FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

# Zadání diplomové práce

| | |
|---|---|
| **Název:** | SOS II - Studentsky odevzdavaci system |
| **Student:** | Bc. Tomáš Pavlůsek |
| **Vedoucí:** | Ing. Jiří Hunka |
| **Studijní program:** | Informatika |
| **Obor / specializace:** | Softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | do konce letního semestru 2023/2024 |

## Pokyny pro vypracování

Již nyní existuje sos.fit.cvut.cz - dále jen portál, který slouží k podpoře týmových cvičení na FIT ČVUT. Tento portál byl již od první verze rozšířen prací Bc. Maxe Hejdy, v rámci bakalářské práce Systém pro správu ročníkových prací. Cílem této práce, je navázat na aktuální stav portálu a zajistit jeho plné použití pro specifické potřeby Softwarových týmových předmětu BI-SP1 a BI-SP2. Dále zajistit stabilizaci systému pro budoucí provoz a možné další rozšiřování.

Postupujte v těchto krocích:
1. Analyzujte současný stav portálu včetně dílčích úprav ostatních autorů.
2. Analyzujte vhodným způsobem a řádně potřeby předmětů BI-SP1 a BI-SP2, dále také analyzujte aktualizované potřeby předmětu NI-NUR, pro který byl portál primárně vyvíjen.
3. Navrhněte a realizujte potřebné úpravy portálu k plnohodnotnému využití dle provedené analýzy.
4. Ověřte použitelnost úprav v praxi, ideálně zajistěte support prvního běhu BI-SP1 nebo BI-SP2.
5. Zajistěte projekt tak, aby bylo snadné a efektivní dalšími lidmi portál nadále vyvíjet a rozvíjet.
6. Zhodnoťte dosažené výsledky a stav portálu.

---

*Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. února 2023 v Praze.*

Diplomová práce

# SOS II - STUDENTSKÝ ODEVZDÁVACÍ SYSTÉM

**Bc. Tomáš Pavlůsek**

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
3. května 2023

# Obsah

# Seznam obrázků

# Seznam tabulek

# Seznam výpisů kódu

*Nejprve bych chtěl poděkovat vedoucímu diplomové práce Ing. Jiřímu Hunkovi za odborné vedení, vstřícný přístup a za příležitost společně s dalšími studenty tvořit a provozovat výukovou aplikaci na FIT ČVUT. Této příležitosti a zkušenosti si nesmírně vážím. Velké poděkování si rovněž zaslouží Bc. Max Hejda, který k projektu významně přispěl svou bakalářskou prací a během tvorby této práce mi pomáhal podáváním zpětné vazby na kód a testováním nových funkcionalit a rozšíření. Jeho zájem o projekt byl pro mě velkou motivací. Také bych rád poděkoval studentům předmětu BI-SP1 Timoteji Adamcovi, Luciánovi Kučerovi, Janu Mrázkovi, Janu Jamnickému, Markovi Hujovi, Janu Pitákovi a Jakubovi Sedláčkovi za jejich usilovnou práci v průběhu letního semestru 2022/23, která byla pro projekt velkým přínosem.*

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 3. května 2023 .....

# Abstrakt

Studentský odevzdávací systém (SOS) je webová aplikace sloužící ke správě týmových i individuálních semestrálních prací v předmětech vyučovaných na Fakultně informačních technologií (FIT) Českého vysokého učení technického v Praze a ročníkových prací na Gymnáziu Jiřího Gutha-Jarkovského. Tato diplomová práce se zabývá rozšířením funkcionalit aplikace pro využití v dalších předmětech vyučovaných na FIT. Práce popisuje analýzu počátečního stavu aplikace, provedenou analýzu požadavků, návrh a implementaci potřebných úprav a rozšíření a proces nasazování nových verzí do provozu. Jedním z realizovaných rozšíření byla i integrace se systémy KOS a FIT Klasifikace, které jsou na FIT používány k zaznamenávání studijních výsledků. Práce se také zabývá stabilizací celého projektu zavedením potřebné infrastruktury a zapojením studentů FIT do vývoje. Výsledkem práce je jednak plně funkční software, který je využíván při výuce několika předmětů na FIT, ale zároveň i stabilizovaný projekt se zavedenou infrastrukturou, který mohou studenti FIT dále efektivně vyvíjet a rozvíjet.

**Klíčová slova** systém pro podporu výuky, semestrální práce, webová aplikace, Django, CI/CD, KOS, FIT Klasifikace

# Abstract

Student submission system (SOS) is a web application used for the administration of both team and individual semestral projects in courses taught at the Faculty of Information Technology (FIT) of the Czech Technical University in Prague, as well as seminar works at the Jiří Guth-Jarkovský Grammar School. The subject matter of this thesis is the extension of the functionalities of the application for use in additional courses taught at FIT. It describes the analysis of the initial state of the application, the analysis of requirements, the design and implementation of the necessary modifications and extensions, and the process of deploying new versions. One of the implemented extensions was the integration with the KOS and Grades FIT systems, which are used at FIT to record study results. The thesis also deals with the stabilization of the entire project by introducing the necessary infrastructure and involving the students of FIT in the development process. The result of the work is a fully functional software, which is used in several courses at FIT, but also a stabilized project with an established infrastructure, that the students of FIT can further efficiently develop and extend.

**Keywords** learning management system, semestral works, web application, Django, CI/CD, KOS, Grades FIT

# Seznam zkratek

| | |
|---|---|
| API | Application Programming Interface |
| BI-PYT | (předmět) Programování v Pythonu |
| BI-SP1 | (předmět) Softwarový týmový projekt 1 |
| BI-SP2 | (předmět) Softwarový týmový projekt 2 |
| BI-SWI | (předmět) Softwarové inženýrství |
| BI-TUR | (předmět) Tvorba uživatelského rozhraní |
| BPMN | Business Process Model and Notation |
| CI/CD | Continuous Integration / Continuous Delivery |
| ČVUT | České vysoké učení technické |
| DNS | Domain Name System |
| ER | Entity Relationship (Diagram) |
| FIT | Fakulta informačních technologií |
| FURPS | Functionality, Usability, Reliability, Performance, Supportability |
| GJGJ | Gymnázium Jiřího Gutha-Jarkovského |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| ICT | Informační a komunikační technologie |
| JS | JavaScript |
| JSON | JavaScript Object Notation |
| KOS | Komponenta Studium |
| LTS | Long-term support (release; verze s dlouhodobou podporou) |
| MoSCoW | Must have, Should have, Could have, Won't have |
| MR | Merge Request |
| NI-NUR | (předmět) Návrh uživatelského rozhraní |
| OAAS | OAuth 2.0 autorizační server |
| ORM | Objektově relační mapování |
| PDF | Portable Document Format |
| PEP | Python Enhancement Proposal |
| PK | Primární klíč |
| REST | Representational state transfer |
| SDK | Software Development Kit |
| SMTP | Simple Mail Transfer Protocol |
| SOS | Studentský odevzdávací systém |
| SP tým | Studentský tým v předmětech BI-SP1 a BI-SP2 |
| SSH | Secure Shell |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| VIC | Výpočetní a informační centrum ČVUT |
| VPN | Virtual Private Network |
| XML | Extensible Markup Language |
| XSS | Cross-site scripting |

# Úvod

V roce 2021 jsem rámci své bakalářské práce vytvořil SOS – Studentský odevzdávací systém [1]. Jedná se o webovou aplikaci sloužící ke kompletní správě týmových projektů – umožňuje tvorbu zadání a týmů které je budou vypracovávat, odevzdávání vypracovaných řešení studenty a jejich ohodnocování vyučujícími. Aplikace byla po dokončení nasazena do provozu při výuce předmětu „NI-NUR – Návrh uživatelského rozhraní“ na Fakultě informačních technologií Českého vysokého učení technického v Praze ve spolupráci s vyučujícím tohoto předmětu a vedoucím mé bakalářské práce Hunkou. SOS tak byl dva roky využíván při reálné výuce. Ačkoliv byl původní záměr vytvořit aplikaci pro podporu výuky na FIT ČVUT, především předmětu NI-NUR a případně i dalších, ukázalo se, že najde využití například i na středních školách, jak ukazuje Hejda ve své bakalářské práci z roku 2022, kde systém rozšiřuje a nasazuje do provozu na pražském Gymnáziu Jiřího Gutha-Jarkovského, kde je využíván ke správě ročníkových prací [2].

Tato práce navazuje na mou bakalářskou práci a bakalářskou práci Hejdy, shrnuje dosavadní vývoj a provoz SOS a věnuje se dalšímu rozvoji projektu. Popisuji zde proces nasazení SOS do výuky předmětu „BI-SP1.21 - Softwarový týmový projekt 1“ a jiných příbuzných předmětů a s tím související úpravy aplikace. Rozebírám také svou další práci na projektu, jejímž cílem je především zajištění celého projektu tak, aby mohl být na FIT ČVUT dlouhodobě a bezpečně využíván při výuce a aby na jeho rozvoji mohli snadno a efektivně pracovat další studenti FIT ČVUT, a to i bez mé přítomnosti po dokončení magisterského studia.

Téma jsem si zvolil především proto, že chci navázat na výsledky své bakalářské práce a zajistit co nejlepší dlouhodobou využitelnost aplikace SOS, kterou jsem v rámci ní vytvořil. Zároveň se zde mohu věnovat problematice, se kterou jsem již dobře obeznámen.

## Cíle práce

Tato práce má dva hlavní cíle. Jedním z nich je SOS rozšířit tak, aby plně vyhovoval požadavkům předmětů BI-SP1 a BI-SP2 vyučovaných na FIT ČVUT, následně jej nasadit do provozu při výuce předmětu BI-SP1 a zajistit jeho bezproblémové fungování po celý semestr. Úpravy je potřeba provést při zachování veškerých dosavadních funkcionalit systému, neboť je již využíván na fakultě v předmětu NI-NUR a také na gymnáziu pro správu ročníkových prací. To zahrnuje kompletní analýzu současného stavu systému, analýzu požadavků předmětů BI-SP1 a BI-SP2 a také aktualizaci požadavků předmětu NI-NUR, mj. s využitím zpětné vazby získané z provozu systému ve dvou běžích tohoto předmětu. Následně přistoupím k implementaci potřebných změn a po dokončení splnění požadavků ověřím s garantem BI-SP1. Poté bude možné systém při výuce využít. Pokud k tomu dojde, bude mým cílem shromáždit co nejvíce zpětné vazby od uživatelů a případně zajistit potřebné úpravy.

Druhým hlavním cílem práce je zajištění dlouhodobé udržitelnosti projektu bez nutnosti mé intenzivní spolupráce. Budu se věnovat zkvalitnění automatických testů a zajištění stabilního

produkčního i testovacího prostředí pro běh systému v rámci fakulty, spolu s plně automatizovaným nasazováním nových verzí do obou prostředí. Budoucnost projektu nepochybně závisí i na tom, jestli jej bude mít kdo udržovat a rozvíjet. Proto v rámci předmětu BI-SP1 povedeme spolu s vedoucím práce Hunkou a kolegou Hejdou tým studentů bakalářského programu, kteří se s projektem seznámí a budou se během semestru (a potenciálně i toho následujícího v rámci předmětu BI-SP2) podílet na jeho rozvoji.

Dílčím cílem práce je zapracování dosud získané zpětné vazby a implementace některých funkcionalit, které jsem v rámci své bakalářské práce už implementovat nestihl, jako je například integrace s aplikací FIT Klasifikace.

# Kapitola 1

# Analýza

*V této kapitole čtenáře seznámím s existujícím systémem SOS. Popisuji vývoj od jeho počátku a aktuální stav a možnosti systému. Shrnuji také technologie, na kterých je systém postaven. Následně se věnuji analýze požadavků, kde se zaměřím především na předmět NI-NUR, kde je již systém dva roky využíván a předměty BI-SP1 a BI-SP2, kde pro jeho využití existuje silný potenciál a především iniciativa ze strany zadavatele a vedoucího práce Hunky. Výsledkem této analýzy je seznam konkrétních funkčních a nefunkčních požadavků, které musí systém SOS pro plnohodnotné využití v těchto předmětech splňovat. Také se zabývám klasifikací jednotlivých požadavků podle míry jejich splnění současnou verzí systému. V závěru kapitoly se také zaměřuji na systémy pro zaznamenávání studijních výsledků používané na FIT ČVUT a možnosti integrace s SOS.*

## 1.1 Analýza existující aplikace

SOS – Studentský odevzdávací systém je webová aplikace pro podporu výuky, používaná na Fakultě informačních technologií Českého vysokého učení technického v Praze (dále jen FIT ČVUT) a pražském Gymnáziu Jiřího Gutha-Jarkovského (dále jen GJGJ). Slouží ke kompletní správě týmových i individuálních semestrálních prací, resp. ročníkových prací. Umožňuje definici zadání a týmů, které je budou vypracovávat, odevzdávání vypracovaných řešení studenty a následné ohodnocování těchto řešení vyučujícími. Podporuje také odevzdávání řešení v předem definovaných iteracích.

Aplikace je konfigurovatelná dle potřeb konkrétního předmětu. Například zadání projektů mohou v závislosti na konfiguraci definovat vyučující, a to jak globálně, tak i pro každý tým zvlášť, nebo sami studenti. Složení týmů mohou také určovat buď vyučující nebo studenti. Konfigurovat lze i jednotlivé kontrolní body, jejich počet, obsah, termíny odevzdání a bodové ohodnocení. Samotnou konfiguraci lze provádět na úrovni předmětu, nebo i konkrétních vyučujících, pokud jich daný předmět vyučuje více. Podrobněji se možnostem konfigurace SOS věnuji ve své bakalářské práci, proto je zde nebudu již detailněji rozebírat [1].

Data se do SOS importují skrze API z datového zdroje, ve kterém škola udržuje veškeré potřebné informace o studentech, vyučujících a vyučovaných předmětech. Na FIT ČVUT datový zdroj představuje KOS, na GJGJ je to systém Bakaláři. Uživatelé se do aplikace přihlašují pomocí svého školního účtu, na FIT ČVUT s využitím fakultního OAuth serveru, na GJGJ prostřednictvím Google OAuth.

### 1.1.1 Vývoj existující aplikace

Po dokončení mé bakalářské práce byl SOS připraven k nasazení do výuky v předmětu NI-NUR na FIT ČVUT, k čemuž také hned v následujícím semestru došlo ve spolupráci s vyučujícím Hunkou. Při používání aplikace se postupně objevovaly nové požadavky a problémy, které jsem po celou dobu aktivně řešil. K projektu se také připojil Hejda, který se rozhodl SOS upravit pro využití pro správu ročníkových prací na GJGJ a začal jej k tomu uzpůsobovat.

#### 1.1.1.1 Úpravy před prvním nasazením a během provozu

Těsně před nasazením do výuky v předmětu NI-NUR se objevil nový požadavek ze strany vyučujícího, a to mít možnost schvalovat studenty vytvořená zadání projektů, čímž bude teprve studentům umožněno začít odevzdávat řešení. Tuto funkcionalitu jsem implementoval ještě před začátkem výuky. Implementace spočívala v uložení informace o datu schválení ke každému zadání, kontrole existence tohoto záznamu při pokusu o odevzdání řešení, pokud bylo související zadání vytvořeno studentem a také přidání prvků uživatelského rozhraní potřebných ke zobrazení informace o stavu schválení zadání a samotnému schvalování zadání vyučujícím.

Dalším požadavkem, který se objevil ke konci semestru, byl mechanismus pro nakládání s členem týmu, který se na týmové práci nepodílí a neměl by tedy mít nárok na zápočet, přičemž studenta z týmu úplně odstranit nebylo vyhovujícím řešením. Přidal jsem tedy ke každému záznamu o členství v týmu informaci, zdali je člen aktivní a uživatel odpovědný za správu týmu, což je v závislosti na konfiguraci předmětu vyučující, nebo vedoucí týmu z řad studentů, získal možnost tuto informaci upravovat. Neaktivní člen týmu neobdrží žádné body za týmem odevzdaná řešení.

Dále jsem během provozu aplikace řešil drobné úpravy a opravy, z nichž asi žádná sama o sobě nestojí za zmínku. Úpravy spočívaly především v uvolnění omezení na maximální délku různých textů zadávaných do aplikace uživatelem, nastalé problémy se týkaly nejčastěji špatného zobrazení různých méně podstatných informací a občasných pádů systému v situacích, které nebyly dostatečně pokryty automatickými testy ani manuálním testováním před nasazením.

#### 1.1.1.2 Semestrální práce v předmětu NI-NUR

V prvním semestru svého magisterského studia jsem si zapsal předmět NI-NUR, kde byl poprvé využit SOS. Používání aplikace jsem si tak sám vyzkoušel z pohledu studenta. V rámci tohoto předmětu jsme se s týmem v naší semestrální práci věnovali návrhu zlepšení učitelské části SOS, převážně tedy procesu konfigurace předmětu a ohodnocování odevzdaných řešení.

Provedli jsme analýzu požadavků a případů užití a na základě toho jsme vytvořili Lo-Fi prototyp v nástroji Balsamiq[1], který je k dispozici v příloze A. Dále jsme vytvořili i Hi-Fi prototyp přímo v tehdejší verzi SOS, který je k dispozici ve formě Merge Requestu[2][3] v GitLab repozitáři projektu SOS. Prototyp byl podroben heuristické analýze a usability testování, což je taktéž zdokumentováno v příloze A. Následně byl upraven tak, aby byly odstraněny nejzávažnější zjištěné problémy. K ostatním problémům byl proveden pouze návrh řešení, jednalo se převážně o vizuální úpravy.

Během tvorby semestrální práce byla naše představa taková, že prototyp bude následně využit jako vzor pro implementaci zcela nového frontendu SOS, který bude vytvořen s využitím odlišné technologie než původní verze, v úvahu připadal především framework Vue.js[4]. K tomu nicméně nikdy nedošlo a ani to není cílem této práce, částečně také proto, že od doby vytvoření prototypu došlo k aktualizaci požadavků. Prototyp nicméně stále může sloužit jako inspirace při dalším

[1] `https://balsamiq.cloud/`
[2] `https://gitlab.fit.cvut.cz/sos/sos/-/merge_requests/130`
[3] `https://gitlab.fit.cvut.cz/sos/sos/-/merge_requests/136`
[4] `https://vuejs.org/`

rozvoji systému, jak v rámci této práce, tak i pro tým, který se na rozvoji projektu bude podílet v rámci předmětu BI-SP1.

#### 1.1.1.3 Rozšíření pro gymnázium

Hejda v roce 2022 v rámci své bakalářské práce [2] systém rozšířil tak, aby jej bylo možné využít pro správu ročníkových prací na GJGJ, kde jej také následně nasadil do provozu. Hlavní novou funkcionalitou je možnost sdílet zadání mezi předměty prostřednictvím tzv. *zdrojů zadání* – předměty mohou být nastaveny jako zdroje zadání pro jiné předměty, které pak agregují vybraná zadání z těchto předmětů. Na GJGJ je pak v SOS vytvořen předmět Ročníková práce, který agreguje zvolená zadání z ostatních středoškolských předmětů, jako je například Matematika nebo Český jazyk. Tato funkcionalita by potenciálně mohla najít využití i na FIT ČVUT, jelikož zde studenti často vypracovávají semestrální práce na stejné či podobné téma ve více předmětech zároveň.

Kromě zdrojů zadání byl systém navíc rozšířen o možnost přihlašování prostřednictvím Google OAuth, e-mailové notifikace a byl také výrazně zjednodušen proces nasazování pomocí technologie Docker. Jako datový zdroj je místo KOS v konfiguraci pro gymnázium využíván systém Bakaláři. Veškeré tyto změny a úpravy Hejda podrobně popisuje ve své bakalářské práci, proto se zde jimi již detailněji zabývat nebudu.

## 1.2 Analýza požadavků

V roce 2022 svolil garant předmětu Softwarové inženýrství (dále jen BI-SWI) a souvisejících předmětů Softwarový týmový projekt 1 a 2 (dále jen BI-SP1 a BI-SP2) k využití SOS při výuce těchto předmětů. Vznesl nové požadavky na tento systém, které v této kapitole dále rozebírám. Využití SOS v BI-SP1 také podmínil provedením akceptačního testování před nasazením.

Ačkoliv byl SOS od začátku navržen jako vysoce konfigurovatelný systém a potenciální využití v BI-SP1 a BI-SP2 jsem se při jeho vývoji v rámci své bakalářské práce snažil zohlednit, ukázalo se, že pro plnohodnotné využití v BI-SP1 a BI-SP2 je nutné jej rozšířit a upravit. V této sekci analyzuji potřeby předmětů BI-SP1 a BI-SP2 a také předmětu NI-NUR, který již SOS používá, a tudíž je k dispozici zpětná vazba z reálného provozu. Výsledkem je seznam funkčních a nefunkčních požadavků.

### 1.2.1 Metodika

Publikace *Software Requirements* uvádí několik obvyklých metod elicitace požadavků [3, kap. 7]:

- **Interview** – zjišťování potřeb uživatelů rozhovory s jednotlivci nebo malými skupinami,
- **Workshop** – strukturovaná a moderovaná schůzka s vybranými zástupci zúčastněných stran,
- **Focus group** – interaktivní schůzka s reprezentativní skupinou uživatelů,
- **Pozorování** – sledování uživatelů při provádění potřebných úkonů (např. s již existujícím produktem),
- **Dotazník** – kvantitativní výzkum se skupinou (budoucích) uživatelů produktu,
- **Analýza systémového rozhraní** – zkoumání dalších systémů, se kterými je třeba vyvíjené řešení integrovat,
- **Analýza uživatelského rozhraní** – zkoumání uživatelského rozhraní existujícího nebo konkurenčního řešení,

- **Dokumentová analýza** – zkoumání dokumentace existujících nebo konkurenčních řešení, souvisejících norem a regulací a dalších relevantních dokumentů.

Analýzu potřeb jsem realizoval zejména metodou nestrukturovaných rozhovorů, které jsem vedl s vyučujícími předmětů, především s vyučujícím NI-NUR, BI-SP1 a BI-SP2 a zadavatelem práce Hunkou. Ten kromě řady vlastních vstupů zprostředkoval také požadavky garanta BI-SP1 a BI-SP2, jejichž splněním podmínil využití SOS ve výuce těchto předmětů.

Dále jsem se zaměřil na dokumentovou analýzu v podobě zkoumání materiálů dostupných pro studenty jednotlivých předmětů v systémech FIT ČVUT Course Pages[5] a Moodle[6]. Důležité byly zejména informace o průběhu jednotlivých předmětů a podmínkách hodnocení.

Doplňujícím zdrojem bylo pozorování reálných uživatelů při práci se současnou verzí SOS během posledních dvou běhů NI-NUR. Pozorování jsem prováděl intenzivně během prvního běhu, jelikož jsem sám byl studentem NI-NUR a měl jsem řadu příležitostí pozorovat ostatní studenty při práci se systémem. Během druhého běhu jsem měl také několik příležitostí pozorovat studenty NI-NUR při práci se systémem nebo s nimi o jejich zkušenostech s ním hovořit.

### 1.2.2 Potřeby předmětů BI-SP1 a BI-SP2

Předměty BI-SP1 a BI-SP2 na sebe navazují a mají podobný průběh, proto jsem je pro účely analýzy spojil do jednoho celku. Studenti v BI-SP2 rozvíjejí projekt, který rozpracovali v BI-SP1. Předměty se sice liší náplní práce, kde v BI-SP1 je kladen důraz především na analýzu a návrh softwarového řešení a v BI-SP2 naopak na samotnou implementaci a ověřování funkčnosti, nicméně z hlediska průběhu jsou tyto předměty téměř totožné.

V analýze potřeb předmětů vycházím jak z oficiálních stránek předmětů BI-SP1 [4] a BI-SP2 [5] v systému Moodle, tak i z informací získaných od vyučujících včetně garanta.

#### 1.2.2.1 Tvorba týmů

Před začátkem semestru vyučující vypíší projekty, na kterých budou studenti pracovat. Podle KOS má předmět BI-SP1 aktuálně pouze dva vyučující, nicméně projekty mohou vypsat po schválení garantem i další vyučující.

Studenti si zvolí projekt dle svého zájmu a požádají vyučujícího který jej vypsal o vstup do tohoto projektu. Vyučující může u projektu uvést také požadované role, jako například Grafik, Projektový manažer apod. Studenti se mohou přihlásit přímo na preferovanou roli. Je doporučeno, aby tato odpovídala jejich zaměření. Vyučující mohou žádosti studentů přijmout nebo zamítnout a utvořit tak týmy přibližně o čtyřech až šesti studentech.

#### 1.2.2.2 Práce v semestru

Tým se se svým vedoucím vyučujícím pravidelně schází a konzultuje práci na projektu. Pokud vyučující neurčí jinak, je práce rozdělena do tří iterací. Na konci každé iterace tým prezentuje dosažené výsledky a vyučující provede jejich bodové ohodnocení. Za nadstandardní práci může vyučující udělit bonusové body.

#### 1.2.2.3 Přerozdělení bodů

Přerozdělení bodů je nástrojem, který umožňuje motivovat jednotlivé členy týmu. V rámci přerozdělení má v jednotlivých iteracích tým možnost přerozdělit body tak, aby bodový zisk každého člena týmu odpovídal jeho odvedené práci. Přerozdělení se udává procentem z počtu bodů získaných za danou iteraci. Maximálně lze v každé iteraci přidat nebo odebrat každému studentovi až 60 % bodů, celkový součet přerozdělení musí být nulový.

[5] `https://courses.fit.cvut.cz/`
[6] `https://moodle-vyuka.cvut.cz/`

#### 1.2.2.4 Organizace

Specifikem těchto dvou předmětů je fakt, že pro ně nejsou v systému KOS vypsány žádné paralelky. Předmět totiž nemá v rozvrhu vypsány žádné přednášky ani cvičení, schůzky jednotlivých týmů s jejich vyučujícími se domlouvají individuálně a teoretickou přípravu poskytuje místo přednášek předmět BI-SWI, který mají studenti většinou zapsaný současně. Z uživatelského hlediska to pro systém SOS není podstatné, nicméně je to třeba zohlednit při návrhu aplikační logiky a importu předmětu z datového zdroje. Podrobněji se tomuto věnuji v následující kapitole.

#### 1.2.2.5 Současné řešení

Dosud byla administrativa projektů v BI-SP1 a BI-SP2 řešena v systému Moodle. Vyučující zde vypsali nabízené projekty, ke kterým se poté studenti přihlásili prostřednictvím ankety. Toto řešení bylo studenty i vyučujícími hodnoceno jako nepřehledné a nepohodlné. Také končilo u tvorby týmu, veškerá další agenda byla řešena v dalších nástrojích, plně v režii daného vyučujícího či týmu. Využití specializovaných nástrojů pro týmovou spolupráci, jako je například GitLab, Slack a Redmine by stále dávalo smysl i při nasazení SOS. Ten by sloužil jako jednotný nástroj pro správu odevzdaných řešení a jejich hodnocení, přičemž samotná odevzdání mohou být realizována například formou odkazů na obsah v již zmíněném systému GitLab nebo kdekoliv jinde.

### 1.2.3 Potřeby předmětu NI-NUR

SOS byl primárně navržen především pro předmět NI-NUR, byl zde již nasazen v posledních dvou běžích a s jeho využitím je počítáno i do budoucna. V analýze tedy vycházím jak z oficiálních stránek předmětu na FIT ČVUT Course Pages [6], tak i z takto získané zpětné vazby od skutečných uživatelů z řad studentů a vyučujícího Hunky, který v NI-NUR odpovídá za semestrální práce.

#### 1.2.3.1 Tvorba týmů

Studenti pracují v týmech o dvou až čtyřech členech. Téma práce si studenti volí sami, může být součástí jiného autorského projektu či práce studentů. Na prvním cvičení studenti do SOS začnou zapisovat své projekty, ke kterým se mohou jejich kolegové přidat prostřednictvím žádosti, kterou autor projektu přijme nebo zamítne. Autoři projektů do aplikace zadávají jeho název a popis zadání, případně i přílohy ve formě nahraných souborů nebo webových odkazů. Utvořené týmy pak vyučující schválí, čímž studentům umožní začít odevzdávat řešení.

#### 1.2.3.2 Práce v semestru

Práce je rozdělena do tří iterací s definovaným termínem odevzdání. V první iteraci tým provádí rozvahu, definuje produkt, persony, uživatelské cíle, případy užití a scénáře. Výsledkem jejich práce je PDF dokument, který nahrají do systému. Součástí první iterace je také Lo-Fi prototyp budoucího produktu, který bývá tvořen zejména v nástroji Balsamiq.

V druhé iteraci studenti vypracovávají rešerši konkurenčních aplikací, která je opět odevzdána ve formě dokumentu. Především ale zpracují Hi-Fi prototyp, který by měl obsahovat vše, co bylo naplánováno v předchozím bodě. Své řešení mohou odevzdat formou archivu s prototypem, nebo jako webový odkaz, na kterém se prototyp nachází.

V poslední iteraci studenti provádí heuristickou analýzu a usability testování prototypu z předchozí iterace. Výstupem jsou opět dokumenty, případně upravený prototyp zohledňující objevené problémy.

Výsledky každé iterace tým na cvičení předvede vyučujícímu, který následně provádí bodové hodnocení. Maximální počet bodů a minimální počet bodů nutný k získání zápočtu je předem definován. Vyučující také může dle svého uvážení udělit bonusové body za mimořádnou práci.

Součástí odevzdání všech iterací je i reportování, kteří z členů realizovali danou část. Pokud některý z členů týmu nepřispěl přiměřenou aktivitou, mohou mu být body z dané iterace odebrány, díky čemuž může i ztratit nárok na zápočet.

#### 1.2.3.3 Testování

V rámci třetí iterace týmy provádí usability testování prototypu své aplikace. Studenti z ostatních týmů se mohou stát testery a za potvrzenou účast na testování obdrží bonusové body. Studentovi se do celkového hodnocení připočtou body maximálně za jedno testování.

#### 1.2.3.4 Zpětná vazba k současné verzi

Z rozhovorů s vyučujícím Hunkou, se studenty NI-NUR kteří SOS používali a také z mé vlastní zkušenosti se systémem jsem vyvodil několik hlavních problémů, které současná verze má. V této sekci je shrnuji ve formě textového popisu, dále jsou pak ve strukturované podobě zahrnuty do jednotlivých funkčních požadavků dále v této kapitole, v sekci 1.2.5.

Prvním z nich je realizace studentského testování v systému, která neodpovídá tomu, jak ve skutečnosti probíhá. Systém studentům umožňuje přihlášení k jednotlivým projektům jako tester. Vyučující v konfiguraci předmětu určí maximální počet testovaných projektů na testera, maximální počet testerů na projekt a počet bodů udělených za otestování. Tým u každého odevzdaného řešení potvrzuje, kteří z přihlášených testerů řešení skutečně otestovali. Testeři pak obdrží definovaný počet bodů v dané iteraci, pokud byla jejich účast na testování potvrzená nejméně u jednoho odevzdaného řešení pro tuto iteraci. Ve skutečnosti ale není testování realizováno u všech iterací, ale pouze u té poslední. Testování navíc probíhá z principu již před odevzdáním třetí iterace. K samotnému odevzdání třetí iterace navíc ani nemusí ze strany týmu dojít, čímž by tester o body přišel, i když se testování účastnil. Tým navíc účast na testování potvrzuje u každého odevzdaného řešení, i když probíhá pouze jednou v rámci iterace. Vyučující se vyjádřil tak, že by účast na testování měla být navázána přímo na daný tým, nikoliv na odevzdaná řešení, pak mohlo být potvrzení účasti provedeno kdykoliv během semestru bez ohledu na týmem odevzdaná řešení. To by oproti současnému řešení více odpovídalo realitě NI-NUR, nicméně já jsem toho názoru, že by takto byla příliš omezena univerzálnost systému. V jiných předmětech by mohlo dávat smysl navázat testování volitelně na jednotlivé iterace. Konkrétně v NI-NUR by pak mohlo být navázáno pouze na druhou iteraci, kde týmy odevzdávají Hi-Fi prototyp, jehož existence je prerekvizitou proveditelnosti usability testování. Toto bude ještě dále diskutováno s vyučujícím a budu se tomu věnovat detailněji v následující kapitole. Současné řešení nicméně realitě určitě neodpovídá a je potřeba jej upravit.

Jinou situací, kde současná verze SOS selhává, je proces tvorby týmů v případě, že je dle konfigurace v režii studentů. V okamžiku, kdy student vytvoří nový projekt, tedy současně zadání a nový tým, je toto zadání odesláno vyučujícímu ke schválení. Nově vzniklý tým nemůže odevzdávat řešení, dokud není zadání schváleno. Vyučujícímu ale před schválením nezáleží pouze na samotném zadání, nýbrž na projektu jako takovém, tedy včetně složení týmu. Zároveň nechce, aby po schválení mohli studenti složení týmu měnit bez jeho explicitního potvrzení této změny. Souvisejícím nedostatkem současného řešení pak může být také to, že pokud v daném předmětu studenti sice sami tvoří týmy, ale zadání jsou již předem definována vyučujícím, neexistuje už pro vyučujícího vůbec žádný nástroj, jak vyjádřit souhlas či nesouhlas se složením týmu. Toto je také potřeba napravit a schválení vyučujícím navázat přímo na projekt/tým, nikoliv pouze na zadání.

Ostatní zjištěné nedostatky se týkaly především nepřehlednosti a ergonomických vad uživatelského rozhraní nebo vyplývaly ze skrytých chyb v aplikační logice. Studenti například často hlásili, že se nemohou nalézt projekt, ke kterému by se přihlásili jako testeři. Problém byl jednak v tom, že u jednotlivých projektů chyběla informace o tom, proč již nepřijímá nové testery a projekty podle toho ani nebylo možné filtrovat, zároveň neměla řada projektů povoleny žádosti

o testování, tudíž testery vůbec nepřijímaly. Výchozí možností bylo žádosti nepovolit a vedoucí týmů často ani nevěděli o tom, že takové nastavení existuje. Důsledkem bylo, že se testování často řešilo zcela mimo SOS, což následně značně znepřehledňovalo bodové hodnocení. S tím souvisí další nedostatek, kterým je zobrazení bodů získaných jednotlivými studenty. Vyučující sice na stránce přehledu klasifikace vidí celkový součet bodů každého studenta za všechny odevzdané iterace a potvrzená testování, nicméně studentovi je na stránce předmětu i detailu projektu k dispozici pouze počet bodů, které za jednotlivé iterace obdržel tým jako takový, není zde započítáno testování. Správný součet má sice student k dispozici na domovské stránce, nicméně tu studenti nenavštěvovali vůbec nebo jen velmi zřídka, vzhledem k tomu, že NI-NUR byl dosud jediným předmětem, který SOS využíval.

Další problémy byly drobného nebo obecného charakteru a podrobně je zde rozebírat nebudu. Pozornost jim bude věnována v následujících kapitolách, kde je demonstrováno jejich případné řešení.

### 1.2.4 Potřeby dalších předmětů

S předměty BI-SP1 a BI-SP2 z hlediska semestrálních prací nepřímo souvisí i předmět BI-SWI – Softwarové inženýrství. Studenti BI-SWI, kteří mají současně zapsaný předmět BI-SP1 realizují semestrální práci v BI-SP1 a počítá se jim i do BI-SWI. Ostatní studenti vypracovávají podobnou semestrální práci v menším rozsahu přímo v BI-SWI [7].

Hlavní rozdíl oproti BI-SP1 a BI-SP2 tkví v tom, že studenti si téma práce volí sami a tvorba týmů je také plně v jejich režii. Práce je konzultována a odevzdávána na cvičení, odevzdání probíhá ve třech iteracích s předem definovanou náplní.

Po důkladném studiu materiálů pro studenty BI-SWI dostupných v systému Moodle jsem došel k závěru, že realizace semestrální práce v tomto předmětu je z organizačního a procesního hlediska téměř totožná s předmětem NI-NUR, pouze odpadá studentské testování. Pokud bude SOS splňovat požadavky NI-NUR, bude pak jistě plně využitelný i pro semestrální práce v BI-SWI.

### 1.2.5 Funkční a nefunkční požadavky

Analýza požadavků slouží k vymezení hranic systému a k zachycení omezení, která jsou na systém kladena. V praxi umožňuje klientovi a zadavateli přesně definovat zadání projektu a rovněž se využívá při kalkulaci ceny. Správně definovaný požadavek musí být jednoznačný, nesmí si odporovat s dalšími požadavky a dodavatel musí být schopen jeho splnění zajistit. Splnění požadavku musí být možné jednoznačně ověřit, což je pak předmětem akceptačního testování. Součástí definice požadavku je určení jeho důležitosti a náročnosti jeho splnění.

Požadavky lze rozdělit do dvou základních kategorií – funkční a nefunkční. Funkční požadavky popisují konkrétní funkcionality a schopnosti systému, nefunkční pak udávají obecné vlastnosti systému a omezení na něj kladená. Kromě tohoto základního rozdělení jsou dalšími nástroji ke klasifikaci a prioritizaci požadavků metody FURPS a MoSCoW.

Metoda FURPS požadavky rozděluje do pěti kategorií – první kategorie představuje zmíněné funkční požadavky, ostatní spadají do obecné kategorie nefunkčních požadavků [8]:

- **Functionality** (funkcionalita) – funkční požadavky,
- **Usability** (použitelnost) – jak lze systém používat, interakce s uživatelem, dokumentace,
- **Reliablility** (spolehlivost) – dostupnost systému, četnost a závažnost chyb a výpadků,
- **Performance** (výkon) – rychlost odezvy systému, technické parametry,
- **Supportability** (podporovatelnost/rozšiřitelnost) – údržba a podpora systému, budoucí možnost rozšíření o další vlastnosti, přizpůsobitelnost novým okolnostem.

Metoda MoSCoW naopak požadavky rozděluje podle priority [9, kap. 6]:

- **Must have** – požadavek musí být splněn v každém případě,
- **Should have** – požadavek by měl být splněn, pokud to bude možné – často se jedná o kritický požadavek, který lze ale v případě nutnosti pokrýt i alternativními cestami,
- **Could have** – požadavek je žádoucí splnit, ale není to nutné,
- **Won't have** – zúčastněné strany se shodly, že požadavek aktuálně nebude implementován, ale je možné jej zvážit v budoucnosti.

Vzhledem k tomu, že tato analýza pojednává o systému SOS, který je již implementován a má být pouze rozšířen, je u každého požadavku také uvedeno, zda jej již aktuální implementace splňuje či nikoliv. U několika požadavků je uvedeno splnění pouze částečné či s výhradami, což bude dále podrobněji rozebráno.

#### 1.2.5.1 Funkční požadavky

Z analýzy vyplynul níže uvedený seznam funkčních požadavků. Částečně se jedná o aktualizované požadavky z analýzy v rámci vývoje první verze SOS, částečně o nové požadavky. Požadavky jsou rozděleny do tří kategorií:

- FP-GEN-* – generické požadavky,
- FP-NUR-* – požadavky vyplývající z potřeb předmětu NI-NUR,
- FP-SP-* – požadavky vyplývající z potřeb předmětů BI-SP1 a BI-SP2.

Rozdělení do těchto kategorií je pouze orientační a značí především, z které části analýzy daný požadavek vzešel. Funkcionality popsané požadavky z kategorií FP-NUR-* a FP-SP-* budou využitelné i dalšími předměty.

- **FP-GEN-1 – Import dat**
  Aplikace umožňuje načítat data o předmětech a uživatelích z externího datového zdroje. Tímto datovým zdrojem je v konfiguraci pro FIT ČVUT systém KOS, v konfiguraci pro GJGJ systém Bakaláři. Při importu z datového zdroje KOS uživatel zvolí, jaký typ paralelek má být importován.
  **Priorita:** must have
  **Splněno:** ano
  **Náročnost:** -

- **FP-GEN-2 – Konfigurace předmětů**
  Aplikace umožňuje vytváření sekcí pro jednotlivé předměty a jejich následnou konfiguraci tak, aby odpovídaly průběhu těchto předmětů. Konfiguraci je možné provádět v závislosti na volbě správce daného předmětu na úrovni celého předmětu, paralelky nebo konkrétního týmu.
  **Priorita:** must have
  **Splněno:** částečně
  **Náročnost:** vysoká

- **FP-GEN-3 – Aktualizace seznamu studentů**
  Aplikace umožňuje již vytvořený předmět aktualizovat z datového zdroje, ve smyslu přidání nově zapsaných studentů a odstranění těch, kteří si zápis předmětu zrušili.
  **Priorita:** should have
  **Splněno:** ne
  **Náročnost:** nízká

- **FP-GEN-4 – Definice zadání semestrálních prací**
  Aplikace umožňuje uživatelům definovat zadání semestrálních prací včetně příloh v podobě nahraných souborů a webových odkazů. Oprávnění studentů a vyučujících vytvářet a upravovat zadání je dáno konfigurací ve smyslu FP-GEN-2.
  **Priorita:** must have
  **Splněno:** ano
  **Náročnost:** -

- **FP-GEN-5 – Tvorba týmů**
  Aplikace umožňuje vytvořit týmy, ve kterých budou studenti pracovat a přiřadit jim zadání semestrální práce. Zadání je dle konfigurace ve smyslu FP-GEN-2 buďto voleno z předem definované nabídky, nebo je přímo pro vznikající tým vytvořeno ve smyslu FP-GEN-4. Do týmů se studenti přihlašují prostřednictvím žádostí, které odpovědný uživatel může přijmout nebo odmítnout. Odpovědný uživatel může také studenty do týmu pozvat nebo přímo přidat, jedná-li se o studenta nebo o vyučujícího. Odpovědný uživatel je ten, který tým vytvořil. Odpovědný uživatel je student nebo vyučující v závislosti na konfiguraci ve smyslu FP-GEN-2.
  **Priorita:** must have
  **Splněno:** ano
  **Náročnost:** -

- **FP-GEN-6 – Filtrování projektů**
  Uživatel může v aplikaci zobrazit seznam relevantních projektů v rámci předmětu. Projekt je tým ve smyslu FP-GEN-5 s přiřazeným zadáním semestrální práce. Seznam projektů může uživatel filtrovat podle názvu, zodpovědného vyučujícího, člena týmu a volné kapacity. Relevantní projekty jsou pro studenta ty, za které odpovídá jeho vyučující, pro vyučujícího projekty za které sám odpovídá. Správce předmětu má přístup ke všem projektům v rámci předmětu.
  **Priorita:** could have
  **Splněno:** ne
  **Náročnost:** střední

- **FP-GEN-7 – Odevzdávání řešení**
  Studenti mohou v aplikaci odevzdávat vypracovaná řešení semestrálních prací včetně příloh v podobě nahraných souborů a webových odkazů. Řešení odevzdávají v iteracích definovaných konfigurací předmětu ve smyslu FP-GEN-2.
  **Priorita:** must have
  **Splněno:** ano
  **Náročnost:** -

- **FP-GEN-8 – Hodnocení**
  Vyučující může v aplikaci provádět hodnocení studenty odevzdaných řešení ve formě bodového ohodnocení a textové poznámky. Toho hodnocení je po potvrzení vyučujícím zobrazeno příslušným studentům. Vyučující má k dispozici také celkový přehled bodových zisků svých studentů v podobě tabulky. Bodové ohodnocení se odvíjí od konfigurace předmětu ve smyslu FP-GEN-2, vyučující může dle vlastního uvážení udělit také bonusové body.
  **Priorita:** must have
  **Splněno:** ano
  **Náročnost:** -

- **FP-GEN-9 – Statistiky**
  Vyučující může v aplikaci v rámci předmětu zobrazit statistiky o členství studentů v týmech, o volných místech v týmech a o splnění jednotlivých kontrolních bodů studenty.
  **Priorita:** nízká
  **Splněno:** ne
  **Náročnost:** nízká

- **FP-GEN-10 – Sdílení zadání mezi předměty**
  Aplikace umožňuje zadání vytvořená v rámci jednoho předmětu zkopírovat do dalších předmětů. Tyto kopie je pak možné samostatně upravovat. V případě úpravy originálního zadání může uživatel volitelně nechat změny propsat i do takto vytvořených kopií.
  **Priorita:** should have
  **Splněno:** ano
  **Náročnost:** -

- **FP-NUR-1 – Schvalování týmů**
  Pokud týmy dle konfigurace ve smyslu FP-GEN-2 tvoří studenti, musí být schváleny vyučujícím, než jim aplikace umožnění začít odevzdávat řešení. Utvořený tým požádá vyučujícího prostřednictvím aplikace o schválení, vyučující může žádost přijmout nebo odmítnout s textovou poznámkou. Od okamžiku schválení ztrácí studenti možnost spravovat složení týmu, tato agenda přechází na odpovědného vyučujícího.
  **Priorita:** should have
  **Splněno:** částečně
  **Náročnost:** -

- **FP-NUR-2 – Studentské testování**
  Vyučující může v konfiguraci předmětu ve smyslu FP-GEN-2 povolit studentské testování. Týmy pak mohou potvrzovat ostatním studentům účast na uživatelském testování jejich projektu, za což studenti obdrží body dle konfigurace. Konfigurace také stanoví maximální počet testerů na projekt a maximální počet projektů testovaných jedním studentem v rámci předmětu.
  **Priorita:** should have
  **Splněno:** s výhradami
  **Náročnost:** vysoká

- **FP-SP-1 – Přístup do předmětu pro vyučující**
  Správce předmětu může v aplikaci do existujícího předmětu přidat i jiné vyučující, než kteří byli importováni z datového zdroje. Vyučujícím může správce učinit libovolného uživatele kromě studentů daného předmětu. Správce také může v aplikaci povolit, aby mu mohli uživatelé posílat žádosti o roli vyučujícího v daném předmětu. Po přijetí žádosti správcem nebo přímém přiřazení role správcem se uživatel stává plnohodnotným vyučujícím předmětu na úrovni vyučujících importovaných z datového zdroje.
  **Priorita:** must have
  **Splněno:** ne
  **Náročnost:** střední

- **FP-SP-2 – Asistent**
  Vyučující odpovědný za tým může do týmu přidat další uživatele v roli asistenta. Asistentem se může stát libovolný uživatel kromě studentů daného předmětu. Asistent může v aplikaci zobrazit informace týkající se daného týmu, především zadání, složení týmu, odevzdaná řešení a jejich hodnocení. Asistent nemůže v týmu provádět žádné akce. Asistent nemá přístup do ostatních částí sekce daného předmětu, pokud není zároveň jeho vyučujícím.
  **Priorita:** could have
  **Splněno:** ne
  **Náročnost:** střední

- **FP-SP-3 – Vlastní role členů týmu**
  Uživatel odpovědný za tým může v týmu definovat vlastní role členů týmu a tyto role jim libovolně přiřazovat. V definici uvede název role a požadovaný počet členů. Studenti mohou při odeslání o členství v týmu požádat o konkrétní roli dle vlastní preference.
  **Priorita:** should have

**Splněno:** ne
**Náročnost:** střední

- **FP-SP-4 – Přerozdělení bodů**
  Vyučující může v konfiguraci předmětu ve smyslu FP-GEN-2 povolit přerozdělení bodů a nastavit maximální procento přerozdělení. Přerozdělení pak může povolit či zakázat u jednotlivých kontrolních bodů. Studenti pak při odevzdání řešení mohou navrhnout přerozdělení bodů v procentech. Jeden student může získat či ztratit maximálně tolik procent, kolik udává konfigurace. Celkový součet přerozdělení je vždy nulový. Aplikace neumožní navrhnout přerozdělení, které tyto dva body nesplňuje. Vyučující může navržené přerozdělení během procesu hodnocení ve smyslu FP-GEN-8 upravit. Přerozdělení bodů je zohledněno na všech místech aplikace, kde je zobrazeno bodové hodnocení jednotlivých studentů.
  **Priorita:** must have
  **Splněno:** ne
  **Náročnost:** střední

#### 1.2.5.2 Nefunkční požadavky

Z analýzy nevyplynuly žádné nové nefunkční požadavky. Pro úplnost zde uvádím aktualizovaný seznam nefunkčních požadavků, který jsem vypracoval v rámci bakalářské práce před zahájením návrhu a implementace první verze SOS.

- **NP-1 – Dostupnost**
  Aplikace je dostupná 24 hodin denně, 7 dní v týdnu.
  **Kategorie:** spolehlivost
  **Priorita:** must have
  **Splněno:** ano

- **NP-2 – Oprávnění pro přístup**
  Aplikace a její data jsou dostupná pouze oprávněným uživatelům. Autentikace a autorizace uživatelů je relaizována protokolem OAuth 2.0 s využitím vlastního OAAS FIT ČVUT nebo Google OAuth.
  **Kategorie:** spolehlivost
  **Priorita:** must have
  **Splněno:** ano

- **NP-3 – Ovládání aplikace**
  Aplikaci je možné ovládat pomocí grafického uživatelského rozhraní v běžném webovém prohlížeči.
  **Kategorie:** použitelnost
  **Priorita:** must have
  **Splněno:** ano

- **NP-4 – Responzivita**
  Aplikace má responzivní uživatelské rozhraní, je ji tedy možné pohodlně používat jak na běžném počítači, tak i na mobilních zařízeních.
  **Kategorie:** použitelnost
  **Priorita:** should have
  **Splněno:** ano

- **NP-5 – Jazyková lokalizace**
  Uživatelské rozhraní aplikace je dostupné v českém a anglickém jazyce.
  **Kategorie:** použitelnost
  **Priorita:** should have
  **Splněno:** ano

**Tabulka 1.1** Přehled splnění požadavků současnou implementací SOS

| Požadavek | Splněno | Poznámka |
|---|---|---|
| FP-GEN-1 | ano | - |
| FP-GEN-2 | částečně | Konfigurace předmětů je aktuálně možná na úrovni celého předmětu nebo vybraného vyučujícího. Ta se pak aplikuje na všechny paralelky za které odpovídá. |
| FP-GEN-3 | ne | - |
| FP-GEN-4 | ano | - |
| FP-GEN-5 | ano | - |
| FP-GEN-6 | ne | - |
| FP-GEN-7 | ano | - |
| FP-GEN-8 | ano | - |
| FP-GEN-9 | ne | - |
| FP-GEN-10 | ano | - |
| FP-NUR-1 | částečně | Aktuálně vyučující schvaluje studenty vytvořená zadání, čímž je splněno že studenti nemohou před schválením odevzdávat řešení. Po schválení mohou však nadále upravovat složení týmu. |
| FP-NUR-2 | s výhradami | Studentské testování je implementováno dle definice požadavku, potvrzení účasti na testování se však vztahuje ke konkrétnímu odevzdanému řešení, nikoliv k celému projektu (viz 1.2.3.4). |
| FP-SP-1 | ne | - |
| FP-SP-2 | ne | - |
| FP-SP-3 | ne | - |
| FP-SP-4 | ne | - |
| NP-1 | ano | - |
| NP-2 | ano | - |
| NP-3 | ano | - |
| NP-4 | ano | - |
| NP-5 | ano | - |

#### 1.2.5.3 Splnění požadavků současnou aplikací

Tabulka 1.1 shrnuje splnění výše uvedených požadavků současnou implementací[7] SOS. Tabulka 1.2 poskytuje přehled o počtech splněných požadavků v daných kategoriích a celkově. Z celkových 21 požadavků je aktuálně 11 splněných, 3 jsou splněny částečně nebo s výhradami a 7 požadavků není splněno vůbec. Rozšíření SOS vedoucí ke splnění všech požadavků je předmětem následujících kapitol Návrh a Realizace.

## 1.3 Technologie

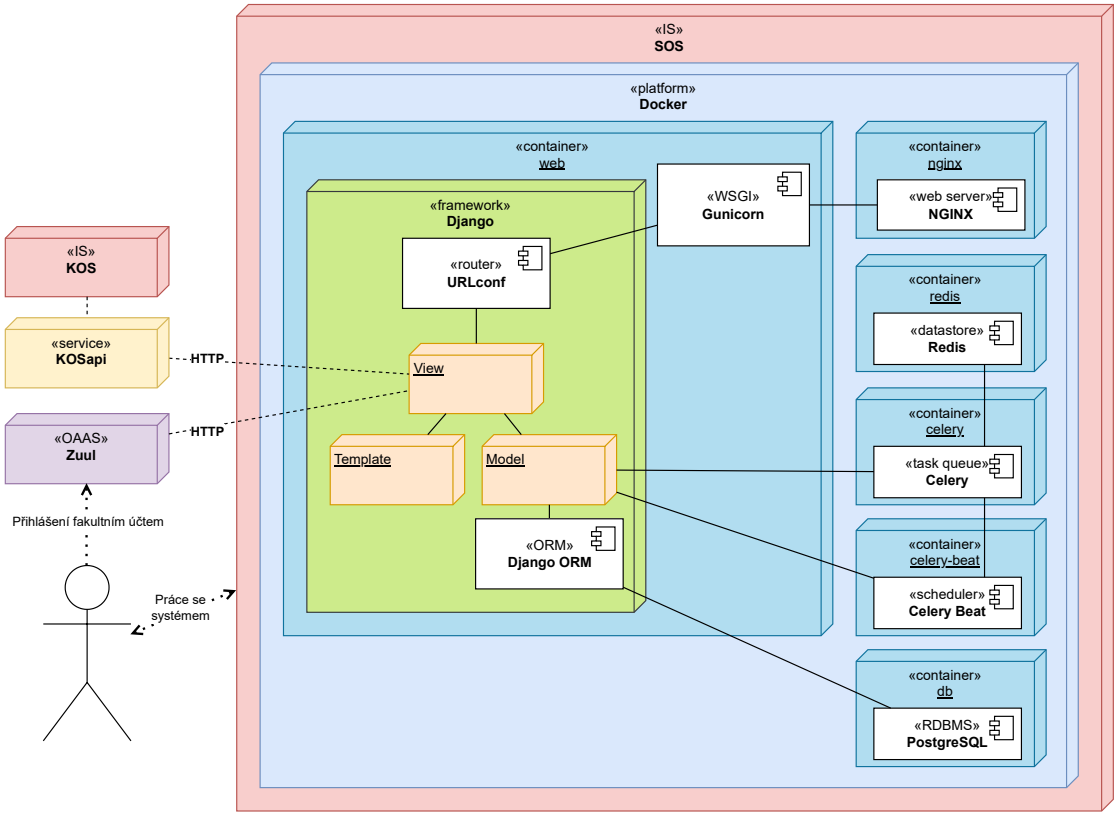
V této sekci shrnuji, na jakých technologiích je současná verze SOS postavena a k čemu každá z nich v rámci systému slouží. Z požadavků na rozšíření zatím nevyplynula potřeba žádnou stávající technologii vyměnit ani začít využívat nějakou další. Kromě popisů jednotlivých položek ilustruje architekturu celého systému diagram na obrázku 1.1.

[7] Současnou nebo původní implementací zde i v dalších částech práce myslím verzi SOS po úpravách v rámci bakalářské práce Maxe Hejdy. Tato verze byla na FIT ČVUT v provozu od zimního semestru 2022/23 a využil ji předmět NI-NUR.

**Tabulka 1.2** Statistika splnění požadavků současnou implementací SOS

| | Ano | Ne | Částečně / s výhradami |
|---|---|---|---|
| FP-GEN-* | 6 | 3 | 1 |
| FP-NUR-* | 0 | 0 | 2 |
| FP-SP-* | 0 | 4 | 0 |
| NP-* | 6 | 0 | 0 |
| Celkem | 11 | 7 | 3 |

**Obrázek 1.1** Architektura současné implementace SOS



### 1.3.1 Python

Python je vysokoúrovňový univerzální programovací jazyk. Jedná se o jazyk interpretovaný, spuštění tedy typicky nepředchází kompilace. Zdrojový kód je přeložen to tzv. *byte-kódu* až při samotném spuštění. Za celý proces je zodpovědný program nazývaný *interpreter*. Python je dynamicky typovaný jazyk, programátor se tedy nemusí při vývoji přímo zabývat datovými typy. [10] Pro zvýšení přehlednosti kódu může ale být vhodné datové typy v kódu vyznačit, k čemuž slouží mimo jiné například balíček *typing*, který je součástí standardní knihovny [11]. Jazyk umožňuje jak procedurální přístup, tak i objektově orientované programování. Python je open-source a jeho vývoji se věnuje především americká nezisková organizace Python Software Foundation [12].

SOS používá Python ve verzi 3.9. Aktuálně je již k dispozici verze 3.11, která oproti 3.9 nabízí řadu nových funkcionalit a slibuje také výrazné zrychlení v řádech desítek procent. Bylo by tedy jistě vhodné na nejnovější verzi jazyka přejít, čímž bude rovněž zajištěna oficiální podpora a bezpečnostní aktualizace až přibližně do října roku 2027. [13]

### 1.3.2 Django

Django je framework pro tvorbu webových aplikací v programovacím jazyce Python. Implementuje architektonický vzor Model-View-Template, což je modifikace známějšího vzoru Model-View-Controller. Framework využívá vlastní ORM k propojení jednotlivých modelů v podobě tříd jazyka Python s tabulkami relační databáze. Django je kompatibilní s řadou databázových strojů, oficiálně jsou podporovány například PostgreSQL, Oracle a SQLite. Změny v datovém modelu lze provádět inkrementálně pomocí tzv. migrací, což vývoj značně usnadňuje. Framework nabízí i řadu dalších komponent, které SOS využívá, zejména serializační a validační systém pro formuláře, šablonovací systém, různé bezpečnostní prvky a nástroje pro realizaci autentizace a autorizace uživatelů. Součástí frameworku je také konfigurovatelné administrátorské prostředí [14].

Při použití frameworku Django je zvykem rozdělit zdrojový kód do tzv. *aplikací*, což jsou Python moduly sdružující veškerý kód související s nějakou logickou částí systému – datový model, aplikační logiku, formuláře, šablony, jednotkové testy a další. SOS toto dodržuje, součástí zdrojových kódů jsou například aplikace *courses*, *submissions*, *users* a *teams*.

Fungování frameworku Django a konkrétnímu využití jeho komponent v SOS se detailněji věnuje má bakalářská práce [1], především její třetí kapitola Realizace.

#### 1.3.2.1 Verze a oficiální podpora

Django ve své dokumentaci uvádí, že používá uvolněnou formu *semantického verzování*[8]. Jednotlivá vydání jsou číslována ve formátu „A.B“ nebo „A.B.C“. Vydání „A.B“ obsahuje nové funkcionality, „A.B.C“ pak pouze opravy chyb a bezpečnostní aktualizace. Každé vydání „X.2“ je tzv. *LTS*, tedy vydání s prodlouženou podporou (z anglického *long-term-support*). Tato vydání typicky dostávají bezpečnostní aktualizace po dobu následujících tří let po jejich uvedení. Jako další vydání po LTS následuje vždy „Y.0“. [15]

Současný SOS používá Django ve verzi 3.2 LTS, která byla vydána během vývoje první verze SOS. Oficiální podpora této verze má skončit v první třetině roku 2024. Vydání verze 4.2 LTS je očekáváno přibližně začátkem dubna 2023, bylo by tedy vhodné SOS na tuto verzi aktualizovat jakmile bude k dispozici, čímž bude zajištěna oficiální podpora minimálně do první třetiny roku 2026. [16]

[8] `https://semver.org/`

### 1.3.3 PostgreSQL

K realizaci persistence SOS využívá databázový stroj PostgreSQL[9]. Jedná se o open-source technologii, která vyniká svou stabilitou, množstvím dostupných funkcionalit a dlouhodobě velmi dobrou kompatibilitou s frameworkem Django [17].

Veškerá práce s databází na aplikační úrovni nicméně probíhá prostřednictvím již zmíněného Django ORM, což je v souladu s oficiálním doporučením z dokumentace frameworku a mimo jiné je tak umožněno technologii PostgreSQL v případě potřeby vyměnit za jinou bez nutnosti upravovat kód aplikace.

### 1.3.4 Celery

Celery je asynchronní úkolová fronta se zaměřením na zpracování úkolů v reálném čase, zároveň ale podporuje i plánování úkolů. Jedná se o open-source technologii vyvinutou v jazyce Python [18]. SOS ji využívá k realizaci automatického odesílání e-mailových notifikací o blížících se termínech odevzdání. Jako zprostředkovatele zpráv Celery v SOS využívá key-value databázi Redis[10].

### 1.3.5 NGINX

NGINX je minimalistický open-source webový server, který podporuje *load-management* a *reverzní proxy* [19]. NGINX oproti konkurenčnímu Apache nedokáže poskytovat dynamický obsah, což ale u SOS řeší technologie Gunicorn, která je standartem při nasazovaní Django aplikací a její použití je popsáno v oficiální dokumentaci [20]. NGINX tak klientovi přímo vrací pouze statické soubory, v ostatních případech se zachová jako reverzní proxy a požadavek přepošle na Gunicorn socket, kde je zpracován na aplikační úrovni.

### 1.3.6 Docker a Docker Compose

Docker je nástroj pro virtualizaci na úrovni operačního systému. Klasická virtualizace v podobě tzv. *virtuálních strojů* spočívá v abstrakci hardware, virtuální stroje pak v sobě obsahují kompletní operační systém a další potřebné položky. Docker oproti tomu využívá tzv. *kontejnery*, což je abstrakce aplikační vrstvy – kontejner obsahuje software a jeho závislosti, ale jádro operačního systému je sdíleno s hostitelským operačním systémem. Kontejnery zajišťují izolované a standardizované prostředí pro běh aplikace, ale s řádově nižším využitím prostředků, než je tomu v případě virtuálních strojů. [21]

Docker Compose je nástroj pro orchestraci více Docker kontejnerů do jednoho funkčního celku. K definici a konfiguraci orchestrace využívá konfigurační soubor ve formátu YAML. [22]

SOS se aktuálně skládá z šesti kontejnerů:

- **nginx** – webový server NGINX, požadavky na dynamický obsah přeposílá na *web*,
- **web** – zde běží Gunicorn a Django aplikace,
- **db** – PostgreSQL databáze,
- **celery** – *worker* proces Celery,
- **celery-beat** – *beat* proces Celery – slouží k rozesílání úkolů,
- **redis** – key-value databáze Redis pro Celery.

[9] `https://www.postgresql.org/`
[10] `https://redis.io/`

### 1.3.7 KOS a KOSapi

Komponenta Studium (dále jen KOS) je systém pro organizaci a správu výuky provozovaný Výpočetním a informačním centrem ČVUT. Mezi jeho klíčové funkce patří mimo jiné tvorba a evidence předmětů, zápis předmětů, tvorba rozvrhů a záznam studijních výsledků. [23]

KOSapi poskytuje aplikační rozhraní v podobně RESTful webových služeb, které zprostředkovává přístup k vybrané části dat v databázi KOS. Umožňuje a podporuje vznik školních i studentských aplikací, které pro svou činnost vyžadují online aplikační přístup k datům souvisejícím s výukou. [24]

SOS aktuálně využívá KOSapi k získávání informací o uživatelích, vyučovaných předmětech a rozvrzích. Tato data slouží především k importu nových předmětů do SOS garantem předmětu. Vyučující projevili zájem také o funkcionalitu exportu hodnocení ze SOS do KOS, nicméně KOSapi v této chvíli nic takového neumožňuje a nic nenasvědčuje tomu, že by se na tom mělo v blízké budoucnosti cokoliv změnit.

Export hodnocení ze systému Moodle do KOS může být v současnosti realizován například pomocí skriptu v jazyce JavaScript formou tzv. *bookmarklets* [25]. Podobné řešení používá i DBS portál[11]. SOS by mohl export hodnocení do KOS prozatím realizovat obdobně, s tím, že robustnější řešení bude vyvinuto, pokud KOSapi v budoucnu umožní kromě čtení dat i jejich zápis.

### 1.3.8 Bakaláři a Datový konektor

Systém Bakaláři je určen pro podporu administrativy základních a středních škol v České republice. Jedná se o nejrozšířenější řešení na trhu, dle tvůrců jej používá přes 60 % všech škol v ČR a přes milion uživatelů. [26] Nabízí mimo jiné evidenci žáků a zaměstnanců, internetovou žákovskou knížku, rozvrh hodin a elektronickou třídní knihu. Na školách tak může plnit obdobnou funkci jako KOS na FIT ČVUT.

SOS dokáže jako datový zdroj použít kromě KOS také systém Bakaláři, byl k tomu uzpůsoben v rámci nasazení na GJGJ. Součástí webové aplikace Bakaláři je komponenta *Datový konektor*, dle její dokumentace se jedná o RESTful webovou službu. [27] V současnosti SOS využívá Datový konektor pouze ke čtení dat, stejně jako KOSapi v konfiguraci pro FIT ČVUT.

Dle dokumentace Datového konektoru je určitým způsobem možné pomocí něj do systému Bakaláři zapisovat známky. Dostupná dokumentace je v tomto ohledu ale velmi stručná a byla by zde nutná další analýza, zřejmě ve spolupráci se společností, která systém Bakaláři vyvíjí a distribuuje. Dle informací od Hejdy navíc o tuto funkcionalitu na GJGJ, kde je SOS aktuálně provozován, nebyl zájem.

### 1.3.9 OAuth

K ověřování totožnosti uživatelů používá SOS protokol OAuth 2.0. Jedná se o moderní autorizační protokol, jehož hlavní výhoda tkví v tom, že uživatel může klientské aplikaci (v tomto případě SOS) poskytnout přístup k jeho datům v jiné službě (v tomto případě KOSapi nebo GSuite), aniž by dané aplikaci musel vyzradit své přistupové údaje do této služby. Klientská aplikace se může autentizovat buď sama za sebe, nebo za jejího uživatele, který k tomu dá explicitní souhlas. Protokol umožňuje vymezit konkrétní pravomoci jednotlivých klientských aplikací. Klientská aplikace se může také v otázce ověřování identity uživatelů spolehnout na OAuth autorizační server (dále jen OAAS) služby a nemusí tak sama vůbec řešit správu přihlašovacích údajů, což přispívá k celkové bezpečnosti. [28]

FIT ČVUT provozuje vlastní OAAS, který je open-source a jmenuje se Zuul OAAS. Jedná se o samostatný autorizační server, který není součástí žádné služby, ale je sdílený mezi řadou

[11] `https://dbs.fit.cvut.cz`

služeb provozovaných na fakultě. [29] Tento OAAS využívá i SOS v konfiguraci pro FIT ČVUT. Na GJGJ je pro tyto účely SOS integrován s Google OAuth[12].

### 1.3.10 Git a GitLab

Pro správu verzí SOS používá Git[13] a GitLab[14]. GitLab je systém pro správu Git repozitářů, který současně poskytuje řadu dalších souvisejících funkcionalit, jako je například Issue Tracker nebo nástroje pro realizaci CI/CD. FIT ČVUT provozuje *self-hosted* variantu systému GitLab, která je dostupná na adrese `https://gitlab.fit.cvut.cz`. Zde se nachází i repozitář SOS.

### 1.3.11 Sentry

Sentry[15] je služba pro monitorování chyb. FIT ČVUT ji provozuje v *self-hosted* variantě na adrese `https://sentry.fit.cvut.cz`. Integrace Sentry s Django aplikací je poměrně přímočará s využitím poskytovaného Python modulu Sentry SDK a je přímo popsána v dokumentaci Sentry.

Využití Sentry se během provozu SOS ukázalo jako mimořádně užitečné. Každá neošetřená výjimka je v Sentry zaznamenána a správce je na novou událost upozorněn e-mailovou notifikací. Ve webovém rozhraní Sentry je pak k dispozici podrobný záznam o události s řadou informací týkajících se prostředí a stavu aplikace. Díky tomu jsem měl od nasazení SOS do provozu o všech takových událostech okamžitý přehled a mohl je aktivně řešit.

## 1.4 Zaznamenávání studijních výsledků

Systém SOS poskytuje studentům a vyučujícím přehled o bodových ziscích za semestrální práce realizované jeho prostřednictvím. V některých předmětech vyučovaných na FIT ČVUT, jako například BI-SP1 a BI-SP2, je semestrální práce jedinou položkou celkového hodnocení studenta. V jiných předmětech může hodnocení zahrnovat i další položky, zejména zkoušku a případně i další práci během semestru.

### 1.4.1 KOS

Oficiálně na FIT ČVUT slouží k zaznamenávání studijních výsledků KOS [23]. Vyučující tedy musí na konci semestru zpracovat bodové zisky studentů, na jejich základě určit známku a tu zapsat do KOS. KOS v současnosti neposkytuje možnosti, jak toto udělat strojově, vyučující si tedy musí vystačit s uživatelským rozhraním webové aplikace. Jak jsem již zmínil v předchozí sekci, KOSapi sice umožňuje získávat data z KOS, není ale nástrojem k jejich zápisu.

Současný SOS shrnuje pro vyučujícího bodové zisky studentů na obrazovce přehledu klasifikace. Zde se nachází tabulka obsahující mimo jiné jména studentů a jejich celkový bodový zisk. Vyučující pak tuto tabulku musí celou projít, sám určit známky a ty zapsat do KOS. Bylo by vhodné, aby SOS vyučujícímu tento proces usnadnil, neboť může být při větším počtu studentů poměrně zdlouhavý a také náchylný k chybám. Konkrétnímu řešení se bude věnovat kapitola Návrh.

[12] `https://developers.google.com/identity/protocols/oauth2`
[13] `https://git-scm.com/`
[14] `https://about.gitlab.com/`
[15] `https://sentry.io/`

### 1.4.2 FIT Klasifikace

Kromě SOS se v některých předmětech na FIT ČVUT zaznamenávají studijní výsledky i do aplikace FIT Klasifikace[16] (dále jen Klasifikace). Jedná se o webovou aplikaci, která umožňuje podrobnější zaznamenávání studijních výsledků, než jen zápočet a celkovou známku. Vyučující zde mohou definovat jednotlivé položky hodnocení, například domácí úkoly, testy v průběhu semestru, semestrální práci či závěrečnou zkoušku. U těchto položek pak definují i možné bodové zisky a minimální hranici, které je nutné dosáhnout k získání zápočtu a absolvování předmětu. V průběhu semestru potom do aplikace zadávají hodnocení jednotlivých studentů. Pro studenty tato aplikace nabízí přehled hodnocení včetně známek ve všech předmětech, které ji využívají a také podrobný výpis položek hodnocení v jednotlivých předmětech.

Nabízí se tedy, aby SOS umožňoval export hodnocení i do aplikace Klasifikace. Ta k tomuto účelu poskytuje REST API, které umožňuje i zápis dat. [30] Návrhu konkrétního řešení se taktéž věnuje následující kapitola Návrh.

### 1.4.3 Bakaláři

V prostředí gymnázia slouží k zaznamenávání studijních výsledků systém Bakaláři. Jak jsem již zmínil, Datový konektor by dle dostupných informací měl umožňovat i zápis dat a mohlo by být možné výsledky ze SOS do systému Bakaláři nějakým způsobem exportovat. Nicméně ze strany GJGJ o tuto funkcionalitu dle mých informací zatím nebyl projeven zájem. Navíc jsem přesvědčen, že by implementaci této funkcionality provedl oproti mě kvalitněji například Hejda, který již řešil import dat z tohoto systému a má s ním tedy zkušenosti. Ve své práci se tedy budu dále zabývat pouze exportem do KOS a Klasifikace, s tím že řešení by mělo být v rámci možností co nejvíce univerzální a rozšiřitelné.

### 1.4.4 Způsoby zakončení předmětů na FIT ČVUT

Předměty vyučované na FIT ČVUT můžeme z hlediska hodnocení a zakončení rozdělit do následujících kategorií [31]:

- Zápočet – předmět je zakončen získáním zápočtu, není hodnocen známkou.
- Zkouška – předmět je zakončen pouze zkouškou.
- Klasifikovaný zápočet – předmět je zakončen získáním zápočtu, známka je určena dle bodového zisku v průběhu semestru.
- Zápočet a zkouška – předmět je zakončen zkouškou, přihlášení ke zkoušce může být podmíněno předchozím získáním zápočtu. Známka je určena dle bodového zisku v průběhu semestru a zkoušky.

Předměty prvních dvou kategorií nejsou předměty, na které SOS primárně cílí, nebudu se jimi proto dále zabývat. Předměty ostatních kategorií mohou SOS využít, pokud je součástí jejich hodnocení semestrální práce. Ta může být pouze jednou z několika položek hodnocení, nebo jedinou položkou. Toto bude třeba zohlednit při návrhu exportu do KOS a Klasifikace. Zejména v prvním případě nejspíš nebude možné exportovat hodnocení z SOS přímo do KOS, vhodnější bude využít jako mezikrok aplikaci Klasifikace, která sama (neoficiálně) umožňuje následný export hodnocení do KOS.

[16] `https://grades.fit.cvut.cz`

## 1.5 Shrnutí

V úvodu kapitoly jsem analyzoval existující systém SOS a jeho vývoj v čase – první verzi vzešlou z mé bakalářské práce, úpravy související s jeho nasazením v předmětu NI-NUR od zimního semestru 2021/22, výsledky semestrální práce z předmětu NI-NUR, kde jsme se s kolegy věnovali návrhu nového uživatelského rozhraní a konečně také rozšíření SOS pro využití na gymnáziích realizované Hejdou v rámci jeho bakalářské práce.

Následně jsem se věnoval analýze požadavků, z níž vzešel seznam šestnácti funkčních a pěti nefunkčních požadavků. Zhodnotil jsem splnění těchto požadavků současnou verzí SOS a identifikoval nedostatky současného řešení a chybějící funkcionality. Návrh a implementace potřebných úprav a rozšíření bude předmětem dalších kapitol.

Dále jsem analyzoval architekturu současného SOS a popsal jednotlivé technologie, které SOS používá. Výstupem této analýzy je i diagram popisující architekturu SOS, komunikaci jeho vnitřních komponent a integraci s externími systémy v prostředí FIT ČVUT.

Nakonec jsem se zabýval systémy pro zaznamenávání studijních výsledků používanými na FIT ČVUT a GJGJ – KOS, FIT Klasifikace a Bakaláři – a možnostmi další integrace SOS s těmito systémy. Rozhodl jsem se přidat do SOS funkcionalitu exportu hodnocení do KOS a FIT Klasifikace, detailnější rozbor možností a návrh konkrétního řešení popíšu v následující kapitole.

# Kapitola 2
# Návrh

*V této kapitole rozebírám změny, které je potřeba v SOS učinit, aby byly splněny všechny funkční požadavky popsané v předchozí kapitole. Popisy změn navržených k naplnění jednotlivých požadavků zahrnují úpravy a rozšíření datového modelu, aplikační logiky i uživatelského rozhraní. Druhá část kapitoly se věnuje exportu hodnocení do KOS a Klasifikace. Navrhuji celý proces exportu do obou systémů, zabývám se možnými způsoby jak jej realizovat a popisuji zvolené řešení z pohledu aplikační logiky i uživatelského rozhraní.*

## 2.1 Splnění funkčních požadavků

Z analýzy vzešlo celkem deset funkčních požadavků, které současnou implementací nejsou splněny vůbec, nebo jen částečně. Tato sekce se věnuje návrhu úprav potřebných ke splnění těchto funkčních požadavků.

Součástí této sekce je několik třídních diagramů. Jedná se o zjednodušené diagramy, ze kterých jsou v zájmu přehlednosti vypuštěny některé vztahy a atributy nesouvisející s demonstrovaným problémem. Navržené změny jsou vyznačeny tučným písmem a silnějšími liniemi.

### 2.1.1 Žádosti o přístup do předmětu

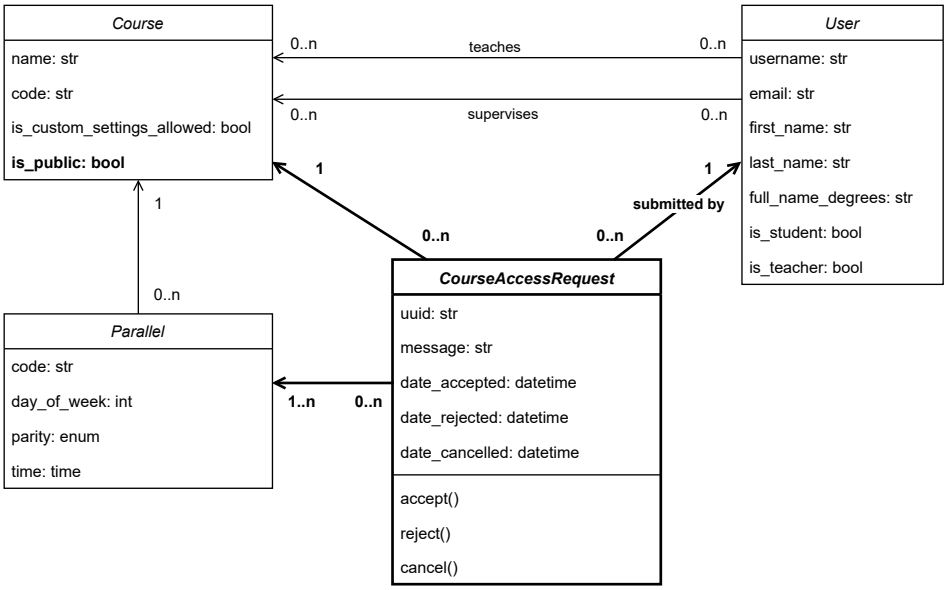
**Požadavek:** FP-SP-1 – Přístup do předmětu pro vyučující

Správce předmětu nově může ostatním uživatelům umožnit odeslat žádost o přístup do předmětu. Tuto žádost může odesílatel opět zrušit. Správce předmětu může žádost odmítnout nebo přijmout, v tom případě se odesílatel žádosti stane dalším vyučujícím daného předmětu. Před odesláním žádosti uživatel specifikuje, ke kterým paralelkám si přeje být přiřazen.

#### 2.1.1.1 Změny modelu

Diagram tříd na obrázku 2.1 znázorňuje ztahy mezi předměty, paralelkami a vyučujícími a navržené úpravy modelu. Hlavní změnou je vytvoření nové třídy `CourseAccessRequest`, která představuje žádost o získání přístupu do předmětu. Tuto žádost odesílá uživatel, který se chce stát vyučujícím daného předmětu, později ji může zrušit. Správce předmětu může žádost přijmout nebo odmítnout. Žádost také odkazuje na minimálně jednu paralelku, ke které má být nový vyučující přiřazen. Také přibyl atribut `is_public` u třídy `Course`, který značí, zda mohou uživatelé vytvářet zmíněné žádosti.

**Obrázek 2.1** Schéma vztahu předmětů a vyučujících s žádostí o přístup




### 2.1.1.2 Změny uživatelského rozhraní

Povolení odesílání žádostí správce předmětu provede na hlavní stránce konfigurace předmětu. Uživatelům, kteří mohou žádost odeslat se předmět zobrazí na domovské stránce vyučujícího v sekci „Další dostupné předměty“. Kliknutím na předmět z nabídky se dostanou na formulář odeslání žádosti o přístup, případně zde bude zobrazena informace o čekající žádosti.

Správce předmětu může také nového vyučujícího přidat do předmětu přímo. Učiní tak rovněž na hlavní stránce konfigurace předmětu, kde zadá uživatelské jméno nového vyučujícího a seznam paralelek, ke kterým má být přiřazen. Po potvrzení systém vyhledá uživatele ve své databázi, případně v datovém zdroji a přiřadí jej k předmětu jako vyučujícího.

## 2.1.2 Nastavení předmětu na úrovni týmu

**Požadavek:** FP-GEN-2 – Konfigurace předmětů

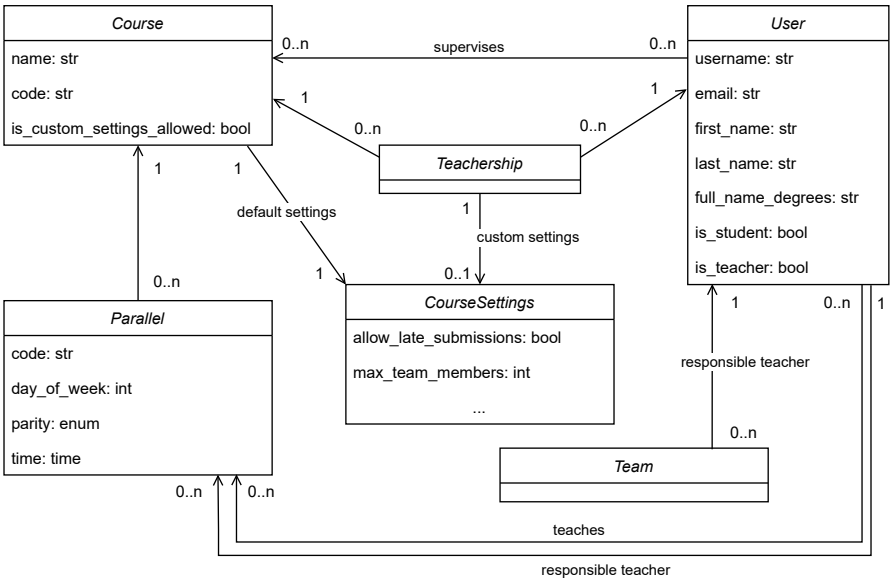
Aktuálně může správce předmětu povolit ostatním vyučujícím, aby mohli vytvářet vlastní konfigurace předmětu. Tyto se pak aplikují na všechny paralelky, za které daný vyučující odpovídá. Nově správce určí, zda má být konfigurace definována vzhledem k celému předmětu, k paralelkám nebo k týmům. Ve druhém a třetím případě je na jednotlivé studenty aplikováno nastavení paralelky ke které patří, resp. jejich týmu v případě, že nějaký mají. Pokud ne, aplikuje se výchozí nastavení předmětu, které vždy určuje jeho správce.

### 2.1.2.1 Změny modelu

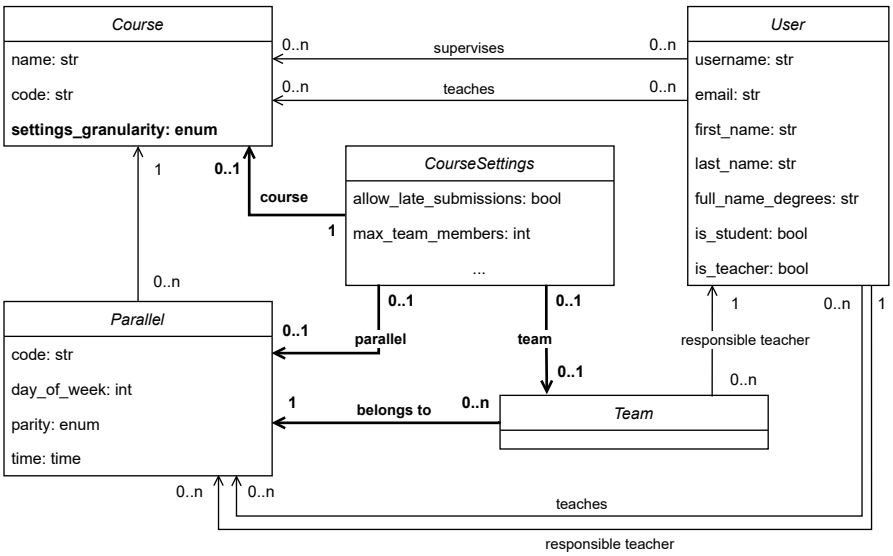
Současný model je vyjádřen diagramem tříd na obrázku 2.2. Třída `Course` obsahuje atribut `is_custom_settings_allowed` určující, zda je vytváření vlastních nastavení povoleno či nikoliv.

Tento atribut je nově nahrazen atributem `settings_granularity`, který může nabývat hodnot `COURSE`, `PARALLEL` a `TEAM` a určuje, které `CourseSettings` mají být uvažovány. Každý předmět má vlastní konfiguraci, která je uvažována v případě hodnoty `COURSE` a v ostatních případech slouží jako výchozí hodnota. V případě nastavení hodnoty `PARALLEL` jsou kopírováním konfigurace předmětu vytvořeny `CourseSettings` pro každou paralelku. Upravovat je může vyučující odpovědný za danou paralelku. V případě hodnoty `TEAM` jsou kopírováním konfigurace předmětu

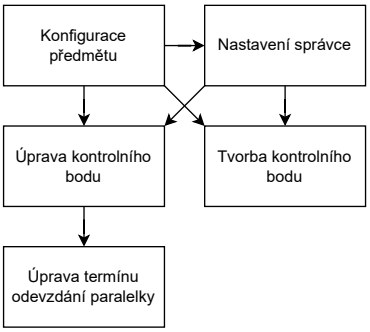
**Obrázek 2.2** Aktuální schéma vztahu předmětů a konfigurací



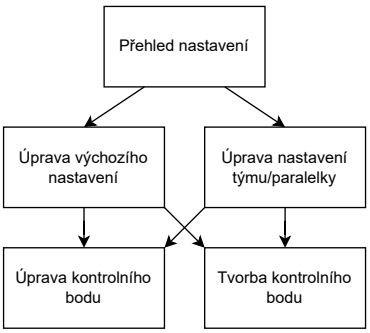

**Obrázek 2.3** Upravené schéma vztahu předmětů a konfigurací



**Obrázek 2.4** Aktuální mapa obrazovek konfigurace předmětu




**Obrázek 2.5** Nová mapa obrazovek konfigurace předmětu



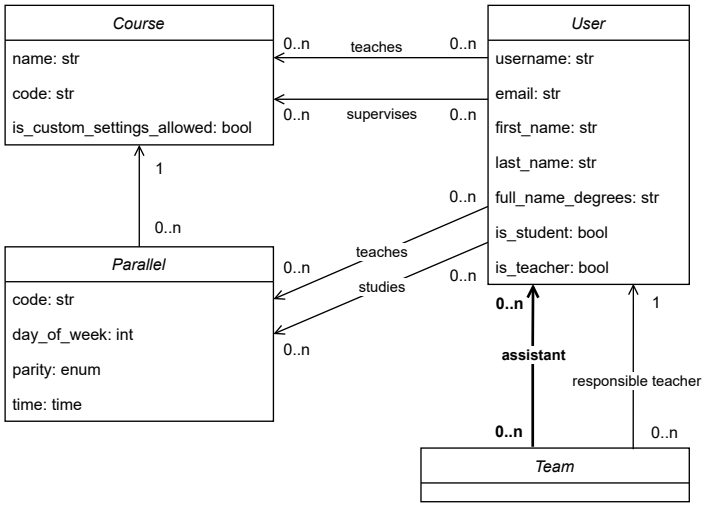

či paralelky (v závislosti na předchozí hodnotě) vytvořeny `CourseSettings` pro každý tým, které může upravovat vyučující odpovědný za daný tým. Změny modelu jsou zachyceny diagramem tříd na obrázku 2.3.

#### 2.1.2.2 Změny uživatelského rozhraní

Aktuální uživatelské rozhraní konfigurace předmětu shrnuje mapa obrazovek na obrázku 2.4. Její první obrazovka je „Konfigurace předmětu“, na které může vyučující editovat své vlastní nastavení, případně zde vidí výchozí hodnoty, není-li vlastní nastavení povoleno správcem. Z této stránky může vyučující přejít na obrazovky sloužící k tvorbě nebo úpravě kontrolního bodu. Z obrazovky „Úprava kontrolního bodu“ může kliknutím na položku v seznamu paralelek na této obrazovce přejít ještě na obrazovku „Úprava termínu odevzdání paralelky“. Správce předmětu může z první obrazovky přejít na „Nastavení správce“, což je obrazovka téměř totožná s obrazovkou vlastního nastavení, jen je místo toho upravováno výchozí nastavení předmětu. Jsou zde také navíc karty s dalšími možnostmi, které se týkají se celého předmětu.

Uživatelské rozhraní konfigurace bude stejně jako model přepracováno, jak ukazuje mapa obrazovek na obrázku 2.5. Hlavní modelovou změnou je, že konfigurace specifické pro jednotlivé vyučující jsou nahrazeny konfiguracemi pro konkrétní týmy nebo paralelky. To musí reflektovat i uživatelské rozhraní. Nově bude první obrazovkou konfigurace „Přehled nastavení“. Na této obrazovce uživatel uvidí výchozí nastavení předmětu a kontrolní body, které k němu patří. Dále zde uvidí seznam paralelek (resp. týmů), za které odpovídá a může editovat jejich konfigurace. Kliknutím na položku v seznamu přejde na obrazovku „Úprava nastavení týmu/paralelky“. V případě, že je uživatel správcem předmětu, může na první obrazovce také editovat obecná nastavení předmětu, jako například granularitu nastavení nebo seznam správců předmětu. Správce také z této obrazovky může přejít na obrazovku „Úprava výchozího nastavení“, která je shodná s nastavením paralelky nebo týmu, ale upravována je výchozí konfigurace předmětu. Z obou

**Obrázek 2.6** Schéma vztahu týmů a asistentů




zmíněných obrazovek může uživatel také přejít na úpravu nebo tvorbu kontrolního bodu, které zůstanou bez výraznějších změn. Poslední změnou je zrušení obrazovky „Úprava termínu odevzdání paralelky“. Ta již v novém schématu nedává smysl, jelikož paralelky mají nyní vlastní konfigurace a tedy i vlastní kontrolní body s termíny odevzdání.

### 2.1.3 Přístup k projektu pro další uživatele

**Požadavek:** FP-SP-2 – Asistent

Jedním z požadavků bylo umožnění povolení přístupu k vybranému projektu pro dalšího uživatele, který není vyučujícím odpovědným za tento projekt, případně není vyučujícím vůbec. Tento uživatel pouze nesmí být studentem předmětu, ke kterému projekt patří.

#### 2.1.3.1 Změny modelu

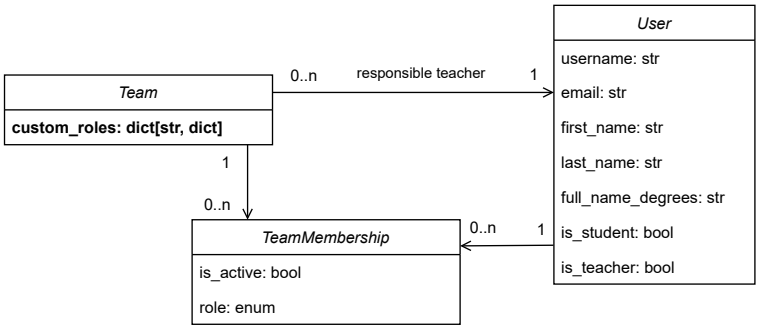
Změnu představuje přidání nové vazby mezi uživatelem a týmem, která značí, že daný uživatel je asistentem tohoto týmu. Uživatel může být asistentem libovolného počtu týmů a tým může mít libovolný počet asistentů. Jedinou podmínkou je, že uživatel nesmí být současně asistentem týmu a studentem paralelky, která patří ke stejnému předmětu jako daný tým. Vazba je znázorněna diagramem na obrázku 2.6.

#### 2.1.3.2 Změny uživatelského rozhraní

Na obrazovku správy týmu přibude karta „Asistenti“, která bude zobrazena pouze vyučujícímu. Zde bude k dispozici aktuální seznam asistentů a formulář pro přidání nového asistenta. Jména asistentů budou zobrazena na detailu týmu.

Z pohledu asistenta přibude seznam asistovaných projektů na domovské obrazovce. Asistent nezískává přístup do celého předmětu (pokud není zároveň jeho vyučujícím), pouze do konkrétních projektů. Kliknutím na položku v seznamu může tedy přejít na detail týmu a odtud se později vrátit zpět na domovskou obrazovku, na detail předmětu asistent přístup nemá. Z detailu projektu může přejít na další obrazovky související s projektem, zejména na detaily odevzdaných řešení.

**Obrázek 2.7** Schéma vztahů týmů, jejich členů a vyučujících




### 2.1.4 Vlastní role členů týmu

**Požadavek:** FP-SP-3 – Vlastní role členů týmu

Uživatel odpovědný za správu týmu, což je dle příslušné konfigurace buďto odpovědný vyučující, nebo vedoucí týmu z řad studentů, může nově definovat vlastní role pro členy týmu. U role uvede její název a požadovaný počet členů. Role si studenti volí při odesílání žádosti o členství v týmu, případně jim je může přiřadit uživatel odpovědný za správu týmu.

Vlastní role mají pouze informativní charakter, nesouvisejí s aplikační logikou ve smyslu oprávnění uživatelů ani validace při schvalování týmu či přijímání nového člena.

#### 2.1.4.1 Změny modelu

Definice rolí (název, požadovaný počet členů) a identifikátory uživatelů, kteří mají roli přiřazenou jsou uloženy v asociativním poli jako atribut třídy `Team`, jak znázorňuje diagram na obrázku 2.7. Kromě těchto vlastních rolí zůstávají paralelně zachovány i původní role, které mají odlišný význam. Původní atribut `role` třídy `TeamMembership` může nabývat hodnot `LEADER`, `MEMBER` a `TESTER` a definuje oprávnění uživatele vzhledem k danému týmu. Vlastní role jsou přiřazovány pouze členům s rolí `LEADER` nebo `MEMBER`.

#### 2.1.4.2 Změny uživatelského rozhraní

Tato funkcionalita bude realizována na obrazovce správy týmu. Zde přibude formulář, ve kterém může uživatel definovat nové role a upravovat existující. U role definuje název a požadovaný počet členů. Na této obrazovce se také nachází tabulka s členy týmu, do které bude přidán sloupec s volbami vlastních rolí. Změny ilustruje wireframe na obrázku 2.8. Realizovaná obrazovka pak bude obsahovat i další karty, které s funkcionalitou vlastních rolí nesouvisejí a z wireframe tedy byly vypuštěny.

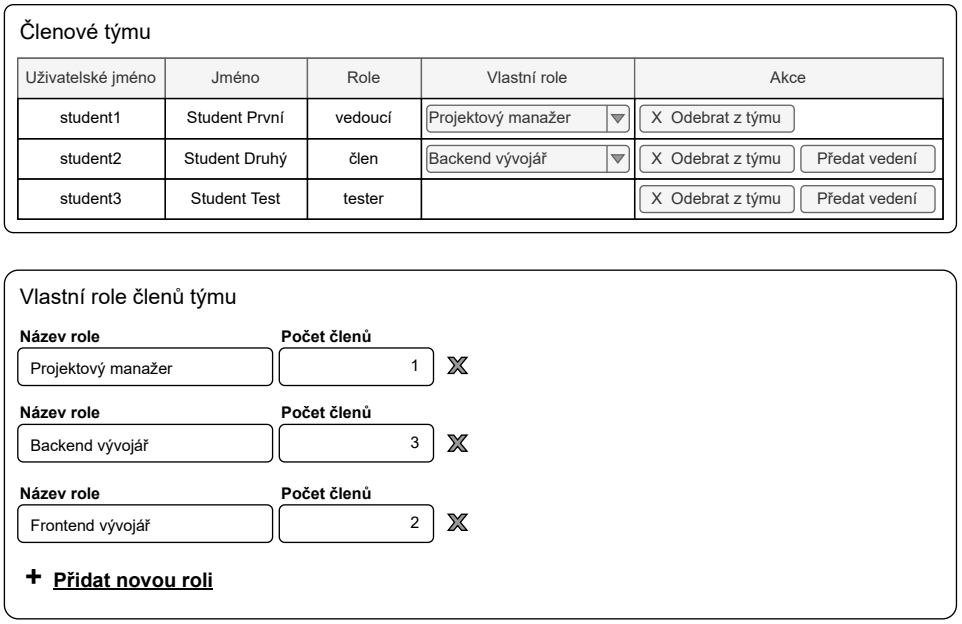
### 2.1.5 Schvalování týmů

**Požadavek:** FP-NUR-1 – Schvalování týmů

V současné implementaci je při vytvoření nového zadání studentem automaticky vytvořena žádost o schválení tohoto zadání, kterou může odpovědný vyučující přijmout nebo odmítnout. V případě odmítnutí může student zadání upravit a vytvořit novou žádost.

Tento koncept je nově zcela nahrazen schvalováním týmů jako takových. Jakmile má tým alespoň minimální počet členů, může student požádat o jeho schválení. Tím je vytvořena žádost o schválení týmu, kterou může odpovědný vyučující přijmout nebo odmítnout. V případě přijetí žádosti je tým schválen, což je navíc zaznamenáno přímo v atributu `date_approved` třídy `Team`. Na základě informace o schválení je dále rozhodováno, jestli je za správu týmu odpovědný student

**Obrázek 2.8** Částečný wireframe obrazovky správy týmu




nebo vyučující. Celá logika schvalování týmů je aktivní pouze pokud jsou dle konfigurace za správu týmů odpovědní studenti.

### 2.1.5.1 Změny modelu

Původní model je zachycen diagramem na obrázku 2.9. Navržené změny ukazuje diagram na obrázku 2.10. Jedná se o zrušení třídy `AssignmentApprovalRequest` a její nahrazení novou třídou `TeamApprovalRequest`. Součástí je také přidání pomocných metod do třídy `Team`.

### 2.1.5.2 Změny uživatelského rozhraní

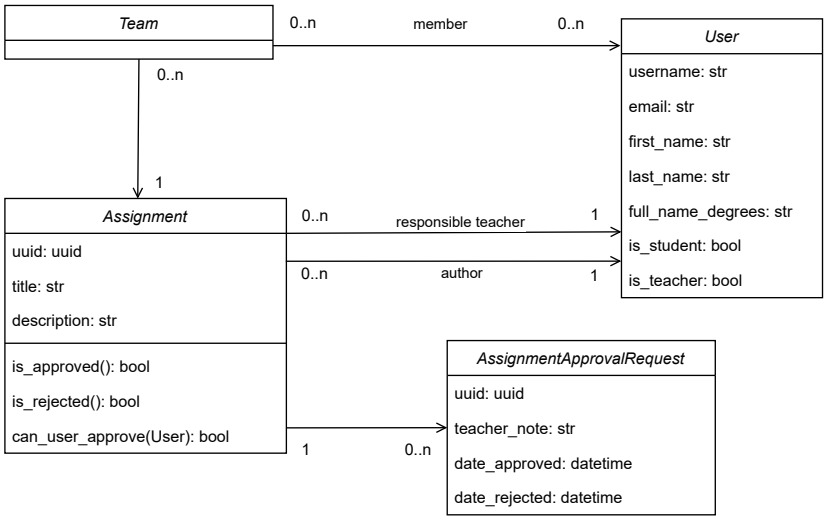
Pro tuto funkcionalitu není třeba výraznějších změn uživatelského rozhraní. Současné schvalování zadání probíhá na detailu týmu, může tedy být přímo nahrazeno schvalováním týmu jako takového pomocí obdobného formuláře. Nově ale k odeslání žádosti o schválení nebude docházet automaticky po vytvoření zadání/týmu. Student musí žádost odeslat sám, tak jako nyní v případě opakované žádosti.

## 2.1.6 Studentské testování

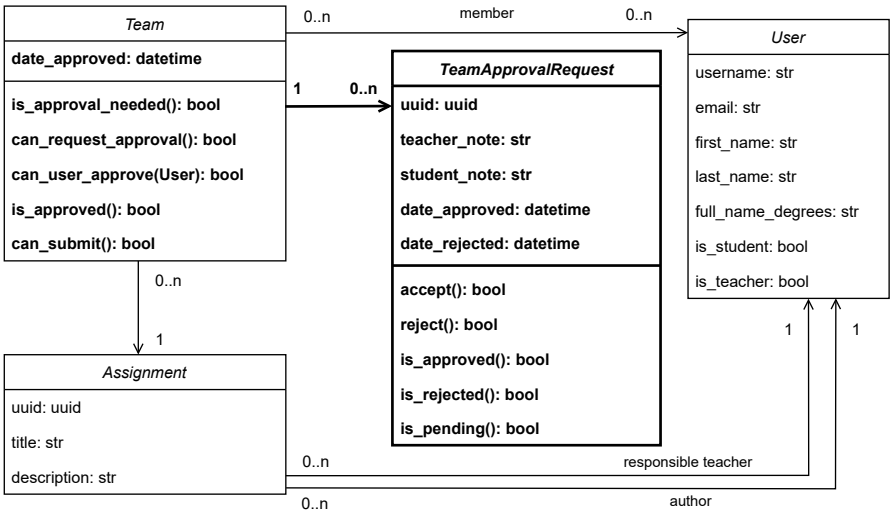
**Požadavek:** FP-NUR-2 – Studentské testování

V současné implementaci se studentské testování vztahuje vždy ke konkrétnímu odevzdanému řešení. Studentské testování je využíváno v předmětu NI-NUR, pro nějž je ale současná implementace nevyhovující (viz 1.2.3.4). Nabízely se dvě alternativní možnosti řešení – navázat potvrzení testování na tým jako takový, nebo na jednotlivé kontrolní body. Návaznost na kontrolní body by potenciálně mohla způsobovat problémy v situaci, kdy má v důsledku různých konfigurací sám tester ve svém projektu jiné kontrolní body, než testovaný tým – ne tak z implementačního hlediska, jako spíše z uživatelského, mohlo by to být matoucí. Také není jasné, jak by se potom měl počítat bodový zisk testera. Pro NI-NUR je zcela dostatečná a asi i vhodnější varianta, kdy se účast na testování vztahuje pouze k týmu jako takovému. I vzhledem k tomu, že

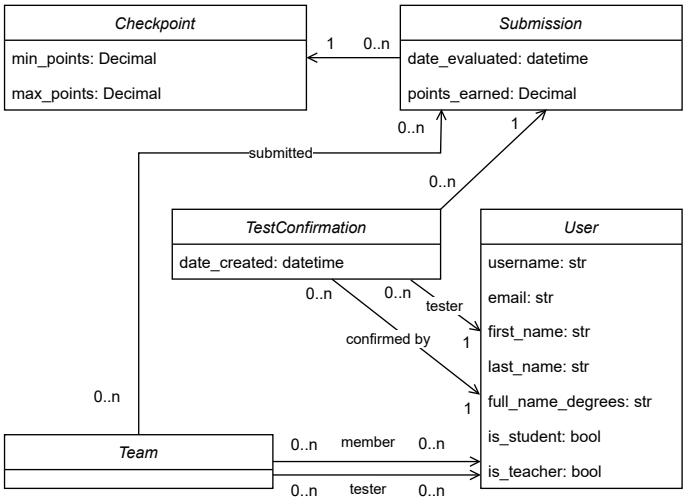
**Obrázek 2.9** Aktuální schéma schvalování zadání



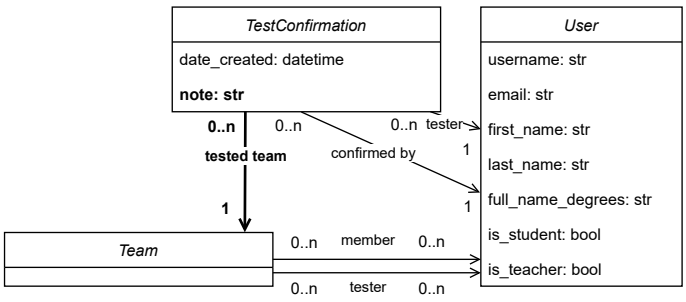

**Obrázek 2.10** Schéma schvalování týmů



**Obrázek 2.11** Aktuální schéma potvrzování účasti na testování




**Obrázek 2.12** Schéma potvrzování účasti na testování




NI-NUR je prozatím jediný známý předmět, který studentské testování využije, jsem se rozhodl pro toto řešení.

### 2.1.6.1 Změny modelu

Aktuální a upravené schéma potvrzování účasti na testování zachycují diagramy tříd na obrázcích 2.11 a 2.12. Potvrzení účasti je realizováno třídou `TestConfirmation`, jejíž instance je vytvořena, když vedoucí týmu potvrdí účast testera. Změnou v návrhu je úplné zrušení návaznosti na `Submission` (odevzdané řešení) a přeneseně i `Checkpoint` (kontrolní bod). Nově je `TestConfirmation` navázáno přímo na `Team` (projekt/tým). Aby mohla být instance vytvořena, musí být uživatel-tester členem daného týmu s rolí „tester“. Potvrzení účasti také nově zahrnuje textovou poznámku potvrzujícího v podobě atributu `note`.

### 2.1.6.2 Změny uživatelského rozhraní

V současnosti realizuje potvrzování účasti na testování vedoucí týmu na detailu existujícího odevzdaného řešení. Zde se nachází karta se seznamem nepotvrzených testerů s tlačítky pro potvrzení a seznamem již potvrzených testerů. Seznam potvrzených testerů zde vidí i ostatní uživatelé, kteří mají přístup na tuto obrazovku, zejména odpovědný vyučující. Tato karta v novém návrhu nedává smysl a bude úplně zrušena.

Nově bude testování navázáno na tým jako takový a proto bude i z uživatelského hlediska

**Obrázek 2.13** Wireframe karty pro potvrzování účasti na testování


řešeno na detailu týmu. Zde přibude rozbalovací karta „Potvrzení testeři“, kterou znázorňuje wireframe na obrázku 2.13. V její pravé části se nachází seznam již potvrzených testerů s daty potvrzení a textovými poznámkami. V levé části je vedoucímu týmu dostupný formulář pro potvrzení účasti na testování dalšího testera. Z dostupných možností vybere správného testera, volitelně vyplní textovou poznámku a záznam uloží tlačítkem „Potvrdit“. Ostatním uživatelům se zobrazí pouze seznam potvrzených testerů a namísto formuláře seznam nepotvrzených testerů.

### 2.1.7 Přerozdělení bodů

**Požadavek:** FP-SP-4 – Přerozdělení bodů

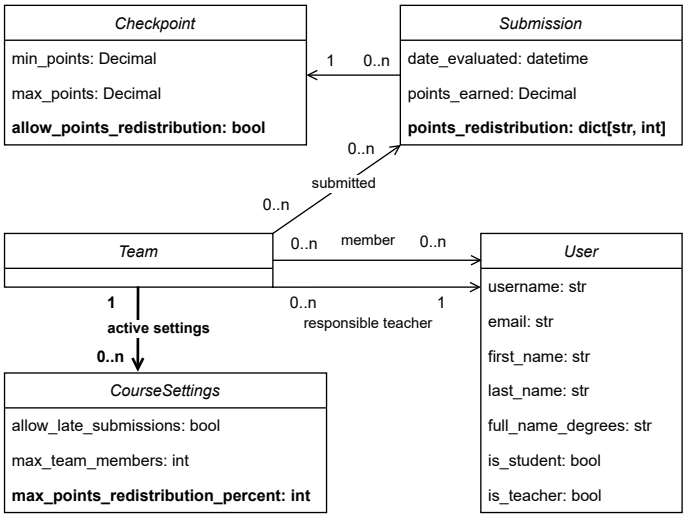
Jedním z požadavků, které nejsou aktuálně řešeny nijak, je přerozdělení bodů. Správce předmětu přerozdělení povolí určením nenulového maximálního procenta přerozdělených bodů v konfiguraci předmětu. Přerozdělení je aktivní pouze u těch kontrolních bodů, kde to povolil příslušný vyučující. Samotné procentuální přerozdělení je pak definováno u každého odevzdaného řešení nejprve studenty, následně může být upraveno vyučujícím v rámci ohodnocování. Uživatelé určí hodnotu v procentech pro každého člena týmu. Z těchto hodnot a bodového ohodnocení daného odevzdaného řešení jsou vypočteny bodové zisky jednotlivých členů týmu.

#### 2.1.7.1 Změny modelu

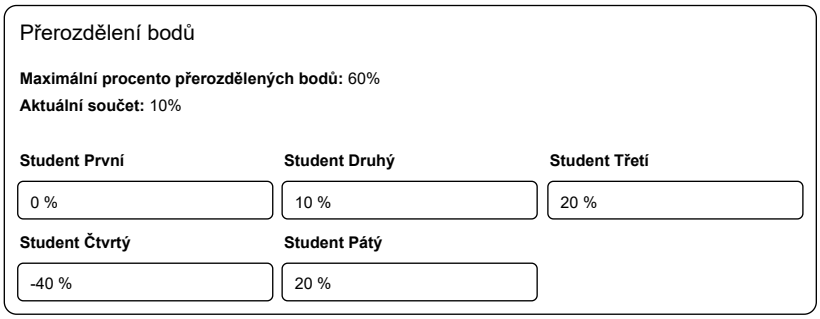
Změny spočívají v přidání několika atributů k třídám souvisejícím s odevzdáváním a hodnocením. Jedná se o atribut `max_points_redistribution_percent` u třídy `CourseSettings`, který značí maximální procento přerozdělených bodů u odevzdání stanovené správcem předmětu. Dále atribut `allow_points_redistribution` u třídy `Checkpoint` reprezentující kontrolní bod. Atribut určuje, zda je u daného kontrolního bodu přerozdělení bodů umožněno. U třídy `Submission`, reprezentující odevzdané řešení, byl přidán atribut `points_redistribution`, jehož hodnota je asociativní pole, kde klíčem je identifikátor člena týmu a hodnotou procento přerozdělených bodů. Nulová hodnota znamená, že student obdrží tolik bodů, kolik vyučující udělí v rámci ohodnocení, záporná či kladná hodnota pak značí, že student oproti udělenému počtu bodů ztrácí či získává navíc touto hodnotou danou část z udělených bodů. Aplikační logika pak zajistí, že součet hodnot u každého přerozdělení je nulový.

Popsané změny jsou znázorněny v diagramu na obrázku 2.14. Je zde také znázorněna vazba „active settings“ mezi týmem a konfigurací předmětu – jedná se o zjednodušení, představující aplikovanou konfiguraci vzhledem k tomu, jak je nastavena granularita nastavení předmětu. Může být uvažována konfigurace ke konkrétnímu týmu, k paralelce, nebo k celému předmětu.

**Obrázek 2.14** Schéma přerozdělení bodů




**Obrázek 2.15** Wireframe formuláře úpravy přerozdělení bodů




### 2.1.7.2 Změny uživatelského rozhraní

Konfigurace možnost přerozdělení bodů z pohledu vyučujícího je z hlediska uživatelského rozhraní přímočará – ve formuláři konfigurace nastaví maximální procento přerozdělení a ve formuláři úpravy kontrolního bodu zvolí, zda je u něj přerozdělení povoleno či nikoliv.

Definice konkrétního přerozdělení pak probíhá na stránce tvorby/úpravy odevzdání. Tam bude přidána karta znázorněná na obrázku 2.15. Na kartě je kromě polí pro zadání procenta přerozdělení pro jednotlivé studenty také informace o maximálním procentu přerozdělení pro jednoho studenta a aktuální součet přerozdělení, což uživateli usnadní dodržení podmínky, že celkový součet musí být vždy nulový. Stejná karta se zobrazí i vyučujícímu při hodnocení odevzdaného řešení a umožní mu přerozdělení upravit.

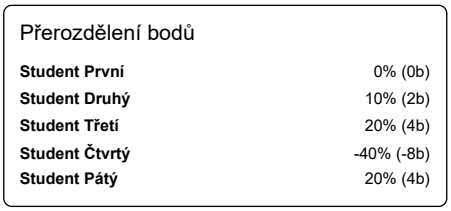
Na detailu odevzdaného řešení se pak zobrazí karta znázorněná na obrázku 2.16. Jsou na ní zobrazeny procentuální hodnoty přerozdělení pro jednotlivé členy týmu a pokud je řešení ohodnoceno vyučujícím, tak i konkrétní bodové zisky či ztráty v důsledku přerozdělení.

## 2.1.8 Statistiky předmětu

**Požadavek:** FP-GEN-9 – Statistiky

Další novou funkcionalitou SOS má být přehled statistik o jednotlivých předmětech. Požadováno je zobrazení informací o tom, kolik studentů má nebo nemá tým, případně mají podanou

**Obrázek 2.16** Wireframe karty s přerozdělením bodů




žádost o členství, dále informace o volných místech v týmech vzhledem k počtu studentů, kteří v žádném nejsou a nakonec statistiky splnění jednotlivých kontrolních bodů studenty.

Všechna potřebná data již SOS ve své databázi uchovává, je tedy pouze potřeba je správně agregovat a zobrazit uživateli. Na detail předmětu pro vyučujícího bude přidána karta, kde budou tato data vizualizována pomocí knihovny Chart.js, která je součástí použité šablony AdminLTE. Aby měl vyučující jednoduchý způsob, jak by mohl studenty z jednotlivých skupin kontaktovat, přibudou zde k tomuto účelu také tlačítka s `mailto` odkazy.

Potenciální problém je ve zobrazení statistik jednotlivých kontrolních bodů. V rámci konfigurace předmětu totiž mohou vyučující definovat jiný počet kontrolních bodů pro různé paralelky nebo týmy a není tedy jasné, jak by měla být data o splnění správně agregována. Jedním z možných řešení je seřadit kontrolní body jednotlivých konfigurací podle termínu odevzdání a seskupit je podle tohoto pořadí. V případě, že všechny dotčené konfigurace mají stejný počet kontrolních bodů, bude zobrazená statistika jistě dávat uživateli smysl, nicméně při odlišných počtech může být potenciálně matoucí. Dá se ale předpokládat, že jednotliví vyučující si pro všechny své paralelky či týmy nastaví stejný počet kontrolních bodů s podobnými termíny odevzdání, problém tedy pravděpodobně nastane pouze u správce předmětu, pokud si zobrazí všechny paralelky najednou. Rozhodl jsem se realizovat toto řešení s tím, že se v jeho vhodnost prakticky ověří během případného nasazení v BI-SP1, kde tento problém může nastat.

### 2.1.9 Vyhledávání a filtrování projektů

**Požadavek:** FP-GEN-6 – Filtrování projektů

Aktuálně SOS žádným způsobem neumožňuje vyhledávání nebo filtrování existujících projektů, pouze zobrazuje jejich seznam. Studentovi jsou zobrazeny projekty, do kterých by se dle konfigurace předmětu mohl přihlásit, vyučujícímu pak projekty za které odpovídá.

Dle požadavku má mít uživatel možnost vyhledat projekt podle názvu, odpovědného vyučujícího nebo člena týmu. Student má mít možnost vyfiltrovat projekty, ke kterým se může připojit, vyučující zase projekty, které ještě nemají dostatečný počet členů stanovený konfigurací předmětu.

Rozhodl jsem se toto realizovat pomocí několika vyhledávacích polí, která se budou nacházet nad seznamem projektů. Správce předmětu zde také bude mít možnost zvolit, zda chce zobrazit všechny projekty, nebo pouze ty, za které jako vyučující odpovídá. Filtrování projektů podle uživatelem zadaných parametrů pak proběhne pomocí dotazu do databáze na straně serveru. Parametry budou zároveň uloženy do cookie[1], aby je nemusel uživatel zadávat znovu pokaždé, když opustí stránku předmětu a následně se na ni vrátí.

### 2.1.10 Aktualizace seznamu studentů existujícího předmětu

**Požadavek:** FP-GEN-3 – Aktualizace seznamu studentů

[1] `https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies`

V současné implementaci jsou studenti k předmětu, resp. k jeho paralelkám, přiřazeni při importu předmětu z datového zdroje. V praxi se ale ukázalo, že není výjimkou, že si předmět student v KOS zapíše či zruší až po jeho importu do SOS. Garant předmětu může předmět do SOS importovat s předstihem, naopak studenti si zápis předmětů často upravují ještě v prvním týdnu po začátku semestru. Současná implementace umožňuje nápravu stavu kdy student v předmětu chybí pouze prostřednictvím administrátorského rozhraní. U BI-SP1 se navíc počítá s importem předmětu do SOS s větším předstihem, protože je cílem mít týmy utvořené již před začátkem semestru.

Nově bude tedy správce předmětu mít možnost manuálně vyvolat aktualizaci seznamu studentů z datového zdroje. Ta proběhne podobným způsobem jako samotný import. Pro každou existující paralelku předmětu v databázi bude proveden dotaz do datového zdroje na seznam studentů. Ten bude následně porovnán se seznamem studentů v databázi. Systém z něj pak přebývající studenty vyřadí a chybějící přidá. V případě, že nový student vůbec není zaznamenán v databázi SOS, záznam bude vytvořen. Nakonec bude správci předmětu zobrazena informace o počtu přidaných a odebraných studentů. Může se stát, že student, který má být z předmětu vyřazen, je již členem nějakého týmu. V tom případě vyřazen nebude, ale seznam takových studentů systém správci předmětu zobrazí, jelikož je tato situace nestandardní a je třeba ji se studenty individuálně řešit.

Tato nová funkcionalita nevyžaduje modelové změny, jedná se pouze o rozšíření aplikační logiky a uživatelského rozhraní. Aktualizaci je také případně možné automatizovat s využitím stejného mechanismu, kterým je již realizováno automatické rozesílání notifikací o blížících se termínech odevzdání. Správce předmětu by pak pouze nastavil, zda a jak často má být automatická aktualizace prováděna.

## 2.2 Export hodnocení

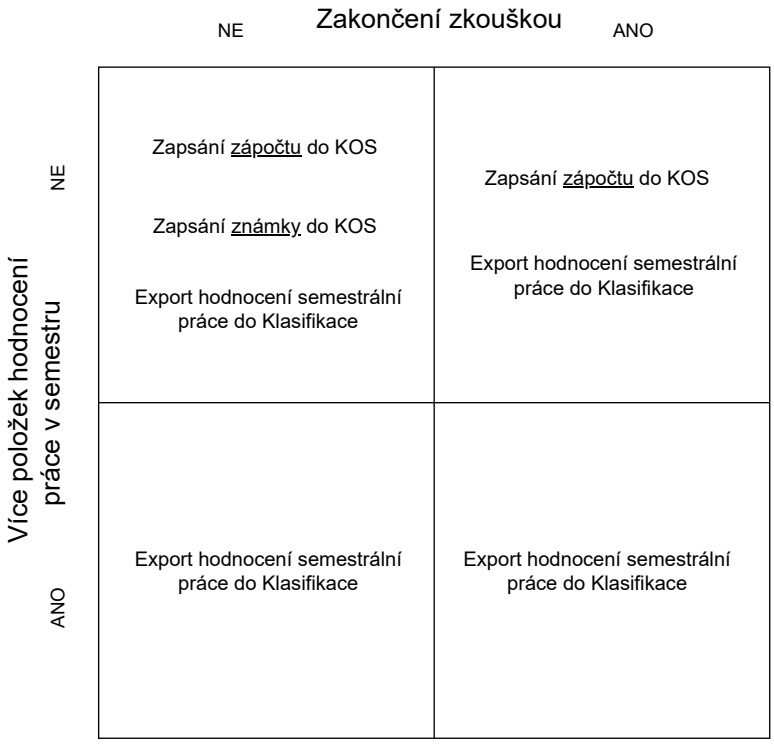
Jak jsem již nastínil v předchozí kapitole, jedním z rozšíření SOS v rámci této práce bude funkcionalita exportu hodnocení studentů do KOS a Klasifikace. V této sekci navrhuji celý proces exportu hodnocení ve dvou variantách – do KOS a do Klasifikace. Dále se zabývám možnými řešeními a popisuji zvolené řešení, které budu později implementovat.

### 2.2.1 Možnosti exportu hodnocení v předmětech

V první řadě je potřeba si říci, kdy přichází v úvahu export do KOS a kdy do Klasifikace. Předměty vyučované na FIT ČVUT jsem již dle zakončení klasifikoval v sekci 1.4.4. Nejjednodušším případem jsou předměty zakončené klasifikovaným zápočtem, kde jedinou položkou hodnocení je semestrální práce. Příkladem mohou být například předměty BI-SP1 a BI-SP2. U těchto předmětů je možné čistě z bodových hodnocení zaznamenaných v SOS určit získání zápočtu i výslednou známku. V úvahu tedy připadá plnohodnotný export do KOS. Do Klasifikace je samozřejmě možné tyto předměty exportovat také, ale funkčně to nepřinese nic navíc. Výhodou je pouze pohodlí studentů, kteří uvidí své hodnocení v předmětu v Klasifikaci spolu s dalšími předměty.

Druhou variantou jsou předměty s klasifikovaným zápočtem, kde se celkové hodnocení skládá kromě semestrální práce i z dalších položek, které nejsou zaznamenávány v SOS. Takovým předmětem je například BI-PYT (Programování v Pythonu), kde se kromě semestrální práce hodnotí i domácí úlohy a testy v průběhu semestru [32]. Domácí úlohy by jistě bylo možné evidovat v SOS, nicméně u testů by toto bylo velmi problematické. Pro předměty tohoto typu tedy export do KOS nepřipadá v úvahu vůbec – na základě bodových zisků evidovaných v SOS není možné stanovit ani to, zda student získal zápočet či nikoliv, nemluvě o výsledné známce. Smysl ale dává propsat bodové zisky ze semestrální práce realizované v SOS do Klasifikace jako jednu z položek hodnocení. Klasifikace pak vyhodnotí zápočet i výslednou známku na základě této a dalších

**Obrázek 2.17** Možnosti exportu hodnocení z SOS podle způsobu hodnocení předmětu




položek.

Nakonec zbývají předměty zakončené zkouškou. Zkoušku samotnou by teoreticky bylo možné zaznamenat v SOS jako poslední iteraci semestrální práce, nicméně pouze v případě, že jsou semestrální práce vypracovávány individuálně (nikoliv v týmech), nebo zkoušku skládá celý tým společně a všichni členové jsou hodnoceni stejně. Ve většině případů by toto řešení bylo absolutně nevhodné a při nejmenším matoucí. U předmětů zakončených zkouškou tedy nepřipadá v úvahu plnohodnotný export hodnocení do KOS. V případě, že je ale semestrální práce jedinou položkou hodnocení práce v semestru, lze na základě bodových zisků studentů alespoň vyhodnotit získání zápočtu a provést částečný export do KOS (pouze zápočty, známka bude zapsána samostatně po složení zkoušky). Do Klasifikace lze opět semestrální práci exportovat jako jednu z položek hodnocení, další položkou pak bude zkouška, jejíž hodnocení vyučující zaznamená sám. Toto je případ například předmětu NI-NUR, pro který byl SOS primárně vyvíjen. Poslední možností jsou předměty zakončené zkouškou, kde semestrální práce není jedinou položkou hodnocení během semestru. Stejně jako u obdobných předmětů bez zkoušky, přímý export do KOS nepřipadá v úvahu ze stejného důvodu. Zbývá opět pouze export semestrální práce do Klasifikace jako jedné z položek hodnocení.

Popsané možnosti exportu hodnocení z SOS shrnuje obrázek 2.17. Závěrem je, že export hodnocení semestrální práce do Klasifikace dává smysl u všech typů předmětů, zapisování zápočtů do KOS pouze u předmětů, kde je semestrální práce jedinou položkou hodnocení práce v semestru a zapisování známek do KOS pouze u těch z nich, které jsou zakončeny klasifikovaným zápočtem.

### 2.2.2 Export do KOS

Do KOS se zaznamenávají pouze zápočty a známky, oproti Klasifikaci neslouží k podrobnému zaznamenávání hodnocení v průběhu semestru. Teoreticky by tedy mohlo stačit do KOS ex-

portovat celkové hodnocení všech studentů předmětu na konci semestru. Problém ale nastává v případě, kdy ne všichni studenti získají zápočet ve stejnou dobu, některé předměty například umožňují odevzdat semestrální práci i ve zkouškovém období. Možnost přihlásit se na zkoušku ale student má až po zapsání zápočtu do KOS. Ideální by tedy bylo, pokud by byly alespoň zápočty do KOS exportovány automaticky při splnění podmínek studentem.

#### 2.2.2.1 Možné způsoby implementace

Ideální by bylo řešit export hodnocení do KOS čistě strojově, běžnou praxí je k těmto účelům využít API služby, do které je třeba zaznamenat data. Aktuálně využívané KOSapi nicméně umožňuje pouze čtení dat, zápis zápočtů ani známek není možný [24].

Aktuálně jsou uživatelům k dispozici dvě verze webového rozhraní KOS – staré a nové [33]. Nové rozhraní bylo spuštěno 1. září 2021, to staré mělo být dostupné do konce akademického roku 2021/22. Dostupné ale stále je i v době psaní tohoto textu v akademickém roce následujícím. Po zběžné analýze nového uživatelského rozhraní s využitím běžného webové prohlížeče jsem zjistil, že je pravděpodobně implementováno s využitím frameworku Vue.js a s backendem komunikuje prostřednictvím REST API, které je dostupné na adrese `https://new.kos.cvut.cz/rest/api/`. Přihlášenému uživateli je dostupný například endpoint[2], který nabízí informace o předmětech, nebo endpoint[3], který slouží k odhlášení ze zkouškového termínu. Na základě těchto zjištění jsem přesvědčen, že musí existovat i způsob, jak pomocí toho API zapsat zápočty a známky, pokud k tomu má přihlášený uživatel příslušná oprávnění. Problém pro využití tohoto API ale představuje právě autentizace uživatele – buďto by bylo nutné nějakým způsobem „ukrást“ již existují session z jiného okna prohlížeče, nebo požadovat po uživateli zadání hesla pro KOS do uživatelského rozhraní SOS. Ani jednu z těchto možností nepovažuji za přijatelnou. Takové řešení by bylo příliš komplikované, dále mám pochybnosti ohledně etické stránky věci, protože by se v podstatě jednalo o XSS útok [34]. Dá se navíc předpokládat, že by se mohlo fungování takového řešení VIC[4] jakožto provozovatel KOS aktivně bránit. Další zásadní překážkou využití tohoto API je chybějící dokumentace. Nejen, že by byl tímto vývoj velmi ztížen, ale rozhraní neveřejného API by se také mohlo kdykoliv bez varování změnit a jakékoliv řešení na něm postavené by mohlo přestat fungovat.

Alternativní možností je nějakým způsobem automatizovat vyplnění příslušného formuláře ve webovém rozhraní KOS, kde změny následně uživatel přihlášený do KOS standardním způsobem potvrdí a uloží. Pro tyto účely by mohlo být možné využít nástroje jako je například Selenium WebDriver[5], který primárně slouží pro automatické testy uživatelských rozhraní webových aplikací a umožňuje simulovat akce uživatelů. Takové řešení by ale mohlo být nepřiměřeně náročné na údržbu vzhledem k tomu, že se může uživatelské rozhraní KOS kdykoliv bez varování změnit, navíc je zde opět problém s přihlašováním uživatelů.

Poslední možností je automatické vyplnění formuláře realizovat pomocí skriptu v jazyce JavaScript, který přihlášený uživatel na příslušné stránce webového rozhraní KOS vloží do konzole internetového prohlížeče. Skript za něj vyplní formulář hodnocení, uživatel v okně prohlížeče uvidí veškeré takto vyplněné údaje a změny standardním způsobem uloží. Takové řešení již existuje například v podobě Moodle2KOS, což je sada skriptů pro přenos hodnocení ze systému Moodle do KOS [25]. Nevýhodou je opět nízká spolehlivost takového řešení, jelikož silně závisí na uživatelském rozhraní KOS. Výhodou je naopak poměrně jednoduchá implementace a v rámci možností uživatelská přívětivost.

[2] `https://new.kos.cvut.cz/rest/api/courses`
[3] `https://new.kos.cvut.cz/rest/api/terms/{id:term}/sign-off-by-student/{id:student}`
[4] `https://www.cvut.cz/vypocetni-a-informacni-centrum`
[5] `https://www.selenium.dev/documentation/webdriver/`

#### 2.2.2.2 Návrh řešení

Po důkladném zvážení všech výše popsaných možností jsem se nakonec rozhodl pouze modifikovat řešení Moodle2KOS pro účely exportu zápočtů a/nebo známek z SOS do KOS. Původní Moodle2KOS sestává ze dvou částí. První část slouží k přečtení dat z webového rozhraní systému Moodle a jejich uložení ve formátu JSON. Toto je pro SOS irelevantní, jelikož data jsou již uložena přímo v něm. Využitelná je naopak druhá část, která se stará o vyplnění formuláře hodnocení ve webovém rozhraní KOS.

Řešení pro SOS bude spočívat ve vygenerování skriptu na serveru. Pro získání dat je s menšími úpravami možné využít již existující kód, který kompletuje přehled hodnocení pro zobrazení v uživatelském rozhraní SOS. Vygenerovaný skript obsahující potřebná data a funkce pro vyplnění formuláře v KOS pak bude zobrazen uživateli spolu s instrukcemi jak jej použít. Uživatel bude mít navíc možnost zvolit, zda má skript v KOS vyplnit pouze zápočty, nebo také známku.

Vzhledem k tomu, že toto řešení není automatizované a export je nutné provádět manuálně je potřeba určit, kdo a kdy by tak měl učinit a jaká data budou exportována. Uživatel může být buďto správce předmětu, nebo jiný vyučující v SOS, zároveň musí mít tento vyučující oprávnění zapisovat vybraným studentům hodnocení do KOS. Pokud uživatel není správce předmětu, skript bude obsahovat pouze data studentů, kteří se uživateli zobrazují na stránce přehledu hodnocení v SOS, tedy studenti, kteří patří do týmů nebo paralelek vyučujícího. Pokud je uživatel správce předmětu, má na této stránce možnost volby, zda zobrazit pouze své studenty nebo všechny. Podle této volby budou vložena data i do vygenerovaného skriptu. Získání zápočtu a případně známky budou určeny podle celkových bodových zisků studentů. Export vyučující neprovádí průběžně, ale až na konci semestru, kdy mají studenti kompletně ohodnocené semestrální práce a mají tedy (ti úspěšní) nárok na zápočet. V případě, že následně dojde ke změnám v hodnocení, někteří studenti například získají zápočet až v průběhu zkouškového období, bude možné export libovolně opakovat.

### 2.2.3 Export do Klasifikace

Aplikace Klasifikace poskytuje rozsáhlé možnosti definice hodnocení. Vyučující může specifikovat jednotlivé položky hodnocení, jejich datový typ (číslo, boolean nebo řetězec) a u číselných položek také minimální, minimální požadované a maximální hodnoty. Dále také může některé položky hodnocení definovat vzorcem, jejich hodnota tak může být vypočtena z jiných položek. To se hodí například pro celkové hodnocení semestrální práce, které bude součtem hodnocení jednotlivých iterací, zápočet, který student získá při splnění podmínek definovaných vzorcem, nebo výpočet celkové známky.

Klasifikace také na rozdíl od KOS oficiálně poskytuje REST API [30], díky čemuž by mělo být poměrně snadné implementovat export hodnocení z SOS.

#### 2.2.3.1 Exportovaná data

Díky konfigurovatelnosti Klasifikace je možné zvolit, jaká data se budou z SOS exportovat. Ideální by bylo exportovat zvlášť hodnocení jednotlivých kontrolních bodů, případně bodový zisk za studentské testování a celkové hodnocení semestrální práce definovat v Klasifikaci vzorcem jako součet těchto položek. Problém ale nastává, pokud je v rámci předmětu v SOS používáno více různých konfigurací, zejména pokud se liší počtem kontrolních bodů. Klasifikace totiž sice umožňuje vyučujícímu definovat jednotlivé položky hodnocení, nicméně pouze na úrovni předmětu, nikoliv zvlášť například pro paralelky nebo jiné skupiny studentů.

První možností řešení tohoto problému by bylo jednoduše v Klasifikaci definovat větší množství kontrolních bodů, u kterých by následně byly získané body vyplňovány pouze u těch studentů, u kterých to dává smysl vzhledem k jejich konfiguraci předmětu. SOS nicméně aktuálně nijak neomezuje počet definovaných kontrolních bodů, není tedy jasné, kolik kontrolních bodů

by mělo být v Klasifikaci definováno. Omezení na počet kontrolních bodů je samozřejmě možné do SOS zavést, ale buď by se tím zbytečně omezila konfigurovatelnost SOS pro potřeby různých předmětů, nebo by to vedlo ke zvýšení komplexity konfigurace, kterou již nyní někteří uživatelé hodnotí jako příliš náročnou.

Druhou možností je exportovat jednotlivé kontrolní body pouze u těch předmětů, kde existuje pouze jedna společná konfigurace a tedy jedna sada kontrolních bodů platná pro všechny studenty předmětu. Kontrolní body pak lze úplně a jednoznačně reprezentovat v Klasifikaci. U ostatních předmětů se exportuje pouze celkové hodnocení semestrální práce, jehož hodnota se v průběhu semestru několikrát změní, jak bude student odevzdávat semestrální práci v jednotlivých iteracích. Dále je v obou případech možné exportovat bodový zisk za studentské testování jako další položku v případě, že má předmět studentské testování v SOS aktivované.

Nakonec jsem zvolil druhé řešení, protože nevyžaduje zásah do jiných částí SOS kromě implementace samotného exportu a povede k nejspolehlivějšímu zobrazení získaných bodů v Klasifikaci pro všechny studenty. Jeho nevýhodou je, že v některých předmětech nebudou v Klasifikaci uvedeny kontrolní body jednotlivě, detailní hodnocení student nalezne pouze v SOS. Nemyslím si ale, že by to byl zásadní problém – student stejně v průběhu semestru SOS používá a své bodové zisky za kontrolní body zde vidí. V Klasifikaci pak bude mít zobrazeny pouze důležité údaje jako celkový bodový zisk za semestrální práci, případně další položky hodnocení (které nejsou zaznamenány v SOS), zápočet a celkovou známku.

#### 2.2.3.2 Vyvolání exportu

Díky využití API mohou být ve většině případů bodové zisky studentů do Klasifikace exportovány automaticky ze strany serveru. Studenti mohou v SOS obdržet body dvěma způsoby:

- vyučující ohodnotí odevzdané řešení iterace nebo změní existující hodnocení,
- vedoucí týmu potvrdí studentovi účast na testování.

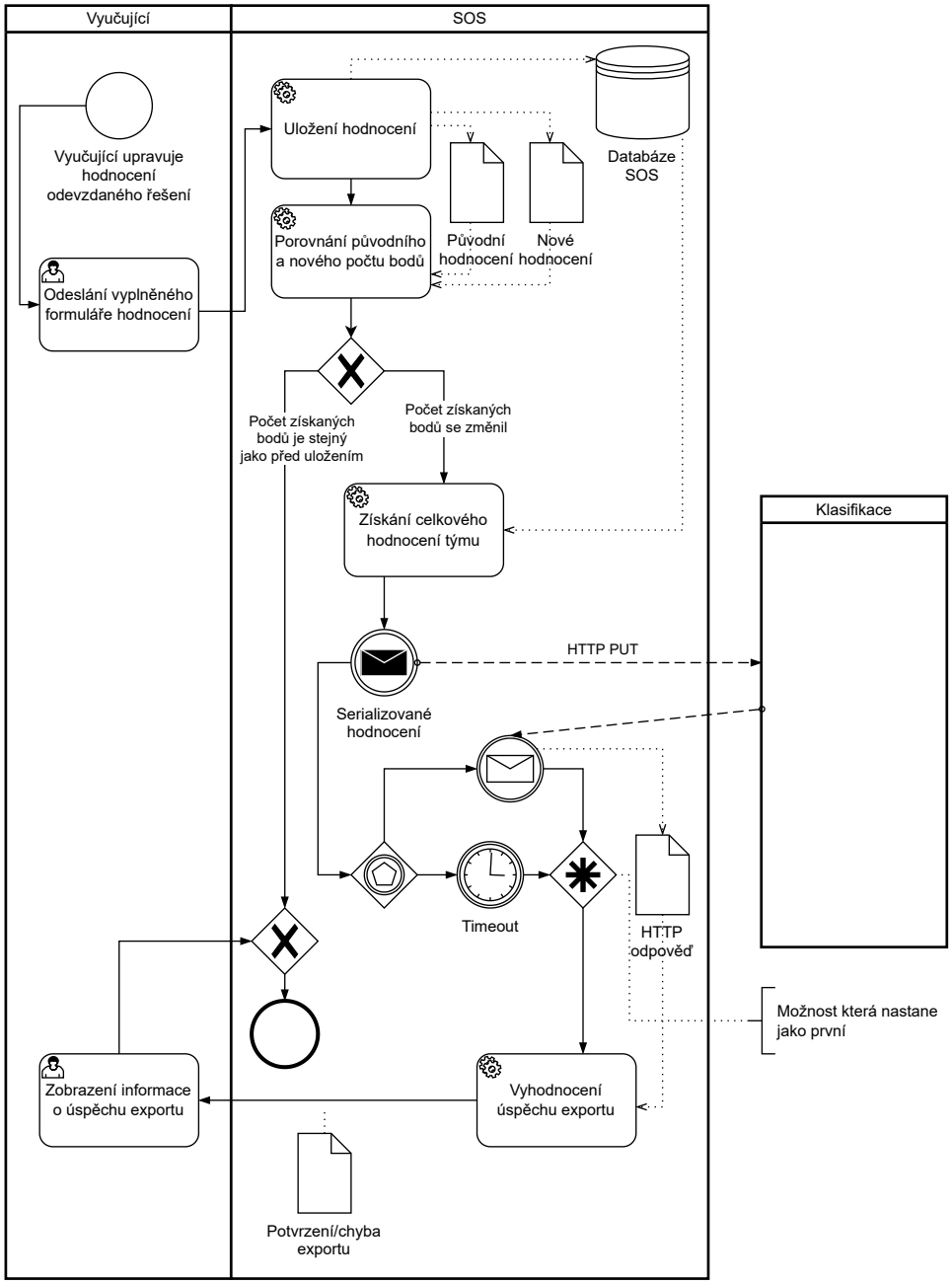
V prvním případě bude využit již existující kód k získání všech bodových zisků členů týmu. Tato data budou následně serializována a odeslána do Klasifikace. V druhém případě bude vypočten pouze bodový zisk testera za účasti na testování, následně bude stejným způsobem serializován a odeslán.

Kromě automatického exportu bude mít vyučující také možnost vyvolat export manuálně. Toto bude realizováno na stránce přehledu klasifikace, podobně jako export do KOS. Kliknutím na tlačítko zde vyučující vyvolá export kompletních hodnocení všech studentů daného předmětu, nebo pouze svých studentů, podle stejné logiky jako v případě exportu do KOS. K získání potřebných dat může být opět využit již existující kód, neboť se jedná o stejná data, jako jsou již na stránce přehledu klasifikace zobrazena. Data budou opět serializována a odeslána do Klasifikace přes její API. Smyslem této funkcionality je řešení situací, kdy z nějakého důvodu nebyla všechna data do Klasifikace odeslána automaticky. Příkladem mohou být následující situace:

- správce předmětu export do Klasifikace aktivoval až v průběhu semestru, kdy již byla nějaká řešení ohodnocena,
- došlo k výpadku Klasifikace nebo selhání komunikace z jiného důvodu,
- nebyla automaticky zapsána hodnocení všech studentů z důvodu neaktuálního seznamu studentů předmětu v jednom ze systémů.

Celý proces uložení hodnocení odevzdaného řešení s automatickým exportem bodů do Klasifikace znázorňuje BPMN diagram [35] na obrázku 2.18. Diagram procesu uložení bodů za potvrzenou účast na testování by vypadal velmi podobně, lišil by se sice v jednotlivých aktivitách a lidským aktérem by byl místo vyučujícího vedoucí týmu, ale logická struktura a především část, kde se komunikuje s Klasifikací by byla totožná.

**Obrázek 2.18** BPMN diagram procesu uložení hodnocení s exportem do Klasifikace



#### 2.2.3.3 Využití API a autorizace

V první řadě je nutné vyřešit autorizaci, aby bylo vůbec možné s API komunikovat. Klasifikace i její API řeší autorizaci protokolem OAuth 2.0 a stejně jako SOS využívá fakultní OAAS. V souladu se specifikací je možné autorizovat přímo uživatele, nebo samotnou aplikaci (tzv. *service account*). Na fakultním OAAS jsou pro Klasifikaci k dispozici scope `cvut:grades:user-write` a `cvut:grades:course-restricted`. První z nich slouží pro zápis s využitím uživatelského tokenu, druhý pro čtení pomocí tokenu vydaného přímo autorizované aplikaci na základě tzv. *client-credentials*[6].

Vzhledem k tomu, že k API je nutné přistupovat i v případech, kdy uživatel přihlášený do SOS není vyučující (konkrétně zápis bodů za potvrzené studentské testování), rozhodl jsem se pro řešení používající service account. První řešení by také nefungovalo v případě, že bude v SOS do předmětu přidán (dle FP-SP-1) vyučující, který podle KOS předmět neučí (nemusel by totiž mít oprávnění zapisovat hodnocení do Klasifikace). Naopak nevýhodou využití scope `course-restricted` je, že musí v Klasifikaci v daném předmětu jeho garant přístup externí aplikaci (v tomto případě SOS) manuálně přidělit.

Dle dokumentace API se hodnocení zapisuje odesláním HTTP PUT požadavku na příslušný endpoint[7]. Do těla požadavku má být ve formátu JSON vloženo pole záznamů jednotlivých klasifikací, které obsahují minimálně tyto informace [30]:

- `classificationIdentifier` – identifikátor sloupce, např. `semestral_work`,
- `studentUsername` – uživatelské jméno studenta,
- `value` – hodnota hodnocení podle datového typu sloupce – číslo, boolean nebo řetězec.

Před odesláním požadavku je tedy potřeba data serializovat do tohoto formátu. Dále je nutné ošetřit některé nestandardní situace, kdy například některý student z SOS v Klasifikaci chybí, nebo vícenásobný zápis stejného hodnocení jednomu studentovi, který může vzhledem k výše popsanému návrhu nastat. Podle informací od administrátora Klasifikace by se API mělo s těmito situacemi umět samo vypořádat, je ale nutné to ještě ověřit během implementace.

### 2.2.4 Uživatelské rozhraní

Automatické exportování hodnocení do Klasifikace samo o sobě nevyžaduje změny uživatelského rozhraní, kromě aktivace této funkcionality na stránce nastavení předmětu z pohledu správce. Rozšíření uživatelského rozhraní je naopak nutné pro realizaci manuální varianty exportu do Klasifikace a exportu do KOS. Spočívat bude v přidání dialogu, který je znázorněn na obrázku 2.19 a vyučující jej otevře kliknutím na tlačítko „Exportovat hodnocení“ na stránce přehledu klasifikace. Dialog má dvě části – KOS a FIT Klasifikace. V obou bude zobrazen textový popis dané funkcionality a instrukce, jak ji použít. Část „KOS“ pak bude obsahovat zaškrtávací pole určující, zda vygenerovaný skript do KOS zapíše i známky a textové pole, ze kterého může uživatel skript zkopírovat. Část „FIT Klasifikace“ obsahuje kromě textu pouze jedno tlačítko, které vyvolá export hodnocení všech studentů do Klasifikace.

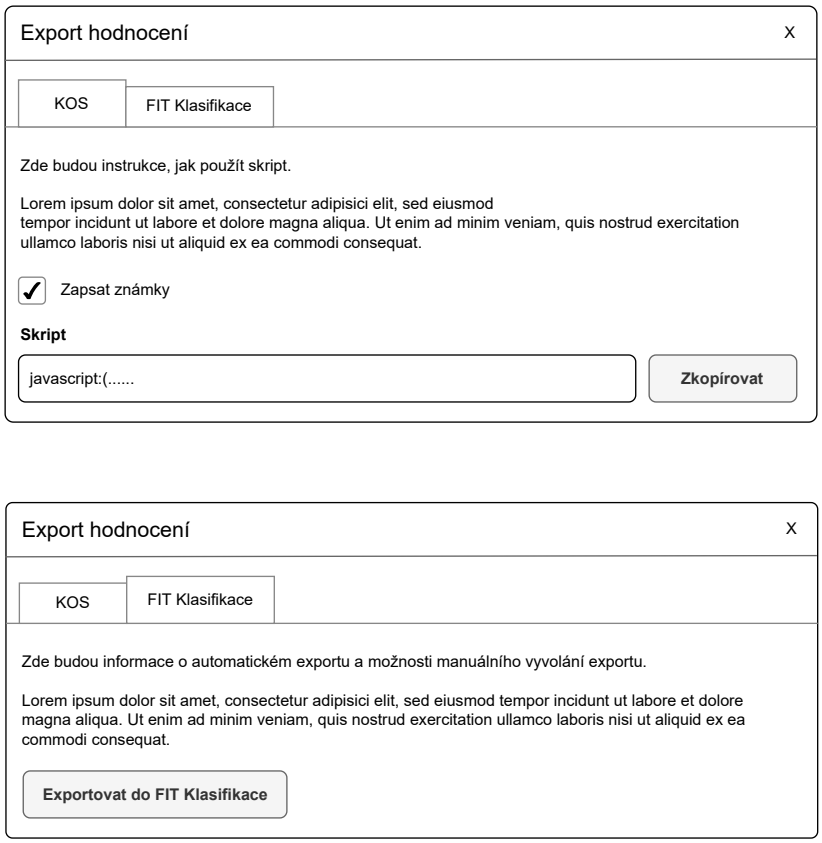
## 2.3 Shrnutí

V první části kapitoly jsem popsal návrh změn v systému, které zajistí splnění všech požadavků uvedených v předchozí kapitole. Návrh zahrnuje úpravy a rozšíření datového modelu, aplikační logiky i uživatelského rozhraní. V druhé části kapitoly jsem se zabýval exportem hodnocení do KOS a FIT Klasifikace, navrhl jsem proces exportu hodnocení z SOS do těchto systémů a

[6] Client ID a Client Secret

[7] `/public/courses/{course-code}/student-classifications`

**Obrázek 2.19** Wireframe dialogu exportu hodnocení



popsal zvolené řešení z pohledu plánované implementace i uživatelského rozhraní. Realizace všech popsaných úprav a rozšíření podle návrhu je předmětem následující kapitoly Realizace.

# Kapitola 3
# Realizace

*V této kapitole se zabývám realizací všech navržených úprav a rozšíření a souvisejícími procesy. Popisuji zvolenou metodiku vývoje a celkový postup vývoje v čase. Dále rozebírám implementaci jednotlivých položek návrhu z funkcionálního a implementačního hlediska a související změny uživatelského rozhraní. V závěru kapitoly se věnuji realizaci ostatních procesů souvisejících s vývojem softwarového produktu, jako je správa verzí a nasazování aplikace do provozu.*

## 3.1 Metodika vývoje a postup

Existuje řada metodik a přístupů k vývoji softwaru, mezi něž se řadí například vodopádový, prototypový a spirálový přístup, RAD, Scrum nebo Extrémní programování. Metodiky lze obecně rozdělit na sekvenční a iterativní. Sekvenční (nebo také lineární) metodiky staví na rozdělení vývoje do oddělených fází, které na sebe navazují a jsou prováděny postupně, s dodáním celého produktu najednou po dokončení vývoje. Iterativní metodiky oproti tomu kladou důraz na flexibilitu a otevřenost ke změnám v průběhu vývoje, jelikož softwarový produkt je dodáván po částech v jednotlivých iteracích. [36]

V této práci jsem pro vývoj zvolil inkrementální přístup. Jedná se o kombinaci sekvenčních a iterativních metodik a jeho cílem je především omezení projektových rizik rozdělením projektu na menší segmenty, čímž je umožněno zavedení změn během procesu vývoje. Základní principy inkrementálního přístupu jsou [37]:

- provádění sérií malých vodopádů, kde každý vodopád je prováděn pro malou část systému a je dokončen před pokračováním na další přírůstek, nebo
- obecné požadavky jsou definovány dříve než se přikročí k evolučnímu vývoji pomocí malých vodopádů, nebo
- prvotní koncept, analýza požadavků, design architektury a systémové jádro jsou definovány vodopádovým přístupem, následuje iterativní prototypový přístup, který vrcholí instalací konečného prototypu jako funkčního systému.

Díky využití inkrementálního přístupu bylo možné nasadit verzi SOS splňující v uspokojivé míře většinu požadavků kritických pro předmět BI-SP1 do provozu již před začátkem letního semestru 2022/23. Během semestru jsem pak mohl iterativně vyvíjet zbývající funkcionality a zdokonalovat aplikaci tak, aby plně reflektovala veškeré požadavky, které vyplynuly z analýzy. Zároveň jsem mohl do výsledné implementace průběžně zapracovávat zpětnou vazbu od uživatelů z řad vyučujících a studentů, včetně své vlastní zkušenosti s používáním systému, jelikož jsem se účastnil projektu v předmětu BI-SP1 v roli asistenta. Nové verze jsem mohl snadno nasazovat do provozu s využitím CI/CD, o čemž se zmiňuji dále v této kapitole.

### 3.1.1 Úvodní fáze před nasazením do provozu

Práci jsem zahájil v prosinci 2022 provedením analýzy a návrhu úprav potřebných pro použití SOS v předmětu BI-SP1 v následujícím semestru. Identifikoval jsem požadavky kritické pro tento předmět, především v jeho úvodní fázi tvorby týmů:

- Žádosti o přístup do předmětu (FP-SP-1)
- Nastavení předmětu na úrovni týmu (FP-GEN-2)
- Aktualizace seznamu studentů existujícího předmětu (FP-GEN-3)

Následující funkcionality nebyly pro nasazení v první fázi nezbytné, byly ale stále důležité pro použitelnost systému a tak jsem je do této fáze taktéž zařadil:

- Vlastní role členů týmu (FP-SP-3)
- Schvalování týmů (FP-NUR-1)
- Přerozdělení bodů (FP-SP-4)
- Statistiky předmětu (FP-GEN-9)
- Vyhledávání a filtrování projektů (FP-GEN-6)

Implementaci v této fázi jsem prováděl ve třech iteracích. Výsledky byly validovány se zadavatelem a vedoucím práce Hunkou a s garantem BI-SP1 Mlejnkem. Na závěr provedl samostatně garant BI-SP1 akceptační testování, čemuž se podrobně věnuje kapitola Testování.

### 3.1.2 Nasazení do provozu v BI-SP1

Přibližně dva týdny před začátkem letního semestru 2022/23 jsem do provozu nasadil verzi SOS, která splňovala požadavky kritické pro předmět BI-SP1. Tuto verzi jsem označil jako 1.0.0 (označení verzí se více věnuje sekce 3.7). SOS jsem nasadil do produkčního[1] a do testovacího[2] prostředí. Zprovoznil jsem také CI/CD, abych mohl následně automaticky nasazovat další verze v průběhu semestru. Jednotlivým prostředím a procesu nasazování se podrobněji věnuje sekce 3.8. S využitím datových migrací, kterým se podrobněji věnuji v sekci 3.3 u popisu implementace jednotlivých změn, jsem mohl i přes poměrně rozsáhlé změny databázového schématu zachovat existující data z minulých let.

### 3.1.3 Projekt v BI-SP1

Jak jsem již zmínil v úvodu práce, v letním semestru 2022/23 vedeme s vedoucím práce a vyučujícím BI-SP1 Hunkou a kolegou Hejdou v předmětu BI-SP1 tým studentů bakalářského programu, abychom zajistili budoucnost projektu tím, že tyto studenty seznámíme s existující aplikací a zapojíme je do vývoje. Aby mohl takový tým efektivně pracovat, musel jsem mu zajistit odpovídající podmínky. To zahrnovalo zejména:

- opravu automatických testů, které přestaly fungovat v důsledku provedených úprav v první fázi vývoje,
- zprovoznění CI/CD,
- sepsání odpovídající dokumentace k lokálnímu spuštění projektu pro vývoj,

[1] Dostupné na `https://sos.fit.cvut.cz`
[2] Dostupné na `https://sos2.fit.cvut.cz`

- aktualizaci seznamu známých nedostatků v systému GitLab.

Zmíněným krokům z mé strany a další práci s týmem v průběhu semestru se detailněji věnuji v kapitole Zapojení dalších studentů.

### 3.1.4 Vývoj a průběžné aktualizace během semestru

V průběhu letního semestru 2022/23 jsem implementoval zbývající funkcionality a průběžně SOS zdokonaloval tak, aby byly veškeré definované požadavky naplněny. Také jsem navrhl a vyvinul řešení pro export hodnocení z SOS do KOS a FIT Klasifikace. Celkově se další implementace týkala těchto oblastí:

- Přístup k projektu pro další uživatele (FP-SP-2)
- Studentské testování (FP-NUR-2)
- Statistiky předmětu (FP-GEN-9; rozšíření o statistiky splnění kontrolních bodů)
- Export do KOS
- Export do FIT Klasifikace

Všechny tyto položky jsem implementoval sám, pouze některé drobné úpravy následně prováděli studenti BI-SP1, což je u popisu implementace vždy patřičně označeno. Během vývoje jsem také zpracovával průběžně získávanou zpětnou vazbu a patřičně upravoval definice funkčních požadavků a návrhy řešení – kapitoly Analýza a Návrh obsahují jejich finální podobu. Nové verze jsem průběžně nasazoval do testovacího a následně i produkčního prostředí na FIT ČVUT. Kromě samotného vývoje jsem se také věnoval zkvalitnění automatických testů, což je popsáno v kapitole Testování.

## 3.2 Vývojové prostředí

K vývoji jsem po celou dobu používal operační systém Microsoft Windows 11. Projekt SOS sice může být přímo spuštěn pouze na operačních systémech unixového typu, nicméně s využitím technologie Docker lze SOS zprovoznit na libovolném operačním systému, kde lze zprovoznit Docker, tedy i na Windows nebo také na Mac OS od společnosti Apple. Pro lokální spuštění projektu během vývoje jsem tedy využíval Docker, prostřednicvím aplikace Docker Desktop. Díky tomu jsem nemusel vůbec řešit instalaci Python interpreteru, virtuálního prostředí ani žádných knihoven a balíčků.

K úpravám zdrojových kódů projektu a dalších souborů jsem využíval zejména vývojové prostředí PyCharm od společnosti JetBrains. Prostředí PyCharm nabízí řadu pokročilých funkcí, jako je například Python debugger, statická analýza kódu a integrace systému pro správu verzí. Prostředí je určeno primárně pro jazyk Python, nicméně umožňuje i editaci dalších typů souborů, jako jsou například HTML, XML a JSON dokumenty, zdrojové kódy jazyka JavaScript či shell skripty. Lze do něj také nainstalovat řadu zásuvných modulů, které přidávají pokročilé funkce specifické pro některé knihovny a frameworky, jako je například Django. PyCharm zároveň umožňuje integraci s Python interpreterem, který se nachází uvnitř Docker containeru. Aplikaci provozovanou lokálně v Dockeru lze tedy vyvíjet, spouštět a debugovat stejně jednoduše, jako by byla spouštěna interpreterem nainstalovaným přímo na hostitelském operačním systému.

K testování vyvíjeného projektu během vývoje jsem používal běžné internetové prohlížeče Mozilla Firefox a Google Chrome na systému Windows, k otestování mobilní verze pak prohlížeče Mozilla Firefox a Safari na mobilním telefonu Apple iPhone 12 mini s operačním systémem iOS 16.

## 3.3 Implementace

Předchozí kapitola Návrh obsahuje návrh realizovaných změn a rozšíření. Jedná se o tyto oblasti:

- Žádosti o přístup do předmětu (FP-SP-1)
- Nastavení předmětu na úrovni týmu (FP-GEN-2)
- Přístup k projektu pro další uživatele (FP-SP-2)
- Vlastní role členů týmu (FP-SP-3)
- Schvalování týmů (FP-NUR-1)
- Studentské testování (FP-NUR-2)
- Přerozdělení bodů (FP-SP-4)
- Statistiky předmětu (FP-GEN-9)
- Vyhledávání a filtrování projektů (FP-GEN-6)
- Aktualizace seznamu studentů existujícího předmětu (FP-GEN-3)
- Export do KOS
- Export do FIT Klasifikace

V této sekci postupně popisuji způsob implementace všech těchto položek a související změny uživatelského rozhraní.

### 3.3.1 Žádosti o přístup do předmětu

**Implementovaný požadavek:** FP-SP-1
**Fáze vývoje:** Úvodní fáze před nasazením, rozšiřováno během semestru (duben 2023)
**Cílové verze:** 1.0.0, 1.4.3 (přímé přidání vyučujícího správcem), 1.4.5 (zohlednění paralelek)
**Hlavní související MR:** !163[3], !245 (přímé přidání vyučujícího správcem)[4], !249 (zohlednění paralelek)

Rozšíření se týká především aplikace `courses`. Jako první jsem provedl potřebné modelové změny. V souboru `apps/courses/models/base.py` se nachází třída `Course`, která v SOS reprezentuje předmět. Do této třídy jsem přidal boolean atribut `is_public`, jehož hodnota značí, zda je tento předmět zobrazen i jiným vyučujícím, než kteří jsou k němu již přiřazeni a zda mohou tito vyučující požádat garanta předmětu o přístup. Definici atributu ilustruje výpis kódu 3.1.

Dále jsem přidal nový model `CourseAccessRequest`[5], který reprezentuje žádost o přístup do předmětu. Třída používá `TimestampedModelMixin`[6], který přidává a automaticky nastavuje atributy `date_created` a `date_updated`. Důležité části kódu jsou viditelné ve výpisu kódu 3.2.

Následně jsem upravil související Views, konkrétně `CourseDetailView` a `TeacherHomeView`. Původně nemohl uživatel `CourseDetailView` navštívit vůbec, pokud neměl přístup do daného předmětu. Nově je tento uživatel, pokud se jedná o vyučujícího a předmět má povoleny žádosti o přístup, přesměrován na nové `CourseRequestAccessView`. Tuto logiku implementuje metoda `dispatch`, která je zobrazena na výpisu kódu 3.3. Úprava `TeacherHomeView` se pak týkala získání seznamu předmětů, u kterých může uživatel požádat o přístup, aby mohly být zobrazeny v uživatelském rozhraní.

[3] Tento MR obsahuje i řadu dalších změn obsažených i v jiných MR.
[4] Autorem tohoto rozšíření je člen SP týmu.
[5] `CourseAccessRequest` je definovaný v souboru `apps/courses/models/access_request.py`
[6] `TimestampedModelMixin` je definovaný v souboru `core/mixins.py`

**Výpis kódu 3.1** Nový atribut `Course.is_public`

```
class Course(models.Model):
    ...
    is_public = models.BooleanField(
        verbose_name=_("allow teachers to request access"),
        default=False
    )
        ...
```

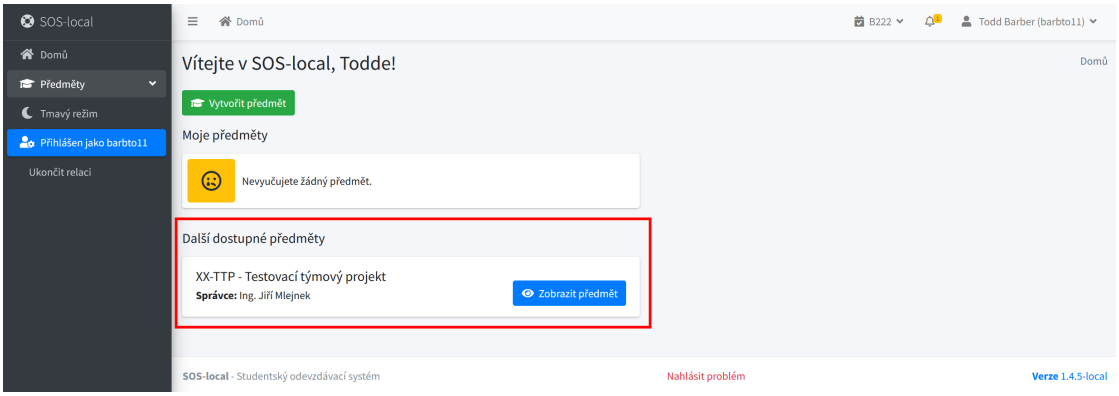
**Výpis kódu 3.2** Nový model `CourseAccessRequest`

```
class CourseAccessRequest(TimestampedModelMixin, models.Model):
    ...

    # fields
    uuid = models.UUIDField(
        default=uuid.uuid4, editable=False, unique=True
    )

    message = models.CharField(...)
    date_accepted = models.DateTimeField(..., blank=True, null=True)
    date_rejected = models.DateTimeField(..., blank=True, null=True)
    date_cancelled = models.DateTimeField(..., blank=True, null=True)

    # relations
    course = models.ForeignKey(
        "courses.Course", on_delete=models.CASCADE, ...
    )
    teacher = models.ForeignKey(
        settings.AUTH_USER_MODEL, on_delete=models.CASCADE, ...
    )
    parallels = models.ManyToManyField(
        "courses.Paralell"
    )

    @transaction.atomic
    def accept(self):
        self.course.add_teacher(self.teacher, self.parallels.all())

        self.date_accepted = timezone.localtime()
        self.save()

    def reject(self)
        self.date_rejected = timezone.localtime()
        self.save()

    def cancel(self)
        self.date_cancelled = timezone.localtime()
        self.save()
        ...
```

**Výpis kódu 3.3** Přesměrování uživatele v `CourseDetailView.dispatch`

```
class CourseDetailView(AccessMixin, ..., DetailView):
    ...

    def dispatch(self, request, *args, **kwargs):
        if not request.user.is_authenticated:
            return self.handle_no_permission()

        course = self.get_object()
        if (
            course.is_user_student(request.user)
            or course.is_user_teacher(request.user)
        ):
            return super().dispatch(request, *args, **kwargs)

        if course.is_public and request.user.is_teacher:
            return redirect(course.get_request_access_url())

        return self.handle_no_permission()
```

**Obrázek 3.1** Dostupné předměty na domovské obrazovce




Vytvořil jsem také dvě nové Views. Prvním z nich je `CourseRequestAccessView`, na které je uživatel přesměrován z `CourseDetailView`, pokud nemá do předmětu přístup a kde může o přístup požádat, nebo upravit či zrušit existující žádost. K tomuto účelu view využívá formulář `CourseAccessForm`[7]. Druhé View `CourseAccessRequestUpdateView` slouží ke změně stavu žádosti – umožňuje ji přijmout nebo odmítnout, což ilustruje výpis kódu 3.4.

Všechna tři Views týkající se předmětu nebo žádosti o přístup do předmětu jsou definována v souboru `apps/courses/views/base.py`. `TeacherHomeView` je součástí aplikace `homepage`.

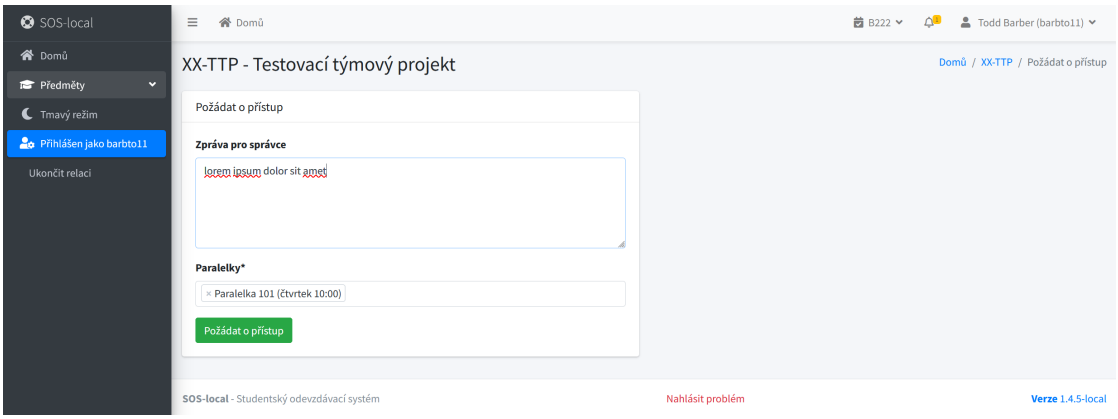
Úpravy a rozšíření uživatelského rozhraní jsem provedl na třech místech. Jako první jsem vytvořil novou obrazovku pro vytvoření a změnu žádosti o přístup do předmětu, která je obsluhována `CourseRequestAccessView`. Dále jsem na stránku předmětu přidal kartu se seznamem aktivních žádostí o přístup, která se zobrazuje garantovi předmětu. Nakonec jsem na domovskou stránku vyučujícího přidal seznam předmětů, kde může požádat o přístup. Úpravy jsou viditelné na snímcích obrazovky na obrázcích 3.1, 3.2, 3.3 a 3.4.

[7] `CourseAccessForm` je definován v souboru `apps/courses/forms.py`

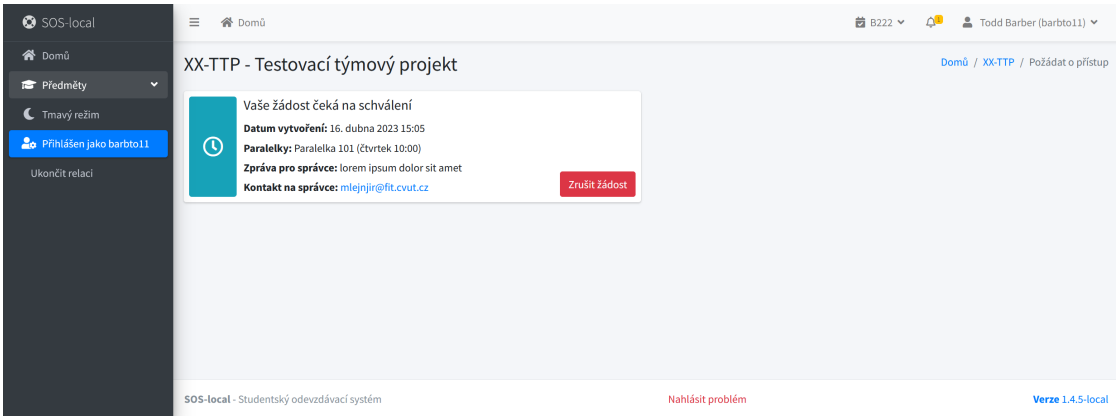
**Výpis kódu 3.4** Změna stavu žádosti o přístup do předmětu

```
class CourseAccessRequestUpdateView(..., UpdateView):
    model = CourseAccessRequest
    ...

    def post(self, request, *args, **kwargs):
        action = request.POST.get("action", "").lower()
        action_method = getattr(self, action)
        return action_method()

    def approve(self):
        ...
        access_request = self.get_object()
        access_request.accept()
        ...

    def reject(self):
        ...
        access_request = self.get_object()
        access_request.reject()
            ...
```

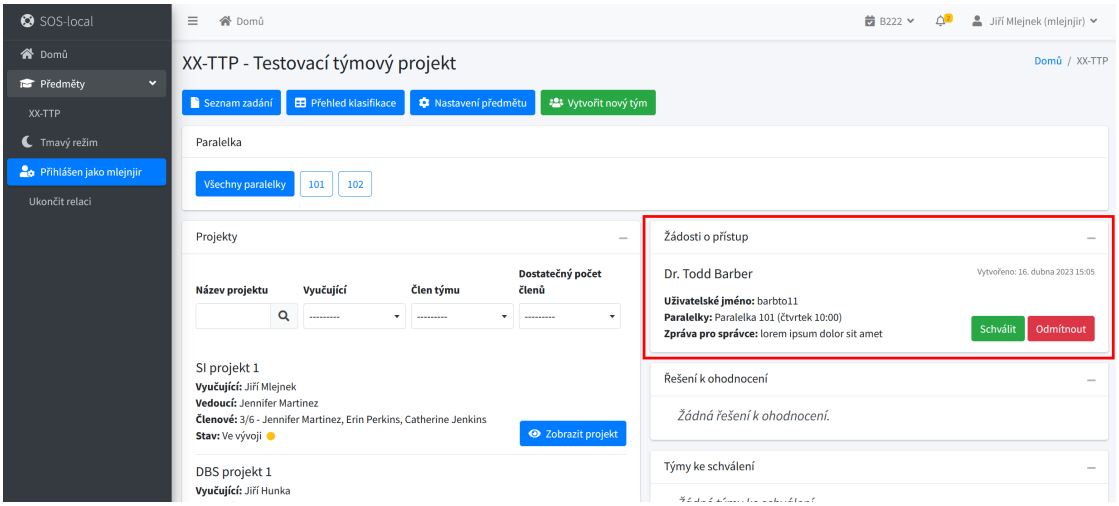
**Obrázek 3.2** Formulář k podání žádosti o přístup do předmětu




**Obrázek 3.3** Aktivní žádost o přístup do předmětu



**Obrázek 3.4** Seznam žádostí o přístup do předmětu z pohledu správce




### 3.3.2 Nastavení předmětu na úrovni týmu

**Implementovaný požadavek:** FP-GEN-2
**Fáze vývoje:** Úvodní fáze před nasazením
**Cílová verze:** 1.0.0
**Hlavní související MR:** !164

Tato změna se opět týká především aplikace `courses`, ale v omezené míře i aplikací `teams`, `checkpoints` a `assignments`. Je zde model `CourseSettings`[8], který reprezentuje konfiguraci předmětu. Původní implementace umožňovala definovat tuto konfiguraci pro samotný předmět a také pro jednotlivé učitele v tomto předmětu, pokud to správce předmětu povolil. Nově se konfigurace vztahují k předmětu, paralelce nebo týmu.

Součástí konfigurace jsou také kontrolní body reprezentované modelem `Checkpoint`[9]. Tento model definuje vlastnosti kontrolního bodu, jako například maximální počet získaných bodů a termín odevzdání. Je navázán na konfiguraci předmětu atributem `course_settings`, což je cizí klíč odkazující na model `CourseSettings`.

#### 3.3.2.1 Vazby mezi konfiguracemi a objekty, ke kterým se vztahují

Původně byly v `CourseSettings` definovány dva cizí klíče, atributy `course` a `teachership`, které odkazovaly na modely `Course` a `Teachership`[10]. Oba byly nepovinné a nastaven byl vždy právě jeden z nich. Oba tyto atributy jsem zrušil a nové vazby vytvořil z opačné strany – z modelu, k jehož instanci se konfigurace vztahuje. Do modelu `Course` přibyl atribut `default_settings`, do modelů `Parallel`[11] a `Team`[12] atribut `course_settings`, všechny reprezentují cizí klíč odkazující na model `CourseSettings`. U `Course` je atribut nastaven vždy, instance `CourseSettings` je vytvořena automaticky při vytvoření předmětu. V ostatních variantách se vytváří instance a nastavuje cizí klíč při změně granularity nastavení.

[8] `CourseSettings` je definován v souboru `apps/courses/models/settings.py`
[9] `Checkpoint` je definován v souboru `apps/checkpoints/models.py`
[10] Oba modely `Course` a `Teachership` jsou definovány v souboru `apps/courses/models/base.py`
[11] `Parallel` je definován v souboru `apps/courses/models/base.py`
[12] `Team` je definován v souboru `apps/teams/models/base.py`

**Výpis kódu 3.5** Úpravy modelu `Course` související s konfigurací

```
class Course(models.Model):
    ...

    class SettingsGranularityChoices(models.IntegerChoices):
        COURSE = 0, capfirst(_("course"))
        PARALLEL = 1, capfirst(_("parallel"))
        TEAM = 2, capfirst(_("team"))

    settings_granularity = models.PositiveSmallIntegerField(
        choices=SettingsGranularityChoices.choices,
        default=SettingsGranularityChoices.COURSE,
    )
    ...
    default_settings = models.ForeignKey(
        "courses.CourseSettings",
        related_name="course",
        on_delete=models.RESTRICT,
    )
    ...
```

#### 3.3.2.2 Granularita nastavení

Granularita nastavení je reprezentována novým atributem `settings_granularity`, definovaným v modelu `Course`, který nahradil původní atribut `is_custom_settings_allowed`. Jedná se o výčtový typ nabývající hodnot `COURSE`, `PARALLEL` a `TEAM`. Definici atributu ukazuji ve výpisu kódu 3.5. Stejně jako původní atribut jej nastavuje správce předmětu. Výchozí hodnotou je `COURSE`, tedy jedna konfigurace pro celý předmět. Další možností je `PARALLEL`, vlastní konfigurace pro každou paralelku. Při změně hodnoty z `COURSE` na `PARALLEL` je pro každou paralelku vytvořena kopie výchozí konfigurace předmětu. Poslední variantou je `TEAM`, vlastní konfigurace pro každý tým. Při nastavení této hodnoty je vytvořena kopie konfigurace pro každý tým. Kopíruje se buďto výchozí konfigurace předmětu, nebo konfigurace paralelky, do které tým patří, v závislosti na předchozím nastavení granularity. Kopie konfigurace pro tým je také vytvořena v okamžiku vytvoření nového týmu. Zvýšit granularitu lze popsaným způsobem vždy, snížit ji lze pouze v případě, že týmy ještě neodevzdaly žádné řešení. Nemuselo by totiž být možné odevzdaná řešení správně navázat na jednotlivé kontrolní body cílové konfigurace.

Při vytváření kopií konfigurace jsou také vytvořeny kopie všech kontrolních bodů souvisejících s kopírovanou konfigurací. Následně jsou vyhledána všechna odevzdání navázaná na kopírované kontrolní body a vazba je změněna na příslušné kopie.

#### 3.3.2.3 Další související změny

Popsané změny byly poměrně významným zásahem do fungování celé aplikace, který vyvolal nutnost dalších původně neplánovaných změn. První a nejjednodušší z nich bylo zrušení modelu `CustomDeadline`[13]. Tento model měl vazbu na kontrolní bod a paralelku a definoval termín odevzdání kontrolního bodu pro danou paralelku. Nově je možné definovat celou konfiguraci včetně kontrolních bodů pro každou paralelku zvlášť, tento model byl tedy redundantní.

Další potřebnou změnou bylo přidání vazby mezi týmem a paralelkou, realizovanou cizím klíčem `parallel` v modelu `Team`. Důvodem bylo to, že by jinak nebylo možné týmu přiřadit platnou konfiguraci předmětu v případě, že konfigurace jsou definované pro paralelky. Původně pro tým platila buďto konfigurace předmětu, nebo konfigurace vyučujícího odpovědného za zadání, na

[13] `CustomDeadline` byl definován v souboru `apps/checkpoints/models.py`

kterém tým pracuje. Nyní se již jen podle vyučujícího rozhodovat nelze, protože může učit více paralelek a bez explicitní vazby není jasné, do které z nich tým patří. Přiřadit tým k paralelce je pak také potřeba, pokud chce vyučující podle paralelky týmy filtrovat. Tuto možnost má na stránce předmětu.

Nakonec bylo potřeba vyřešit, kde bude definované společné zadání pro týmy v případě, že jej vyučující chce určit. Původně bylo určeno cizím klíčem `default_common_assignment` v modelu `Teachership` a aktivováno boolean atributem `common_assignment` v modelu `CourseSettings`. De facto tak bylo součástí konfigurace předmětu. Bylo by sice možné tento stav zachovat, myslím si ale, že by to pro uživatele bylo matoucí a zároveň jsem nepřišel na způsob, jak situaci uspokojivě vyřešit z hlediska uživatelského rozhraní. Proto jsem společné zadání skutečně navázal na konfiguraci – zrušil jsem atribut `common_assignment` a cizí klíč `default_common_assignment` jsem přesunul do `CourseSettings`. Tento cizí klíč je nepovinný, vyučující tedy společné zadání může neurčit tím, že nechá hodnotu společného zadání prázdnou, místo explicitní aktivace a deaktivace. V důsledku těchto úprav zůstal model `Teachership` čistě vazební tabulkou a je možné jej jako takový zrušit, Django ORM by vazební tabulku vygenerovalo samo i bez explicitní definice [38]. Nepovažoval jsem to ale za příliš důležité a bylo by potřeba upravit všechna místa v kódu, kde je tento model využíván k vytvoření nebo přečtení záznamu o přiřazení vyučujícího k předmětu. Model jsem tedy prozatím zachoval.

#### 3.3.2.4 Zjednodušení nalezení aktivní konfigurace

Díky popsaným změnám schématu bylo možné značně zpřehlednit a zjednodušit způsob, jakým jsou přiřazovány relevantní konfigurace jednotlivým týmům a studentům. Do modelu `Course` jsem přidal tři *properties*[14] pro ověření nastavení granularity. Dále jsem upravil metody pro získání konfigurace pro vybraný tým nebo studenta s využitím těchto properties. Pokud je student členem týmu, je mu automaticky přiřazena konfigurace pro tento tým, pokud ne, je vrácena konfigurace pro jeho paralelku nebo výchozí konfigurace předmětu, v závislosti na nastavení granularity. Změny ilustruje výpis kódu 3.6.

Dále jsem přidal *cached property* `active_settings` do modelu `Team`, pro efektivnější nalezení příslušné konfigurace. Hodnotu je možné jednoduše využít ve Views a Templates a při opakovaném použití v rámci zpracování jednoho HTTP požadavku již nejsou opakovány dotazy do databáze [39]. Definici je možné vidět na výpisu kódu 3.7.

#### 3.3.2.5 Migrace

Při modelových změnách v Django aplikaci je vždy nutné upravit databázové schéma spravované Django ORM pomocí tzv. *migrací*. V tomto případě je navíc potřeba transformovat existující data tak, aby mohla být uložena v novém schématu a nedošlo ke ztrátě uložených informací. K těmto účelům existují typy migrací - migrace schématu a datové migrace [40]. Migrace schématu mohou být a měly by být generovány automaticky spuštěním příkazu `manage.py makemigrations`. Datové migrace musí být vytvořeny ručně, protože Django nedokáže samo rozpoznat, jak by měla být existující data transformována do nového schématu.

V Django aplikaci jsou migrace uložené ve formě Python souborů v adresářích jednotlivých aplikací v podsložce `migrations`. Jednotlivé soubory jsou číslovány podle pořadí, ve kterém mají být v rámci aplikace provedeny. U migrací je také možné definovat závislosti mezi migracemi více aplikací. Pro realizaci potřebných změn jsem vytvořil migrace v aplikaci `courses` s pořadovými čísly 0008–0016 a v aplikaci `teams` s pořadovými čísly 0004 a 0005.

Při přechodu mezi dvěma způsoby uložení stejné nebo podobné informace pomocí migrací jsem postupoval tak, že jsem nejprve vygeneroval migraci schématu, která umožnila uložit data novým způsobem při zachování toho původního, následně jsem vytvořil datovou migraci, která transformovala data a uložila je novým způsobem a nakonec jsem opět vygeneroval migraci

[14] `https://docs.python.org/3/library/functions.html#property`

**Výpis kódu 3.6** Získání konfigurace relevantní pro tým nebo studenta

```
class Course(models.Model):
    ...

    @property
    def has_global_settings(self) -> bool:
        return self.settings_granularity == \
            Course.SettingsGranularityChoices.COURSE

    @property
    def has_parallel_settings(self) -> bool:
        return self.settings_granularity == \
            Course.SettingsGranularityChoices.PARALLEL

    @property
    def has_team_settings(self) -> bool:
        return self.settings_granularity == \
            Course.SettingsGranularityChoices.TEAM

    ...

    def get_settings_for_team(self, team):
        if self.has_team_settings:
            return team.course_settings

        if self.has_parallel_settings:
            return team.parallel.course_settings

        return self.default_settings

    def get_settings_for_student(self, student):
        if student.has_team(self):
            return self.get_settings_for_team(student.current_team(self))

        if self.has_parallel_settings:
            if not (parallel := self.get_parallel_for_student(student)):
                return None
            return parallel.course_settings

        # granularity == COURSE or TEAM and student doesn't have a team
        return (
            self.default_settings
            if self.is_user_student(student)
            else None
        )
    ...
```

**Výpis kódu 3.7** `cached_property` pro nalezení konfigurace pro tým

```
from django.utils.functional import cached_property

class Team(models.Model):
    ...

    @cached_property
    def course(self):
        return self.parallel.course

    @cached_property
    def active_settings(self):
        return self.course.get_settings_for_team(self)

    ...
```

schématu, která odstranila původní a nyní redundantní způsob uložení dat. Jako příklad mohu uvést migrace 0008–0010 aplikace `courses`, mění způsob uložení výchozí konfigurace předmětu z cizího klíče v modelu `CourseSettings` na cizí klíč v modelu `Course`. Nejprve je vytvořen nový cizí klíč v modelu `Course`, následně jsou podle původní vazby nastaveny hodnoty nového cizího klíče a v poslední migraci je původní cizí klíč v modelu `CourseSettings` odstraněn. Pro ilustraci ve výpisu 3.8 uvádím kód migrace číslo 0009, která transformuje transformuje data do nové podoby.

#### 3.3.2.6 Uživatelské rozhraní

Všechny popsané změny se musely promítnout i do uživatelského rozhraní, jak popisuji již v návrhu v sekci 2.1.2.2. Vytvořil jsem několik nových obrazovek: obrazovku přehledu nastavení obsluhovanou `CourseSettingsMainView`, obrazovku úpravy výchozího nastavení obsluhovanou `CourseSettingsDefaultView`, nastavení paralelky obsluhovanou `CourseSettingsParallelView` a nastavení týmu obsluhovanou `CourseSettingsTeamView`. Poslední dvě Views jsou ve smyslu Python tříd potomky `CourseSettingsDefaultView`, pouze definují vlastní způsob, jak získat upravovanou instanci konfigurace. Toto ilustruje výpis kódu 3.9. Všechny čtyři zmíněné Views jsou definovány v souboru `apps/courses/views/settings.py`. Snímek obrazovky úpravy nastavení týmu je na obrázku 3.6.

První zmíněné View, `CourseSettingsMainView`, neslouží k úpravě konfigurace v podobě instance modelu `CourseSettings`, ale k úpravě globálních atributů předmětu, které vždy upravuje pouze jeho správce a vztahují se na celý předmět. Jedním z těchto atributů je například granularita nastavení. K úpravě slouží formulář `CourseSupervisorForm`[15]. Vedle toho View navíc umožňuje například jmenovat další správce předmětu a zobrazuje aktuální hodnoty výchozí konfigurace předmětu a také seznam týmů nebo paralelek, kterým může uživatel upravovat konfigurace. Pokud obrazovku obsluhovanou tímto View navštíví vyučující, který není správce předmětu, zobrazí se mu pouze hodnoty výchozí konfigurace předmětu a seznam týmů a paralelek, jejichž konfigurace může upravovat. Globální atributy předmětu upravuje pouze správce. Snímek obrazovky přehledu nastavení se nachází na obrázku 3.5.

Kromě samotných konfigurací je také potřeba vytvářet a upravovat kontrolní body. Na všech třech obrazovkách, které slouží k úpravě konfigurace se nachází seznam kontrolních bodů, které může uživatel upravit a tlačítko pro přidání nového kontrolního bodu. Obrazovky a Views pro tvorbu a úpravu kontrolních bodů zůstaly bez výraznějších změn oproti původní implementaci.

[15] `CourseSupervisorForm` je definován v souboru `apps/courses/forms.py`

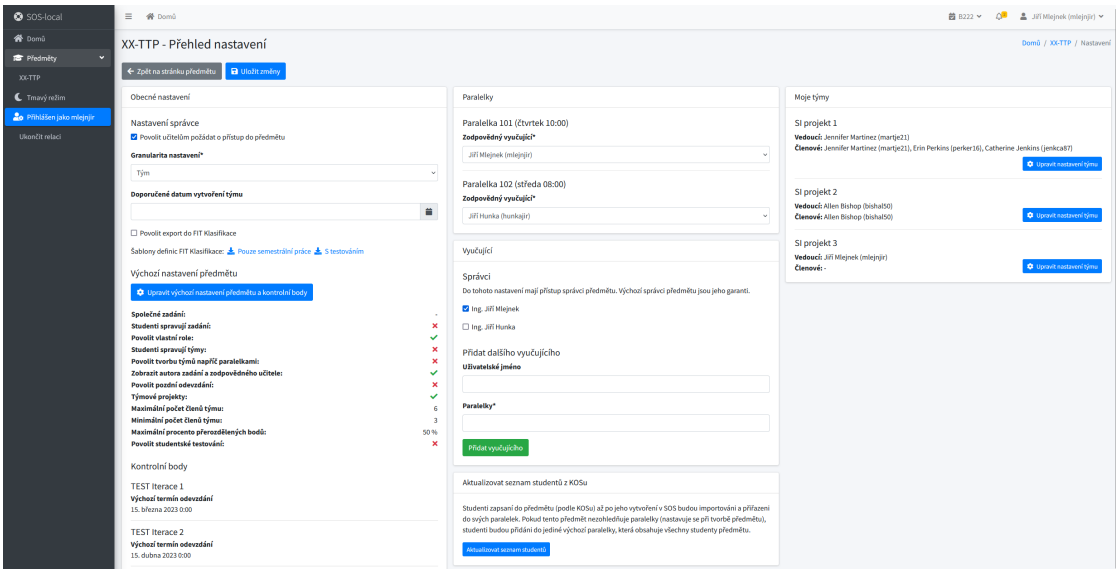
**Výpis kódu 3.8** Transformace způsobu uložení výchozí konfigurace předmětu v migraci č. 0009 aplikace `courses`

```
from django.db import migrations


def forwards_func(apps, schema_editor):
    Course_model = apps.get_model("courses", "Course")
    CourseSettings_model = apps.get_model("courses", "CourseSettings")

    db_alias = schema_editor.connection.alias

    all_courses, all_settings = [], []
    for course_settings in CourseSettings_model.objects.filter(
        course__isnull=False
    ):
        course = course_settings.course
        course_settings.course = None
        course.default_settings_x = course_settings

        all_courses.append(course)
        all_settings.append(course_settings)

    Course_model.objects.using(db_alias).bulk_update(
        all_courses, ["default_settings_x"]
    )
    CourseSettings_model.objects.using(db_alias).bulk_update(
        all_settings, ["course"]
    )


class Migration(migrations.Migration):
    dependencies = [
        ("courses", "0008_auto_20221210_1135"),
    ]

    operations = [
        migrations.RunPython(forwards_func, migrations.RunPython.noop)
    ]
```

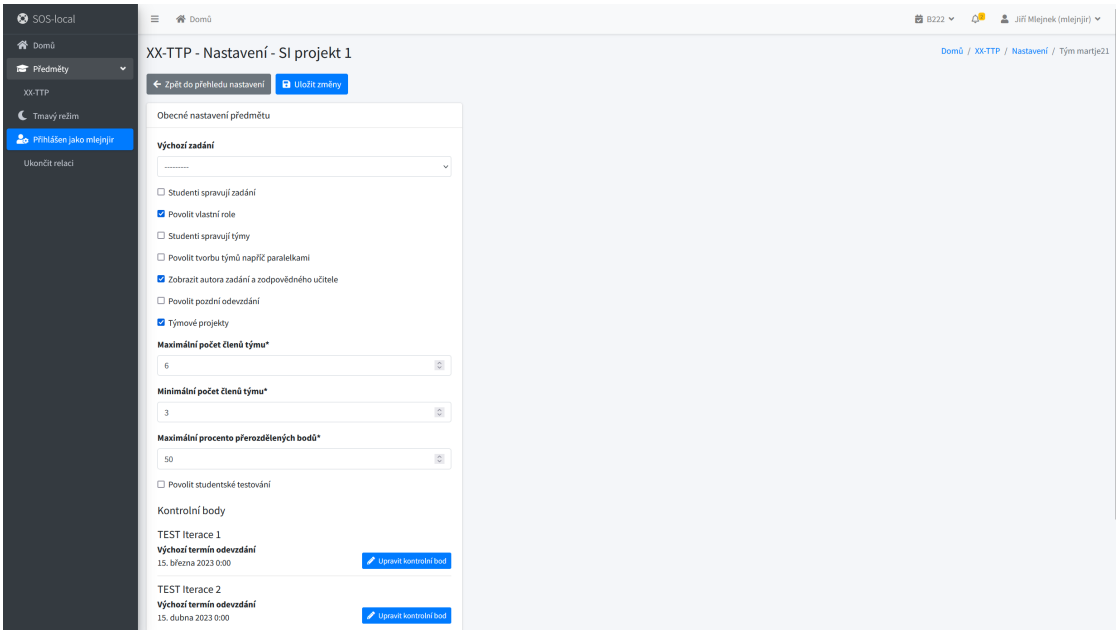
**Výpis kódu 3.9** Využití dědičnosti v definici Views pro úpravu konfigurace předmětu

```
class CourseSettingsDefaultView(..., CourseMixin, ...):
    settings_form_class = CourseSettingsForm

    ...

    def get_object(self):
        return self.course.default_settings

    ...


class CourseSettingsParallelView(CourseSettingsParallelView):
    ...

    @cached_property
    def parallel(self):
        return self.course.parallels.get(
            code=self.kwargs.get(self.slug_url_kwarg)
        )

    def get_object(self):
        return self.parallel.course_settings

    ...


class CourseSettingsTeamView(CourseSettingsDefaultView):
    ...

    @cached_property
    def team(self):
        return Team.objects.get(
            uuid=self.kwargs.get(self.slug_url_kwarg)
        )

    def get_object(self):
        return self.team.course_settings

    ...
```

**Obrázek 3.5** Obrazovka přehledu nastavení z pohledu správce




**Obrázek 3.6** Obrazovka úpravy nastavení týmu



**Výpis kódu 3.10** Many-to-many relace mezi týmem a asistenty

```
class Team(models.Model):
    ...

    assistants = models.ManyToManyField(
        settings.AUTH_USER_MODEL,
        related_name="assisted_teams",
        blank=True,
    )
    ...
```

### 3.3.3 Přístup k projektu pro další uživatele

**Implementovaný požadavek:** FP-SP-2
**Fáze vývoje:** Na začátku semestru, v pozdní fázi procesu tvorby týmů v BI-SP1 (únor 2023)
**Cílová verze:** 1.1.0
**Hlavní související MR:** !194

Přidání role asistenta bylo z implementačního hlediska jednou z nejjednodušších úprav. Jedinou modelovou změnou bylo vytvoření many-to-many relace mezi modely `Team` a `User`. Tuto změnu ilustruje výpis kódu 3.10. Vedle toho jsem implementoval pomocnou metodu pro získání všech týmů, kde je daný uživatel asistentem. Tato metoda se nachází ve třídě `TeamQuerySet`[16] a je implementována podobným způsobem, jako metody pro získání všech týmů za které odpovídá vybraný vyučující, nebo které patří k nějakému předmětu.

#### 3.3.3.1 Autorizace

Dále bylo potřeba zajistit, aby měl asistent na obrazovky související s daným týmem přístup. Realizaci v `TeamDetailView`, které obsluhuje obrazovku detailu týmu, zobrazuje výpis kódu 3.11. Toto View stejně jako řada dalších využívá `UserPassesTestMixin`, který umožňuje definovat metodu `test_func` sloužící k autorizaci uživatele [41]. Posledním místem, které bylo potřeba upravit byla metoda `can_user_access` modelu `Submission`[17], který reprezentuje odevzdané řešení. Metodu jsem upravil tak, aby vracela kladnou odpověď i v případě, že uživatel je asistentem týmu, který řešení odevzdal. Díky této změně má asistent nově přístup na obrazovku detailu odevzdaného řešení a může zobrazit i související přílohy – na obou místech se k autorizaci používá tato metoda.

#### 3.3.3.2 Přístup k souvisejícímu předmětu

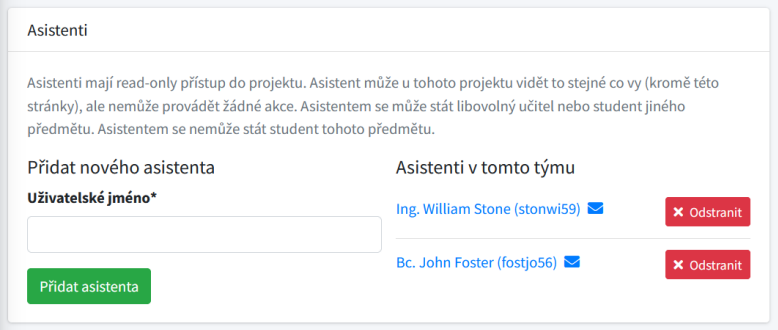
Asistent má sice přístup na obrazovky související s asistovaným týmem, nezískává tím ale přístup k ostatním částem sekce předmětu, ve kterém byl tým vytvořen. Tento přístup má pouze, pokud je zároveň vyučujícím daného předmětu. Vzniká tak situace, kdy je asistentem buďto vyučující nebo student jiného předmětu, který nemá do souvisejícího předmětu přístup. To má dva dopady – zaprvé by tomuto uživateli byla zobrazena chyba, pokud by na obrazovce detailu předmětu použil tlačítko „Zpět na stránku předmětu“, zadruhé by tento uživatel nemohl na tuto obrazovku přejít standardním způsobem ze obrazovky detailu předmětu. Vyřešil jsem to tak, že jsem zmíněné tlačítko z obrazovky detailu týmu odstranil, pokud `TeamDetailView` detekuje tuto situaci a na domovskou obrazovku (pro studenta i vyučujícího) jsem přidal seznam týmů, kde je uživatel asistentem, který zároveň obsahuje kódy souvisejících předmětů a odkazy pro zobrazení

[16] `TeamQuerySet` je definován v souboru `apps/teams/models/base.py`
[17] `Submission` je definován v souboru `apps/submissions/models/base.py`

**Výpis kódu 3.11** Přístup pro asistenta na detail týmu

```
class TeamDetailView(..., UserPassesTestMixin, ..., DetailView):

    model = Team

    def test_func(self):
        return self.is_user_teacher or \
            self.is_user_student or self.is_user_assistant

    @cached_property
    def is_user_assistant(self):
        return self.get_object().is_user_assistant(
            self.request.user
        )
    ...
```

**Obrázek 3.7** Karta „Asistenti“ na obrazovce správy týmu




jednotlivých týmů. Uživatel, který nemá přístup do předmětu jako takového, tedy přechází přímo mezi domovskou obrazovkou a detailem týmu.

#### 3.3.3.3 Správa asistentů

Asistenta může do týmu přidat vyučující, který je za tým odpovědný a má přístup na obrazovku správy týmu. Přichází to tedy v úvahu pouze v případě, že jsou týmy dle konfigurace spravovány vyučujícím, nebo je tým již schválen (schvalování týmů nahrazuje schvalování zadání a pojednává o něm sekce 3.3.5). Na obrazovku správy týmů byl dle návrhu přidán seznam aktuálních asistentů a formulář pro přidání nového asistenta, kde vyučující vyplní uživatelské jméno a potvrzením asistenta do týmu přidá. Karta se seznamem asistentů a formulářem pro přidání, která se nachází na obrazovce správy týmu, je zobrazena na obrázku 3.7.

Logiku přidání asistenta zajišťuje `TeamAddAssistantView`[18]. Tento View neobsluhuje žádnou obrazovku a přijímá pouze HTTP POST požadavky, které jsou odesílány pomocí zmíněného formuláře z obrazovky správy týmu. View během zpracování požadavku zkontroluje, zda již uživatel, který má být nastaven jako asistent, není asistentem daného týmu a zda se nejedná o vyučujícího odpovědného za tento tým, nebo studenta předmětu, ve kterém byl tým vytvořen. Pokud se uživatel může stát asistentem, pokusí se jej View podle zadaného uživatelského jména nalézt v databázi. Pokud neuspěje, pokusí se uživatele nalézt v KOS dotazem na KOSapi. V případě, že KOSapi vrátí potřebná chybějící data o uživateli, vytvoří View záznam o novém uživateli a pokračuje dál, stejně jako by u uživatel již v databázi existoval. V opačném případě

[18] `TeamAddAssistantView` je definováno v souboru `apps/teams/views/base.py`

skončí s chybou, která je uživateli zobrazena. Nakonec View přiřadí uživatele jako asistenta týmu a přesměruje uživatele zpět na stránku správy týmu.

Vyhledání uživatele v KOS a vytvoření nového záznamu v databázi je realizováno stejným způsobem, jako při prvním přihlášení uživatele. Používá se k tomu metoda `get_user_info` třídy `KosApi`[19] a metoda `create_from_oauth` třídy `UserQuerySet`[20]. Vyhledávání uživatele v KOS je nutné, protože asistentem se může stát i uživatel, který se nikdy předtím do SOS nepřihlásil a není studentem ani vyučujícím žádného předmětu, který byl do SOS importován. Takový uživatel v databázi SOS neexistuje.

Logiku odstranění asistenta zajišťuje `TeamDeleteAssistantView`[21]. To také obsluhuje pouze HTTP POST požadavky. Při zpracování požadavku View pouze zkontroluje, zda je uživatel určený uživatelským jménem v těle požadavku skutečně asistentem daného týmu a tuto vazbu mezi uživatelem a týmem odstraní. Uživatele pak přesměruje zpět na obrazovku správy týmu.

### 3.3.4 Vlastní role členů týmu

**Implementovaný požadavek:** FP-SP-3
**Fáze vývoje:** Úvodní fáze před nasazením, rozšířeno během semestru (duben 2023)
**Cílové verze:** 1.0.0, 1.4.0 (úprava zobrazení)
**Hlavní související MR:** !163[22], !225 (úprava zobrazení)[23]

Vlastní role členu týmu jsem implementoval dle návrhu s využitím asociativního pole, které je uloženo ve formátu JSON. Jak již zmiňuje i návrh, možnou zvláštností realizovaného řešení je fakt, že kapacity rolí nejsou žádným způsobem vynucovány. Je tedy možné přiřadit k roli více členů, než udává její kapacita. Není to tedy kapacita v pravém slova smyslu, jedná se spíše o požadovaný počet členů s danou rolí. Důvodem byla především časová úspora, protože bylo potřeba verzi SOS podporující vlastní role členů týmu nasadit do provozu před začátkem letního semestru 2022/23 a variantu, která spolehlivě validuje kapacity rolí bych implementovat nestihl. Nejednalo by se totiž pouze o kontrolu při pokusu o přiřazení role, například by také bylo nutné nějakým způsobem reflektovat existující členy týmu a čekající žádosti o členství v případě, že uživatel upraví kapacitu již definované role, případně validovat i zde a úpravu v určitých případech neumožnit. Tento a další potenciální problémy by vyžadovaly důkladnější analýzu tohoto požadavku, protože aktuálně není jasné, jak by měly být kapacity validovány a jaké dopady by to mělo například na uživatelské rozhraní. Místo toho jsem se rozhodl pro jednodušší implementaci, která může být případně časem upravena s využitím poznatků z jejího používání.

#### 3.3.4.1 Reprezentace a definice vlastních rolí

K uložení vlastních rolí členů týmu jsem se rozhodl použít *JSONField* [42]. Jeho hodnota je v PostgreSQL databázi ukládána jako datový typ `jsonb` [43], který slouží k ukládání JSON dokumentů a zároveň podporuje indexování. V Python kódu je hodnota reprezentována jako `dict`, což je asociativní pole [44]. Vlastní role členů týmu reprezentuje atribut `custom_roles` modelu `Team`, definici a příklad uložené hodnoty ukazuje výpis kódu 3.12. Definování vlastních rolí může vyučující dle požadavků povolit nebo zakázat, k čemuž slouží nový boolean atribut `allow_custom_roles` v modelu `CourseSettings`.

Vlastní role členů týmu může dle návrhu uživatel definovat na obrazovce správy týmu, kterou obsluhuje `TeamManageView`[24]. K tomuto účelu jsem vytvořil formulář `TeamCustomRoleForm`,

[19] Třída `KosApi` je definována v souboru `apps/oauth/data_provider/kos_api.py`
[20] Třída `UserQuerySet` je definována v souboru `apps/users/models.py`
[21] `TeamDeleteAssistantView` je definováno v souboru `apps/teams/views/base.py`
[22] Tento MR obsahuje i řadu dalších změn obsažených i v jiných MR.
[23] Autorem tohoto rozšíření je člen SP týmu.
[24] `TeamManageView` je definováno v souboru `apps/teams/views/base.py`

**Výpis kódu 3.12** Reprezentace vlastních rolí členů týmu

```
class Team(TimestampedModelMixin, models.Model):
    ...

    custom_roles = models.JSONField(
        verbose_name=_("custom roles"), blank=True, null=False, default=dict
    )
    """
    Example value / structure:

    {
        "backend-developer" {   # id is slugified user-provided label
            "label": "Backend Developer",   # display name of the role
            "members": [2, 34, 51], # primary keys of members with this role
            "wanted_count": 5   # how many members with this role are wanted
        },
        "project-manager" {
            "label": "Project Manager",
            "members": [3],
            "wanted_count": 1
        },
        ...
    }
    """
    ...
```

který umožňuje definovat popisek a počet členů, kteří mají mít tuto roli. Rolí je ale potřeba definovat několik najednou, proto jsem napsal ještě funkci `create_custom_role_formset`[25], která formulář převede na tzv. *Formset* [45]. Formset je abstrakce pro práci s více formuláři stejného typu na jedné stránce. Jak je definován formulář a Formset ukazuje výpis kódu 3.13, jeho použití k validaci a uložení dat v `TeamManageView` pak výpis kódu 3.14. Formulář pro definici rolí na obrazovce správy týmu je viditelný na obrázku 3.8.

### 3.3.4.2 Přiřazování rolí

Kromě samotné definice rolí je následně potřeba role přiřadit jednotlivým členům týmu. To může uživatel provést dvěma způsoby. První z nich má k dispozici taktéž na obrazovce správy týmu, v tabulce s jednotlivými členy týmu. Do tabulky jsem přidal sloupec „Vlastní role“, kde může uživatel vybrat roli pro každého člena týmu. Při výběru role je odeslán HTTP POST požadavek, který je následně zpracován `TeamSetMemberCustomRoleView`[26]. V těle požadavku je odesláno uživatelské jméno člena týmu a identifikátor role. View zkontroluje validitu požadavku, tedy jestli uživatelské jméno skutečně odpovídá některému ze členů týmu typu `LEADER` nebo `MEMBER` a jestli je role v týmu definována. Potom roli nastaví voláním metody `set_custom_role_for_member` modelu `Team` a uživatele přesměruje zpět na obrazovku správy týmu. Nový sloupec v tabulce je viditelný na obrázku 3.9

Druhý způsob, jak může být role přiřazena, je přímo při přidání nového člena týmu. Když student požádá o členství v týmu, může nově zvolit roli, o kterou má zájem. K dispozici má role, které jsou v týmu definovány. K tomuto účelu jsem do modelu `TeamJoinRequest`[27], který reprezentuje pozvánku do týmu nebo žádost o členství v týmu, přidal atribut `custom_role`. Jeho

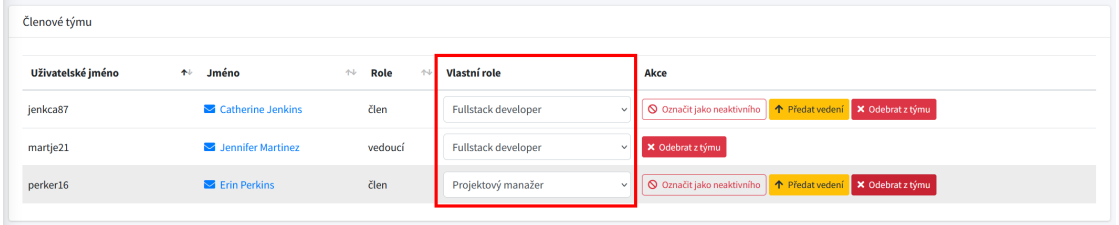
[25] `TeamCustomRoleForm` a `create_custom_role_formset` jsou definovány v souboru `apps/teams/forms.py`
[26] `TeamSetMemberCustomRoleView` je definován v souboru `apps/teams/views/base.py`
[27] `TeamJoinRequest` je definován v souboru `apps/teams/models/join.py`

**Výpis kódu 3.13** Formset pro definici vlastních rolí členů týmu

```
from django import forms
from django.forms import formset_factory


class TeamCustomRoleForm(forms.Form):
    label = forms.CharField(label=capfirst(_("role name")))
    wanted_count = forms.IntegerField(label=capfirst(_("member count")))


def create_custom_role_formset(request_data, json_data):
    if not request_data:
        role_count = len(json_data)
        formset_class = formset_factory(TeamCustomRoleForm, extra=role_count)
        formset = formset_class(None)
        for form, (key, data) in zip(formset, json_data.items()):
            form.initial = {
                "label": data.get("label", ""),
                "wanted_count": data.get("wanted_count", ""),
            }
        return formset
    else:
        return formset_factory(TeamCustomRoleForm)(request_data)
```

**Obrázek 3.8** Formulář pro definici vlastních rolí na obrazovce správy týmu

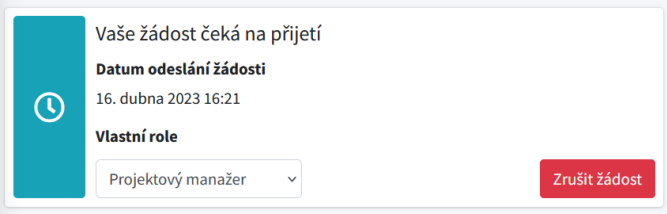

**Obrázek 3.9** Přiřazení vlastních rolí členům týmu v tabulce na obrazovce správy týmu



**Výpis kódu 3.14** Uložení vlastních rolí členů týmu

```
class TeamManageView(..., UpdateView):
    ...

    def get_custom_roles_formset(self):
        ...
        return create_custom_role_formset(
            self.request.POST or None,
            self.object.custom_roles,
        )

    def form_valid(self, form):
        team = self.object
        ...
        if self.course_settings.allow_custom_roles:
            formset = self.get_custom_roles_formset()

            if any(not f.is_valid() for f in formset):
                return self.form_invalid(form)

            roles_data = {
                slugify(f.cleaned_data["label"]): f.cleaned_data
                for f in formset
                if f.cleaned_data
            }
            team.update_custom_roles_data(roles_data)

        return super().form_valid(form)
```

**Obrázek 3.10** Karta s čekající žádostí o vstup do týmu z pohledu studenta/žadatele




hodnotu student nastaví na obrazovce detailu týmu, kam je přesměrován po odeslání žádosti o členství. Je zde karta s informacemi o jeho čekající žádosti, na níž může zvolit roli, o kterou má zájem. Po zvolení je role k pozvánce přiřazena obdobným způsobem, jako při nastavení role uživateli, který již je členem týmu, prostřednictvím `TeamSetRequestCustomRoleView`[28]. Při přijetí žádosti, která má definovanou roli, je tato role rovnou přiřazena novému členovi týmu, pokud je tato role v týmu definována. Pokud roli mezitím uživatel odpovědný za správu týmu například odstranil a tedy definována není, je role v žádosti ignorována člen týmu je přidán bez přiřazené role. Karta s čekající žádostí je na obrázku 3.10.

#### 3.3.4.3 Zobrazení přiřazených rolí

Kromě stránky správy týmu jsou vlastní role členů týmu zobrazeny i na obrazovce detailu týmu, na kartě „Tým“ v pravé části obrazovky v sekci „Členové“. V případě, že vlastní role nejsou v konfiguraci povoleny, je zde pouze jednoduchý seznam členů týmu. Pokud povoleny jsou, je seznam rozdělen na několik částí pro jednotlivé role. V poslední části seznamu jsou členové týmu, kteří žádnou roli přiřazenu nemají. Původně jsem kartu navrhl tak, že byly v kartě podle rolí rozděleni pouze členové týmu. Během používání se ale ukázalo, že by bylo vhodnější zde zobrazit i studenty, kteří o členství teprve požádali. Karta byla tedy takto upravena členem SP týmu, který se v rámci předmětu BI-SP1 na projektu podílel. Finální podoba karty je na obrázku 3.11.

Nakonec je role, o kterou student požádal, zobrazena také na kartě čekající žádosti o členství týmu, která je zobrazena uživateli, který ji může přijmout nebo odmítnout. Karta se může nacházet na obrazovce detailu týmu nebo na obrazovce detailu předmětu. Na tuto kartu jsem zároveň přidal informaci o kapacitě týmu, kapacitě dané role a upozornění, pokud je kapacita týmu nebo role již vyčerpána. V případě, že je vyčerpána kapacita týmu, žádost není možné přijmout. V případě, že je vyčerpána pouze kapacita dané role, žádost je možné i tak přijmout a roli přiřadit, ale uživatel je na tuto skutečnost upozorněn. Čekající žádost je zobrazena na obrázku 3.12.

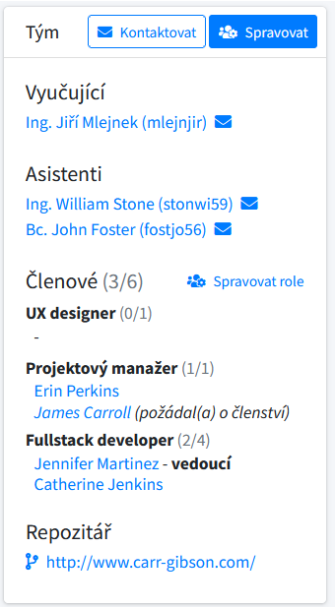
### 3.3.5 Schvalování týmů

**Implementovaný požadavek:** FP-NUR-1
**Fáze vývoje:** Úvodní fáze před nasazením
**Cílová verze:** 1.0.0
**Hlavní související MR:** !165

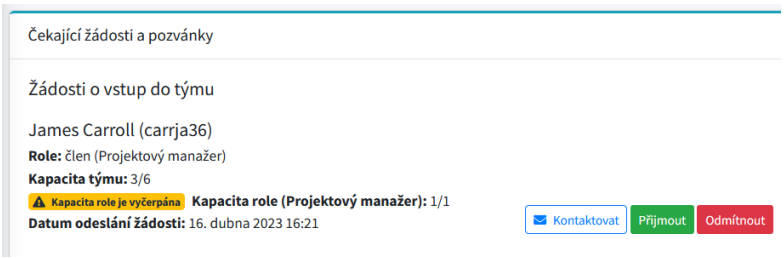
Implementace funkcionality schvalování týmu do jisté míry spočívala v úpravě stávajícího schvalování zadání. Nový model `TeamApprovalRequest`[29] jsem vytvořil přejmenováním původního modelu `AssignmentApprovalRequest`[30] a jeho přesunutím z aplikace `assignments` do aplikace `teams`. Dále jsem jej také rozšířil o nové atributy a metody dle návrhu. Pomocí datové migrace

[28] `TeamSetRequestCustomRoleView` je definován v souboru `apps/teams/views/join.py`
[29] `TeamApprovalRequest` je definován v souboru `apps/teams/models/base.py`
[30] `AssignmentApprovalRequest` byl definován v souboru `apps/assignments/models.py`

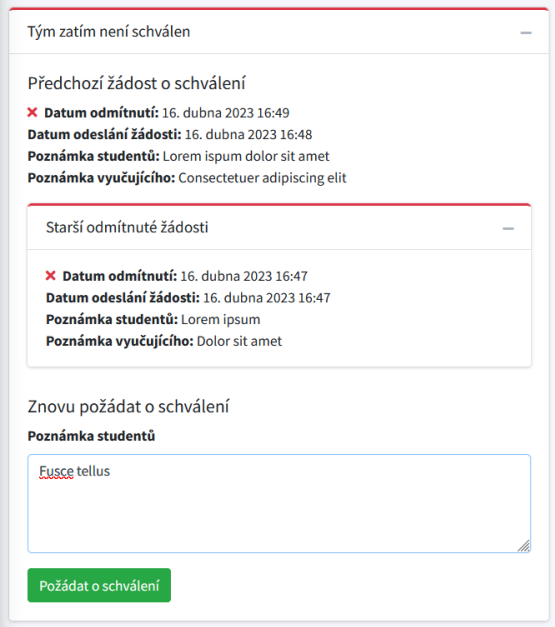
**Obrázek 3.11** Karta „Tým“ se zobrazením vlastních rolí




**Obrázek 3.12** Čekající žádost o vstup do týmu s upozorněním na překročení kapacity role



**Obrázek 3.13** Karta s formulářem pro žádost o schválení týmu z pohledu studenta




jsem transformoval existující schválení zadání na schválení týmů, které tato zadání vypracovávaly. Dále jsem do modelu `Team` přidal nový atribut `date_approved`, který jen duplikuje stejnojmenný atribut v modelu `TeamApprovalRequest`. To jsem učinil z důvodu optimalizace databázových dotazů – k získání informace o schválení týmů tak není potřeba vyhledávat příslušnou žádost, protože je informace uložena přímo u týmů. Hodnota je zkopírována v okamžiku schválení týmu.

Jedním z podstatných rozdílů oproti původnímu schvalování zadání je, že není žádost odeslána automaticky při vytvoření týmu, student tak činí ručně až v okamžiku, kdy chce tým prohlásit za kompletní. Učiní tak na obrazovce detailu týmu, kde nově přibyla karta s nadpisem „Tým zatím není schválen“ nebo „Tým je schválen“, v závislosti na stavu schválení. Na této kartě je k dispozici formulář, kde může student zadat textovou poznámku a požádat o schválení týmu. V případě, že o schválení v minulosti již požádal, zobrazí se na kartě i seznam předchozích žádostí a jejich stav. Vytvoření žádosti o schválení týmu obsluhuje `TeamDetailView`[31], které nově pro tento účel přijímá i HTTP POST požadavky. Stejně je implementováno i schválení či odmítnutí týmu vyučujícím. Ten na obrazovce vidí stejnou kartu jako student a má zde k dispozici obdobný formulář, tentokrát ale s možnostmi „Schválit“ a „Odmítnout“. Schválení či odmítnutí týmu je opět realizováno odesláním HTTP POST požadavku, který zpracuje `TeamDetailView`. U schváleného týmu je stále zobrazena karta s přehledem existujících žádostí o schválení, pouze je ve výchozím stavu sbalena a neobsahuje žádný formulář. Zpracování obou typů HTTP POST požadavku v `TeamDetailView` ilustruje výpis kódu 3.15. Karta s formulářem z pohledu studenta je zobrazena na obrázku 3.13, z pohledu vyučujícího na obrázku 3.14 a ve finální podobě kdy je tým schválený na obrázku 3.15.

Nakonec je potřeba na různých místech ověřovat, zda je již tým schválen a zda je to vůbec potřeba, aby mohl například odevzdat řešení. Celá logika schvalování týmů je totiž aktivní pouze v případě, že jsou týmy spravovány studenty. Jsou-li schvalovány vyučujícím, týmy schváleny být nemusí a tak jak jsem funkcionalitu implementoval ani nemohou. K zjištění stavu schválení jsem přidal do modelu `Team` několik properties, které jsou ukázány ve výpisu kódu 3.16.

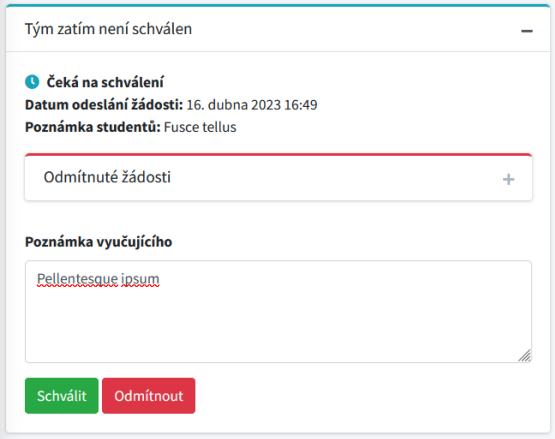
Výše jsem již popsal, že kompletní informace o stavu schválení týmu a související formuláře jsou zobrazeny v kartě na obrazovce detailu týmu. Vyučující má kromě toho na obrazovce detailu

[31] `TeamDetailView` je definováno v souboru `apps/teams/views/base.py`

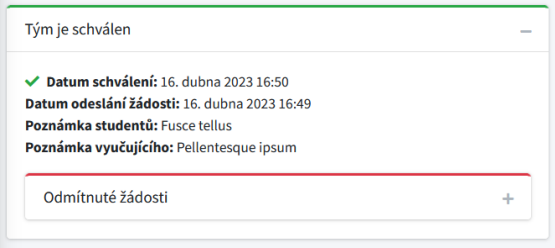
**Výpis kódu 3.15** Zpracování požadavků souvisejících se schvalováním týmu

```
class TeamDetailView(..., DetailView):
    ...

    def post(self, request, *args, **kwargs):
        self.form = self.get_approval_form()
        if self.form is None:
            raise PermissionDenied()

        team = self.get_object()
        if not self.form.is_valid():
            return self.render_to_response(
                self.get_context_data(object=team)
            )

        if self.is_user_teacher:
            action = self.request.POST.get("action", "").lower()
            teacher_note = self.form.cleaned_data.get("teacher_note", "")
            if action == "approve":
                accepted = team.latest_approval_request.accept(
                    teacher_note
                )
                ...    # notifikace uzivatele o (ne)uspechu
            elif action == "reject":
                team.latest_appproval_request.reject(teacher_note)
            else:
                raise BadRequest(...)
        elif self.is_user_student:
            requested = team.request_approval(
                self.form.cleaned_data.get("student_note", "")
            )
            ...    # notifikace uzivatele o (ne)uspechu
        else:
            raise PermissionDenied()

        return HttpResponseRedirect(team.get_absolute_url())
```

**Obrázek 3.14** Karta s formulářem pro schválení týmu s pohledu vyučujícího



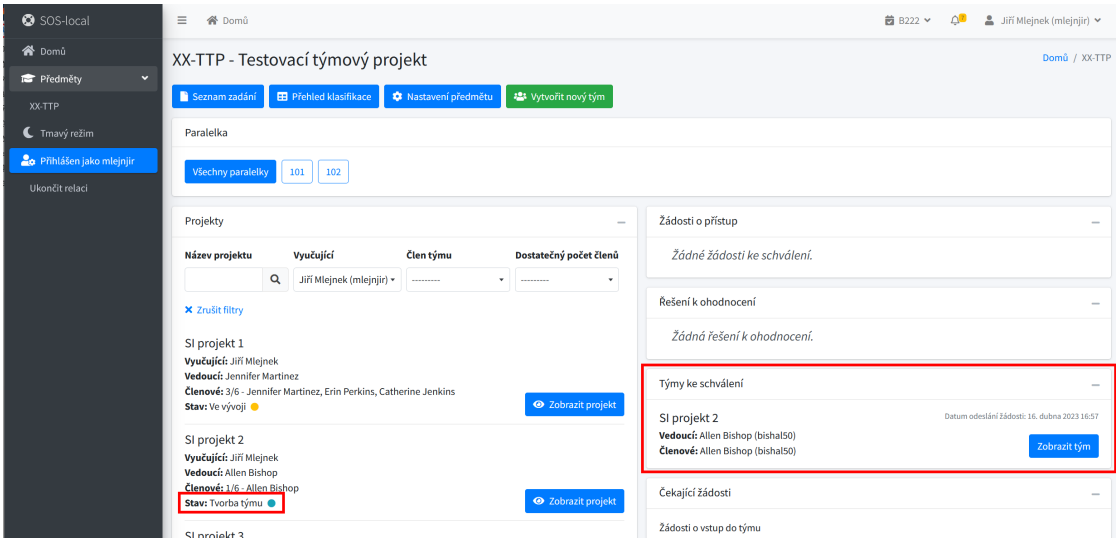

**Obrázek 3.15** Karta s informacemi o schválení týmu




**Výpis kódu 3.16** Zjištění stavu schválení týmu

```
class Team(TimestampedModelMixin, models.Model):
    ...

    @cached_property
    def is_approval_needed(self) -> bool:
        return self.is_student_managed

    @cached_property
    def is_student_managed(self) -> bool:
        return self.active_settings.student_managed_teams

    @property
    def is_approved(self) -> bool:
        return self.date_approved is not None

    @cached_property
    def can_submit(self) -> bool:
        return not self.is_approval_needed or self.is_approved

    @cached_property
    def latest_approval_request(self):
        return self.approval_requests.last()

    ...
```

**Obrázek 3.16** Seznam žádostí o schválení týmu na obrazovce detailu předmětu z pohledu vyučujícího




předmětu k dispozici kartu se seznamem týmů, které čekají na jeho schválení, jak je vidět na obrázku 3.16. Fakt, že tým čeká na schválení, je také indikován stavem „Tvorba týmu“ nebo „Čeká na schválení“, který je zobrazen u položek v hlavním seznamu týmu na obrazovce detailu předmětu a také na kartě „Můj tým“, kterou na této obrazovce vidi student v případě, že nějaký tým má.

### 3.3.6 Studentské testování

**Implementovaný požadavek:** FP-NUR-2
**Fáze vývoje:** V průběhu semestru (březen 2023)
**Cílová verze:** 1.2.0
**Hlavní související MR:** !214

Modifikace funkcionality studentského testování měla dvě hlavní části – upravit model tak, aby bylo potvrzení účasti na testování vázáno na tým jako takový místo na odevzdané řešení a odpovídajícím způsobem této nové skutečnosti přizpůsobit uživatelské rozhraní.

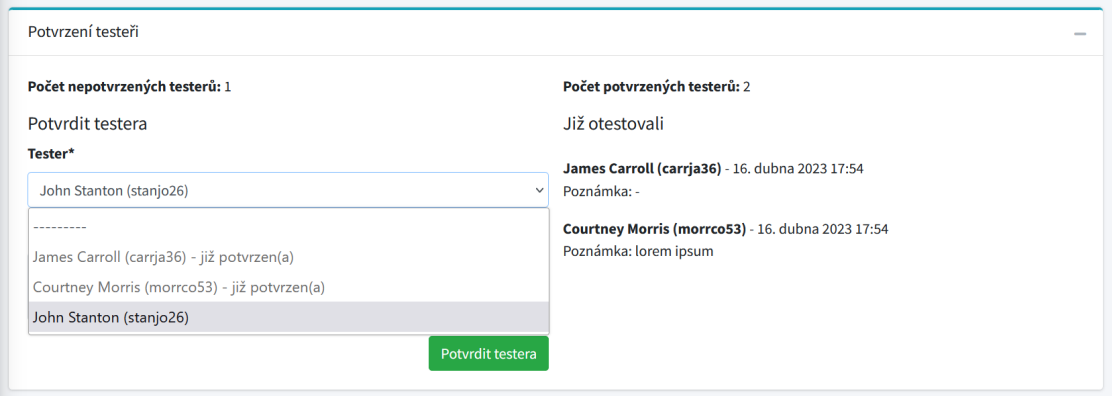
#### 3.3.6.1 Úpravy modelu a aplikační logiky

Potvrzení účasti na testování je realizováno modelem `TestConfirmation`[32]. Tento model jsem zachoval, pouze jsem jej přesunul z aplikace `submissions` do aplikace `teams`. Dále jsem do něj přidal atribut `note`, jenž reprezentuje textovou poznámku, kterou nově může vedoucí týmu při potvrzování uvést. Hlavní změna spočívala v odstranění cizího klíče `submission` odkazujícího na testované odevzdané řešení a jeho nahrazení novým cizím klíčem `team`, odkazujícím na tým, jehož práce byla předmětem testování.

Samotné potvrzení je realizováno `TestConfirmationView`[33], které jsem zachoval téměř v původní podobě a opět jsem jej pouze přesunul z aplikace `submissions` do aplikace `teams`. View přijímá pouze HTTP POST požadavky a jeho činnost spočívá ve validaci legitimity požadavků a vytvoření instance `TestConfirmation`, čímž je vytvořen záznam o potvrzení účasti na testování.

[32] `TestConfirmation` je definován v souboru `apps/teams/models/testing.py`
[33] `TestConfirmationView` je definováno v souboru `apps/teams/views/testing.py`

**Obrázek 3.17** Karta „Potvrzení testeři“ na obrazovce detailu týmu z pohledu vedoucího týmu




#### 3.3.6.2 Transformace existujících dat

Existující data jsem opět transformoval pomocí datové migrace[34]. Transformace u každého `TestConfirmation` spočívala v nastavení atributu `team` na hodnotu týmu, který odevzdal otestované řešení odkazovaného atributem `submission`. Potenciálně by mohla nastat situace, kdy by více potvrzení účasti na testování nově odkazovaly na stejný tým, protože původně souvisely s různými odevzdanými řešeními stejného týmu. Avšak z povahy předmětu NI-NUR, který jako jediný funkcionalitu dosud využil, taková situace v produkčních datech nenastala a nebylo tedy potřeba ji řešit.

#### 3.3.6.3 Uživatelské rozhraní

Dle návrhu jsem uživatelské rozhraní této funkcionality přesunul na obrazovku detailu týmu. Přidal jsem kartu „Potvrzení testeři“, která zobrazuje seznam potvrzených testerů s poznámkami a počty potvrzených a nepotvrzených testerů. Na kartě je také formulář, který se zobrazuje vedoucímu týmu, který má oprávnění testery potvrzovat. Ostatním uživatelům je místo něj zobrazen seznam nepotvrzených testerů. V případě, že ještě zbývají testeři k potvrzení, je karta ve výchozím stavu rozbalena a její záhlaví je obarveno modře. Pokud jsou již všichni přihlášení testeři potvrzení, je tento stav indikován zeleným obarvením záhlaví karty. Karta je také ve výchozím stavu sbalena. Formulář pro potvrzení účasti na testování obsahuje dvě pole – výběr testera a volitelnou textovou poznámku. Je definován třídou `TestConfirmationForm`[35]. Karta s otevřeným výběrem testera k potvrzení je na obrázku 3.17.

### 3.3.7 Přerozdělení bodů

**Implementovaný požadavek:** FP-SP-4
**Fáze vývoje:** Úvodní fáze před nasazením
**Cílová verze:** 1.0.0
**Hlavní související MR:** !163[36]

Jako první jsem přidal atributy konfigurace přerozdělení dle návrhu. V modelu `CourseSettings` reprezentujícího konfiguraci předmětu přibyl atribut `max_points_redistribution_percent` jehož kladná celočíselná hodnota udává maximální procento přerozdělení bodů. Výchozí hodnotou

[34] Migrace je definovaná v souboru `apps/submissions/migrations/0005_testconfirmation_team.py`
[35] `TestConfirmationForm` je definován v souboru `apps/teams/forms.py`
[36] Tento MR obsahuje i řadu dalších změn obsažených i v jiných MR.

**Výpis kódu 3.17** Příklad hodnoty přerozdělení bodů u odevzdaného řešení

```
{
    1: 10,    # clen tymu s primarnim klicem (id) 1 ziskava 10 % bodu navic
    22: 0,
    385: -20,
    97: 10
}
```

je 0, která značí, že přerozdělování není vůbec povoleno provádět. Druhým přidaným atributem je boolean `allow_points_redistribution` v modelu `Checkpoint` reprezentujícím kontrolní bod. Tento atribut určuje, zda je přerozdělování povoleno pro daný kontrolní bod.

#### 3.3.7.1 Reprezentace přerozdělení

Samotné přerozdělení je pak definováno atributem `points_redistribution` v modelu `Submission` reprezentujícím odevzdané řešení. Atribut je definován jako JSONField, jeho Python hodnotou je vždy `dict` a v databázi je uložen jako datový typ `jsonb`. Jedná se o stejný přístup, jako u implementace vlastních rolí členů týmu popsané v sekci 3.3.4. Struktura uložené hodnoty je ukázána ve výpisu 3.17. Jedná se o asociativní pole, jehož klíče jsou primární klíče uživatelů (členů týmu) a hodnoty jsou procenta přerozdělení v rozsahu -100–100.

#### 3.3.7.2 Definice přerozdělení

Přerozdělení bodů může upravovat student nebo vyučující na příslušných obrazovkách spolu s ostatními atributy odevzdaného řešení. Konkrétně student při odevzdávání nového řešení nebo úpravě existujícího a vyučující při ohodnocování. Pro manipulaci s přerozdělením bodů jsem vytvořil třídu `PointsRedistributionMixin`[37]. Tento *mixin* pak využívají celkem tři Views[38], které obsluhují příslušné obrazovky.

K získání a zpracování uživatelského vstupu mixin používá Formset, který je generovaný funkcí `create_redistribution_formset`[39]. Jedná se o sadu formulářů, jeden pro každého člena týmu, z nichž každý obsahuje číselné pole pro zadání procenta přerozdělení a skryté pole s identifikátorem uživatele.

#### 3.3.7.3 Uživatelské rozhraní

První změnou uživatelského rozhraní je přidání karty „Přerozdělení bodů“ na výše zmíněné tři obrazovky. Karta kromě formuláře pro úpravu přerozdělení obsahuje také stručné textové vysvětlení, jak přerozdělování funguje, informaci o maximálním procentu přerozdělení a aktuální součet zadaného přerozdělení, který je automaticky přepočítáván při změnách hodnot v číselných polích. Karta s formulářem pro definici přerozdělení je viditelná na obrázku 3.18. Obrazovka je zde v chybovém stavu, kdy se uživatel pokusil odeslat formulář s nenulovým součtem přerozdělení.

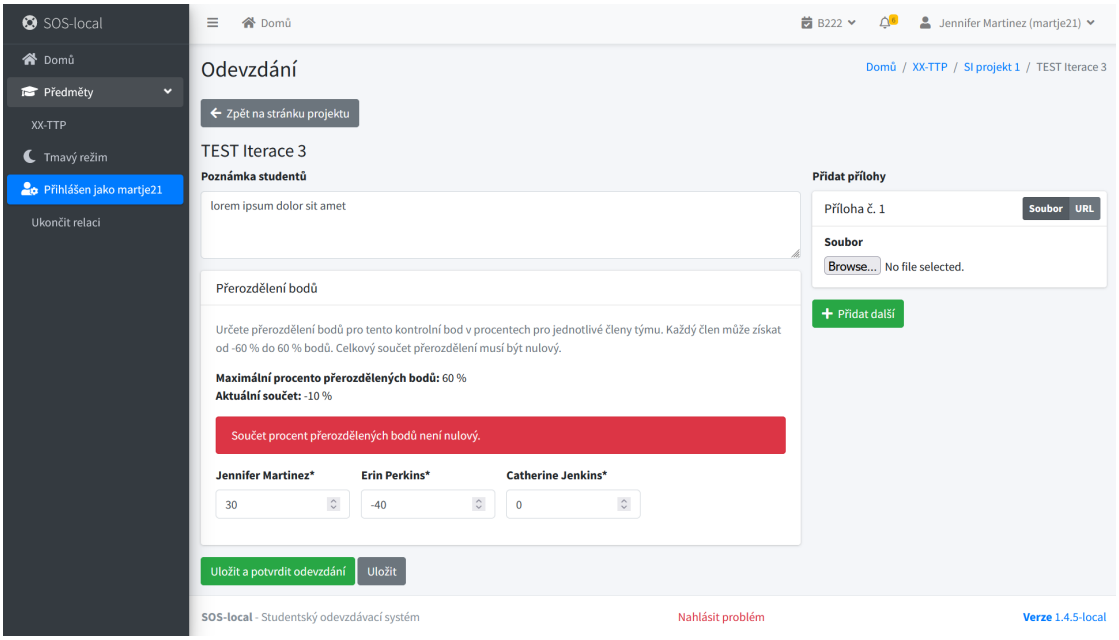
Dále jsem přidal na obrazovku detailu odevzdaného řešení kartu „Přerozdělení bodů“, na které je zobrazen seznam členů týmu a jejich procentuální a případně (pokud je řešení ohodnoceno) bodové zisky či ztráty v důsledku přerozdělení. Tato karta je zobrazena na obrázku 3.19. Také jsem rozšířil kartu „Body“ na obrazovce detailu týmu. Do ní jsem přidal sekci „Body členů týmu“, ve které je seznam členů týmu s jejich celkovými bodovými zisky. Pokud je aktivní přerozdělení bodů nebo studentské testování, bodový zisk týmu jako takového nemusí totiž být pro jednotlivé

[37] Třída `PointsRedistributionMixin` je definována v souboru `apps/submissions/views/base.py`

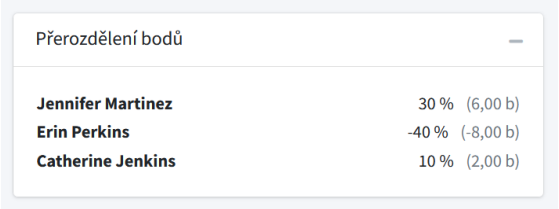
[38] `SubmissionCreateView`, `SubmissionUpdateView` a `SubmissionEvaluateView`, všechna definována v souboru `apps/submissions/views/base.py`

[39] `create_redistribution_formset` je definována v souboru `apps/submissions/forms.py`

**Obrázek 3.18** Obrazovka odevzdání řešení s formulářem pro definici přerozdělení bodů




**Obrázek 3.19** Karta zobrazující přerozdělení bodů na detailu ohodnoceného řešení




členy týmu vypovídající, jejich bodové hodnocení se může a pravděpodobně bude lišit. Karta je zobrazena na obrázku 3.20.

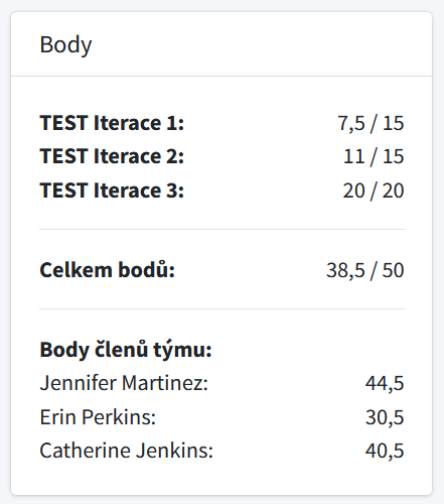
### 3.3.8 Statistiky předmětu

**Implementovaný požadavek:** FP-GEN-9
**Fáze vývoje:** Úvodní fáze před nasazením a bezprostředně po něm, rozšířeno během semestru
**Cílové verze:** 1.0.0, 1.0.1 (volná místa v týmech), 1.4.0 (kontrolní body)
**Hlavní související MR:** !167, !168 (volná místa v týmech), !230 (kontrolní body)

Implementace statistik má dvě hlavní části – získání dat a jejich zobrazení. K získání potřebných dat slouží `CourseStatsView`[40]. Zobrazení dat je pak na obrazovce detailu předmětu realizováno s využitím knihovny Chart.js[41], jež je součástí šablony AdminLTE, kterou uživatelské rozhraní SOS využívá.

[40] `CourseStatsView` je definováno v souboru `apps/courses/views/base.py`
[41] `https://www.chartjs.org/`

**Obrázek 3.20** Karta „Body“ na obrazovce přehledu týmu




#### 3.3.8.1 Získání dat

`CourseStatsView` přijímá HTTP GET požadavky se dvěma URI parametry – `parallel` a `show_other_teachers`. Oba parametry jsou nepovinné. Hodnotou prvního z nich je kód paralelky, pro kterou mají být statistiky vypočteny. Pokud hodnota chybí, jsou statistiky vypočteny pro celý předmět. Hodnota druhého parametru je buďto „true“ nebo „false“, v případě že chybí, uvažuje se druhá možnost. Ta slouží k určení, zda mají být do statistik zahrnuti pouze studenti, jejichž vyučujícím je přihlášený uživatel (tzn. buďto vyučuje jejich paralelku, nebo je odpovědný za jejich tým), nebo všichni studenti.

Celkem je třeba získat data pro tři různé statistiky. První je rozdělení studentů podle členství v týmu. Studenti jsou rozděleni do čtyř skupin - „Má tým“, „Má tým (neschválený)“, „Aktivní žádost o vstup do týmu“ a „Bez týmu“. View získá z databáze seznam studentů a jejich týmů a z něj zjistí velikosti jednotlivých skupin. Druhou statistikou je porovnání počtu studentů, kteří nemají tým, a počtu volných míst v existujících týmech. První informace byla již získána pro účely první statistiky, druhou View opět vypočte na základě seznamu všech týmů získaného z databáze.

Nakonec zbývají statistiky splnění jednotlivých kontrolních bodů – počty studentů, kteří jednotlivé kontrolní (ne)splnili. Zde nastává problém popsaný již v návrhu – různí studenti vypracovávají různé kontrolní body, kterých ani nemusí být stejný počet. Realizoval jsem tedy řešení také popsané v návrhu – kontrolní body jsem agregoval podle pořadí podle termínu odevzdání. To řeší metoda `get_classification` modelu `Course`, která je detailněji rozebrána v sekci 3.3.11 spolu s exportem hodnocení.

Kromě vypočtených statistik je součástí odpovědi ještě celkový počet studentů a týmů a seznamy e-mailových adres studentů s týmem a bez týmu.

View nevrací odpověď ve formě HTML dokumentu nebo přesměrování jako většina ostatních, místo toho odpovídá vypočtenými daty ve formátu JSON. Obsah odpovědi je tedy typu `application/json`. Příklad dat, která může View vrátit, je ve výpisu kódu 3.18.

#### 3.3.8.2 Zobrazení dat

Všechny popsané statistiky jsem implementoval jako koláčové grafy vytvořené knihovnou Chart.js. Grafy jsou zobrazeny uživateli na obrazovce detailu předmětu. Do šablony, která slouží k vykres-

**Výpis kódu 3.18** Příklad JSON dat pro zobrazení statistik předmětu

```
{
  "number_of_students": 10,
  "number_of_teams": 3,
  "students_and_teams": {
    "has_team": 5,
    "not_approved": 2,
    "pending_request": 2,
    "without_team": 1,
    "students_with_team_emails": [
      "student1@fit.cvut.cz",
      "student2@fit.cvut.cz",
      ...
    ],
    "students_without_team_emails": [
      "student3@fit.cvut.cz",
      "student4@fit.cvut.cz",
      ...
    ]
  },
  "free_slots": {
    "free_students": 3,
    "free_slots": 8
  },
  "checkpoints": [
    {
      "completed": 5,
      "not_completed": 5
    },
    {
      "completed": 0,
      "not_completed": 10
    }
  ]
}
```

lení této stránky, jsem vložil[42] dvě šablony – `content/stats.html` a `js/stats.html`[43]. V první z nich je definováno rozmístění grafů na stránce. Jsou zde definovány elementy `<canvas>`, které dle dokumentace Chart.js [46] mají sloužit k vykreslení grafů. Dále jsou zde také dvě tlačítka pro kontaktování studentu s týmem nebo bez týmu e-mailem.

V druhé šabloně je uvnitř HTML tagu `<script>` kód v jazyce JavaScript, který se stará o získání dat odesláním HTTP GET požadavku na `CourseStatsView` a přečtením odpovědi a následnou inicializaci koláčových grafů. Data jsou získána asynchronně pomocí Fetch API [47]. K požadavku jsou přidány URI parametry `parallel` a `show_other_teachers` z původního požadavku, který vedl k vykreslení stránky, na které se uživatel právě nachází. Důvodem, proč jsem zvolil toto na první pohled komplikovanější asynchronní řešení, místo varianty, kdy by byly statistiky vypočteny přímo v `CourseDetailView`, které obsluhuje obrazovku detailu předmětu a zobrazeny ihned při načtení stránky, je rychlost – výpočet statistik může v případě většího množství studentů trvat i menší jednotky sekund a tak dlouhá odezva systému na tak kritické obrazovce jako je detail předmětu pro vyučujícího by byla nepřijatelná.

Po získání dat ve formátu JSON popsaném v předchozí sekci jsou s jejich využitím pomocí Chart.js vytvořeny koláčové grafy pro jednotlivé statistiky. Inicializace grafů probíhá standardním způsobem dle dokumentace Chart.js. Pro statistiky jednotlivých kontrolních bodů musí být navíc nejprve dynamicky vytvořeny `<canvas>` elementy, protože není dopředu znám jejich počet. Kromě koláčových grafů jsou také na stránku také doplněny informace o celkových počtech studentů a týmů a do tlačítek v sekci „Kontaktovat studenty“ jsou přidány `mailto` odkazy s emailovými adresami studentů. Karta „Statistiky“, která na obrazovce detailu předmětu zobrazuje popsané statistiky vyučujícímu, je na obrázku 3.21.

### 3.3.9 Vyhledávání a filtrování projektů

**Implementovaný požadavek:** FP-GEN-6
**Fáze vývoje:** Úvodní fáze před nasazením a bezprostředně po něm
**Cílové verze:** 1.0.0, 1.0.1 (filtrování podle kapacity)
**Hlavní související MR:** !163[44], !168 (filtrování podle kapacity)

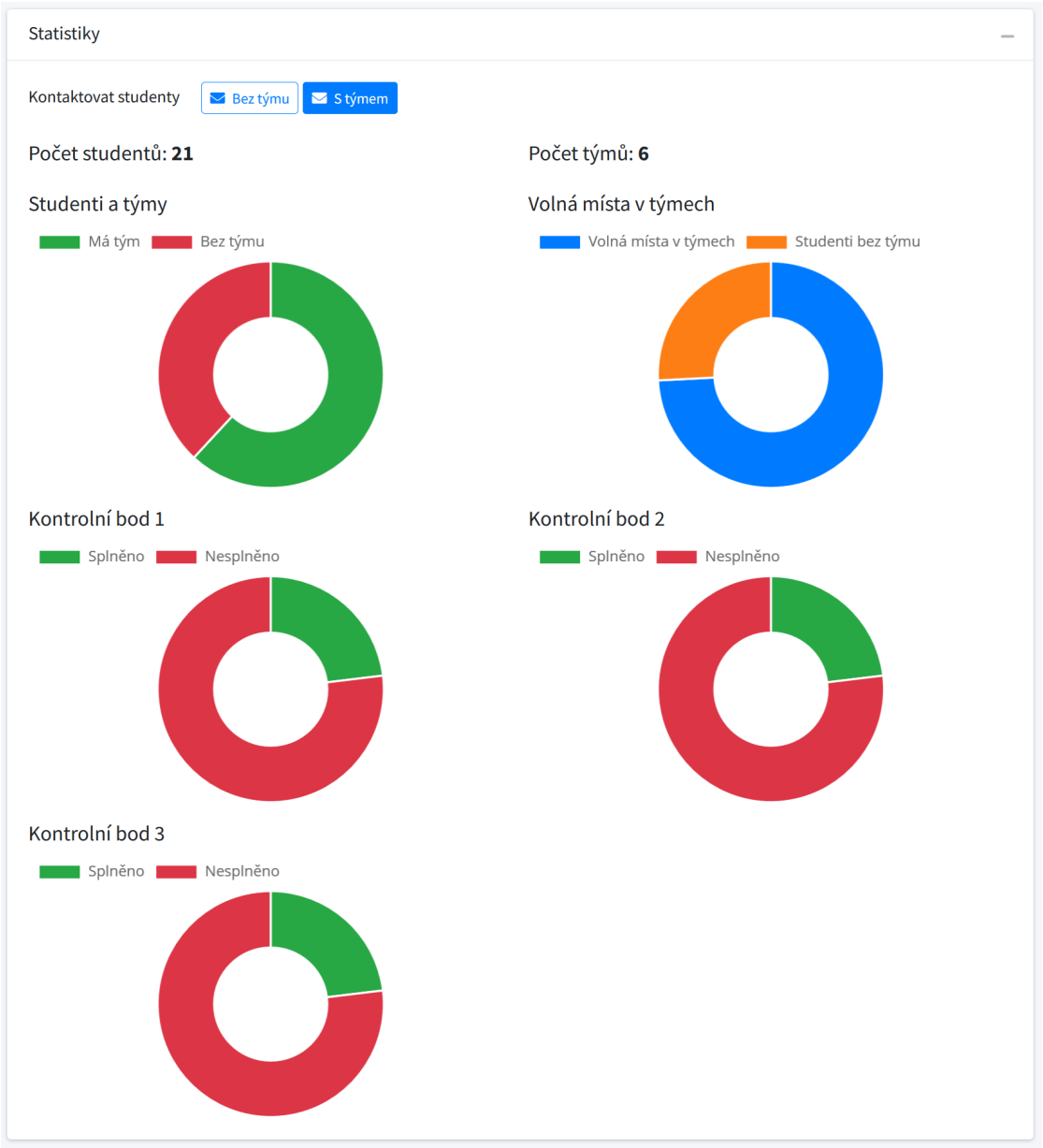
Vyhledávání a filtrování týmů/projektů je realizováno na obrazovce detailu předmětu v obou variantách – pro studenta i pro vyučujícího. Tuto obrazovku obsluhuje `CourseDetailView`, který také implementuje veškerou potřebnou logiku. Na obrazovce měl již původně vyučující možnost filtrovat projekty podle paralelky do které patří. To bylo užitečné například v předmětu NI-NUR, nicméně nevyužitelné v předmětech BI-SP1 a BI-SP2, které paralelky vůbec vypsané nemají. Jedním z nedostatků původní implementace bylo také to, že nebyl filtr žádným způsobem zachován v případě, že uživatel obrazovku opustil a později se na ni vrátil.

#### 3.3.9.1 Možnosti vyhledávání a filtrování

Na kartu se seznamem projektů jsem přidal pole pro vyhledávání podle názvu, člena týmu a odpovědného vyučujícího. Poslední pole se objeví v závislosti na typu uživatele. Student má možnost vyfiltrovat pouze týmy, do kterých se může přihlásit – tedy ty, které ještě mají volnou kapacitu a mají povoleny žádosti o členství. Vyučující zase může filtrovat týmy podle toho, zda již mají dostatečný počet členů. Smyslem této funkcionality je dát vyučujícímu způsob, jak identifikovat týmy, u kterých hrozí, že se nenaplní a případně některé z nich zrušit, aby se studenti přesunuli do jiných týmů, které tím dostatečný počet členů získají. Správce předmětu vidí ve výchozím stavu všechny projekty, ostatní vyučující vidí pouze ty, za které odpovídají a

[42] Django šablony je do sebe možné vkládat pomocí tagu `include`
[43] Do šablony `apps/courses/templates/courses/teacher_detail.html` jsou vloženy šablony `apps/courses/templates/courses/inc/content/stats.html` a `apps/courses/templates/courses/inc/js/stats.html`
[44] Tento MR obsahuje i řadu dalších změn obsažených i v jiných MR.

**Obrázek 3.21** Karta „Statistiky“ na obrazovce detailu předmětu




*Poznámka: Číselné hodnoty jednotlivých položek grafů se zobrazí při najetí kurzorem myši nebo při dotyku v případě dotykové obrazovky.*

navíc mají k dispozici zaškrtávací pole „Zobrazit projekty ostatních vyučujících“. Zaškrtnutím se jim zobrazí i ostatní projekty a zpřístupní se jim filtrování podle odpovědného vyučujícího.

Formuláře jsou definovány pomocí několika tříd[45] v souboru `apps/courses/forms.py`. Do kontextu vykreslované šablony jsou vkládány dynamicky podle typu uživatele a aktuálních parametrů vyhledávání.

#### 3.3.9.2 Implementace vyhledávání

Do View jsou všechny parametry vyhledávání předány pomocí URI parametrů. Děje se tak automaticky když uživatel některou hodnotu změní – v tom okamžiku je odeslán nový HTTP GET požadavek, který již obsahuje aktuální hodnoty parametrů, požadavek je zpracován a stránka je znovu načtena, již se správnými daty.

Prvním krokem během zpracování požadavku ve View je uložení hodnot parametrů, případně načtení již uložených hodnot v případě, že požadavek žádné parametry neobsahoval. Požadavek navíc může obsahovat speciální parametr `clear`, který nabývá hodnot „search“ a „parallel“ a slouží k vymazání uložených filtrů. Tento parametr je do požadavku vložen kliknutím na jedno z tlačítek „Všechny paralelky“ nebo „Zrušit filtry“. K uložení hodnot parametrů View využívá „SessionMiddleware“, který slouží právě k uchovávání stavu systému pro přihlášeného uživatele napříč požadavky [48]. Jinak je totiž Django navrženo jako *stateless* – žádný stav v paměti uchováván není. Do session jsou ukládány hodnoty jednotlivých parametrů pro každý předmět separátně. Uživatel tedy může mít uloženy různé filtry pro různé předměty, navzájem se neovlivňují. O uvedenou práci s filtry se v `CourseDetailView` stará metoda `_handle_filters`.

Nakonec je potřeba využít hodnoty parametry k samotnému vyfiltrování projektů, které mají být uživateli zobrazeny. Seznam projektů je ve View v průběhu zpracování požadavku uchováván v atributu `projects`. Nejprve je hodnotou tohoto atributu tzv. QuerySet, který uchovává konstruovaný dotaz do databáze pomocí Django ORM [49]. V první fázi jsou do konstruovaného dotazu přidány některé podmínky vyplývající z hodnot parametrů, a to v metodě `_apply_filters`. V konečné fázi zpracování požadavku View použije metodu `_apply_qs_breaking_filters`. Ta QuerySet převede na obyčejný seznam Python objektů skutečným provedením databázového dotazu. Dále pak ještě na tento seznam aplikuje zbytek filtrů definovaných parametry. Některé části vyhledávání jsem totiž nedokázal efektivně implementovat na straně databáze pomocí Django ORM a musel jsem je tedy řešit na úrovni aplikační logiky. Konkrétně se jedná o filtry podle možnosti přihlášení se k projektu a podle dosažení minimálního požadovaného počtu členů.

Seznam projektů uložený v atributu `projects` je pak zobrazen uživateli stejně jako v původní implementaci, jediným rozdílem je, že nyní obsahuje pouze ty položky, které odpovídají filtrům. Karta se seznamem projektů s aktivním filtrováním na obrazovce detailu předmětu z pohledu studenta je zobrazena na obrázku 3.22.

### 3.3.10 Aktualizace seznamu studentů existujícího předmětu

**Implementovaný požadavek:** FP-GEN-3
**Fáze vývoje:** Úvodní fáze před nasazením, rozšířeno během semestru (únor 2023)
**Cílové verze:** 1.0.0, 1.0.7 (odstraňování studentů)
**Hlavní související MR:** !163[46], !187

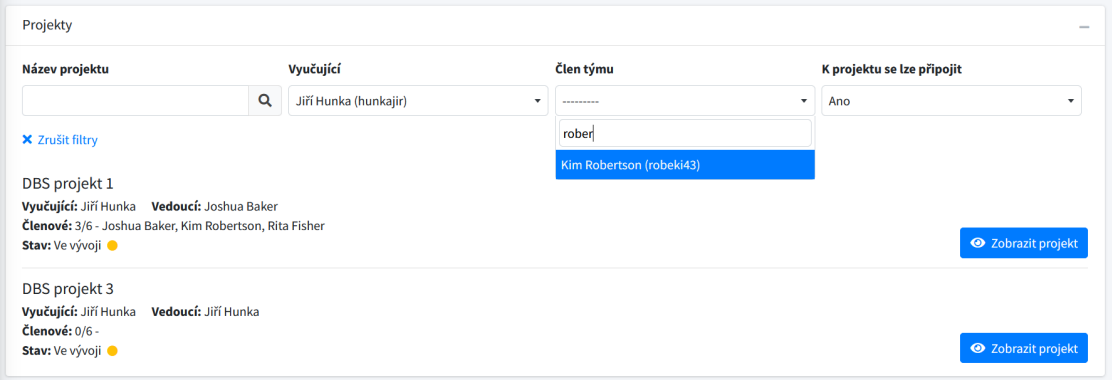
Aktualizaci seznamu studentů existujícího předmětu řeší `CourseRefreshView`[47], které přijímá HTTP POST požadavky a vrací odpověď typu `application/json`. Logika zpracování požadavku odpovídá návrhu – pro všechny paralelky definované v předmětu (nebo pro celý předmět

[45] `StudentSearchForm`, `TeacherSearchForm`, `NameSearchForm`, `OtherTeachersForm`, `MemberCountSearchForm`, `AvailableSearchForm`

[46] Tento MR obsahuje i řadu dalších změn obsažených i v jiných MR.

[47] `CourseRefreshView` je definováno v souboru `apps/courses/views/settings.py`

**Obrázek 3.22** Filtrování projektů z pohledu studenta




**Výpis kódu 3.19** Příklad odpovědi `CourseRefreshView`

```
{
    "type": "success",
    "message": "Added 8 students. No students removed."
}
```

najednou, v případě, že definovány nejsou) je z datového zdroje získán seznam studentů, který je následně porovnán se studenty uloženými v databázi SOS. Chybějící studenti jsou pak přidáni, přebývající jsou odebráni, pokud už nejsou členy nějakého týmu. Nakonec je vrácena odpověď ve formátu zobrazeném na výpisu kódu 3.19. Odpověď obsahuje informaci o úspěchu nebo neúspěchu operace a zprávu s počtem přidaných/odebraných studentů a seznamem studentů, kteří nebyli odebráni kvůli členství v týmu, která je určená k zobrazení uživateli.

Aktualizaci seznamu studentů může v SOS provést správce předmětu. Akci vyvolá na obrazovce přehledu konfigurace, kam jsem pro tyto účely přidal kartu „Aktualizovat seznam studentů“. Na kartě je text se stručným popisem funkcionality a tlačítko „Aktualizovat seznam studentů“. Po kliknutí na tlačítko je pomocí JavaScriptu odeslán HTTP POST požadavek na `CourseRefreshView`, po obdržení odpovědi je na kartu opět pomocí JavaScriptu přidána zpráva o provedené aktualizaci nebo o chybě. Karta i se skripem pro odeslání požadavku a zobrazení odpovědi je definována v šabloně `refresh.html`[48]. Karta ve stavu po úspěšně provedené aktualizaci je zobrazena na obrázku 3.23.

## 3.3.11 Export hodnocení obecně

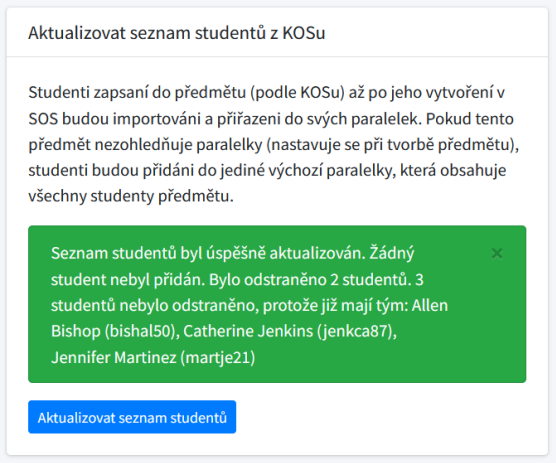
Pro realizaci exportu hodnocení jsem vytvořil novou aplikaci `export`. Aplikace neobsahuje žádné Modely, pouze Views, Templates a modul `exporter`, který implementuje veškerou logiku exportování hodnocení. Struktura aplikace je znázorněna na obrázku 3.24.

### 3.3.11.1 Získání potřebných dat

Aby mohlo být hodnocení exportováno, je nejprve potřeba zkompletovat potřebná data. Toto již v původní implementaci řešila cached property `classification` v `CourseClassificationView`[49]. Její hodnotou byl `dict`, který obsahoval informace o kontrolních bodech a seznam studentů spolu s jejich bodovými zisky za jednotlivé kontrolní body, za účast na testování a celkově za předmět.

[48] `apps/courses/templates/courses/inc/content/refresh.html`
[49] `CourseClassificationView` je definováno v souboru `apps/courses/views/base.py`

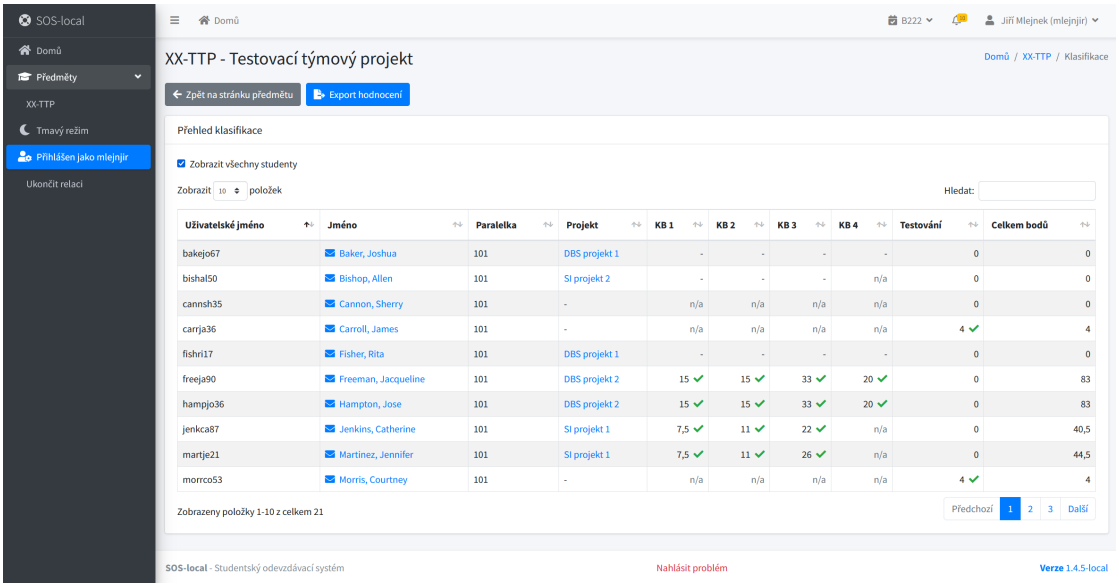
**Obrázek 3.23** Karta obsluhující aktualizaci seznamu studentů na obrazovce přehledu konfigurace




**Obrázek 3.24** Struktura aplikace `export`

```
export/
├── exporter/
│   ├── tests/ ..... Unit testy modulu exporter
│   ├── base.py ..... Implementace abstraktní třídy BaseExporter
│   ├── grades.py ..... Implementace exportu do KOS
│   └── kos.py ..... Implementace exportu do Klasifikace
├── templates/ ..... HTML a JS šablony
├── tests/ ..... Unit testy aplikace
├── urls.py ..... Definice směrování požadavků
└── views.py ..... Views pro obsluhu exportů
```

**Obrázek 3.25** Obrazovka přehledu klasifikace




Abych mohl implementaci využít, přesunul jsem kód do metody `get_classification`, kterou jsem vytvořil v modelu `Course`. Metoda má nově nepovinný argument `teacher`, který slouží k získání dat o hodnocení studentů pouze pro vybraného vyučujícího. Novou metodu využívá původní property v `CourseClassificationView` pro zachování původní funkcionality a nově i aplikace `export` a také `CourseStatsView` pro výpočet statistiky splnění kontrolních bodů, jak jsem popsal v sekci 3.3.8.1. Kromě přesunutí kódu jsem jej také patřičně upravil, aby reflektoval všechny modelové změny, zejména granularitu nastavení a studentské testování navázané přímo na týmy namísto odevzdaných řešení. Také je zde řešena agregace kontrolních bodů pro různé konfigurace – v principu jde o to, že jsou získány všechny relevantní konfigurace předmětu a u každé jsou její kontrolní body seřazeny podle data odevzdání. Ke každému studentovi je pak vytvořen seznam záznamů o kontrolním bodu, jehož položky jsou počty získaných bodů a informace, zda kontrolní bod splnil. Do tohoto seznamu je zapisováno podle indexu kontrolního bodu v uvedeném pořadí. Relevantní části kódu metody `get_classification` jsou ukázány ve výpisu 3.20.

### 3.3.11.2 Přehled klasifikace

Vzhledem ke všem popsaným změnám bylo potřeba upravit také obrazovku přehledu klasifikace, která která zobrazuje uživateli data, která mohou být následně exportována do dalších systémů. Data jsou získávána metodou `get_classification` modelu `Course` popsanou v předchozím odstavci. Obrazovka nově zohledňuje rozdílné počty kontrolních bodů dle výše popsané logiky. Ve sloupcích pro kontrolní bod který se ke studentovi v daném řádku nevztahuje je uvedena místo bodového ohodnocení hodnota „n/a“ (*not applicable*). Jinak je uvedeno vždy bodové ohodnocení daného studenta nebo „-“, pokud tým studenta své řešení zatím neodevzdal. Zelená značka se v buňkách tabulky zobrazuje, pokud dané řešení bylo ohodnoceno alespoň minimálním požadovaným počtem bodů (tato logika záměrně nezohledňuje přerozdělení bodů).

Pokud je uživatel garantem předmětu, zobrazí se mu nad tabulkou pole „Zobrazit všechny studenty“, jehož zaškrtnutím může místo svých studentů zobrazit všechny studenty předmětu. Toto nastavení se později bude vztahovat i na manuální exporty popsané dále. Poslední změnou v tabulce je pak přidání odkazů na detaily týmů a `mailto` odkazů pro kontaktování jednotlivých studentů. Obrazovka přehledu klasifikace je ukázána na obrázku 3.25.

**Výpis kódu 3.20** Agregace kontrolních bodů v modelu `Course`

```
def get_classification(self, teacher=None) -> dict:
    ...
    students_dict = ... # klice = PK studentu, hodnoty = instance User
    all_teams = ... # QuerySet vsech tymu
    team_settings_dict = ... # klice = PK tymu, hodnoty = CourseSettings
    ...
    for settings_ in set(team_settings_dict.values()):
        settings_.ordered_checkpoints = list(
            settings_.checkpoints.order_by("default_deadline")
        )
    ...
    for team in all_teams:
        ordered_checkpoints = team_settings_dict[team.pk].ordered_checkpoints
        order_mapping = {c.pk: i for i, c in enumerate(ordered_checkpoints)}

        for member in team.actual_members_with_activity:
            if (student := students_dict.get(member.pk)) is None:
                continue
            ...
            student.total_points = team.member_earned_points[member]
            student.checkpoint_count = len(team.checkpoints)
            for c in team.checkpoints:
                points = ... # body clena tymu po prerozdeleni

                try:
                    student.checkpoint_points[order_mapping[c.pk]] = (
                        points,
                        c.earned_points is not None
                        and c.earned_points >= c.min_points
                    )
                except IndexError:
                    pass
    ...
    return {
        "students": list(students_dict.values()),
        "checkpoints": [
            f"CHKP {i + 1}" for i in range(max_checkpoint_count)
        ],
        "is_any_testing_enabled": is_any_testing_enabled,
    }
```

**Tabulka 3.1** Stupnice hodnocení používaná při exportu do KOS

| Známka | Bodové rozmezí |
|---|---|
| A | 90 a více |
| B | 80 až 89 |
| C | 70 až 79 |
| D | 60 až 69 |
| E | 50 až 59 |
| F | méně než 50 |

#### 3.3.11.3 Uživatelské rozhraní pro export

Části funkcionality exportu hodnocení které neprobíhají automaticky jsou vyučujícímu dostupné na obrazovce přehledu klasifikace. Přidal jsem zde tlačítko „Export hodnocení“, které zobrazí dialog se stejným názvem. Dialog obsahuje dvě karty – „KOS“ a „FIT Klasifikace“, detailněji je popisuji v následujících sekcích. Tlačítko i dialog se zobrazí pouze v prostředí FIT ČVUT, jelikož na gymnáziu bude export do KOS i FIT Klasifikace deaktivován nastavením příslušných proměnných prostředí[50].

### 3.3.12 Export do KOS

**Fáze vývoje:** V průběhu semestru (březen 2023)
**Cílová verze:** 1.3.0
**Hlavní související MR:** !218

Export hodnocení do KOS implementuje třída `KosExporter`[51]. Konstruktor přijímá jeden povinný a dva nepovinné argumenty – předmět, pro který má být hodnocení exportováno, vyučující, jehož studenti mají být zahrnuti (ve výchozím stavu všichni studenti) a zda mají být zapsány i známky, nebo jen zápočty.

#### 3.3.12.1 Zpracování dat

Nejprve jsou vypočteny celkové bodové zisky relevantních studentů voláním výše popsané metody `get_classification`. Celkové bodové zisky studentů jsou přeškálovány tak, aby maximální možný počet bodů získaných standardním způsobem (bez bonusových bodů apod.) byl 100. Následně jsou studentům vypočteny známky dle tabulky 3.1, v případě že tak bylo určeno při inicializaci. Také je určeno, zda student získal zápočet. Implementované řešení uděluje zápočet studentům, kteří získali více než polovinu maximálního počtu bodů. V budoucnu by bylo vhodné udělat škálování bodů a hranici zápočtu konfigurovatelné správcem předmětu, s čímž třída `KosExporter` počítá, ale zbytek implementace funkcionality zatím nikoliv.

#### 3.3.12.2 Skript

K vytvoření skriptu používá `KosExporter` šablonu `kos.js`[52]. Šablona obsahuje celý skript, do kterého je pouze potřeba doplnit data o studentech ve formátu JSON. Šablona je zobrazena ve výpisu 3.21. Jedná se o modifikaci skriptu pro zápis známek z řešení Moodle2KOS zmíněného v návrhu. Vytvoření skriptu doplněním dat do šablony je ukázáno ve výpisu 3.22.

[50] `DATA_PROVIDER=[Kosapi, Bakalari]` a `CLASSIFICATION_EXPORT_TARGET=[GradesFIT]`
[51] Třída `KosExporter` je definována v souboru `apps/export/exporter/kos.py`
[52] `apps/export/templates/export/js/kos.js`

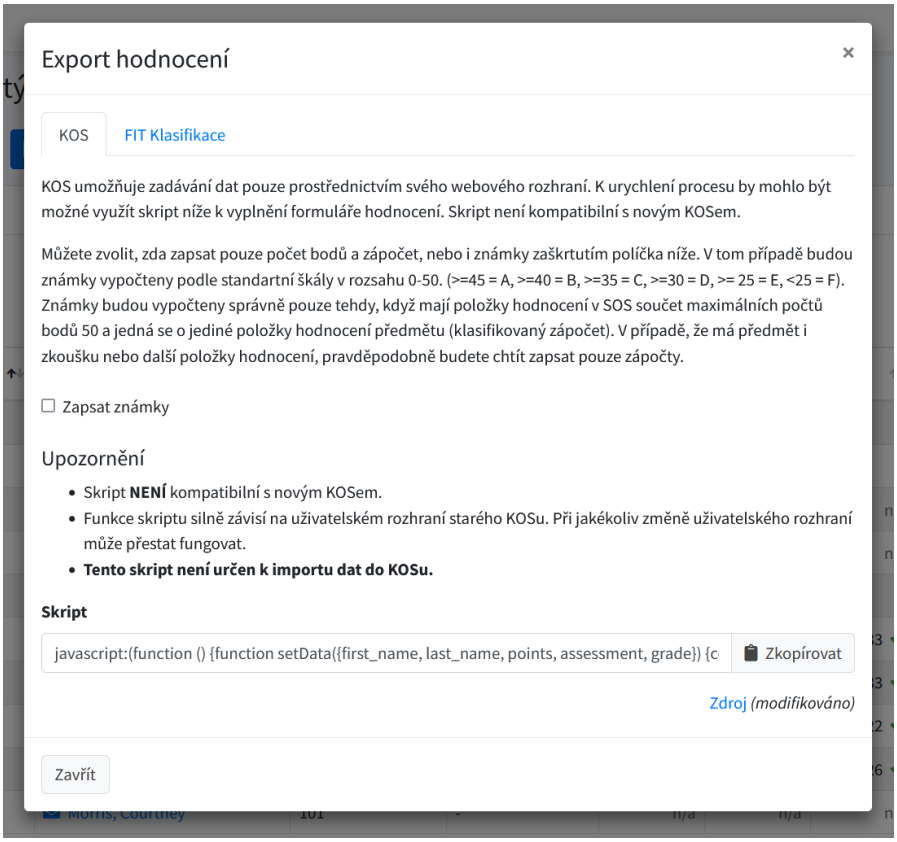
**Výpis kódu 3.21** Šablona pro vytvoření skriptu pro zápis do KOS

```
(function () {
    function setData({first_name, last_name, points, assessment, grade}) {
        const rows = $(`a:contains(${last_name})`).parent().parent();
        const row = rows.find(`td:contains(${first_name})`).parent();
        if (row.length === 0) {
            return;
        }
        if (row.length > 1) {
            alert(
                "Error importing grades: (last_name, first_name) \
                is not unique"
            );
            return;
        }
        row.find("input[type=checkbox]").prop('checked', assessment);
        row.find("[title='Poznámka']").find("input").val(points);
        if (grade !== null) {
            row.find("[title=<~popisek pole pro zadani znamky>]")
                .find("select").val(grade);
        }
    }

    function fillResults() {
        const gradeBook = JSON.parse('{{ data }}');
        $.map(gradeBook, (data) => setData(data));
    }

    fillResults();
})()
```

**Výpis kódu 3.22** Generování skriptu pro zápis do KOS

```
import json

from django.template.loader import render_to_string
from django.utils.safestring import SafeString


class KosExporter:
    ...

    def get_script(self) -> str:
        json_data = json.dumps(self._classification)

        script: str = render_to_string(
            self.script_template_path, {"data": SafeString(json_data)}
        )

        script = script.replace("\n", "").replace("    ", "")

        return "javascript:" + script
```

**Obrázek 3.26** Dialog obsluhující export do KOS




### 3.3.12.3 Proces exportu

Vyučující má funkcionalitu k dispozici v dialogu „Export hodnocení“ na obrazovce přehledu klasifikace. Na této obrazovce také vyučující nastaví, zda řeší klasifikaci pouze svých studentů, nebo celého předmětu (pokud je správcem předmětu; jinak vždy pouze svých studentů). Dialog obsahuje textový popis celé funkcionality exportu do KOS spolu s upozorněním na nedostatky implementovaného řešení. Dále se zde nachází zaškrtávací pole „Zapsat známky“ a textové pole, do kterého se při otevření dialogu vloží vygenerovaný skript. Uživatel může skript zkopírovat do schránky kliknutím na tlačítko „Zkopírovat“. Zkopírovaný skript pak může vložit do konzole prohlížeče na příslušné obrazovce webového rozhraní KOS, čímž do něj hodnocení zapíše. Dialog je zobrazen na obrázku 3.26.

Požadavek na vygenerování skriptu zpracovává `KosExportView`[53]. View přijímá HTTP GET požadavky a vrací odpověď typu `application/x-javascript`. V těle oďpovědi je skript vygenerovaný třídou `KosExporter` dle popisu výše.

Jedním z možných nedostatků popsané implementace je fakt, že je kompatibilní pouze se starým rozhraním KOS. Bohužel jsem neměl možnost analyzovat nové rozhraní KOS a tedy vytvořit řešení, které by s ním bylo kompatibilní. Staré rozhraní KOS je ale v době psaní tohoto textu stále dostupné, uživatel je na tuto skutečnost upozorněn a v případě potřeby je v budoucnu možné kompatibilitu s novým rozhraním KOS zajistit výměnou šablony, která slouží ke generování skriptu. Pokud je součástí nového rozhraní KOS obrazovka, na které vyučující vyplňuje tabulku hodnocení studentů podobně jako ve starém rozhraní, může většina implementace exportu zůstat stejná a bude potřeba pouze napsat nový skript, do kterého budou data

[53] `KosExportView` je definováno v souboru `apps/export/views.py`

z SOS vložena. Optimálním řešením by samozřejmě bylo celou funkcionalitu realizovat s využitím oficiálního API systému KOS, jak jsem již zmínil v návrhu, pokud by bylo v budoucnu zveřejněno.

### 3.3.13 Export do FIT Klasifikace

**Fáze vývoje:** V průběhu semestru (březen 2023)
**Cílová verze:** 1.3.0
**Hlavní související MR:** !222

Export do FIT Klasifikace po jeho patřičné konfiguraci probíhá automaticky, lze jej také v případě potřeby vyvolat manuálně. Implementace využívá oficiální API, které Klasifikace poskytuje [30].

#### 3.3.13.1 Problém s chybějícími studenty

V poslední sekci návrhu této funkcionality zmiňuji, že je třeba během implementace ověřit, zda se API dokáže vypořádat s požadavkem na zápis, který obsahuje hodnocení studenta, který podle Klasifikace daný předmět nestuduje. Ukázalo se, že API takový požadavek vůbec nepřijme a nezapíše ani ostatní validní hodnocení, které požadavek obsahoval, pouze vrátí HTTP odpověď „400 BAD REQUEST“.

Tento problém jsem vyřešil tak, že je před každým odesláním požadavku na hromadný zápis hodnocení nejprve z Klasifikace získán seznam studentů, které má Klasifikace přiřazeny k danému předmětu. API poskytuje endpoint[54], který přijímá GET požadavky a vrací seznam všech známých studentů předmětu spolu s jejich existujícími hodnoceními. Z tohoto seznamu jsou získána uživatelská jména studentů, jimž je možné zapisovat v Klasifikaci hodnocení. Tento seznam je následně použit k filtraci odesílaných dat tak, aby byl požadavek validní.

#### 3.3.13.2 Implementace exportu

Za realizaci exportu hodnocení odpovídá třída `GradesExporter`[55]. Konstruktor třídy přijímá jeden povinný a jeden nepovinný argument – předmět, pro který má být export proveden a vyučujícího, jehož studenti mají být zahrnuti (v případě chybějící hodnoty jsou zahrnuti všichni studenti). Třída poskytuje čtyři důležité metody:

- `export_course()` – serializuje kompletní hodnocení všech studentů předmětu,
- `export_team(team: Team)` – serializuje kompletní hodnocení členů daného týmu,
- `export_tester(tester: User)` – serializuje hodnocení účasti na testování daného testera,
- `publish() -> tuple[int, str]` – odešle HTTP PUT požadavek s připravenými daty do Klasifikace, návratovou hodnotou je HTTP status kód odpovědi API a zpráva o úspěchu nebo chybová hláška.

Toto rozhraní je ve skutečnosti definováno třídou `BaseExporter`[56], jejímž je `GradesExporter` potomkem. Účelem této abstraktní[57] třídy je definovat jednotné rozhraní pro provádění exportu typu Klasifikace. Domnívám se totiž, že stejným způsobem by bylo možné realizovat export do systému Bakaláři při využití SOS na gymnáziu. To může být předmětem budoucího rozvoje

[54] `/public/courses/{course-code}/group/{group-code}/student-classifications`; pro získání dat všech studentů předmětu slouží group-code „ALL“

[55] Třída `GradesExporter` je definována v souboru `apps/export/exporter/grades.py`

[56] Třída `BaseExporter` je definována v souboru `apps/export/exporter/base.py`

[57] `BaseExporter` není abstraktní třídou v pravém slova smyslu. Lze ji inicializovat a na jejích instancích je možné volat některé metody. Jedná se spíše o něco mezi abstraktní třídou a null-objektem.

**Výpis kódu 3.23** Příklad dat připravených k odeslání do FIT Klasifikace

```
[
    {
        "classificationIdentifier": "semestral_work",
        "studentUsername": "student1",
        "value": 46,
    },
    {
        "classificationIdentifier": "testing",
        "studentUsername": "student1",
        "value": 4,
    },
    {
        "classificationIdentifier": "semestral_work",
        "studentUsername": "student2",
        "value": 31,
    },
    {
        "classificationIdentifier": "testing",
        "studentUsername": "student2",
        "value": 0,
    },
]
```

SOS, samozřejmě za podmínky kompatibility API tohoto systému. Součástí abstrakce exportu je také funkce `get_exporter_class`, která je definována ve stejném souboru jako `BaseExporter` a její návratovou hodnotou je správná třída pro realizaci exportu dle nastavení proměnných prostředí[58]. Třída `KosExporter` není potomkem `BaseExporter`, jelikož princip exportu do KOS je odlišný, třídy mají odlišné rozhraní a odlišným způsobem se používají.

Metoda `export_course` k získání hodnocení studentů volá již několikrát zmiňovanou metodu `get_classification` modelu `Course`. Ostatní metody používají `member_earned_points` a `member_testing_points` modelu `Team`. Serializace těchto dat do formátu předepsaného dokumentací [30] probíhá s pomocí privátní metody `_create_dto`. Výsledkem serializace je seznam jednotlivých hodnocení v tomto formátu pro všechny relevantní studenty. Hodnocení jsou serializována pouze pro ty studenty, kterým je možné hodnocení do Klasifikace zapsat, jak jsem již popsal v předchozí sekci. Příklad serializovaných dat je uveden ve výpisu 3.23. Export počítá s tím, že v Klasifikaci je definován sloupec „semestral_work“, případně i „testing“ v případě, že je aktivováno studentské testování.

#### 3.3.13.3 Autorizace

Implementované řešení dle návrhu používá service-account pro autorizaci požadavků na API Klasifikace. Ke každému požadavku je třeba přiložit tzv. *access token*, který lze získat od fakultního OAAS. K tomu slouží funkce `get_client_token`[59], kterou jsem implementoval v aplikaci `oauth`. Implementace je odlišná od původní implementace komunikace s OAAS, jelikož ta namísto service-account pracovala s uživatelskými tokeny. Aby nemusel být token vytvářen znovu pro každý požadavek, je do doby jeho expirace uložen pomocí cachovacího frameworku, který je součástí frameworku Django. Cache je používáno ve výchozí konfiguraci dle dokumentace [50]. Implementace funkce pro získání tokenu je zobrazena na výpisu 3.24.

[58] `CLASSIFICATION_EXPORT_TARGET=[GradesFIT]`; potenciálně např. i „Bakalari“

[59] Funkce `get_client_token` je definována v souboru `apps/oauth/utils/token.py`

**Výpis kódu 3.24** Funkce pro získání client tokenu pro komunikaci s Klasifikací

```
import httpx

from django.conf import settings
from django.core.cache import cache

from .. import conf as oauth_conf


def get_client_token():
    if (access_token := cache.get(oauth_conf.CLIENT_CACHE_KEY)) is None:
        token_response = httpx.post(
            oauth_conf.TOKEN_ENDPOINT,
            auth=(
                settings.SERVICE_CLIENT_ID,
                settings.SERVICE_CLIENT_SECRET
            ),
            data={
                "grant_type": "client_credentials",
                "scope": "cvut:grades:course-restricted",
            },
        )
        token_info = _parse_token_response(
            token_response, require_refresh=False
        )
        access_token = token_info["access_token"]

        cache.set(
            oauth_conf.CLIENT_CACHE_KEY,
            access_token,
            int(token_info["expires_in"]),
        )

    return access_token
```

#### 3.3.13.4 Konfigurace

Konfigurace funkcionality se skládá ze tří částí. První z nich je konfigurace proměnných prostředí pro systém SOS. Je potřeba nastavit následující proměnné prostředí:

- `CLASSIFICATION_EXPORT_TARGET` – pro aktivaci exportu do Klasifikace je třeba nastavit hodnotu „GradesFIT“,
- `SERVICE_CLIENT_ID` – Client ID pro service account,
- `SERVICE_CLIENT_SECRET` – Client secret pro service account.

Dalším krokem je konfigurace exportu pro vybraný předmět v rámci SOS. Export do Klasifikace je aktivován nastavením boolean atributu `is_grades_export_enabled`, který jsem pro tento účel přidal do modelu `Course`. Správce předmětu jej nastaví na obrazovce přehledu konfigurace pomocí stejného formuláře, jako například granularitu nastavení.

K rozpoznání, zda je export do Klasifikace aktivní slouží metody `is_enabled_globally` a `is_enabled` ve třídě `BaseExporter` a jejích potomcích. U instancí rodičovské třídy tyto modely vždy vrací hodnotu `False`, což značí, že export není aktivní. U instancí `GradesExporter` vrací první metoda pravdivou hodnotu v případě, že jsou patřičně nastaveny proměnné prostředí a druhá v případě, že je navíc export aktivován správcem pro daný předmět.

Třetí krok konfigurace se týká samotné Klasifikace. Jako první je potřeba zaregistrovat SOS do Klasifikace jako „externí aplikaci“, a to zadáním názvu a použitého Client ID. To může provést vyučující, který je alespoň editorem nějakého předmětu v Klasifikaci. Aplikaci je potřeba zaregistrovat pouze jednou, následně ji může využívat více předmětů. Dále je třeba SOS přidat jako „externí aplikaci předmětu“ k definici vybraného předmětu v Klasifikaci. To je důsledek využití scope `course-restricted` při autorizaci – SOS musí být zaregistrován u daného předmětu v Klasifikaci, aby bylo možné komunikovat přes API v rámci tohoto předmětu. Registraci externí aplikace SOS k předmětu v Klasifikaci může provést vyučující, který je alepoň editorem tohoto předmětu. Jak tento proces provést je poměrně snadno pochopitelné z uživatelského rozhraní Klasifikace pro vyučujícího, který má patřičná oprávnění.

Nakonec zbývá v Klasifikaci správně definovat sloupce hodnocení, aby mohl export korektně proběhnout. Je potřeba definovat sloupec „semestral_work“, případně i „testing“ v případě, že je aktivováno studentské testování. K urychlení tohoto procesu jsem připravil dvě varianty definice hodnocení předmětu, které obsahují potřebné sloupce. Klasifikace totiž umí ukládat a načítat definice ve formátu CSV. Soubory s definicemi[60] si může správce předmětu stáhnout pomocí odkazu pod formulářem v SOS, kde export do Klasifikace aktivoval.

#### 3.3.13.5 Automatický export

Export probíhá převážně automaticky, ve dvou situacích – při ohodnocení odevzdaného řešení vyučujícím a při potvrzení účasti na testování vedoucím týmu, jak jsem popsal již v návrhu.

V prvním případě je v `SubmissionEvaluateView`[61] při uložení hodnocení vytvořena instance `GradesExporter`, na které jsou následně zavolány metody `export_team` a `publish`. Hodnocení je v SOS vždy uloženo bez ohledu na úspěch exportu. Informace o úspěchu exportu je vyučujícímu zobrazena na další obrazovce.

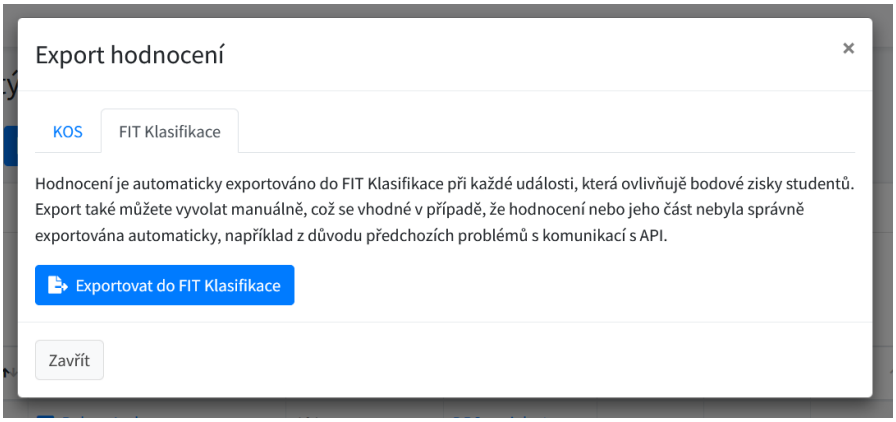
V druhém případě je v `TestConfirmationView`[62] při uložení potvrzení účasti taktéž vytvořena instance `GradesExporter`, na které jsou zavolány metody `export_tester` a `publish`. Potvrzení je taktéž vždy uloženo v SOS bez ohledu na úspěch exportu. V tomto případě se uživateli informace o úspěchu exportu vůbec nezobrazuje, pro studenta to není podstatné.

[60] Definice jsou uloženy v souborech `staticfiles/grades_templates/sos_grades.csv` a `staticfiles/grades_templates/sos_grades_testing.csv`

[61] `SubmissionEvaluateView` je definováno v souboru `apps/submissions/views/base.py`

[62] `TestConfirmationView` je definováno v souboru `apps/teams/views/testing.py`

**Obrázek 3.27** Dialog obsluhující manuální export do FIT Klasifikace




#### 3.3.13.6 Manuální export

V případě potřeby může dle návrhu vyučující vyvolat export celého předmětu do Klasifikace manuálně prostřednictvím dialogu na obrazovce přehledu klasifikace. Dialog obsahuje pouze textový popis funkcionality, tlačítko „Exportovat do FIT Klasifikace“ a po dokončení exportu také zprávu o úspěchu nebo chybě. V případě, že není export do Klasifikace nakonfigurován, obsahuje místo toho dialog pouze informaci kdo a kde může konfiguraci provést. Dialog je zobrazen na obrázku 3.27.

Export celého předmětu do Klasifikace realizuje `GradesCourseExportView`[63], které přijímá HTTP POST požadavky a vrací odpověď typu `application/json`. V těle odpovědi je status kód odpovědi od API a zpráva pro zobrazení uživateli. Princip zpracování je v podstatě totožný jako například u aktualizace seznamu studentů popsané v sekci 3.3.10.

## 3.4 Aktualizace závislostí

**Fáze vývoje:** V průběhu semestru (duben 2023)
**Hlavní související MR:** !251

V analýze v sekcích 1.3.1 and 1.3.2 jsem uvedl, že by bylo vhodné, aby SOS používal novější verze jazyka Python a frameworku Django. Původní implementace používala Python 3.9.6 a Django 3.2 LTS. Rozhodl jsem se tedy přejít na nejnovější dostupnou verzi jazyka Python, což je v době psaní tohoto textu Python 3.11.3 a nejnovější LTS verzi frameworku Django, 4.2 LTS. Kromě nových funkcionalit a zrychlení aplikace [13] je hlavním argumentem pro přechod na novější verze především podporovatelnost – jak jsem již uvedl v analýze, bezpečnostní aktualizace bude Django 4.2 LTS dostávat do roku 2026 a Python 3.11 do roku 2027.

Samotná aktualizace a nasazení nové verze SOS byly díky využití technologie Docker snadné – stačilo pouze změnit čísla verzí v konfiguračních souborech pro Docker[64] a v souborech pro specifikaci Python zavislostí[65].

Při přechodu na novější verze je také vždy potřeba prostudovat všechny změny a zhodnotit, jestli je potřeba s aktualizací provést nějaké úpravy. K tomu slouží tzv. *release notes*. Prostudoval jsem tedy tyto dokumenty pro všechny vydané verze až po nejnovější, konkrétně Python 3.10 [51], Python 3.11 [52], Django 4.0 [53], Django 4.1 [54] a Django 4.2 LTS [55] a porovnal je s

[63] `GradesCourseExportView` je definováno v souboru `apps/export/views.py`
[64] Soubory `Dockerfile`, `Dockerfile.stage` a `Dockerfile.prod` v kořenovém adresáři projektu
[65] Soubor `requirements.txt` v kořenovém adresáři projektu a další textové soubory ve složce `requirements/`

implementací SOS. Ukázalo se, že Python lze přímo aktualizovat bez dalších zásahů, naopak aktualizace frameworku Django vyžadovala určité úpravy, které jsem musel provést. Jednalo se o poměrně jednoduché změny v konfiguračních souborech, výměnu některých zastaralých[66] tříd poskytovaných frameworkem za jejich novější alternativy a úpravy kódu reflektující změny rozhraní tříd poskytovaných frameworkem. Kromě toho jsem také provedl celkovou revizi souborů pro specifikaci Python závislostí a odstranil jsem z nich některé zbytečné záznamy – byla zde definována řada závislostí, které ve skutečnosti byly jen závislostmi jiných použitých modulů a nebylo třeba je uvádět explicitně. Jejich přítomnost zároveň komplikovala aktualizaci. Ostatní závislosti jsem opět aktualizoval na nejnovější dostupné verze kompatibilní s nově použitými verzemi jazyka Python a frameworku Django.

Nakonec bylo potřeba celý systém po aktualizaci řádně otestovat a vyřešit případné problémy způsobené aktualizací. To obnášelo spuštění unit testů, manuální otestování průchodem aplikací se zaměřením na potenciálně problematická místa (především části aplikace využívající moduly třetích stran) a několikadenní provoz aktualizované verze v testovacím prostředí[67]. Při spouštění unit testů a manuálním testování se objevilo několik problémů menšího rozsahu souvisejících se změněným rozhraním některých modulů třetích stran nebo strukturou konfiguračních souborů, což bylo vzhledem k četným a významným změnám verzí očekávané. Během provozu v testovacím prostředí se pak žádné další problémy neobjevily.

## 3.5 Finální stav architektury

Z hlediska architektury a použitých technologií nedošlo během vývoje k žádné výraznější změně, kromě použití novějších verzí jazyka Python, frameworku Django a dalších závislostí, jak uvádím v předchozí sekci 3.4. Nově přibyla pouze integrace s aplikací FIT Klasifikace skrze její API popsaná v sekci 3.3.13 a částečně automatizovaný export hodnocení do KOS realizovaný generovaným skriptem, popsaný v sekci 3.3.12. Tuto novou situaci ilustruje diagram na obrázku 3.28, který oproti diagramu původní architektury v sekci 1.3 znázorňuje i komunikaci SOS a jeho uživatele s KOS a FIT Klasifikací.

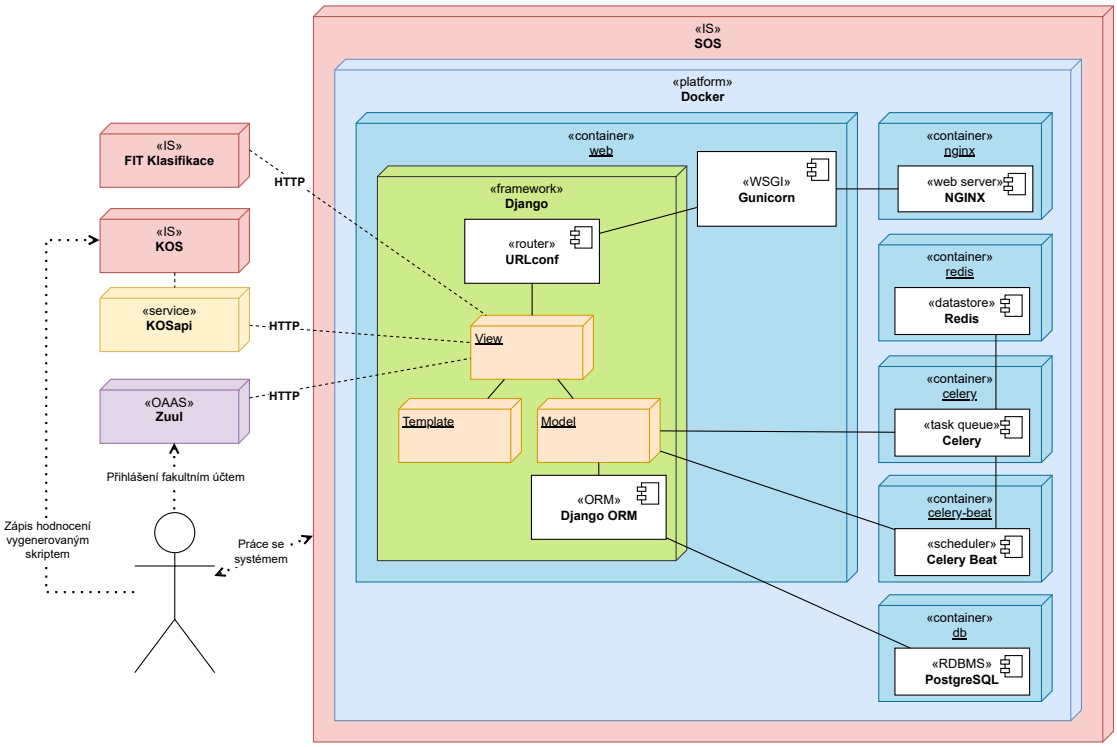
Součástí implementace byly změny datového modelu, které jsem implementoval podle návrhu a jsou popsané v sekci 3.3. Se změnami Modelů souvisí i změny databázového schématu, které proběhly automaticky pomocí databázových migrací generovaných frameworkem Django, které taktéž popisuji u konkrétních realizovaných úprav v sekci 3.3. Konceptuální model a vygenerovaný ER diagram jsou dostupné v Redmine Wiki projektu[68] jako součást výstupů práce SP týmu, které je věnována kapitola Zapojení dalších studentů. Konceptuální model je také k dispozici v příloze B.

## 3.6 Zdrojové kódy

Veškeré zdrojové kódy SOS jsou uloženy ve webovém repozitáři GitLab, který se nachází na adrese `https://gitlab.fit.cvut.cz/sos/sos`. Původně byl repozitář vytvořen v mém osobním namespace[69], nicméně vzhledem k tomu, že po dokončení magisterského studia již nebudu studentem FIT ČVUT, bylo potřeba repozitář přesunout. Vedoucí práce Hunka proto vytvořil GitLab skupinu *SOS*[70], do které jsem repozitář přesunul. Přesun repozitáře do jiného namespace je popsán v oficiální dokumentaci [56]. V nově vzniklé skupině mi byla udělena role „Maintainer“, díky čemuž jsem mohl přesun realizovat a také mi zůstala zachována veškerá oprávnění potřebná

[66] V anglicky psané dokumentaci uváděno jako „deprecated“
[67] `https://sos2.fit.cvut.cz`
[68] `https://dbs.fit.cvut.cz/redmine/projects/sos-fit-cvut-cz/wiki/Datab%C3%A1zov%C3%BD_model`
[69] `https://gitlab.fit.cvut.cz/pavlutom/sos`
[70] `https://gitlab.fit.cvut.cz/sos`

**Obrázek 3.28** Finální architektura SOS




ke správě repozitáře. Kromě mě jsou aktuálně členy této skupiny také Hunka, Hejda a studenti, kteří se na vývoji SOS podílejí v rámci předmětu BI-SP1.

### 3.6.1 Branching model

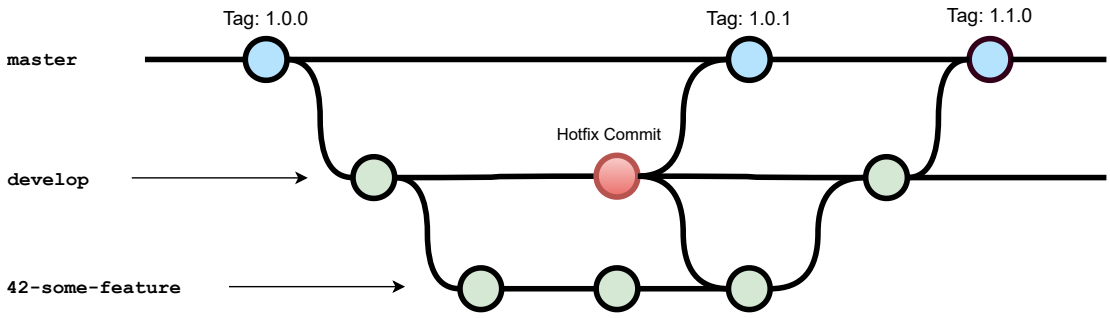
V repozitáři existují dvě hlavní větve – `develop` a `master`. Větev `master` slouží pouze pro uchovávání produkční verze projektu, jediná aktivita zde je merge větve `develop`. Větev `develop` slouží k integraci ostatních větví a k realizaci drobných oprav, pro které nemá smysl vytvářet vlastní větve. Obě zmíněné větve jsou chráněné, pro přidání commitů musí mít uživatel v repozitáři alespoň roli „Maintainer“. U obou větví je také zakázáno přepisovat jejich historii.

Pro vývoj samotný slouží tzv. *feature branch*. Ty jsou většinou spjaty s konkrétní Issue v GitLab Issue Trackeru[71], což značí prefix s číslem Issue v názvu větve. Větve tohoto typu jsou vždy tvořeny z `develop`, po dokončení implementace je proveden merge opět do `develop`. Pokud se v průběhu implementace změnil kód v `develop` takovým způsobem, že konflikty brání sloučení větví, musí být *feature branch* nejprve aktualizována změnami provedenými v `develop` a konflikty manuálně vyřešeny. K označení produkčních verzí SOS jsou využity tzv. tagy. *Git tag* je reference na určitý commit v repozitáři. Jako název tagu jsem se rozhodl používat čistě samotné číslo verze. Označení verzí je věnována sekce 3.7.

Popsaný branching model i s označováním verzí znázorňuje obrázek 3.29. Model je zjednodušenou variantou *Gitflow* [57] a rozhodl jsem se jej používat především proto, že se mi dlouhodobě osvědčil v mém zaměstnání, kde mimo jiné odpovídám za správu verzí, integraci funkcionalit vyvíjených ostatními členy týmu a distribuci softwarového produktu. Produktem je sice na rozdíl od této práce desktopová aplikace, nicméně nastavený proces včetně CI/CD je velmi podobný, až na finální krok distribuce ke koncovým uživatelům, který už s branching modelem příliš nesouvisí.

[71] `https://gitlab.fit.cvut.cz/sos/sos/-/issues`

**Obrázek 3.29** Znázornění nastaveného branching modelu v repozitáři SOS




## 3.7 Označení verzí

Produkční verze SOS byly označovány již od prvního nasazení na FIT ČVUT v roce 2021. Označování však probíhalo poměrně náhodným způsobem, podle aktuálního uvážení vývojáře který prováděl změny, jímž jsem byl vždy buď já nebo Hejda. Poslední taková verze byla 0.6.2 nasazená v listopadu 2022.

### 3.7.1 Specifikace

Rozhodl jsem se zavést systém označování verzí, který se volně drží specifikace *Semantického verzování*. Ta definuje označení verzí ve formátu „MAJOR.MINOR.PATCH“, kde navyšování jednotlivých čísel probíhá následovně [58]:

- MAJOR – když nastala změna, která není zpětně kompatibilní s ostatními,
- MINOR – když byla přidána funkcionalita se zachováním zpětné kompatibility,
- PATCH – když byla opravena chyba a zůstala zachována zpětná kompatibilita.

Specifikace je určena především pro softwarové produkty, které poskytují veřejné API a smyslem označení verzí je zejména zajistit kompatibilitu s dalším software, který toto API používá, což není případ SOS. SOS se dále od specifikace odchyluje tak, že:

- MAJOR verze byla zatím inkrementována pouze jednou, a to z 0 na 1 při zavedení tohoto způsobu označování verzí. Není jasné, kdy by měla být opět inkrementována, jelikož se přímo nepočítá se zaváděním zpětně nekompatibilních změn, ani vzhledem k čemu by tato kompatibilita měla být vlastně posuzována.
- PATCH verze je kromě oprav chyb někdy inkrementována také v případě, kdy je upravena/rozšířena nějaká existující funkcionalita v rozsahu, který není dle subjektivního uvážení vývojářů dostatečný k inkrementaci MINOR verze.
- V případě skutečně drobných nebo z uživatelského hlediska nepodstatných oprav chyb je někdy místo inkrementace PATCH verze k označení přidáno ještě čtvrté číslo označující takovou aktualizaci. Tato situace nastávala hlavně v počátcích provozu verze SOS vznikající v rámci této práce, kdy se objevovalo větší množství drobných chyb, které ale bylo potřeba akutně řešit.

### 3.7.2 Přehled nasazených verzí

První verzí označenou podle tohoto systému byla verze 1.0.0 implementující požadavky kritické pro využití SOS v předmětu BI-SP1, nasazená v lednu 2023. V době psaní tohoto textu[72] je aktuální verze 1.4.5. Přehled všech verzí nasazených do produkčního prostředí poskytuje *Changelog*, který je dostupný v českém i anglickém jazyce jak v GitLab repozitáři projektu, tak i v uživatelském rozhraní SOS[73].

### 3.7.3 Implementace

Changelog je realizován pomocí souboru ve formátu *Markdown*[74]. Tento soubor může být zobrazen buď webovým rozhraním systému GitLab, nebo uživatelským rozhraním SOS, kde jeho zpracování a transformaci do HTML dokumentu zajišťuje `ChangelogView`[75].

Samotné označení verze je definováno souborem `.version` v kořenovém adresáři projektu. Z něj je označení verze načteno do proměnné prostředí a později zobrazeno v patičce všech obrazovek SOS jako odkaz vedoucí právě na Changelog.

## 3.8 Nasazení do provozu

SOS byl od prvního nasazení v roce 2021 provozován na virtuálním stroji na platformě CloudFIT, který mi přidělilo pro účely mé bakalářské práce oddělení ICT FIT ČVUT. Stroj existoval ve starém rozhraní OpenNebula, které je nově nahrazováno rozhraním VMware. Staré rozhraní již aktuálně není podporováno a má být postupně zcela nahrazeno novým rozhraním VMware [59]. Pro budoucí spolehlivý provoz SOS v prostředí FIT ČVUT bylo tedy nutné přejít na nové rozhraní. Zároveň zde byl stejný problém jako u GitLab repozitáře – bylo potřeba správu tohoto virtuálního stroje převést na zadavatele a vedoucího práce Hunku.

### 3.8.1 Nové virtuální stroje

Nakonec jsme se s vedoucím práce dohodli na vytvoření dvou nových virtuálních strojů v novém rozhraní VMware, pro produkční a pro testovací prostředí, které budeme do ukončení mého magisterského studia spravovat společně, následně zůstane jejich správa na něm. Naší žádosti o jejich vytvoření ICT FIT ČVUT vyhovělo a přidělilo nám dva takřka identické virtuální stroje s následujícími parametry:

- Počet CPU: 2
- Velikost operační paměti: 4 GB
- Kapacita pevného disku: 50 GB (v testovacím prostředí 30 GB)
- Operační systém: Debian GNU/Linux 11 (64-bit)

Nové virtuální stroje se stejně jako ten původní nacházejí za fakultní proxy, není tedy třeba v rámci konfigurace webového serveru pro SOS řešit SSL. Jsou k nim také přiřazeny DNS záznamy a stroje jsou tak dostupné na adresách `https://sos.fit.cvut.cz` (produkční) a `https://sos2.fit.cvut.cz` (testovací). Produkční stroj může také využívat SMTP server na `relay.fit.cvut.cz` k odesílání e-mailů. K těmto účelům byl zřízen mailový alias *sos@fit.cvut.cz*.

[72] ke dni 16. dubna 2023
[73] `https://sos.fit.cvut.cz/changelog/`
[74] `https://www.markdownguide.org/basic-syntax/`
[75] `ChangelogView` je definováno v souboru `apps/homepage/views.py`

Instalace SOS na tyto virtuální stroje byla opět s využitím Dockeru snadná – pouze jsem v systému vytvořil uživatele pro správu SOS, nainstaloval podle oficiálního návodu Docker [60] a Docker Compose [61] a nakonfiguroval SSH klíče pro komunikaci s GitLab repozitářem. Zbytek procesu instalace proběhl automaticky v rámci CI/CD, které popíšu v sekci 3.8.2. Nakonec jsem ještě v systému zaregistroval a aktivoval *systemd službu* `sos-app`[76], která zajišťuje automatické spuštění SOS při startu systému, pro případ že se server restartuje.

Poprvé bylo také potřeba na produkční stroj přenést data z původního stroje, na kterém byl SOS dosud provozován – obsah PostgreSQL databáze a soubory nahrané uživateli. Vzhledem k tomu, že se všechny stroje nacházely uvnitř stejné sítě v rámci FIT, mohl jsem celý proces migrace realizovat jednoduchým skriptem, který jsem původně vytvořil pro kopírování produkčních dat do testovacího prostředí. Skript[77] popisuji spolu s dalšími v sekci 3.8.3. Po migraci dat zbýval poslední a jediný manuální krok, a to úprava konfigurace fakultní proxy, aby požadavky na `https://sos.fit.cvut.cz` předávala novému virtuálnímu stroji. Tento krok musel provést administrátor CloudFIT. Díky automatizaci celého procesu migrace (až na poslední krok) se podařilo jej realizovat bez ztráty dat a s výpadkem služby v řádu nižších desítek sekund.

### 3.8.2 Continuous Integration / Continuous Deployment

SOS využíval jednoduché CI/CD realizované pomocí nástrojů dostupných ve fakultou provozovaném GitLabu ještě před jeho prvním využitím na FIT ČVUT, nastavil jsem jej již v rámci své bakalářské práce [1, kap. 3.9]. Původní pipeline měla tři fáze:

- `build` – Vytvoření virtuálního prostředí a instalace potřebných Python balíčků.
- `test` – Spuštění testů. Pokud testy selžou, pipeline dál nepokračuje.
- `deploy` – Nasazení nové verze spuštěním skriptu přes SSH.

Hejda v rámci své bakalářské práce proces nasazování upravil s využitím technologie Docker. Do pipeline přidal po spuštění testů krok, který sestavil Docker image pomocí nástroje Buildah[78] a nahrál jej do tzv. *Container registru* v GitLab repozitáři projektu. Pro účely nasazování SOS v prostředí GJGJ také vytvořil skript, který z repozitáře tento image stáhne a nainstaluje, čímž byl nahrazen poslední krok pipeline, který zajišťuje nasazení nové verze. Když jsem na podzim roku 2022 nasazoval tehdejší verzi SOS do prostředí FIT ČVUT, použil jsem stejný mechanismus jako Hejda a nové verze jsem pak nasazoval ručně. Funkční tedy zůstalo CI, ale nikoliv CD.

V letním semestru 2022/23 vznikla vzhledem k této práci potřeba často a efektivně nasazovat nové verze do prostředí FIT ČVUT. Pipeline jsem tedy upravil tak, aby bylo CI/CD opět plně funkční. Celý proces jsem intenzivně konzultoval s Hejdou, jelikož měl jednak více zkušeností s technologií Docker než já a byl autorem změn souvisejících se sestavením image a také proto, že bylo potřeba proces nastavit tak, aby byl alespoň částečně kompatibilní s prostředím GJGJ, kde za provoz SOS odpovídá právě Hejda.

Jednou z nových vlastností pipeline je, že automatické nasazení probíhá do produkčního i testovacího prostředí. Do produkčního prostředí jsou automaticky nasazovány verze z větve `master`, do testovacího verze z větve `develop`. Finální verze pipeline je definována v souboru `.gitlab-ci.yml` v kořenovém adresáři projektu a skládá se z osmi *jobů* rozdělených do čtyř *stage*: `build`, `test`, `image-build` a `deploy`. Jednotlivé *joby* jsou spouštěny podmíněně podle větve, které se pipeline týká. Rozdělení *jobů* do *stage* je patrné například z vizualizace z webového rozhraní GitLab na obrázku 3.30. Jedná se o tyto fáze:

- **Job:** `install-dependencies`
  **Stage:** `build`

[76] Služba je definována v souboru `sos-app.service`
[77] Jedná se o skript `sync_prod_db.sh`
[78] `https://buildah.io/`

**Větev:** *všechny*
**Popis:** Instalace Python závislostí podle specifikace pro CI určené souborem `ci.txt` v adresáři `requirements/`.

- **Job:** `static-tests-black`
  **Stage:** `test`
  **Větev:** *všechny*
  **Popis:** Spuštění statických testů v podobě kontroly formátování kódu nástrojem Black[79]. Pipeline se zde ukončí, pokud není formátování v pořádku. Artefaktem je report o úspěchu statických testů ve formátu JUnit XML[80], který později slouží ke zobrazení ve webovém rozhraní GitLabu.

- **Job:** `unit-tests`
  **Stage:** `test`
  **Větev:** *všechny*
  **Popis:** Spuštění unit testů nástrojem Pytest[81]. Pokud některý z testů selže, pipeline se zde ukončí. Tento job má tři artefakty: HTML report o pokrytí kódu unit testy (slouží ke stažení a zobrazení v internetovém prohlížeči), JUnit XML report o pokrytí kódu unit testy (slouží pro zobrazení pokrytí testy ve webovém rozhraní GitLabu) a JUnit XML report o úspěchu unit testů (opět pro zobrazení ve webovém rozhraní GitLabu).

- **Job:** `create-stage-image`
  **Stage:** `image-build`
  **Větev:** `develop`
  **Popis:** Sestavení Docker image nástrojem Buildah pro použití v testovacím prostředí a jeho nahrání do *Container registru*.

- **Job:** `create-prod-image`
  **Stage:** `image-build`
  **Větev:** `master`
  **Popis:** Sestavení Docker image nástrojem Buildah pro použití v produkčním prostředí a jeho nahrání do *Container registru*.

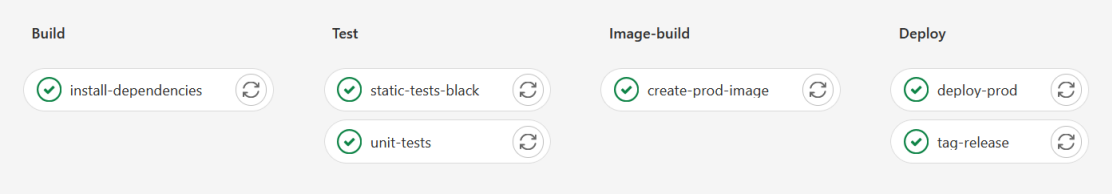
- **Job:** `deploy-stage`
  **Stage:** `deploy`
  **Větev:** `develop`
  **Popis:** Vyvolání aktualizace z *Container registru* v testovacím prostředí přes SSH.

- **Job:** `deploy-prod`
  **Stage:** `deploy`
  **Větev:** `master`
  **Popis:** Vyvolání aktualizace z *Container registru* v produkčním prostředí přes SSH.

- **Job:** `tag-release`
  **Stage:** `deploy`
  **Větev:** `master`
  **Popis:** Automatické vytvoření *Git tagu* při nasazení nové verze do produkčního prostředí. Tag se vygeneruje ze souboru `.version` v kořenovém adresáři projektu a vytvoří se pouze v případě, že tag s danou verzí v repozitáři zatím neexistuje.

[79] `https://github.com/psf/black`
[80] `https://www.ibm.com/docs/en/developer-for-zos/16.0?topic=formats-junit-xml-format`
[81] `https://docs.pytest.org/en/7.3.x/`

**Obrázek 3.30** Znázornění pipeline na větvi `master` ve webovém rozhraní GitLab




### 3.8.3 Administrační skripty

V kořenovém adresáři projektu se nachází složka `scripts/`, která obsahuje skripty pro využití např. v CI/CD pipeline nebo při lokálním vývoji. Jedná se o tyto soubory:

- `.config` a `.config.example` – soubor (a šablona k jeho vytvoření) obsahující konfigurační proměnné pro ostatní skripty (přihlašovací údaje, adresy atp.)
- `clear_db.sh` – shell skript, který zahodí existující PostgreSQL databázi, vytvoří novou a spustí databázové migrace pro vytvoření schématu. Slouží pro lokální vývoj.
- `deploy.sh` – shell skript, který stáhne nejnovější Docker image z *Container registru*, restartuje službu `sos-app` a spustí databázové migrace. Je spouštěn vzdáleně z CI/CD pipeline v produkčním i testovacím prostředí, slouží k nasazení nové verze SOS.
- `run_static_tests.sh` – shell skript, který spustí statické testy a vygeneruje XML výstup pro GitLab. Primárně slouží ke spuštění v CI/CD pipeline, ale lze jej použít i lokálně.
- `run_unit_tests.sh` – shell skript, který spustí unit testy a vygeneruje XML výstupy pro GitLab a HTML výstup pro přímé zobrazení. Primárně slouží ke spuštění v CI/CD pipeline, ale lze jej použít i lokálně.
- `sync_prod_db.sh` – shell skript, který slouží ke stažení aplikačních dat (PostgreSQL databáze a uživatelské soubory) z produkčního do testovacího prostředí. Primárně je určen k manuálnímu spuštění z testovacího serveru, ale lze jej se správnou konfigurací v souboru `.config`, znalostí příslušných přístupových údajů a připojením přes fakultní VPN použít i ke stažení produkčních dat do lokálního systému vývojáře.
- `tag_release.sh` – shell skript, který slouží k automatickému vytvoření *Git tagu*. Je určen ke spuštění v CI/CD pipeline s větví `master`.

## 3.9 Dokumentace

Již od úplného začátku vývoje projektu jsem se snažil především o to, aby k porozumění zdrojovým kódům nebylo třeba rozsáhlé dokumentace v podobě komentářů v kódu. Při vývoji a také v rámci *code review* změn prováděných dalšími vývojáři, které popisuji v kapitole Zapojení dalších studentů, jsem dbal na smysluplné pojmenovávání všech tříd, metod a proměnných, aby bylo z jejich názvů co nejlépe pochopitelné, k čemu slouží. I tak jsou ale důležité složitější třídy a metody – zejména Views, Modely a Formuláře – opatřeny tzv. *docstrings*, tedy dokumentačními komentáři shrnujícími poskytovanou funkcionalitu. S využitím balíčku `django.contrib.admindocs` je pak z těchto komentářů a ze samotné implementace generována základní programátorská dokumentace, která je dostupná přímo v administrátorském rozhraní SOS[82].

[82] `https://sos.fit.cvut.cz/admin/doc/`; pro přístup je potřeba administrátorské oprávnění; dokumentace je přístupná i v testovacím a lokálním prostředí, kde lze toto oprávnění snadno získat

Kromě této dokumentace kódu je pro všechny vývojáře projektu dostupná i dokumentace projektu v podobě provedené analýzy v Redmine Wiki, kterou zpracovali studenti BI-SP1 během semestru. Studenti BI-SP1 také zpracovali uživatelskou dokumentaci, která je dostupná na `https://sos.pages.fit/dokumentace/` a popisuje jednotlivé aktivity, které mohou uživatelé v SOS realizovat, ilustrované snímky obrazovky. Těmto položkám se více věnuji v kapitole Zapojení dalších studentů.

## 3.10 Shrnutí

V úvodu kapitoly jsem popsal zvolenou metodiku vývoje – inkrementální přístup. Část požadavků jsem implementoval před začátkem letního semestru 2022/23 a po akceptačním testování jsem nasadil novou verzi SOS 1.0.0 do prostředí FIT ČVUT, kde byla následně využita při výuce předmětu BI-SP1. Během semestru jsem průběžně implementoval zbylé požadavky a věnoval se zdokonalování SOS. Také jsem implementoval export hodnocení z SOS do KOS a FIT Klasifikace. V době psaní tohoto textu[83] je poslední nasazená verze 1.4.7.

Kromě vývoje nových funkcionalit jsem také aktualizoval většinu závislostí systému na nejnovější verze pro zajištění lepší podporovatelnosti systému. Dále jsem aktualizoval CI/CD pipeline, která je nyní schopna plně automaticky nasazovat nové verze do produkčního i testovacího prostředí.

Veškeré zdrojové kódy SOS jsou dostupné v GitLab repozitáři, který se nachází na adrese `https://gitlab.fit.cvut.cz/sos/sos`. V repozitáři jsem také vytvořil tag `DP-pavlutom`, který označuje stav repozitáře v době odevzdání této práce.

Během semestru se na projektu SOS kromě mě podíleli i studenti předmětu BI-SP1, což rozebírám v následující kapitole Zapojení dalších studentů. Testování a ověřování splnění požadavků se věnuji v poslední kapitole Testování.

[83] 2. května 2023

# Kapitola 4
# Zapojení dalších studentů

*Tato kapitola shrnuje zapojení studentů předmětu BI-SP1 do projektu SOS během letního semestru 2022/23. Popisuji cíle, kterých má tým během semestru dosáhnout, organizaci práce týmu a v závěru shrnuji výsledky práce týmu v době odevzdání této práce před koncem semestru.*

Jak jsem již uvedl v úvodu práce a v předchozí kapitole v sekci 3.1.3, v letním semestru 2022/23 vedeme spolu s vedoucím práce Hunkou a kolegou Hejdou tým studentů v předmětu BI-SP1, který se podílí na rozvoji projektu SOS (dále jen Tým). Tým začal na projektu pracovat na začátku semestru v únoru 2023 a jeho aktivita tak z vetší části probíhala paralelně s mou činností v rámci této práce. Bylo tedy potřeba zajistit efektivní koordinaci mnou realizovaného vývoje a práce Týmu, aby bylo z této příležitosti pro projekt SOS vytěženo maximum.

## 4.1 Cíle BI-SP1 projektu

Hlavní cíle zapojení Týmu do projektu SOS jsou zejména:

- **Zajištění lidských zdrojů pro budoucí rozvoj a údržbu projektu** – Studenti se s projektem seznámí, začnou se podílet na jeho rozvoji v rámci BI-SP1 a potenciálně mohou pokračovat v předmětu BI-SP2 a následně ve svých bakalářských pracích.
- **Zlepšení současné verze SOS** – V rámci této práce v SOS provádím řadu změn velkého rozsahu a implementuji nové funkcionality. Tým může na projektu pracovat současně se mnou a věnovat se dalšímu rozšiřování nových i existujících funkcionalit, opravám chyb a zdokonalování uživatelského rozhraní.
- **Návrh budoucí podoby uživatelského rozhraní** – Naše současná představa je, že by měla být v horizontu přibližně dvou let architektura SOS upravena tak, aby došlo k úplnému oddělení frontendu a backendu. Nový frontend bude nejpravděpodobněji implementován s využitím Vue.js a s backendem bude komunikovat prostřednictvím REST API. Uživatelské rozhraní SOS bude tedy muset být implementováno znovu od začátku, což je dobrou příležitostí k provedení řádného návrhu s využitím již existujícího systému, jeho známých problémů a veškeré zpětné vazby a zkušeností získaných z jeho provozu.

### 4.1.1 Požadavky předmětů BI-SP1 a BI-SWI

Průběh předmětu BI-SP1 jsem již shodou okolností popsal v rámci analýzy požadavků na SOS v sekci 1.2.2. Zde přiblížím náplň tohoto předmětu, která je shodná s projektem v souběžném předmětu BI-SWI, tak jak ji specifikují oficiální materiály pro studenty v systému Moodle [7].

V předmětu BI-SP1 se studenti v rámci práce na projektu věnují činnostem souvisejícím s vývojem softwarového produktu. Mezi tyto činnosti spadá analýza, návrh řešení a částečně i jeho implementace a dokumentace. Zpravidla pak studenti projekt dále rozvíjejí v předmětu BI-SP2. Mezi konkrétní položky které mají studenti v rámci BI-SP1 vypracovat patří zejména:

- **Analýza** – analýza business procesů, požadavků a případů užití,
- **Návrh** – doménový model, databázový model, návrh logické architektury,
- **Implementace** – implementace aplikace podle návrhu, zdrojové kódy verzované systémem pro správu verzí, funkční pipeline pro sestavení aplikace,
- **Dokumentace** – programátorská dokumentace implementovaného řešení, Wiki stránka s postupem pro zprovoznění aplikace.

Materiály také uvádějí, že pokud si to povaha projektu vyžaduje, může vyučující tuto náplň upravit. Je tak běžné, že studenti místo tvorby úplně nové aplikace rozvíjejí již existující projekt, jako je například DBS portál[1] nebo v tomto případě SOS.

### 4.1.2 Využití pro projekt SOS

Dohodli jsme se, že Tým bude provádět analýzu stavu *as-is*, tedy se bude zabývat již existujícím řešením – nejaktuálnější verzí SOS, tak jak vzniká v rámci této práce. Sám v této práci sice analyzuji potřeby předmětů využívajících SOS a z nich vyplývající funkční a nefunkční požadavky, nevěnuji se už ale analýze případů užití a business procesů. Tým může také analyzovat současnou implementaci z doménového hlediska atp. Výstupy této analýzy budou v případě kvalitního provedení cennou dokumentací projektu využitelnou dalšími lidmi, kteří by se v budoucnu mohli na projektu podílet.

Tým by se měl také věnovat návrhu. Položky jako doménový a databázový model jsou vzhledem k již existující aplikaci zahrnuty v analýze. Studenti se dále mohou věnovat návrhu nové podoby uživatelského rozhraní, jak jsem již uvedl výše.

V rámci BI-SP1 se mají studenti věnovat také implementaci. To je opět dobře využitelné, jak jsem již uvedl, studenti se mnou mohou v průběhu semestru spolupracovat na průběžném zdokonalování SOS v průběhu semestru. Při tom si také osvojí práci se všemi používanými technologiemi, seznámí se s existující implementací a v budoucnu tak budou moct projekt nadále rozvíjet a udržovat.

## 4.2 Lidské zdroje

Tým tvoří celkem sedm členů z řad studentů BI-SP1: Timotej Adamec, Lucián Kučera, Jan Mrázek, Jan Jamnický, Marko Hujo, Jan Piták a Jakub Sedláček v roli vedoucího týmu. Dále s Týmem spolupracujeme já a Max Hejda v rolích asistentů. Vyučujícím odpovědným za celý projekt je vedoucí a zadavatel této práce, Jiří Hunka.

Jedním z prvních kroků před zahájením práce které jsem inicioval bylo zjištění znalostí a zkušeností členů Týmu, především s technologiemi, se kterými budou v rámci projektu pracovat. Dle jejich vlastních vyjádření měli na začátku semestru studenti tyto znalosti a zkušenosti:

- **Git** – všichni znají,
- **Python** – 4 znají dobře, zbytek okrajově,
- **Django** – 1 zná dobře, 2 okrajově, zbytek vůbec,

[1] `https://dbs.fit.cvut.cz`

- **Docker** – 1 zná dobře, 1 okrajově, zbytek vůbec,
- **Bootstrap** – 3 znají, zbytek neuvedl,
- **Další technologie** – studenti uvedli také zkušenosti s technologiemi Next.js, Symfony, Twig, Kotlin a Spring.

Celkově se tedy podařilo sestavit tým, který má s technologiemi potřebnými při práci na SOS na poměry předmětu BI-SP1 nadprůměrné zkušenosti. Pro porovnání mohu uvést například projekt, kterého jsem se v BI-SP1 v rámci svého bakalářského studia účastnil já – pracovali jsme s jazykem Kotlin, který byl nový téměř pro všechny členy týmu, pro řadu z nás to také bylo první setkání s Gitem.

### 4.2.1 Rozdělení odpovědnosti

Projektu SOS se aktuálně účastní celkem deset osob. Aby mohla být práce efektivní, bylo třeba definovat odpovědnosti všech zúčastněných. Kromě „řadových členů týmu“, kteří se věnují především plnění zadaných úkolů, se jedná o následující:

- **Vedoucí týmu** – Odpovídá především za organizaci práce studentů. Práce je týmu zadávána jako celku, vedoucí ji přerozděluje mezi jednotlivé členy, kontroluje jejich výstupy a prezentuje je vyučujícímu. Jeho odpovědností je rovněž skrze komunikaci s vyučujícím a asistenty zajistit dostatek úkolů, aby byl tým efektivně vytěžován. Jeho nástrojem pro motivování členů týmu k co nejlepším výkonům je především mechanismus přerozdělování bodů zmiňovaný v předchozích kapitolách této práce.
- **Tomáš Pavlůsek – asistent** – Mojí rolí v týmu je především zadávání implementačních úkolů a dohled nad jejich zpracováním prostřednictvím *code review*. Další mou odpovědností je kontrola analytických úkolů z věcného hlediska – ze všech zúčastněných osob mám největší přehled o funkcionalitách SOS a systémem realizovaných procesech, jelikož jsem autorem většiny implementace. Také jsem zajišťoval *onboarding* studentů, tedy jejich seznámení s projektem z praktické stránky a s používanými technologiemi. Po celý semestr jsem jim také k dispozici pro konzultace ohledně používaných technologií a jakýchkoliv problémů, na které během implementace narazí.
- **Max Hejda – asistent** – Role Maxe Hejdy je podobná jako ta moje, přičemž se soustředí především na udržení kompatibility SOS s prostředím gymnázia. Kromě spolupráce při vedení Týmu jsem také s Hejdou konzultoval velkou část mnou implementovaných úprav a rozšíření skrze *code review*. Cílem bylo jednak zkvalitnění mých výstupů, ale především udržení Hejdy v obraze ohledně prováděných změn, o což během semestru sám jevil zájem.
- **Jiří Hunka – vyučující** – Kromě standardních záležitostí spojených s předmětem BI-SP1, jako je hodnocení práce studentů v průběhu semestru a odborný dohled nad celým projektem, je Jiří Hunka v projektu především v roli mentora a soustředí se zejména na předání dlouhodobé vize projektu. Poskytuje rovněž řadu cenných vstupů při návrhu nového uživatelského rozhraní, díky jeho bohatým zkušenostem a také skutečnosti, že je po celou dobu provozu SOS jeho aktivním uživatelem.

## 4.3 Úkony před zahájením týmové práce

Před zahájením práce s Týmem bylo potřeba, abych repozitář projektu uvedl do stavu umožňujícího efektivní zapojení dalších vývojářů. Z praktického hlediska se jednalo především o nutné úpravy existujících unit testů, které zatím nereflektovaly mnou provedené změny v úvodní fázi vývoje. Pokrytí zdrojových kódů unit testy bylo na začátku semestru 51 %. Dalšími úkony pak

bylo zprovoznění CI/CD, produkčního a testovacího prostředí, vytvoření administračních skriptů a zavedení branching modelu a systému označování verzí, jak popisuji v závěru předchozí kapitoly.

Kromě zmíněných praktických záležitostí jsem také aktualizoval dokumentaci s instrukcemi k lokálnímu spuštění projektu a dalšími základními informacemi v podobě souboru `README.md` v kořenovém adresáři projektu. Posledním krokem pak byla aktualizace GitLab Issue Trackeru, který původně obsahoval především nedostatky systému identifikované během uživatelského testování v rámci mé bakalářské práce a několik dalších, které se objevily během provozu. Část z nich jsem uzavřel, jelikož už byly v nové verzi vyřešeny a přidal řadu dalších, které vyplynuly z právě prováděných úprav. V Issue Trackeru bylo na začátku semestru přibližně 80 otevřených úkolů, Tým tak měl hned od začátku k dispozici dostatek úkolů různých obtížností, které mohli jeho členové využít k seznámení se s projektem prostřednictvím jejich řešení.

## 4.4 Nástroje a infrastruktura

Pro efektivní týmovou práci je nezbytná také jistá infrastruktura – tedy sada nástrojů a procesů využívaných při vývoji. Infrastruktura projektu SOS je záměrně podobná infrastruktuře dalšího projektu na FIT ČVUT, který rovněž spadá pod Jiřího Hunku. Tímto projektem je DBS portál, jehož infrastrukturu popisuje Malec ve své práci z roku 2017 [62, kap. 4]. Projekt SOS zatím nedosahuje takových rozměrů jako DBS portál, a proto je i jeho infrastruktura o něco jednodušší. V budoucnu by ale mohl narůst do podobných rozměrů a jeho infrastruktura by se měla DBS portálu více přiblížit.

### 4.4.1 Studentský odevzdávací systém

SOS je v letním semestru 2022/23 poprvé využíván i předmětem BI-SP1, a tedy i tento projekt vznikl právě v SOS. Před začátkem semestru vytvořil Jiří Hunka v SOS projekt *Vývoj tohoto portálu SOS.fit.cvut.cz*[2], do kterého následně přijal sedm studentů a přiřadil mě a Maxe Hejdu jako asistenty.

V projektu je definováno celkem šest iterací s odevzdáním přibližně jednou za dva týdny. Tým v SOS odevzdává svá řešení v podobě přiložených odkazů do Redmine, kterému se věnuji níže, a využívá SOS k přerozdělování bodů udělovaných vyučujícím – Tým má možnost přerozdělit až 100 % udělených bodů. SOS tak kromě prvotního utvoření týmu slouží především jako primární „zdroj pravdy“ ve vztahu k časům odevzdání a hodnocení.

### 4.4.2 GitLab

Pro projekt jsou vedle uchovávání zdrojových kódů a CI/CD důležité zejména dvě funkcionality GitLabu – *Issue Tracker* a *Merge Requesty*. Jak jsem již zmínil výše, na začátku semestru bylo otevřených přibližně 80 úkolů, které průběžně řešil Tým nebo já sám. Tým zároveň vytvořil řadu nových úkolů, které vzešly jak z Týmem prováděné analýzy, tak i ze samotného používání SOS. Během první poloviny semestru bylo vytvořeno přibližně 50 nových úkolů a přibližně 30 úkolů bylo vyřešeno. Jeden z členů Týmu také pro zvýšení přehlednosti definoval šablonu pro nově vznikající úkoly, díky které jsou požadavky jasně a jednotně specifikovány.

Druhou kritickou funkcionalitou GitLabu je systém tzv. *Merge Requestů*. Pro každý implementační úkol řešitel vytvoří *feature branch* a po dokončení implementace otevře v GitLabu *Merge Request* do větve `develop`. Následně provede jiný člen Týmu *code review* a v okamžiku, kdy se na kvalitě řešení takto shodnou alespoň dva členové týmu včetně autora, je *Merge Request*

[2]Dostupný na `https://sos.fit.cvut.cz/courses/bi-sp121/teams/074312ec-e141-4c79-8bb3-8e14960a55ae/` s nastaveným semestrem *B222*; k zobrazení je třeba mít příslušná oprávnění v SOS

přiřazen mně, abych provedl finální *code review*. Jakmile i já shledám řešení dostatečně kvalitním, *Merge Request* potvrdím, provedu *merge* a úkol v GitLabu tím uzavírám. Alternativně může místo mě finální *code review* a *merge* provádět Max Hejda, v případě mého přílišného vytížení, pokud se implementace týká spíše částí SOS, které vytvořil nebo upravil on, nebo v budoucnu až dokončím studium a projektu se již nebudu aktivně účastnit. Celý životní cyklus úkolu v GitLabu jsem Tým nechal podrobně popsat na Wiki projektu v Redmine, zároveň je patrný z již uzavřených *Merge Requestů* otevřených členy Týmu. Stejným procesem procházely i mnou řešené úkoly v rámci této práce během semestru, s tím rozdílem, že bylo prováděno pouze jedno *code review* a to právě Hejdou.

### 4.4.3 Redmine

Celou koncepci a použití aplikace Redmine již dostatečně popsal ve své práci Malec na příkladu projektu DBS portál [62, kap. 4.1.1], proto zde uvádím pouze specifika projektu SOS. Redmine je při vývoji SOS využíván, podobně jako u DBS portálu, především pro:

- shromažďování úkolů k řešení (i těch, které se netýkají implementace a nejsou tak vedeny v GitLabu),
- záznam času stráveného řešením úkolu či u projektu,
- shromažďování návodů a dokumentace (Wiki).

SOS využívá instanci Redmine provozovanou v rámci DBS portálu[3]. Pro naše účely zde byl zřízen projekt *SOS.fit.cvut.cz* a podprojekt *SOS SP2023*. Úkolům v Redmine se příliš nevěnuji, odpovídám především za úkoly v GitLabu. Tým si úkoly z většiny vytváří i předává v Redmine sám, následně je kontroluje a vyhodnocuje Jiří Hunka. U implementačních úkolů nastává situace, kdy stejný úkol existuje v GitLabu i v Redmine – proto byla k úkolu v Redmine přidána pole „Gitlab Issue“ a „Merge request“, určená k vložení příslušných odkazů na GitLab. Tato pole i se všemi dalšími jsou zobrazena na snímku obrazovky formuláře pro zadání nového úkolu do Redmine na obrázku 4.1. Úkol v Redmine pak slouží především k zaznamenávání stráveného času a k přiřazení k vybrané iteraci.

Kromě správy úkolů slouží Redmine také ke shromažďování dokumentace. V hlavím projektu byla vytvořena Wiki, kam Tým vkládá výstupy veškerých úkolů, které nejsou implementační. To jsou například výstupy analýzy a další záležitosti vyžadované předmětem BI-SWI nebo zprávy o průběhu a výsledcích výzkumů prováděných s uživateli. Na Wiki jsou také vedeny zápisy ze všech schůzek Týmu. Jsou zde také dokumentovány důležité procesy, jako například životní cyklus úkolu v GitLabu nebo s práce s větví v Gitu. Nakonec se zde nachází sekce s důležitými odkazy souvisejícími s projektem.

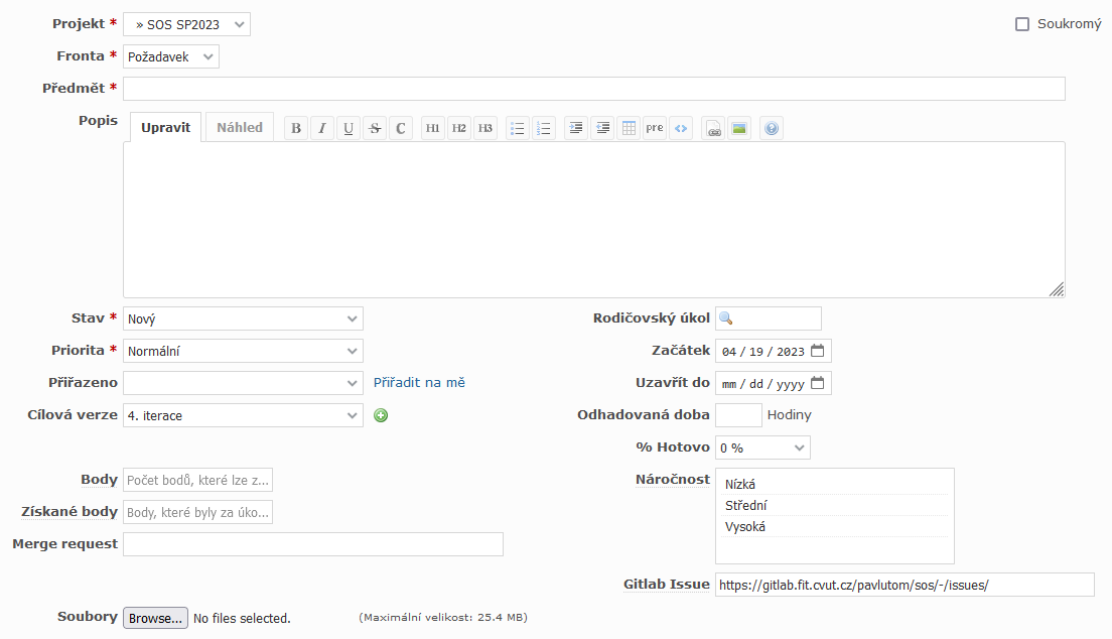
### 4.4.4 Slack

Po vzoru DBS portálu jsem pro projekt SOS vytvořil Slack Workspace[4]. Slack umožňuje komunikovat jak přes soukromé zprávy, tak i prostřednictvím kanálů zaměřených na určité téma. Kanály mohou být otevřené nebo uzavřené. Workspace SOS obsahuje následující kanály:

- *#django* – dotazy a diskuze o frameworku Django,
- *#docker* – dotazy a diskuze o technologii Docker,
- *#general* – základní a výchozí kanál,

[3] `https://dbs.fit.cvut.cz/redmine/`
[4] `https://sos-fit.slack.com`

**Obrázek 4.1** Formulář pro vytvoření nového úkolu v Redmine




- *#gitlab-integration* – kanál pro integraci Slack–GitLab, automaticky se zde objevují notifikace o důležitých událostech v GitLabu, především o CI/CD, Issues a Merge Requestech,
- *#python* – dotazy a diskuze o jazyku Python,
- *#random* – druhý z výchozích kanálů, slouží pro neprojektovou diskuzi a spam,
- *#sp2023* – společná komunikace Týmu, asistentů a vyučujícího
- *#vedení* – uzavřený kanál vedení projektu, jehož členem jsem aktuálně já, Max Hejda a Jiří Hunka.

Kromě těchto kanálů pravděpodobně existuje ještě nejméně jeden uzavřený kanál pro interní komunikaci Týmu.

### 4.4.5 Schůzky

Kromě všech zmíněných nástrojů, které více či méně slouží ke komunikaci a organizaci týmové práce, jsou pro projekt důležité také osobní schůzky, které probíhají jednou týdně a trvají přibližně jednu hodinu. Schůzek se až na výjimečné případy účastní všichni členové Týmu, asistenti i vyučující. Hlavním účelem těchto schůzek je zadávání nových úkolů, prezentace dokončených úkolů a konzultace ohledně rozpracovaných úkolů nebo jakýchkoliv problémů týkajících se projektu. Týdenní schůzky jsou standardním a ověřeným konceptem předmětu BI-SP1 [4] a pro účely projektu SOS nebyl důvod je nevyužít nebo na konceptu cokoliv měnit.

## 4.5 Odvedená práce a výstupy

V předchozích sekcích popisuji, jakým způsobem bylo do projektu SOS zapojeno sedm studentů předmětu BI-SP1. To je pro budoucnost projektu velmi pozitivní, především s vyhlídkou jejich dalšího zapojení v předmětu BI-SP2 v následujícím semestru a potenciálně i v rámci jejich

bakalářských prací. Ještě cennější jsou ale výsledky jejich práce, kterou během semestru odvedli a ještě odvedou.

### 4.5.1 Analýza

Redmine Wiki projektu[5] obsahuje veškeré výstupy analýzy, kterou Tým prováděl pro splnění požadavků předmětů BI-SP1 a BI-SWI. Jedná se o následující položky:

- Funkční a nefunkční požadavky
- Seznam účastníků/aktérů
- Model případů užití a detailní scénáře
- Popisy šesti vybraných business procesů
- Diagramy aktivit
- Analytický (doménový) model s popisy všech entit a jejich atributů
- Stavové diagramy pro pět vybraných entit
- Popis logické architektury aplikace (Model–View–Template)
- Databázový model – konceptuální model a relační schéma
- Diagram balíčků (Django aplikace)
- Vyčerpávající seznam použitých technologií s popisy jejich využití

Vybrané položky jsou kromě Redmine dostupné také v příloze B. Pro budoucí vývojáře projektu SOS jsou velmi cennou dokumentací, jelikož téměř vyčerpávajícím způsobem popisují celý systém SOS. Forma strukturovaných popisů a UML diagramů v Redmine Wiki je zároveň pro budoucí vývojáře nepochybně mnohem přístupnější, než bakalářské či diplomové práce související s SOS, které dosud byly jediným dostupným zdrojem těchto informací. Ty zároveň není možné odpovídajícím způsobem aktualizovat současně s prováděním změn v systému.

Všechny tyto položky vytvořili sami členové Týmu. Já jim byl k dispozici pro konzultace a kontroloval jsem výstupy, aby odpovídaly skutečnosti. Položka „Funkční a nefunkční požadavky“ se již podle názvu kryje se sekcí 1.2.5 této práce. Požadavky jsme já a Tým analyzovali nezávisle na sobě, jelikož jsem jednak potřeboval mít tuto analýzu provedenou ještě před začátkem semestru, zároveň jsem nechtěl Tým o zkušenost s analýzou požadavků připravit. V budoucnu by ale bylo vhodné obě analýzy požadavků nějakým způsobem sloučit a ve Wiki uvést finální verzi.

### 4.5.2 Výzkumy s uživateli

Jedním z prvních úkolů Týmu bylo shromáždění zpětné vazby na existující SOS. Na začátku semestru tedy provedli kvantitativní i kvalitativní výzkum. Kvantitativní výzkum byl v podobě dotazníku, který byl rozeslán studentům předmětu BI-SP1 a studentům, kteří absolvovali NI-NUR v posledních dvou letech. Z odpovědí vyplynulo několik problémů týkajících se nepřehlednosti uživatelského rozhraní, které studenti uváděli opakovaně, a objevilo se několik návrhů nových funkcionalit.

Kvalitativní výzkum Tým prováděl formou strukturovaných rozhovorů s vyučujícími BI-SP1 včetně garanta. Jejich průběh a získané poznatky jsou opět zdokumentovány v Redmine. Dva

[5] `https://dbs.fit.cvut.cz/redmine/projects/sos-fit-cvut-cz/wiki`

členové Týmu, kteří mají současně zapsaný předmět BI-TUR[6], také prováděli *usability* testování se studenty. Vytvořili testovací data a scénáře a následně subjekty pozorovali při průchodech systémem. Audiovizuální záznamy jsou taktéž dostupné v Redmine.

### 4.5.3 Identifikace problémů současného řešení

Na základě všech uvedených vstupů Tým vytvořil seznam zjištěných nedostatků a návrhů na zlepšení, který je dostupný v příloze B a obsahuje přes 50 položek. Nad tímto seznamem jsme následně společně diskutovali, například o prioritách, o možných řešeních a o náročnosti jejich implementace. Některé položky se již překrývaly s existujícími úkoly v GitLabu, k většině ostatních byly úkoly vytvořeny.

### 4.5.4 Návrh nového uživatelského rozhraní

Jak jsem již avizoval v první sekci této kapitoly, jedním z cílů Týmu je i vytvoření návrhu nového uživatelského rozhraní SOS, které má být implementováno v budoucnu spolu s plánovanou změnou architektury systému. Návrhy vytvářejí především členové Týmu, kteří mají současně zapsaný i předmět BI-TUR. K tvorbě prototypu využívají nástroj Figma[7]. Figma projekt je sdílený a odkaz na něj se rovněž nachází v Redmine Wiki, spolu s průběžnými exporty v podobě archivů a PDF dokumentů. Prototyp je průběžně prezentován a validován na pravidelných projektových schůzkách.

Dlouhodobým cílem je vytvořit kompletní prototyp celého uživatelského rozhraní SOS. Aktuálně[8] je rozpracovaná nová podoba domovské obrazovky a nová obrazovka detailu týmu, na které by nově měla být realizována i veškerá agenda spojená s odevzdáváním a ohodnocováním řešení studentů. Vybrané části prototypu jsou dostupné v příloze B.

### 4.5.5 Implementace

V neposlední řadě se Tým přímo podílí na implementaci SOS. Během semestru jsem zatím uzavřel celkem 14 Merge Requestů vytvořených členy Týmu, u kterých jsem provedl *code review*. Přehled konkrétních Merge Requestů je na obrázku 4.2. Jednotlivé úkoly byly většinou ze seznamu nedostatků a návrhů na zlepšení, o kterém pojednává sekce 4.5.3, nebo se jednalo o řešení chyb, které se během semestru objevily. Některé Merge Requesty se také týkaly vývojářské dokumentace v GitLabu.

Ze začátku semestru byla efektivita členů Týmu coby vývojářů nižší, neboť se s projektem teprve seznamovali. Velmi rychle se ale v projektu a v použitých technologiích zorientovali a nyní již jsou schopni implementaci provádět efektivně. Do konce semestru očekávám ještě nejméně jednou tolik vyřešených implementačních úkolů.

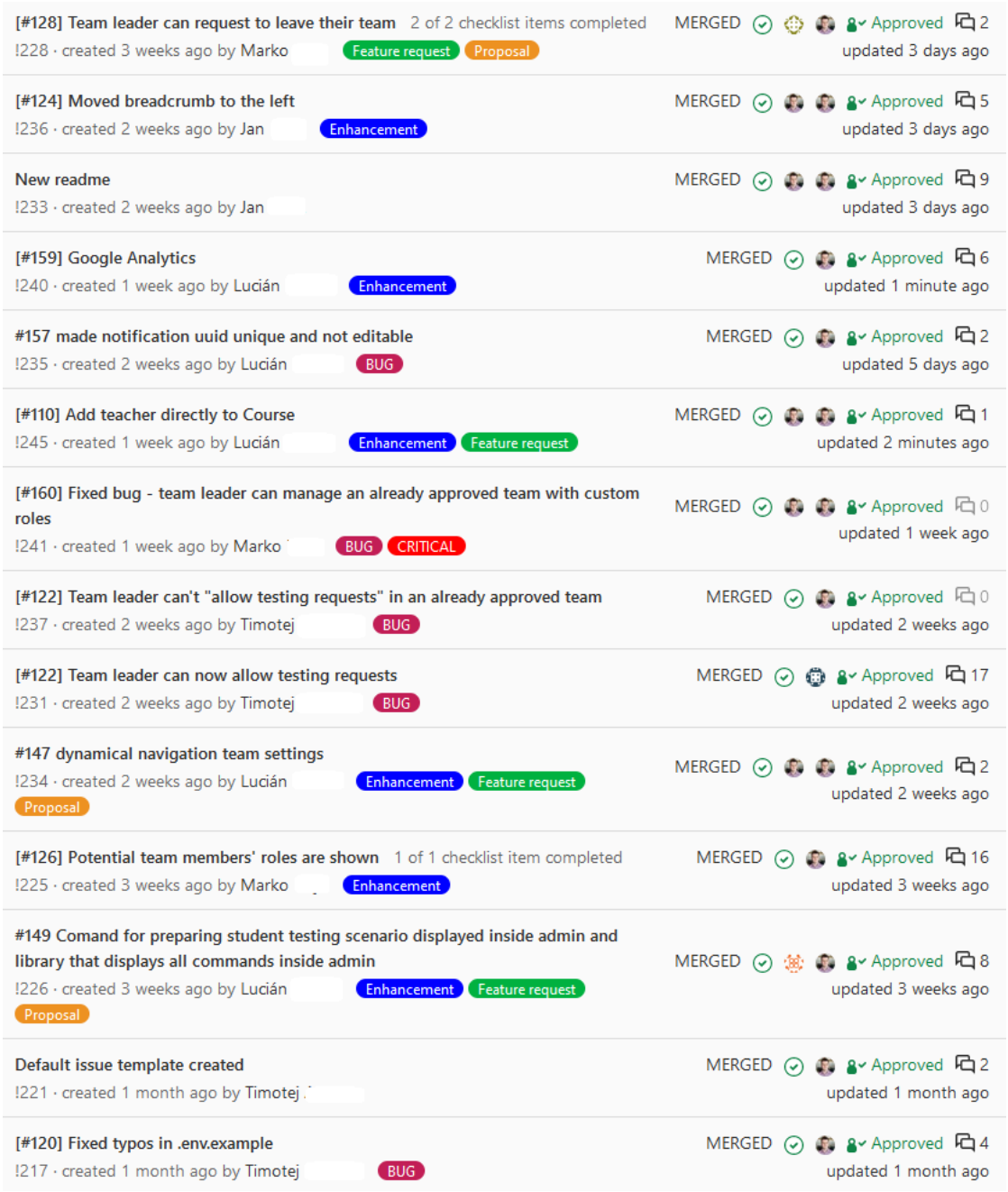
### 4.5.6 Uživatelská dokumentace

Tým zpracoval také uživatelskou dokumentaci, která je dostupná na `https://sos.pages.fit/dokumentace/`. Dokumentace je vytvořena v technologii VuePress[9], která umožňuje generování statických stránek ze souborů ve formátu Markdown a je nasazená pomocí služby PagesFIT[10].

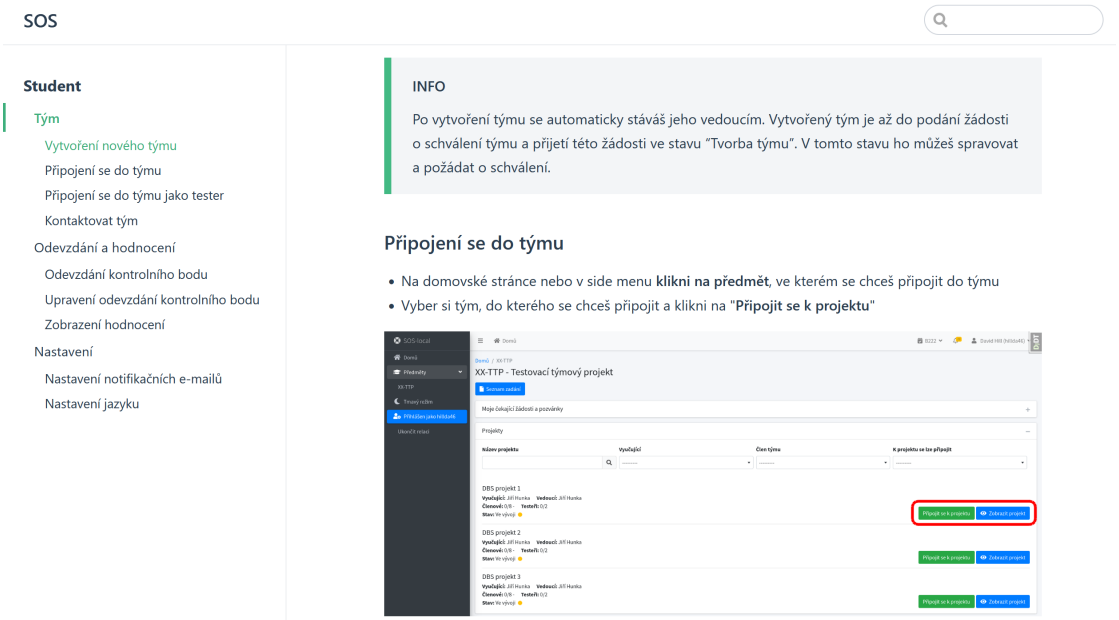
Uživatelská dokumentace vychází ze scénářů vypracovaných v rámci analýzy popsané v sekci 4.5.1. Scénáře popisují aktivity, které mohou studenti a vyučující v SOS realizovat a pro názornost

[6] *BI-TUR: Tvorba uživatelského rozhraní* – bakalářský předmět s podobným zaměřením jako NI-NUR
[7] `https://www.figma.com`
[8] ke dni 20. dubna 2023
[9] `https://vuepress.vuejs.org/`
[10] `https://help.fit.cvut.cz/pages-fit/index.html`

**Obrázek 4.2** Uzavřené Merge Requesty, jejichž autory jsou členové Týmu



**Obrázek 4.3** Ukázka uživatelské dokumentace na PagesFIT




jsou doplněny o snímky souvisejících obrazovek. Ukázka uživatelské dokumentace je na obrázku 4.3.

### 4.5.7 Dokumentace pro vývojáře

Pro usnadnění budoucího zapojení dalších vývojářů do projektu Tým kromě již zmiňované dokumentace v Redmine Wiki také v průběhu semestru aktualizoval instrukce ke zprovoznění projektu v souboru `README.md`, jak je patrné například z Merge Requestu !233. Součástí této aktualizace je i videonávod k lokálnímu spuštění aplikace vytvořený členem Týmu, dostupný na YouTube[11].

## 4.6 Shrnutí

V této kapitole jsem popsal zapojení týmu studentů předmětu BI-SP1 do projektu SOS. Na začátku jsem identifikoval cíle, kterých by měl Tým během semestru v projektu dosáhnout a popsal, jakým způsobem jsou tyto cíle v souladu s požadavky předmětů BI-SP1 a BI-SWI, které musí studenti během semestru splnit. Popsal jsem rozdělení odpovědností a role jednotlivých účastníků projektu, nastavenou infrastrukturu projektu a použití nástrojů pro týmovou spolupráci, včetně samotného SOS. Věnoval jsem se také úkonům, které jsem realizoval před zahájením týmové spolupráce, aby byla co nejefektivnější.

V poslední části kapitoly jsem shrnul veškeré výsledky, kterých Tým během semestru dosáhl. Tým vypracoval komplexní analýzu aktuální verze SOS, která je pro všechny aktuální i budoucí vývojáře systému velmi cennou dokumentací. Po celou dobu jsem byl Týmu k dispozici pro konzultace ohledně fungování systému a dohlížel jsem na faktickou správnost výstupů analýzy. Tým také realizoval kvantitativní i kvalitativní výzkumy se skutečnými uživateli systému a zpětnou vazbu odpovídajícím způsobem zpracoval, aby mohla být zohledněna při dalším vývoji. Dále také Tým rozpracoval prototyp nového uživatelského rozhraní, které má být v budoucnu implementováno.

[11] `https://youtu.be/fthXCmbAZYs`

Členové Týmu se zároveň na projektu aktivně podíleli jako vývojáři. Během semestru využili již známé nebo i jimi identifikované problémy systému k tomu, aby se při jejich řešení seznámili s jeho implementací a s použitými technologiemi. V průběhu semestru jsem na implementaci prováděnou členy Týmu dohlížel skrze *code review*, s členy Týmu jsem diskutoval jejich řešení a snažil jsem se jim projekt a používané technologie co nejvíce zpřístupnit. V průběhu semestru jsem pozoroval zvyšující se efektivitu členů Týmu jakožto vývojářů a jelikož už jsou nyní s projektem i s používanými technologiemi dostatečně obeznámeni, nebojím se říct, že budou například pod vedením Maxe Hejdy schopni projekt udržovat a vyvíjet i nadále, bez mé přítomnosti.

# Kapitola 5
# Testování

*V poslední kapitole se zabývám ověřováním správné funkčnosti jak realizovaných úprav a rozšíření, tak systému jako celku. Popisuji existující automatické testy a jejich realizované rozšíření. Dále se věnuji začlenění automatického testování do nastavené infrastruktury projektu. V druhé části kapitoly se zabývám testovacím prostředím, manuálními testy a provedeným akceptačním testováním. Nakonec shrnuji poznatky získané během provozu SOS v prostředí FIT ČVUT v letním semestru 2022/23 a identifikuji oblasti, na které by měl být budoucí vývoj zaměřen.*

## 5.1 Automatické testování

Žádný softwarový projekt většího rozsahu se neobejde bez automatických testů. Automatické testy jsou základním nástrojem k zajištění stability vyvíjeného systému a k ověřování funkčnosti všech jeho částí. V této sekci se věnuji existujícím automatickým testům a jejich aktualizaci, především vzhledem k provedeným změnám. Dále také popisuji spouštění automatických testů v rámci CI/CD pipeline, interpretaci výsledků ve webovém rozhraní systému GitLab a integraci s komunikačním nástrojem Slack.

### 5.1.1 Unit testy

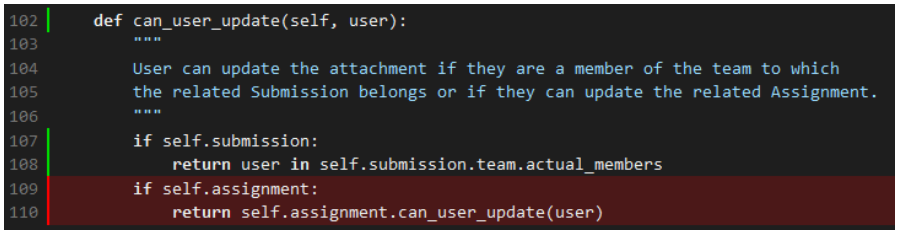
Jedním ze základních nástrojů k ověření funkčnosti jednotlivých částí systému jsou unit testy. Většinou se jedná o tzv. *white-box* testy, tedy je znám kód testovaných tříd a metod a je přímo testováno jejich chování. Proto unit testy většinou píše přímo vývojář, který je autorem testovaného kódu. Unit testy kromě ověřování správného chování vyvíjených rozšíření slouží také jako testy *regresní* – průběžně ověřují, zda prováděné změny nenarušily funkčnost již funkčních částí systému.

#### 5.1.1.1 Pokrytí kódu testy

Jednou z metrik týkajících se unit testů je množství kódu pokrytého těmito testy, většinou uváděné v procentech. Ke sledování pokrytí využívá SOS nástroj Coverage.py[1], který dokáže označit jednotlivé řádky kódu podle toho, zda byly nebo nebyly během spouštění testů provedeny. Výstupem je jednak textový report pokrytí v procentech, celkově a pro jednotlivé moduly, ale dle konfigurace také například HTML dokument s viditelným označením řádků nebo JUnit XML pro

[1] `https://coverage.readthedocs.io/en/7.2.3/`

**Obrázek 5.1** Ukázka výstupu nástroje Coverage.py ukazující pouze částečné pokrytí kódu testy




zpracování dalšími nástroji. Ukázka HTML dokumentu s označenými řádky kódu podle pokrytí testy je na obrázku 5.1.

#### 5.1.1.2 Existující testy

Pro SOS jsem již v rámci své bakalářské práce vytvořil sadu unit testů, které se zaměřovaly především na funkčnost jednotlivých Modelů[2] [1, kap. 4.1]. Další testy pak vytvořil v rámci své práce Hejda, když SOS rozšiřoval pro využití na gymnáziu – testy pokryl nové funkcionality, které v systému realizoval [2, kap 4.1].

#### 5.1.1.3 Aktualizace testů

V rámci této práce jsem v systému provedl některé výrazné změny a především implementoval řadu nových funkcionalit. Z časových důvodů jsem ale bohužel nezvládl průběžně odpovídajícím způsobem rozšiřovat i sadu unit testů, především v úvodní fázi vývoje před začátkem semestru, v důsledku čehož kleslo celkové pokrytí kódu testy z původních 62 % na 51 % ve verzi 1.0.0.

Pro zajištění stability projektu jsem se tedy v závěrečné fázi vývoje věnoval tvorbě unit testů, abych pokrytí kódu výrazně zvýšil. Některé nové unit testy souvisely funkcionalitami implementovanými během semestru a byly přidány již v průběhu vývoje, ať už mnou nebo členy Týmu, čímž se pokrytí zvýšilo řádově o jednotky procent. Na zvýšení pokrytí ale měly největší podíl Merge Requesty !227, !256, !257, !258 a !259, ve kterých jsem se zaměřil čistě na tvorbu unit testů pro již existující kód.

Verzi SOS obsahující tyto testy a opravy drobných chyb, které jsem během tvorby testů objevil, jsem označil 1.4.7. Ve verzi 1.4.7 existuje celkem 508 unit testů a pokrytí kódu testy je 95 %, což považuji za dostatečné[3]. Pokryté jsou veškeré funkcionality implementované v rámci této práce, veškeré funkcionality využívané v prostředí FIT ČVUT a celkově většina funkcionalit, které tato verze SOS nabízí. Výjimkou jsou pouze některé funkcionality používané v prostředí GJGJ, které jsem neměl možnost lokálně otestovat a analyzovat, abych k nim vytvořil odpovídající automatické testy.

#### 5.1.1.4 Nástroje pro spouštění testů

Framework Django nabízí řadu nástrojů k tvorbě a spouštění unit testů, které jsou popsány v jeho dokumentaci [63]. Nástroje poskytované frameworkem jsou z velké části postavené na modulu *unittest*, který je součástí standadní knihovny jazyka Python. Pro samotnou definici a spouštění testů by tyto nástroje byly i nadále vyhovující, nastal ovšem problém při integraci s GitLabem, kterou popisuji v sekci 5.1.3. Výchozí spouštěč testů frameworku Django totiž nedokáže vytvořit výstup ve formátu JUnit XML sloužící ke zobrazení výsledků ve webovém rozhraní GitLabu.

[2] Modelem je v tomto případě myšlena třída, která je potomkem `django.models.Model` a reprezentuje nějakou entitu v systému.

[3] Počet testů a pokrytí kódu jsou patrné například z této pipeline: `https://gitlab.fit.cvut.cz/sos/sos/-/pipelines/271286`

**Výpis kódu 5.1** Ukázka JUnit XML souboru s výsledky testů

```
<?xml version="1.0" encoding="utf-8"?>
<testsuites>
    <testsuite name="pytest" errors="0" failures="0" skipped="0" tests="160"
               time="200.557" timestamp="2023-04-14T12:55:57.333662"
               hostname="bae2b6d2ec60">
        <testcase classname="... .test_models.AssignmentTestCase"
                  name="test_assignment_create_basic"
                  file="apps/assignments/tests/test_models.py"
                  line="157" time="15.795"/>
        <testcase classname="... .tests.test_models.AssignmentTestCase"
                  name="test_assignment_create_with_attachments"
                  file="apps/assignments/tests/test_models.py"
                  line="192" time="5.799"/>
        ...
    </testsuite>
</testsuites>
```

Z tohoto důvodu jsem místo něj nasadil framework Pytest[4]. Ten JUnit XML výstup podporuje a kromě vlastního způsobu definice testů dokáže pracovat i s testy založenými na modulu *unittest* [64] a spolu s pluginem *pytest-django*[5] tak dokáže spustit i všechny existující testy bez nutnosti jakýchkoli úprav.

Pro spuštění testů lokálně i v CI/CD pipeline slouží skript `run_unit_tests.sh`, který zajišťuje identifikaci a spuštění všech testů pomocí frameworku Pytest, analýzu pokrytí kódu testy nástrojem Coverage.py a vygenerování všech potřebných výstupů.

### 5.1.2 Code style

PEP 8 je tzv. *style guide* přijímaný Python komunitou od roku 2001. Jedná se o soubor pravidel a konvencí, jakým způsobem má být Python kód formátován, aby byl vždy co nejsrozumitelnější [65]. K zajištění dodržování *code style* projekt SOS využívá nástroj Black[6], který je vyvíjen a udržován organizací Python Software Foundation. Kromě podpory obecné přehlednosti kódu je výrazným přínosem využití tohoto nástroje také snazší analýza provedených změn v kódu, například v rámci *code review* – v kódu se totiž téměř nevyskytují změny týkající se pouze formátování, jsou zobrazeny pouze skutečné úpravy kódu.

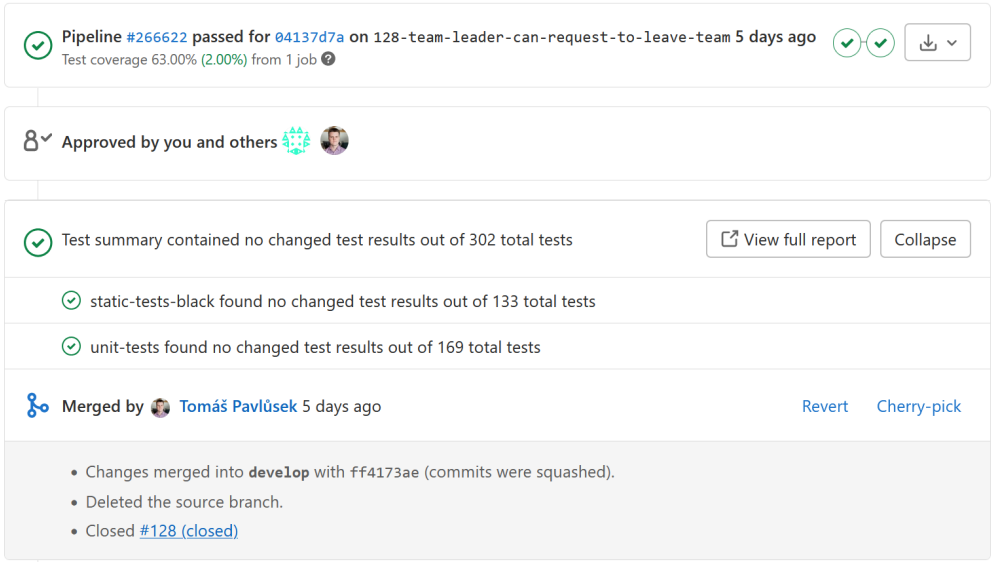
Black je potřeba používat k formátování kódu minimálně před vytvořením nového commitu, manuálně nebo například pomocí *pre-commit* hooku[7]. Black je totiž spouštěn i skriptem `run_static_tests.sh` v CI/CD pipeline a při detekci nedodržení pravidel pipeline ukončí.

### 5.1.3 Integrace s infrastrukturou

Jak jsem již zmínil v přechozích sekcích, výstupem skriptů provádějících testy jsou reporty ve formátu JUnit XML [66]. Jedná se o standardizovaný formát pro reportování výsledků automatických testů. Ukázka formátu je ve výpisu 5.1. Tyto XML soubory jsou v pipeline uloženy jako tzv. *artefakty*. Pokud jsou správně označeny, GitLab je pak dokáže načíst a zobrazit na příslušných obrazovkách svého webového rozhraní. Celý proces konfigurace je podrobně popsán v oficiální dokumentaci [67].

[4] `https://docs.pytest.org/en/7.3.x/`
[5] `https://pytest-django.readthedocs.io/en/latest/`
[6] `https://github.com/psf/black`
[7] `https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks`

**Obrázek 5.2** Ukázka shrnutí výsledků testů u Merge Requestu




Pipeline produkuje celkem tři JUnit XML artefakty – výsledky unit testů, výsledky kontroly formátování a report o pokrytí kódu testy. Report o pokrytí kódu testy je generován přímo nástrojem Coverage.py příkazem `coverage xml`. Report o výsledcích unit testů je generován frameworkem Pytest jeho zavoláním s přepínačem `--junitxml`, což byl hlavní důvod jeho použití namísto výchozího spouštěče testů frameworku Django. Pro vygenerování reportu o výsledcích kontroly formátování kódu jsem použil nástroj *black-junit*[8], jelikož sám Black tuto funkcionalitu přímo nepodporuje.

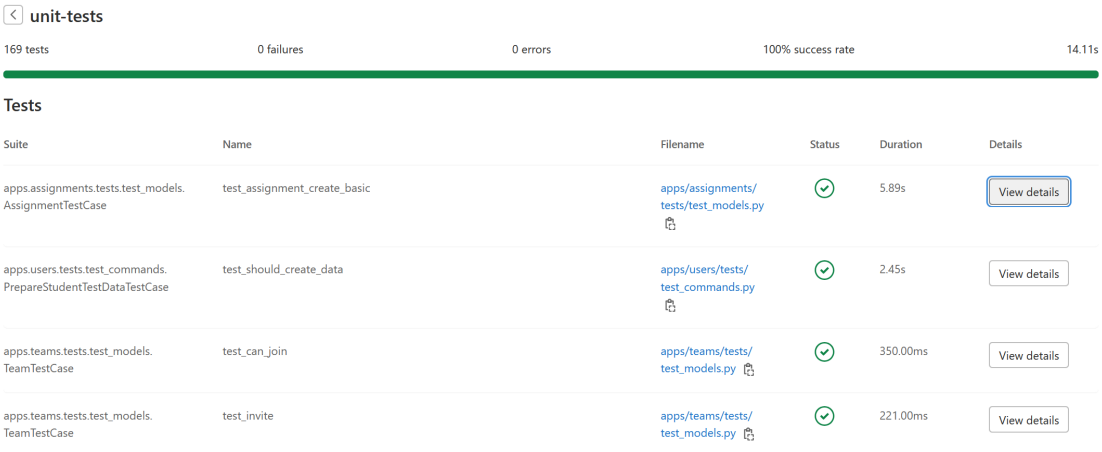
Prvním místem, kde se uvedené artefakty používají je základní shrnutí Merge Requestu. Jak je patrné z obrázku 5.2, zobrazuje se zde celkové pokrytí kódu testy a informace po počtech úspěšně a neúspěšně provedených testů. Dalším místem je tzv. *diff*, tedy přehled konkrétních změn v kódu v Merge Requestu. Zde jsou jednotlivé řádky označeny zeleným nebo červeným pruhem na levé straně výpisu, podle toho, zda byly během spouštění testů provedeny, jak je vidět na obrázku 5.3. To je velmi užitečné při *code review*, jelikož tak lze snadno identifikovat neotestovaný kód, který je potřeba důkladněji zanalyzovat, nebo autorovi vrátit, aby k němu vytvořil testy. Posledním místem je pak kompletní přehled všech jednotlivých testů a jejich výsledků, viditelný na obrázku 5.4. Po otevření některé z položek se zobrazí detailní přehled s informacemi o testu, případně chybové hlášky pokud byl test neúspěšný. Díky tomu není třeba tyto informace nepohodlně hledat v konzolovém výstupu CI/CD pipeline.

Další integrací, kterou jsem nakonfiguroval a částečně s testy také souvisí, jsou Slack notifikace z GitLabu. Konfiguraci jsem provedl v GitLabu[9] i ve Slacku[10] podle jejich dokumentací. V dedikovaném Slack kanálu *#gitlab-integration* se pak objevují notifikace, především o úspěchu nebo neúspěchu CI/CD pipeline na větvích `develop` a `master`. Díky tomu lze snadno identifikovat selhání testů v důležitých větvích a problém hned řešit, bez nutnosti neustále sledovat webové rozhraní GitLabu nebo e-mailové notifikace. Ukázka Slack notifikací je k dispozici na obrázku 5.5.

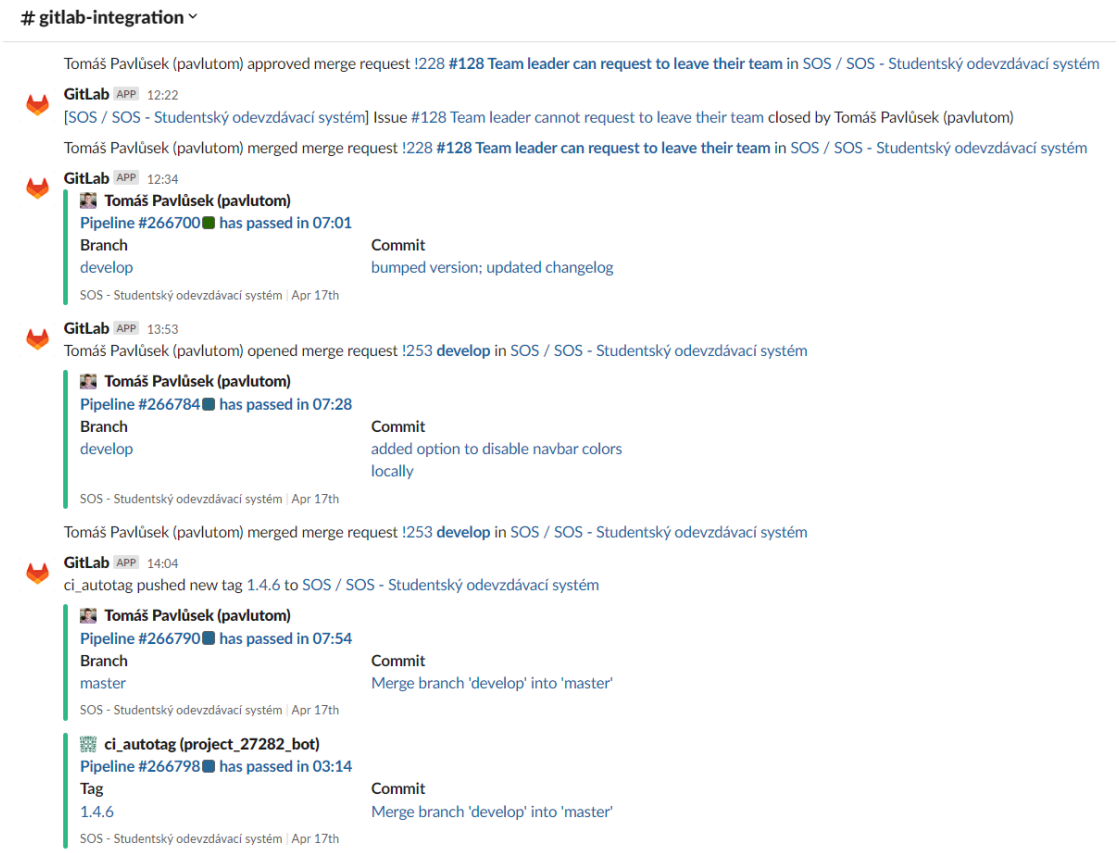
[8] `https://pypi.org/project/black-junit/0.2.2/`
[9] `https://docs.gitlab.com/15.10/ee/user/project/integrations/slack.html`
[10] `https://api.slack.com/messaging/webhooks`

**Obrázek 5.3** Ukázka označení pokrytí řádků kódu testy v GitLab rozhraní


**Obrázek 5.4** Ukázka kompletního reportu výsledků testů v GitLabu



**Obrázek 5.5** Ukázka Slack notifikací o událostech v GitLabu



## 5.2 Manuální testování

Manuální testování je proces, při kterém je funkčnost systému ověřována člověkem prostřednictvím průchodů systémem, ať už neřízených nebo podle předem určených scénářů. Tento proces doplňuje automatické testování a obecně se často zaměřuje na ověření funkčnosti a použitelnosti uživatelského rozhraní vyvíjeného systému. Lze jej provádět v různých fázích vývoje a může sloužit jak k odhalení problémů již v raných fázích vývoje, tak i k ověření celkové správné funkčnosti systému v realistických podmínkách. V této sekci popisuji testovací prostředí a jeho význam, procesy manuálního testování během vývoje a nakonec se zabývám akceptačním testováním, které bylo provedeno před využitím SOS ve výuce předmětů BI-SP1 a BI-SWI v letním semestru 2022/23.

### 5.2.1 Testovací prostředí

Jak jsem již zmínil v sekci 3.8, kromě produkčního prostředí existuje ještě testovací prostředí dostupné na `https://sos2.fit.cvut.cz/`. Jedná se o prakticky totožné prostředí jako produkční, pouze s tím rozdílem, že je zde nasazována vývojová verze SOS z větve `develop`. Od produkčního prostředí je plně izolované, má vlastní virtuální stroj, vlastní úložiště i vlastní databázi. To je rozdíl například oproti projektu DBS portál. Ten rovněž využívá testovací prostředí[11], které ale pracuje s produkční databází namísto vlastní. To má určité výhody i nevýhody. Výhodou společné databáze je především snadnější testování nových verzí na produkčních datech. Je zde ale nebezpečí, že se v důsledku chyby ve vývojové verzi poškodí produkční data. Proto jsem zvolil pro projekt SOS plně oddělené testovací prostředí. Pro kopírování produkčních dat do testovacího prostředí jsem pak vytvořil skript[12], který zkopíruje obsah databáze a soubory nahrané uživateli. Skript je podle potřeby spouštěn manuálně.

#### 5.2.1.1 Význam testovacího prostředí

Jedním z významů testovacího prostředí je testování databázových migrací, především těch datových. Právě k tomu je nezbytné mít k dispozici jednoduchý mechanismus pro zkopírování produkčních dat. V případě, že se v důsledku vadných migrací nějakým způsobem poškodí data, je možné problém v testovacím prostředí vyšetřit, migrace opravit, a pak znovu zkopírovat zdravá produkční data a test opakovat.

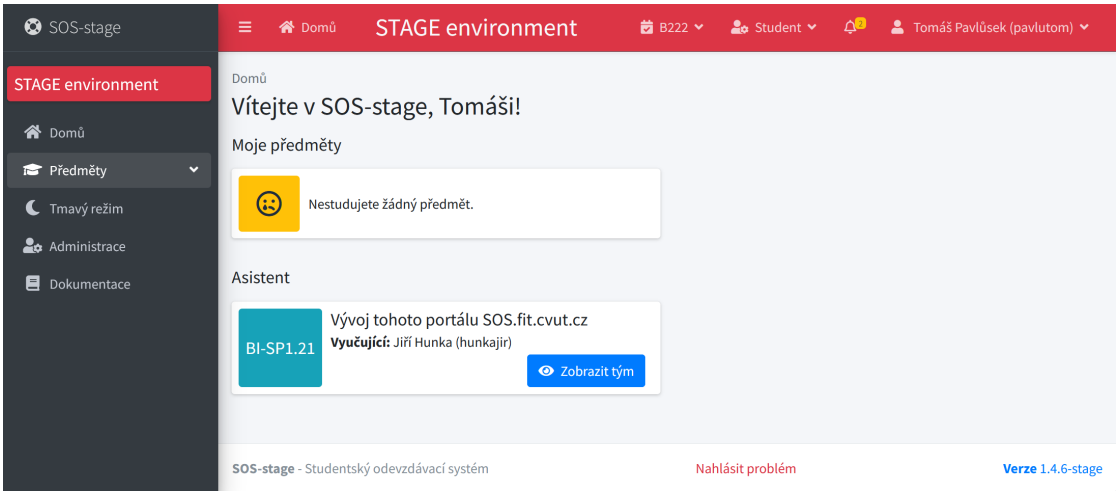
Dalším významem testovacího prostředí je pohodlné manuální testování nových verzí. Veškeré úpravy a rozšíření by sice měly být odpovídajícím způsobem pokryty automatickými testy, nicméně ne vždy dokáží automatické testy odhalit veškeré problémy. Testovací prostředí s kopií produkčních dat dává testerovi možnost důkladně vyzkoušet všechny funkcionality s reálnými daty, a to i agresivním způsobem – může se záměrně pokoušet o vyvolání chyby a testovat chování systému v okrajových případech. Také může volně provádět akce, které sice data nepoškodí ve smyslu jejich konzistence, ale stále vedou k jejich znehodnocení, tedy například zápisy nereálných hodnot nebo neautorizované zásahy v režimu impersonace. Díky využití kopie produkčních dat se zároveň výrazně redukuje nutnost vytvářet pro tyto účely umělá testovací data, u nichž je vždy riziko, že nebudou plně vystihovat reálnou situaci, v důsledku čehož by mohly některé problémy zůstat skryty.

#### 5.2.1.2 Vizuální odlišení testovacího prostředí

V praxi se později ukázalo, že je pro vývojáře či testera velmi snadné splést si testovací a produkční prostředí, především pokud má v obou z nich stejná oprávnění a v obou jsou stejná nebo podobná data. Proto jsem testovací prostředí vizuálně odlišil změnou barvy navigační lišty z bílé

[11] `https://dbs2.fit.cvut.cz`
[12] `scripts/sync_prod_db.sh`

**Obrázek 5.6** Odlišný vzhled uživatelského rozhraní v testovacím prostředí




na výraznou červenou a zobrazením textu „STAGE environment“ jak v navigační liště, tak i v postranním panelu. Odlišný vzhled testovacího prostředí je viditelný na obrázku 5.6.

### 5.2.2 Testování během vývoje

Kromě automatických testů a využití testovacího prostředí jsem SOS také neustále podroboval manuálnímu testování lokálně během vývoje nových funkcionalit. Hlavní zásadou bylo projít v aplikaci s využitím testovacích dat všechny scénáře, které měla vyvíjená funkcionalita řešit, a to ještě předtím, než byla nová implementace podrobena *code review*. Účelem této aktivity bylo především odhalit chyby, které neodhalily unit testy, ať už z důvodu jejich nedokonalého návrhu, nebo jejich absence, což se dělo především v úvodní fázi vývoje. V případě, že jsem během manuálního testování objevil problémy a součástí implementace byly i unit testy, soustředil jsem se kromě samotné opravy také na rozšíření těchto testů tak, aby objevenou chybu pokryly.

#### 5.2.2.1 Testování v rámci code review

Kromě manuálního testování vývojářem bylo lokální manuální testování často realizováno i v rámci *code review*, ať už Maxem Hejdou provádějícím *code review* mnou implementovaných úprav a rozšíření, nebo mnou a členy Týmu v rámci *code review* změn prováděných členy Týmu. Dělo se tak především v případě rozsáhlejších změn nebo zcela nových funkcionalit.

#### 5.2.2.2 Testování před nasazením první verze

Jednou ze situací, kdy bylo manuální testování obzvlášť důležité bylo nasazování verze 1.0.0 do prostředí FIT ČVUT. Přechod mezi verzemi 0.6.2 a 1.0.0 obnášel rozsáhlé změny datového modelu a s tím související velké množství datových migrací. V tu dobu navíc zatím nebylo k dispozici testovací prostředí. Bylo potřeba ověřit, zda datové migrace fungují správně na všech existujících produkčních datech, tedy nejen z FIT ČVUT, ale i z GJGJ. Proto jsme si s Maxem Hejdou oba vytvořili kopie produkčních dat z námi spravovaných prostředí a lokálně na nich databázové migrace otestovali. Při tom jsme objevili několik problémů, které jsem následně opravil. Některé problémy spočívaly v pádu migrací a jsou patrné například z GitLab Issue č. 48[13], nebo

[13] `https://gitlab.fit.cvut.cz/sos/sos/-/issues/48`

z komentářů u GitLab Issue č. 47[14]. Další byly zjištěny právě až při manuální kontrole dat po provedení migrací.

#### 5.2.2.3 Manuální testování v testovacím prostředí

Později již bylo k dispozici testovací prostředí a bylo tedy možné jej využívat také k odhalování chyb, které se dostaly až do nasazených verzí nebo tam byly skryté dlouhodobě. Testovací prostředí využívali také členové Týmu při analýze prováděné pro splnění požadavků BI-SP1 a BI-SWI, při čemž objevili celkem 12 chyb, z nichž 6 následně i vyřešili a až na jednu výjimku se jednalo o méně závažné problémy, většinou související se špatným zobrazením uživatelského rozhraní. Lze je dohledat v GitLab Issue Trackeru[15]. Kromě těchto problémů se ale Týmu podařilo provést v SOS všechny analyzované scénáře[16], což lze považovat za úspěch a za důkaz funkčnosti systému.

### 5.2.3 Akceptační testování

Jak jsem již uvedl na začátku práce v sekci 1.2, nasazení SOS do výuky předmětu BI-SP1 v letním semestru 2022/23 podmínil jeho garant provedením akceptačního testování. Po implementaci funkcionalit kritických pro tento předmět, tak jak je uvádím v sekci 3.1.1, jsem tedy navíc pro garanta zkompletoval obrázkový návod k průchodu systémem, který demonstruje nově implementované funkcionality, a umožnil mu akceptační testování samostatně provést. Návod je k dispozici v multimediální příloze v souboru `navod-sp1.pdf`.

#### 5.2.3.1 Realizace testování

Testování se odehrálo v lednu 2023, tedy ještě v době, kdy existovalo pouze produkční prostředí ve starém CloudFIT a v něm nasazená verze 0.6.2. Pro testování jsem tedy musel místo testovacího prostředí použít alternativní metodu v podobě nástroje Ngrok[17]. S jeho využitím jsem vytvořil HTTP tunel a vývojovou verzi SOS spustil lokálně na svém počítači. Přes odkaz, který Ngrok vygeneroval, se pak dalo k systému přistupovat odkudkoliv.

Stejný nástroj jsem využíval i při uživatelském testování první verze SOS v rámci své bakalářské práce. Z pohledu uživatele se při této metodě systém chová stejně, jako by byl provozován běžným způsobem na serveru, snad kromě delší doby odezvy. Nevýhodou byla pouze z bezpečnostních důvodů nefunkční integrace s fakultním OAAS a tudíž i s KOSapi, což ale bylo pro účely tohoto testování irelevantní.

#### 5.2.3.2 Výsledky testování

Během testování garant narazil pouze na několik problémů nízké závažnosti, převážně souvisejících s nepřehledností uživatelského rozhraní. Jednalo se o tyto položky:

- chybějící chybová hláška při nevyplnění popisu zadání,
- nejasný význam pojmů „tým“ a „projekt“,
- nevěděl, jak přiřadit vlastní role členům týmu.

Toto testování se týkalo verze 1.0.0. Ta akceptačním testováním prošla a mohl jsem ji tedy nasadit do provozu, jak popisuji v sekci 3.1.2, aby mohla být následně použita ve výuce.

[14] `https://gitlab.fit.cvut.cz/sos/sos/-/issues/47`
[15] Jedná se o úkoly vytvořené členy Týmu od poloviny března 2023 s labelem „BUG“.
[16] tak jak jsou uvedeny na Redmine Wiki a v uživatelské dokumentaci na PagesFIT
[17] `https://ngrok.com/product/secure-tunnels`

## 5.3 Provoz systému na FIT ČVUT

Od nasazení verze 1.0.0 do produkčního prostředí v lednu 2023 je funkčnost systému také neustále ověřována reálným provozem. Od začátku je SOS využíván především v předmětu BI-SP1, jak bylo zamýšleno. Navíc se ale garant předmětu rozhodl jej využít i v předmětu BI-SWI, jehož průběh je shodný s předmětem NI-NUR, jak jsem již popsal v analýze v sekci 1.2.4. SOS je také využíván variantami těchto předmětů pro kombinované studium a studium v anglickém jazyce[18]. SOS tak má v letním semestru 2022/23 přes 250 uživatelů z řad studentů a projekty v něm vypsalo 9 vyučujících. Celkem bylo tento semestr vytvořeno 47 projektů. Většina týmů má již odevzdané i ohodnocené řešení prvního kontrolního bodu, řada týmů již odevzdala i řešení dalších kontrolních bodů.

Do produkčního prostředí jsem průběžně nasazoval nové verze, tak jak uvádí Changelog[19] a jak je popsáno v kapitole Realizace. Provoz byl po celou dobu monitorován s využitím služby Sentry. V produkčním prostředí nedošlo za celou dobu k žádným pádům systému nebo jiným chybám, které by byly v Sentry zaznamenány. Výjimkou bylo pouze několik hlášení souvisejících s výpadkem KOSapi. Zde je zřetelný přínos testovacího prostředí, které je rovněž monitorováno Sentry a několik problémů zde bylo zachyceno dříve, než by byla vadná verze nasazena do produkčního prostředí.

Během provozu se projevilo několik nedostatků, které byly zaznamenány v GitLab Issue Trackeru[20] a řada z nich byla během semestru opravena, buďto mnou nebo členy Týmu. Vyřešené problémy jsou uvedeny v Changelogu. Aktuálně[21] také Tým provádí další kvantitativní výzkum prostřednictvím dotazníku, který byl rozeslán studentům předmětu BI-SP1. Výsledky do konce semestru Tým vyhodnotí a zohlední při budoucím vývoji SOS. Účelem dotazníku je především zhodnotit přívětivost uživatelského rozhraní, identifikovat nepřehledná místa a sesbírat návrhy nových funkcionalit, které by uživatelům usnadnily práci se systémem. Rozdíl oproti prvnímu dotazníku rozeslaném na začátku semestru, o kterém píšu v sekci 4.5.2, jsou jednak konkrétněji položené otázky, ale i skutečnost, že studenti BI-SP1 již mají zkušenosti i s prací se systémem v průběhu semestru, nikoli pouze s tvorbou týmů.

## 5.4 Známé nedostatky systému

Z provozu systému během semestru vyplynulo několik nedostatků, které by bylo vhodné vyřešit ideálně před začátkem zimního semestru 2023/24. Jejich řešení už z časových důvodů nebude předmětem této práce a zabývat se jimi bude Max Hejda nebo členové Týmu. Většinou se nejedná přímo o chyby, jako spíš o vhodná rozšíření funkcionalit. Všechny položky jsou uvedeny v GitLab Issue Trackeru projektu, ty nejdůležitější uvádím zde:

- **Popis:** Chybí mechanismus pro zkopírování konfigurace předmětu do jiného předmětu. Funkcionalita by byla užitečná pro kopírování konfigurace a případně i vyučujícími vytvořených zadání z předchozích semestrů, případně mezi příbuznými předměty, jako jsou například BI-SP1 a BI-SP2.
  **Kategorie:** Nová funkcionalita
  **GitLab Issue:** #132
- **Popis:** Chybí mechanismus pro přenos projektu z jednoho předmětu do druhého. Funkcionalita by byla velmi užitečná pro předmět BI-SP2, kde studenti často navazují na projekt rozpracovaný v rámci BI-SP1.
  **Kategorie:** Nová funkcionalita
  **GitLab Issue:** #137

[18] BIK-SWI, BIK-SP1, BIE-SWI a BIE-SP1
[19] `https://sos.fit.cvut.cz/changelog/`
[20] `https://gitlab.fit.cvut.cz/sos/sos/-/issues/`
[21] dotazník byl rozeslán 21. dubna 2023

- **Popis:** Chybí mechanismus pro manuální přiřazení studenta k předmětu. V případě, že z nějakého důvodu není student importován z datového zdroje (KOS, Bakaláři), je sice možné jej manuálně přidat z administrátorského rozhraní, nicméně je tak potřeba zásah administrátora a navíc bude student opět odstraněn, pokud správce předmětu provede aktualizaci předmětu z datového zdroje dříve, než se student přidá k nějakému projektu.
  **Kategorie:** Nová funkcionalita
  **GitLab Issue:** #113, #131

- **Popis:** Není funkční automatická tvorba týmů. Jedná se o funkcionalitu, kterou realizoval v rámci své práce Max Hejda a která je využívána v prostředí GJGJ, na FIT ČVUT zatím využita nebyla. Funkcionalita není momentálně dostupná, jelikož je nutné její implementaci aktualizovat vzhledem ke změnám datového modelu, které jsem v rámci této práce provedl.
  **Kategorie:** Chyba
  **GitLab Issue:** #70

- **Popis:** Aktualizace předmětu z datového zdroje nedokáže zpracovat nově přidanou paralelku. Aktuálně jsou aktualizovány pouze seznamy studentů paralelek, které byly do SOS importovány při vytvoření předmětu. Situaci lze řešit manuálním přidáním paralelky pomocí administrátorského rozhraní a následným vyvoláním aktualizace, která do paralelky již správně přiřadí studenty.
  **Kategorie:** Rozšíření existující funkcionality
  **GitLab Issue:** #95

- **Popis:** V případě, že některé předměty mají společnou semestrální práci, jako je tomu v případě BI-SP1 a BI-SWI, je potřeba tuto skutečnost v SOS nějakým způsobem zaznamenat a v jednom z předmětů tyto studenty vůbec nezobrazovat. Aktuálně jsou jednak znehodnoceny statistiky v předmětu BI-SWI, jelikož přibližně tři čtvrtiny jeho studentů realizují semestrální práci v BI-SP1 a v BI-SWI jsou tedy nesprávně uvedeni jako „bez týmu“. Situace je zároveň matoucí i pro studenty BI-SP1, jelikož je systém vybízí k připojení se k projektu BI-SWI, i když mají projekt v BI-SP1, který ho plně nahrazuje.
  **Kategorie:** Nová funkcionalita
  **GitLab Issue:** #118

Žádná z uvedených položek nebrání provozu systému v plném rozsahu. Ve všech případech existuje tzv. *workaround*, tedy alternativní způsob, jak dosáhnout stejného výsledku.

## 5.5 Shrnutí

V této kapitole jsem popsal veškeré procesy, kterými byla během vývoje a během provozu systému v prostředí FIT ČVUT ověřována správná funkce všech existujících i nově přidaných funkcionalit a rozšíření SOS. Systém má funkční CI/CD pipeline, která ověřuje funkčnost jeho jednotlivých částí pomocí unit testů s pokrytím kódu 95 % a také kontroluje dodržování *code style*. Veškeré prováděné změny jsou pro zajištění odpovídající kvality implementace podrobovány *code review* a v případě potřeby i manuálnímu testování. Vývojové verze jsou automaticky nasazovány do testovacího prostředí, kde je manuálně ověřována jejich spolehlivá funkčnost před nasazením do produkčního prostředí. Testovací i produkční prostředí je po celou dobu monitorováno službou Sentry, díky čemuž mohou být vážné problémy okamžitě identifikovány a díky množství informací získaných z těchto hlášení o chybách také efektivně analyzovány a následně vyřešeny.

V testovacím prostředí byly také veškeré funkcionality ověřeny členy Týmu, kteří prováděli komplexní analýzu celého systému. Kromě ověření funkčnosti Tým také testoval použitelnost uživatelského rozhraní skrze kvantitativní i kvalitativní výzkumy se skutečnými uživateli systému, z čehož vyplynuly některé problémy, které byly odpovídajícím způsobem zaznamenány a již byly nebo co nejdříve budou adresovány.

Nejnovější verze SOS je po celý letní semestr 2022/23 používána více než 250 uživateli v prostředí FIT ČVUT v předmětech BI-SP1, BI-SWI a jejich variantách pro kombinované studium a studium v anglickém jazyce. Z existujících dat v systému je zřejmé, že uživatelé systém zamýšleným způsobem používají a nic nenasvědčuje tomu, že by se vyskytovaly závažnější problémy, ať už po stránce funkčnosti nebo použitelnosti systému. Aktuálně je také Týmem prováděn dotazníkový průzkum s cílem zajistit další zpětnou vazbu od uživatelů z řad studentů.

V závěru kapitoly jsem uvedl přehled aktuálně známých nedostatků systému, které budou co nejdříve řešeny ostatními vývojáři. Nejedná se však o závažné problémy, ve většině jde pouze o návrhy nových funkcionalit, které by uživatelům usnadnily práci se systémem. Na základě všech výše uvedených skutečností mohu prohlásit, že bylo ověřeno splnění všech funkčních i nefunkčních požadavků uvedených v první kapitole v sekci 1.2.5. Systém je v zamýšleném rozsahu plně funkční a použitelný pro výuku předmětů BI-SP1, BI-SP2, BI-SWI a NI-NUR.

# Závěr

Jedním z cílů této diplomové práce bylo rozšířit existující SOS – Studentský odevzdávací systém tak, aby plně vyhovoval požadavkům předmětů BI-SP1 a BI-SP2 vyučovaných na FIT ČVUT. To zahrnovalo analýzu současného stavu systému, návrh řešení, jeho implementaci, nasazení do provozu a nakonec ověření splnění identifikovaných požadavků. Dalším cílem bylo zajištění dlouhodobé udržitelnosti projektu, především prostřednictvím zapojení studentů předmětu BI-SP1.

Nejprve jsem popsal existující systém SOS a jeho vývoj v čase – první verzi vzešlou z mé bakalářské práce, úpravy související s jeho nasazením v předmětu NI-NUR a konečně také rozšíření pro využití na gymnáziích, které realizoval Max Hejda v rámci jeho bakalářské práce. Dále jsem se věnoval analýze požadavků, a to nejen předmětů BI-SP1 a BI-SP2, ale také s nimi souvisejícího předmětu BI-SWI. Zároveň jsem znovu analyzoval požadavky předmětu NI-NUR, který již SOS dva roky využívá a mohl jsem tedy využít i zpětnou vazbu získanou z reálného provozu systému. Vytvořil jsem seznam šestnácti funkčních a pěti nefunkčních požadavků a následně zhodnotil jejich splnění existující implementací SOS. Řada těchto požadavků byla již existující implementací splněna, identifikoval jsem ale některé nedostatky a chybějící funkcionality, související především se specifickými požadavky předmětů BI-SP1 a BI-SP2.

Na základě provedené analýzy jsem navrhl konkrétní řešení vedoucí ke splnění všech identifikovaných požadavků. Návrh zahrnoval úpravy datového modelu, aplikační logiky i uživatelského rozhraní. Navíc jsem také navrhl řešení pro export hodnocení studentů z SOS do systémů pro zaznamenávání studijních výsledků používaných v prostředí FIT ČVUT – KOS a FIT Klasifikace.

Implementaci jednotlivých rozšíření jsem provedl ve více fázích. Nejprve jsem implementoval položky kritické pro předmět BI-SP1, aby mohl být v letním semestru 2022/23 v tomto předmětu SOS nasazen. V průběhu semestru jsem pak postupně implementoval zbylé položky, včetně exportu hodnocení do KOS a FIT Klasifikace s využitím dostupných rozhraní. Mezi nejdůležitější nové funkcionality, které slouží zejména předmětům BI-SP1 a BI-SP2, patří například možnost konfigurace předmětu na úrovni jednotlivých týmů, možnost přiřadit do předmětu i vyučující, kteří nejsou importováni z datového zdroje, možnost udělení přístupu k projektu i dalším uživatelům kromě vyučujícího a členů týmu nebo funkcionalita přerozdělování bodů mezi členy týmů.

V letním semestru 2022/23 jsme také s vedoucím práce Jiřím Hunkou a kolegou Maxem Hejdou vedli tým studentů předmětu BI-SP1, které jsme do projektu SOS zapojili. Členové týmu se seznámili s projektem a s používanými technologiemi a nyní se aktivně podílí na vývoji. Zpracovali také komplexní analýzu systému, jejíž výstupy jsou velmi cennou dokumentací pro všechny budoucí vývojáře projektu. Tým také zpracoval uživatelskou dokumentaci popisující veškeré aktivity, které v systému jeho uživatelé realizují. Kromě vývoje tým také aktuálně pracuje na prototypu nového uživatelského rozhraní, které má být v budoucnu implementováno.

V neposlední řadě jsem se v rámci práce věnoval zajištění stability systému a zefektivnění

procesu nasazování nových verzí. Systém má nyní plně funkční CI/CD pipeline, která s využitím technologie Docker automaticky nasazuje nové verze do testovacího i produkčního prostředí. Před nasazením ověřuje funkčnost jednotlivých částí systému pomocí unit testů, které aktuálně pokrývají 95 % kódu. Testovací i produkční prostředí je monitorováno službou Sentry, díky čemuž mohou být chyby včas identifikovány a díky množství informací získaných z hlášení o chybách také efektivně analyzovány a řešeny.

Splnění jednotlivých požadavků bylo ověřeno jednak akceptačním testováním provedeným před nasazením systému do předmětu BI-SP1, ale také následně v rámci analýzy systému provedené studenty BI-SP1 a nakonec i reálným provozem SOS v letním semestru 2022/23. SOS v tomto semestru využívají předměty BI-SP1, BI-SWI a jejich varianty pro kombinované studium a studium v anglickém jazyce. Systém tak má v tomto semestru přes 250 uživatelů včetně 9 vyučujících, kteří v něm vypsali celkem 47 projektů. V následujícím semestru předpokládám využití SOS v předmětech NI-NUR a BI-SP2. Z provozu systému vyplynulo několik nedostatků, které jsou ale buďto málo závažnými chybami, nebo spíše návrhy nových funkcionalit. Všechny jsou zaznamenány v GitLab Issue Trackeru projektu a jsou průběžně řešeny dalšími vývojáři. Žádný z nich ale neovlivňuje funkčnost a použitelnost systému v zamýšleném rozsahu.

Výsledkem diplomové práce je plně funkční SOS rozšířený tak, aby vyhovoval potřebám předmětů BI-SP1, BI-SP2, BI-SWI a NI-NUR. Projekt je díky CI/CD, automatickým testům, existující dokumentaci a nastavené infrastruktuře možné efektivně a bezpečně provozovat, vyvíjet a dále rozšiřovat. Moje aktivita v projektu pravděpodobně skončí s dokončením mého magisterského studia na FIT ČVUT, projekt ale budou dále rozvíjet členové SP týmu v rámci předmětu BI-SP2 v následujícím semestru a také Max Hejda, který má zájem SOS dále rozvíjet pro potřeby gymnázia. Cíle práce tedy považuji za splněné.

# Příloha A

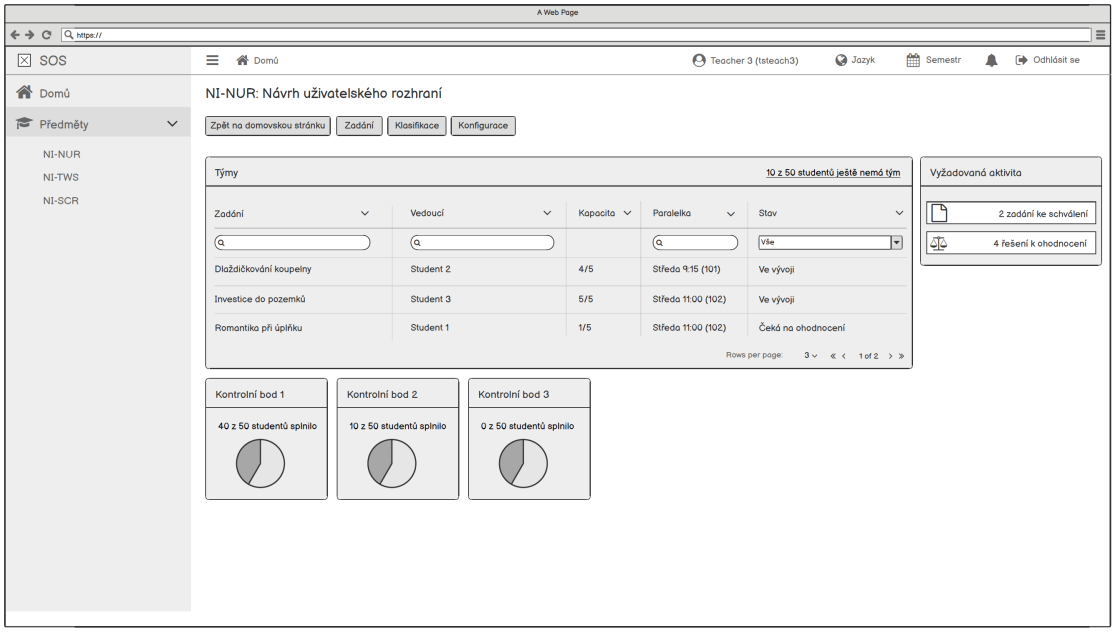
# Semestrální práce z NI-NUR

Tato příloha obsahuje vybrané části ze semestrální práce, kterou jsem vypracoval společně s kolegy v předmětu NI-NUR v zimním semestru 2021/22. V semestrální práci jsme se zabývali návrhem nového uživatelského rozhraní pro SOS, přičemž jsme vycházeli z původní verze, která vzešla z mé bakalářské práce.

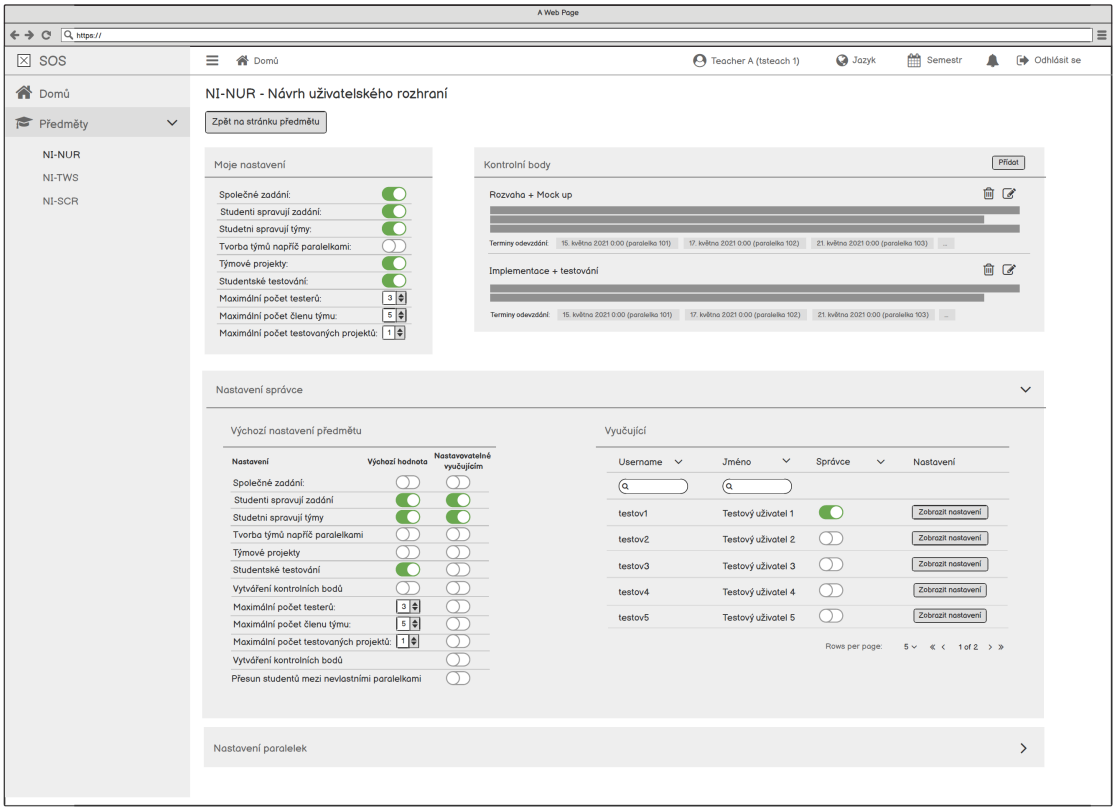
## A.1 Lo-Fi prototyp

Na obrázcích A.1–A.5 jsou vybrané části Lo-Fi prototypu vytvořeného v nástroji Balsamiq. Prototyp pokrývá učitelskou část systému, tedy domovskou stránku, stránku předmětu, přehled klasifikace, konfiguraci předmětu a ohodnocování řešení. Celý prototyp je dostupný v multimediální příloze ve složce `nur-semestralni-prace/` ve formátu PDF (soubor `wireframes.pdf`) nebo ve formátu nástroje Balsamiq (soubor `wireframes.bpmr`).

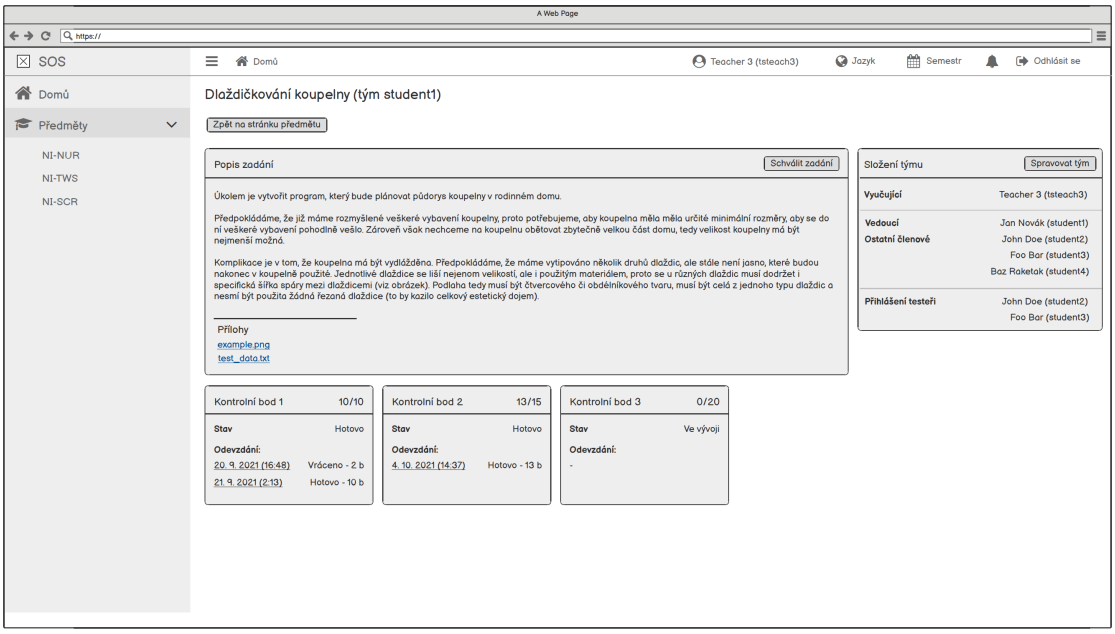
**Obrázek A.1** Wireframe obrazovky detailu předmětu



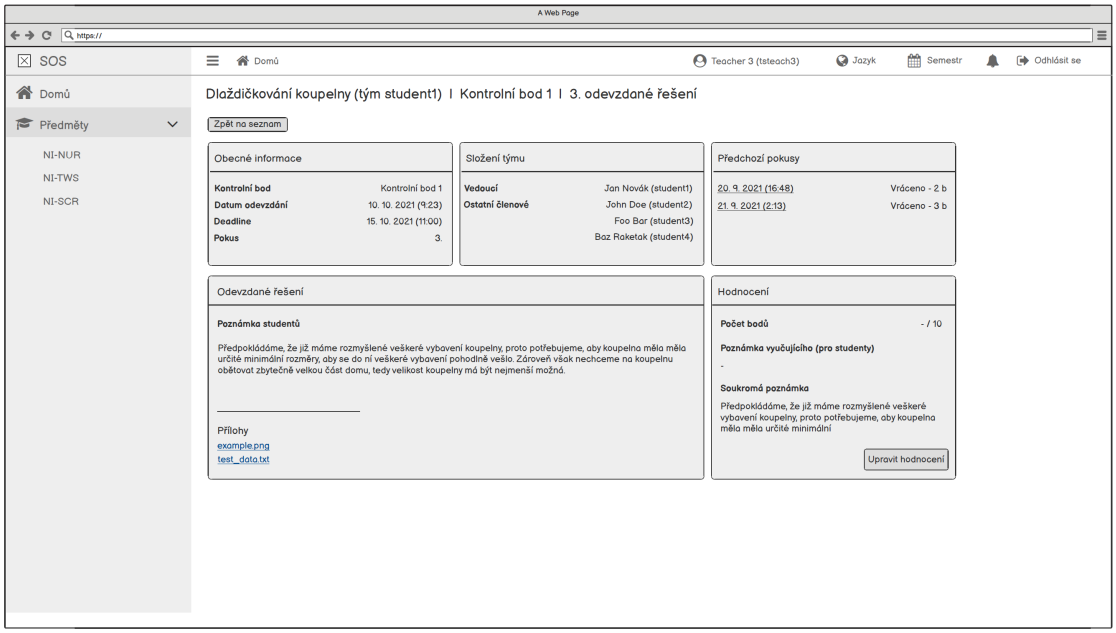
**Obrázek A.2** Wireframe obrazovky nastavení předmětu



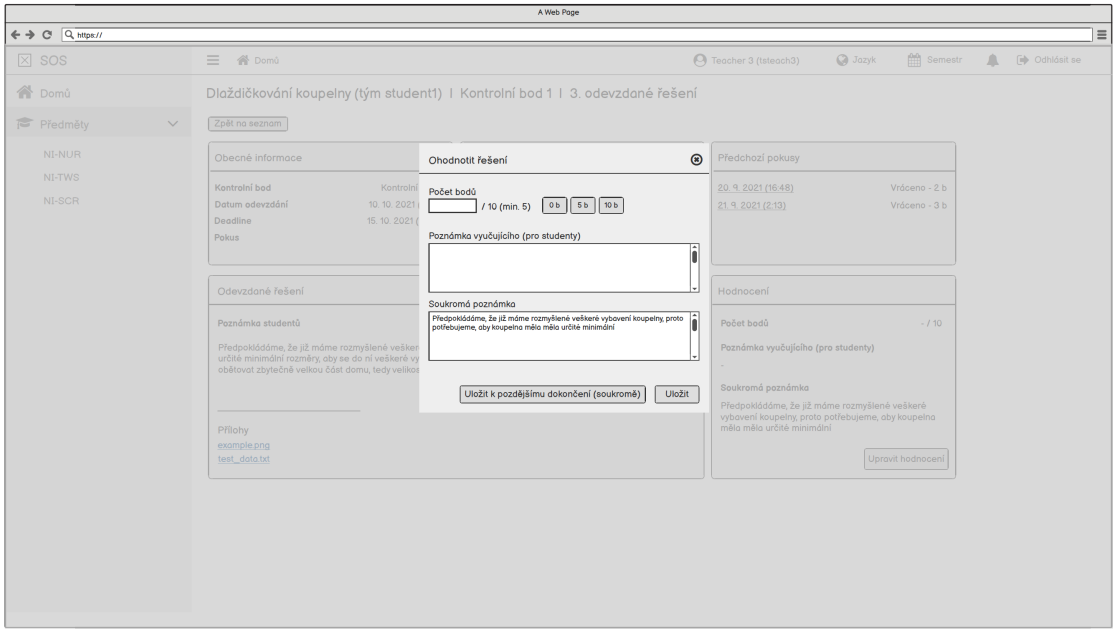

**Obrázek A.3** Wireframe obrazovky detailu týmu



**Obrázek A.4** Wireframe obrazovky detailu odevzdaného řešení




**Obrázek A.5** Wireframe dialogu ohodnocování řešení

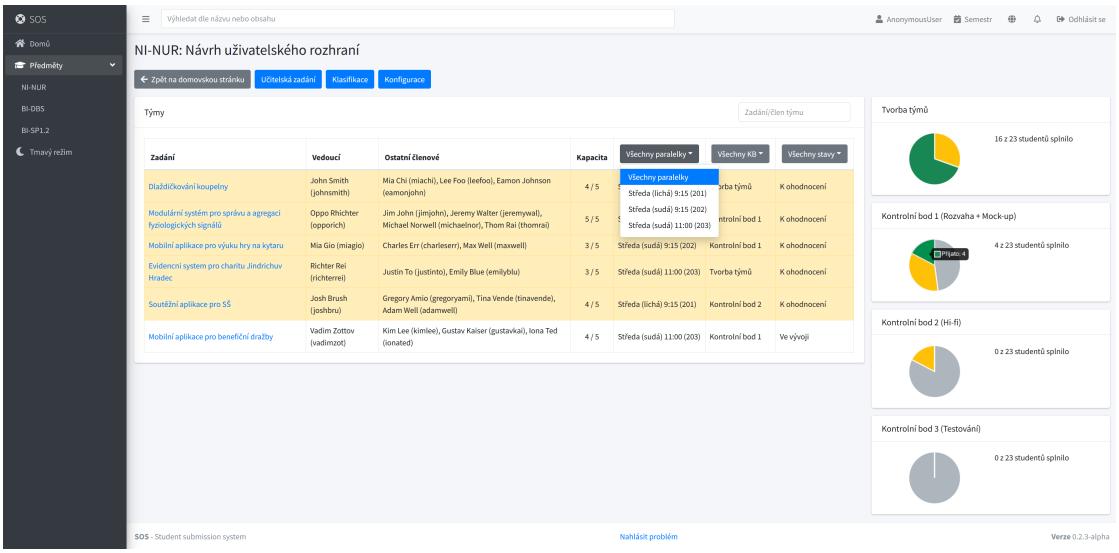


## A.2 Hi-Fi prototyp

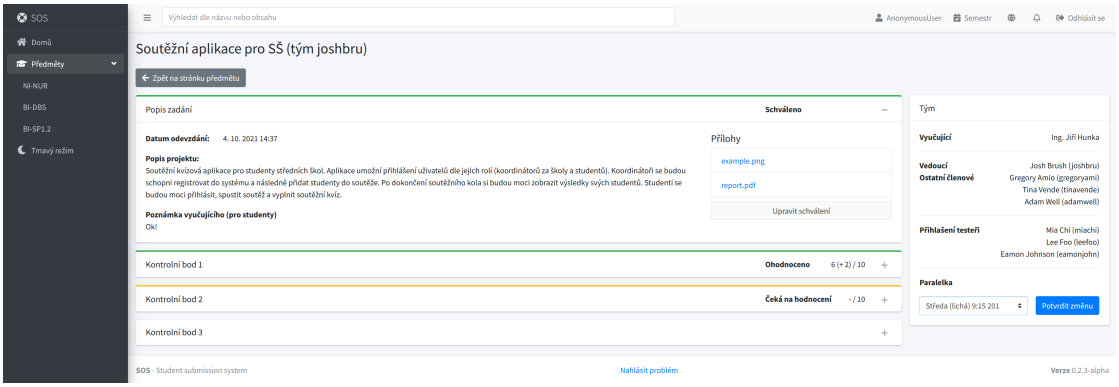
Na obrázcích A.6–A.9 jsou vybrané části Hi-Fi prototypu vytvořeného přímo v původní verzi SOS. Prototyp pokrývá pouze obrazovky detailu předmětu a týmu. Celý prototyp je dostupný v repozitáři SOS v následujících Merge Requestech:

- První verze prototypu – !130
- Upravená verze po usability testování – !136
- Upravená verze spustitelná v Dockeru – !254

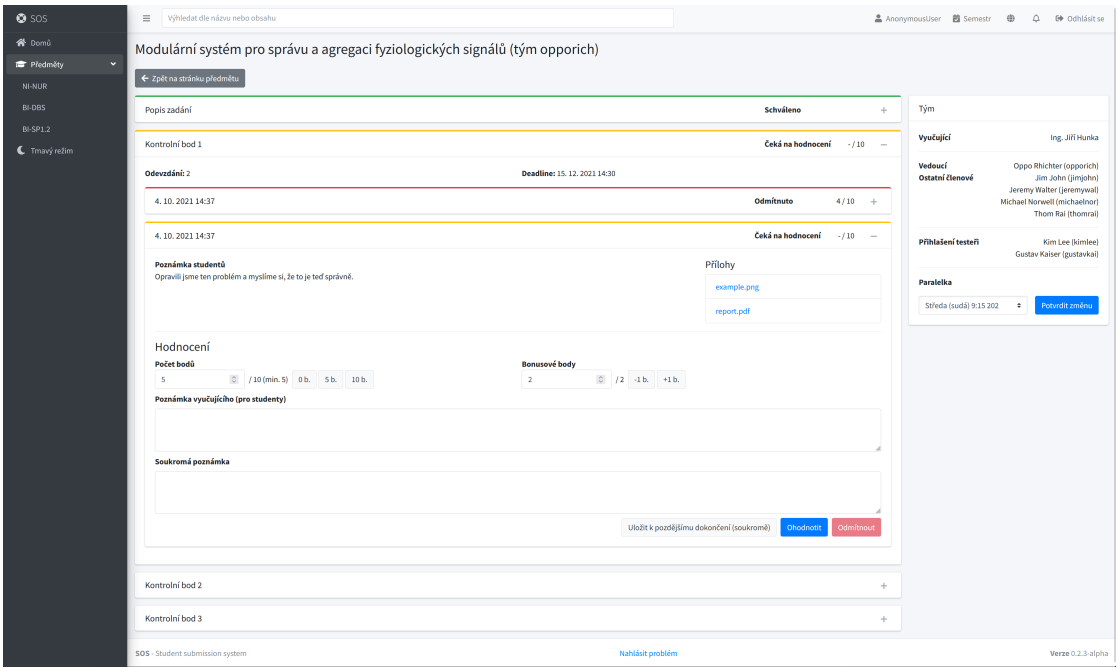
**Obrázek A.6** Hi-Fi prototyp obrazovky detailu předmětu



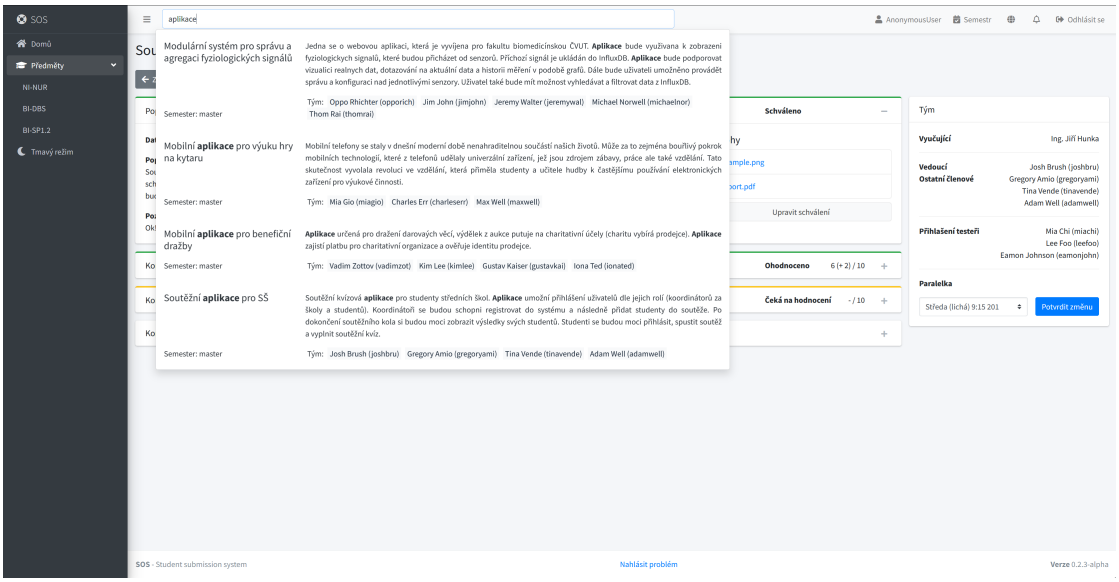

**Obrázek A.7** Hi-Fi prototyp obrazovky detailu týmu – zadání



**Obrázek A.8** Hi-Fi prototyp obrazovky detailu týmu – hodnocení řešení




**Obrázek A.9** Hi-Fi prototyp vyhledávače



## A.3 Další dokumenty

Součástí semestrální práce byly i další dokumenty, jako například analýza případů užití a scénářů, Person, uživatelských cílů, business požadavků a task groups, Nielsenova heuristická analýza, záznam z usability testování a přehled nalezených problémů. Tyto dokumenty se nacházejí v multimediální příloze ve složce `nur-semestralni-prace/`.

# Příloha B

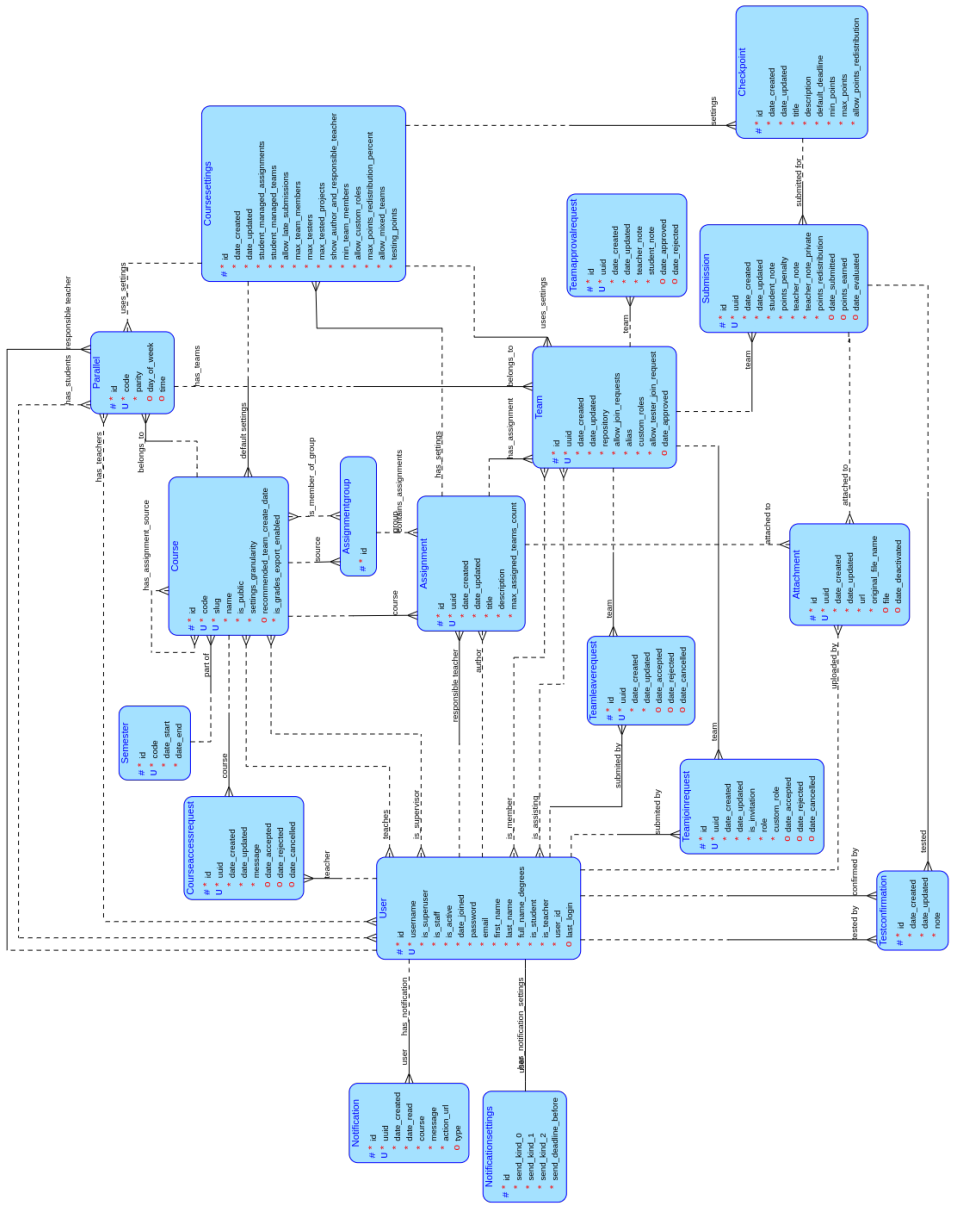
# Výsledky práce studentů BI-SP1

Tato příloha obsahuje vybrané výstupy práce Týmu v rámci letního semestru 2022/23. Veškeré tyto i další výstupy práce Týmu jsou dostupné v Redmine Wiki projektu, která se nachází na adrese `https://dbs.fit.cvut.cz/redmine/projects/sos-fit-cvut-cz/wiki`.

## B.1 Analýza

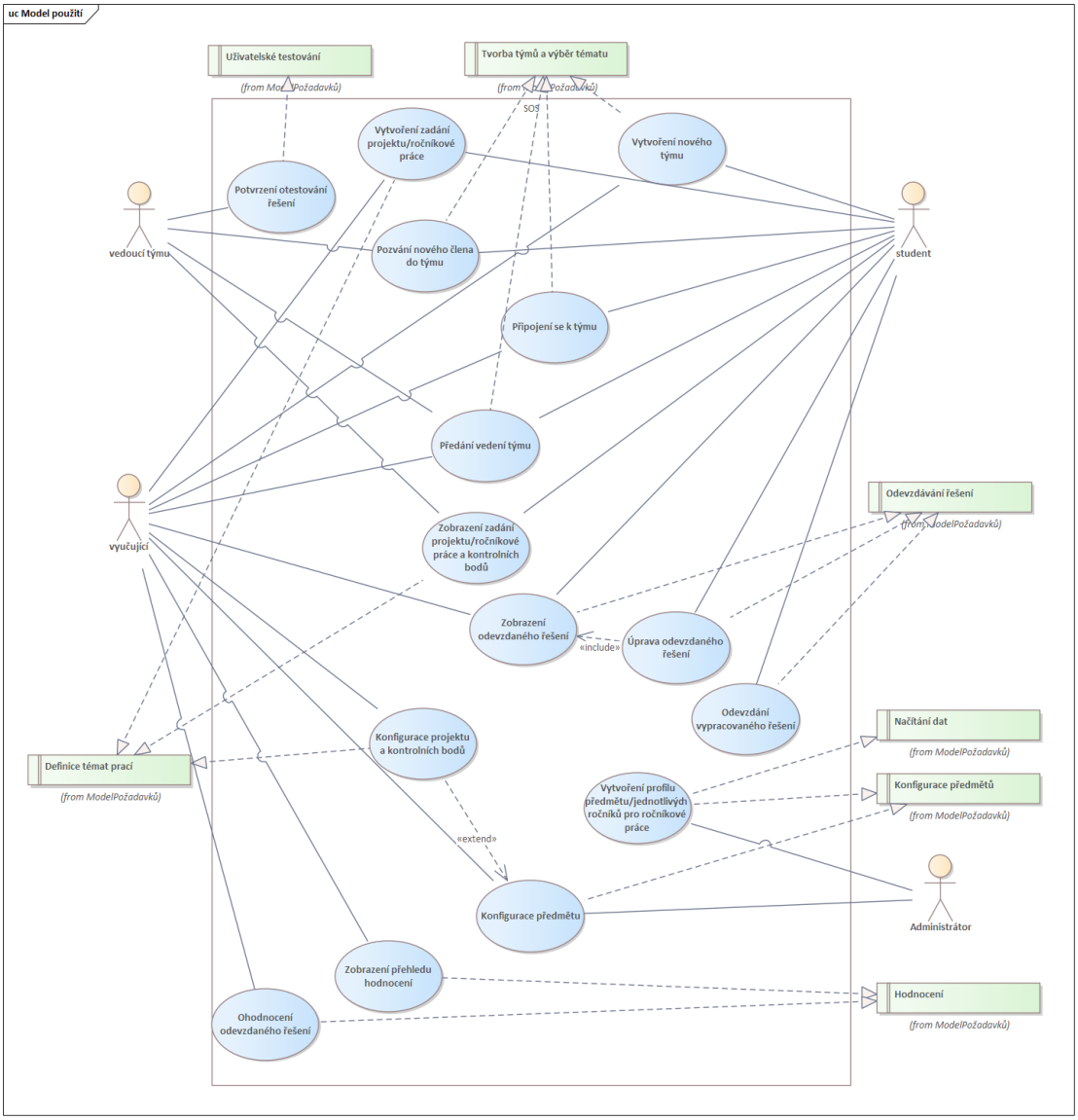
V této sekci se nachází vybrané výstupy analýzy, kterou Tým provádí pro splnění požadavků semestrální práce v předmětech BI-SP1 a BI-SWI a na jejíž faktickou správnost jsem během semestru dohlížel. Na obrázcích B.1–B.5 ukazuji konceptuální model, diagram případů užití, některé diagramy aktivit a popisy vybraných business procesů. Diagramy na obrázcích jsou také k dispozici v multimediální příloze ve složce `prace-sp1/analyza/`. Veškeré výstupy analýzy jsou k dispozici v Redmine Wiki a jsou průběžně aktualizovány.

*Poznámka: Konceptuální model na obrázku B.1 obsahuje chybu – ve skutečnosti již neexistuje vazba mezi entitami TestConfirmation a Submission. V době psaní tohoto textu je k dispozici pouze tato verze konceptuálního modelu, chyba bude co nejdříve opravena a model v Redmine Wiki aktualizován.*

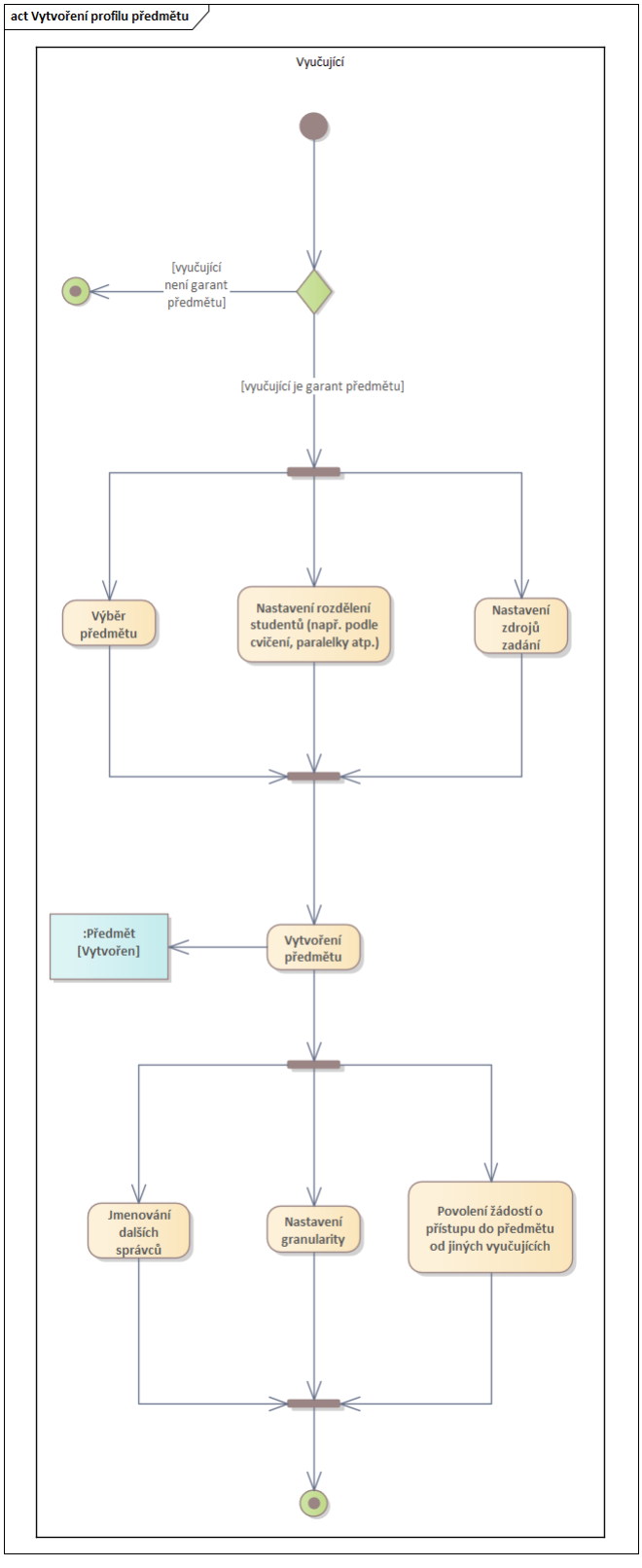



**Obrázek B.1** Konceptuální model

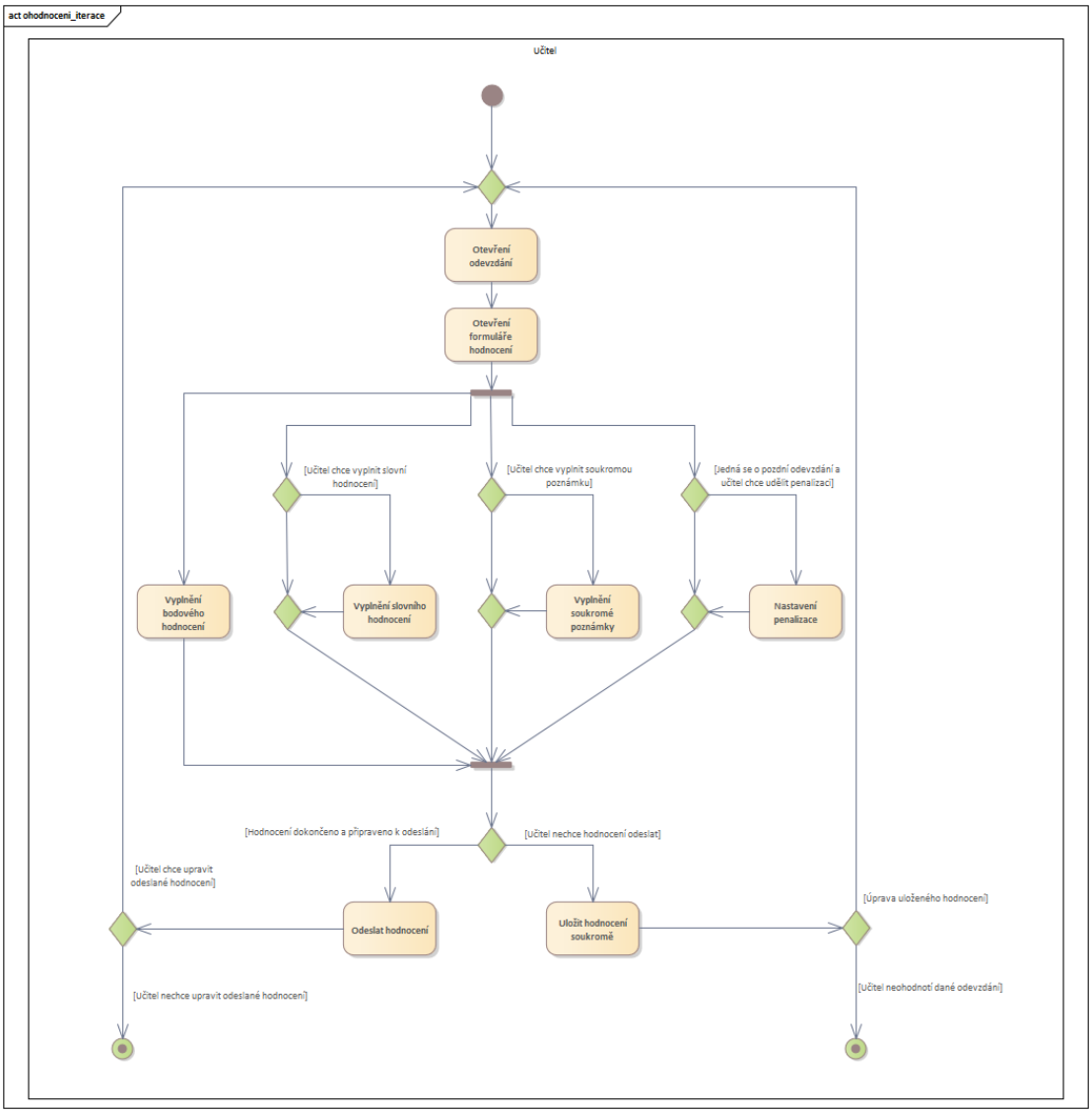
**Obrázek B.2** Diagram případů užití

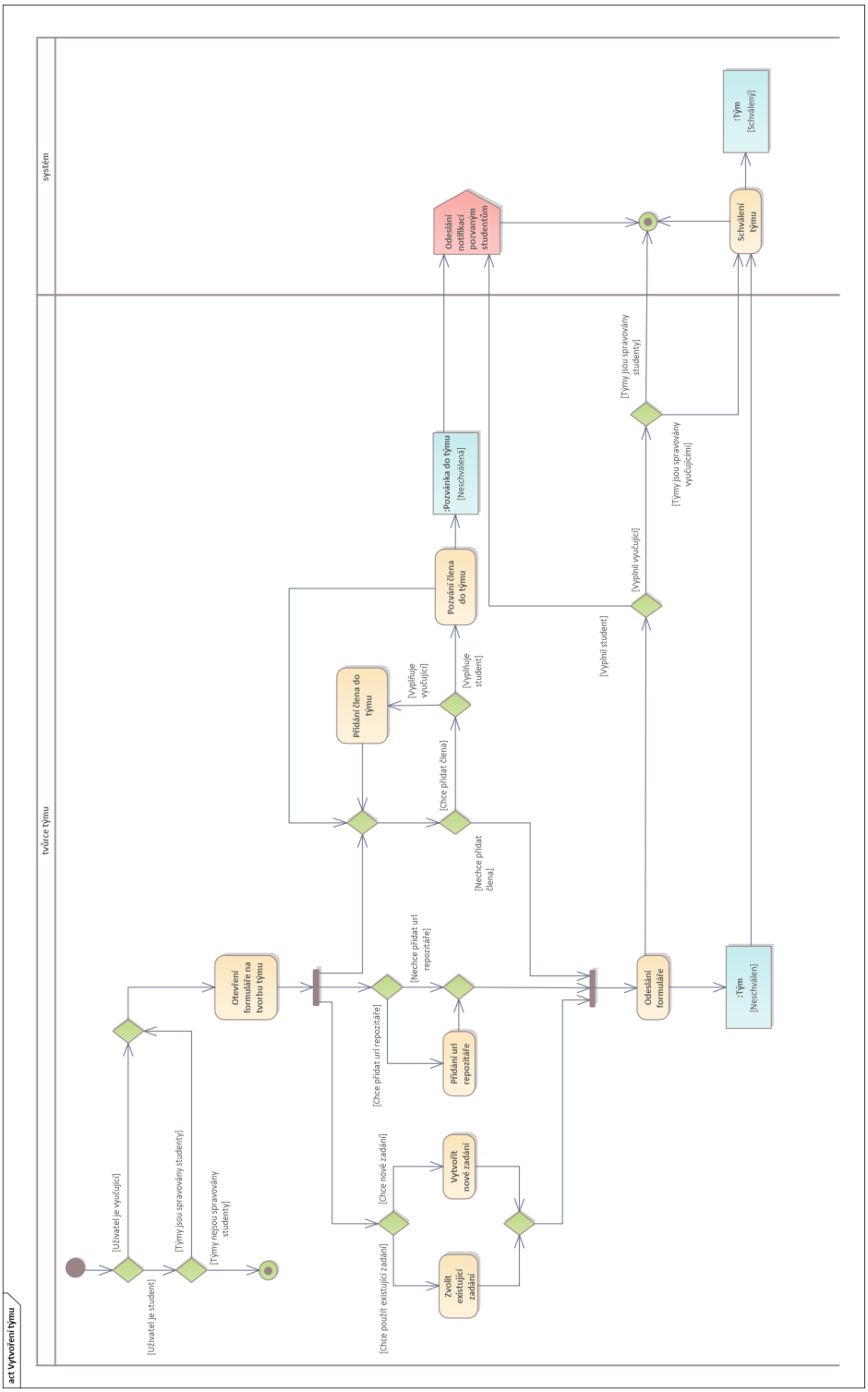


**Obrázek B.3** Activity diagram – vytvoření předmětu



**Obrázek B.4** Activity diagram – ohodnocení kontrolního bodu vyučujícím






**Obrázek B.5** Activity diagram – tvorba týmu

## B.2 Seznam nedostatků a návrhů na zlepšení

V první polovině semestru Tým vypracoval seznam nedostatků a návrhů na zlepšení, vzhledem k tehdy nasazené verzi SOS. Při jeho tvorbě vycházeli členové týmu z provedené analýzy a z výzkumů s uživateli SOS, které realizovali. Seznam se nachází na obrázku B.6 a také v multimediální příloze v souboru `navrhy.xlsx` ve složce `prace-sp1/`. Seznam obsahuje přes 50 položek, jednotlivé položky byly následně zaznamenány v GitLab Issue Trackeru v repozitáři SOS a část z nich již byla vyřešena v aktuální verzi SOS nebo v prototypu nového uživatelského rozhraní.

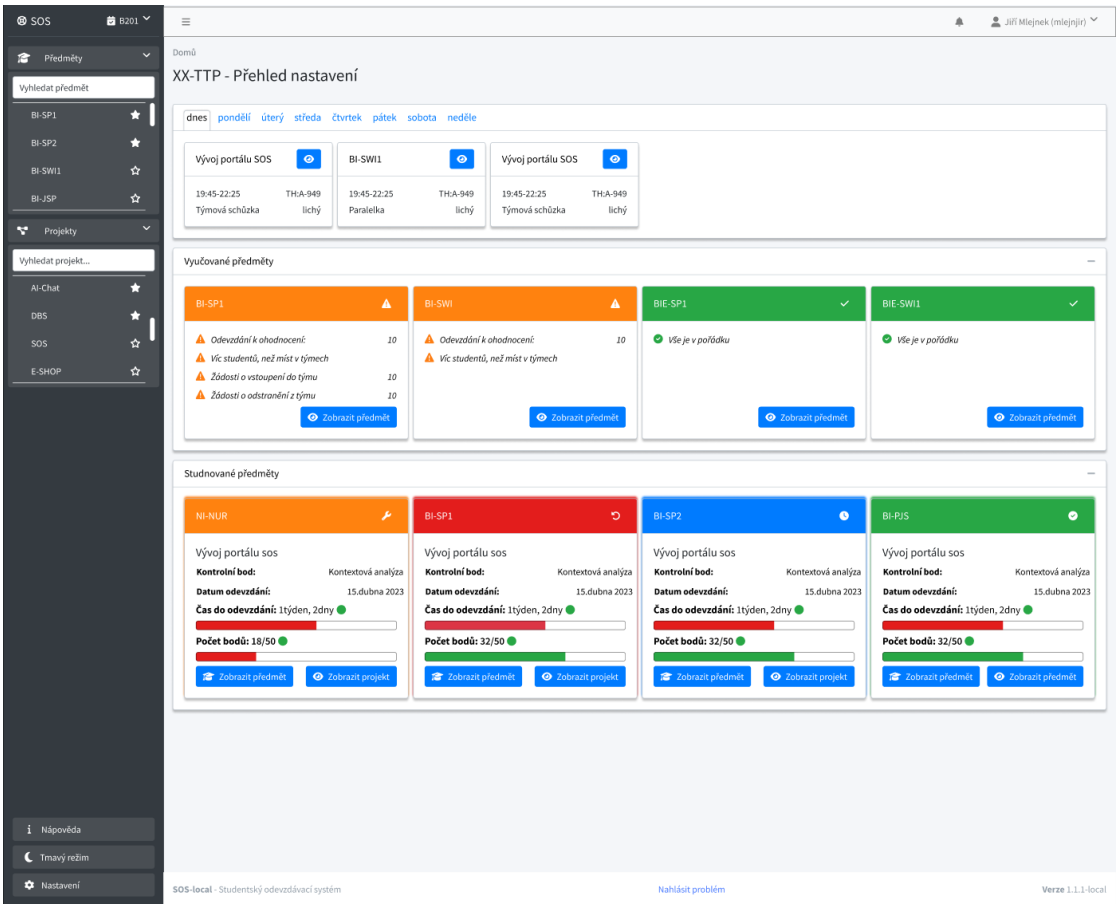
**Obrázek B.6** Seznam nedostatků a návrhů na zlepšení

| Stav | Název | Typ | Priorita | Náročnost | Popis | Zpracovává | Issue link |
|---|---|---|---|---|---|---|---|
| K diskuzi | SWI+SP1 - chybí info, že student nemusí řešit projekt v SWI | Fix | 5 | 3 | zároveň v grafu to zobrazuje, že většina z SWI nemá tým -> znehodnocuje graf, při vyhledávání stu | | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/118 |
| Navrženo | Vymyslet nové uspořádání v sidebaru | Fix | 5 | 3 | Domů a předměty znamenají to samé | Lucián Kučera | |
| Navrženo | Side menu nezobrazuje, kde uživatel je | Fix | 5 | 2 | | Lucián Kučera | |
| Schváleno | Přesunout breadcrumb vlevo | Fix | 5 | 2 | | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/124 |
| K diskuzi | Konzistence tlačítek | Fix | 5 | 2 | mělo by jít poznat, že se jedná o tlačítko (karty předmětů) | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/125 |
| Navrženo | Manuální aktualizování seznamu studentů | Feature | 5 | 2 | chybějící studenty přidat ručně tak, aby jej aktualizace nezrušila | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/131 |
| Navrženo | Klonování nastavení předmětů | Feature | 5 | 2 | při vytváření předmětu import nastavení z jiného předmětu i napříč semestry | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/132 |
| Ve vývoji | Termín odevzdání kontrolních bodů | Fix | 5 | 1 | Termín odevzdání je vidět až po rozkliknutí dané iterace | Marko Hujo | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/123 |
| K diskuzi | Míchá se terminologie (tým a projekt je to samé) | Fix | 5 | 1 | | | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/75 |
| Navrženo | Tvorba Git repozitářů | Feature | 4 | 4 | preference p. Mlejnka založení přes GIT a poté natáhnutí projektu do SOSu | Lucián Kučera | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/129 |
| Navrženo | NUR - management testerů | Feature | 4 | 3 | | | Potřeba další informace |
| Navrženo | Filtr na projekty, kde se ještě hledají lidi | Feature | 4 | 3 | | | Existující filtr - dostatečný počet lidí?? |
| Navrženo | Filtry v nabídce projektů | Feature | 4 | 3 | filtrování podle technologií, rolí nebo volných míst, systém tagů | Lucián Kučera | Probrat později |
| K diskuzi | Update/save akce neoznamují, že se něco stalo | Fix | 4 | 2 | Vytvořit nejspíše informační pop-upy | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/133 |
| K diskuzi | Cancel operace | Fix | 4 | 2 | odebrání učitele, smazání předmětu | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/135 |
| Schváleno | Zkontrolovat a opravit tlačítka zpět | Fix | 4 | 2 | Detail týmu->Upravit nastavení->Zpět->Jsem někde úplně jinde | Lucián Kučera | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/147 |
| K diskuzi | Nastavení nezohledňování paralelek | Fix | 4 | 2 | Nezohledňování paralelek lze nastavit pouze během vytváření předmětu | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/136 |
| Navrženo | Přenést tým z projektu SP1 do SP2 | Feature | 4 | 2 | | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/137 |
| Ve vývoji | U přihlášení k projeku zobrazit na jakou roli se kdo hlásí | Feature | 4 | 1 | | Marko Hujo | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/126 |
| Navrženo | Burger menu není asociované se sidebarem | Fix | 4 | 1 | Předem skrýt sidebar | Lucián Kučera | |
| Navrženo | Logo SOS není kliknutelné | Fix | 4 | 1 | | Marko Hujo | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/127 |
| Ve vývoji | Vedoucí nemůže podat žádost o opuštění | Fix | 4 | 1 | | Marko Hujo | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/128 |
| K diskuzi | Potvrzovací okna | Fix | 4 | 1 | | Lucián Kučera | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/130 |
| Navrženo | Nemizející popisek | Fix | 4 | 1 | SWI>Hover na tlačítko připojit se k týmu, kam se ale připojit nemohu>Info box, že se nemůžu připoj | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/143 |
| Navrženo | Personalizovaná homepage | Feature | 3 | 4 | info, do kdy se může měnit tým, SP1 nezobrazovat SWI, zobrazovat rovnou projekty v předmětu? (v | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/141 |
| K diskuzi | Nepřehledný systém odevzdávání | Fix | 3 | 4 | | | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/96 |
| Navrženo | Pomoc uživatelům s orientací | Fix | 3 | 3 | Po přihlášení nevidí žádný krok, co dál | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/140 |
| Navrženo | Nepřehledné odevzdání a hodnocení iterací | Fix | 3 | 3 | Co znamená potvrdit odevzdání a může to pouze vedoucí? Přerozdělování bodů neintuitivní - co zn | | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/96 |
| Navrženo | Upravit nastavení x spravovat tým | Fix | 3 | 3 | v nastavení je povolení rolí a ve správě týmu je až vytváření vlastních rolí | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/144 |
| Navrženo | Přecházení projekt zadání | Feature | 3 | 3 | | Jan Mrázek | Potřeba další informace |
| Navrženo | Informační boxy | Fix | 3 | 2 | Co obnáší "Připojit se k projektu?" nebo další operace | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/139 |
| Navrženo | Druhý graf je matoucí | Feature | 3 | 2 | není dobře poznat, co představuje (matoucí barvy a číslo uprostřed, změnit typ grafu?) | | Solved? |
| Navrženo | Formulář nastavování rolí | Fix | 3 | 2 | není poznat, zda se role uloží, na stránce se to kombinuje s tlačítkem uložit, které ale ukládá něco ji | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/133 |
| Navrženo | U přidání URL nelze poznat, jestli bylo uloženo | Fix | 3 | 1 | Zbarvit border boxu (indikace uložení), poté odbarvit zpět | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/138 |
| Navrženo | Možnost oblíbených projektů | Feature | 2 | 3 | souvisí s personalizovanou homepage | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/141 |
| Navrženo | Vyhledávání u přidávání asistenta + zpětná vazba | Feature | 2 | 3 | nyní se musí napsat přesně username, zpětná vazba při přidání | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/145 |
| Navrženo | ~~Popis chybných stavů~~ | Feature | 2 | 2 | error 500, když nejsem v kurzu | | Probrat později |
| Navrženo | Text v kartách na homepage se rozhazuje | Fix | 2 | 2 | | Jakub Sedláček | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/142 |
| Navrženo | Synchronizovat studenty, kteří mají předmět X a NE předmět Y | Feature | 2 | 2 | | | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/118 |
| Navrženo | Více grafů | Feature | 2 | 2 | | Tomáš Pavlůsek | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/71 (minimalne cast) |
| Navrženo | Nastavit vedoucího jako neaktivního | Feature | 2 | 2 | | | Feature? Nejdřív předat vedení, a poté nastavit jako inactive |
| Navrženo | Datepicker deadlinu u kontrolních bodů | Feature | 2 | 2 | ruční zadání času v tom formátu, nebo tam předvolit, který čas se má hodit -> kdyby tam bylo možn | | Probrat později |
| Navrženo | Možnost přidat do projektu další odkazy kromě gitlabu | Feature | 2 | 2 | např pro odkazy na RM a toggli a obecně custom odkazy do své sekce | | Solved? |
| Navrženo | Viditelný mail ostatních členů čekajících na žádost | Feature | 2 | 1 | | Marko Hujo | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/146 |
| Navrženo | Chybové hlášky | Fix | 2 | 1 | Některé moc odborné a nekomunikují řešení | Marko Hujo | https://gitlab.fit.cvut.cz/pavlutom/sos/-/issues/148 |
| Navrženo | Redukovat počet SQL query | Fix | 1 | 2 | | Lucián Kučera | |
| Navrženo | Popisek při přihlašování k týmu? | Feature | 1 | 2 | přidat komentář, který student může napsat při přihlašování k týmu | | Probrat později |
| Navrženo | Ikony stavu projektu | Fix | 1 | 1 | přesýpací hodiny znamenají, že se čeká na schválení nebo na odevzdání | | Co tím chtěl asi básník říct? |
| Navrženo | Maximální počet týmů u zadání lze nastavit na <0 | Fix | 1 | 1 | | | Nejspíš není třeba řešit |
| Navrženo | Emoji v editoru | Feature | 1 | 1 | | | Probrat později |
| Navrženo | Špatně datum přidání se do týmu | Fix | 1 | 1 | hover na členy týmu ukazuje nesprávné datum přidání | | Probrat později |
| Navrženo | | | | | | | |

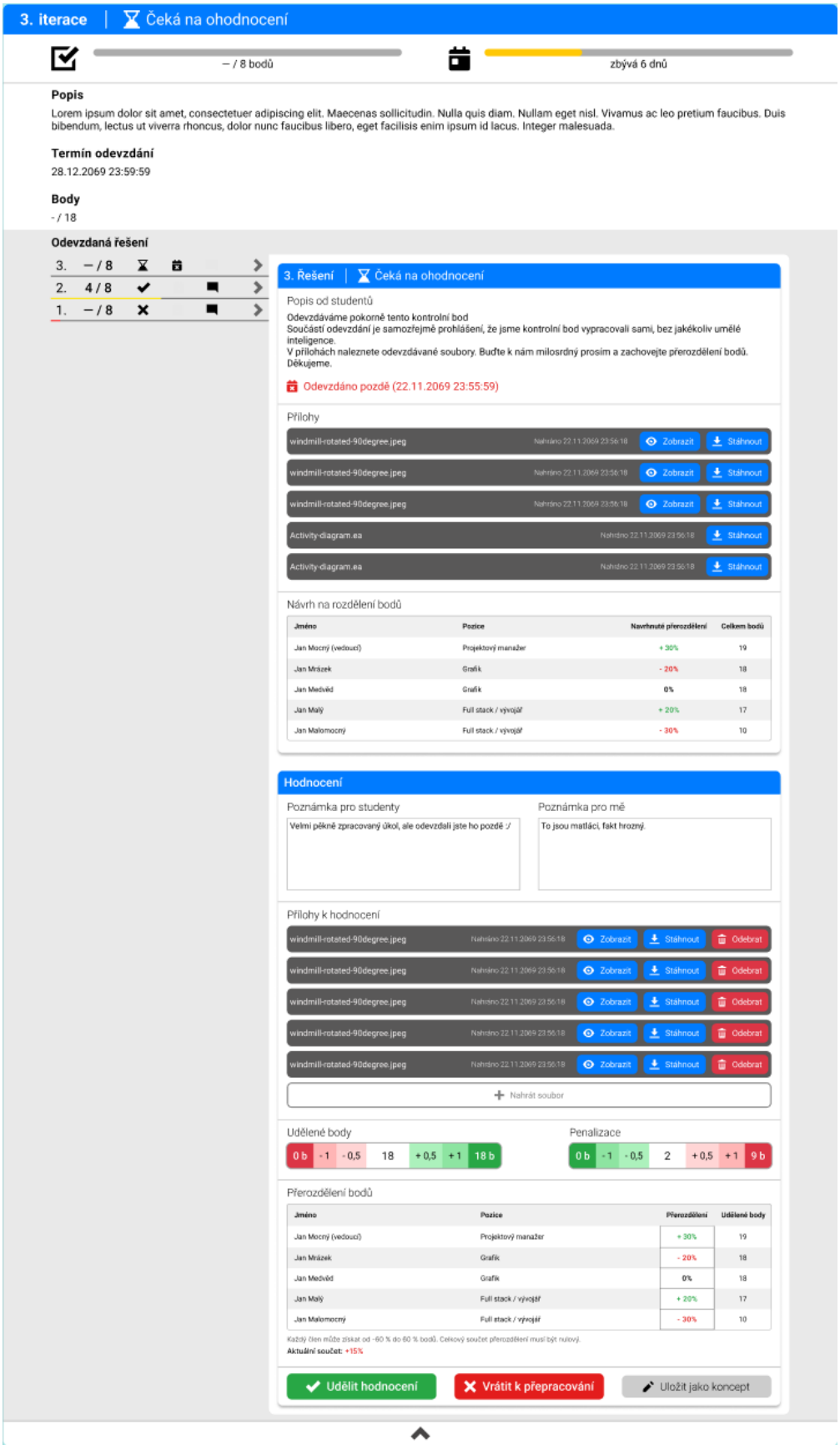
## B.3 Prototyp nového uživatelského rozhraní

Jedním z cílů Týmu je vytvořit prototyp nového uživatelského rozhraní SOS, které bude časem implementováno s využitím technologie Vue.js. K tvorbě prototypu členové týmu využívají nástroj Figma. Na obrázcích B.7–B.9 jsou vybrané části rozpracovaného prototypu. Další části prototypu se nacházejí v multimediální příloze ve složce `prace-sp1/figma/`. Prototyp je také z nástroje Figma průběžně exportován do Redmine Wiki.

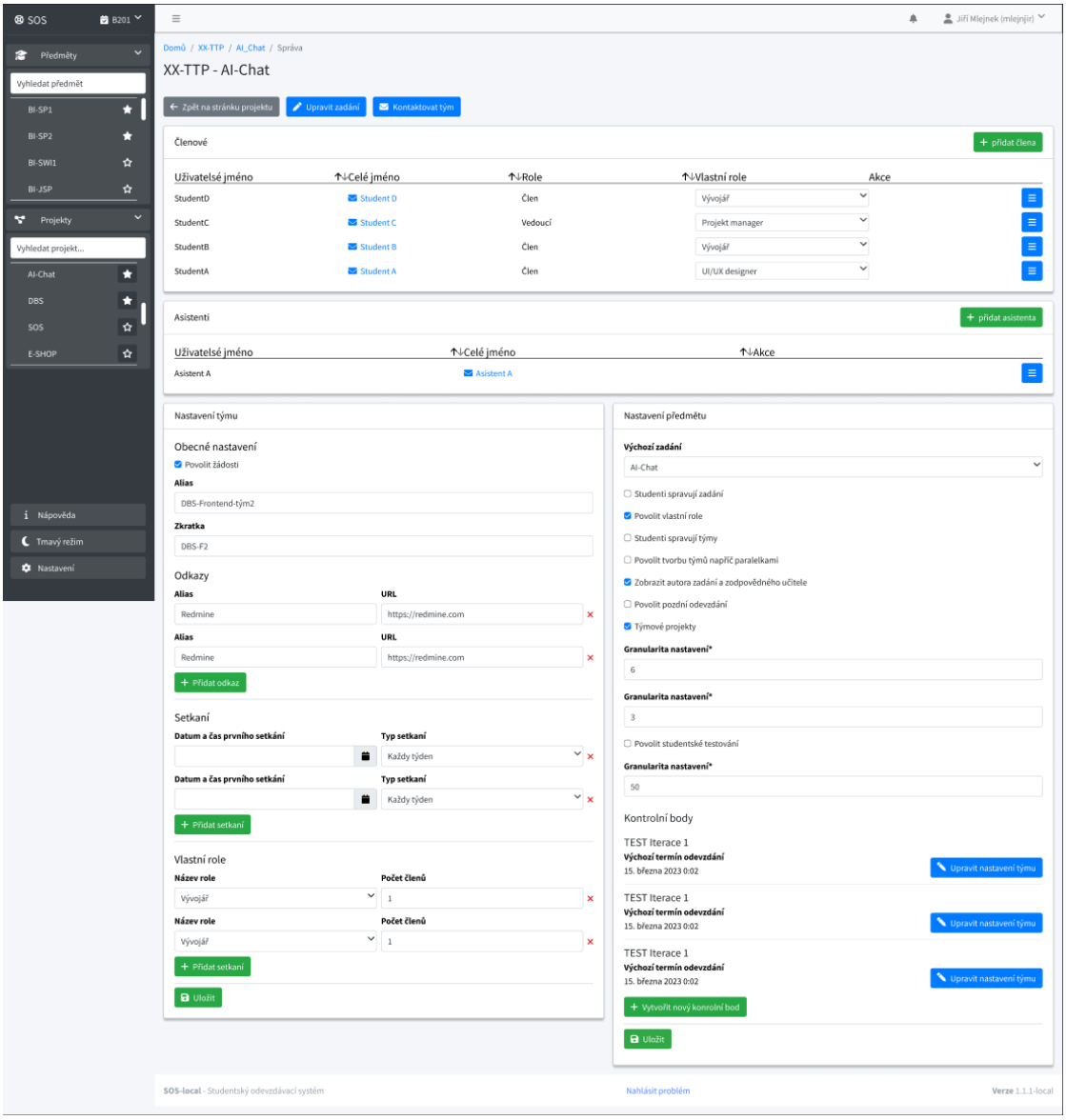
**Obrázek B.7** Prototyp domovské obrazovky



**Obrázek B.8** Prototyp karty kontrolního bodu z pohledu vyučujícího, který provádí hodnocení



**Obrázek B.9** Prototyp obrazovky konfigurace předmětu



# Bibliografie


1. PAVLŮSEK, Tomáš. *SOS - Studentský odevzdávací system*. Praha, 2021. Dostupné také z: http://hdl.handle.net/10467/95107. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
2. HEJDA, Max. *Systém pro správu ročníkových prací*. Praha, 2022. Dostupné také z: http://hdl.handle.net/10467/102038. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
3. WIEGERS, Karl; BEATTY, Joy. *Software Requirements (Developer Best Practices)*. Paperback. Microsoft Press, 2013. ISBN 978-0735679665.
4. *B212-BI-SP1,BIK-SP1 - Softwarový týmový projekt 1* [online]. 2021. [cit. 2023-04-12]. Dostupné z: https://moodle-vyuka.cvut.cz/course/view.php?id=6916.
5. *B221-BI-SP2.1,BIK-SP2.1 - Softwarový týmový projekt 2* [online]. 2022. [cit. 2023-04-12]. Dostupné z: https://moodle-vyuka.cvut.cz/course/view.php?id=7753.
6. HUNKA, Jiří. *Návrh uživatelského rozhraní: Hodnocení a semestrální práce / The semestral work classification* [online]. 2022. [cit. 2023-04-12]. Dostupné z: https://courses.fit.cvut.cz/MI-NUR/classification/index.html.
7. *B222-BI-SWI.21 - Softwarové inženýrství* [online]. 2023. [cit. 2023-04-12]. Dostupné z: https://moodle-vyuka.cvut.cz/course/view.php?id=8689.
8. EELES, Peter. *Capturing Architectural Requirements* [online]. 2005. [cit. 2023-04-12]. Dostupné z: https://web.archive.org/web/20201112020231/http://www.ibm.com/developerworks/rational/library/4706.html.
9. BRENNAN, Kevin (ed.). *A guide to the business analysis body of knowledge(R) (BABOK(R) guide)*. Toronto, ON, Canada: International Institute of Business Analysis, 2009. ISBN 978-1927584026.
10. KUHLMAN, Dave. *A Python Book: Beginning Python, Advanced Python, and Python Exercises* [online]. 2012. [cit. 2023]. Dostupné z: https://web.archive.org/web/20120623165941/http://cutter.rexx.com/~dkuhlman/python_book_01.html#part-1-beginning-python.
11. ROSSUM, Guido van. *Type Hints*. 2014-09. PEP, 484. Dostupné také z: https://peps.python.org/pep-0484/.
12. *About the Python Software Foundation* [online]. [B.r.]. [cit. 2023-04-14]. Dostupné z: https://www.python.org/psf/about/.
13. SALGADO, Pablo Galindo. *Python 3.11 Release Schedule*. 2021-07. PEP, 664. Dostupné také z: https://peps.python.org/pep-0664/.

14. HOLOVATY, Adrian; KAPLAN-MOSS, Jacob. *The definitive guide to Django: Web development done right*. 2nd ed. Berkeley: Apress, c2009. ISBN 978-1-4302-1936-1.
15. DJANGO SOFTWARE FOUNDATION. *Django's release process* [online]. 2023. Ver. 4.2 [cit. 2023-04-14]. Dostupné z: `https://docs.djangoproject.com/en/4.2/internals/release-process/`.
16. DJANGO SOFTWARE FOUNDATION. *Supported Versions* [online]. 2023. [cit. 2023-04-14]. Dostupné z: `https://www.djangoproject.com/download/#supported-versions`.
17. DJANGO SOFTWARE FOUNDATION. *django.contrib.postgres* [online]. 2023. Ver. 4.2 [cit. 2023-04-14]. Dostupné z: `https://docs.djangoproject.com/en/4.2/ref/contrib/postgres/`.
18. SOLEM, Ask. *Celery – Distributed Task Queue* [online]. 2021. Ver. 5.2.7 [cit. 2023-04-14]. Dostupné z: `https://docs.celeryq.dev/en/v5.2.7/`.
19. *nginx* [online]. [B.r.]. [cit. 2023-04-14]. Dostupné z: `https://nginx.org/en/`.
20. DJANGO SOFTWARE FOUNDATION. *How to use Django with Gunicorn* [online]. 2023. Ver. 4.2 [cit. 2023-04-14]. Dostupné z: `https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/gunicorn/`.
21. DOCKER, INC. *Use containers to Build, Share and Run your applications* [online]. [B.r.]. [cit. 2023-04-14]. Dostupné z: `https://www.docker.com/resources/what-container/`.
22. DOCKER, INC. *Docker Compose overview* [online]. [B.r.]. [cit. 2023-04-14]. Dostupné z: `https://docs.docker.com/compose/`.
23. VÝPOČETNÍ A INFORMAČNÍ CENTRUM ČVUT. *Organizace a správa výuky (KOS)* [online]. 2023. [cit. 2023-04-14]. Dostupné z: `https://ist.cvut.cz/nase-sluzby/kos/`.
24. JIRŮTKA, Jakub. *KOSapi* [online]. 2015. [cit. 2023-04-14]. Dostupné z: `https://kosapi.fit.cvut.cz/projects/kosapi/wiki`.
25. PERGL, Robert. *Moodle2KOS* [online]. 2017. [cit. 2023-04-14]. Dostupné z: `https://users.fit.cvut.cz/perglr/dokuwiki/doku.php?id=moodle2kos`.
26. BAKALÁŘI SOFTWARE S.R.O. *Mezi školou a rodinou* [online]. 2023. [cit. 2023-04-14]. Dostupné z: `https://www.bakalari.cz/`.
27. BAKALÁŘI SOFTWARE S.R.O. *Datový konektor* [online]. 2023. [cit. 2023-04-14]. Dostupné z: `https://www.bakalari.cz/Static/konektor`.
28. HARDT, D. *The OAuth 2.0 Authorization Framework*. 2012-10. RFC, 6749. IETF. Dostupné také z: `http://tools.ietf.org/rfc/rfc6749.txt`.
29. JIRŮTKA, Jakub. *OAuth 2.0* [online]. 2017. [cit. 2023-04-14]. Dostupné z: `https://rozvoj.fit.cvut.cz/Main/oauth2`.
30. *FIT Classification REST API* [online]. [B.r.]. [cit. 2023-04-14]. Dostupné z: `https://grades.fit.cvut.cz/api/v1/swagger-ui.html`.
31. *Předměty* [online]. 2023. [cit. 2023-04-14]. Dostupné z: `https://bk.fit.cvut.cz/cz/predmety/predmetyAll.html`.
32. ŠLAPÁK, Martin. *BI-PYT Programování v Pythonu: Hodnocení* [online]. 2023. [cit. 2023-04-12]. Dostupné z: `https://courses.fit.cvut.cz/BI-PYT/classification.html`.
33. VÝPOČETNÍ A INFORMAČNÍ CENTRUM ČVUT. *Rozdíly mezi současným a Novým Webovým KOS* [online]. 2022. [cit. 2023-04-14]. Dostupné z: `https://new.kos.cvut.cz/changelog`.
34. SYNOPSYS, INC. *Cross Site Scripting (XSS)* [online]. [B.r.]. [cit. 2023-04-14]. Dostupné z: `https://www.synopsys.com/glossary/what-is-cross-site-scripting.html`.

35. OBJECT MANAGEMENT GROUP. *Business Process Model and Notation (BPMN) Version 2.0.2* [online]. 2014. [cit. 2023-04-14]. Dostupné z: https://www.omg.org/spec/BPMN/2.0.2.

36. PATIL, Aishwarya Vijay. Comparative Analysis between Sequential and Iterative Project Management Approaches. *International Research Journal of Engineering and Technology* [online]. 2019 [cit. 2023-04-15]. ISSN 2395-0056. Dostupné z: https://www.irjet.net/archives/V6/i7/IRJET-V6I7460.pdf.

37. CENTERS FOR MEDICARE & MEDICAID SERVICES, OFFICE OF INFORMATION SERVICES (ed.). *Selecting a development approach* [online]. 2005. [cit. 2023-04-15]. Dostupné z: https://web.archive.org/web/20100331173931/http://www.cms.hhs.gov/SystemLifecycleFramework/Downloads/SelectingDevelopmentApproach.pdf.

38. DJANGO SOFTWARE FOUNDATION. *Model field reference: ManyToManyField* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/ref/models/fields/#manytomanyfield.

39. DJANGO SOFTWARE FOUNDATION. *Django Utils: django.utils.functional* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/ref/utils/#django.utils.functional.cached_property.

40. DJANGO SOFTWARE FOUNDATION. *Django Migrations* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/topics/migrations/.

41. DJANGO SOFTWARE FOUNDATION. *Using the Django authentication system: Limiting access to logged-in users that pass a test* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/topics/auth/default/#limiting-access-to-logged-in-users-that-pass-a-test.

42. DJANGO SOFTWARE FOUNDATION. *Model field reference: JSONField* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/ref/models/fields/#jsonfield.

43. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *JSON Types* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://www.postgresql.org/docs/13/datatype-json.html.

44. PYTHON SOFTWARE FOUNDATION. *Data Structures: Dictionaries* [online]. 2023. [cit. 2023-04-16]. Dostupné z: https://docs.python.org/3.11/tutorial/datastructures.html#dictionaries.

45. DJANGO SOFTWARE FOUNDATION. *Formsets* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/topics/forms/formsets/.

46. *Getting Started: Let's get started with Chart.js!* [Online]. 2023. [cit. 2023-04-15]. Dostupné z: https://www.chartjs.org/docs/4.2.1/getting-started/.

47. MOZILLA FOUNDATION. *Fetch API* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

48. DJANGO SOFTWARE FOUNDATION. *How to use sessions* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/topics/http/sessions/.

49. DJANGO SOFTWARE FOUNDATION. *QuerySet API reference* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/ref/models/querysets/.

50. DJANGO SOFTWARE FOUNDATION. *Django's cache framework* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/topics/cache/.

51. SALGADO, Pablo Galindo (ed.). *What's New In Python 3.10* [online]. 2021. [cit. 2023-04-15]. Dostupné z: https://docs.python.org/3/whatsnew/3.10.html.

52. SALGADO, Pablo Galindo (ed.). *What's New In Python 3.11* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.python.org/3/whatsnew/3.11.html.
53. DJANGO SOFTWARE FOUNDATION. *Django 4.0 release notes* [online]. 2021. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.0/releases/4.0/.
54. DJANGO SOFTWARE FOUNDATION. *Django 4.1 release notes* [online]. 2022. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.1/releases/4.1/.
55. DJANGO SOFTWARE FOUNDATION. *Django 4.2 release notes* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.djangoproject.com/en/4.2/releases/4.2/.
56. GITLAB. *Project settings: Transfer a project to another namespace* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.gitlab.com/ee/user/project/settings/index.html#transfer-a-project-to-another-namespace.
57. *Gitflow Workflow* [online]. [B.r.]. [cit. 2023-04-15]. Dostupné z: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.
58. PRESTON-WERNER, Tom. *Semantic Vesioning 2.0.0* [online]. [B.r.]. Ver. 2.0.0 [cit. 2023-04-15]. Dostupné z: https://semver.org/spec/v2.0.0.html.
59. MACHÁČEK, Jiří. *CloudFIT* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://help.fit.cvut.cz/cloud-fit/index.html.
60. DOCKER, INC. *Install Docker Engine on Debian* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.docker.com/engine/install/debian/.
61. DOCKER, INC. *Install the Compose standalone* [online]. 2023. [cit. 2023-04-15]. Dostupné z: https://docs.docker.com/compose/install/other/.
62. MALEC, Oldřich. *Řízení projektu a infrastruktury portálu pro podporu výuky předmětu BI-DBS*. Praha, 2017. Dostupné také z: http://hdl.handle.net/10467/69399. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
63. DJANGO SOFTWARE FOUNDATION. *Writing and running tests* [online]. 2023. Ver. 4.2 [cit. 2023-04-22]. Dostupné z: https://docs.djangoproject.com/en/4.2/topics/testing/overview/.
64. KREKEL, Holger. *How to use unittest-based tests with pytest* [online]. 2015. [cit. 2023-04-22]. Dostupné z: https://docs.pytest.org/en/7.3.x/.
65. ROSSUM, Guido van; WARSAW, Barry; COGHLAN, Nick. *Style Guide for Python Code*. 2001-07. PEP, 8. Dostupné také z: https://peps.python.org/pep-0008/.
66. IBM CORPORATION. *JUnit XML format* [online]. 2023. [cit. 2023-04-22]. Dostupné z: https://www.ibm.com/docs/en/developer-for-zos/16.0?topic=formats-junit-xml-format.
67. GITLAB. *Unit test reports* [online]. [B.r.]. [cit. 2023-04-22]. Dostupné z: https://docs.gitlab.com/ee/ci/testing/unit_test_reports.html#unit-test-reports.

# Obsah přiloženého archivu

Mimo přiložený archiv se veškeré zdrojové kódy nacházejí v GitLab repozitáři dostupném na adrese `https://gitlab.fit.cvut.cz/sos/sos`. Výstupy práce studentů BI-SP1 jsou dostupné v Redmine Wiki na adrese `https://dbs.fit.cvut.cz/redmine/projects/sos-fit-cvut-cz/wiki`. Do repozitáře i Wiki má přístup má přístup vedoucí práce Ing. Jiří Hunka a oponent práce Ing. Michal Valenta, Ph.D. Na požádání mohou být přístupy uděleny i dalším osobám.

```
readme.txt ..... stručný popis obsahu archivu
src/
    impl/ ..... zdrojové kódy implementace
    thesis/ ..... zdrojová forma práce ve formátu LaTeX
text/ ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
media/ ..... multimediální příloha
    nur-semestralni-prace/ ..... semestrální práce z NI-NUR odkazovaná v příloze A
    prace-sp1/ ..... výstupy práce studentů BI-SP1 odkazované v příloze B
    navod-sp1.pdf ..... obrázkový návod demonstrující funkcionality verze 1.0.0
```