FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	SOS III - Studentský odevzdávací systém - hodnocení, odevzdání
Student:	Jan Mrázek
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 21. ledna 2024 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Pokyny pro vypracování

Již nyní existuje sos.fit.cvut.cz - dále jen portál, který slouží k podpoře týmových cvičení na FIT ČVUT. Jeho aktuální stav je provozuschopný a reálně na něm aktivně probíhají například předměty NI-NUR, BI-SP1 a BI-SP2. Cílem této práce, je na základě aktuálních zkušeností a dle aktuální funkcionality realizovat nový, snadno udržitelný a dobře rozšiřitelný portál SOS, který v budoucnu nahradí aktuálně dostupný a funkční portál. Dílčím cílem je také reálně vyzkoušet iterativní formu tvorby software a práci v týmu s kolegy Janem Jamnickým a Markem Hujo, kteří řeší ostatní části systému.
Přímo tato práce je primárně zaměřena na část systému hodnocení a odevzdání.

Postupujte v těchto krocích:
1. Analyzujte současný stav portálu včetně dílčích úprav ostatních autorů.
2. Analyzujte vhodným způsobem aktuální možnosti převážně hodnocení a odevzdání současného portálu SOS s ohledem na možné budoucí využití nejen na FIT ČVUT ale i v jiných školních prostředích.
3. Navrhněte a realizujte za pomocí iterativního vývoje ve spolupráci s ostatními členy týmu nový portál SOS.
4. Připravte s ostatními členy týmu aplikaci pro reálné využití na FIT ČVUT.
5. Výsledný portál řádně otestujte vhodně zvolenými testy.
6. Zajistěte projekt tak, aby bylo snadné a efektivní dalšími lidmi portál nadále vyvíjet a rozvíjet.
7. Zhodnoťte dosažené výsledky a stav portálu, navrhněte možná budoucí vylepšení.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 21. ledna 2024 v Praze.

Bakalářská práce

SOS III - STUDENTSKÝ ODEVZDÁVACÍ SYSTÉM - HODNOCENÍ, ODEVZDÁNÍ

Jan Mrázek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Hunka
16. května 2024

České vysoké učení technické v Praze
Fakulta informačních technologií

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat své rodině a přítelkyni za neustálou podporu při celém studiu a tvorbě této práce. Dále bych chtěl poděkovat Ing. Jiřímu Hunkovi za vedení práce, odborné konzultace a projevenému nadšení pro toto téma. Dále děkuji také kolegům z týmu BI-SP1 a BI-SP2, Timoteji Adamcovi, Janu Jamnickému a Marku Hujovi, se kterými jsme postavili základ tohoto systému a tedy i této práce. Rovněž děkuji studentům BI-SP1 2024, jmenovitě Františku Holému, Matěji Hrbáčkovi, Matěji Procházkovi, Oldřichu Macháčkovi a Saši Vobořilovi, kteří se zapojili do projektu a poskytli výstupy, které budou užitečné při dalším vývoji tohoto řešení. V neposlední řadě bych chtěl poděkovat Bc. Maxu Hejdovi za cenné rady a pomoc, které poskytoval v průběhu tohoto projektu.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 16.5.2024

Abstrakt

Tato bakalářská práce se zabývá vypracováním částí hodnocení a odevzdávání v nové verzi webové aplikace SOS – Studentský Odevzdávací Systém. Výsledkem práce jsou vypracované a otestované sekce, včetně seznamu nedostatků a budoucích vylepšení. Nová verze tohoto systému je vyvíjena společně s týmovými kolegy za pomoci iterativního vývoje. Frontend aplikace je postaven na frameworku Nuxt 3 a backend na ASP.NET Core 8.

Klíčová slova Studentský Odevzdávací Systém, iterativní vývoj, webová aplikace, Vue.js, Nuxt 3, C#, .NET Core 8

Abstract

This bachelor's thesis deals with the development of assessment and submission sections in the new version of the SOS – Student Submission System web application. The outcome of the work consists of developed and tested sections, including a list of deficiencies and future enhancements. The new version of this system is being developed collaboratively with team colleagues using iterative development. The frontend of the application is built on the Nuxt 3 framework, and the backend on ASP.NET Core 8.

Keywords Student Submission System, iterative development, web application, Vue.js, Nuxt 3, C#, .NET Core 8

Seznam zkratek

AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CD	Continuous Deployment
CI	Continuous Integration
CORS	Cross-Origin Request Sharing
CSRF	Cross-Site Request Forgery
CSS	Cascading Style Sheets
ČVUT	České vysoké učení technické
DOM	Document Object Model
FIT	Fakulta informačních technologií ČVUT
GJGJ	Gymnázium Jiřího Gutha-Jarkovského
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
MVC	Model View Controller
MVT	Model View Template
MR	Merge Request
ORM	Objektově-relační mapper
REST	Representational State Transfer
SID	Session ID
SOS	Studentský Odevzdávací Systém
SQL	Structured Query Language
UI	User Interface (Uživatelské rozhraní)
URL	Uniform Resource Locator
UX	User Experience
XML	Extensible Markup Language
XSS	Cross Site Scripting

Úvod

V současné době existuje mnoho aplikací pro správu úkolů nebo projektů pro školní prostředí. Tyto systémy poskytují různé funkce, které jsou často specifické pro průběh předmětu a usnadňují tím práci jak vyučujícím, tak studentům. Jedním z takovýchto systémů je také Studentský Odevzdávací Systém – SOS, kterým se zabývá následující text.

Tento systém slouží pro správu složitějších projektů, na kterých typicky pracuje více lidí a jsou odevzdávány po částech. Pro tyto účely je systém vybaven mnoha funkcemi a rozsáhlou konfigurací, která je dobře připravena na téměř libovolný předmět. Toto ostatně dokazuje také jeho rozšíření mimo půdu FIT ČVUT.

Tento systém byl vyvinut Ing. Tomášem Pavlůskem pod vedením Ing. Jiřího Hunky. Záminkou pro vznik SOSu byla převážně potřeba sjednotit řešení, která se používají pro výše zmíněné projekty. Inspirací pro tento systém byl *NURIS*, který sloužil pro potřeby magisterského předmětu NI-NUR. Během jeho provozu byl systém dále rozšiřován a upravován a to jak autorem systému, tak jinými lidmi. Výsledkem je portál, který slouží na půdě FIT ČVUT pro správu společných projektů v rámci předmětů BI-SP1, BI-SP2, BI-SWI a také NI-NUR. Dále se tento systém využívá na Gymnáziu Jiřího Guthа-Jarkovského a to pro správu ročníkových prací. Na gymnázium systém rozšířil Bc. Max Hejda v rámci jeho bakalářské práce a přinesl tím i další vylepšení, využitelná i na půdě FIT ČVUT.

Systém SOS jsme používali v rámci předmětů SP (společný týmový projekt). Trochu ironicky jsme se v tomto předmětu podíleli právě na rozvoji tohoto systému a sbírali jsme požadavky a zpětnou vazbu lidí, kteří s ním přišli do kontaktu. Díky tomu jsme zjistili, že systém trápí některé nedostatky, které je potřeba vyřešit. Avšak toto řešení nebude zcela přímočaré, jelikož některé problémy, které uživatele nejvíce trápí, mají hluboké kořeny.

V návaznosti na analýzu možných řešení jsme se rozhodli pro přepsání systému do nové podoby a to spolu s kolegy Janem Jamnickým a Markem Hujem. Našim cílem je využít všechny aktuální znalosti a podklady a vytvořit nový systém, který postupem času nahradí stávající a funkční řešení. Navíc bude

nový systém implementován v atraktivnější technologiích, které se používají i na mnoha jiných projektech a to nejen na půdě FIT ČVUT, ale i v praxi.

Přepsáním sekce hodnocení a odevzdávání projektů se zabývá právě tato práce. Analýzou a nedostatky této částí systému jsem se zabýval již v předmětu BI-SP1 a BI-SP2 a mám proto velkou motivaci pokračovat ve své práci. Změny, které v této práci realizuji, vyzkouším na uživatelském testování s reálnou cílovou skupinou uživatelů. Díky tomu je velká šance, že se odladí většina nedostatků, které byly nalezeny a vznikne tak uživatelsky přívětivé rozhraní pro odevzdávání projektů a jejich hodnocení.

Cíl práce

Tato práce si klade za cíl analyzovat potřeby a nedostatky sekcí hodnocení a odevzdávání systému SOS. Následně budou postupně prováděny návrhy a implementace, které ve výsledku přinesou novému systému SOS nové uživatelské rozhraní a tím i lepší uživatelský zážitek při odevzdávání a hodnocení projektů. Finálním výsledkem by měla být funkční sekce hodnocení a odevzdávání projektů, která bude nasazená na fakultním serveru sloužícím pro testovací provoz nového portálu.

Při vývoji bude kladen důraz na rozšiřitelnost a udržitelnost kódu a také dokumentaci relevantních částí, aby bylo možné výsledný projekt dále rozvíjet.

Dílčím cílem je také vyzkoušení procesu iterativního vývoje s kolegy Janem Jamnickým a Markem Hujem, kteří zpracovávají ostatní části systému. Tento režim spolupráce jsme začali využívat v rámci BI-SP2 a rozhodli jsme se v něm pokračovat i při dalším vývoji nového systému SOS.

Kapitola 1

Vývoj webových aplikací

V této kapitole se věnuji tomu, co vlastně webové aplikace jsou a jaké architektury lze při jejich psaní využít. Dále zde analyzuji bezpečnostní rizika a možnosti obrany proti nim. Poté zkoumám různé způsoby testování, nasazování a monitorování webových aplikací. Nakonec se věnuji populárním technologiím pro tvorbu webových aplikací a jaké přínosy přinášejí.

1.1 Definice webových aplikací

Webovou aplikací se myslí aplikace, která běží na serveru a je uživatelům přístupná přes nějakého klienta. Klientem se v kontextu webových aplikací myslí většinou webový prohlížeč, ve kterém je vykresleno uživatelské rozhraní, pomocí kterého uživatel interaguje s aplikací. Oproti klasické aplikaci nainstalované přímo v zařízení, se webová aplikace liší tím, že není potřeba nic instalovat. Stačí již zmíněný internetový prohlížeč [1]. Komunikace mezi klientem a serverem poté probíhá pomocí protokolu HTTP.

1.1.1 HTTP

HTTP (Hypertext Transfer Protocol) je protokol fungující na aplikační vrstvě ISO/OSI modelu. Definuje způsob přenosu informací mezi zařízeními fungujícími přes síť. Typický průběh zahrnuje požadavek od klienta a následnou odpověď od serveru [2].

1.1.1.1 Požadavek

HTTP požadavek se skládá z dat potřebných pro vytvoření odpovědi. Tyto data obsahují verzi HTTP, URL adresu, HTTP metodu, hlavičky a volitelně také tělo požadavku obsahující data v určitém formátu [2].

HTTP metody se používají pro rozlišení typu akce prováděné na serveru. Nejvíce využívané jsou metody *GET* a *POST*. Standard však definuje i další.

Mezi často používané u webových aplikací patří *PUT*, *PATCH*, *DELETE* a *OPTIONS* [3].

1.1.1.2 Odpověď

Odpověď obvykle následuje po požadavku. Obsahuje stavový kód, hlavičky a volitelně také tělo odpovědi, obsahující data v různých formátech [2].

1.2 Architektury webových aplikací

V této sekci se podrobněji věnuji různým přístupům k návrhu architektury a dalšího členění kódu podle odpovědností a funkcionalit. Je nutné říci, že webové aplikace nelze často striktně zařadit do některé z architektur, proto je níže zmíněno jemnější rozdělení. Zástupci níže zmíněných kategorií se často doplňují – volba architektury plně neomezuje volbu vrstvení, nebo návrhového vzoru.

1.2.1 Architektura

Architektura webové aplikace přímo ovlivňuje vlastnosti, jako škálovatelnost, výkonnost nebo udržitelnost [4]. Architektury mohou dále i omezovat, v jakém prostředí může aplikace běžet, čemuž se věnuji převážně v sekci o serverless architektuře.

1.2.1.1 Monolitická architektura

Aplikace založená na monolitické architektuře se vyznačuje tím, že všechny důležité části jako byznys logika, přihlašování a přístup k datům jsou obsaženy v jednom balíku. Aplikace samozřejmě může interagovat s jinými službami, kterými jsou například databáze, nebo API třetích stran, pro potřeby ukládání, nebo získávání dat [5].

Ač se může zdát v porovnání s mikroslužbami nebo serverless horší volbou, opak je pravdou. Pro menší aplikace převažují výhody monolitu, jako například rychlý vývoj, snadné ladění a jednoduché nasazování nad jeho nevýhodami. Nevýhody se typicky projevují až v rozsáhlejších aplikacích a patří mezi ně hlavně obtížné aktualizace, provázanost kódu a omezené škálování [6].

1.2.1.2 Mikroslužby

Mikroslužby, neboli *microservices*, je označení pro architekturu, která se skládá z mnoha malých, soběstačných částí – mikroslužeb. Každá mikroslužba má předem určenou funkci a obsahuje vše potřebné pro její provedení. Zároveň je samostatně nasaditelná a udržovatelná [7]. Jednotlivé mikroslužby spolu poté komunikují prostřednictvím HTTP požadavků, nebo zpráv zasílaných přes sběrnici událostí [8].

Mezi výhody mikroslužeb patří především výborná škálovatelnost a při rozsáhlejších aplikacích také rychlost vývoje a udržitelnost, která je zajištěna oddělením do menších celků. Mezi nevýhody poté patří hlavně složitost propojení a komunikace jednotlivých mikroslužeb a také složité monitorování a ladění [9].

1.2.1.3 Serverless architektura

Serverless architektura využívá k provozu cloudové funkce u různých poskytovatelů (například Azure, AWS nebo Google Cloud). Zatímco monolitická architektura, nebo mikroslužby většinou fungují jako tzv. IaaS[1], nebo PaaS[2], tak serverless architektura používá FaaS[3]. To znamená, že poskytovatel dodává výpočetní výkon v podobě provedení funkce a to pouze, když je o to požádáno [10].

Mezi hlavní výhody serverless architektury patří hlavně nízké náklady na provoz, dobré škálování a zajištění odolnosti vůči výpadku přímo poskytovatelem cloudu. Mezi hlavní nevýhody těchto aplikací poté spadá obtížnější testování, složité zabezpečení a vysoká prvotní odezva, jelikož při prvním požadavku se musí spustit celá architektura, na které aplikace funguje [11].

1.2.2 Vrstvení

Vrstvení určuje logické seskupení kódu nebo služeb podle jejich funkcionalit a odpovědností. Každá vrstva by měla být co nejvíce oddělená od ostatních a závislosti by měly být definovány pomocí rozhraní. Díky tomu se poté urychluje navigace v kódu a usnadňuje se údržba či případně změny [5]. Často se vrstvení udává i jako architektura, ale subjektivně jej chápu spíše jako logické uskupení jednotlivých částí kódu.

1.2.2.1 Rozdělení do N-vrstev

Typickým zástupcem vrstvení je třívrstvá architektura. V případě nutnosti je možno ubrat nebo přidat vrstvy, ale minimum jsou zpravidla dvě vrstvy. Dále budu uvažovat nejčastěji používané vrstvení – tedy na tři vrstvy [5]. Rozdělení do tří vrstev je znázorněno na obrázku 1.1. V takto vrstvených aplikacích vyšší vrstva závisí na nižší (tedy prezentační vrstva, závisí na logické atd.). Typická třívrstvá aplikace obsahuje následující vrstvy:

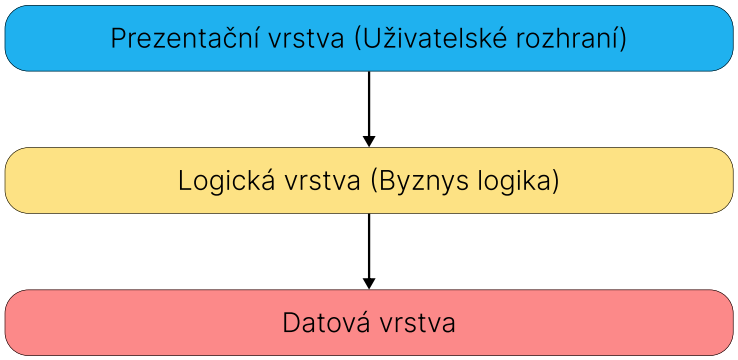
1. **Prezentační vrstva:** zde je uložena logika pro zobrazení dat a zpracování uživatelských vstupů.
2. **Logická vrstva:** jedná se o vrstvu, kde je uložena byznys logika aplikace. Jinak se této vrstvě říká také servisní, byznysová, nebo aplikační.

[1] Infrastructure-as-a-Service – jedná se například o pronajatý server, nebo virtuální stroj

[2] Platform-as-a-Service – umožňuje pohodlný vývoj bez starostí o infrastrukturu - např. Vercel, nebo Platform.sh

[3] Function-as-a-Service

3. **Datová vrstva:** označuje vrstvu, která se stará o komunikaci s databází nebo jiným úložištěm dat (i vzdáleným).



Obrázek 1.1 Třívrstvá architektura, přeloženo z e-knihy od Microsoftu [5]

Dependency injection

Než se budu věnovat dalšímu způsobu vrstvení, tak zadefinuji pojmy *dependency injection* (DI) a *inversion of control* (IoC).

Inversion of control je návrhový vzor zvyšující modularitu a testovatelnost aplikací. Principem tohoto vzoru je předání kontroly nad programovými toky z rukou programátora do rukou externí entity, většinou frameworku. Dependency injection je potom konkrétní realizace tohoto principu. Závislosti objektů jsou poskytovány zvenčí, prostřednictvím konstruktoru nebo metod [12]. Tyto principy jsou hojně využívány moderními frameworky bez nutnosti rozsáhlých konfigurací. Díky tomu je možné tvořit modulární a dobře testovatelné aplikace.

1.2.2.2 Čistá architektura

Čistá architektura (přeloženo z *clean architecture*) je označení pro další způsob vrstvení aplikace. Na rozdíl od klasické N-vrstvé architektury je však ve spodní vrstvě jádro aplikace a aplikační logika.

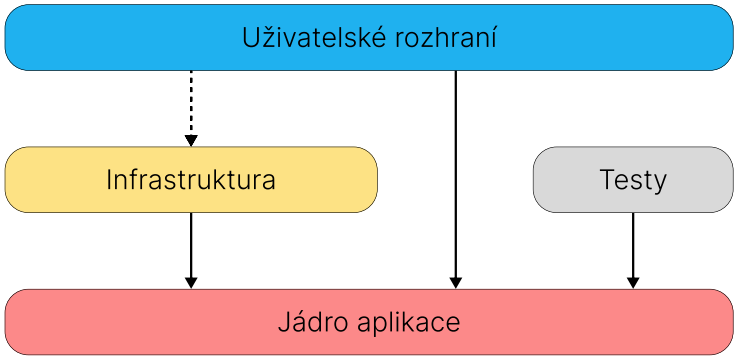
Pro čistou architekturu je stěžejní využití *dependency injection*. Díky tomu je možné využít při kompilaci závislostí přímo na vrstvě jádra. Za běhu jsou potom implementace závislostí poskytnuty frameworkem díky dependency injection. Tento mechanismus znázorňuje obrázek 1.2, kde plné čáry značí závislosti při kompilaci a přerušované značí závislosti při běhu [5]. Aplikace vyvíjeny podle tohoto principu, obsahují typicky následující vrstvy:

1. **Jádro aplikace:** obsahuje byznys logiku, entity, DTO[4] a rozhraní. Rozhraní definovaná v této vrstvě jsou implementována další vrstvou.

[4]Data Transfer Object - objekty sloužící pro přenos dat

2. **Infrastruktura:** zahrnuje implementaci rozhraní definovaných v jádru aplikace. Implementace jsou za běhu aplikace poskytovány prostřednictvím dependency injection.

3. **Uživatelské rozhraní:** označuje vstupní bod aplikace. Obsahuje controllery, šablony pro zobrazení, nebo vlastní middleware.



Obrázek 1.2 Čistá architektura, přeloženo z e-knihy od Microsoftu [5]

1.2.3 Návrhové vzory

Návrhové vzory představují obecné řešení problému a slouží jako další vodítko pro strukturování kódu a separaci odpovědnosti v aplikaci. Jedná se o obecné koncepty, které lze využít v jakémkoliv jazyce a frameworku [13]. V doplnění s architekturou a vrstvením aplikace dávají dohromady kompletní sadu pravidel, kterými by se vývojář při implementaci měl řídit, aby byl výsledný kód dobře rozšiřitelný a udržitelný. Existuje celá řada návrhových vzorů aplikací, každopádně zaměřím se zde na dva, které považuji za relevantní pro tento text. Dalšími vzory, které se v tomto kontextu používají, je například MVP[5], nebo MVVM[6]. Rozdíly mezi těmito návrhovými vzory jsou v dnešní době poměrně malé a většinou jde o odlišné zodpovědnosti a pojmenování jednotlivých částí daného vzoru. V této sekci se věnuji architektonickým návrhovým vzorům a ne tzv. GoF[7], které se však v aplikacích také hojně používají.

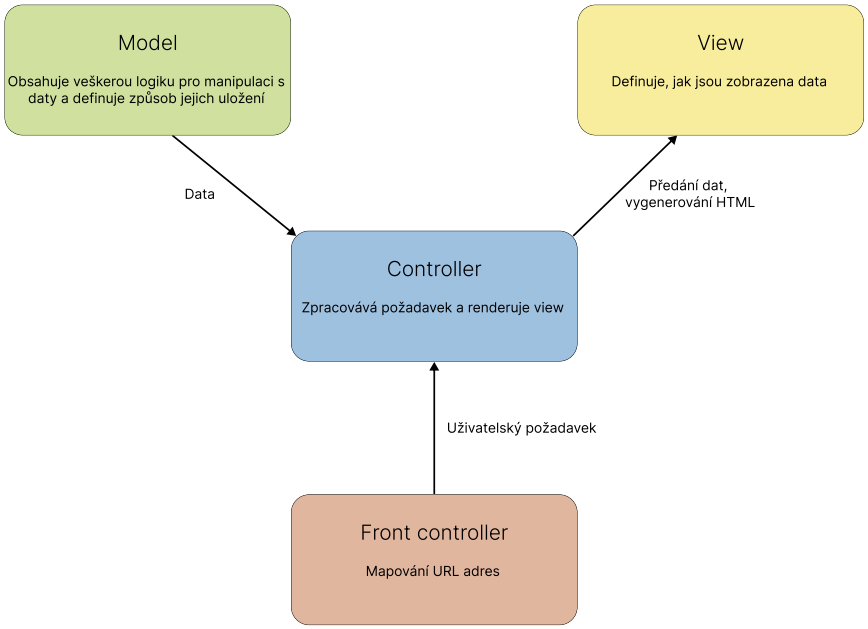
MVC – Model View Controller

Model View Controller je označení pro návrhový vzor, velmi populární pro vývoj moderních webových aplikací. Oproti klasickému MVC je varianta MVC určena pro webové aplikace je trochu odlišná. Tato verze vychází z původní

[5] Model View Presenter
[6] Model View ViewModel
[7] Gang of Four – všeobecně uznávaná kniha o řešení obecných problémů v objektově orientovaném kódu [14]

Model 2 architektury složené pro Javu [15]. Skládá se ze 4 částí, které mezi sebou interagují. Tyto části jsou znázorněny na obrázku 1.3.

1. **Model:** jedná se o nejdůležitější část aplikace. Spravuje data a byznys logiku naší aplikace.
2. **View:** tato vrstva zajišťuje prezentaci dat uživateli. Za view můžeme považovat například vygenerovanou HTML stránku.
3. **Controller:** vrstva, která se stará o spojení view a modelu. Přijímá vstupy od view a předává je modelu ke zpracování. Dále také vyřizuje zpětné předání dat z modelu zpátky do view.
4. **Front controller:** představuje vstupní bod aplikace. Provádí akce jako například autentizaci, autorizaci a předání požadavku příslušnému controlleru.



Obrázek 1.3 Princip fungování MVC, přeloženo z archivního článku [16]

Modifikace oproti původnímu návrhovému vzoru spočívá v tom, že view je při každém požadavku znovu vyrenderován. V původním MVC vzoru je u view použit návrhový vzor pozorovatel [17], který zajišťuje aktualizaci dat oproti modelu. V modifikaci MVC pro webové aplikace se návrhový vzor pozorovatel nepoužívá [15].

MVT – Model View Template

Jedná se návrhový vzor podobný MVC. Controller je zde reprezentován frameworkem jako takovým. Ten se stará o předání požadavků view vrstvě a vrácení odpovědi klientovi. Tento návrhový vzor je implementován frameworkem využitým při vývoji aktuálního SOSu – Djangem. Obsahuje opět tři hlavní vrstvy a navíc framework, jak je znázorněno na obrázku 1.4. Tyto vrstvy jsou:

1. **Model:** zodpovídá za manipulaci s daty v databázi, je reprezentovaný třídami v Pythonu a mapovaný na databázové tabulky prostřednictvím ORM.
2. **View:** zpracovává uživatelské požadavky. V této vrstvě se nachází veškerá logika aplikace, autentizace, autorizace, získávání a zpracování dat, a interakce s model a template vrstvami.
3. **Template:** generuje kód uživatelského rozhraní. Data získává z vrstvy view a šablonovací jazyk frameworku umožňuje pokročilou manipulaci s daty včetně využití podmínek, smyček a předdefinovaných funkcí pro zobrazení.
4. **Framework:** zajišťuje mapování URL adres na příslušné view, předávání uživatelských požadavků do view a další podpůrné úkony

Ve výsledku je tento model velmi podobný právě webové variantě MVC. Rozdíl je zde hlavně v rozdělení odpovědností. View v případě MVT obsahuje většinu aplikační logiky a rozhoduje, jaká data zobrazíme. V případě MVC view rozhoduje, jak data zobrazíme [18].

1.3 Zabezpečení webových aplikací

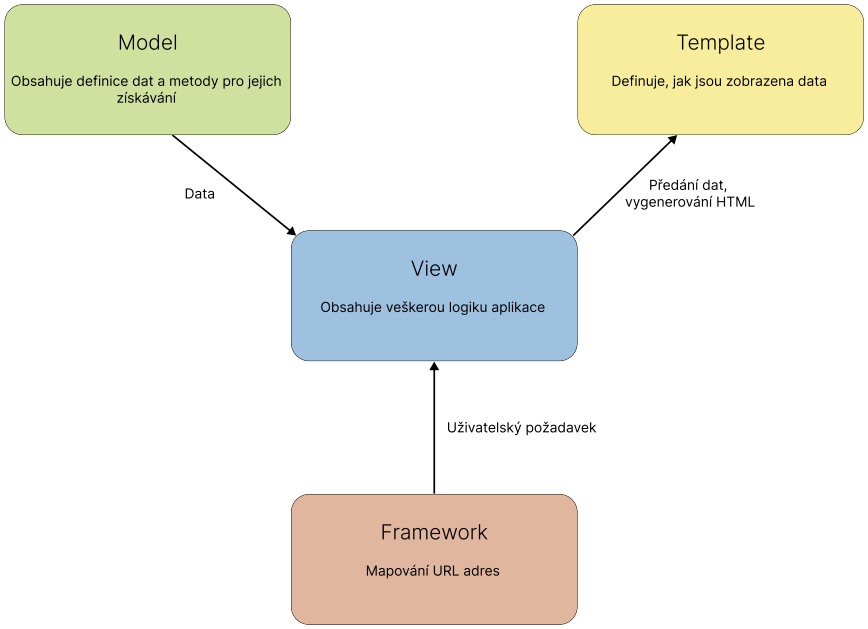
Bezpečnost aplikace je jednou z klíčových věcí, na které musí vývojáři při tvorbě a provozu webové aplikace myslet. Kvůli dostupnosti webových aplikací přes internet narůstá riziko ztráty uživatelských dat. Dále jsou tyto aplikace náchylnější na útoky, které cílí na zpomalení služeb, nebo zkopírování obsahu pro účely získání konkurenční výhody [19].

1.3.1 Útoky a zranitelnosti

V této sekci se věnuji nejčastějším útokům na webové aplikace, které se často objevují na seznamu OWASP top 10 [20]. Pro zajímavost také uvádím příklady známých útoků.

1.3.1.1 XSS – Cross Site Scripting

Jedná se útok, kdy útočník vloží škodlivý kód na věrohodnou stránku. Škodlivý kód je odeslán serverem oběti do prohlížeče, kde se vykoná. Cílem tohoto útoku



Obrázek 1.4 Princip fungování MVT

je většinou získání citlivých dat uživatele, jako například cookies (to může vést na session hijacking, popsaný v sekci 1.3.1.2) nebo provedení jiné škodlivé akce [21]. Tento typ útoku se dále dělí na:

1. **Reflektovaný:** tento typ útoku většinou vyžaduje, aby uživatel klikl na odkaz, který obsahuje škodlivý kód. Škodlivý kód se pošle na server a ten jej vzápětí zašle odpovědí uživateli. Serverem vrácený kód je potom proveden v prohlížeči, jelikož jej vrátil věrohodný server.
2. **Persistentní:** využívá škodlivý kód, který je uložen přímo na serveru, klasicky v databázi (například uložený komentář). Když uživatel zobrazí daný komentář, tak mu server pošle dříve uložený škodlivý kód a ten se opět provede v prohlížeči. [21]

Ochrana

V dnešní době naštěstí mnoho frameworků implementuje nativní obranu proti XSS formou tzv. *escapování* nedůvěryhodných (čtěte uživatelských) vstupů. Tuto formu ochrany poskytují téměř všechny JavaScriptové frameworky, jako například React nebo Vue.js [22] a je také možné jí použít ve většině šablonovacích jazyků (například Django template language). I přesto je třeba být

opatrný při vkládání uživatelského vstup zpět na stránku. Dále je vhodné prostudovat dokumentace technologií, které používáme. V těchto dokumentacích se často dá dozvědět, jaký mechanismus ochrany je k dispozici [23].

Známé útoky

- **British Airways:** v roce 2018 využila skupina Magecart XSS k odesílání zákaznických dat na jejich server. Takto útočníci získali data ze 380 000 transakcí, než byli odhaleni. K XSS použili zranitelnost v JavaScriptové knihovně, kterou web používal [24].
- **eBay:** na začátku roku 2016 používaly stránky eBay parametr *url*, který nebyl nijak validován. Tohoto mohli využít útočníci k provedení XSS útoku, nicméně zranitelnost byla opravena [25].

1.3.1.2 Session hijacking

Tento typ útoku spočívá ve zneužití způsobu, jakým většina webových služeb řeší autentizaci. Na serveru se vytvoří session – ten obsahuje údaje o uživateli a je na serveru uložen pod vlastním ID. Toto ID se obecně označuje jako Session ID (SID). SID je zároveň uloženo jako cookie do prohlížeče, přes který proběhlo přihlášení. Při každém dalším požadavku jsou tyto cookies posílány v hlavičce na server a díky nim je server schopný identifikovat přihlášeného uživatele [26].

Ochrana

Standardně jsou tyto cookies chráněny parametry jako *http-only* a *secure*. Pokud však toto není použito nebo je cookie odposlechnuta, či jinak získána útočníkem, může se útočník vydávat za oběť. Je několik různých možností získání této cookie, jednou z nich je již zmíněný XSS, další zahrnují odposlechy síťové komunikace nebo phishing[8] [27]. Zajímavostí je, že historicky se SID posílalo v URL query parametru, v dnešní době se však tento způsob považuje na méně bezpečný [28].

Známé útoky

- **Slack:** v roce 2019 byla odhalena zranitelnost, která umožnila útočníkům získat SID uživatelů. Útočníci potom mohli získat data ze společností, kde oběti pracovaly [29].
- **GitLab:** v roce 2017 byla odhalena zranitelnost v GitLabu, kde byly SID dostupné přímo v URL. Tyto ID ještě navíc neměly omezenou platnost,

[8] Technika, která zahrnuje manipulaci obětí a snahu vylákat z ní citlivá data, nebo ji přimět provést nějakou akci

což by v případě jejich získání poskytlo útočníkům časově neomezený přístup k účtu oběti [30].

1.3.2 SQL Injection

Tento útok využívá neošetřené vstupy pro provedení škodlivé akce oproti databázi. Tato akce může mít různé formy – ať už získání dat, jejich úprava, nebo smazání. V nejhorších případech může dojít k úniku dat a následnému smazání celé databáze, kterou nelze obnovit, pokud nebyla zálohovaná [31].

Ochrana

Obranu proti tomuto typu útoku již má hodně frameworků přímo integrovanou. Zpravidla se jedná o takzvanou parametrizaci dotazů [32].

Známé útoky

- **Heartland Payment Systems:** v roce 2008 útočníci pomocí SQL Injection získali údaje až 130 milionů kreditních a debetních karet. Jedná se o jeden z nejhorších útoků v historii [33].
- **Sony Pictures:** v roce 2011 postihla tuto společnost vlna SQL Injection útoků, díky kterým unikla data až 100 milionů účtů ze služby PlayStation Network [34].

1.3.3 CSRF – Cross Site Request Forgery

Tento útok spočívá v neautorizovaném či neúmyslném provedení akce uživatelem. Oběť často ani neví, že akci provedla. Útok spočívá v tom, že uživatel, který má v prohlížeči uložen SID pro cílenou stránku, navštíví škodlivý web. Ne tomto webu poté klikne na odkaz, který provede akci na cílené stránce. Provedení je možné díky tomu, že prohlížeče automaticky posílají cookies při požadavcích na doménu, na které byly cookies vytvořeny. Díky tomuto může útočník provést takřka jakoukoliv akci a vydávat se přitom za poškozeného uživatele. Typicky jsou tímto útokem cíleny akce, které dělají změny stavu v systému. Akce získávající data nejsou příliš užitečné, jelikož odpověď serveru nezískává útočník, ale přímo poškozený [35].

Ochrana

Bránit tomuto útoku se typicky dá pomocí tzv. CSRF tokenu. Tento token je vygenerován při přihlášení (nebo při navštívení stránky, obsahující formulář) a je zaslán pouze jednou klientovi. Úkolem klienta je poté poslat tento CSRF token při každém požadavku, který mění stav na serveru. Tento token se potom typicky ukládá do cookies, lokálního úložiště prohlížeče, nebo skrytého

formulářového vstupu. Potom jej zbývá připojit k požadavku a odeslat na server, který jej spáruje s SID a tak ověří, že uživatel provádějící danou akci je ten, který se přihlásil [36].

Známý útok

- **TikTok:** v roce 2020 byl na této populární sociální síti objevena zranitelnost, která pomocí CSRF umožnila změnit údaje k účtu, který byl propojen přes aplikaci třetí strany [37].

1.3.4 DoS a DDoS

DoS (Denial of Service) a DDoS (Distributed Denial of Service) jsou útoky, které cílí na zpomalení nebo úplné zahlcení webové služby. Tyto útoky si kladou za cíl znepřístupnit službu běžným uživatelům. Většinou se tyto útoky zaměřují na známé, důležité a hojně využívané weby, kde může v důsledku útoku vzniknout značná finanční ztráta. Rozdíl mezi DoS a DDoS je ten, že DoS probíhá typicky z jednoho stroje, zatímco DDoS se provádí z mnoha strojů nebo tzv. botnetu[9] [38]. Díky tomu jsou DDoS často mnohem účinnější v zahlcení cílové služby.

Ochrana

Existuje více druhů DoS útoků, které cílí na zahlcení různých zdrojů webu a proto nelze jednoduše navrhnout ochranu. Běžné typy ochrany zahrnují například limitování počtu požadavků z IP adresy, nebo využití Load balanceru [39].

Známé útoky

- **Google, Cloudflare a AWS:** v roce 2023 čelilo několik společností největšímu DDoS útoku v historii. Útočníci byli schopni poslat na servery Googlu až 398 milionů požadavků za vteřinu. Cloudflare byl zahlcen 201 miliony požadavky a AWS 155 miliony [40].
- **Estonsko:** v roce 2007 byly státní služby Estonska napadeny DDoS útokem. To znepřístupnilo vládní služby, ale také finanční a mediální instituce. Estonsko bylo v té době průkopníkem digitalizace, což přidalo na vážnosti útoku [41].

1.3.5 Instituce a komunity

Bezpečnost webových aplikací je nesmírně důležité a obsáhlé téma. Vzhledem k popularizaci webových aplikací, které ukládají více a více dat, je nutné na

[9] Síť počítačů ovládaná útočníkem.

bezpečnost myslet. Proto se bezpečnosti aplikací a webů věnuje mnoho organizací a komunit. Nejdůležitější organizace zmiňuji v seznamu níže.

1.3.5.1 OWASP

OWASP (Open Web Application Security Project) je nezisková organizace, která se zabývá osvětou ohledně bezpečnosti softwaru a snaží se o zlepšení celkové situace v tomto ohledu. Tato organizace vytváří různé dokumenty a nástroje, které mohou používat lidé po celém světě ke zlepšení bezpečnosti jejich aplikací [42]. Nejvýznačnějšími dokumenty, které tato organizace vydává, je seznam deseti nejběžnějších zranitelností či útoků proti aplikacím [20], a také seznam pravidel jak se proti nim bránit [43].

1.3.5.2 CSIRT.CZ

Jedná se o českou instituci, která se podobně jako OWASP věnuje osvětě a školení v oblasti bezpečnosti, sledování informací o chystaných útocích, ale také řešení bezpečnostních incidentů. Je provozována sdružením CZ.NIC a spolupracuje s různými institucemi. Mezi tyto instituce patří například ČTU[10], banky, ISP[11]. Provozují také službu Skener Webu[12], který se používá pro ověřování zranitelností převážně v neziskové a státní sféře, ale je možné si zažádat o objednávku i pro komerční projekty [44].

1.3.5.3 CVE

CVE (Common Vulnerabilities and Exposures) je celosvětová databáze zranitelností vyskytujících se v softwaru. Může jít o zranitelnosti v aplikacích nebo pouhých knihovnách. Každá zranitelnost má jednoznačný identifikátor. Identifikátor poté využívají nástroje, které skenují závislosti aplikací na bezpečnostní chyby. Tyto zranitelnosti jsou přidávány jak uživateli, tak samotnými vývojáři [45].

1.3.5.4 CWE

CWE (Common Weakness Enumeration) je obdobně jako CVE databáze běžně se vyskytujících slabých míst v aplikacích, ale i hardwaru. Tato slabá místa mohou být posléze příčinami pro zranitelnosti. Narozdíl od CVE se ale zaměřuje na prevenci vzniku těchto zranitelností, které plynou z nebezpečného návrhu nebo jiných praktik [46].

[10] Český Telekomunikační Úřad
[11] Internet Service Providers – poskytovatelé připojení k internetu
[12] `https://www.skenerwebu.cz/cs/`

1.4 Testování webových aplikací

Testování webových aplikací je nedílná součást jejich vývoje. Kromě zaručení funkčnosti kritických částí aplikace se může testování zabývat také výkonností, použitelností a ostatními aspekty webových aplikací [47]. V následující sekci uvádím přehled důležitých kategorií testů.

1.4.1 Dělení podle rozsahu

Dělení podle rozsahu řadí testy do 3 kategorií. Jedná se o nejčastěji používaný druh dělení při funkčním testování a obsahuje následující kategorie [48]

- **Jednotkové (unit) testy:** zaměřené na jednu třídu, popřípadě metodu. Nejmenší z hlediska rozsahu, nejrychlejší na provedení a implementaci. Jsou však typicky náchylné na změny implementace. Z hlediska počtu by jich mělo být nejvíce.
- **Integrační testy:** zaměřují se na testování propojení více tříd. Jsou časově náročnější na implementaci i provedení než jednotkové testy.
- **Systémové testy:** nejrozsáhlejší testy z hlediska rozsahu, jejich implementace je časově náročná a provedení trvá dlouho. Nejsou typicky náchylné na změny implementace. [48]

1.4.2 Dělení podle znalosti implementace

- **Black box:** testy, které se provádějí oproti rozhraní. Nezajímá nás konkrétní implementace a díky tomu není nutné je často upravovat.
- **Gray box:** při testování známe použité algoritmy, ne však jejich přesnou implementaci.
- **White box:** jedná se o strukturální testy, při kterých přihlížíme k implementaci. Zpravidla jsou křehčí a změny v implementaci znamenají, že je musíme upravit také. [48]

1.4.3 Dělení podle způsobu provedení

- **Statické:** tyto testy nevyžadují spuštění kódu. Pomáhají odhalit například anti-patterny[13], vracení správných datových typů, odchytávání výjimek, nebo zranitelné závislosti.
- **Dynamické:** jedná se o testy prováděné na běžící aplikaci. [48]

[13] Jedná se o způsob, jak NEřešit problém.

1.4.4 Dělení podle způsobu vyhodnocování

- **Automatické:** testy kontrolované programem. Většinou využívané pro funkční testování, ale pomocí různých nástrojů se dají automatizovat téměř všechny testy z kategorie 1.4.5. Automatizování některých testů však může být nákladné a nemusí poskytovat hlubší analýzu výsledků (například testování UI).
- **Manuální:** tyto testy jsou prováděny a vyhodnocovány manuálně člověkem, nebo testerem. Většinou se jedná například o kvalitativní testování uživatelského rozhraní, nebo akceptační testování zákazníkem. Některé druhy testů, zejména funkční, není však vhodné provádět manuálně z důvodu náchylnosti na chyby a nemožnosti je jednoduše opakovat. [48]

1.4.5 Dělení podle způsobu zaměření – FURPS

- **Functionality:** mezi vývojáři asi nejznámější kategorie testování. Účelem tohoto typu testování je zaručit funkčnost aplikace, převážně v nejvíce důležitých částech aplikace. Tyto testy je vhodné tvořit již při implementaci a zautomatizovat je.
- **Usability:** jedná se o testy použitelnosti uživatelského rozhraní. Je žádoucí tyto testy provést na prototypu aplikace, nebo alespoň před nasazením aplikace do provozu. Provádí se manuálně, formou kvalitativních a kvantitativních metod. Zkoumají, jestli je pro cílové uživatele rozhraní intuitivní a pomáhá jim dosáhnout jejich cíle efektivně.
- **Reliability:** testování spolehlivosti má mnoho podob. Může se zde posuzovat například doba zotavení se z chyb, ale můžeme sem zařadit například i regresní testy.
- **Performance:** výkonnostní testy určují, zdali aplikace reaguje v rozumném čase a zdali dokáže v případě nutnosti dobře škálovat.
- **Supportability:** tyto testy mohou kontrolovat, zdali bude aplikaci snadné udržovat a opravovat. Jedná se tedy například o statickou analýzu antipatternů, nebo o kontrolu vytváření logů, nebo jiného druhu zaznamenání chyb a dodatečných dat o ní [49].

Toto základní dělení se dá doplnit o další kategorie, jako je například testování bezpečnosti nebo testování kompatibility. Každopádně jsou zde ještě dvě kategorie testů, které bych chtěl zvlášť zmínit.

- **Smoke testy:** jedná se o velmi rychlé testy zkoušející dostupnost různých částí aplikace. Zpravidla se používají pro otestování jednotlivých API endpointů, zdali všechny vrací příslušný HTTP kód [50].

- **Akceptační testy:** jedná se o testy prováděné zákazníkem na implementovanou funkcionalitu. Tyto testy ověřují, že aplikace splňuje všechny funkční požadavky, které zákazník vydefinoval [51].

1.4.6 Závislosti při testování

Při psaní testu se často naráží na situaci, kdy je nutné nahradit závislosti testované funkcionality něčím, co má deterministické a předem dané chování nezávislé na prostředí. Často se jedná o funkce související s databází nebo přístupy k API třetí strany. Tyto závislosti se při testování nahrazují objekty, které mají předdefinované chování pro provedení testu [52]. Takovému objektu se obecně říká *mock*, ale správně se tyto objekty rozlišují na:

- **Stub:** který vrací odpověď pro očekávané volání a při neočekávaném volání nedělá nic, nebo vrací NULL,
- **Mock:** jedná se o stub, který navíc umožňuje ověřit, že volání bylo provedeno dle očekávání.

Díky těmto objektům je možné v testech vytvořit rychlou a deterministickou náhradu závislostí, které jsou za běžného běhu programu reprezentovány plnou implementací a vkládány frameworkem přes dependency injection 1.2.2.1.

1.4.7 Pravidla pro psaní testů

Při implementaci funkčních testů by se vývojář měl řídit následujícími zásadami. Tyto zásady zahrnují především FIRST [53], což je zkratka pro:

- **Fast:** testy by měly být rychlé. To umožní vývojářům je rychle spustit a ověřit funkcionalitu bez dlouhých prodlev.
- **Independent:** testy by neměly být závislé na výsledku jiných testů.
- **Repeatable:** testy musí být deterministické.
- **Self-Validating:** test samotný musí vyhodnotit, zdali byl úspěšný.
- **Thorough:** testy by měly pokrýt všechny ideální případy, snažit se pokrýt veškeré mezní případy, nesprávné vstupy a vysoké hodnoty.

Kromě pravidla FIRST by se měly testy řídit jistým flow, které určuje, fáze, které by měl každý test obsahovat. Nejpopulárnějším metodou v tomto ohledu je metoda 3A [54]. Tato metoda říká, že každý test by měl obsahovat následující 3 fáze:

- **Arrange:** v této fázi jsou připravena veškerá data a závislosti.
- **Act:** tato fáze testu provádí samotnou testovací akci.

- **Assert:** v této fázi se kontrolují výsledky předchozí fáze s očekávanými výsledky. Popřípadě se zde může také kontrolovat neúspěšné provedení akce – například při testování nesprávných vstupů.

1.5 Způsoby řízení vývoje

V této sekci se věnuji metodikám vývoje webových aplikací a softwaru obecně. Metodika v sobě typicky zahrnuje určité postupy a procesy, kterými se řídí projektový tým při návrhu, realizaci a provozu projektu [55]. Budu se zde věnovat převážně dvěma nejpopulárnějším metodikám – vodopádu a iterativnímu vývoji.

1.5.1 Vodopád

Jedná se o jednodušší metodiku vývoje, která je strukturována do navazujících fází. Tato metodika se hodí převážně na projekty menšího rozsahu. Je také vhodná v případě, že se požadavky na aplikaci příliš nemění a je tak možné držet se návrhu [55]. Tato metodika je většinou dělena do následujících fází:

1. **Analýza:** zjištění požadavků na software.
2. **Návrh:** návrhy uživatelského rozhraní a implementace.
3. **Implementace:** zrealizování předešlých návrhů.
4. **Testování:** testování implementace a uživatelského rozhraní.
5. **Nasazení:** zprovoznění aplikace, sběr zpětné vazby.
6. **Údržba:** průběžné opravy chyb, implementace nových funkcionalit.

1.5.2 Iterativní vývoj

Tato metodika vývoje využívá rychlých, většinou 2týdenních sprintů, neboli iterací. V každé iteraci poté probíhá analýa, návrh, implementace a testování. Tyto fáze jsou poté opakovány, dokud není dodán kompletní produkt. Po každé iteraci vzniká nová verze aplikace, která je spustitelná a nasazená [56].

1.5.2.1 SCRUM

V dnešní době nejpopulárnější iterativní metodikou je SCRUM [57]. Při použití této metodiky vývoj probíhá v krátkých, většinou 2–4týdenních *sprintech*. Úkoly do těchto sprintů jsou plánovány z *backlogu*, který obsahuje prioritizovaný seznam požadavků a chyb, které je potřeba vyřešit. Dále se klade důraz na pravidelnou komunikaci formou *stand-upů*, které se pořádají typicky každý den v předem stanovený čas. SCRUM pro své fungování vyžaduje v týmu tzv.

Product Ownera a *Scrum Mastera*. Kromě těchto dvou rolí jsou samosebou přítomni i vývojáři, kteří mají předem definované odpovědnosti. Product Owner zastupuje zájmy zákazníka a zajišťuje, že požadavky kladené na software jsou splněny a hlavně komunikovány směrem k týmu. Scrum Master zastupuje roli project managera. Zajišťuje, že jsou dodržovány nejlepší praktiky SCRUMu, plánuje práci do jednotlivých sprintů, týmová setkání, projektové retrospektivy a řeší také případné konflikty, které v týmu nastanou [58].

1.6 Nasazení

Nasazení označuje proces, při kterém je kód nové verze webové aplikace nahrán na server a jsou provedeny další potřebné úkony pro to, aby nová verze aplikace mohla být spuštěna pro uživatele. Zpravidla je při nasazování aplikace dbán také důraz na zachování dat v databázi, popřípadě nahraných uživatelských souborů [59].

1.6.1 Prostředí

V praxi je potřeba novou verzi aplikace testovat také v podmínkách, které jsou co nejvíce podobné těm reálným. K tomu je nutné mít zprovozněno více prostředí, do kterých aplikaci nasazujeme postupně. Pokud by testování probíhalo na produkčním prostředí a produkčních datech, mohlo by mít katastrofální následky [52].

Pojem prostředí označuje izolovanou instanci aplikace s potřebnými závislostmi. Některé závislosti mohou však být v určitém prostředí nahrazeny, aby nedocházelo k nežádoucím vedlejším efektům, kterým může být odeslání emailu, nebo změna dat v propojených informačních systémech [60].

Příprava takového prostředí a jeho provoz vždy něco stojí. Je totiž žádoucí, aby testovací prostředí aplikace bylo co nejvíce podobné produkčnímu, aby při testování nedocházelo k neočekávanému chování způsobeného infrastrukturou, na které aplikace běží. Zpravidla se provozují alespoň 2 prostředí (nepočítaje vývojové), ale v závislosti na potřebách aplikace a jejich uživatelů jich může být i více. Minimálně bychom měli mít k dispozici produkční a testovací prostředí, ale pro přehled uvádím běžně používané typy prostředí [52]:

- **Produkční (PROD):** označuje prostředí, které používají reální uživatelé pro jejich potřeby. Není vhodné na něm testovat nové funkcionality a při manipulaci s ním musíme být opatrní.
- **Preprodukční (UAT / PREPROD):** označuje prostředí, které používají testeři, popřípadě zákazník, k otestování implementovaných funkcionalit a jejich akceptaci. Je velmi podobné produkčnímu prostředí.
- **Testovací (TEST):** označuje prostředí, které používají testeři dodavatele k otestování implementovaných funkcionalit.

- **Vývojové (DEV):** označuje prostředí, které používají vývojáři k vývoji aplikace. Je dobré mít toto prostředí snadno replikovatelné, aby bylo možné jej lehce zprovoznit a mít nastavenou aplikaci tak, aby poskytovala dodatečná data pro debugování.

1.6.2 Databáze a jiné závislosti

Při nasazování nové verze aplikace se může stát, že je potřeba udělat změny v databázi. Může se jednat o přidání sloupce, nebo editaci dat. Tyto změny je dobré také verzovat a kombinovat verzování těchto změn s pravidelnými zálohami databáze. Změny v databázích se obvykle nazývají *migrace*, které obsahují SQL (nebo jiný) kód potřebný pro provedení změn. Typicky migrace obsahují i kód, který změny provedené touto migrací vrátí zpět [52].

Další závislosti, na které je dobré myslet, jsou nahrané uživatelské soubory. Ty by se při nasazení aplikace neměly mazat, ale je dobré je nanejvýš přesunout na jiné místo. Umožňuje to opět snadný návrat k předešlé verzi v případě kritického problému.

1.6.3 Sémantické verzování

Jako poslední zmíním sémantické verzování. Tento způsob verzování je všeobecně uznávaný a používaný způsob, jak označovat jednotlivé verze aplikace podle toho, co přinášejí a zdali se v nové verzi vyskytují změny, které nemusí fungovat se staršími nástroji.

Při použití sémantického verzování, je každá verze programu zapsána ve formátu **major.minor.patch**, kde jednotlivé pojmy značí následující

major: označuje verzi aplikace, která není zpětně kompatibilní s předchozí verzí.

minor: označuje verzi aplikace, která je zpětně kompatibilní s předchozí verzí a přidává funkcionalitu.

patch: označuje verzi, ve které jsou opraveny chyby při zachování zpětné kompatibility. Funkčnosti aplikace se zpravidla nemění.

Příklad sémantické verze je: 1.12.34. Dále je dáno, že major verze nesmí být 0 – toto je vyhrazeno pro počáteční rozvoj aplikace před prvním vydáním [52].

1.7 Provoz a monitorování

Po nasazení aplikace přichází na řadu její provoz, údržba a monitorování. Monitorování zajišťuje včasné upozorněni, pokud aplikace nefunguje, dochází místo na disku, nebo nestačí výkon [61]. V praxi mohou aplikace monitorovat hned několik metrik, mezi ně patří převážně:

- Výjimky a chyby
- Dostupnost aplikace
- Bezpečnost
- Výkon, doba odezvy

Na monitorování těchto metrik je dostupné množství placených i neplacených nástrojů. Základním řešením každé aplikace je však tvorba logů, které mohou sloužit pro potřeby ladění v případě výjimek, nebo pro účely kontroly bezpečnosti [61]. Pro potřeby tohoto textu bude však nejdůležitější monitorování výjimek a chyb, které se v aplikaci vyskytnou.

1.7.1 Monitorování výjimek a chyb

Pro tento druh monitorování můžeme použít několik přístupů. Jedním z běžně používaných jsou logy ukládané na serveru v případě výjimek. Dalším způsobem mohou být služby, které nám k výjimkám zobrazují ještě podpůrná data a notifikují správce systému, když k výjimce dojde. V praxi se běžně používá nástroj Sentry[14] [62].

Sentry je dostupná ve variantě SaaS[15], ale také jako self-hosted služba. Podporuje monitorování neošetřených výjimek a poskytuje detailní informace o místě v kódu, kde se výjimka objevila. Kromě toho zaznamenává také mnoho dalších informací týkajících se prostředí, ve kterém došlo k chybě [63]. Nabízí také možnost integrace s komunikační platformou Slack[16], což poskytuje okamžité upozornění, když dojde k chybě v produkčním prostředí.

1.8 Technologie webových aplikací

Pro vývoj webových aplikací dnes existuje množství technologií, které se každým dnem mění. Existují také dva hlavní přístupy k tvorbě webových aplikací, které většinou tyto technologie podporují a které přinášejí určité výhody i nevýhody.

1.8.1 Aplikace obsahující všechno

Jedná se o klasický a starší způsob pro vývoj aplikací. Je podporován většinou moderních webových frameworků a ty se snaží, aby byl tento způsob pohodlný a dobře provázaný s funkcionalitami frameworku. Při použití tohoto způsobu je komunikace backendu a frontendu zajištěna předáváním dat přímo přes metody

[14] `www.sentry.io`
[15] Software as a Service
[16] `https://slack.com/`

frameworku, většinou do šablon. Tyto šablony potom jen definují, jak data zobrazit, popřípadě mohou obsahovat formuláře, na jejich změny.

Hlavní výhodou těchto aplikací je rychlost a jednoduchost vývoje. Na druhé straně svázání frontendu s aplikací může přinést problémy, když se aplikace rozroste [6].

1.8.2 Aplikace rozdělená na REST API a frontend

Tento přístup rozděluje aplikaci do dvou částí. V jedné části běží frontend, který se stará o renderování HTML, komunikaci s klientem atp. Tento frontend může mít podobu klasického client-side frontendu, nebo komplexnějšího serveru, který může zajišťovat i server-side rendering. Na straně druhé běží backendový server v podobě REST API, díky kterému může frontend získávat a upravovat data.

Samotné získání a úpravy dat jsou možné díky využití AJAX. AJAX (Asynchronous JavaScript and XML) je mechanismus, který umožňuje JavaScriptu získávat data i po načtení stránky. Tento mechanismus je dále podpořen existencí FetchAPI v prohlížeči [64].

1.8.2.1 REST API

V předchozí části zmiňuji pojem REST API, který zaslouží také samostatné vysvětlení. API, neboli *Application Programming Interface* je sada definic a protokolů pro výstavbu aplikací. Definuje, jaká data jsou vyžadována od klienta a specifikuje formát dat vracených serverem. API je typicky implementováno právě ve webových službách a slouží pro manipulaci a získávání dat ze serveru pro následné zpracování nebo zobrazení na klientské straně [65].

REST, neboli *REpresentational State Transfer* je architektonicky styl pro vývoj webových aplikací založený Royem Fieldingem v jeho doktorské disertaci [66]. Je důležité říci, že REST je sada architektonických omezení, nejedná se o protokol, nebo standard. Díky tomu mohou vývojáři implementovat REST několika různými způsoby. Systémy sestavené podle této architektury poté poskytují klientovi data prostřednictvím HTTP a to v mnoha různých formátech, jako například JSON, XML, HTML a další. V dnešní době je nejvíce populární formát JSON. Ve spojení s API se poté často mluví o tzv. *RESTful API*, což značí API, sestavené podle těchto architektonických omezení:

- Architektura klient-server s požadavky zasílanými přes HTTP protokol.
- Bezstavová komunikace – žádné klientské informace se na serveru neukládají.
- Jednotné rozhraní mezi částmi systému, aby mohly být informace zasílány standardizovanou formou. To vyžaduje aby:

1. jednotlivé zdroje na serveru byly identifikovatelné a oddělené od jejich reprezentace zasílané klientovi,
2. zdroje byly manipulovatelné skrze reprezentace, které jsou zasílány klientovi, jelikož obsahují všechny potřebné informace,
3. výstižné odpovědi vracené serverem, aby měl klient dostatečné informace jak s informacemi naložit,
4. odpovědi ze serveru obsahují hypertext, takže klient má informace o dalších relevantních datech, nebo akcích dostupných pro daný zdroj.

- Vrstvená architektura, která obstarává například vyrovnávání zátěže, nebo zabezpečení je pro klienta neviditelná.

Pro RESTful API tedy platí, že klient se dotazuje na data pomocí HTTP požadavků, většinou ve formátu JSON a server poté zasílá odpověď ve stejném formátu. Díky tomu je možné dobře oddělit klientskou část aplikace a umožňuje to větší flexibilitu při budoucích změnách. Dále díky oddělení mohou vznikat další podpůrné aplikace, které s tímto API mohou komunikovat a poskytovat uživatelům přizpůsobené funkcionality pro jejich specifické potřeby [65].

Hlavní výhodou tohoto přístupu je oddělení frontendu od backendu, čímž se otevírá možnost vyměnit frontendové technologie bez potřeby změny backendu v budoucnu. Další výhodou je možnost volby vhodnější technologie, se kterou bude stavba UI pohodlnější a rychlejší. Na druhou stranu se jedná o časově náročnější variantu na vývoj oproti klasické aplikaci. Oba dva způsobu vývoje aplikací se stále hojně používají, ale většina společností se přiklání k použití právě druhého přístupu, tedy REST API a separátní klientské aplikace [67]. V předchozí části jsem použil pojmy *client-side rendering* a *server-side rendering*. Jedná se o režimy vykreslování HTML, v další části textu se věnuji jejich krátkému popisu.

1.8.3 Client-side rendering

Tento druh renderování nepotřebuje další server. Stačí nám poskytnout soubor, který stáhne JavaScript a ten následně vykreslí požadované HTML. Výhodou je, že nepotřebuje separátní server, ale má vyšší latenci [68].

1.8.4 Server-side (universal) rendering

Tento druh renderování potřebuje server. Typicky se jedná o Node.js nebo alternativu. Tento server potom zajišťuje prvotní vykreslení HTML, které potom posílá klientovi. Na klientské části poté typicky probíhá proces tzv. *hydratace*, což je proces, který v zásadě obohatí vygenerovanou statickou stránku JavaScriptem, který byl zpracován na serveru. Díky tomu je poté zajištěná interaktivita aplikace i na její klientské části [69]. Výhodou tohoto přístupu

je jeho rychlost a nízká latence. Mezi nevýhody se typicky řadí složitější vývoj a nákladnější provoz, kvůli potřebě dalšího serveru [69]. Pro potřeby nové aplikace SOS jsme kvůli nízké latenci volili právě tento způsob provozu.

Kapitola 2

Analýza stávajícího řešení

V následující kapitole se věnuji rozboru aktuálního systému SOS, který se používá na FIT ČVUT a GJGJ. V první části zmiňuji vývoj v čase. Poté analyzuji funkční požadavky, které se týkají sekcí odevzdání a hodnocení. Dále analyzuji technické řešení aplikace a nevýhody, které se k tomuto řešení vážou.

Systém SOS slouží pro odevzdávání rozsáhlejších projektů, které vyžadují spolupráci týmu. Je vhodný také pro rozdělení projektu do kontrolních bodů, které jsou individuálně hodnoceny. Typickými zástupci úloh, které se přes SOS odevzdávají jsou semestrální projekty, společné týmové projekty, nebo ročníkové práce.

Aktuálně se systém využívá na Fakultě informačních technologií ČVUT a to v bakalářských předmětech BI-SP1, BI-SP2 a BI-SWI na správu společných týmových projektů, magisterském NI-NUR pro semestrální práce a dále také na Gymnáziu Jiřího Gutha-Jarkovského pro správu ročníkových prací.

2.1 Historický vývoj

2.1.1 První verze

První verze systému vznikla z bakalářské práce Ing. Tomáše Pavlůska pod vedením Ing. Jiřího Hunky. Původní systém vycházel z již existujícího systému *NURIS* [70]. Ing. Tomáš Pavlůsek tedy částečně zakomponoval potřeby uživatelů ze systému NURIS právě do SOS.

V rámci bakalářské práce se autor práce věnoval analýze požadavků různých předmětů, které by SOS mohly využívat. Dále se věnoval návrhu architektury i následné implementaci a nasazení systému na server. Systém, který autor implementoval, prošel uživatelským testováním s vyučujícími i studenty a byl nasazen do provozu v předmětu NI-NUR [71]. Konkrétní funkcionality a nedostatky popisuje autor v jeho bakalářské práci [72].

2.1.2 Druhá verze

Po nasazení do předmětu NI-NUR se k systému SOS dostal Bc. Max Hejda. Ten systém rozšířil na Gymnázium Jiřího Gutha-Jarkovského – GJGJ. Na gymnáziu systém slouží ke správě ročníkových prací. V rámci rozšíření na GJGJ bylo učiněno několik rozsáhlejších úprav v systému. Z některých těchto úprav každopádně benefituje i FIT ČVUT, což mimo jiné prokazuje také flexibilitu této verze systému.

Mezi jednu z největších úprav, využitelnou především pro gymnázium, patřila možnost přihlášení přes Google OAuth, který se na gymnáziu využívá. Dalším, pro gymnázium specifickým požadavkem, byla také možnost importu dat ze systému Bakaláři[1]. Toho bylo docíleno zavedením jednotného rozhraní pro import data – *DataProvider*. Další úpravy, které jsou využitelné i na půdě FIT ČVUT, jsou především následující:

- úpravy logiky a nastavení tak, aby bylo možné propojit zadání několik předmětů (ty reprezentují ročníkové práce),
- kategorizace notifikací a konfigurovatelný systém emailových notifikací,
- zakomponování plánovače úloh *Celery* [2], který slouží pro výše zmíněné emailové notifikace.

Tato verze systém taktéž prošla uživatelským testování a byla úspěšně nasazena na GJGJ [73].

2.1.3 Třetí verze

Na verzi, kterou Bc. Max Hejda nasadil na GJGJ, poté navázal opět Ing. Tomáš Pavlůsek jeho diplomovou prací. Cílem práce bylo rozšířit systém do předmětů BI-SP1, BI-SP2 a BI-SWI, zvýšit pokrytí systému automatickými testy a také implementovat funkcionality, které nestihl v rámci bakalářské práce.

V této diplomové práci vzniklo několik úprav systému, které byly potřebné pro nasazení na výše zmíněné předměty. Tyto úpravy zahrnovali především:

- rozšíření možností konfigurace předmětu i na úroveň týmů, aby bylo možno měnit kontrolní body,
- možnost přerozdělení bodů mezi jednotlivé členy týmu,
- možnost aktualizovat seznam studentů i po vytvoření předmětu v SOS,
- možnost exportu hodnocení do FIT Klasifikace pomocí API [74]
- možnost exportu hodnocení do KOS pomocí starého webového rozhraní – tento bod již funkční není, viz. sekce 2.4.1,

[1] `https://www.bakalari.cz`
[2] `https://docs.celeryq.dev`

V rámci diplomové práci byly splněny všechny požadavky a systém byl nasazen v letním semestru 2022/2023 do provozu na předměty BI-SP1 a BI-SWI. V tomto semestru jsem se poprvé se systémem setkal také já a kolegové z mého týmu a na této verzi jsme začali zpracovávat první analýzy a také identifikovat možná místa pro rozšíření a zlepšení systému.

V diplomové práci autor zmiňuje, že poslední verze, která byla v rámci diplomové práce nasazena byla verze s tagem 1.4.7 v GitLab repozitáři projektu SOS.

2.1.4 Aktuální verze

Aktuální verze se velmi podobá výše zmíněné třetí verzi, která vzešla z diplomové práce Ing. Tomáše Pavlůska. V rámci této verze bylo učiněno několik oprav, převážně se jednalo o drobné nedostatky, které nejvíce trápili některé uživatele. Tyto úpravy prováděli oba autoři závěrečných prací. Mezi provedené úpravy patří především:

- přidání časové osy vývoje systému, včetně spolupracovníků,
- oprava importu studentů Erasmu,
- drobné opravy jiných chyb, překlepů.

Z této verze vychází tato práce. Zároveň všechny požadavky a nedostatky, jsou rovněž analyzovány na této verzi aplikace.

2.2 Funkcionality aktuálního řešení

Pro potřeby předmětů využívajících SOS, jsou v systému implementovány různé funkcionality a konfigurace. V rámci této práce se budu soustředit na funkcionality, které se týkají kontrolních bodů, jejich odevzdávání a hodnocení. Kromě toho se ale také dotknu týmů a projektů, jelikož jsou pro tyto potřeby také důležité.

Nejprve je nutné zadefinovat pojmy, které se v této části vyskytnou a kterých se dotkne tato práce

- **Projekt:** označuje celek, na kterém pracuje tým, popřípadě jednotlivec. Nedílnou součástí projektu je popis zadání, přiřazení uživatelé – jedná se o učitele, jeho asistenty a řešící tým.
- **Kontrolní bod:** představuje nejmenší odevzdatelný celek. Kontrolní bod je dílčí část projektu a obsahuje hlavně vlastní zadání, které má být v rámci kontrolního bodu odevzdáno. Dále obsahuje termín odevzdání a vymezení bodového ohodnocení.

- **Odevzdání:** představuje studentské řešení kontrolního bodu. V rámci odevzdání jsou uloženy také informace o přerozdělení bodů a přiložené soubory.
- **Hodnocení:** značí bodové ohodnocení v rámci kontrolního bodu. Obsahuje zpětnou vazbu pro studenty, samotné bodové ohodnocení a případně přiložené soubory.

2.2.1 Funkční požadavky

V následující části se věnuji popisu funkčních požadavků, které souvisí s touto prací. Tyto funkční požadavky jsou sepsány tak, jak je podporuje aktuální systém. Udávají však rámcovou představu o funkcionalitách, kterým se bude tato práce věnovat. V pozdějších kapitolách tyto funkční požadavky poslouží převážně jako zdroj dat pro pro dílčí analýzy. V těchto analýzách budou upraveny, popřípadě přidány další funkční požadavky, podle kterých bude vznikat implementovaná sekce v novém systému.

FP01 – Kontrolní body

Důležitost: Vysoká

Náročnost: Střední

Vyučující může nastavit, na jaké části je projekt rozdělen. Tyto části se nazývají kontrolní body a mají vlastní zadání a termín odevzdání. Tyto kontrolní body vyučující může posléze upravit, popřípadě smazat. Smazání je podmíněno neexistencí odevzdaného řešení.

FP02 – Odevzdávání řešení

Důležitost: Vysoká

Náročnost: Střední

Studenti mohou v iteracích odevzdávat svá vypracovaná řešení včetně příloh v podobě nahraných souborů a webových odkazů.

FP03 – Hodnocení

Důležitost: Vysoká

Náročnost: Střední

Vyučující může v aplikaci provádět hodnocení odevzdaných řešení pro iteraci a celou práci. Zároveň aplikace umožní hodnocení editovat.

FP04 – Přerozdělení bodů

Důležitost: Vysoká

Náročnost: Střední

V konfiguraci předmětu je možné povolit a nastavit maximální procento přerozdělení bodů mezi jednotlivými členy týmu. Přerozdělení pak může vyučující povolit či zakázat u jednotlivých kontrolních bodů. Při odevzdání řešení studenti navrhnou 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 v rámci FP03 upravit.

2.3 Technologie aktuálního řešení

V následujících části se věnuji popisu všech technologií, které byly použity při vývoji a rozvoji systému SOS. Co se architektury týče, tak systém SOS je vícestránková monolitická webová aplikace [72]. Monolitické aplikace více popisuji v sekci 1.2.1.1. Vzhledem k počtu použitých technologií znázorňuje aplikační technologie i obrázek 2.1

2.3.1 Django

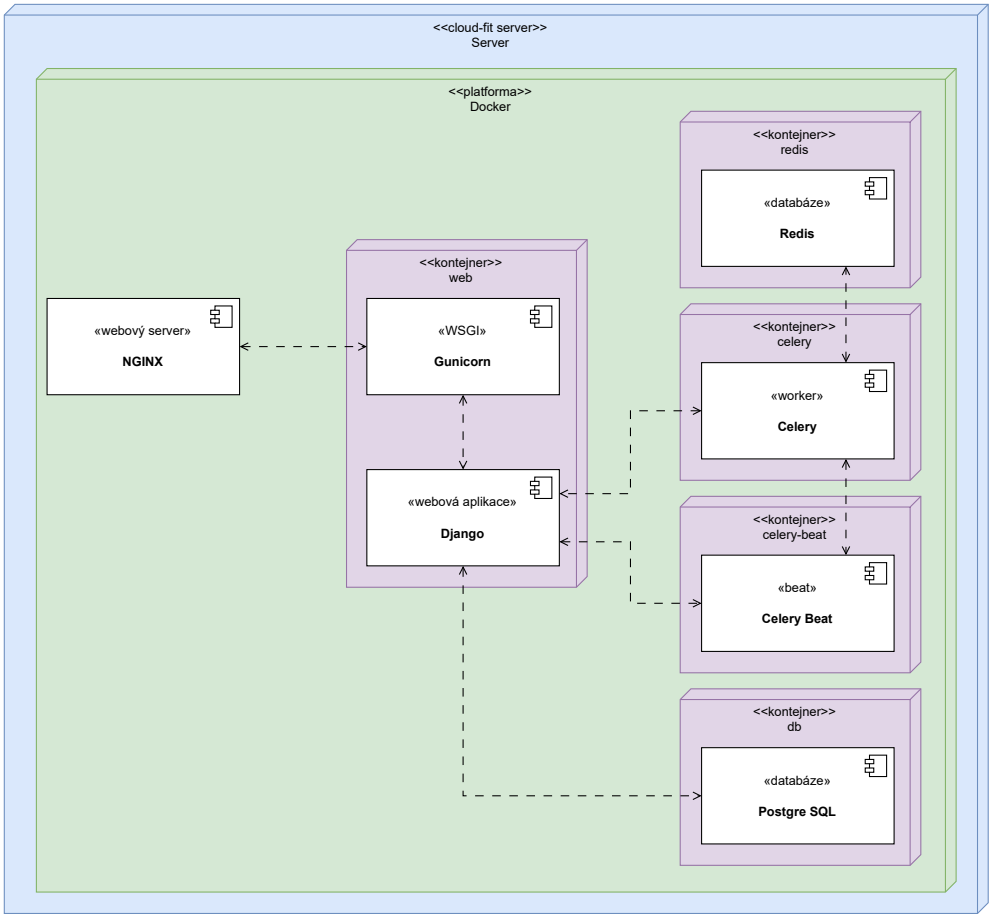
Aktuální webová aplikace je implementována ve frameworku Django. Autor aktuálního řešení si Django zvolil z důvodu, že už měl s tímto frameworkem zkušenosti. V době psaní systému autor zvažoval také využití JavaScriptového frameworku pro potřeby frontendu. Nakonec se však rozhodl nevyužít pro frontend JavaScriptové řešení, ale postavit systém pouze na frameworku Django. Zdůraznil však, že ačkoliv by byly tyto technologie vhodné, tak nebylo časově efektivní je kvůli nedostatku zkušeností využít [72].

Django je open-source webový framework pro jazyk Python. Jedná se o zástupce architektury MVT, které se věnuji v sekci 1.2.3. Zároveň obsahuje spoustu nástrojů a funkcí už v základu a není nutné je doinstalovávat.

Důležitou částí Django frameworku je i snadno konfigurovatelná ochrana proti mnoha častým útokům a zranitelnostem, popsaným v sekci 1.3.1. Podrobněji popisuje implementaci těchto ochran Ing. Tomáš Pavlůsek v jeho bakalářské práci [72, kapitola 3.5].

Frontend

Součástí Django aplikace je také frontend. Ten je psán v Django templates s využitím podpůrných knihoven. Základní vzhled portálu určuje šablona AdminLTE. Jedná se o šablonu pro administrační weby postavenou na populárním CSS frameworku Bootstrap. Obsahuje různé komponenty, jako například



Obrázek 2.1 Architektura aktuálního řešení, zjednodušeno z [71]

formuláře, rozbalovací nabídky, grafy a celou řadu dalších komponent. Umožňuje tak rychle postavit poměrně dobře vypadající administrační stránky bez nutnosti hluboké znalosti CSS a JavaScriptu [72].

2.3.2 PostgreSQL

Autor zvolil pro systém objektově-relační databázi PostgreSQL, jelikož dobře spolupracuje s využitým frameworkem. PostgreSQL je pokročilá a výkonná open-source databáze, která mimo ukládání relačních dat umí pracovat i s nerelačními daty ve formátu JSON. Jak už bylo zmíněno, tak se jedná o objektově-relační databázi, umožňuje ukládat seznamy, množiny nebo i vnořené objekty [75].

2.3.3 NGINX

Autor systému zvolil jako webový server NGINX. Úkolem webového serveru je zpracovat a předat HTTP požadavek příslušné aplikaci a to buď přes port, nebo unixový socket (toto využívá například Python s Gunicornem).

NGINX se v kontextu této aplikace chová jako vstupní bod pro všechny uživatelské požadavky, které potom dále přeposílá skrze Gunicorn do aplikace.

2.3.4 Gunicorn

Jedná se o opensource implementaci WSGI (Web Service Gateway Interface) rozhraní, které umožňuje komunikaci webového serveru s aplikacemi psanými v Pythonu. Více informací o WSGI je možné zjistit z oficiálního PEP [76].

2.3.5 Celery

Pro potřeby emailových notifikací a jiných periodických úloh je v SOS použitý nástroj Celery. Celery k fungování využívá dva procesy, které běží v aplikačních kontejnerech.

- **Beat:** producent zpráv pro worker proces.
- **Worker:** konzument zpráv vytvořených beat procesem.

Nástroj Celery vyžaduje systém pro přenos zpráv pro jeho správné fungování. V systému SOS je pro tyto potřeby použit Redis [77].

2.3.6 Docker

Kvůli rozšíření aplikace o Celery a Redis, vznikla potřeba usnadnění nasazení aplikace a také zprovoznění aplikace při lokálním vývoji. K tomu se Bc. Max Hejda rozhodl použít Docker v kombinaci s Docker Compose.

Docker je virtualizační platforma, na které je možné provozovat OCI (Open Container Initiative) kontejnery. Kontejner je izolované prostředí se všemi potřebnými závislostmi pro provozování aplikace [78].

Docker Compose je nástroj pro orchestraci[3] několika kontejnerů. Konfigurace probíhá ve speciálním YAML souboru, ve kterém je možné definovat jednotlivé názvy služeb, proměnné a porty, které kontejner vystavuje do hostujícího prostředí. Dále tato služba vytváří vlastní síť, po které komunikují kontejnery a umožňuje se na jiný odkazovat pomocí jména služby. Tím se usnadňuje komunikace mezi jednotlivými částmi aplikace [79].

2.3.7 GitLab

Jedná se o systém na správu kódu, který je hojně využíván na FIT. Kromě verzování kódu poskytuje GitLab množství dalších nástrojů, které jsou využitelné při vývoji [80]. Mezi ně patří například:

- diskuzní funkce pro pohodlné diskuze a code reviews v týmu, které zvyšují kvalitu kódu,
- registr kontejnerů pro nativní hostování sestavených verzí aplikací pro pohodlné nasazení na server,
- nástroje pro řízení rozvoje aplikace, jako GitLab issues.

Díky jeho podpoře na FIT se jedná o snadno obhajitelnou volbu pro verzování zdrojového kódu pro potřeby vývoje systému.

2.3.8 Redmine

Jedná se o systém pro správu úkolů. Je používán hlavně v předmětech BI-SP1 a BI-SP2 k zadávání úkolů a tvorbě přehledů o stráveném času. Zároveň slouží jako hlavní zdroj informací a materiálů, které jsou v průběhu těchto předmětů tvořeny do tzv. *wiki* projektu SOS. Záznam času poté studenti realizují skrze aplikaci Toggl Track[4], která je synchronizována pomocí externího nástroje *Timer2Ticket*[5] s Redminem.

2.3.9 Sentry

Pro potřeby monitorování chyb v produkčním a testovacím prostředí používá aktuální SOS fakultní Sentry[6]. O Sentry jsem více psal v sekci 1.7.1.

Kromě monitorování výjimek na webovém portálu obsahuje projekt i integraci s komunikační platformou Slack, takže každá nová chyba je viditelná v příslušném kanálu.

[3] Řízení a konfigurace
[4] `https://track.toggl.com`
[5] `https://app.timer2ticket.com/`
[6] `https://sentry.ksi.fit.cvut.cz/`

2.3.10 Slack

Pro potřeby rychlé komunikace se při práci na projektu využívá komunikační platforma Slack.

Slack je byznysová aplikace nabízející mnoho funkcí pro komunikaci, jako je chat, skupiny a také hlasové hovory. Podporuje navíc integrace s jinými aplikacemi, mezi které patří například již zmíněná Sentry. Slack nabízí i variantu zdarma, která má několik omezení. Žádné z nich však není tak závažné, že by se musela hledat alternativní platforma [81].

2.4 Integrace s jinými systémy

Kromě výše zmíněných technologií, ze kterých je SOS sestaven, se využívá i několik integrací s ostatními systémy. Tyto integrace slouží především pro získávání dat, nebo přihlašování.

2.4.1 KOS a KOSapi

Pro potřeby FIT ČVUT je v SOSu implementováno propojení s KOSem (Komponentou Studia) pomocí služby KOSapi.

„KOSapi poskytuje aplikační rozhraní (API) v podobě RESTful webových služeb, které zprostředkovává přístup k vybrané části dat v databázi KOS. Odstraňuje nutnost zpracovávání exportů, neustálou duplikaci všech dat a potíže s jejich udržováním. RESTové služby staví na osvědčených konceptech webu jakožto distribuovaného prostředí vzájemně provázaných informací. KOSapi umožňuje a podporuje vznik školních i studentských aplikací, které pro svou činnost vyžadují online aplikační přístup k datům souvisejícím s výukou“ [82].

V SOS se toto API používá aktuálně pouze k získání informací o uživatelích, vyučovaných předmětech a rozvrzích. Slouží především pro garanty předmětů a to pro potřeby založení předmětu v semestru.

Ve své diplomové práci práci Ing. Tomáš Pavlůsek zmiňuje i přání vyučujících o export hodnocení ze systému SOS do KOS. To bohužel nelze realizovat přes KOSapi, jelikož to poskytuje pouze přístup pro čtení. V diplomové práci bylo řešení exportu realizováno obdobně, jako *Moodle2Kos*. Funguje tak, že klasifikaci exportuje do JavaScriptové funkce, kterou posléze uživatel musí zkopírovat a vložit do vývojářské konzole na příslušné obrazovce v KOSu. Tam tato funkce hodnocení hodnocení prakticky „nakliká“ a tím je export hotov. Autor sám popisuje, že řešení nefunguje s novým rozhraním KOS a při každé změně uživatelského rozhraní může přestat fungovat [71, sekce 1.3.7]. Vzhledem k tomu, že staré rozhraní KOSu již není k dispozici, je tato funkcionalita nefunkční. Podle Bc. Maxe Hejdy zde existuje možnost, že by se mohlo zpřístupnit přímo API systému KOS. Podrobnější informace ohledně této možnosti

však v době psaní této práce nemám a nepředpokládám, že by se v dohledné době něco změnilo.

2.4.2 Bakaláři a Datový konektor

Systém Bakaláři je systém pro podporu administrativy a komunikace na základních a středních škol v ČR. Tento systém používá přes 60% procent základních a středních škol [83].

V rámci rozšíření na GJGJ implementoval Bc. Max Hejda připojení pro import dat ze systému Bakaláři přes RESTové rozhraní – Datový konektor. Ve své práci popisuje kostrbatý proces napojení na toto API a jaké kompromisy musel v SOS udělat, aby probíhal import dat podle očekávání [73, sekce 1.6.1].

2.4.3 OAuth 2.0

OAuth 2.0 je standardizovaný protokol pro autorizaci. Soustředí se na jednoduchost pro vývojáře přičemž definuje specifický autorizační proces pro webové, desktopové a mobilní aplikace. Specifikaci vyvíjí IETF[7] skupina. V současné době je ve vývoji i verze OAuth 2.1, která si klade za cíl sjednotit běžná rozšíření OAuth 2.0 pod jedno jméno [84].

Proces definovaný OAuth 2.0 standardem je poměrně jednoduchý. Klientská aplikace (v tomto případě SOS) požádá autorizační server (jinak také OAuth server) o přístup. Uživatel je přesměrován na příslušnou stránku autorizačního serveru, kde se přihlásí a udělí souhlas o přístup k datům. Při udělování souhlasu navíc vidí, která data aplikace žádá a může se tedy rozhodnout přístup neudělit. Autorizační server poté klientské aplikaci předá přístupový token, kterým se prokazuje na resource serveru (server s daty, v tomto případě také SOS).

2.5 Celkový stav a možnosti systému

2.5.1 Sběr požadavků a zpětné vazby uživatelů

Během celého průběhu předmětů BI-SP1 a BI-SP2 jsme sbírali zpětnou vazbu od uživatelů systému s cílem vytvořit si lepší představu o jejich potřebách. Kromě jejich potřeb jsme také zjišťovali, jaké nedostatky systému uživatele nejvíce trápí a se kterými se nejvíce setkávají. Převážná (a cennější) část získaných informací byla získána pomocí *kvalitativních* metod. Vedlejší a převážně orientační byly potom *kvantitativní* metody.

[7] Internet Engineering Task Force

2.5.1.1 Kvalitativní metody

Kvalitativní metody zahrnují přímou interakci, většinou formou rozhovoru, s jednotlivými uživateli systému. Díky tomu, že systém už je nějakou dobu využíván, jsme měli k dispozici několik různých uživatelů, kteří v systému zastávají odlišné role a mají tedy odlišné potřeby.

Pro získání dobré zpětné vazby na systém jsme použili **hloubkový rozhovor**. Hloubkový rozhovor se provádí s reálnými, popřípadě budoucími, uživateli. Tyto rozhovory nám poskytují pohled na to, co uživatelé od systému očekávají a také poskytují vhled do toho, jak uživatelé své problémy řeší [85].

Základem pro tento typ rozhovoru je příprava. Přípravu jsme sestavovali podle doporučení knihy [85] a také vedoucího práce tak, že jsme se zkusili „vžít“ do uživatele a realizovat pro něj typický *use-case*. Při tom jsme zaznamenávali otázky, subjektivní nedostatky, nebo jiné postřehy. Tímto vznikla pro nás potřebná osnova, podle které jsme poté rozhovor s uživatelem vedli.

Kromě připravených otázek a postřehů jsme samozřejmě probrali i postřehy uživatelů, na které narazili při práci se systémem a konzultovali s nimi možná řešení.

Data z těchto rozhovorů poskytly velmi cenné informace a zpětnou vazbu od uživatelů systému a díky tomu jsme získali dobrý podklad pro další funkcionality, které by budoucí systém měl podporovat a také nedostatky, které je potřeba vyřešit.

Poznatky a odpovědi uživatelů z těchto rozhovorů byly zaznamenávány a jsou přiloženy v dokumentu *Rozhovory.pdf*, který je odevzdán spolu s prací. Alternativně je dostupný také ve sdílené složce na fakultním OneDrive.

2.5.1.2 Kvantitativní metody

Kvantitativní metody se na problematiku zaměřují zeširoka. Mají většinou formu dotazníku, který se rozešle uživatelům. Data z takového dotazníku by měla být sbírána tak, aby bylo jednoduché je statisticky vyhodnotit a použít pro potvrzení naší domněnky [86]. Pro sběr zpětné vazby na stávající systém byly využity dva dotazníky.

První dotazník byl pouze o jedné otázce a nesplňoval tedy řádně stanovená kritéria pro dobré vyhodnocení. Nicméně zpětná vazba studentů, ač strohá, nám poskytla nějaká data a velmi obecný pohled na to, co uživatele v systému nejvíce trápí. Data jsou dostupná v souboru *První dotazník.ods* odevzdané s touto prací, nebo na fakultním OneDrive.

Druhý dotazník jsme rozesílali s cílem zjistit, jak moc je pro uživatele snadné porozumět názvosloví systému a jak moc je pro ně složitá navigace. Tento dotazník však nedosáhl dostatečného množství odpovědí a tím pádem jsou data z něj také velmi orientační. Data jsou dostupná v souboru *Druhý dotazník.ods* odevzdaném s touto prací, nebo na fakultním OneDrive.

Data z těchto dotazníků sloužili spíše jako orientační pomůcka, když jsme prioritizovali požadavky, které vyvstali z kvalitativních metod. Dotazníky ne-

byly objektivně vyhodnotitelné a poskytly pouze malý vzorek dat, které byly převážně orientační. Pro úplnost je ale anonymizované uvádím v přiložených ODS souborech, které jsou odevzdány spolu s touto prací, nebo ve sdílené složce na fakultním OneDrive.

2.5.2 Vyhodnocení získaných informací

Z informací získaných výše popsanými metodami jsme získali mnoho možných podnětů, které jsme zhodnotili a případně zprioritizovali pomocí hodnot 1 (nejmenší priorita) až 5 (největší priorita). Ty nejvíce závažné požadavky a nedostatky, související s touto prací, uvádím formou tabulky 2.1. Některé z těchto požadavků byly zaneseny do systému, který aktuálně funguje – tyto požadavky jsou označeny jako hotové. Jeden z požadavku je připravený v merge requestu. V tabulce je tato skutečnost zaznamenána jako MR. První bod, který si za-

Požadavek / Nedostatek	Priorita	Hotovo
Termín kontrolních bodů je vidět až po rozkliknutí	5	Ne
Hodnocení za kontrolní bod je vidět až po rozkliknutí	5	Ne
Možnost nahrát přílohu k hodnocení	5	MR
Nepřehledný seznam odevzdávání	3	Ne
Zamknutí odevzdání pro úpravy	3	Ne
Přidání grafů hodnocení kontrolních bodů	3	Ano
Neužitečnost grafů hodnocení kontrolních bodů	3	Ne

Tabulka 2.1 Tabulka požadavků a nedostatků

slouží komentář, je *Neužitečnost grafů hodnocení kontrolních bodů*. Původně totiž tyto grafy vznikly z požadavků, které jsme sbírali v BI-SP1. Avšak provedení a také některé nevyřešené situace v systému vedli k tomu, že grafy jsou v podstatě nepoužitelné a to především z těchto důvodů:

- Souběh BI-SP1 a BI-SWI způsobuje, že v předmětu BI-SWI jsou uživatelé, kteří splní projekt v BI-SP1 a kvůli tomu jsou v předmětu BI-SWI „navíc“ a tím zkreslují grafy.
- Různé projekty mají různé počty kontrolních bodů. Některé projekty jich mají mnohem více, než jiné a tím potom zvyšují počet grafů, které pro cílového uživatele nenesou podstatnou informační hodnotu.
- Grafy jsou pouze dvoustavové – *Neohodnoceno* a *Ohodnoceno*. Pokud už je odevzdané řešení, které není ohodnoceno tak se informace v grafu nezobrazí.

2.5.3 Celkové zhodnocení stavu aplikace

Systém je aktuálně dobře použitelný a mohou na něm nadále probíhat drobné opravy a úpravy. Nicméně z některých nedostatků se začínají projevovat hlubší problémy a to převážně s technologiemi použitými na vytvoření uživatelského rozhraní. Z rozhovorů a dotazníků plyne, že uživatelské rozhraní a celkový UX je v současné době to, co nejvíce uživatele aktuálního SOS trápí.

2.5.3.1 Uživatelské rozhraní

Jak již bylo zmíněno výše, jedním z největších problému SOSu je uživatelské rozhraní. Aplikace je v současné době navržená jako vícestránková aplikace. To často způsobuje, že formuláře jsou umístěny na separátních stránkách, což vede k častému proklikávání. Toto proklikávání je problém hlavně pro odevzdávání a hodnocení, jelikož jde o často prováděnou akci a tudíž každý proklik navíc způsobuje prodlení a ztrátu kontextu. Toto podotkl v rozhovoru i Ing. Jiří Hunka, který demonstroval proces hodnocení, kdy vyučující musí udělat celkem tři prokliky, aby se dostal na samotné ohodnocení. Toto je znázorněno na ukázce 2.2.

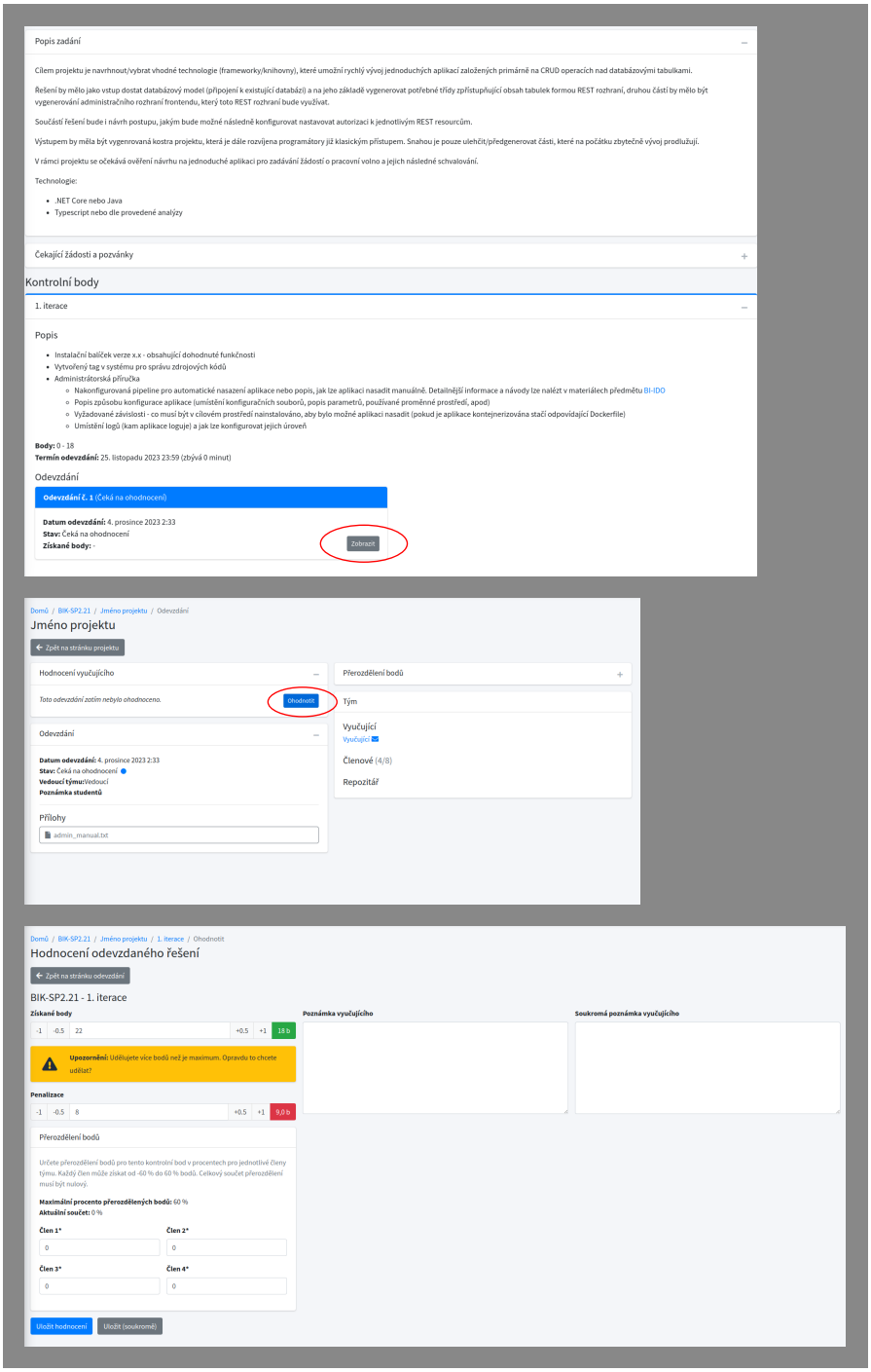
Samotná organizace do více stránek a to převážně u formulářů, je především způsobena technologiemi a tím, že je při použití Django templates výhodné, realizovat formuláře na jiné stránce. Realizace více formulářů na jedné stránce je sice možná, ale je výhodné je oddělit, jelikož obsluha více formulářů z jedné šablony je na úrovni backendu je složitá [87].

2.5.3.2 Technologie

Další problém, který trápí současnou aplikaci je také volba technologií. Na frontendu navíc funguje dnes již poměrně zastaralá knihovna jQuery, která se již při vývoji moderních webových stránek prakticky nepoužívá a ustoupila novějším JavaScriptovým frameworkům, jako je React[8], nebo Vue.js[9].

Dále z našeho průzkumu plyne, že větší část studentů, kteří se s největší pravděpodobností budou podílet na rozvoji portálu SOS[10] se moc nechce učit Django a spíše jsou zvyklí na práci s MVC frameworky, jako je například Symfony, Spring Boot, nebo ASP.NET Core. Studenti FIT ČVUT se mimo jiné během studia setkají právě s jinými frameworky, něž je Django. Například studenti Webového inženýrství se s nejvyšší pravděpodobností setkají s MVC frameworkem Spring boot v rámci předmětů BI-TJV[11] a s frameworkem Symfony v BI-TWA[12]. Tím pádem pro ně architektura aplikace, která je napsaná

[8] `https://react.dev/`
[9] `https://vuejs.org/`
[10] Studenti softwarového inženýrství, webového inženýrství a počítačové grafiky
[11] Technologie Javy
[12] Tvorba Webových Aplikací



Obrázek 2.2 Ukázka procesu pro hodnocení odevzdaného řešení

v Djangu, nemusí být tak srozumitelná. Toto může působit jako zpomalující faktor při nástupu projektu v rámci BI-SP1.

Konkrétní výsledky zasílaného dotazníku na průzkum technologií jsou dostupné v příloze *Dotazník na technologie.ods*, která byla odevzdána s touto prací, nebo ve složce na fakultním OneDrive.

2.5.3.3 Funkcionality

Jak plyne z analýzy výše, tak funkcionality v systému jsou vyhovující. Systém před nasazením do předmětů BI-SP1, BI-SP2 a BI-SWI prošel akceptačním testováním a funkcionality byly ověřeny autorem systému i garantem předmětů. Z rozhovorů s uživateli vyplynulo pouze pár chybějících funkcionalit. V kontextu této práce se jedná především o možnost nahrávat přílohy k hodnocení.

Kapitola 3
Nové řešení

V této kapitole se věnuji možným způsobům dalšího rozvoje portálu SOS a také tomu, který nakonec zvolil náš tým. Dále se věnuji navrženému a implementovanému technickému řešení, které jsme pro další vývoj zvolili. Nakonec se věnuji podpůrným aplikacím, díky kterým byl vývoj nového portálu možný.

Vzhledem k nedostatkům aktuálního řešení se nabízelo několik cest, kterými jsme se mohli v rámci dalšího vývoje systému vydat. Tyto cesty jsme konzultovali s vedoucím projektu Ing. Jiřím Hunkou a také s asistentem Bc. Maxem Hejdou. Pro přehled níže uvádím možnosti, které jsme pro další vývoj uvažovali a jejich výhody a nevýhody.

Rozvoj aktuálního frontendu a backendu

Rozvoj aktuálního řešení postaveného na Djangu a Django templates s Bootstrapem byla první uvažovanou variantou. Z hlediska časové náročnosti by se jednalo zřejmě o krátkodobě nejméně časově náročné řešení. Po týmové konzultaci jsme však dospěli k tomu, že vyřešení aktuálních nedostatků by pouze odložilo nevyhnutelné přepsání uživatelského rozhraní do jiné technologie. Navíc bychom museli tyto limitace složitě obcházet, což by vedlo na spoustu kompromisů a obtížně udržitelný kód.

Dalším problémem, se kterým jsme se potýkali, bylo celkové porozumění technologiím a to převážně backendu. Nikdo z našeho týmu nemá s Pythonem ani Djangem žádné zkušenosti. To by vedlo na nižší týmovou motivaci na projektu pracovat. Důvodem je, že bychom museli investovat značné úsilí do analýzy a procházení dokumentace k technologii, kterou nemáme zájem se učit, protože pro ni nevidíme jiné využití, než pro aktuální SOS. Krom toho není Django využíván v žádném dalším školním projektu[1], což vede na nejed-

[1]Podle dat v systému SOS pro letní semestr 2024

notnost technologií v rámci školních projektů a oslovuje jen omezenou skupinu studentů.

Výhody

- Nízká časová náročnost.
- Stavění na vyzkoušeném základu, který prošel reálným provozem.

Nevýhody

- Složité obcházení limitací na frontendu.
- Nezkušenost s technologií backendu.
- Nejednotnost technologií s jinými projekty v rámci FIT ČVUT.

Vystavení REST API a implementace nového frontendu

Druhou uvažovanou variantou bylo vystavět nové REST API na již existujícím backendu, s použitím balíčku *Django REST framework*[2]. Tím bychom získali možnost, vytvořit pro systém zcela nový frontend, postavený na JavaScriptovém frameworku, což je v současnosti velmi populární možnost pro tvorbu interaktivních uživatelských rozhraní. Více se tomuto přístupu věnuji v sekci 1.8.2.

Tento přístup přináší jednu velkou výhodu ve formě oddělení frontendu od stávajícího backendu. Toto řešení přináší nejen flexibilitu moderního JavaScriptového frameworku, ale také do jisté míry, sjednocuje technologie s jiným projektem – *DBS Portálem*[3]. Na tomto portálu právě probíhá přepsání frontendu do frameworku *Vue.js*. Díky tomu by bylo možné sdílet některé komponenty, nebo umožnit členům týmů plynulý přechod mezi projektem DBS Portál a SOS.

Na druhou stranu zde stále přetrvává problém s backendovou technologií. Django není v době psaní textu používáno na žádném jiném školním projektu. Kvůli tomu není možné sdílet části kódu, nebo vývojáře z jiných projektů. Druhým problémem je, jak bylo výše zmíněno, naše nezkušenost s Pythonem a Djangem a také nízká míra motivace se tyto technologie učit. Z toho jsme měli v kontextu projektu obavy, jelikož bychom museli nad stávající logikou stavět REST API. Toto by mohlo vést k četným chybám a situacím, se kterými původní systém nepočítal, a které by pro nás nemusely být snadno vyřešitelné.

[2] `https://www.django-rest-framework.org/`
[3] `https://dbs.fit.cvut.cz/`

Výhody

- Rozdělení frontendu a backendu na dvě nezávislé aplikace.
- Využití velké části logiky existujícího systému.
- Sjednocení frontendové technologie s DBS Portálem.

Nevýhody

- Potřeba mnoho zásahů do existujícího řešení.
- Backend stále není jednotný s žádným jiným projektem.

Implementace nového backendu a frontendu

Tato cesta zahrnuje kompletní přepsání systému od návrhu, přes implementaci a testování až po nasazení. Technologie, do které by se přepisoval frontend by byl framework Vue.js, především kvůli rozšíření na několika projektech realizovaných v rámci SP projektů a také díky relativně mírné učící křivce.

Pro backend by přicházelo v úvahu několik kandidátů, například PHP framework *Symfony*, nebo framework *ASP.NET Core*, který se nejčastěji kombinuje s jazykem C#. V Symfony byly vyvíjeny mikroslužby DBS Portálu, projekt ale neuspěl[4]. Samotný DBS Portál je aktuálně napsán v několika technologiích, které většinou fungují na jazyce PHP. V blízké době se ale chystá přepsání DBS Portálu právě do ASP.NET Core.

I přesto, že se jedná o časově nejvíce nákladnou variantu, tak přináší hned několik benefitů. Mezi hlavními je sjednocení technologií s DBS Portálem. Další výhodou je motivace budoucích členů SP předmětů, jelikož projekt bude rozdělený a mohou se na vývoji podílet jak čistě frontendové týmy, tak čistě backendové, podle potřeby. V kombinaci tyto výhody znamenají převážně lepší udržitelnost systému a snadnější rozvoj v delším časovém horizontu.

Nevýhodami tohoto přístupu je bezpochyby potřeba přepsat celý systém a tím pádem „zahodit“ to, co je již implementované. Jsou zde ale i světlé stránky, například možnost z velké části využít požadavky, podle kterých byl systém vyvinut a také promítnout do vývoje nového systému zkušenosti získané provozem starého řešení.

Výhody

- Sjednocení technologií s ostatními projekty.
- Motivace našeho týmu na řešení pracovat.
- Rozdělení systému na dva projekty.
- Lepší udržitelnost projektu v delším časovém horizontu.

[4] Podle slov Ing. Jiřího Hunky

Nevýhody

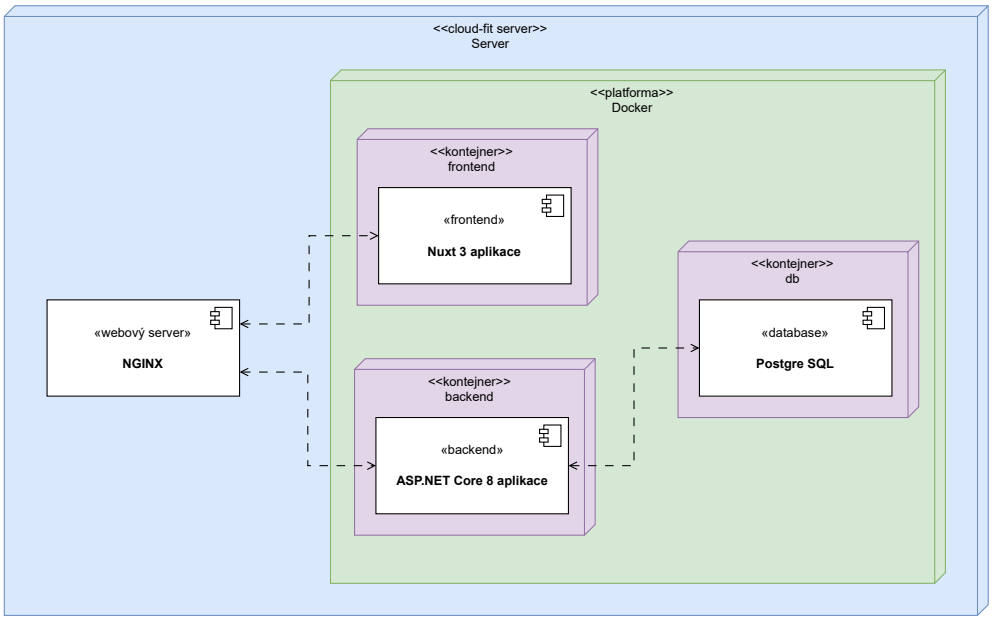
- Potřeba přepsat celý systém, bez možnosti znovu použít stávající logiku.
- Vysoká časová náročnost.

Finální rozhodnutí

Vzhledem ke všem zmíněným možnostem a jejich nevýhodách jsme se v týmu a s vedoucím shodli na poslední variantě, tedy kompletnímu přepisu systému do nových technologií. Technologie, které jsme zvolili, již byly výše zmíněny. Jedná se o JavaScriptový framework Vue.js pro frontend a framework ASP.NET Core, s použitím jazyka C#.

3.1 Architektura nového systému

Nový systém bude, na rozdíl od starého, implementován jako backendové API a oddělený frontend. Webový server zůstane stejný, jako v aktuálním řešení, tedy NGINX. Beze změny zůstane rovněž i databáze, tedy PostgreSQL. Zachování PostgreSQL navíc zmírní nepříjemnosti, které nastanou při pozdější migraci dat ze starého do nového systému. Narozdíl od aktuálního řešení, nebudeme potřebovat Celery, ani žádné jiné systémy, které slouží pro plánované úlohy. Zbavíme se tím celkem 3 kontejnerů, jmenovitě Redisu, Celery a Celery beat. Plánované úlohy obstará ASP.NET Core díky třídám `BackgroundService` a `PeriodicTimer` . Co se týká správy zdrojových kódů,



Obrázek 3.1 Architektura nového systému

tak zůstaneme u fakultního GitLabu. Jediný rozdíl je, že aplikace bude rozložena do dvou repozitářů, na frontend a backend.

Systém bude opět realizován jako shluk kontejnerů s tím rozdílem, že kontejnerizovaný je pouze frontend, backend a databáze. NGINX bude fungovat přímo na serveru, nikoliv jako kontejner. Toto řešení pro NGINX je žádoucí, jelikož nám umožní využít stejný server, který byl již poskytnut pro testovací účely aktuálního SOS a je tedy potřeba podle domény směrovat požadavky do různých aplikací. K orchestraci kontejnerů využíváme zatím *Docker compose*, ale v otázce je také *Docker swarm*, který zatím bohužel nefunguje kvůli komplikacím s ICT. Konfigurace orchestrátoru je dostupná pouze přes přímé připojení na server a není verzovaná v systému GitLab. Nasazení aplikace probíhá skrze GitLab CI/CD a je blíže popsáno v sekci 3.2.1. Architektura celého systému je pro přehlednost zachycena také diagramem 3.1.

3.1.1 Backend

Pro backendovou část aplikace byl zvolen framework ASP.NET Core verze 8 a jazyk C# verze 12. Jedná se o open-source cross-platformový[5] framework od společnosti Microsoft, sloužící k implementaci moderních aplikací a cloudových služeb [88]. Konkrétně verze 8 je *Long Term Support*, pro kterou vývojáři frameworku zaručují podporu do 10. listopadu 2026 [89].

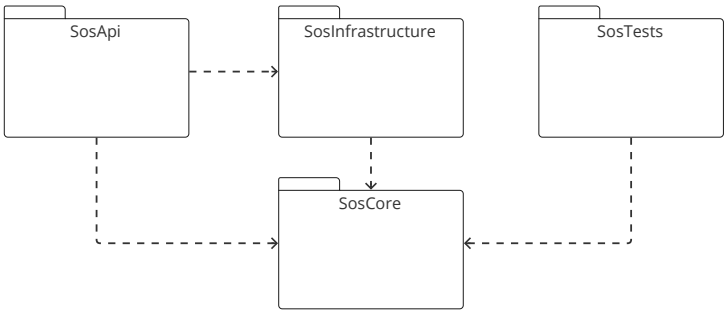
Framework ASP.NET Core je modulární – to znamená, že nemá vše k dispozici v základu, jako například Django. Funkcionality jsou dostupné v rámci modulů, o které lze aplikaci rozšířit. Tento přístup je výhodný, jelikož výsledná aplikace neobsahuje zbytečné závislosti navíc a tím se zmenšuje velikost obrazu pro kontejner.

Jak již bylo naznačeno výše, tento framework byl zvolen převážně kvůli dobré podpoře a rozšíření na FIT ČVUT. Tento framework je navíc velmi podobný frameworku Spring Boot, se kterým se mnoho studentů setká v předmětu BI-TJV a díky tomu předpokládáme, že pro ně nebude složité pochopit zdrojový kód systému.

Struktura aplikace

Pro backendovou část aplikace jsme se rozhodli vytvořit monolitickou aplikaci podle principů čisté architektury. Čisté architektuře se více věnuje sekce 1.2.2.2. Při vývoji jsme se po konzultaci s Bc. Maxem Hejdou rozhodli některé omezení čisté architektury porušit. Toto porušení bylo hlavně z důvodu zavedení uživatelů přes modul `Identity` a také pro urychlení vývoje. Dále se však respektuje čistá architektura, což mimo jiné vedlo na rozdělení aplikace do několika *projektů*, které reprezentují jednotlivé vrstvy čisté architektury. Níže uvádím výčet těchto projektů spolu s jejich rolí v kontextu aplikace. Pro přehlednost je také struktura projektů zobrazena diagramem balíčku 3.2.

[5] Jeden kód funguje na více operačních systémech



Obrázek 3.2 Diagram balíčků backendové části aplikace

Každý z těchto projektů má svoje vlastní závislosti. To přináší výhodu hlavně v tom, že je těžší porušit princip architektury a zhoršit udržitelnost kódu do budoucna. Závislosti každého projektu se specifikují v souboru *csproj*, kam se automaticky přidávají balíčkovacím nástrojem NuGet.

SosCore

Základním projektem je projekt SosCore. Jedná se o projekt obsahující definice všech rozhraní a výjimek. Kromě těchto definicí je zde obsažená aplikační logika a to v podobě servisních tříd a také entity.

Hodně entit obsahuje kromě datových polí také byznys logiku. Jedná se o přístup návrhu, který je v kontrastu s programovacím anti-patternem *anemic domain model*. Tento problém nastává, když v samotných entitách, které představují doménový model, nemáme obsaženou také logiku, která s těmito daty pracuje. Jedná se o velmi rozšířený anti-pattern, který vede většinou ke špatné testovatelnosti a menší přehlednosti výsledného řešení. Je také velmi obtížně najít v servisní vrstvě patřičnou metodu, kterou potřebujeme vyřešit problém [90].

Neznamená to však, že aplikace neobsahuje servisní vrstvu. Jak bylo řečeno výše, servisní vrstva je také součástí projektu SosCore. Hlavním úkolem této vrstvy je připravit potřebná data z databáze pro provedení akce v samotné entitě. V jistých případech je obsažena malá část aplikační logiky i v těchto servisních třídách, avšak jedná se většinou o validaci.

SosInfrastructure

Dalším důležitým projektem je SosInfrastructure. V tomto projektu jsou implementovaná rozhraní pro práci se souborovým systémem a databází. Pro potřeby práce s databází je zde implementováno rozhraní `IDataContext` , které slouží jako poskytovatel databázových tabulek `DbSet` v aplikaci. Dále jsou zde uloženy také databázové migrace, které slouží pro potřeby inkrementálních změn v databázi při tvorbě nových funkcionalit. Databázové migrace jsou

generovány pomocí nástroje EF Core tools[6]. Při lokálním vývoji je možné pomocí stejného nástroje tyto migrace rovněž provést. V produkčním prostředí se ale pro migrace využívá tzv. *bundle*, který je sestaven při tvorbě obrazu v GitLab CI.

SosApi

Posledním projektem důležitým pro běh aplikace je samotné API. Ten obsahuje převážně definice všech kontrolerů a API endpointů. Kromě kontrolerů obsahuje také validace a systém pro zpracování výjimek a jejich přetvoření do formátu JSON, který je následně zpracován na frontendu.

Pro validaci dat se používá modul *Fluent Validation*, který validuje základní formát dat a běžná omezení – délku řetězce, maximální hodnoty atp. Validace je uložena ve speciálních třídách – *validátorech*. Ukázka této validace je přiložena ve výpisu kódu 3.1.

```
public class AssignmentRequestValidator :
AbstractValidator<AssignmentRequest>
{
    public AssignmentRequestValidator()
    {
        RuleFor(req => req.Summary)
            .NotEmpty()
            .WithMessage(ValidationMessageCodes.Required)
            .MaximumLength(255)
            .WithMessage(
                ValidationMessageCodes.MaxLengthExceeded
            );

        RuleFor(req => req.Description)
            .NotEmpty()
            .WithMessage(ValidationMessageCodes.Required);
    }
}
```

Výpis kódu 3.1 Ukázka validátoru

SosTests

Posledním, pro běh produkčního prostředí nedůležitým projektem, je SosTests. Ten obsahuje závislosti potřebné pro testování a samotné testy. Jak již bylo naznačeno, tento projekt není zahrnut ve výsledném produkčním obrazu. Tím je snížená výsledná velikost produkčního obrazu.

[6] https://learn.microsoft.com/en-us/ef/core/cli/dotnet

Moduly aplikace

V backendové části projektu používáme několik modulů. Tyto moduly jsou použity v různých projektech aplikace a to hlavně z důvodu oddělení odpovědnosti. Výčet nejpodstatnějších modulů je dán tabulkou 3.1.

Modul	Funkce	Projekty
AspNetCore	Základní modul webové aplikace	SosApi
Fluent Validation	Validace dat	SosApi
Sentry	Integrace se Sentry	SosApi
SwashBuckle	Generování OpenAPI	SosApi
EF Core	Objektově relační mapper	SosInfrastructure
Identity Framework	Autentizace a autorizace	SosInfrastructure, SosCore
AutoMapper	Mapování entit na DTO	SosCore
SonarAnalyzers	Statická kontrola kódu	Všechny

Tabulka 3.1 Tabulka modulů backendové části aplikace

3.1.2 Frontend

Frontendovou část jsme se rozhodli postavit v TypeScriptu s použitím JavaScriptových a CSS frameworků. Pro přehlednost jsem všechny použité technologie popsal zvlášť.

3.1.2.1 TypeScript

TypeScript je nadstavba nad standardním JavaScriptem, kterému poskytuje typovou kontrolu. TypeScript je pro nás vhodnější, jelikož díky podpoře typování lépe funguje navigace v editoru a je možné hodně chyb zachytit už při psaní kódu, nebo při statické kontrole. Při výsledném sestavení se veškerý TypeScriptový kód kompiluje do JavaScriptu, který běží v prohlížeči nebo Node.js serveru. Nastavení toho, jak silná má být typová kontrola, popřípadě vypnutí některých pravidel kontrolovaných kompilátorem, se nachází v souboru *tsconfig.json* [91].

3.1.2.2 Vue.js 3

Vue.js je JavaScriptový framework pro tvorbu uživatelských rozhraní. Staví na základech standardního HTML, CSS a JavaScriptu a nabízí deklarativní programovací model postavený na komponentách. Díky tomu je možné efektivně vyvíjet uživatelská rozhraní, ať už jednoduchá nebo složitá [92].

Využití Vue.js bylo jednoznačnou volbou a to převážně díky dobré dokumentaci, rozšíření na ostatních projektech FIT ČVUT a mírné učící křivce. Jak

již bylo zmíněno, tak Vue.js využívá komponenty, které se dále dělí na několik částí. V našem případě se jedná o:

Skriptovou část: obsahuje TypeScriptový kód a definuje hlavně reaktivní proměnné, jejichž změna se automaticky renderuje do HTML.

Šablonovou část: obsahuje HTML šablonu komponenty, která je obohacená o metody a direktivy frameworku, které umožňují například iteraci přes pole nebo výpis proměnných ze skriptové části.

Lokalizační část: definuje překlady pro i18n knihovnu pomocí *yaml* zápisu.

Kromě komponent se ve frameworku hojně používají tzv. *composables*, což je pojem označující funkci, která se používá ve skriptové části k obalení stavové logiky [93].

3.1.2.3 Tailwind CSS

Tailwind CSS je velmi oblíbený framework, který se řídí *utility-first* přístupem. To znamená, že poskytuje HTML třídy, u kterých jde na první pohled vidět, co dělají. Dále poskytuje předpřipravené *breakpointy*, které slouží pro definování responzivity. Díky breakpointům můžeme jednoduše definovat, jak bude UI vypadat na různých velikostech zařízení. Tailwind navíc při kompilaci eliminuje nepoužívané, nebo přepsané styly a tím snižuje výslednou velikost přenesených dat. Spolu s komponentami frameworku Vue.js tvoří logický celek, který se snadno a rychle upravuje [94].

3.1.2.4 Nuxt 3 a další podpůrné moduly

Jako nadstavbu nad Vue.js používáme *metaframework* Nuxt 3. Pojem metaframework označuje nadstavbu nad jiným frameworkem nižší úrovně. To znamená, že oproti základnímu Vue.js 3 máme k dispozici více funkcí, které urychlují vývoj a zlepšují celkový výkon aplikace. Nuxt 3 poskytuje především systém směrování, který reflektuje strukturu souborů na URL adresy [95]. Díky tomu je možné jednoduše definovat stránku, včetně adresy, pouhým vložením do složky **pages**. Navíc je možné používat i speciální jména souborů, například **[id].vue** pro zachycení parametru ID, který je poté dále využitelný v komponentě.

Dále framework poskytuje funkce jako jsou automatické importy komponent a funkcí nebo podporu server-side renderingu, o kterém se zmiňuji v sekci 1.8.4. Kromě těchto základních funkcí, využíváme ve frontendové části projektu i další moduly.

NuxtUI: modul obsahující základní komponenty jako tlačítka a ikony. Pro stylování těchto komponent využívá Tailwind CSS.

Nuxt Security: modul zvyšující bezpečnost aplikace pomocí různých hlaviček, nebo middlewarů. Aplikaci chrání především proti XSS, kterému se věnuji v sekci 1.3.1.1 [96].

Pinia: modul pro správu globálního stavu. V aplikaci slouží pro potřeby autentizace a drobečkové navigace, neboli *breadcrumbs*.

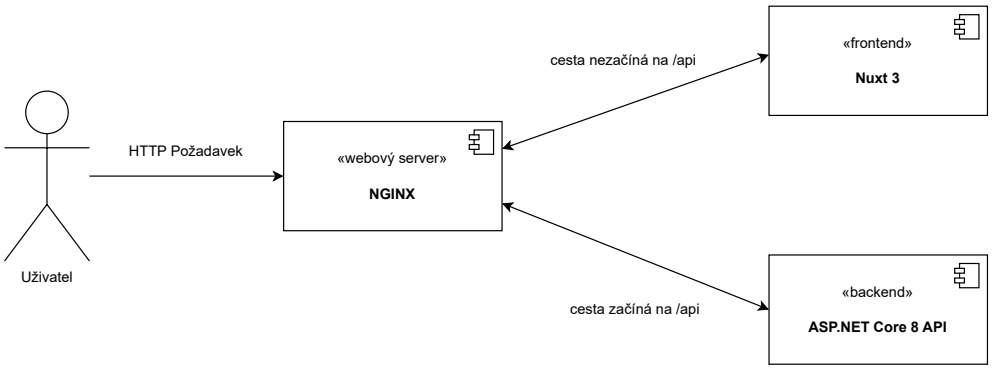
Nuxt i18n: modul pro lokalizaci do jiných jazyků. Obsahuje funkce pro překlad textu pomocí definovaných klíčů a také lokalizaci čísel, času a data.

zod: modul sloužící pro validaci formulářů.

Sentry: modul sloužící k propojení s nástrojem Sentry, pro monitorování výjimek.

3.1.3 NGINX

NGINX bude, stejně jako v aktuálním řešení, sloužit jako webový server naší aplikace. Jeho úlohou je převážně přeposílání požadavků do správné aplikace na základě cesty URL adresy. Princip fungování je ukázán na obrázku 3.3. Díky využití NGINX se v produkčním a testovacím prostředí nemusíme sta-



Obrázek 3.3 Princip směrování požadavků pomocí NGINX

rat o CORS. CORS je mechanika implementována webovými prohlížeči, pro snížení rizika CSRF útoků, které popisuji v sekci 1.3.3.

3.2 Podpůrné nástroje

3.2.1 GitLab

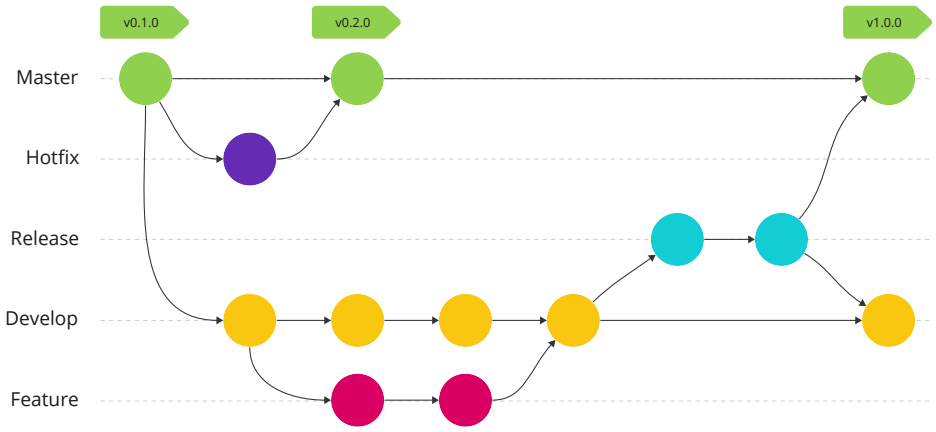
GitLab slouží ke správě zdrojových kódů a rovněž také pro potřeby nasazení aplikačních kontejnerů na server. Pro naše potřeby používáme fakultní GitLab, který obsahuje vše potřebné pro práci na nové verzi systému. Důležitými funkcemi, které při práci na systému využíváme, jsou především *GitLab issues*, které

slouží pro hlášení chyb v programu, nebo *Container registry*, který slouží pro ukládání a následné poskytování vytvořených obrazů pro Docker kontejnery.

Další velmi užitečnou funkcí jsou *merge requesty*, které slouží pro vzájemnou diskuzi nad kódem mezi členy týmu – tzv. *code reviews*. Tento proces slouží hlavně pro včasné zachycení a opravu nekvalitního kódu, ale také k tomu, aby celý tým mohl udržovat přehled o tom, které změny se v aplikaci chystají. Tím je zaručena dobrá kvalita kódu, který se slučuje do master větve.

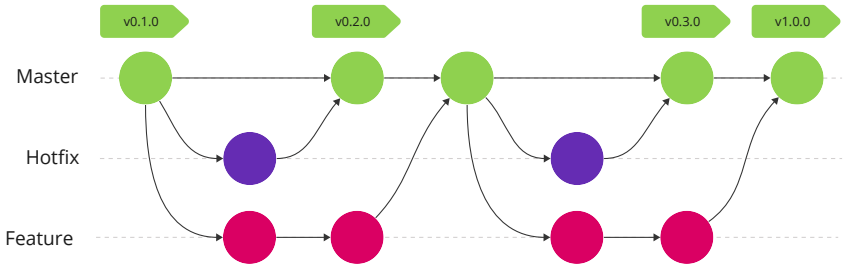
Práce s Gitem

Pro jednotnost jsme v rámci týmu stanovili také *workflow*. Workflow určuje sadu pravidel, jak by měly být vytvářeny větve a jak by měly být slučovány změny a nasazovány nové verze. Při práci s Gitem jsme nejprve využívali workflow Git Flow. Jedná se o poměrně komplikovanou, avšak všeobecně uznávanou workflow, která se hodí pro větší projekty. Popis této workflow je znázorněn obrázkem 3.4. Každopádně při práci s Git Flow, jsme zjistili, že pro náš projekt nepřináší žádné výhody, spíše vnáší komplexitu do verzování a nasazování nových verzí.



Obrázek 3.4 Ukázka Git Flow, převzato z webu Leanpub [97]

To vedlo k tomu, že jsme v průběhu projektu workflow zjednodušili a přešli jsme na mnohem jednodušší GitHub Flow. Tato workflow poměrně dobře ladí s deploy procesem, který popisuji níže. Výslednou workflow znázorňuje obrázek 3.5. Přechod na novou workflow proběhl hladce. Nejprve jsme uzavřeli veškeré merge requesty, které byly otevřené a následně jsme změny sloučili do větve master. Poté jsme pouze odstranili větev develop a tím byl proces změny workflow ukončen.



Obrázek 3.5 Výsledná workflow projektu

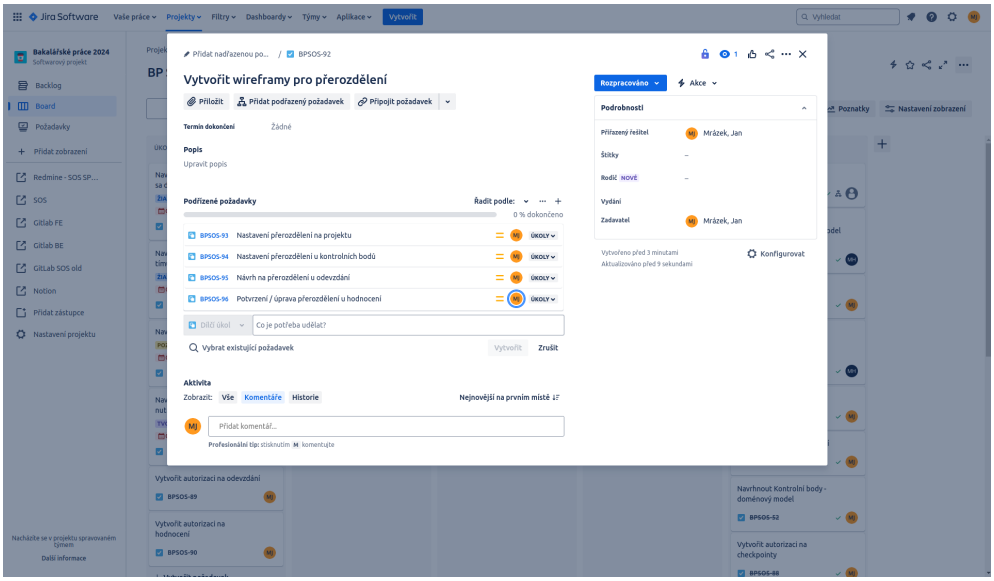
Nasazení

Nasazení probíhá zatím do jednoho prostředí – testovací (STAGE). Až výsledná aplikace projde akceptačními testy a bude připravená do ostrého provozu, tak se přidá produkční prostředí. Tím se splní zásada, podle které by měla existovat alespoň dvě prostředí, jak je popsáno v sekci 1.6.1. Nasazení probíhá tak, že se změny sloučí do větvě *master*, která slouží jako hlavní větev. Automaticky poté probíhá nasazení na testovací prostředí. Po otestování na tomto prostředí, musí vývojář udělat tzv. *tag*, a tím proběhne nasazení na produkční prostředí (až bude připraveno). Tag slouží pro označení verze aplikace, která byla nasazena do produkce. Jedná se prakticky o pojmenovaný ukazatel v historii repozitáře, který navíc může obsahovat stručné informace o provedených změnách. Více o tagu se můžete dočíst například v oficiální Git dokumentaci [98]. To, že tag značí v našem workflow verzi programu implikuje, že jeho název by měl být unikátní a měl by se řídit sémantickým verzováním, které více popisuji v sekci 1.6.3.

3.2.2 Redmine + Jira

Dalším podpůrným nástrojem je nástroj Redmine, který slouží pro projektové řízení. S Redminem jsme pracovali při práci na projektu v rámci BI-SP1 a BI-SP2. Obsahuje hlavně wiki projektu, do které jsou ukládány všechny důležité informace a výstupy z analýz. Dále do Redminu vytváříme požadavky, ve kterých specifikujeme zadání a výstupy úkolů.

Při práci v BI-SP2 jsme přišli na to, že práce s Redminem je pro iterativní vývoj, který obsahuje členění na spoustu menších podúkolů, relativně krkolomná. Systém je poměrně zastaralý a spravování úkolů v něm je časově náročné. Proto jsme se s kolegy dohodli použít pro sledování úkolů na jednotlivých iteracích nástroj Jira a to v bezplatné verzi. Rámcové úkoly jsou však stále vedeny v Redminu projektu pro potřeby zpětného dohledání a trackování času.



Obrázek 3.6 Ukázka rozdělení úkolu v Jiře (snímek obrazovky)

3.2.3 Sentry

Stejně jako předchozí řešení, využíváme pro monitorování výjimek Sentry 1.7.1, konkrétně fakultní řešení [7]. K dispozici máme dva projekty, jeden slouží pro monitorování frontendové části a druhý pro backendovou část. Po zprovoznění aplikace na produkčním prostředí zůstane struktura stejná, jen se přidají prostředí přímo v Sentry.

3.2.4 Slack

Nezměněná zůstala také komunikační platforma, kterou využíváme při práci na projektu. Pro potřeby projektu je Slack vyhovující řešení a poskytuje nám možnosti komunikace v týmu, s vedoucím a asistentem projektu a také s týmem, který se do projektu zapojil v rámci BI-SP1 v letním semestru 2024.

3.2.5 Figma

Figma je nástroj pro vytváření návrhů a prototypů uživatelského rozhraní. Podporuje spolupráci více uživatelů a to i v reálném čase [99].

Pro potřeby SOS využívám Figmu pro návrhy komponent uživatelského rozhraní, popřípadě drátěných modelů obrazovek, podle kterých poté implementuji výsledné rozhraní.

Pro studenty navíc nabízí Figma studentskou licenci, která poskytuje neomezený počet souborů a je tedy ideální volbou pro tvorbu návrhů pro tento

[7] `https://sentry.ksi.fit.cvut.cz/`

projekt. Pro další rozvoj je poté možné převést Figmu projektu do správy vyučujícího a díky tomu se neztratí již vytvořené návrhy a bude možné v nich pokračovat i po skončení této práce.

3.3 Proces vývoje

Vzhledem k časovým možnostem všech členů týmu bylo potřeba vyřešit, jakým způsobem budeme systém vyvíjet a jak nastavíme celkový workflow projektu, aby to vyhovovalo všem účastníkům projektu. V rámci předmětu BI-SP2 jsme se s kolegy rozhodli vyzkoušet principy iterativního vývoje, který nám mohl přinést jistou flexibilitu v ohledu plánované práce, vzhledem k omezeným časovým možnostem všech členů. Zvažovali jsme různé agilní metodiky vývoje softwaru, z nichž nejvíce používanou v praxi je metodika SCRUM, které jsem se více věnoval v předchozí kapitole 1.5.2.1.

Ačkoliv je SCRUM z mnoha důvodů populární technika, rozhodli jsme se s týmem, že jí nevyužijeme. Toto rozhodnutí padlo hlavně z časových a týmových důvodů, jelikož nemáme dedikovaného člověka, který by představoval Scrum mastera a také je SCRUM časově náročný z hlediska denních standupů. Rozhodli jsme se tedy použít některé jeho principy, jako jsou rychlé sprinty rozdělené do vícero menších úkolů a stand-upy jsme zavedli na konci každého sprintu. Na těchto stand-upech jsme řešili individuální pokroky, rámcové plánování na další sprint, popřípadě také problémy, na kterých jsme se zasekli a potřebujeme s nimi pomoc.

Dále se na těchto stand-upech řešily úlohy, které vyžadovaly užší spolupráci celého týmu, jako například požadavky, které zasahovaly do více částí systému a mohly by posléze některého člena týmu zablokovat.

Nic se ale nemění na tom, že na konci každé iterace vznikla spustitelná aplikace, která obsahovala funkcionality definované pro danou iteraci. Vzhledem k občasnému zdržení v procesu code review však byly tyto výsledky nasazovány jednotlivě.

3.4 Požadavky na nový systém

3.4.1 Funkční požadavky

Funkční požadavky pro vývoj této práce jsem skládal při každé iteraci. Pro sběr těchto požadavků jsem dle doporučení [100] využili primárně dvě metody.

První metodou využitou pro sběr požadavků byla analýza požadavků stávajícího systému. Díky tomu, že systém prošel akceptačním testováním, tak většina požadavků z tohoto systému byla využitelná i pro nový systém.

Druhou metodou byly již zmíněné rozhovory s uživateli systému. Rozhovory jsme vedly s celkem třemi vyučujícími, kteří se SOSem pracují. V kontextu této

práce bylo zmíněna především možnost nahrávat soubory k hodnocení. Další požadavky se týkaly prací týmových kolegů.

3.4.2 Nefunkční požadavky

Nefunkční požadavky doplňují ty funkční a slouží pro určení omezení kladených na použití systému. Mezi tyto požadavky patří například požadavky na výkon aplikace, její dostupnost, zabezpečení nebo rozšiřitelnost [101]. Pro přehled zde uvádím seznam nefunkčních požadavků, který je kladený na celý systém.

NP01 – Oprávnění pro přístup

Důležitost: Vysoká

Kategorie: Zabezpečení

Aplikace je dostupná pouze přihlášeným uživatelům.

NP02 – Lokalizace

Důležitost: Vysoká

Kategorie: Použitelnost

Aplikace je dostupná ve dvou jazycích – češtině a angličtině.

NP03 – Dostupnost

Důležitost: Vysoká

Kategorie: Spolehlivost

Aplikace je dostupná 24 hodin denně, 7 dní v týdnu.

NP04 – Responzivita

Důležitost: Střední

Kategorie: Použitelnost

Aplikace je primárně určena pro desktopová zařízení, je však nutné aby se dala rozumně zobrazit i na mobilním zařízení.

NP05 – Rozšiřitelnost a dokumentace

Důležitost: Střední

Kategorie: Rozšiřitelnost

Aplikace je jednoduše rozšiřitelná o další funkce a to jak na frontendu tak na backendu. Aplikace obsahuje dokumentaci API.

NP06 – Odezva

Důležitost: Střední

Kategorie: Výkon

Aplikace při použití adekvátně rychlého připojení k internetu zpracuje většinu požadavků do půl vteřiny.

Kapitola 4
Vývoj nového řešení

V této kapitole se věnuji průběhu vývoje nového systému SOS. Kapitola začíná vývojem v rámci BI-SP2 a končí poslední iterací zhotovenou před uživatelským testováním.

Cíle vývoje

Cílem vývoje v rámci projektu BI-SP2 a této práce bylo co možná nejvíce se přiblížit funkcionalitám aktuálního řešení a vyřešit nedostatky, které se týkají sekcí kontrolních bodů, odevzdávání a hodnocení. Jak již bylo zmíněno, tak tyto funkcionality byly implementovány v iteracích, které navazovaly na vývoj v rámci BI-SP2.

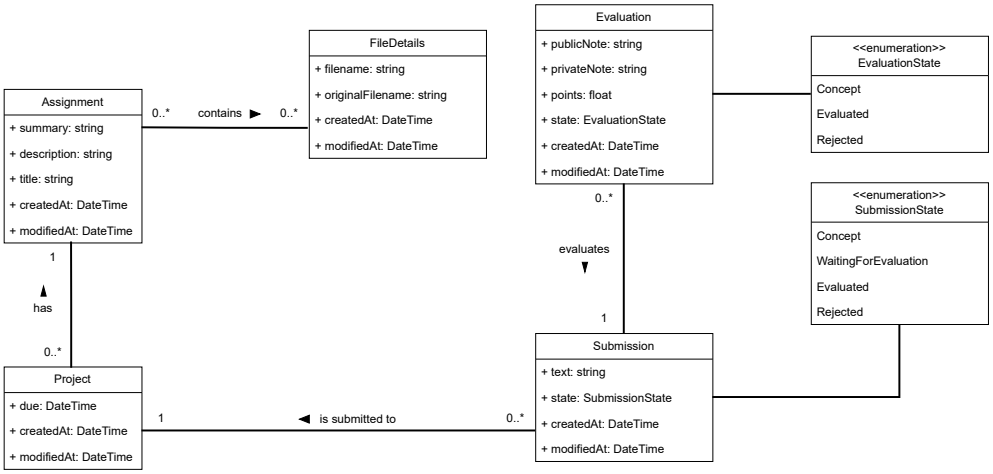
4.1 Vývoj v rámci BI-SP2

Pro kontext této práce je důležité říct, že právě v zimním semestru 2023 jsme se s kolegy Markem Hujem a Janem Jamnickým rozhodli, že budeme na novém systému SOS pracovat v rámci našich bakalářských prací. Práci v průběhu BI-SP2 jsme tedy pojali hlavně jako přípravu pro bakalářské práce. Cílem bylo položit základy dílčích prací a také vyzkoušet práci v týmu při použití naší iterativní metody.

Jednalo se o naše první setkání s iterativním vývojem a tak jsme hlavně potřebovali stanovit proces, který nám na práci vyhovoval. S tímto nám velmi pomohl Timotej Adamec, který v běhu BI-SP2 působil jako náš projektový manažer a řešil vše ohledně zadávání iterací a plánování.

Stanovili jsme zde také režim našeho iterativního vývoje, který se skládá z podobných částí jako iterativní vývoj v praxi, jak je popsáno v sekci 3.3.

Pro potřeby tohoto textu není nutné zmiňovat detaily pro všechny iterace, které jsme s týmem realizovali. Záznamy o těchto sprintech a konkrétních úko-



Obrázek 4.1 Doménový model sekce odevzdání a hodnocení

lech v nich realizovaných jsou zaznamenány v Redmine u projektu SOS[1]. Níže budou uvedeny pouze výsledky, které se týkají této sekce, nebo které jsem realizoval v rámci průběhu BI-SP2.

4.1.1 Návrh a implementace sekce odevzdání a hodnocení

Velkou částí mé práce v BI-SP2 byl návrh a následná implementace základu pro odevzdání a hodnocení. Sestavil jsem návrhy některých komponent a rovněž doménový model, který je přiložen na obrázku 4.1. Podle tohoto modelu jsem poté implementoval funkční část a také velmi ranou verzi uživatelského rozhraní, skrze které mohl uživatel vytvořit odevzdání přímo k projektu a to poté mohl příslušný vyučující ohodnotit.

Výsledkem této práce bylo připravení komponent a formulářů a systém na zpracování souborů, který ale musel projít dalšími změnami, neboť nepodporoval úpravy již nahraných souborů.

4.1.2 Další podíl na projektu

V této sekci uvádím části projektu, na kterých jsem pracoval. Jedná se o práci, která se většinou týkala větší části systému, nebo funkcionality, která je pro fungování systému kritická.

Přihlašování na frontendu

Nejprve zmíním přihlašování na straně frontendu. To zahrnovalo vytvoření formuláře pro zadání uživatelského jména a hesla, odeslání požadavku na při-

[1] `https://dbs.fit.cvut.cz/redmine/projects/sos-fit-cvut-cz/roadmap`

hlášení a následné uložení autentizačního tokenu. Jako token používáme JWT.

JSON Web Token (JWT) je kompaktní způsob pro reprezentaci práv a dat uživatele. Tato data jsou kódována jako JSON objekt, který je navíc uložen do obalujícího objektu opatřeného digitálním podpisem [102].

Získaný JWT je posléze odesílán s každým požadavkem v HTTP hlavičce *Authorization* a tím se frontendová část autentizuje na backendu. V případě, že je uživatel nepřihlášený, systém ho vždy přesměruje na stránku přihlášení.

S tímto souvisela i potřeba snadno získat aktuálně přihlášeného uživatele a informace o něm. Pro tyto účely jsem do systému přidal composable `useCurrentUser`, která udržuje kontext o přihlášeném uživateli pomocí modulu *Pinia*.

Implementaci této části ukazuje výpis kódu 4.1. Na ukázce lze vidět datový typ `DecodedUser`, který představuje data, která jsou vracena z této composable a dále využívána v uživatelském rozhraní. V samotné composable `useCurrentUser` se poté využívá získaný JWT, ze kterého se pomocí funkce `jwtDecode` dekóduje přihlášený uživatel a ten se uloží do proměnné. Tato funkce je volaná v computed proměnné `currentUser`, což znamená, že se volá jen v případě změny JWT.

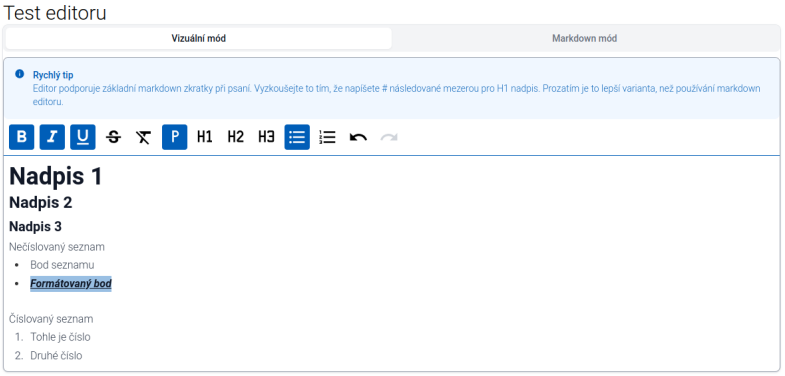
```
type DecodedUser = {
  id: string;
  username: string;
  email: string;
  firstName: string;
  lastName: string;
  role: UserRole;
};

export const useCurrentUser = defineStore("currentUser", () => {
  const authTokenStore = useAuthTokenStore();
  const currentUser = computed<UserDetailResponse | null>(
    () => {
      const token = authTokenStore.getAuthToken();
      if (token === null || token === undefined) return null;
      const decoded = jwtDecode<DecodedUser>(token);
      return { ...decoded, id: parseInt(decoded.id) };
    });
  const loggedIn = computed<boolean>(
    () => currentUser.value !== null
  );
  return { currentUser, loggedIn };
});
```

Výpis kódu 4.1 Composable pro získání přihlášeného uživatele

Textový editor

Pro potřeby specifikace zadání a delších popisů vyvstala potřeba implementovat uživatelsky přívětivý textový editor, který bude vzhledem odpovídat moderním standardům. Pro tento účel jsem zvolil jako základ editor *TipTap*[2] s rozšířením pro markdown[3]. Jedná se o velmi rozšířený *headless* textový editor – neobsahuje žádné styly, ale pouze API na ovládání.

Editor rovněž podporuje markdownové zkratky, což z něj činí velmi mocný nástroj v rukou někoho, kdo se již s markdownem setkal. Pro méně zkušené uživatele je ovšem stále k dispozici ovládání skrze tlačítka, která lze používat, jako ve standardních editorech. Editor dále podporuje funkce jako *undo* a *redo*, což přináší uživateli možnost jak rychle opravit chyby. Výsledný textový editor je ukázán na snímku obrazovky 4.2.



Obrázek 4.2 Ukázka textového editor (snímek obrazovky)

Drobečková navigace

Pro potřeby navigace v uživatelském rozhraní a udržení přehledu o stránce, na které se uživatel nachází, vznikla potřeba implementovat drobečkovou navigaci. Drobečkovou navigaci jsem implementoval s pomocí modulu *Pinia*. V aktuálním stavu si každá stránka zadefinuje, jaké položky se v drobečkové navigaci zobrazí pomocí funkce `setPages`, která je dostupná v composable `useBreadcrumbsStore`. Každá z těchto částí je definovaná v separátním souboru, kde lze zadefinovat i možnost překladu a tudíž i klíč pro překlad. Díky tomu je na frontendu plně přizpůsobitelná drobečková navigace.

Jak lze vidět v ukázce kódu 4.2, tak stránky jsou definované objektem `SosPage`, který obsahuje samotnou URL, příznak přeložitelnosti a nakonec také název, nebo překladový klíč. V této funkci se poté každá položka drobečkové navigace překládá do výsledného jazyka, pokud je nastaven parametr `translatable`.

[2] `https://tiptap.dev/`
[3] `https://github.com/aguingand/tiptap-markdown`

```
export const useBreadcrumbsStore = defineStore("breadcrumbs",
() => {
  const { t } = useI18n({
    useScope: "global",
  });
  const sosPages = ref<SosPage[]>([Pages.Home()]);

  const breadcrumbs = computed<BreadcrumbLink[]>(() => {
    return sosPages.value.map((sosPage) => {
      const breadcrumbLink: BreadcrumbLink = {
        label: sosPage.translatable ?
          t(`pages.${sosPage.key}`) :
          sosPage.label,
        to: sosPage.to,
      };
      return breadcrumbLink;
    });
  });

  const setPages = (newPages: SosPage[]) => {
    sosPages.value = [Pages.Home(), ...newPages];
  };

  return {
    breadcrumbs,
    setPages,
  };
});
```

Výpis kódu 4.2 Ukázka breadcrumbs

Systém pro nahrávání souborů

Pro potřeby doplnění zadání, odevzdání či hodnocení o soubory vznikla potřeba implementovat do systému práci se soubory. Tento úkol zahrnoval vytvoření komponenty na frontendu, která umožňuje stáhnutí a vkládání souborů. Tato komponenta se dá jednoduše integrovat do všech formulářů a výpisů a poskytuje tak poměrně pohodlný přístup pro manipulaci se soubory. Na backendu bylo nutné vytvořit systém, který kontroluje soubory na nepovolené koncovky a také endpointy, které slouží pro uložení, stažení a mazání souborů.

Verze vytvořená v průběhu BI-SP2 poté prošla velkým vylepšením, které přineslo nové možnosti týkající se editace souborů. Tomuto vylepšení se více věnuje druhá iterace 4.3.

Refaktor frontendu

Ke konci běhu BI-SP2 vznikla potřeba sjednotit styly v celé frontendové části a provést refaktor často používaných komponent. Při tomto refaktoru jsme se také rozhodli přejít z UI knihovny Quasar[4] na více přizpůsobitelnější NuxtUI (více popsáno v sekci 3.1.2.4). Toto vyřešilo řadu problémů, například nekonzistentí styly, které se kvůli rozdílnému CSS frameworku našeho projektu a knihovny Quasar v projektu vyskytly.

Refaktor backendu

Stejně jako u frontendu, tak i na backendu byla potřeba výsledné řešení refaktorovat. V rámci tohoto refaktoru jsem provedl také rozdělení backendového řešení do projektů, jak je uvedeno na obrázku 3.2. Dále jsem do projektu přidal balíček AutoMapper, který velmi usnadnil práci s mapováním objektu do objektu, ale také práci s databázovými dotazy. Tyto změny byly konzultovány s asistentem projektu Bc. Maxem Hejdou.

V ukázce kódu 4.3 lze vidět využití zmíněného AutoMapperu. Ten se používá i na úrovni databáze, při potřebě výpisu DTO. Díky tomu se minimalizují datové přenosy a hydratace nepotřebných atributů objektu, jelikož projekce probíhá přímo v SQL a vrací přímo specifikované DTO.

```
public async Task<CheckpointDetailResponse> GetById(
    long checkpointId,
    CancellationToken cancellationToken = default)
{
    var response = await _dataContext.Checkpoints
        .Where(c => c.Id == checkpointId)
        .ProjectTo<CheckpointDetailResponse>(
            _mapper.ConfigurationProvider)
        .FirstOrDefaultAsync(cancellationToken);

    Guard.Against.NotFound(checkpointId, response);

    return response;
}
```

Výpis kódu 4.3 Ukázka mapování přes projekci

Sestavení obrazu v CI pro frontend i backend

Pro potřeby iterativního vývoje, kdy na konci každé iterace je funkční produkt, bylo nutné vyřešit sestavení obrazu pro produkční kontejner. Výsledkem byly

[4] https://quasar.dev/

soubory *Dockerfile*, které slouží jako sada instrukcí pro sestavení obrazu a také upravené *.gitlab-ci.yml* soubory, které definují kroky pro provedení v GitLab CI. Obraz se sestaví při každém pushi do větve *master* v GitLabu. Takto sestavený obraz je poté automaticky nasazen na testovací server. Při vytvoření tagu ve větvi master probíhá sestavení znovu, ale výsledný obraz nese jiný štítek. Tento obraz bude nahráván na produkční server, až bude k dispozici.

Code style pro frontend

Pro potřeby jednotného vzhledu kódu a dodržování určitých pravidel bylo potřeba do projektu zanést statickou kontrolu kódu. Pro tyto účely jsem do projektu nasadil program ESLint[5], který nahlašuje chyby, jako například nevyužité proměnné nebo porušení best-practices Vue.js.

Pro kontrolu stylu kódu, jako je například maximální délka řádků nebo odsazování, byl do projektu přidán plugin Prettier.

V poslední řadě byla do projektu přidána kontrola typů a to přímo TypeScriptem. Všechny tyto nástroje je možné spustit lokálně a automaticky se spouští také v GitLab CI.

4.1.3 Souhrn práce v BI-SP2

V rámci BI-SP2 jsme se seznámili s procesem iterativního vývoje. Vytvořili jsme společný základ pro dílčí bakalářské práce a připravili potřebnou infrastrukturu pro vývoj projektu. Součástí této infrastruktury bylo jak nasazování na server, tak GitLab CI, které zaručuje, že nemůže být nasazena funkcionalita nesplňující standardy kódu, nebo automatické testy. Díky tomu bude zároveň snazší zapojit do projektu nové lidi a udržet alespoň částečnou kontrolu nad jednotným stylem kódu a jeho kvalitou.

4.2 První iterace – kontrolní body

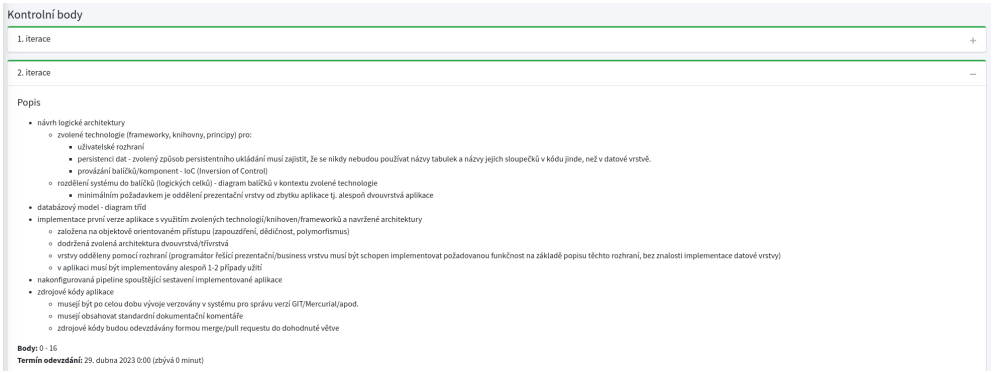
4.2.1 Analýza

Kontrolní body se nachází již v současném systému SOS. Využívají se aktuálně ve všech předmětech, které SOS používají, a to sice ke stanovení dílčích zadání a také termínu, do kdy má být tato část hotová.

V případě předmětu NI-NUR se dále v rámci posledního kontrolního bodu provádí uživatelské testování. Tohoto testování se účastní testeři z řad studentů, kteří poté za otestování projektu dostanou bonusové body. Studentské testování každopádně není předmětem této iterace a tak jej nebudu zahrnovat do funkčních požadavků zde specifikovaných.

Ze získané zpětné vazby uživatelů nevyplynuly pro kontrolní body žádné významné nedostatky, které by potřebovaly hlubší zásah do chování této sekce.

[5] `https://eslint.org/`



Obrázek 4.3 Snímek obrazovky ukazující kontrolní body v aktuálním systému

Dvě menší změny, které bude nutné v rámci této iterace navrhnout a zpracovat, se týkají prvních dvou požadavků z tabulky 2.1.

Oba tyto požadavky se týkají zobrazení relevantních informací ohledně kontrolního bodu – jmenovitě termínu odevzdání a bodového ohodnocení. V současné verzi systému se kontrolní bod zobrazuje pouze jako rozbalovací sekce, kde jsou veškeré informace, až na její název, viditelné až po rozkliknutí. Pro ilustraci přikládám snímek obrazovky z aktuálního systému 4.3.

4.2.2 Funkční požadavky iterace

FP-1 – Přidání kontrolního bodu

Systém umožňuje k jednotlivým projektům přidat kontrolní body. Přidání probíhá přes nabídku nastavení projektu. Kontrolní body obsahují název, dílčí zadání ve formě formátovaného textu, maximální počet bodů a datum odevzdání. Tato vstupní data systém kontroluje a v případě neplatných dat systém zobrazí patřičnou chybovou hlášku. Systém umožňuje přidat k projektu libovolný počet kontrolních bodů.

FP-2 – Upravení kontrolního bodu

Systém umožňuje upravit jednotlivé kontrolní body u projektu a to v plném rozsahu. Systém rovněž ověřuje vstupní data uživatele, stejně jako u přidání kontrolního bodu, a v případě chyby zobrazí patřičnou chybovou hlášku.

FP-3 – Odebrání kontrolního bodu

Systém umožňuje odebrat kontrolní body z projektu. Podmínkou pro odebrání kontrolního bodu je to, že k němu ještě nebylo vytvořeno žádné odevzdání. V tomto případě systém znemožní odebrání kontrolního bodu a zobrazí chybovou hlášku.

FP-4 – Zobrazení kontrolních bodů

Systém zobrazuje kontrolní body formou seznamu. U každého kontrolního bodu systém zvýrazní získaný počet bodů a stav, ve kterém se nachází. Stav kontrolního bodu se mění v závislosti na odevzdáních a jejich hodnoceních.

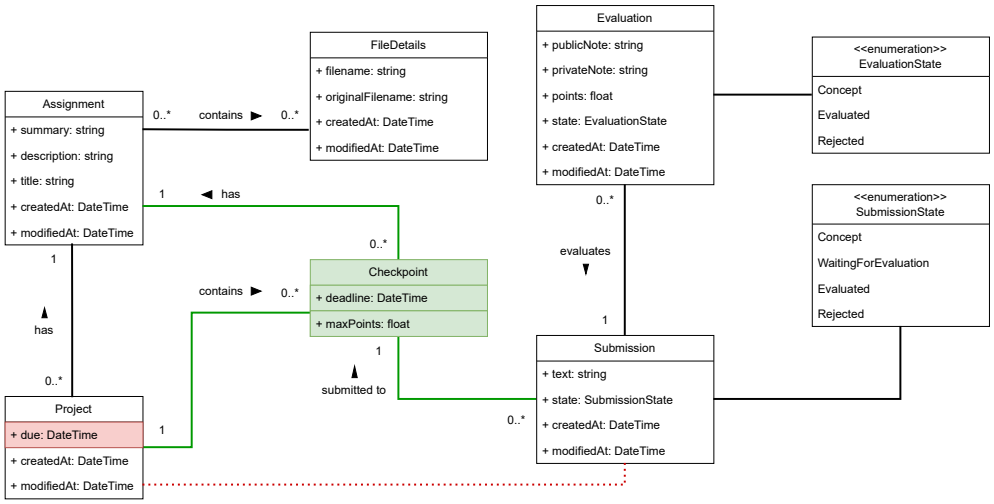
4.2.3 Návrhy iterace

4.2.3.1 Wireframy obrazovek

V rámci návrhu kontrolních bodů byly ve Figmě zpracovány wireframy obrazovek a to pro zobrazení A.1 a nastavení A.2.

4.2.4 Změny v doménovém modelu

Vzhledem k tomu, že kontrolní body v novém systému zatím nebyly, tak bylo potřeba provést rozsáhlejší změny v doménovém modelu nového systému. Tyto změny znázorňuje UML diagram 4.4. Zeleně znázorněné spojnice a atributy představují přidané prvky, červené naopak prvky odebrané.



Obrázek 4.4 Doménový model pro kontrolní body

4.2.5 Výsledná implementovaná část

Implementace kontrolních bodů s sebou nesla jeden konečný zádrhel, kterým byla migrace databáze. Ta nefungovala dobře v případě, když byly v systému data, jelikož se značně měnila struktura systému. Problém byl vyřešen promazáním testovací databáze. Toto řešení bylo jednorázově přijatelné, jelikož se v systému nacházela pouze testovací data. Je však nutno podotknout, že

by k takovým situacím při dalším vývoji nemělo dojít a budou se muset více kontrolovat vygenerované databázové migrace.

Kromě problémů s migracemi však proběhlo zprovoznění této iterace bez větších problémů. Změny provedené v této iteraci prošly vzájemným code review a jsou dostupné v merge requestech:

Backend: `https://gitlab.fit.cvut.cz/sos/new-sos/backend/-/merge_requests/51`

Frontend: `https://gitlab.fit.cvut.cz/sos/new-sos/frontend/-/merge_requests/53`

Po dokončení code review a sloučení změn do větve *master* byla verze systému nasazena na testovací server a označena tagem **v0.3.0** v GitLabu a to jak v repozitáři pro frontend, tak v repozitáři pro backend.

Vzhledem k závislosti na dalších iteracích je výsledná část ukázána na snímcích obrazovky z konečného stavu projektu.

4.3 Druhá iterace – odevzdání

4.3.1 Analýza

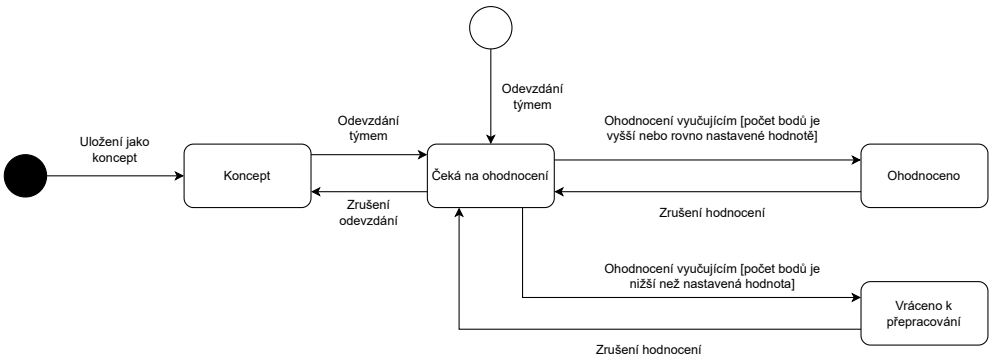
Rovněž jako kontrolní body, tak i odevzdání se nachází v aktuálním systému. Slouží pro odevzdání vyřešeného kontrolního bodu, včetně souborů, nebo odkazů. Toto odevzdání je později ohodnoceno vyučujícím a student získává udělené body.

V současném systému jsem pomocí rozhovorů s uživateli systému identifikoval několik požadavků a rozdílů, které bude nutné oproti stávajícímu řešení navrhnout a vyřešit.

Prvním tímto rozdílem je situace, do které se dostávalo poměrně značné množství týmů při práci na společných projektech. Často docházelo k situaci, kdy bylo vytvořeno více odevzdání po sobě, což vedlo na znepřehlednění výsledného seznamu, jak je uvedeno i v tabulce 2.1. Studenti poté nevěděli, které odevzdání je to, které se bude hodnotit, a v systému tak občas vzniklo odevzdání, které se nikdy neohodnotilo. Tato situace nastala i v našem týmu, při průchodu předmětem BI-SP1. V ideálním případě by systém měl povolit pouze jedno odevzdání, které je rozpracované, nebo čeká na ohodnocení. V případě ohodnoceného odevzdání se však může jednat o úpravy za účelem vyššího bodového zisku, takže stále zde zůstává potřeba mít možnost odevzdat další odevzdání, pokud bylo předchozí již ohodnoceno.

Dalším rozdílem, který se týká odevzdání, jsou jeho stavy. V současném systému může mít odevzdání pouze 3 stavy, které se mění v závislosti na hodnocení tohoto odevzdání. Diagram stavů v současném systému je znázorněn diagramem 4.5. Jak je vidět z diagramu, tak systém v aktuální podobě neobsahuje explicitní vrácení odevzdání. Vrácení odevzdání je možné docílit, pokud

vyučující udělí minimální počet bodů, který je uvedený v nastavení. Toto chování není pro uživatele příliš transparentní a může působit problémy, pokud má kontrolní bod nulové hodnocení.



Obrázek 4.5 Stavový diagram odevzdání v aktuálním systému SOS

Posledním, byť velmi nepravděpodobným případem, je také situace, kdy vyučující začne odevzdání hodnotit a tým jej mezitím upraví. Vyučující se v tomto případě dostává do situace, kdy nemá kontrolu nad tím, jestli studenti mohou odevzdání měnit či nikoliv. Tento případ byl objeven a diskutován ještě v rámci běhu BI-SP1 a bylo navrženo řešení v podobě zámku pro odevzdávání. Tento zámek by se automaticky aplikoval, když vyučující začne hodnotit odevzdání, a tím je zaručeno, že bez jeho vědomí nelze odevzdání již měnit. Je zde však nutné myslet i na situaci, kdy se tým dodatečně s vyučujícím domluví (například na projektové schůzce), že by před ohodnocením chtěli ještě odevzdání upravit. Vyučující tedy musí mít možnost takto zamknuté odevzdávání odemknout a tím umožnit týmu úpravy.

4.3.2 Funkční požadavky

Z výše provedené analýzy tedy plynou pro odevzdání v novém systému následující funkční požadavky:

FP-1 – Vytvoření nového odevzdání

Systém umožňuje studentovi přiřazenému k projektu vytvořit nové odevzdání pro kontrolní bod. Nové odevzdání je možné vytvořit pouze v případě, že poslední odevzdání není ve stavu *koncept*, nebo *čeká na ohodnocení*. V případě, že je poslední odevzdání ve stavu *vráceno k přepracování*, nebo *ohodnoceno*, systém umožní vytvořit nové odevzdání. Systém rovněž kontroluje vstupní data a v případě nekompletních nebo neplatných dat zobrazí chybovou hlášku.

FP-2 – Zamknutí odevzdání pro úpravy

Systém umožňuje vyučujícímu zamknout odevzdání ve stavu *čeká na ohodnocení*. Toto zamknutí posléze znemožní studentovi upravit odevzdání a to do chvíle, dokud jej vyučující opět neodemkne.

FP-3 – Nahrání příloh k odevzdání

Systém umožňuje studentovi přiřazenému k projektu nahrávat přílohy k odevzdání, které je ve stavu *koncept*, nebo k odemčenému odevzdání ve stavu *čeká na ohodnocení*.

FP-4 – Úpravy odevzdání

Systém umožňuje studentovi přiřazenému k projektu upravit vytvořené odevzdání. Úpravy je možné provádět v celém měřítku, tedy včetně přiložených souborů. Systém tyto úpravy umožňuje pouze u odevzdání ve stavu *koncept*, nebo *čeká na ohodnocení*, pokud nebylo zamknuto pro úpravy.

FP-5 – Uložení konceptu odevzdání

Systém umožňuje studentovi vytvořit koncept odevzdání. Koncept je pro vyučujícího neviditelný a slouží pro týmové účely, kdy každý z týmu může do konceptu přidat vlastní výstup. Tento koncept je poté možné odevzdat. V případě, že je u kontrolního bodu vytvořený koncept odevzdání, systém neumožní vytvořit další odevzdání.

4.3.3 Návrhy iterace

4.3.3.1 Wireframy

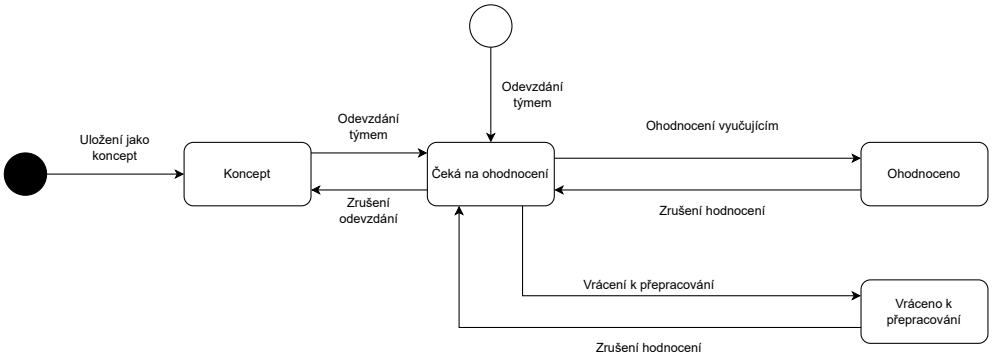
V rámci této iterace byly navrženy wireframy A.3-A.8, které lze nalézt v příloze.

4.3.3.2 Stavový diagram

Pro potřeby odevzdání a možnosti explicitně vrátit odevzdání, nebo udělit nulové hodnocení, byl lehce upraven stavový diagram. Při přechodu do stavu *vráceno k přepracování* se již nepoužívá podmínka při hodnocení, ale explicitní akce uživatele, jak je vidět na diagramu 4.6.

4.3.3.3 Doménový model

Změn v doménovém modelu nebylo oproti předchozí iteraci potřeba příliš mnoho, jak je ostatně vidět i na diagramu 4.7. Jednou z potřebných změn



Obrázek 4.6 Stavový diagram s upraveným přechodem – nový stav

bylo přidání vazby na entitu reprezentující soubory, což umožní implementovat logiku nahrávání příloh k odevzdání, jak specifikuje funkční požadavek 4.3.2.

Další změnou bylo přidání zámku, který je reprezentovaný výčtovým typem **SubmissionLockState**. Tento typ nese 3 hodnoty, což může působit jako nadbytečná logika, ale je to z prostého důvodu. Po odeslání odevzdání k hodnocení se zámek nachází ve stavu *Unlocked*. Tento stav je automatický a dává systému informaci, že vyučující se zámkem ještě nikterak nemanipuloval. Při prvním rozkliknutí formuláře na hodnocení systém automaticky změní stav zámku na *Locked*, ale pouze v případě, že je zámek odevzdání ve stavu *Unlocked*. Pokud je zámek ve stavu *UnlockedManually*, systém nijak do tohoto stavu nezasahuje a zamknutí odevzdání pro úpravy je předáno do režie vyučujícího.

4.3.4 Výsledná implementovaná část

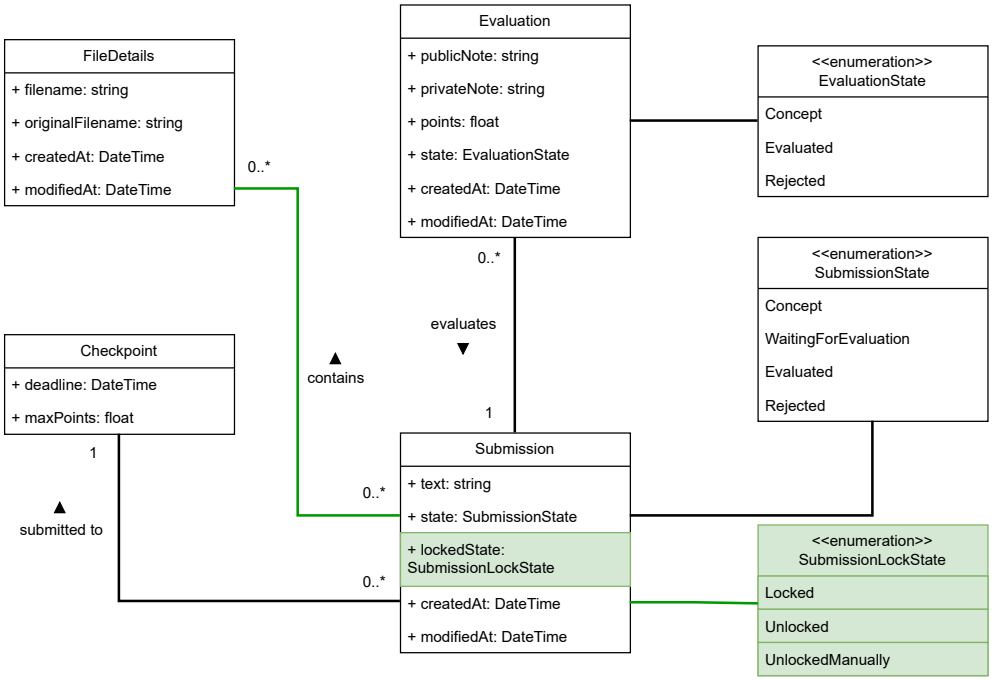
V této iteraci jsem se nakonec rozhodl pro neimplementování posledního funkčního požadavku, kterým byl koncept odevzdání. Neimplementoval jsem jej z časových důvodů, jelikož hodně práce zabrala analýza a přepracování komponenty pro práci se soubory. Tento požadavek jsem přesunul do konečné iterace, která byla zpracována těsně před uživatelským testováním.

Výsledná implementovaná část této iterace obsahuje splněné funkční požadavky definované pro tuto iteraci, kromě požadavku na koncept odevzdání.

Na straně API byla implementována logika pro zamezení úprav v závislosti na stavu zámku odevzdání a byla vytvořena API dokumentace pokrývající tuto část systému.

Na frontendu bylo odevzdání integrováno s komponentou kontrolního bodu a rovněž byly přidány také informační hlášky, které uživateli sdělují, proč nemůže vytvořit nové odevzdání nebo upravit stávající.

Implementace této sekce proběhla bezproblémově a bezproblémové bylo i nasazení na testovací server. Změny prošly procesem code review a jsou dostupné v merge requestech:



Obrázek 4.7 Výseč doménového modelu se změnami pro odevzdání

Backend: `https://gitlab.fit.cvut.cz/sos/new-sos/backend/-/merge_requests/59`

Frontend: `https://gitlab.fit.cvut.cz/sos/new-sos/frontend/-/merge_requests/56`

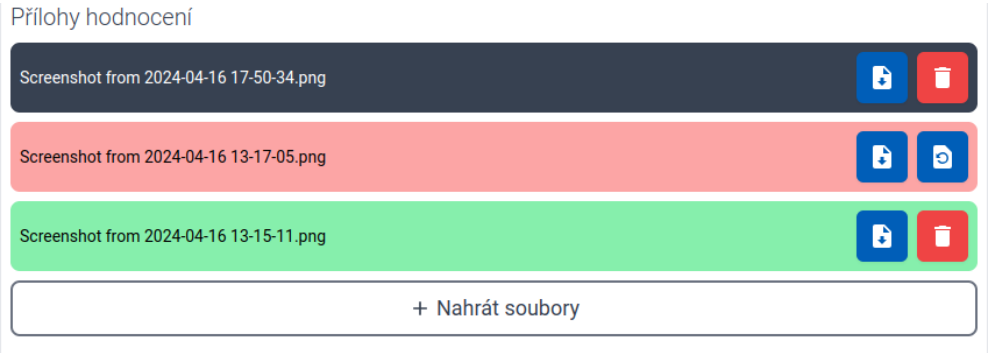
Výsledná verze byla po nasazení na testovací server označena štítkem **v0.5.0** a to v obou projektových repozitářích.

4.3.5 Další výstupy této iterace

4.3.5.1 Práce se soubory

Pro potřeby editace příloh k odevzdávání bylo nutné přepracovat systém pro práci se soubory a to především ve frontendové části. Do této iterace bylo možné soubory pouze nahrát a smazat, nikoliv později editovat. Soubory se nahrály vždy s odesláním formuláře. Při mazání byl soubor ihned smazán, bez jakéhokoliv potvrzení, nebo možnosti vidět, které soubory budou po výsledném uložení formuláře smazány.

Tento stav nebyl vyhovující z hlediska použitelnosti ani funkčnosti. Bylo proto nutné udělat návrh pro novou práci se soubory ve formulářích, který bude rovněž podporovat editaci souborů. Pro tyto účely jsem navrhnul řešení, které funguje na bázi sledování stavu jednotlivých souborů a tyto stavy reflektuje



Obrázek 4.8 Snímek obrazovky ukazující přepracovanou komponentu pro práci se soubory

do uživatelského rozhraní. Po uložení proběhne synchronizace souborů s API a až tehdy jsou změny uloženy.

Pro potřeby sledování stavu jednotlivých souborů bylo do frontendu přidáno rozhraní `SosFile`, které reprezentuje soubor uložený ve formuláři, nebo na serveru. Pro potřeby formuláře byl rovněž přidán typ `FileWrapper`, který kromě souboru obsahuje pole `FileState`, které označuje stav souboru ve formuláři. Tento stav je reflektován do barvy a dostupných akcí, jak je vidět na snímku obrazovky 4.8. Šedý řádek představuje soubor, který je nahraný na serveru a po uložení tam zůstane. Červený znázorňuje soubor, který bude po uložení smazán, a zelený zase soubor, který bude nahrán na server. Samotnou synchronizaci souborů s API poté můžete vidět ve výpisu kódu 4.4. Jedná se o funkci **syncFiles**, která podle stavů jí předaných souborů zajistí synchronizaci oproti poskytnuté URL adrese serveru. Poté vrátí nahrané soubory pro zpětné zobrazení v uživatelském rozhraní. V rozhraní jsou poté nahrané soubory zobrazeny tmavě šedou barvou.

4.3.6 Souhrn iterace

V této iteraci vznikla pro nový systém SOS řada nových funkcionalit a byly vylepšeny funkcionality, které se již v systému nacházely. Převážně se jednalo o předělání formulářů týkajících se hodnocení a také přepracování komponenty pro práci se soubory.

4.4 Třetí iterace – hodnocení

4.4.1 Analýza

Hodnocení v systému SOS slouží pro udělení bodů a případné penalizace za odevzdané řešení. Provádí jej vyučující a to vždy k odevzdání, které čeká na ohodnocení. V současném systému lze rovněž nakonfigurovat, kolik bodů

za odevzdání pro kontrolní bod musí tým dostat, aby byl považován za splněný. Jak již bylo zmíněno v druhé iteraci 4.3, vyučující nyní nemá explicitní možnost vrátit odevzdání, které nesplňuje kritéria k přepracování. Jediný způsob, jak toho lze docílit, je udělit bodové hodnocení nižší než stanovený limit.

Vyučující má kromě udělení hodnocení v systému možnost hodnocení upravit, popřípadě uložit jako koncept, pro pozdější revizi a dokončení. Takto uložené hodnocení není pro studenty viditelné a nemění stav příslušného odevzdání – to zůstává ve stavu *čeká na ohodnocení*.

V současném systému nemá vyučující možnost přiložit k hodnocení žádné soubory. Jak plyne z rozhovorů a také z tabulky 2.1, tak se jedná o poměrně žádanou funkcionalitu pro uživatele z řad vyučujících. Přílohy k odevzdání poskytují možnost, jak opravit odevzdané soubory v nástroji tomu určeném a poté je týmu poslat se zpětnou vazbou. Tato funkcionalita je ve stávajícím systému připravena, avšak v době psaní tohoto textu není známo, jestli a kdy bude dokončena.

Z rozhovorů ohledně hodnocení dále vyplynulo, že navigace do formuláře je pro vyučující náročná a při přechodu do hodnocení ztrácí kontext o projektu. Celý průchod touto sekcí je vidět na obrázku z analýzy 2.2. Pro

Bodové ohodnocení v aktuálním systému poté může projít přes tzv. *přerozdělení bodů*, tomu se však věnuji až v následující iteraci.

4.4.2 Funkční požadavky

FP-1 – Přidání hodnocení k odevzdání

Systém umožňuje vyučujícímu ohodnotit studentem odevzdané řešení. Toto řešení musí být ve stavu *čeká na ohodnocení*, jinak nelze hodnocení udělit. Hodnocení obsahuje detailní bodové ohodnocení, včetně bonusových bodů, popřípadě bodové penalizace. Dále hodnocení obsahuje textovou poznámku pro studenty a také soukromou poznámku pro vyučujícího. Kromě textových údajů systém umožňuje vyučujícímu nahrát k hodnocení přílohy. Vyučující má také možnost nevyhovující odevzdání vrátit k přepracování. Systém za takto vrácené odevzdání neudělí studentům žádné body a zvýrazní jej v uživatelském rozhraní.

FP-2 – Upravení hodnocení

Systém umožňuje vyučujícímu dodatečně upravit již odeslané hodnocení a to v plném rozsahu. V případě, že vyučující hodnocení uloží jako koncept, tak jej student nevidí a příslušné odevzdání se označí stavem *čeká na ohodnocení*.

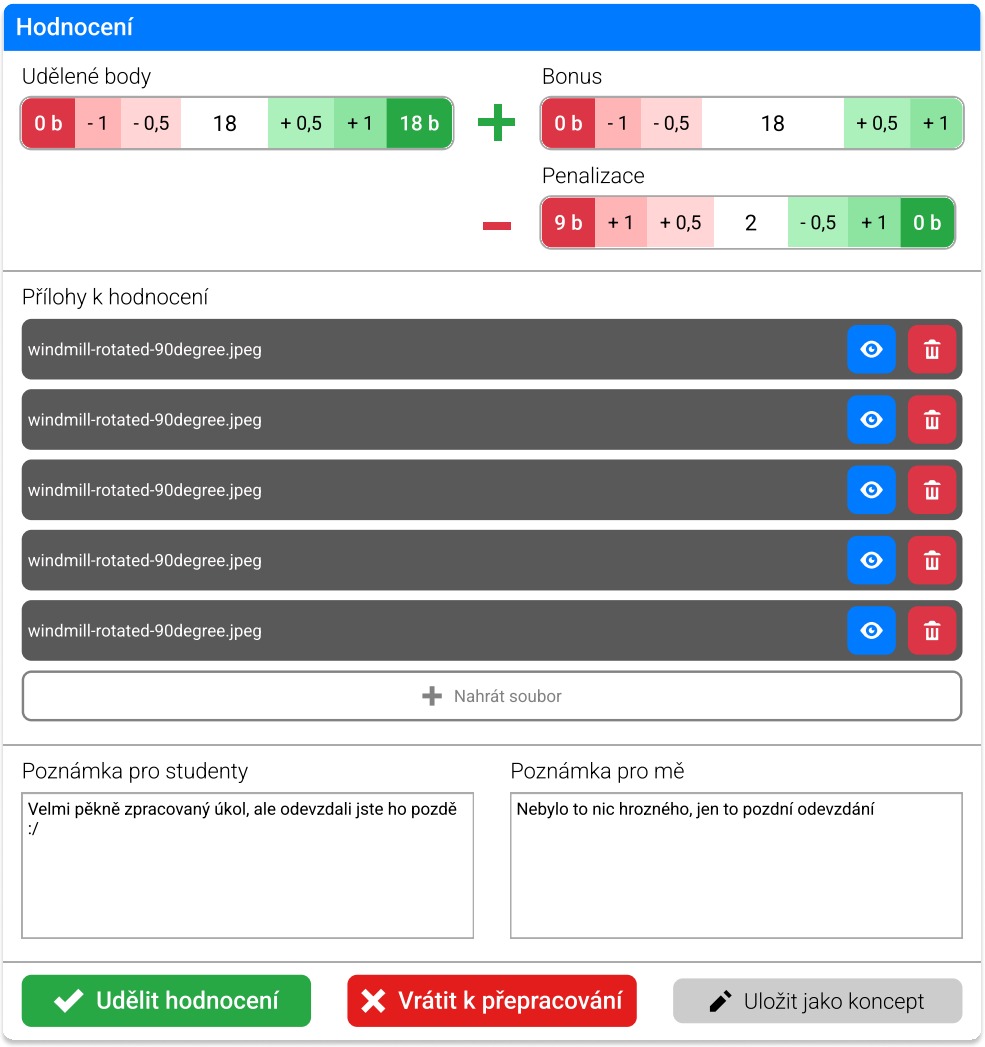
4.4.3 Návrhy iterace

4.4.3.1 Hi-Fi návrh komponenty hodnocení

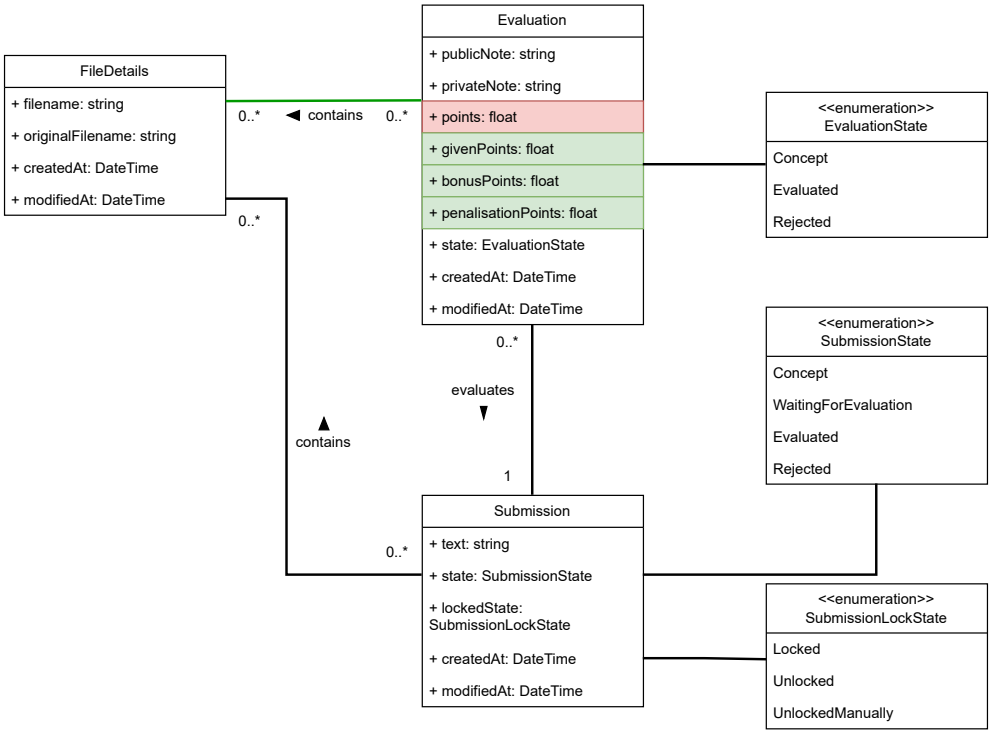
Vzhledem k tomu, že se jednalo pouze o jednu komponentu, tak jsem se rozhodl revidovat a upravit návrh, který byl vytvořen a konzultován s vedoucím práce v BI-SP1.

Vytvořený návrh je přiložen na obrázku 4.9. V budoucí iteraci do něj bude integrováno rovněž přerozdělení bodů. Jak je vidět na návrhu, tak nová komponenta hodnocení umožňuje explicitně vrátit odevzdání k přepracování a tím poskytuje vyučujícímu větší kontrolu nad výsledným stavem.

Dalším rozdílem oproti původnímu hodnocení je možnost přidávat soubory, aby byl úplně splněn funkční požadavek 4.4.2. Přidávání souborů je realizováno komponentou, která byla zpracována v předchozí iteraci.



Obrázek 4.9 Hi-Fi návrh komponenty hodnocení



Obrázek 4.10 Úpravy v doménovém modelu pro sekci hodnocení

4.4.3.2 Změny v doménovém modelu

Pro splnění funkčních požadavků bylo potřeba upravit rovněž doménový model. K hodnocení bylo potřeba přidat detailní rozpis bodového hodnocení, aby bylo možné tyto hodnoty následně upravovat a také přehledně zobrazit uživateli v rozhraní.

Kromě rozšíření bodového hodnocení bylo potřeba také přidat vazbu na soubory, aby mohla být realizována funkcionalita umožňující vyučujícím nahrávat soubory.

Změny v doménovém modelu jsou opět barevně vyznačeny na jeho výseči 4.10.

4.4.4 Výsledná implementovaná část

V rámci této iterace se podařilo implementovat všechny stanovené funkční požadavky. Zároveň proběhl také částečný refaktor komponenty kontrolního bodu. Jednalo se ale pouze o menší refaktor, na který jsem dále navázal v následující iteraci.

Změny v této iteraci prošly rovněž procesem code review a jsou dostupné v následujících merge requestech:

Backend: `https://gitlab.fit.cvut.cz/sos/new-sos/backend/-/merge_requests/62`

Frontend: `https://gitlab.fit.cvut.cz/sos/new-sos/frontend/-/merge_requests/58`

Po nasazení změn na testovací server byla výsledná verze označena štítkem **v0.6.0** a to jak na frontendu, tak na backendu.

4.5 Čtvrtá iterace – přerozdělení bodů

4.5.1 Analýza

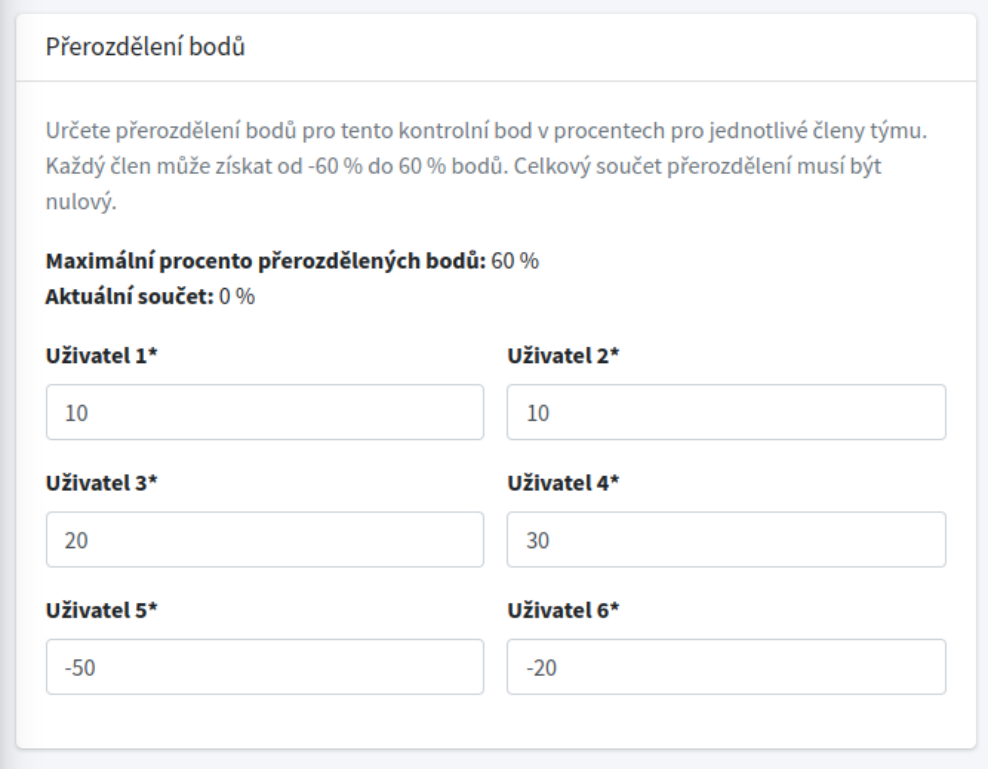
Přerozdělení bodů je funkcionalita, která souvisí s týmovými projekty. Poskytuje členům týmu (převážně vedoucímu) možnost, jak odměnit jednotlivé členy týmu vzhledem k jejich jednotlivým výstupům. Jak vyplývá z názvu této funkcionality, jedná se o přerozdělení, tudíž když některý z členů týmu dostává body navíc, jiný člen týmu musí ztratit stejný počet bodů. Vyučující může dále nastavit hranici pro to, kolik procent může každý člen z týmu ztratit, nebo získat. Ve výchozím nastavení je tato hodnota nastavena na 60%.

V aktuálním systému je přerozdělení řešeno formulářovými poli s přidaným JavaScriptem, který kontroluje, že konečný součet přerozdělených procent je rovný nule. Pro lepší představu je aktuální řešení přiloženo na snímku obrazovky 4.11. V praxi se toto řešení ukázalo jako dostačující, avšak z rozhovorů vyplynulo pár ergonomických vad, na které bych se chtěl v novém řešení zaměřit.

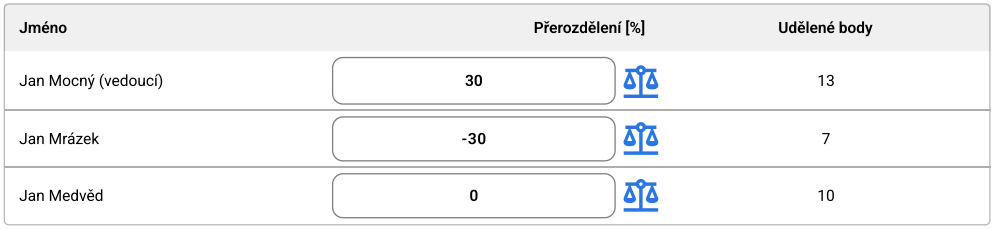
První touto vadou je, že studenti nemají možnost jednoduše vypočítat, kolik bodů jednotliví členové dostanou po udělení hodnocení vyučujícím. Musí odhadnout, kolik procent udělit, aby dosáhli požadované známky z předmětu. Může se zdát, že se jedná o věc, kterou by studenti neměli řešit – měli by se přeci objektivně řídit tím, kdo kolik práce vykonal. V praxi však do tohoto přerozdělení často spadají i mezilidské vztahy v týmu a také snaha jednotlivých studentů si vylepšit průměr. Tím se poté dostávají do situace, kdy by bylo dobré mít jednoduchou kalkulačku pro výsledné body. Do této kalkulačky by student zadal předpokládaný bodový zisk a systém by poté vypočítal body pro jednotlivé členy týmu po přerozdělení.

Druhou, byť drobnou vadou, je absence rychlé možnosti pro vyrovnání součtu přerozdělených procent. Přerozdělení bodů se často dělá postupně, přičemž u posledního člena je jasné, kolik procent mu bude uděleno. Pro tyto případy by bylo žádoucí přidat jednoduchou funkcionalitu na vyrovnání součtu procent na nulu.

Poslední nevýhodou, která ale zasahuje i do nastavení projektu, je samotný limit přerozdělených bodů. Ten limituje, kolik procent může dostat nebo ztratit každý člen v týmu. Když ale nastane situace, že ve 4členném týmu tři členové neudělají nic, ale poslední člen odpracuje celou iteraci sám, dostává se systém



Obrázek 4.11 Aktuální přerozdělení bodů (snímek obrazovky)



Obrázek 4.12 Komponenta přerozdělení

do úskalí. Jelikož horní i spodní hranice je dána předchozí konfigurací, tak i kdybychom zbylým třem členům odebrali všechny body, tak poslední člen nemůže získat více, než maximální počet procent, který je udaný konfigurací. Tato situace však často nenastávala, neaktivní členové spíše opustili tým, nebo byli označení jako neaktivní a nebyly jim uděleny žádné body. Je však dobré mít tuto situaci na paměti, hlavně pro další iteraci.

4.5.2 Funkční požadavky

FP-1 – Definování přerozdělení u odevzdání

Systém umožňuje studentům definovat přerozdělení pro jednotlivé členy. Přerozdělení je definováno jako procentuální zisk, respektive ztráta pro každého člena projektu. Systém kontroluje případné nastavení mezní hodnoty přerozdělení, a také zda je celkový součet přerozdělených bodů nulový. V případě porušení zobrazí systém chybovou hlášku.

FP-2 – Upravení přerozdělení vyučujícím

Systém umožňuje vyučujícímu upravit studenty navržené přerozdělení bodů. Systém kontroluje stejné parametry, jako u studenta, tedy mezní hodnoty přerozdělení a celkový součet. V případě porušení těchto omezení zobrazí systém chybovou hlášku.

4.5.3 Návrhy iterace

Hi-Fi návrh komponenty přerozdělení

Pro potřeby přerozdělení byl proveden opět pouze Hi-Fi návrh komponenty, která bude přerozdělení obstarávat. Jedná se o komponentu, která se bude vykreslovat u odevzdání a hodnocení v případě, že bude přerozdělení povoleno v nastavení kontrolního bodu. Návrh této komponenty je vidět na obrázku 4.12. Pro potřeby rychlého vyrovnání na nulový součet je u každého studenta přidán symbol vah, který po kliknutí vyváží součet procent přerozdělení na nulu.

Wireframy

Pro potřeby budoucí implementace nastavení a také pro znázornění zakomponování komponenty do výsledných obrazovek byly dále upraveny wireframy pro odevzdání a hodnocení. Tyto wireframy jsou dostupné v příloze tohoto textu pod označením A.7-A.10.

Změny v doménovém modelu

Pro potřeby přerozdělení bodů bylo potřeba vymyslet, jak přerozdělení bodů ukládat. V aktuálním řešení se pro přerozdělení bodů používá pole v databázi, které je datového typu *jsonb*. Do tohoto pole se serializuje Python *dict*, což je datová struktura běžně známá jako mapa. Ta obsahuje jako klíč identifikátor uživatele a jako hodnotu procenta, která mu byla udělena.

Toto řešení, ač funkční, může vést k chybám, jelikož datová struktura není kontrolována na úrovni databáze cizími klíči do tabulky uživatelů. Když by se stalo, že se do objektu uloží špatný identifikátor, tak databáze nemá žádný mechanismus, jak tuto neplatnou hodnotu zjistit a zabránit jejímu uložení.

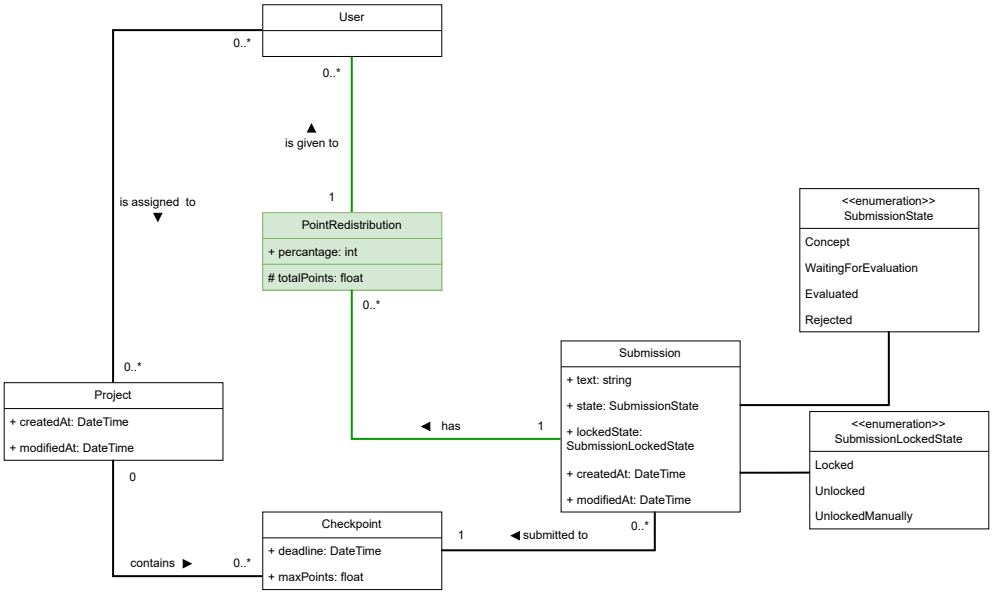
Kvůli tomu jsem se rozhodl pro klasickou reprezentaci formou tabulky, která obsahuje procentuální zisk / ztrátu uživatele, na kterého je navázaná. Dále obsahuje také volitelnou hodnotu *totalPoints*, která značí celkový bodový zisk uživatele po přerozdělení bodů, aby nemusela být tato hodnota vypočítávána stále dokola.

Je také nutné říci, že tento návrh je víc připravený na možné změny, které mohou přijít. Mezi tyto změny můžeme například zařadit možnost historie v případě, že student dostane od vyučujícího jiné procentuální přerozdělení, než které navrhnul tým. Tento případ zatím není řešený, ale v případě jeho implementace by stačilo upravit přidat další sloupec, který by značil hodnotu skutečně dosaženou po hodnocení.

4.5.4 Výsledná implementovaná část

Při implementaci přerozdělení jsem narazil na frontendu na určitou limitaci, kterou představovala struktura komponent, která byla na frontendu implementována. Vzhledem k mnoha iteracím týkajících se této sekce vznikl poměrně hluboký strom komponent a bylo čím dál obtížnější spravovat stavy, které souvisely s celým kontrolním bodem, ale měnily se až v některém z potomků. Pro aktualizaci všech dat se používaly tzv. *eventy*, které se definovaly v komponentě, která stav měnila, a každá komponenta je poté přeposílala do jejího rodiče, nebo jej zpracovávala. Tento problém se označuje jako **event bubbling** [103].

Kromě event bubblingu se vyskytnul také „opačný“ problém, který souvisel s předáváním dat (tzv. *props*) do potomků. Tato data se často jen předávala další komponentě, kterou daný potomek vykresloval, a to i přesto, že samotná



Obrázek 4.13 Změny v doménovém modelu pro potřeby přerozdělení

komponenta s těmito daty jiným způsobem nepracovala. Tento problém se označuje jako **prop drilling** [104].

Bylo tedy nezbytné pro další snadnou rozšiřitelnost tento kód refaktorovat a vymyslet řešení, které centralizuje správu a modifikaci stavů, které se využívají napříč celým stromem. Vue.js pro tyto problémy naštěstí nabízí nativní řešení a to ve formě direktiv *provide* a *inject*. Myšlenka tohoto návrhu je taková, že kořenová komponenta slouží jako poskytovatel dat a centrální místo pro správu stavů, které pomocí direktivy *provide* poskytuje všem potomkům a to nezávisle na jejich hloubce. Když poté potomek potřebuje nějaká data, tak místo přímo předaných *props* využije direktivu *inject*, která poskytne data dodaná kořenovou komponentou.

Díky tomuto refaktoru se omezilo ukládání stavu a jeho správa do jedné komponenty, která pomocí *provide* poskytuje jak data, tak metody pro aktualizaci těchto dat. Díky tomu jsou data v celém kontrolním bodu konzistentní a budoucí rozšíření bude méně náchylné na chyby.

Část refaktoru je vidět také v ukázce kódu 4.5. Zde je vidět využití direktivy *provide* a to po načtení dat odevzdání. V celém stromu komponent je poté možné direktivou *inject* s odpovídajícím klíčem tato data načíst. Všimněte si také direktivy *readonly*, která obaluje reaktivní proměnnou zámkem proti úpravám. Díky tomu nemůže žádný potomek tato data změnit přímo, ale musí využít poskytnutých funkcí, například *addSubmission*.

Po vyřešení tohoto problému proběhla implementace přerozdělení bez problémů. Změny opět prošly vzájemným code review a jsou k dispozici v následujících MR:

Backend: `https://gitlab.fit.cvut.cz/sos/new-sos/backend/-/merge_requests/69`

Frontend: `https://gitlab.fit.cvut.cz/sos/new-sos/frontend/-/merge_requests/63`

Po nasazení na testovací server byla tato verze označena štítkem **v0.7.0** v obou projektových repozitářích.

4.6 Pátá iterace – dokončení nedokončených požadavků

Cílem této iterace bylo dokončit potřebné funkcionality, které jsem nestihl implementovat v rámci ostatních iterací. Jednalo se hlavně o:

1. Nastavení přerozdělení 4.5
2. Možnost uložení konceptu odevzdání 4.3.2
3. Validace dat u přerozdělení a hodnocení
4. Autorizace akcí na backendu

4.6.1 Analýza nastavení přerozdělení

Pro potřeby přerozdělení bodů je nutné v systému mít i nastavení, které toto přerozdělení bude povolovat, popřípadě omezovat. V aktuálním systému probíhá nastavení ve dvou rovinách. První nastavení je příznak u kontrolního bodu, který určuje, zdali bude možné u kontrolního bodu přerozdělit body.

Druhé nastavení je maximální procento přerozdělených bodů, které se provádí pro celý projekt. Toto nastavení určuje celočíselnou hodnotu, která posléze omezuje, kolik procent může každý člen týmu získat, nebo ztratit. V tomto případě se však dá narazit na situaci, kdy máme například 4členný tým, ale dva z členů vůbec nepracovali. To znamená, že by neměli dostat za iteraci žádné body, ale to aktuální systém nepodporuje.

Po konzultaci s vedoucím projektu Ing. Jiřím Hunkou jsme došli k závěru, že toto omezení by mělo být volitelné a systém by měl umožnit také „neomezené“ přerozdělení. To by mělo umožnit vzít členovi týmů všechny body za iteraci a zároveň udělit jinému členovi i více, než 100 % bonusových bodů. Tímto lze také vyřešit situaci, kdy se tým domluví, že někteří členové budou vypracovávat například první iteraci, ostatní zase druhou.

4.6.2 Funkční požadavky

FP-1 – Uložení konceptu odevzdání

Systém umožňuje studentovi vytvořit koncept odevzdání. Koncept je pro vyučujícího neviditelný a slouží pro týmové účely, kdy každý z týmu může do konceptu přidat vlastní výstup. Tento koncept je poté možné odevzdat. V případě, že je u kontrolního bodu vytvořený koncept odevzdání, systém neumožní vytvořit další odevzdání.

FP-2 – Povolení přerozdělení pro kontrolní bod

Systém umožňuje vypnout nebo zapnout přerozdělení bodů pro kontrolní bod. V případě vypnutého přerozdělení systém skryje komponentu pro zadání přerozdělení a veškeré body kontrolního bodu se zobrazují bez přepočítání. V případě zapnutého přerozdělení systém zobrazuje uživateli vždy body, které mu náleží po přerozdělení a rovněž zobrazuje informaci, jaké přerozdělení náleží jednotlivým členům. Kromě zobrazení těchto informací systém umožňuje přerozdělení definovat formou komponenty z předchozí iterace.

FP-3 – Nastavení omezení přerozdělení

Systém umožňuje volitelně omezit maximální procento přerozdělených bodů, které může každý člen z týmu dostat. Toto omezení je určeno formou čísla, ale je možné jej také vypnout a neomezovat maximální hodnotu přerozdělených bodů. V případě omezení maximální hodnoty se v příslušných formulářích hodnota omezí a systém zobrazí hlášku v případě, že bude nastavena vyšší procentuální hodnota přerozdělení, než je povolena.

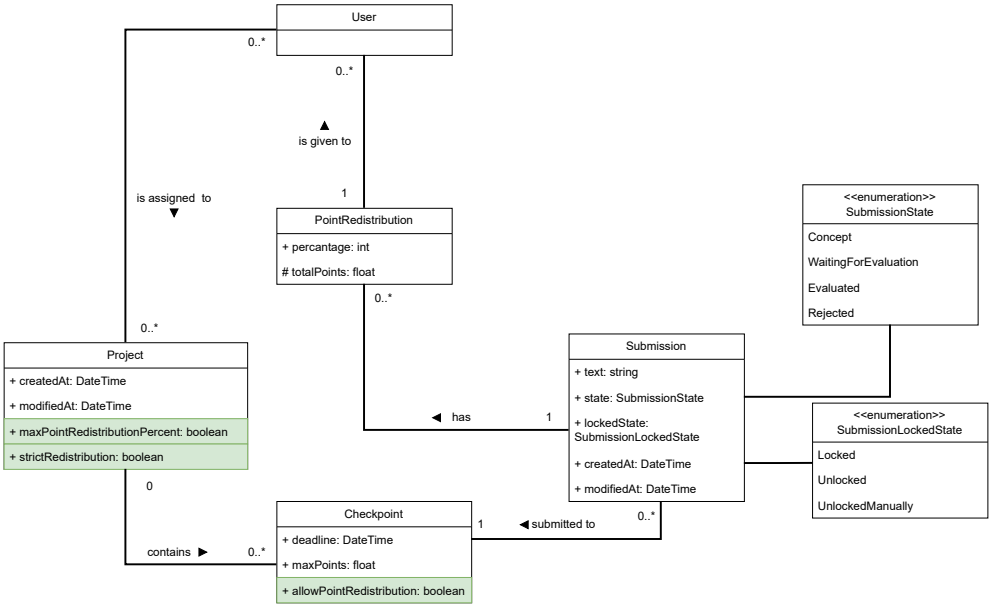
4.6.3 Návrhy iterace

Wireframy

V rámci této iterace byly navrženy wireframy pro nastavení přerozdělení na projektu A.9 a také na kontrolním bodu A.10.

Změny v doménovém modelu

Pro potřeby nastavení přerozdělení bylo potřeba trochu upravit doménový model. Tyto úpravy znázorňuje výseč z doménového modelu 4.14. Jedná se pouze o přidání příznaku ke kontrolnímu bodu a také přidání maximální hodnoty přerozdělených procent k projektu. Kromě toho byl k projektu přidán také příznak *strictRedistribution*, který určuje, zdali má systém omezit hodnotu přerozdělených procent či nikoliv.



Obrázek 4.14 Výseč doménového modelu se změnami pro nastavení přerozdělení bodů

4.6.4 Výsledná implementovaná část

V této části byly splněny a dodělány všechny požadavky, které jsem si v iteracích této práce stanovil. Jelikož se také také jedná o poslední verzi před uživatelským testováním, tak přikládám také snímky obrazovky mnou vypracované části systému. Tyto snímky obrazovky jsou dostupné v příloze B.

Část autorizace, kterou jsem provedl u kontrolních bodů je vidět také v ukázce kódu 4.6. Tato autorizace využívá `HttpContextu` k získání ID příslušného zdroje. Tento zdroj poté načte z databáze a provede potřebnou kontrolu oprávnění. Díky umístění do samostatných tříd je autorizace oddělena od zbytku logiky a tudíž lehce upravitelná.

Implementované části této iterace jsou dostupné v MR:

Backend: `https://gitlab.fit.cvut.cz/sos/new-sos/backend/-/merge_requests/76`

Frontend: `https://gitlab.fit.cvut.cz/sos/new-sos/frontend/-/merge_requests/67`

Po nasazení na testovací server byla tato verze označena štítkem **v0.9.0** v obou projektových repozitářích.

```
export const syncFiles = async (
    files: FileWrapper[],
    fileUploadUrl: string
): FileResponse[] => {
  const { $api } = useNuxtApp();
  const deleteFileRequests = [];
  const formData = new FormData();
  for (const fileWrapper of files) {
    if (fileWrapper.state === "Keep") continue;
    if (fileWrapper.state === "Delete") {
      deleteFileRequests.push($api(
        ApiPath.Files.Download(fileWrapper.file.id), {
          method: "DELETE",
      }));
    }
    if (fileWrapper.state === "New") {
      const file = await fileWrapper.file
        .getDownloadObject();
      formData.append("files", file);
    }
  }
  await Promise.all(deleteFileRequests);
  const newFiles = await $api<FileResponse[]>(
    fileUploadUrl, {
      method: "POST",
      body: formData,
    });
  const remainingFiles = convertWrapperToFile(
    files.filter((file) => file.state === "Keep"),
  );
  return [...remainingFiles, ...newFiles];
};
```

Výpis kódu 4.4 Ukázka funkce pro synchronizaci souborů s API

```
watch(
  submissions,
  () => {
    if (!submissions.value) return;

    provide(checkpointSubmissionsKey, {
      submissions: readonly(submissions),
      addSubmission,
      evaluateSubmission,
      updateSubmission,
      submissionLockStateChange,
    });
  },
  { immediate: true, once: true, deep: true },
);
```

Výpis kódu 4.5 Ukázka provide direktivy

```
protected override async Task HandleRequirementAsync(
    AuthorizationHandlerContext context,
    CheckpointSubmitRequirement requirement)
{
    var checkpointId = PathParamExtractor.
        ExtractIdFromAuthorizationContext
        (
            context,
            "checkpointId"
        );

    var checkpoint = await _dataContext.Checkpoints
        .Where(checkpoint => checkpoint.Id == checkpointId)
        .Include(checkpoint => checkpoint.Project)
        .ThenInclude(project => project.Team)
        .ThenInclude(team => team.Members)
        .FirstOrDefaultAsync();
    Guard.Against.NotFound(checkpointId, checkpoint);

    var currentUser = await _currentUser.GetUser();

    if (checkpoint.Project.IsAssignee(currentUser) &&
        currentUser.Role == Role.Student)
    {
        context.Succeed(requirement);
    }
    else
    {
        context.Fail();
    }
}
```

Výpis kódu 4.6 Ukázka autorizace úpravy kontrolního bodu

Kapitola 5

Příprava aplikace k provozu a dalšímu rozvoji

V této kapitole se věnuji přípravě projektu pro další rozvoj. Rozebírám zde připravené repozitáře a automaticky prováděné úkony, které zajišťují kvalitu kódu. Dále popisuji proces dokumentace a automatického nasazení na testovací server. Věnuji se zde také zapojení BI-SP1 2024 týmu do vývoje.

5.1 Další rozvoj systému

Pro efektivní rozvoj projektu bylo potřeba připravit především automatické úkony prováděné v GitLabu a také dokumentaci. Díky automatickým testům a kontrole kvality kódu bude větší jistota, že nasazovaný kód splňuje definované standardy a nenarušuje funkčnost již implementovaných částí. Zároveň díky automatickému nasazení nemusí vývojáři řešit manuální stahování kódu přímo na serveru.

5.1.1 Repozitáře

Pro potřeby vývoje jsou připraveny dva GitLab repozitáře. Přístup do těchto repozitářů má vedoucí práce, asistent projektu Bc. Max Hejda, členové týmu BI-SP2 2023 a nově také členové týmu BI-SP2 2024.

5.1.1.1 Backend repozitář

V tomto repozitáři se kromě kódu nachází také základní příručka týkající se zprovoznění projektu pro lokální vývoj. Ta je udělána formou markdownového souboru **running-project.md**, na který lze přejít rovněž z úvodního **README.md** souboru. Toto rozdělení je z důvodu online dokumentace, které se věnuji v další sekci. Repozitář obsahuje také definici GitLab CI pi-

peline. Tato pipeline se spouští při každém pushi do repozitáře a provádí následující úkony:

1. kompilace projektu pomocí *dotnet build*,
2. kontrola kvality kódu pomocí nástroje *dotnet format*,
3. automatické testy pomocí *dotnet test*,
4. sestavení produkčního obrazu (pouze ve větvi master nebo při vytvoření tagu),
5. nasazení sestaveného obrazu na server (pouze master nebo při vytvoření tagu),
6. nasazení dokumentace na FIT Pages (pouze master nebo při vytvoření tagu).

Takto definovaná pipeline je dostatečná pro zapojení dalších studentů do projektu a také pro potřeby dalšího rozvíjení při zachování kvality kódu a funkčnosti již implementovaných funkcionalit.

Pro potřeby splnění požadavku dokumentace je touto pipeline vygenerována také kompletní API dokumentace, která je nahrána na FIT Pages[1].

5.1.1.2 Frontend

Ve frontendovém repozitáři je rovněž obsažen readme soubor s instrukcemi pro lokální vývoj a také definice GitLab CI pipeline. V této pipeline se provádí následující úkoly:

1. kontrola stylu kódu *prettier*,
2. spuštění linteru *eslint*,
3. statická typová kontrola *typescript*,
4. sestavení produkčního obrazu (pouze ve větvi master a tagy),
5. nasazení sestaveného obrazu na server (pouze master a tagy).

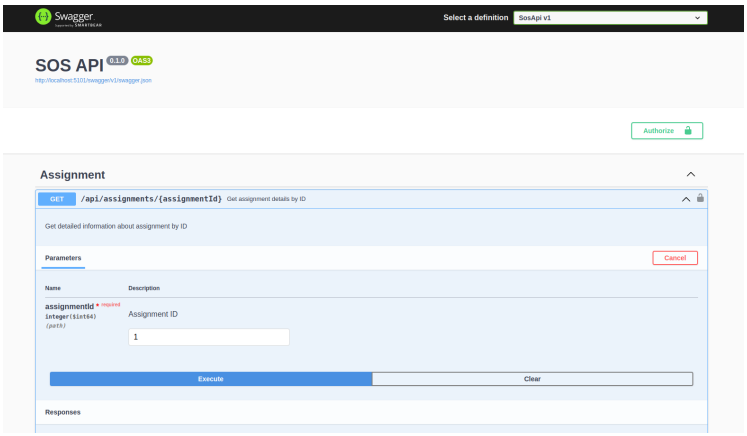
5.1.2 Dokumentace

Dokumentace projektu má dvě úrovně, první z nich je API dokumentace backendového projektu, druhá je samotná vývojářská dokumentace. Dokumentace je zprovozněná formou volně dostupného webového rozhraní[2]. Pro sestavení této dokumentace se používá nástroj *docfx*[3].

[1] Dokumentace je dostupná na adrese `https://sos.pages.fit/new-sos/backend/index.html`

[2] Dostupné na adrese `https://sos.pages.fit/new-sos/backend/index.html`

[3] `https://dotnet.github.io/docfx/`



Obrázek 5.1 Ukázka webového rozhraní pro lokální testování API

5.1.2.1 API dokumentace

API dokumentace obsahuje všechny relevantní informace pro použití REST API backendové aplikace. V této sekci dokumentace může uživatel nalézt definice API endpointů. Ty kromě URL cesty a obsahují také výčet všech HTTP kódů, které vrací, a také formát odpovědi v případě, že je akce úspěšná.

Tato dokumentace je sestavovaná nástrojem Swagger[4]. Jedná se o populární nástroj pro tvorbu API dokumentací. Swagger je v základu integrován do projektu SosApi a díky tomu nebylo potřeba moc věcí konfigurovat. Tento nástroj je dostupný i při lokálním vývoji pro potřeby potřeby testování API. Pro tyto potřeby nástroj poskytuje interaktivní webové rozhraní, které je ukázáno na snímku obrazovky 5.1. V dokumentaci vystavené na FIT Pages je tato API dokumentace dostupná pod záložkou **Swagger API**.

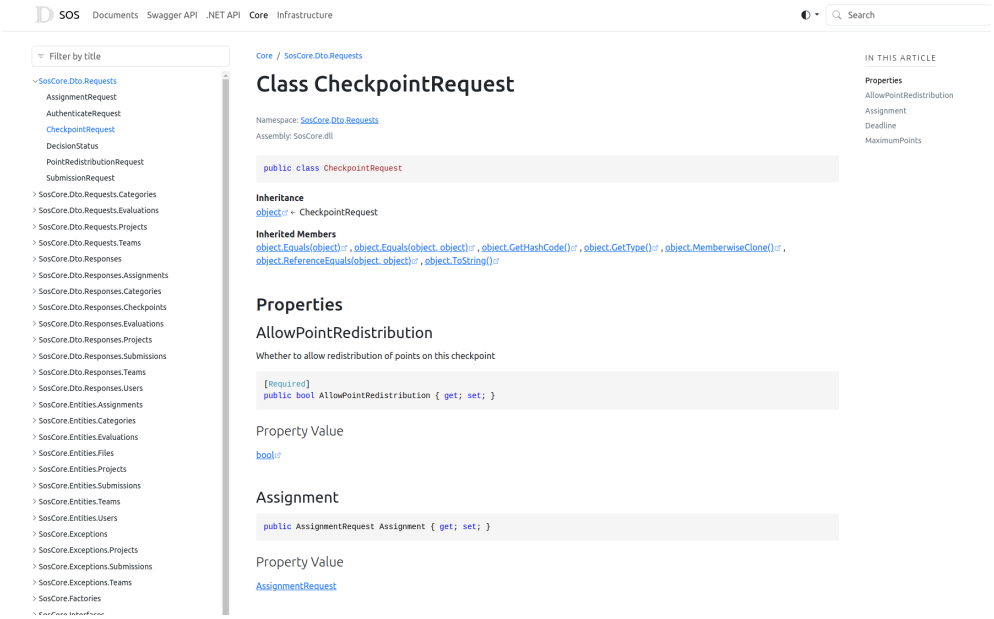
5.1.2.2 Vývojářská dokumentace

Pro vývojáře byla sestavena také vývojářská dokumentace relevantních částí kódu. Ta je dostupná formou XML po kompilaci projektu a je rovněž součástí dokumentace dostupné na webu, jak je vidět na snímku obrazovky 5.2.

5.1.3 Zapojení dalších studentů do projektu

Kromě týmových kolegů, se kterými vypracováváme bakalářské práce, se do projektu zapojil také BI-SP1 tým. Tento tým se skládal z celkem 5 členů, konkrétně Františka Holého, Matěje Hrbáčka, Matěje Procházky, Oldřicha Macháčka a Saši Vobořila. Tento tým měl pro projekt velký přínos a to hlavně při přípravě dokumentace, různých analýz a také při uživatelském testování.

[4] `https://swagger.io/`



Obrázek 5.2 Ukázka webové dokumentace kódu

Dva studenti z tohoto projektu se rovněž zapojili do tvorby balíčku pro poskytnutí snadných filtrovacích funkcí nad databází – **Filter bundlu**[5]. Cílem tohoto projektu je vytvořit NuGet balíček, který poskytne snadnou a rychlou možnost, jak přidat filtrování entit pomocí URL. V současné době zatím není dokončený a na jeho vývoji se stále aktivně pracuje. Po dokončení bude využitelný také v DBS Portálu, který se rovněž přepisuje do nových technologií.

5.1.3.1 Realizované úkoly na novém portálu

Mezi úkoly, které SP tým realizoval, patří především zprovoznění dokumentace. Kromě dokumentace se SP tým věnoval implementaci temného režimu na frontendu, opravě Sentry a několika menším úkolům.

5.1.3.2 Aktuální úkoly

Aktuálně se SP tým zabývá úkoly, které se týkají důležitých funkcí portálu. Mezi tyto funkce patří hlavně již zmíněný filter bundle, přihlášení pomocí Google OAuth a FIT Auth. Kromě těchto funkcí je rozpracované také integrační testování aplikace. Dále se tým bude zabývat nedostatky a nedokončenými úkoly, které vyvstaly z našich bakalářských prací.

[5] Dostupný v GitLab repozitáři `https://gitlab.fit.cvut.cz/common-projects/filter-bundle`

5.2 Příprava projektu na provoz

Aby mohl být projekt spuštěn, muselo být vyřešeno nasazení do produkčního a testovacího prostředí. Vzhledem k celkovému stavu portálu, kterému se věnuji v kapitole 7, není zatím produkční prostředí k dispozici, nasazení tedy probíhá pouze do testovacího. Až budou vyřešeny všechny funkční požadavky a provedeno akceptační testování spolu s migrací dat, tak teprve může započít zprovoznění produkčního prostředí a nasazení systému do provozu.

5.2.1 Nasazení

V rámci BI-SP2 a této práce jsem spolu s Bc Maxem Hejdou připravil infrastrukturu pro CD. Nasazení probíhá na stejný server poskytnutý pro testovací účely starého systému SOS. Pro běh aplikace využíváme platformu Docker a Docker compose, kterým se více věnuji v sekci 2.3.6. Níže popsaný proces funguje analogicky pro frontendový i backendový repozitář

Po sloučení změn do větve master se pomocí pipeline sestaví obraz kontejneru. Tento obraz se uloží do registru kontejnerů, který slouží jako úložiště pro všechny obrazy sestavované v rámci GitLabu. Po nahrání do registru kontejnerů proběhne připojení na cílový server pomocí *ssh*, kde je spuštěn skript na stáhnutí nového obrazu a spuštění kontejneru. Backendový obraz obsahuje navíc spuštění databázových migrací, které jsou uloženy v migračním *bundle*.

5.3 Aktuální stav sekce hodnocení a odevzdání

Zadáním této práce bylo přepracovat sekce odevzdání a hodnocení do nového systému a nasadit do produkčního prostředí. Výsledek mojí práce byl nasazen na testovací prostředí, které je dostupné na adrese `https://new.sos2.ksi.fit.cvut.cz/`. Přihlašovací údaje jsou v případě potřeby dostupné na vyžádání. Produkční prostředí zatím není vzhledem ke stavu projektu připravené. Mnou zhotovená sekce je však dokončená a po zprovoznění produkčního prostředí projektu může být bez problémů nastazena i tam.

Kapitola 6
Testování aplikace

V této kapitole se věnuji testování aplikace a způsobům, které jsem pro toto testování zvolil. Věnuji se zde automatickým testům, spouštěným v GitLab CI, a také uživatelskému testování, které proběhlo na mnou zhotovené sekci výsledného systému. Nakonec se věnuji nalezeným nedostatkům a jejich řešení.

Pro testování části aplikace zabývající se odevzdáváním a hodnocením jsem zvolil celkem dvě metody. První metodou, která se provádí automaticky při každém nahrání do repozitáře jsou funkční testy. Ty v současné době probíhají pouze na backendové části systému a testují aplikační logiku. Tyto testy jsou rozdělené do kategorií, které jsou více zmíněny v sekci 1.4.

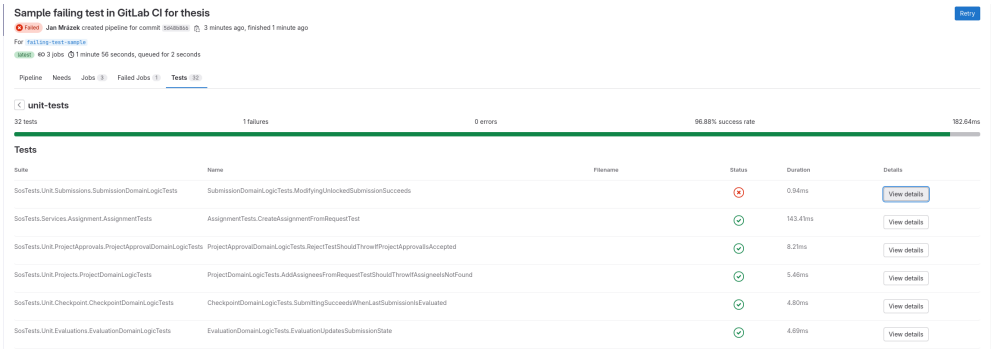
Druhým a pro účely tohoto textu podstatnějším testováním je uživatelské testování použitelnosti. Tímto testováním jsem odhalil nedostatky, které zhoršují UX při práci se systémem. Následně jsem část těchto nedostatků vyřešil a zbylou část zaznamenal formou GitLab issue trackeru.

6.1 Automatické testování

Pro automatické testování byl na backendu zvolen testovací nástroj *xUnit*[1]. Jedná se o velmi populární nástroj pro testování ASP.NET Core aplikací, který se dá využít pro jednotkové, ale i integrační a systémové testy. Tyto testy jsou poté podle principů čisté architektury zahrnuty v separátním C# projektu *SosTests*. Pro potřeby testování jsou v tomto projektu obsaženy také další závislostí, které ovšem nejsou pro běh aplikace potřebné, což je hlavním účelem tohoto oddělení.

Automatické testování probíhá také při každém pushnutí do repozitáře a výstup testu je zobrazen v uživatelském rozhraní GitLabu. Z výsledku testu

[1] `https://xunit.net/`



Obrázek 6.1 Ukázka neúspěšného testu v GitLab CI

jde snadno poznat, který test byl neúspěšný, jak je vidět na snímku obrazovky 6.1.

6.1.1 Jednotkové testy

Jednotkové testy jsou nejrychlejšími a nejmenšími testy. Více se popisu a struktuře těchto testů věnuji v kapitole 1.4.

Pro potřeby hodnocení a odevzdání je většina funkcionalit testována právě jednotkovými testy a nevyžadují tak spuštění celé aplikace, nebo databáze. Díky tomu jsou tyto testy velmi rychlé a zároveň pokrývají většinu aplikační logiky.

V nástroji xUnit se k testovacím metodám přidává atribut `[Fact]` , díky kterému poté knihovna rozezná testy od pomocných metod. Pro potřeby kontroly výstupu, popřípadě vyhození výjimky se poté používají funkce z třídy `Assert` , které při neshodě označí test jako neúspěšný.

6.1.2 Integrační testy

Integrační testy často zahrnují spuštění databáze, nebo jiné části infrastruktury. Pro potřeby otestování této sekce nejsou tyto testy kritické, jelikož převážná většina aplikační logiky se testuje pomocí jednotkových testů. Aktuálně jsme požadavek na zpracování integračních testů předali SP týmu. Kvůli nižší prioritě se však tomuto požadavku zatím nestihli plně věnovat.

6.2 Testování použitelnosti

Pro testování použitelnosti jsem se rozhodl využít moderovaný průchod předem určeným scénářem. Při použití této metody prochází respondent uživatelským rozhraním podle předem připraveného scénáře. Dále je u testování přítomný i moderátor, který se případně respondenta doptává, aniž by jej nějak naváděl k vyřešení úlohy. Roli moderátora jsem při testování zastával sám.

```
[Fact]
public void EvaluationConceptUpdatesSubmissionState()
{
    var submission = new Submission();

    var evaluation = new Evaluation
    {
        Submission = submission,
        State = EvaluationState.Concept,
    };

    evaluation.State = EvaluationState.Evaluated;

    Assert.Equal(SubmissionState.Evaluated, submission.State);
}
```

Výpis kódu 6.1 Ukázka jednotkového testu pro aktualizaci stavu odevzdání

Doptávání může být například na to, co právě uživatel hledá nebo proč danou funkcionalitu hledá na daném místě. Kromě toho moderátor během průchodu zaznamenává i známky neverbální komunikace, díky čemuž může sledovat například nejistotu testovacího respondenta [105]. Může se poté například doptat, zdali je uživateli jasné, co má udělat a případně vyjasnit zadání. Je však nutné myslet na to, aby uživatele nijak nenaváděl k tomu, jak daný úkol vyřešit.

Pro testování jsem se rozhodl použít metodu podle Kruga. Jedná se o metodu, při níž se testují všichni respondenti stejnou verzí uživatelského rozhraní (pro účely tohoto testování verze **v0.9.0**) a poté se z testování vyvodí závěry. Po vyvození závěrů následuje úprava návrhu a jeho implementace, která by měla být poté podrobena dalším testům [105].

Při použití moderovaného průchodu se často řeší počet respondentů. Podle všeobecně uznávaného experta na použitelnost software Jacoba Nielsena, stačí pro odhalení většiny chyb 3-5 respondentů [106].

6.2.1 Příprava

Nejprve jsem zhotovil dva testovací scénáře – jeden pro studenta a jeden pro vyučujícího. Tyto testovací scénáře jsem prošel s vedoucím práce, který poukázal na úpravy, které bylo potřeba ve scénářích udělat, aby bylo testování použitelnosti přínosné. Konkrétně se jednalo o to, že původně navržené scénáře byly příliš návodné tomu, jak má uživatel akci provést. Změnil jsem tedy scénáře tak, aby byl jasný pouze cíl a uživatel se mohl více vžít do situace. Takto upravené scénáře poté vedoucí práce odsouhlasil. Tím vznikla podoba scénářů, které jsou přiloženy níže.

Kromě konzultace scénářů poukázal vedoucí práce na nedostatky, se kterými se při průchodu systémem setkal. Konkrétně se jednalo o:

1. omezené možnosti rozbalování prvků a navigace,
2. složité nahrávání souborů,
3. nevhodně zobrazený stav odevzdání,
4. nezobrazení konkrétního času u zadávání termínu odevzdání kontrolního bodu při zvolené možnosti „Odevzdat do půlnoci“,
5. možnost zadání sekund u termínu odevzdání kontrolního bodu,
6. malý font v tooltipech.

6.2.2 Scénář pro studenty

Pro otestování důležitých funkcionalit z pohledu studenta byl připraven následující scénář. Student při jeho průchodu začínal přihlášením a z úvodní obrazovky.

1. Úkol – Příprava odevzdání

V projektu **Nejlepší projekt** se Vám blíží termín odevzdání **1. Kontrolního bodu**. S týmovými kolegy jste se domluvil, že své podklady nahrají do systému k připravenému odevzdání. Odevzdání bude obsahovat pouze popis pro vyučujícího, který bude obsahovat podtržený nadpis první úrovně “Naše odevzdání”. Uložte odevzdání tak, aby jej vyučující neviděl, ale ostatní členové týmu k němu mohli nahrát své podklady.

> *Tímto úkolem jsem chtěl zjistit, jak intuitivní je vytvoření konceptu odevzdání, které je pro vyučujícího neviditelné.*

2. Úkol – Záchrana odevzdání

Zavolal Vám vyplašený kolega **Josef Student-2** z týmu **Lepší projekt**. Když jste ho trochu uklidnil, tak jste se dozvěděl, že odevzdal velmi nevhodný soubor k **1. Kontrolnímu bodu**. Kdyby tento soubor viděl vyučující, tak nebude příliš nadšený. Jste však zrovna na cestách a nemáte k dispozici správný soubor, musíte tedy zajistit, aby odevzdání vyučující neviděl a vy jste ho mohli později opravit.

> *Tento úkol měl ověřit schopnost studenta najít možnost vrácení odevzdání. Vrácení již odevzdaného řešení zneviditelní toto odevzdání pro vyučujícího a tým neztratí možnost jej později upravit.*

3. Úkol – Opravení odevzdání

Dorazil jste domů z Vaší cesty a jdete opravit odevzdání z úkolu 2. Upravte odevzdání tak, že nahradíte nahraný soubor souborem **odevzdani2.txt**. Kolega Vás tak naštval, že jste rozhodl mu vzít nějaké body za toto odevzdání. Zajistěte, aby platilo následující, když dostanete za odevzdání 10 bodů:

1. Josef Student-0 dostane 12 bodů
2. Josef Student-1 dostane 11 bodů
3. Josef Student-2 dostane zbytek

Takto připravené odevzdání už je připravené na hodnocení vyučujícím. Uložte jej tak, aby jej vyučující mohl ohodnotit.

> *Tímto úkolem jsem chtěl zjistit intuitivnost přerozdělování bodů a bodové kalkulačky, pro zjednodušení výpočtu procent. Dále tento úkol ověřoval, zdali je pro uživatele dostatečně intuitivní komponenta pro práci se soubory.*

6.2.2.1 1. respondent

První respondentem byl bývalý student FIT ČVUT. Na fakultě studoval ještě v dobách, kdy nebyl ve výuce zaveden SOS. Jednalo se tedy o jeho první setkání s tímto systémem a všemi jeho funkcemi. Student není navíc původem z České republiky, což lehce zkomplikovalo porozumění testovacímu scénáři. Nicméně jsem díky tomu zjistil, které pojmy nejsou dobře srozumitelné pro cizince, pro které není čeština mateřským jazykem.

Respondentovi se podařilo úspěšně dokončit úkoly 1 a 3 úspěšně. U druhého úkolu sice dosáhl částečně cíle, nicméně neskryl odevzdání úplně, nýbrž jen smazal nevhodný soubor.

Respondent poukázal na nestandardně umístěné ovládací prvky a místy nejasnou terminologii. Dále také poznamenal, kde by dané ovládací prvky hledal. Jeho zpětná vazba je pro výsledky testování velmi hodnotná, jelikož má sám bohaté zkušenosti s vývojem webových aplikací a má rovněž přesah do UX.

Připomínky respondenta

1. Při vytváření konceptu není dostatečně jasné, že odevzdání tým stále uvidí a může jej měnit.
2. Možnost vrácení odevzdání je dostupná až při úpravě. Není příliš jasné, že tlačítko „Vzít zpět“ vrátí odevzdání do konceptu. Navíc je viditelné až po rozkliknutí formuláře, což pro respondenta působilo zmatečně.

3. DragNDrop pro nahrání souborů přetáhnutím při tomto průchodu nezafungoval a bylo potřeba soubor nahrát přes výběr ze souborů.

4. Nastavení přerozdělení bodů působilo na respondenta zmatečně. Konkrétně se jednalo o způsob orientačního výpočtu bodů, který není dostatečně vysvětlen v uživatelském rozhraní.

6.2.2.2 2. respondent

Druhým respondentem byl student magisterského studia na FIT ČVUT. Konkrétně se jednalo o asistenta projektu SOS. Respondent vypracovával na projektu SOS i jeho bakalářskou práci a má tedy s aktuálním systémem bohaté zkušenosti.

Při průchodu scénářem byla jediným větším zádrhelem práce s přerozdělením bodů. Simulované body, které mají za úkol pomoci s přerozdělením, nebyly na první pohled příliš dobře vidět.

Respondent mimo jiné poukázal, že by možná bylo vhodnější zvolit jiné barvy pro tlačítko „Vzít zpět“ u odevzdání, jelikož se jedná o tlačítko, které slouží spíše jako okrajová funkcionalita. Dále navrhl novou funkcionalitu, která by přerozdělila body zbylým studentům rovnoměrně.

Připomínky respondenta

1. Při braní odevzdání zpět respondent poukázal na doladění barev u možnosti „Vzít zpět“ odevzdání. Nebyl si dále příliš jistý, jestli se akce provedla úspěšně a že odevzdání není pro vyučujícího viditelné.

2. DragNDrop při přetahování souborů opět nefungoval. Zafungoval až na druhý pokus.

3. Respondenta zmátlo nastavení procent, že se dá jít do mínusu. Podotkl, že by čekal na místě nějakou vysvětlivku, jelikož takto nastavení smysl dává.

6.2.2.3 3. respondent

Třetím respondentem byl student bakalářského studia FIT ČVUT. Student se setkal se systémem SOS v rámci předmětu BI-SP1. Shodou náhod v tomto předmětu také pracoval na systému SOS, nepřišel však do kontaktu s touto sekcí, tudíž jeho pohled na toto testování nebyl ovlivněn.

Respondent zvládl projít scénářem bez větších problémů. Nicméně, objevily se opět opakující se nedostatky tohoto rozhraní, které shrnuji v seznamu níže.

Připomínky respondenta

1. Uložení odevzdání jako konceptu nebylo pro respondenta příliš vypovídající. Nebyl si jistý, zda úkol splnil.

2. Simulované body pro přerozdělení jsou zobrazeny až pod tabulkou, což respondenta vedlo k tomu, že je bral jako vstup nesouvisející s tabulkou přerozdělení.

3. DragNDrop pro nahrávání souborů v tomto průchodu také nefungoval. Soubory se musely nahrát ručně přes výběr.

4. Navigace při přechodu na projekt funguje pouze při kliknutí na tlačítko „Zobrazit detail“. Respondent by očekával, že bude fungovat kliknutí na celý blok.

5. Tlačítko „Vzít zpět“ u odevzdání u respondenta evokovalo, že se tím zruší změny u formuláře a nebylo pro něj zřejmé, že se jedná o skrytí odevzdání.

6.2.2.4 4. respondent

Čtvrtým respondentem byla studentka 1. Lékařské fakulty Univerzity Karlovy. Se systémem se nikdy nesetkala. Tímto se dobře vyzkoušelo, jak je rozhraní srozumitelné i studentům z jiných prostředí.

Respondentka prošla scénářem bez větších problémů. Jediný větší problém byl s názvoslovím, konkrétně stavu koncept. Dále bylo odhaleno pár nedostatků, které se ale ve výsledku neliší od těch, které již byly odhaleny dříve.

Připomínky respondenta

1. Názvosloví konceptu bylo nejasné. Respondentka nevěděla, jestli je odevzdání viditelné pro tým, či nikoliv.

2. Nahrání souboru respondentka nejdříve zkusila přetažením přímo na komponentu, bez otevření modálu.

3. Tlačítko „Vzít zpět“ u odevzdání působí vedle tlačítka „Zahodit úpravy“ zmatečně.

6.2.2.5 5. respondent

Pátým respondentem byl týmový kolega, který realizoval správu týmů. S novým rozhraním systému se tedy již setkal a je dobře obeznámený s tím, jak má sekce fungovat.

Respondent dokončil scénář bez problémů, nicméně podotkl několik prvků, které ho při průchodu rozhraním zmátly. Většina nedostatků se opakovala i u předchozích respondentů, jediným novým nedostatkem byla hláška o pozdním odevzdání.

Připomínky respondenta

1. Hláška o pozdním odevzdání se zobrazuje i při vytvoření konceptu. Respondent by čekal, že se hláška zobrazí až po skutečném odevzdání vyučujícímu.
2. Tlačítko na vrácení odevzdání by hledal spíše přímo v komponentě odevzdání a ne v jeho úpravě.
3. U přerozdělení chybí vysvětlivky.

6.2.3 Vyučující

Pro moderovaný průchod s vyučujícím jsem sestavil následující scénář, který obsahuje důležité situací z jeho pohledu. Stejně jako studentský scénář, byl i tento odsouhlasen vedoucím práce.

1. Úkol – Nastavení kontrolních bodů

Jste odpovědný vyučující projektu **Nejlepší projekt**. S týmem, který projekt vypracovává jste se domluvili, že rozdělíte práci do 3 kontrolních bodů.

Všechny kontrolní body budou ohodnoceny maximálně 15 body a umožní studentům rozdělit si body v závislosti na jimi odvedené práci. Kontrolní body nastavte podle následující tabulky. Kurzívou je uvedeno případné formátování, text v závorkách tedy nekopírujte, ale použijte formátování v nich uvedené.

1. Kontrolní bod obsahuje v zadání tučný text **Analýza** a na druhém řádku normální text **Analyzovat aplikaci**. Deadline tohoto kontrolního bodu je konec dne 10.5.2024.

2. Kontrolní bod obsahuje v zadání nadpis první úrovně **Návrh** a na druhém řádku normální text **Navrhnout usecases**. Deadline tohoto kontrolního bodu je konec dne 16.5.2024 v 16:00.

3. Kontrolní bod obsahuje v zadání podtržený nadpis třetí úrovně **Implementace**. Deadline tohoto kontrolního bodu je konec dne 23.5.2024 v 16:00.

> *Tento úkol měl ověřit schopnost uživatele pracovat s textovým editorem a také intuitivnost nastavení kontrolních bodů.*

2. Úkol – Hodnocení

Studenti v projektu **Lepší projekt** odevzdali jejich řešení **1. Kontrolního bodu**. Dnes však nestíháte hodnocení řádně dokončit, ale nechcete přijít o rozpracovanou práci. Do poznámky, kterou studenti neuvidí, napište následující text: **Dokončit hodnocení** a uložte si hodnocení k pozdějšímu dokončení tak, aby jej studenti týmu neviděli.

> *Tento úkol měl ověřit srozumitelnost různých možností ukládání, konkrétně pro studenty neviditelného konceptu.*

3. Úkol – Opravy odevzdání

Studenti projektu **Lepší projekt** se Vám ozvali, že by potřebovali opravit odevzdání u **1. Kontrolního bodu**. Ověřte, že studenti mohou odevzdání upravit a pokud ne, zajistěte, aby tak mohli učinit.

Tento úkol cílil na možnost zamykání odevzdaného řešení na úpravy.

4. Úkol – Dokončení hodnocení

Studenti opravili odevzdání u **1. Kontrolního bodu** v projektu **Lepší projekt**. Je tedy třeba je řádně ohodnotit.

Práce odvedená v rámci tohoto kontrolního bodu byla v pořádku, studentům náleží plný bodový zisk a také 1 bonusový bod. Musíte ale zjistit, jestli studenti odevzdali vypracovanou úlohu pozdě. Pokud ano, tak jim udělte 2 bodovou penalizaci za pozdní odevzdání.

Ze studenty přiloženého souboru **odevzdani.txt** zkopírujte text a vložte jej do poznámky, kterou studenti uvidí.

Pro účely ohodnocení jste připravil soubor **opravene-odevzdani.txt**, ten je potřeba nahrát jako zpětnou vazbu studentům. Z vykázaného času a odvedené práce jste dále zjistil, že **Josef Student-2** téměř nepracoval a nezaslouží si tedy plný bodový zisk. U ostatních dvou studentů byla práce v pořádku a zasloužili by si nějaké body navíc. Až budete mít hodnocení vyplněné, udělte jej studentům, aby jej viděli.

Tento úkol měl především ověřit schopnost vyučujícího ovládat přerozdělení studentských bodů a také manipulaci se soubory.

6.2.3.1 1. respondent

Prvním respondentem z řad vyučujících byl vedoucí katedry softwarového inženýrství. Má zkušenosti se stávajícím systémem SOS, jelikož v posledních třech semestrech vedl tým studentů, kteří odevzdávali své projekty právě přes systém SOS.

Respondentovi se podařilo dokončit všechny úkoly zadané v testovacím scénáři, byly však odhaleny některé nedostatky, které průchod zkomplikovali. Kromě těchto nedostatků hodnotil respondent uživatelské rozhraní pozitivně. Líbilo se mu, že je vše ohledně zadání a hodnocení na jedné stránce a nemusí se proklikávat na několik různých formulářů, což vede na ztrátu kontextu.

Pro mě neočekávaný byl pouze průchod prvním úkolem, který se týkal nastavení kontrolních bodů. Respondenta zmátlo deadline uvedený v hlavičce již existujícího kontrolního bodu a myslel, že nejde změnit. Toto byl jeden z nejvážnějších nedostatků, který byl během tohoto testování odhalen.

Připomínky respondenta

1. Režim úprav kontrolního bodu není dostatečně zvýrazněný.
2. Při úpravách kontrolního bodu působí matečně deadline uvedený v jeho hlavičce, který je pouze pro zobrazení. Samotný výběr deadlinu je ve formuláři níže.
3. Slovo koncept při ukládání konceptu hodnocení není příliš vypovídající.
4. Po uložení hodnocení jako konceptu si nebyl respondent jistý, zdali bylo dosaženo cíle.
5. Po vytvoření kontrolního bodu se neresetuje formulář na jeho vyplnění. Při dalším rozkliknutí jsou potom předvyplněné údaje z předchozího vytvoření.
6. Nastavení konkrétního času deadlinu respondent při prvním průchodu neobjevil, jelikož bylo skryté pod možností „Odevzdat do půlnoci“.

6.2.3.2 2. respondent

Druhým respondentem byl vyučující, který pravidelně se systémem SOS pracuje, jelikož je odpovědný za projekty v BI-SP1 a BI-SP2. Pomáhal také testovat úplně první verzi systému, kterou vyvíjel Ing. Tomáš Pavlůsek.

Respondentovi se podařilo dokončit všechny úkoly bez větších potíží. Poukázal přitom na poměrně značné množství drobností, které ho v novém rozhraní potěšili, jako například podpora markdownu v editoru. Na druhou stranu poukázal i na některé nedostatky a poskytl velmi cennou zpětnou a připomínky.

Připomínky respondenta

1. Formulář při úpravě kontrolního bodu se nezavře. Objeví se sice oznámení o uložení, ale formulář zůstane otevřený.
2. Respondent podotkl, že v hlavičce úpravy kontrolního bodu by rád viděl i body.
3. Reprezentace stavu „čeká na ohodnocení“ se jeví, jakoby se něco načítalo.
4. Nejasná reprezentace konceptu hodnocení.
5. Tlačítko „Zrušit změny“ u formuláře na hodnocení je matoucí, mělo by se zobrazovat pouze když se provedou nějaké úpravy.
6. Při hodnocení se při přechodu na novou stránku neupozorní na rozpracovaný formulář.

7. V indikaci pozdního odevzdání není uvedeno, o kolik pozdě bylo odevzdáno řešení.
8. Při nahrávání souborů je zbytečné otevření modálu.
9. Při přerozdělování bodů by to chtělo vysvětlivku, kolik bodů lze přerozdělit, popřípadě jak přerozdělování funguje.
10. Při přerozdělování bodů není pro vyučujícího jednoduchá možnost, jak se vrátit k tomu navrženému studenty.
11. Při chybě ve formuláři by bylo dobré mít vyskakující chybovou hlášku.
12. Rychlý tip u textového editoru by chtěl mít možnost skrýt.

6.2.3.3 3. respondent

Třetím respondentem byl vyučující, který se se systémem SOS setkal poprvé. Podotkl, že rozhraní je intuitivní a podařilo se mu dokončit všechny úkoly bez větších problémů. Jediným novým poznatkem z tohoto testování byla výrazná hláška o soukromé poznámce hodnocení, kvůli které poté není příliš výrazná ta pro studenty.

Připomínky respondenta

1. Formulář po úpravě kontrolního bodu se nezavře.
2. Koncept hodnocení lze poznat jen podle barvy záhlaví.
3. Soubory nejde prohlížet bez stažení.
4. Soukromá poznámka je u hodnocení více vidět, než ta pro studenty.

6.2.4 Zhodnocení uživatelského testování

Z provedeného uživatelského testování, kterého se zúčastnilo 5 respondentů v roli studenta a 4 respondenti v roli vyučujícího vyplynulo několik nedostatků, které bude potřeba vyřešit. Některé z těchto nedostatků jsou méně závažné, nebo ojedinělé. Vyskytlo se však několik problémů, které se opakovaly napříč scénáři a tím zhoršovaly celkový UX této sekce.

V následující sekci uvádím souhrn nedostatků, které z uživatelského testování vyplynuly a budou řešeny. Tyto nedostatky byly většinou vnímány více respondenty a nejednalo se tedy o ojedinělý případy.

Záznamy všech provedených testů jsou dostupné na vyžádání, nebo do srpna 2024 na fakultním OneDrive.

6.2.4.1 Úprava přerozdělení vyučujícím

Úprava přerozdělení bodu vyučujícím přepíše původně uloženou hodnotu definovanou studenty. Toto je nežádoucí chování, systém by měl tuto historii uchovávat.

Závažnost: vysoká

Řešení: upravení reprezentace přerozdělení bodů z iterace 4.5 tak, aby obsahovala jak procenta nevržená studenty, tak procenta udělená vyučujícím

Náročnost řešení: vysoká

6.2.4.2 Nahrávání souborů

U nahrávání souborů šlo převážně o zbytečné otevření modálu na výběr, popřípadě přetáhnutí souboru. Při přetahování byla dále cílová plocha příliš malá, což zapříčinilo občasné otevření souboru přímo v prohlížeči.

Závažnost: vysoká

Řešení: odstranění modálu, implementace nahrání přetažením rovnou do cílové plochy

Náročnost řešení: střední

6.2.4.3 Zobrazení stavů u odevzdání

Při zobrazení odevzdání ve stavu „Čeká na hodnocení“ nebo „Koncept“ se bodový ukazatel u odevzdání tváří, jakoby něco načítal. Toto je pro většinu uživatelů matoucí.

Závažnost: vysoká

Řešení: nahrazení pohyblivého stavu statickým

Náročnost řešení: nízká

6.2.4.4 Reprezentace konceptů

Pro uživatele není příliš jasný pojem koncept a jeho reprezentace v systému není dostatečně znázorněna. Tento problém je nejvíce patrný u hodnocení.

Závažnost: vysoká

Řešení: doplnění vysvětlivek ke stavu konceptu, stav odevzdání slovně doplnit i do rozbalovací nabídky

Náročnost řešení: nízká

6.2.4.5 Tlačítko na vrácení odevzdání / hodnocení

Pro uživatele je matoucí tlačítko na vrácení odevzdání a hodnocení na spodní části formuláře. Je pro ně často matoucí v kombinaci s tlačítkem „Zahodit úpravy“, které manipuluje s formulářem.

Závažnost: střední

Řešení: přesunutí tlačítka mimo formulář, k tlačítku editace

Náročnost řešení: nízká

6.2.4.6 Přerozdělení bodů

Uživatelé nechápou, jak funguje přerozdělení. Neví, co dělají tlačítka vah a jaké jednotky se rozdělují.

Závažnost: střední

Řešení: přidání jednotky ke vstupu, přidání vysvětlivky k celku a tlačítku vah

Náročnost řešení: nízká

6.2.4.7 Zpoždění u odevzdání

U pozdního odevzdání není zobrazen údaj s konkrétním zpožděním. Mělo by být zobrazeno i o kolik pozdě bylo odevzdání vytvořeno.

Závažnost: střední

Řešení: přidání konkrétního zpoždění k pozdnímu odevzdání

Náročnost řešení: nízká

6.2.4.8 Vysvětlivky k tlačítkům

Mnoho uživatelů se dostalo do situace, kdy chyběly vysvětlivky u tlačítek nebo různých ovládacích prvků. Například se jednalo o tlačítko vah u přerozdělení, tlačítka u souborů.

Závažnost: střední

Řešení: opravení a přidání vysvětlivek k tlačítkům u souborů a přerozdělení

Náročnost řešení: nízká

6.2.4.9 Chyby ve formuláři

U chyb ve formuláři nejsou občas chyby dostatečně vidět, jelikož systém vypadá že neodpovídá.

Závažnost: střední

Řešení: přidání chybové hlášku při chybě ve formuláři, ideálně doplněné o scroll k cílovému prvku.

Náročnost řešení: střední

6.2.4.10 Termín odevzdání kontrolního bodu

Konkrétní čas kontrolního bodu je neviditelný, pokud je zvolená možnost „Odevzdat do půlnoci“. Tento prvek by měl být viditelný vždy, aby bylo jasné, jaký čas bude finálně nastaven. U nastavení času je dále nutné nastavovat i sekundy, což aktuálně nelze.

Závažnost: střední

Řešení: zobrazit čas termínu odevzdání vždy, umožnit zadání sekund

Náročnost řešení: nízká

6.2.4.11 Úpravy kontrolního bodu

Při úpravách kontrolního bodu není jasně vidět, že se kontrolní bod upravuje. Dále se po uložení formuláře formulář nezavře. Některé informace jsou dále duplikovány jak v hlavičce kontrolního bodu, tak i v těle formuláře. Někteří uživatelé je poté zkoušeli upravit v hlavičce.

Závažnost: nízká

Řešení: přidání tlačítka na úpravu, místo šipky. Zajištění zavření formuláře po úspěšném uložení.

Náročnost řešení: nízká

6.2.4.12 Navigace do projektu a kontrolních bodů

Při navigaci do projektu a kontrolních bodů by bylo vhodné, nepoužívat tlačítka, ale celé ovládací prvky. Většina respondentů s tímto však problémy neměla, jedná se proto o méně závažný nedostatek.

Závažnost: nízká

Řešení: přidání odkazů i do komponenty

Náročnost řešení: nízká

6.2.4.13 Rychlý tip u editoru nelze skrýt

Rychlý tip zabírá značnou část místa v každém editoru a nelze jej skrýt

Závažnost: nízká

Řešení: přidání tlačítka na skrytí rychlého tipu

Náročnost řešení: nízká

6.2.4.14 Zobrazení času odevzdání u konceptu

Při vytvoření konceptu se zobrazí komponenta, která indikuje zdali bylo odevzdáno včas či nikoliv.

Závažnost: nízká

Řešení: zobrazit tuto komponentu pouze při dokončeném odevzdání

Náročnost řešení: nízká

6.2.4.15 Viditelnost poznámky pro studenty

Soukromá poznámka je kvůli ohraničení více vidět, než poznámka pro studenty.

Závažnost: nízká

Řešení: orámovat poznámku pro studenty stejným stylem, jako tu soukromou

Náročnost řešení: nízká

6.2.5 Shrnutí nedostatků

Níže uvádím přehled nedostatků také formou tabulky. Některé nedostatky byly ještě v rámci této práce vyřešeny a jsou v tabulce označeny, jako hotové. Ostatní nedostatky byly přeformulovány do podoby GitLab issues a věnuji se jim také v sekci 7.3.1. Po opravení níže zmíněných nedostatků je finální verze aplikace **v0.11.0** a to v backendovém i frontendovém repozitáři. Vybrané opravy jsou rovněž ukázány v příloze C.

Nedostatek	Závažnost	Vyřešeno
Úprava přerozdělení vyučujícím	Vysoká	Ne
Nahrávání souborů	Vysoká	Ano
Zobrazení stavů u odevzdání	Vysoká	Ano
Reprezentace konceptů	Vysoká	Ano
Tlačítko na vrácení odevzdání / hodnocení	Střední	Ne
Přerozdělení bodů	Střední	Ano
Zpoždění u odevzdání	Střední	Ano
Vysvětlivky k tlačítkům	Střední	Ano
Chyby ve formuláři	Střední	Ne
Termín odevzdání kontrolního bodu	Střední	Ano
Úpravy kontrolního bodu	Nízká	Ano
Navigace do projektu a kontrolních bodů	Nízká	Ano
Rychlý tip u editoru nelze skrýt	Nízká	Ano
Zobrazení času odevzdání u konceptu	Nízká	Ano
Viditelnost poznámky pro studenty	Nízká	Ano

Tabulka 6.1 Nalezené UX nedostatky a jejich stav

Kapitola 7

Zhodnocení a další rozvoj

V této kapitole se věnuji zhodnocení celkového stavu systému, který jsme vyvinuli v rámci bakalářských prací s týmovými kolegy. Dále se věnuji nedokončeným částem a také možnostem dalšího rozvoje, který bude třeba na projektu realizovat.

7.1 Zhodnocení stavu systému

I přes snahu dotáhnout projekt tak, aby funkcemi odpovídal stávajícímu řešení, bohužel nebyly naplněny všechny funkční požadavky a systém bude potřeba dále rozvíjet. V době psaní této práce je potřeba dodělat předměty, import a export dat, přihlašování přes OAuth 2.0 Kromě potřeby dalšího rozvoje bude také potřeba celý systém otestovat uživatelskými testy a doplnit automatické testy.

Systém jako takový je ale dobře připraven na další rozvoj a to hlavně díky dokumentaci kódu a API endpointů a fungujícímu GitLab CI, ve kterém probíhá kontrola kvality kódu a také automatické nasazení. Samotná dokumentace je dostupná formou webu hostovaného službou FIT Pages[1].

Co se týká sekce hodnocení a odevzdání tak ta byla dopracována do konce, se všemi funkčními požadavky, které byly v rámci iterací stanoveny. Funkčně je tato sekce stejná, jako ta v aktuálním systému SOS. Mnou vypracovanou část jsem rovněž podrobil uživatelskému testování, ze kterého vyplynulo pár nedostatků, z nichž většina byla v rámci této práce vyřešena. Rovněž je tato sekce dobře pokryta automatickými testy, takže při dalších zásazích do této části je větší jistota, že funkčnost nebude narušena.

[1] Dokumentace je na dostupná na adrese `https://sos.pages.fit/new-sos/backend/index.html`

7.1.1 Nedokončené části projektu

V této části textu zmiňuji funkcionality, které bude potřeba během dalšího vývoje implementovat. Tento seznam je platný v době psaní tohoto textu. Od té doby se stav některých funkcionalit mohl změnit. Nedokončené funkcionality se přímo netýkají mojí práce, většina z nich souvisí s prací kolegy Jamnického.

7.1.1.1 Kategorizace projektů a uživatelů

Náročnost: vysoká

Důležitost: vysoká

Jedná se o funkcionalitu, díky které je možné rozlišit běhy jednotlivých předmětů podle času, nebo jiných kritérií. Kategorizace umožní následné filtrování v uživatelském rozhraní a omezení přístupu. V době psaní textu je tato část nedokončená a je nutné dokončit vytváření těchto kategorií, jejich úpravy a výsledné řešení otestovat.

7.1.1.2 Filtrování projektů

Náročnost: vysoká

Důležitost: vysoká

V rámci filtrování projektů bude potřeba přidat funkcionality, které budou filtrovat projekty podle předem zmíněných kategorií. S touto funkcionalitou souvisí také tzv. *filter bundle*, kterému se více věnuji v sekci 5.1.3. Po dokončení bude nasazen do nového systému SOS a využíván pro filtrování dat zobrazených v uživatelském rozhraní.

7.1.1.3 Jednotné přihlášení ČVUT

Náročnost: Vysoká

Důležitost: Vysoká

Pro potřeby využití na FIT ČVUT je nutné přidat možnost přihlášení přes ČVUT účet. Tato možnost existuje v aktuálním řešení a jedná se o jedinou možnost přihlášení v produkčním prostředí. V aktuálním řešení je možnost tohoto přihlášení umožněna využitím fakultního OAuth 2.0 serveru Zuul OAAS[2].

V novém řešení je zatím implementované pouze přihlašování pomocí uživatelského jména a hesla, uložených v systému. Tento způsob je vhodný pro testovací účely, nicméně jednotné přihlášení je pro provoz na FIT ČVUT vhodnější.

[2] `https://help.fit.cvut.cz/dev/oauth2.html`

V době psaní této práce se vývojem jednotného přihlášení přes fakultní OAuth server zabývá tým BI-SP2 2024. Tato funkcionalita je vyvíjena v rámci MR 81[3] v backendové části projektu. Ve frontendové části projektu je tato část dostupná pod MR 68[4].

V souvislosti s touto funkcionalitou je však dobré podotknout, že se možná nebude využívat příliš dlouho a to kvůli plánované migraci na Microsoft přihlášení.

7.1.1.4 Jednotné přihlášení Google

Náročnost: Vysoká

Důležitost: Střední

Pro potřeby využití na GJGJ a jiných institucích využívajících Google Classroom, je potřeba přidat také jednotné přihlášení přes Google. Stejně jako pro ČVUT, je tato možnost v aktuálním systému implementována. V případě nového systému SOS na této možnosti aktuálně pracuje tým BI-SP2 2024 v rámci MR 77[5]

7.1.1.5 Studentské testování

Náročnost: Střední

Důležitost: Střední

Studentské testování je funkcionalita využívaná hlavně v předmětu NI-NUR. Jedná se o funkci, která umožňuje vyučujícímu 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. Následně studenti mohou v aplikaci potvrzovat, že jejich vypracovaná řešení byla otestována jinými studenty. V době psaní tohoto textu není tato funkcionalita implementována, ani rozpracována.

7.1.1.6 Export hodnocení do FIT Klasifikace

Náročnost: Střední

Důležitost: Střední

[3] Merge request je dostupný na `https://gitlab.fit.cvut.cz/sos/new-sos/backend/-/merge_requests/81`

[4] `https://gitlab.fit.cvut.cz/sos/new-sos/frontend/-/merge_requests/68`

[5] Merge request je dostupný na `https://gitlab.fit.cvut.cz/sos/new-sos/backend/-/merge_requests/77`

Export hodnocení do FIT Klasifikace umožní vyučujícímu jednoduše exportovat hodnocení za projekt členům týmu. Tento export bude probíhat přes API klasifikace.

7.2 Problémy, které zapříčinili nestihnutí projektu

7.2.1 Styl vývoje

Jak bylo řečeno v sekci 3.3, systém jsme vyvíjeli pomocí iterativního vývoje s týdenními stand-upy. Tento styl vývoje byl pro nás nový, setkali jsme se s ním poprvé během předmětu BI-SP2.

V průběhu BI-SP2 jsme měli v týmu ještě Timoteje Adamce, který působil jako náš projektový manažer a řídil tak tento proces. Obnášelo to i společné plánování práce a to pro všechny členy týmu. Jeho účast na projektu však skončila právě po BI-SP2.

Absence projektového manažera, znamenala značné komplikace z hlediska plánování a průběhu projektu. Každý z členů se začal soustředit více na vlastní práci a její plánování. To však vedlo k poměrně chaotickému způsobu vývoje, který nebyl bez projektového manažera udržitelný.

7.2.2 Objem práce

Dalším důvodem nestihnutí zprovoznění systému byl objem práce. Aktuální systém je aktivně vyvíjen již 3 roky a podílela se na něm celá řada studentů. Pokud by se jednalo pouze o vývojovou část, tak by projekt nejspíše byl zvládnutelný. Náš tým se kromě implementace věnoval mnoha návrhům, analýzám. Kromě toho jsme se trávili značné množství času na týmových schůzkách, vzájemných code review a konzultacích pro členy SP týmu.

7.2.3 Technologie

Ačkoliv jsou zvolené technologie pro projekt objektivně vhodné, náš tým s nimi neměl příliš velké zkušenosti. Pro všechny členy projektu se jednalo o první setkání s frameworkem Vue.js a ASP.NET Core. Tento fakt poměrně zkomplikoval vývoj a to hned z několika důvodů.

Prvním důvodem bylo to, že jsme museli věnovat značné množství času naučení se novým technologiím. Výraznější byl tento problém na frontendu, jelikož všichni členové týmu působí spíše jako backendoví vývojáři nemají mnoho zkušeností s vývojem frontendu. I přes poměrně pozvolnou učící křivku u frameworku Vue.js 3 a Nuxt 3 zabralo studování těchto technologií několik desítek hodin. Na backendu nebyl tento problém tak výrazný, neboť se všichni členové týmu setkali s podobnými MVC frameworky a jednalo se tak spíše o záležitosti specifické pro tuto technologii.

Druhý důvod plyne z předchozího a také ze způsobu vývoje, kterým jsme se rozhodli projekt vyvíjet. Při iterativním vývoji se stále přidávají nové funkcionality, které se často nabalují na ty již implementované. To vede na více částí, které je třeba průběžně refaktorovat, aby systém zůstal v udržitelném stavu.

Kvůli průběžným refaktorům a učení se s novými technologiemi se často vývoj protahoval a více časů se strávilo také na vzájemných code review a jejich opravách.

7.2.4 Práce v týmu

Jak je zřejmé i ze zadání této práce, tak projekt jako celek byl rozdělen do několika částí. Ačkoliv se na první pohled může zdát, že tyto části spolu příliš nesouvisí, tak opak byl pravdou. Často jsme na konci iterace naráželi na problém s tzv. *merge konflikty*, které způsobovali další prodlení vývoje. Tyto konflikty značí části kódu, které byly změněny současně ve více větvích a Git je nedokáže automaticky sloučit. Sloučení změn proto musí probíhat manuálně a po jeho sloučení musí být provedena řádná kontrola, že byly zachovány obě vyvíjené funkcionality. Řešení těchto konfliktů, které vznikali prakticky při každé iteraci rovněž prodlužovalo vývoj a zatěžovalo vývojáře.

7.3 Návrhy na další rozvoj projektu

V následující části se věnuji možným vylepšením a novým funkcionalitám, které by se ohledně odevzdání a hodnocení daly implementovat. Zahrnuji zde také nedostatky, které vyplynuly z uživatelského testování, ale nebyly vyřešeny.

7.3.1 Nedostatky z uživatelského testování

Většinu nedostatků z uživatelského testování se podařilo vyřešit ještě v rámci práce. Zbývají následující tři nedostatky, které by bylo vhodné před nasazením systému do provozu vyřešit. Tyto nedostatky jsou zaznamenány rovněž formou GitLab issues v projektových repozitářích.

UX01 – Historie přerozdělení bodů

Popis: upravení přerozdělení vyučujícím přepíše studentský návrh nevratně

Issues: Backend#12, Frontend#14

UX02 – Chyby ve formulářích

Popis: při chybě ve formuláři je prvek nevýrazný a není na něj zascrollováno

Issues: Frontend#15

UX03 – Tlačítko pro vrácení odevzdání / hodnocení

Popis: tlačítka na změnu stavu zpět do konceptu jsou vidět až ve formuláři

Issues: Frontend#13

7.3.2 Další návrhy

Tyto návrhy nebyly zpracovány ve funkčních požadavcích, ale pro další rozvoj je zde uvádím, neboť souvisí s touto prací. Jedná se především o přehledy, nebo vylepšení UX.

7.3.2.1 Přehled odevzdání čekajících na hodnocení

Náročnost: Vysoká

Důležitost: Vysoká

Pro potřeby rychlé navigace a přehledu práce, která je potřeba udělat, by bylo vhodné doplnit seznam odevzdání, které čekají na ohodnocení. Tento seznam by mohl být umístěn na domovské obrazovce, nebo na stránce daného předmětu. Při kliknutí na odevzdání z tohoto seznamu by se uživateli rovnou zobrazil patřičný formulář pro hodnocení.

7.3.2.2 Konfigurace pozdního odevzdání

Náročnost: Střední

Důležitost: Vysoká

V současné době systém podporuje pozdní odevzdání a umožní týmu odevzdat i po termínu. Bylo by však žádoucí mít možnost pozdní odevzdání rovnou odmítnout. Tato možnost by byla nastavitelná buď pro celý projekt, nebo pro jednotlivé kontrolní body.

7.3.2.3 Zobrazení hodnocení v přehledu projektů

Náročnost: Střední

Důležitost: Vysoká

Z uživatelských požadavků vyplynulo, že by vyučující chtěli mít rychlý přehled formou jednoduchých barevných bodů u projektu. Tyto body by značily stavy jednotlivých kontrolních bodů. Sloužily by defacto jako náhrada současných grafů, které dle slov vyučujících špatně reprezentují skutečnost.

7.3.2.4 Rozbalení kontrolního bodu

Náročnost: Střední

Důležitost: Střední

Při přechodu na stránku projektu by bylo dobré rozbalit kontrolní bod, který bude pravděpodobně uživatele zajímat. Může to být ten, kterému se blíží termín odevzdání, ale není zatím odevzdán. Je však nutné provést hlubší analýzu uživatelských očekávání.

Kapitola 8
Závěr

Cílem této bakalářské práce bylo za pomoci iterativního vývoje analyzovat problémy, které se týkají sekcí odevzdávání a hodnocení v současném řešení systému SOS. Na tyto nedostatky mělo být navrženo adekvátní řešení, které jsem měl za úkol implementovat a otestovat. Toto implementované a otestované řešení mělo být nasazeno na testovací server nového portálu SOS.

Nedostatky systému byly analyzovány již s týmovými kolegy v průběhu předmětu BI-SP1 a BI-SP2. V této fázi jsme s týmem získali dobrou představu o nedostatcích a také o tom, jakým způsobem bude stavěno nové řešení. Potřeby uživatelů a také nedostatky jsme sbírali ze závěrečných prací, které se zabývaly systémem SOS. Konkrétně bakalářské a diplomové práce Ing. Tomáše Pavlůska a bakalářské práce Bc. Maxe Hejdy. Kromě těchto závěrečných prací bylo hodně nedostatků a požadavků získáno také rozhovory s uživateli systému a vedoucím práce. Na základě těchto poznatků bylo rozhodnuto opustit vícestránkovou architekturu stávající aplikace ve prospěch modernějšího řešení v podobě aplikace rozdělené na frontend a backend.

Dále bylo také rozhodnuto o použití nových technologií a to zejména na frontendu, kde jsme se pro nové řešení rozhodli využít populární JavaScriptový framework Vue.js doplněný o Nuxt 3. Díky tomu mohlo vzniknout dobře rozšiřitelné a přizpůsobitelné uživatelské rozhraní, které může být při dalším rozvoji dále upravováno a vylepšováno, aby lépe pokrylo uživatelské požadavky.

Na backendu bylo rozhodnuto o použití webového frameworku ASP.NET Core 8, který byl použit na postavení REST API. Toto REST API slouží jako zdroj dat pro výše zmíněný frontend. Díky tomu je možné nezávisle rozvíjet obě části projektu a do budoucna toto řešení poskytuje větší flexibilitu pro další rozvoj, či výměnu technologií na frontendové části projektu.

Pro vývoj systému jsme s kolegy využili principů iterativního vývoje, s cílem si tento proces řádně vyzkoušet a zhodnotit si jeho pozitiva a negativa. Vývoj byl rozdělen do několika 14denních iterací a také interních týdenních sprintů, na konci kterých jsme s týmem probrali, čím jsme se zabývali, zkonzultovali a zhodnotili implementované změny a poté velmi rámcově naplánovali

práci na další iteraci. Z mojí zkušenosti byl tento proces pro rozvoj vhodným řešením, avšak bylo by potřeba dedikovat práci project managementu na dalšího člena týmu, jelikož plánování úkolů a sprintů se ukázalo jako velmi obtížné.

V rámci iterací byla vždy provedena analýza implementované části systému a také návrhy potřebné pro její realizaci. Při dílčích analýzách bylo vycházeno z výše zmíněných závěrečných prací a také z vlastních poznatků sesbíraných pomocí rozhovorů od reálných uživatelů systému a také vedoucího Ing. Jiřího Hunky.

Po dokončení analýzy a návrhu pro iteraci byla provedena implementace, která následně prošla procesem code review. Výsledkem každé iterace byla funkční verze aplikace s přidanými funkcionalitami. Tyto verze byly také pomocí GitLab CI testovány na kvalitu kódu a posléze nasazovány na testovací server dostupný pro projekt.

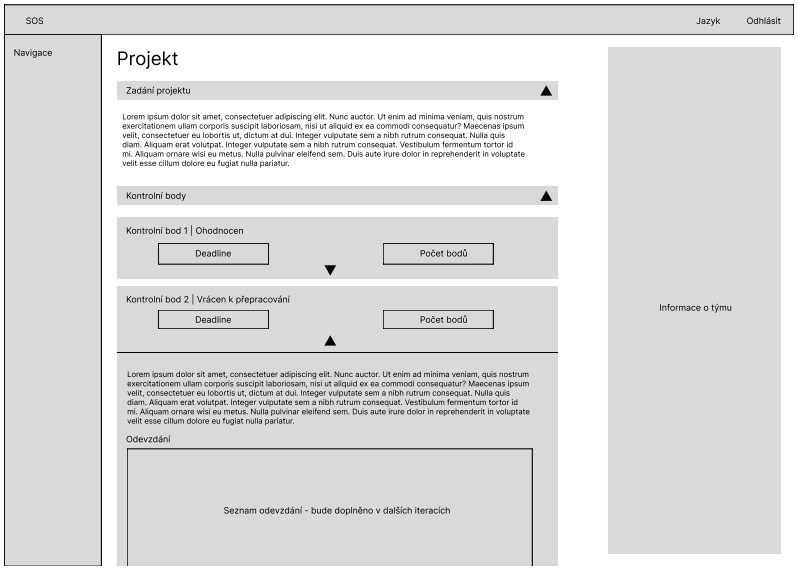
Při vývoji byl kladen důraz na rozšiřitelnost řešení a proto byla také vedena dokumentace relevantních částí kódu a hlavně celého API, které je dostupné i díky snaze studentů BI-SP1 2024 formou webového rozhraní. Toto webové rozhraní obsahuje definice všech API endpointů. Tyto definice obsahují formát dat přijímaný endpointem, požadovaná oprávnění a formát odpovědi včetně stavového HTTP kódu.

Na závěr bylo provedeno také uživatelské testování se studenty a vyučujícími. Na testování navázala oprava některých nedostatků, které z něj vyplynuly. Nevyřešené nedostatky byly zaznamenány do GitLabu a předány k vyřešení SP týmu. Poslední verzí před odevzdáním této práce je verze **v0.11.0** v obou projektových repozitářích.

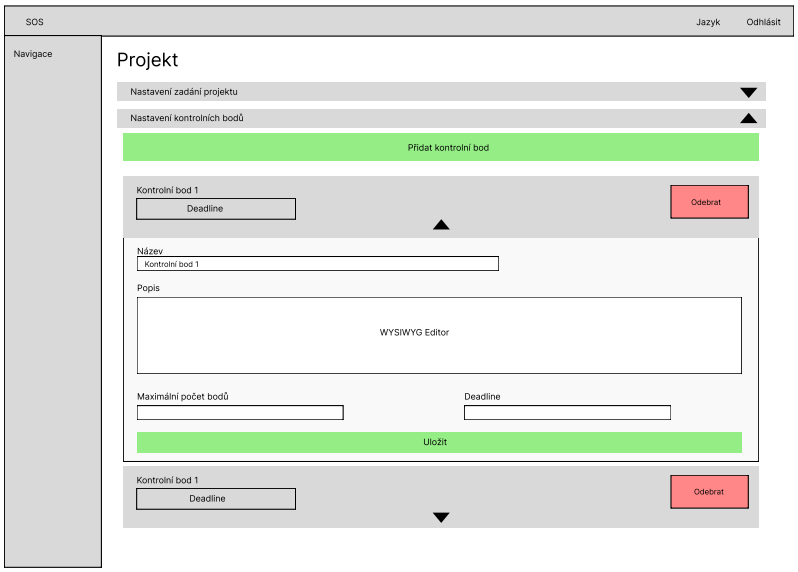
Příloha A

Wireframy

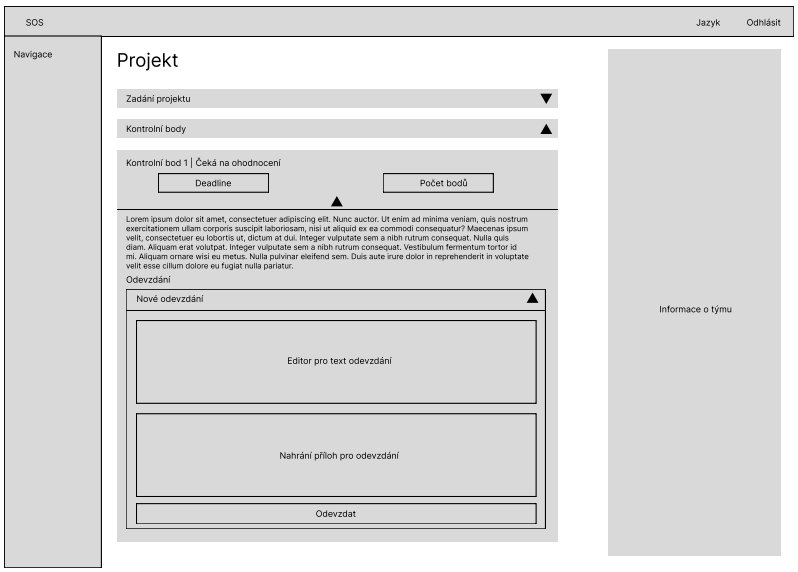
V této příloze se nachází veškeré wireframy sestavené v nástroji Figma pro potřeby jednotlivých iterací. Wireframy jsou rovněž dostupné ve Figmě projektu, kam mají přístup aktuální členové SP týmu SOS. Na požádání je možné udělit přístup i dalším osobám.



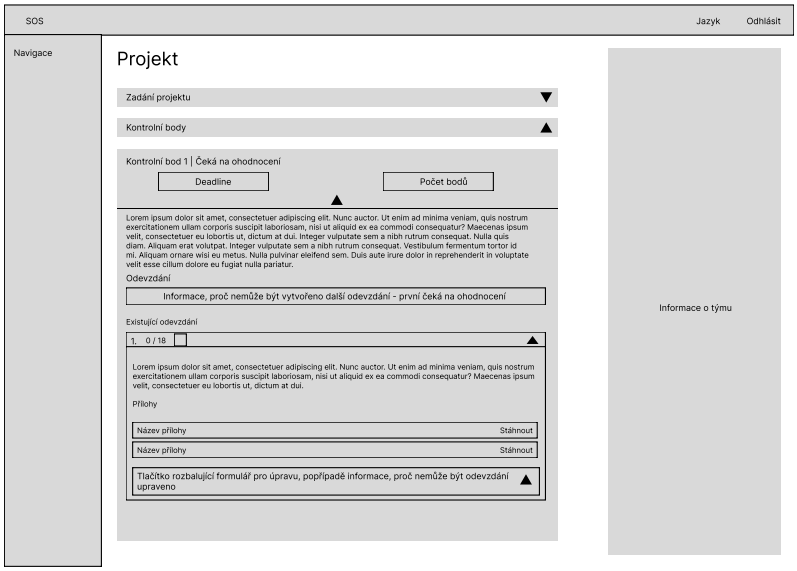
Obrázek A.1 Wireframe zobrazení kontrolních bodů



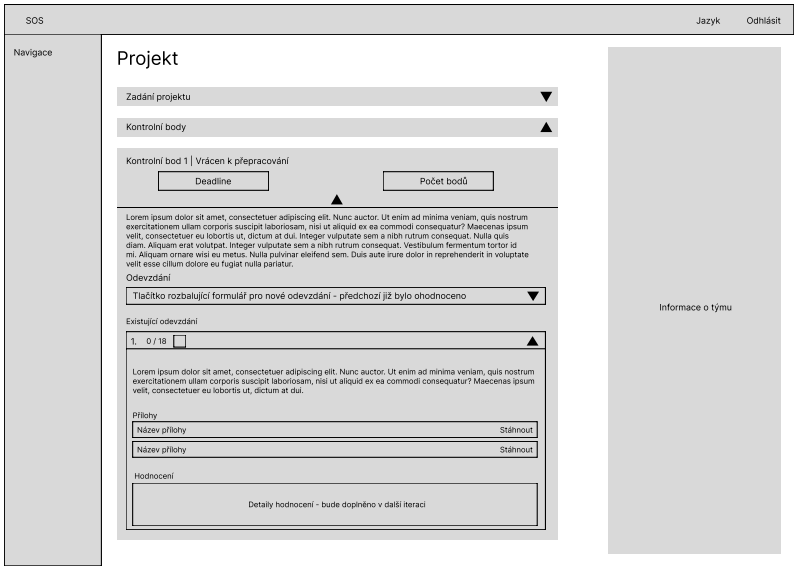
Obrázek A.2 Wireframe nastavení kontrolních bodů



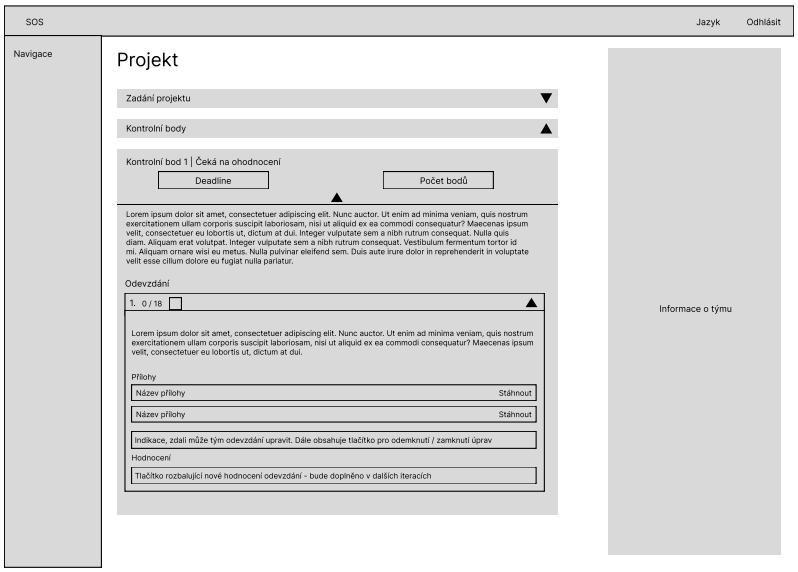
Obrázek A.3 Wireframe vytvoření odevzdání – pohled studenta



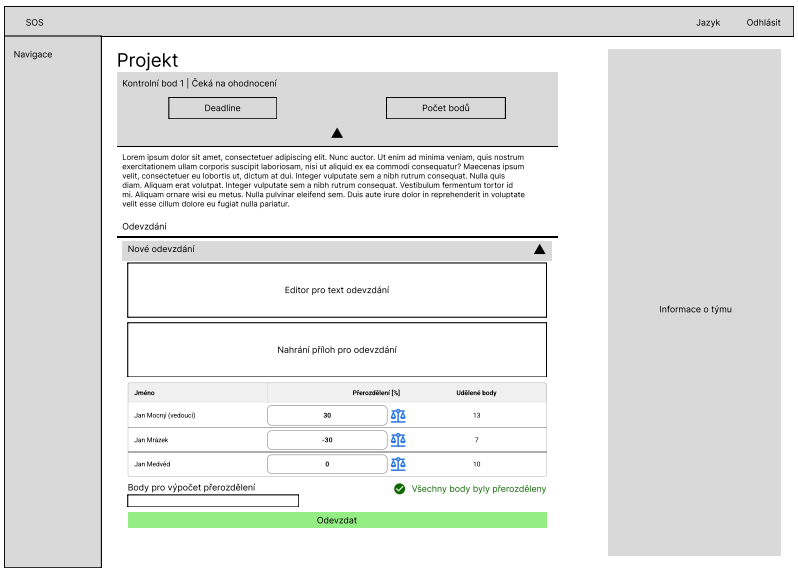
Obrázek A.4 Wireframe seznamu odevzdání – pohled studenta



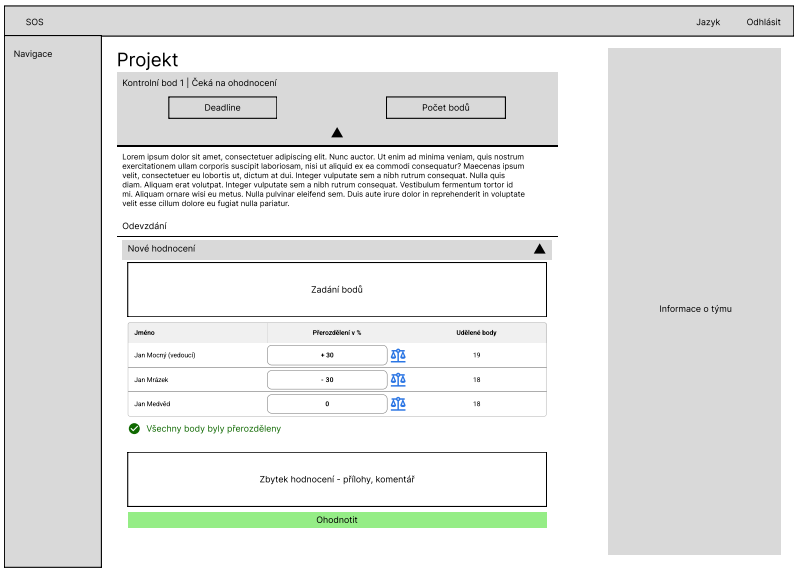
Obrázek A.5 Wireframe pro nové odevzdání, pokud bylo předchozí ohodnoceno, nebo vráceno – pohled studenta



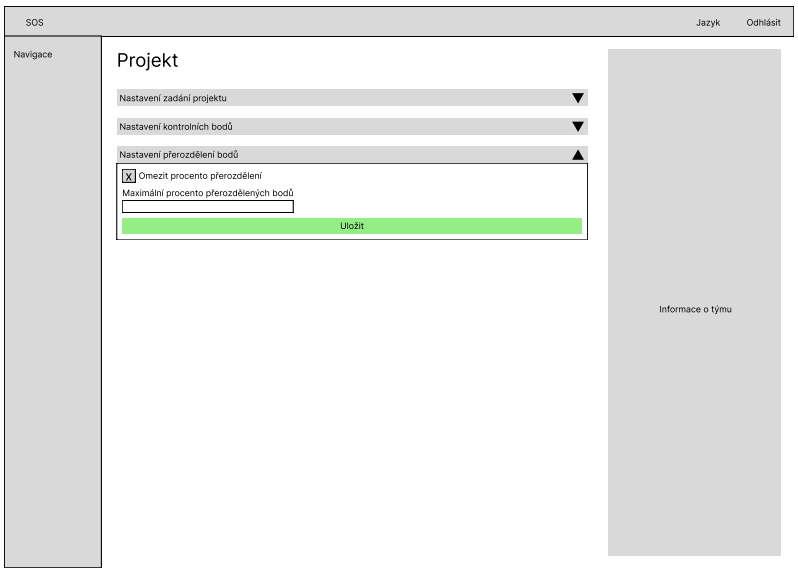
Obrázek A.6 Wireframe detailu odevzdání včetně zámku – pohled vyučujícího



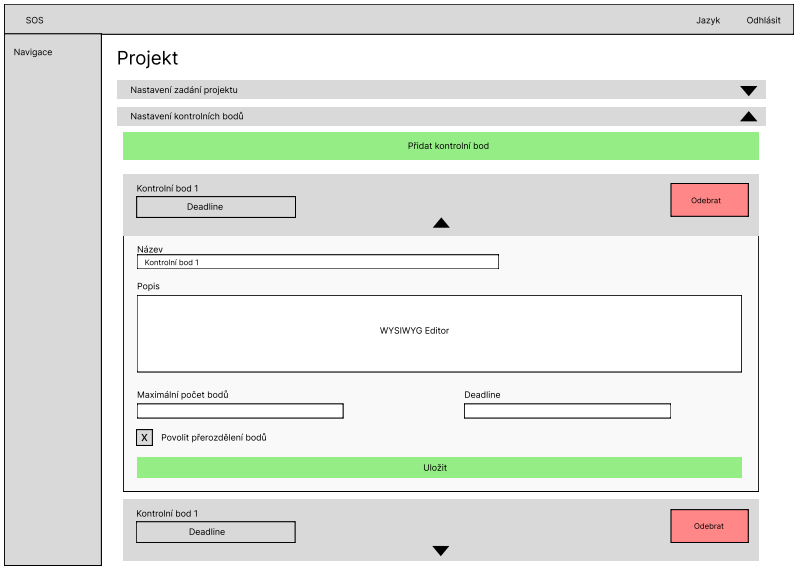
Obrázek A.7 Wireframe odevzdání s přerozdělením – pohled studenta



Obrázek A.8 Wireframe hodnocení s přerozdělením – pohled vyučujícího



Obrázek A.9 Wireframe nastavení omezení přerozdělení na projektu

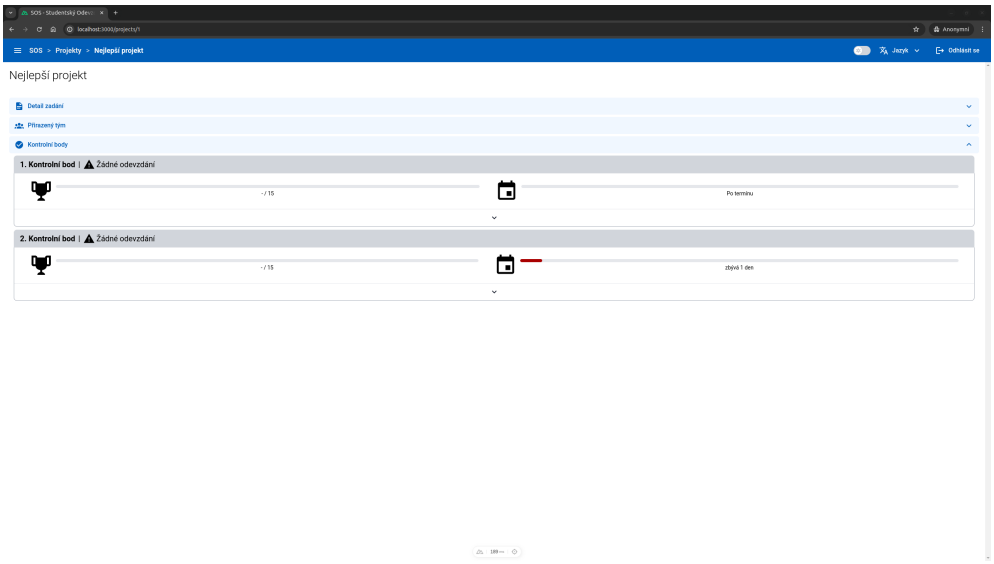


Obrázek A.10 Wireframe povolení přerozdělení na kontrolním bodu

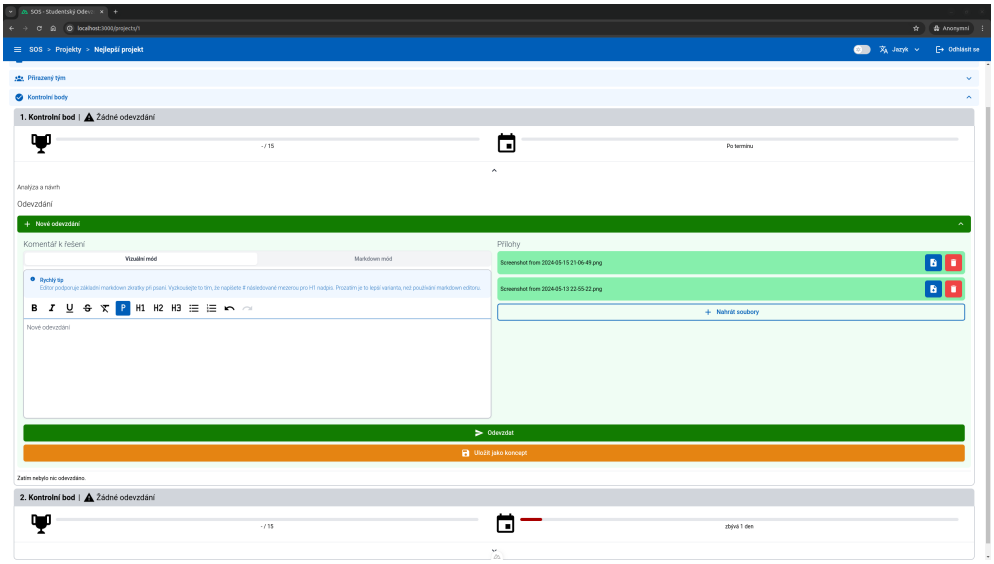
Příloha B

Snímky obrazovky realizované části

V této části se nachází snímky obrazovky z mnou vypracované části systému. Snímky obrazovky pochází z verze před uživatelským testováním, takže se v ní nacházejí nedostatky, které uživatelské testování odhalilo.

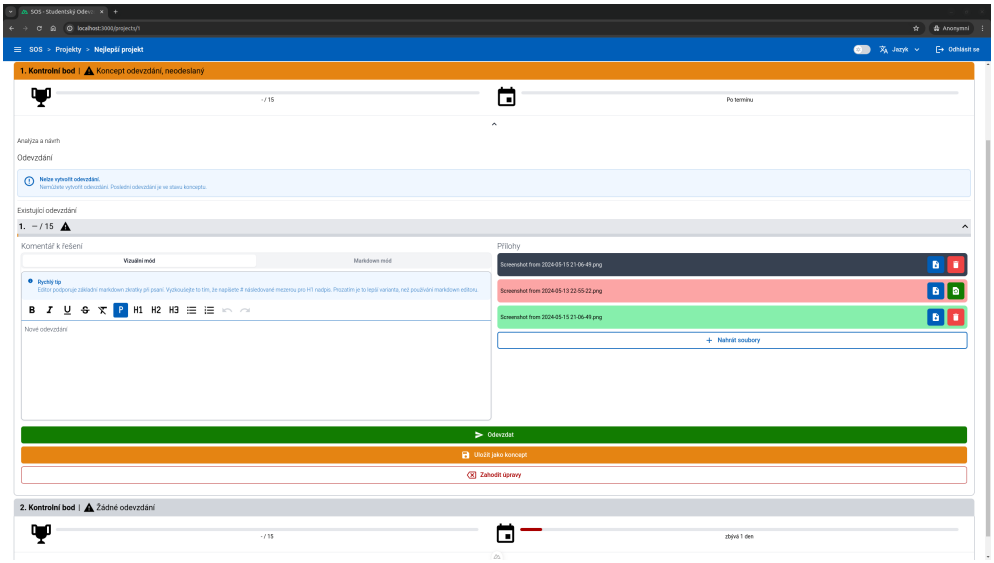


Obrázek B.1 Pohled na kontrolní body – student

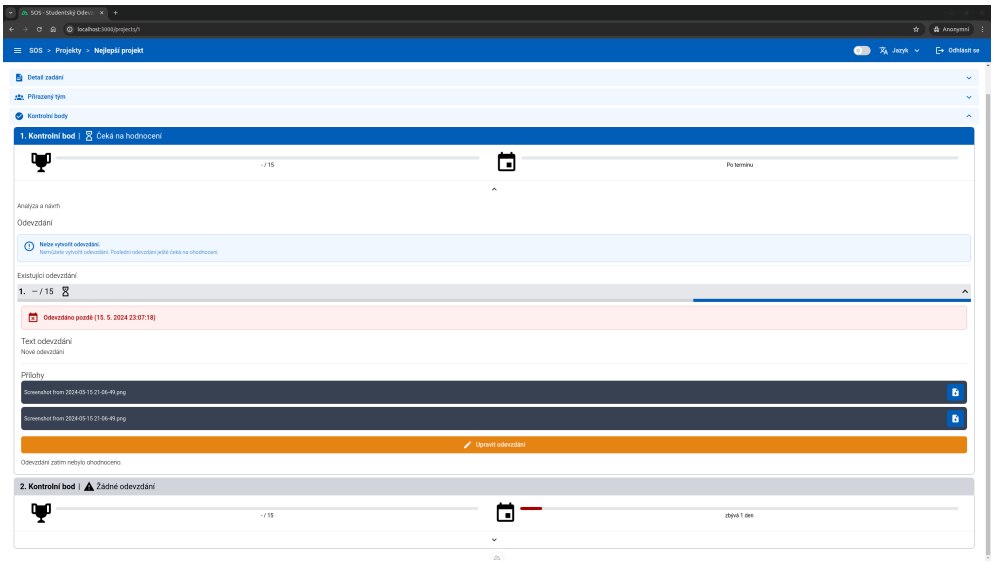


Obrázek B.2 Vytvoření odevzdání – student

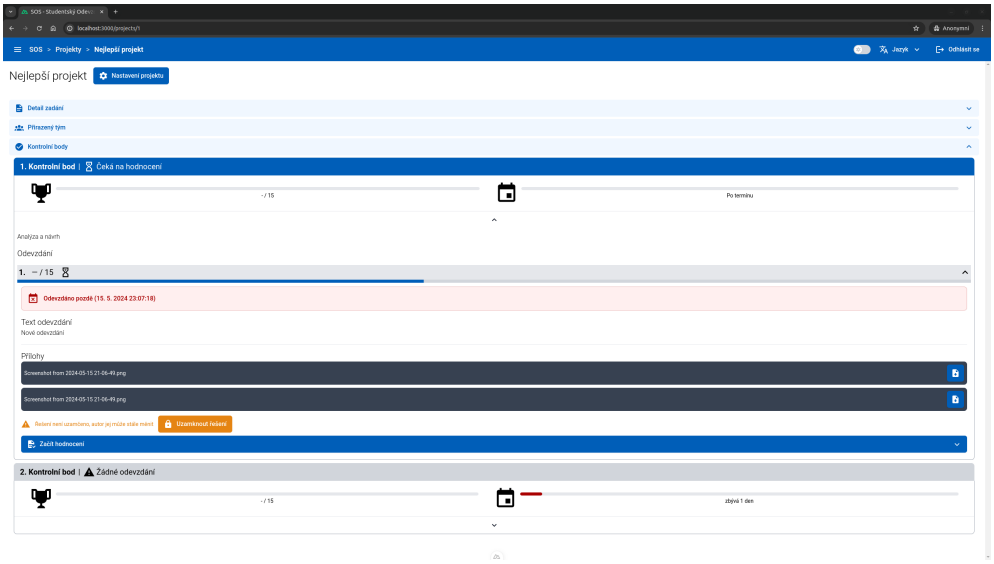
Obrázek B.3 Koncept odevzdání



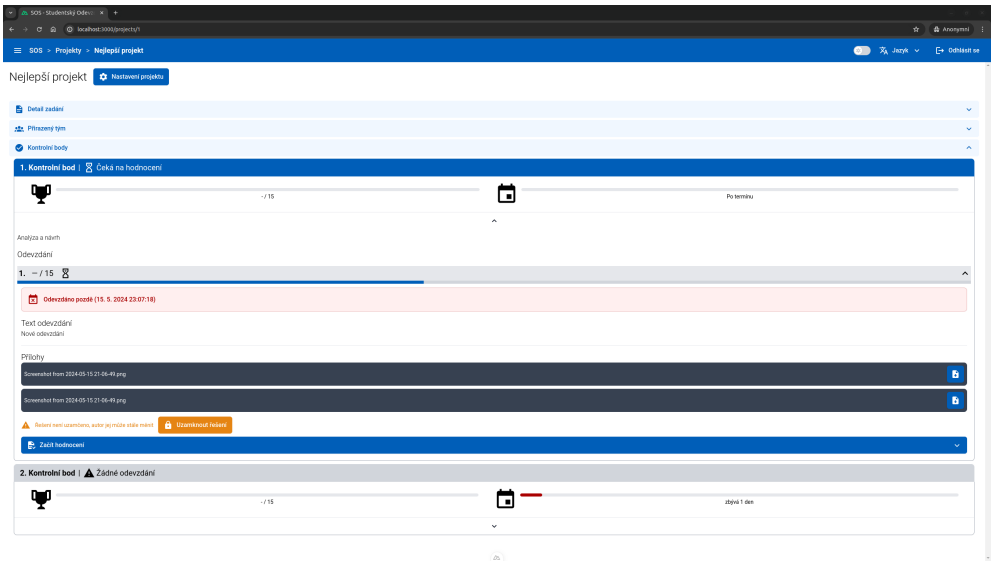
Obrázek B.4 Úprava konceptu a souborů



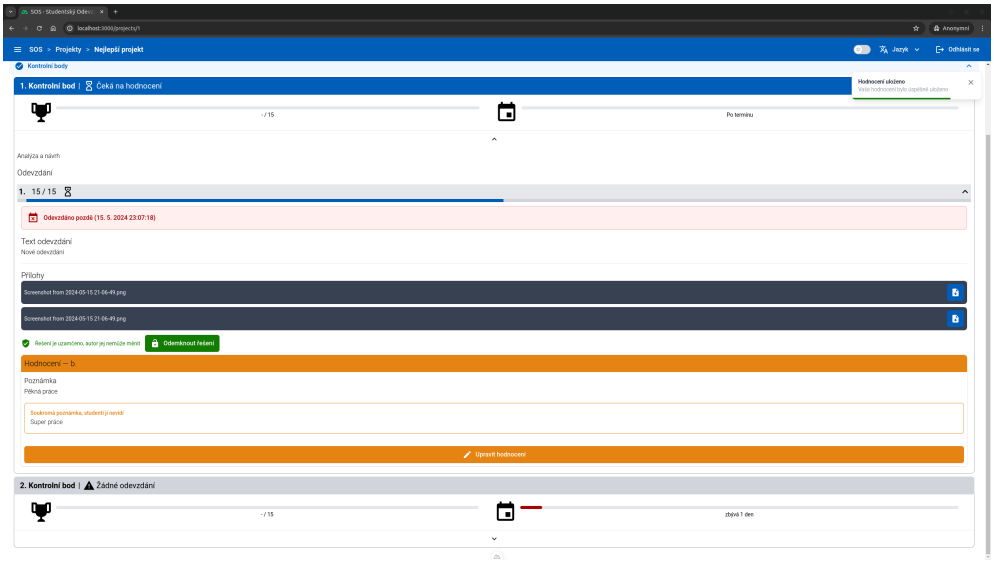
Obrázek B.5 Odevzdané řešení



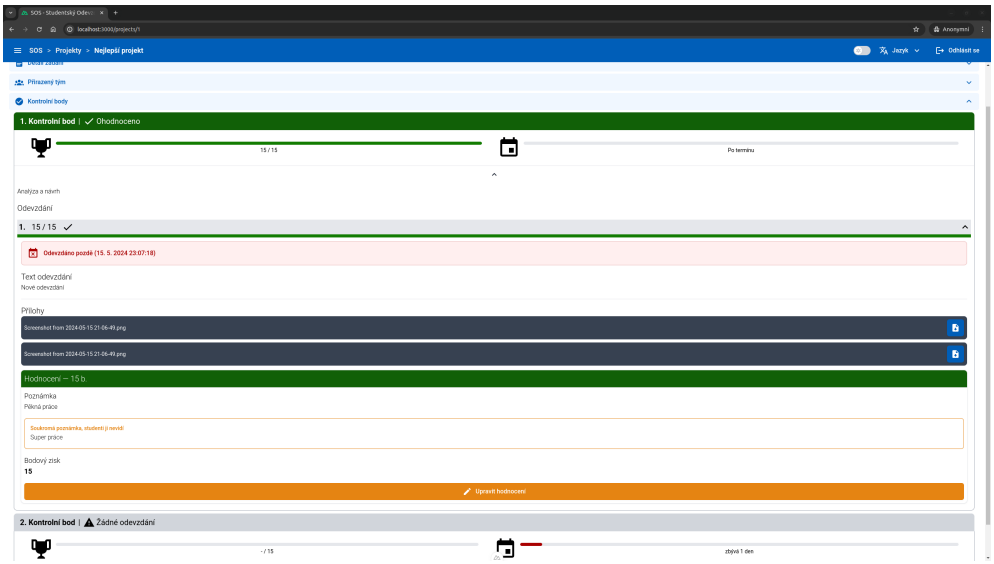
Obrázek B.6 Odevzdání – pohled vyučujícího



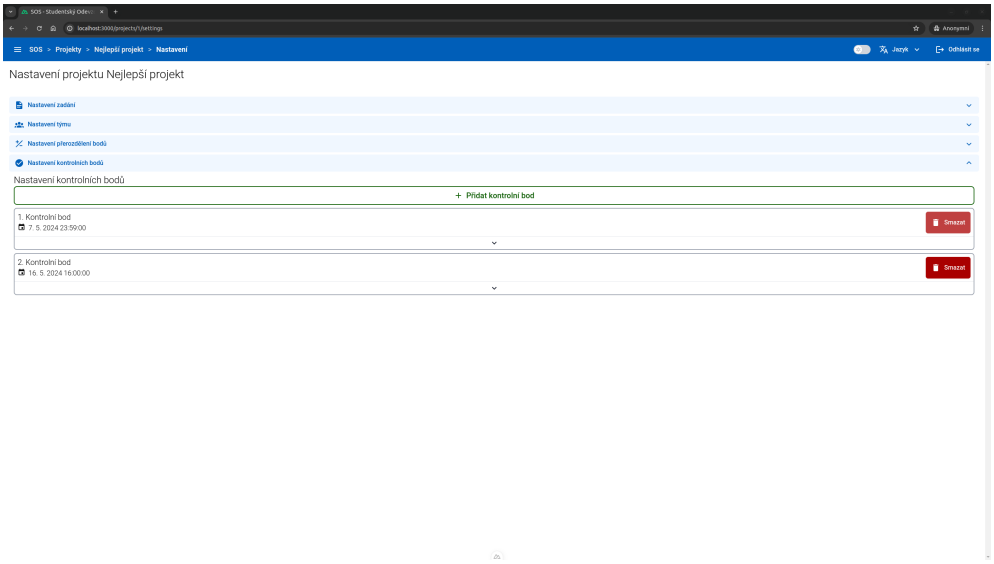
Obrázek B.7 Formulář pro hodnocení



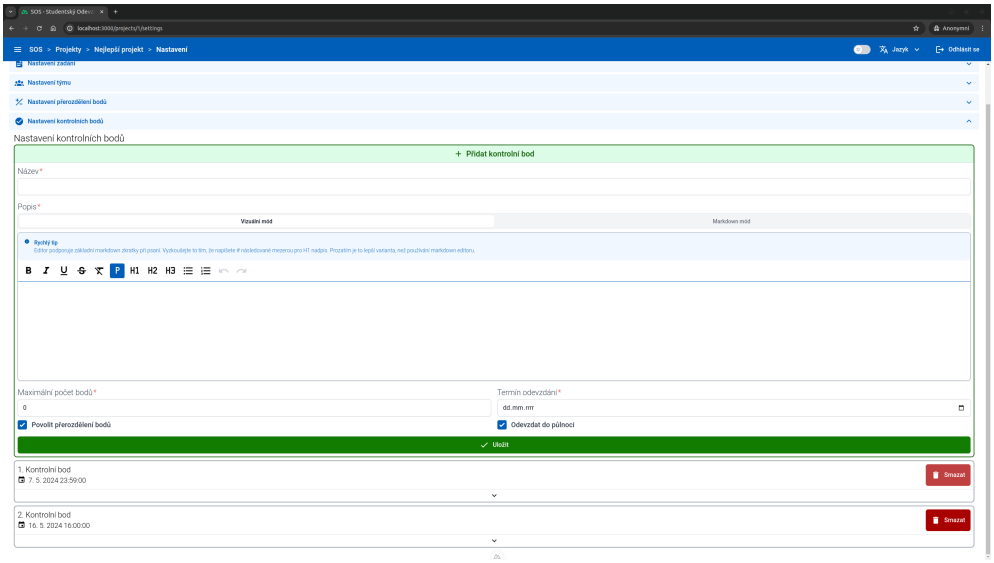
Obrázek B.8 Koncept hodnocení



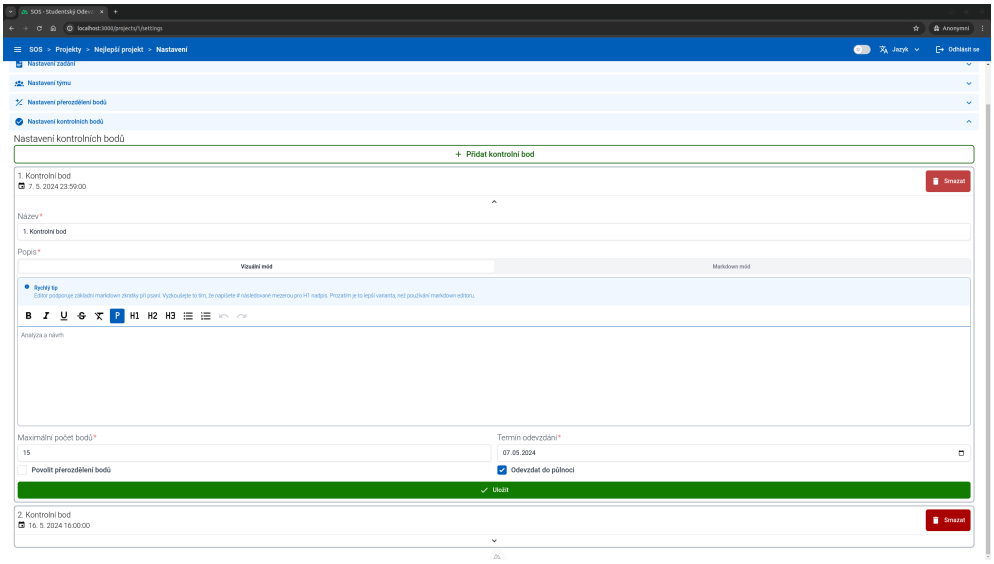
Obrázek B.9 Ohodnocené odevzdání



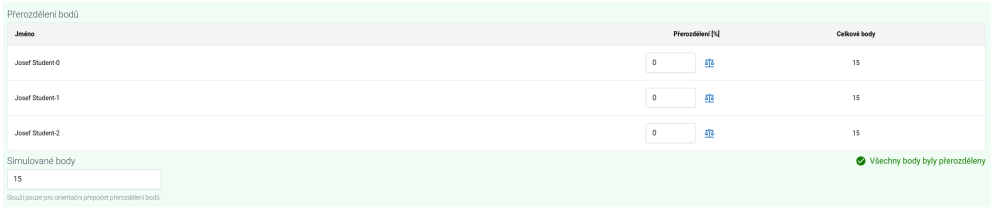
Obrázek B.10 Nastavení projektu



Obrázek B.11 Vytvoření kontrolního bodu



Obrázek B.12 Upravení kontrolního bodu

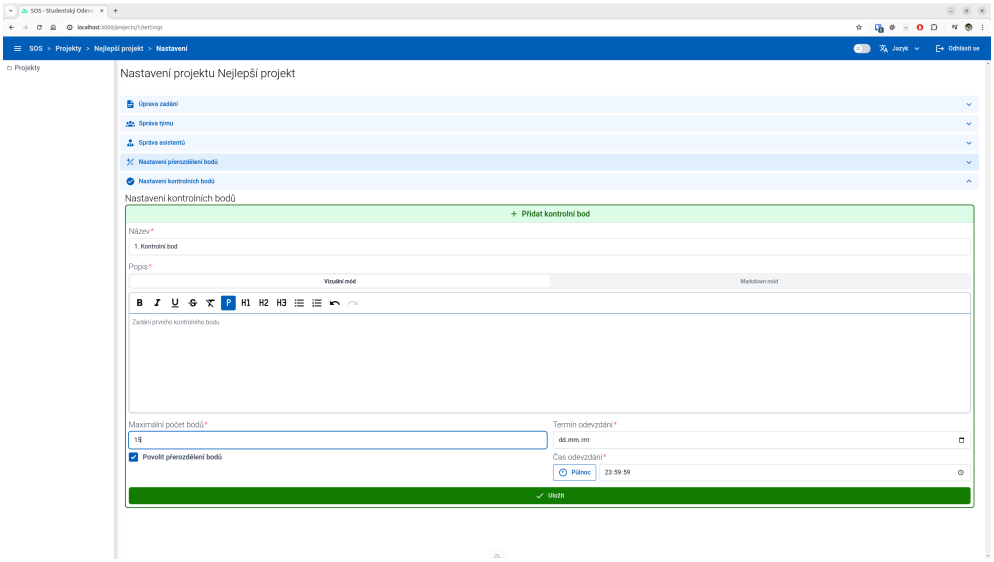


Obrázek B.13 Komponenta přerozdělení bodů – student

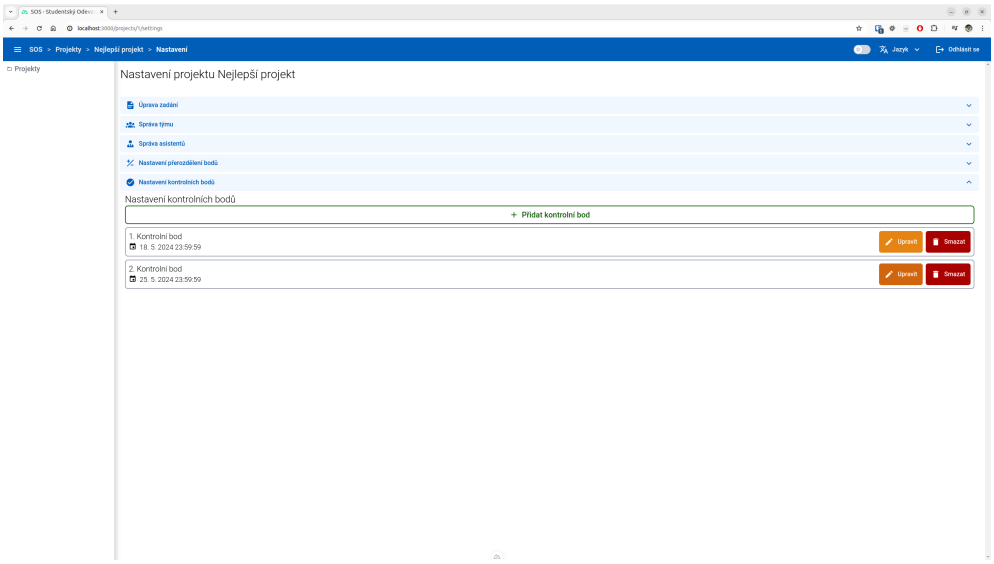
Příloha C

Snímky obrazovky po uživatelském testování

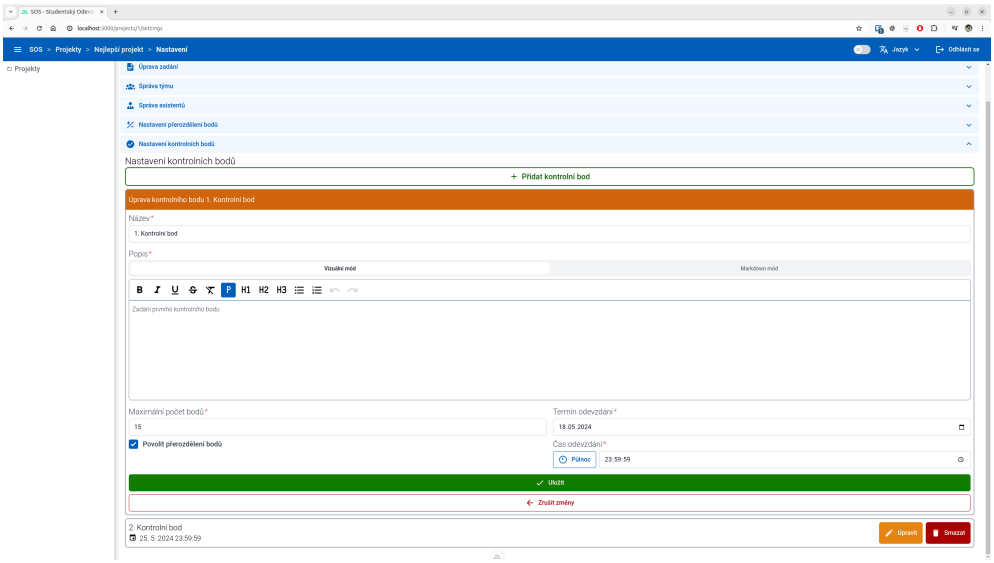
Následující snímky obrazovky ukazují některé změny, které byly realizovány po uživatelském testování.



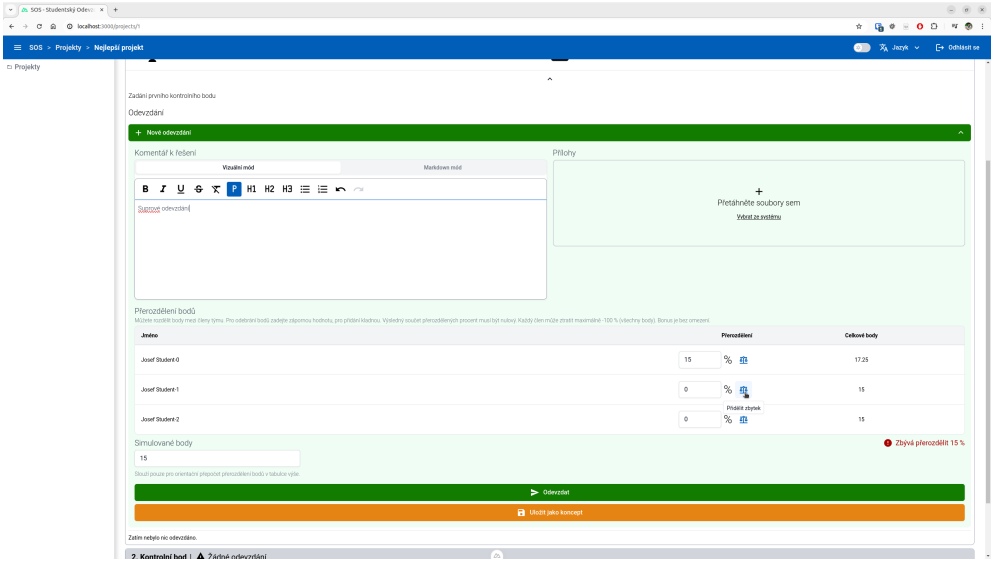
Obrázek C.1 Vytvoření kontrolního bodu – oprava zadání času



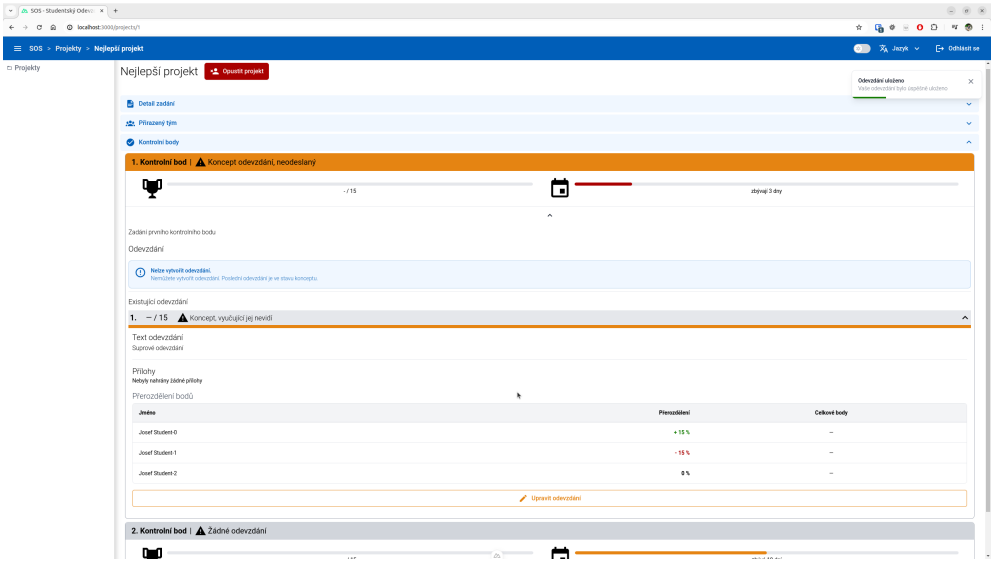
Obrázek C.2 Nastavení kontrolních bodů



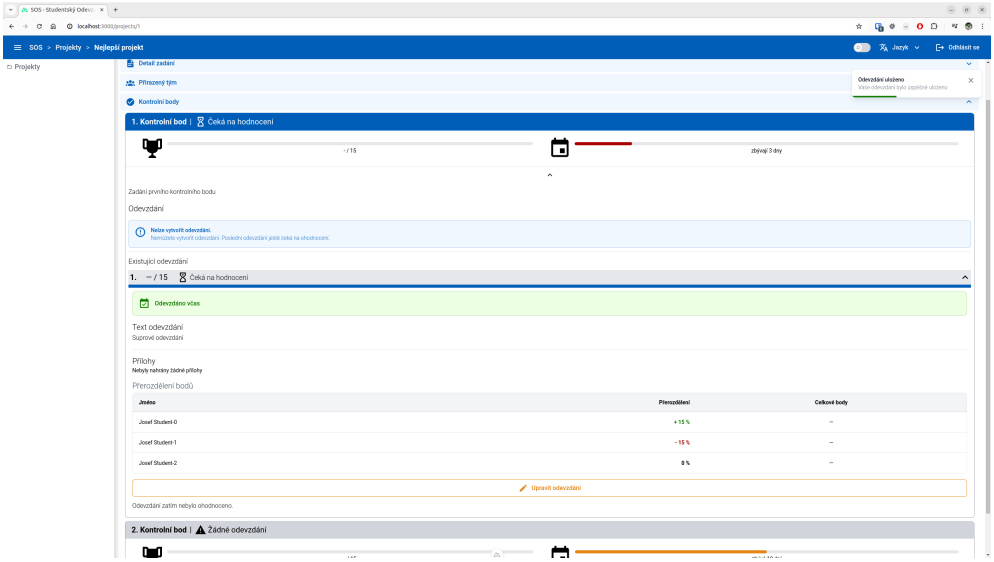
Obrázek C.3 Úprava kontrolního bodu



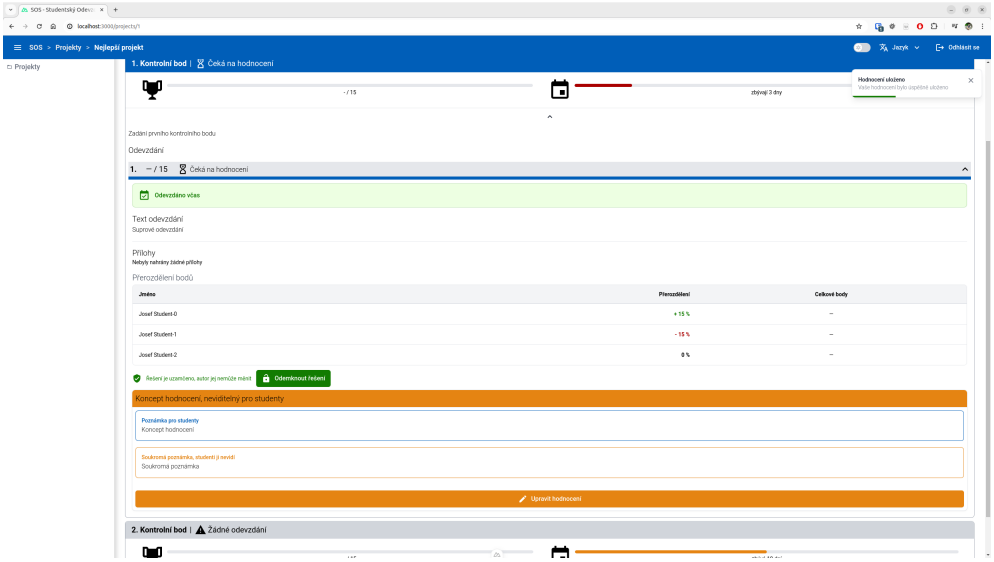
Obrázek C.4 Vytvoření odevzdání s přerozdělením



Obrázek C.5 Koncept odevzdání



Obrázek C.6 Odevzdané řešení



Obrázek C.7 Koncept hodnocení

Bibliografie

1. CZ.NIC. *Jak na Internet - Webové aplikace* [online]. [B.r.]. [cit. 2024-02-26]. Dostupné z: https://www.jaknainternet.cz/page/1262/webove-aplikace/.
2. CLOUDFLARE. *What is HTTP?* [Online]. [B.r.]. [cit. 2024-05-16]. Dostupné z: https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/.
3. MDN. *HTTP request methods* [online]. [B.r.]. [cit. 2024-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods.
4. LUCHANINOV, Yurii. *Web Application Architecture: Choosing the Right Type in 2024* [online]. 2024. [cit. 2024-05-14]. Dostupné z: https://mobidev.biz/blog/web-application-architecture-types.
5. *Navrhování aplikací .NET nativních pro cloud pro Azure | Microsoft Learn* [online]. [B.r.]. [cit. 2024-02-26]. Dostupné z: https://learn.microsoft.com/cs-cz/dotnet/architecture/cloud-native/?WT.mc_id=dotnet-35129-website.
6. DUSHENIN, Oleksii. *Monolithic Architecture. Advantages and Disadvantages | by Oleksii Dushenin | Medium* [online]. 2021. [cit. 2024-05-14]. Dostupné z: https://datamify.medium.com/monolithic-architecture-advantages-and-disadvantages-e71a603eec89.
7. *What are microservices?* [Online]. [B.r.]. [cit. 2024-02-26]. Dostupné z: https://microservices.io/.
8. GOEBELBECKER, Eric. *How does microservice communication work?* [Online]. 2021. [cit. 2024-05-14]. Dostupné z: https://www.architect.io/blog/2023-03-21/microservices-communication/.
9. ATLASSIAN. *Advantages and disadvantages of Microservices* [online]. [B.r.]. [cit. 2024-05-14]. Dostupné z: https://www.atlassian.com/microservices/cloud-computing/advantages-of-microservices.

10. *What is serverless?* [Online]. 2022. [cit. 2024-02-26]. Dostupné z: https://www.redhat.com/en/topics/cloud-native-apps/what-is-serverless.
11. CLOUDFLARE. *Pros and cons of serverless* [online]. [B.r.]. [cit. 2024-05-14]. Dostupné z: https://www.cloudflare.com/learning/serverless/why-use-serverless/.
12. FOWLER, Martin. *Inversion of Control Containers and the Dependency Injection pattern* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.martinfowler.com/articles/injection.html.
13. MESROPYAN, Elen. *Design Patterns* [online]. 2023. [cit. 2024-05-14]. Dostupné z: https://stackify.com/introduction-to-design-patterns-in-software-development/.
14. *GOFPattern (Behavioral, Creational, Structural)* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.gofpattern.com/.
15. *Interactive Application Architecture Patterns · Los Techies* [online]. 2007. [cit. 2024-02-26]. Dostupné z: https://lostechies.com/derekgreer/2007/08/25/interactive-application-architecture/.
16. *Archiv z 25.8.2007 | Interactive Application Architecture Patterns* [online]. 2007. [cit. 2024-02-26]. Dostupné z: https://web.archive.org/web/20201107060523/https://lostechies.com/derekgreer/2007/08/25/interactive-application-architecture/.
17. *Observer Pattern (Flexibly broadcast messages to Receivers)* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.gofpattern.com/behavioral/patterns/observer-pattern.php.
18. *Django documentation | FAQ* [online]. 2024. [cit. 2024-02-26]. Dostupné z: https://docs.djangoproject.com/en/5.0/faq/general/%5C#django-appears-to-be-a-mvc-framework-but-you-call-the-controller-the-view-and-the-view-the-template-how-come-you-don-t-use-the-standard-names.
19. CLOUDFLARE. *Web security* [online]. [B.r.]. [cit. 2024-05-14]. Dostupné z: https://www.cloudflare.com/learning/security/what-is-web-application-security/.
20. OWASP FOUNDATION. *OWASP Top Ten* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://owasp.org/www-project-top-ten/.
21. OWASP FOUNDATION. *Cross Site Scripting (XSS)* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://owasp.org/www-community/attacks/xss/.
22. VUE.JS. *Security* [online]. [B.r.]. [cit. 2024-05-14]. Dostupné z: https://vuejs.org/guide/best-practices/security#html-content.

23. OWASP FOUNDATION. *Cross Site Scripting Prevention* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html%5C#introduction.

24. *British Airways data theft demonstrates need for cross-site scripting restrictions | TechRepublic* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.techrepublic.com/article/british-airways-data-theft-demonstrates-need-for-cross-site-scripting-restrictions/.

25. *XSS Flaw Exposed eBay Users to Phishing Attacks - SecurityWeek* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.securityweek.com/xss-flaw-exposed-ebay-users-phishing-attacks/.

26. OWASP FOUNDATION. *Session hijacking attack* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://owasp.org/www-community/attacks/Session_hijacking_attack.

27. OWASP FOUNDATION. *Session Management* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html.

28. *Why is passing the session id as url parameter insecure?* [Online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://security.stackexchange.com/questions/14093/why-is-passing-the-session-id-as-url-parameter-insecure.

29. ZDNET. *Slack fixes vulnerability exploitable for session hijacking, account takeovers* [online]. 2020. [cit. 2024-02-27]. Dostupné z: https://www.zdnet.com/article/slack-vulnerability-allowed-session-hijacking-account-takeovers/.

30. THREATPOST. *Session Hijacking Bug Exposed GitLab Users Private Tokens* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: 2017.

31. OWASP FOUNDATION. *SQL Injection* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://owasp.org/www-community/attacks/SQL_Injection.

32. OWASP FOUNDATION. *SQL Injection Prevention* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.

33. COMPUTERWORLD. *SQL injection attacks led to Heartland, Hannaford breaches* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.computerworld.com/article/2527185/sql-injection-attacks-led-to-heartland--hannaford-breaches.html.

34. COMPUTERWORLD. *Sony Pictures falls victim to major data breach* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.computerworld.com/article/2508871/sony-pictures-falls-victim-to-major-data-breach.html.

35. OWASP FOUNDATION. *Cross Site Request Forgery (CSRF)* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://owasp.org/www-community/attacks/csrf.

36. OWASP FOUNDATION. *Cross-Site Request Forgery Prevention* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html.

37. *TikTok fixed security issues that could have led one-click account takeover* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://securityaffairs.com/111336/hacking/tiktok-domains-security-flaws.html.

38. *DoS Attack vs. DDoS Attack: Key Differences? | Fortinet* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.fortinet.com/resources/cyberglossary/dos-vs-ddos.

39. *Elastic Load Balancing (BP6) - AWS Best Practices for DDoS Resiliency* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://docs.aws.amazon.com/whitepapers/latest/aws-best-practices-ddos-resiliency/elastic-load-balancing-bp6.html.

40. ZDNET. *Google Cloud, AWS, and Cloudflare report largest DDoS attacks ever | ZDNET* [online]. 2023. [cit. 2024-02-27]. Dostupné z: https://www.zdnet.com/article/google-cloud-aws-and-cloudflare-report-largest-ddos-attacks-ever/.

41. *Famous DDoS attacks | Biggest DDoS attacks | Cloudflare* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/.

42. OWASP FOUNDATION. *About the OWASP Foundation* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://owasp.org/about/.

43. OWASP. *OWASP Cheat Sheet Series* [online]. [B.r.]. [cit. 2024-05-15]. Dostupné z: https://cheatsheetseries.owasp.org/index.html.

44. *O nás - CSIRT* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://csirt.cz/cs/o-nas/.

45. *CVE* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://www.cve.org/About/Overview.

46. *CWE* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://cwe.mitre.org/about/index.html.

47. IBM. *What Is Software Testing?* [Online]. [B.r.]. [cit. 2024-05-15]. Dostupné z: https://www.ibm.com/topics/software-testing.

48. MLEJNEK, Jiří. *Softwarové inženýrství - Přednáška 9* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://moodle-vyuka.cvut.cz/course/view.php?id=10710.

49. HIMSELF, Alexey. *Test Cases Taxonomy: FURPS+* [online]. 2017. [cit. 2024-05-15]. Dostupné z: https://medium.com/practical-software-testing/test-cases-taxonomy-a32900e1d994.

50. GAT. *The Ultimate Guide to Smoke Testing* [online]. 2024. Dostupné také z: https://www.globalapptesting.com/blog/the-ultimate-guide-to-smoke-testing. 2024-05-15.

51. KUBÁTOVÁ, Barbora. *Akceptační testování: Jak odhalit chyby na webu* [online]. 2018. [cit. 2024-05-15]. Dostupné z: https://www.blueghost.cz/clanek/akceptacni-testovani-weby/.

52. MLEJNEK, Jiří. *5. Přednáška BI-IDO: Release a nasazení • FIT ČVUT* [online]. 2024. [cit. 2024-02-26]. Dostupné z: https://courses.fit.cvut.cz/BI-IDO/lectures/5-lecture.html.

53. RAHMAN, Tasdik. *F.I.R.S.T principles of testing* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://medium.com/@tasdikrahman/f-i-r-s-t-principles-of-testing-1a497acda8d6.

54. WAKE, Bill. *3A - Arrange, Act, Assert - XP123* [online]. [B.r.]. [cit. 2024-02-27]. Dostupné z: https://xp123.com/3a-arrange-act-assert/.

55. BADKAR, Akshay. *Software Development Methodologies* [online]. 2023. [cit. 2024-05-15]. Dostupné z: https://www.simplilearn.com/software-development-methodologies-article.

56. MARTINS, Julia. *Iterative development* [online]. 2024. [cit. 2024-05-15]. Dostupné z: https://asana.com/resources/iterative-process.

57. METZ, Tim. *What Are The Most Popular Agile Methodologies in 2023?* [Online]. 2023. [cit. 2024-04-25]. Dostupné z: https://www.parabol.co/blog/most-popular-agile-methodologies/.

58. LUTKEVICH, Ben. *What is Scrum?* [Online]. 2021. [cit. 2024-04-25]. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Scrum.

59. PATEL, Bhaval. *A Detailed Guide to Web App Deployment [Process + Methods]* [online]. 2024. [cit. 2024-05-15]. Dostupné z: https://www.spaceotechnologies.com/blog/web-app-deployment/.

60. PERRY, Morgan. *Deployment Environments* [online]. 2023. [cit. 2024-05-15]. Dostupné z: https://www.qovery.com/blog/everything-you-need-to-know-about-deployment-environments/.

61. CYCLEOPS. *What is Web Application Monitoring?* [Online]. [B.r.]. [cit. 2024-05-15]. Dostupné z: https://cycleops.io/understanding-web-application-monitoring/.

62. *Best Error And Exception Monitoring Software in 2024 | 6sense* [online]. 2024. [cit. 2024-05-15]. Dostupné z: https://6sense.com/tech/error-an-exception-monitoring.

63. SENTRY. *Developer first Error Monitoring platform* [online]. [B.r.]. [cit. 2024-05-16]. Dostupné z: https://sentry.io/for/error-monitoring/.

64. MDN. *Fetching data from the server* [online]. [B.r.]. [cit. 2024-05-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Fetching_data.

65. REDHAT. *What is a REST API?* [Online]. 2020. [cit. 2024-04-25]. Dostupné z: https://www.redhat.com/en/topics/api/what-is-a-rest-api.

66. FIELDING, Roy. *Representational State Transfer (REST)* [online]. 2000. [cit. 2024-04-25]. Dostupné z: https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

67. YALCIN, Gozde Saygili. *Why is the REST API so popular?* [Online]. 2022. [cit. 2024-04-25]. Dostupné z: https://medium.com/@saygiligozde/why-is-rest-api-is-so-popular-ed9ca370b541.

68. NUXT.JS. *Clientside rendering* [online]. [B.r.]. [cit. 2024-05-15]. Dostupné z: https://nuxt.com/docs/guide/concepts/rendering#client-side-rendering.

69. NUXT. *Server-side (universal) rendering* [online]. [B.r.]. [cit. 2024-04-25]. Dostupné z: https://nuxt.com/docs/guide/concepts/rendering#universal-rendering.

70. JIROUT, David. *Informační systém pro MI-NUR*. Praha, 2019. Dostupné také z: http://hdl.handle.net/10467/80475. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.

71. PAVLŮSEK, Tomáš. *SOS II - Studentský odevzdávací systém*. Praha, 2023. Dostupné také z: http://hdl.handle.net/10467/108866. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.

72. PAVLŮSEK, Tomáš. *SOS - Studentský odevzdávací systém*. Praha, 2021. Dostupné také z: http://hdl.handle.net/10467/95107. Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií.

73. 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.
74. *FIT Klasifikace - API* [online]. [B.r.]. [cit. 2024-03-13]. Dostupné z: https://grades.fit.cvut.cz/api/v1/swagger-ui.html.
75. *PostgreSQL* [online]. [B.r.]. [cit. 2024-02-29]. Dostupné z: https://www.postgresql.org/about/.
76. EBY, Phillip J. *Python Web Server Gateway Interface*. 2010-09. PEP, 3333. Dostupné také z: https://peps.python.org/pep-3333/.
77. *Celery 5.3.6 documentation* [online]. [B.r.]. [cit. 2024-02-29]. Dostupné z: https://docs.celeryq.dev/en/stable/getting-started/introduction.html.
78. *OCI vs Docker: what's a container and how does it work? | Padok* [online]. [B.r.]. [cit. 2024-02-29]. Dostupné z: https://www.padok.fr/en/blog/container-docker-oci.
79. *Docker Compose overview | Docker Docs* [online]. [B.r.]. [cit. 2024-02-29]. Dostupné z: https://docs.docker.com/compose/.
80. *GitLab Documentation* [online]. [B.r.]. [cit. 2024-02-29]. Dostupné z: https://docs.gitlab.com/.
81. *What is Slack? | Slack* [online]. [B.r.]. [cit. 2024-04-09]. Dostupné z: https://slack.com/help/articles/115004071768-What-is-Slack-.
82. JIRŮTKA, Jakub. *Main - KOSapi - Projekt KOSapi* [online]. [B.r.]. [cit. 2024-02-29]. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki.
83. *Bakaláři – mezi školou a rodinou | Bakaláři* [online]. [B.r.]. [cit. 2024-02-29]. Dostupné z: https://www.bakalari.cz/.
84. *OAuth 2.0* [online]. [B.r.]. [cit. 2024-02-29]. Dostupné z: https://oauth.net/2/.
85. STICKDORN, Marc; HORMESS, Markus; LAWENCE, Adam; SCHNEIDER, Jakob. *This Is Service Design Doing*. Sebastopol, CA: O'Reilly Media, Inc, 2018. ISBN 978-1-491-92718-2.
86. JUŘENÍKOVÁ, Petra. *Kvantitativní výzkum* [online]. [B.r.]. [cit. 2024-04-09]. Dostupné z: https://is.muni.cz/do/rect/el/estud/lf/js19/metodika_zp/web/pages/07-kvantitativni.html.
87. *Multiple forms in Django templates* [online]. [B.r.]. [cit. 2024-03-13]. Dostupné z: https://www.django-antipatterns.com/antipattern/using-multiple-forms-on-the-same-page-without-prefixing.html.
88. *.NET* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://dotnet.microsoft.com/en-us/.

89. *.NET and .NET Core official support policy* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://dotnet.microsoft.com/en-us/platform/support/policy/dotnet-core.

90. FOWLER, Martin. *Anemic Domain Model* [online]. 2003. [cit. 2024-04-28]. Dostupné z: https://martinfowler.com/bliki/AnemicDomainModel.html.

91. CODEJOURNEY. *TypeScript Compiler Explained - CodeJourney.net* [online]. 2023. [cit. 2024-05-13]. Dostupné z: https://www.codejourney.net/typescript-compiler-explained/.

92. *Introduction | Vue.js* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://vuejs.org/guide/introduction.html%5C#what-is-vue.

93. VUE.JS. *Composables* [online]. 2024. [cit. 2024-04-15]. Dostupné z: https://vuejs.org/guide/reusability/composables.

94. *Tailwind CSS* [online]. [B.r.]. [cit. 2024-03-18]. Dostupné z: https://tailwindcss.com/.

95. *Routing* [online]. [B.r.]. [cit. 2024-03-17]. Dostupné z: https://nuxt.com/docs/getting-started/routing.

96. *Nuxt Security* [online]. [B.r.]. [cit. 2024-03-17]. Dostupné z: https://nuxt-security.vercel.app/.

97. LEANPUB. *Git Flow* [online]. [B.r.]. [cit. 2024-03-16]. Dostupné z: https://leanpub.com/git-flow/read.

98. *Git - Tagging* [online]. [B.r.]. [cit. 2024-03-16]. Dostupné z: https://git-scm.com/book/en/v2/Git-Basics-Tagging.

99. *Figma* [online]. [B.r.]. [cit. 2024-04-06]. Dostupné z: https://www.figma.com/design/.

100. WIEGERS, Karl; BEATTY, Joy. *Software Requirements*. Microsoft Press, 2013. ISBN 978-0-7356-7966-5.

101. *Nefunkční požadavky* [online]. [B.r.]. Dostupné také z: https://managementmania.com/cs/nefunkcni-pozadavky-non-functional-requirements. 2024-05-15.

102. JONES, Michael B.; BRADLEY, John; SAKIMURA, Nat. *JSON Web Token (JWT)* [RFC 7519]. RFC Editor, 2015 [cit. 2024-05-13]. Request for Comments, č. 7519. Dostupné z DOI: 10.17487/RFC7519.

103. MEGIDA, Dillion. *Event Bubbling in JavaScript* [online]. 2022. [cit. 2024-04-23]. Dostupné z: https://www.freecodecamp.org/news/event-bubbling-in-javascript/.

104. KENT C. DODDS. *Prop Drilling* [online]. 2018. Dostupné také z: https://kentcdodds.com/blog/prop-drilling. 2024-04-23.

105. JANÁSEK, Robin. *Jak na uživatelské testování* [online]. 2022. [cit. 2024-04-27]. Dostupné z: https://www.proofreason.com/blog/metody-uzivatelskeho-testovani.

106. NIELSEN, Jacob. *Why You Only Need to Test with 5 Users* [online]. 2000. [cit. 2024-04-27]. Dostupné z: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `source`
  - `backend` ..... zdrojové kódy backendové části aplikace
  - `frontend` ..... zdrojové kódy frontendové části aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF