FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	SOS III - Studentský odevzdávací systém - týmové řešení projektů
Student:	Marko Hujo
Vedoucí:	Ing. Jiří Hunka
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 28. 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 Janem Mrázkem, kteří řeší ostatní části systému.
Přímo tato práce je primárně zaměřena na řešení týmů a projektů.

Postupujte v těchto krocích:
1. Analyzujte současný stav portálu včetně dílčích úprav ostatních autorů.
2. Analyzujte vhodným způsobem převážně aktuální možnosti týmů a týmových i soukromých projektů 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. Michal Valenta, Ph.D. dne 28. ledna 2024 v Praze.

Bakalárska práca

SOS III – STUDENTSKÝ ODEVZDÁVACÍ SYSTÉM – TÝMOVÉ ŘEŠENÍ PROJEKTŮ

Marko Hujo

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedúci: Ing. Jiří Hunka
16. mája 2024

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Chcel by som poďakovať najmä vedúcemu bakalárskej práce Ing. Jiřímu Hunkovi za vedenie mojej práce, cenné odborné rady a za skvelú príležitosť vytvoriť systém podporujúci výuku viacerých predmetov na FIT ČVUT v Prahe. Moje poďakovanie si tiež zaslúži Bc. Max Hejda za jeho rady, konzultácie a podávanie spätnej väzby na kód a Jan Jamnický, Jan Mrázek a Timotej Adamec za spoluprácu hlavne v predmete BI-SP2 počas zimného semestra 2023/2024. Tiež by som rád poďakoval študentom predmetu BI-SP1, ktorí na projekte pracovali v letnom semestri 2023/24. V neposlednom rade ďakujem svojej rodine a kamarátom za podporu a motiváciu počas celého štúdia.

Vyhlásenie

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 Prahe dňa 16. mája 2024

Abstrakt

Studentský odevzdávací systém (SOS) je webová aplikácia pre správu tímových semestrálnych prác na Fakultně informačních technologií (FIT) Českého vysokého učení technického (ČVUT) v Praze a ročníkových prác na Gymnáziu Jiřího Gutha-Jarkovského (GJGJ). Táto bakalárska práca sa venuje analýze tejto aplikácie a návrhu a realizácii novej aplikácie, ktorá tú súčasnú v budúcnosti nahradí. Cieľom práce je navrhnúť architektúru a užívateľské rozhranie novej aplikácie, aby bola nová aplikácia škálovateľná, jednoducho udržateľná a rozšíriteľná, jednoduchá na použitie, prehľadná a intuitívna. Ďalším cieľom je reálne vyskúšať iteratívny vývoj softwaru a prácu v tíme na bakalárskom projekte. Výsledkom práce je funkčná nová aplikácia, ktorá podporuje najprioritnejšie funkcionality vrátane tvorby a správy tímových projektov a priradeného projektového tímu, schvaľovania študentmi vytvorených projektov, odovzdávania a hodnotenia. Počas vývoja bola aplikácia testovaná jednotkovými a integračnými testami. Po implementácii bolo otestované a zvalidované aj nové užívateľské rozhranie užívateľským testovaním na piatich vybraných užívateľoch. Nová aplikácia je nasadená a pripravená pre ďalší rozvoj a skoré produkčné využitie na FIT ČVUT.

Kľúčové slová webová aplikácia, správa tímových projektov, podpora výuky, návrh webovej aplikácie, iteratívný vývoj softwaru, .NET, ASP.NET Core, Vue.js, Nuxt

Abstract

Student Submission System is a web application for managing team projects at the Faculty of Information Technology (FIT) of the Czech Technical University (CTU) in Prague and seminar works at the Jiří Guth-Jarkovský Gymnasium. This bachelor thesis is devoted to the analysis of this application and the design and implementation of a new application that will replace the current

one in the future. The aim of this thesis is to design the architecture and user interface of the new application in order to make the new application scalable, easily sustainable and extensible, simple to use, clear and intuitive. Another goal is to realistically experience iterative software development and team cooperation. The result of the thesis is a functional new application that supports the most prioritized functionalities, including the creation and management of team projects and assigned project team, approval of student-created projects, submission and evaluation. During development, the application was tested using unit and integration tests. After implementation, the new user interface was also tested and validated by usability testing on 5 selected users. The new application is deployed and ready for further development and early production use at FIT CTU.

Keywords web application, team project management, teaching support, web application design, iterative software development, .NET, ASP.NET Core, Vue.js, Nuxt

Zoznam skratiek

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
BI-KOM	Konceptuální modelování (bakalársky predmet na FIT ČVUT)
BI-OOP	Object-Oriented Programming (bakalársky predmet na FIT ČVUT)
BI-PRR	Projektové řízení (bakalársky predmet na FIT ČVUT)
BI-PST	Pravděpodobnost a statistika (bakalársky predmet na FIT ČVUT)
BI-SP1	Softwarový týmový projekt 1 (bakalársky predmet na FIT ČVUT)
BI-SP2	Softwarový týmový projekt 2 (bakalársky predmet na FIT ČVUT)
BI-SWI	Softwarové inženýrství (bakalársky predmet na FIT ČVUT)
BI-TIS	Tvorba informačních systémů (bakalársky predmet na FIT ČVUT)
ČVUT	České vysoké učení technické
FIT	Fakulta informačních technologií
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
JSON	JavaScript Object Notaion
KOS	Komponenta Studium (Študijný informačný systém na ČVUT)
MVP	Minimum Viable Product
MVC	Model View Controller
MPA	Multi Page Application
MVT	Model View Template
NI-NUR	Návrh uživatelského rozhraní (magisterský predmet na FIT ČVUT)
REST	REpresentational State Transfer
SPA	Single Page Application
SQL	Structured Query Language
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UML	Unified Modeling Language
XP	Extrémne programovanie

Úvod

S nárastom počtu študentov sa nevyhnutne zvyšuje potreba efektívnej správy vzdelávacieho procesu a využitia softwarových systémov pre podporu výuky. Táto potreba je aktuálna aj na FIT ČVUT, kde sa vyučujú predmety, ako NI-NUR (Návrh užívateľského rozhrania), BI-OOP (Objektovo orientované programovanie), BI-SP1 (Softvérový týmový projekt 1) a BI-SP2 (Softvérový týmový projekt 2), BI-PRR (Projektové řížení) a BI-TIS (Tvorba informačních systémů), kde sú semestrálne práce vypracovávané v tímoch kľúčovou súčasťou výuky.

Donedávna však chýbal jednotný odovzdávací systém, ktorý by dokázal uspokojiť špecifické požiadavky jednotlivých predmetov. Súčasné univerzálne systémy, ako napríklad Moodle, nedokážu adekvátne podporovať všetky požiadavky všetkých predmetov. Práve z tohto dôvodu vznikol systém SOS, ktorý je prispôsobený potrebám jednotlivých predmetov, ale zároveň univerzálny a rozšíriteľný pre viaceré predmety. V minulom roku som sa aj ja aktívne podieľal na vývoji tohto systému. Moje zapojenie do projektu sa začalo vo februári 2023, konkrétne v predmete BI-SP1. Následne som, v rámci nadväzujúceho predmetu BI-SP2, začal pracovať na novom systéme s novým užívateľským rozhraním postavenom na novej architektúre a nových technológiách.

Hlavným cieľom tejto práce je zaistiť lepší technický stav systému SOS pre budúci vývoj, teda zaistiť jednoduchú udržateľnosť a rozšíriteľnosť tohto systému. Ďalším hlavným cieľom je zo získaných skúseností so súčasným systémom a zo zistených nedostatkov užívateľského rozhrania súčasného systému, navrhnúť jednoduchšie, prehľadnejšie a intuitívnejšie užívateľské rozhranie. Cieľom práce je teda analyzovať súčasný systém SOS vrátane úprav ostatných predchádzajúcich autorov a funkcionalít týkajúcich sa tvorby a správy tímov a tímových projektov. Pri analýze je dôležité brať ohľad na budúce rozšírenia a využitie nie len na FIT ČVUT, ale aj v iných prostrediach. Ďalším cieľom je získať spätnú väzbu od užívateľov systému SOS, zvalidovať existujúce požiadavky a získať a analyzovať nové. Na základe tejto analýzy je cieľom navrhnúť

a realizovať nový systém SOS, ktorý v budúcnosti nahradí ten súčasný. Ďalšími dôležitými cieľmi práce je výsledný systém otestovať vhodne zvolenými testami a zaistiť a pripraviť projekt na budúce jednoduché a efektívne rozšírenia systému ďalšími ľudmi a na reálne využitie na FIT ČVUT.

Čiastočným cieľom práce je reálne vyskúšať iteratívnu formu vývoja softwaru a prácu v tíme s kolegami Janom Jamnickým a Janom Mrázkom, ktorí sa v rámci svojich bakalárskych prác venujú ďalším častiam projektu. Mojím zameraním práce sú primárne tímy a tímové projekty, kolega Jamnický sa venuje predmetom, semestrom, paralelkám a užívateľom a kolega Mrázek kontrolným bodom, odovzdávaniu a hodnoteniu.

Kapitola 1

Vývoj softwaru v tíme

V prvej kapitole sa zaoberám vývojom softwaru v tíme a porovnávam rôzne spôsoby a prístupy k tímovému vývoju. Definujem vodopádový model vývoja, iteratívny vývoj softwaru a agilné metodiky – extrémne programovanie a Scrum. V druhej časti tejto kapitoly sa venujem predstaveniu základných troch verzovacích modelov, ktoré sú pri vývoji softwaru v tíme potrebné.

1.1 Spôsoby a metodiky vývoja softwaru

Jedným z cieľov tejto práce je reálne vyskúšať iteratívnu formu vývoja softwaru a prácu v tíme na bakalárskom projekte. Na začiatku vývoja nového systému sme preto s tímom čelili výberu vhodného spôsobu vývoja pre náš prípad. Rozhodovali sme sa aj na základe tejto analýzy spôsobov a metodík vývoja softwaru. V tejto časti vysvetľujem, ako a prečo jednotlivé spôsoby a metodiky vznikali, aké postupy priniesli a aké sú ich výhody, nevýhody a vhodné použitia. Pred samotným predstavením a analýzou jednotlivých metód a metodík vývoja by som ale pre lepšie pochopenie tejto problematiky čitateľa rád oboznámil s pojmom softwarový proces a základnými činnosťami softwarového inžinierstva.

Softwarový proces označuje množinu súvisiacich aktivít a krokov, ktoré vedú k úspešnému dokončeniu softwarového projektu a vzniku nového softwarového systému. Základnými činnosťami softwarového inžinierstva sú analýza, návrh, implementácia, testovanie a údržba softwaru. Softwarový proces v určitej forme zahŕňa všetky uvedené základné činnosti softwarového inžinierstva. Tieto činnosti sú samy o sebe komplexné a zahŕňajú podčinnosti, ako sú zber požiadaviek, návrh architektúry systému alebo užívateľské testovanie. Okrem základých činností softwarového inžinierstva zahŕňa tvorba softwaru aj vedľajšie činnosti softwarového inžinierstva ako napríklad riadenie projektu, za-

bezpečenie kvality softwaru, tvorbu dokumentácie alebo konfiguráciu systému, ktoré vykonávame počas celého procesu vývoja softwaru. [1, kap. 2]

Metodiky vývoja softwaru môžeme chápať ako komplexné pravidlá, postupy a návody pre vývoj softwarového systému. Metodiky vývoja ale nerozoberajú konkrétne spôsoby, ako niečo vykonať alebo pomocou čoho niečo vykonať. Skôr sa zaoberajú pohľadom z výšky a hľadaním nadhľadu nad celým softwarovým procesom a jeho fázami a venujú sa tejto problematike ako celku. Určujú teda časové závislosti jednotlivých etáp vývoja, ich frekvenciu opakovania, očakávané vstupy a výstupy a nároky na ich vykonanie. [2]

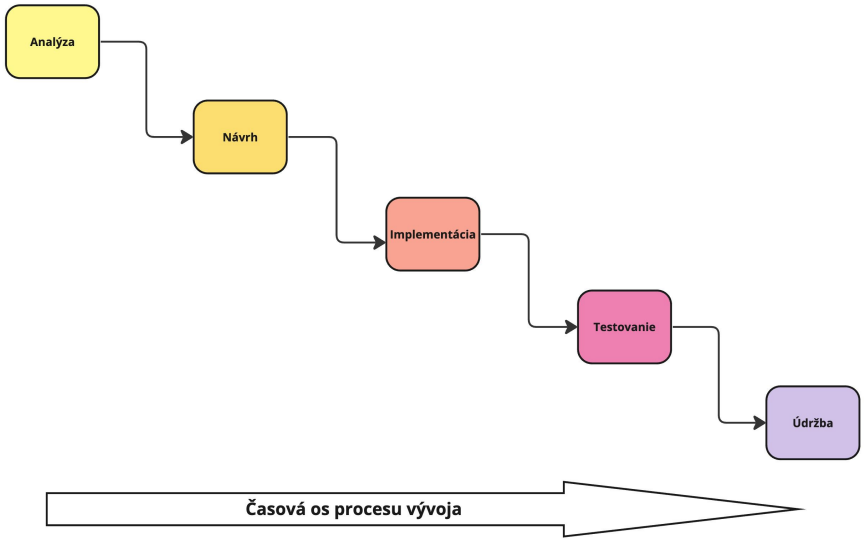
Adaptovanie metodík vývoja softwaru je spojené s rozličnými výhodami. V prípade menších softwarových projektov môže byť použitie špecifickej metodiky považované za zbytočné. Avšak pri práci na väčších projektoch v tíme alebo dokonca vo viacerých tímoch sa prirodzene ukazuje ako nevyhnutné riadiť sa vopred stanovenými a jasne formulovanými pravidlami. Práve toto považujem za jednu z kľúčových výhod používania metodík vývoja. Metodiky vývoja nám totižto poskytujú pravidlá, ktoré uľahčujú spoluprácu a komunikáciu nielen v rámci jedného vývojového tímu, ale aj medzi tímami a všetkými zainteresovanými stranami vrátane zákazníkov. Existuje viacero metodík, ktoré prinášajú rôzne benefity, ktoré budú podrobnejšie rozobraté v ďalších častiach textu. Výber tej správnej metodiky závisí od konkrétnych potrieb projektu, a preto nie je možné určiť jednu, ktorá by bola najlepšia vo všetkých situáciách, pretože rôzne typy softwarových projektov vyžadujú rôzne prístupy. V praxi sa často stretávame s tým, že tímy striktne nedodržiavajú jednu metodiku, ale upravujú alebo dokonca kombinujú prvky z rôznych metodík tak, aby vyhovovali ich špecifickým potrebám.

1.1.1 Vodopád

Vodopád nie je úplne metodikou, je skôr modelom životného cyklu vývoja softwaru. Ako vidieť na obrázku 1.1, vodopád je špecifický oddelenými fázami softwarového procesu, ktorými sú analýza požiadavkov, návrh, implementácia, testovanie a údržba. Fáze vodopádu teda priamo reflektujú základné činnosti softwarového inžinierstva. Tieto fázy sú vykonávané sekvenčne za sebou v stanovenom poradí so žiadnymi alebo minimálnymi iteráciami. Vývoj teda postupuje tak, že až po skončení jednej fázy je zahájená nasledujúca fáza a preto nie je možné vykonávať viac fáz naraz. [2]

Pri vývoji softwaru sa ale fázy vývoja prirodzene prekrývajú. Počas návrhu sa identifikujú problémy s požiadavkami, počas implementácie sa zistia problémy s návrhom atď. Softwarový proces nikdy nie je jednoduchý lineárny model. V praxi je často nemožné od zákazníka zistiť a pochopiť všetky požiadavky ešte pred samotnou implementáciou. Často sa stretávame s tým, že zákaz-

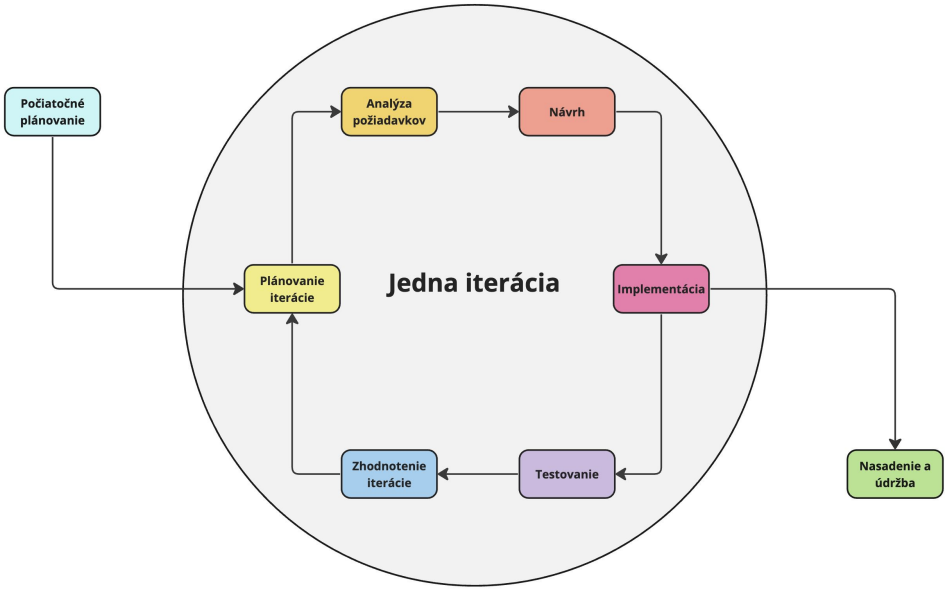
ník svoje požiadavky v priebehu vývoja mení alebo zistíme, že sme zákazníka nesprávne pochopili. Pri vodopádovom vývoji ale nie sme schopní na takéto zmeny rýchlo reagovať. Vodopádový model preto nie je vhodným v situáciách, kedy je potrebné sa rýchlo prispôsobovať zmenám. Iteratívny vývoj a agilné metodiky, ktoré analyzujem nižšie, sú v týchto situáciách pravdepodobne vhodnejšie.



Obr. 1.1 Vodopádový model vývoja softwaru

1.1.2 Iteratívny vývoj

Iteratívny vývoj, ktorý je znázornený na obrázku 1.2, je založený na myšlienke vyvinúť počiatočnú verziu systému, získať spätnú väzbu od zákazníka a užívateľov a postupne vyvíjať ďalšie verzie, kým nevznikne požadovaný systém. Každá iterácia alebo verzia systému zahŕňa niektoré funkcionality požadované zákazníkom. Vo všeobecnosti platí, že prvé verzie systému obsahujú najdôležitejšie alebo najnaliehavejšie funkcionality. To znamená, že zákazník alebo užívateľ môže systém vyhodnotiť v relatívne skorej fáze vývoja, aby zistil, či poskytuje to, čo požaduje. Včasné dodanie a nasadenie softwaru je možné, aj keď nie sú naimplementované všetky funkcionality. Zákazníci teda môžu software používať oveľa skôr ako pri vývoji vodopádom. Iteratívny vývoj, ktorý je základom agilných metodík, je preto lepší ako vodopád pre systémy, ktorých požiadavky sa počas procesu vývoja môžu často meniť, čo platí pre väčšinu softwarových systémov. Na druhú stranu je pravdepodobné, že pravidelné zmeny môžu viesť k neprehľadnému kódu, následkom čoho je pridávanie nových funkcionalít čoraz ťažšie a nákladnejšie. [1, kap. 2]



Obr. 1.2 Iteratívny vývoj softwaru

Iteratívny vývoj vlastne reflektuje spôsob, akým bežne riešime problémy a prichádzame s riešeniami. Predstavme si, že máme nejaký problém, ktorý potrebujeme vyriešiť. Málokedy nájdeme kompletné riešenie celého problému, ale skôr sa problém snažíme rozdeliť na menšie časti a postupujeme k riešeniu v niekoľkých krokoch, pričom sa vraciame späť, keď si uvedomíme, že sme urobili chybu. Presne takýto postup riešenia problémov sme sa učili aj vo viacerých matematických a algoritmických predmetoch na FIT ČVUT.

1.1.3 Agilné metodiky

Keďže dnes žijeme a pracujeme v rýchlo sa meniacom prostredí, rýchly vývoj a dodanie softwaru je jednou z najdôležitejších požiadaviek väčšiny zákazníkov. Práve táto potreba stála za vznikom myšlienok o agilnom vývoji softwaru, ktoré prináša zmenu myslenia celého tímu a všetkých zúčastnených strán vrátane zákazníkov.

Základom agilného vývoja je, že software je vyvíjaný iteratívne s veľmi krátkymi iteráciami. Úlohou zákazníka je špecifikovať a prioritizovať nové požiadavky do ďalších iterácií a hodnotiť výsledky každej iterácie. Zákazníci sú teda súčasťou vývoja a s vývojovými tímami úzko spolupracujú a osobne komunikujú takmer na každodennej báze, čím dávajú vývojárom rýchlu spätnú väzbu na meniace sa požiadavky. Okrem toho poskytujú vývojárom rýchlu podporu a odpovede na ich otázky. V praxi je agilný vývoj veľmi používaný a obľúbený

kvôli kontinuálnemu a skorému dodávaniu nového softwaru, čo samozrejme prispieva k vyššej spokojnosti zákazníka. [1, 2, 3]

V ďalších častiach predstavím agilné metodiky XP (Extreme Programming[1]) a Scrum, ktorý je v praxi najpopulárnejšou agilnou metodikou.

1.1.3.1 Extrémne programovanie

Extrémne programovanie je agilnou metodikou, ktorej hlavným prínosom je súbor jednoduchých a konkrétnych postupov.

Krátke iterácie: Pri aplikovaní XP vyvíjame a dodávame funkčný software v krátkych iteráciách, zvyčajne každé dva týždne. Každá z týchto iterácií vytvára funkčný software, ktorý rieši nejaký súbor funkcionalít vybraných zákazníkom. Tento nový software môže, ale nemusí byť uvedený do produkcie. Na konci každej iterácie sa systém predvedie zákazníkovi, prípadne aj vybraným užívateľom s cieľom získať ich spätnú väzbu. [4, kap. 2]

***User story*:** *User story* je neformálne, všeobecné vysvetlenie funkcionality softwaru napísané z pohľadu koncového užívateľa. Jeho cieľom je vyjadriť, ako daná funkcionalita softwaru užívateľovi pomôže vyriešiť jeho problém. [5]

Plánovanie a *story points*: Na začiatku projektu sa spoločne so zákazníkom snažíme identifikovať všetky dôležité *user stories* a určiť ich prioritu. Zvyčajne však nevieme identifikovať všetky, pretože v priebehu projektu budú zákazníci naďalej pridávať nové. [4, kap. 3] Zložitosť úloh odhadujeme pomocou tzv. *story points*, ktoré vyjadrujú celkové úsilie potrebné na úplnu realizáciu *user story*. [6] Na začiatku každej iterácie vývojári rozdelia *user stories* na úlohy. Úloha by ideálne mala byť niečo, čo vie jeden vývojár implementovať za 4-16 hodín. Vývojári spoločne odhadujú náklady implementácie úloh v *story pointoch* a zákazníkom dajú rozpočet *story points* na základe toho, koľko sa im podarilo dokončiť v poslednej iterácii. Zákazníci si potom podľa priority vyberú úlohy, ktorých náklady v súčte dosahujú, ale nepresahujú tento stanovený rozpočet. [4, kap. 3]

Refaktoring: Pri aplikovaní agilného vývoja a XP navrhujeme softwaru čo najjednoduchšie, pri čom sa sústredíme len na funkcionality danej iterácie. Snažíme sa rýchlo a efektívne reagovať na časté zmeny. Časté zmeny v kóde ale majú tendenciu zhoršovať čitateľnosť a štruktúru kódu. Tento problém agilného vývoja sa ale snaží riešiť práve refaktoring. Refaktoring je proces zmeny softwarového systému takým spôsobom, ktorý nemení jeho funkcionalitu a vonkajšie správanie pre koncového užívateľa, ale zlepšuje vnútornú štruktúru zdrojového kódu, aby bol jednoduchší na pochopenie

[1] V preklade „extrémne programovanie“

a lacnejší na úpravu. Podstatou refaktoringu je vykonávanie viacero malých zmien, ktoré zachovávajú funkcionalitu, pričom každá zo zmien je sama o sebe dosť malá a nepodstatná, avšak celkový efekt týchto zmien je pomerne významný. [7]

Párové programovanie: Všetok kód je písaný párom programátorov, ktorí spolupracujú za jedným počítačom. Jeden člen páru píše kód a druhý hľadá chyby a navrhuje vylepšenia. [4, kap. 2]

***Test-driven development*:** Testovanie je automatizované a je základným prvkom vývojového procesu. Vývoj nemôže pokračovať, kým sa úspešne nevykonajú všetky testy. [1, kap. 3] Najprv napíšeme jednotkový test, potom napíšeme kód, ktorý spôsobí, že tento test prejde. Výsledkom je, že spolu s kódom rastie aj súbor testov. Okrem toho máme tiež silnú motiváciu oddeliť moduly od seba, aby sa dali testovať nezávisle a jednoducho. Kód, ktorý je napísaný týmto spôsobom, má teda tendenciu byť oveľa menej previazaný, vďaka čomu vieme dodržiavať princípy objektovo orientovaného návrhu, ako je napríklad *Single responsibility principle*[2]. [4, kap. 2 a 4]

Akceptačné testy: Podrobnosti o *user stories* sú zachytené vo forme akceptačných testov zadaných zákazníkom. Ich úlohou je overiť, či sa systém správa tak, ako zákazník určil. [4, kap. 2] Ak vývojár implementuje kód potrebný na splnenie akceptačných testov, *user story* sa považuje za správne implementovaný a dokončený.

Cieľom XP je zlepšiť kvalitu softwaru a schopnosť reagovať na meniace sa požiadavky zákazníkov. V praxi sa ale ukázalo, že aplikovanie XP v pôvodne navrhovanej forme, teda aplikovanie všetkých vyššie spomenutých postupov, je náročnejšie, ako sa očakávalo. Preto si tímy vyvíjajúce agilne vyberajú tie postupy a praktiky extrémneho programovania, ktoré sú pre ich spôsob práce najvhodnejšie. Často sú tieto vybrané praktiky používané v spojení s inou agilnou metodikou ako je napríklad Scrum. [1, kap. 3]

1.1.3.2 Scrum

Scrum a agilný vývoj sa často považujú za to isté [8], čo ale nie je úplne pravda. Agilný vývoj je len zoznam princípov pre tvorbu softwaru z agilného manifesta. Scrum je konkrétna agilná metodika pre vývoj a údržbu komplexných softwarových produktov, ktorá sa síce riadi zásadami z agilného manifesta, ale nenariaďuje používanie konkrétnych vývojových postupov. To znamená, že ho možno ľahšie integrovať do zaužívaných postupov v tímoch. [1, 9]

Základom Scrumu je malý tím s maximálne desiatimi členmi. V praxi sa ale ukazuje, že menšie tímy lepšie komunikujú a sú aj viac produktívne. Scrum

[2] V preklade „Princíp jednej zodpovednosti“

tímy sú multifunkčné, čo znamená, že všetci členovia majú všetky zručnosti potrebné pre dodanie softwaru zákazníkovi. V Scrum tíme majú členovia na základe svojich zodpovedností jednu z nasledovných troch rolí [9, 10]:

Vývojár: Úlohou vývojarov je v jednoduchosti vykonávať prácu, ktorá vytvorí akýkoľvek použiteľný a viditeľný pokrok. Vývojový tím by mal byť schopný samoorganizácie, aby mohli vývojari sami robiť rozhodnutia na dokončenie ich práce.

***Product Owner*:** *Product Owner*[3] je člen tímu, ktorý rozumie požiadavkám zákazníka a biznisu a vytvára a spravuje produktový *backlog*, ktorý popíšem nižšie. Jeho najväčšou zodpovednosťou je prioritizácia práce.

***Scrum Master*:** *Scrum Master* celý tím spája, je zodpovedný za to, aby sa zabezpečilo, že sa Scrum vykonáva dobre a že všetko v tíme funguje. V praxi to znamená, že pomáha *Product Ownerovi* aj vývojovému tímu, čím zlepšuje fungovanie celého tímu.

Scrum je definovaný nasledujúcimi praktikami, ktoré agilné tímy používajúce Scrum, dodržiavajú:

***Scrum Sprint*:** *Scrum Sprint* označuje iteráciu v rámci Scrumu, ktorá zvyčajne trvá jeden alebo dva týždne. Hlavným cieľom týchto krátkych iterácií je tímom pomáhať dodržiavať agilné princípy častého dodávania softwaru a rýchleho reagovania na zmeny. [8, 9]

Backlog*:** **Produktový *backlog je prioritizovaný zoznam úloh pre vývojový tím, ktorý je odvodený z plánu projektu a jeho požiadaviek. Existuje ešte aj ***backlog* jednotlivých šprintov**, ktorý definuje úlohy, na ktorých tím pracuje počas daného šprintu. [9, 11]

***Sprint Planning*:** *Sprint Planning*[4] je stretnutie celého Scrum tímu, ktoré odštartuje každý šprint. Účelom tohto stretnutia je definovať, na akých úlohách z *backlogu* bude tím pracovať a ako vie tím túto prácu čo najefektívnejšie dosiahnuť. [9]

***Stand-up*:** *Stand-up* je krátke každodenné stretnutie, na ktorom celý Scrum tím stručne diskutuje o pokroku, ktorý každý člen učinil za posledný deň a pláne na ďalší deň. Každý člen tímu by mal na *stand-upe* odpovedať na nasledujúce tri otázky, najlepšie v tomto poradí [9, 12]:

1. Na čom som pracoval včera?
2. Na čom plánujem pracovať dnes?
3. Narazil som sa nejaké problémy, ktoré mi prekážajú v práci?

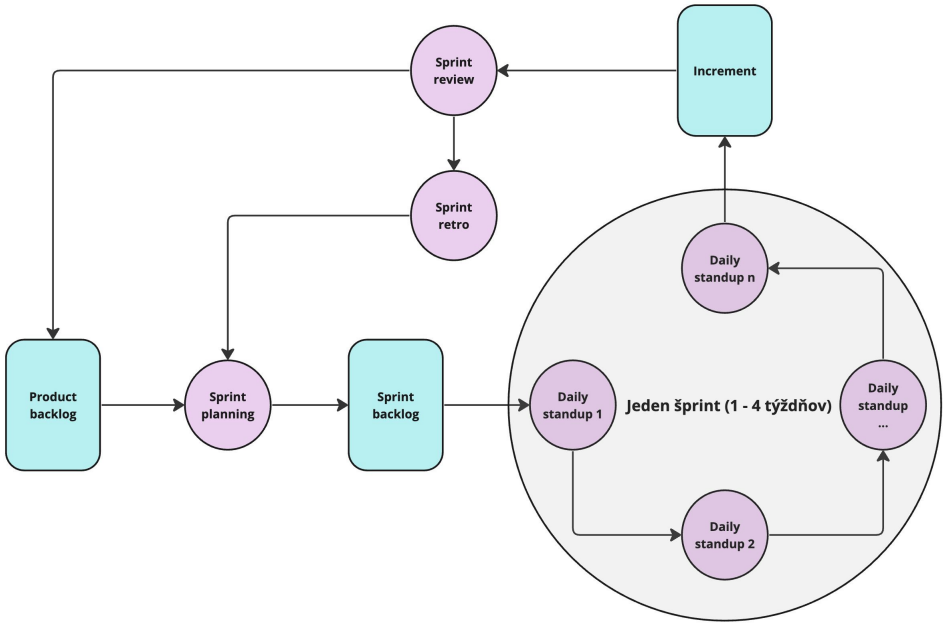
[3] V preklade „vlastník produktu“

[4] Alebo v preklade „plánovanie šprintu“

***Sprint Review*:** Ďalším stretnutím, ktorého sa Scrum tím účastní je *Sprint Review*, ktorého cieľom je zákazníkom odprezentovať dokončené úlohy šprintu. Členovia tímu teda neformálne predvedú výsledky svojej práce v danom šprinte. Počas tohto stretnutia sa zákazníci pýtajú otázky, skúšajú nové funkcionality a poskytujú spätnú väzbu. [9]

***Sprint Retrospective*:** Účelom tohto stretnutia je prísť na to, ako do budúcna zefektívniť a skvalitniť prácu celého tímu na základe skúseností z predošlého alebo predošlých šprintov. [9]

Pre lepšiu ilustráciu metodiky Scrum som vytvoril obrázok 1.3.



Obr. 1.3 Scrum

1.2 Verzovacie modely

Verzovacie systémy, ako je Git, sú neoddeliteľnou súčasťou moderného vývoja softwaru poskytujúce základ pre efektívnu spoluprácu a správu zmien v kóde. Konsenzus o postupoch v tíme je jedným z kľúčových faktorov úspechu projektu. Neexistuje ale nejaký jednotný alebo štandardizovaný postup, ktorý by bol vhodný pre všetky projekty a tímy. Je avšak kľúčové, aby si členovia tímu ujasnili spoločný prístup k používaniu Gitu. Tento proces vyberania a dohodnutia sa na spôsobe práce s Gitom sa označuje ako výber verzovaciaho modelu (alebo anglicky *Git workflow*). *Git workflow* predstavuje súbor

dohodnutých postupov v rámci tímu, ktoré určujú, ako sa Git bude v projekte využívať.

Existuje viacero rôznych variant, z ktorých každý je navrhnutý tak, aby vyhovoval rôznym typom projektov. Tieto varianty môžu zahŕňať rôzne stratégie pre vetvenie, zlúčenie a správu zmien, a sú navrhnuté tak, aby zefektívňovali spoluprácu a minimalizovali potenciálne konflikty v kóde.

Jedná sa ale skôr o nejaké usmernenie ako konkrétne pravidlá, takže v praxi si tímy často vytvoria vlastný model, ktorý reflektuje konkrétne potreby a preferencie daného tímu. Takéto prispôsobené prístupy sú zvyčajne odvodené z osvedčených modelov, ale sú upravené tak, aby čo najlepšie vyhovovali danému projektu.

Výber vhodného verzovacieho modelu je teda základným krokom k zabezpečeniu hladkého priebehu projektu a efektívnej spolupráce v rámci tímu. Je dôležité, aby bol tento výber urobený s ohľadom na špecifické potreby projektu. Pri výbere vhodného verzovacieho modelu pre tento projekt sme s tímom preskúmali tri z najčastejšie používaných modelov, ktoré nižšie definujem.

1.2.1 Git Flow

Git Flow predstavuje najkomplexnejší workflow z nasledujúcich troch založený na efektívnom využívaní rôznych typov vetiev. V rámci Git Flow existujú v repozitári projektu nasledujúce dve permanentné vetvy, každá s odlišnými úlohami [13, 14]:

Hlavná vetva (*main* alebo *master*): Táto vetva obsahuje kód, ktorý bol otestovaný, skontrolovaný ostatnými vývojármi z tímu a je pripravený na nasadenie do produkčného prostredia. Jednotlivé commity na tejto vetve sú označené tagmi, čo umožňuje jednoduchú identifikáciu a správu rôznych vydaní softvéru.

Vývojová vetva (*develop*): Slúži ako hlavná vetva pre vývoj, do ktorej sú pridávané nové funkcionality pripravené na budúce vydanie a nasadanie do produkčného prostredia. Táto vetva teda predstavuje základ pre tvorbu nových funkcionalít a integráciu zmien.

Okrem hlavných vetiev Git Flow zahŕňa aj niekoľko typov podporných vetiev, ktoré umožňujú paralelný vývoj, jednoduchú spoluprácu a efektívnu správu rôznych fáz vývojového cyklu. Podporné vetvy vieme kategorizovať nasledovne podľa svojho účelu [13, 14]:

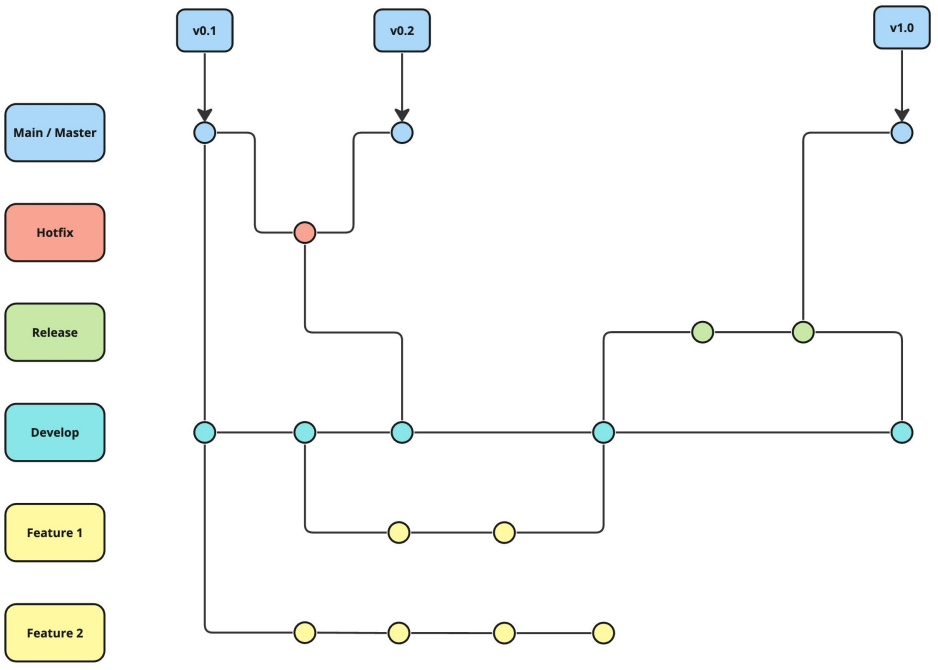
***Feature* vetvy:** *Feature* vetvy sú vytvárané z vetvy *develop* a slúžia na vývoj nových funkcionalít. Po dokončení vývoja a revízii sú zmeny zlúčené späť do vetvy *develop*. Každá nová funkcionalita by mala mať svoju vlastnú *feature*

vetvu a preto sa pravdepodobne jedná o najčastejší typ vetvy počas celého procesu vývoja.

***Release* vetvy:** Po integrácii všetkých plánovaných funkcionalít do vetvy *develop* je vytvorená *release* vetva, ktorá slúži na prípravu, zhromaždenie kódu a finálne úpravy pred vydaním novej verzie do produkčného prostredia. Do *release* vetve už pridávame iba drobnosti súvisiace s vydaním danej verzie napríklad opravy menších chýb nových funkcionalít, úpravy dokumentácie týkajúce sa nových funkcionalít alebo iné záverečné úpravy pred nasadením. Po dokončení sú zmeny zlúčené do vetvy *main* a označené tagom reprezentujúcim vydanú verziu, ako aj do vetvy *develop* pre zachovanie konzistencie.

***Hotfix* vetvy:** V prípade objavenia kritických chýb v produkčnej verzii slúžia *hotfix* vetvy na ich rýchlu opravu. Tieto vetvy vychádzajú priamo z vetvy *main*, a po oprave sú zmeny zlúčené späť do *main* a tiež do *develop*, aby sa zabezpečilo, že oprava bude zahrnutá aj v budúcich vydaniach.

Obrázok 1.4 ilustruje štruktúru modelu a vzájomných vzťahov medzi jednotlivými vetvami, zjednodušujúc tak pochopenie celého modelu a jeho použitia.



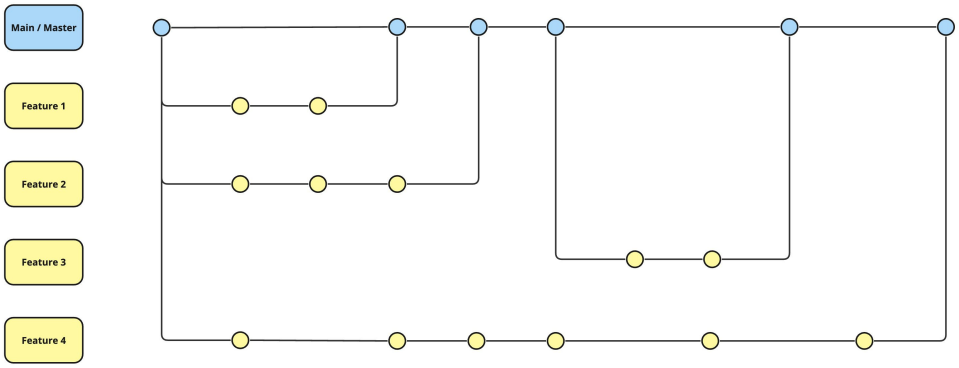
Obr. 1.4 Git Flow

1.2.2 GitHub flow

GitHub Flow je zjednodušený workflow, ktorý efektívne odstraňuje komplexitu spojenú s Git Flow a je vhodný pre menšie tímy a aplikácie. GitHub Flow umožňuje vďaka svojej jednoduchosti časté a rýchle nasadzovanie nových verzií, nerieši ale problém viacerých prostredí, na ktoré chceme software nasadzovať. [14, 15] Ako znázorňujem na obrázku 1.5, v rámci GitHub flow exisutjú v projekte nasledujúce typy vetví:

Hlavná vetva (*main* alebo *master*): Hlavná vetva, ktorá je základom pre všetky ďalšie vetvy, obsahuje otestovaný a skontrolovaný kód a slúži na nasadzovanie do produkčného prostredia.

***Feature* vetvy:** Nové funkcionality sú vyvíjané v samostatných vetvách, ktoré vznikajú z hlavnej vetvy *main* a po dokončení a následnom schválení sú zlúčené späť. Každá takáto vetva by mala byť zameraná na jednu konkrétnu funkcionalitu alebo opravu jednej konkrétnej chyby.



Obr. 1.5 GitHub Flow

1.2.3 GitLab flow

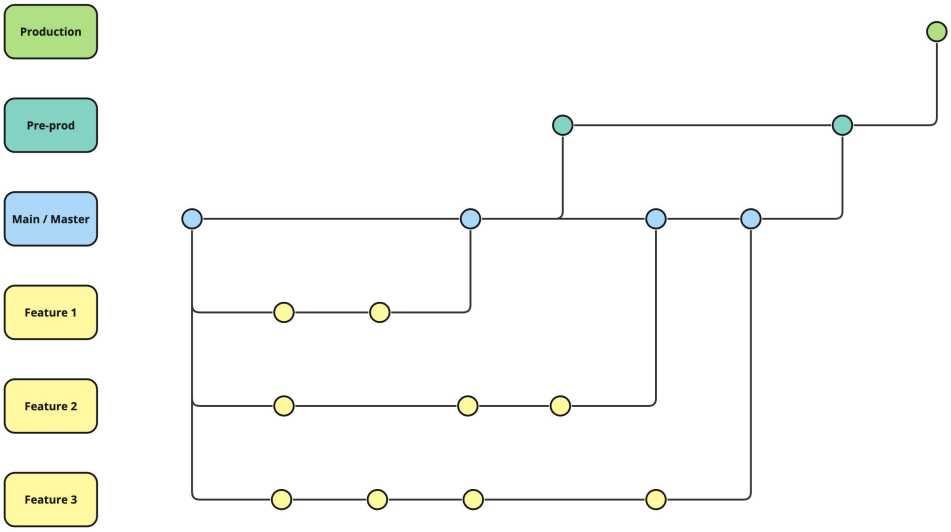
Hoci je GitLab Flow podobný GitHub flow, snaží sa riešiť niektoré jeho nedostatky. Hlavným rozdielom je pridanie riešenia pre správu nasadzovacích procesov na viacero prostredí. Pre každé prostredie (napríklad produkčné, predprodukčné, testovacie) totiž existuje stabilná permanentná vetva. [14, 16, 17] V GitLab flow modely teda existujú nasledujúce typy vetví:

Hlavná vetva (*main* alebo *master*): Podobne ako v GitHub Flow, aj tu je hlavná vetva základom, do ktorej smerujú všetky schválené zmeny a nové funkcionality a ktorá slúži ako základ pre nasadenie do daného prostredia.

***Feature* vetvy:** Nové funkcionality sú vyvíjané v samostatných vetvách, ktoré vznikajú z hlavnej vetvy *main*.

***Release* alebo *environment* vetvy:** Pre každé prostredie, na ktoré chceme software nasadzovať existuje stabilná vetva.

Ako je vidieť na obrázku 1.6, ktorý sumarizuje model GitLab Flow vrátane vzťahov medzi jednotlivými vetvami, pravdepodobne jediný rozdiel medzi GitHub flow a GitLab flow je zavedenie vetví pre jednotlivé prostredia.



Obr. 1.6 GitLab Flow

Kapitola 2

Architektúra webových aplikácií

V druhej kapitole predstavím čitateľovi rôzne architektúry webových aplikácií. Na úvod predstavím základný model klient-server, popíšem, ako funguje protokol HTTP a architektonický štýl REST. Následne vysvetlím viacvrstvovú architektúru aplikácií, porovnám architektonické vzory MVC a MVT a predstavím aj tzv. čistú architektúru. Na koniec porovnám jednostránkové a viac-stránkové webové aplikácie.

Návrh softwarovej architektúry je vo všeobecnosti o návrhu organizácie celkovej štruktúry systému. Ide teda o pochopenie štruktúry a zodpovedností jednotlivých častí systému a o tom, ako budú tieto časti systému interagovať. Je to prvá fáza procesu návrhu softwaru, keďže to vieme považovať za kľúčové prepojenie medzi zberom a analýzou požiadaviek a návrhom aplikácie. V agilných metodikách sa všeobecne akceptuje, že prvé iterácie agilného vývojového procesu by sa mali zamerať na návrh celkovej architektúry systému. Iteratívny návrh architektúry zvyčajne nie je úspešný, pretože refaktorovanie architektúry systému je nákladné. Dôvodom je to, že pri zmene architektúry systému musíme upraviť väčšinu systémových komponent. Návrh architektúry sa čiastočne týka splnenia nefunkčných požiadaviek systému, z tohto dôvodu by sa mal návrh architektúry odrážať aj od nefunkčných požiadaviek systému. [1, kap. 6]

Užívatelia a zákazníci si nevšimnú výber architektúru systému okamžite, pretože zvyčajne nemení nič na funkcionalitách a správaní softwaru. Zlá architektúra však významne prispieva nielen k nesplneniu niektorých nefunkčných požiadaviek systému, ale aj k nárastu prvkov softwaru, ktoré bránia vývojárom pochopiť daný software. Software, ktorý obsahuje veľa takýchto prvkov

sa oveľa ťažšie upravuje, čo vedie k tomu, že nové funkcionality dodávame pomalšie a s väčším počtom chýb. [18]

Pri návrhu webovej aplikácie je veľmi dôležité si uvedomiť, že architektúra nie je definovaná jedným súborom pravidiel a nasledujúce architektúry, ktoré nižšie analyzujem, nepredstavujú úplnú kategorizáciu. Namiesto toho architektúra webových aplikácií zahŕňa celý rad konceptov, vzorov a technológií, ktoré sa síce odlišujú, ale navzájom sa dopĺňajú a reprezentujú rôzne úrovne pohľadu na riešenie rôznych výziev pri vývoji webových aplikácií, od interakcií s užívateľmi až po efektívne spracovanie dát a škálovateľnosť služieb.

2.1 Klient-server

Základným princípom fungovania webových aplikácií je klient-server model, ktorý vymedzuje úlohy medzi serverom, ktorý má úlohu poskytovateľa rôznych zdrojov alebo služieb a klientom, ktorý o tieto zdroje a služby žiada. V kontexte webových aplikácií sú príkladom tohto modelu webové servery, ktoré poskytujú webové stránky alebo dáta a webové prehliadače (alebo iné klientske aplikácie), ktoré požadujú prístup k týmto stránkam alebo dátam. Klientské aplikácie komunikujú s webovým serverom pomocou HTTP protokolu.

2.2 HTTP protokol

Protokol HTTP je založený na odosielaní správ medzi klientom a serverom. Definuje dva typy správ – žiadosti[1] a odpovede[2]. Prehliadač odošle HTTP žiadosť na webový server. Servery čakajú na klientské HTTP žiadosti, ktoré po príchode spracujú a po spracovaní klientovi odošlú HTTP odpoveď. Pred vysvetlením, ako vyzerá HTTP žiadosť a HTTP odpoveď by som čitateľovi najprv rád predstavil HTTP metódy, stavové kódy a hlavičky. [19]

2.2.1 HTTP metódy

Cieľ HTTP žiadosti sa nazýva zdroj[3], čo je veľmi abstraktný pojem, ktorý nie je bližšie definovaný a môže to byť v podstate čokoľvek (napríklad HTML dokument alebo obrázok). Každý zdroj je identifikovaný pomocou URI (Uniform Resource Identifier, v preklade unikátny identifikátor zdroja). Protokol HTTP definuje súbor metód na označenie požadovanej akcie, ktorá sa má danou HTTP žiadosťou pre daný zdroj vykonať.

[1] Anglicky „request“
[2] Anglicky „response“
[3] Anglicky „resource“

Pri vývoji webových aplikácií sa podľa môjho názoru a skúseností stretne vývojár hlavne s nasledujúcimi šiestimi HTTP metódami: [20]

GET - používa sa na získanie dát z určeného zdroja

HEAD - žiada o podobnú odpoveď ako GET, ale bez tela odpovede, žiada teda len o hlavičky

POST - používa sa na odoslanie dát na server

PUT - nahradí všetky aktuálne reprezentácie cieľového zdroja dátami žiadosti, hlavným rozdielom medzi POST a PUT metódami je nižšie vysvetlená idempotentnosť

PATCH - používa sa na čiastočnú zmenu zdroja, umožňuje aktualizovať časť zdroja bez nutnosti nahradzovať celý zdroj

DELETE - odstráni špecifikovaný zdroj

Vývojárovi webových aplikácií sa tiež môže hodiť znalosť a pochopenie niektorých vlastností HTTP metód. HTTP metódy môžu byť bezpečné, idempotentné a cachovateľné.

Metóda je **bezpečná**, ak nemení stav servera a iba poskytuje dáta na čítanie. Medzi bezpečné metódy patrí GET, HEAD a OPTIONS. [21]

Metóda je **idempotentná** práve vtedy, ak zamýšľaný efekt jednej žiadosti je rovnaký ako efekt viacerých rovnakých žiadostí. Nutno podotknúť, že pri definícii idempotentnosti sa berie do úvahy len stav servera, napríklad odpovede servera sa môžu líšiť alebo každá žiadosť môže mať nejaké iné unikátne vedľajšie účinky, ktoré neovplyvňujú stav servera. Okrem všetkých bezpečných metód sú idempotentné aj metódy PUT a DELETE. Za zmienku ešte stojí, že server negarantuje idempotenciu metód a niektoré aplikácie môžu idempotenciu porušovať, takže chápanie idempotentnosti je dôležité napríklad pri návrhu API. [22]

HTTP odpoveď serveru označujeme ako cachovateľnú, keď si vieme túto odpoveď uložiť a neskôr použiť namiesto posielania novej žiadosti. Nie všetky odpovede ale sú cachovateľné, pretože existuje viacero obmedzení na to, aby bola odpoveď cachovateľná. Jedným z týchto obmedzení je, že musí byť metóda žiadosti **cahovateľná**. Medzi cachovateľné HTTP metódy patrí GET a HEAD, pretože sú bezpečné a nemenia teda stav servera. Ďalšie metódy sa vo všeobecnosti nepovažujú za cachovateľné, ale je dôležité poznamenať, že možnosť cachovania odpovedí na žiadosti metód, ktoré nie sú cachovateľné, sa dá kontrolovať pomocou HTTP hlavičiek. [23]

2.2.2 HTTP stavové kódy

Stavové kódy HTTP sú číselné kódy HTTP odpovede, ktoré označujú, či bola konkrétna HTTP žiadosť klienta úspešne dokončená. Stavové kódy HTTP delíme do nasledujúcich piatich kategórií, z ktorých vyberiem niektoré konkrétne kódy, ktoré aj stručne vysvetlím: [24]

1. **Informatívne odpovede (1xx)**
2. **Úspešné odpovede (2xx)**
   - **200 OK** – žiadosť bola úspešná, takže napríklad pre GET bol zdroj úspešne nájdený, načítaný a odoslaný v tele odpovede
   - **201 Created** – žiadosť bola úspešná a výsledkom bolo vytvorenie nového zdroja, 201 je teda typická odpoveď pre POST žiadosti
   - **204 No Content** – žiadosť bola úspešná, server ale nevracia žiadny obsah, avšak môže vrátiť hlavičky
3. **Presmerovania (3xx)**
4. **Chyby klienta (4xx)**
   - **400 Bad Request** – server nevie spracovať žiadosť kvôli chybe klienta (napríklad nesprávna syntax žiadosti)
   - **401 Unauthorized** – žiadosť klienta nebola dokončená, pretože chýbajú platné autentifikačné údaje pre získanie požadovaného zdroja
   - **403 Forbidden** – server odmieta poskytnúť požadovaný zdroj, keďže klient nemá prístupové práva k danému zdroju, na rozdiel od kódu 401 je ale identita klienta serveru známa
   - **404 Not Found** – server nevie nájsť požadovaný zdroj, čo znamená, že server buď nevie rozpoznať URL adresu žiadosti alebo v kontexte API to môže tiež znamenať, že aj keď je API endpoint validný, tak daný zdroj neexistuje
5. **Chyby servera (5xx)**
   - **500 Internal Server Error** – Server narazil na neočakávanú situáciu, ktorú nevie vyriešiť a preto nevie na žiadosť úspešne odpovedať

2.2.3 HTTP hlavičky

Hlavičky HTTP umožňujú posielať dodatočné informácie spolu s HTTP žiadosťou alebo odpoveďou. Hlavička HTTP sa skladá z názvu, za ktorým nasleduje dvojbodka a potom hodnota danej hlavičky. Hlavičky sa môžu použiť

napríklad na poskytnutie autentifikačných údajov, správu cachovania alebo poskytnutie kontextu k žiadosti či odpovedi. [25]

2.2.4 HTTP žiadosť

HTTP žiadosť je správa odosielaná klientom na server. Každá HTTP žiadosť sa skladá z nasledujúcich komponentov: [19]

- HTTP metóda, ktorá definuje, aká akcia sa so zdrojom vykoná
- URL zdroja
- Verzia protokolu HTTP (buď HTTP/1.1 alebo HTTP/2)
- Nepovinné hlavičky, ktoré poskytujú dodatočné informácie pre server
- Nepovinné telo, ktoré obsahuje zdroj, ktorý chce klient poslať na server (telo obsahujú často hlavne POST žiadosti)

2.2.5 HTTP odpoveď

HTTP odpoveď je správa odoslaná serverom klientovi ako odpoveď na HTTP žiadosť. Informuje klienta o výsledku žiadosti a môže klientovi vrátiť údaje zo servera. Každá odpoveď HTTP sa skladá z nasledujúcich komponentov: [19]

- Verzia protokolu HTTP (buď HTTP/1.1 alebo HTTP/2)
- Stavový kód označujúci, či bola žiadosť úspešná alebo nie a prípadne prečo nie (napríklad 200)
- Stavová správa bližšie vysvetľujúca výsledok (napríklad „OK“)
- Hlavičky (tak ako pri žiadosti)
- Nepovinné telo, ktoré obsahuje získaný zdroj

2.3 REST

Po predstavení klient-server modelu a HTTP protokolu sa teraz zameriam na kľúčové aspekty architektúry moderných webových aplikácií – REST a REST API – ktoré zvyčajne využívajú práve HTTP na komunikáciu medzi klientmi a servermi štruktúrovanejším spôsobom.

Pred samotným predstavením REST a REST API, by som ale v krátkosti rýchlo vysvetlil, čo je vlastne API (Application Programming Interface). Ide

o rozhranie aplikácie, ktoré umožňuje dvom alebo viacerým komponentom softwaru spolu komunikovať a predávať si dáta pomocou súboru pravidiel a protokolov.

REST (REpresentational State Transfer) je architektonický štýl pre tvorbu API, ktorý predstavil Roy Fielding v roku 2000 vo svojej dizertačnej práci. Nutno podotknúť, že sa ale nejedná o protokol alebo štandard, ale o architektonický štýl, ktorý vývojári môžu implementovať rôzne. REST dáva vývojárom určité princípy a obmedzenia, ktorých sa musia držať, aby rozhranie ich služby bolo označované ako RESTful[4], ale nevynucuje žiadne pravidlá týkajúce sa implementácie. Webové API, ktoré je v súlade s REST štýlom a jeho pravidlami označujeme ako REST API. REST definuje nasledujúcich šesť pravidiel: [26, 27, 28]

Jednotné rozhranie: REST definuje jednotné rozhranie pre interakcie medzi jednotlivými komponentmi. Napríklad pri použití HTTP, REST API využíva štandardné metódy HTTP ako GET, POST, PUT alebo DELETE a URI identifikátory na identifikáciu zdrojov.

Klient-server: Toto obmedzenie v podstate znamená, že klientske a serverové aplikácie sa musia vyvíjať samostatne bez akejkoľvek vzájomnej závislosti, čo vynucuje oddelenie zodpovedností. Výhodou je, že server aj klienti sa môžu vymieňať a vyvíjať aj nezávisle na sebe, pokiaľ sa medzi nimi nezmení stanovené rozhranie.

Bezstavovosť: Bezstavovosť znamená, že každá žiadosť prebieha úplne izolovane. Klient je zodpovedný za poskytnutie všetkých potrebných informácií v každej žiadosti, ako sú autentifikačné tokeny alebo iné kontextové údaje, aby server danú žiadosť mohol splniť. Server sa nikdy nespolieha na informácie z predchádzajúcich žiadostí klienta a nikdy neukladá žiadne *session* dáta špecifické pre klienta. *Session* dáta aplikácie sa preto uchovávajú výlučne na klientovi. Ak je nejaká takáto informácia dôležitá, klient ju odošle ako súčasť aktuálnej žiadosti.

Cachovateľnosť Cachovateľnosť vyžaduje, aby bola odpoveď implicitne alebo explicitne označená ako cachovateľná alebo necachovateľná. Ak je odpoveď cachovateľná, klientská aplikácia môže na vopred určené obdobie opätovne použiť dáta odpovede pre ekvivalentné žiadosti.

Vrstvený systém: V REST rozhraniach môžu žiadosti a odpovede prechádzať viacerými vrstvami, preto pri návrhu REST API nemôžme predpokladať, že klient a server komunikujú priamo. REST teda umožňuje, aby sa architektúra systému skladala z viacerých vrstiev, pričom každá vrstva vidí iba tú vrstvu, s ktorou priamo komunikuje. REST rozhrania musia byť teda navrhnuté tak, aby klient ani server nevedeli určiť, či komunikujú

[4] Alebo v preklade RESTové

s koncovou aplikáciou alebo s nejakým iným sprostredkovateľom.

Kód na vyžiadanie: Toto nepovinné pravidlo hovorí, že v prípade potreby vie server odpovedať aj spustiteľným kódom namiesto statických reprezentácií zdrojov vo formáte JSON.

2.4 Viacvrstvová architektúra

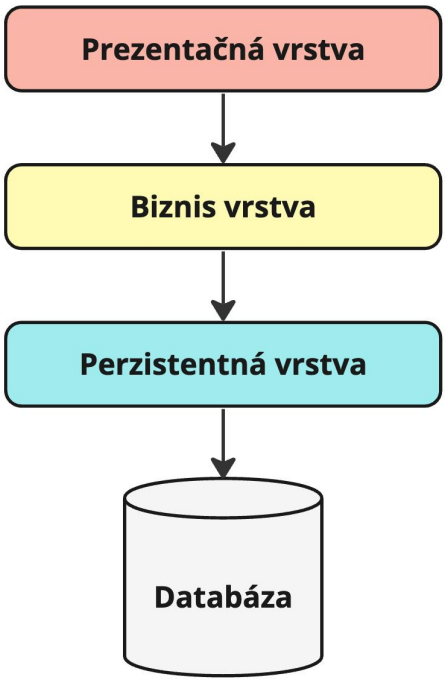
Jedným zo spôsobov, ako zvládnuť a vyriešiť rastúcu zložitosť aplikácie je rozdeliť aplikáciu do vrstiev podľa zodpovedností. Viacvrstvová architektúra je jedna z najčastejšie sa vyskytujúcich architektúr moderných softwarových aplikácií, pretože je v súlade so štruktúrou tradičných vývojových tímov. Väčšina tímov je totižto organizovaná podľa technických oblastí (napríklad frontend tím, backend tím, tím databázových špecialistov alebo DevOps tím). Komponenty v rámci viacvrstvovej architektúry sú usporiadané do horizontálnych vrstiev, z ktorých každá plní špecifickú úlohu v rámci aplikácie.

Ako ukazuje obrázok 2.1, väčšina viacvrstvových aplikácií sa skladá z nasledujúcich štyroch základných vrstiev[5].

1. **Prezentačná vrstva** - stará sa o užívateľské rozhranie a interakciu s užívateľom
2. **Biznis vrstva** - zapuzdruje biznis logiku aplikácie
3. **Perzistentná vrstva** - abstrahuje ukladanie a získavanie dát z databáze
4. **Databáza** - zodpovedná za ukladanie a spravovanie dát, môže sa jednať buď o relačné (napríklad PostgreSQL alebo Oracle) alebo nerelačné NoSQL databázy (napríklad MongoDB)

Jednou z kľúčových vlastností viacvrstvovej architektúry je oddelenie zodpovedností medzi komponentmi. Komponenty v rámci konkrétnej vrstvy sa zaoberajú len logikou, ktorá sa týka tejto vrstvy. Napríklad prezentačná vrstva sa zameriava na zobrazovanie informácií bez potreby podrobného vyhľadávania dát. Biznis vrstva spracováva logiku, ako sú výpočty, agregácia dát alebo odosielanie e-mailov, bez toho, aby sa starala o zdroj dát alebo formátovanie zobrazenia, pričom sa pri prístupe k dátam spolieha na perzistentnú vrstvu. [29, 30, 31]

[5] Niektoré zdroje uvádzajú tri základne vrstvy a databázu nepovažujú za vrstvu aplikácie, prípadne spájajú databázu a perzistentnú vrstvu, čo potom nazývajú napríklad dátová vrstva



Obr. 2.1 Viacvrstvová architektúra

2.5 Architektonické vzory

Na základe nadobudnutých poznatkov o viacvrstvovej architektúre, ktorá organizuje vývoj aplikácií do samostatných vrstiev, sa teraz zameriam na architektonické vzory ako MVC a MVT. Kľúčovým cieľom týchto vzorov je oddelenie zodpovedností, slabá väzba[6] a vysoká súdržnosť[7] jednotlivých komponentov. Dosahujú to rozdelením aplikácie na niekoľko nezávislých modulov, z ktorých každý má svoju špecifickú zodpovednosť. Toto rozdelenie nielen zjednodušuje vývoj a testovanie, ale tiež zlepšuje opätovnú použiteľnosť kódu a jeho údržbu.

V tejto sekcii primárne predstavím architektonický vzor MVC. Okrem MVC existuje viacero jeho variácií (napríklad MVP alebo MVVM), ktorých cieľ je rovnaký, ale majú určité odlišnosti. Pre účely tejto práce ale nie je potrebné tieto architektonické vzory bližšie predstavovať. Namiesto toho čitateľovi predstavím architektonický vzor MVT, na ktorom, ako uvádzam nižšie v časti 3.1, je postavený súčasný systém SOS.

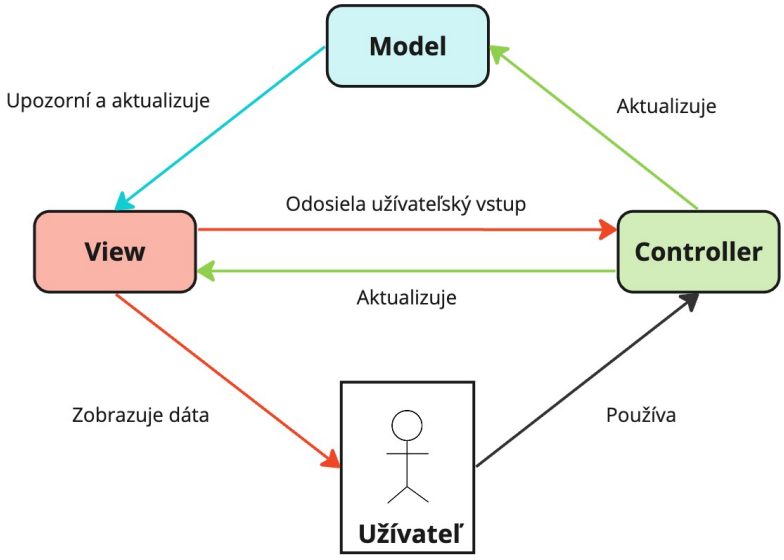
[6] Anglicky známe pod pojmom "loose coupling", jedná sa o princíp, ktorý hovorí, že by triedy, respektíve moduly mali mať čo najmenej závislostí

[7] Anglicky známe pod pojmom "high cohesion", jedná sa o princíp, ktorý hovorí, že by triedy, respektíve moduly mali riešiť len súvisiace funkcionality

2.5.1 MVC - Model-View-Controller

MVC rozdeľuje aplikáciu do nasledujúcich troch modulov, ako je videiť na obrázku 2.2: [32]

1. **Model** - dáta a biznis logika aplikácie
2. **View** - užívateľské rozhranie, ktoré zobrazenie dáta a logiku užívateľovi
3. **Controller** - rozhoduje o zmenách v Modely, prípadne View na základe užívateľského vstupu, ktorý spracováva



Obr. 2.2 MVC (Model-View-Controller)

Model reprezentuje stav, štruktúru a správanie dát, ktoré užívateľ vidí a s ktorými manipuluje. Dôležité ale je, že Model neobsahuje žiadnu závislosť na View a Controller, teda na tom ako budú tieto dáta zobrazované užívateľovi a ako sa bude spracovávať užívateľský vstup. View a Controller sú ale úzko prepojené, keďže sa spoločne starajú o to, aby vedel užívateľ interagovať s Modelom.

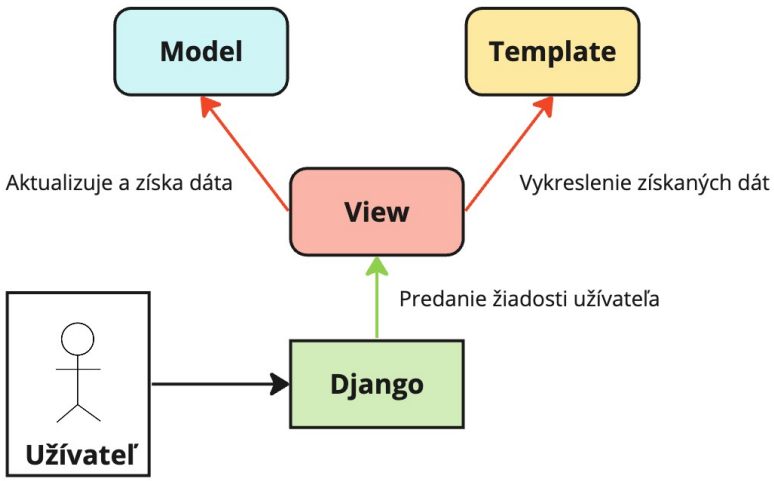
Existujú dve hlavné implementácie MVC s drobnou odlišnosťou týkajúcou sa aktualizácie View. V prvej implementácii je za aktualizáciu View zodpovedný Controller. Po spracovaní vstupu teda Controller upraví Model a na základe týchto zmien upraví aj View. V druhej implementácii Controller spracuje vstup od užívateľa, upraví Model a Model upozorní View o vykonaných zmenách. Je ale nutné podotknúť, že toto nepridáva závislosť Modelu na View, pretože sa tu využíva návrhový vzor Observer. [33] Observer je vo všeobec-

nosti návrhový vzor, ktorý umožňuje definovať mechanizmus, vďaka ktorému vedia objekty pozorovať iný objekt a na základe toho sú upozornené o rôznych udalostiach a zmenách, ktoré sa s týmto pozorovaným objektom stanú. [34] V tomto prípade View pozoruje Model, ktorý View upozorní na všetky zmeny, ktoré sa s Modelom uskutočnili po spracovaní vstupu Controllerom. [33]

2.5.2 MVT - Model-View-Template

Cieľ MVT je rovnaký ako cieľ MVC a to oddelenie zodpovedností. MVT ale rozdeľuje aplikácie do iných troch modulov, ktoré majú iné zodpovednosti ako moduly vzoru MVC. Vzor MVT implementuje hlavne webový framework Django, ktorý sa používa na vývoj webových aplikácií. Komponenty Django aplikácie postavenej na architektúre MVP sú nasledovné [35]:

1. **Model** - abstraktná vrstva na štruktúrovanie a manipuláciu s dátami
2. **View** - zapuzdruje všetku logiku aplikácie, zodpovedá za spracovanie žiadosti užívateľa a vrátenie odpovede
3. **Template** - vykresľovanie informácii užívateľovi



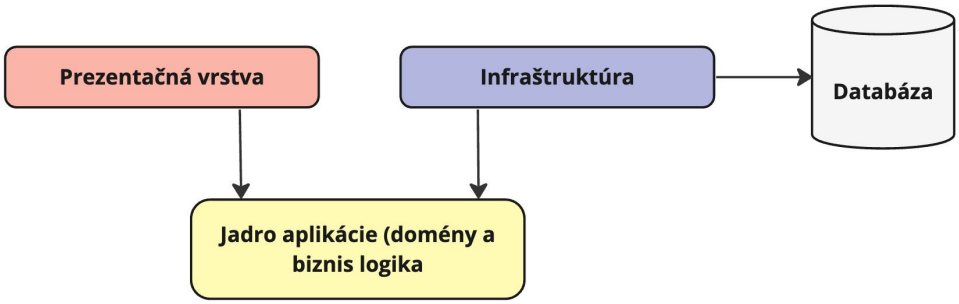
Obr. 2.3 MVT (Model-View-Template)

Ako ukazuje diagram 2.3, Django – konkrétne jeho smerovač požiadaviek – presmeruje žiadosť užívateľa na základe URL adresy do správneho View, ktorý túto žiadosť spracuje a požiada Model o dáta. Model získa dáta z databázy a vráti ich View. View po získaní dát vykreslí užívateľovi príslušný Template spolu s načítanými dátami.

2.6 Čistá architektúra

Pri použití tradičnej viacvrstvovej architektúry, ktorú som definoval v časti 2.4, existujú v aplikácii tranzitívne závislosti, keďže na databáze je závislá perzistentná vrstva, na ktorej je závislá biznis vrstva, na ktorej je závislá prezentačná vrstva, takže každá komponenta aplikácie závisí na databáze. Cieľom čistej architektúry[8] je doménovo orientovaný prístup k organizácii aplikácie a jej závislostí. Ako je vidieť na obrázku 2.4, aplikácie postavené na čistej architektúre sú typicky delené na nasledujúce vrstvy:

1. **Jadro aplikácie** - obsahuje entity, biznis logiku a rozhrania, napríklad pre prístup k dátam, ktoré sú potom implementované vo vrstve infraštruktúry
2. **Infraštruktúra** - obsahuje implementáciu rozhraní hlavne pre prístup k dátam
3. **Prezentačná vrstva** - stará sa o užívateľské rozhranie, patria sem napríklad Controller triedy, ktoré definujú a realizujú API endpointy



Obr. 2.4 Čistá architektúra

Kľúčovou vlastnosťou je, že závislosť na infraštruktúre, napríklad na databáze, sa minimalizuje a sústredí sa na domény a biznis logiku. Namiesto toho, aby biznis logika závisela na prístupu k dátam alebo iných problémov týkajúcich sa infraštruktúry, je táto závislosť obrátená, keďže infraštruktúra a implementačné detaily závisia od jadra aplikácie. Táto funkčnosť sa dosahuje definovaním rozhraní v jadre aplikácie, ktoré sú potom implementované triedami definovanými vo vrstve infraštruktúry. [36]

2.7 SPA a MPA

Viac stránková aplikácia (MPA z anglického „Multi Page Application“) je webová aplikácia s viacerými HTML stránkami so statickým obsahom a odkazmi

[8] Anglicky „Clean architecture“

na iné stránky. Pri každej HTTP žiadosti, ktorú webový prehliadač odošle na server je ako odpoveď klientovi vrátená celá novo vygenerovaná HTML stránka s požadovaným obsahom. [37]

Alternatívou tradičných MPA sú jednostránkové aplikácie (SPA z anglického „Single Page Application“). SPA je webová aplikácia, ktorá obsahuje, ako napovedá názov, len jednu HTML stránku. Pre SPA je charakteristické, že sa časť biznis logiky presúva na klienta. Keď je potrebné zobraziť nový obsah, klient pomocou AJAX (Asynchronous JavaScript and XML) odošle žiadosť na server, ktorý klientovi namiesto celej HTML stránky vráti len potrebné dáta napríklad formou JSON (JavaScript Object Notation) prostredníctvom REST API. Klient tieto dáta zobrazí na tu istú stránku, ktorú načítal pri prvotnom spustení aplikácie. [37, 38]

Kapitola 3

Analýza súčasného systému

V tejto kapitole čitateľa zoznámim s existujúcim systémom SOS. Predstavím štruktúru aplikácie a na akých technológiách a architektúre je SOS postavený. Ďalej popíšem históriu a časovú os vývoja systému, predstavím pôvodnú myšlienku a dôvody pre vznik systému SOS. Následne čitateľovi vysvetlím, kedy som sa do rozvoja systému SOS zapojil a ako som sa na rozvoji podieľal. Na záver uvediem zistené nedostatky a zhodnotím jeho aktuálny stav a budúcnosť a uvediem dôvody, ktoré nás s tímom viedli k rozhodnutiu vyvinúť nový systém vychádzajúci z toho súčasného.

SOS je webová aplikácia pre podporu výuky, ktorej cieľom je zjednodušenie správy a hodnotenia tímových a individuálnych projektov v školskom prostredí. Aplikácia umožňuje vytváranie zadaní projektov, formovanie tímov na ich realizáciu, správu tímov a projektov, rozdelenie projektov na iterácie, odovzdávanie práce v rámci týchto iterácií a hodnotenie odovzdanej práce vyučujúcimi. Dnes je SOS aktívne využívaný na tímové semestrálne práce v predmetoch NI-NUR (Návrh uživatelského rozhraní), BI-SWI (Softwarové inženýrství), BI-SP1 (Softwarový týmový projekt 1) a BI-SP2 (Softwarový týmový projekt 2) na FIT ČVUT a na ročníkové práce na GJGJ.

3.1 Použité technológie

Systém SOS je napísaný v programovacom jazyku Python[1] s využitím frameworku Django[2]. Pre ukladanie dát používa databázový stroj PostgreSQL[3] a pre

[1] `https://www.python.org/`
[2] `https://www.djangoproject.com/`
[3] `https://www.postgresql.org/`

UI (užívateľské rozhranie, z anglického „User Interface“) šablónovací systém v rámci frameworku Django a CSS knižnicu Bootstrap[4]. V tejto časti čitateľovi predstavím tieto technológie a následne stručne vysvetlím aj architektúru systému.

3.1.1 Python

Python je interpretovaný, objektovo orientovaný, vysoko-úrovňový univerzálny programovací jazyk s dynamickou sémantikou. Fakt, že ide o dynamicky typovaný jazyk znamená, že sa programátor nemusí pri vývoji zaoberať dátovými typmi objektov a premenných. Toto určite uľahčuje a urýchľuje vývoj softwaru, no na základe mojich vlastných skúseností s dynamicky typovanými jazykmi ako sú Python a Groovy[5], si dovolím tvrdiť, že používanie takýchto jazykov značne znižuje prehľadnosť kódu hlavne väčších aplikácií v porovnaní so staticky typovanými jazykmi ako je napríklad Java[6] alebo C#[7]. Jeho vysoko-úrovňové dátové štruktúry v kombinácii so spomínaným dynamickým typovaním ho ale robia veľmi atraktívnym na rýchly vývoj aplikácií. Okrem toho nájde uplatnenie ako skriptovací jazyk alebo jazyk, ktorým vývojári často spájajú existujúce komponenty napísané v iných jazykoch. [39]

3.1.2 Django

Pri vývoji webových aplikácií je v praxi veľmi časté využiť nejaký framework, ktorý je určený na podporu a urýchlenie vývoja. Webové frameworky nám poskytujú spôsob organizácie projektu a často obsahujú knižnice pre prístup k databázam, autentifikáciu užívateľov, spracovanie HTTP žiadostí a ďalšie funkcionality, vďaka čomu nemusíme písať tieto komponenty od začiatku. Pôvodny autor súčasného systému Ing. Tomáš Pavlůsek zvolil pre implementáciu tohto systému framework Django. Django je webový framework založený na programovacom jazyku Python, ktorý implemetuje architektonický vzor MVT, ktorý som bližšie predstavil a porovnal so vzorom MVC vyššie v časti 2.5.2. Ako uvádza autor vo svojej bakalárskej práci z roku 2021, dôvodom pre výber tejto technológie boli primárne jeho skúsenosti, pretože framework Django v tej dobe používal vyše roka na komerčnom projekte vo svojom zamestnaní a preto nad inou technológiou ani nerozmýšľal. [40]

[4] https://getbootstrap.com/
[5] https://groovy-lang.org/
[6] https://www.java.com/en/
[7] https://dotnet.microsoft.com/en-us/languages/csharp

3.1.3 PostgreSQL

PostgreSQL je open-source objektovo-relačný databázový stroj, ktorý využíva a rozširuje jazyk SQL. [41] Podporuje veľkú časť štandardu SQL a ponúka mnoho moderných funkcií, vrátane komplexných dotazov, cudzích kľúčov, triggerov, aktualizovateľných pohľadov a transakčnej integrity. Okrem toho môže užívateľ PostgreSQL rozšíriť mnohými spôsobmi, napríklad pridaním nových dátových typov, funkcií, operátorov, agregačných funkcií a indexových metód. [42]

3.1.4 Bootstrap

Autor pre frontend aplikácie využil šablónovací systém, ktorý ponúka framework Django v kombinácii s Bootstrap knižnicou a šablónou AdminLTE. Bootstrap je CSS knižnica, ktorá slúži pre vývoj responzivných webových aplikácií pomocou tzv. grid systému. Bootstrap obsahuje viacero komponentov užívateľského rozhrania, formulárov alebo ikon, ktoré môžu vývojári vo svojich aplikáciach použiť. [43] Ako tvrdí autor súčasného systému vo svojej bakalárskej práci, zvažoval aj použitie nejakého populárneho frontend frameworku postaveného na jazyku JavaScript[8], ako napríklad Vue.js[9]. Hlavným dôvodom výberu Bootstrap knižnice v kombinácii s Django šablónovacím systémom bola ale autorova malá skúsenosť s týmito JavaScriptovými technológiami. [40]

3.2 Štruktúra aplikácie

Pre webové aplikácie vo frameworku Django je typické členiť zdrojový kód do tzv. aplikácií vo forme Python modulov. Každá aplikácia je zodpovedná za určitú logickú časť systému. Vo všeobecnosti sa jedná o tzv. Domain Partitioning, teda o architektúru s doménovým členením. Komponenty v architektúrach s doménovým členením sú organizované podľa doménových oblastí, všetok zdrojový kód vrátane dátových modelov, aplikačnej logiky a testov je teda združený podľa doménových oblastí, nie podľa technického použitia ako pri viacvrstvovej architektúre, [29] ktorú definujem v časti 2.4.

V aplikácii SOS existujú nasledovné aplikácie:

Assignments: Rieši tvorbu a správu zadaní projektov.

Attachments: Slúži na prácu s prílohami, ktoré užívatelia nahrávajú k zadaniam a riešeniam.

[8] `https://developer.mozilla.org/en-US/docs/Web/JavaScript`
[9] `https://vuejs.org/`

Checkpoints: Rieši kontrolné body a ich termíny odovzdaní.

Courses: Obsahuje modely a logiku reprezentujúcu semestre, predmety a paralelky.

Export: Slúži na export dát do externých systémov (napríklad KOS).

Homepage: Rieši zobrazenie domovskej stránky.

Notifications: Pracuje s upozorneniami na udalosti v systéme.

OAuth: Rieši autentizáciu užívateľov.

Submissions: Realizuje odovzdávanie riešení a ich následne hodnotenie.

Teams: Rieši správu tímov, ktoré pracujú na nejakom zadaní.

Users: Slúži na správu užívateľov systému.

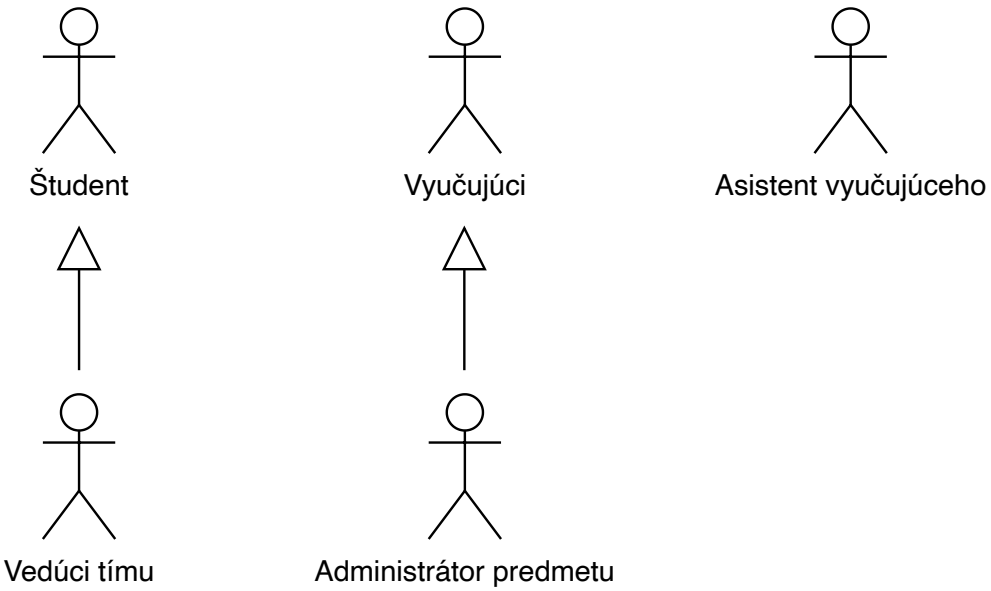
V praxi ale môžu byť tieto domény v prípade potreby ďalej organizované do technických vrstiev, [29] čo je aj prípadom aplikácie SOS. Keďže je systém SOS postavený na frameworku Django, ktorý implementuje architektonický vzor MVT, jednotlivé moduly systému v sebe obsahujú príslušné modely, formuláre, biznis logiku vo views, konfiguráciu smerovača žiadostí a šablóny pre generovanie odpovedí.

Jedná sa teda o MPA (Multi Page Application - viac stránková aplikácia) bez oddeleného backendu a frontendu v jednom projekte s viacerými Python modulmi. Hoci je toto monolitické riešenie s jedným projektom jednoduché, má určité nevýhody. S veľkosťou a zložitosťou projektu rastie aj počet súborov a priečinkov a časom nemusia byť jasne určené závislosti medzi jednotlivými triedami a modulmi, čo často vedie k neprehľadnému a ťažko udržateľnému a rozšíriteľnému kódu. [36]

3.3 Funkcionality systému

Pred samotnou analýzou funkcionalít systému SOS je potrebné predstaviť, aké role môžu užívatelia v systéme mať. Okrem toho je potrebné definovať, aké typy tímov v systéme existujú.

V systéme existuje celkom päť užívateľských rolí, je ale nutné spomenúť, že dané role neprislušia užívateľovi globálne, ale vzťahujú sa ku konkrétnemu predmetu v konkrétnom semestri. Môže sa napríklad stať, že jeden užívateľ môže byť v rovnakom semestri študentom nejakého magisterského predmetu a zároveň vyučujúcim v nejakom bakalárskom predmete. Aktéri systému sú vidieť na obrázku 3.1.



Obr. 3.1 Aktéri systému SOS

Rozlišujeme dva základe typy tímov – tímy spravované študentmi a tímy spravované vyučujúcim. Tím spravovaný študentmi sa ale po schválení vyučujúcim stáva tímom spravovaným týmto vyučujúcim.

Systém SOS podporuje nasledujúce funkcionality súvisiace s tvorbou a správou tímov a tímových projektov. Všetky funkcionality som analyzoval metódou prípadov použitia vrátane hlavných scenárov. Analyzoval som iba funkcionality týkajúce sa môjho zamerania práce – tímy a tímove projekty. Analýzu ďalších funkcionalít súčasného systému nájde čitateľ v bakalárskych prácach mojich kolegov z tímu – Jana Jamnického a Jana Mrázka.

UC1 – Zobrazenie zadaní

Aktéri: Vyučujúci
Popis: Vyučujúci si vie zobraziť zoznam všetkých svojich zadaní. Kliknutím na zadanie vidí stránku s detailami o danom zadaní vrátane počtu priradených tímov. Okrem detailov vie užívateľ na tejto stránke upravovať zadanie a priradiť zadanie novému tímu, ktorý najprv vytvorí.

Hlavný scenár:

1. Vyučujúci na domovskej stránke klikne na predmet.
2. Systém zobrazí detail predmetu.
3. Vyučujúci klikne na zoznam zadaní.

4. Systém zobrazí zoznam všetkých zadaní vytvorených daným vyučujúcim.

UC2 – Tvorba zadania

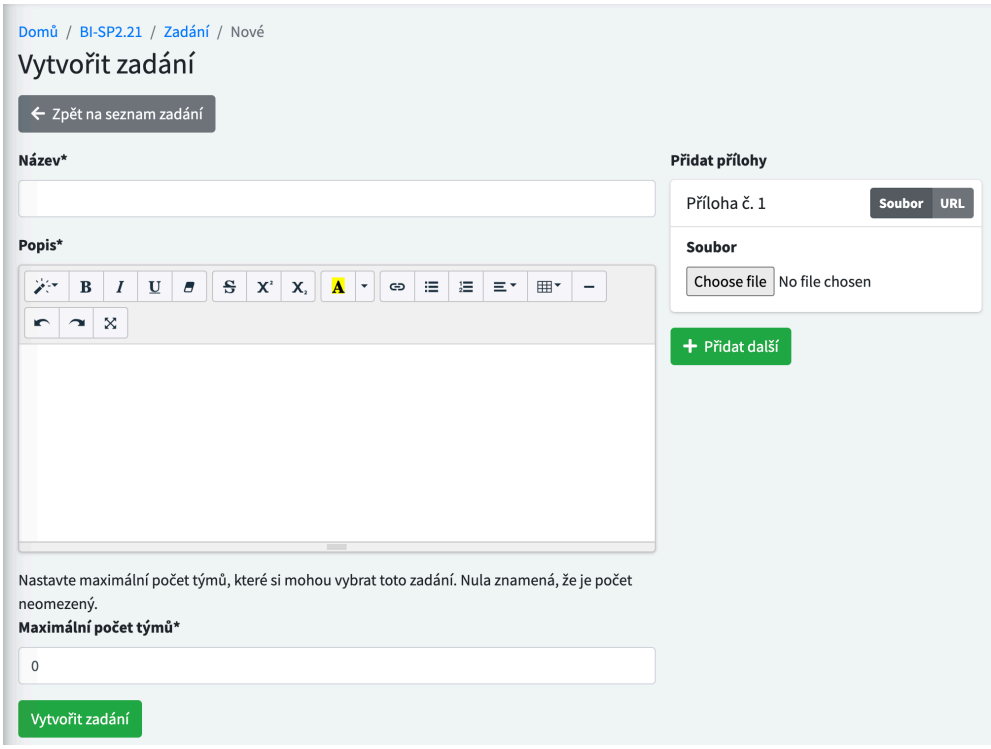
Aktéri: Vyučujúci, študent
Popis: Systém umožňuje vyučujúcim aj študentom vytvoriť zadanie projektu vrátane názvu, textového popisu a príloh v podobe súborov a webových odkazov, vyučujúci vie ešte určiť aj maximálny počet tímov, ktoré si toto zadanie môžu vybrať.

Hlavný scenár:

1. Vyučujúci aj študent vie vytvoriť zadanie v rámci tvorby tímu (UC6).

Alternatívny scenár:

1. Vyučujúci si zobrazí zoznam zadaní (UC1)
2. Vyučujúci klikne na tlačítko, ktoré ho presmeruje na formulár tvorby zadania, ktorý je vidieť na obrázku 3.2.
3. Po vyplnení a odoslaní formulára vytvorí systém zadanie s vyplnenými údajmi, ktoré je viditeľné študentom.



Obr. 3.2 Snímka obrazovky - tvorba zadania

UC3 – Úprava zadania

Aktéri: Vyučujúci, študent
Popis: Študenti aj vyučujúci vedia svoje zadania upravovať a to opäť vrátane názvu, textového popisu a príloh. Nutno podotknúť, že upravením existujúceho zadania sa toto zadanie automaticky aktualizuje všetkým priradeným tímom. Užívateľ si ale vie zvoliť, či chce aktualizovať zadanie vo všetkých predmetoch, v ktorých sa toto zadanie používa.

Hlavný scenár:

1. Vyučujúci aj študent pre úpravu zadania na domovskej stránke klikne na predmet.
2. Systém zobrazí detail predmetu.
3. Užívateľ vyberie projekt, ktorého zadanie chce upraviť.
4. Systém zobrazí detail projektu.
5. Užívateľ klikne na tlačítko úpravy zadania.
6. Systém zobrazí formulár úpravy zadania.
7. Po vyplnení a odoslaní formuláru je zadanie aktualizované.

Hlavný scenár:

1. Vyučujúci si zobrazí zoznam zadaní (UC1).
2. Vyučujúci vyberie zadanie, ktoré chce upravovať.
3. Systém zobrazí detail zadania.
4. Vyučujúci klikne na tlačítko úpravy zadania.
5. Systém zobrazí formulár úpravy zadania.
6. Po vyplnení a odoslaní formuláru je zadanie aktualizované.

UC4 – Zobrazenie a filtrovanie projektov

Aktéri: Vyučujúci, študent
Popis: Systém umožňuje vyučujúcim zobraziť zoznam všetkých ich projektov. Študentom umožňuje vidieť zoznam všetkých viditeľných projektov, ak nie sú v danom predmete a v danom semestri členom iného tímu, v tom prípade vidia iba svoj projekt. Systém umožňuje aj filtrovanie projektov podľa názvu projektu, vyučujúceho a člena tímu. Študenti vedia ďalej projekty filtrovať podľa toho, či je možné sa k projektu pripojiť, vyučujúci podľa toho, či má tím projektu dostatočný počet členov.

Hlavný scenár:

1. Užívateľ na domovskej stránke klikne na predmet.
2. Systém zobrazí detail predmetu, kde sa nachádza zoznam projektov.

UC5 – Zobrazenie detailu projektu

Aktéri: Vyučujúci, študent
Popis: Študenti aj vyučujúci si vedia zobraziť a vidieť detaily projektu vrátane tímu, jednotlivých kontrolných bodov a ich termínov odovzdania a kompletného zadania vrátane príloh.

Hlavný scenár:

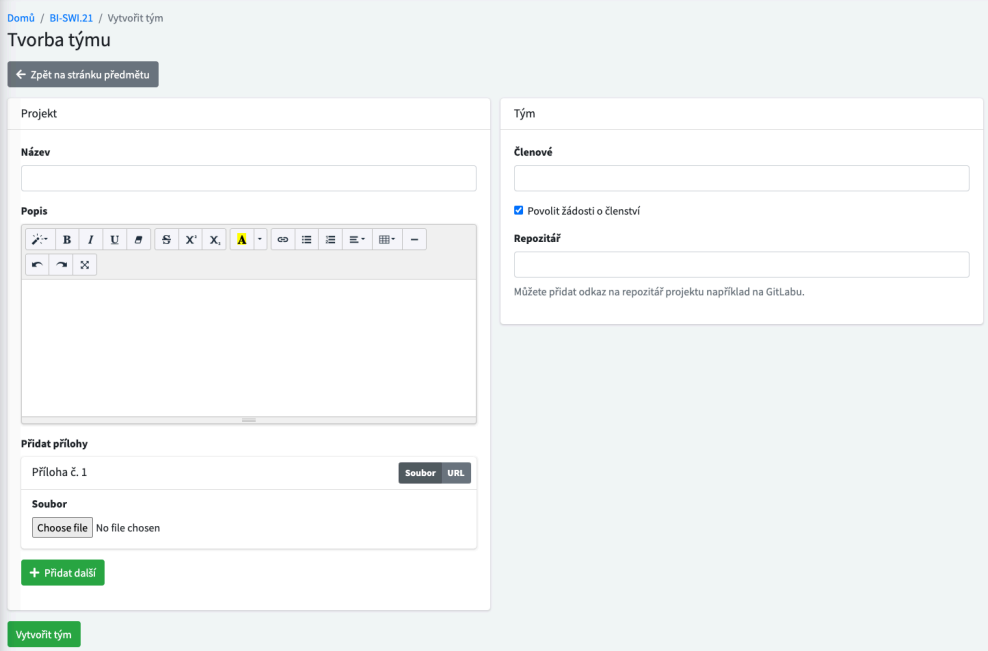
1. Užívateľ si zobrazí zoznam projektov (UC4).
2. Následne klikne na vybraný projekt.
3. Systém zobrazí stránku detailu projektu.

UC6 – Tvorba tímu

Aktéri: Vyučujúci, študent
Popis: Systém umožňuje užívateľom vytvoriť nový tím, ktorý pracuje na realizácii zadania. Okrem toho vie užívateľ pri tvorbe tímu povoliť žiadosti o členstvo ostatným študentom a pridať odkaz na Git repozitár projektu. Študent vie vytvoriť tím, len ak to povoľuje konfigurácia predmetu a v prípade vytvorenia tímu študentom sa daný študent stáva automaticky jeho vedúcim.

Hlavný scénar:

1. Užívateľ si zobrazí detail predmetu a zoznam projektov (UC4).
2. Užívateľ klikne na tlačítko tvorby nového tímu.
3. Systém zobrazí formulár tvorby nového tímu, ktorý je vidieť na obrázku 3.3..
4. Vyučujúci pri tvorbe tímu buď vyberie už existujúce zadanie alebo vytvorí nové, vyberie členov tímu, ktorí budú do tímu automaticky pridaní. Študent vytvorí nové zadanie a pozve do tímu študentov, ktorí budú do tímu pridaní až po prijatí tejto pozvánky.
5. Po vyplnení a odoslaní formuláru systém vytvorí nový tím.



Obr. 3.3 Snímka obrazovky - tvorba tímu

UC7 – Pripojenie sa do tímu

Aktéri: Študent
Popis: Študent vie požiadať o pripojenie sa do už existujúceho tímu. Žiadosť o prijatie, ktorá ešte nebola prijatá vie študent zrušiť. Po prijatí žiadosti sa študent automaticky stáva členom tímu. Žiadosť o prijatie vie prijať buď vedúci tímu, ak sa jedná o študentmi spravovaný tím pred schválením alebo vyučujúci v opačnom prípade.

UC8 – Opustenie tímu

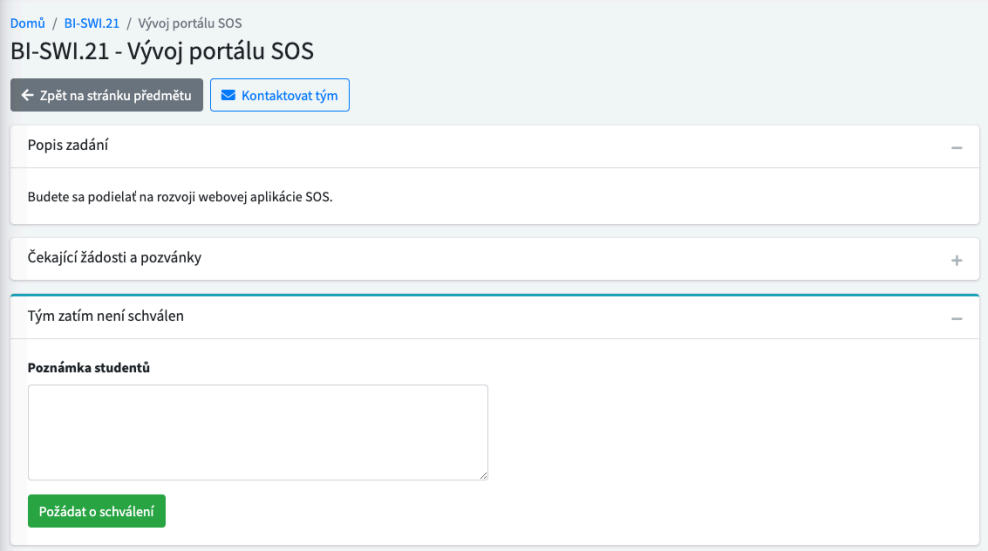
Aktéri: Študent
Popis: Systém umožňuje študentovi opustiť tím alebo požiadať o opustenie tímu na základe nastavení predmetu a stavu, v akom sa jeho tím nachádza. Študent vie opustiť tím bez požiadania v prípade, ak ide o študentmi spravovaný tím pred schválením vyučujúcim. V opačnom prípade musí študent požiadať o opustenie tímu. Po prijatí tejto žiadosti vyučujúcim už nie študent členom tímu.

UC9 – Predanie vedenia tímu

Aktéri: Vedúci tímu
Popis: Vedúci tímu vie predať vedenie tímu inému členovi tímu. Vedúci tímu je študent, ktorý vie navyše pozývať nových členov do tímu, prijímať žiadosti o pripojenie sa k tímu a požiadať o schválenie tímu. Nutno podotknúť, že tieto činnosti vie vykonávať len v prípade študentmi spravovaného tímu, ktorý ešte nebol schválený vyučujúcim. Okrem toho je ale ešte zodpovedný za odovzdanú prácu.

UC10 – Požiadanie o schválenie tímu

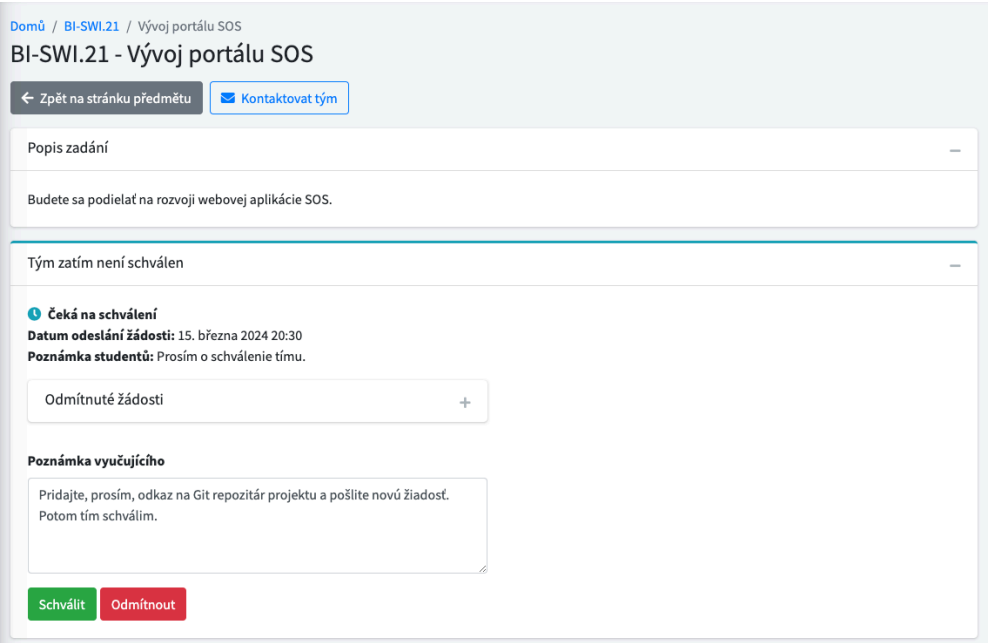
Aktéri: Vedúci tímu
Popis: Vedúci študentmi spravovaného tímu vie požiadať vyučujúceho o schválenie tímu. Po schválení vyučujúcim ďalej tento tím spravuje vyučujúci. V rámci podania žiadosti o schválenie tímu vie vedúci tímu vyučujúcemu poslať aj textovú poznámka, ako je možné vidieť na obrázku 3.4.



Obr. 3.4 Snímka obrazovky - žiadosť o schválenie tímu

UC11 – Schválenie alebo odmietnutie tímu

Aktéri: Vyučujúci
Popis: Vyučujúci vie buď schváliť alebo odmietnuť tím vytvorený študentmi. Ako vidieť na obrázku 3.5, aj vyučujúci vie poslať poznámku členom tímu, napríklad s dôvodom odmietnutia ich žiadosti o schválenie tímu.



Obr. 3.5 Snímka obrazovky - schválenie tímu

UC12 – Správa tímu

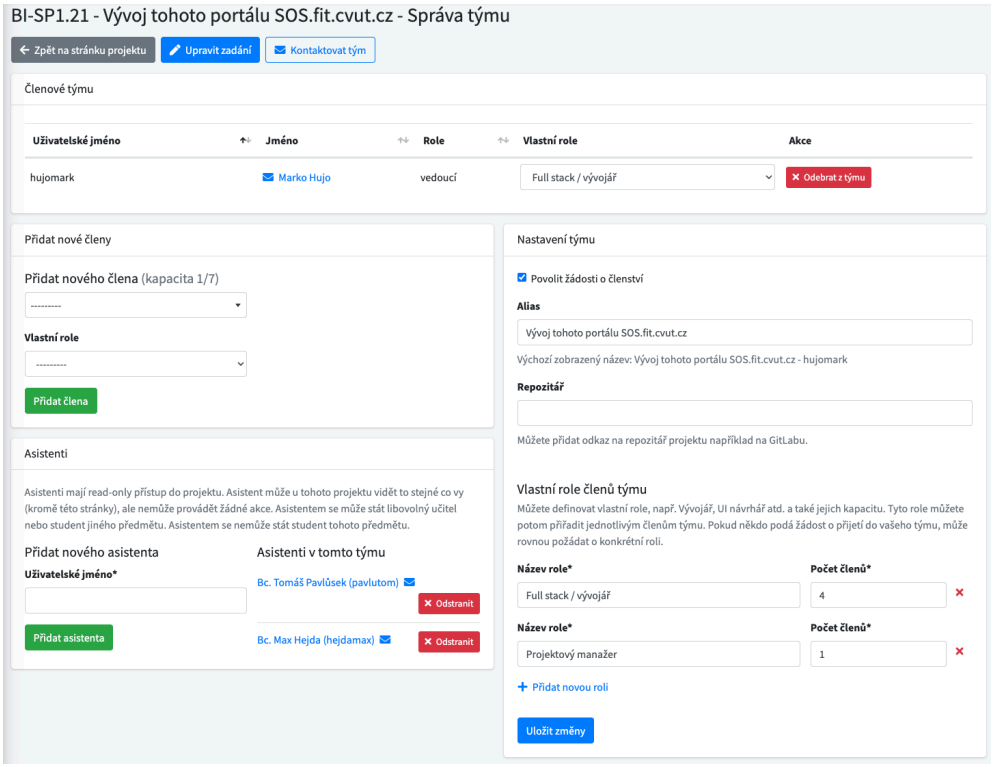
Aktéri: Vyučujúci, vedúci tímu
Popis: Systém umožňuje vyučujúcemu spravovať tím vrátane pridávania a odoberania členov tímu a svojich asistentov a definovania a priraďovania vlastných rolí členom tímu. Spravovať neschválený študentmi spravovaný tím vie aj vedúci tímu a to ale s jedným rozdielom – nevie spravovať svojich asistentov, keďže nie je vyučujúcim. Pohľad vyučujúceho na obrazovku správy tímu je vidieť na obrázku 3.6.

3.4 História vývoja

V tejto časti stručne predstavím históriu vývoja súčasného systému od pôvodných myšlienok cez úpravy a rozšírenia jednotlivých autorov až po moje zapojenie do vývoja v predmete BI-SP1.

3.4.1 Prvé nasadenie systému v predmete NI-NUR

V roku 2019 vznikol na FIT ČVUT systém NURIS určený pre odovzdávanie semestrálnych prác v predmete NI-NUR. Tento systém bol ale navrhnutý špecificky pre predmet NI-NUR a nebol preto využiteľný pre ďalšie predmety na



Obr. 3.6 Snímka obrazovky - správa tímu

FIT ČVUT. Z toho dôvodu vznikol v roku 2021 v rámci bakalárskej práce Ing. Tomáša Pavlůska [40] základ systému SOS s cieľom poskytnúť univerzálne riešenie pre odovzdávanie prác, použiteľné vo viacerých predmetoch na FIT ČVUT vrátane NI-NUR. Po dokončení práce bol systém SOS nasadený a uvedený do prevádzky pri výuke predmetu NI-NUR v zimnom semestri 2021/2022. Reálne využitie tohto systému v predmete NI-NUR odhalilo ďalšie požiadavky vrátane možnosti pre vyučujúceho schvaľovať zadania projektov vytvorené študentmi. Okrem toho bolo potrebné realizovať aj ďalšie menšie úpravy a opravy. V tom istom semestri práve v rámci predmetu NI-NUR sa do vývoja systému zapojili okrem kolegu Pavlůska aj ďalší študenti, ktorí sa vo svojej semestrálnej práci venovali návrhu zlepšenia UI pre vyučujúcich. Išlo primárne o konfiguráciu predmetu a hodnotenie študentských riešení.

3.4.2 Ročníkové práce na GJGJ

Na bakalársku prácu kolegu Pavlůska nadviazal v roku 2022 svojou bakalárskou prácou Bc. Max Hejda. V rámci tejto práce systém rozšíril a využil pre správu ročníkových prác na GJGJ. Hlavnou funkcionalitou, ktorú pre potreby ročníkových prác pridal kolega Hejda, bola možnosť zdieľania zadaní medzi

predmetmi. Okrem toho bol systém rozšírený o e-mailové notifikácie a možnosť prihlasovania prostredníctvom Google OAuth. V neposlednom rade bol do systému pridaný nový dátový zdroj, pretože GJGJ namiesto systému KOS[10] využíva systém Bakaláři[11]. [44]

3.4.3 Využitie systému v predmetoch BI-SWI, BI-SP1 a BI-SP2

V roku 2022 garant predmetov BI-SWI, BI-SP1 a BI-SP2 Ing. Jiří Mlejnek vybral systém SOS k využitiu pri výuke týchto predmetov. Ako popisujem vyššie v časti 3.4.1, systém SOS bol navrhnutý s ohľadom na rozšíriteľnosť a konfigurovateľnosť pre ďalšie predmety na FIT ČVUT. Ukázalo sa ale, že vznikli nové požiadavky na systém a pre plnohodnotné použitie v týchto predmetoch bolo potrebné systém rozšíriť a upraviť. Pôvodný autor súčasného systému Ing. Tomáš Pavlůsek teda systém ďalej vyvíjal v rámci svojej diplomovej práce [45], ktorú dokončil a obhájil v roku 2023. V letnom semestri 2022/2023 bol systém SOS uvedený do prevádzky v predmetoch BI-SWI a BI-SP1 a v zimnom semestri 2023/2024 v predmete BI-SP2, ktorý ale na systém kladie rovnaké požiadavky ako predmet BI-SP1. V letnom semestri 2022/2023 som mal predmet BI-SP1 zapísaný aj ja a z ponuky projektov, na ktorých som mohol pracovať, som si vybral práve vývoj systému SOS. Zaujala ma hlavne možnosť sa spolupodielať na zlepšovaní systému, ktorý sa prvýkrát používa v predmete, ktorý som v tej dobe študoval.

3.5 Vývoj v predmete BI-SP1

Na rozvoji systému SOS sa podieľam od februára 2023. V predemete BI-SP1 som na vývoji systému SOS pracoval v sedemčlennom tíme spolu s kolegami Janom Jamnickým, Janom Mrázkom, Timotejom Adamcom, Jakubom Sedláčkom, Luciánom Kučerom a Janom Pitákom. Našim vyučujúcim bol Ing. Jiří Hunka, okrem ktorého s nami spolupracovali aj Ing. Tomáš Pavlůsek a Bc. Max Hejda, ktorí pôsobili ako asistenti vyučujúceho Hunky.

S tímom sme pracovali v dvojtýždenných iteráciách, pričom na konci každej sme mali stretnutie s celým tímom vrátane vyučujúceho a jeho asistentov, na ktorom sme vedúcemu a jeho asistentom odprezentovali odvedenú prácu a získali spätnú väzbu.

Podieľali sme sa na viacerých činnostiach, ktorými sme prispeli k rozvoju systému SOS. V nasledujúcich častiach textu popíšem, akým činnostiam sme

[10] `https://kos.cvut.cz/`
[11] `https://www.bakalari.cz/`

sa venovali a upresním, akým som sa venoval ja.

3.5.1 Analýza a návrh

Pre splnenie požiadaviek predmetov BI-SWI a BI-SP1 sme sa s tímom venovali nasledovnými analytickým a návrhovým činnostiam:

- Analyzovali sme funkčné a nefunkčné požiadavky
- Vytvorili sme model prípadov použitia vrátane zoznamu aktérov, detailných scenárov jednotlivých prípadov použitia a diagramu
- Vytvorili sme diagramy niektorých vybraných aktivít
- Vytvorili sme doménový model
- Vytvorili sme stavové diagramy pre vybrané entity
- Analyzovali sme vybrané technológie, návrh logickej architektúry a rozdelenie aplikácie do logických celkov v kontexte zvolených technológií
- Vytvorili sme konceptuálny databázový model

Ja som sa venoval primárne analýze funkčných požiadaviek, vytvoreniu modelu prípadov použitia a vytvoreniu niektorých diagramov aktivít.

Za hlavné výhody, ktoré táto analýza systému priniesla, považujem podrobné pochopenie celého systému SOS a vytvorenie obsiahlej vývojárskej dokumentácie, čo potvrdzuje aj pôvodný autor systému Ing. Tomáš Pavlůsek vo svojej diplomovej práci. [45]

3.5.2 Rozhovory s užívateľmi a identifikácia nedostatkov

Mimo požiadavky predmetov BI-SWI a BI-SP1 sme dostali aj iné úlohy. Jednou z nich bolo získať a zhromaždiť spätnú väzbu na súčasný systém. S tímom sme sa rozhodli získať spätnú väzbu od študentov predmetov BI-SP1 a NI-NUR formou kvantitatívneho výskumu v podobe dvoch dotazníkov a od učiteľov predmetu BI-SP1 formou kvalitatívneho výskumu v podobe štrukturovaných rozhovorov. Formám výskumu a osvedčeným technikám zberu dát sa bližšie venujem v časti 4.2.1.

Z odpovedí, ktoré sme od študentov dostali a ktoré sme následne analyzovali vyplynuli nasledujúce nedostatky a návrhy na zlepšenie, z ktorých sme niektoré implementovali rovno v rámci predmetu, viac o implementácii v predmete BI-SP1 v časti 3.5.4:

- Viditeľný e-mail ostatných študentov čakajúcich na potvrdenie žiadosti o prijatie do tímu
- Viditeľné role ostatných študentov čakajúcich na potvrdenie žiadosti o prijatie do tímu
- Chýba upozornenie užívateľa na záväzné pripojenie sa do tímu (po prijatí žiadosti o vstup do tímu už nie je možné tím meniť)
- Niektorým študentom prišiel prehľad všetkých odovzdaní a iterácií projektu neprehľadný
- Študentom chýba v ponuke projektov zhrnutie technológií a možnosť filtrovať projekty podľa technológií, s ktorými budú na projekte pracovať
- Študentom tiež chýba možnosť filtrovania projektov podľa volných pozícií v tíme na konkrétne role
- Navigácia v systéme bola hodnotená priemerne, niektorým študentom prišla navigácia v systéme náročnejšia, nevedeli sa na začiatku v systéme zorientovať
- Väčšina študentov by uvítala možnosť návratu alebo zrušenia vykonaných akcií

Ako tím sme viedli aj štrukturované rozhovory s viacerými učiteľmi predmetu vrátane Ing. Zdeněka Rybolu, Ph.D., Ing. Jana Matouška a garanta predmetu Ing. Jiřího Mlejnka. Ja som sa zúčastnil rozhovoru s garantom predmetu Ing. Jiřím Mlejnkom. Naše zodpovednosti zahŕňali prípravu otázok a štruktúry rozhovoru, samotné prevedenie a vedenie rozhovoru, zapisovanie poznámok počas rozhovoru a spracovanie výsledkov. Z rozhovoru s garantom predmetu Ing. Jiřím Mlejnkom vyplynuli nasledovné nedostatky a návrhy na zlepšenie:

- Ak má študent súbežne zapísané predmety BI-SWI a BI-SP1, vidí v systéme oba predmety, pričom by ale mal pracovať len na projekte v predmete BI-SP1 a túto prácu si nechať uznať v predmete BI-SWI
- Import projektu z fakultného GitLabu do systému SOS
- Filtrovanie projektov, na ktorých je ešte voľné miesto
- Filtrovanie projektov podľa technológií
- U kontrolných bodov je vidieť termín odovzdania až po rozkliknutí
- Chýba možnosť aktualizovať zoznam študentov predmetu
- Chýba možnosť kopírovať alebo importovať nastavenia predmetov (napríklad pri tvorbe predmetu mať možnosť importovať nastavenia iného predmetu aj naprieč semestrami)
- Chýba možnosť kopírovať tímy z jedného predmetu do druhého

- Chýba možnosť obľúbených projektov
- Niektoré grafy nie sú prehľadné
- Zložitá terminológia systému, nechápe napríklad rozdiel medzi termínmi „tím“ a „projekt“

3.5.3 Užívateľská dokumentácia

Ďalšou úlohou nášho tímu, ktorá nás napadla na základe spätnej väzby od študentov, bolo vytvorenie dokumentácie pre užívateľov oboch rolí, teda študentov aj učiteľov, ktorá je dostupná na adrese `https://sos.pages.fit/dokumentace`. Konkrétne mojou úlohou bolo vytvoriť užívateľskú dokumentáciu pre študentov v češtine. Dokumentáciu sme vytvorili pomocou technológie VuePress[12], ktorá umožňuje generovanie statických stránok z Markdown súborov, ktoré sú kompilované do HTML (Hypertext Markup Language). [46] Pri tvorbe dokumentácie sme vychádzali zo scenárov prípadov užitia pripravených počas analýzy a návrhu, ktorú popisujem vyššie v časti 3.5.1.

3.5.4 Implementácia

V rámci implementácie sme sa s tímom venovali hlavne oprave chýb a vývoji menších nových funkcionalít. Ja som pracoval na nasledujúcich troch implementačných úlohách, ktoré súvisia s mojím zameraním práce - tímy a tímové projekty:

- Role študentov čakajúcich na potvrdenie žiadosti o prijatie do tímu sú viditeľné ostatným študentom (implementácia vykonaná v rámci Merge Requestu !225 na fakultnom GitLabe)
- Pridanie možnosti podať žiadosť o opustenie tímu vedúcim daného tímu (implementácia vykonaná v rámci Merge Requestu !228 na fakultnom GitLabe)
- Oprava kritickej chyby pri správe testerov vedúcim tímu (implementácia vykonaná v rámci Merge Requestu !241 na fakultnom GitLabe)

Okrem implementácie som sa ešte analyticky venoval nasledujúcim dvom záležitostiam. K implementácii som sa ale nedostal hlavne z časových dôvodov a kvôli prioritnejším úlohám.

- Zbytočný vedúci tímu – v systéme je globálne vynucované, aby každý tím mal práve jedného vedúceho, aj keď je to niekedy zbytočné (viac info v issue 92 na fakultnom GitLabe

[12] `https://vuepress.vuejs.org/`

- Žiadosť študenta o opustenie tímu sa nevymaže po odstránení študenta v správe tímu (viac info v issue 155 na fakultnom GitLabe

3.5.5 Návrh nového užívateľského rozhrania

Náš tím sa okrem vyššie spomenutých činností venoval aj návrhu nového UI na základe spätnej väzby získanej od užívateľov systému. Ako spomína už aj pôvodný autor systému Ing. Tomáš Pavlůsek vo svojej diplomovej práci, dôvodom pre návrh nového užívateľského rozhrania bola plánovaná zmena architektúry systému, aby bol do dvoch rokov[13] oddelený backend a nový frontend systému. [45] Je ale nutné podotknúť, že ja som sa priamo nezapojil do návrhu nového UI, pretože som na to nemal znalosti a zručnosti. Moje zapojenie do návrhu preto spočívalo len v podávaní spätnej väzby na návrhy, ktoré pripravovali kolegovia z tímu Jan Mrázek a Lucián Kučera, ktorí mali súbežne zapísaný aj predmet BI-TUR.

3.6 Zhodnotenie a budúcnosť systému

Systém je dnes použiteľný a nemá žiadne kritické chyby, ktoré by bránili v jeho používaní pri výuke. Známe nedostatky systému sa týkajú hlavne užívateľského rozhrania, ktoré nie je vždy prehľadné a intuitívne. Ako popisujem v časti 3.5.2, väčšinu týchto nedostatkov sme objavili s tímom v predmete BI-SP1 získaním spätnej väzby od užívateľov, prípadne manuálnym prechádzaním scenárov prípadov použitia. Niektoré z nedostatkov boli opravené buď mnou alebo kolegami z tímu, ako popisujem v časti 3.5.4 alebo kolegami Ing. Tomášom Pavlůskom a Bc. Maxom Hejdom. V systéme ale stále pretrvávajú nasledujúce nedostatky:

- Chýba upozornenie užívateľa na záväzné pripojenie sa do tímu (po prijatí žiadosti o vstup do tímu už nie je možné tím meniť)
- Neprehľadný prehľad iterácií, odovzdaní a termínov odovzdaní
- Chýba možnosť filtrovať projekty podľa technológií
- Navigácia v systéme asi stále ostáva náročná a mätúca hlavne pre nových užívateľov
- Chýba možnosť návratu alebo zrušenia vykonaných
- Občas chýba spätná väzba na vykonané akcie – potvrdenie úspešnej akcie a upozornenie na akcie, ktoré sa nepodarilo uskutočniť

[13] Kolega Pavlůsek dokončil svoju diplomovú prácu v máji roku 2023

- Niektoré akcie sa dajú vykonať viacerými tlačítkami a spôsobmi, čo je pre užívateľov niekedy mätúce
- Študenti nevidia email ostatných študentov čakajúcich na potvrdenie žiadosti o prijatie do tímu
- Nie je vyriešené súbežné zapísanie predmetov BI-SWI a BI-SP1
- Chýba možnosť kopírovať tímy z jedného predmetu do druhého
- Chýba možnosť obľúbených projektov
- Užívatelia nevidia na domovskej stránke najdôležitejšie veci, ktoré musia riešiť
- Chýba možnosť personalizácie domovskej stránky
- Niektoré grafy nie sú prehľadné
- Zložitá terminológia systému, užívatelia nerozumejú pojmom ako napríklad „tím“ a „projekt“ a rozdielom medzi nimi
- Niektoré ikony nie sú užívateľom zrozumiteľné
- Zahraničný študenti majú problém nájsť zmenu jazyka
- Je zbytočne vynucovaný vedúci tímu, aj keď to niekedy nedáva zmysel
- Je zmätok v tom, čo patrí do nastavenia tímu a čo do správy tímu

Vzhľadom na použité technológie a architektúru systému bude podľa mňa odstránenie a oprava niektorých z vyššie uvedených nedostatkov pomerne náročná. Vzhľadom na rastúcu komplexitu systému sa budúce nevyhnutné úpravy už existujúcich stránok môžu stať komplikovanými. Okrem toho sa domnievam, že pre SP tímy nebude jednoduché pridávať nové stránky. Dôvodom je podľa mňa fakt, že ide o šablónovací systém určený primárne na rýchlu tvorbu užívateľského rozhrania, ktorý nie je určený pre tvorbu pokročilejších užívateľských rozhraní rozsiahlych aplikácií, ktoré vyžadujú, aby ich UI bolo pohodlné, intuitívne a s jednoduchou navigáciou.

Systém sa zatiaľ javí ako udržateľný a pomerne jednoducho rozšíriteľný, čoho dôkazom je fakt, že systém vznikol ako náhrada systému NURIS primárne pre podporu výuky predmetu NI-NUR, no dnes sa používa aj v ďalších troch predmetoch na FIT ČVUT a dokonca aj mimo ČVUT na správu ročníkových práci na GJGJ. Do rozvoja aplikácie sa okrem pôvodného autora kolegu Ing. Tomáša Pavlůska zapojilo už viacero študentov FIT ČVUT vrátane kolegu Hejdy a mňa a mojich kolegov z tímu v predmete BI-SP1, čo podľa mňa tiež potvrdzuje udržateľnosť systému. Všetky zmeny, ktoré som implementoval, neboli náročné a systém a jeho štruktúru a architektúru som pochopil pomerne rýchlo. Na druhú stranu je nutné podotknúť, že som neimplementoval žiadne veľké rozšírenie, ktoré by vyžadovalo rozsiahle zásahy do systému. Myslím si,

že vzhľadom na architektúru súčasného systému by toto ale v prípade narastajúceho množstva kódu a náročnosti celého systému mohlo v budúcnosti tiež spôsobovať problémy.

Moje zapojenie do rozvoja systému pokračovalo aj v predmete BI-SP2 v zimnom semestri 2023/2024 s kolegami Janom Jamnickým, Janom Mrázkom a Timotejom Adamcom, s ktorými sme spolu na projekte pracovali už aj v predmete BI-SP1. Ako uvádza pôvodný autor systému Ing. Tomáš Pavlůsek v závere svojej diplomovej práce, jeho aktivita na rozvoji systému SOS pravdepodobne skončí s dokončením jeho magisterského štúdia na FIT ČVUT. Kolega Pavlůsek ďalej tvrdí, že projekt budú ďalej rozvíjať členovia SP tímov a Bc. Max Hejda.[45] Toto tvrdenie sa naplnilo, kolega Pavlůsek s nami už v zimnom semestri 2023/2024 nespolupracoval. Na začiatku semestra sme preto s tímom a s naším vyučujúcim Ing. Jiřím Hunkom a jeho asistentom Bc. Maxom Hejdom čelili otázke, ako ďalej projekt vyvíjať. Naším hlavným cieľom bolo zlepšiť UI systému a zaistiť lepší a udržateľnejší technický stav systému pre budúci vývoj ďalšími študentami. Z vyššie uvedených dôvodov sme sa rozhodli, že začneme pracovať na zmene architektúry systému SOS a oddelení backendu a frontendu, tak ako to uvádzal už aj kolega Pavlůsek vo svojej diplomovej práci [45].

Kapitola 4

Zber a analýza požiadaviek

V tejto kapitole sa venujem získavaniu, analýze a validácii funkčných aj nefunkčných požiadaviek kladených na systém. Pred samotným zberom požiadaviek ešte analyzujem potreby a priebeh niektorých predmetov na FIT ČVUT, ktoré súčasný systém SOS buď využívajú alebo sú vhodnými kandidátmi pre budúce využitie nového systému. Výsledkom tejto kapitoly je kategorizovaný a prioritizovaný zoznam funkčných a nefunkčných požiadaviek. Získané funkčné požiadavky v nasledujúcej kapitole analyzujem metódou prípadov použitia.

4.1 Potreby predmetov na FIT ČVUT

Pred zberom, analýzou a validáciou požiadaviek som sa rozhodol analyzovať predmety na FIT ČVUT, ktoré súčasný systém využívajú. Okrem prípravy na zber požiadaviek by mohla táto analýza v budúcnosti pomôcť aj pri možnom budúcom využití v ďalších predmetoch na FIT ČVUT, ale aj v iných prostrediach mimo FIT ČVUT. Pozerám sa na priebeh týchto predmetov počas celého semestru, úlohy a semestrálne práce, na ktorých študenti pracujú, spôsob komunikácie medzi učiteľmi a študentmi a formu hodnotenia a zakončenia predmetu. Snažím sa prísť na to, čo majú tieto predmety spoločné a prečo je pre tieto predmety výhodné systém SOS používať. Na základe tejto analýzy a mojej osobnej skúsenosti s niektorými z týchto predmetov určujem ďalšie predmety, ktoré by v budúcnosti mohli systém SOS používať.

4.1.1 Možnosti zakončenia predmetu

Na FIT ČVUT existujú nasledujúce spôsoby zakončenia a záverečného hodnotenia predmetu [47]:

- **Zápočet** – predmet je úspešne zakončený len zápočtom a študentovi nie je udelená žiadna známka
- **Klasifikovaný zápočet** – predmet je zakončený zápočtom, študentovi je udelená známka na základe získaného počtu bodov v semestri
- **Skúška** – predmet je zakončený zložením skúšky bez potreby získania zápočtu, známka je študentovi udelená na základe získaného počtu bodov na skúške
- **Zápočet a skúška** – predmet je zakončený skúškou, ktorej sa študent môže zúčastniť po získaní semestru, známka je študentovi udelená na základe súčtu získaného počtu bodov v semestri a na skúške

4.1.2 NI-NUR

Predmet NI-NUR je zakončený zápočtom a skúškou. Študenti predmetu NI-NUR môžu za prácu v semestri získať päťdesiat bodov, pričom získanie dvadsiatich piatich bodov predstavuje minimum pre udelenie zápočtu. Prácou na semestrálnom projekte môžu študenti získať štyridsaťštyri bodov, za účasť na usability testovaní ďalších šesť bodov. Študenti počas celého semestra pracujú na semestrálnej práci v dvoj až päťčlenných tímoch. Tému práce si zvolia sami alebo im môže byť pridelená vyučujúcim. Semestrálny projekt sa skladá z troch blokov, pričom z každého musia študenti získať určený minimálny počet bodov pre udelenie zápočtu. Všetky tri bloky práce vrátane účasti na usability testovaní sú odovzdávané prostredníctvom systému SOS. [48]

4.1.3 BI-SP1 a BI-SP2

Predmety BI-SP1 a BI-SP2 som sa rozhodol zjednotiť, pretože na seba nadväzujú a ich priebeh je takmer rovnaký. Cieľom týchto predmetov je si prakticky vyskúšať analýzu, návrh a realizáciu rozsiahlejšieho softwarového systému a prácu v tíme. Študenti preto pracujú v štyri až šesť členných tímoch na konkrétnom softwarovom projekte. Výsledok práce v predemte BI-SP1 je väčšinou ďalej rozvíjaný v predmete BI-SP2. Študenti si pred začiatkom výuky v systéme SOS vyberú jeden z ponúkaných projektov. Tím počas semestru pracuje na projekte a odovzdáva prácu v rámci viacerých kontrolných bodov, ktorých výstup a termíny sú individuálne dohodnuté v rámci projektu. Na

záver každej iterácie si môžu členovia tímu prerozdeliť body podľa odvedenej práce jednotlivých členov tímu. [49, 50]

4.1.4 BI-SWI

Práca na projekte v predmete BI-SWI je tiež podobná práci v predmetoch BI-SP1 a BI-SP2. Hlavnými rozdielmi sú menší rozsah projektu v BI-SWI a to, že si študenti tvoria tím a tému práce sami. Tvorba tímu a témy projektu ako aj odovzdávanie práce v troch iteráciách vo vopred definovaných termínoch a s vopred definovanými požiadavkami prebieha v systéme SOS. [51]

4.1.5 Potreby ďalších predmetov

Po analýze predmetov, ktoré systém SOS používajú, som sa pozrel na niektoré bakalárske predmety na FIT ČVUT, ktoré som počas štúdia absolvoval a ktoré systém SOS zatiaľ nepoužívajú, ale myslím si, že sú vhodnými kandidátmi.

4.1.5.1 BI-PST

Predmet BI-PST (Pravděpodobnost a statistika) je zakončený zápočtom a skúškou. Pre udelenie zápočtu je nutné zo semestru získať dvadsať bodov z možných štyridsať. Desať bodov môžu študenti získať za prácu na domácej úlohe, na ktorej pracujú v trojčlennom tíme. Výstupom úlohy je písomná správa vo formáte PDF, ktorú študenti posielajú učiteľovi e-mailom. [52, 53]

4.1.5.2 BI-KOM

Predmet BI-KOM (Konceptuální modelování) je zakončený zápočtom a skúškou. Za prácu počas semestra môžu študenti získať až šesťdesiat bodov, z toho až štyridsať môžu získať zo semestrálnej práce. Nutné minimum pre udelenie zápočtu je zisk aspoň tridsiatich bodov. Každý študent musí vypracovať individuálne alebo vo dvojici semestrálnu prácu, ktorá je rozdelená na dve časti. Tému práce si študenti volia sami a po zvolení si ju registrujú v systéme Moodle do konca tretieho výukového týždňa. Odovzdanie oboch častí prebieha v systéme Moodle vo vopred stanovených termínoch formou PDF súboru, ktorý obsahuje text a diagramy a súboru vo formáte .oop, v prípade, že študenti vypracovali prácu v programe OpenPonk. V prípade dvojčlenného tímu odovzdá prácu jeden z dvojice a druhý člen musí toto odovzdanie ešte potvrdiť. [54, 55]

4.1.5.3 BI-PRR

Predmet BI-PRR (Projektové řízení) je zakončený zápočtom a skúškou. Pre udelenie zápočtu je potrebné odovzdať osemdesiat percent úloh, ktoré sú tímové aj individuálne. Študenti počas semestra pracujú na štyroch individuálnych úlohách a štyroch tímových úlohách. Na prvom cvičení študenti vytvoria troj až šesťčlenné tímy, v ktorých počas semestra pracujú na úlohách v rámci projektu. Tému projektu si vyberú študenti a následne je im vytvorený kanál v systéme Microsoft Teams, v ktorom študenti komunikujú s tímom ale aj s učiteľom a odovzdávajú všetky tímové aj individuálne úlohy. V systéme Microsoft Teams prebieha aj hodnotenie a podávanie spätnej väzby na prácu. Alternatívne je možné úlohy odovzdať v systéme Moodle. Výstupom tímových úloh je väčšinou prezentácia na cvičení s využitím odovzdaných a konzultovaných podkladov. Výstupom individuálnych úloh je väčšinou krátka správa vo formáte nejakého dokumentu. [56, 57]

4.1.5.4 BI-TIS

Predmet BI-TIS (Tvorba informačních systémů) je zakončený zápočtom a skúškou, pričom až šesťdesiat bodov vedia študenti získať za prácu na semestrálnom projekte. Študenti na začiatku semestra utvoria troj až štvorčlenné tímy a vyberú si jednu z viacerých ponúkaných tém. Projekt je rozdelený na viac častí, výstupom väčšiny častí je správa vo formáte PDF. Prácu vždy odovzdáva jeden z členov tímu v systéme Moodle vo vopred stanovených termínoch. Vyučujúci tohto predmetu sú z FEL ČVUT, kde vedú podobné predmety, kde by systém SOS podľa mňa tiež našiel uplatnenie. [58]

4.1.5.5 BI-OOP

Predmet BI-OOP (Object-Oriented Programming) je zakončený zápočtom a skúškou. Študenti sú počas semestra povinný pracovať na individuálnom programovacom projekte, z ktorého môžu získať až päťdesiatdva bodov. Na začiatku semestra si študenti zvolia jednu z navrhnutých tém a prácu odovzdajú do vopred stanoveného termínu. Keďže sa jedná o programovací projekt, študenti využívajú systém GitLab, kde si vytvoria privátny repozitár s prístupom pre vyučujúcich. Samotné odovzdanie projektu prebieha vyplnením dotazníku, kde študent uvedie zvolenú tému, adresu projektu v systéme GitLab a prípadne nejakú poznámku. Hodnotenie a podávanie spätnej väzby prebieha formou správ v systéme Microsoft Teams. [59, 60]

4.2 Zber a analýza požiadaviek

V tejto sekcii teoreticky analyzujem, aké sú osvedčené možnosti a formy výskumu a zberu dát v kontexte získavania požiadaviek na software. Na základe tejto analýzy a potrieb tohto projektu potom uvediem, aké metódy získavania požiadaviek sme s kolegami z tímu aplikovali. Ďalej vysvetlím, ako vieme požiadavky na software kategorizovať a v neposlednom rade uvediem zoznam požiadaviek kladených na systém SOS. Tieto požiadavky následne analyzujem metódou prípadov užitia až v časti 5.3.

4.2.1 Metodika zberu dát

Na úvod definujem a porovnám dve základné formy výskumu, na čo nadviažem predstavením techník zberu požiadaviek na software. Na základe tejto analýzy uvediem ako som s tímom získaval požiadavky na nový systém.

4.2.1.1 Formy výskumu

Existujú dve hlavné formy výskumu a zberu dát, ktoré sa líšia spôsobom, akým dáta zbierajú a typom dát, ktoré zbierajú. Jedná sa o kvalitatívny a kvantitatívny výskum.

Kvalitatívny výskum: Zber, analýza a interpretácia nečíselných dát, ako sú slová, obrázky, videá alebo zvuky. Tieto údaje je často možné získať prostredníctvom napríklad rozhovorov. Cieľom kvalitatívneho výskumu je pochopiť, ako jednotlivci alebo skupiny subjektívne interpretujú a vnímajú svet, pričom sa snaží zachytiť ich skúsenosti čo najbližšie k tomu, ako ich skutočne vnímajú a prežívajú. Kvalitatívny výskum nám teda pomáha skúmať danú problematiku viac do hĺbky. [61]

Kvantitatívny výskum: Proces objektívneho zberu a analýzy číselných údajov, pričom sa zameriava na zovšeobecnenie výsledkov na širšiu populáciu. Cieľom kvantitatívnych výskumníkov je pochopiť, predpovedať a validovať ich domnienky a teórie a stanoviť všeobecné zákony v rôznych prostrediach a kontextoch. Jednou z metód kvantitatívneho výskumu sú napríklad dotazníky s uzavretými otázkami, ktoré poskytujú buď číselné údaje alebo údaje, ktoré môžeme nejako kategorizovať. [61]

4.2.1.2 Získavanie požiadaviek

Existuje viacero osvedčených techník získavania požiadaviek [62, kap. 7]:

Rozhovor: Zisťovanie potrieb zákazníka alebo užívateľa pýtaním sa otázok na súčasné riešenia a na nový systém, ktorý sa má vytvoriť. Získame tým celkové pochopenie toho, čo užívatelia robia, na čo súčasné riešenie používajú, s akými ťažkosťami sa stretávajú pri súčasných riešeniach a ako by mohli komunikovať s novým systémom.

Workshop: Štrukturované stretnutie s vybranou skupinou zákazníkov a iných odborníkov vrátane vývojárov, testerov alebo návrhárov.

Focus group: Interaktívne stretnutie s vybranou skupinou užívateľov a zákazníkov.

Pozorovanie: Pozorovanie užívateľov pri práci so súčasným systémom s cieľom overiť informácie získané z iných zdrojov, identifikovať nové témy na rozhovory, zistiť problémy so súčasným systémom a určiť spôsoby, ako môže nový systém lepšie podporovať danú prácu, ktorú užívateľ vykonáva.

Dotazník: Kvantitatívny výskum s veľkou skupinou užívateľov.

Analýza systémového rozhrania: Skúmanie systémov, s ktorými budeme náš systém integrovať.

Analýza užívateľského rozhrania: Skúmanie užívateľského rozhrania súčasných systémov.

Analýza dokumentácie: Skúmanie dokumentácie súčasných systémov vrátane špecifikácie požiadaviek, biznis procesov, užívateľských príručiek, slovníčkov pojmov, dátových modelov, súvisiacich noriem a regulácií a ďalších relevantných dokumentov.

Okrem vyššie uvedených techník získavania požiadaviek sa považuje za osvedčený postup aj skúmanie problémov, nedostatkov a návrhov na vylepšenie od užívateľov súčasných systémov s cieľom získať nápady na požiadavky na nový systém, ktorý budeme implementovať. Ďalšou osvedčenou technikou, ktorá využíva súčasné riešenie, je opätovné použitie už existujúcich požiadaviek na súčasný systém. Opätovné používanie požiadaviek môže zvýšiť produktivitu, zlepšiť kvalitu, urýchliť dodanie, znížiť náklady, ako aj viesť k väčšej konzistentnosti medzi súvisiacimi systémami, teda medzi súčasným systémom a novým systémom, ktorý vyvíjame. Práve vývoj novej aplikácie, ktorá nahradí tú súčasnú je jedným z najbežnejších prípadov, kedy je možné a vhodné opätovne využiť už existujúce požiadavky. [62, kap. 3 a 18]

4.2.1.3 Zvolené metódy získavania požiadaviek

V tejto časti popíšem, ako som spolu s kolegami z tímu získaval požiadavky po tom, ako sme sa rozhodli vytvoriť nový systém.

Na začiatku vývoja nového systému sme sa rozhodli najprv validovať súčasné požiadavky na súčasný systém, ktoré sme analyzovali už v letnom semestri 2022/2023 v predmete BI-SP1. Tieto požiadavky boli získané kolegami Ing. Tomášom Pavlůskum a Bc. Maxom Hejdom pri vývoji súčasného systému v rámci ich záverečných prác. Okrem toho sme sa ale zamýšľali aj nad ďalšími možnými prípadmi použitia s cieľom vybrať najpotrebnejšie funkcionality nového systému. Už v tejto etape vývoja sme sa snažili požiadavky vhodne prioritizovať podľa potrieb jednotlivých predmetov.

Na začiatku letného semestra 2023/2024, kedy sme úspešne dokončili implementáciu a nasadenie prvej verzie nového systému sme sa zamerali na analýzu užívateľského rozhrania súčasného systému formou prechádzania scenárov prípadov použitia.

Analýza existujúcich požiadaviek a užívateľského rozhrania súčasného systému nám spolu so zistenými nedostatkami súčasného systému slúžila ako podklad pre získavanie nových požiadaviek a ďalšiu a precíznejšiu validáciu tých existujúcich, čo sme vykonali formou rozhovorov s nasledujúcimi vyučujúcimi na FIT ČVUT:

- **Ing. Jiří Hunka** – vyučujúci predmetov BI-SP1, BI-SP2 a NI-NUR
- **Ing. Jiří Mlejnek** – garant a vyučujúci predmetov BI-SWI, BI-SP1 a BI-SP2
- **Ing. Zdeněk Rybola, Ph.D.** – vyučujúci predmetov BI-SWI, BI-SP1 a BI-SP2

Na začiatku rozhovorov sme najprv validovali body z rozhovorov v minulých semestroch. Zisťovali sme detaily k odhaleným nedostatkom, navrhovali sme úpravy, pýtali sme sa na návrhy vyučujúcich a zisťovali sme, či sú vyučujúci spokojní s novými funkcionalitami a opravami chýb. Následne sme validovali existujúce požiadavky. Snažili sme sa určiť prioritu jednotlivých požiadaviek podľa metódy MoSCoW, ktorú definujem nižšie v časti 4.2.2. Zisťovali sme tiež, ako učitelia so systémom pracujú, na čo ho najčastejšie používajú a s čím majú najväčšie problémy. Na koniec sme získavali nové požiadavky formou otvorenej diskusie, kedy učitelia sami prichádzali s nápadmi alebo s nami zdieľali postrehy od ich študentov.

Z rozhovorov sme zistili nové nedostatky, návrhy na zlepšenie a funkčné požiadavky. Niektoré nové funkčné požiadavky ale len modifikujú alebo určitým spôsobom dopĺňajú tie existujúce. Okrem toho boli spomenuté aj známe nedostatky, ku ktorým ale učitelia pridali návrhy riešenia alebo detailnejšie informácie po roku skúseností s používaním systému. Uvádzam len zistenia týkajúce sa tímov a tímových projektov. Ďalším zisteniam týkajúcich sa iných doménových oblastí sa venujú kolegovia Jan Jamnický a Jan Mrázek vo svojich bakalárskych prácach.

Kopírovanie projektu: Jedná sa o kopírovanie projektu do iného predmetu alebo v rámci rovnakého predmetu ale do iného semestra. Na FIT ČVUT by toto mohlo byť využité napríklad pre kopírovanie projektu z BI-SP1 do BI-SP2 a potom zase do BI-SP1, pretože je veľmi časté, že študenti v predmete BI-SP2 pokračujú v projekte, na ktorom pracovali v predmete BI-SP1.

Z rozhovorov vyplynuli aj návrhy, ako kopírovanie projektov riešiť. Predstava učiteľov je taká, že by si mohli vybrať, čo by chceli kopírovať. Primárne by sa ale kopírovalo zadanie a adresa GitLab repozitára, prípadne tím s tými členmi, ktorí majú zapisaný predmet, do ktorého bude projekt skopírovaný. Kopírovanie tímu ale nie je vždy žiadané, keďže napríklad pri prechode z BI-SP2 do BI-SP1 neostáva rovnaký žiadny študent.

Pri kopírovaní členov tímu vznikajú ešte ďalšie dva problémy. Prvým je automatické zapísanie študenta do projektu bez jeho vedomia a súhlasu. Toto by sa dalo riešiť napríklad pozvánkou do tímu, o ktorej by študentovi prišla notifikácia. Druhým je počet členov tímu viditeľný ostatným študentom. Ostatní študenti by mali vidieť počet členov tímu bez študentov, ktorí ešte túto pozvánku neprijali, aby sa do tímu mohli tiež hlásiť.

Vytvorenie tímu bez nutnosti priradenia k projektu: Občas sa stáva, že skupina kamarátov by chcela na projekte pracovať spolu. Radi by si teda vytvorili tím a prihlásili sa na nejaký projekt spolu.

Možnosť pridať poznámku k žiadosti o pripojenie sa k tímu: V poznámke by sa študenti mohli napríklad predstaviť a uviesť, aké majú skúsenosti s danými technológiami, na základe čoho by mohli učitelia študentov vyberať.

Možnosť pridať termíny konzultácií pri tvorbe projektu: Pri tvorbe projektu by učiteľ mohol zadať termíny, kedy preferuje konzultácie s tímom. Študenti by to videli a vedeli by sa na základe toho rozhodnúť, či o projekt majú záujem.

Možnosť deaktivovania projektu: Garant predmetu by mohol mať možnosť deaktivácie projektov v danom predmete, aby ich študenti nevideli a nevedeli sa do nich hlásiť. Dôvodom je to, že sa stáva, že je veľa prázdnych projektov a málo študentov. V takom prípade je nutné, aby sa prázdne projekty deaktivovali, aby sa študenti rozdelili na voľné miesta v iných už čiastočne zaplnených projektoch. S týmto súvisí aj možnosť jasne vidieť, ktoré projekty študenti vidia v ponuke a ktoré sú deaktivované. Pre túto požiadavku existuje aj issue 79 na fakultnom GitLabe v repozitári súčasného systému.

Zbytočný vedúci tímu: Niekedy je táto rola zbytočná, napríklad v predmetoch BI-SP1 a BI-SP2. Pre túto požiadavku existuje aj issue 92 na fakult-

nom GitLabe v repozitári súčasného systému.

Nezobrazovanie projektov študentom s už zvoleným projektom: V súčasnom systéme študenti so zvoleným projektom nevidia ponuku projektov v tom istom predmete. Dôvodom je to, aby študenti neprechádzali v semestri z projektu na projekt a nespôsobovali tak chaos. Toto rozhodnutie má ale aj negatívne následky. Stáva sa totiž, že študent je jediným členom tímu, ktorý nevie opustiť, pretože žiadosť o opustenie tímu musí prijať učiteľ, ktorý s ním ale nekomunikuje. Rád by si daný študent našiel iný projekt, ale nemá ako, pretože ponuku nevidí kvôli tomu, že už je členom iného tímu.

Možnosť zvýšiť kapacitu tímu pri prijímaní žiadosti o členstvo: V súčasnom systéme je táto možnosť len v nastaveniach projektu. Môže sa ale stať, že sa do tímu hlási viac študentov ako je maximálna kapacita tímu. Učiteľ by rád prijal všetkých študentov a v rámci prijímania žiadosti študenta nad kapacitu by mohol navýšiť kapacitu tímu a vďaka tomu študenta prijať.

Okrem týchto návrhov a požiadaviek by som rád ešte spomenul ďalšie dve zistenia, ktoré sa netýkajú nikoho časti projektu, ale sú považované za dôležité a prioritné.

Domovská stránka neobsahuje užitočné informácie: O tomto nedostatku už vieme, ale kolegovia Mlejnek a Rybola po prvom roku práce so súčasným systémom navrhli, že by na domovskej stránke mohli byť činnosti, ktoré musia vyriešiť v jednotlivých predmetoch, napríklad žiadosti o schválenie tímu alebo ohodnotenie odovzdanej práce.

Nikto z trojice nepoužíva notifikácie: Vďaka rozhovorom sme zistili, že všetci traja kolegovia ignorujú notifikácie. Dôvodom je podľa kolegu Hunky to, že nie sú dostatočne agresívne a nenútia ho danú akciu vykonať. Ďalším problémom je to, že notifikácia sa nezruší v prípade vyriešenia toho, na čo upozorňuje bez zobrazenia danej notifikácie.

4.2.2 Kategorizácia požiadaviek

Požiadavky vieme podľa ich zamerania rozdeliť na dve základne kategórie – funkčné a nefunkčné. Funkčné požiadavky predstavujú konkrétne funkcionality a správanie systému, ktoré sa prejavia za určitých podmienok. Definujú, čo majú vývojári implementovať, aby splnili potreby užívateľov. Nefunkčné požiadavky popisujú vlastnosti systému a obmedzenia, ktoré sú na systém kladené. [62, kap. 1] Okrem tohto základného a jednoduchého rozdelenia vieme pre kategorizáciu a prioritizáciu požiadaviek použiť rôzne pokročilejšie metódy, napríklad FURPS a MoSCoW.

Metóda FURPS rozdeľuje funkčné aj nefunkčné požiadavky do nasledujúcich piatich kategórii [63]:

- **Functionality** (funkčnosť) – funkčné požiadavky
- **Usability** (použiteľnosť) – ako bude užívateľ alebo externé systémy systém používať, ako bude prístupný a aké definuje rozhrania
- **Reliability** (spoľahlivosť) – definuje dostupnosť, spoľahlivosť, a odolnosť systému voči chybám a výpadkom
- **Performance** (výkon) – efektivita, rýchlosť a škálovateľnosť systému
- **Supportability** (podporovateľnosť) – udržateľnosť, modularita a jednoduchosť vylepšení systému

V praxi sa často stretávame s tým, že chceme takto kategorizovaným požiadavkám stanoviť priority. Na to vieme použiť napríklad metódu MoSCoW, ktorá požiadavky rozdeľuje do nasledujúcich štyroch tried podľa priority [62, kap. 16]:

- **Must have** – požiadavka musí byť splnená, aby bol projekt úspešný
- **Should have** – požiadavka je dôležitá a mala by byť zahrnutá v systéme, ak je to možné, ale nie je nevyhnutná na to, aby sa projekt považoval za úspešný
- **Could have** – žiaduca požiadavka, ktorú by sme mali realizovať, nie je to ale nutné a vieme ju odložiť alebo zrušiť
- **Won't have** – táto požiadavka sa v súčasnosti nebude realizovať, môžme to ale zvážiť v budúcnosti

4.2.3 Zoznam funkčných požiadaviek

Z analýzy a validácie existujúcich funkčných požiadaviek a zberu nových sme objavili nasledujúce funkčné požiadavky na nový systém, ktoré sa týkajú tímov a tímových projektov. Ku každej požiadavke uvádzam aj prioritu podľa MoSCoW metódy a pridávam aj to, či sa jedná o všeobecnú požiadavku alebo požiadavku konkrétneho predmetu, čo budem označovať ako „Kategória FIT".

FP1 – Tvorba a definícia zadaní

Popis: Užívateľ vie vytvoriť a definovať zadanie projektu vrátane názvu, textového popisu a príloh vo forme súborov alebo URL odkazov
Kategória FIT: Všeobecná
Priorita: Must have

FP2 – Tvorba tímov

Popis: Užívateľ vie vytvoriť vytvoriť tímy skladajúce sa zo študentov. Do tímu sa môžu študenti prihlasovať sami formou žiadostí, ktoré musí schváliť zodpovedná osoba. Zodpovedná osoba vie tiež buď pozvať alebo automaticky pridať študentov do tímu na základe konfigurácie a role užívateľa.
Kategória FIT: Všeobecná
Priorita: Must have

FP3 – Tvorba projektov

Popis: Užívateľ vie tímu priradiť zadanie na realizáciu, čím vzniká projekt. Aplikácia by mala užívateľom umožňovať vytvoriť projekt bez toho, aby bolo potrebné vopred vytvoriť tím a zadanie.
Kategória FIT: Všeobecná
Priorita: Must have

FP4 – Schvaľovanie projektov

Popis: Ak je projekt vytvorený študentmi, musí byť pred začiatkom realizácie zadania schválený vyučujúcim. Aplikácia umožňuje členom tímu požiadať o schválenie projektu. Vyučujúci môže túto žiadosť prijať alebo odmietnuť.
Kategória FIT: NI-NUR, BI-SWI
Priorita: Should have

FP5 – Filtrovanie projektov

Popis: Systém umožňuje užívateľom filtrovať projekty podľa názvu, učiteľa, členov tímu, voľnej kapacity v tíme a technológií.
Kategória FIT: Všeobecná
Priorita: Should have

FP6 – Deaktivovanie projektu

Popis: Systém umožňuje garantovi predmetu deaktivovať ľubovolný projekt v danom predmete a učiteľovi deaktivovať ľubovoľný projekt, za ktorý je zodpovedný. Deaktivácia projektu znamená, že tento projekt študenti nevidia. V systéme je tiež jasne vidieť, ktoré projekty študenti vidia a ktoré sú deaktivované.
Kategória FIT: Všeobecná
Priorita: Could have

FP7 – Pridanie asistenta

Popis: Učiteľ vie do projektu pridať svojho asistenta. Asistent je užívateľ, ktorý nemusí byť učiteľom a nesmie byť študentom daného predmetu. Asistent vidí v aplikácii informácie o danom projekte, no nemôže vykonávať žiadne akcie.
Kategória FIT: BI-SP1, BI-SP2
Priorita: Could have

FP8 – Vlastné role členov tímu

Popis: Aplikácie umožňuje definovať vlastné role členov tímu a tieto role členom priradiť. Pri žiadaní o pripojenia sa k tímu vie študent požiadať o konkrétnu rolu.
Kategória FIT: BI-SP1, BI-SP2
Priorita: Could have

FP9 – Kopírovanie projektov

Popis: Systém umožňuje kopírovať projekty vrátane zadania a tímu medzi predmetmi, prípadne v rovnakom predmete medzi rôznymi semestrami. Kopírovanie projektu je konfigurovateľné podľa predstáv učiteľa.
Kategória FIT: BI-SP1, BI-SP2
Priorita: Won't have (Túto požiadavku som sa rozhodol zatiaľ nerealizovať, pretože sa jedná o pomerne rozsiahlu implementačnú úlohu, ktorá navyše nadväzuje na moju prácu ako aj prácu oboch kolegov z tímu. Keďže sa ale jedná o pomerne novú požiadavku, ktorá vyplynula z používania súčasného systému, vykonal som analytické a návrhové činnosti, ktoré uľahčia implementáciu tejto požiadavky. Požiadavku som analyzoval, získal detaily od viacerých vyučujúcich používajúcich súčasný systém a spolu s nimi a kolegami z tímu uvažoval, ako k celému problému pristúpiť. Návrhu riešenia sa bližšie venujem v časti 5.3.)

4.2.4 Zoznam nefunkčných požiadaviek

Na nový systém sú kladené nasledujúce nefunkčné požiadavky.

NP1 – Dostupnosť

Popis: Aplikácia je dostupná nonstop.
Kategória FURPS: R

Priorita: Must have

NP2 – Oprávnenie pre prístup

Popis: Aplikácia je dostupná len oprávneným užívateľom.
Kategória FURPS: R
Priorita: Must have

NP3 – Jazyková lokalizácia

Popis: Aplikácia je dostupná v češtine aj angličtine pre zahraničných študentov. Prepnutie jazyka je ihneď viditeľné.
Kategória FURPS: U
Priorita: Must have

NP4 – Škálovateľnosť aplikácie

Popis: Aplikácia musí byť škálovateľná a zvládať v budúcnosti rastúce množstvo záťaže, užívateľov a dát bez toho, aby došlo k výraznému zníženiu výkonu
Kategória FURPS: P
Priorita: Should have

NP5 – Rozšíriteľnosť a udržateľnosť aplikácie

Popis: Aplikácia bude v budúcnosti jednoducho rozšíriteľná o ďalšie funkcionality.
Kategória FURPS: S
Priorita: Should have

NP6 – Responzivita

Popis: Aplikácia má responzívne užívateľské rozhranie, dá sa teda používať aj na mobilných zariadeniach.
Kategória FURPS: U
Priorita: Should have

NP7 – Intuitívne užívateľské rozhranie

Popis: Užívateľské rozhranie aplikácie je intuitívne, prehľadné a jednoduché aj s narastajúcim počtom funkcionalít. Navigácia v systéme je tiež jednoduchá

a intuitívna.
Kategória FURPS: U
Priorita: Should have

Kapitola 5

Návrh nového systému

Cieľom tejto kapitoly je čitateľovi predstaviť návrh nového systému vrátane toho, ako som sa s kolegami z tímu rozhodol na vývoji nového systému pracovať. Na úvod teda predstavím zvolený spôsob vývoja, ktorý sme si s tímom vybrali pre vývoj nového systému ako aj zvolený verzovací model. Následne analyzujem funkčné požiadavky metódou prípadov použitia. Pred samotným vývojom bolo dôležité navrhnúť architektúru nového systému a vybrať vhodné technológie, čo tiež v tejto kapitole popíšem. Následne čitateľovi predstavím návrh MVP, teda minimálneho životaschopného produktu a predstavím viaceré návrhy doménového modelu, API a užívateľského rozhrania.

5.1 Spôsob vývoja

V rámci projektu sme s tímom čelili výzve prispôsobiť spôsob vývoja obmedzenej časovej dostupnosti členov tímu. Všetci sme študentmi v poslednom ročníku bakalárskeho štúdia a zároveň získavame pracovné skúsenosti z praxe mimo školu. Rolu zákazníkov v našom prípade suplujú garanti a vyučujúci predmetov na FIT ČVUT, ktorí mimo výuky pracujú aj na iných pozíciách. Tento faktor mal zásadný vplyv na výber a uplatňovanie metodík vývoja softwaru. Situácia nás viedla k záveru, že agilné metodiky ako Scrum, ktoré vyžadujú denné stretnutia a plné nasadenie od vývojového tímu aj zákazníkov, nie sú pre nás ideálne. Napriek tomu sme sa rozhodli vyvíjať iteratívne a prijať niektoré agilné postupy, ktoré sme modifikovali a prispôsobili našim potrebám. Keďže sme pracovali iteratívne, všetky návrhy, ktoré v tejto kapitole uvediem, som vykonával vo viacerých iteráciách, dôsledkom čoho sa návrhy menili, preto aj v tejto kapitole budem uvádzať viaceré návrhy, ktoré už nemusia byť aktuálne, no slúžili ako podklad pre novšie návrhy v ďalších iteráciách.

S tímom sme na vývoji nového systému pracovali v krátkych dvojtýždňových

iteráciách. Raz za iteráciu sme organizovali interné tímové stretnutia, na ktorých sme s kolegami z tímu zdieľali naše pokroky, spoločne riešili problémy, na ktoré sme narazili, pripadne sme spoločne pracovali na úlohách, ktoré vyžadovali komunikáciu a úzku spoluprácu celého tímu, ako napríklad výber vhodného spôsobu vývoja alebo definíciu a návrh MVP a jeho doménového modelu. Na konci letného semestra, kedy každý z členov tímu pracoval primárne na implementácii a testovaní svojho zamerania práce, sme sa začali stretávať nepravidelne len v prípade potreby.

Vedúci tejto práce Ing. Jiří Hunka je vyučujúcim viacerých predmetov používajúci systém SOS. Systém teda aktívne využíva pri výuke týchto predmetov, vďaka čomu má jasnú predstavu o prednostiach a nedostatkoch súčasného systému a o požiadavkách kladených na systém. Okrem toho má prehľad o celom vývoji a histórii vývoja systému, pretože viedol každú záverečnú prácu, ktorá sa systémom SOS zaoberala a viedol všetky projekty týkajúce sa systému SOS v predmetoch BI-SP1 a BI-SP2. V kontexte agilného vývoja softwaru, ktorý sme s tímom pri vývoji nového systému čiastočne aplikovali, teda pôsobil ako náš zákazník, s ktorým sme úzko spolupracovali. Stretnutia s vedúcim práce prebiehali na konci každej iterácie. Cieľom týchto stretnutí bolo odprezentovať výsledky našej práce, diskutovať o výsledkoch našej práce, získať cennú spätnú väzbu a spoločne naplánovať ďalšie kroky a úlohy do ďalšej iterácie. Aj tieto stretnutia sme v letnom semestri organizovali nepravidelne len v prípade potreby.

Pred samotným vývojom sme s tímom pripravili plán práce na celý zimný semester, ktorý jasne definoval, čomu sa budeme v každej iterácii venovať a aký výsledok od nás môže vedúci práce na konci každej iterácie očakávať. Okrem toho sme definovali a naplánovali, aký výsledok dodáme na konci zimného semestra. Tento plán sme iteratívne validovali a postupne upravovali na základe spätnej väzby od vedúceho práce a iných nečakaných zmien a okolností, ktoré prichádzali počas semestra.

Plánovanie jednotlivých iterácií zvyčajne prebiehalo na stretnutí s vedúcim práce na konci predošlej iterácie. Plánované a očakávané výstupy našej práce sme ale často nedefinovali presne a jasne, kvôli čomu vznikali nejasnosti medzi vedúcim a nami. Výsledkom toho bolo, že vedúci práce nás pochopil inak a očakával, že na konci iterácie uvidí iný výstup ako sme mu v skutočnosti predstavili. V jednej z prvých iterácií sme do plánu zaradili úlohu s popisom „Návrh zadaní, projektov a odovzdania riešení“. My sme touto úlohou mysleli jednoduchú prvú verziu návrhu UI hlavných obrazoviek a konceptuálny návrh, ktorým by sme si vyjasnili, ako chceme projekty a ich zadania a odovzdávanie riešiť. Vedúci práce to ale samozrejme takto nepochopil a nevedel, čo si má pod tým predstaviť. Z chýb na začiatku sme sa ale poučili a v ďalších iteráciach sme prišli s jasnejšie definovaným plánom a úlohami. Plán na poslednú iteráciu bol napríklad nasledovný:

- Napísať modelový jednotkový test na vytvorenie odovzdania
- Nakonfigurovať GitLab *pipeline*, aby sa tento test spustil a aby bol výstup testu v JUnit a vďaka tomu viditeľný v *Merge Requeste* v systéme GitLab
- Zostaviť Docker *image*
- Po nasadení rozchodiť Sentry
- Pridať zobrazenie príloh zadania k detailnému zobrazeniu projektu
- Unifikovať už použité komponenty na frontende
- Vytvoriť dokumentáciu pre BI-SP1 tím, ktorý s nami bude od februára spolupracovať vrátane inštrukcií pre prvotné stiahnutie a spustenie projektu, vysvetlenia dohodnutých pravidiel a použitého verzovacieho modelu

Ďalšou agilnou praktikou, ktorú sme s tímom aplikovali bol častý refaktoring kódu ako backendu tak aj frontendu. Dôvodom bolo väčšinou skvalitnenie kódu a zvýšenie prehľadnosti kódu.

Okrem vyššie spomenutých praktík agilného vývoja sme s tímom mali v úmysle vyskúšať aj odhadovanie úloh *story pointami*, ktoré definujem vyššie v časti 1.1.3. Hlavnou výhodou vykonávania odhadov pomocou *story pointov* je podpora konsenzu členov tímu o náročnosti jednotlivých úloh. Ďalším dôvodom, prečo agilné softwarové tímy odhadujú úlohy pomocou *story pointov*, je lepšie plánovanie iterácii a cieľov celého tímu. Zhodli sme sa ale na tom, že v našom prípade to tímu a projektu neprinesie veľa výhod, pretože v našom prípadne sa jedná hlavne o individuálnu prácu na oddelených častiach projektu každého člena tímu. Malo by to ale pravdepodobne zmysel v predmetoch BI-SP1 a BI-SP2 alebo pri vývoji MVP, kde práca jednotlivých členov tímu ovplyvňuje prácu celého tímu.

5.2 Verzovací model

Verzovací model, ktorý sme si s tímom zvolili vychádza primárne z Git Flow, ktorý definujem v časti 1.2.1. V projekte teda existujú dve hlavné vetvy – *master* a *develop*. Vetva *master* slúži na uchovávanie produkčných verzií. Vetva *develop* je hlavnou vývojovou vetvou, v ktorej sa integrujú ostatné vetvy pridávajúce nové funkcionality alebo opravujúce chyby. Pri nasadzovaní novej verzie na produkčné prostredie sme využívali tzv. *release* vetve, ktoré vznikli z vetve *develop* a boli zlúčené do vetví *master* aj *develop*.

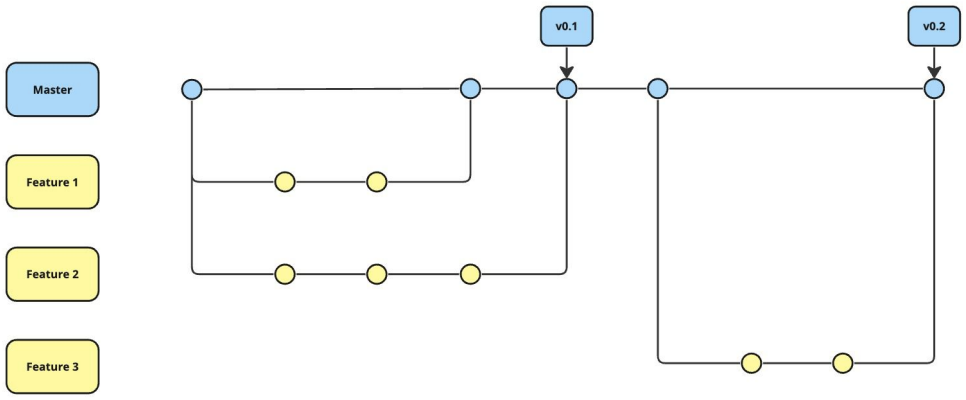
Dôvodom pre výber tohto verzovacieho modelu bolo primárne to, že sa podobný model používa aj na súčasnom systéme. Boli sme naň zvyknutí a vedeli sme s ním efektívne pracovať. Pôvodný autor Ing. Tomáš Pavlůsek tento model

pre súčasný systém zvolil z dôvodu jeho pozitívnych skúseností zo zamestnania. [45]

V jednej z iterácií na začiatku letného semestra sme sa ale s tímom rozhodli zvolený model značne upraviť. Dôvodom pre toto rozhodnutie bolo hlavne to, že sme v tom období všetci členovia tímu často dodávali nové funkcionality, čo znamenalo časté nasadzovanie na testovacie prostredie. Komplexný Git Flow ale na časté nasadzovanie nových verzií softwaru nie je vhodný, ako uvádzam aj v časti 1.2.1.

Novozvolený model, na ktorý sme prešli, vychádza z GitHub Flow. Ako uvádzam v časti 1.2.2, GitHub Flow odstraňuje zložitosť Git Flow, vďaka čomu je vhodný práve pre menšie tímy a aplikácie. Práve vďaka spomínanej jednoduchosti umožňuje časté nasadzovanie nových verzií, nerieši ale problém viacerých prostredí, kvôli čomu sme sa rozhodli pre účely nášho projektu GitHub Flow čiastočne modifikovať.

V projekte teda existuje hlavná vetva *master*, ktorá slúži na uchovávanie otestovaného a skontrolovaného kódu pripraveného na nasadenie na produkciu. Vo vetve *master* prebieha aj integrácia ostatných vetví, z vetvy *master* sú teda vytvárané tzv. *feature* vetvy, ktoré sú opäť v rámci GitLab *Merge Requestu* zlúčené do vetvy *master*. Kód zlúčený do vetvy *master* je vďaka CI/CD (Continuous integration and continuous delivery/deployment[1]) automaticky nasadený na testovacie prostredie. Nasadenie na produkčné prostredie bude v budúcnosti prebiehať commitom vo vetvi *master* označeným tagom. Upravený model, na ktorý sme prešli je vidieť na diagrame na obrázku 5.1.



Obr. 5.1 Zvolený verzovací model vychádzajúci z GitHub flow

V ďalších iteráciách sa podľa môjho názoru zmena verzovacieho modelu osvedčila, pracovalo sa mi totiž na projekte pohodlnejšie a efektívnejšie. Je ale možné, že v budúcnosti kedy nebudú často vznikať nové verzie projektu a kedy

[1] V preklade „Neustála integrácia a neustále doručovanie a nasadzovanie“

bude projekt zložitejší, bude výhodnejšie prejsť opäť na Git Flow alebo minimálne na nejakú jednoduchšiu modifikáciu Git Flow.

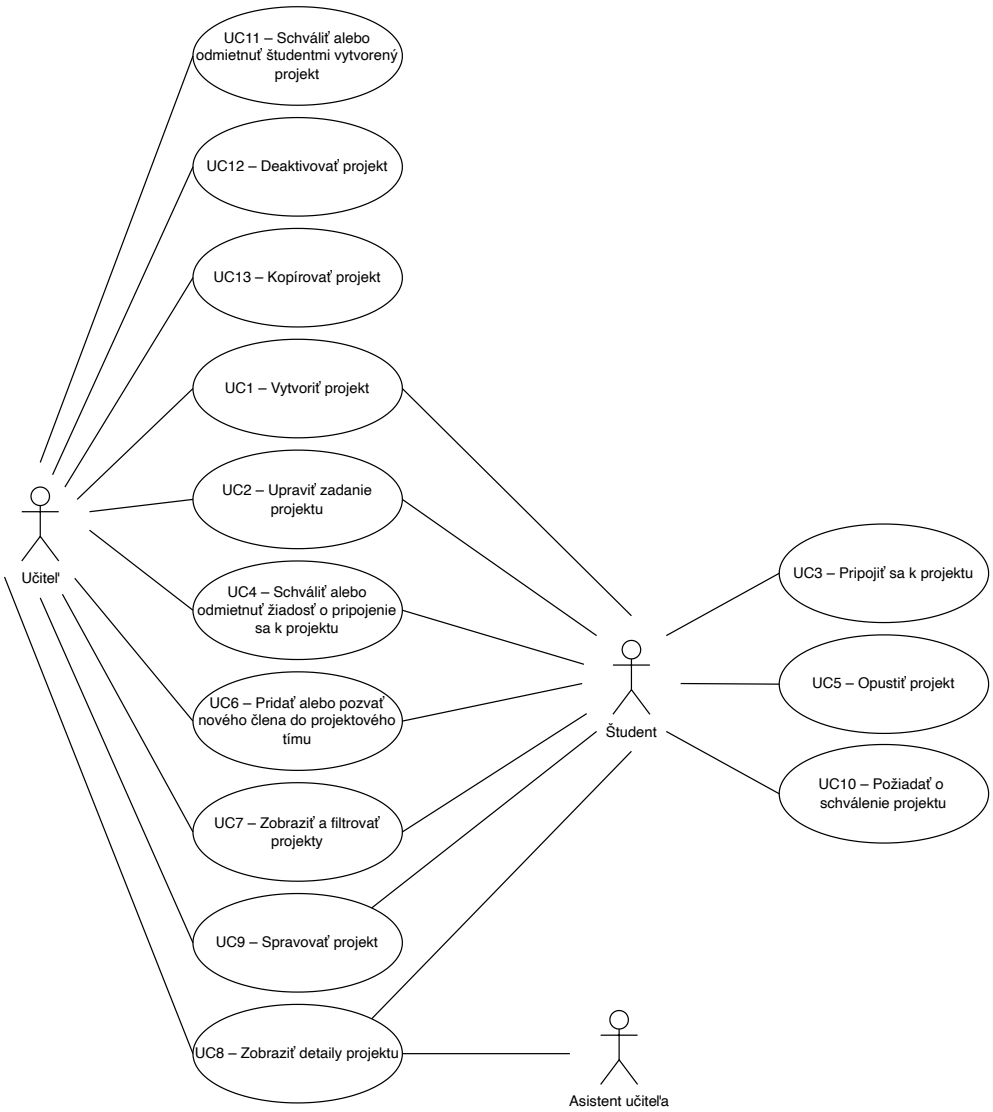
5.3 Model prípadov použitia

Cieľom analýzy požiadaviek je vytvoriť si systematickú predstavu o tom, čo klient a užívatelia skutočne potrebujú. Najčastejšie používanou technikou na skúmanie a analýzu funkčných požiadaviek je model prípadov použitia. Prípad použitia (UC z anglického „Use Case“) opisuje postupnosť interakcií medzi systémom a aktérom, ktoré vedú k nejakému výsledku. Aktér je osoba (niekedy aj čas alebo iný software či hardware), ktorá interaguje so systémom s cieľom vykonať UC. Je nesprávne domnievať sa, že všetci užívatelia softwarového systému patria do monolitickej skupiny s podobnými charakteristikami a potrebami. V praxi väčšinu produktov používajú rôznorodí užívatelia s rôznymi očakávaniami a cieľmi. Je preto dobré zoskupiť užívateľov do niekoľkých odlišných tried na základe rozdielov, ako sú ich prístupové oprávnenia alebo úrovne zabezpečenia, úlohy, ktoré vykonávajú, a funkcionality, ktoré používajú. [62, kap. 6 a 8]

UC väčšinou pozostáva z názvu, popisu, aktérov, ktorí daný UC môžu v systéme vykonať, vstupných a výstupných podmienok vykonania a hlavných a alternatívnych scenárov. Pre lepšiu vizualizáciu je dobré modelovať UC pomocou UML (Unified Modeling Language) diagramu prípadov použitia. [62, kap. 8] Je ale nutné myslieť na to, že pri práci na analýze požiadaviek je stále najdôležitejšia komunikácia s užívateľmi a zákazníkmi, ktorí zvyčajne softwarovému inžinierstvu nerozumejú a nepoznajú teda ani jazyk UML. [64, kap. 2]

5.3.1 Diagram modelu prípadov použitia

Jednotlivé UC týkajúce sa tímov a tímových projektov som prehľadne zachytil diagramom na obrázku 5.2. Okrem diagramu prípadov použitia nám ale pri analýze požiadaviek pomáhajú aj iné UML techniky. Konceptuálny model môže byť dobrým spôsobom pre vytvorenie precízneho slovníku pojmov. Diagramy aktivít pomáhajú vizualizovať, ako sa software a užívatelia vzájomne ovplyvňujú v rámci zložitých procesov. Okrem toho sú vhodné pre lepšie zobrazenie kontextu prípadov užitia a detailov komplikovaných prípadov užitia. Stavové diagramy zjednodušujú pochopenie komplexných stavov entít a udalostí, ktoré tieto stavy menia. [64, kap. 2]



Obr. 5.2 Diagram modelu prípadov použitia

5.3.2 Zoznam prípadov použitia

Nižšie uvádzam textový popis, ktorý bližšie špecifikuje jednotlivé prípady použitia, a scenáre niektorých vybraných prípadov použitia.

UC1 – Vytvoriť projekt

Aktéri: Učiteľ, študent
Popis: Užívateľ vie vytvoriť projekt. Projekt reprezentuje tím pracujúci na ne-

jakom zadaní so zodpovedným učiteľom, jeho asistentmi a kontrolnými bodmi, ku ktorým vie tím odovzdávať svoju prácu, ktorú vie následne učiteľ hodnotiť.

Hlavný scenár:

1. Užívateľ klikne na tlačítko tvorby projektu.
2. Systém zobrazí formulár tvorby projektu.
3. Užívateľ vyplní potrebné údaje, vyberie zadanie alebo vytvorí nové a pridá alebo pozve členov tímu. Vyučujúci si vie v rámci tvorby projektu rovno zvoliť aj svojich asistentov.
4. Systém vytvorí nový projekt s novým tímom, prípadne aj nové zadanie.

UC2 – Upraviť zadanie projektu

Aktéri: Učiteľ, študent
Popis: Učiteľ aj študent môžu upraviť nimi vytvorené zadanie vrátane názvu, popisu a príloh.

UC3 – Pripojiť sa k projektu

Aktéri: Študent
Popis: Študent vie požiadať o pripojenie sa k projektu. Po prijatí žiadosti sa stáva členom tímu priradeného k projektu. Do prijatia žiadosti vie žiadosť zrušiť. Študent sa nevie pripojiť k projektu bez podania žiadosti.

UC4 – Schváliť alebo odmietnuť žiadosť o pripojenie sa k projektu

Aktéri: Učiteľ, študent
Popis: Po odoslaní žiadosti o pripojenie sa k projektu, ju vedia schváliť alebo odmietnuť ostatní členovia tímu (v prípade, že je projekt spravovaný študentmi) alebo učiteľ zodpovedný za projekt (v opačnom prípade).

UC5 – Opustiť projekt

Aktéri: Študent
Popis: Študent vie buď automaticky opustiť projekt alebo požiadať o opustenie projektu, záleží na stave projektu a konfigurácii.

UC6 – Pridať alebo pozvať nového člena do projektového tímu

Aktéri: Učiteľ, študent
Popis: Členovia tímu vedia do tímu pozvať iného študenta daného predmetu. Ak študent túto pozvánku príjme, stáva sa členom daného tímu. Do prijatia pozvánky ju môžu členovia tímu zrušiť. Ak tím vypracováva nejaké zadanie v rámci projektu, môže učiteľ do tohto tímu automaticky pridať študenta daného predmetu.

UC7 – Zobraziť a filtrovať projekty

Aktéri: Učiteľ, študent
Popis: Užívatelia si vedia zobraziť zoznam im relevantných projektov. Pre garanta predmetu budú v budúcnosti relevantné všetky projekty v predmete, pre učiteľa jeho projekty, pre študenta všetky viditeľné projekty. Užívatelia vedia tento zoznam projektov aj filtrovať podľa názvu, učiteľa, študenta v tíme.

UC8 – Zobraziť detaily projektu

Aktéri: Učiteľ, študent, asistent
Popis: Užívateľ vie zobraziť detaily o projekte vrátane zadania, tímu a kontrolných bodov.

UC9 – Spravovať projekt

Aktéri: Učiteľ, študent
Popis: Užívateľ vie spravovať projekt vrátane úpravy zadania a pridávania členov tímu. Učiteľ vie navyše odobrať členov tímu, pridať a odobrať svojich asistentov a nastavovať kontrolných bodov.

UC10 – Požiadať o schválenie projektu

Aktéri: Študent
Popis: Projekt je v prípade vytvorenia študentmi aj spravovaný študentmi až do schválenie tohto projektu učiteľom. Členovia tímu vedia preto požiadať o schválenie projektu. V rámci žiadosti o schválenie projektu vedia študenti učiteľovi poslať aj textovú správu. Je nutné, aby bol projekt schválený ešte pred prvým odovzdaním. Po schválení projektu tento projekt už spravuje zodpovedný učiteľ.

UC11 – Schváliť alebo odmietnuť študentmi vytvorený projekt

Aktéri: Učiteľ
Popis: Učiteľ vie buď prijať alebo odmietnuť žiadosť o schválenie projektu vytvoreného študentmi. V rámci rozhodnutia o schválení vie študentom poslať aj textovú správu. V prípade schválenia projektu tento projekt namiesto študentov spravuje zodpovedný učiteľ. V prípade odmietnutia žiadosti o schválenie projektu

UC12 – Deaktivovať projekt

Aktéri: Učiteľ
Popis: Učiteľ vie deaktivovať projekt, čím sa stane neviditeľným pre študentov. Garant predmetu bude v budúcnosti vedieť okrem svojich projektov deaktivovať ľubovoľný projekt v predmete.

UC13 – Kopírovať projekt

UC zatiaľ nebude implementovaný.
Aktéri: Učiteľ
Popis: Učiteľ bude v budúcnosti vedieť skopírovať projekt do druhého predmetu vrátane zadania, odkazu na GitLab repozitár a tímu. Kopírovanie projektu bude konfigurovateľné podľa predstáv učiteľa, učiteľ si bude môcť napríklad vybrať, že chce skopírovať len zadanie bez tímu. Pri kopírovaní projektu aj s tímom budú skopírovaní len tí členovia tímu, ktorí majú zapísaní predmet, do ktorého sa projekt kopíruje. Nutno podotknúť, že študenti nebudú do projektu pridaný automaticky, ale pravdepodobne obdržia pozvánku a notifikáciu.

5.3.3 Mapovanie funkčných požiadaviek na prípady použitia

Na záver overím pokrytie všetkých funkčných požiadaviek pomocou mapovania funkčných požiadaviek na prípady použitia. Cieľom mapovania je skontrolovať, či sme analyzovali a zachytili všetky požiadavky. Pokrytie jednotlivých funkčných požiadaviek prípadmi použitia som zachytil tabuľkou 5.1.

	FP1	FP2	FP3	FP4	FP5	FP6	FP7	FP8	FP9
UC1	✓	✓	✓						
UC2	✓								
UC3		✓	✓						
UC4		✓	✓						
UC5		✓	✓						
UC6		✓	✓						
UC7					✓				
UC8			✓						
UC9		✓	✓				✓	✓	
UC10				✓					
UC11				✓					
UC12						✓			
UC13									✓

Tabuľka 5.1 Tabuľka pokrytia funkčných požiadaviek

5.4 Architektúra a technológie

Pri vývoji systému SOS bolo a je potrebné riešiť jeho nedostatky, ktoré vyplynuli z reálnej prevádzky a využívania tohto systému vo viacerých predmetoch na FIT ČVUT. Tieto nedostatky bližšie predstavujem v rámci analýzy súčasného systému SOS v kapitole 3. Jednou z možností, ako systém SOS ďalej vyvíjať bolo pokračovať vo vývoji súčasnej monolitickej viac-stránkovej aplikácie vo frameworku Django a šablónovacom systéme, ktorý tento framework ponúka. S tímom sme sa rozhodli pre zmenu architektúry systému na oddelenú jedno-stránkovú frontend aplikáciu a backend aplikáciu, ktorá frontendu poskytuje REST API. Dôvodov bolo viacero vrátane jednoduchšej udržateľnosti a rozšíriteľnosti systému do budúcna a jednoduchšieho riešenia známych nedostatkov týkajúcich sa neintuitívneho a neprehľadného UI, ktoré by bolo v súčasnej architektúre náročnejšie zlepšiť. Výhodou SPA a oddeleného backendu a frontendu v kontexte systému SOS je, že bude možné podporovať bohaté funkcionality na strane klienta, ktoré nevyžadujú opätovné načítanie stránky, počas toho, čo užívatelia s aplikáciou pracujú alebo prechádzajú medzi stránkami aplikácie. SPA sa zvyčajne načítavajú rýchlejšie, načítavajú dáta na pozadí a na jednotlivé akcie užívateľov reagujú rýchlejšie, pretože úplné načítanie stránky je zriedkavé. Jedno-stránkové aplikácie teda nájdu svoje uplatnenie, ak požiadavky aplikácie zahŕňajú bohatú funkcionalitu, ktorá presahuje možnosti typických HTML formulárov. [36]

Po rozhodnutí zmeniť architektúru systému SOS sme s tímom mali viacero možností týkajúcich sa výberu vhodných technológií a všeobecne toho, ako budeme túto zmenu uskutočňovať. Vo všeobecnosti asi platí, že výber konkrétnej

technológie je často vec osobných preferencií a skúseností. V tejto sekcii sa ale pokúsim uviesť, aké možnosti sme mali a akú sme si z akých dôvodov nakoniec vybrali. Následne bližšie predstavím zvolené technológie a vhodným diagramom aj architektúru nového systému.

5.4.1 Frontend

Pre novú jedno-stránkovú frontend aplikáciu sme sa s tímom rozhodli použiť programovací jazyk TypeScript a viaceré JavaScriptové a CSS frameworky a knižnice vrátane Vue.js, Nuxt, Nuxt UI a Tailwind CSS, ktoré v tejto časti čitateľovi bližšie predstavím.

Pri výbere týchto technológií sme s tímom ale nevykonali žiadnu precíznu analýzu všetkých možností. Vychádzali sme primárne z našich skúsenosti a subjektívnych názorov, zo skúsenosti vedúceho práce Ing. Jiřího Hunky napríklad z DBS portálu používaného v predmete BI-DBS (Databázové systémy) a z porovnania tzv. learning curve alebo jednoduchosti danej technológie a času potrebného pre jednoduché pochopenie a prvotné vniknutie do danej technológie. Nikto z nášho trojčlenného tímu totižto nemal pred vývojom tejto aplikácie pokročilejšie znalosti a bohatšie skúsenosti s frontendovými technológiami. Okrem toho môže byť jednoduchosť frontend technológie výhodná aj v budúcnosti, kedy na projekte budú pracovať študenti predmetov BI-SP1 a BI-SP2, z ktorých pravdepodobne väčšina tiež nebude mať pokročilé znalosti frontendových technológií. Podľa študijného plánu špecializácie *Softwarové inženýrství* v bakalárskom štúdiu na FIT ČVUT totiž študenti nemajú žiadny povinný predmet, ktorý by sa podrobnejšie venoval vývoji frontendových aplikácií v JavaScripte a JavaScriptových frameworkoch. [65] Študenti špecializácie *Webové inženýrství* majú takéto predmety povinné dva – BI-PJS (Programování v jazyku Javascript) a BI-TWA (Tvorba webových aplikací). Tí ale zase nemajú povinné predmety BI-SP1 a BI-SP2, [66] v ktorých je práve priestor pre ďalší rozvoj systému SOS.

5.4.1.1 Vybrané technológie

Typescript

TypeScript (TS) je silno typovaný programovací jazyk, ktorý vychádza z jazyka JavaScript, ale pridáva ďalšiu syntax primárne pre podporu statického typovania. JavaScript (JS) je dynamicky typovaný, interpretovaný skriptovací a programovací jazyk často označovaný ako tzv. „jazyk webu“, pretože je primárne používaný pre vývoj webových stránok. [67, 68]

Dôvodom pre výber TypeScriptu namiesto JavaScriptu bolo zvýšenie čitateľnosti a udržateľnosti kódu. Keďže sa bude jednať o jedno-stránkovú frontend aplikáciu s určitou časťou logiky systému aj na frontende, potrebujeme, aby okrem backend aplikácie bola udržateľná aj frontend aplikácia. Výber TypeScriptu s typovou kontrolou nám totižto môže pomôcť oveľa skôr nájsť chyby a dodať spoľahlivejší a udržateľnejší kód.

Vue.js

Vue.js je JavaScriptový frontend framework pre tvorbu užívateľských rozhraní webových aplikácií ako aj tých jedno-stránkových. Stavia na štandardných webových jazykoch HTML, CSS a JavaScript a poskytuje deklaratívny programovací model založený na komponentoch, ktoré pomáhajú efektívne vyvíjať aplikácie akejkoľvek zložitosti a veľkosti. [69]

Ako naznačujem vyššie, jedným z dôvodov pre výber frameworku Vue.js bola jeho jednoduchosť. Ako uvádzajú viaceré porovnania troch momentálne najpoužívanejších frontend frameworkov – Angular, React a Vue.js – Vue.js je jednoznačne najjednoduchší na pochopenie a okamžité používanie aj pre začiatočníkov. [70, 71]

Ďalším dôvodom bola kladná skúsenosť kolegu z tímu Jana Mrázka zo zamestnania a vedúceho práce Ing. Jiřího Hunky z DBS portálu. Frontend DBS portálu je totiž postavený tiež na frameworku Vue.js s grafickou knižnicou komponent Quasar. Výhodou tohto výberu je teda aj možnosť zdieľania skúseností a znalostí všetkých vývojárskych tímov pracujúcich na vývoji oboch portálov.

Nuxt

Nuxt je frontendový framework postavený na Vue.js. Okrem samotného frameworku sme sa rozhodli používať aj Nuxt UI. Ide o knižnicu komponentov užívateľského rozhrania, ktorej cieľom je vývojárom poskytnúť všetky potrebné prvky užívateľského rozhrania vrátane komponentov, ikon alebo farieb pre rýchly vývoj frontendových webových aplikácií vo frameworku Nuxt. [72, 73]

Ako uvádzam vyššie, DBS portál je momentálne postavený na frameworku Vue.js s knižnicou Quasar. Prechod DBS portálu do frameworku Nuxt, ktorý je postavený práve na Vue.js by mohol v budúcnosti priniesť ešte viac výhod ako použitie samotného Vue.js v oboch aplikáciach. Okrem možnosti zdieľať skúsenosti by tento prechod mohol priniesť ďalšie výhody pre vývojárov, ale aj pre koncových užívateľov, pretože by sme mohli zdieľať spoločné komponenty

a prvky užívateľského rozhrania, čo by vývojárom šetrilo čas a užívateľom prinieslo komfort vo forme podobného užívateľského rozhrania.

Tailwind CSS

Tailwind je CSS framework, ktorý vývojárom ponúka množstvo tzv. *utility* CSS tried, ktoré je možné ľubovoľne skladať dohromady a použiť tak pre definíciu štýlu ľubovolného prvku. Iné podobné CSS knižnice ako napríklad Bootstrap fungujú odlišne – poskytujú sadu preddefinovaných tried pre konkrétne prvky ako napríklad tlačítka alebo tabuľky. Rozdiel je teda v tom, že narozdiel od sady tried pre konkrétne komponenty a prvky, Tailwind prináša triedy vytvorené okolo konkrétneho štýlu ľubovolného prvku, napríklad farby alebo veľkosti textu. [43, 74]

5.4.2 Backend

Pri výbere vhodných backend technológií sme mali primárne dve možnosti. Jednou z nich bolo pokračovať vo vývoji súčasnej aplikácie vo frameworku Django. Tento prístup by znamenal zachovanie doménového modelu a väčšiny biznis logiky súčasnej aplikácie. Bolo by ale stále potrebné navrhnúť API pre nový frontend a toto API vytvoriť pomocou frameworku Django REST[2], čo je rozšírenie frameworku Django určené práve pre tvorbu REST API. Výhodou tohto prístupu by bola určite rýchlosť vývoja, pretože by sme mohli zachovať podstatnú časť logiky systému, nemuseli by sme navrhovať nový doménový model a rozmýšľať nad všetkými možnými krajnými prípadmi, ktoré môžu nastať. Mohli by sme sa teda sústrediť primárne na návrh API, návrh nového UI a implementáciu frontendu a riešiť tak konkrétne nedostatky súčasného systému.

Druhou možnosťou bolo vytvoriť úplne novú aplikáciu aj s novým backendom pomocou iných technológií. Nová backend aplikácia by ale z tej súčasnej samozrejme vychádzala. Tento prístup by ale okrem návrhu API, návrhu nového UI a implementácie frontendu zahŕňal aj kompletne nový návrh databázového a doménového modelu a implementáciu doménových tried a celej biznis logiky novej aplikácie. Je zrejmé, že tento prístup by bol časovo oveľa viac náročnejší a bolo by takmer nemožné do začiatku zimného semestra 2024/2025 vytvoriť v trojčlennom tíme kompletne nový systém so všetkými potrebnými funkcionalitami pre predmety, ktoré súčasný systém využívajú.

Na druhú stranu fakt, že by v tomto prípade bolo nevyhnutné celý systém navrhnúť znova považujem čiastočne aj za výhodu. Ako uvádzam v časti 1.1.3,

[2] `https://www.django-rest-framework.org/`

jedným z dôvodov pre vznik a zároveň aj úspech agilných metodík vývoja softwaru, aplikujúcich iteratívny vývoj, je to, že pri tvorbe softwarových produktov stále vznikajú nové požiadavky, užívatelia prichádzajú s novými návrhmi na zlepšenie a odhaľujú nové nedostatky. Tento fakt som si aj ja sám overil pri práci na tomto projekte – počas rozhovorov s užívateľmi, analýze UI súčasného systému, validácie existujúcich požiadaviek na súčasný systém a zbere tých nových. Tieto nové požiadavky, návrhy na zlepšenie a nedostatky, ktoré boli zistené po vyše dvoch rokoch produkčného používania súčasného systému, by sme mohli zahrnúť v návrhoch, čím by sme určite vytvorili do budúcna udržateľnejší a rozšíriteľnejší systém, čo je jednou z nefunkčných požiadaviek, ako uvádzam v časti 4.2.4. Prvý prístup nám naopak neumožňuje rozsiahlo zasahovať do jadra aplikácie a meniť kľúčovú biznis logiku či základ doménového modelu.

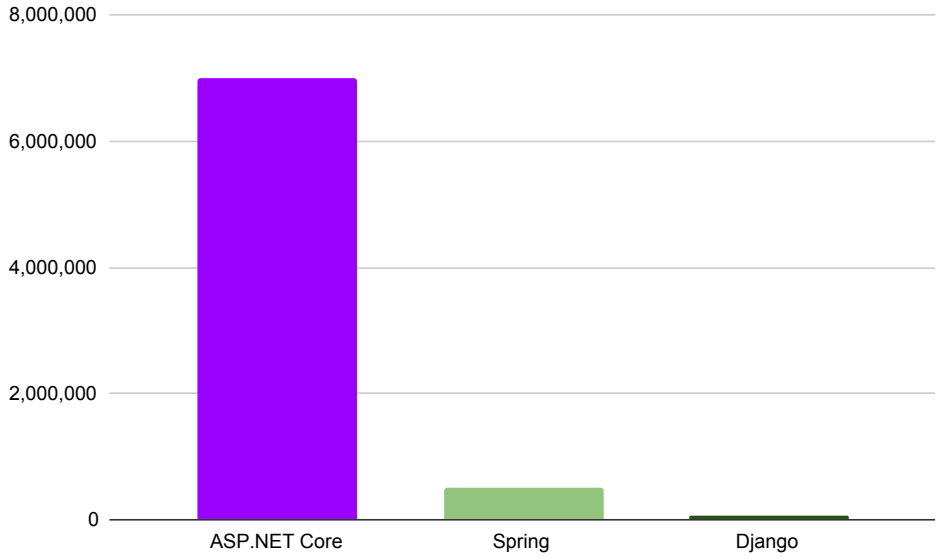
Ďalšou výhodou tohto prístupu je možná voľba nových technológií na základe určitých výhod a nevýhod jednotlivých technológií. Ako uvádzam v časti 4.2.4, jednou z ďalších nefunkčných požiadaviek kladených na nový systém je škáloveteľnosť aplikácie. Aplikácia musí teda zvládať rastúce množstvo záťaže bez výrazného zníženia výkonu. Tejto nefunkčnej požiadavke vieme vyhovieť viacerými spôsobmi, napríklad optimalizáciou zložitejších častí kódu, optimalizáciou databázových dotazov napríklad efektívnym používaním indexov alebo rozložením zátaže na viaceré servery. Ďalšou možnosťou je ale aj výber vhodných technológií a návrh architektúry. Ako uvádzam v časti 2, návrh architektúry aplikácie, s čím úzko súvisí aj výber vhodných technológií, sa čiastočne týka splnenia nefunkčných požiadaviek systému, keďže zlá architektúra významne prispieva k nesplneniu niektorých nefunkčných požiadaviek systému.

V prípade vývoja novej backend aplikácie sme s tímom zvažovali Javovský webový framework Spring, pretože sa povinne vyučuje v predmete BI-TJV na FIT ČVUT, ktorý je úvodom do vývoja podnikových webových aplikácií s viacvrstvovou architektúrou [75] a ASP.NET Core kvôli našej domnienke o vysokej efektivite a doporučeniu vedúceho práce, ktorý má s touto technológiou kladné skúsenosti zo svojej bohatej praxe.

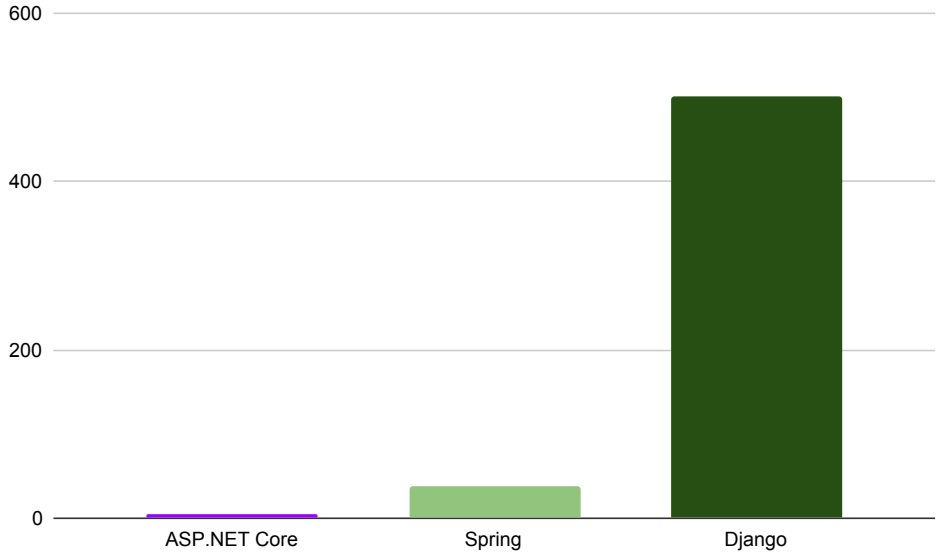
Ako je vidieť na grafoch na obrázkoch 5.3 a 5.4 od nezávislých TechEmpower[3] meraní a ich oficiálnych výsledkov testov, ktoré porovnávajú frameworky pre vývoj webových aplikácií pri rôznych úlohách, domnienka o vysokej efektivite frameworku ASP.NET Core sa potvrdila a zo spomínaných technológií sa naozaj jedná o tú najefektívnejšiu, pričom Django, na ktorom je postavený súčasný systém, je na tom zo spomínaných troch webových frameworkov najhoršie. V oboch testoch server odpovedal na HTTP žiadosť klienta jednoduchou textovou správou „Hello, World“. [76]

Práve z vyššie uvedených dôvodov sme sa s tímom rozhodli pre vývoj novej

[3] `https://www.techempower.com/benchmarks/`



Obr. 5.3 Porovnanie webových frameworkov - celkový počet spracovaných žiadostí za sekundu (čím vyššie, tým lepšie). Dáta získané z TechEmpower meraní.



Obr. 5.4 Porovnanie webových frameworkov - priemerné oneskorenie v ms (čím nižšie, tým lepšie). Dáta získané z TechEmpower meraní.

backend aplikácie vo frameworku ASP.NET Core aj za cenu pomalšieho vývoja oproti vývoji súčasnej aplikácie vo frameworku Django. Ďalšou výhodou výberu platformy .NET je zjednotenie technológií s DBS portálom, ktorého

backend je síce primárne napísaný v jazyku PHP a frameworku Nette, no momentálne je tiež v procese obmeny technológií a jeho nová verzia, ktorá sa ešte len vyvíja, je vyvíjaná práve na platforme .NET. Ako spomínam vyššie pri výbere frontend technológií, toto zjednotenie môže v budúcnosti priniesť portálom SOS a DBS viaceré výhody vrátane možnosti využívania viacerých spoločných projektov a vlastných knižníc a zdieľania skúseností a znalostí členov tímov vyvíjajúcich oboje aplikácie. Využitie spoločných projektov dokonca nie je záležitosťou budúcnosti, na projekte *Filter Bundle*[4] už začal pracovať tím študentov predmetu BI-SP1 v letnom semestri 2023/2024. Jedná sa o knižnicu zjednodušujúcu filtrovanie a stránkovanie. Ďalšie spoločné projekty budú tiež v rovnakej GitLab skupine *Common Projects*[5] založenej Bc. Maxmom Hejdom.

5.4.2.1 Vybrané technológie

C#

C# je vysoko-úrovňový, staticky a silno typovaný programovací jazyk. Je založený na objektovo orientovaných princípoch, ale obsahuje aj mnoho prvkov z iných paradigiem, v neposlednom rade aj z funkcionálneho programovania. Síce sa jedná o vysoko-úrovňový jazyk s prvkami ako napríklad dátovo orientované *record* triedy, C# podporuje kvôli vysokej efektivite aj nízko-úrovňové prvky bez nutnosti písania nebezpečného kódu ako napríklad ukazovatele na funkcie. [77, 78]

.NET

.NET je open-source aplikačná platforma podporovaná spoločnosťou Microsoft pre vývoj viacero druhov aplikácií. Vie spúšťať programy napísané vo viacerých jazykoch, pričom najpopulárnejším je práve jazyk C#. Poskytuje automatickú správu pamäte vďaka *garbage collectoru* (GC). Práve vďaka GC a prísnym jazykovým kompilátorom je platforma .NET a jazyk C# typovo a pamäťovo bezpečný. Podporuje asynchrónne programovanie a ponúka veľkú sadu knižníc so širokou funkcionalitou, ktoré boli optimalizované na výkon vo viacerých operačných systémoch a na viacerých architektúrach čipov. [79, 80]

ASP.NET Core

ASP.NET Core je webový framework postavený na .NET platforme. .NET je teda aplikačná a vývojárska platforma pozostávajúca z rôznych nástrojov,

[4] `https://gitlab.fit.cvut.cz/common-projects/filter-bundle`
[5] `https://gitlab.fit.cvut.cz/common-projects`

jazykov a knižníc na vývoj rôznych typov aplikácií. ASP.NET Core vychádza z tejto platformy a pridáva ďalšie nástroje a knižnice špeciálne pre vývoj webových aplikácií. [81, 82, 83]

Entity Framework Core

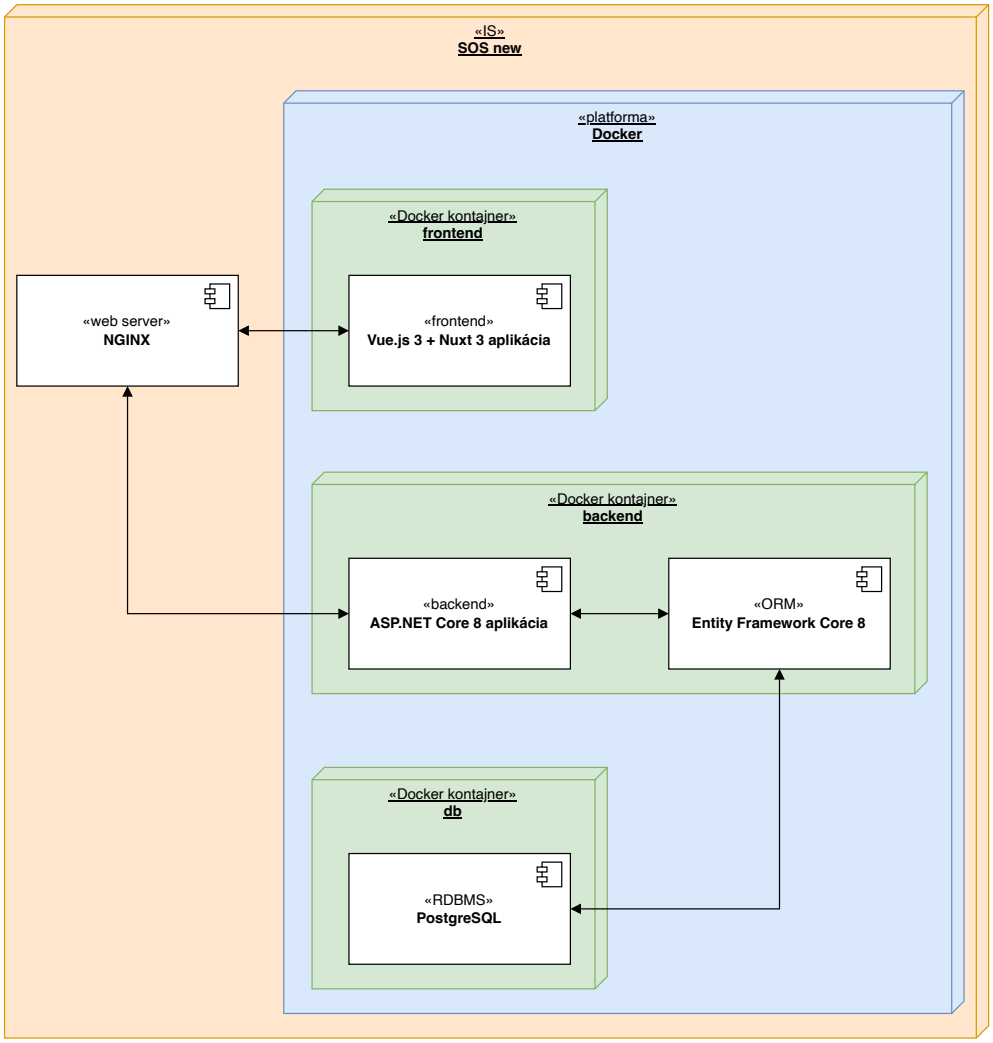
Entity Framework Core je tzv. ORM *(Object-Relational-Mapping)* framework alebo framework poskytujúci objektovo relačné mapovanie, ktorý umožňuje vývojárom vytvoriť vysoko-úrovňovú dátovú vrstvu aplikácií pomocou C# objektov a eliminovať potrebu veľkej časti SQL dotazov potrebného pre prístup k dátam. Vo všeobecnosti je ORM programátorská technika pre automatický prevod dát medzi objektami v objektovo orientovaných jazykoch a relačnou databázou. [84, 85]

5.4.3 Databáza

Výber vhodnej technológie pre perzistentné ukladanie dát bol asi najjednoduchší. S tímom sme len čiastočne zvažovali aj iné možnosti, rýchlo a jednohlasne sme sa zhodli na výbere objektovo relačného databázového stroja PostgreSQL, ktorý som čitateľovi predstavil už v časti 3.1, keďže sa používa aj v súčasnom systéme. PostgreSQL sme vybrali hlavne preto, že s ním máme všetci viacročné skúsenosti z komerčných projektov v zamestnaní a zo súčasného systému SOS. Okrem toho sa PostgreSQL používa aj v predmete BI-DBS vyučovanom v druhom semestri na FIT ČVUT, ktorý je v rámci bakalárskeho štúdia úvodom do sveta relačných databáz.

5.4.4 Architektúra nového systému

Nový systém sa teda bude skladať z dvoch samostatných aplikáci. Prvou aplikáciou bude backend aplikácia napísaná v jazyku C# s pomocou webového frameworku ASP.NET Core, ktorá bude frontend aplikácií poskytovať REST API. Návrhu samotného REST API sa venujem v časti 5.7. Pre objektovo relačné mapovanie medzi doménovými triedami v jazyku C# a relačnou databázov PostgreSQL budeme používať Entity Framework Core. Druhou aplikáciou bude jedno-stránková frontend aplikácia napísaná v programovacom jazyku TypeScript a frameworku Nuxt s použitím knižnice komponentov užívateľského rozhrania Nuxt UI a CSS frameworku Tailwind CSS. Backend a frontend aplikácie budú spolu komunikovať pomocou HTTP protokolu, na preposielanie HTTP žiadostí bude slúžiť webový server NGINX. Architektúru novej aplikácie najlepšie sumarizuje nasledujúci diagram na obrázku 5.5.



Obr. 5.5 Návrh architektúry nového systému

5.5 Definícia a návrh MVP

Po rozhodnutí oddeliť backend a frontend systému a vyvíjať kompletne nový systém aj s novým backendom, analýze funkčných požiadaviek a návrhu architektúry a voľbe technológií nového systému sme s tímom v jednej z prvých iterácií na začiatku zimného semestra čelili ďalšej výzve – definícii a návrhu MVP (Minimum Viable Product[6]). Vo všeobecnosti sa jedná o najjednoduchšiu verziu produktu, vytvorenú v prvých iteráciách, ktorú vieme ukázať zákazníkom prípadne malej časti koncových užívateľov s cieľom predstaviť a validovať nový produkt a získať spätnú väzbu, návrhy na zlepšenia a nové požiadavky,

[6] V preklade minimálny životaschopný produkt

na ktorých vieme pracovať v ďalších iteráciách vývoja. [86] V tejto sekcii čitateľovi vysvetlím dôvody, prečo sme sa rozhodli navrhnúť a implementovať MVP a ukážem, ako sme s tímom pri definícii a návrhu MVP postupovali.

5.5.1 Dôvody návrhu MVP

Definovať a navrhnúť MVP nového systému SOS sme rozhodli z viacerých dôvodov. V prvom rade je dôležité pripomenúť, že jedným z cieľov tejto práce je vyskúšať iteratívny vývoj softwaru a prácu v tíme. Koncept MVP čiastočne súvisí s agilným vývojom softwaru, ktorý na iteratívnom vývoji stavia, keďže cieľom MVP je predstavenie prvej vezie produktu zákazníkovi v prvých iteráciách vývoja a získanie spätnej väzby. Ďalším dôvodom je, že sme si s tímom chceli overiť, že sme sa vybrali správnou cestou. Chceli sme mať na konci zimného semestra 2023/2024 produkt, ktorý by sme vedeli ukázať vedúcemu práce a presvedčiť ho vďaka tomu o tom, že má zmysel pokračovať vo vývoji nového systému. Uvedomovali sme si, že v prípade, že by sme postupovali tradičným vodopádovým štýlom vývoja a vedúcemu práce by sme nový systém ukázali až na konci letného semestra 2023/2024, tak by sme počas celého vývoja nedostali žiadnu spätnú väzbu. Toto by ale mohlo zapríčiniť nesplnenie požiadaviek vedúceho práce a ďalších učiteľov, kvôli čomu by sa všetka naša odvedená práca mohla zahodiť.

5.5.2 Funkcionality MVP

Rozhodli sme sa, že náš MVP bude podporovať najprioritnejšie funkcionality všeobecného charakteru, nie potreby konkrétnych predmetov. MVP sme teda definovali na základe už existujúcich požiadaviek na súčasný systém, ktoré sme analyzovali už v letnom semestri 2022/2023 v predmete BI-SP1. Okrem toho sme sa zamysleli nad ďalšími možnými prípadmi použitia nového systému s cieľom vybrať najpotrebnejšie a najprioritnejšie funkcionality, ktoré by sme implementovali ešte v zimnom semestri.

V našom MVP sme užívateľov rozdeľovali na dve kategórie vo vzťahu k zadaniu – zadávateľ a riešiteľ. MVP sme navrhli, aby podporoval následujúce prípady použitia.

UC1 - Prihlásiť sa do aplikácie

Aktéri: Neprihlásený užívateľ
Popis: Neprihlásený užívateľ sa vie prihlásiť do aplikácie užívateľským menom a heslom.

UC2 - Vytvoriť zadanie

Aktéri: Prihlásený užívateľ
Popis: Prihlásený užívateľ vie vytvoriť zadanie, ktoré sa skladá z krátkeho zhrnutia a textového popisu.

UC3 - Priradiť zadanie riešiteľovi

Aktéri: Zadávateľ, riešiteľ
Popis: Zadávateľ vie priradiť vytvorené zadanie riešiteľovi. Riešiteľ si vie priradiť ľubovoľné zadanie z ponuky.

UC4 - Odovzdať riešenie

Aktéri: Riešiteľ
Popis: Riešiteľ vie odovzdat riešenie úlohy vo forme textového popisu.

UC5 - Spracovať odovzdané riešenie

Aktéri: Zadávateľ
Popis: Zadávateľ vidí odovzdané riešenie riešiteľa a vie ho potvrdiť.

5.6 Doménový model

V tejto sekcii čitateľovi predstavím návrhy doménového modelu. Keďže sme s tímom pracovali iteratívne, na návrhoch som pracoval vo viacerých iteráciách. Návrhy sa teda časom menili, čitateľovi preto predstavím návrh, ktorý som vytvoril pri každej novej verzii aplikácie. Podľa jednotlivých iterácií alebo verzií aplikácie bude členený aj text tejto sekcie na ďalšie podsekcie.

Nutno podotknúť, že budem uvádzať len návrhy, na ktorých som pracoval ja a ktoré sa týkajú môjho zamerania práce – tímy a tímové projekty. V prípade potreby zavedenia aj iných doménových oblastí, napríklad kontrolných bodov alebo užívateľov, nebudem tieto domény navrhovať detailne, budem ich používať len pre lepšie znázornenie vzťahu s doménami, ktoré súvisia s tímami a tímovými projektami.

Dovolím si čitateľa ešte upozorniť na to, že na začiatku textu o danej iterácií najprv uvediem, akým prípadom použitia som sa v danej iterácií venoval, po čom uvediem, aké zmeny som vykonal v doménovom modele, čo na záver každej sekcie ukážem na diagrame. Okrem toho si dovolím čitateľa ešte upozorniť na to, že iterácie v tejto sekcii sú myslené ako iterácie v kontexte návrhu

doménového modelu, teda ucelené časové obdobia, v ktorých som iteratívne upravil doménový model pre splnenie daných požiadaviek, nie ako dvojtýždňové iterácie v kontexte iteratívneho vývoja, ktorý sme s tímom pri vývoji aplikovali. Dôvodom je to, že sa napríklad stalo, že som sa návrhu doménového modelu, týkajúceho sa schvaľovania projektov, venoval aj spolu s návrhom API a užívateľského rozhrania a implementáciou až dve tieto dvojtýždňové iterácie, zatiaľ čo na konci tej druhej som už ale začal pracovať na návrhu doménového modelu týkajúceho sa ďalšej funkcionality.

5.6.1 MVP

V prvých iteráciách som sa venoval návrhu MVP a teda základnému splneniu najprioritnejších „Must Have“ požiadaviek a nasledujúcim prípadom použitia.

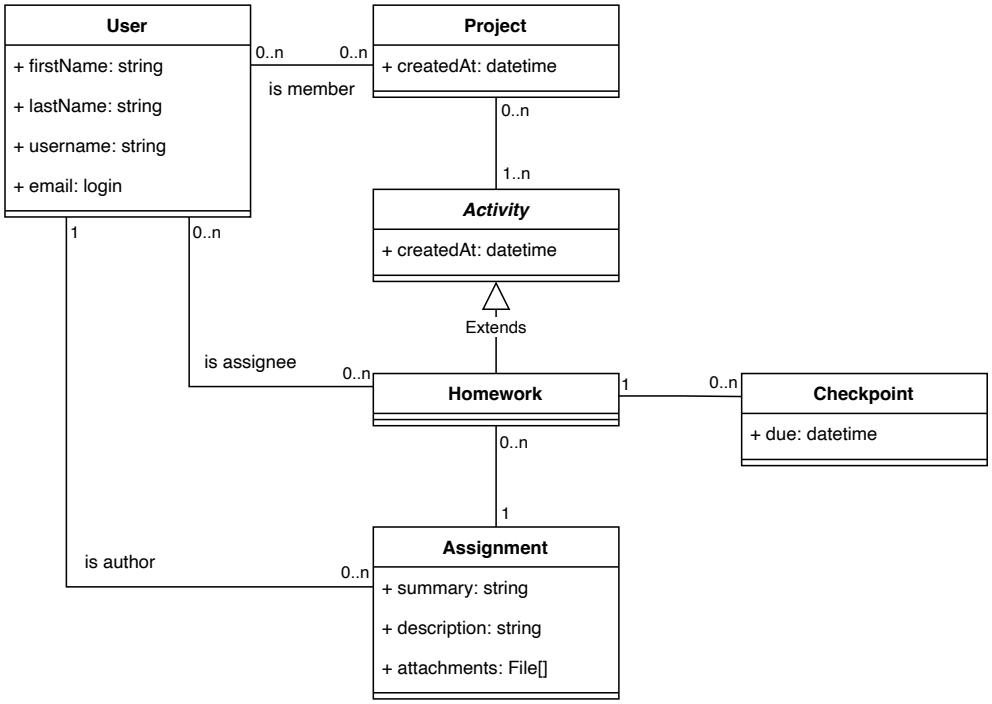
Prípady použitia:

- UC1 – Vytvoriť zadanie
- UC3 – Vytvoriť tím
- UC4 – Pripojiť sa k tímu
- UC9 – Vytvoriť projekt
- UC10 – Zobraziť a filtrovať projekty
- UC11 – Zobraziť detaily projektu

Pri návrhu MVP ešte pred samotnou implementáciou sme s tímom, s cieľom vytvoriť čo najuniverzálnejší a jednoducho rozšíriteľný systém, pracovali s nápadom vytvoriť doménu `Activity`. Išlo by o abstraktnú doménu zastrešujúcu nejaké zadanie a jeho riešenie. Mohlo by sa jednať napríklad o domácu úlohu pre jednotlivca, skupinový projekt, ročníkovú prácu, písomku alebo kvíz. Z tohto nápadu vychádzali nasledujúce dva moje návrhy doménového modelu, ktoré som v jednej z prvých iterácií predstavil tímu a vedúcemu práce.

Ako vidieť na obrázku 5.6, prvý návrh pracuje s myšlienkou, že by existovala doména `Project`, ktorá by reprezentovala nejakú veľkú skupinovú prácu. `Project` by sa skladal z viacerých aktivít alebo nejakých podúloh, ktoré by boli definované zadaním. Riešiteľmi týchto úloh by bola nejaká podmnožina tímu projektu. Riešenie jednotlivých úloh by sa odovzdávalo v kontrolných bodoch.

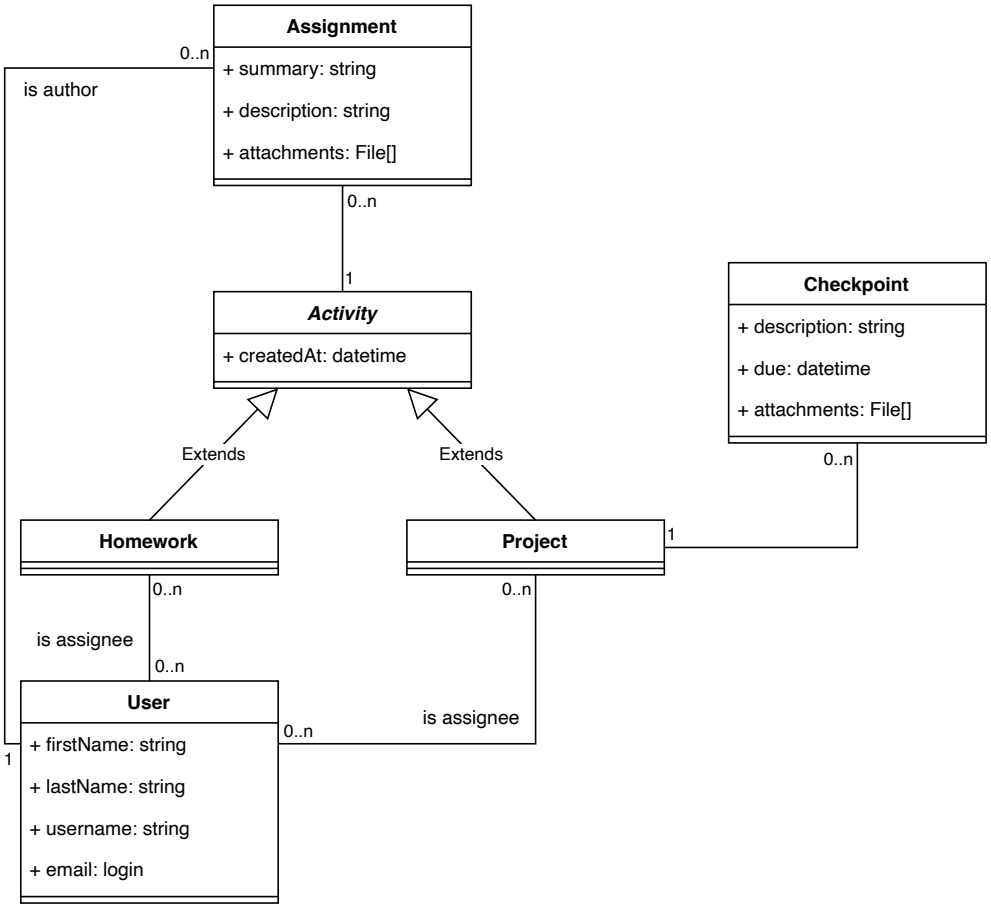
Na obrázku 5.7 je naopak vidieť druhý návrh, ktorého rozdiel oproti prvému spočíva hlavne v tom, že neexistuje žiadna entita reprezentujúca veľkú skupinovú prácu, ktorá by zoskupovala viacero rôznych typov aktivít. Pracuje skôr s myšlienkou, že všeobecná aktivita je definovaná zadaním a môže byť buď



Obr. 5.6 Prvý návrh doménového modelu pred implementáciou MVP

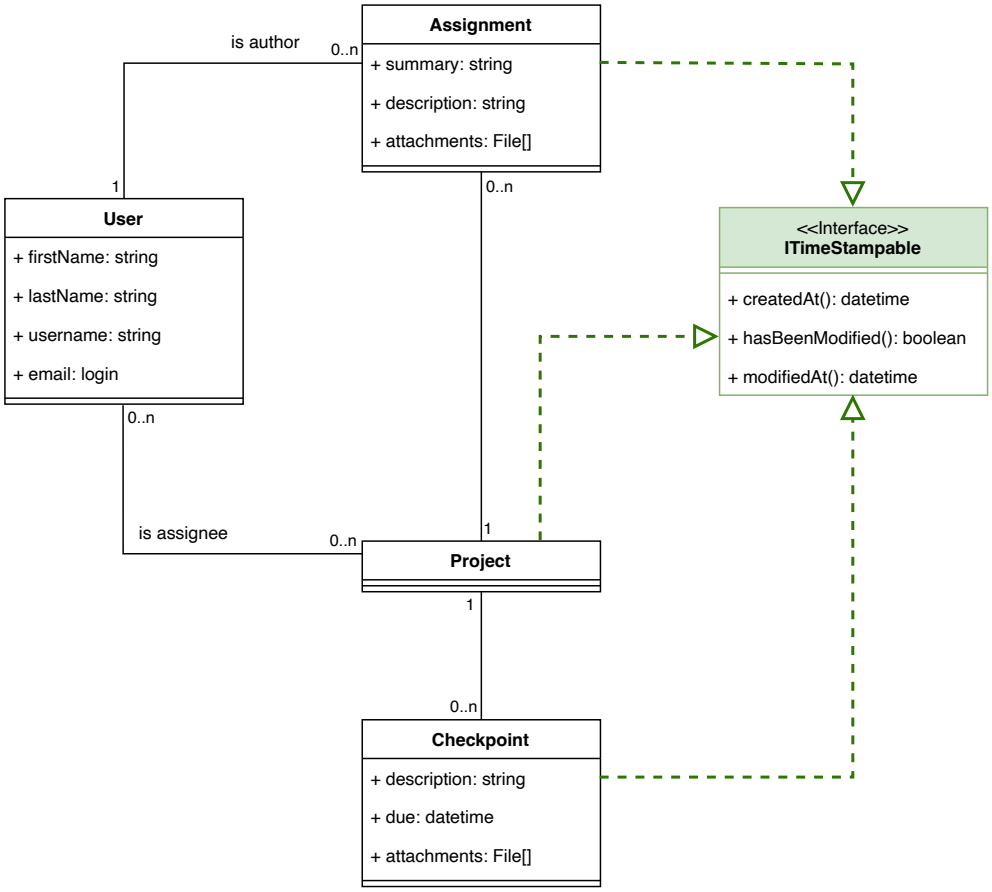
úloha alebo projekt priradený riešiteľom. Rozdiel medzi jednoduchšou úlohou a projektom by bol v tom, že úlohu by riešitelia odovzdávali ako celok, zatiaľ co projekt by riešitelia odovzdávali v rámci viacerých kontrolných bodoch, ktoré by okrem termínu odovzdania obsahovali aj zadanie vo forme textového popisu a príloh. Dôvod, prečo nie je v tomto návrhu žiadna asociácia medzi doménou `Activity` a doménou `User` je ten, že som pracoval s myšlienkou skutočne všeobecnej aktivity, ktorá by z dôvodu rozšíriteľnosti nemusela mať žiadnych riešiteľov.

Po predstavení a prediskutovaní vyššie vysvetlených návrhov s tímom a vedúcim práce sme sa zhodli, že väčší zmysel nám dáva druhý návrh, takže pri ďalších návrhoch som sa odrážal od neho. S kolegom Janom Mrázkom, ktorý v rámci svojej práce pracuje na kontrolných bodoch, odovzdávaní a hodnotení, sme sa dohodli na zovšeobecnení kontrolných bodov, teda domény `Checkpoint`. Ako už vidieť na návrhu na obrázku 5.7, doména teda bude mať okrem dátumu odovzdania aj textový popis, prílohy, v budúcnosti prípadne aj zadanie vo forme asociáce s doménou `Assignment` či priradených riešiteľov, ktorí budú podmnožinou členov tímu projektu. Uvedomil som si tiež, že nezáleží, či aktivita má alebo nemá kontrolné body a že nie je teda potrebné mať dve domény `Homework` a `Project`, ktorých jediný rozdiel spočíval v tom, či doména obsahuje kontrolné body alebo nie. Doména `Homework` vie byť totižto reprezentovaná ako doména `Project` s jedným kontrolným bodom. Následne som



Obr. 5.7 Druhý návrh doménového modelu pred implementáciou MVP

tiež začal pochybovať o potrebnosti abstraktnej doménovej triedy `Activity`, pretože som v tom nevidel žiadnu výhodu. Práve naopak mi zavedenie tejto abstraktnej triedy prišlo zbytočné, dokonca si myslím, že to zvyšovalo komplexitu celého systému a robilo kód neprehľadnejším. Myslím si, že v tomto prípade bude jednoduchšie abstraktného predka alebo rozhranie pridať neskôr až v prípade potreby. Počas samotnej implementácie sme sa s tímom rozhodli pridať ešte rozhranie `ITimeStampable`, ktoré implementujú všetky doménové triedy potrebujúce časovú evidenciu ich vzniku a poslednej úpravy. Výsledný návrh, ktorý som upravil na základe mojich rozhodnutí a diskusií s tímom a podľa ktorého som implementoval prvú verziu MVP je vidieť na obrázku 5.8.



Obr. 5.8 Návrh doménového modelu počas implementácie MVP

5.6.2 Pridanie schvaľovania projektov

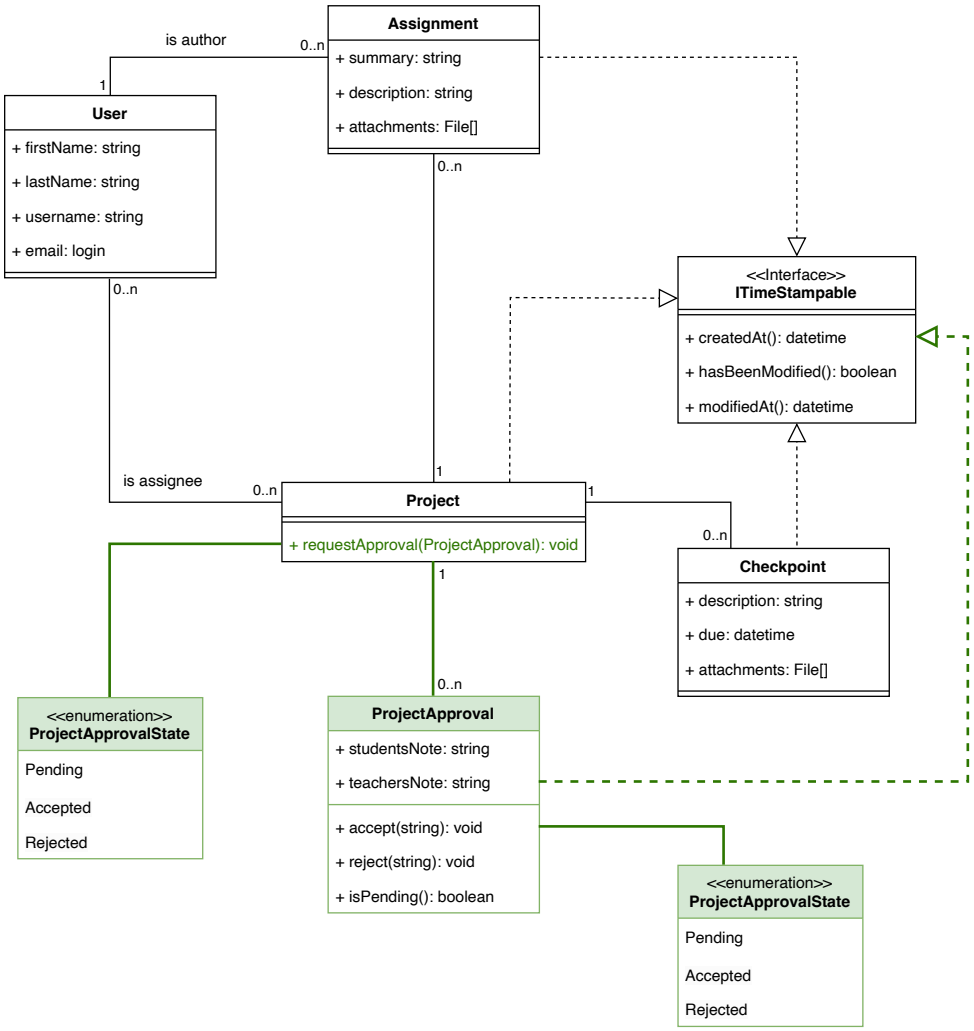
Po splnení väčšiny „Must Have“ požiadaviek som sa v ďalšej iterácií venoval nasledujúcej „Should Have“ požiadavke a nasledujúcim prípadom použitia.

Prípady použitia:

- UC12 – Spravovať projekt
- UC15 – Požiadať o schválenie projektu
- UC16 – Schváliť alebo odmietnuť študentmi vytvorený projekt

Pre potreby pridania žiadostí o schválenie študentmi vytvorených projektov bolo potrebné pridať doménovú triedu `ProjectApproval`, ktorá môže byť v troch rôznych stavoch, čo reprezentuje výčtový typ `ProjectApprovalState`. Okrem toho je potrebné vedieť, v akom stave je momentálne projekt – teda či už bol schválený alebo či študenti už požiadali o schválenie. Z tohto dôvodu som pridal ďalší výčtový typ `ProjectState`. Výsledný doménový model s pri-

danými doménami potrebnými pre schvaľovanie projektov je vidieť na obrázku 5.9.



Obr. 5.9 Návrh doménového modelu - pridanie schvaľovania projektov

5.6.3 Pridanie žiadostí a pozvánok do tímu

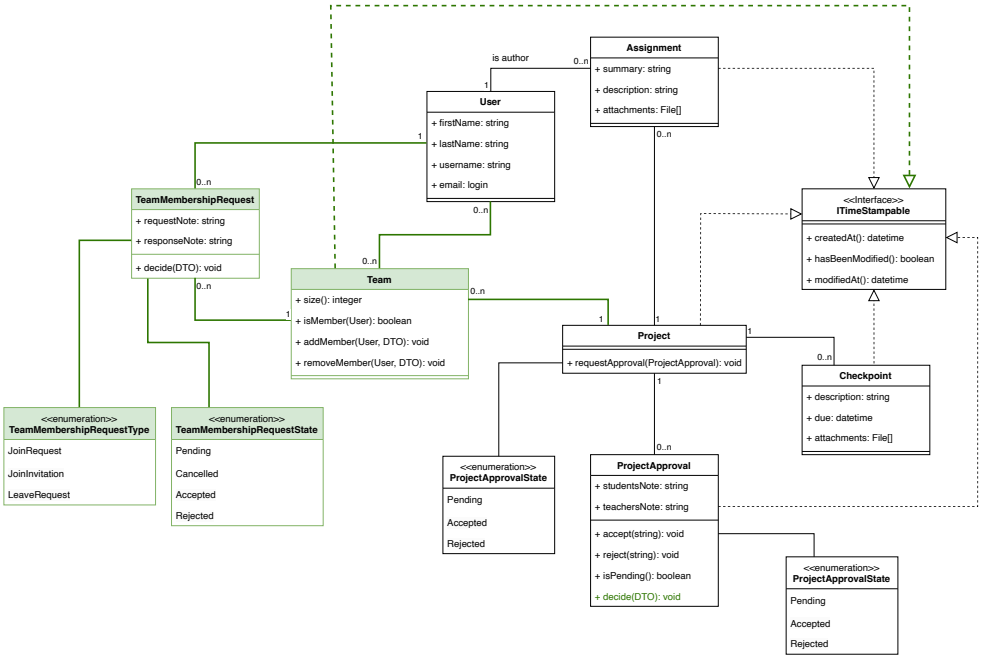
V ďalšej iterácií som pracoval na pokročilejších a menej prioritných funkcionalitách týkajúcich sa „Must Have“ požiadavky na tvorbu študentských tímov.

Prípady použitia:

- UC4 – Pripojiť sa k tímu

- UC5 – Schváliť alebo odmietnuť žiadosť o pripojenie sa k tímu
- UC6 – Opustiť tím
- UC7 – Spravovať tím
- UC8 – Pridať alebo pozvať nového člena do tímu

Pre pridanie nových funkcionalít týkajúcich sa správy tímu, žiadostí o pripojenie sa do tímu, žiadostí o opustenie tímu a pozvánok do tímu bolo v prvom rade potrebné pridať novú doménovú triedu `Team` reprezentujúcu tím, teda skupinu študentov a žiadosti o členstvo v danom tíme. Samotná žiadosť o členstvo v tíme, žiadosť o opustenie tímu ako aj pozvánka do tímu je reprezentovaná doménovou triedou `TeamMembershipRequest`. To či sa jedná o žiadosť o členstvo, žiadosť o opustenie tímu alebo pozvánku do tímu je reprezentované výčtovým typom `TeamMembershipRequestType`. Ďalej je potrebné rozlišovať, či táto žiadosť čaká na rozhodnutie alebo už bola prijatá alebo odmietnutá. Pre toto som pridal ďalší výčtový typ `TeamMembershipRequestState`. Výsledný doménový model po pridaní domény `Team` a žiadostí o členstvo v tíme, opustenie tímu a pozvánok do tímu je vidieť na obrázku 5.10.



Obr. 5.10 Návrh doménového modelu - pridanie žiadostí a pozvánok do tímu

5.7 API

Opäť si dovolím čitateľa upozorniť na to, že text tejto sekcie bude členený na podsekcie podľa jednotlivých iterácií a verzií aplikácie. Na začiatku textu o danej iterácií najprv uvediem akým prípadom použitia som sa v danej iterácií venoval. V rámci každej iterácie uvediem zmeny vykonané v návrhu API. Kompletnú dokumentáciu aktuálneho API vygenerovanú nástrojom Swagger[7] nájde čitateľ na PagesFIT.

5.7.1 MVP

Prípady použitia:

- UC1 – Vytvoriť zadanie
- UC3 – Vytvoriť tím
- UC4 – Pripojiť sa k tímu
- UC9 – Vytvoriť projekt
- UC10 – Zobraziť a filtrovať projekty
- UC11 – Zobraziť detaily projektu

Pre splnenie najprioritnejších požiadaviek sme spolu s tímom v rámci návrhu MVP pridali nasledujúce API endpointy pre autentifikáciu a správu užívateľov a tvorbu a zobrazenie zadaní a projektov.

Autentifikácia užívateľa a získanie JWT tokenu:
`POST /api/auth/authenticate`

Overenie validity JWT tokenu: `GET /api/auth/token`

Zoznam užívateľov: `GET /api/users`

Detail užívateľa `GET /api/users/{userId}`

Vytvorenie zadania: `POST /api/assignments`

Zoznam zadaní: `GET /api/assignments`

Detaily zadania: `GET /api/assignments/{assignmentId}`

Vytvorenie projektu: `POST /api/projects`

Zoznam projektov: `GET /api/projects`

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

Detaily projektu: `GET /api/projects/{projectId}`

Okrem toho som pridal nasledujúce endpointy podporujúce primárne pridanie a odobratie riešiteľov projektu.

Pripojenie sa k projektu: `POST /api/projects/{projectId}/assignees`

Opustenie projektu: `DELETE /api/projects/{projectId}/assignees`

Pridanie riešiteľov projektu:
`POST /api/projects/{projectId}/assignees/batch`

Odobratie riešiteľa projektu:
`DELETE /api/projects/{projectId}/assignees/{assigneeId}`

5.7.2 Pridanie schvaľovania projektov

Prípady použitia:

- UC12 – Spravovať projekt
- UC15 – Požiadať o schválenie projektu
- UC16 – Schváliť alebo odmietnuť študentmi vytvorený projekt

Pre podporu žiadostí o schválenie študentmi vytvoreného projektu a následné prijatie alebo odmietnutie tejto žiadosti vyučujúcim som pridal nasledovné API endpointy. Jedná sa o požiadanie vyučujúceho o schválenie projektu, teda o vytvorenie zdroja `ProjectApproval` v prípade úspechu a validných dát v tele žiadosti a o operácie nad danou žiadosťou o schválenie ako zobrazenie žiadosti, zrušenie žiadosti či prijatie alebo zamietnutie žiadosti.

Žiadosť o schválenie projektu:
`POST /api/projects/{projectId}/approval-requests`

Detaily žiadosti o schválenie projektu:
`GET /api/project-approvals/{projectApprovalId}`

Zrušiť čakajúcu žiadosť o schválenie projektu:
`DELETE /api/project-approvals/{projectApprovalId}`

Prijať žiadosť o schválenie projektu:
`PATCH /api/project-approvals/{projectApprovalId}/accept`

Zamietnuť žiadosť o schválenie projektu:
`PATCH /api/project-approvals/{projectApprovalId}/reject`

5.7.3 Pridanie žiadostí a pozvánok do tímu

Prípady použitia:

- UC4 – Pripojiť sa k tímu
- UC5 – Schváliť alebo odmietnuť žiadosť o pripojenie sa k tímu
- UC6 – Opustiť tím
- UC7 – Spravovať tím
- UC8 – Pridať alebo pozvať nového člena do tímu

Pre potrebu pridania žiadostí o členstvo v tíme, opustenie tímu a pozvánok do tímu bolo potrebné backend aplikáciu čiastočne zrefaktorovať. Ako uvádzam vyššie v časti 5.6.3, pridal som novú doménovú triedu `Team`, ktorý je zodpovedný za riešenie projektu a nahradil tak riešiteľov projektu. V rámci návrhu API pre podporu tímov bolo potrebné pridať endpointy pre tvorbu a zobrazenie tímu, pre pridanie a odobranie členov tímu a pre rozhodnutie alebo zrušenie žiadosti o členstvo v tíme.

Vytvorenie tímu: `POST /api/teams`

Detaily tímu: `GET /api/teams/{teamId}`

Pridanie nového člena tímu: `POST /api/teams/{teamId}/member`

Odobratie člena tímu: `DELETE /api/teams/{teamId}/members`

Rozhodnúť žiadosť o členstvo v tíme:
```
POST
/api/team-membership-requests/
{teamMembershipRequestId}/decisions
```

Zrušiť žiadosť o členstvo v tíme:
`DELETE /api/teams/{teamId}/members`

Je nutné podotknúť, že z dôvodu vysokej univerzálnosti backend aplikácie a jej API som sa rozhodol vytvoriť jeden endpoint pre akýkoľvek typ pridania členov do tímu. Jediným zavolaním tohto endpointu je totiž možné ľubovoľný počet užívateľov do tímu pridať priamo, zároveň vytvoriť žiadosť o vstup do tímu pre ľubovoľný počet užívateľov ako aj vytvoriť pozvánku do tímu pre ľubovoľný

počet užívateľov. To o aký typ pridania, teda či sa daný užívateľ pridá priamo alebo či sa vytvorí žiadosť o členstvo v tíme, sa určuje v tele žiadosti, ktorú je možné vidieť na výpise kódu 5.1.

```
{
  "members": [
    {
      "memberId": 42,
      "note": "",
      "requestType": 1,
    },
    {
      "memberId": 43,
      "note": "",
      "requestType": 1,
    },
  ]
}
```

Výpis kódu 5.1 Očakávané telo žiadosti POST /api/teams/teamId/member vo formáte JSON

To isté platí aj pre priame odstránenie člena tímu a podanie žiadosti o opustenie tímu, kde je typ opäť určený v tele žiadosti, očakávane telo žiadosti `DELETE /api/teams/{teamId}/member` je rovnaké ako na výpise 5.1.

Ako píšem vyššie, hlavnou výhodou tohto rozhodnutia je univerzálnosť. Využívam v tomto prípade jednu z výhod jedno-stránkovej frontend aplikácie, konkrétne toho, že určitú biznis logiku systému nechávam na frontend, vďaka čomu môže backend slúžiť ako univerzálne rozhranie pre viacero frontend aplikácií.

Okrem toho bolo potrebné zmeniť telo žiadosti pri tvorbe projektu. Pôvodný stav, kde backend aplikácia očakáva pole celých čísel reprezentujúcich ID riešiteľov projektu, je vidieť na výpise 5.2, upravený stav, kde je očakávané práve jedno celé číslo reprezentujúce ID tímu, ktorý bude daný projekt riešiť, je znázornený na výpise 5.3.

5.7.4 Refaktoring schvaľovania projektov

Prípady použitia:

- UC15 – Požiadať o schválenie projektu

```
{
    "assignmentId": 1,
    "assigneesIds": [1, 2, 3, 4, 5],
}
```

Výpis kódu 5.2 Pôvodné očakávané telo žiadosti POST /api/projects vo formáte JSON

```
{
    "assignmentId": 1,
    "teamId": 1,
}
```

Výpis kódu 5.3 Upravené očakávané telo žiadosti POST /api/projects vo formáte JSON

- UC16 – Schváliť alebo odmietnuť študentmi vytvorený projekt

V rámci refaktoringu som odstránil dva endpointy na prijatie a zamietnutie žiadosti o schválenie projektu.

Prijať žiadosť o schválenie projektu:
`PATCH /api/project-approvals/{projectApprovalId}/accept`

Zamietnuť žiadosť o schválenie projektu:
`PATCH /api/project-approvals/{projectApprovalId}/reject`

Nahradil som ich jedným endpointom, ktorý v tele žiadosti dostane informáciu o tom, či bude daná žiadosť prijatá alebo zamietnutá.

Rozhodnúť žiadosť o schválenie projektu:
`PATCH /api/project-approvals/{projectApprovalId}`

Očakávane telo žiadosti nového endpointu, kde hodnota `status = 0` znamená prijatie žiadosti a hodnota `status = 1` zamietnutie žiadosti, je znázornené na výpise 5.4.

Dôvodom tohto refaktoringu bolo zjednotenie akceptovania a zamietnutia žiadostí so žiadosťami o členstvo v tíme. Ďalším dôvodom je fakt, že počas implementácie som si uvedomil, že sú si tieto funkcionality veľmi podobné, dokonca takmer zhodné. Vďaka novému návrhu s jedným endpointom teda predchá-

```
{
    "status": 0,
    "note": ""
}
```

Výpis kódu 5.4 Očakávané telo žiadosti PATCH /api/project-approvals/projectApprovalId vo formáte JSON

dzam zbytočnému duplikovaniu kódu.

5.8 Užívateľské rozhranie

Pre vytvorenie návrhu užívateľského rozhrania som pracoval na tvorbe Lo-Fi prototypu pomocou nástroja Balsamiq[8]. Zvažoval som aj použitie nástrojov Figma[9] a Sketch[10], s ktorými som si tiež vyskúšal pracovať, ale kvôli jednoduchosti som nakoniec uprednostnil nástroj Balsamiq.

Pri protypovaní vieme definovať tzv. fidelitu prototypu. Fidelita prototypu znamená, do akej miery zodpovedá prototyp vzhľadu a fungovaniu konečného systému. Lo-Fi (z anglického Low Fidelity) prototyp – často tiež označovaný ako tzv. wireframe – označuje prototyp s nízkou fidelitou. Lo-Fi prototyp teda narozdiel od Hi-Fi (z anglického High Fidelity) prototypu s vysokou fidelitou neobsahuje žiadne klikateľné odkazy alebo navigáciu, nemusí nutne obsahovať všetky prvky finálneho návrhu, často sa môže jednať o čierno biele náčrty so schematickým znázornením obrázkov a inej grafiky a iba zhrnutím finálneho obsahu. Inými slovami sa Lo-Fi prototyp zameriava skôr na funkcionalitu a prioritu obsahu, nie na štýl a to ako bude užívateľské rozhranie vyzerať. [87]

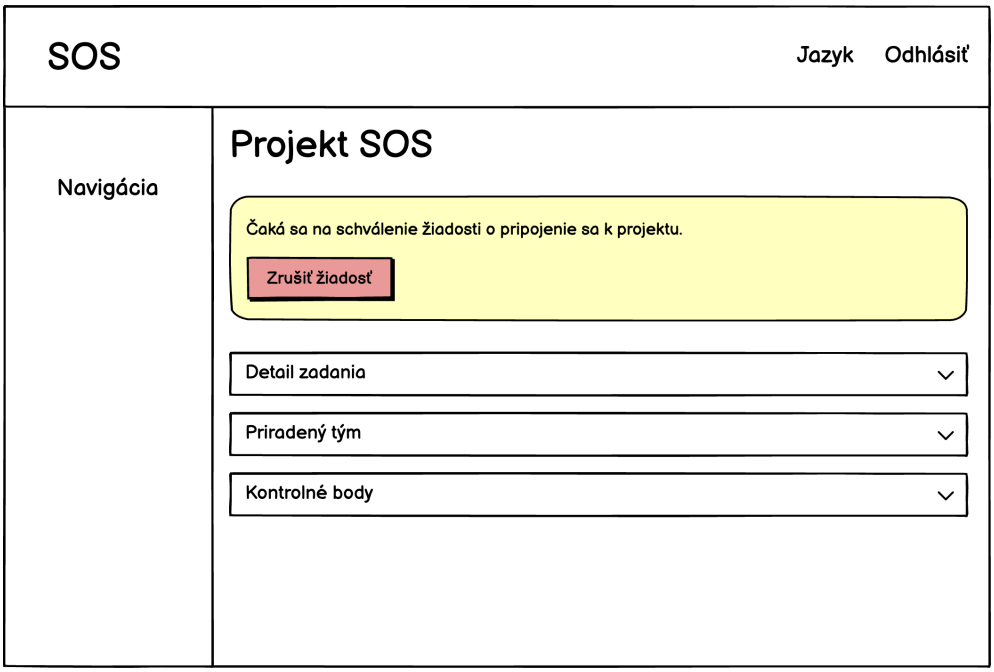
Prototypovanie som rovnako ako návrh doménového modelu a návrhu API uskutočňoval vo viacerých iteráciách, návrhy UI ale nebudem takto uvádzať, pretože som jednotlivé obrazovky počas alebo po implementácii takmer nemenil, keďže mi Lo-Fi prototyp poslúžil hlavne ako predstava o rozložení a usporiadaní jednotlivých prvkov na obrazovkách.

5.8.1 Odoslaná žiadosť o pripojenie sa k projektu

Na obrázku 5.11 je znázornená obrazovka, kedy študent požiadal o pripojenie sa k projektu a čaká na schválenie jeho žiadosti buď učiteľom alebo členom tímu pracujúcom na danom projekte v závislosti na stave projektu. Študent vidí

[8] https://balsamiq.com/
[9] https://figma.com/
[10] https://www.sketch.com/

farebne zvýraznené upozornenie na obrazovke detailného zobrazenie projektu a môže odoslanú žiadosť zrušiť.



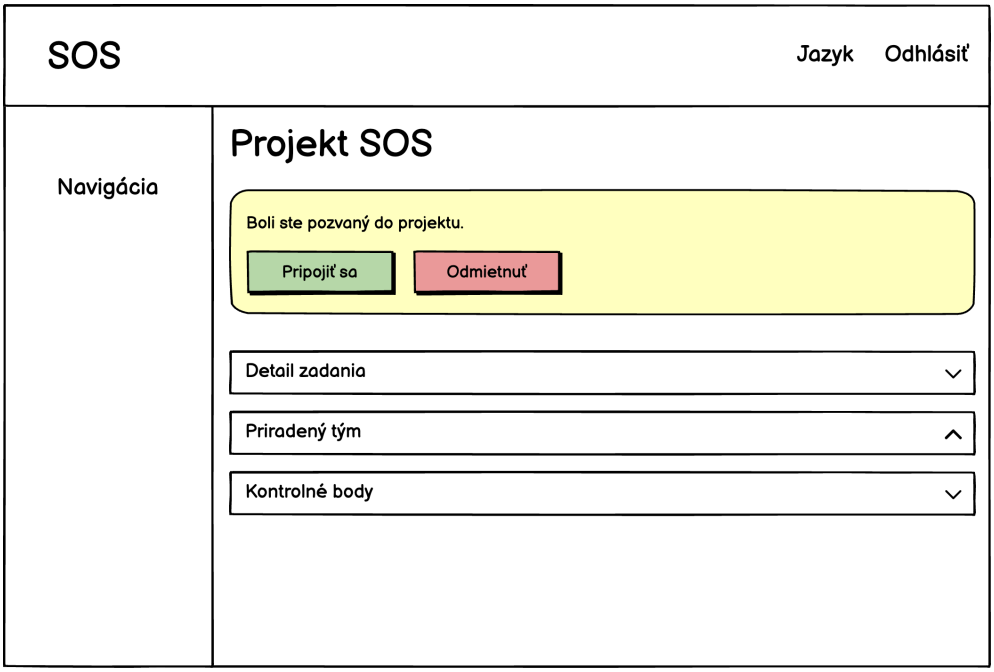
Obr. 5.11 Lo-Fi prototyp - odoslaná žiadosť o pripojenie sa k projektu

5.8.2 Pozvánka do projektu

Keď je študent pozvaný iným študentom k pripojeniu sa do projektu, vidí na obrazovke detailného zobrazenia daného projektu farebne zvýraznené upozornenie. Pozvánku vie buď prijať zeleným tlačítkom alebo odmietnuť červeným tlačítkom. Ako vidieť na obrázku 5.12, na tlačítko pre prijatie pozvánky je text „Pripojiť sa“, aby bolo užívateľovi jasné, že prijatím pozvánky sa stane členom tímu a pripojí sa tak k projektu.

5.8.3 Členovia tímu priradeného k projektu

Obrázok 5.13 ukazuje obrazovku detailného zobrazenia projektu s informáciami o členoch priradeného tímu. Jedná sa v podstate o tabuľku všetkých členov tímu s ich menom, e-mailom, bodovým ziskom, prípadne ďalším údajmi. Členovia tímu, ktorí požiadali o opustenie budú zobrazení na spodku tabuľky a budú farebne odlíšený od ostatných členov. Pod členmi tímu, ktorí požiadali o opustenie budú ešte študenti, ktorí buď požiadali o pripojenie alebo boli pozvaní. Títo študenti budú tiež farebne odlíšení.



Obr. 5.12 Lo-Fi prototyp - pozvánka do projektu

5.8.4 Čakajúce žiadosti

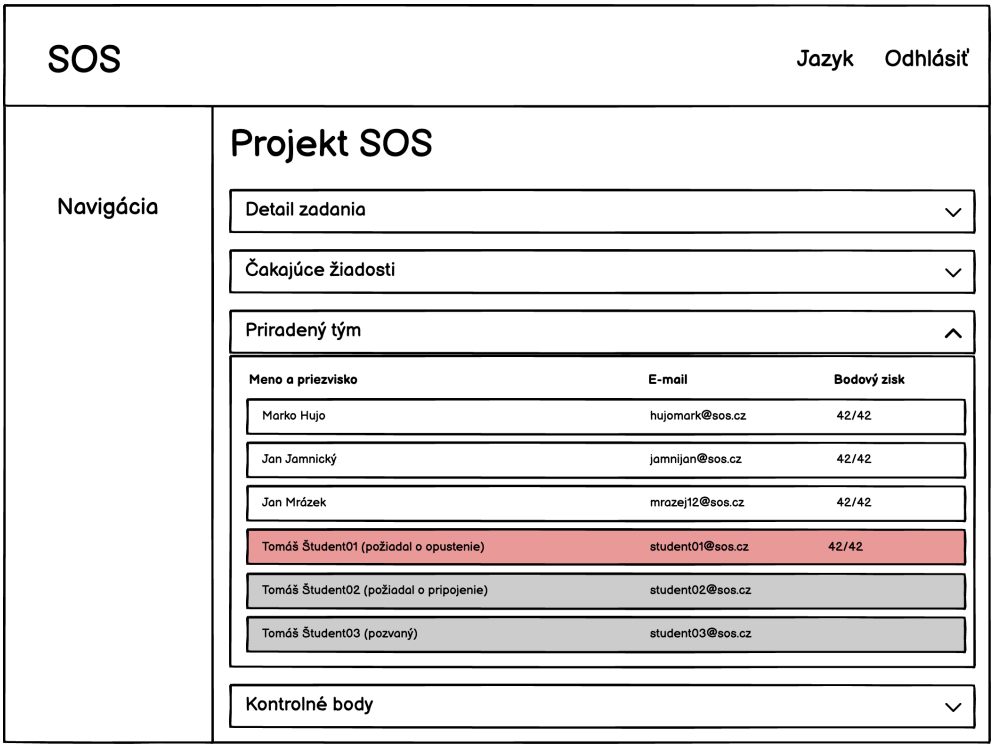
Študenti neschváleného tímu alebo učiteľ schváleného tímu môže rozhodovať o žiadostiach o členstvo v projektovom tíme ako aj o opustenie. Ako je vidieť na obrázku 5.14, čakajúce žiadosti vidí užívateľ v klikateľnej karte v rámci obrazovky detailného zobrazenia projektu.

5.8.5 Požiadanie o schválenie projektu

Pred prvým odovzdaním je v prípade študentmi vytvoreného projektu nutné, aby študenti požiadali o schválenie projektu vyučujúcim a preniesli tak správu projektu na vyučujúceho. Obrázok 5.15 znázorňuje farebne zvýraznené upozornenie na ešte neschválený projekt a možnosť požiadať o schválenie.

5.8.6 Schválenie projektu

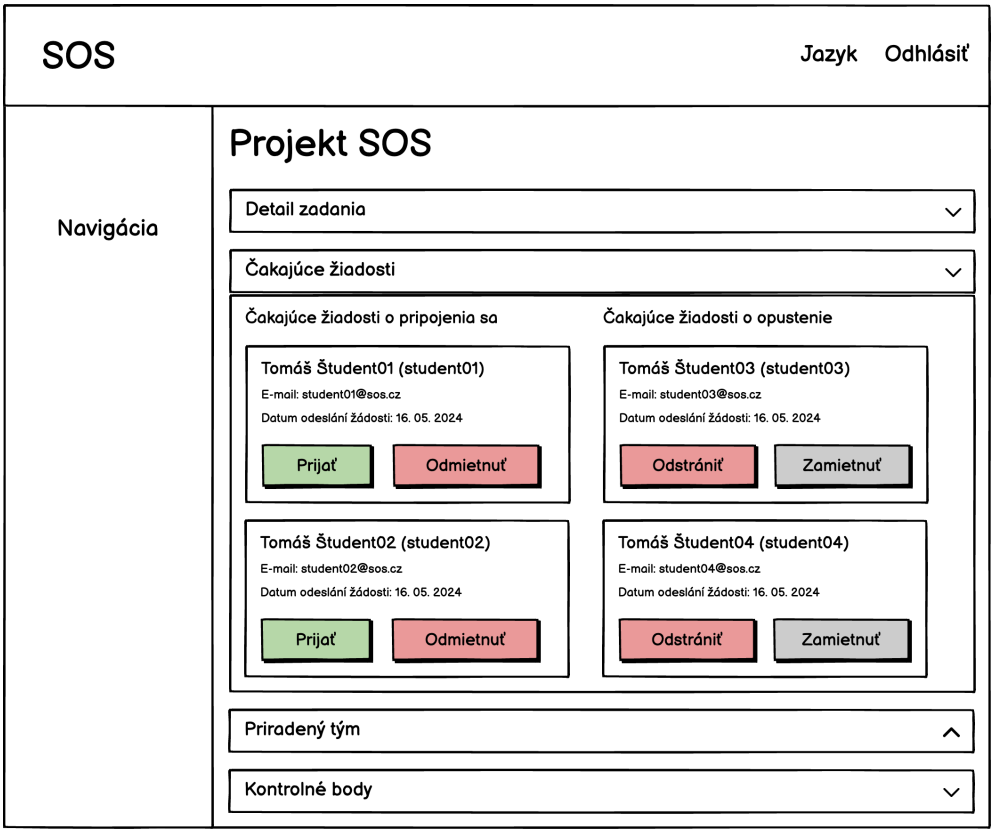
Obrázok 5.16 znázorňuje farebne zvýraznené upozornenie o nutnosti schváliť študentmi vytvorený projekt na obrazovke detailného zobrazenie projektu.



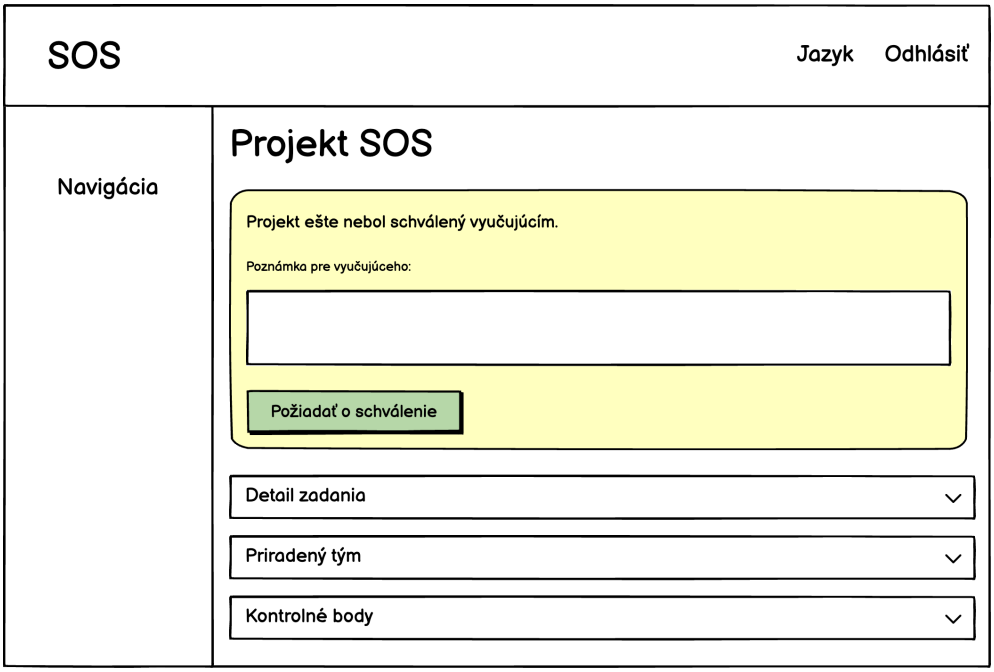
Obr. 5.13 Lo-Fi prototyp - členovia tímu

5.8.7 Správa tímu

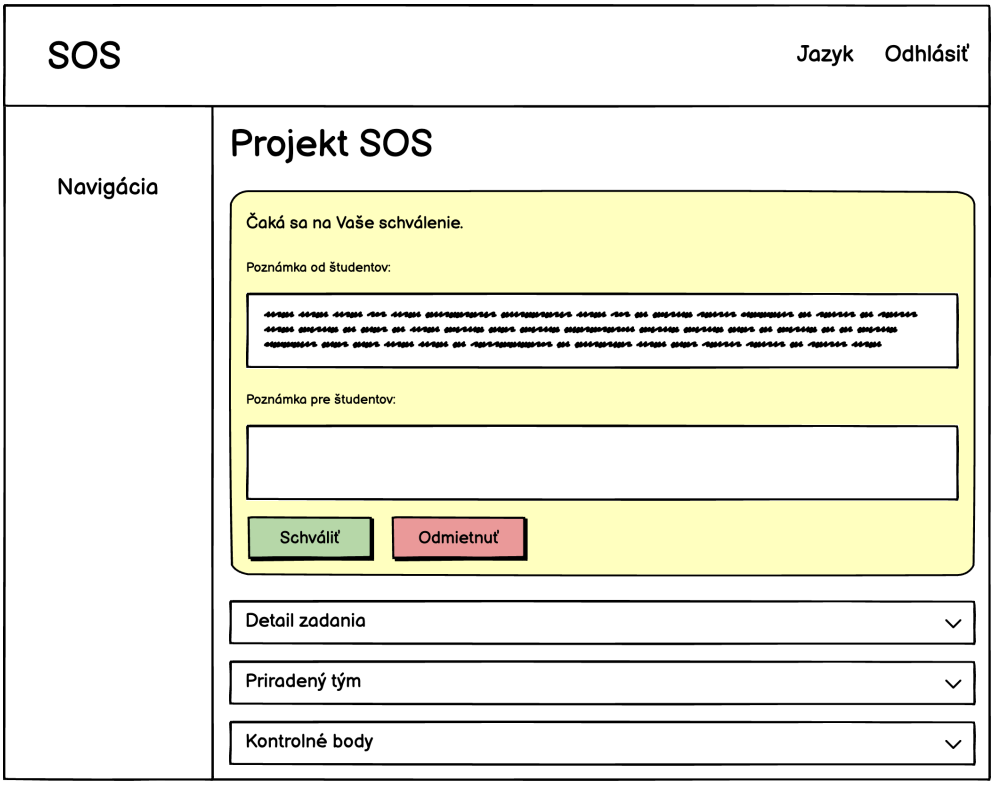
V rámci obrazovky nastavení projektu, na ktorú sa užívateľ dostane kliknutím na tlačítko na obrazovke detailného zobrazenia projektu, vidí užívateľ klikateľnú kartu pre správu tímu. V rámci správy tímu vie užívateľ odobrať člena tímu a pridať alebo pozvať nových členov tímu. Obrazovku nastavení projektu so správou tímu nájde užívateľ na obrázku 5.17.



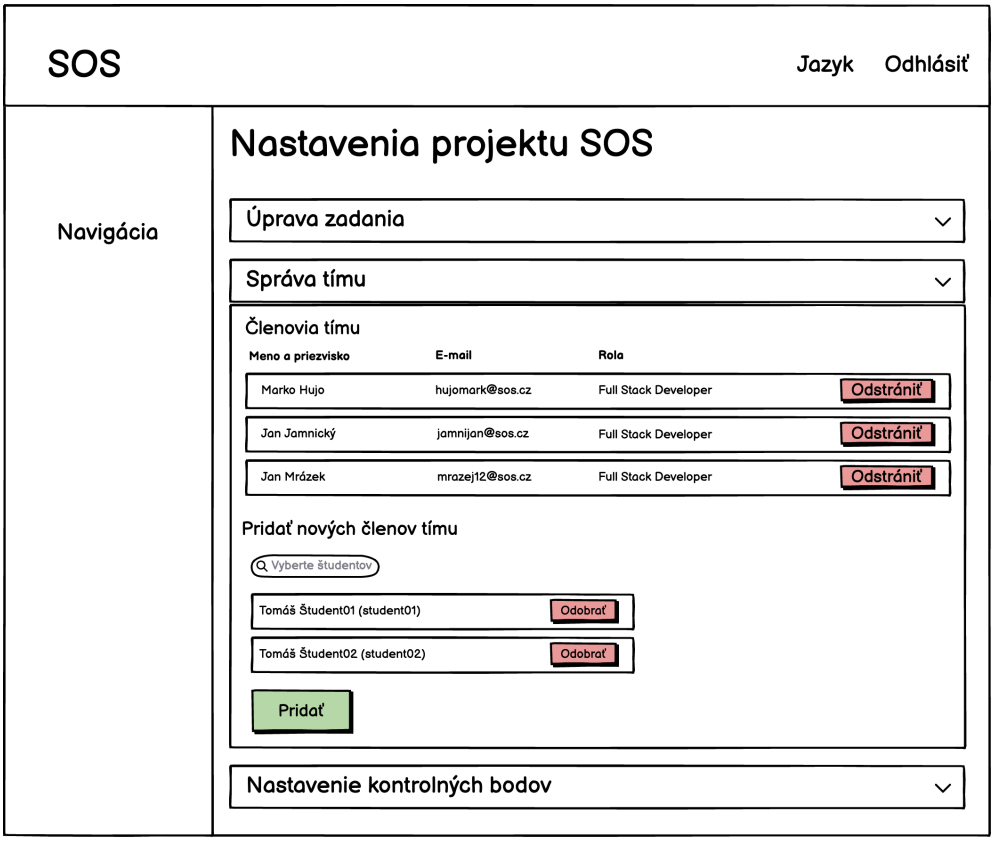
Obr. 5.14 Lo-Fi prototyp - čakajúce žiadosti



Obr. 5.15 Lo-Fi prototyp - požiadanie o schválenie projektu



Obr. 5.16 Lo-Fi prototyp - schválenie projektu



Obr. 5.17 Lo-Fi prototyp - správa tímu

Kapitola 6

Implementácia

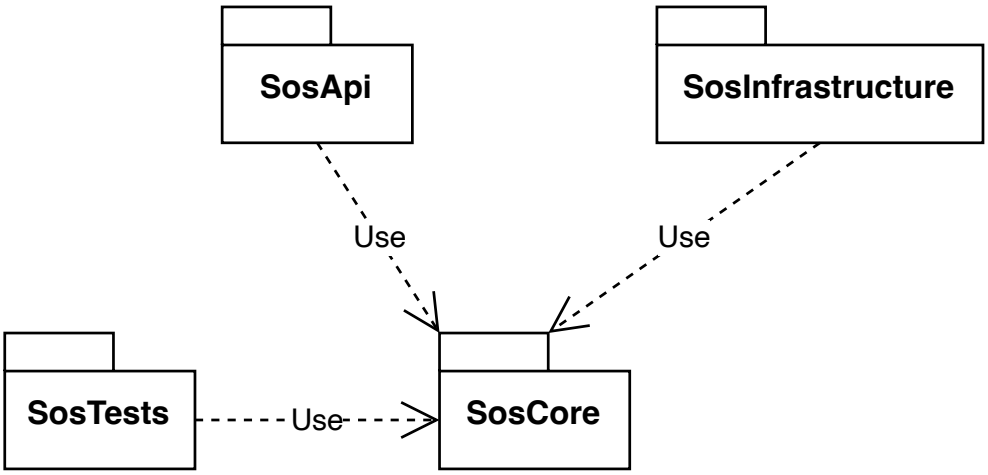
V tejto kapitole popíšem implementáciu nového systému, zameriam sa hlavne na dôležité časti kódu a celkovú štruktúru kódu. Najprv popíšem implementáciu backendu a následne implementáciu frontendu.

6.1 Backend

Na úvod predstavím štruktúru a jednotlivé projekty backend aplikácie, následne definujem návrhový vzor Dependency Injection a ukážem, ako ho využívame a vysvetlím, ako funguje autorizácia a autentifikácia užívateľov.

6.1.1 Štruktúra aplikácie

V jednej z iterácií na začiatku letného semestra 2023/2024, teda v období, kedy sme mali na testovacom prostredí nasadenú jednu z prvých a najzákladnejších verzií novej aplikácie, sme sa s tímom rozhodli pre rozsiahlejší refaktoring celej backend aplikácie za účelom zavedenia tzv. čistej architektúry, ktorú teoreticky definujem v časti 2.6. Jednotlivé vrstvy sú v našej aplikácii reprezentované .NET projektami. Štruktúru a závislosti týchto vrstiev je možné vidieť na diagrame balíčkov na obrázku 6.1. Pred refaktoringom štruktúra aplikácie spĺňala definíciu trojvrstvovej architektúry definovanej v časti 2.4, kde každá vrstva bola reprezentovaná ako priečinok v jednom .NET projekte.



Obr. 6.1 Diagram balíčkov backend aplikácie

6.1.1.1 SosCore

Projekt `SosCore` obsahuje všetky biznis modely a všetku biznis logiku aplikácie. Jedná sa o DTO pre zapuzdrenie prenášaných dát, entity pre mapovanie štruktúry dát v databáze na C# triedy a konfigurácie automatického mapovania medzi DTO a entitami. Ďalej obsahuje `Service` triedy zapuzdrujúce biznis logiku a rôzne rozhrania, ktoré poskytujú abstrakcie nad operáciami z projekt `SosInfrastructure` ako napríklad prístup k databáze alebo správu súborov.

DTO

DTO (z anglického Data Transfer Object) je objekt, ktorý prenáša dáta medzi procesmi. Jedná sa teda o akési zapuzdrenie dát potrebných pre prenos medzi rôznymi vrstvami alebo komponentmi aplikácie, ktoré neobsahuje žiadnu biznis logiku a je nezávislé na implementácii. [88]

V našej aplikácii používame dva typy DTO – DTO pre žiadosti a DTO pre odpovede. Pre DTO používame v projekte C# *record* triedy, ktoré poskytujú funkcionalitu pre zapuzdrenie dát a sú primárne určené na podporu nemenných dátových modelov.

Ukážka DTO objektov pre schválenie projektu je vidieť na výpisoch kódu 6.1 a 6.2.

```
public record ProjectApprovalRequest
{
    [MaxLength(2500)]
    public string? Note { get; set; }
}
```

Výpis kódu 6.1 Ukážka DTO triedy pre žiadosti

```
public record ProjectApprovalResponse
{
    public long Id { get; set; }

    public ProjectBriefResponse Project { get; set; }

    public string? StudentsNote { get; set; }

    public string? TeachersNote { get; set; }

    public string State { get; set; }

    public DateTime CreatedAt { get; set; }

    public DateTime ModifiedAt { get; set; }
}
```

Výpis kódu 6.2 Ukážka DTO triedy pre odpovede

Entity

Entitné triedy v ASP.NET Core aplikácii reprezentujú doménový model aplikácie a mapujú štruktúra dát v databáze. Zvyčajne obsahujú atribúty, ktoré zodpovedajú stĺpcom v tabuľke a môžu obsahovať aj vzťahy k iným entitným triedam, ktoré odrážajú relácie medzi tabuľkami v databáze (napríklad *one-to-many* alebo *many-to-many*). Entitné triedy sú kľúčové pre silno typovanú prácu s dátami z databáze v rámci celej aplikácie. Ukážka entitnej triedy `Project` je na výpise kódu 6.3.

Entitné triedy môžu okrem dát obsahovať aj metódy a validáciu, vďaka čomu môžu nezávisle pracovať s vlastnými dátami. Vyhýbame sa tak *anti-patternu* známemu pod pojmom *Data Class*, teda triedam, ktoré obsahujú len atribúty a CRUD metódy pre prístup k týmto atribútom (`Get` a `Set` metódy). Jedná sa jednoducho o akési kontajneri pre dáta používané v iných triedach bez žiadnej ďalšej funkcionality. [89] Ja som sa rozhodol používať metódy v entitných

```
public class Project : ITimeStampable
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public long Id { get; set; }

    public Assignment Assignment { get; set; }

    public Team Team { get; set; }

    public ICollection<Checkpoint> Checkpoints { get; set; }

    public ICollection<ProjectApproval> Approvals { get; set; }

    public ProjectState State { get; set; }

    public DateTime CreatedAt { get; set; }

    public DateTime ModifiedAt { get; set; }
```

Výpis kódu 6.3 Ukážka entitnej triedy

triedach pre tú logiku aplikácie, ktorá sa týka stavu a práce s atribútmi daného objektu. Biznis logiku, ktorá sa netýka priamo práce s objektom entitnej triedy prenechávam na príslušné vrstvy aplikácie. Príklad takejto metódy je vidieť na ukážke 6.4, kde overím, že projekt ešte nebol schválený alebo že nečaká na schválenie, vyhodím prípadne výnimku, ale v prípade chceného stavu pridám žiadosť o schválenie a upravím stav projektu, aby bolo zrejmé, že študenti už požiadali o schválenie a projekt teraz čaká na schválenie vyučujúcim.

Profile

Pre automatické mapovanie DTO objektov na entitné objekty používame knižniču AutoMapper[1]. Ako uvádza jej dokumentácia, jedným zo spôsobov organizácie konfigurácií mapovania sú tzv. profily, teda triedy dediace z triedy `Profile`. [90] Príklad použitia v našom projekte je znázornený na triede `ProjectApprovalProfile` na výpise kódu 6.5. Knižnica AutoMapper vyžaduje, aby pomenovanie atribútov cieľa aj zdroja bolo rovnaké. Pre mapovanie atribútov, ktoré nemajú identické pomenovanie je potrebné manuálne definovať mapovanie pomocou ďalších metód.

[1] `https://docs.automapper.org/en/stable/index.html`

```
public Project RequestApproval(ProjectApproval projectApproval)
{
    ThrowIfNotUnapproved();

    Approvals.Add(projectApproval);
    State = ProjectState.WaitingForApproval;

    return this;
}

private void ThrowIfNotUnapproved()
{
    if (State != ProjectState.Unapproved)
    {
        throw new ProjectIsNotUnapprovedException();
    }
}
```

Výpis kódu 6.4 Ukážka metód v entitnej triede

```
public class ProjectApprovalProfile : Profile
{
    public ProjectApprovalProfile()
    {
        CreateMap<ProjectApproval, ProjectApprovalResponse>();
    }
}
```

Výpis kódu 6.5 Ukážka konfigurácie automatického mapovania

Service

`Service` triedy sú komponenty aplikácie, ktoré zapuzdrujú komplexnú biznis logiku, prístup k dátam a iné záležitosti týkajúce sa infraštruktúry ako napríklad mapovanie entintných objektov na DTO a vice versa.

Na ukážke 6.6 je vidieť trieda `ProjectApprovalService` a jedna z jej metód `DecideProjectApproval`, ktorá slúži na prijatie alebo odmietnutie žiadosti o schválenie projektu. V metóde sú najprv získané dáta z databáze, následne sa overí, že pre hľadané ID sa entitu podarilo nájsť. Potom sa určitá logika prenechá metóde entitnej triedy, ktorá overí, že študentmi podaná žiadosť ešte nebola rozhodnutá, zmení stav projektu na schválený alebo neschválený a nastaví stav žiadosti na prijatú alebo zamietnutú. Následne sa opäť

v metóde `DecideProjectApproval` v triede `ProjectApprovalService` uloží všetky uskutočnené zmeny do databáze v rámci jednej transakcie a vráti sa DTO získané z entitného objektu.

```
public class ProjectApprovalService(
    IDataContext dataContext,
    IMapper mapper
) : IProjectApprovalService
{
    public async Task<ProjectApprovalResponse>
        DecideProjectApproval(
            long projectApprovalId,
            User teacher,
            ProjectApprovalDecisionRequest
                projectApprovalDecisionRequest,
            CancellationToken cancellationToken = default
         )
    {
        var projectApproval = await dataContext.ProjectApprovals
            .Where(pa => projectApprovalId == pa.Id)
            .Include(pa => pa.Project)
            .FirstOrDefaultAsync(cancellationToken);

        Guard.Against.NotFound(
            projectApprovalId,
            projectApproval
        );

        projectApproval.Decide(projectApprovalDecisionRequest);

        await dataContext.FlushChanges(cancellationToken);
        return mapper.Map<ProjectApprovalResponse>(
            projectApproval
        );
    }
```

Výpis kódu 6.6 Ukážka `Service` triedy

6.1.1.2 SosInfrastructure

Projekt `SosInfrastructure` obsahuje implementácie rozhraní z projektu `SosCore`. Jedná sa o prístup k databáze a konfiguráciu modelu, prácu so súbormi alebo

zabezpečenie aplikácie a generovanie JWT pre užívateľa. Okrem toho obsahuje súbory definujúce databázové migrácie.

Tvorba a konfigurácia modelu a databázového kontextu

V rámci vyššie spomínaného refaktoringu celej backend aplikácie na začiatku letného semestra 2023/2024 sme okrem zavedenia spomínanej čistej architektúry odstránili aj tzv. *Repository* triedy alebo repozitáre, ktoré v trojvrstvovej architektúre implementujú perzistentnú vrstvu. Úlohou repozitárov je sprostredkovávať akúsi izolovanú vrstvu medzi entitnými objektami a prístupu k databáze pomocou jednoduchého rozhrania pre prístup ku kolekcii entitných objektov uložených v pamäti. [91] Dôvodom odstránenia našich *Repository* tried bolo to, že sme si uvedomili, že pri jednoduchosti našej aplikácie nie sú potrebné a zbytočne robia štruktúru aplikácie zložitejšiu vytváraním zbytočnej triedy pre každú `Entity` triedu. Ďalším dôvodom je fakt, že `DbContext` trieda, ktorá v Entity Framework Core slúži na získavanie dát z databáze a zoskupovanie zmien, ktoré sa spolu do databáze zapíšu ako celok, už vlastne plní účel spomínaného repository vzoru, pretože poskytuje veľmi jednoduché rozhranie pre získavanie doménových objektov, [92] ako je napríklad vidieť na ukážke kódu 6.7, kde je z databáze získaný objekt reprezentujúci schválenie projektu s hľadaným ID vrátane projektu, ktorého sa schválenie týka a tímu, ktorý daný projekt rieši vrátane jeho členov vo forme entitného objektu triedy `ProjectApproval`.

```
var projectApproval = await dataContext.ProjectApprovals
            .Where(pa => pa.Id == projectApprovalId)
            .Include(pa => pa.Project)
            .ThenInclude(p => p.Team)
            .ThenInclude(t => t.Members)
            .FirstOrDefaultAsync(cancellationToken);
```

Výpis kódu 6.7 Ukážka použitia `DbContext` pre získanie dát z databáze

Pre definíciu doménového modelu a databázového kontextu je potrebné implementovať triedu dediacu vyššie spomínanú triedu `DbContext`. Ako vidieť na úkažke kódu 6.8, v našom prípade trieda `DataContext` dedí triedu `IdentityDbContext`, ktorá dedí `DbContext`, ale navyše poskytuje aj užívateľskú identitu.

Databázové migrácie

Keď sa pridajú alebo odstránia nové entity alebo atribúty entít, je potrebné zodpovedajúcim spôsobom zmeniť aj databázovú schému tak, aby bola synch-

```
public class DataContext(
    DbContextOptions<DataContext> options
) : IdentityDbContext<User, IdentityRole<long>, long>(options),
    IDataContext
{
    public DbSet<Project> Projects { get; set; } = null!;

    public DbSet<Team> ProjectApprovals { get; set; } = null!;

    public DbSet<Team> Teams { get; set; } = null!;
```

Výpis kódu 6.8 Ukážka konfigurácie modelu a databázového kontextu

ronizovaná s dátovým modelom aplikácie a zachovať pritom existujúce dáta v databáze. Entity Framework Core poskytuje spôsob, ako postupne aktualizovať databázovú schému pomocou tzv. migrácií, ktoré vie automaticky vygenerovať framework, vďaka čomu nemusí migrácie písať vývojár. Na druhú stranu si myslím, že je potrebné vygenerovaným migráciám rozumieť a vedieť ich správne nakonfigurovať. Z tohto dôvodu aspoň stručne uvediem, ako fungujú. Každá migrácia je C# trieda dediaca abstraktnú triedu `Migration` a obsahuje metódu `Up`, ktorá slúži pre aktualizáciu databázovej schémy a metódu `Down`, ktorá databázovú schému dostane do pôvodného stavu. Na výpise kódu 6.9 je napríklad vidieť pridanie stĺpca `State` do tabuľky `Projects`, čo bolo potrebné kvôli schvaľovaniu projektov.

```
migrationBuilder.AddColumn<int>(
                name: "State",
                table: "Projects",
                type: "integer",
                nullable: false,
                defaultValue: 0);
```

Výpis kódu 6.9 Ukážka pridania stĺpca do tabuľky

6.1.1.3 SosApi

Projekt SosApi v kontexte čistej architektúry má rolu prezentačnej vrstvy. Je teda vstupným bodom do aplikácie a zodpovedá za prijímanie žiadostí od klienta, ktoré validuje a ďalej ich spracováva. Obsahuje `Controller` triedy, ktoré spracúvajú žiadosti klienta, `Validator` triedy, ktoré slúžia pre validáciu DTO ako aj konfiguračný súbor `Program.cs`, ktorý obsahuje spúšťací kód aplikácie

vrátane konfigurácie rôznych komponentov a služieb aplikácie.

Controller

`Controller` triedy zodpovedajú za prijímanie HTTP žiadostí od klienta, špecifikujú aké endpointy API poskytuje, aké dáta jednotlivé endpointy prijímajú a aké dáta sa vrátia ako odpoveď klientovi.

Ako je vidieť na ukážke `Controller` triedy na výpise kódu 6.10, pomocou tzv. atribútov definujeme URI identifikátor, HTTP metódu a autorizáciu endpointu ako aj aké HTTP stavové kódy a aké parametre alebo telo žiadosti očakávame a aké dáta posielame ako odpoveď. Metóda po prijatí žiadosti získa prihláseného užívateľa a zavolá príslušnú metódu v `Service` triede, kde sa vykoná potrebná biznis logika pre spracovanie žiadosti. Po úspešnom spracovaní žiadosti v `Service` triede metóda vráti príslušný HTTP stavový kód. V prípade ukážky vráti 200 aj s telom odpovede získaným z metódy `Service` triedy.

DTO Validator

Pre validáciu DTO objektov pre žiadosti používame knižnicu FluentValidation[2]. Ako je znázornené na výpise kódu 6.11, pre definíciu súboru validačných pravidiel pre nejaký konkrétny objekt je potrebné vytvoriť triedu dediacu z abstraktnej triedy `AbstractValidator<T>`, kde `T` je typ triedy, ktorý chceme validovať. Samotné pravidlá validácie sú definované v konštruktore tejto triedy pomocou metódy `RuleFor`, ktorá príjima lambda výraz indikujúci atribút, ktorý chceme validovať.

6.1.2 Dependency Injection

ASP.NET Core podporuje návrhový vzor „dependency injection“ (v preklade vkladanie závislostí), čo je technika na dosiahnutie tzv. inverzie kontroly (z anglického „inversion of control“) medzi triedami a ich závislosťami. Vo všeobecnosti sa jedná o návrhový vzor, ktorý sa stará o tvorbu objektov (alebo služieb v kontexte vkladania závislostí) a ich závislostí. Závislosť vieme definovať ako objekt, na ktorom závisí iný objekt (alebo inými slovami ktorý je potrebný pre vytvorenie iného objektu). Vždy keď potrebujeme vytvoriť nejaký objekt, vieme namiesto ručného vytvárania všetkých závislostí použiť framework, ktorý automaticky skonštruuje všetky závislosti objektu, závislosti ich závislostí a celý tranzitívny uzáver závislostí. Okrem toho sa framework stará aj o likvidáciu objektu, keď už nie je potrebný. Výhodou aplikovania vkladania

[2] `https://docs.fluentvalidation.net/en/latest/`

```
[Route("/api/project-approvals/")]
public class ProjectApprovalController(
    ICurrentUser currentUser,
    IProjectApprovalService projectApprovalService
) : SecuredController
{
    [HttpPatch("{projectApprovalId}")]
    [Authorize(Roles = $"{Role.Admin},{Role.Teacher}")]
    [ProducesResponseType<ProjectApprovalResponse>(
        StatusCodes.Status200OK
    )]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    [ProducesResponseType(StatusCodes.Status403Forbidden)]
    [ProducesResponseType(StatusCodes.Status404NotFound)]
    [SwaggerOperation(
        "Decide students' request for project approval by ID"
    )]
    public async Task<IActionResult> DecideProjectApproval(
        [FromRoute] long projectApprovalId,
        [FromBody] ProjectApprovalDecisionRequest req,
        CancellationToken cancellationToken = default
    )
    {
        var teacher = await currentUser
            .GetUser(cancellationToken);

        var result = await projectApprovalService
            .DecideProjectApproval(
                projectApprovalId,
                teacher,
                req,
                cancellationToken
            );

        return Ok(result);
    }
```

Výpis kódu 6.10 Ukážka `Controller` triedy

závislostí je podpora modulárnej architektúry, v ktorej sú komponenty voľne prepojené a vysoko koherentné. [93]

ASP.NET Core má zabudovanú podporu pre vkladanie závislostí prostredníc-

```
public class ProjectApprovalRequestValidator :
    AbstractValidator<ProjectApprovalRequest>
{
    public ProjectApprovalRequestValidator()
    {
        RuleFor(req => req.Note)
            .MaximumLength(2500)
            .WithMessage(
                ValidationMessageCodes.MaxLengthExceeded
            );
    }
}
```

Výpis kódu 6.11 Ukážka `Validator` triedy

tvom rozhrania `IServiceCollection`, ktoré sa zvyčajne konfiguruje v súbore `Program.cs`. ASP.NET Core podporuje rôzne doby životnosti služieb: [93]

- **Transient** - služba je vytvorena vždy pri každom vyžiadaní z kontajnera služieb
- **Scoped** - služba je vytvorená raz na každú žiadosť klienta, na konci žiadosti sú tieto služby zlikvidované
- **Singleton** - služba je vytvorená raz a zdieľaná počas celého behu aplikácie

Na ukážke 6.12 je znázornená konfigurácia služby `FileHandler` pre prácu so súbormi ako `Singleton` služby.

```
var destinationFolder = builder
    .Configuration
    .GetSection("FileSaver")["destinationFolder"];

if (!string.IsNullOrEmpty(destinationFolder))
{
    builder.Services.AddSingleton<IFileHandler>(provider =>
        new FileHandler(destinationFolder)
    );
}
```

Výpis kódu 6.12 Ukážka konfigurácie `Singleton` služby

Na ukážke 6.13 je znázornená konfigurácia služby `DataContext` pre prístup k databáze a služby `ProjectApprovalService` ako `Scoped` služieb.

```
builder.Services.AddScoped<IDataContext, DataContext>();
builder.Services.AddScoped<
    IProjectApprovalService, ProjectApprovalService
>();
```

Výpis kódu 6.13 Ukážka konfigurácie `Scoped` služieb

6.1.3 Autorizácia a autentifikácia

Keďže každý užívateľ aplikácie potrebuje vidieť rôzne dáta a vykonávať rôzne akcie na základe svojej role, je nutné, aby bol užívateľ pri každej žiadosti na backend overený a autorizovaný. Autorizácia užívateľov je v prípade tejto aplikácie realizovaná pomocou JSON webového tokenu (JWT).

JWT je štandardom pre prístupové tokeny, ktorý definuje kompaktný a samostatný spôsob bezpečného prenosu informácii medzi dvomi stranami vo forme JSON objektu. JWT sa mimo iného primárne používajú práve k overeniu a autorizácii užívateľov aplikácií. [94, 95]

Keď sa užívateľ úspešne prihlási pomocou svojich prihlasovacích údajov, je na backende vygenerovaný JWT, ktorý je vrátený ako odpoveď frontendu. Frontend si vrátený token uloží pomocou cookie a každá následná žiadosť pre prístup k chráneným zdrojom obsahuje tento JWT zvyčajne v HTTP hlavičke „Authorization“.

Je ale nutné podotknúť, že v budúcnosti bude prihlásenie do systému realizované výhradne pomocou ČVUT alebo Google účtu s využitím protokolu OAuth 2.0[3], ktorý ale vyššie vysvetlený JWT často využíva. Tento protokol umožňuje užívateľom udeliť prístup k ich dátam v aplikácii bez potreby zdieľania svojich prístupových údajov s danou aplikáciou. [96].

6.2 Frontend

V tejto sekcii čitateľovi ukážem, ako som implementoval frontend. Predstavím štruktúru Nuxt aplikácie, vysvetlím ako funguje smerovanie a zobrazovanie jednotlivých stránok, ako členíme a používame jednotlivé komponenty, ako funguje komunikácia s backendom, ako funguje autorizácia a autentifikácia a ako funguje internacionalizácia a lokalizácia.

[3] `https://oauth.net/2/`

6.2.1 Smerovanie

Každému Vue súboru v priečinku `pages/` prísluší práve jedna URL adresa, ktorá zobrazuje obsah daného Vue súbora. Tieto súbory sa nazývajú stránky alebo *pages* a sú to jednoducho Vue komponenty, ktoré bližšie vysvetľujem v časti 6.2.2. Pre tvorbu dynamických a vnorených adries sú použité definované konvencie pre pomenovávanie, ako je znázornené na obrázku 6.2, na ktorom je vidieť adresárová štruktúra stránok v priečinku `pages/` v našej aplikácii.

```
pages/
├── projects/
│   ├── [id]/
│   │   ├── index.vue
│   │   └── settings.vue
│   ├── create.vue
│   └── index.vue
├── index.vue
└── login.vue
```

Obr. 6.2 Adresárová štruktúra priečinku `pages/`

6.2.2 Komponenty

Komponenty alebo tzv. *Components* sú nezávislé, izolované a opakovateľne použiteľné časti užívateľského rozhrania. Pre Nuxt aplikáciu je bežné, že je usporiadaná do stromu vnorených komponentov. Komponenty vo frameworku Nuxt sú Vue súbory nachádzajúce sa v priečinku `components/` dodržujúce tzv. `Single-File Component` (SFC). SFC je špeciálny formát súborov, ktorý má syntax podobnú HTML a umožňuje zapuzdriť šablónu, logiku, štýl ako aj lokalizáciu komponenty v jednom súbore. SFC komponenta sa typicky skladá z troch hlavných blokov `<template>`, `<script>`, `<style>`. Komponenty vo frameworku Nuxt sú automaticky k dispozícii v celej aplikácii a nie je ich teda potrebné explicitne importovať. Blok `<template>` je znázornený na výpise kódu 6.14. Na výpise kódu 6.15 je ukázaný blok `<script>`, v ktorom sa nachádza logika frontend aplikácie. Blok `<style>` používame veľmi zriedkavo, pretože pre definíciu štýlu prvkov používame framework Tailwind CSS. Je možné pridať aj ďalšie bloky, internacionalizácia a lokalizácia v našej aplikácie je napríklad definovaná v bloku `<i18n>`, ktorý je znázornený na výpise kódu 6.16.

```
<template>
  <div class="flex flex-grow flex-col">
    <h2>{{ t("teamMembers") }}</h2>
    <UTable :rows="tableRows" :columns="teamMembersCols">
      <template #name-data="{ row }">
        {{ row.name.value }}
      </template>
      <template #empty-state>
        <div class="
          flex flex-col items-center justify-center gap-3 py-6
        ">
          <span class="text-sm italic">
            {{ t("noTeamMembers") }}
          </span>
        </div>
      </template>
    </UTable>
  </div>
</template>
```

Výpis kódu 6.14 Ukážka bloku `<template>`

```
1  const pendingJoinRequestsUsers = computed(() =>
2    props.membershipRequests
3      .filter(
4        (request) =>
5          request.type === "JoinRequest" &&
6          request.state === "Pending",
7      )
8      .map((request) => request.member),
9  );
```

Výpis kódu 6.15 Ukážka časti bloku `<script>`

6.2.2.1 Props

Pre predávanie dát do komponentov sa vo Vue.js používajú tzv. *props*[4]. Jedná sa o atribúty, vďaka ktorým vieme predávať dáta z rodiča do danej komponenty. Pre definíciu *props* používame makro `defineProps()`. Vďaka použitiu jazyka TypeScript vieme *props* určiť pri definícii aj typ. Definícia *props* v komponente `ProjectAssignees.vue` je znázornená na ukážke 6.17. Použitie

[4] Z anglického „properties“, v preklade teda „vlastnosti“

```
<i18n lang="yaml">
cs:
  requestedToJoin: "Požádal o připojení"
  requestedToLeave: "Požádal o opuštení"
  invited: "Pozván"
en:
  requestedToJoin: "Requested to join"
  requestedToLeave: "Requested to leave"
  invited: "Invited"
</i18n>
```

Výpis kódu 6.16 Ukážka bloku `<i18n>` pre lokalizáciu

definovaných `props` je znázornené už na ukážke 6.15 na riadkoch 2 a 11.

```
type ProjectAssigneesProps = {
  teamMembers: UserBriefResponse[];
  membershipRequests: TeamMembershipRequestResponse[];
  totalMaximumPoints: number;
};

const props = defineProps<ProjectAssigneesProps>();
```

Výpis kódu 6.17 Ukážka definície `props`

6.2.2.2 Dependency Injection

Zvyčajne pre predávanie dát používame vyššie definované *props*. Avšak v prípade, že máme veľký strom komponentov a nejaká hlboko vnorená komponenta potrebujeme nejaké dáta zo vzdialeného rodiča, museli by sme predávať rovnaké dáta cez všetky komponenty. Práve pre tieto potreby existujú vo Vue.js funkcie `provide` a `inject`. Ako je znázornené na výpise kódu 6.18, rodičovská komponenta môže slúžiť ako nejaký poskytovateľ závislostí pre všetky svoje potomky pomocou použitia funkcie `provide`. Ktorýkoľvek potomok potom môže tieto závislosti, poskytované nejakým komponentom vyššie v strome komponentov, používať, bez ohľadu na to, ako hlboko sa nachádza. Závislosti sú do komponenty vložené použitím funkcie `inject`, ako je znázornené na výpise kódu 6.19. Definovanie typu poskytovanej závislosti je znázornené na výpise kódu 6.20.

Je ale nutné podotknúť, že funkcie `provide` a `inject` sme začali používať, až keď to bolo skutočne potrebné v jednej z posledných iterácii v letnom semestri

2023/2024. Kód je teda potrebné dôladnejšie zrefaktorovať a v prípade potreby nahradiť používanie `props` práve funkciami `provide` a `inject`.

```
const { data, refresh } = await useApiFetch<
  ProjectDetailResponse
>(ApiPath.Projects.Detail(projectId));

provide(projectAssigneesKey, data.value.team.members);
provide(projectTeacherKey, data.value.responsibleTeacher);
```

Výpis kódu 6.18 Ukážka použitia funkcie `provide()`

```
const teacher = inject(projectTeacherKey) as UserBriefResponse;
const assignees = inject(projectAssigneesKey) as
  UserBriefResponse[];
```

Výpis kódu 6.19 Ukážka použitia funkcie `inject()`

```
export const projectTeacherKey = Symbol(
  "Project teacher",
) as InjectionKey<UserBriefResponse>;

export const projectAssigneesKey = Symbol(
  "Project assignees"
) as InjectionKey<UserBriefResponse[]>;
```

Výpis kódu 6.20 Ukážka definície typu závislosti

6.2.2.3 Emits

Emits sú vo Vue.js spôsobom, ako môžu komponenty posielať správy svojim rodičom. Vďaka tomu môžu tieto komponenty upozorniť rodičov, keď sa niečo stane, bez toho, aby priamo zmenili akékoľvek dáta. Komponenty teda nemusia priamo manipulovať so stavom rodičovských komponentov, ale napriek tomu môžu rodiča informovať o určitých akciách alebo zmenách. Definovanie *emits* je znázornené na výpise kódu 6.21. Použitie definovanej funkcie je znázornené v ukážke 6.22 na riadku 21 v rámci volania API pre pridanie asistentov vyučujúceho.

```
const emit = defineEmits<{
  (event: "refresh"): void;
}>();
```

Výpis kódu 6.21 Ukážka definície `emits`

```
try {
  await $api<ProjectDetailResponse>(
    ApiPath.Projects.Detail(props.project.id),
    {
      method: "PUT",
      body: requestBody,
    },
  );
  add({
    color: "success",
    title: t("assistantsAdded"),
  });
  emit("refresh");
  formState.value.assistants = [];
} catch (e) {
  add({
    title: t("couldNotAddAssistant"),
    color: "error",
  });
} finally {
  isLoading.value = false;
}
```

Výpis kódu 6.22 Ukážka použitia definovaných `emits` a odosielania žiadosti na backend

6.2.3 Komunikácia s backendom

Pre komunikáciu s backendom a získavanie dát z backendu je možné použiť viacero možností, ktoré framework Nuxt ponúka. My používame funkciu alebo tzv. *composable* `useFetch` ako aj *ofetch*[5] knižnicu, ktorá je autoimportovaná ako alias `$fetch`. V kontexte Vue aplikácií je *composable* funkcia, ktorá zapuzdruje a ponúka opätovné využitie stavovej logiky aplikácie, teda logiku, ktorá zahŕňa správu nejakého stavu, ktorý sa v priebehu času mení. *Composable* `useFetch` je vo frameworku Nuxt najjednoduchší spôsob získavania dát,

[5] `https://github.com/unjs/ofetch`

ktorý využívame na získanie potrebných dát pri prvotnom načítaní stránky. Toto *composable* ďalej obaľujeme do vlastnej funkcie `useApiFetch`, ktorá navyše nastaví hlavičky odosielanej žiadosti ako napríklad JWT pre autorizáciu prihláseného užívateľa a spracováva prípadne chyby. Použiť naopak `$fetch` je vhodné pre odosielanie žiadostí na základe interakcie užívateľa. `$fetch` ďalej obaľujeme do vlastného Nuxt pluginu `$api`, ktorý navyše nastaví potrebné hlavičky vrátane JWT. Implementácii funkcie `useApiFetch` ako aj pluginu `$api` som sa ale nevenoval ja, ale kolega Jan Mrázek, preto ich v mojej práci nebudem uvádzať. Použitie funkcie `useApiFetch` pre načítanie dát o projekte na stránke `pages/projects/[id]/index.vue` je znázornené na výpise kódu 6.23. Použitie `$api` pre odoslanie žiadosti pre pridanie asistentov je znázornené na výpise kódu 6.22 na riadku 2.

```
const { data, refresh } = await useApiFetch<
  ProjectDetailResponse
>(ApiPath.Projects.Detail(projectId));
```

Výpis kódu 6.23 Ukážka použitia `useApiFetch` pre získanie dát o projekte s daným ID

Endpointy API poskytované backendom sú na frontende definované v súbore `apiPaths.ts`, ktorého časť je vidieť na výpise kódu 6.24. Použitie definovaných endpointov je znázornené na výpise kódu 6.22 na riadku 3 alebo tiež na výpise kódu 6.23.

Vďaka výberu jazyka TypeScript vieme definovať typy pre telá žiadostí aj odpovedí, čo nám uľahčuje prácu, zvyšuje čitateľnosť kódu a pomáha skôr nájsť chyby. Definícia typov `ProjectDetailResponse` a `ProjectRequest` je znázornená na výpise kódu 6.25.

6.2.4 Autorizácia a autentifikácia

Ako už uvádzam vyššie v časti 6.1.3, pre prístup k chráneným zdrojom a dátam vyžaduje backend JWT v hlavičke HTTP žiadosti. Po zadaní prihlasovacích údajov v prihlasovacom formulári sú tieto údaje odoslané na backend, ktorý v prípade úspešného overenia odošle v odpovedi JWT, ktorý frontend následne uloží pomocou *cookie*. Implementácii prihlasovania a uloženia JWT sa tiež venoval kolega Jan Mrázek.

Pre overenie užívateľa na frontende napríklad pre podmienené zobrazovanie určitých komponentov alebo autorizáciu rôznych akcií používame vlastné *composable* funkcie, ktoré na základe uloženého JWT nájdu prihláseného užívateľa a overia potrebné dáta o danom užívateľovi. Na výpise kódu 6.26 je napríklad znázornená funkcia, ktorá overí, či je prihlásený užívateľ členom daného

```
const ProjectsApi = {
  base: `${ApiBase}/projects`,
  List: (): string => ProjectsApi.base,
  Create: (): string => ProjectsApi.base,
  Detail: (projectId: number | string): string =>
    `${ProjectsApi.base}/${projectId}`,
  Files: (projectId: number | string): string =>
    `${ProjectsApi.Detail(projectId)}/files`,
  Submissions: (projectId: number | string): string =>
    `${ProjectsApi.Detail(projectId)}/submissions`,
  AddCheckpoint: (projectId: number | string) =>
    `${ProjectsApi.Detail(projectId)}/checkpoints`,
  RequestApproval: (projectId: number | string): string =>
    `${ProjectsApi.Detail(projectId)}/approval-requests`,
};

export const ApiPath = {
  Projects: ProjectsApi,
};
```

Výpis kódu 6.24 Ukážka definície ciest k API endpointom

tímu. Použitie tejto funkcie pre zobrazenie tlačítka, ktoré slúži pre opustenie projektu, je na výpise kódu 6.27.

6.2.5 Internacionalizácia a lokalizácia

Pre internacionalizáciu a lokalizáciu aplikácie používame I18n modul frameworku Nuxt, ktorý je postavený na Vue I18n[6] plugine, ktorý ponúka jednoduchú integráciu lokalizačných funkcionalít Vue.js aplikácií. Nový systém, tak ako aj ten súčasný, podporuje dva jazyky – češtinu a angličtinu. Jazykové preferencie užívateľa sú ukladané pomocou *cookie* `i18n_redirected`. Pri zmene jazyka sa nemení URL, takže obidva jazyky používajú rovnakú schému URL adresy bez žiadnej predpony alebo prípony, ktorá by označovala zvolený jazyk. Keď užívateľ pristúpi k aplikácii, systém skontroluje, či je v *cookie* uložená jazyková preferencia. Ak sa `i18n_redirected` *cookie* v prehliadači nenájde, používa aplikácia jazyk prehliadača a preferovaný jazyk užívateľa uloží až v prípade zmeny jazyky. Preklady pre určený jazyk sa načítajú z príslušných JSON súborov v priečinku `localization/lang` pomocou tzv. *lazy loading*, čo znamená, že vždy je načítaný len práve zvolený jazyk, aby sa docielil vyšší výkon

[6] `https://kazupon.github.io/vue-i18n/`

```
export type ProjectState =
  | "Unapproved"
  | "WaitingForApproval"
  | "Approved";

export type ProjectDetailResponse = {
  id: number;
  assignment: AssignmentDetailResponse;
  team: TeamDetailResponse;
  responsibleTeacher: UserBriefResponse;
  teachersAssistants: UserBriefResponse[];
  checkpoints: CheckpointBriefResponse[];
  maxPointRedistributionPercent: number;
  strictRedistribution: boolean;
  approvals: ProjectApprovalResponse[];
  state: ProjectState;
  visible: boolean;
};

export type ProjectRequest = {
  assignmentId: string;
  teamId: string;
  teachersAssistantsIds: string[];
};
```

Výpis kódu 6.25 Ukážka definície typov pre žiadosť a odpoveď

```
const usePermissions = () => {
  const { currentUser } = useCurrentUser();

   const isTeamMember = (members: UserBriefResponse[]) => {
    return (
      members.find((assignee) =>
        assignee.id === currentUser?.id) !== undefined
    );
  };
```

Výpis kódu 6.26 Ukážka `isTeamMember` *composable*, ktorá zistí, že či je prihlásený užívateľ členom daného tímu

aplikácie skrátením počiatočného času načítania. Okrem lokalizačných JSON súborov je väčšina prekladov definovaná v komponente, kde je daný preklad

```
<UButton
  v-if="
    isTeamMember(data.team.members) &&
    !hasPendingLeaveRequest
  "
  :label="t('leaveProject')"
  icon="i-mdi-person-remove"
  color="error"
  @click="leaveProject()"
/>
```

Výpis kódu 6.27 Ukážka použitia `isTeamMember` *composable* pre podmienené zobrazenie komponenty

potrebný. Toto je docielené použitím už vyššie spomenutého `<i18n>` bloku, ktorý je vidieť na výpise kódu 6.16.

Kapitola 7

Testovanie

V tejto kapitole popíšem, ako som aplikáciu testoval. Na úvod ukážem jednotkové testy, ktorými som testoval najdôležitejšiu biznis logiku aplikácie. Následne vysvetlím, na čo slúžia integračné testy, ktoré testujú spoločné fungovanie viacerých komponentov a uvediem, kedy budú do projektu pridané. Na záver popíšem ako prebiehalo užívateľské testovanie a aké prinieslo výsledky.

7.1 Jednotkové testy

Cieľom jednotkových testov je automatické overenie funkčnosti najmenších testovateľných častí kódu, ktoré je možné logicky izolovať od zvyšku kódu. Jedná sa teda zvyčajne o metódy. Jednotkové testy by mali testovať len kód, ktorý má vývojár pod kontrolou, nemali by teda testovať problémy týkajúce sa infraštruktúry ako napríklad prístup k databáze alebo prácu so súbormi.

Na tomto projekte používame pre jednotkové testy knižnicu *xUnit*[1]. Jednotkovými testami som sa rozhodol testovať väčšinu biznis logiky, ktorá sa nachádza v entitných triedach. Ukážka jednotkového testu, ktorý očakáva úspech metódy je na výpise kódu 7.1. Naopak na výpise kódu 7.2 je znázornený jednotkový test, ktorý očakáva vyhodenie výnimku v testovanej metóde. Jednotkové testy sa podľa princípov čistej architektúry nachádzajú v oddelenom .NET projekte *SosTests*.

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

```
[Fact]
public void DecideTestAccept()
{
    var projectApproval = CreateProjectApproval();

    var projectApprovalDecisionRequest =
        new ProjectApprovalDecisionRequest
    {
        Status = DecisionStatus.Accept,
        Note = "",
    };

    projectApproval.Decide(projectApprovalDecisionRequest);

    Assert.Equal(
        ProjectState.Approved,
        projectApproval.Project.State
    );

    Assert.Equal(
        ProjectApprovalState.Accepted,
        projectApproval.State
    );
}
```

Výpis kódu 7.1 Ukážka jednotkového testu entitnej triedy

7.2 Integračné testy

Integračné testy sú automatické testy, ktoré overujú funkčnosť viacerých komponentov a ich schopnosť fungovať spoločne. Integračné testy môžu testovať aj infraštruktúru aplikácie, napríklad prístup k databáze alebo integráciu s inými systémami alebo inými API. Keďže sa jedná o časovo náročnejšie a menej prioritné testy ako jednotkové testy, prípravu pre integračné testovanie sme sa s kolegami Janom Jamnickým a Janom Mrázkom rozhodli prenechať na tím študentov predmetu BI-SP1, ktorý začal pracovať na rozvoji aplikácie v letnom semestri 2023/2024. Zapojeniu ďalších študentov do rozvoja nového systému sa bližšie venujem v kapitole 8, kde aj bližšie uvediem ich prácu týkajúcu sa integračného testovania.

```
[Fact]
public void
    DecideTestAcceptShouldThrowIfProjectApprovalIsAccepted()
{
    var projectApproval = CreateProjectApproval();
    projectApproval.State = ProjectApprovalState.Accepted;
    projectApproval.Project.State = ProjectState.Approved;

    var projectApprovalDecisionRequest =
        new ProjectApprovalDecisionRequest
    {
        Status = DecisionStatus.Accept,
        Note = "",
    };

    Assert.Throws<ProjectApprovalAlreadyDecidedException>(() =>
        projectApproval.Decide(projectApprovalDecisionRequest)
    );

    Assert.Equal(
        ProjectState.Approved,
        projectApproval.Project.State
    );

    Assert.Equal(
        ProjectApprovalState.Accepted,
        projectApproval.State
    );
}
```

Výpis kódu 7.2 Ukážka jednotkového testu entitnej triedy, ktorý očakáva vyhodenie výnimky

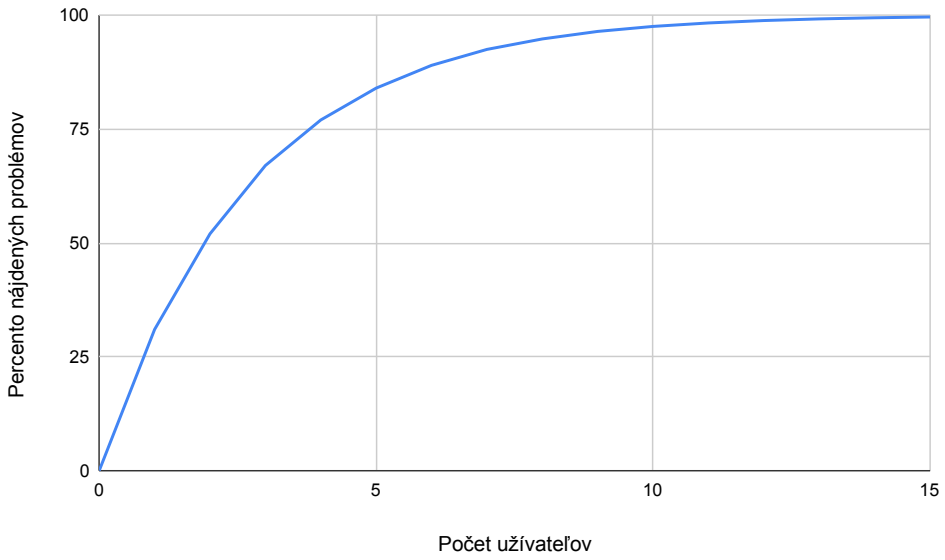
7.3 Užívateľské testovanie

Pre potreby identifikovania problémov v návrhu nového užívateľského rozhrania, odhalenia ďalších príležitostí pre zlepšenie a lepšieho poznania preferencií koncových užívateľov som sa rozhodol pre kvalitatívne užívateľské testovanie, ktoré som realizoval po implementácii väčšiny prípadov použitia. Vo všeobecnosti vieme užívateľské testovanie definovať ako istý výskum, pri ktorom výskumník požiada účastníka o vykonanie nejakých pripravených úloh pomocou špecifikovaného užívateľského rozhrania. Zvyčajne sa jedná buď o funkčnú

aplikáciu alebo Hi-Fi prototyp. Výskumník účastníka pri vykonávaní úloh detailne pozoruje a snaží sa získať jeho spätnú väzbu. [97] V tejto sekcii popíšem, ako som užívateľské testovanie vykonal, konkrétne akých som vybral účastníkov, aké testovacie scenáre som pre potreby užívateľského testovania pripravil a na záver zhodnotím, ako testovanie prebiehalo a aké výsledky prinieslo.

7.3.1 Výber účastníkov

Ako vidieť na grafe na obrázku 7.1, najviac nových nedostatkov odhalia prvý traja užívatelia, zatiaľ čo šiesty užívateľ nám nepomôže identifikovať už takmer žiadny nový problém. Práve z tohto dôvodu je odporúčané testovať s piatimi užívateľmi, na základe ich pozorovaní opraviť nájdené nedostatky a vylepšiť tak užívateľské rozhranie, ktoré následne opäť otestujeme s ďalšími piatimi užívateľmi v ďalšej iterácii. [98, 99]



Obr. 7.1 Optimálna veľkosť skupiny testovacích užívateľov pri kvalitatívnom užívateľskom testovaní. Dáta získané z článku od Jakoba Nielsena.

Keďže cieľom systému SOS je byť univerzálny a použiteľný pre viaceré predmety na FIT ČVUT ako aj mimo neho, vybral som pre užívateľské testovanie nasledovných piatich užívateľov:

- Vyučujúci predmetov BI-SP1, BI-SP2 a NI-NUR na FIT ČVUT
- Učiteľ na Bilingválnom gymnáziu C. S. Lewisa[2] v Bratislave

[2] `https://bilgym.sk/`

- Študent posledného ročníka bakalárskeho štúdia na FIT ČVUT, ktorý absolvoval predmety BI-SP1 a BI-SP2 s použitím súčasného systému SOS, ako aj ďalšie povinné predmety, ktoré by v budúcnosti mohli systém SOS využívať
- Študent magisterského štúdia na FIT ČVUT, ktorý absolvoval predmet NI-NUR s použitím súčasného systému SOS a predmety BI-SP1 a BI-SP2 bez použitia súčasného systému
- Absolvent bakalárskeho štúdijného programu *Softwarové inženýrství a technologie* na Fakulte elektrotechnické ČVUT

7.3.2 Testovacie scenáre

Pre účely užívateľského testovania som pripravil nasledujúce testovacie scenáre, ktoré účastníci vykonávali. Pripravené scenáre sú aplikovateľné aj na ďalšie iterácie testovania v budúcnosti, keďže sa väčšinou jedná o všeobecné úlohy, ktoré spájajú viaceré prípady použitia.

7.3.2.1 Testovacie scenáre pre vyučujúceho

Definícia témy semestrálnej práce I

Chcete definovať tému semestrálnej práce pre študentov s názvom „Môj prvý projekt“, ktorého cieľom je zlepšiť celkovú použiteľnosť portálu SOS pre všetky užívateľské role. K projektovej téme chcete priložiť aj dve prílohy — súbory „priloha1.pdf“ a „priloha2.pdf“ v priečinku „Downloads“. Chcete, aby sa študenti mohli sami hlásiť, ak ich téma práce zaujme.

Definícia témy semestrálnej práce II

Nedávno ste vytvorili tému semestrálnej práce s názvom „Vývoj portálu SOS“. Cieľom tejto semestrálnej práce je vytvoriť webovú aplikáciu pre odovzdávanie tímových semestrálnych práci. Na projekte už nejakí študenti začínajú pracovať. Ďalší vaši študenti (Josef Student-90, Josef Student-91, Josef Student-92) tiež prejavili záujem pracovať na tomto projekte. Tím pracujúci na tomto projekte už však naplnil kapacitu a pridaním troch ďalších študentov do tímu by bol tím už veľký nad rámec limitu predmetu. Chcete preto, aby na tejto práci pracovali dva tímy súčasne. Vytvoríte teda druhý projekt s rovnakým zadaním a hneď do projektu pridajte aj študentov, o ktorých viete, že majú o projekt záujem.

Odmietnuť žiadosť študentov o schválenie projektu

Skupina študentov sa dohodla, že chce spolu pracovať na projekte. Vytvorili si teda projekt s vlastným zadaním a tímom. Projekt má názov Náš študentský projekt. Pred tým, ako študenti môžu začať na projekte pracovať a začať odovzdávať prácu, je nutné, aby zadanie projektu a tím schválil vyučujúci. Študenti Vás preto požiadali o schválenie. Po kontrole zadania ste si ale všimli, že študentom chýba v zadaní špecifikácia konkrétnych výstupov projektu, na ktorých ste sa dohodli na stretnutí. Okrem zadania chcete skontrolovať aj tím, ktorí si študenti vytvorili. Tím je v poriadku, no projekt nechcete schváliť kvôli nedostatkom zadania. O nedostatkoch zadania chcete dať vedieť aj študentom.

Odmietnuť žiadosť študenta o pripojenie sa k projektu

Ste zodpovedným vyučujúcim projektu „Vývoj portálu DBS“. Študent Josef Student-42 požiadal o pripojenie sa k projektu. Študenta ale nepoznáte a neviete, či rozumie technológiám, ktoré sa na projekte používajú. Žiadosť študenta preto odmietnete.

Pridať ďalších študentov do projektu

Ste zodpovedným vyučujúcim projektu „Vývoj portálu DBS“. So študentami, ktorí pracujú na tomto projekte máte práve stretnutie. Študenti vám oznámili, že ak by bolo potrebné posilniť tím o ďalších kvalitných vývojárov, tak majú ďalších dvoch spolužiakov, ktorí by sa k nim radi pridali a pracovali spolu s nimi na tomto projekte. Študenti im ale nemôžu poslať pozvánku, pretože im to systém v tomto predmete neumožňuje. Do projektu teda pridáte študentov Josef Student-80 a Josef Student-81.

7.3.2.2 Testovacie scenáre pre študenta

Definícia vlastnej témy semestrálnej práce a následné požiadanie o schválenie

So spolužiakmi (Josef Student-90, Josef Student-91, Josef Student-92) ste sa dohodli, že budete spolu pracovať na semestrálnej práci. Cieľom vašej práce bude vytvoriť webovú aplikáciu pre odovzdávanie tímových semestrálnych prác. Následne ste sa s vašim vyučujúcim dohodli, že pripravíte Lo-Fi prototypy ešte pred prácou na projekte, ktoré uvediete ako prílohy „príloha1.pdf“ a „príloha2.pdf“ v priečinnku „Downloads“. Následne požiadate vyučujúceho o schválenie vytvoreného zadania práce.

Pozvanie ďalšieho študenta

Nedávno ste si spolu so skupinou spolužiakov vytvorili projekt s názvom „Aplikácie pre správu tímov“. Na projekte ste ešte nezačali pracovať. Napadlo vás, že by sa k vám mohol pridať ďalší váš spolužiak, ktorý má dlhoročné skúsenosti s technológiami, s ktorými budete projekt vyvíjať. Rozhodnete sa teda spolužiaka Josef Student-42 do projektu pozvať.

Upravenie zadania

Nedávno ste požiadali vyučujúceho o schválenie projektu „Aplikácia pre správu športových klubov“. Vyučujúci ale ešte nie je spokojný s vaším zadaním. Upravte zadanie projektu podľa požiadaviek vyučujúceho a následne znovu požiadajte o schválenie.

7.3.3 Priebeh testovania

Prvým užívateľom, ktorý sa testovania zúčastnil bol vedúci práce, ktorý mi okrem spätnej väzby na predstavené UI dal spätnú väzbu aj na priebeh samotného testovania. Keďže sa jednalo o moju prvú skúsenosť s vedením užívateľského testovania, naučil som sa nasledujúce tri veci, ktoré som následne aplikoval na ďalších štyroch užívateľoch:

- pred samotným testovaním je potrebné užívateľovi podrobne predstaviť aplikáciu a spýtať sa na jeho skúsenosti s podobnými systémami a užívateľa tak ešte lepšie kategorizovať
- po skončení testovania je potrebné sa užívateľa spýtať na celkový dojem užívateľa z predstaveného UI a dať mu priestor pre poskytnutie ďalšej spätnej väzby týkajúcej sa jednoduchosti, prehľadnosti a intuitívnosti predstaveného UI, môže tiež prichádzať s vlastnými návrhmi na zlepšenie
- v pripravených scenároch nie je vhodné dávať užívateľovi návod, ako scenár úspešne dokončiť, scenáre som teda po testovaní s prvým užívateľom čiastočne upravil, aby žiadne indície neobsahovali

7.3.4 Výsledky testovania

Užívateľské testovanie odhalilo viaceré nedostatky, ktoré v tejto časti popíšem a určím ich závažnosť ako aj náročnosť opravy.

7.3.4.1 Terminológia systému

Ukázalo sa, že nejasná terminológia súčasného systému čiastočne pretrváva aj v tom novom. Viacerí účastníci sa zhodli na tom, že keď chcú definovať tému semestrálnej práce, tak nevedia, či chcú vytvoriť zadanie alebo projekt.

Závažnosť: Vysoká

Náročnosť: Nízka

Bol nedostatok opravený: Áno

Oprava spočívala v tom, že bol ponechaný len formulár pre tvorbu projektu, v rámci ktorého je zadanie vytvorené. Ešte pri zbere a validácii požiadaviek som zistil, že na FIT ČVUT neexistuje potreba pre vytvorenie samostatného zadania, z ktorého si sami študenti vedia tvoriť projekty. Objavil sa ale požiadavok vytvorenia zadania, ku ktorému sa zatiaľ študenti nemôžu hlásiť. Toto je ale vyriešené deaktiváciou projektu, čo skryje projekt pred študentami. V prípade požiadavky na možnosť vytvorenia zadania, z ktorého vedia študenti sami tvoriť projekty, napríklad na GJGJ, bude táto možnosť tiež v rámci jedného formuláru pre tvorbu projektu, kedy vyučujúci vyberie nejakú možnosť, ktorá toto bude umožňovať.

7.3.4.2 Čiastočne neprehľadný formulár tvorby projektu

Užívatelia sa zhodli, že možnosť výberu vytvorenia nového zadania a výberu z existujúcich zadaní nie je ihneď viditeľný. Jeden užívateľ si pri tvorbe projektu nebol istý, či vytvára projekt alebo zadanie, čo ale čiastočne súvisí s vyššie spomenutým nedostatkom 7.3.4.1 týkajúceho sa nejasnej terminológie.

Závažnosť: Vysoká

Náročnosť: Nízka

Bol nedostatok opravený: Nie

Tento nedostatok som sa rozhodol neopravovať, pretože si myslím, že bol čiastočne vyriešený a odstránený vyriešením nejasnej terminológie. V prípade, že by ale nejasnosť formulára tvorby projektu pretrvávala, navrhujem lepšie farebne odlíšiť možnosti, z ktorých si užívateľ vie vybrať a tieto možnosti užívateľovi stručne vysvetliť.

7.3.4.3 Zoznam pridávaných užívateľov

Užívateľom vadilo, že pri pridávaní študentov do projektového tímu nevidia, akých študentov pridávajú.

Závažnosť: Vysoká

Náročnosť: Nízka

Bol nedostatok opravený: Áno

Oprava spočívala vo farebnom zvýraznení vybraných študentov a zobrazenie výberu študentov vyššie, aby bola viditeľná tabuľka pridávaných užívateľov, ktorá už bola súčasťou UI.

7.3.4.4 Neviditeľný stav projektu

Užívatelia sa zhodli v tom, ze je pri zobrazení zoznamu projektov nejasne viditeľné, v akom stave sa projekt nachádza a či vyžaduje nejakú akciu užívateľa, napríklad schválenie projektu alebo ohodnotenie odovzdanej práce.

Závažnosť: Vysoká

Náročnosť: Nízka

Bol nedostatok opravený: Áno

Po opravení užívatelia vidia užívatelia farebne zvýraznený stav projektu, ktorý očakáva ich akciu.

7.3.4.5 Filtrovanie projektov

Viacerým užívateľom chýbala možnosť filtrovať projekty podľa názvu alebo stavu, napríklad projekty čakajúce na schválenie vyučujúcim.

Závažnosť: Stredná

Náročnosť: Stredná

Bol nedostatok opravený: Áno

Po užívateľskom testovaní som pridal možnosť jednoduchého filtrovania projektov podľa názvu, člena tímu, zodpovedného vyučujúceho a stavu projektu, teda napríklad, či projekt čaká na schválenie alebo čaká na ohodnotenie.

7.3.4.6 Schovaná správa tímu

Štyria užívatelia si neboli istý, kde správu tímu nájdu. Najprv ich totiž nenapadlo, že správa tímu spadá do nastavení projektu. Nakoniec všetci užívatelia správu tímu ale našli a scenár úspešne dokončili. Jeden z užívateľov, ktorý je aktívnym užívateľom súčasného systému ale ocenil, že všetky nastavenia týkajúce sa projektu sú na jednom mieste na rozdiel od súčasného systému, kde

pre správu tímu a iné nastavenia projektu netýkajúce sa projektu, existujú dve obrazovky.

Závažnosť: Nízka

Náročnosť: Stredná

Bol nedostatok opravený: Nie

Nedostatok nebol odstránený, pretože je potrebné detailnejšie navrhnúť jeho opravu a návrh následne zvalidovať s užívateľmi. Jeden z mojich návrhov je užívateľov informovať, na čo nastavenia projektu slúžia a čo všetko je v rámci projektu konfigurovateľné. Druhým návrhom je nemať oddelenú obrazovku pre nastavenia, ale umožniť užívateľom konfigurovať projekt v rámci zobrazenia projektu.

7.3.4.7 Možnosť požiadať o schválenie projektu rovno v rámci tvorby projektu

Jeden zo študentov navrhol, že by mohlo byť možné požiadať o schválenie projektu rovno v rámci tvorby projektu. Vo formuláre tvorby tímu by si teda študent mohol zvoliť, či chce požiadať o schválenie projektu. Tento návrh som ale nerealizoval, implementácia síce bude jednoduchá, no je najprv potrebné overiť, či je to naozaj potrebné a či to nebude pre užívateľov mätúce.

Závažnosť: Nízka

Náročnosť: Nízka

Bol nedostatok opravený: Nie

7.3.4.8 Nahrávanie súborov

Užívatelia odhalili viacero menších nedostatkov pri nahrávaní súborov, týmto nedostatkom sa ale bližšie venuje vo svojej práci kolega Jan Mrázek.

Kapitola 8

Zapojenie ďalších študentov

V tejto kapitole sa zameriavam na tímovú spoluprácu na projekte a zapojenie ďalších študentov do rozvoja nového systému. Popisujem, ako sme s tímom spolupracovali na vývoji systému, ako sme projekt pripravili pre zapojenie ďalších študentov a ukážem úlohy, na ktorých pracovali študenti predmetu BI-SP1, ktorí sa do rozvoja projektu zapojili v letnom semestri 2023/2024.

8.1 Tímová spolupráca

Na novom systéme sme od začiatku vývoja pracovali v tíme s kolegami Timotejom Adamcom, Janom Jamnickým a Janom Mrázkom, pričom spolu s kolegami Janom Jamnickým a Janom Mrázkom pracujeme na rozvoji nového systému aj v rámci našich bakalárskych prác. V zimnom semestri zastával kolega Timotej Adamec v tíme rolu vedúceho. V letnom semestri s nami kolega Adamec už nespolupracoval a rolu vedúceho som sa snažil suplovať spoločne s kolegami Janom Jamnikcým a Janom Mrázkom. Tento tím s kolegami Janom Jamnickým a Janom Mrázkom budem ďalej v texte označovať ako „BP tím“. Ešte pred samotným vývojom novej aplikácie na začiatku zimného semestra 2023/2024 sme spolu s kolegami z BP tímu a kolegom Timotejom Adamcom museli pripraviť projekt tak, aby bolo možné a jednoduché na vývoji pracovať spoločne v tíme. V prvom rade bolo potrebné rozdeliť systém na tri čo najmenej súvisiace doménové oblasti. V prípade nadväznosti našich častí práce bolo potrebné komunikovať a spolupracovať a spoločne sa dohodnúť na postupe. Ďalej bolo nutné vytvoriť repozitáre projektu na GitLabe a dohodnúť sa na ďalších procesoch súvisiacich s tímovou spoluprácou pri vývoji softwaru. Okrem nižšie uvedených pravidiel som spolu s kolegami z BP tímu vybral vhodný verzovací model. Výberu verzovacieho modelu vrátane jeho úpravy v jednej z iterácií som sa venoval podrobnejšie už v sekcii 5.2.

8.1.1 Zdieľanie zdrojových kódov

Zdrojové kódy oboch aplikácií sú uložené v dvoch GitLab repozitároch dostupných na adrese `https://gitlab.fit.cvut.cz/sos/new-sos/backend` a `https://gitlab.fit.cvut.cz/sos/new-sos/frontend`. Oba repozitáre sú v skupine *New SOS*[1], ktorá je podskupinou skupiny *SOS*[2], ktorá okrem iného obsahuje aj repozitár súčasného systému ako aj repozitár pre užívateľskú dokumentáciu súčasného systému.

8.1.2 Pomenovávanie vetví a Merge Requestov

Pri práci na projekte pomenovávame Git vetve nasledovne:
`<typ>/<ID-redmine-úlohy>-<krátky-popis>`, kde

- `<typ>` označuje účel vytvorenia vetve a môže byť jeden z nasledujúcich:
  - `feature` pre implementáciu nových funkcionalít
  - `fix` pre opravu chýb
  - `refactor` pre refaktoring existujúceho kódu
  - `docs` pre tvorbu a úpravu dokumentácie
  - v prípade potreby môžu byť pridané ďalšie typy, ale je odporúčané použiť jeden z vyššie uvedených
- `<ID-redmine-úlohy>` je ID príslušnej úlohy v systéme Redmine, ktorý používame pre správu úloh, tvorbu podúloh, odhadovanie časovej náročnosti ako aj pre vykazovanie času stráveného na úlohách
- `<krátky-popis>` je stručné zhrnutie úlohy v angličtine, ktoré slúži pre vysvetlenie toho, čo daná vetva implementuje

Po dokončení implementácie v rámci vetve je vytovrený *Merge Request* (MR) v systéme GitLab. Jedná sa o akýsi návrh na začlenenie zmien zo zdrojev do cieľovej vetve. MR typicky obsahuje všetky zmeny kódu v danej vetvi, komentáre kolegov z tímu k jednotlivým zmenám, informácie o CI/CD procesoch a zoznam Git commitov.

Merge Requesty pomenovávame nasledovne:
`RM-<ID-redmine-úlohy> <krátky-popis>`, kde

- `<ID-redmine-úlohy>` je ID príslušnej úlohy v systéme Redmine
- `<krátky-popis>` je stručné zhrnutie úlohy v angličtine, ktoré slúži pre vysvetlenie toho, aké zmeny daný MR prináša

[1] https://gitlab.fit.cvut.cz/sos/new-sos
[2] https://gitlab.fit.cvut.cz/sos

8.1.3 Revízie kódu

Súčasťou tímového vývoja softwaru je aj revízia kódu alebo tzv. *code review*. Vo všeobecnosti sa jedná o vzájomnú kontrolu kódu, ktorej účelom je zaistiť vyššiu kvalitu kódu a jednoducho a rýchlo identifikovať chyby ešte pred zlúčením kódu do vyššej vetvy, napríklad do vetvy *master* a následným nasadením zmien. Okrem toho tieto kontroly pomáhajú vývojárom poznať zdrojový kód celej aplikácie, nie len častí kódu, ktoré napísali oni a zdieľať znalosti a skúsenosti v rôznych programovacích jazykoch a frameworkoch.

V oboch repozitároch sme na začiatku vývoja zaviedli pravidlo, že každý MR musí mať minimálne dvoch členov BP tímu, ktorí skontrolujú kód a po opravách či diskusii s riešiteľom MR následne schvália zmeny. Okrem toho každú zmenu na backende skontroloval aj asistent projektu Bc. Max Hejda. V priebehu letného semestra sme sa ale z časových dôvodov rozhodli pre upravenie tohto pravidla na minimálne jedno schválenie navrhovaných zmien, pričom pri práci na backend aplikácii ostalo pravidlo schválenia kolegom Bc. Maxom Hejdom.

Ďalším pravidlom, ktoré sme zaviedli na začiatku letného semestra, kedy sa k nám pripojil SP tím, bola nutnosť kontroly kódu minimálne jedného vývojára práve z BP tímu alebo asistenta projektu Bc. Maxa Hejdu a nie vývojára z SP tímu. Dôvodom tohto rozhodnutia a výberu konkrétnych vývojárov bolo, že okrem toho, že poznáme aj súčasný systém SOS, tak sme boli od začiatku pri vzniku toho nového a poznáme teda celý kód frontendu aj backendu.

8.2 Príprava projektu pre zapojenie ďalších študentov

V letnom semestri 2023/2024 sa do vývoja nového systému zapojili ďalší študenti, konkrétne päťčlenný tím študentov predmetu BI-SP1 vedený Ing. Jiřím Hunkom (ďalej len „SP tím“). Pred začiatkom semestru bolo teda nutné pripraviť projekt do takého stavu, aby bolo pre ďalších študentov jednoduché na projekte pracovať. Pripravili sme stručnú a zrozumiteľnú dokumentáciu pre vývojárov pozostávajúcu hlavne z viacerých dôležitých informácií o systéme SOS a jeho vývoji. Cieľom poskytnutia týchto informácii v zrozumiteľnej forme bolo uľahčenie práce SP tímu na projekte a prvotné vniknutie do novej aplikácie ako aj celej problematiky vývoja systému SOS. Táto dokumentácia obsahuje stručné predstavenie súčasného aj nového systému vrátane užívateľskej dokumentácie súčasného systému, výstupov z rozhovorov s užívateľmi súčasného systému, dôvody pre vývoj nového systému, vybrané technológie a architektúru ako aj slovníček pojmov nového systému. Študentom sme tiež poskytli vybrané a relevantné diagramy, na ktorých sme pracovali v predmete

BI-SP1 my vrátane konceptuálneho ER modelu a diagramu prípadov použitia. Okrem iného sú súčasťou tejto dokumentácie pokyny a usmernenia, ako pri vývoji backend aj frontend aplikácie postupovať. Tieto pokyny obsahujú napríklad konvencie pomenovania tried, komponentov, Git vetví a Merge Requestov v oboch GitLab repozitároch, fungovanie databázových migrácií, fungovanie lokalizácie na frontende a vysvetlenie používaného verzovacieho modelu. Okrem toho dokumentácia obsahuje aj inštrukcie pre lokálne spustenie backend aj frontend aplikácie vrátane inštalácie potrebných nástrojov a odporúčaných vývojárskych prostredí.

V rámci vývoja backendu som spolu s kolegami tiež nezanedbal prípravu API dokumentácie. API dokumentácia obsahuje potrebné informácie pre použitie REST API. Pre tvorbu API dokumentácie sme použili nástroj Swagger[3]. Ako popisujem už v časti 5.7, API dokumentácie je dostupná na Pages-FIT.

Okrem pripravenia dokumentácie sme s SP1 tímom na začiatku semestra mali krátke stretnutie, ktorého cieľom bolo tímu predstaviť súčasný ako aj nový systém, vysvetliť časovú os vývoja súčasného systému a dôvody a motiváciu pre vývoj nového systému a dohodnúť sa na spolupráci v letnom semestri. SP1 tím, ktorý sa skladá zo študentov Františka Holého, Matěja Hrbáčka, Matěja Procházku, Oldřicha Macháčka a Sašy Vobořila, na stretnutí prejavil záujem projektu naozaj pomôcť a aktívne sa zapojiť do jeho rozvoja. Dohodli sme sa teda, že v priebehu ďalších týždňov od stretnutia vymyslíme úlohy, na ktorých by mohli pracovať a naozaj tak pomôcť rozvoju nového systému.

8.3 Práca tímu študentov predmetu BI-SP1

SP tím pracoval počas letného semestra na viacerých úlohách týkajúcich sa požiadaviek predmetu BI-SP1 a ďalších úlohách, ktoré im zadal vyučujúci Ing. Jiří Hunka alebo jeho asistent Bc. Max Hejda. Okrem toho sme ale aj po vzájomnej dohode s SP tímom spolupracovali, komunikovali, odpovedali na ich otázky a zadávali im úlohy, ktoré v tejto časti bližšie špecifikujem.

8.3.1 Filter Bundle

Jedná sa o vyčlenený .NET projekt, ktorý bude slúžiť ako knižnica pre jednoduché filtrovanie dát z databázy. Knižnica bude okrem nového systému SOS použitá aj v DBS portále, ktorý sa tiež začal prepisovať do rovnakých technológií. Je potrebné podotknúť, že keďže sa jedná o náročnejšiu a komplexnejšiu úlohu,

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

knižnica ešte nie je dokončená a SP tím na jej vývoji ešte stále pracuje. Repozitár knižnice *Filter Bundle* je takisto na fakultnom GitLabe.

8.3.2 Analýza scenárov pre schvaľovanie žiadostí o vstup do tímu

Zo zberu a analýzy požiadaviek vyplynulo, že by vyučujúci uvítali možnosť navýšiť kapacitu tímu pri schvaľovaní žiadostí o pripojenie sa k projektu, aby vedeli prijať všetkých študentov, ktorí chcú pracovať na projekte a požiadali o pripojenie sa. V súčasnom systéme je možné navyšovať kapacitu tímu v nastaveniach projektu, cieľom je teda do nového pridať funkcionalitu rýchleho zvýšenia kapacity v rámci schvaľovania žiadosti. Cieľom tejto konkrétnej úlohy bolo analyzovať možné AS-IS scenáre a rôzne krajné prípady v súčasnom systéme. Na základe tejto analýzy bude neskôr navrhnuté, ako chceme k tomu problému pristúpiť vrátane návrhu UI a biznis logiky potrebnej pre implementáciu tejto požiadavky.

Výstupom úlohy je prehľadná a pomerne detailná analýza scenárov. Tím najprv analyzoval možnosť, ako je v súčasnom systéme možné navýšiť kapacitu tímu. Tím si všimol jedného zo známych nedostatkov súčasného systému a to, že toto nie je možné uskutočniť v nastaveniach tímu, ale v nastaveniach projektu, čo študentom z tímu prišlo mätúce ako aj viacerým užívateľom súčasného systému. Ďalej študenti analyzovali nasledujúce štyri scenáre:

- možnosť podať žiadosť do tímu, keď je tím plný
- možnosť podať žiadosť do tímu, keď je už podaný rovnaký počet žiadostí ako kapacita tímu
- možnosť podať žiadosť do tímu, keď je počet už podaných žiadostí spolu s počtom členom tímu rovnaký ako kapacita tímu
- možnosť pridať študenta do plného tímu

8.3.3 Vývojárska dokumentácia

Študenti SP tímu podrobne zdokumentovali potrebné časti zdrojového kód na backende v rámci MR !57. Vývojárska dokumentácia je nasadená na PagesFIT spoločne s dokumentáciu API, na ktorej spoločne so mnou a kolegami z BP tímu tiež pracoval aj SP tím a túto dokumentáciu vylepšil a dokončil.

8.3.4 Integračné testovanie

SP tím v rámci MR !80 implementoval potrebnú infraštruktúru pre integračné testovanie na backende spolu s ukážkou použitia. Ďalšími úlohami buď SP tímu alebo tímu v predmete BI-SP2 v nasledujúcom zimnom semestri bude napísať integračné testy kľúčových častí backend aplikácie.

8.3.5 E-mailové notifikácie

V rámci MR !73 pripravil SP tím podporu jednoduchého rozosielania e-mailových notifikácií.

8.3.6 Prihlásenie pomocou ČVUT a Google účtu

V rámci MR !73 pracuje SP tím na podpore prihlásenia pomocou ČVUT účtu pre potreby produkčného využitia na FIT ČVUT a v rámci MR !73 na podpore prihlásenia pomocou Google účtu pre potreby produkčného využitia v iných prostrediach mimo ČVUT.

Kapitola 9

Výsledky práce a stav nového systému

V poslednej kapitole mojej práce stručne zhrniem výsledky práce a stav nového systému. Ukážem čitateľovi, kde je nový systém nasadený a kde sa nachádza dokumentácia nového systému vrátane vývojárskej dokumentácie a dokumentácie API. Uvediem, aké ďalšie kroky je nutné vykonať pre ďalší rozvoj systému a pre možné nahradenie starého systému tým novým v predmetoch na FIT ČVUT, ktoré momentálne systém SOS používajú. Na záver zhodnotím iteratívnu formu vývoja, ktorú sme spolu s kolegami z tímu pri vývoji aplikovali a uvediem, aké výhody a nevýhody nám tento štýl vývoja priniesol.

V rámci mojej práce a práce kolegov z tímu – Jana Jamnického a Jana Mrázka – bola pomocou iteratívneho vývoja vytvorená nová webová aplikácia s novým minimalistickým UI, ktoré bolo validované v rámci prvej iterácie užívateľského testovania. Niektoré nedostatky, ktoré vyplynuli z užívateľského testovania, boli ihneď opravené. V rámci vývoja sme si s kolegami navzájom kontrolovali kód a každá zmena prešla revíziou a schválením aspoň jedného ďalšieho člena tímu, ktorý danú zmenu neimplementoval. Okrem toho každú zmenu na backende schválil kolega Bc. Max Hejda, ktorý má s platformou .NET viac skúseností ako ja alebo kolegovia z tímu a podieľal sa na vývoji súčasného systému, takže dôkladne rozumie všetkým funkčným požiadavkám. Kód je teda prehľadný a udržateľný, vďaka čomu je projekt pripravený na ďalší rozvoj. Aplikácia je nasadená na testovacom prostredí na adrese `https://new.sos2.ksi.fit.cvut.cz/`. Bola vytvorená pomerne obsiahla dokumentácia uľahčujúca prácu na projekte ďalším vývojárom. Dokumentácia obsahuje pokyny o dodržiavaní konvencií pri písaní kódu a tímovej kolaborácii, postupy, ako spustiť aplikáciu pre potreby lokálneho vývoja, vývojársku

dokumentáciu vysvetľujúcu potrebné časti kódu a v neposlednom rade dokumentáciu API. Dokumentácia je nasadená na PagesFIT. Repozitáre projektu sa nachádzajú na fakultnom GitLabe v skupine *New SOS*. Vo vývoji aplikácie je možné ihneď pokračovať ďalšími vývojármi, na začiatku letného semestra sa začal na rozvoji nového systému podieľať aj päťčlenný tím študentov predmetu BI-SP1. Je pravdepodobné, že v ďalšom semestri sa na rozvoji systému bude podieľať tím študentov predmetu BI-SP2 a v budúcnosti ďalšie tímy študentov predmetov BI-SP1 a BI-SP2, prípadne NI-NUR.

9.1 Ďalší rozvoj systému

9.1.1 Nedokončené funkcionality potrebné pre využitie na FIT ČVUT

Pred samotným uvedením nedokončených funkcionalít nového systému považujem za potrebné pripomenúť, že spolu s kolegami z tímu sme si systém na začiatku vývoja rozdelili na tri časti. Ja som sa v rámci svojej práce venoval primárne tímovému riešeniu projektov. Považujem za dôležité podotknúť, že väčšina nesplnených požiadaviek a nedokončených funkcionalít sa priamo netýka tímov a projektov, ide skôr o funkcionality, ktoré nadväzujú a spájajú moju prácu s prácou kolegov, teda kombinujú viaceré doménové oblasti a časti, na ktoré sme systém pred začiatkom rozdelili alebo sa netýkajú ani jednej zo spomínaných troch častí.

Takisto považujem za nutné podotknúť, že síce nižšie uvedené funkcionality neboli dokončené a implementované, boli analyzované a pripravené pre jednoduchú implementáciu ďalšími vývojármi. Neznáme funkčné požiadavky bolo totižto potrebné získať od vyučujúcich, známe požiadavky bolo potrebné validovať a bližšie špecifikovať. Požiadavky sme s tímom bližšie analyzovali a spolu s vyučujúcimi v rámci rozhovorov uvažovali nad riešením. Niektoré z funkcionalít sme aj podrobnejšie navrhovali. Niektoré funkcionality sú už dokonca v procese implementácie ďalšími ľudmi. Táto časť práce môže okrem iného poslúžiť aj ako zoznam úloh pre tímy, ktoré budú v budúcnosti na rozvoji nového systému pracovať.

9.1.1.1 Prihlásenie ČVUT účtom

Nový systém zatiaľ podporuje len prihlásenie pomocou užívateľského mena a hesla ukladaných priamo v systéme. Pre využitie v predmetoch na FIT ČVUT je ale potrebné implementovať prihlásenie pomocou ČVUT účtu. Na implementácii prihlásenia pomocou ČVUT účtu už začal pracovať SP tím, ako uvádzam vyššie v časti 8.3.6.

Dôležitosť: Vysoká

Náročnosť: Vysoká

9.1.1.2 Import dát

Nový systém v dobe odovzdania mojej práce nepodporuje import dát o predmetoch a užívateľoch z externých dátových zdrojov, napríklad zo systému *KOS* pre potreby využitia na FIT ČVUT. Na realizácii importu dát zo systému *KOS* už ale pracuje kolega Jan Jamnický vo svojej bakalárskej práci, v rámci ktorej realizuje podporu predmetov, semestrov a import užívateľov.

Dôležitosť: Vysoká

Náročnosť: Vysoká

9.1.1.3 Zoskupovanie projektov

Nový systém zatiaľ nepodporuje zoskupovanie projektov vo vybranom predmete a semestri. Podporuje zatiaľ len zobrazenie všetkých projektov globálne. Z tohto dôvodu tiež nie je implementované pravidlo, že študent môže byť členom tímu len jedného projektu v danom predmete a semestri. Okrem toho bude tiež potrebné implementovať požiadavku vyučujúcich neumožniť študentom podať viac ako jednu žiadosť o pripojenie sa k projektu v danom predmete a semestri. Pre implementáciu zoskupovania projektov je ale potrebná podpora predmetov a semestrov. Implementácia teda veľmi úzko súvisí a nadväzuje na prácu Jana Jamnického.

Dôležitosť: Vysoká

Náročnosť: Stredná

9.1.1.4 Študentské testovanie

V budúcnosti budú vedieť študenti testovať iný projekt, na ktorom nepracujú, za čo budú daní študenti obmenení bodmi. Túto požiadavku sme neimplementovali, pretože sa jedná o veľmi špecifickú potrebu predmetu NI-NUR a úzko súvisí aj s mojou časťou práce ako aj s prácou kolegu Jana Mrázka, ktorý sa v rámci svojej práce zaoberá hodnotením. Pri realizácii tejto požiadavky v rámci našich bakalárskych prác by teda práca jedného silno závisela na práci druhého.

Dôležitosť: Stredná (pre potreby predmetu NI-NUR)

Náročnosť: Stredná

9.1.1.5 Vlastné role členov tímu a kapacita tímu

Jedná sa o *Could Have* funkčnú požiadavku FP8, ktorá je tiež bližšie vysvetlená v časti 4.2. Túto požiadavku som nakoniec neimplementoval z dôvodu nižšej priority a potreby len niektorých predmetov. Systém je ale pripravený pre okamžitú a pomerne jednoduchú implementáciu tejto funkcionality ďalšími vývojarmi.

Dôležitosť: Stredná (hlavne pre potreby predmetov BI-SP1 a BI-SP2)

Náročnosť: Nízka

9.1.1.6 Vedúci tímu

Jedná sa o ďalšiu užívateľskú rolu. Vedúci tímu má v tíme a v projekte, ku ktorému je daný tím priradený, vyššie právomoci ako bežný člen tímu. Ako som ale zistil v rámci rozhovorov s vyučujúcimi, jedným z nedostatkov súčasného systému je, že je často vedúci tímu zbytočne vyžadovaný. Toto má za následok viacero ďalších komplikácií. Prvý člen tímu sa napríklad automaticky bez jeho vedomia a súhlasu stane vedúcim tímu. Komplikuje sa tak tiež riešenie scenárov, kedy a za akých pravidiel môže vedúci tímu tím opustiť alebo kto sa stane vedúcim tímu v prípade, že pôvodný vedúci tímu tím opustí. Potrebu tejto role som teda bližšie analyzoval a validoval v rámci zberu a analýzy požiadaviek. Zistil som, že v predmetoch BI-SP1 a BI-SP2 je rola vedúceho tímu primárne potrebná až pri odovzdaní práce. Je totižto požadované, aby bol za odovzdanie nejaký študent zodpovedný. Je ale potrebné dôkladnejšie overiť, či garanti predmetov, ktoré súčasný systém používajú, túto rolu vyžadujú aj pre iné činnosti, napríklad pre správu tímu.

Ďalej je potrebné navrhnúť voľbu vedúceho tímu. Pri návrhu je dôležité myslieť na to, že účelom systému SOS je zjednodušenie správy projektov a spolupráce učiteľa s tímom pracujúcim na projekte. Voľba vedúceho tímu by teda mala byť jednoduchá a priamočiara. Automatické nastavenie vedúceho tímu, ako to je v súčasnom systéme, preto teda nie je potrebné a ani chcené, pretože je to pre užívateľov mätúce. Myslím si, že je dôležité sa zamyslieť nad tým, ako voľba vedúceho tímu v praxi prebieha. Tímy majú na začiatku semestra zvyčajne stretnutie aj so zodpovedným vyučujúcim, kde sa na role vedúceho tímu dohodnú. Myslím si teda, že v prípade učiteľom spravovaných projektov, by zodpovednosť voľby vedúceho mohla byť na učiteľovi. Existoval by teda stav projektu, kedy je projekt automaticky schválený, keďže bol vytvorený a je spravovaný učiteľom, ale ešte nie je zvolený vedúci, takže študenti nemôžu odovzdávať svoju prácu. V prípade študentmi vytvorených projektov by sa voľba vedúceho tímu tiež presunula na vyučujúceho, kedy by vyučujúci v rámci schválenia projektu vybral jedného z členov tímu. Ľubovoľný členovia

tímu by ale mohli nejakou formou vyjadriť záujem, čo by vyučujúci pri schvaľovaní projektu videl. Je potrebné tieto návrhy ale zvalidovať s vyučujúcimi predmetov. Keďže sa ale jedná o menej prioritnú funkcionalitu, tak som sa tomuto z časových dôvodov a kvôli prioritnejším požiadavkám viac nevenoval a rolu vedúceho tímu neimplementoval.

Dôležitosť: Nízka

Náročnosť: Nízka

9.1.2 Dôvody nedokončenia niektorých funkcionalít

Okrem vyššie spomenutých dôvodov nerealizovania jednotlivých funkcionalít ako napríklad nízka priorita danej funkcionality, existujú podľa môjho názoru všeobecné dôvody, prečo systém v dobe odovzdania tejto práce nie je možné produkčne využiť na FIT ČVUT a prečo sme s tímom boli nútení venovať sa len prioritným funkcionalitám a ďalšie funkcionality len analyzovať a realizáciu prenechať ďalším študentom.

9.1.2.1 Rozsah práce a komplexita systému

V prvom rade je nutné podotknúť, že na vývoji súčasného systému pracovalo viac ako desať študentov FIT ČVUT po dobu troch rokov. Vývoji súčasného systému sa venoval kolega Ing. Tomáš Pavlůsek vo svojej bakalárskej aj diplomovej práci, kolega Bc. Max Hejda vo svojej bakalárskej práci, sedemčlenný tím študentov predmetu BI-SP1, v ktorom som bol aj ja a tím študentov NI-NUR, ktorý sa venoval návrhu UI.

Mimo rozsah a komplexitu systému SOS je potrebné spomenúť aj rozsah analytických a návrhových prác, ktoré som spolu s kolegami z tímu vykonal. Okrem zberu a analýzy nových požiadaviek bolo potrebné podrobnejšie analyzovať, validovať a prioritizovať aj tie existujúce. Pred samotným vývojom novej aplikácie sme sa s tímom skutočne venovali dôkladnej analýze existujúcich požiadaviek ako aj analýze súčasného systému a jeho funkcionalitám, UI a nedostatkom, ako uvádzam v časti 3. Myslím si ale, že práve dôkladná analýza existujúcich požiadaviek a súčasného systému ako aj skúsenosti, ktoré sme nadobudli vďaka vývoju súčasnej aplikácie, nám priniesli viacero výhod vrátane univerzálnejšie navrhnutého systému a prehľadnejšieho UI.

9.1.2.2 Tímová spolupráca

Ďalším dôvodom je podľa môjho názoru čiastočne aj práca v tíme a zlyhanie tímovej spolupráce. Myslím si, že sme v letnom semestri s kolegami z tímu

nedokázali suplovať rolu vedúceho tímu a vhodne sa dohodnúť na pláne práce, aby naše práce vzájomne nenadväzovali na seba. Ako uvádzam vyššie, systém sme si s kolegami z BP tímu rozdelili na tri časti, no pre implementáciu viacerých funkcionalít bola potrebná dokončená implementácia mojej časti práce ako aj práce kolegov.

Okrem absencie vedúceho tímu je tiež dôležité spomenúť časovú náročnosť, ktorá s vývojom v tíme a zapojením SP tímu do vývoja súvisela. Ako uvádzam vyššie v časti 8.1.3, s kolegami z BP tímu sme si navzájom revidovali všetok kód. Neskôr sme kontrolovali kód aj kolegom z SP tímu, aby sme sa uistili, že všetok kód je naozaj kvalitný a jednoducho pochopiteľný, aby sme tak zaistili čo najjednoduchšiu udržateľnosť nového systému. Síce tieto vzájomne kontroly kódu prinášajú množstvo výhod, vývoj to ale určite spomaľuje, pretože sme sa s kolegami okrem implementácii našich častí práce museli venovať aj kontrole kódu týkajúcich sa ďalších častí systému. Po schválení zmien sme tiež často narážali aj na tzv. *merge konflikty*, ktoré označujú časti kódu, ktoré boli súčasne zmenené vo viacerých vetvách. Dôvodom častých konfliktov bolo určite aj to, že naše časti práce súvisia, aj keď sme sa s kolegami na začiatku vývoja pokúsili systém rozdeliť na tri čo najmenej súvisiace doménové oblasti. Tieto konflikty je potrebné riešiť manuálne, čo vývojárov značne spomaľuje a predlžuje tak vývoj.

Časovo náročná bola aj spolupráca s SP tímom. Okrem vývoja som aj s kolegami z BP tímu často komunikoval s SP tímom, vysvetľoval potrebné informácie o systéme SOS, odpovedal na ich otázky a zadával im úlohy, ako uvádzam v časti 8.3.2.

Na druhú stranu si ale myslím, že vďaka práci na novom systéme v tíme vrátane návrhu architektúry systému, návrhu základného doménového modelu MVP, návrhu UI MVP, návrhu API MVP, vzájomným revíziám kódu a zapojeniu SP tímu do rozvoja nového systému sme pripravili udržateľný nový systém, ktorý je pripravený pre jednoduchý rozvoj ďalšími vývojármi, čo v budúcnosti prinesie množstvo výhod, čo sme už aj vďaka SP tímu skutočne overili.

9.1.2.3 Nové technológie

Výber technológií je často otázkou osobných preferencií a skúsenosti konkrétnych vývojárov. Pri vývoji veľkého systému je ale výber vhodných technológií na základe objektívnej analýzy správnejší, pretože na jeho rozvoji bude pracovať veľké množstvo vývojárov s rôznymi skúsenosťami a preferenciami. Práve z tohto dôvodu sme s kolegami vybrali technológie, s ktorými síce nemáme najviac skúseností, ale ich výber prináša viac objektívnych výhod, ako uvádzam v časti 5.4.

Práve kvôli nie príliš bohatým skúsenostiam so zvolenými technológiami sa vý-

voj tiež predĺžil. Pre všetkých členov tímu sa jednalo o jeden z prvých projektov vo vybraných technológiách. Ja som konkrétne nemal žiadnu skúsenosť s platformou .NET a asi mesačnú skúsenosť s technológiou Vue.js. Je nutné podotknúť, že síce je framework ASP.NET Core podľa môjho subjektívnu názoru čiastočne podobný frameworku Spring, s ktorým mám skúsenosti z predmetu BI-TJV a zo zamestnania alebo iným MVC webovým frameworkom, s ktorými majú skúsenosti ďalší členovia BP tímu, venovali sme stále značné množstvo času pochopeniu celej platforme .NET a frameworku ASP.NET Core. Naučiť sa pracovať s frontend technológiami Vue.js a Nuxt na takú úroveň, aby sme v nich boli schopní vytvoriť kvalitné UI, tiež zabralo všetkým členom tímu niekoľko desiatok hodín. Aj z tohto dôvodu sme vybrali práve framework Vue.js, ktorý je považovaný za jednu z jednoduchších frontend technológií, ako uvádzam v časti 5.4, pretože v opačnom prípade by sa vývoj ešte viac predĺžil.

9.1.3 Ďalšie možné funkcionality

V tejto časti uvediem ďalšie možné funkcionality nového systému a návrhy na zlepšenie, ktoré by mohli byť v budúcnosti implementované, no nie sú vyžadované pre produkčné využitie na FIT ČVUT.

9.1.3.1 Prihlásenie Google účtom

Systém bude v budúcnosti podporovať prihlásenie pomocou Google účtu pre potreby produkčného využitia mimo ČVUT. Na implementácii prihlásenia pomocou Google účtu už začal pracovať SP tím, ako uvádzam vyššie v časti 8.3.6.

Dôležitosť: Stredná

Náročnosť: Vysoká

9.1.3.2 Export hodnotenia

Systém by v budúcnosti mohol podporovať export hodnotenia do externých systémov, napríklad do systému *KOS* a *Klasifikace*.

Dôležitosť: Stredná

Náročnosť: Vysoká

9.1.3.3 Notifikácie v rámci systému

V rámci zberu nových požiadaviek a validácie tých existujúcich sme s tímom zistili, že v súčasnom systéme užívatelia nepoužívajú notifikácie a zvyčajne ich ignorujú. Podľa užívateľov, s ktorými sme viedli rozhovory je dôvodom to, že nefungujú podľa ich predstáv. Notifikácie v súčasnom systéme totižto fungujú tak, že sa konkrétna notifikácia prestane zobrazovať, keď si ju užívateľ zobrazí. Užívateľ nemusí ale udalosť, na ktorú daná udalosť upozorňuje, vôbec vyriešiť. Naopak po vyriešení udalosti, na ktorú daná notifikácia upozorňuje, bez zobrazenia danej notifikácie, ostáva táto notifikácia nezobrazená a aktívna. Užívatelia, s ktorými sme viedli rozhovory, majú teda v aplikácii desiatky až stovky nezobrazených notifikácií.

Dôležitosť: Stredná

Náročnosť: Vysoká

9.1.3.4 E-mailové notifikácie

Systém bude v budúcnosti posielať e-mailové notifikácie napríklad o blížiacich sa termínoch odovzdania. Notifikačné e-maily budú konfigurovateľné v zmysle, že si užívateľ bude vedieť zvoliť, aké e-maily chce prijímať. Ako píšem v časti 8.3.5, podporu e-mailových notifikácii už pridal SP tím.

Dôležitosť: Stredná

Náročnosť: Stredná

9.1.3.5 Kopírovanie projektov

Jedná sa o *Won't Have* požiadavku F9. Požiadavka nebola realizovaná, pretože nadväzuje na moju prácu ako aj prácu oboch kolegov z tímu. Pre jej implementáciu sú totiž potrebné ako predmety a semestre, na čom pracuje kolega Jamnický tak aj kontrolné body, na čom pracuje kolega Mrázek. Ide ale o novú požiadavku, ktorá nie je realizovaná ani v súčasnom systéme. Mojou prácou teda bolo túto požiadavky objaviť, získať od užívateľov podrobnosti, bližšie analyzovať, určiť prioritu a náročnosť a navrhnúť riešenie. Navrhnuté riešenie, ktoré som konzultoval aj s užívateľmi v rámci rozhovorov pri zbere požiadaviek, je uvedené v časti 5.3, konkrétne sa jedná o prípad použitia UC16.

Dôležitosť: Nízka

Náročnosť: Stredná

9.1.3.6 Hromadné akcie

Nový systém by v budúcnosti mohol podporovať hromadné akcie nad nejakou množinou projektov. Užívateľ by si v systéme vyfiltroval určitú množinu projektov (túto funkcionalitu som už implementoval v rámci tejto práce) a pre všetky projekty z tejto množiny vykonal nejakú akciu z ponuky. Vyučujúci by si napríklad mohol vyfiltrovať všetky projekty s prázdnym tímom a všetky tieto projekty hromadne deaktivovať a skryť tak pre študentov (deaktivovanie jedného vybraného projektu som už tiež implementoval v rámci tejto práce). Toto je ale len jeden z prípadov použitia, ktorý by vyučujúci v systém uvítali, čo som zistil pri zbere požiadaviek. Inak sa ale jedná o všeobecnú funkcionalitu aplikovateľnú pre viacero prípadov použitia.

Dôležitosť: Nízka

Náročnosť: Stredná

9.1.3.7 Vytvorenie tímu bez nutnosti priradeného projektu

Ďalšiu funkcionalitu, ktorú by užívatelia radi uvítali, je možnosť vytvorenia tímu bez nutnosti priradeného projektu. Išlo by o skupinu študentov, ktorá by mohla v určitých situáciach vystupovať ako celok, napríklad spoločne žiadať o pripojenie sa k projektu. Systém je na to pripravený, na backende existuje doménová trieda `Team` ako aj API endpointy pre správu tímov.

Dôležitosť: Nízka

Náročnosť: Nízka

9.1.3.8 Poznámka k žiadostiam týkajúcich sa členstva v tíme

Viacerí vyučujúci by v systéme uvítali, keby študenti vedeli poslať textovú poznámku k žiadosti o členstvo v projektovom tíme, prípadne keby mohol vyučujúci zaslať študentovi otázku okrem prijatia alebo odmietnutia žiadosti. Systém ja na túto požidavku na backende pripravený, doménova trieda `TeamMembershipRequest` obsahuje atribúty pre poznámku od študenta aj učiteľa. Ďalej je nutné navrhnúť UI a implementovať frontend. Na základe výsledkov užívateľského testovania bolo aj čiastočne navrhnuté UI tejto funkcionality. Návrh spočíval v tom, že by okrem tlačítok pre prijatie a odmietnutie žiadosti existovalo tretie tlačítko, ktoré by po stlačení umožnilo vyučujúcemu zadať otázku, v rámci ktorej by sa mohol vyučujúci opýtať študenta napríklad na jeho skúsenosti. Počas čakania na odpoveď by ale vyučujúci mohol stále žiadosť prijať alebo odmietnuť. Študent by tiež mohol v rámci podania žiadosti

poslať poznámku, v ktorej by sa mohol napríklad stručne predstaviť. Aby užívatelia vedeli na čo to slúži, bolo by potrebné im túto funkcionalitu vysvetliť. V budúcnosti by sa tiež dalo nastaviť v konfigurácii projektu, či sú poznámky povinné alebo nie.

Dôležitosť: Nízka

Náročnosť: Nízka

9.1.3.9 Možnosť zvýšiť kapacitu tímu pri pridávaní nových členov

V súčasnom systéme je možné zvýšiť kapacitu tímu len v nastaveniach projektu. Nový systém by ale po pridaní kapacity tímu mohol tiež podporovať zvýšiť kapacitu tímu v rámci procesu prijímaní žiadosti o členstvo v tíme. Analýze scenárov súčasného systému týkajúcich sa prijímania študentov do tímu a podávania žiadostí o vstup do tímu sa venoval SP tím, ako uvádzam v časti 8.3. Keďže sa ale jedná o požiadavku jedného konkrétneho vyučujúceho predmetov BI-SP1, BI-SP2 a NI-NUR, je nutné podrobnejšie analyzovať potrebu tejto funkcionality, na základe čoho aj lepšie navrhnúť realizáciu.

Dôležitosť: Nízka

Náročnosť: Nízka

9.1.3.10 Možnosť uviesť preferované termíny stretnutí

Vyučujúci by v budúcnosti mohli mať možnosť pri tvorbe projektu uviesť preferované termíny konzultácií. Študenti, ktorí sa na projekt chcú prihlásiť, by túto informáciu videli a vedeli by si projekt vyberať aj podľa týchto kritérií. Jedná sa o požiadavku viacerých vyučujúcich viacerých predmetov, požiadavka je preto určite potrebná a žiadaná, nie je ale veľmi prioritná. Je potrebné navrhnúť, ako túto požiadavku realizovať. Z dôvodu nízkej priority sa nemusí jednať o zložitú implementáciu. Požiadavku by pravdepodobne splnilo zadanie nejakého rozsahu alebo textovej poznámky. Možností je samozrejme viacero a funkcionalita sa dá implementovať aj zložito v prípade, že by v budúcnosti bolo zistené, že sa jedná o prioritnejšiu požiadavku.

Dôležitosť: Nízka

Náročnosť: Nízka

9.1.4 Ďalšie potrebné činnosti

Okrem implementácie vyššie uvedených funkcionalít bude v budúcnosti potrebné pre skvalitnenie novej aplikácie vykonať nasledujúce činnosti.

9.1.4.1 Integračné testy

Ako uvádzam v časti 8.3, podporu pre integračné testy pridal SP tím v rámci MR !80. Ďalej bude potrebné napísať integračné testy kľúčových častí backend aplikácie, primárne `Service` tried, ktoré pracujú s biznis logikou v entitných triedach a s databázou a pridávajú ďalšiu biznis logiku.

9.1.4.2 Druhá iterácia užívateľského testovania

Po implementácii vyššie spomenutých funkcionalít bude potrebné znovu otestovať a zvalidovať užívateľské rozhrania aplikácie na vybraných užívateľoch. V rámci ďalších iterácií navrhujem, aby bolo užívateľské testovanie primárne zamerané na nové funkcionality, ale čiastočne validovalo aj už existujúce funkcionality.

9.1.4.3 Akceptačné testovanie pred produkčným využitím

Pred produkčným využitím na FIT ČVUT bude potrebné realizovať akceptačné testovanie garantmi predmetov, v ktorých sa nový systém začne používať.

9.1.4.4 Získanie kvantitatívnej spätnej väzby

Po prvom semestri produkčného využitia v predmetoch na FIT ČVUT navrhujem získať kvantitatívnu spätnú väzbu od študentov na novú aplikáciu a nové UI formou dotazníka.

9.1.4.5 Iteratívné vylepšovanie UI

Na základe ďalších iterácii užívateľského testovania a odpovedí z dotazníka navrhujem ďalej vylepšovať UI nového systému, aby systém čo najviac odpovedal požiadavkám užívateľov.

9.1.4.6 Využitie v ďalších predmetoch na FIT ČVUT a mimo ČVUT

Systém SOS by mohol byť v budúcnosti využitý pre podporu výuky a semestrálnych prác aj v ďalších predmetoch na FIT ČVUT, ako uvádzam vyššie v časti 4.1. Okrem toho by mohol byť aj nový systém využitý na správu ročníkových prác na GJGJ ako súčasný systém ale aj na ďalších gymnáziách, na správu semestrálnych prác na ďalších fakultách ČVUT alebo aj v iných inštitúciách mimo ČVUT. Snažili sme sa totiž o návrh a implementáciu o čo najuniverzálnejšieho backendu systému, nevenovali sme sa ale konkrétnym požiadavkám GJGJ alebo iných inštitúcií mimo FIT ČVUT, keďže to nebolo cieľom tejto práce.

9.2 Zhodnotenie spôsobu vývoja

Ako plynie zo zadania mojej práce a ako som viackrát v práci spomínal, jedným z cieľov mojej práce bolo prakticky vyskúšať iteratívnu formu vývoja softwaru s nejakými agilnými postupmi a prácu v tíme s kolegami Janom Jamnickým a Janom Mrázkom. Bližšie sa analýze metódam vývoja softwaru venujem v časti 1.1, na základe čoho následne v časti 5.1 vysvetľujem spôsob vývoja, ktorý som spolu s kolegami z BP tímu pre vývoj nového systému zvolil.

Zvolený spôsob vývoja nám priniesol viacero výhod ako aj nevýhod. V tejto časti práce zhrniem, aké výhody a nevýhody nám podľa môjho názoru zvolený spôsob vývoj priniesol a čo by som nabudúce zmenil na základe mojich skúseností s vývojom nového systému v rámci tejto bakalárskej práce.

Za hlavnú výhodu zvoleného spôsobu vývoja považujem to, že sme sa skutočne s tímom pravidelne stretávali. Vďaka tomu sme spoločne riešili problémy, na ktoré narazil niekto z tímu. Okrem iného sme spoločne navrhovali architektúru nového systému, vyberali technológie a navrhovali a definovali MVP vrátane návrhu základného doménového modelu, návrhu API a UI práve pre najprioritnejšie funkcionality, ktoré boli v rámci MVP implementované. Výhodou tímového návrhu bola určite možnosť argumentácie všetkých členov tímu, čo určite zvýšilo kvalitu nového systému.

Ďalšou výhodou bolo, že sme skutočne iteratívne validovali našu prácu a plán práce na ďalšie iterácie s vedúcim práce, takže sme mohli náš plán prispôsobovať potrebám a prioritám, ktoré určoval vedúci práce.

V neposlednom rade považujem za veľkú výhodu iteratívneho vývoja to, že už na konci zimného semestra 2023/2024 bola nasadená prvá funkčná verzia nového systému, ktorá podporovala tie najprioritnejšie funkcionality, ktoré po-

pisujem v časti 5.5. Vďaka tomu sme mohli už prvú verziu systému predstaviť vedúcemu práce a dostať spätnú väzbu na užívateľské rozhranie a podporované funkcionality.

Na druhú stranu nás zvolený spôsob vývoja aj čiastočne zdržoval a spomaľoval, ako už naznačujem v časti 9.1.2. V prvom rade bolo na začiatku vývoja potrebné analyzovať možné spôsoby vývoja, analyzovať výhody, ktoré by nám iteratívny vývoj mohol priniesť ako aj výhody jednotlivých agilných postupov. Taktiež bolo potrebné, aby všetci členovia tímu zmenili svoje myslenie a rýchlo sa prispôsobili iteratívnemu vývoju. Konkrétne ja som mal na začiatku zimného semestra 2023/2024 potrebu navrhnúť celý nový systém so všetkými možnými funkcionalitami. Myslel som si, že to, že myslím do budúcna a snažím sa všetky pravdepodobné rozšírenia zohľadniť v návrhu bude prospešné a že presne takto by mal správny softwarový inžinier postupovať. Na jednom zo stretnutí, ktoré sme s tímom mali, mi ale vedúci práce pripomenul, že sa snažíme vyvíjať iteratívne. Vysvetlil mi, prečo to vlastne robíme, aké to prináša benefity a prečo detailný a kompletný návrh pred začiatkom implementácie nedáva zmysel a že to pravdepodobne bude strata času, pretože sa časom veľa požiadaviek zmení. V praxi vidím, že agilný vývoj funguje a že častý refaktoring kódu, ktorý je pri agilnom vývoji nutný, je naozaj bežný a prospešný. Keďže som ale pred tým nikdy nový systém agilne nenavrhoval, mal som práve s týmto problém.

Vyššie spomenutý refaktoring kódu ale v našom prípade stále považujem za nevýhodu zvoleného spôsobu vývoja. Je nutné podotknúť, že sme kód refaktorovali skutočne často a že nás podstatne zdržoval pri vývoji a implementácii ďalších potrebných funkcionalít.

Kvôli zvolenému spôsobu vývoja a vývoja nového systému v tíme bolo potrebné zaistiť správne fungovanie tímu počas celého vývoja. Za správne fungovanie tímu je zodpovedný vedúci tímu alebo *Product Owner* a *Scrum Master* v kontexte agilnej metodiky Scrum. v zimnom semestri s nami spolupracoval kolega Timotej Adamec, ktorý práve rolu vedúceho tímu zastával a práve vďaka nemu fungovala spolupráca celého tímu skvele. V letnom semestri s nami už ale kolega Adamec nespolupracoval. Rolu vedúceho tímu teda nezastával nikto z tímu a spolupráca už nebola tak kvalitná. Práve kvôli absencii vedúceho tímu v letnom semestri 2023/2024 sa nám nepodarilo dokončiť niektoré funkcionality, ktoré nadväzujú na viaceré doménové oblasti systému a teda na moju prácu aj prácu kolegov.

Myslím si ale, že nevýhody zvoleného vývoja sú skôr spôsobené našimi chybami, ktoré sa vďaka skúsenostiam nadobudnutým v rámci tejto práce, budú dať v budúcnosti jednoducho odstrániť.

V prvom rade bude potrebné zaistiť, aby mal každý tím pracujúci na rozvoji nového systému vedúceho tímu, ktorý sa nebude venovať implementačným činnostiam, ale bude primárne riadiť celý tím, rozhodovať o prioritách úloh,

bude zodpovedný za nadväznosť úloh všetkých členov tímu, bude spoločne s celým tímom pracovať na analytických činnostiach a bude s ostatnými členmi tímu konzultovať ich návrhy a postup.

V rámci predmetov BI-SP1 a BI-SP2 by bolo teda vhodné mať na toto vyčleneného jedného študenta. V rámci práce na bakalárskych alebo diplomových prácach v tíme sa to bude uskutočňovať ťažšie, ako to bolo aj v našom prípade. Podľa môjho názoru by ale bolo ideálne, keby bolo zameranie záverečnej práce jedného člena tímu práve na riadenie tímu a dodanie systému ako celku.

V časti 5.1 popisujem, že sme sa s kolegami z BP tímu rozhodli neodhadovať náročnosť úloh, pretože sme sa na začiatku vývoja domnievali, že naše časti prác spolu nesúvisia a že implementačné úlohy budú väčšinou individuálne. To nakoniec ale nebola pravda, ako uvádzam vyššie v časti 9.1.2. Okrem vedúceho tímu by práve spoločné odhadovanie úloh pomohlo rýchlejšiemu vývoju a rýchlejšiemu dodaniu väčšieho množstva funkcionalít.

Rozhodne teda navrhujem pokračovať v iteratívnom vývoji pri ďalšom rozvoji nového systému či už v BI-SP1, BI-SP2 alebo v rámci záverečných prác, pretože vyššie spomenuté výhody určite prevyšujú nevýhody. Bude ale potrebné myslieť na naše chyby a snažiť sa im vyvarovať.

Záver

Cieľom tejto práce bolo analyzovať súčasný systém SOS a jeho funkcionality týkajúce sa tímov a tímových projektov. Na základe analýzy súčasného systému, jeho nedostatkov, získaných skúseností z vývoja súčasného systému a aktuálnych funkcionalít súčasného systému, bolo cieľom navrhnúť a realizovať nový systém, ktorý v budúcnosti ten súčasný nahradí a bude tak reálne využitý na FIT ČVUT. Ďalším cieľom bolo zaistiť jednoduchú udržateľnosť a rozšíriteľnosť nového systému, aby bolo jednoduché nový systém aj naďalej rozvíjať ďalšími ľuďmi.

Ďalším cieľom bolo vyskúšať iteratívnu formu vývoja softwaru a prácu v tíme s kolegami Janom Jamnickým a Janom Mrázkom. Doménovým zameraním tejto práce boli tímy a tímové projekty, pričom ostatní členovia tímu riešili ďalšie doménové oblasti systému. Kolega Jan Jamnický sa vo svojej práci venuje užívateľom, predmetom, semestrom a paralelkám a kolega Jan Mrázek kontrolným bodom, odovzdávaniu a hodnoteniu. Síce som pracoval primárne na návrhu, implementácii a testovaní funkcionalít týkajúcich sa práve tímov a tímových projektov, počas celého vývoja bola tímová spolupráca skutočne potrebná. Spoločne s kolegami z tímu sme totižto navrhovali architektúru systému, vyberali vhodné technológie, navrhovali a definovali MVP vrátane výberu najprioritnejších funkcionalít, ktoré MVP podporovalo, a návrhu základného doménového modelu, návrhu API a UI práve pre tieto najprioritnejšie funkcionality. Ďalej som spoločne s tímom spolupracoval aj s tímom študentov predmetu BI-SP1, ktorý sa do rozvoja nového systému zapojil v letnom semestri 2023/2024. V neposlednom rade sme si spolu s kolegami z tímu navzájom kontrovali kód a pomáhali si navzájom riešiť rôzne problémy, na ktoré sme počas implementácie narazili. Ako ale píšem v kapitole 9, spolupráca v našom tíme čiastočne zlyhala, pretože nám chýbal vedúci tímu.

Keďže jedným z cieľov tejto práce bolo vyskúšať iteratívny vývoj softwaru a prácu v tíme, tak sa v kapitole 1 zaoberám teoretickou analýzou tímového vývoja softwaru. Definujem vodopádový model vývoja, iteratívny vývoj a agilné

metodiky – XP a Scrum. Ďalej sa venujem predstaveniu troch hlavných verzovacích modelov – Git Flow, GitHub Flow a GitLab Flow – ktoré sú pre vývoj softwaru v tíme veľmi potrebné.

V kapitole 2 sa venujem architektúram webových aplikácií. Predstavil som HTTP protokol a architektonický štýl REST. Následne som definoval viacvrstvovú architektúru, architektonické vzory MVC a MVT a čistú architektúru. Na záver som porovnal jedno-stránkové a viac-stránkové webové aplikácie.

Následne v kapitole 3 analyzujem súčasný systém SOS vrátane použitých technológií a architektúry, na ktorej je SOS postavený. Ďalej analyzujem funkcionality súčasného systém metódou prípadov použitia. Uvádzam aj hlavné a alternatívne scenáre a snímky obrazovky. Následne rozoberá moje zapojenie do rozvoja súčasného systému v predmete BI-SP1. Na záver analyzujem stav systému a uvádzam jeho hlavné nedostatky, ktoré som spolu s kolegami z tímu v predmete BI-SP1 odhalil.

V kapitole 4 analyzujem potreby a priebeh predmetov na FIT ČVUT, ktoré systém SOS využívajú a niekoľkých ďalších podobných predmetov, ktoré by v budúcnosti tiež mohli nový systém SOS používať. Následne analyzujem a validujem existujúce požiadavky a získavam, analyzujem, kategorizujem a prioritizujem nové požiadavky.

Po získaní a analýze požiadaviek sa v kapitole 5 venujem návrhu nového systému. Najprv predstavím zvolený spôsob vývoja a verzovací model. Následne sa venujem návrhu architektúry a voľbe technológií. Nová aplikácia sa bude skladať z oddeleného frontend a backendu, ktorý frontendu poskytuje REST API. Pre vývoj backend bol použitý framework ASP.NET Core, pre vývoj frontendu framework Nuxt postavený na Vue.js. Následne predstavujem MVP a návrh doménového modelu, API a UI vo forme Lo-Fi prototypov.

V kapitole 6 sa venujem implementácii kľúčových častí backendu aj frontendu. Popisujem štruktúru oboch aplikácií, ako funguje autorizácia a autentifikácia, komunikácia backendu a frontendu a internacionalizácia a lokalizácia na frontende.

Následne sa v kapitole 7 zaoberám testovaním novej aplikácie. Rozhodol som sa použiť jednotkové testy pre testovanie kľúčovej biznis logiky, integračné testy pre overenie funkčnosti viacerých komponentov vrátane entít, *Service* tried a prístupu k databáze. Okrem automatického testovania som testoval a validoval aj užívateľské rozhranie užívateľským testovaním na vybraných piatich užívateľoch.

V kapitole 8 popisujem, ako sme spolu s kolegami z tímu pripravili projekt pre zapojenie ďalších študentov a ako sa na rozvoji nového systému podieľal tím študentov predmetu BI-SP1.

V poslednej kapitole 9 som zhodnotil výsledky práce a stav nového systému. Popisujem, čo je potrebné implementovať pre produkčné využitie na FIT ČVUT a pridávam aj ďalšie možnosti rozvoja, na ktorých by mohli pracovať ďalšie študentské tímy v predmetoch BI-SP1 a BI-SP2.

Nová aplikácia je funkčná, otestovaná a je nasadená na adrese `https://new.sos2.ksi.fit.cvut.cz/`. Nové užívateľské rozhranie bolo tiež otestované na piatich užívateľoch. Aplikácia bude po dokončení implementácie niektorých ďalších funkcionalít uvedených v kapitole 9 pripravená na produkčné využitie na FIT ČVUT.

Bibliografia

1. SOMMERVILLE, Ian. *Software Engineering*. Pearson Education Limited, 2015. ISBN 978-1-292-09613-1.
2. KADLEC, Václav. *Agilní programování : metodiky efektivního vývoje softwaru*. Brno : Computer Press, 2004. ISBN 80-251-0342-0.
3. *Manifesto for Agile Software Development*. 2001. Dostupné tiež z: `https://agilemanifesto.org/`. (cit. 2024-02-05).
4. MARTIN, Robert C. *Agile software development, principles, patterns, and practices*. Pearson Education Limited, 2014. ISBN 978-1-292-02594-0.
5. REHKOPF, Max. *User stories with examples and a template*. [B.r.]. Dostupné tiež z: `https://www.atlassian.com/agile/project-management/user-stories`. (cit. 2024-02-06).
6. RADIGAN, Dan. *User stories with examples and a template*. [B.r.]. Dostupné tiež z: `https://www.atlassian.com/agile/project-management/estimation`. (cit. 2024-02-06).
7. FOWLER, Martin. *Refactoring*. [B.r.]. Dostupné tiež z: `https://refactoring.com/`. (cit. 2024-02-21).
8. REHKOPF, Max. *Scrum sprints*. [B.r.]. Dostupné tiež z: `https://www.atlassian.com/agile/scrum/sprints`. (cit. 2024-02-27).
9. SCHWABER, Ken; SUTHERLAND, Jeff. *The Scrum Guide*. 2020. Dostupné tiež z: `https://scrumguides.org/scrum-guide.html`. (cit. 2024-02-16).
10. WEST, Dave. *Agile scrum roles and responsibilities*. [B.r.]. Dostupné tiež z: `https://www.atlassian.com/agile/scrum/roles`. (cit. 2024-02-27).
11. RADIGAN, Dan. *Product Backlog - What is it & How to create one*. [B.r.]. Dostupné tiež z: `https://www.atlassian.com/agile/scrum/backlogs`. (cit. 2024-02-27).

12. RADIGAN, Dan. *What is a stand up meeting & tips to run one*. [B.r.]. Dostupné tiež z: https://www.atlassian.com/agile/scrum/standups. (cit. 2024-02-27).
13. ATLASSIAN. *Gitflow workflow*. [B.r.]. Dostupné tiež z: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow. (cit. 2024-02-25).
14. GITKRAKEN. *What is the best Git branch strategy?* [B.r.]. Dostupné tiež z: https://www.gitkraken.com/learn/git/best-practices/git-branch-strategy. (cit. 2024-02-25).
15. GITHUB. *GitHub Flow*. [B.r.]. Dostupné tiež z: https://githubflow.github.io. (cit. 2024-02-25).
16. GITLAB. *Introduction to GitLab Flow*. [B.r.]. Dostupné tiež z: https://docs.gitlab.co.jp/ee/topics/gitlab_flow.html. (cit. 2024-02-26).
17. GITLAB. *What is GitLab Flow?* [B.r.]. Dostupné tiež z: https://about.gitlab.com/topics/version-control/what-is-gitlab-flow. (cit. 2024-02-26).
18. FOWLER, Martin. *Software Architecture Guide*. 2019. Dostupné tiež z: https://martinfowler.com/architecture/. (cit. 2024-02-24).
19. *An overview of HTTP*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview. (cit. 21.02.2024).
20. *HTTP request methods*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods. (cit. 21.02.2024).
21. *Safe (HTTP Methods)*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Glossary/Safe/HTTP. (cit. 21.02.2024).
22. *Idempotent*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Glossary/Idempotent. (cit. 21.02.2024).
23. *Cacheable*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Glossary/Cacheable. (cit. 21.02.2024).
24. *HTTP response status codes*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. (cit. 21.02.2024).
25. *HTTP headers*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers. (cit. 21.02.2024).
26. GUPTA, Lokesh. *What is REST?: REST API Tutorial*. 2023. Dostupné tiež z: https://restfulapi.net/. (cit. 13.03.2024).

27. GUPTA, Lokesh. *REST Architectural Constraints: REST API Tutorial*. 2023. Dostupné tiež z: https://restfulapi.net/rest-architectural-constraints/. (cit. 13.03.2024).

28. GUPTA, Lokesh. *REST Architectural Constraints: REST API Tutorial*. 2023. Dostupné tiež z: https://restfulapi.net/statelessness/. (cit. 13.03.2024).

29. RICHARDS, Mark. *Software Architecture Patterns*. O'Reilly Media, Inc., 2022. ISBN 978-1-098-13427-3.

30. IBM. *What is three-tier architecture?* [B.r.]. Dostupné tiež z: https://www.ibm.com/topics/three-tier-architecture. (cit. 2024-02-20).

31. FOWLER, Martin. *Presentation Domain Data Layering*. [B.r.]. Dostupné tiež z: https://www.martinfowler.com/bliki/PresentationDomainDataLayering.html. (cit. 2024-02-20).

32. *MVC*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Glossary/MVC. (cit. 2024-02-19).

33. *NI-ADP: MVC*. Django Software Foundation, [b.r.]. Dostupné tiež z: https://courses.fit.cvut.cz/NI-ADP/materials/architectures/MVC.html. (cit. 13.03.2024).

34. *Observer*. [B.r.]. Dostupné tiež z: https://refactoring.guru/design-patterns/observer. (cit. 13.03.2024).

35. *Django documentation*. Django Software Foundation, [b.r.]. Dostupné tiež z: https://docs.djangoproject.com/en/5.0/. (cit. 13.03.2024).

36. SMITH, Steve. *Common web application architectures*. [B.r.]. Dostupné tiež z: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures. (cit. 13.03.2024).

37. DAVIDSON, Tim. *Single Page Application (SPA) vs Multi Page Application (MPA): Which Is The Best?* 2023. Dostupné tiež z: https://cleancommit.io/blog/spa-vs-mpa-which-is-the-king/. (cit. 13.03.2024).

38. *SPA (Single-page application)*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Glossary/SPA. (cit. 13.03.2024).

39. *What is Python? Executive Summary | Python.org*. [B.r.]. Dostupné tiež z: https://www.python.org/doc/essays/blurb/. (cit. 10.03.2024).

40. PAVLŮSEK, Tomáš. *SOS - Studentský odevzdávací systém*. 2021. Dostupné tiež z: https://dspace.cvut.cz/handle/10467/95107. (cit. 10.03.2024).

41. *PostgreSQL: About*. [B.r.]. Dostupné tiež z: https://www.postgresql.org/about/. (cit. 13.03.2024).

42. *What Is PostgreSQL?* [B.r.]. Dostupné tiež z: https://www.postgresql.org/docs/current/intro-whatis.html. (cit. 13.03.2024).

43. *Bootstrap – The most popular HTML, CSS, and JS library in the world.* [B.r.]. Dostupné tiež z: https://getbootstrap.com/. (cit. 14.03.2024).

44. HEJDA, Max. *Systém pro správu ročníkových prací*. 2022. Dostupné tiež z: https://dspace.cvut.cz/handle/10467/102038. (cit. 13.03.2024).

45. PAVLŮSEK, Tomáš. *SOS II- Studentský odevzdávací systém*. 2023. Dostupné tiež z: https://dspace.cvut.cz/handle/10467/108866. (cit. 11.03.2024).

46. *Introduction | VuePress*. [B.r.]. Dostupné tiež z: https://vuepress.vuejs.org/guide/%5C#how-it-works. (cit. 6. 3. 2024).

47. *Studijní a zkušební řád pro studenty ČVUT*. 2018. Dostupné tiež z: https://www.cvut.cz/sites/default/files/content/74c76d2e-7f4d-4cb1-ac28-b0765c7f88f2/cs/20200624-studijni-a-zkusebni-rad-pro-studenty-cvut-v-praze-ii-uplne-zneni-ucinnost-od-29-11-2018.pdf. (cit. 14.03.2024).

48. *NI-NUR Hodnocení a semestrální práce*. 2023. Dostupné tiež z: https://courses.fit.cvut.cz/MI-NUR/classification/index.html. (cit. 19.03.2024).

49. *B222-BI-SP1.21,BIK-SP1.21 - Softwarový týmový projekt 1*. [B.r.]. Dostupné tiež z: https://moodle-vyuka.cvut.cz/course/view.php?id=8719. (cit. 14.03.2024).

50. *B231-BI-SP2.21,BIK-SP2.11 - Softwarový týmový projekt 2*. [B.r.]. Dostupné tiež z: https://moodle-vyuka.cvut.cz/course/view.php?id=9560. (cit. 14.03.2024).

51. *B222-BI-SWI.21 - Softwarové inženýrství*. [B.r.]. Dostupné tiež z: https://moodle-vyuka.cvut.cz/course/view.php?id=8689. (cit. 14.03.2024).

52. *BI-PST Domácí úkol*. 2023. Dostupné tiež z: https://courses.fit.cvut.cz/BI-PST/homework/index.html. (cit. 17.03.2024).

53. *BI-PST Hodnocení*. 2023. Dostupné tiež z: https://courses.fit.cvut.cz/BI-PST/classification/index.html. (cit. 17.03.2024).

54. *BI-KOM Hodnocení*. 2022. Dostupné tiež z: https://courses.fit.cvut.cz/BI-KOM/classification.html. (cit. 17.03.2024).

55. *BI-KOM Semestrální práce*. 2022. Dostupné tiež z: https://courses.fit.cvut.cz/BI-KOM/semestral.html. (cit. 17.03.2024).

56. *BI-PRR Hodnocení*. 2023. Dostupné tiež z: https://courses.fit.cvut.cz/BI-PRR/classification/index.html. (cit. 19.03.2024).

57. *B222-BI-PRR.21 - Projektové řízení*. [B.r.]. Dostupné tiež z: https://moodle-vyuka.cvut.cz/course/view.php?id=8709. (cit. 13.03.2024).

58. *B231-BI-TIS.21 - Tvorba informačních systémů*. [B.r.]. Dostupné tiež z: https://moodle-vyuka.cvut.cz/course/view.php?id=9568. (cit. 13.03.2024).
59. *BI-OOP Classification*. 2023. Dostupné tiež z: https://courses.fit.cvut.cz/BI-OOP/classification/index.html. (cit. 17.03.2024).
60. *BI-OOP Projects*. 2023. Dostupné tiež z: https://courses.fit.cvut.cz/BI-OOP/projects/index.html. (cit. 17.03.2024).
61. *Qualitative vs Quantitative Research: What's the Difference?* [B.r.]. Dostupné tiež z: https://www.simplypsychology.org/qualitative-quantitative.html. (cit. 2024-02-19).
62. WIEGERS, Karl; BEATTY, Joy. *Software Requirements*. Microsoft Press, 2013. ISBN 978-0-7356-7966-5.
63. DYSON, Jonathan. *Conjoining FURPS and MoSCoW to Analyse and Prioritise Requirements*. 2019. Dostupné tiež z: https://www.linkedin.com/pulse/conjoining-furps-moscow-analyse-prioritise-jonathan-dyson. (cit. 13.03.2024).
64. FOWLER, Martin. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, 2003. ISBN 9780321193681.
65. *Studijní plány a jejich doporučené průchody oboru/specializace Softwarové inženýrství 2021*. [B.r.]. Dostupné tiež z: https://bk.fit.cvut.cz/cz/obory/obor1203491441605.B.B.P.1203490917005.html. (cit. 18.03.2024).
66. *Studijní plány a jejich doporučené průchody oboru/specializace Webové inženýrství 2021*. [B.r.]. Dostupné tiež z: https://bk.fit.cvut.cz/cz/obory/obor1203491579105.B.B.P.1203490917005.html. (cit. 18.03.2024).
67. *Typescript*. Microsoft, [b.r.]. Dostupné tiež z: https://www.typescriptlang.org/. (cit. 17.03.2024).
68. *JavaScript*. Mozilla Foundation, [b.r.]. Dostupné tiež z: https://developer.mozilla.org/en-US/docs/Web/JavaScript. (cit. 17.03.2024).
69. YOU, Evan. *Introduction | Vue.js*. [B.r.]. Dostupné tiež z: https://vuejs.org/guide/introduction.html. (cit. 17.03.2024).
70. NAGAR, Tarun. *Angular vs React vs Vue: Core Differences*. [B.r.]. Dostupné tiež z: https://devtechnosys.com/insights/tech-comparison/angular-vs-react-vs-vue/. (cit. 17.03.2024).
71. DEINMA, Simeon. *Angular, React, Vue – A Three-Way Comparison*. [B.r.]. Dostupné tiež z: https://blog.openreplay.com/angular-react-vue--a-three-way-comparison/. (cit. 17.03.2024).
72. *Nuxt: The Intuitive Vue Framework*. [B.r.]. Dostupné tiež z: https://nuxt.com/. (cit. 17.03.2024).

73. *Nuxt: Introduction*. [B.r.]. Dostupné tiež z: https://nuxt.com/docs/getting-started/introduction. (cit. 17.03.2024).

74. *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. [B.r.]. Dostupné tiež z: https://tailwindcss.com/. (cit. 17.03.2024).

75. *BI-TJV*. 2023. Dostupné tiež z: https://courses.fit.cvut.cz/BI-TJV/index.html. (cit. 21.03.2024).

76. *Web Framework Benchmarks*. Techempower, [b.r.]. Dostupné tiež z: https://www.techempower.com/benchmarks. (cit. 18.03.2024).

77. *C#*. Microsoft, [b.r.]. Dostupné tiež z: https://dotnet.microsoft.com/en-us/languages/csharp. (cit. 15.03.2024).

78. *A tour of the C# language*. Microsoft, [b.r.]. Dostupné tiež z: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. (cit. 15.03.2024).

79. *What is .NET?* Microsoft, [b.r.]. Dostupné tiež z: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet. (cit. 15.03.2024).

80. *Introduction to .NET*. Microsoft, [b.r.]. Dostupné tiež z: https://learn.microsoft.com/en-gb/dotnet/core/introduction. (cit. 15.03.2024).

81. *ASP.NET Core*. Microsoft, [b.r.]. Dostupné tiež z: https://dotnet.microsoft.com/en-us/apps/aspnet. (cit. 17.03.2024).

82. *What is ASP.NET Core?* Microsoft, [b.r.]. Dostupné tiež z: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet-core. (cit. 17.03.2024).

83. *Overview of ASP.NET Core*. Microsoft, [b.r.]. Dostupné tiež z: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-8.0. (cit. 17.03.2024).

84. *Entity Framework documentation hub*. Microsoft, [b.r.]. Dostupné tiež z: https://learn.microsoft.com/en-us/ef/. (cit. 17.03.2024).

85. *Entity Framework Core*. Microsoft, [b.r.]. Dostupné tiež z: https://learn.microsoft.com/en-us/ef/core/. (cit. 17.03.2024).

86. RAJ, Nidhi. *Minimum Viable Product (MVP) - What is it & how to start*. [B.r.]. Dostupné tiež z: https://www.atlassian.com/agile/product-management/minimum-viable-product. (cit. 13.03.2024).

87. PERNICE, Kara. *UX Prototypes: Low Fidelity vs. High Fidelity*. 2016. Dostupné tiež z: https://www.nngroup.com/articles/ux-prototype-hi-lo-fidelity/. (cit. 29.03.2024).

88. FOWLER, Martin. *Patterns of Enterprise Application Architecture: Data Transfer Object*. 2002. Dostupné tiež z: https://martinfowler.com/eaaCatalog/dataTransferObject.html. (cit. 29.03.2024).

89. *Data Class*. [B.r.]. Dostupné tiež z: https://refactoring.guru/smells/data-class. (cit. 21.04.2024).

90. BOGARD, Jimmy. *AutoMapper*. [B.r.]. Dostupné tiež z: https://docs.automapper.org/en/stable/index.html. (cit. 21.04.2024).

91. FOWLER, Martin. *Patterns of Enterprise Application Architecture: Repository*. 2002. Dostupné tiež z: https://martinfowler.com/eaaCatalog/repository.html. (cit. 01.04.2024).

92. *.NET: DbContext Class*. Microsoft, [b.r.]. Dostupné tiež z: https://learn.microsoft.com/en-us/dotnet/api/system.data.entity.dbcontext?view=entity-framework-6.2.0. (cit. 02.04.2024).

93. *Dependency injection in ASP.NET Core*. Microsoft, 2012. Dostupné tiež z: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-8.0. (cit. 23.04.2024).

94. *Introduction to JSON Web Tokens*. [B.r.]. Dostupné tiež z: https://jwt.io/introduction. (cit. 29.04.2024).

95. *JSON Web Tokens*. [B.r.]. Dostupné tiež z: https://auth0.com/docs/secure/tokens/json-web-tokens. (cit. 29.04.2024).

96. *What is OAuth 2.0?* [B.r.]. Dostupné tiež z: https://auth0.com/intro-to-iam/what-is-oauth-2. (cit. 29.04.2024).

97. MORAN, Kate. *Usability Testing 101*. 2019. Dostupné tiež z: https://www.nngroup.com/articles/usability-testing-101/. (cit. 23.04.2024).

98. NIELSEN, Jakob. *Why You Only Need to Test with 5 Users*. 2000. Dostupné tiež z: https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/. (cit. 23.04.2024).

99. NIELSEN, Jakob. *How Many Test Users in a Usability Study?* 2012. Dostupné tiež z: https://www.nngroup.com/articles/how-many-test-users/. (cit. 23.04.2024).

Obsah príloh

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `impl` ..... zdrojové kódy implementácie
    - `backend` ..... zdrojové kódy backend aplikácie
    - `frontend` ..... zdrojové kódy frontend aplikácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `hujomark-thesis.pdf` ..... text práce vo formáte PDF