FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Anviron - automatizované prostředí
Student:	Adam Šánta
Vedoucí:	Ing. Filip Glazar
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat webový nástroj pro snadnou tvorbu prostředí. Prostředí bude primárně sloužit studentům a softwarovým inženýrům pro jednoduchou konfiguraci softwarového prostředí, které je možné využít například pro vývoj aplikací, studium technologií atd. Pro samotnou realizaci využijte kontejnerizaci. Proveďte analýzu vhodných nástrojů například Docker nebo Podman.

Postupujte dle následujících kroků:

1) Analyzujte potřeby cílových uživatelů
2) Na základě analýzy specifikujte funkční a nefunkční požadavky
3) Vyberte vhodné nástroje pro realizaci systému
4) Navrhněte kompletní systém včetně podpůrných nástrojů
5) Implementujte prototyp softwarového řešení
6) Vytvořte ukázkové prostředí např. pro vhodné předměty v rámci ČVUT FIT
7) Otestujte vytvořená prostředí a samotnou aplikaci
8) Na základě poznatků z testování zhodnoťte stav prototypu a případně navrhněte vhodná vylepšení, či úpravy

---

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 4. ledna 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Anviron – automatizované prostředí

Adam Šánta

Katedra softwarového inženýrství
Vedúci práce: Ing. Filip Glazar

10. mája 2023

Poďakovanie

Rád by som poďakoval svojmu vedúcemu práce, Ing. Filipovi Glazarovi za pomoc so smerovaním práce, jeho trpezlivosť, čas a rady. Poďakovanie tiež patrí mojej rodine — Viere, Danielovi, Milanovi a Valérii, ktorí mi nielen počas písania tejto práce, ale najmä počas celého bakalárskeho štúdia poskytovali svoju bezhraničnú podporu. Špeciálne poďakovanie by som rád venoval Davidovi, bez ktorého by táto práca nikdy neexistovala.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb, autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k používaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo používať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, ale len pre nezárobkové účely. Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 10. mája 2023 . . . . . .

Odkaz na túto prácu

Šánta, Adam. *Anviron – automatizované prostředí.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Táto práca sa zameriava na vývoj prototypu webovej aplikácie s názvom Anviron, ktorej cieľom je zjednodušenie procesu nastavenia vývojového prostredia pre študentov softvérového inžinierstva a podobných študijných programov. Cieľovou užívateľskou skupinou sú primárne študenti nižších ročníkov, ktorí s nastavovaním prostredí a riešením prípadných problémov, ktoré počas neho môžu nastať, často nemajú dostatok skúseností. Aplikácia pre zostavenie generovaných prostredí a réžiu ich behu využíva princíp kontajnerizácie. Implementovaný prototyp je následne testovaný automatickými testovacími metodikami a metódou užívateľského testovania.

Kľúčové slová vývojové prostredie, automatizácia, cloud, Docker, virtualizácia

Abstract

This thesis is focused on development of a web application prototype named Anviron, which aims to simplify the process of development environment setup for students of software engineering or similar study branches. The targeted user segment consists primarily of lower grade university students, who often lack experience not only with development environments setup, but mainly

with the process of troubleshooting any related problems, which may appear during the setup. The application uses the principle of containerization for assembly and run of the generated environments. The implemented prototype is consequently tested using both automatic testing methods and user testing.

Keywords development environment, automation, cloud, Docker, virtualization

Obsah

Úvod **1**

1 Analýza **5**
1.1 Popis problému 5
1.2 Formulácia zadania 6
1.3 Analýza externých systémov 7
1.4 Aktéri systému 8
1.5 Funkčné požiadavky 9
1.6 Nefunkčné požiadavky 10
1.7 Doménový model 10
1.8 Analýza podobných nástrojov 15

2 Návrh **19**
2.1 Správa infraštruktúry 19
2.2 Vývoj softvéru 23
2.3 Architektúra webovej aplikácie 26
2.4 Modelovanie systému 28
2.5 Návrh procesov 28

3 Implementácia **33**
3.1 Nastavenie vývojového prostredia 33
3.2 Modul EasyAdmin 34
3.3 Zabezpečenie 35
3.4 Správa fakultných dát 36
3.5 Registrácia a prihlasovanie užívateľov 37
3.6 Správa obrazov pre vývojové prostredia 37
3.7 Spustenie a vypnutie vývojového prostredia 38
3.8 Integrácie 39

4 Testovanie **41**

4.1 Metodiky testovania 41
4.2 Výber vhodných testovacích metodík 42
4.3 Implementácia vybraných testovacích metodík 43
4.4 Zhrnutie 45

5 Zhrnutie a výhľad do budúcna **47**
5.1 SWOT Analýza 47

Záver **51**

Literatúra **53**

A Zoznam použitých skratiek a termínov **57**

B Obsah priloženého média **59**

C Kompletný doménový model **61**

D Kompletný diagram tried **63**

E Kompletný relačný model **65**

F Ukážka použitia prototypu **69**

Zoznam obrázkov

1.1 Doménový model: Správa užívateľov 12
1.2 Doménový model: Jadro aplikácie 15

2.1 Porovnanie virtualizácie a kontajnerizácie 20
2.2 Architektúra nástroja Docker 21
2.3 Diagram aktivít: Synchronizácia študijných dát 29
2.4 Príklad behov procesu: Spustenie vývojového prostredia 32

3.1 Bežné členenie zdrojového kódu aplikácie v Symfony 34
3.2 Konfiguračný súbor Dockerfile pre vzor obrazu prostredí 38
3.3 Diagram tried: Spustenie a vypnutie prostredia 39

C.1 Kompletný doménový model 62

D.1 Kompletný diagram tried 63

E.1 Kompletný relačný model — časť 1 66
E.2 Kompletný relačný model — časť 2 67

F.1 Domovská obrazovka pred autentizáciou užívateľa 69
F.2 Autentizácia užívateľa pomocou SSO 70
F.3 Domovská obrazovka po autentizácii užívateľa 70
F.4 Zoznam obrazov pre vývojové prostredia 71
F.5 Vytvorenie obrazu pre vývojové prostredie 71
F.6 Detail obrazu pre vývojové prostredie 72
F.7 Úspešné zostavenie obrazu pre prostredie 72
F.8 Zobrazenie všetkých zostavení obrazu pre prostredie 73
F.9 Zobrazenie dostupných prostredí 73
F.10 Vytvorenie nového prostredia 74
F.11 Priradenie potreby alokácie portu k prostrediu 74
F.12 Priradenie potreby alokácie súborov k prostrediu 75

F.13 Detail prostredia 75
F.14 Prihlásenie do virtuálneho prostredia pomocou vygenerovaného hesla 76
F.15 Ukážka práce na zdrojovom kóde vo virtuálnom prostredí 76

Zoznam tabuliek

4.1 Znenie SUS dotazníka 44
4.2 Vyhodnotenie SUS dotazníka 45

Úvod

V bakalárskej práci sa budem zaoberať analýzou, návrhom, implementáciou a testovaním prototypu webovej aplikácie určenej na podporu výuky predmetov ohľadom programovania. Za modelových používateľov navrhovaného systému budem v práci najčastejšie považovať študentov a vyučujúcich kurzov programovania v rôznych programovacích jazykoch, akými sú napríklad C, C++, Java, Python alebo PHP.

Mojou hlavnou motiváciou pre voľbu tejto témy boli skúsenosti zo štúdia na Fakulte informačních technologií ČVUT, kde som pri výuke niektorých predmetov vnímal, že na úvodných laboratórnych cvičeniach niektorých predmetov je neúmerne veľké množstvo času venované nastavovaniu lokálneho vývojového prostredia. To je nevyhnutné pre účasť na praktickej časti výuky predmetu. Kladúc si otázku, či by tento proces nešiel zefektívniť, ma moje myšlienky priviedli k predstave, že vývojové prostredia by mohli byť pre študentov predgenerované na vzdialenom serveri. Po hlbšom zamyslení som dospel až k finálnej predstave o systéme, ktorý by túto funkcionalitu umožňoval.

Ako hlavný prínos potenciálnej prevádzky takéhoto systému vidím uľahčenie práce s nastavovaním vývojového prostredia novým študentom informatiky a podobných odborov. Prototyp navrhovaného systému si kladie za cieľ uľahčiť začínajúcim študentom programovania nastavenie vývojového prostredia. Možnosťou prístupu z ľubovoľného konvenčného webového prehliadača sa pokúsim užívateľov ušetriť od riešenia problémov s kompatibilitou systémov používaných pre výuku.

V práci sa budem zaoberať prevažne analýzou, návrhom, implementáciou a testovaním prototypu technického riešenia. Súčasťou práce je ukážka jeho funkčnosti na nízkej škále počtu užívateľov.

V nadväznosti na túto prácu vidím priestor rozšíriť vyvíjaný prototyp webovej aplikácie o adaptéry zabezpečujúce kompatibilitu s inými kontajnerizačnými platformami (napr. Podman), adaptéry na rôzne úložiská dát (napr. S3, Git repozitáre, atp.), adaptéry na bežne používané nástroje pre správu

infraštruktúry (napr. Kubernetes), alebo už spomínané šablóny pre viaceré textové editory. Priestor na vylepšenie tiež vidím v oblasti užívateľského rozhrania a podporu konfigurácie podľa potrieb užívateľa.

Verím, že táto práca bude inšpiráciou pre fakulty, ktoré organizujú výuku predmetov zameraných na základy programovania, prípadne špecifiká konkrétneho programovacieho jazyka, aby svojim študentom poskytli jednoducho dostupný, predpripravený priestor na vývoj zdrojového kódu. Tým môžu pomôcť svojim študentom upriamiť pozornosť na štúdium pred problémami s nastavovaním prostredia na vlastnom fyzickom zariadení a inými súvisiacimi technikáliami.

Cieľ a štruktúra práce

Primárnym cieľom práce je vytvoriť webovú aplikáciu pre automatické generovanie vývojových prostredí. Tento cieľ dosiahnem splnením nasledovných podcieľov, ktoré sú členené podľa príslušnosti k jednotlivým kapitolám.

- Analýza
  - Identifikovať aktérov, ktorí so systémom budú interagovať.
  - Zadefinovať prípady použitia systému a ich vzťahy k jednotlivým aktérom.
  - Identifikovať relevantné externé zdroje dát pre vyvíjaný systém.
  - Vytvoriť doménový model na základe definovaných požiadaviek na systém.
  - Zadefinovať konkurenčnú výhodu vyvíjaného prototypu oproti podobným, už existujúcim nástrojom.
- Návrh
  - Zmapovať princípy a technológie relevantné pre návrh prototypu.
  - Vybrať vhodné nástroje a technológie pre implementáciu prototypu.
  - Transformovať doménový model na diagram tried a relačný (databázový) model.
- Implementácia
  - Navrhnúť a zdokumentovať architektúru implementovaného prototypu webovej aplikácie.
  - Popísať implementačné špecifiká vzniknutej webovej aplikácie v nadväznosti na návrhovú časť práce.
- Testovanie

  - Zvoliť vhodnú metodiku testovania pre jednotlivé časti aplikácie.
  - Vybrať vhodné technológie a prostriedky pre realizáciu testovania.
  - Implementovať alebo jednorázovo vykonať testovanie aplikácie pomocou zvolených princípov a technológií.

- Zhrnutie a výhľad do budúcna
  - Pomenovať silné a slabé stránky navrhnutého prototypu webovej aplikácie.
  - Navrhnúť možnosti ďalšieho rozšírenia prototypu.

Naopak, cieľom tejto práce nie je navrhovať optimalizované užívateľské rozhranie pre aplikáciu, prípadne akokoľvek podrobnejšie testovať a ladiť rozhranie medzi človekom a počítačom. Pretože sa jedná o prototyp, za ambíciu si nekladiem ani úplnú elimináciu bezpečnostných zraniteľností, ktoré v prototype aplikácie môžu hypoteticky vzniknúť. Zameriavať sa nebudem ani na variabilitu textových editorov, ktoré môže užívateľ systému používať.

Kapitola 1

Analýza

V analytickej časti tejto práce sa zameriam na modelové prostredie FIT ČVUT. Budem sa snažiť definovať systémové požiadavky tak, aby čo najlepšie a čo najpresnejšie vystihli potreby jej študentov, vyučujúcich a správcov interných systémov.

V kontexte študentov sa zameriam najmä na študentov nižších ročníkov. Práve na riešenie problémov typických pre nich sa vyvíjanú aplikáciu snažím cieliť najviac. Verím totiž, že sú to práve oni, kto trávi najviac času snahou o splnenie technických požiadaviek na vývojové postredia, ktoré dostávajú zo strany vyučujúcich.

V priebehu kapitoly zadefinujem problém, ktorý bude moja aplikácia riešiť. Z definície problému bude zrejmé, kto problému čelí a kto bude s navrhovaným systémom interagovať. Tieto osoby budem považovať za aktérov navrhovaného systému. K jednotlivým aktérom následne zhrniem ich prípady použitia systému, pomocou ktorých vymedzím požadovaný rozsah implementácie prototypu. Zmapujem existenciu podobných nástrojov a zadefinujem konkurenčnú výhodu svojej aplikácie.

1.1 Popis problému

Súčasťou hodnotenia vysokoškolských kurzov programovania je zväčša aj praktická časť. Jedná sa o drobnejšie úlohy hodnotené automaticky (jednotkovými testmi neviditeľnými pre študenta) alebo semestrálne projekty väčšieho rozsahu. Tieto úlohy sú hodnotené buď automaticky – sadou jednotkových testov, alebo manuálne – vyučujúcimi daného predmetu, prípadne kombináciou oboch možností.

Dosiahnutie nenulového bodového zisku z praktickej časti je zvyčajne podmienené tým, že študent rieši úlohy v určenom programovacom jazyku na počítači. Aby mu toto bolo umožnené na vlastnom zariadení, je nutné, aby si pri zápise do takéhoto kurzu nainštaloval potrebné nástroje. Tým sa rozumejú

predovšetkým:

- kompilátory (napr. *gcc* pre programovací jazyk C),
- interprety (napr. interpret programovacieho jazyka PHP),
- ladiace nástroje (napr. *valgrind* pre ladenie chýb v programoch napísaných v jazyku C)
- pokročilé editory a vývojové prostredia (IDE) umožňujúce jednoduchšiu prácu so zdrojovým kódom (napr. *Visual Studio Code*)

Táto požiadavka zo strany vyučujúcich smerom k študentovi kurzu môže so sebou priniesť viaceré problémy, z ktorých uvediem niekoľko príkladov.

- **Nedostatočný výkon študentovho zariadenia.** Študent nemusí disponovať dostatočne výkonným zariadením, aby spĺňalo minimálne systémové požiadavky na inštaláciu potrebných nástrojov. Nákup zariadenia spĺňajúceho minimálne systémové požiadavky môže byť mimo finančných možností študenta.
- **Problémy s kompatibilitou.** Požadované nástroje (resp. ich požadované verzie) nemusia byť kompatibilné s operačným systémom prevádzkovaným na študentovom počítači. Zmena operačného systému je časovo náročná a vo väčšine prípadov nie je zo strany študenta – používateľa žiadúca.
- **Čas potrebný na inštaláciu.** Inštalácia nástrojov vyžaduje, aby ju študent na svojom zariadení spustil a počkal do jej úspešného dokončenia. Tento časový interval môže trvať na menej výkonných zariadeniach trvať rádovo desiatky minút až jednotky hodín a stáva sa tak blokujúcim prvkom pre vývoj softvéru.
- **Zlyhanie inštalácie.** Pri inštalácii nástrojov môže nastať neočakávaná chyba, kvôli ktorej systém nemohol inštaláciu dokončiť. Vzhľadom na fakt, že inštalované nástroje môžu byť pre študenta neznáme, chybová hláška môže vyžadovať ďalšie štúdium dokumentácie inštalovaného nástroja. Toto vyžaduje alokáciu väčšieho množstva študentovho času.

Uvedené problémy často vedú k neúmerným časovým stratám a frustrácii na strane študenta.

1.2 Formulácia zadania

Nástroj umožní vyučujúcim kurzu nakonfigurovať obraz prostredia pre vývoj softvéru s presnou definíciou verzií používaného vývojového softvéru a potrebných ladiacich nástrojov. Vďaka napojeniu na informačné systémy FIT

ČVUT dokáže nástroj vygenerovať študentovi vývojové prostredia pre predmety, ktoré aktuálne študuje, prípadne v minulosti študoval.

Systém by mal byť schopný pracovať nezávisle od voľby programovacích jazykov a iných podporných nástrojov. Jediným obmedzením je možnosť inštalácie daného nástroja na operačnom systéme linuxového typu. Pre účely spúšťania virtuálnych prostredí na strane servera systém využije princíp kontajnerizácie.

Návrh systému by mal klásť dôraz na jednoduchú nasaditeľnosť aj na menej výkonné (ergo lacnejšie) webové servery, kde sa očakáva menšia záťaž. V prípade nutnosti škálovania na vyššiu záťaž systém poskytne administrátorovi možnosť dopracovať vlastnú implementáciu rozhrania pre spúšťanie jednotlivých vývojových prostredí. V rámci prototypu bude zahrnutá výhradne jednoduchá implementácia spúšťania prostredí pomocou kontajnerizačnej služby priamo na webovom serveri, kde beží prototyp aplikácie. Možnosť inštalácie bez nutnosti správy vlastného orchestračného nástroja (resp. platieb zaň) by mala byť jednou zo silných stránok tohoto projektu. Niekoľko podobných dostupných nástrojov totiž už podobnú funkcionalitu ponúka, no žiaden z nich nedokáže pracovať so servermi bez orchestračného nástroja (akým je napr. Kubernetes).

Vyvíjaný nástroj by mal byť navrhnutý v súlade s aktuálnym trendom v kontexte architektúry a návrhu webových aplikácii. Tým sa myslí najmä presun výpočtového výkonu z klientskych zariadení na server. Tento prístup znižuje systémové nároky na klientske zariadenia, čím sa poskytovaná služba stáva dostupnou pre užívateľov s nižšou úrovňou technickej výbavy a umožňuje prístup k dátam užívateľa z viacerých zariadení pripojených do siete (počítač, tablet, mobil, atp.). Server webovej aplikácie tiež čiastočne preberá zodpovednosť za zabezpečenie obsahu vo vlastníctve užívateľa. Správnym použitím autentizačných a autorizačných metód prináša užívateľovi benefit vo forme zvýšeného zabezpečenia dát.

1.3 Analýza externých systémov

Za externý systém považujem ľubovoľný informačný systém, ktorý disponuje údajmi relevantnými pre funkčnosť navrhovaného prototypu aplikáce a zároveň pomocou API umožňuje prístup k takýmto údajom. Cieľom analýzy externých systémov je identifikovať, ktoré systémy prevádzkované v modelovom prostredí FIT ČVUT sú pre moju prácu relevantné, resp. ktoré z týchto systémov disponujú údajmi potrebnými pre prevádzku navrhovanej aplikácie. Pre každý systém vymedzím rozsah údajov, ktoré sú pre túto prácu relevantné.

Systém KOS

Systém KOS je študijný informačný systém využívaný na všetkých fakultách ČVUT. Sú v ňom zaznamenávané všetky úkony spojené so štúdiom od prijatia

na univerzitu, zaznamenávanie študijných výsledkov, až po ukončenie štúdia štátnou záverečnou skúškou. [1]

Systém KOS umožňuje pomocou rozhrania KOS API prístup k dátam o výuke na FIT ČVUT. [2]

Pre túto prácu sú relevantné najmä zdroje týkajúce sa semestrov (Semesters), predmetov (Courses), študentov (Students), vyučujúcich (Teachers) a vzťahov medzi nimi. Vyvíjaný prototyp aplikácie môže KOS API používať ako primárny zdroj údajov o uvedených entitách. Automatická synchronizácia eliminuje prípadnú potrebu zadávania takýchto dát do aplikácie manuálne spávcom systému.

Prístup do KOS API je zabezpečený fakultným autorizačným serverom na báze štandardu OAuth 2.0.

Systém Usermap

Usermap je aplikácia pre správu vybraných osobných údajov, hlavný kontaktný adresár na ČVUT a nástroj pre správu užívateľských rolí väčšiny informačných systémov ČVUT. [3]

Systém Usermap je pre túto prácu relevantným, pretože poskytuje možnosť dohľadania podrobnejších údajov o užívateľovi podľa užívateľského mena, najmä celé meno užívateľa.

Podobne, ako je tomu pri systéme KOS, prístup do Usermap API je zabezpečený fakultným autorizačným serverom na báze štandardu OAuth 2.0.

1.4 Aktéri systému

Pod pojmom aktér rozumiem externú entitu interagujúcu so systémom, ktorá si plní svoje ciele pomocou služieb poskytovaných systémom. Pri interakcií so systémom zastáva aktér niektorú zo špecifikovaných rolí aktéra. Aktér môže iniciovať viaceré prípady použitia paralelne (súbežne). Jednotlivé prípady použitia sa môžu vzájomne vylučovať. [4]

Počas analýzy som vytipoval tri hlavné role užívateľov, t.j. aktérov, ktorí so systémom budú interagovať:

1. študent predmetu v rámci daného semestra,
2. vyučujúci predmetu v rámci daného semestra,
3. správca systému.

Každá rola Študent a Vyučujúci je viazaná na beh predmetu. Ten je identifikovaný predmetom (napr. „Programování v jazyku PHP“) a aktuálnym semestrom (napr. „B211“). Z bežnej praxe pozorovateľnej na fakulte plynie, že je prípustné, aby študent daného predmetu v danom semestri bol zároveň

učiteľom iného predmetu v tom istom semesti. Nie je však prípustné, aby jeden užívateľ bol študentom a zároveň učiteľom toho istého predmetu v tom istom semestri. V systéme ako aktér vystupuje aj čas.

1.5 Funkčné požiadavky

V práci zachytím funkčné požiadavky prostredníctvom prípadov použitia priradených k jednotlivým aktérom. Prípady použitia sú centrálnym mechanizmom odporúčaným pre vyhľadanie a definíciu funkčných požiadaviek. Prípady použitia vymedzujú správanie systému. Pri analýze požiadaviek je nežiadúce hľadať riešenia, ako požiadavky realizovať, namiesto toho považujeme navrhovaný systém za „čiernu skrinku“. Až v návrhovej časti práce navrhneme riešenie, ktoré požiadavky spĺňa. [5, s. 48]

Ľubovoľný užívateľ

F1.1 Prihlásenie pomocou školského účtu (SSO)

F1.2 Odhlásenie zo systému

Študent predmetu

F2.1 Zobrazenie zoznamu dostupných prostredí pre vývoj, v závislosti od predmetov, ktoré študent študuje.

F2.2 Spustenie ľubovoľného prostredia pre vývoj.

F2.3 Možnosť spustenia ľubovoľného príkazu v termináli bez administrátorských práv (napr. kompilácia zdrojového kódu, spustenie skompilovaného programu, ...).

F2.4 Interakcia so súborovým systémom prostredia (napr. vytvorenie súboru, úprava súboru, zmazanie súboru).

F2.5 Uloženie vyvíjaného projektu vo vývojovom prostredí na úložisko poskytované serverom.

F2.6 Vypnutie spusteného prostredia pre vývoj.

Učiteľ predmetu

F3.1 Vytvorenie nového vývojového prostredia rozšírením základného prostredia poskytnutého ľubovoľným správcom systému.

F3.2 Testovacie spustenie vytvoreného vývojového prostredia.

F3.3 Sprístupnenie vytvoreného vývojového prostredia všetkým študentom zvoleného predmetu v zvolenom semestri.

Užívateľ s rolou Správca systému

F4.1 Vytvorenie nového základného prostredia reprezentovaného súborom pre vytvorenie obrazu konrajnera (napr. *Dockerfile*).

F4.2 Zverejnenie vytvoreného základného prostredia všetkým vyučujúcim pre ďalšie použitie.

F4.3 Spustenie synchronizácie údajov o výuke so systémom KOS pomocou rozhrania KOS API.

Čas

F5.1 Vypnutie prostredia pre vývoj spusteného študentom po dlhšom čase neaktivity.

1.6 Nefunkčné požiadavky

Pod nefunkčnou požiadavkou rozumiem najmä takú požiadavku na systém, ktorá sa týka napríklad použiteľnosti, spoľahlivosti, výkonnosti, podpory, implementácie, rozhrania alebo právnych záležitostí. [5, s. 42–43] Nefunkčné požiadavky by nemali zasahovať do logiky fungovania systému.

N1 Systém musí byť v záujme maximalizácie prístupnosti dostupný a plne použiteľný z ľubovoľného bežne používaného webového prehliadača.

N2 Systém pre internú reprezentáciu a trvalé ukladanie dát využíva relačnú databázu.

N3 Systém pre svoju prevádzku nebude vyžadovať orchestračný nástroj (akým je napr. Kubernetes) pre správu infraštruktúry, môže však využívať kontajnerizačné služby (Docker, Podman, atp.)

1.7 Doménový model

Tak, ako je popísané v [6], ciele doménového modelovania sú zvyčajne predovšetkým:

- zaviesť projektový slovník pojmovov, ktorý zabezpečí jednoznačnosť v komunikácii medzi jednotlivými diskutujúcimi v kontexte projektu,
- zachytiť vzťahy medzi týmito pojmami pomocou agregácie a generalizácie,
- vymedziť rámec projektu.

Agregácia je vzťah, ktorý upresňuje kardinalitu, resp. početnosť medzi jednotlivými objektmi.

Napríklad, jednen *zámok* dokáže mať priradených $1, 2, ...n$ objektov typu *kľúč*, ktoré do daného zámku pasujú. Na druhej strane, každý kľúč musí mať v tomto vzťahu priradený práve jeden objekt zámok, ktorý dokáže odomknúť.

Agregácia nám v tomto konkrétnom príklade zachytáva, že za žiadnych okolností nie je možné, aby existoval taký kľúč, ktorý nedokáže odomknúť žiaden zámok. Rovnako nie je žiadúce, aby jeden kľúč dokázal odomknúť viacero zámkov.

Zaužívanou praxou v softvérovom inžinierstve je pre vymedzenie kardinality používať výrazy *one-to-one*, *one-to-many*, *many-to-one* alebo *many-to-many*. Tieto výrazy však v sebe nezachytávajú nulovosť vzťahu na oboch stranách.

Generalizácia je vzťah, ktorý medzi jednotlivými pojmami (objektmi) vytvára hierarchiu. Určuje, ktorý z dvoch uvedených objektov je nadtypom, resp. podtypom. Dôležitým faktom je, že jednotlivé podtypy sa navzájom vylučujú, t.j. sú disjunktné.

Napríklad, objekty *osobné auto* alebo *motorka* významovo rozširujú objekt *motorové vozidlo*. Nemôže sa však stať, že jedno motorové vozidlo by bolo zároveň aj motorkou, aj osobným autom.

Doménový tiež slúži ako základ pre diagram tried, ktorý je štandardne súčasťou návrhu aplikácie. Podľa [6] je doménový model zjednodušeným diagramom tried, s čiarami medzi jednotlivými triedami (doménovými objektmi), ktoré poukazujú na ich vzájomné vzťahy.

V tejto časti práce uvádzam pre ilustráciu len náčrty doménového modelu s objektami a vyznačenými vzťahmi medzi nimi. Následne textovo popisujem význam objektu a jeho najpodstatnejšie atibúty. Kompletný doménový model vrátane všetkých atribútov je súčasťou práce ako príloha.

Pre tvorbu diagramov bol použitý nástroj PlantUML. Kardinalita je na diagramoch značená podobne ako pri ERD (Entity Relationship Diagram, [7]) kvôli lepšej prehľadnosti.

1.7.1 Správa užívateľov a dát o štúdiu

Prvým krokom pri tvorbe doménového modelu bolo zachytenie potrieb aplikácie v kontexte správy užívateľov a správy dát o vzťahu užívateľov k fakulte. Model zachytáva nasledovné objekty:

User

Reprezentuje osobu užívateľa systému. K nevyhnutným atribútom patrí najmä:

- meno a priezvisko užívateľa, prípadne tituly pred menom a za menom,

- užívateľské meno užívateľa, zhodné s užívateľským menom používaným pre prihlásenie do ostatných interných systémov fakulty,
- množinu rolí, ktorá bola užívateľovi pridelená v rámci systému.

Subject

Reprezentuje predmet vyučovaný na fakulte. K nevyhnutným atribútom patrí najmä textový kód predmetu zo študijného informačného systému.

Term

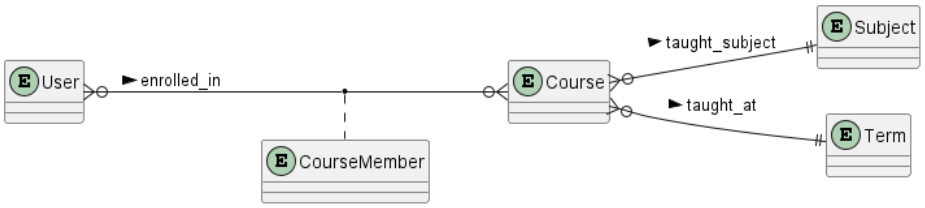
Reprezentuje semester, počas ktorého prebieha výkuka jednotlivých predmetov. K nevyhnutným atribútom patrí najmä textový kód semestra zo študijného informačného systému a dátumy jeho začiatku a konca.

Course

Dáva do súvislosti predmet a semester, čím reprezentuje beh konkrétneho predmetu v rámci konkrétneho semestra (ďalej len "kurz"). Žiadnymi nevyhnutnými atribútmi objekt nedisponuje.

CourseMember

Definuje vzťah medzi užívateľom systému a kurzom, špecifikuje vzťah medzi nimi (učiteľ / študent). V prípade rozširovania systému o zachytenie študijných skupín by bolo potrebné pridať informáciu o takejto skupine. Pre implementáciu prototypu to však nebude potrebné.



Obr. 1.1: Doménový model: Správa užívateľov

1.7.2 Jadro aplikácie

Táto časť doménového modelu zachytáva potreby aplikácie pre splnenie všetkých vyššie zadefinovaných, najmä funkčných požiadaviek na vyvíjaný prototyp systému. Jedná sa predovšetkým o doménu správy jednotlivých prostredí, ich sprístupnenie prostredníctvom internetu a ukladania potebných dát.

EnvironmentImage

Reprezentuje recept na zostavenie obrazu systému, ktorý môže byť užívateľovi spustený ako vývojové prostredie. Nevyhnutným atribútom je súbor reprezentujúci takýto recept.

EnvironmentImageBuild

Reprezentuje konkrétne zostavenie obrazu systému s väzbou na úspešne zostavený obraz pomocou kontajnerizačného systému. Nevyhnutnými atribútmi sú:

- súbor reprezentujúci recept, podľa ktorého bol obraz zostavený,
- referencia na užívateľa, ktorý zostavenie vykonal.

EnvironmentFactory

Obohacuje zostavenie obrazu systému o inicializačný skript, ktorý dokáže nastaviť vývojové prostredie do takého stavu, v ktorom je pripravené pre používateľa. Objekt musí mať väzbu na konkrétny predmet, ktorý sa na fakulte vyučuje. Ďalej je potrebné špecifikovať, koľko portov, resp. ktoré konkrétne porty je nutné sprístupniť pre sieťovú komunikáciu medzi vývojovým prostredím a užívateľom. Rovnako je nutné reprezentovať, kde budú ukladané dáta, ktoré vzniknú používaním vývojového prostredia jeho cieľovým užívateľom. Nevyhnutnými atribútmi sú:

- textový názov prostredia,
- textová dokumentácia pre užívateľa alebo referencia na súbor, ktorý ju obsahuje,
- shell skript, ktorý sa vykoná pri každom spustení prostredia alebo referencia na súbor, ktorý tento skript obsahuje,
- shell skript, ktorý sa vykoná pred každým ukončením prostredia alebo referencia na súbor, ktorý tento skript obsahuje,
- referencia na užívateľa, ktorý za toto prostredie zodpovedá,
- kurzy, ku ktorým je toto prostredie priradené,
- príznak, ktorý určuje, či je prostredie viditeľné pre študentov kurzu, ku ktorému je toto prostredie priradené.

EnvironmentFactoryVolume

Reprezentuje potrebu perzistentnej pamäte (súborového adresára) pre potreby vývojového prostredia. Taktiež definuje spôsob interakcie tohoto adresára a zostaveného vývojového prostredia. Nevyhnutným atribútom je cesta (v rámci kontajnera), do ktorej sa má adresár mapovať.

EnvironmentFactoryPort

Reprezentuje potrebu sieťového portu pre sieťovú komunikáciu medzi serverom (prototypom vyvíjanej aplikácie) a užívateľom (klientom). Nevyhnutnými atribútmi sú:

- číselný kód portu, ktorého vystavenie (t.j. sprístupnenie prostredníctvom siete) vyžaduje prostredie pre svoj beh,
- typ pripojenia, ktoré bude prostredníctvom tohoto portu realizované (napr. TCP, UDP, ...)

EnvironmentRuntime

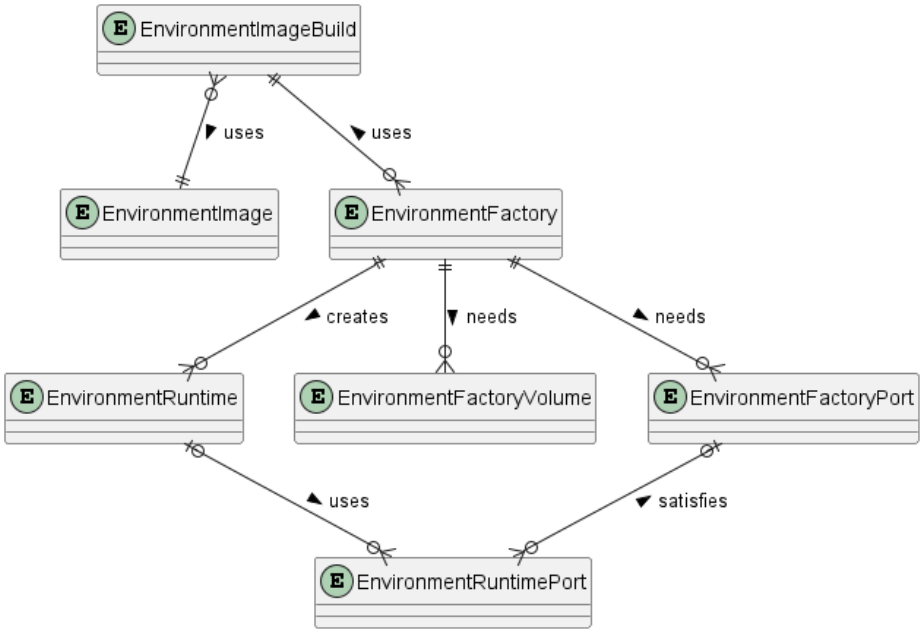
Reprezentuje spustenie vývojového prostredia a jeho následný beh. Nevyhnutnými atribútmi sú:

- referencia na užívateľa, ktorý beh vývojoveho prostredia využíva,
- referencia na EnvironmentFactory, kt. definuje spôsob, akým sa má vývojové prostredie spustiť,
- referencie na porty (EnvironmentRuntimePort), pomocou ktorých bude aplikácia sprístupnená na webe,
- textové súbory reprezentujúce štandardný a chybový výstup z behu inicializačného skriptu (definovaného v objekte EnvironmentFactory),
- textové súbory reprezentujúce štandardný a chybový výstup z behu skriptu pred ukončením behu prostredia (definovaného v objekte EnvironmentFactory),
- čas, v ktorom bol beh prostredia spustený,
- čas, v ktorom automat ukončí beh prostredia,
- čas, v ktorom bol beh prostredia ukončený.

EnvironmentRuntimePort

Reprezentuje konkrétnu alokáciu sieťového portu pre zvolené spustenie vývojového prostredia. Nevyhnutnými atribútmi sú:

- číselný kód portu, ktorý bude zverejnený na serveri, t.j. pomocou ktorého bude možné spojiť sa so spusteným vývojovým prostredím,
- referenciu na potrebu alokácie portu (EnvironmentFactoryPort).



Obr. 1.2: Doménový model: Jadro aplikácie

1.8 Analýza podobných nástrojov

V tejto časti práce sa zameriam na existujúce riešenia podobné tomu, aké sa snažím navrhnúť. Pre každé z riešení, ktoré považujem za podobné, uvediem jeho stručný popis, niekoľko silných a slabých stránok, minimálne technické požiadavky na prevádzku riešenia, prípadne jeho ďalšie charakteristické vlastnosti. Zhodnotím tiež cenovú dostupnosť riešení.

Analýza bola vykonaná v týždni 25. až 30. 04. 2022. Všetky údaje k analýze sú uvádzané a platné k tomuto dátumu.

1.8.1 Coder

Coder je platforma umožňujúca organizáciám bezpečne poskytovať vývojové prostredia (vrátane vzdialených IDE a ďalších súvisiacich zdrojov) pre tímy správcov systémov, správcov nástrojov a softvérových inžinierov. [8] Prevádzkovateľom služby je spoločnosť Coder Technologies Inc.

Nástroj pre svoju prevádzku vyžaduje orchestračný nástroj Kubernetes. K inštalácii do Kubernetes je v online dokumentácii dostupný užívateľský manuál pre správcu systému.

Cena nástroja je fixná bez ohľadu na využitie a záťaž. Činí 35 USD za každého užívateľa mesačne, pričom cena zahŕňa iba licenciu na používanie nástroja a aktualizácie. Nezahŕňa náklady na infraštruktúru, ktorú si musí organizácia využívajúca nástroj spravovať sama.

Firma, ktorá za nástrojom stojí, je tiež autorom a správcom riešenia „code-server“, ktoré umožňuje spustenie textového editora Visual Studio Code a prácu v ňom vo webovom prehliadači. Tento projekt má zverejnené zdrojové kódy pod MIT licenciou, t.j. je možné používať ho bez záruky aj na komerčné účely.

Za silné stránky platormy môžeme bezpochyby považovať kvalitne spracovanú dokumentáciu a rozsah systému, ktorý plne pokrýva požiadavky priemerného softvérového inžiniera vyvíjajúceho webové aplikácie. Naopak, vysoká cenovka a nevyhnutnosť správy infraštruktúry pre nástroj sú obrovskou nevýhodou tohoto nástroja.

1.8.2 Github Codespaces

Codespaces [9] je webová služba umožňujúca tvorbu virtuálnych vývojových prostredí podporovaných vysokým výpočtovým výkonom. Službu prevádzkuje spoločnosť GitHub, Inc., ktorá je dcérskou spoločnosťou Microsoft Corporation.

Pretože služba je prevádzkovaná výhradne v modeli SaaS, nemôže byť spustená na samostatnej infraštruktúre a z pohľadu zákazníka nemá žiadne minimálne systémové požiadavky.

Pre jednotlivcov je služba bezplatná, poplatok za používanie sa týka iba tímov a firiem. Cena za používanie nástroja je dynamická, v závislosti od využívaného výpočtového výkonu a množstva využívanej pamäte. Za jednu hodinu behu na dvojjadrovom procesore je účtovaných 0.18 USD, za každý využívaný gigabajt pamäte je účtovaných 0.07 USD mesačne.

Služba v aktuálnom stave podporuje iba jeden textový editor, ktorým je Visual Studio Code.

Silnou stránkou tohoto nástroja je správa infraštruktúry, ktorú za užívateľa preberá poskytovateľ služby, z čoho plynú aj nízke systémové požiadavky na infraštruktúru. Cenová dostupnosť pôsobí v porovnaní s ostatnými nástrojmi výhodne. Nevýhodou je príliš silná väzba na platformu Github, ktorá slúži

na vzdialené ukladanie a verzovanie zdrojových kódov. Pri použití alternatív, akými sú napr. Gtilab alebo Bitbucket už nie je tento nástroj vôbec použiteľný.

1.8.3 Gitpod

Gitpod [10] je aplikácia poskytujúca predzostavené a kolaboratívne vývojové prostredia vo webovom prehliadači. Službu zabezpečuje spoločnosť Gitpod GmbH.

Nástroj je možné používať na zdieľanej infraštruktúre (SaaS), ako aj v tzv. „self-hosted“ režime na vlastnej infraštruktúre. V režíme SaaS si užívateľ aj správca systému vystačia s webovým prehliadačom, pri voľbe „self-hosted“ režimu je potrebný orchestračný nástroj Kubernetes, resp. dostatočné skúsenosti s prácou s týmto nástrojom za účelom správneho nastavenia aplikácie.

Od voľby medzi týmito režimami sa odvíja cena. Dostupný je aj bezplatný program, ktorý je v režime SaaS limitovaný 50 hodinami mesačne, v režime „self-hosted“ toto časové obmedznie neplatí, avšak po prekročení hranice 10 užívateľov sa stanú niektoré funkcie nástroja nedostupnými. Platené programy bez výraznejších obmedzení začínajú na cenovej hladine 23 EUR za užívateľa mesačne (SaaS), resp. 29 EUR za užívateľa mesačne („self-hosted“).

Nástroj obsahuje viacero užitočných nástrojov pre uľahčenie tímovej spolupráce a párové programovanie.

Silnou stránkou tohoto nástroja je existujúca integrácia s často používanými platformami pre vývoj softvéru, akými sú napríklad GitHub, GitLab alebo Bitbucket. Nevýhodou nástroja je vysoká cena za použitie plnej verzie nástroja, či obmedzenie počtu hodín práce s nástrojom v bezplatnej verzii.

1.8.4 Zhrnutie analýzy podobných nástrojov

Technológií s podobným účelom, cieľmi a funkcionalitou, ako plánovaný nástroj existuje hneď niekoľko. Z existencie týchto systémov plynie, že technická realizovateľnosť by pre túto prácu nemala byť limitáciou.

Medzi najčastejšie silné stránky existujúcich nástrojov patrí:

- možnosť využiť riešenie v režime SaaS, čo užívateľa oslobodí od nutnosti spravovať infraštruktúru,
- integrácia s často používaným verzovacím systémom Git,
- možnosť využiť nástroje pre zjednodušenú tímovú spoluprácu.

Naopak, ako najčastejšie slabé stránky som identifikoval nasledovné:

- vysoká cena za nákup licenie pre používanie nástroja,
- v prípade modelu „self-hosted“ je vo všetkých prípadoch nevyhnutnou prerekvizitou inštalácia orchestračného nástroja Kubernetes,

- absencia možností pre distribúciu jednotlivých vývojových prostredí učiteľom pre jednotlivých študentov.

Všetky z uvedených slabých stránok sa pri návrhu prototypu pokúsim eliminovať. Pretože vyvíjaný nástroj bude zverejnený v plnom rozsahu, jeho použitie bude bezplatné. Nástroj nebude vyžadovať pre svoj beh orchestračný nástroj Kubernetes. Nástroj bude obsahovať možnosti pre správu prostredí vyučujúcimi a ich distribúciu študentom. Práve tieto body odlišujú navrhovaný systém od už existujúcich systémov, teda definujú jeho konkurenčnú výhodu.

Kapitola 2

Návrh

V úvode kapitoly sa budem venovať bližšiemu spracovaniu technologických princípov, ktoré sú pre vývoj prototypu relevantné. V neskorších častiach kapitoly sa pokúsim porovnať dostupné technológie, ktoré tieto princípy využívajú a vybrať z nich tie najvhodnejšie pre implementáciu. Na záver sa vrátim k doménovému modelu z predošlej kapitoly - na jeho základe vytvorím diagram tried a relačný diagram pre uloženie dát v relačnej databáze.

2.1 Správa infraštruktúry

V tejto časti popíšem relevantné technologické princípy v kontexte správy infraštruktúry, vďaka ktorým bude prototyp vyvíjanej aplikácie fungovať efektívne z hľadiska spotreby systémových zdrojov. Tými sú najmä trvalá pamäť (pevný disk), vyrovnávacia pamäť (pamäť RAM) a výpočtový výkon (procesor).

2.1.1 Princípy

Pri návrhu aplikácie čelím problému, ako na jednom fyzickom počítači spustiť viacero virtuálnych prostredí. Pre návrh riešenia tohoto problému sú relevantnými najmä princíp virtualizácie a kontajnerizácie.

Virtualizácia

Virtualizácia je technické riešenie umožňujúce spustenie viacerých „logických“ počítačov zároveň na jednom fyzickom počítači [11]. Pojmom **virtuálny počítač** budem označovať jednu takúto bežiacu inštanciu operačného systému. Nad fyzickým hardvérom je spustený **hypervízor**, ktorý riadi prístup jednotlivých virtuálnych počítačov k systémovým zdrojom fyzického počítača.

Oddelenie aplikácií do samostatných virtuálnych strojov prináša mnoho výhod. Snáď najdominantnejšou z nich je zapúzdrenie virtuálneho počítača,

t.j. v prípade akéhokoľvek poškodenia integrity softvéru stačí virtuálny počítač odstrániť a znovu vytvoriť jeho novú inštanciu. Toto je obzvlášť výhodné pre testovanie aplikácii alebo investigáciu škodlivého sofvéru.

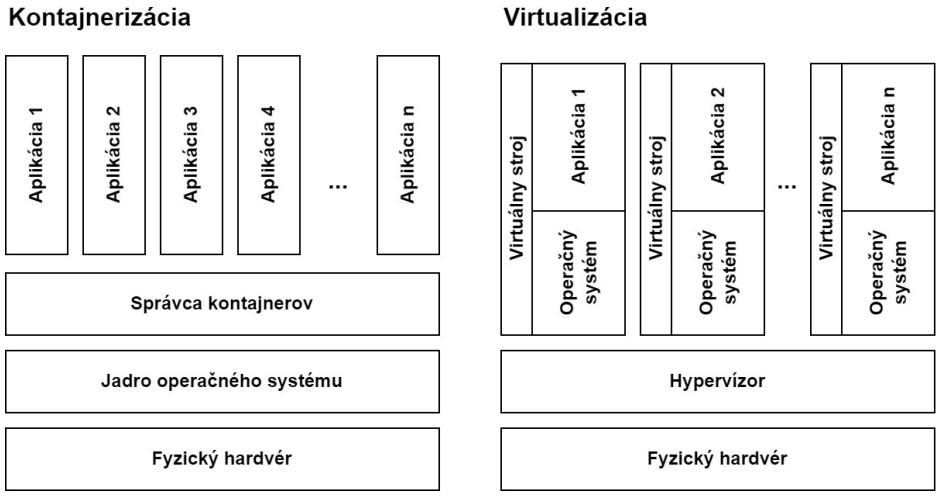
Nevýhodou virtualizácie je nutnosť spustenia samostatného operačného systému pre každú aplikáciu, čo zapríčiní príliš veľkú spotrebu systémových zdrojov na réžiu.

Kontajnerizácia

Kontajnerizácia je technické riešenie umožňujúce spustenie viacerých izolovaných aplikácii na jednom operačnom systéme, ktorý beží na jednom fyzickom počítači. Pod pojmom **kontajner** rozumiem proces bežiaci v (hostiteľskom) operačnom systéme fyzického počítača, ktorý má prístup k izolovanej pamäti. Na jednom fyzickom počítači smie byť spustených viacero kontajnerov, ktoré medzi sebou neinteragujú, avšak zdieľajú spoločný operačný systém. V ňom beží okrem jednotlivých aplikácii ešte jeden ďalší proces, ktorého úlohou je spravovať jednotlivé kontajnery. [12]

Výhodou tohoto prístupu v porovnaní s virtualizáciou je výrazné zníženie nárokov na systémové zdroje, nakoľko si vystačím s jedným operačným systémom pre viacero izolovaných prostredí pre beh aplikácie.

Zhrnutie



Obr. 2.1: Porovnanie virtualizácie a kontajnerizácie [12]

Pre vývoj prototypu aplikácie mi zadanie ukladá využiť princíp kontajnerizácie. Jedná sa o vhodný princíp, pretože vývojové prostredia používané na väčšine softvérovo zameraných predmetov zvyčajne nijak nezasahujú do operačného systému a sú od neho zcela nezávislé. Využitie virtualizácie, ktoré

by priniselo zapúzdrenie jednolivých vývojových prostredí až na úrovni samostatnéhp operačného systému, nepovažujem za prínosné. Práve naopak, réžia s ním súvisiaca by spôsobila zbytočne vysokú spotrebu systémových zdrojov.

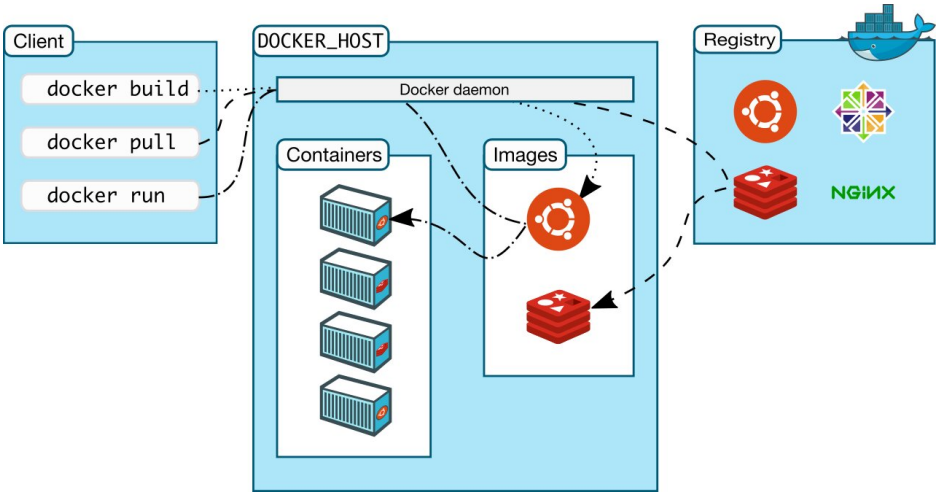
2.1.2 Technológie

Zadanie práce mi ponúka možnosť výberu medzi kontajnerizačnými technológiami Docker a Podman. V tejto časti popíšem tieto bežne používané implementácie kontajnerizácie. Následne sa pokúsim vybrať pre svoj projekt vhodnejšiu alternatívu.

Docker

Docker [13] je otvorený nástroj používaný na automatizáciu vývoja, dodania a nasadenia počítačových aplikácii v kontajneroch, aby mohli aplikácie pracovať efektívne v rôznych prostrediach.

Pre Docker je typická architektúra klient – server. Klient a server (reprezentovaný démonom spusteným na hostiteľovi) spolu komunikujú prostredníctvom REST API, UNIX socketov alebo sieťového rozhania. Hostiteľ môže ďalej interagovať s registrom, ktorý slúži ako primárne ako zdroj obrazov pre zostavenie kontajnerov.



Obr. 2.2: Architektúra nástroja Docker [14]

Dôležitou vlastnosťou Docker obrazov je ich mimoriadne jednoduchá rozšíriteľnosť. Obraz je často len rozšírením nejakého iného obrazu s doplnením želaného prispôsobenia. Táto kľúčová vlastnosť by umožnila vyučujúcim veľmi jednoducho reprezentovať vývojové prostredie prostredníctvom Docker obrazu. Stačí vytvoriť základný obraz, ktorý bude následne rozšíriteľný vytvorením nového obrazu na jeho základe.

Pomocou klienta je následne možné z obrazu zostaviť kontajner, spustiť ho a ukončiť ho. Klient tiež umožňuje reguláciu systémových zdrojov pridelených jednotlivým spusteným kontajnerom, sprístupnenie sieťových portov kontajnera alebo mapovanie častí súborového systému medzi kontajnerom a hostiteľom. Tým sa síce čiastočne poruší zapúzdrenosť kontajnera, avšak prínosom je možnosť vystavenia behu aplikácie sieťovej prevádzke. Dôležité je, že beh aplikácie je od stavu hostiteľského systému nezávislý - všetky závislosti pre beh spravuje kontajner.

Podman

Podman [15] je nástroj pre réžiu kontajnerov, nie však pre ich zostavenie. Za účelom zostavenia sa spolieha na iné nástroje, ako je napr. Buildah alebo už spomínaný Docker.

Jeho špecifikom je možnosť spustenia kontajnerov bez koreňových práv, pod hlavičkou bežného užívateľa systému. Toto vytvára vrstvu zabezpečenia pre prípad, že sa útočníkovi podarí do kontajnera akokoľvek preniknúť.

Porovnanie a výber kontajnerizačnej technológie

Oba nástroje sú kompatibilné s najpoužívanejšími operačnými systémami (Windows, Linux a Mac). Rozdiel medzi uvedenými technológiami vidím primárne v troch oblastiach – v ich architektúre, rozsahu a veľkosti užívateľskej bázy.

Zatiaľ čo Docker pre svoj beh využíva démona, ktorý slúži ako mediátor medzi užívateľom a infraštruktúrou, Podman priamo interaguje s jadrom operačného systému.

Z hľadiska rozsahu pokrýva Docker väčší rozsah úkonov, ako Podman. Docker spravuje kontajnery počas ich celého životného cyklu - poskytuje spôsob pre ich vytvorenie, nástroje pre ich správu a beh v produkčnom prostredí. Docker je komplexným a nezávislým riešením, zatiaľ čo Podman sa sploieha na viacero ďalších špecifických nástrojov.

Docker má veľkú výhodu vo veľkosti užívateľskej báze. Podľa výsledkov [16] z dotazníka jedného z najväčších vývojárskych fór StackOverflow vyhodnotenom v máji 2021 používa Docker 48.85% respondentov. U respondentov, ktorí sa označili ako profesionálni vývojári, dosiahla táto hodnota až 55.06%. Žiadna iná kontajnerizačná technológia sa na dominantných miestach neumiestnila. Z toho plynie, že väčšina profesionálnych vývojárov je na prácu s nástrojom Docker zvyknutá a žiadna iná technológia nemá konkurencieschopný trhový podiel. Z pohľadu jednoduchosti používania aplikácie správcami systému a vyučujúcimi považujem Docker za vhodnejšiu voľbu.

Po zvážení popísaných špecifík oboch nástrojov som sa rozhodol, že pre účely tejto práce bude vhodnejšie použiť nástroj Docker, pričom pri návrhu bude prihliadnuté k jednoduchej nahraditeľnosti za iný nástroj.

2.2 Vývoj softvéru

V nasledujúcej časti predstavím niekoľko užitočných princípov, ktorých sa budem pri následnej implementácii aplikácie držať. Vyberiem vhodné implementácie týchto princípov pre zvolený jazyk PHP, resp. zvolenú kontajnerizačnú technológiu Docker.

2.2.1 Princípy

V skratke popíšem jednotlivé princípy, ktoré budem pri následnej implementácii používať. Zdôvodním, prečo som sa ich rozhodol použiť a aké prípadné výhody mi ich využitie prinesie.

Objektovo orientované programovanie

Objektovo orientované programovanie (OOP) je programátorská paradigma založená na koncepte objektov zložených z dát (atribútov, resp. vlastností obejktu) a kódu (procedúr, resp. metód objektu).

Táto paradigma stojí na štyroch základných pilieroch, ktorými sú [17]:

abstrakcia (angl. *abstraction*) združujúca logické celky kódu do jednej metódy, skrývajúc implementačné detaily,

zapúzdrenie (angl. *encapsulation*) umožňujúce objektom mať a riadiť svoj vlastný stav, napr. pomocou obmedzenia prístupu k neverejným atribútom a metódam objektu,

dedičnosť (angl. *inheritance*) sprístupňujúca vlastnosti a metódy nadobjektu jeho podobjektom,

polymorfizmus (angl. *polymorphism*) umožňujúci odlíšiť správanie podobjektov pomocou preťaženia jeho vlastností a metód.

Nejedná sa o exaktnú definíciu, v literatúre sa často uvádzajú len tri základné piliere miesto štyroch, viď napr. [18].

Relačné databázové systémy

Databáza je súbor záznamov, ktorých systematická štruktúra umožňuje vyhľadávanie v týchto správach pomocou počítača. Databázový systém je zložený zo samotných dát (databázy) a DBMS (angl. database management system), teda systému pre správu dát. [19]

Vzdľadom na nefunkčnú požiadavku N2 bude pre moju prácu relevantný relačný systém správy databázy (RDBMS).

Relačný databázový systém je databázový systém, ktorý implementuje relačný model. Jeho podrobné vysvetlenie je dostupné v [20].

Objektovo-relačné mapovanie

Princíp ORM spravuje zobrazenie objektov (používaných počas behu programu) na záznamy v relačnej databáze (a naopak). Hlavným cieľom tejto vrstvy je automatizácia mapovacieho procesu a odbremenenie programu od technických detailov implementácie takéhoto mapovania. [21]

Základným kameňom ORM je tzv. databázová abstrakčná vrstva (angl. database abstraction layer, DBAL [22]). Táto vrstva zodpovedá za mapovanie objektovej reprezentácie relačnej logiky do jazyka SQL, bez ohľadu na konkrétny typ použitej relačnej databázy a jej špecifiká. V prípade zmeny typu použitej relačnej databázy (napr. zámena PostgreSQL za MySQL) táto vstva automaticky zabezpečí použitie správneho SQL dialektu, čo odstráni nutnosť manuálne prepisovať SQL dotazy v zdrojovom kóde.

Daňou za jednoduchosť pri používaní ORM je oproti manuálnej interakcii s databázou často nižší výkon pri komplexnejších dotazoch. V rámci optimalizácie sa ale vždy naskytá možnosť rozsiahlejšie dotazy implementovať pomocou databázového jazyka SQL a spúšťať ich priamo nad relačnou databázou.

Jednotlivé technológie, ktoré implementujú princíp ORM sa najčastejšie riadia návrhovými vzormi „Active record“, kde objekt reprezentujúci riadok v databáze obsahuje aj logiku pre operácie nad celou tabuľkou (vyhľadávanie, úprava, mazanie) alebo „Data mapper“, ktorý túto logiku presúva mimo dátového objektu. [23]

Autentizácia a autorizácia

Pojem „autorizácia“ označuje proces získania privilégií na výkon akcií a prístup k zdrojom. Naopak, „autentizácia“ označuje proces overenia a identifikácie užívateľa. Po spoľahlivom a dôveryhodnom overení, že so systémom skutočne interaguje daný užívateľ nasleduje vyhodnotenie privilégií, práv a povolených akcií užívateľa, ktorý bol autentizovaný. [24]

V tejto práci budem musieť pre zabezpečenie korektného fungovania aplikácie implementovať oba spomínané procesy. Pre účely autentizácie využijem protokol OAuth, vďaka ktorému môžem zodpovednosť za autentizáciu užívateľa delegovať na príslušné webové služby prevádzkované univerzitou. Výhodou je, že užívateľ nepotrebuje pre prístup do aplikácie samostatné prihlasovacie údaje, pre prístup využíva svoj účet študenta ČVUT.

Autorizačné mechanizmy, ktoré budem implementovať, budú primárne vychádzať z dát získaných z webových služieb, ktoré poskytujú údaje o fakultnom personále a organizácii výuky (napr. KosAPI).

OAuth

OAuth je otvorený protokol umožňujúci aplikáciam tretích strán získať obmedzený prístup do HTTP služby. Pri interakcii s ňou sa tretia strana identi-

fikuje buď ako vlastník zdroja (po schválení vlastníkom) alebo ako aplikácia samotná. Štandard je podrobne popísaný v norme RFC 6749. [25].

V práci budem implementovať oba spomenuté identifikačné mechanizmy. Proces identifikácie ako „vlastník zdroja“ oproti systému Usermap, ktorý autorizáciu pomocou OAuth podporuje, mi poslúži ako vierohodný zdroj pre prihlásenie užívateľa. Naopak, pod hlavičkou aplikácie bude možné pristupovať k službe KosAPI a získať tak dáta o výuke (semestre, predmety, atp.).

2.2.2 Technológie

PHP

Pre implementáciu projektu som sa rozhodol použiť programovací jazyk PHP. Veľkou výhodou PHP oproti iným jazykom, ktorá sa nám vzhľadom na charakter projektu bude hodiť, je jeho nenáročnosť na infraštruktúru – pre spustenie programu v PHP stačí mať nainštalovaný interpret jazyka PHP, nie je potrebné žiadne špeciálne prostredie ani kompilácia zdrojového kódu, keďže sa jedná o jazyk interpretovaný. Zároveň sa jedná o jazyk, ktorý je stále aktívne vyvíjaný a patrí k najpoužívanejším jazykom pre návrh webových aplikácii.

Táto výhoda nie je výrazná pri lokálnom vývojovom prostredí, no v prípade nasadenia do produkcie nám výrazne zníži nároky na infraštruktúru. S PHP dokážu pracovať bežné (najlacnejšie) webové hostingy, zatiaľ čo pre beh programov v iných jazykoch (Java, Python, ...) je potrebné spravovať virtuálny server, kde je možnosť doinštalovania potrebných závislostí.

Ako som už spomínal v analytickej časti, cieľom práce je splniť zadanie s čo najmenšími nárokmi na infraštruktúru a tým vytvoriť konkurenčnú výhodu oproti iným alternatívam vyvíjanej aplikácie. Túto výhodu PHP teda považujem za kľúčovú.

Ďalšie podrobnosti sú dostupné v dokumentácii k jazyku PHP [26].

Symfony

Symfony [27] je open-source webový framework pre jazyk PHP určený na vývoj webových aplikácii rôznej komplexity. Framework je navrhnutý s prístupom objektovo orientovaného programovania a je celosvetovo rozšírený. Framework podporuje vývoj webových aplikácii s architektúrou MVC / MVP.

Alternatívne by bolo možné pre vývoj použiť aj iné frameworky, z ktorých uvediem dva príklady a ich porovnanie so Symfony.

Nette [28] Tento framework som sa napriek jeho zameraniu na bezpečnosť webových aplikácii rozhodol nepoužiť, nakoľko mimo oblasti Česka a Slovenska nie je medzi programátormi rozšírený. Pre personálne zabezpečenie ďalšieho rozšírenia nad rámec tejto práce by to mohlo byť komplikáciou.

Laravel [29] Laravel sa svojím návrhom sústredí na rýchlosť dodania prototypu, často aj na úkor udržateľnosti a rozšíriteľnosti aplikácie, čo robí Symfony vhodnejšou voľbou pre komplexnejšie webové aplikácie.

Doctrine

Doctrine [30] je rozšírením frameworku Symfony určeným na objektovo-relačné mapovanie a správu relačnej databázy, ktoré sa dá využiť aj ako samostatná knižnica. Jedná sa o nástroj priamo odporúčaný autormi frameworku Symfony pre tento účel. Práve toto široké prepojenie so zvoleným frameworkom Symfony je dôvodom, prečo som sa rozhodol použiť Doctrine oproti iným alternatívam.

PostgreSQL

PostgreSQL [31] je jedným z najpokročilejších a najpoužívanejších relačných databázových systémom s otvoreným zdrojovým kódom. Pre účely tejto práce som sa ju rozhodol použiť najmä z dôvodu širokej užívateľskej komunity, čo zjednoduší potenciálny ďalší vývoj aplikácie.

2.3 Architektúra webovej aplikácie

Cieľom tejto časti bude popísať návrh z pohľadu na celú aplikáciu, bez ohľadu na detailný návrh jej jednotlivých ďalších komponent. Dôležitou poznámkou je, že uvedené typy architektúr sa vzájomne nevylučujú. Jedná sa o rôzne uhly pohľadu na separáciu zodpovednosti medzi jednotlivé časti aplikácie.

2.3.1 Dvojvrstvová architektúra

Dvojvrstvová architektúra [32] delí aplikáciu na serverovú (vzdialenú) časť a klientsku (lokálnu) časť a pojednáva o rozdelení zodpovednosti jednotlivých aplikačných komponent medzi tieto dve časti.

thin-client / smart-server Väčšia časť výpočtového výkonu náleží serveru webovej aplikácie, klient slúži predovšetkým ako ovládací panel.

thick-client / dumb-server Aplikačná logika je z väčšej časti alebo úplne presunutá na stranu klienta, server obsahuje len nevyhnutné funkcionality.

Pretože cieľom práce je spúšťať virtuálne prostredia na serveri, túto časť nie je možné presunúť do klientskeho zariadenia. Pri vývoji sa teda budem držať prístupu „thin-client / smart-server“.

2.3.2 Trojvrstvová architektúra

Trojvrstvová architektúra [32] delí aplikáciu podľa jej logických celkov na nasledovné vrstvy:

1. Prezentačná
2. Logická
3. Dátová

Cieľom delenia aplikácie na tieto tri vrstvy je predovšetkým separácia zodpovedností za jednotlivé úkony, ktoré v rámci cyklu spracovania požiadavky prebiehajú. Dôsledkom je zníženie závislostí a zvýšenie udržateľnosti a prehľadnosti zdrojového kódu.

Dôležitým znakom je, že každá vrstva komunikuje len s najbližšou nižšou vrstvou. Ak komunikuje s viacerými nižšími vrstvami, hovoríme o tzv. rozvolnenej trojvrstvovej architektúre.

2.3.3 MVC

MVC (Model–View–Controller) je typ architektúry aplikácie. Rozdeľuje aplikáciu na tri logické časti, ktoré je možné upravovať samostatne. [33]:

Model Reprezentuje dáta a doménovú (business) logiku aplikácie. V kontexte webových aplikácií sa jedná nielen o databázu a súčasti aplikácie pre interakciu s ňou, ale aj o vrstvu, ktoré tieto dáta ďalej spracováva v doménovom kontexte. V súvislosti s trojvrstvovou architektúrou zastupuje dátovú a logickú vrstvu.

View Reprezentuje zobrazenie dát užívateľovi. Nejedná sa len o samotné pixely vykreslené na obrazovke užívateľa – spadá sem aj časť aplikácie zodpovedná za správne vykreslenie tohoto rozhrania na serveri. V súvislosti s trojvrstvovou architektúrou zastupuje prezentačnú vrstvu.

Controller Zodpovedá za tok udalostí v aplikácii a riadenie aplikačnej logiky. Zachytáva užívateľské akcie a podľa nich náležite manipuluje s modelom a zabezpečí aktualizáciu užívateľského rozhrania podľa upraveného stavu. V súvislosti s trojvrstvovou architektúrou zastupuje prezentačnú a logickú vrstvu.

Jedným z dôležitých pozorovaní tejto architektúry je fakt, že model by nemal mať žiadnu závislosť na view, čoho dôsledkom sú samotné aplikačné dáta a doménová logika v dostatočnej miere izolované od prezentačnej vrstvy. Táto izolácia prináša benefit v podobe zlepšenia udržovateľnosti jednotlivých častí aplikácie.

Štandardný spôsob spracovania užívateľskej akcie je pri použití MVC nasledovný [33]:

1. Užívateľ vykoná akciu na užívateľskom rozhraní (view).
2. Controller zachytí vykonanie akcie.
3. Controller rozhodne, ako na akciu reagovať (interaguje s modelom a priamo ovplyvní view).
4. View zobrazí zmeny užívateľovi.

Bohužiaľ, nedostatočná exaktnosť tejto definície prináša v priebehu implementácie mnoho otáznikov. Vytvára priestor pre vznik viacero podtypov MVC architektúry, akými sú napríklad Model – View – Presenter alebo Model – View – ViewModel.

2.4 Modelovanie systému

V nasledujúcej časti práce popíšem proces vzniku jednotlivých diagramov, ktoré budú slúžiť ako primárny podklad pre implementáciu webovej aplikácie v ďalšej časti práce (diagram tried), resp. pre inicializáciu databázy (databázový diagram). Oba budú vychádzať z doménového modelu, ktorý vznikol v analytickej časti tejto práce.

Pretože v práci budem využívať princíp ORM, databázový model a diagram tried musia byť na seba navzájom prevoditeľné. Jednotlivé vzťahy medzi triedami sú implementované cudzím kľúčom (vzťahy typu 1:1 alebo 1:N / N:1) alebo pomocou dekompozičnej tabuľky (vzťahy typu M:N).

2.4.1 Diagram tried

Jednotlivé triedy reflektujú domény popísané v doménovom modeli. Názvoslovie je zachované. Oproti doménovému modelu sú súčasťou diagramu tried konkrétne dátové typy, ktoré sú závislé na programovacom jazyku zvolenom pre implementáciu. Kompletný diagram tried je prílohou tejto práce.

2.4.2 Databázový diagram

Diagram popisuje reprezentáciu jednotlivých tried a vzťahov medzi nimi v databázi. Je odvodený z diagramu tried pomocou dekompozičných pravidiel. Jednotlivé dátové typy jazyka PHP sú nahradené ekvivalentnými dátovými typmi databázy PostgreSQL, ktorá bude pre implementáciu použitá. Kompletný databázový diagram je prílohou tejto práce.

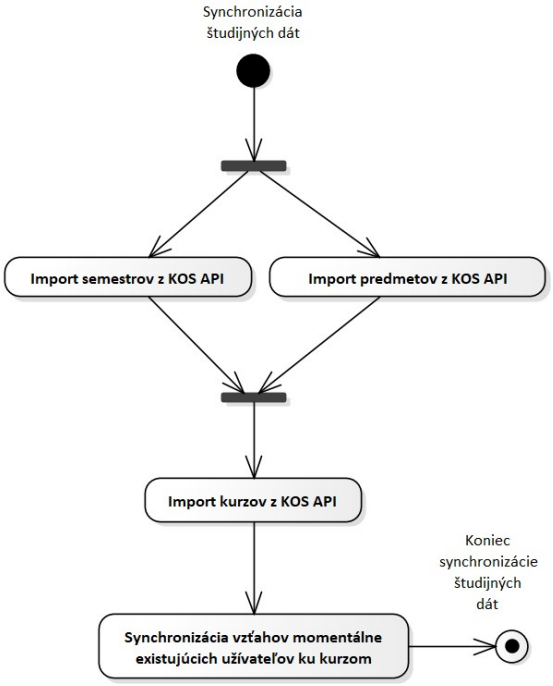
2.5 Návrh procesov

V tejto časti práce navrhnem procesy, ktoré zabezpečia správne fungovanie vyvíjanej aplikácie. V nasledujúcej kapitole budem tieto navrhnuté procesy implementovať pomocou zvolených technológií.

2.5.1 Správa študijných dát

Jedná sa o automatické vytváranie inštancií tried Term, Subject, Course a CourseMember podľa aktuálneho stavu zodpovedajúcich zdrojov webovej služby KOS API. Tento proces (viď diagram 2.3) je nevyhnutný pre možnosť priradenia vytvoreného vývojového prostredia k predmetu, čím sa určí viditeľnosť prostredia jednotlivým študentom.

Synchronizáciu je potrebné spúšťať na pozadí (napr. pomocou známeho nástroja cron) v pravidelných intervaloch (napr. 1x denne), aby systém vždy odrážal reálny stav študijných dát. Pretože nasadenie projektu do produkčnej infraštuktúry nie je súčasťou zadania práce, ponechávam spôsob implementácie spúšťania tohoto procesu otvorený. Cieľom z hľadiska implementácie bude mať pripravený jeden bash skript, ktorý bude zastrešovať celý tento proces. Proces za žiadnych okolností nemaže inštancie akejkoľvek triedy, ktorá v KOS API nebola dohľadaná.



Obr. 2.3: Diagram aktivít: Synchronizácia študijných dát

2.5.2 Správa užívateľov

Systém obsahuje triedu „User“, ktorá reprezentuje užívateľa aplikácie. Inštancia tejto triedy je vytvorená v momente prvého prihlásenia daného užívateľa.

Pretože aplikácia môže využívať fakultou prevádzkovaný autentizačný a au-

torizačný server Zuul, nie je vo fáze prototypu potrebné v aplikácií implementovať iné autentizačné schémy.

Po úspešnej autentizácií si aplikácia uloží token získaný od autorizačného serveru a všetky ďalšie dotazy na služby KOS API a Usermap API môže vykonávať na základe tohoto tokenu.

V prípade, že token expiruje, aplikácia sa pokúsi token obnoviť. Ak sa automatické obnovenie nepodarí, vyzve užívateľa k opätovnému prihláseniu.

Pri prvom spustení nie je správcom systému žiaden užívateľ. Užívateľ, ktorý systém spustil, si rolu „Správa systému“ musí pridať manuálne na databázovej úrovni editáciou tabuľky „user“. Tento užívateľ následne môže menovať alebo odvolávať ďalších správcov systému.

2.5.3 Správa obrazov

Ak je užívateľ správcom systému, má právo pristupovať k jednotlivým obrazom a pridať nový obraz. Primárnou vlastnosťou obrazu je „Dockerfile“, teda súbor, ktorý nástroju Docker popíše spôsob na zostavenie požadovaného obrazu.

Správca systému môže obraz editovať. Akonáhle je obraz pripravený na použitie, môže správca obraz zostaviť. Výsledkom tejto operácie je nová inštancia entity „EnvironmentImageBuild“. Pri prijatí požiadavky na zostavenie aplikácia predá pokyn na zostavenie obrazu nástroju Docker pomocou definovaného obrazu.

Zostavenie obrazu kopíruje Dockerfile z obrazu v momente zostavenia. V prípade, že by došlo k úprave obrazu, táto zmena neovplyvní už vytvorené zostavenia tohoto obrazu.

Ak je zostavenie obrazu úspešné, učitelia budú môcť tento obraz používať pre vytvorenie definícií vývojových prostredí („EnvironmentFactory“). V opačnom prípade systém ohlási chybu a správca systému môže definíciu obrazu upraviť, resp. pokúsiť sa obraz zostaviť opäť po úprave.

2.5.4 Správa vývojových prostredí

Každý užívateľ, ktorý je učiteľom aspoň jedného predmetu, má právo vytvoriť prostredie pre ľubovoľný predmet, ktorý vyučuje. Pre vytvorenie prostredia je potrebné zvoliť zostavenie obrazu (vytvorené správcom systému) a dodefinovať prípadné špecifiká tohoto prostredia pomocou bash skriptu, ktorý sa zavolá pri každom spustení, resp. vypnutí prostredia.

Po zadaní potrebných údajov do systému môže učiteľ prostredie vytvoriť. Následne má možnosť spustiť si testovacie prostredie alebo sprístupniť možnosť spustenia prostredia študentom predmetu, ku ktorému sa toto prostredie viaže.

2.5.5 Spustenie vývojového prostredia

Každý užívateľ vidí vo svojom profile výpis prostredí, ktoré sa viažu na také kurzy, do ktorých je tento užívateľ zapísaný – či už ako študent alebo vyučujúci. Ak má užívateľ v kurze študentskú rolu, prostredie musí byť pre viditeľnosť študentmi označené ako zverejnené študentom.

Užívateľ má možnosť toto prostredie spustiť. Aplikácia následne:

1. skontroluje, či toto prostredie pre daného užívateľa už nie je v danom čase spustené,
2. vytvorí súborové adresáre, do ktorých budú ukladané dáta vytvorené študentom, resp. z ktorých sa načítajú pri ďalšom spustení aplikácie v tomto kroku,
3. alokuje webové porty potrebné pre beh tohoto prostredia,
4. vytvorí nový Docker kontajner pre beh prostredia,
5. spustí v prostredí kontajnera inicializačný skript definovaný pre toto prostredie,
6. vykoná všetky ďalšie potrebné kroky, aby spustila prostredie pre daného študenta.

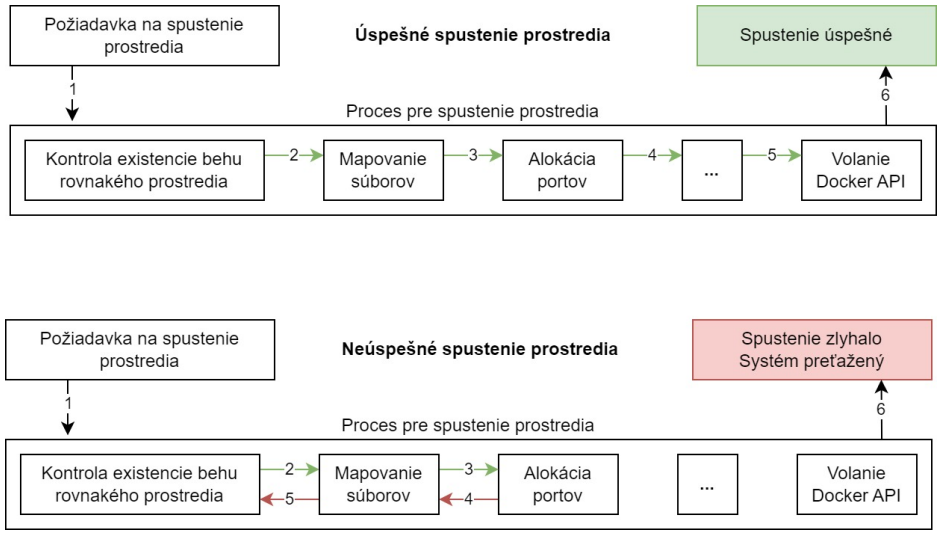
V prípade, že spustenie zlyhá, aplikácia ohlási chybu. V prípade úspešného spustenia vidí študent webovú adresu, na ktorej môže k spustenému prostrediu pristúpiť.

V neskoršej časti práce bude potrebné implementovať mechanizmus, pomocou ktorého bude vyššie uvedený zoznam úkonov reprezentovaný v zdrojovom kóde. Každý jeden z krokov totiž môže potenciálne zlyhať, pričom na zlyhanie musia vedieť všetky predošlé kroky zareagovať vykonaním operácie, ktorá vráti systém do konzistentného stavu (napr. nezostanú alokované porty pre beh prostredia, ktorý nikdy nebežal, pretože zlyhal jeden z krokov spustenia). Návrh tohoto mechanizmu ilustruje obrázok 2.4.

2.5.6 Ukončenie vývojového prostredia

Študent má možnosť po ukončení práce beh prostredia ukončiť. Ak tak neučiní, aplikácia by mala prostredie ukončiť automaticky po troch hodinách behu. Pre zlepšenie užívateľského zážitku by bolo možné doimplementovať logiku, ktorá by prostredia vypínala na základe reálnej aktivity, resp. neaktivity vo vývojovom prostredí. Do prototypu aplikácie som sa však rozhodol túto súčasť neimplementovať.

Ukončenie prostredia by malo využívať analogický mechanizmus ako jeho spustenie - skladá sa z viacerých krokov, pričom každý z nich môže zlyhať a predošlé kroky musia mať možnosť na zlyhanie zareagovať.



Obr. 2.4: Príklad behov procesu: Spustenie vývojového prostredia

Kapitola 3

Implementácia

V tejto kapitole práce popíšem štruktúru zdrojového kódu aplikácie. V zvolenom implementačnom jazyku PHP (resp. s využitím frameworku Symfony) budem reprezentovať jednotlivé entity a služby z návrhovej časti práce. V súlade s predošlou kapitolou implementujem autentifikáciu užívateľa prostredníctvom faktultného SSO. Na záver do aplikácie zapracujem autorizačné mechanizmy, ktoré zabezpečia, že užívatelia budú môcť v rámci aplikácie vykonávať len tie akcie, na ktoré sú oprávnení.

Táto kapitola obsahuje len popis implementácie. Ukážky práce s vyvinutým prototypom sú dostupné v samostatnej kapitole.

3.1 Nastavenie vývojového prostredia

Princíp kontajnerizácie využijem aj na vytvorenie vývojového prostredia pre navrhovaný prototyp aplikácie. Pomocou nástroja *Docker Compose* zadefinujem zoznam služieb, ktoré budú potrebné pre spustenie aplikácie v lokálnom prostredí. Konkrétne sa jedná o nasledovné služby:

webový server určený na priebežné spracovávanie prichádzajúcich webových dotazov a ich smerovanie aplikácii na spracovanie,

aplikačný kontajner s interpretom jazyka PHP, v ktorom bude prebiehať vyhodnocovanie jednotlivých webových dotazov,

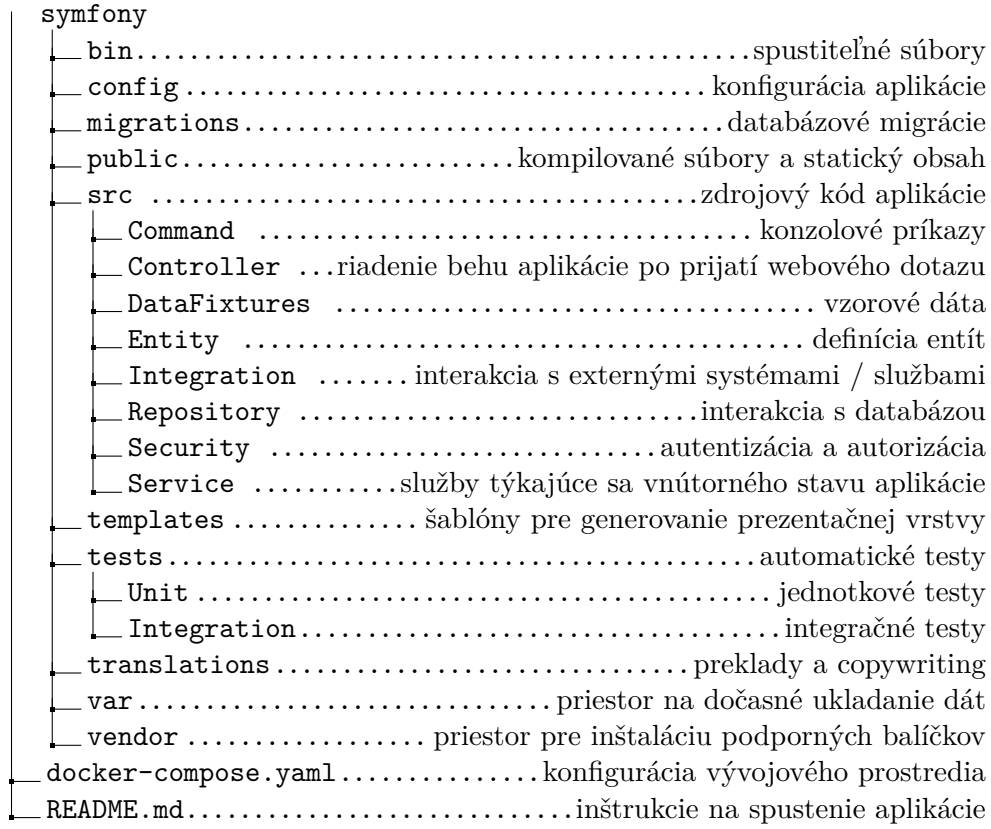
relačnú databázu ako perzistentné dátové úložisko pre aplikáciu,

databázové administračné rozhranie uľahčujúce investigáciu a riešenie prípadných problémov na databázovej úrovni. Pre korektný beh aplikácie však nie je nevyhnutné.

Po vytvorení konfiguračného súboru *docker-compose.yaml* a spustení prostredia pomocou volania *docker compose up* budú do počítača stiahnuté použité

obrazy z registra *Docker Hub*, vytvoria sa z nich kontajnery a vývojové prostredie je pripravné na použitie.

Ďalším krokom je vytvorenie štruktúry samotnej aplikácie. Pre ukážku je k dispozícií diagram 3.1 popisujúci bežné členenie kódu pri vývoji aplikácii vo frameworku Symfony s pomocou služby Docker. Toto členenie sa budem snažiť v priebehu implementácie dodržiavať. Podobnú štruktúru je možné vygenerovať napr. prostredníctvom *Symfony CLI*.



Obr. 3.1: Bežné členenie zdrojového kódu aplikácie v Symfony

3.2 Modul EasyAdmin

Modul EasyAdmin umožňuje vytvorenie administračného rozhrania automaticky z definície jednotlivých entít – pre každú entitu je vytvorený samostatný controller, ktorý je rozšírením triedy AbstractDashboardController z balíčka EasyAdmin. Konkrétne controllery následne pripomínajú skôr konfiguračný súbor, v ktorom sú definované primárne nasledovné:

- verejný názov danej entity, verejné názvy jednotlivých jej atribútov, atp.,

- mapovanie jednotlivých vstupných polí na atribúty danej entity, ku ktorej sa controller viaže,
- logika zobrazovania vstupných polí podľa typu stránky,
- logika zobrazovania riadiacich prvkov pre výkon bežných akcii, akými sú najmä výpis aktuálne existujúcich inštancii, vytvorenie novej inštancie, úprava existujúcej inštancie, zobrazenie detailu inštancie, zmazanie existujúcej inštancie,
- logika zobrazovania riadiacich prvkov pre výkon vlastných akcii (napr. spustenie vývojového prostredia) a ich logika, resp. volanie príslušnej služby,
- logika prístupových práv užívateľa naviazaná k jednotlivým akciám.

Vďaka tomuto modulu nie je potrebné implementovať vlastné užívateľské rozhranie a zaoberať sa jeho následným štýlovaním – to všetko je v réžii modulu EasyAdmin.

Rovnako tak nie je potrebné implementovať logiku vyššie uvedených rutinných akcií - stačí k nim nastaviť príslušné úrovne prístupových práv, aby bola zaručená autorizácia užívateľa na všetky akcie, ktorých výkon je mu v aplikácií umožnený.

V rámci implementácie prototypu som vytvoril takýto controller pre všetky z implementovaných entít.

3.3 Zabezpečenie

Zabezpečenie aplikácie je možné rozdeliť na dve časti - autentizáciu, ktorá sa stará o overenie identity užívateľa, ktorý sa k aplikácií pripája a autorizáciu, ktorá zabezpečuje, že autentizovaný užívateľ smie v aplikácii vykonávať len také akcie, na ktoré je oprávnený.

3.3.1 Autentizácia

V súlade s návrhovou časťou práce som do prototypu aplikácie implementoval overenie užívateľovej identity pomocou fakultného autorizačného servera Zuul OAAS, ktorý funguje na báze protokolu OAuth. Pre implementáciu som využil možnosť rozšíriť triedu AbstractAuthenticator z frameworku Symfony. Implementoval som triedu ZuulOaasAuthenticator, ktorej metóda „authenticate“ dostáva na vstupe jednotlivé dotazy prichádzajúce na server a páruje ich s identitou užívateľa, t.j. vytvorí inštanciu triedy „Passport“, na základe čoho framework Symfony považuje užívateľa za autorizovaného.

Táto logika pre svoje fungovanie vyžaduje možnosť čítania a zápisu do relačných (session) premenných.

V prípade, že systém potrebuje pristupovať k fakultým službám (KOS API, Usermap API, ...) pod vlastnou hlavičkou, t.j. nie pod hlavičkou konkrétneho užívateľa, využije autorizáciu pomocou servisného účtu v súlade s OAuth štandardom.

3.3.2 Autorizácia

Prototyp aplikácie využíva dva kľúčové prístupy k autorizácií užívateľa:

1. autorizáciu pomocou užívateľských rolí - rola správcu systému vs. bežný užívateľ,

2. autorizáciu pomocou tried typu Voter.

Voter je trieda, ktorá rozhoduje o oprávneniach užívateľa v prípade, že sa akcia viaže na konkrétnu inštanciu danej entity. Napr. užívateľ smie zmazať len také prostredie, v ktorom je nastavený ako správca.

V rámci implementácie prototypu som pre každú entitu implementoval samostatnú Voter triedu, ktorá obsahuje:

- definície jednotlivých práv, ktorých získanie je nutné pre výkon akcii nad entitou. Napr. akcia „ENVIRONMENT_FACTORY_RUN“ oprávňuje užívateľa spustiť konkrétne vývojové prostredie.

- logiku pre udelenie alebo neudelenie daného práva danému užívateľovi s väzbou na predmet akcie (napr. užívateľ U nesmie zmazať prostredie P, pretože nie je jeho správcom).

V jednotlivých controlleroch sú následne naviazané akcie na jednotlivé práva, ktoré pre výkon danej akcie nad konkrétnou inštanciou musia byť prostredníctvom Voter tried získané.

3.4 Správa fakultných dát

Správca systému má možnosť spustiť synchronizáciu dát z KOS API, t.j. dát o aktuálne dostupných predmetoch, semestroch a kurzoch (t.j. behoch daného predmetu v rámci daného semestra). V produkčnom prostredí by táto synchronizácia mala prebiehať pravidelne automaticky, aspoň 1x za deň.

Nad rámec tejto synchronizácie sa po každom prihlásení sa pre daného užívateľa synchronizuje zoznam predmetov, ktoré študuje.

Pretože tieto dáta sú v aplikácii potrebné len na čítanie, implementované sú len stránky pre výpis týchto dát. Akékoľvek úpravy alebo mazanie týchto dát užívateľmi nie sú prípustné, keďže náš systém nemá za cieľ nijak zasahovať do životného cyklu týchto dát.

3.5 Registrácia a prihlasovanie užívateľov

Užívateľ sa do aplikácie nemusí žiadnym spôsobom registrovať. Jeho užívateľský profil sa vytvorí automaticky pri prvom prihlásení do aplikácie.

Autorizačná služba Zuul OAAS poskytuje užívateľské meno a email úspešne autorizovaného užívateľa. Meno a priezvisko užívateľa sú dohľadávané s využitím Usermap API podľa užívateľského mena (naprieč ČVUT je využívané ako unikátny identifikátor osoby).

Implementácia vytvorenia užívateľa v prípade jeho neexistecie si vyžiadala rozdelenie logiky prihlasovania do viacerých controllerov:

LoginFormController zodpovedá za vykreslenie domovskej stránky s tlačidlom, ktoré užívateľa po kliknutí presmeruje na autorizačný server,

ZuulOaasController zodpovedá za presmerovanie užívateľa na autorizačný server, a následné spracovanie prihlásenia užívateľa po návrate z autorizačného serveru. Tento dotaz však ešte nie je odchytený službou ZuulOaasAuthenticator, ktorá k dotazom priraďuje identitu užívateľa. Ak je potrebné vykonať akcie závislé od toho, či je užívateľ prihlásený, resp. aký užívateľ je prihlásený, je potrebné užívateľa presmerovať,

SyncCourseMembersController zodpovedá za synchronizáciu vťahu užívateľa k predmetom s KOS API po každom prihlásení.

3.6 Správa obrazov pre vývojové prostredia

V analytickej časti práce som v sekcii 1.8.1 popisoval riešenie „code-server“, ktoré je dostupné aj vo forme obrazu pre Docker. V kontajneri definovanom takýmto obrazom beží samostatná webová aplikácia – IDE Visual Studio Code. Jedná sa o integrované vývojové prostredie, pomocou ktorého je možné pristupovať k súborovému systému, vytvárať a upravovať súbory, prípadne spustiť terminál vnútri takéhoto kontajnera.

Izolovanosť kontajnera od systému, ktorý kontajnery spravuje by mala zabezpečiť elimináciu bezpečnostných rizík. Užívateľovi je navyše v aplikácii sprístupnený terminál pod hlavičkou neprivilegovaného užívateľa „abc“, Vďaka tomu by nemalo byť možné z pozície študenta útočiť na infraštruktúru aplikácie.

Vývojové prostredie VS Code je navyše známe a do veľkej miery používané, preto verím, že jeho použitie umožní jednoduchšiu adopciu aplikácie užívateľmi.

Nižšie uvádzam ukážku konfiguračného súboru Dockerfile, ktorý rozširuje obraz „linuxserver/code-server“ verzie 4.0.2, viď zdrojový kód 3.2:

1. doinštalovaním kompilátora „gcc“ pre programy v jazyku C,
2. definíciou adresára, ktorý sa v IDE VS Code má zobrazovať ako domovský,

3. definíciou zdieľaného úložiska dát medzi hostiteľským súborovým systémom a kontajnerom

4. vystavením portu kontajnera č. 8443 pre spojenia zo siete.

```
FROM linuxserver/code-server:4.0.2
RUN apt-get update && \
    apt-get -y install gcc
ENV DEFAULT_WORKSPACE=/home/student/workspace
VOLUME /home/student/workspace
EXPOSE 8443
```

Obr. 3.2: Konfiguračný súbor Dockerfile pre vzor obrazu prostredí

Dôležitou poznámkou je, že za zabezpečenie aplikácie na úrovni transportnej vrstvy aktuálne zodpovedá výhradne generované prostredie - aplikačný server cestu k nemu nijak hlbšie nezabezpečuje. Implementáciu prípadného zabezpečenia prostredí na transportnej vrstve považujem za dôležitý bod budúceho vývoja aplikácie, no v rámci fázy implementácie prototypu ju vynechávam.

3.7 Spustenie a vypnutie vývojového prostredia

Z návrhovej časti práce plynie, že spustenie prostredia by malo byť atomické, a to aj napriek veľkému množsvu operácii, ktoré je potrebné vykonať. Spustenie sa buď podarí, alebo zlyhá, pričom v oboch prípadoch zostáva systém v konzistentnom stave. Pre tvorbu takéhoto mechanizmu sa inšpirujem návhrovým vzorom Chain of Responsibility.

Jednotlivé kroky spustenia majú definovaný nielen spôsob, ako ich vykonať, ale aj spôsob, ako ich výkon vrátiť naspäť v prípade zlyhania niektorého z neskorších krokov - metóda „unhandle“.

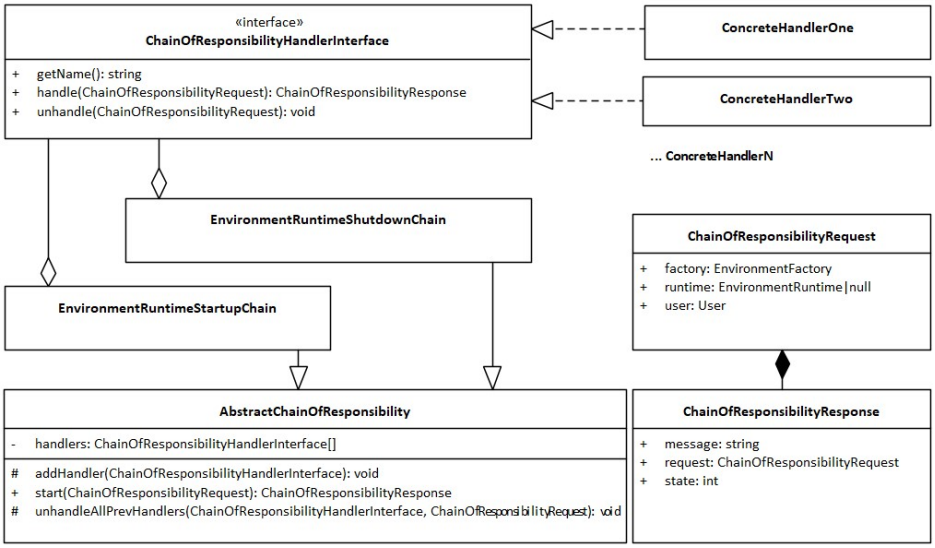
Trieda „AbstractChainOfResponsibility“ zaisťuje réžiu - umožňuje pridať krok potrebný k jej spusteniu a spustiť ju. V prípade, že počas jej behu nastane chyba, zabezpečí, že všetky kroky, ktoré už boli úspešne vykonané, vykonajú návrat do pôvodného stavu.

Jej konkrétni potomkovia „EnvironmentRuntimeStartupChain“ a „EnvironmentRuntimeShutdownChain“ následne už len definujú, aké kroky je potrebné vykonať pre spustenie alebo vypnutie prostredia.

Jednotlivé tiedy implementujúce „ChainOfResponsibilityHandlerInterface“ definujú operácie potrebné k úspešnému výkonu jednotlivých krokov, resp. ich anuláciu v prípade zlyhania. Vďaka rozdeleniu komplexného procesu do jed-

notlivých krokov má každá takáto trieda len jednu zodpovednosť, čo zlepšuje čitateľnosť a prehľadnosť zdrojového kódu.

Vypnutie vývojového prostredia funguje analogicky.



Obr. 3.3: Diagram tried: Spustenie a vypnutie prostredia

3.8 Integrácie

V tejto sekcii popíšem implementačné detaily jednotlivých integrácii s externými službami potrebnými pre chod aplikácie. Napriek tomu, že sa nejedná o jadro aplikačnej logiky, práca s nimi zohrala v implementačnej fáze práce významnú úlohu. Vďaka nim dokáže aplikácia využívať externé webové služby.

Ak to bolo možné, pre integráciu s externými systémami som sa snažil využiť už existujúce PHP balíčky. Ak ich rozsah nebol dostatočný, požadovanú funkcionalitu som doimplementoval.

3.8.1 Docker API

Rozhranie „DockerEngineInterface“ obsahuje zoznam metód, ktoré prototyp aplikácie vyžaduje pre správu prostredí. V teoretickej rovine je možné vytvoriť novú implementáciu tejto triedy, ktorá by v prípade potreby umožnila použiť aj iný kontajnerizačný nástroj ako Docker.

V rámci implementácie prototypu som navrhol jednu implementáciu tohoto rozhrania „DockerEngineSocketApi“, ktoré dokáže komunikovať s Docker Engine pomocou UNIX socketu. Z bezpečnostného hľadiska toto nie je vhodné dlhodobé riešenie, nakoľko vystavenie tohoto socketu na hostiteľskom

hardvéri môže prinášať rôzne bezpečnostné riziká, resp. nemusí byť podporované. Pre overenie funkčnosti prototypu na lokálnej inštancií však poslúži dostatočne.

Veľmi kvalitný základ pre implementáciu poskytol už existujúci balíček „vdauchy/docker-php-api“ [34]. Doimplementovať bolo potrebné len podporu pre odosielanie dotazov prostredníctvom UNIX socketu.

3.8.2 Kos API

Návrh integrácie s Kos API je inšpirovaná návrhovým vzorom „Command“ [35]. Jedná sa primárne o triedu „KosApiIntegration“, ktorá má len jednu metódu „get“ na vyhodnotenie požadovaného príkazu. Jednotlivé príkazy reprezentujúce volania na endpointy Kos API sú implementované ako samostatní potomkovia triedy „AbstractKosApiCommand“. Z výsledku volania zároveň dokážu v prípade úspechu vrátiť predspracovanú štruktúrovanú odpoveď reprezentovanú objektom (viď potomkov triedy „AbstractKosApiDto“).

3.8.3 Usermap API

Jedná sa o jednoduchú fasádu nad volaním zdroja „people“ z Usermap API, ktoré v aplikácií slúži na dohľadanie krstného mena a priezviska podľa užívateľského mena. V aktuálnom stave služba nie je využívaná pre realizáciu iných prípadov použitia.

Pre účely autorizácie dotazu na Usermap API je využitý princíp OAuth, viď nižšie.

3.8.4 OAuth

Jedná sa o rozšírenie implementácie OAuth klienta „league/oauth2-client“ [36] implementovaného v jazyku PHP za účelom kompatibility s fakultným autorizačným serverom a následnú reprezentáciu prístupových údajov v relačných (session) premenných.

Rozšírenie tejto knižnice je využité v triede „ZuulOaasAuthenticator“, ktorej úlohou je rozhodovať o tom, či jednotlivé dotazy, ktoré na aplikáciu prichádzajú, sú autorizované. V prípade, že je potrebné užívateľa presmerovať na autorizačnú službu, presmerovanie a návrat prebieha podľa logiky definovajnej triedou „ZuulOaasController“. Tá následne využíva rozhranie spomínanej knižnice za účelom uloženia získaných prístupových kľúčov.

Kapitola 4

Testovanie

V tejto kapitole sa budem venovať testovaniu navrhnutého prototypu webovej aplikácie. V úvode kapitoly popíšem bežne používané metodiky testovania aplikácii. V ďalšom kroku z nich zvolím tie metodiky, ktoré v kontexte tohoto projektu dáva zmysel realizovať. Následne sa pokúsim pomocou vhodných technológii tieto testovacie metodiky realizovať. V závere kapitoly popíšem výsledky testovania a prípadné zistenia, ktoré boli pri testovaní odhalené.

4.1 Metodiky testovania

V tejto časti popíšem bežne používané metodiky testovania aplikácii. Uvediem, pre ktoré časti aplikácie je ich použitie vhodné, prípadne ich ďalšie špecifické vlastnosti.

4.1.1 Automatické testovanie

Automatické testy sú zvyčajne súčasťou zdrojového kódu projektu, ktorý testujú. Ich úspešný beh a ukončenie je často používaným kontrolným mechanizmom, či už pre overenie všetkých požadovaných funkcionalít aplikácie pred vytvorením novej verzie, alebo pre nasadenie aplikácie na webový server.

Jednotkové testy

Cieľom jednotkových testov (angl. *Unit tests*) je overiť, že jednotlivé komponenty (jednotky) aplikácie, ktoré sú výstupom z implementačnej fázy projektu, fungujú korektne s ohľadom na implementáciu. Za jednotku považujem akúkoľvek pomenovanú sadu príkazov, ktorá je inými časťami zdrojového kódu volaná [37]. Jednotkové testy naopak netestujú vedľajšie efekty, ktoré môžu testovanými akciami nastať (napr. vytvorenie súboru).

Častým problémom pri realizácii jednotkových testov sú vyžadované závislosti pre vytvotenie novej inštancie testovanej triedy. Riešením tohoto proble-

mu je technika nazývaná **mocking**. Vďaka nej pred vytvorením novej inštancie testovanej triedy nie je potrebné inštancovať všetky ďalšie triedy, na ktorých je testovaná trieda závislá. Namiesto týchto inštancii sú do konštruktoru testovanej triedy dosadené "mocky". Tieto objekty majú totožné vonkajšie rozhranie, ako objekty, ktoré "zastupujú", avšak nevykonávajú kód volaných metód - iba overujú, či boli metódy volané so správnymi argumentmi, prípadne či počet volaní metód testovanou jednotkou zodpovedá želanému počtu volaní.

Integračné testy

Cieľom integračného testovania je posúdiť, či rozhrania medzi jednotlivými modulmi aplikácie komunikujú správne. Integračné testovanie musí predpokladať, že moduly fungujú korektne. [37]

V kontexte tejto práce považujem integračné testy za vhodnú metodiku pre overenie korektnosti vedľajších efektov testovaných akcii. Tými môže byť napr. vytvorenie súboru s validným obsahom, volanie konkrétnej webovej metódy alebo uloženie dát do databázy.

Systémové testy

Systémové testy overujú, či systém ako celok spĺňa svoje špecifikácie. Predpokladajú, že jednotlivé súčasti systému fungujú korektne a testujú, či systém funguje ako celok. [37]

Systémové testy by mali byť na rozdiel od integračných testov úplne nezávislé od zdrojového kódu aplikácie. Príkladom využitia integračného testu môže byť test načítania, vyplnenia a odoslania webového formulára, resp. jeho validácie v prípade zadania nesprávnych vstupov.

4.1.2 Užívateľské testovanie

Užívateľské testovanie je vhodné na analýzu miery použiteľnosti systému, predovšetkým však jeho užívateľského rozhrania. Pre vyhodnotenie užívateľského testu je pre svoju jednoduchosť často používaná metóda SUS (*z angl. System usability scale*) dotazníku. V prvej fáze testu užívateľ dostáva úlohy, ktoré má v aplikácii splniť. V druhej fáze vyplní dotazník s 8 otázkami cieliacich na mieru použiteľnosti aplikácie, na škále od 1 do 5. Skóre je následne naškálované na rozsah 1-100, pričom hranica skóre 68 sa považuje za priemer - väčšie skóre je skôr dobrým výsledkom, menšie skóre pondetom k zlepšeniu. [38]

4.2 Výber vhodných testovacích metodík

Pre túto prácu som vybral dve kľúčové metodiky testov, ktoré by mali zodpovedať štádiu prototypu:

1. pokrytie kľúčových častí aplikácie automatickými testmi, primárne jednotkovými a doplnkovo integračnými,
2. testovanie s užívateľom a jeho vyhodnotenie metódou SUS dotazníku.

Systémové testy by v kontexte navrhnutého prototypu nemali veľký prínos oproti už realizovaným jednotkovým testom - užívateľské rozhranie je totiž automaticky generované prostredníctvom komponenty EasyAdmin, ktorá je súčasťou webového frameworku Symfony. Pretože do vytvoreného užívateľského rozhrania nie je nijak ďalej zasahované, priestor pre vznik chyby nepovažujem za významný.

Nakoľko sa v tejto práci nezameriavam na tvorbu užívateľského rozhrania, je vysoko pravdepodobné, že užívateľské rozhranie aplikácie nebude optimalizované pre jednotlivých aktérov (užívateľov) systému. Z toho je v tejto fáze vývoja aplikácie prínos testovania s užívateľom otázny. Napriek pochybe som ho však zvolil, pretože ako jediný pokryje ukážku samotného generovaného vývojového prostredia.

4.3 Implementácia vybraných testovacích metodík

V rámci testovacej fázy práce som pokryl relevantné časti aplikácie jednotkovými a integračnými automatickými testmi. Vytvárané prostredia som podrobil užívateľskému testovaniu metódou SUS dotazníku.

Iné testovacie metodiky som neimplementoval, nakoľko ich najväčší prínos vidím až v neskorších životných fázach tohoto projektu, prípadne v nadväznosti na túto prácu.

4.3.1 Automatické testy

Pre implementáciu automatických testov som zvolil testovací framework PHPUnit.

Pri triedach typu Repository nepovažujem pokrytie testmi za dôležité, nakoľko sa jedná o automaticky generovaný kód prostredníctvom použitého frameworku Symfony, bez ďalšieho zásahu. Vznik chýb teda považujem za málo pravdepodobný.

Rovnako som testmi nepokrýval triedy typu Controller, keďže z veľkej časti vďaka použitiu modulu EasyAdmin obsahujú skôr konfiguráciu, než logiku. Logika aplikácie je prítomná skôr v službách - ktorých kľúčové časti testmi pokryté sú.

Ďalej som považoval za dôležité pre bezpečnosť aplikácie implementovať jednotkové testovanie tried typu Voter, ktoré obsahujú autorizačnú logiku.

Fragmenty zdrojového kódu, ktoré generujú vedľajšie efekty, akými je predovšetkým zápis do súboru, sú pokryté integračnými testmi.

Inštrukcie pre spustenie automatických testov sú súčasťou návodu na spustenie aplikácie.

4.3.2 Užívateľské testovanie

Pre otestovanie generovaných prostredí som zvolil metodiku užívateľského testovania. Pre účely testovania som oslovil päť študentov, resp. absolventov bakalárskeho štúdia na FIT ČVUT, vďaka čomu patria do cieľovej skupiny užívateľov aplikácie.

Počas testu mali užívatelia za úlohu používať prostredie, ktoré im zverejnil ich učiteľ predmetu, ktorý študujú. V prostredí implementovali ukážkovú domácu úlohu z predmetu BI-PA1. Jednalo sa o jednoduchý program, ktorý má na vstupe celé číslo od 1 do 5 a vypíše preň príslušný citát. Program musel detekovať správnosť vstupu. Po implementácii programu bolo úlohou program vo vývojovom prostredí skompilovať a otestovať aspoň jedným jednotkovým testom s využitím príkazového riadku.

Po splnení týchto úloh vypĺňali respondenti (ďalej označení ako R1-R5) nasledovný dotazník. Na každú otázku odpovedali na škále od 1 (rozhodne nesúhlasím) do 5 (rozhodne súhlasím).

Číslo	**Otázka**
1	Rád by som systém používal opakovane.
2	Systém je zbytočne zložitý.
3	Systém sa jednoducho používa.
4	Potreboval by som pomoc človeka z technickej podpory, aby som systém mohol používať.
5	Rôzne funkcie systému sú doň dobre začlenené.
6	Systém je príliš nekonzistentný.
7	Myslím si, že väčšina používateľov sa so systémom naučí pracovať rýchlo.
8	Systém je príliš neohrabaný.
9	Pri práci so systémom sa cítim isto.
10	Musel som sa veľa naučiť, než som so systémom dokázal pracovať.

Tabuľka 4.1: Znenie SUS dotazníka

Z odpovedí v dotazníku bolo následne spočítané SUS skóre:

1. Pre otázky s nepárnym indexom bolo skóre každej otázky určené vzorcom odpoveď - 1,

2. Pre otázky s párnym indexom bolo skóre každej otázky určené vzorcom 5 - odpoveď,

3. Súčet skóre jednotlivých otázok bol prenásobený konštantou 2.5 (naškálované na interval 0-100).

Ako som už spomenul vyššie, hranica skóre 68 sa považuje za priemerný

Otázka	R1	R2	R3	R4	R5	Priemer
1	2	4	5	4	3	3.6
2	5	1	1	1	2	2
3	1	5	4	3	5	3.6
4	4	1	1	1	2	1.8
5	2	4	5	4	4	3.8
6	4	1	1	1	1	1.6
7	1	5	5	5	5	4.2
8	2	2	1	1	3	1.8
9	1	5	5	4	4	3.8
10	5	1	1	1	2	2
Skóre:	17.5	92.5	97.5	87.5	77.5	**73.75**

Tabuľka 4.2: Vyhodnotenie SUS dotazníka

výsledok. Z výsledného skóre 73.75 môžeme predpokladať, že použiteľnosť systému by nemala byť problémom ani v aktuálnom stave.

4.4 Zhrnutie

V súlade so zadaním práce som vybral vhodné testovacie metodiky a otestoval vytvorený prototyp webovej aplikácie. Kritické časti aplikácie boli pokryté jednotkovými a integračnými automatickými testmi.

Vývojové prostredia generované touto aplikáciou boli podrobené užívateľskému testovaniu metódou SUS dotazníku. Výsledkom testu bolo nadpriemerné skóre aj napriek tomu, že optimalizácia užívateľského rozhrania nebola cieľom práce.

Kapitola 5

Zhrnutie a výhľad do budúcna

V tejto časti práce sa pokúsim metódou SWOT analýzy zhodnotiť stav implementovaného prototypu.

5.1 SWOT Analýza

SWOT analýza je metóda, ktorá sa bežne používa pri definovaní konkurenčnej výhody alebo stratégie firmy, prípadne produktu. Skratka SWOT (podľa Weihrlicha [39]) znamená:

S Strengths (Silné stránky): charakteristiky produktu, ktoré ho robia lepším, atraktívnejším alebo kvalitnejším v porovnaní s alternatívami

W Weaknesses (Slabé stránky): charakteristiky produktu, ktoré ho robia horším, menej atraktívnym alebo menej kvalitným v porovnaní s alternatívami

O Opportunities (Príležitosti): externé prvky, ktoré by produkt dokázal využiť vo svoj prospech

T Threats (Hrozby): externé prvky, ktoré produktu škodia alebo spomaľujú jeho rozvoj

Na nasledujúcich riadkoch popíšem jednotlivé SWOT charakteristiky práce.

5.1.1 Silné stránky

Za silné vnútorné charakteristiky vyvinutého prototypu považujem predovšetkým:

1. Nízke nároky na prevádzku systému. Aplikácia nevyžaduje Kubernetes cluster a malo byť možné spustiť ju na akomkoľvek počítači, na ktorom je možné spustiť Docker Engine.

2. Zameranie na vzdelávací segment. Súčasťou aplikácie je logika sprístupňovania jednotlivých prostredí študentom a učiteľom podľa toho, v akých kurzoch sú zapísani. V prípade použitia aplikácie mimo FIT ČVUT však nepredpokladám výrazne odlišnú dátovú štruktúru dát o kurzoch, predmetoch, a semestroch. To zjednoduší prípadnú cestu aplikácie za brány FIT ČVUT.

3. Vysoká modulárnosť riešenia v kontexte generovaných prostredi. Vďaka kontajnerizačnej technológii je možné v aplikácii prakticky okamžite začať používať iný textový editor alebo iné vývojové nástroje.

5.1.2 Slabé stránky

Za slabé vnútorné charakteristiky vyvinutého prototypu považujem predovšetkým:

1. Absencia nasadenia aplikácie do produkčnej infraštruktúry a testovania v produkčnej infraštruktúre. Tento krok technicky blokuje použitie aplikácie v produkčnom prostredí.

2. Nedostatočné zabezpečenie pri komunikácii s API, ktoré vystavuje Docker Host pre správu kontajnerov. Pri spustení súčasnej verzie aplikácie je nutné sprístupniť aplikácii UNIX socket, čo je z bezpečnostného hľadiska problematické. Návrh aplikácie však umožňuje pridať možnosť komunikácie s Docker API iným spôsobom veľmi jednoducho podľa rozhrania *DockerEngineInterface*.

3. Nedostatočná prepracovanosť užívateľského rozhania. Síce nebolo prioritou v prototypovacej fáze, z dlhodobého hľadiska považujem prispôsobenie užívateľského rozhrania za jednu z hlavných priorít pre rozvoj projektu, keďže má priamy dopad na zážitok pri používaní aplikácie.

5.1.3 Príležitosti

Za vonkajšie okolnosti, ktoré by stav práce dokázali posunúť bližšie k plneniu svojej misie, považujem:

1. Rozšírenie na iné fakulty. V prípade pozitívnej skúsenosti s používaním aplikácie by mohla byť konkurenčnou výhodou FIT ČVUT oproti iným fakultám, prípadne by ju mohol iným fakultám ponúkať ako produkt.

2. Rozšírenie aplikácie mimo vzdelávací segment. Jadro aplikácie, ktorým je práca s generovanými prostrediami, je možné oddeliť od logiky prístupov k prostrediam podľa fakultných dát - čo môže uľahčiť tvorbu podobnej aplikácie cielenej na iný segment používateľov.

3. Rastúca požiadavka po cloudových službách. Vďaka tomu, že všetky užívateľské súbory sú uložené na serveri aplikácie, môžu k nim užívatelia pristupovať z viacerých zariadení a nemusia myslieť na synchronizáciu dát medzi svojimi zariadeniami.

5.1.4 Hrozby

Za vonkajšie okolnosti, ktoré by na prácu mohli mať negatívny vplyv, považujem:

1. Dostupnosť systému Linux na užívateľských zariadeniach. Vďaka WSL (Windows subsystem for Linux) sa užívateľom OS Windows v posledných rokoch zjednodušil prístup k vývojovým nástrojom a ich inštalácia, čo môže kontradikovať s potrebou užívateľa zjednodušiť si prácu s nastavovaním lokálneho vývojového prostredia, ergo používaním vyvinutej aplikácie.

2. Zrýchlenie vývoja podobných aplikácií vďaka pandémii COVIDu, kedy cloudové nástroje na tímovú spoluprácu v tímoch softvérových inžinierov zaznamenali rastúci dopyt. To znamenalo urýchlenie vývoja konkurenčných nástrojov.

3. Potreba technickej podpory pre užívateľov aplikácie. V prípade používania veľkým množstvom študentov môže vzniknúť množstvo prevádzkových problémov, zvlášť v dobe tesne po nasadení aplikácie. Tie bude potrebné odbavovať, na čo nemusia byť na inštitúciách, kde je aplikácia nasadená, dostupné kapacity v kontexte ľudských zdrojov.

Záver

Primárnym cieľom práce bolo navrhnúť a implementovať webový nástroj pre jednoduchú tvorbu vývojových prostredí pre softvérových inžinierov, resp. študentov softvérového inžinierstva s využitím princípu kontajnerizácie.

V analytickej časti práce som zadefinoval potreby jednotlivých aktérov a identifikoval ich súčasné problémy, na ktoré poskytuje vyvinutý prototyp riešenia. Výsledkom bola definícia funkčných požiadaviek. Ďalej som spracoval prehľad existujúcich riešení, z ktorého som následne vychádzal v návrhovej časti práce. Aby som vyvinutý prototyp odlíšil od jeho alternatív, dbal som na nízke nároky pre beh systému a zameranie na prostredie školy, kedže nástroj má primárne slúžiť študentom. Na základe požiadaviek na systém som spracoval doménový model, ktorý vymedzuje a popisuje jednotlivé časti systému.

V návrhovej časti práce som s využitím podporných nástrojov a princípu kontajnerizácie implementovaného nástrojom Docker spracoval podklady pre vyhotovenie prototypu webovej aplikácie. Zvolil som a popísal som všetky relevantné nástroje, princípy a technológie, ktoré som neskôr využil v implementačnej časti. Navrhol som tiež kľúčové procesy, ktoré sú potrebné pre korektné fungovanie aplikácie pri nasadení do produkcie.

V implementačnej časti práce som popísal implementačné špecifiká aplikácie. Najvýraznejšími z nich boli integrácia modulu EasyAdmin, vďaka ktorému mi bolo umožnené jednoducho navrhnúť užívateľské rozhranie aplikácie a mechanizmus spúšťania vývojových prostredí inšpirovaný návrhovým vzorom Chain of Responsibility. Popísal som tiež integrácie so službami, ktoré sú pre beh aplikácie nevyhnutné.

Implementovaný prototyp som následne otestoval. Kritické súčasti aplikácie boli pokryté automatickými testami. Vytvorené prostredia som podrobil užívateľskému testovaniu s nadpriemerným výsledkom.

Súčasťou práce je aj ukážka použitia prototypu a zhodnotenie stavu prototypu v čase odovzdania tejto práce metódou SWOT analýzy, resp. návrh jeho ďalšieho rozšírenia.

Všetky ciele práce popísané v úvode práce, resp. v zadaní práce považujem

za úspešne splnené.

Svojou prácou som dokázal, že je technicky uskutočniteľné, aby fakulta svojim študentom sprístupňovala predzostavené obrazy vývojových prostredí používaných v jednotlivých predmetoch s využitím kontajnerizačnej služby Docker. Tým sa dosiahne zníženie času potrebného na nastavenie prostredí študentmi, čo prispeje k zlepšeniu zážitku zo štúdia.

Vyvinutý nástroj sa od svojich alternatív odlišuje primárne v potrebných prerekvizitách pre spustenie systému. Systém totiž riadi beh kontanjerov sám a pre svoju prevádzku nevyužíva orchestračný nástroj, akým je napr. Kubernetes, vďaka čomu je spustiteľný na ľubovoľnom serveri. To znižuje náklady na prevádzku systému.

Priestor pre nadviazanie na túto prácu vidím primárne v zvýšení úrovne zabezpečenia aplikácie, najmä v oblasti prístupu ku generovaným vývojovým prostrediam, prípadne v oblasti komunikácie s Docker Engine. Ďalším možným rozšírením je implementácia podpory iných kontajnerizačných nástrojov, akým je napr. Podman.

Literatúra

[1] VIC ČVUT: Organizace a správa výuky – Kos. https://ist.cvut.cz/nase-sluzby/kos/, 2021, [Webová stránka], zobrazené dňa 21. 04. 2022.

[2] Jirůtka, J.: KOSApi. https://kosapi.fit.cvut.cz/projects/kosapi/wiki, 2015, [Webová stránka], zobrazené dňa 21. 04. 2022.

[3] VIC ČVUT: Usermap. https://ist.cvut.cz/nase-sluzby/usermap/, 2021, [Webová stránka], zobrazené dňa 29. 04. 2022.

[4] Zheng, M.: Software Verification and Validation. Informal Semantics of UML Use Case Diagram. https://cs.uwlax.edu/~mzheng/CS743Fall19/UseCaseDiagrams.html, [Webová stránka], zobrazené dňa 29. 05. 2022.

[5] Larman, C.: *Applying UML and Patterns. An Introduction to Object-Oriented Analysis and Design and Iterative Development*. ISBN 0-13-148906-2, 616 s. Dostupné z: https://personal.utdallas.edu/~chung/SP/applying-uml-and-patterns.pdf

[6] *Domain Modeling*. Berkeley, CA: Apress, 2007, ISBN 978-1-4302-0369-8, str. 24, doi:10.1007/978-1-4302-0369-8_2. Dostupné z: https://doi.org/10.1007/978-1-4302-0369-8%5F2

[7] Dybka, P.: Crow's foot notation. https://vertabelo.com/blog/crow-s-foot-notation, Apr 2016, [Webová stránka], zobrazené dňa 15. 06. 2022.

[8] Coder Technologies Inc.: Coder Docs. https://coder.com/docs/coder/v1.31, 2022, [Webová stránka], zobrazené dňa 29. 04. 2022.

[9] GitHub Inc.: Blazing fast cloud developer environments. https://github.com/features/codespaces, 2022, [Webová stránka], zobrazené dňa 29. 04. 2022.

[10] Gitpod GmbH: Always ready to code. https://www.gitpod.io/, 2022, [Webová stránka], zobrazené dňa 29. 04. 2022.

[11] Portnoy, M.: *Virtualization essentials*, kapitola 1: The Essentials and Beyond. Wiley Pub. Inc., 2012, ISBN 9781119267720.

[12] Containers or VMS? – pros and cons. https://open-telekom-cloud.com/en/blog/cloud-computing/container-vs-vm, [Webová stránka], zobrazené dňa 23. 04. 2023.

[13] Docker: Accelerated, Containerized Application Development. Developfaster.Runanywhere., [Webová stránka], zobrazené dňa 31. 01. 2023.

[14] Docker Inc.: Docker Overview. https://docs.docker.com/get-started/overview/#docker-architecture, Aug 2022, [Webová stránka], zobrazené dňa 15. 06. 2022.

[15] What is Podman? https://podman.io/whatis.html, [Webová stránka], zobrazené dňa 15. 06. 2022.

[16] Stack Exchange Inc.: Stack overflow developer survey 2021. https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-other-tools, [Webová stránka], zobrazené dňa 15. 06. 2022.

[17] Turing School of Software and Design: Back-end Engineering Curriculum. https://backend.turing.edu/module1/lessons/four_pillars_of_oop, [Webová stránka], zobrazené dňa 23. 04. 2023.

[18] Kong, Q.; Siauw, T.; Bayen, A. M.: *Python programming and numerical methods: A guide for engineers and scientists*. Elsevier, 2021, ISBN 9780128195505, 127–134 s.

[19] Valenta, M.: BI-DBS: Databázové systémy – úvod. [prednáška], Praha, ČR, FIT ČVUT, 2020. Dostupné z: https://courses.fit.cvut.cz/BI-DBS/@B201/materials/slides/pres-les01-uvod.pdf

[20] Valenta, M.: BI-DBS: Relační model, relační algebra. [prednáška], Praha, ČR, FIT ČVUT, 2020. Dostupné z: https://courses.fit.cvut.cz/BI-DBS/@B201/materials/slides/hand-les03-relacni-model-a-ra.pdf

[21] Lorenz, M.; Rudolph, J.-P.; Hesse, G.; aj.: Object-Relational Mapping Revisited - A Quantitative Study on the Impact of Database Technology on O/R Mapping Strategies. In *Hawaii International Conference on System Sciences*, 2017, doi:10.24251/HICSS.2017.592.

[22] Doctrine DBAL documentation. https://www.doctrine-project.org/projects/doctrine-dbal/en/current/reference/introduction.html, [Webová stránka], zobrazené dňa 15. 06. 2022.

[23] Fowler, M.: *3: Mapping to relational databases*. Boston, MA: Addison-Wesley, 2015, ISBN 9780321127426, str. 33–36.

[24] Fraser, B. Y.: RFC 2196: Site security handbook. Sep 1997. Dostupné z: https://datatracker.ietf.org/doc/html/rfc2196#section-4.4

[25] Hardt, D.: RFC 6749: The OAuth 2.0 authorization framework. https://www.rfc-editor.org/rfc/rfc6749, [Webová stránka], ISSN 2070-1721.

[26] Achour, M.; Betz, F.; Dovgal, A.; aj.: PHP Manual. https://www.php.net/manual/en/, Mar 2023, [Dokumentácia].

[27] Symfony: Symfony, high performance PHP framework for web development. https://symfony.com/, [Webová stránka], zobrazené dňa 15. 06. 2022.

[28] Grudl, D.: Nette – Comfortable and safe web development in PHP. https://nette.org/en/, [Webová stránka], zobrazené dňa 15. 06. 2022.

[29] The PHP framework for web artisans. https://laravel.com/, [Webová stránka], zobrazené dňa 15. 06. 2022.

[30] Doctrine. https://www.doctrine-project.org/, [Webová stránka], zobrazené dňa 15. 06. 2022.

[31] PostgreSQL: About. https://www.postgresql.org/about/, [Webová stránka], zobrazené dňa 15. 06. 2022.

[32] Bernhauer, D.: BI-TWA: Návrhové vzory a architektura aplikace. [prednáška], Praha, ČR, FIT ČVUT, 2021. Dostupné z: https://courses.fit.cvut.cz/BI-TWA/media/topics/t08-architecture.pdf

[33] Bernard, B.: Prezentační vzory z rodiny MVC. https://zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/, [Webová stránka], zobrazené dňa 15. 06. 2022.

[34] Dauchy, V.: vdauchy/docker-php-api. https://github.com/vdauchy/docker-php-api, [Knižnica], verzia 1.41.

[35] Shvets, A.: Command. https://refactoring.guru/design-patterns/command, [Webová stránka], zobrazené dňa 23. 04. 2023.

[36] The League of Extraordinary Packages: league/oauth2-client. https://oauth2-client.thephpleague.com/, [Knižnica], verzia 2.6.

[37] Amman, P.; Offutt, J.: *Introduction to softwate testing*, kapitola 2.3 Testing levels based on software activity. Cambridge University Press, 2017, ISBN 9781107172012, str. 52–56.

[38] Brooke, J.: SUS: A quick and dirty usability scale. *Usability Eval. Ind.*

[39] Weihrich, H.: The TOWS matrix—A tool for situational analysis. *Long Range Planning*, ročník 15, č. 2, 1982: s. 60–61, ISSN 0024-6301, doi: https://doi.org/10.1016/0024-6301(82)90120-0. Dostupné z: `https://www.sciencedirect.com/science/article/pii/0024630182901200`

Dodatok **A**

Zoznam použitých skratiek a termínov

OS Operačný systém

SSO Single sign-on (jednotné prihlásenie)

API Application programming interface (rozhranie pre programovanie aplikácií)

IDE Integrated development environment (integrované vývojové prostredie)

SaaS Software as a service (softvér ako služba)

CLI Command line interface (konzolové rozhranie)

Klient (angl. *client*) je softvérový nástroj, ktorý umožňuje užívateľovi interagovať s poskytovanou službou.

Hostiteľ (angl. *host*) je fyzický stroj zodpovedný za beh jedného alebo viacerých kontajnerov.

Démon (ang. *daemon*) je proces spustený na pozadí, ktorý zodpovedá za spracovanie požiadaviek pre správu kontajnerov (vytvorenie z určeného obrazu, spustenie, vypnutie, atp.)

Obraz (angl. image) je strojovo spracovateľnou sadou inštrukcii, ktorá je základom pre vytvorenie kontajnera.

Kontajner (angl. *container*) je spustiteľnou inštanciou obrazu. Pozostáva z obrazu, behového prostredia a sady inštrukcii.

Register (angl. *registry*) je webová služba, ktorá umožňuje ukladanie, zverejnenie a distribúciu obrazov. Register umožňuje užívateľom uložiť alebo zverejniť svoje obrazy, vyhľadávať v zozname zverejnených obrazov alebo stiahnuť si zverejnený obraz.

Framework Sada podporných balíčkov zdrojového kódu uľahčujúca vývoj aplikácie. Poskytuje implementáciu rutinných procesov počas behu aplikácie (napr. validácia HTTP dotazu), čím oslobodzuje logiku aplikácie od riešenia problematiky nižších úrovní.

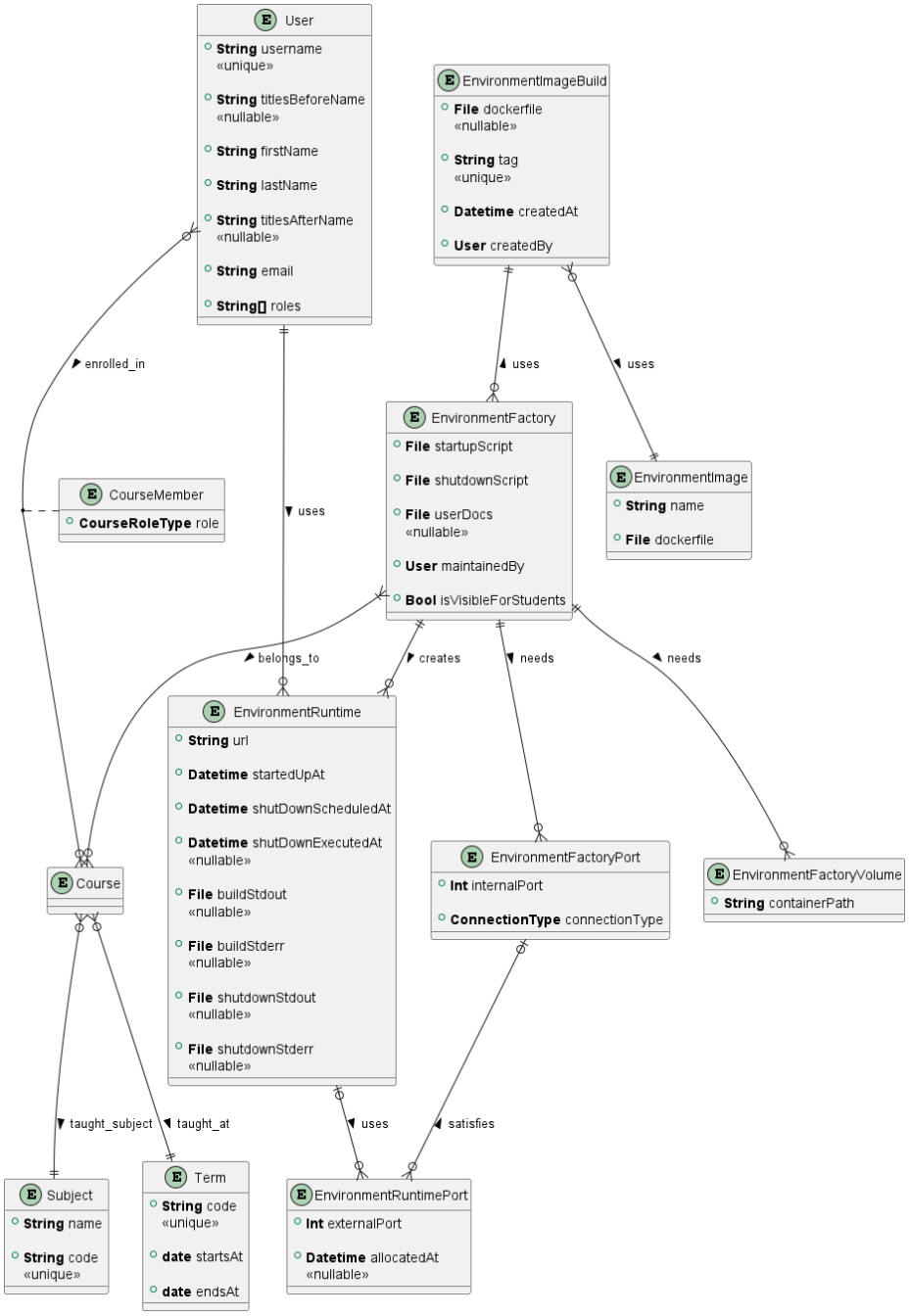
Dodatok B

Obsah priloženého média

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

Dodatok C

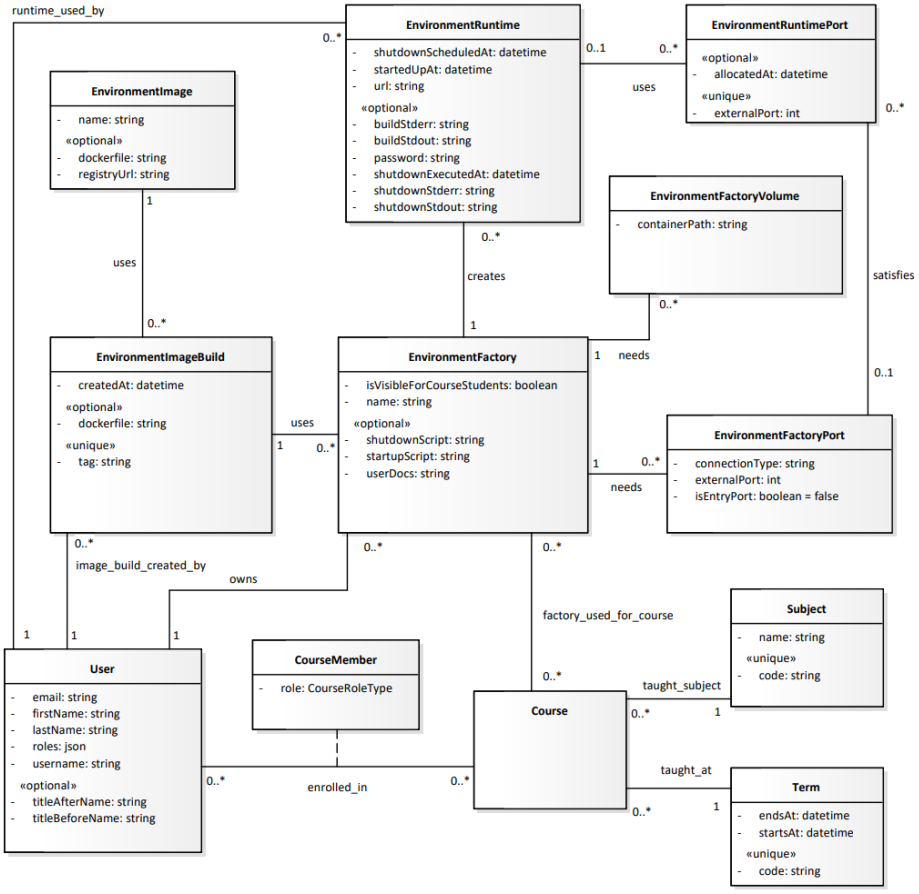
Kompletný doménový model



Obr. C.1: Kompletný doménový model

Dodatok D

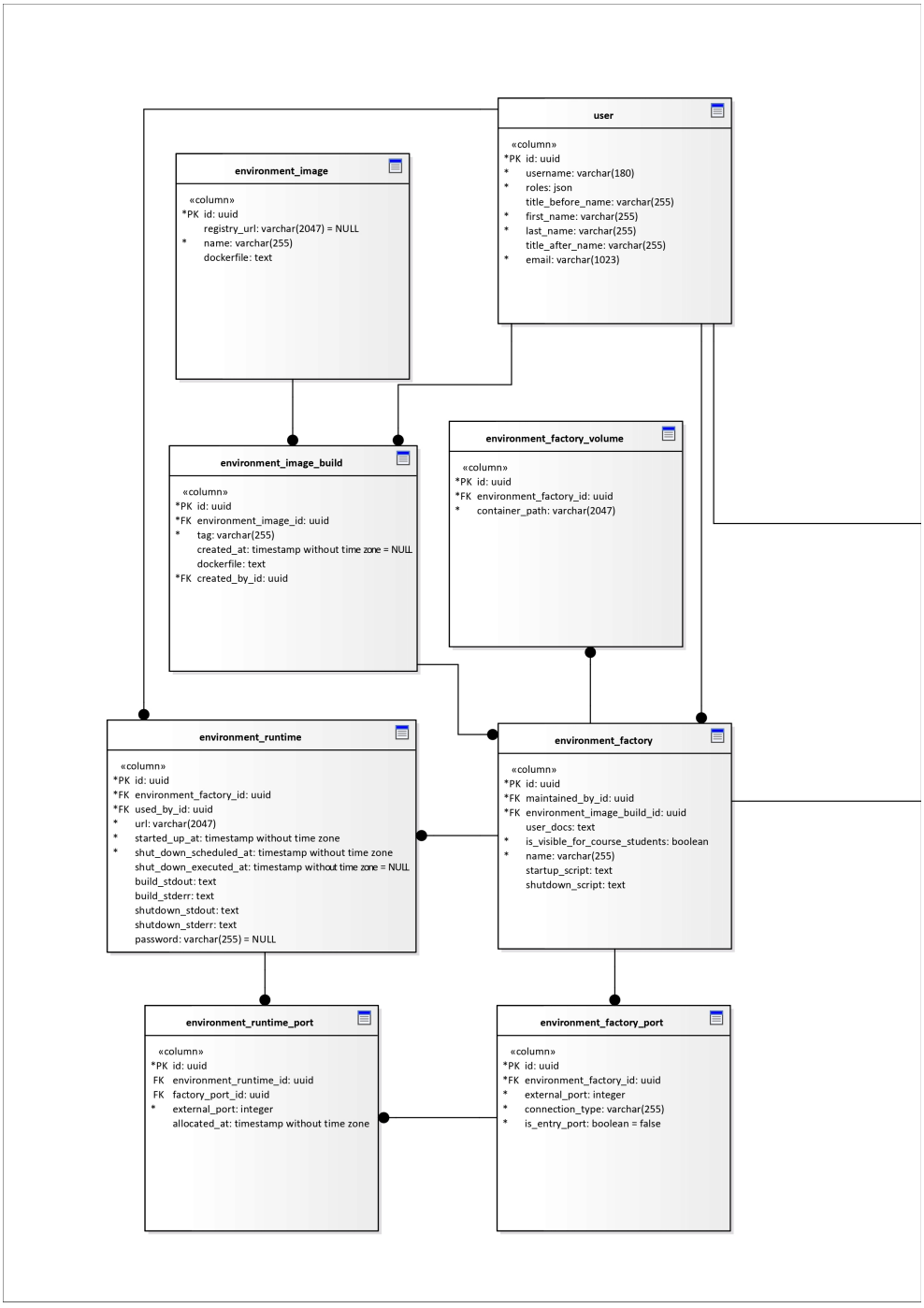
Kompletný diagram tried



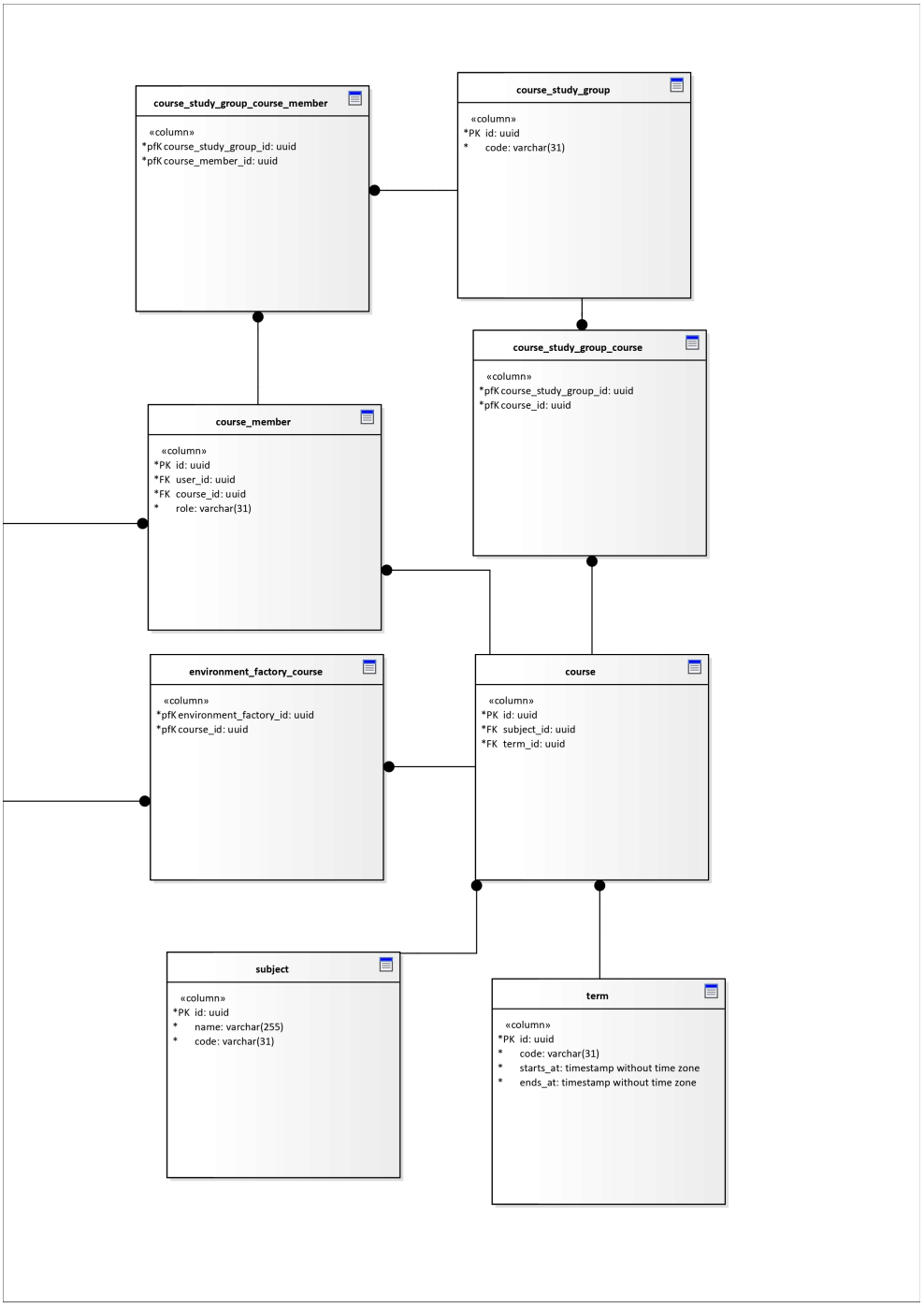
Obr. D.1: Kompletný diagram tried

Dodatok E

Kompletný relačný model



Obr. E.1: Kompletný relačný model — časť 1

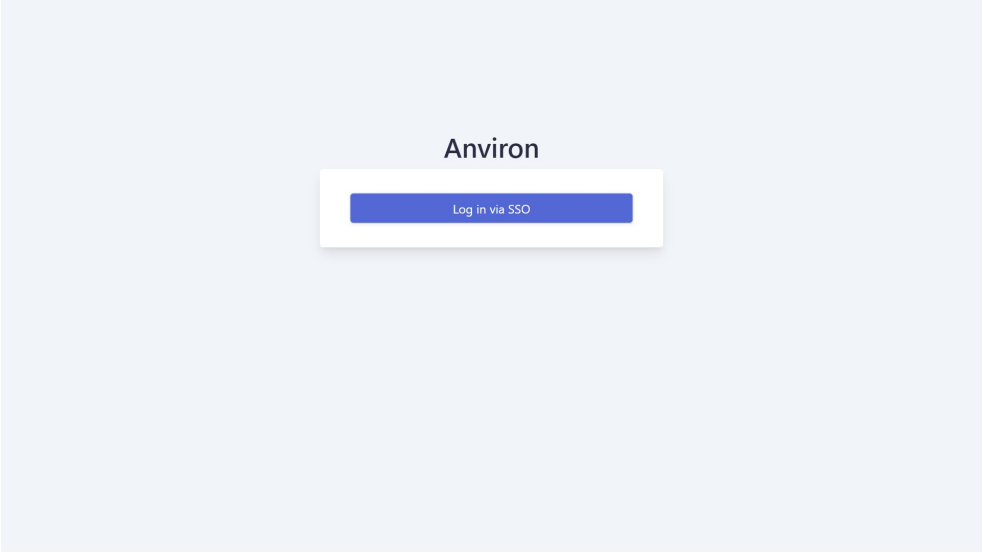


Obr. E.2: Kompletný relačný model — časť 2

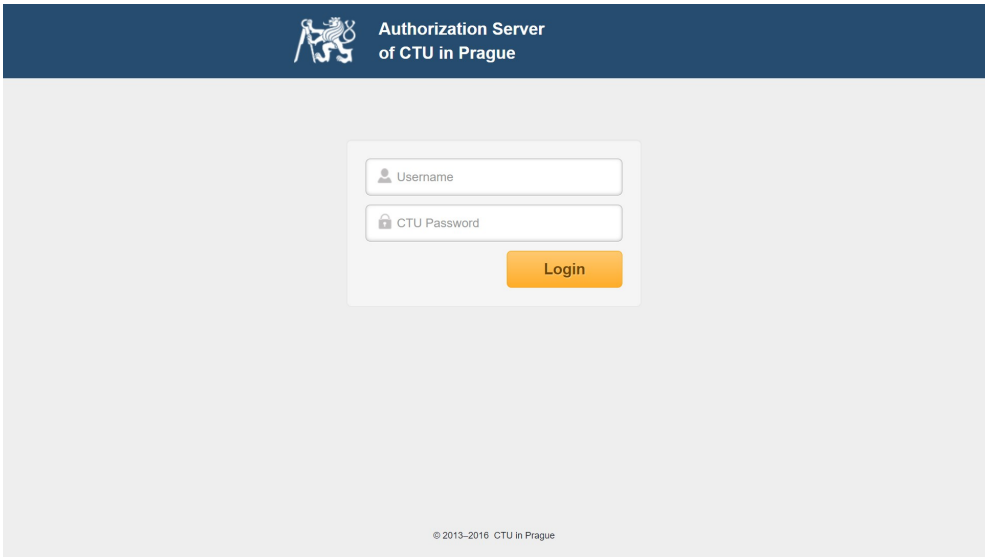
Dodatok F

Ukážka použitia prototypu

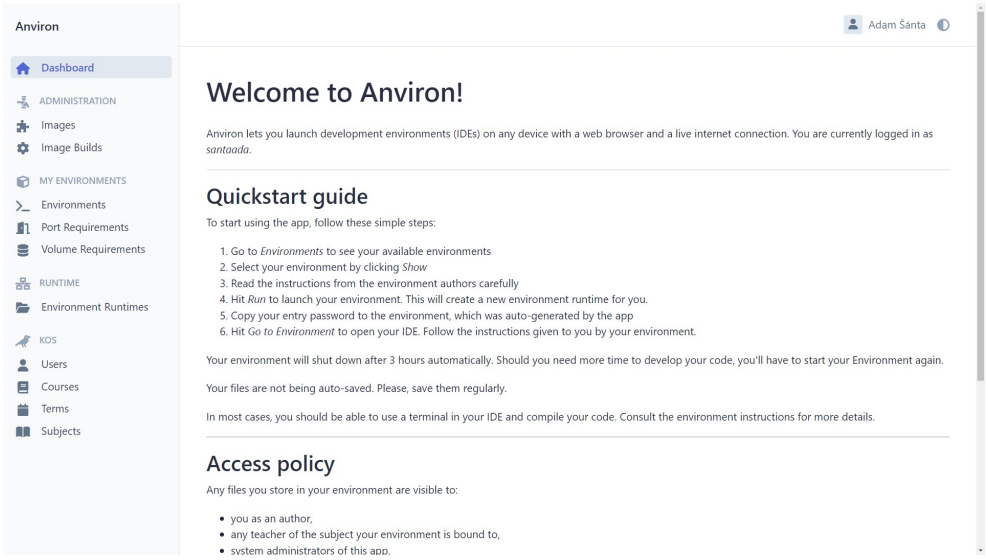
Aby bolo možné demonštrovať kompletnú funkcionalitu aplikácie, v ukážke použitia je prítomné rozhranie správcu systému.



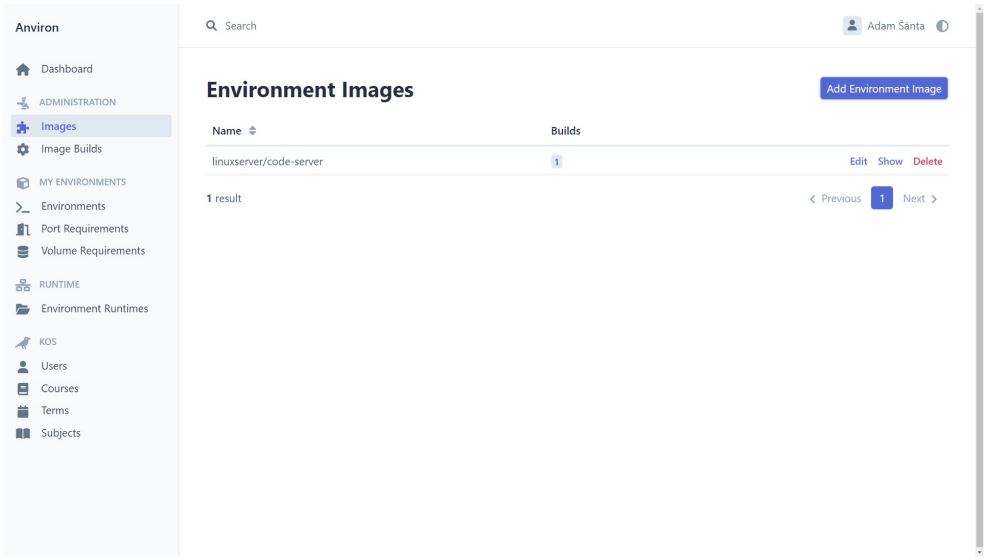
Obr. F.1: Domovská obrazovka pred autentizáciou užívateľa



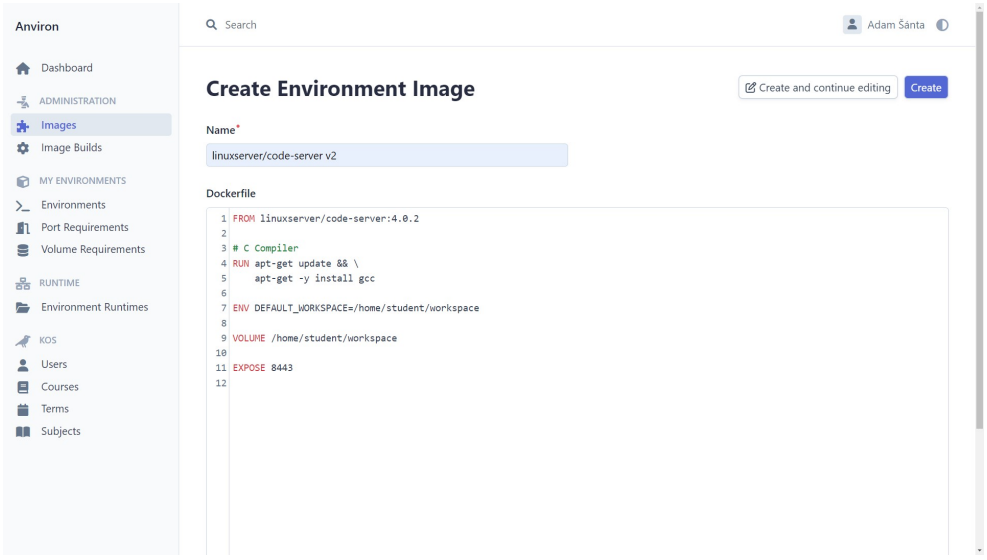
Obr. F.2: Autentizácia užívateľa pomocou SSO



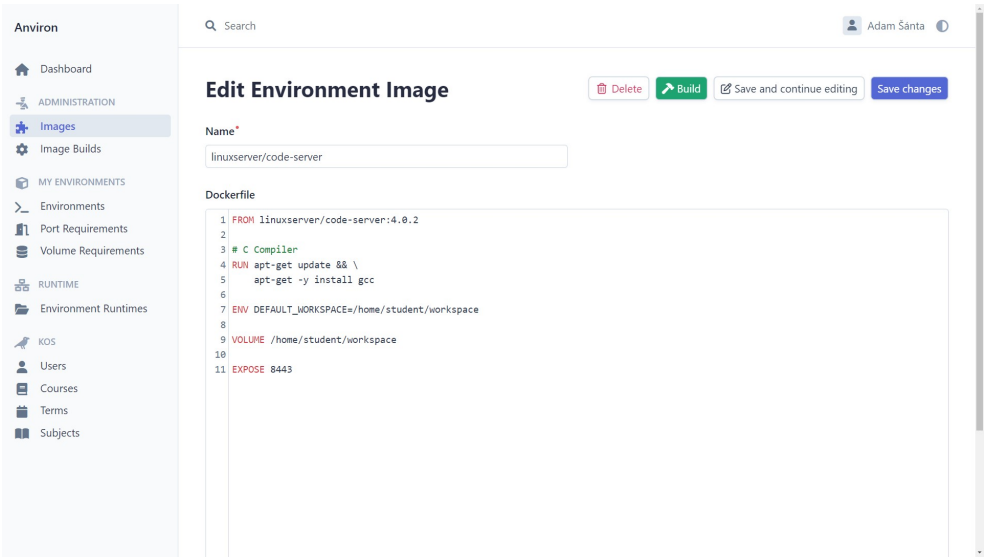
Obr. F.3: Domovská obrazovka po autentizácii užívateľa



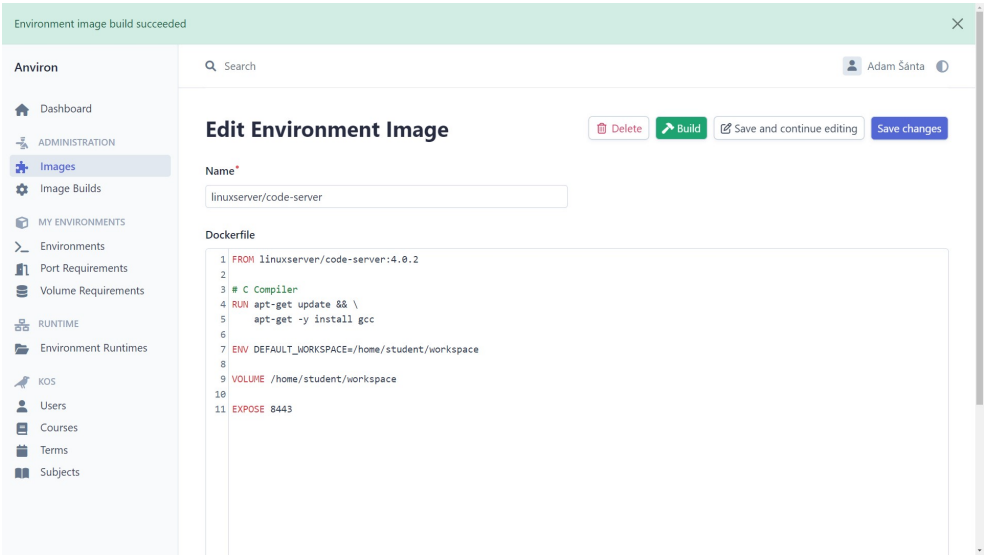
Obr. F.4: Zoznam obrazov pre vývojové prostredia



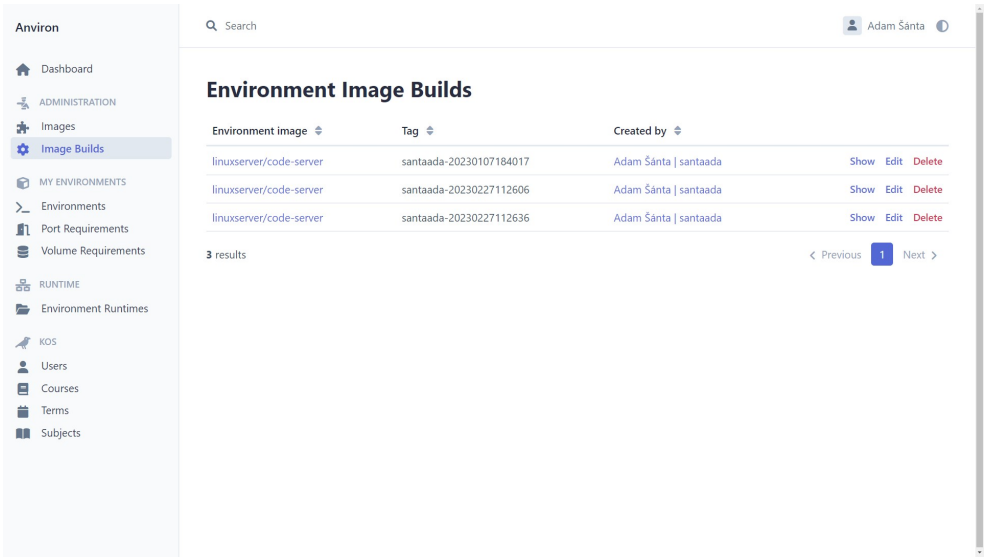
Obr. F.5: Vytvorenie obrazu pre vývojové prostredie



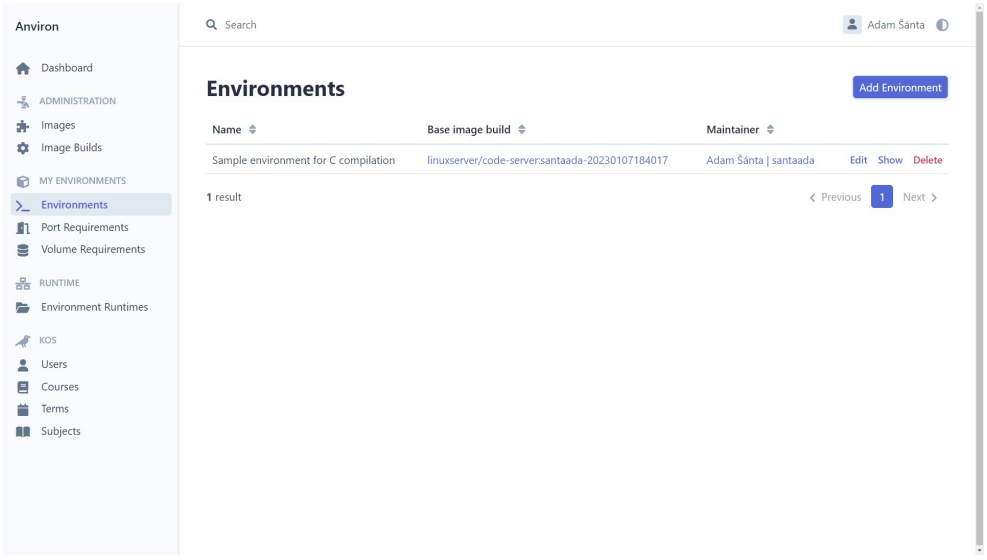
Obr. F.6: Detail obrazu pre vývojové prostredie



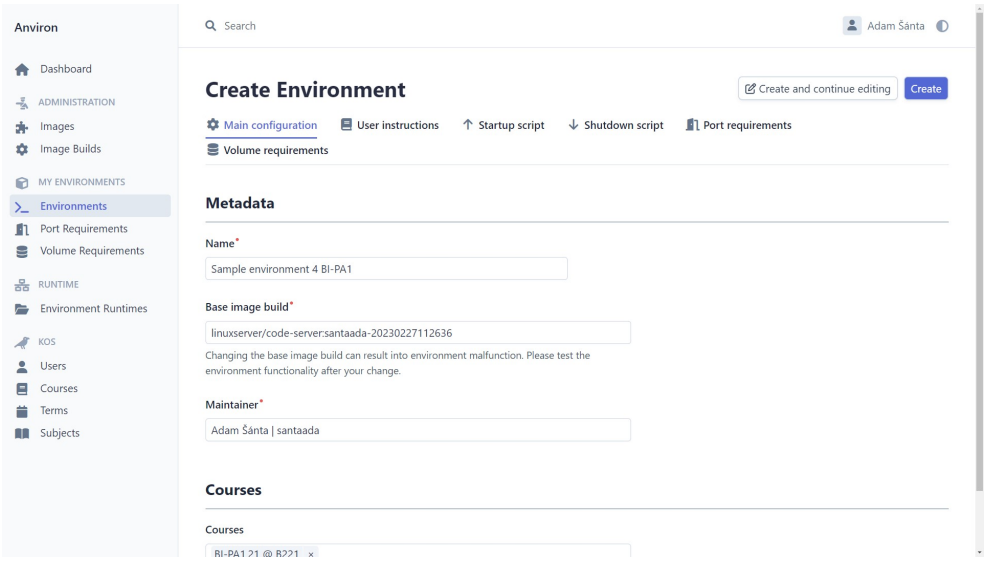
Obr. F.7: Úspešné zostavenie obrazu pre prostredie



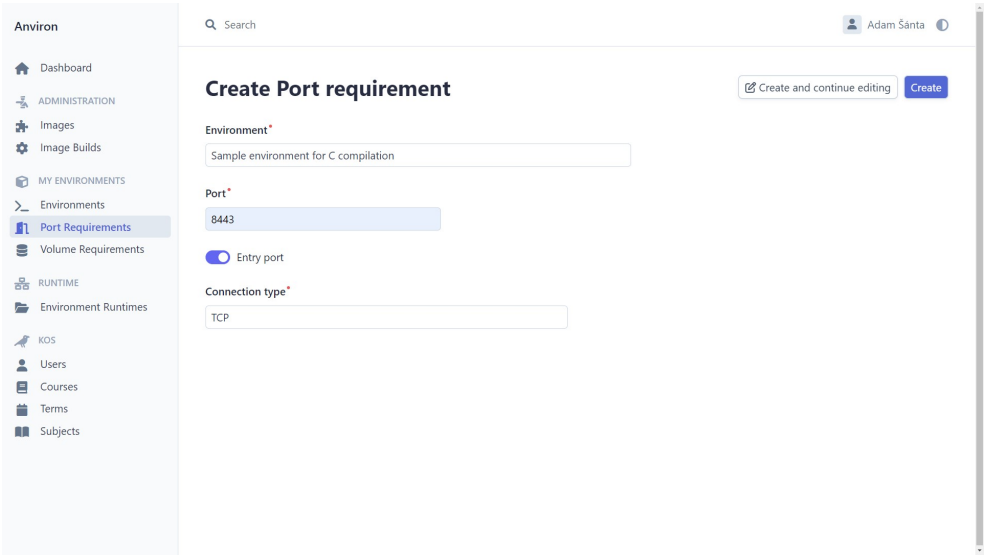
Obr. F.8: Zobrazenie všetkých zostavení obrazu pre prostredie



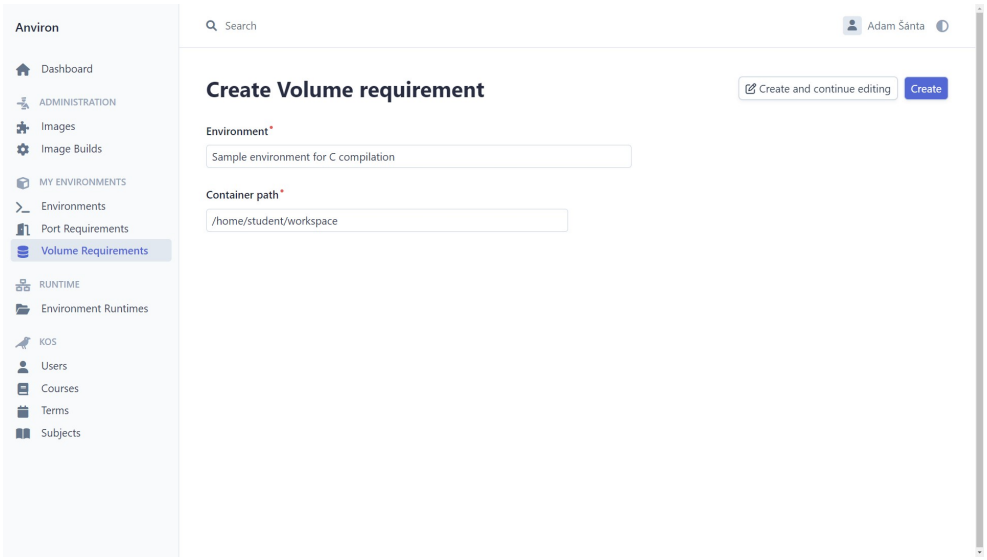
Obr. F.9: Zobrazenie dostupných prostredí



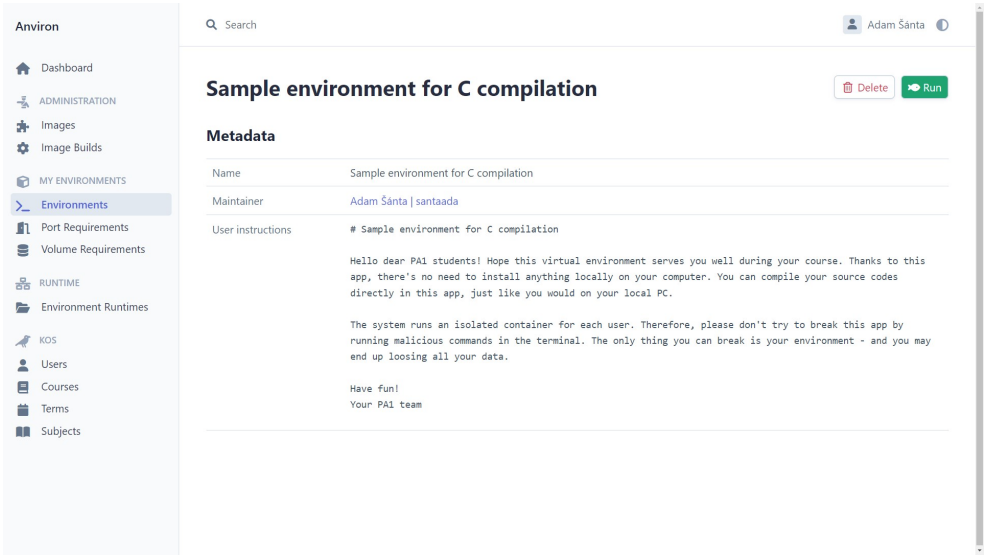
Obr. F.10: Vytvorenie nového prostredia



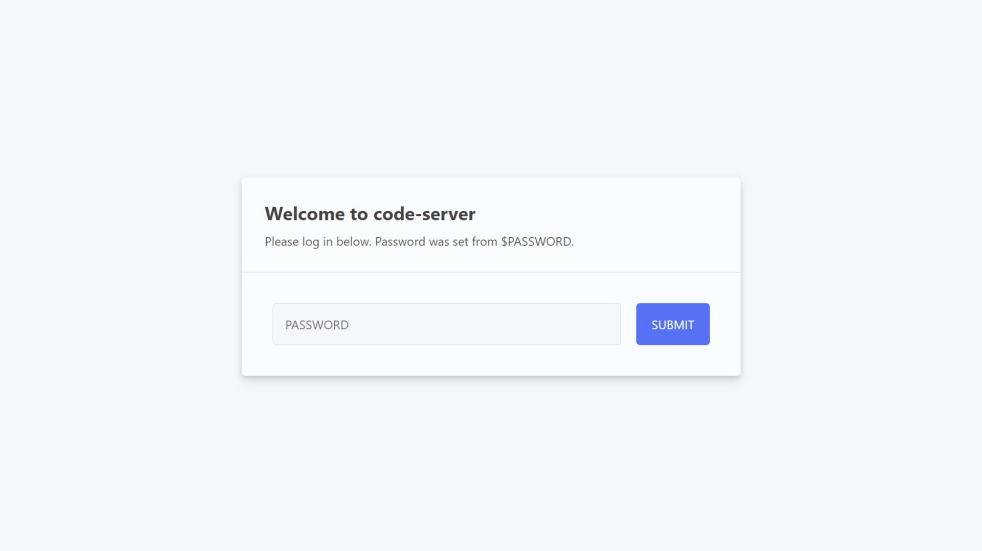
Obr. F.11: Priradenie potreby alokácie portu k prostrediu



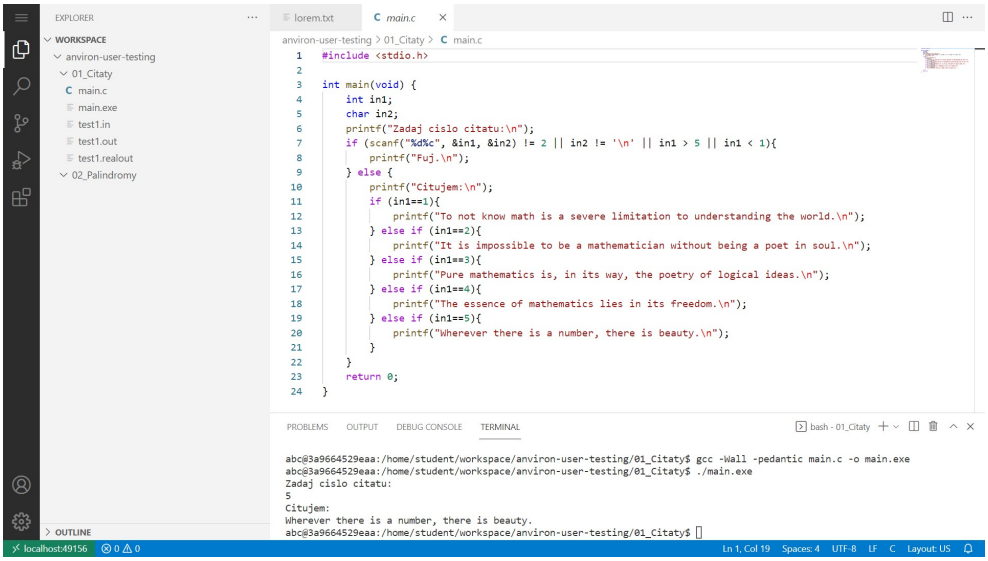
Obr. F.12: Priradenie potreby alokácie súborov k prostrediu



Obr. F.13: Detail prostredia



Obr. F.14: Prihlásenie do virtuálneho prostredia pomocou vygenerovaného hesla



Obr. F.15: Ukážka práce na zdrojovom kóde vo virtuálnom prostredí