Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Webové a softwarové inženýrství

Diplomová práce

XCalc: Systém pro správu intenzivních výpočtů

Bc. Tomáš Polívka

Vedoucí práce: Ing. Tomáš Bartoň

5. května 2015

Poděkování

Tímto bych rád poděkoval hlavně své rodině a přítelkyni za podporu po celou dobu mého vysokoškolského studia. Dále bych chtěl poděkovat vedoucímu své práce, Ing. Tomáši Bartoňovi, za pomoc během tvorby této práce a trpělivost s mými neustálými dotazy. Nakonec bych poděkoval všem svým přátelům, kteří obětovali svůj čas a pomohli mi s gramatickou korekturou této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 5. května 2015 .....

Odkaz na tuto práci

Polívka, Tomáš. *XCalc: Systém pro správu intenzivních výpočtů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Cílem této práce je provedení analýzy požadavků a následná implementace webového rozhraní pro aplikaci *xCalc*, jejímž úkolem je správa extenzivních výpočtů na počítačovém clusteru. Realizace je zaměřena především na implementaci backendové funkcionality a interakce s ostatními systémy.

Klíčová slova xCalc, extenzivní výpočty, Ruby on Rails, API, KOSapi, Gitlab

Abstract

The goal of this thesis is to create analysis of requirements, followed by implementation of web interaface for application *xCalc*. Its main interest is management of extensive calculations on the a computer cluster. Realization is mainly focused on implentation of the backend functionality and interactions with the other systems.

Keywords xCalc, extensive calculations, Ruby on Rails, API, KOSapi, Gitlab

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato magisterská práce se zabývá návrhem a implementací webového rozhraní aplikace *xCalc*, jejíž název vychází ze slovního spojení „extensive calculations“. Pomocí této aplikace by měl student či zaměstnanec ČVUT fakulty informačních technologií (dále pouze ČVUT FIT) spouštět časově náročné výpočetní úlohy na fakultním clusteru *storm* a umožnit přístup k jejich výstupům.

V průběhu bakalářského studia na ČVUT FIT se student mimo jiné učí základům programování. Jeho nabyté znalosti jsou během semestru zevrubně testovány pomocí aplikace *Progtest*, kterou vytvořil Ing. Ladislav Vagner, Ph.D. Studentův algoritmus je po odevzdání do této aplikace podroben testům extrémních situací, aby bylo ověřeno, že student rozumí danému problematice a hlavně zda si je vědom jeho slabých míst. Na magisterském studiu je již umění programování považováno za samozřejmost a pozornost se přesouvá spíše ke studiu složitějších algoritmů. Praktické úkoly jsou zaměřeny více na experimentování prováděném na studentem naprogramovaném algoritmu.

Jedním z takovýchto magisterských předmětů je například *Problémy a algoritmy*[1], který se zabývá teorií okolo výpočetně složitých problémů a jejich potenciálními řešeními. Přednášky a cvičení jsou doplněny několika praktickými domácími úlohami, kdy studenti dostanou zadaný problém, pro který mají při použití daného přístupu implementovat jeho řešení. Volba programovacího jazyka je volná a u výsledků není přikládána priorita časovým měřením, které může být silně ovlivněno studentovým hardwarem či zvoleným programovacím jazykem, ale spíše počtu výsledných cyklů. Výstupem této úlohy je zpráva, ve které student charakterizuje výsledky a statistiky implementovaného řešení. Zdrojový kód je odevzdáván především z důvodu zamezení plagiátorství. Vlastní forma odevzdávání není nutně definována, ale u většiny cvičících se jedná o nahrání zprávy a archivu se zdrojovými kódy do studentova

[1] další informace o předmětu MI-PAA dostupné na `https://edux.fit.cvut.cz/courses/MI-PAA/`

prostoru na studijním portále *EDUX*[2]. Protože aplikace *Progtest* je zaměřena na testování programů vytvořených v jazyku C nebo C++, její použití by mnohé studenty omezovalo. Zde by měla přijít ke slovu aplikace *xCalc*.

Aplikace *xCalc* by měla studentům zjednodušit spouštění těchto časově náročných úloh a zpracování výsledků, které mnohdy zabírá studentům i stejné množství času, jaké stráví nad návrhem a implementací samotné úlohy. Důsledkem má být stav, kdy student čas věnuje hlavnímu předmětu zadané úlohy, tedy návrhu algoritmu, a je minimalizován čas strávený aktivitami, které nejsou pro problematiku zadané úlohy relevantní. Na druhou stranu není cílem vytvoření aplikace, která by udělala všechnu práci za studenta.

Kromě spouštění domácích úkolů by měla aplikace *xCalc* sloužit i pro provádění výzkumných výpočtů mimo definované předměty, například v rámci doktorandského studia.

Autor na tomto projektu spolupracoval s vedoucím této magiterské práce Ing. Tomášem Bartoněm, který má na starosti vývoj systému umožňujícího vytvoření virtuálního kontejneru pro odeslanou výpočetní úlohu na školním clusteru *storm*. Vybraný kontejner obsahuje předpřiravenou sadu nástrojů pro spuštění zdrojového kódu v daném programovacím jazyce.

[2]dostupné na `https://edux.fit.cvut.cz`

Kapitola 1

Cíl práce

Prvním cílem této práce je provést analýzu existujících systémů pro správu extenzivních výpočtů. Výsledkem bude výběr systému, který bude rozšířen o potřebnou funkcionalitu nebo rozhodnutí o implementaci vlastního řešení. Následuje analýza požadavků, ze které vychází návrh uživatelského rozhraní. Hlavním cílem této práce je návrh a implementace webového rozhraní, které bude umožňovat odesílání zdrojových kódů z webové aplikace *Gitlab* k výpočtu na clusteru ČVUT FIT. Navržená aplikace bude umožňovat import dat z *KOSapi*. Toto řešení bude doplněno o samostatnou komponentu pro zpracovávání výstupu v zadaném formátu, čímž bude umožňovat flexibilní přizpůsobení různým typům úloh a předmětů. Jako příklad budou implementovány ukázky úloh z předmětů MI-PAA a MI-ADM.

Kapitola 2

Analýza požadavků

Tato kapitola nejprve seznámí čtenáře s analýzou aktivit v jednotlivých sekcích webové aplikace. Následuje obecný popis chování uživatelských rolí. Další sekce popisuje vytvořené persony pro jednotlivé role.

2.1 Gitlab

Gitlab Community Edition (dále pouze „Gitlab“) je open-source webová aplikace[3], která slouží jako správce repozitářů aplikace *git* pro verzování kódu. Základní funkcionalitu rozšiřuje o možnost interaktivní spolupráce vývojářů, jako například code review včetně komentářů nebo vytváření merge requestů. Od stejných tvůrců existuje ještě druhá komerční verze této aplikace *Gitlab Enterprise Edition*.

ČVUT FIT instanci *Gitlabu*[4] provozuje na svých serverech a poskytuje ho studentům k použití na jejich vlastních projekt, ale i pro správu fakultních projektů nebo materiálů k některým předmětům, čímž umožňují studentům se podílet na jejich vytváření a zdokonalování.

Koncept použití aplikace *xCalc* počítá s tím, že student implementuje zadanou úlohu, jejíž zdrojový kód bude verzovaný na školním *Gitlabu*. Ten pak pomocí webového rozhranní *xCalc* odešle k výpočtu na školní cluster Storm.

2.2 Role uživatelů

Již ze zadání jsou zřejmé dvě skupiny uživatelů, a to studenti a cvičící. Tyto skupiny jsou doplněny o administrátory, tedy správce aplikace *xCalc*. Následuje charakteristika těchto rolí.

[3] další informace dostupné na `https://about.gitlab.com`
[4] dostupné na `https://gitlab.fit.cvut.cz`

2.2.1 Administrátor

Jedná se o malý počet uživatelů, kteří spravují aplikaci a mohou nastavovat její vlastnosti. Jsou také zodpovědní za přípravu prostředí pro import uživatelů a cvičících pro aktuální semestr. Tato skupina uživatelů nese velkou zodpovědnost za funkcionalitu systému a je nutné, aby jejími členy byli pouze uživatelé, kteří jsou dobře seznámeni s interními procesy v aplikaci.

2.2.2 Cvičící

Ačkoliv je tato skupina označena jako cvičící, jsou do této skupiny zařazení i přednášející a garanti předmětu. Označení je odvozeno od faktu, že aplikace se týká z valné většiny domácích úloh, které jsou standardně řešeny v rámci cvičení nebo laboratoří.

2.2.3 Student

Tuto skupinu není nutné detailně popisovat. Jedná se o studenty daného předmětu, kteří během semestru dostanou za úkol vypracovat zadané domácí úlohy a následně je odevzdat pomocí aplikace *xCalc*.

2.3 Seznam aktivit

Tato podkapitola detailně popisuje aktivity v jednotlivých sekcí aplikace. Výsledek analýzy je primárním zdrojem informací pro návrh struktury webu a UI.

2.3.1 Studijní struktura

V této části webové aplikace bude probíhat manipulace s daty vztahujícími se k studiu, které budou získávány z fakultního informačního systému *KOS*. Jedná se o import dat a případné doplnění nebo úpravy jednotlivých entit a relací mezi nimi.

2.3.1.1 Administrátor

Na začátku nového semestru ještě neexistují vazby cvičících na předměty a není tedy možné, aby import předmětu byl spouštěn přímo cvičícími. Za tento iniciální import tedy zodpovídá Administrátor.

- Import předmětů
- import cvičících
- Zobrazení výsledku importu

2.3.1.2 Cvičící

Cvičící má na starosti správu importovaných paralelek a studentů. Má možnost upravit relace mezi studenty, cvičícími a paralelek. Zejména před začátkem a v prvních týdnech semestru probíhá mnoho přesunů mezi paralelkami. Jedná se důsledek snahy studentů optimalizovat svůj studijní časový rozvrh nebo dokonce změna zapsaných předmětů. Protože systém spolupracuje s mnoha externími službami, je potřeba provést nutné kroky pro přípravu budoucí komunikace a umožnit zobrazení stavu synchronizace. Tato synchronizace dat se službami třetích stran (dále pouze synchronizace dat se službami) je pro funkcionalitu aplikace *xCalc* klíčová.

- Import paralelek
- Import studentů
- Zobrazení výsledku importu
- Přesunutí studenta do jiné paralelky
- Dodatečné přidání nového studenta
- Dodatečné přidání cvičícího
- Odebrání studenta z paralelky, resp předmětu
- Zobrazení stavu synchronizace dat se službami u paralelky
- Zobrazení stavu synchronizace dat se službami u předmětu
- Zobrazení stavu synchronizace dat se službami u studenta

2.3.2 Správa úkolů

Tato část webové aplikace je zaměřena na aktivity spojené se zadáváním, vypracováváním a odevzdáváním domácích úloh.

2.3.2.1 Cvičící

Cvičící jsou zodpovědní za zadání, rozdělení a údržbu jednotlivých úloh.

- Vytvoření nové úlohy
- Vytvoření nové úlohy při použití jiné jako šablony
- Úprava úlohy
  - Publikace úlohy
  - Skrytí úlohy

  - Úprava zadání
  - Změna parametrů (např.: možné programovací jazyky a maximální paměťové nároky řešení)
  - Změna nejzazšího termínu odevzdání úlohy
  - Zobrazení výstupu běhu studentovy výpočetní úlohy

2.3.2.2 Student

Student pomocí aplikace *xCalc* odevzdává zadané domácí úlohy napříč vícero předměty.

- Zobrazení seznamu všech studentových úloh
- Zobrazení všech aktuálních studentových úloh
- Zobrazení seznamu nových událostí
- Pdeslání řešení v podobě výpočetní úlohy k výpočtu
  - Výběr verze zdrojového kódu pro odeslání k výpočtu
- Odeslání výpočetní úlohy k testovacímu výpočtu

2.3.3 Zpracování výsledků

Jedná se o téměř nezávislou komponentu, jejíž úkolem je parsovat syrová výstupní data podle zadaného formátu a z těchto dat následně graficky zobrazit výsledky. Operace na datech a grafech jsou konfigurovány správcem, v případě aplikace *xCalc* se jedná o cvičícího.

2.3.3.1 Cvičící

- Změna definovaného formátu pro vstupní data
- Změna nastavení algoritmu pro generování grafického výstupu
- Uložení šablony
- Načtení ze šablony
- Změna odkazu na repozitář s vstupními a validačními daty

2.3.3.2 Student

- Zobrazení neupraveného výstupu výpočtu odeslaného řešení
- Zobrazení výstupu výpočtu odeslaného řešení (filtrovaný pouze na relevantní výstupní data)
- Zobrazení výstupu do předdefinovaných diagramů a grafů
- Stažení výstupu v textovém formátu
- Stažení formátovaných výstupních dat ve formátu CSV
- Nahrání souboru s lokálním výstupem (místo použití výstupu odevzdaného řešení)

Kapitola 3

Analýza existujících řešení

Tato kapitola se zabývá nalezením existujících aplikací potenciálně vhodných k rozšíření. Cílovou skupinou jsou nástroje pro tzv. průběžnou integraci („Continuous Integration“).

Existující aplikaci by bylo nutné rozšířit o funkcionalitu správy úkolů a komunikaci s ostatními službami.

3.1 Aplikace pro správu extezivních výpočtů

Primární funcionalitou těchto aplikací je zajištění splnění kvalitativních kritérií při vydání nové verze, jako je úspěšné dokončení testů, případně provedení nasazení do daného prostředí.

K analýze již existujících řešení byly vybrány webové aplikace *Gitlab CI* a *Travis CI*. Tato dvojice je doplněna aplikací pro správu úkolů Progtest.

3.1.1 Gitlab CI

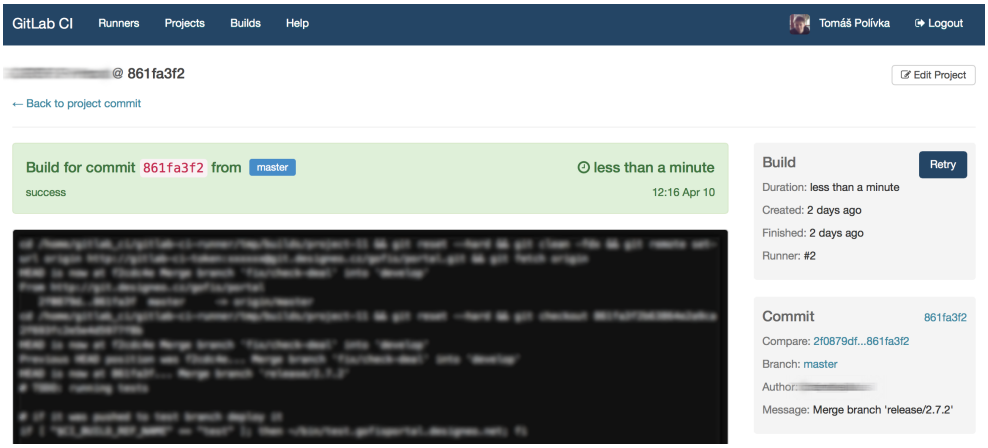
Gitlab CI je řešení, jehož primárním cílem poskytovat funkcionalitu typu průběžné integrace pro aplikace *Gitlab*.

Umožňuje autentifikaci pomocí aplikace *Gitlab*, který v procesu figuruje jako provider pro *OAuth2* autentizaci. Dále umožňuje synchronizaci zpracovávání signálů z Gitlabu a spouštění vytvoření úlohy například při commitu do dané větve.

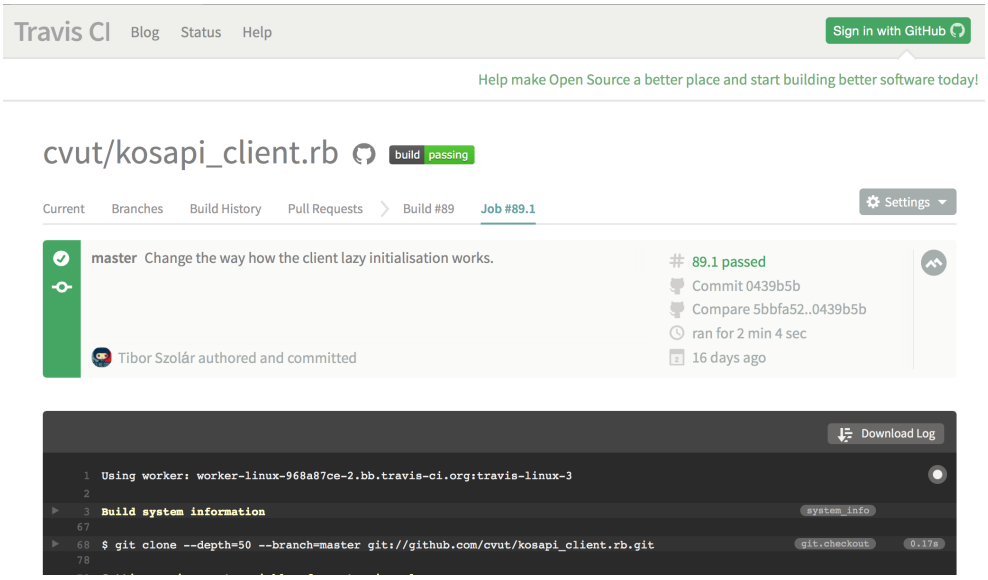
3.1.2 Travis CI

Jedná se o open-source projekt, zprostředkující služby typu průběžné integrace pro projekty na portálu *Github*. Zdrojové kódy jsou volně dostupné z repozitáře na Githubu[5].

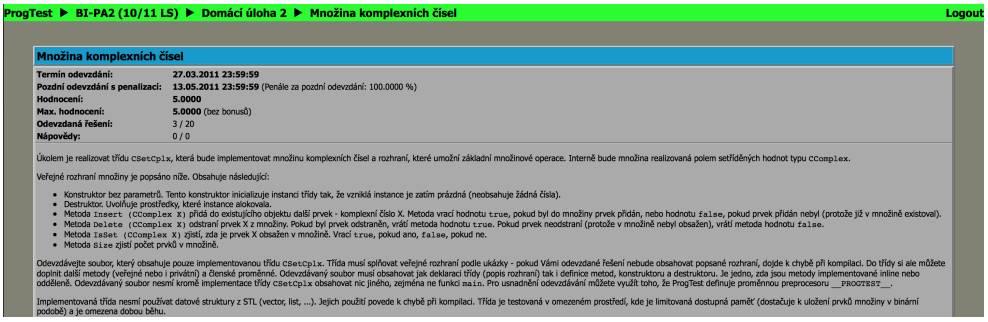
[5]Dostupný na `https://github.com/travis-ci/travis-ci`



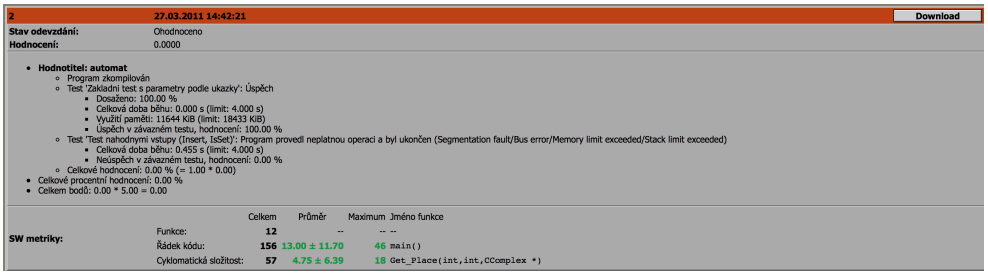
Obrázek 3.1: Ukázka aplikace *Gitlab CI*



Obrázek 3.2: Ukázka aplikace *Travis CI*



Obrázek 3.3: Ukázka zadání úkolu z aplikace *Progtest*



Obrázek 3.4: Ukázka výhodnocení odevzdané úlohu v aplikaci *Progtest*

Aplikace *Travis CI* existuje také jako webová služba, pomocí které je možné bezplatně využívat spouštění buildů u open-source projektů na webu Github. Pro privátní repozitáře je služba zpoplatněna.

Web disponuje velmi jednoduchým a líbivým uživatelským rozhraním 3.2.

3.1.3 Progtest

Aplikace *Progtest* je již delší dobu používána na ČVUT FEL a ČVUT FIT primárně pro výuku základních principů programování v jazyce C a C++. Aplikace se specializuje na kontrolu kvality a integrity odevzdaných zdrojových kódů. Odezvdané kódy bývají fragmenty většího programu a implementují rozhraní specifikováné v zadání úkolu, ke kterému bývá zpravidla přiložen příklad testovaných vstupních dat s vzorovými výstupy. Aplikce umožňuje i náhodné přiřazení úloh jednotlivým studentům.

Progtest poskytuje velké množství funkcionality požadované od aplikace *xCalc*. Jedná se hlavně o komunikaci s *KOSapi*, tedy o import studentů a cvičících. Odevzdaný kód je zařazen do fronty, zkompilován a spuštěn s omezeními například na velikost paměti. Narozdíl od požadavků pro *xCalc* jsou všechny výstupy běhu programu studentovi skryté, vidí pouze procentuální úspěšnost v jednotlivých testovaných bodech a případné penalizace za nesplnění daných podmínek.

3.2 Závěr

Ačkoliv aplikace Progtest z hlediska funkcionality splňuje většinu požadavků kladených na *xCalc*, jeho hlavním problémem je vysoká specializace směrem ke kontrole správnosti implementace. Narozdíl od toho *xCalc* by měl být spíše o výstupu algoritmů a jejich grafické reprezentaci. Navíc *xCalc* by měl umožňovat studentovi vybrat si svůj preferovaný programovací jazyk a případně i využít již existující knihovny pomocí balíčkovacích manažerů (například Composer pro jazyk PHP nebo Bundler pro RUBY).

Z výše uvedených důvodu bylo rozhodnuto o implementaci vlastního řešení, které bude komunikovat s upraveným *Gitlab CI*, jehož modifikace a implementaci runneru má na starosti vedoucí této práce Ing. Tomáš Bartoň.

Kapitola 4

Návrh UI

Tato kapitola je věnována dokumentaci průběhu návrh uživatelského rozhraní. Nejprve jsou vyprofilovány uživatelské persony. Následuje návrh základního layoutu aplikace. V další podkapitole je prezentován návrh struktury aplikace. Dále se autor věnuje popisu navržených komponent. Kapitola je zakončena popisem vytvořeného prototypu.

Při návrhu uživatelského rozhraní byl kladen velký důraz na využití obecně známých postupů a návrhových vzorů. Návrh staví na filosofii a konceptu frontendové knihovny *Twitter bootstrap*[6]. Při návrhu interakce s artefaktem je již u návrhu dbáno na dodržování pravidel použitelnosti podle Jacoba Nielsena [1].

Návrh se z důvodu rozsahu této práce orientuje především na UI pro správu úkolů.

Hlavním subjektem zájmu aplikace *xCalc* je pro studenta zobrazení aktuálního stavu jeho odeslaných řešení.

4.1 Persony

Tato kapitola definuje persony pro jednotlivé uživatelské role v aplikaci. Persona je hypotetický archetyp reálných uživatelů. Nejedná se o reálné osoby, ale reprezentují je v průběhu návrhu UI [2].

4.1.1 Administrátor

Jedná se muže ve věku 30 let. Bydlí v Praze a je zaměstnancem na ČVUT FIT. Mezi jeho dovednosti patří návrh informačních systému, programování v několika různých jazycích a výborná znalost UNIX systémů. Jeho koníčky jsou vývoj open-source projektů a turistika.

[6] Dostupná na `http://getbootstrap.com`

4.1.2 Cvičící

Jedná se muže ve věku 27 let. Mezi jeho znalosti patří hlavně hluboká teoretická znalost IT. V rámci svého doktorandského studia vede několik cvičení magisterského předmetu.

4.1.3 Student

Jedná se o muže ve věku 23 let. Bydlí na koleji s trvalým bydlištěm v severních Čechách. Je studentem magisterského studia. Na poloviční úvazek pracuje v malé IT firmě jako vývojář.

4.2 Hlavní layout

Návrh využívá obrazovku na celou šířku. Standardně bývá obsah u webových stránek omezený na užší centrovaný sloupec. V aplikaci *xCalc* ovšem není mnoho textu ke čtení a většinou se jedná o krátké informace o velké kvantitě.

4.3 Struktura aplikace

Jako příklad je na diagramu 4.1 znázorněna struktura aplikace z pohledu studenta.

4.4 Navržené komponenty

4.4.1 Hlavní navigace

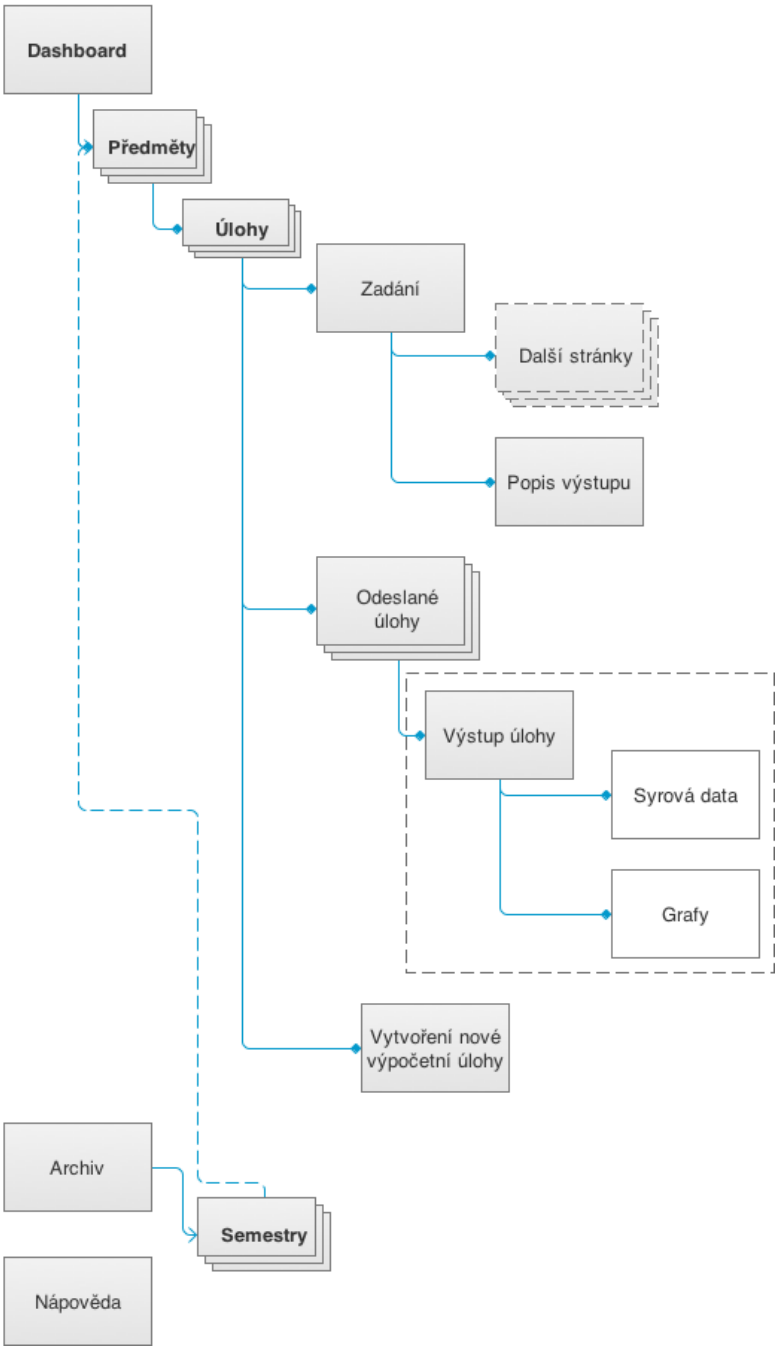
Hlavní navigace webu je fixovaná na horní okraj obrazovky. Její návrh je silně inspirován aplikacemi *Travis CI* a *Gitlab CI*. Je rozdělena na tři hlavní části.

Pro jednoduchou orientaci v aplikaci a z důvodu nízkého zatížení paměti uživatele je využit návrhový vzor drobečkové navigace [3]. Jejím obsahem jsou jednotlivé segmenty stromové struktury aplikace. Pro zjednodušení bude při procházení struktury vztahující se k aktuálnímu semestru položka semestru v drobečkové navigaci skrytá.

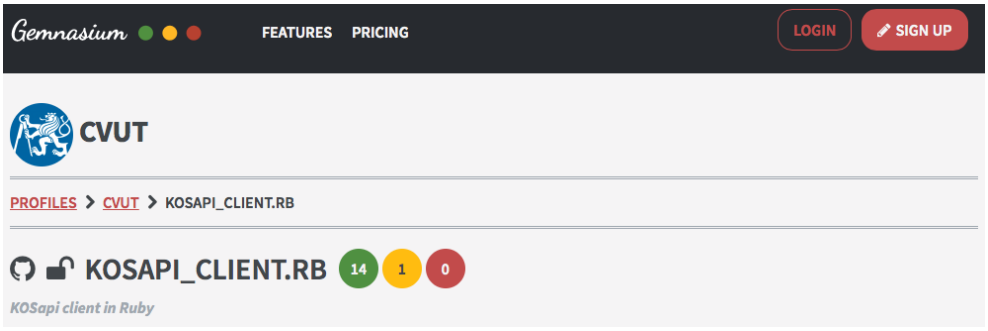
Dále hlavní navigace obsahuje základní informace o přihlášeném uživateli a tlačítko pro odhlášení. Samotnou akci odhlášení je nutné potvrdit ve vyskakujícím okně.

Pro zjednodušení a zpřehlednění UI jsou využívány ikony.

Z důvodu použitelnosti UI je v pravém rohu hlavní navigace vždy dostupné tlačítko nápovědy. Po jeho stisknutí jsou u elementů zobrazeny bubliny — standardně je tento způsob prezentace dodatečných informací nazýván *tooltip* — s nápovědou určenou k jednotlivým komponentám.



Obrázek 4.1: Návrh struktury aplikace z pohledu studenta



Obrázek 4.2: Ukázka komponenty semafor z webové aplikace Gemnasiun

4.4.2 Dashboard

Pod pojmem dashboard si můžeme představit jako reálný model klasickou nástěnku, na kterou jsou umístěny bloky se souhrnnými informacemi. Tato komponenta bude využívána k výpisu seznamu úkolů, resp. předmětů. Seznam úkolů je, pokud má uživatel dostatečnou úroveň oprávnění, doplněn o možné další akce, jako například v případě cvičícího odkaz na seznam studentů.

Blok pro úkol obsahuje všechny důležité informace. Nejdůležitější je časová osa, na níž je zobrazeno datum zadání a datum poslední možnosti odevzdání. Časová osa je doplněna o textovou informaci kolik času do posledního termínu odevzdání zbývá. Dále je u úlohy zobrazen název a semafor událostí (viz 4.4.3).

Na úvodní stránce jsou vždy vyfiltrovány pouze aktuální úkoly s možností zobrazení budoucích či historických úloh.

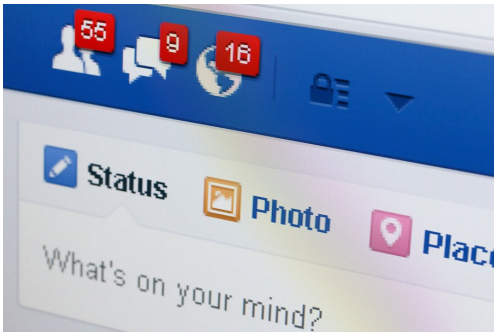
4.4.3 Semafor událostí

Myšlenka této komponenty je převzata z webové aplikace *Gemnasium*[7], která využívá analogii s konceptem semaforu na dopravní křižovatce ke znázornění závažnosti zastaralosti závislostí daného projektu na knihovnách třetích stran (viz obrázek 4.2).

Komponenta zobrazuje následující informace o aktuálním stavu odeslaných úloh:

- odeslané úlohy čekající ve frontě na zpracování
- právě zpracovávané úlohy
- úspěšně dokončená zpracování odeslané úlohy
- dokončená zpracování odeslané úlohy, které skončily chybou

[7] dostupné na adrese `https://gemnasium.com`



Obrázek 4.3: Ukázka zobrazení notifikací ve webové aplikaci *Facebook* [4]

Protože je nutné zobrazit čtyři typy informací, je potřeba standardní semafor rozšířit o další barvu. Barvy k jednotlivým typům informací jsou přiřazeny následovně:

- **šedá** - odeslané úlohy čekající ve frontě na zpracování;
- **žlutá** - právě zpracovávané úlohy;
- **zelená** - úspěšně dokončená zpracování odeslané úlohy;
- **červená** - dokončená zpracování odeslané úlohy, které skončily chybou.

Dále je zde problém s pořadím jednotlivých barev. Existují dvě možnosti:

1. pořadí určeno analogií s reálným semaforem (viz obrázek 4.5)
2. pořadí určeno směrem zleva do prava podle toho, v jakém pořadí jsou jednotlivé, notifikace aktivovány (viz obrázek 4.6).

Jako finální varianta byla zvolena, podle výsledku uživatelského testování, varianta druhá, tedy pořadí zachovávající směr dle životního cyklu výpočetní úlohy.

Semafor aplikace *xCalc* využívá podobný princip zobrazování notifikací, jako je používán v sociální síti *Facebook*[8] (viz obrázek 4.3).

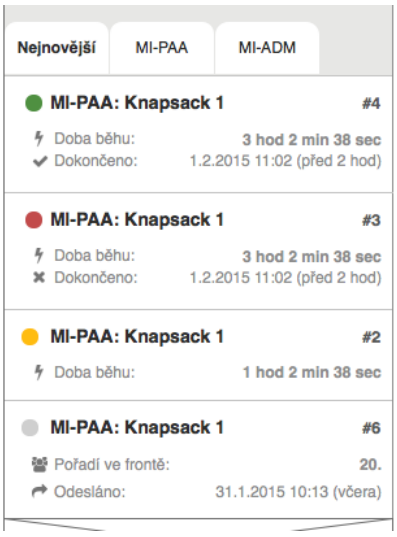
U čekajících a probíhajících úloh se jedná o informaci „statusu“ a zůstává viditelná po celou dobu její platnosti. Dokončená zpracování, ať už s úspěšným nebo chybovým výstupem, jsou prezentovány formou notifikací. Tedy pokud si uživatel zobrazí zprávu, notifikace bude odstraněna i ze semaforu.

Jednotlivé ukazatele v sobě zobrazují počet úloh v daném statusu nebo počet nezobrazených notifikací. Pokud pro daný typ žádná není, zobrazí se prázdné kolečko se šedým pozadím.

Obrázek 4.4: Semafor událostí ve stavu, kdy nejsou žádné události k zobrazení

Obrázek 4.5: Varianta semaforu řazená podle pořadí barev na reálném semaforu

Obrázek 4.6: Varianta semaforu řazená pořadí událostí v životním cyklu odeslaného rešení



Obrázek 4.7: Návrh tickeru

4.4.4 Ticker

Tato komponenta je zobrazena na každé stránce fixovaná na levou stranu obrazovky. Obsahem je stream („proud“) notifikací v reálném čase. Opět se jedná o inspiraci z webové aplikace sociální sítě *Facebook*. Vnější postranní panel (viz obrázek 4.7) obsahuje záložky, které fungují jako filtr pro zobrazované notifikace.

Každá notifikace obsahuje barvou stavového kolečka znázorněný stav do kterého se úloha přesunula (textový název stavu je zobrazen při přejetí myší přes stavové kolečko). Dále obsahuje informace specifické pro každý stav. Obzvláště důležitou informací pro studenta je pořadí ve frontě.

4.5 Správa úkolů

V sekci správy úkolů je nutné vyřesit případné problémy při vytváření úkolů, což je nejsložitější část celé této sekce. Zbytek obrazovek obsahuje pouze jednoduché výstupy a z pohledu interaktivity se jedná o jednoduché koncepty CRUD.

Zadání těchto úkolů vyžaduje nutnost zápisu složitějších kontrukcí, jako tabulek nebo vzorců. Autor se snažil vyhnout WYSIWYG editorům, neboť jejich použití není vždy optimální a při cílové skupině cvičícího je možné si dovolit použít značkovací jazyk. Autor původně uvažoval o volbě formátu *Mardown* [5], nicméně na doporučení vedoucího práce byl použit formát *Asciidoc*[9], který využívá konzistentnějších definicí a poskytuje kvalitní dokumentaci.

Z důvodů jednoduché opravy chyb a dohledání změn v zadání bude artefakt zachovávat historii změn v podobě revizí.

Cvičící i student bude mít možnost kofingurovat výpočetní úlohu různými parametry. V první iteraci vývoje bude využito standardního textového pole a manuální zadávání ve formátu YAML. V budoucnu bude možno toto pole nahradit za komplexnější a interaktivní systém zadávání.

4.6 Prototyp

Na základě analýzy byl vytvořen prototyp pomocí aplikace *Axure RP 7*[10]. Ukázky vybraných obrazovek jsou umístěny v příloze C.

[8] dostupné na adrese `https://facebook.com`

[9] Dostupné na `http://www.methods.co.nz/asciidoc/`.

[10] Dostupné na adrese `http://www.axure.com`.

Kapitola 5

Analýza a návrh backendu

Tato kapitola se zabývá analýzou požadavků z pohledu návrhu implementace, procesů a funkcionality. Dále se zabývá procesy v jednotlivých segmentech aplikace. Následuje návrh způsobu autentizace. Kapitola je zakončena návrhem API pro komunikaci s *Gitlab CI* a databázovým schématem.

5.1 Rozdělení implementace

Dle analýzy požadavků je backend rozdělen do tří téměř nezávislých sekcí - *studijní struktura*, *správa úkolů* a *komponenta pro zpracovávání výsledků*.

5.1.1 Studijní struktura

Tato sekce se zabývá analýzou a návrhem importu dat z *KOSapi* a návrhu struktury se systémy *Gitlab* a *Gitlab CI* včetně mapování importovaných struktur.

5.1.2 Import z KOSapi

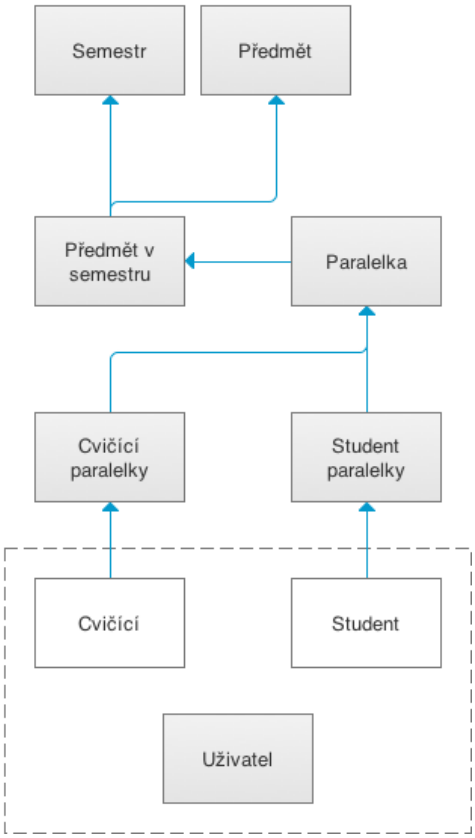
Jak je uvedeno v požadavcích, data semestrů, předmětů, paralelek a studentů mají být importována z celoškolního studijního informačního systému KOS[11]. Tento systém poskytuje vybraná data pomocí aplikačního rozhraní v podobě RESTful webových služeb [6]. Rozhraní je přístupné po autentizaci přístupovými údaji studenta. Pro aplikace je třeba vytvořit uživatelský účet na portálu pro správu aplikací[12]. Aplikaci je následně přidělen přístupový klíč.

Aplikace bude z *KOSapi* využívat následující zdroje:

Semestry [7] Stažení dat k semestru.

[11] Pro studenty a zaměstnance je dostupný na `https://kos.is.cvut.cz`.

[12] Portál je dostupný na `https://auth.fit.cvut.cz/manager/index.xhtml`.



Obrázek 5.1: Diagram vztahů mezi entitami ve studijní struktuře

Předměty [8] K získání entity *coursein* [9](dále „kurz v semestru“) z důvodu zjištění vyučujících (zkoušející, garanti, cvičící a přednášející).

Paralelky [10] Pro získání studentů paralelek.

5.1.2.1 Synchronizace s *Gitlab* a *Gitlab CI*

Při synchronizaci do externích aplikací je také potřeba dbát na návrh struktury uživatelských oprávnění, aby bylo možné splnit dané požadavky. V případě externích aplikací totiž nelze daný model oprávnění změnit tak, jak je tomu při implementaci vlastní aplikace.

V aplikaci gitlab je nutné umožnit následující operace:

- cvičící vídí do repozitářů vytvořených pro úlohy aplikaci *xCalc* jednotlivým studentům;
- umožnit volitelně studentům vidět pouze do svého repozitáře, nebo umožnit zobrazení obsahu repozitářů i ostatních studentů;

- zamezení studentovi manipulaci s repozitářem tak, aby nemohl znemožnit správné fungování aplikace *xCalc*;
- proces provádějící výpočet odeslaných úloh musí mít možnost stáhnout zdrojové kódy z přiřazeného repozitáře.

Následují tabulky zjednodušeně znázorňující strukturu oprávněnní v aplikaci *Gitlab* [11].

Akce/role v repozitáři	Vývojář	Vedoucí	Vlastník
Pull přes program git	✓	✓	✓
Stáhnout projekt	✓	✓	✓
Vytvořit žádost o merge request	✓	✓	✓
Vytvořit novou větev	✓	✓	✓
Přidat štítek	✓	✓	✓
Přidat nového člena		✓	✓
Přepsat nebo smazat štítek		✓	✓
Editovat vlastnosti repozitáře		✓	✓
Změnit viditelnost repozitáře			✓
Přesunoutí repozitáře[13]			✓
Smazat repozitář			✓

Tabulka 5.1: Struktura oprávnění pro repozitáře v aplikaci *Gitlab*

Akce/role ve skupině	Vývojář	Vedoucí	Vlastník
Prohlížení skupiny	✓	✓	✓
Úprava skupiny			✓
Vytvořit repozitář ve skupině		✓	✓
Spravovat členy skupiny			✓
Smazat skupinu			✓

Tabulka 5.2: Struktura oprávnění pro skupiny v aplikaci *Gitlab*

Důležitý je fakt, že pro uživatele, který je členem týmu repozitáře i skupiny, je pro repozitář použito nejvyšších oprávnění z těchto dvou rolí.

5.1.2.2 Navržená struktura

V aplikaci *Gitlab* bude vytvořen standardní uživatel[14] *xcalc-app*, jehož privátní klíč bude součástí konfigurace aplikace *xCalc*, která na jeho základě bude komunikovat s *Gitlabem*. Stejný klíč bude používat i *Gitlab CI* pro umožnění stažení zdrojového kódu odeslaného k výpočtu.

[14]Uživatel, který nemá administrátorská práva.

Pomocí uživatele *xcalc-app* bude v *Gitlab* vytvořena skupina, která bude reflektovat relaci „kurz v semestru“. Viditelnost skupiny je nastavena na hodnotu „privátní“. Název je zvolen tak, aby pro studenta zůstala práce v *Gitlabu* přehledná a repozitáře pro úlohy jednoduše identifikovatelné i při vícenásobném zapsání předmětu. Název bude mít následující formát:

<kód semestru> <kód předmětu> (5.1)

Uživatel *xcalc-app* se po provedení požadavku stává vlastníkem vytvořené skupiny, čímž mu náleží práva vlastníka i pro každý repozitář v této skupině. Díky tomuto faktu může aplikace vytvořit projekt v *Gitlab CI* a stáhnout zdrojové kódy projektu.

Do vytvořené skupiny jsou následně přidáni cvičící s rolí „vedoucí“, což umožní nahlížení do repozitářů studentů.

Repozitáře pro studenty jsou vytvářeny separátně pro každý předmět v daném semestru (repozitář *per* předmět *per* semetr). Do něj je následně přiřazen student s rolí „vývojář“. To mu dává pouze nutná minimální práva pro správu kódu a neomezuje nijak možnosti, které verzovací systém git nabízí (vytváření štítků, větví, spojování atd.). Zároveň mu nedává možnost destruktivních akcí, které by narušili synchronizaci mezi aplikacemi *Gitlab* a *xCalc*.

Největší výhodou tohoto návrhu je absence nutnosti využívání administrátorského účtu v *Gitlabu*. Zároveň se vyhýbá použití deployovacích klíčů.

Úroveň rozlišující jednotlivé paralelky nebyla do struktury zahrnuta, neboť není nutné omezovat práva cvičícím, aby nemohli nahlédnout do repozitářů ostatních studentů. Navíc zobrazení zdrojového kódu probíhá primárně přes odkaz v aplikaci *xCalc*. Další výhodou je menší počet vytvořených skupin na *Gitlabu*.

5.1.3 Správa úkolů

Tato sekce se zabývá všemi aktivitami spojenými se zadanými úlohami a k tomu vztahujícím se uživatelským rolím.

Aby bylo možné aplikaci využívat k co nejširší oblasti úloh, je třeba umožnit flexibilní systém nastavavování a předávání parametrů. V ideální případě, aby nebylo konfigurační data nutné přizpůsobovat pevné struktuře a při jejich změně nebylo nutné upravovat model aplikace. Zároveň je ale nutné pro zadávání úkolů studentům zakázat konfiguraci některých proměnných.

Pro ukládání parametrů bude využito takového datového typu nebo formy serializace, která umožňuje ukládání množiny párů klíč - hodnota i s možností zanoření. Tento požadavek bude jedním z kritérií při výběru databázového systému při implementaci.

5.1.3.1 Návrh formátu a struktury odesílaných dat

Kromě volitelných parametrů je třeba definovat parametry povinné, které přímo ovlivňují prostředí, ve kterém odeslané řešení poběží. Jedná se například o výběr vlastního prostředí nebo omezení velikosti paměti dostupné běhu programu. I množina těchto parametrů může být v budoucnu rozšířena.

Výsledkem je následující návrh struktury a zpracovávání parametrů. Příklady budou reprezentovány ve formátu YAML[15].

Následuje příklad nastavení vytvořené úlohy cvičícím:

```
containers:
  ruby-2.1:
    variables:
      cmd: ./paa %params
      build_cmd: bundle install
    constants:
      max-memory: 100
  c++:
    variables:
      cmd: rake paa %params
      build_cmd: make
    constants:
      max-memory: 100
variables:
  x: test-value
constants:
  configuration:
    a:
      - 1
      - 2
    b:
      - 10
      - 20
```

Následuje příklad nastavení, která jsou zpracována z formuláře pro vytváření výpočetní úlohy při jejím vytváření studentem.

```
container:
  id: ruby-2.1
  variables:
  build_cmd: bundle install --without test
variables:
  x: test-student
```

[15] Další informace a dokumentace dostupná na `http://yaml.org`.

Definujme množinu parametrů jako množinu, jejíž prvky jsou dvojice klíč - hodnota. Tou může být skalární hodnota či další množina parametrů. Hodnotu uloženou pod daným klíčem je možné získat pomocí operace *[]* s daným klíčem jako parametrem.

Nejprve definujme operaci *merge* se vstupními parametry množinou parametrů A a množinou parametrů B následovně:

pokud pouze v jedné z množin existuje klíč x s hodnotou y, je do výsledku zapsána hodnota y pod daným klíčem x;

pokud v množině A i v množině B existuje hodnota s klíčem x je do výsledku zapsána hodnota s klíčem x z množiny A.

Omezení na úpravu jednotlivých parametrů je reprezentováno rozdělením na podmnožiny z *variables* a *constants*. Hodnoty uvedené v podmnožině *constants* nemohou být již přepsány při aplikaci dalších konfigurací.

Množina parametrů *containers* u definice úlohy obsahuje informace, které budou použity k iniciální konfiguraci kontejneru pro spuštění úlohy. Obsahuje prvky jejichž klíči jsou unikátní identifikátory kontejnerů přístupných pro tuto úlohu. Při odeslání student vybere jeden z těchto kontejnerů a případně změní jeho volitelné parametry. Základními parametry jsou:

build_cmd Příkaz, resp. sekvence příkazů, která například v případě programu napsaném v jazyce C++ spustí kompilaci.

cmd Příkaz, resp. sekvence příkazů, která spustí implementovaný program. Umožňuje použití zástupného symbolu *%params*, který bude při provádění nahrazen za vygenerovanou sadu parametrů.

Sekce *configurations* obsahuje kofiguraci vytváření jednotlivých výpočetních úloh na straně *Gitlab CI*. Z těchto dat bude vytvořen kartézský součin a výsledná data budou předána jako parametry spuštěnému programu uvnitř kontejneru - dojde k nahrazení za zástupný symbol *%params*. Při použití této proměnné bude vytvořena samostatná výpočetní úloha pro každou konfiguraci. Pokud bude tento klíč chybět nebo obsahovat prázdnou hodnotu, bude vytvořena pouze jedna výpočetní úloha. Výše zmíněný zástupný symbol bude nahrazen za prázdný řetězec znaků.

Výsledná data odeslaná do aplikace *Gitlab CI* v případě kombinace výše uvedených konfigurací vypadají následovně:

```
container:
  id: ruby-2.1
  cmd: ./paa %params
  build_cmd: bundle install --without test
x: test-student
```

```
configuration:
  a:
    - 1
    - 2
  b:
    - 10
    - 20
```

5.1.4 Komponenta pro zpracovávání výsledků

Tato část aplikace je navrhována jako samostatná knihovna, neboť není nijak vázána na aplikaci *xCalc*. Zabývá se nejprve zpracováním syrových dat výstupu, a to z pohledu definice jejich struktury, kterou musí student následovat. Následuje návrh operací nad výsledky a podkapitola je zakončena analýzou existujících frontendových knihoven pro tvorbu grafů.

Komponenta není hlavním předmětem této práce a není tedy rozváděna tolik do hloubky. V první iteraci implementace je uvažováno s tím, že aplikace získá výstup z běhu výpočetní úlohy až po jeho skončení. Při implementaci složitějšího získávání výstupu v reálné času je nutné uvažovat o vytvoření mezistupně za použití moderních technologií, například pro správu logovacích zpráv pomocí systému *Elastic*[16] v kombinaci s dashboardovým systémem *Kibana*[17].

Autor se při návrhu snažil docílit nezávislosti na tom, zda bude proces zpracovávání probíhat na backendu nebo na frontendu.

5.1.5 Návrh definice formátu výstupu

Zpracování výstupu bude probíhat pomocí dále popsaného formátu, který bude následně převeden do regulárního výrazu. Formát bude zadáván jako textový řetězec a jako separátory budou použity mezery.

V první iteraci bude umožněno vytváření následujících datových typů:

integer - celé číslo;

float - desetinné číslo.

Tyto datové typy budou dekódovány pomocí nativních parserů jazyka, ve kterém bude probíhat výsledná interpretace.

Každá proměnná bude pojmenována řetězcem splňujícím regulární výraz *[a-z_]+*. Následuje příklad definice:

```
integer:id integer:instance_size integer:price
```

[16] Dostupné na `https://www.elastic.co`.

[17] Dostupné na `https://github.com/elastic/kibana`.

Tyto definice budou převedeny do jednoho regulárního výrazu a mapovaním, mapováním, použitým při následném přiřazování, do proměnných. Regulární výraz je postupně aplikován na všechny řádky výstupu. Řádky, které nesplňují vygenerovaný regulární výraz, jsou ignorovány.

Výsledkem zpracování dat bude pro každý validní řádek objekt s přiřazenými proměnnými dle daného pojmenování.

5.1.6 Návrh definice operací s výsledky

Zde je nutné uvažovat, jak velký by mělo provádění operací nad výslednými daty pomocí backendu dopad na výkonnost aplikace. Pokud by zpracovávání výsledků aplikaci příliš zatěžovalo, bylo by potřeba tuto funkcionalitu delegovat na frontend – do prohlížeče. Protože rozhodnutí nemůže být podloženo měřeními, snaží se autor dosáhnout co největší flexibility, která by v ideální případě mohla vyústit v dynamický systém rozklájící zátež mezi frontend a backend dle možností systému.

Aby byla umožněna co největší flexibilita této komponenty, budou operace nad daty definovány pomocí vlastního DSL. Díky tomu lze provádět i složitější operace nad strukturovanými daty.

Pro zobrazení dat je třeba umožnit seskupování výsledných řádků a jejich obsahu provádět agregační výpočty jako například minumum nebo relativní chyba. V prostředí budou dostupné následující proměnné:

results *(dostupné vždy)* Asociativní pole dat, zpracovaných ze syrového výstupu výpočetní úlohy.

solutions Asociativní pole dat, přiřazených k vstupním datům, jako například řešení dané instance problému.

5.1: Koncept formátu pro definici operací na výstupními daty

```
1  map_solutions :id
2
3  group :instance_size do
4    group :id do
5      relative_mistake :price, :valid_result
6    end
7
8    average :relative_mistake
9  end
10
11 graph :logarithmic, relative_mistake
```

Popis ukázky definice operací 5.1 Výstupem příkladu by měl být graf průměrné relativní chyby (metoda *graph* - řádek #11) dle velikosti instance. Nejprve jsou namapována výsledná data na refereční výsledky (metoda *map_solutions* - řádek #1). Poté jsou seskupeny podle velikosti instance (metoda *group* - řádek #3). Dalším krokem je vypočtení relativní chyby (metoda *relative_mistake* - řádek #5) pro každou instanci (metoda *group* - řádek #4). Posledním krokem je vypočtení průměrné relativní chyby pro každou velikost instance (metoda *average* - řádek #8).

Jelikož se jedná o DSL, nelze jej pouze interpretovat, ale je nutné ho nejříve zpracovat. Právě v při tomto zpracování by bylo možné, například za použití kompilátoru *Opalrb*[18], generovat kód v javascriptu. Ten by mohl být interpretován přímo v prohlížeci.

Výsledná podoba DSL bude specifikována v implementační části, neboť může být přizpůsobena možnostem, které jazyk *RUBY* poskytuje, a potřebami úloh. Cílem by nemělo být vytvoření kompletního jazyka, ale příprava komponenty a implementace základní funkcionality s možností případného rozšíření v budoucnu.

5.1.7 Knihovny pro použití v pro generování grafů

Pro generování grafů existuje mnoho frontendových knihoven. Některé z nich umožňují také vykreslení grafu do statického obrázku na backendu pomocí knihovny *phantomjs*[19]. Autor pro implementaci první iterace použije knihovnu *Highcharts*[20], která je zaměřena především na tvorbu grafů. Pro *Hightcharts* existuje i knihovna[21] pro konfiguraci pomocí *RUBY*. V budoucnu může být případně DSL jazyk pro definici operace rozšířen natolik, že by umožňoval využití i dalších knihoven, které se zabývají i tvorbou složitějších diagramů, jako například knihovna *d3js*[22].

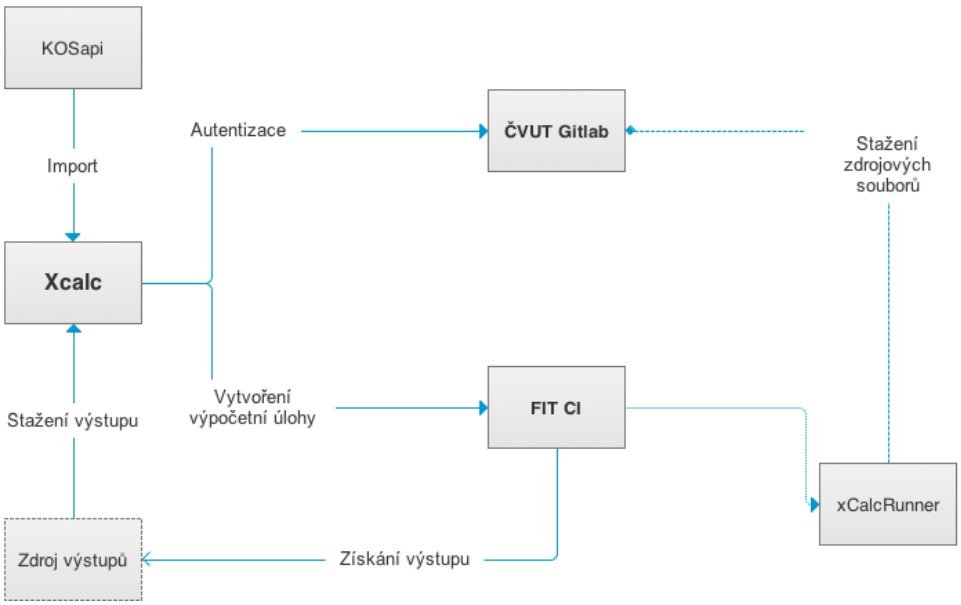
5.2 Integrace s ostatními systémy

Na diagramu 5.2 je zobrazeno základní schéma integrace s ostatními službami.

5.3 Autentizace

Pro autentizaci studentů a cvičích je na ČVUT FIT možno využít několika systémů, jako například *LDAP* nebo *Shibboleth*. Fakultní *Gitlab* využivá pro přihlašování studentů právě *LDAP* s tím, že zachovává možnost vytváření klasických lokálních účtů.

[18] Dostupný na `http://opalrb.org`.
[19] Dostupné na `http://phantomjs.org`.
[20] Dostupné na `http://www.highcharts.com`.
[21] Dostupná na `https://github.com/PerfectlyNormal/highcharts-rails`.
[22] Dostupná na `http://d3js.org`.



Obrázek 5.2: Diagram komunikace mezi jednotlivými systémy

Aplikace *xCalc* potřebuje pro vytváření repozitářů a přiřazování uživatelů tyto uživatele identifikovat. Proto bylo jako autentizační metoda vybráno přihlašování pomocí aplikace *Gitlab*, který od verze 7.7 umožňuje i správu aplikací, které ho využívají jako autentizační systém. Autentizace pomocí *Gitlab* umožňuje použití mapování uživatelů *Gitlabu* na ty získané z *KOSapi* pomocí emailové adresy. Uživatelé se do *Gitlabu* přihlašují pomocí *LDAP* loginu. Zároveň umožňuje vytvoření lokálních účtů a tím jednoduše umožní přístup do aplikace *xCalc* i externím uživatelům.

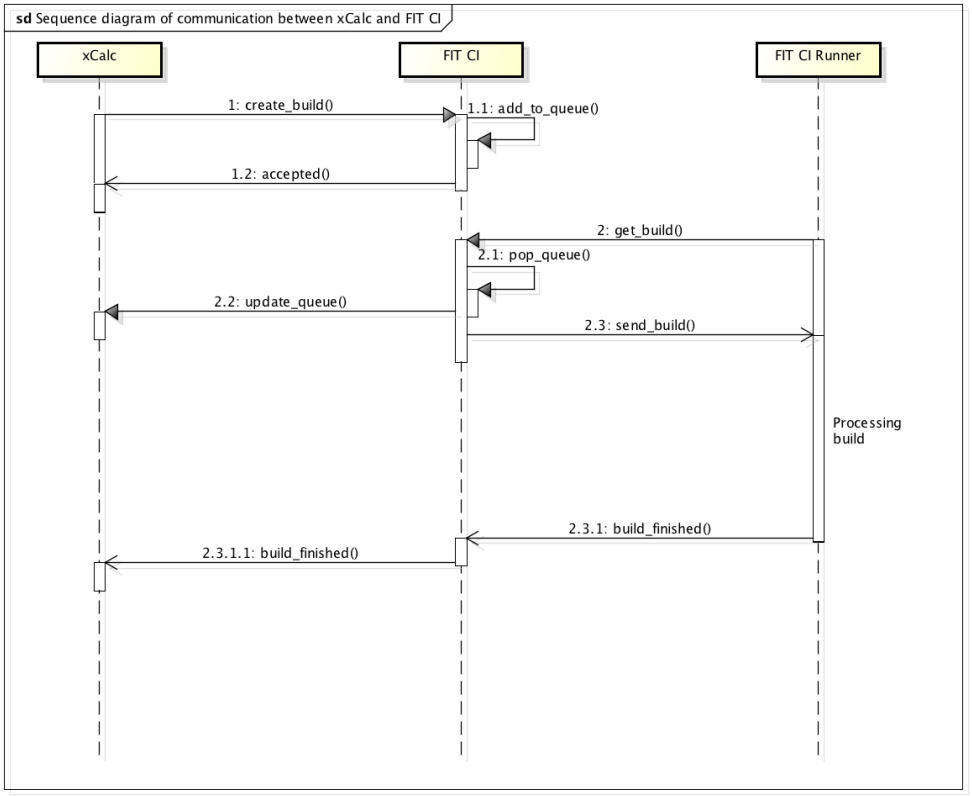
5.4 Návrh API pro komunikaci s *FIT CI*

Komunikace s API *FIT CI* bude rozdělena z pohledu aplikace do dvou bloků. První část bude implementována do použitého klienta *FIT CI* a bude zajišťovat komunikaci, kdy aplikace *xCalc* kontaktuje *FIT CI*. Druhá bude zajišťovat zpracovávní požadavků odeslaných z *FIT CI* do aplikace *xCalc*.

Všechna komunikace probíhá ve formátu JSON.

5.4.1 Komunikace vyvolaná aplikací *xCalc*

Tato komunikace se sestává pouze z jedné metody, a to vytvoření nové výpočetní úlohy do aplikace *xCalc*.



Obrázek 5.3: Sekvenční diagram komunikace *xCalc* a *FIT CI*

5.4.1.1 Vytvoření nové výpočetní úlohy

Aby bylo možné využít velkou část již hotové implementace na straně *FIT CI*, jsou odeslána při vytváření výpočetní úlohy stejná data [12] a je vygenerováno speciální ID pro identifikaci výpočetní úlohy při aktualizaci informací ze strany *FIT CI*.[23]

```
POST xcalc/commits
{
  project_id: integer
  project_token: string
  submit_hash: string
  settings: {
   ...
  }
  data: {
```

[23]Tato metoda musela být dvakrát refaktorována, aby se přizpůsobila změnám při upgradu aplikace *FIT CI* a nekonzistenci v dokumentaci.

```
    ...
  }
}
```

settings *(Povinný)* Obsahuje serializovanou množinu parametrů, která je výstupem z procesu detailně popsaného v 5.1.3.1.

submit_sha *(Povinný)* Unikátní identifikátor automaticky vygenerovaný pro každou výpočetní úlohu vytvořenou pomocí aplikace *xCalc*.

5.4.2 Komunikace vyvolaná aplikací *FIT CI*

Díky použití oboustranně inicializované komunikace je možné se vyvarovat nutnosti periodického dotazu na *xCalc* stav systému. Za benefit je možné považovat fakt, že zásluhou toho bude mít aplikace přípravenu strukturu na rozšiřování API, které by umožňovalo možnost implementace nástrojů využívajících funkcionality aplikace *xCalc*.

Tato komunikace se sestává ze dvou metod. První zasílá aplikaci *xCalc* infomaci o spuštění výpočetní úlohy a aktuálním stavu fronty v *FIT CI*. Druhá notifikuje *xCalc* o ukončení procesování výpočetní úlohy spolu s infomacemi o statusu s jakým byl výpočet ukončen.

5.4.2.1 Notifikace o spuštění výpočetní úlohy a aktualizace fronty

```
PUT service/queue
{
  'time': <timestamp>
  'started': {
    'id': <submit_sha>
  },
  'queue': [
    <submit_sha>,
    <submit_sha>,
    ...
  ]
}
```

time *(Povinný)* Čas spuštění výpočetní úlohy.

started.id *(Povinný)* Identifikátor spuštěné výpočetní úlohy.

queue *(Povinný)* Pole reprezentující frontu výpočetních úloh na *FIT CI*. Položka s indexem 0 je první v řadě a bude zpracována jako následující.

5.4.2.2 Notifikace o ukončení výpočetní úlohy

```
POST service/build-finished/<submit_sha>
{
  'time': <timestamp>
  'status': (success|fail),
  'cause': (unknown|exceed-memory-limit|exceed-time-limit|segfault)
}
```

time *(Povinný)* Čas, kdy byl výpočet dokončen.

status *(Povinný)* Status, zda výpočet byl ukončen úspěšně nebo s chybou.

cause *(Povinný)* Enumerátor příčiny chyb. Hodnoty jsou pouze předběžné a i v budoucnu budou omezeny faktem, že ne vždy je možné určit příčinu selhání.

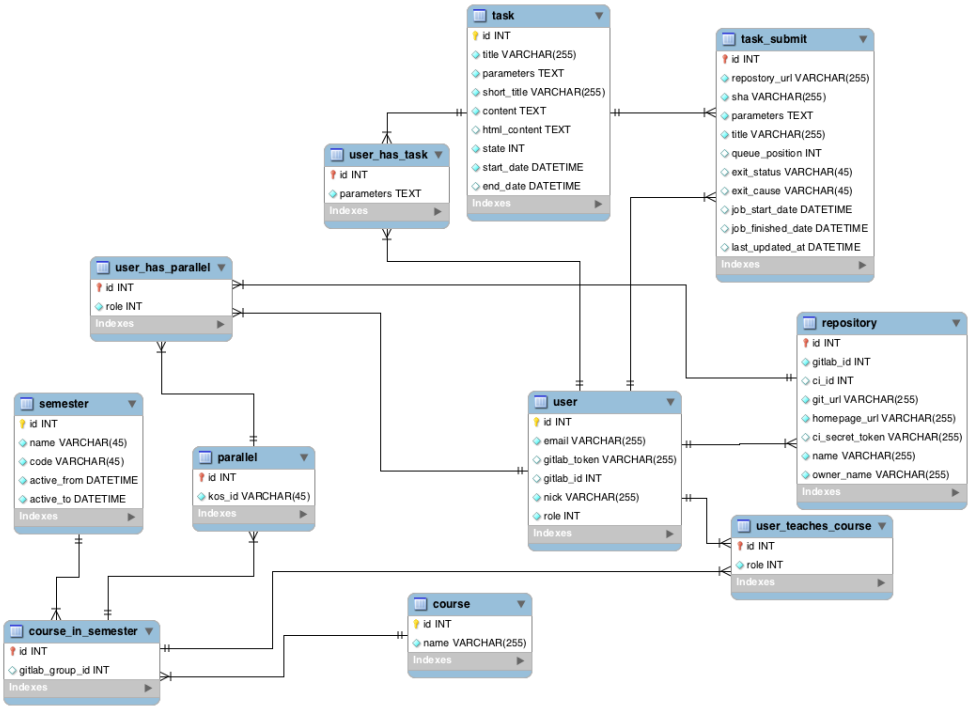
5.5 Databázový systém

Při výběru databázového systému autor s přihlédnutím ke struktuře entit v aplikaci uvažoval pouze mezi objektově-relačními databázemi. Na základě velmi pozitivní zkušenosti a několikaleté praxe vybral autor databázový systém *PostgreSQL*[24]. Od verze *9.1* tento databázový systém splňuje daný požadavek (viz 5.1.3) pro ukládání množin dvojic klíč – hodnota. Pro tyto potřeby je použit datový typ *hstore* [13], který framework *Ruby on Rails* od verze *4* nativně podporuje.

5.6 Databázové schéma

Databázové schéma aplikace je zobrazeno na diagramu 5.4. Tento model byl použit při návrhu funkcionality a není dokonalým obrazem reálného schématu. Důvodem je přístupem k vytváření entit u dabázové vrstvy v *Ruby on Rails*. Zároveň některé knihovny si automaticky v dabaázi vytvoří strukturu nutnou pro svou funkcionalitu. Takovou knihovnou je například *Devise*, která využívá tabulky *identities* pro správu přihlášených identit jednotlivých uživatelů.

[24] Dostupný na `http://www.postgresql.org`.



Obrázek 5.4: Databázové schéma aplikace

Kapitola 6

Realizace

Tato kapitola popisuje postup implementace webové aplikace. Charakterizuje přístup, jaký autor k řešení daného požadavku zvolil, a popisuje technologie nebo knihovny třetích stran, které byly použity při implementaci.

6.1 Programovací jazyk

Pro vývoj aplikace byl po konzultaci s vedoucím práce zvolen programovací jazyk RUBY[25]. Jedná se o plně interpretovaný open-source objektový jazyk. Klade důraz na minimalizaci kódu za účelem zvýšení jeho čitelnosti a snížení doby, kterou potřebuje programátor při zorientování se ve zdrojovém kódu cizího projektu. S lehkou nadsázkou lze říci, že zdrojový kód napsaný v *RUBY* — pokud dodržuje doporučené konvence — může sloužit i jako samotná dokumentace.

6.2 Vývojový framework

V roce 2014 byl při použití programovacího jazyka *RUBY* jasnou volbou framework pro webové aplikace *Ruby on Rails*[26]. Jedná se o framework využívající struktury MVC [14], jehož první verze byla vydána v roce 2005. Filosofie frameworku je založena na následujících principech [15]:

***DRY* - Don't Repeat Yourself** Jedná se o způsob vývoje software, který říká, že „každá znalost musí mít jednu a jednotnou reprezentaci v systému". Pokud se vývojář vyvaruje duplikaci zdrojového kódu, dochází tím k zvýšení udržitelnosti, zjednodušení rozšiřitelnosti a zmenšení pravděpodobnost výskytu chyb.

[25] Další informace a návod na instalaci dostupný na adrese `https://www.ruby-lang.org/en/`.

[26] Další informace dostupné na adrese `http://rubyonrails.org`.

***CoC* - Convention Over Configuration** Princip říká, že nejjednodušším přístupem k implementaci ve webové aplikaci je dodržování postupů a konvencí, oproti nutnosti specifikování každé maličkosti pomocí dlouhých konfiguračních souborů.

6.2.1 Správa závislostí

Standardním správcem balíčků a závislostí je pro framework *Ruby on Rails* *Bundler*[27], který umožňuje jednoduchou instalaci, správu kompatibility a update závislostí definovaných v souboru *Gemfile*.

6.2.2 Rozhraní pro terminálové příkazy

Pro administraci aplikace a provádění pokročilých operací používá *Ruby on Rails* utilitu *Rake* [16]. Jedná se o alternativu utility *make* pro *UNIX* systémy. Oproti *make* umožňuje *Rake* definici příkazů v mnohem více uživatelsky přívětivém formátu a využití flexibility programovacího jazyka *RUBY*.

6.2.3 MVC v Ruby on rails

Ruby on Rails standardně používá svou vlastní databázovou vrstvu, která je implementována podle návrhového vzoru *active record*. Jednotlivé entity jsou reprezentovány třídami, které jsou potomky základní entity[28]. Tyto entity ovšem přímo nedefinují jednotlivé atributy, které jsou automaticky získané z vlastní struktury databáze. Soubory s těmito třídami jsou umístěny ve složce *app/models*.

K vytvoření struktury jsou používány migrace, které jsou standardní součástí této databázové vrstvy. Jedná se o třídy s daným rozhraním, které jsou potomky základní migrační třídy [29]. Soubory jsou umístěny ve složce *db* a vytváření probíha spuštěním skriptu *rails migration create <název>*. Název souboru je prefixován časovým razítkem[30], podle kterého je pak určeno pořadí, v jakém se jednotlivé migrace provádí. Zároveň je časové razítko používáno jako unikátní identifikátor migrace, podle kterého je určeno, zda daná migrace již proběhla.

6.2.4 Controller

Jedná se o třídu jejíž primární funkcí je zpracování příchozích požadavků, provedení přiřazených aktivit a výběr view odpovědi k odeslání. Bývá zvykem, že každá veřejná metoda v controller třídě odpovídá na jeden typ požadavků. Při příchozím požadavku je potřeba zjistit cílovou akci, tedy jakou metodu

[27] Dostupné na *http://bundler.io*.
[28] Základní třídou entity je *ActiveRecord::Base*.
[29] Základní třídou entity je *ActiveRecord::Migration*.
[30] v originále - unix timestamp.

controlleru spustit. O to se stará směrovací komponenta, jejíž konfigurace se nachází v souboru *config/routes.rb*, která obsahuje mapování url požadavku na controller a jeho akci. Metody controlleru většinou tvoří skupinu týkají se operací nad jedním datovým zdrojem.

6.2.5 View

Tato vrstva spravuje výstup dat, která jsou následně odeslána jako odpověď na příchozí požadavek. U webový aplikací se ve valné většině jedná o odpověď ve formátu *HTML*. Ten není vykreslen manuálně, ale pomocí šablonovacího systému, který snižuje opakování kódu, umožňuje vnořování jednotlivých šablon, a tím i zlepšení jejich znovupoužitelnosti. Standardním formátem ve frameworku *Ruby on Rails* je *ERB* [17], ale při implementaci této práce byl zvolen formát *HAML*[31], který poskytuje možnost tvorby jednodušších a kratších zdrojových kódů šablony. *HAML* používá například i aplikace *Gitlab*.

6.2.6 Použitá verze

V první fázi byla aplikace *xCalc* vyvíjena na verzi *4.1*. Během vývoje došlo k upgrade na nově vydanou verzi *4.2*[32], neboť přinesla mnoho přínosných inovací [18]. Pro vývoj byla nejdůležitější nativní podpora cizích klíčů v modelu a datového typu *hstore* objektově relačního databázového systému *PostgreSQL*[33].

6.2.7 Alternativní varianta

V době začátku implementace (rok 2014) se pro *RUBY* objevil nový framework *Volt*[34]. Tento framework přináší nový přístup k implementaci webových aplikací. Díky flexibilitě programovacího jazyka a použití knihovny *Opal*[35], která umožňuje kompilaci *RUBY* do *javascriptu*, je programován backend i frontend simultálně. *Volt* nebyl použit hlavně z důvodu, že se jedná o mladý framework a je zatím cílen hlavně na menší projekty.

6.3 Vývojové prostředí

Tato sekce popisuje nástroje a postupy použité při implementaci.

[31] Dostupné na `http://haml.info`.
[32] Vezre 4.2 byla vydána 19. prosince 2014.
[33] Výběr databázového systému je zdokumentován v 5.6.
[34] Další infomrace dostupné na `http://voltframework.com`.
[35] Další informace dostupné na `http://opalrb.org`.

6.3.1 Git

K verzování zdrojového kódu byl použit program *git*. Jedná se o distribuovaný verzovací systém, který umožňuje nelineární vývojové postupy a udržitelný kód, i pokud více programátorů pracuje na stejném projektu. Tento systém byl vyvinut pro správu kódu Linuxového jádra a v roce 2015 je velmi rozšířený hlavně v open-source komunitě, která se koncentruje kolem webového portálu *Github*[36].

6.3.1.1 Git-flow

Jedná se o soubor doporučení a „best practices“ pro logickou správu větví v *git* repozitáři během průběhu životního cyklu projektu. Tento koncept byl aplikován při vývoji aplikace *xCalc*. Základní větví repozitáře je *master*, jehož obsahem by měl být vždy stabilní poslední verze kódu. V této větvi se vytváří štítky, které označují jednotlivé verze aplikace. Hlavní větví pro vývoj je *develop*. Každá funkcionalita (resp. logický blok funkcionalit) je implementována ve vlastní větvi, které jsou pojmenovány dle formátu *feature/<název_funkcionality>*. Každá větev je pro funkcionalitu vytvořená z *develop* a při dokončení je do ní zakomponována zpět.

6.3.2 Rubymine

Jako vývojový editor byl zvolen program *Rubymine*[37]. Důvody byly hlavně široká podpora funkcionality *Ruby on Rails*, a to včetně doplňování některý dynamicky generovaných částí kódu, a autorova předchozí zkušenost s tímto IDE.

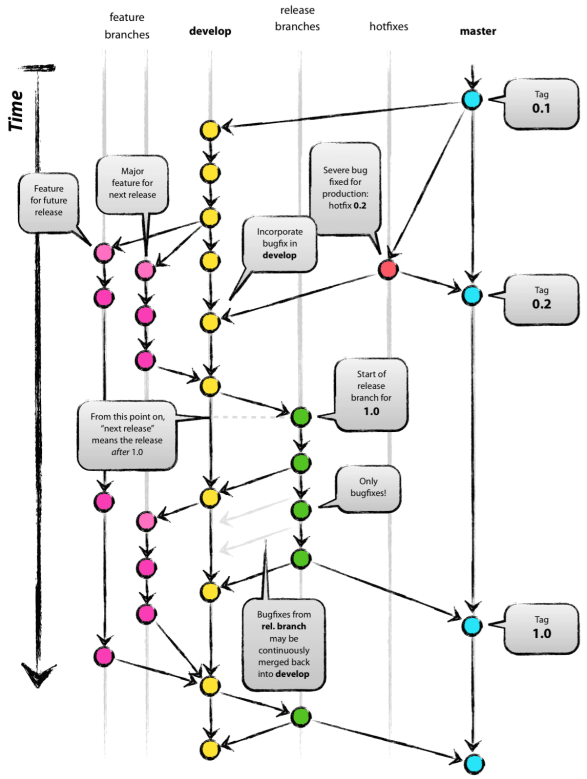
6.3.3 Lokální instance aplikace

Z důvodu, že *xCalc* využívá pro autorizaci aplikaci *Gitlab*, je nutné mít tento systém k dispozici i při lokálním vývoji. Použití produkčního systému k autorizaci by přinášelo mnoho omezení. Autor používá při vývoji virtuální webový server. Jako virtualizační nástroj byl použit *Virtual box*. Virtuální server beží na linuxové distribuci *Ubuntu*. Na tomto serveru běží pro potřeby školy upravené aplikace *Gitlab* a *Gitlab CI*, které jsou interně nazývány *ČVUT Gitlab* a *FIT CI*[38]. K lokálnímu spuštění aplikace *xCalc* je použit do *RUBY* vestavěný webový server *WebBrick*. Komunikace s aplikacemi ve virtuálním serveru je umožněna pomocí přidaných záznamů do lokálního souboru */etc/hosts*, který překládá nastavené domény na IP adresy prioritně před dotazem na DNS server.

[36] Dostupné na `https://github.com`.

[37] Více informací dostupných na `https://www.jetbrains.com/ruby/`.

[38] Zdrojové kódy jsou dostupné na `https://github.com/jirutka/gitlabhq` resp. `https://gitlab.fit.cvut.cz/xcalc/fit-ci`.



Obrázek 6.1: Grafické schéma vývoje projektu pomocí konceptu *git-flow* [19]

6.1: Ukázka záznamu mapování *IP - doména* ze souboru */etc/hosts*

```
# Gitlab & Gitlab CI – virtual
192.168.56.101 gitlab.local ci.gitlab.local
```

6.4 Implementace

Tato sekce se věnuje postupu implementace aplikace *xCalc*. Autor se při použití knihoven a přístupům k implementaci jednotlivých funkcionalit silně inspiruje u open-source zdrojových kódů. Za hlavní zdroj inspirace je možné označit právě kódy aplikace *Gitlab*.

6.4.1 Základní struktura webové aplikace

Základní struktura je znázorněna v 6.2. Následuje podrobnější popis funkce a obsahu vybraných složek.

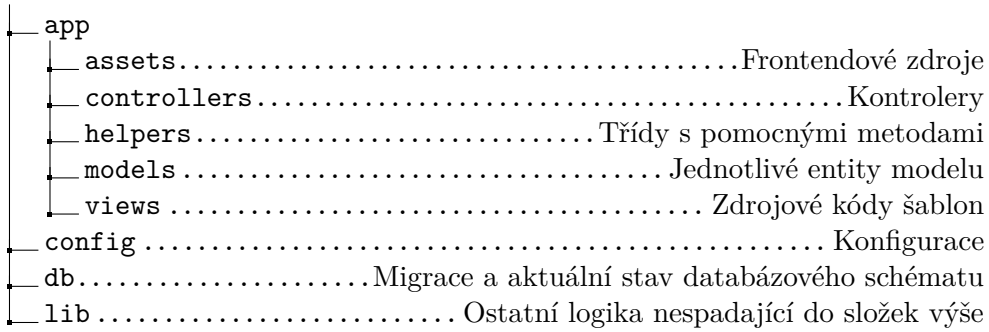


Diagram 6.2: Struktura souborů projektu

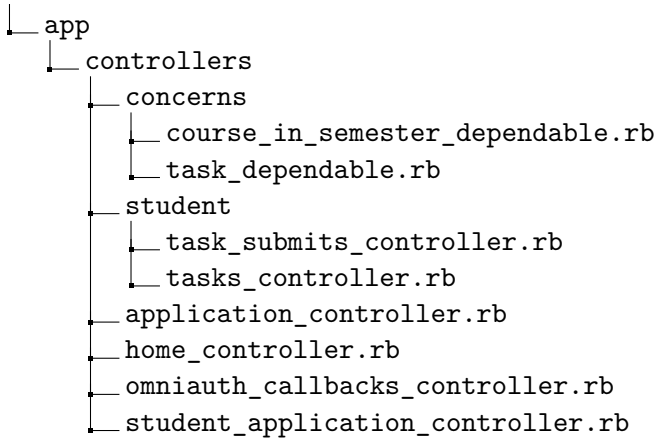


Diagram 6.3: Struktura zdrojových souborů kontrolerů

6.4.1.1 Kontrolery

Kontrolery jsou rozděleny do jednotlivých modulů, čímž kopírují různé pohledy do aplikace dle uživatelské role. Pro integraci základní funkcionality mají kontrolery z modulů stejného předka, který sám má za předchůdce základní aplikační kontroler. Na diagramu 6.3[39] je ukázka struktury souborů vztahujících se ke kontrolerům.

omniauth_callbacks_controller.rb Tento kontroler obsahuje koncové body pro zpracovávání požadavků spojených s autentizací.

application_controller.rb Jedná se o základní kontroler, který je předkem všech ostatních kontrolerů. Obsahuje inicializaci zdrojů ACL potřebných pro

[39] Ukázka z důvodu názornosti obsahuje pouze obecné kontrolery, a ty které se vztahují k pohledu studenta.

autorizaci a vyžaduje pro všechny akce přihlášeného uživatele (kromě akcí týkající se samotného přihlášení a přihlašovacího okna).

student_application_controller.rb Tento kontroler je předkem všech kontrolerů pro zpracovávání akcí z pohledu studenta.

student Složka obsahující všechny kontrolery pro studentův pohled do aplikace.

***concerns* (koncerny)** Jedná se o moduly určené k implementaci sdílené funkcionality. Jde o flexibilnější řešení než pomocí standardní dědičnosti tříd, neboť *RUBY* nepodporuje vícenásobnou dědičnost. Oproti té je modulů možné ve třídě použít i více. Zároveň umožňují využívat i lokálně definovaná DSL[40]. Ačkoliv základní modul *ActiveSupport::Concern* [20] se nachází v jmenném prostoru komponenty databázové vrstvy frameworku, je využíván i u ostatních typů objektů. Pro vytvoření koncernu stačí do příkladu (viz 6.2) přidat metody a nebo využít bloku *included*, který je automaticky proveden po zahrnutí koncernu do cílové třídy.

6.2: Ukázka struktury koncernu

```
module MyConcern
  extend ActiveSupport::Concern

  included do
        ...
  end
end
```

6.4.1.2 Pomocné metody

V těchto souborech jsou definovány pomocné metody určené pro použití v šablonách. Jejich učelem je snížení přítomnosti logiky přímo ve zdrojových souborech šablony.

6.4.1.3 Frontendové zdroje

Struktura frontendlových zdrojů je detailně popsána v podkapitole 6.4.12.

6.4.1.4 Konfigurace

Všechny inicializace konfigurovatelných objektů a samotného frameworku *Ruby on Rails* jsou umístěny ve složce *config*.

[40] Například u kontrolerů se jedná o filtr *:before_action <metoda>*, který před zavoláním akce provede uvedenou metodu.

Prostředí Soubory v této složce jsou používány ke konfiguraci frameworku *Ruby on Rails* pro aktuálně aktivní typ prostředí. Základními prostředími jsou *development*, *test* a *production*.

Zavaděče V souborech této složky jsou definované inicializace a nastavení konfiguračních hodnot použitých knihoven. Zavaděče jsou prováděny v pořadí dle jména ve vzestupném pořadí.

Settingslogic Standardní konfigurační mechanismus frameworku *Ruby on Rails* je v aplikaci *xCalc* rozšířen knihovnou *settingslogic*[41], která umožňuje definici konfiguračních proměnných pomocí speciálního konfiguračního souboru *config/settings/application.yml* ve formátu YAML.

6.4.1.5 Lokálně implementované knihovny

Funkcionalita, která logicky nespadá do žádné dříve definované složky, je umístěna v *lib*. Tento adresář není standardně automaticky načítán a bylo nutné jej registrovat v souboru *config/application.rb*(viz 6.3).

6.3: Registrace adresáře *lib* pro automatické načítání

```
...
config.autoload_paths += Dir[
  "#{config.root}/lib",
  "#{config.root}/lib/**/"
]
...
```

Obsah této složky je pak rozdělen na moduly, které již implementují přímo danou funkcionalitu. Obsahem může být například knihovna, která není nainstalována pomocí balíčkovacího systému, ale pomocí manuálního přidání zdrojových kódů. V aplikaci *xCalc* bude tímto způsoben například implementována komponenta pro import dat pro studijní strukturu.

6.4.2 Směrování

Směrování v aplikaci má za úkol přidělit zpracování přijatého požadavku na určitou url adresu správne metodě kontroleru. Následuje popis jednotlivých sekcí souboru *config/routes.rb*:

6.4: Definice akce která bude zobrazena při požadavku na adresu obsahující pouze doménu aplikace

```
root 'home#dashboard'
```

[41] Dostupná na `https://github.com/binarylogic/settingslogic`.

```
...
```

6.5: Ukázka definice směrovacích pravidel rozdělených podle uživatelských pohledů v aplikaci

```
...
namespace :student do
  resources :semester, only: [], path: "" do
    resources :course, only: [], path: "" do
      resources :tasks, only: [:show] do
        resources :task_submits, only: [:new, :create, :show], path: :submits
      end
    end
  end
end

namespace :teacher do
  resources :semester, only: [], path: "" do
    resources :course, only: [], path: "" do
      resources :tasks do
        resources :user, path: :students do
          resources :task_submits, only: [:index, :show], path: :submits
        end
      end
    end
  end
end
...
```

6.6: Registrace metod pro zpracovávání požadavků při autentizaci

```
devise_for :users,:controllers =>{omniauth_callbacks:'omniauth_callbacks'}
```

Zde i přes doporučení dokumentace zvolil autor hlubší zanoření jednotlivých *resources* [21][42] – „zdrojů“. Díky tomu je možné získat logicky správnou url, která reprezentuje místo ve stromu struktury aplikace (například */student/B142/MI-PAA/task/1*). Tato struktura také umožňuje načítat jednotlivé entity automaticky v kontroleru. To je realizováno pomocí koncernů (viz 6.4.1.1). Například pro automatické načítání entity „kurz v semestru“ stačí přidat koncern *CourseInSemesterDependable* a před začátkem vykonávání cílové metody je tato entita automaticky načtena.

```
module CourseInSemesterDependable
```

[42] Diskuze k tématu dostupná na [22] a [23].

```
  extend ActiveSupport::Concern

  included do
    before_action :set_course_in_semester

    protected

    def set_course_in_semester
      @course_in_semester = CourseInSemester
        .find_by_route_param(params)
    end
  end
end
```

Negativním efektem je výskyt dlouhých názvů jednotlivých cest s mnoha parametry. Neboť entity kopírují stromovou strukturu aplikace, je vždy možné získat všechny parametry z jejích dat nebo pomocí objecktů v relaci s danou instancí. Převedení entity na parametry, které je nutné předat generátoru url adres, obstarává metoda *routerize*, kterou implementují entity podílející se na tvorbě struktury aplikace. Díky stromové struktuře je možné využívat tuto funkci rekurzivně. Parametrem metody je namespace, který určuje zda výsledná url vede například na zobrazení úlohy z pohledu studenta nebo cvičícího.

```
class TaskSubmit < ActiveRecord::Base
  ...
  def routerize(namespace=nil)
    task.routerize(namespace) << self
  end
end
...
class CourseInSemester < ActiveRecord::Base
  ...
  def routerize(namespace=nil)
    params = []

    params << namespace unless namespace.nil?

    params.concat([
      semester,
      course
    ])
  end
end
```

Původní doporučení je v diskuzích v komunitě podkládáno faktem, že při automatickém generování názvů jednotlivých cest jsou výsledkem velmi dlouhé řetězce znaků. Nejedná se tedy o výkonnostní problém, ale pouze o kosmetický problém. Autor tento problém kompenzuje návrhem metody *routerize* a implementací pomocných metod, pokud již je nutné nezbyteně použít přímé vytvoření url adresy. Návrh implementace byl konzultován s vedoucím práce.

6.4.3 Autentizace

Pro autentizaci v *Ruby on Rails* aplikacích je komunitou velmi často využívána knihovna *Devise*[43]. Hlavním důvodem k využití právě této knihovny byla hlavně kompatibilita s rozšířením *Omniauth*, které umožňuje pro jednoho uživatele využívat více přihlašovacích mechanismů. Existuje pro něj mnoho implementovaných strategií využívající například přihlašování pomocí uživatelského účtu na *Facebook* nebo *Google*[44]. Existuje i neoficiální přihlašovací strategie pro *Gitlab*, která existuje ve dvou variantách:

larrylv/omniauth-gitlab[45] Tento přístup využívá standardního *Devise* přihlašovacího formuláře, jehož data jsou následně odeslána do *Gitlab* API na akci */session*, která po autorizaci vrátí entitu uživatele včetně jeho privátního klíče. Tento klíč je používán jako autentizační parameter pro další komunikaci s *Gitlab* API.

linchus/omniauth-gitlab[46] Tato strategie využívá modernějšího přístupu, a to otevřeného protokolu *OAuth2* [25][47], který je už z hlediska návrhu mnohem bezpečnější. *Gitlab* je možné používat jako poskytovatele uživatelovy identity od verze 7.7[48]. Uživatel se pak namísto v klientské aplikaci zadává přihlašovací informace přímo na *Gitlab* a následně je vygenerován autentizační klíč.

V době začátku vývoje ještě *Gitlab* nepodporoval autentizace pomocí *OAuth2*, a proto byla původně vybrána první varianta. Po update fakultní verze *Gitlab* byl kód refaktorován pro použití druhé varianty, která je bezpečnější a modernější.

Životní cyklus OAuth2 v *Gitlab* Nejříve je pro každou externí aplikaci, která bude využívat jako autentizačního poskytovatele *Gitlab*, provést registraci. Po vytvoření jsou vygenerovány bezpečnostní klíče pro komunikaci mezi aplikacemi, které je nutné přidat do konfigurace autentizačního klienta umístěného na straně klientské aplikace. Přihlašovací stránka na klientské aplikace

[43] Dostupná na adrese `https://github.com/plataformatec/devise`.
[44] Kompletní seznam dostupný na [24].
[47] Další informace dostupné na `http://oauth.net`.
[48] Kompletni seznam změn dostupný na [26].

přesměruje uživatele na web *Gitlab*, kde proběhne vlastní autentizace, a to pomocí zadání přihlašovacích údajů nebo aktivního sezení při dlouhodobém přihlášení. Při prvním přihlášení musí uživatel povolit použití klientské aplikace. Uživateli je přímo pro danou aplikaci vygenerována dvojice klíčů – přístupový a obnovovací. Přístupový klíč má danou dobu platnosti a po jeho vypršení automaticky klientská aplikace zažádá pomocí požadavku obsahujícího obnovovací klíč na *Gitlab* o vygenerování nového přístupového klíče.

Při instalaci po zvolení uživatelské entity *User* byla pomocí databázové migrace rozšířena o vlastnosti a vazby potřebné k umožnění autentizace. Následně byla implementována metoda *find_for_oauth*(viz 6.4.3)

```
def self.find_for_oauth(auth, signed_in_resource = nil)

      ...

  # Create the user if needed
  if user.nil?
        if auth.info.email
      user = User.where(:email => auth.info.email).first
    end

    # Create the user if it's a new registration
    if user.nil?
      user = User.new(
          name: auth.info.name,
          email: auth.info.email,
          password: Devise.friendly_token[0,20],
          gitlab_token: auth.credentials.token,
          gitlab_id: auth.uid
      )
      user.save!
    elsif user.gitlab_token.nil?
      # User was already prepared with import
      user.update(
          name: auth.info.name,
          password: Devise.friendly_token[0,20],
          gitlab_token: auth.credentials.token,
          gitlab_id: auth.uid
      )
    end
  end

  # if token is changed save new token
  unless user.gitlab_token == auth.credentials.token
```

```
      user.gitlab_token = auth.credentials.token
      user.save!
    end

    ...

  end
end
```

6.4.4 Autorizace

Autorizací je označováno umožnění uživatelům přístupu pouze ke zdrojům a akcím, na které mají dostatečná oprávnění. Pro implementaci autorizace v *Ruby on Rails* jsou nejrozšířenějšími variantami:

Cancancan[49] Jedná se o pokračování již zaniklé knihovny *cancan*. Tato knihovna byla využívána hlavně pro její jednoduché použití, které se drží principu *CoC* (viz 6.2). To bylo umožněno hlubokou integraci do objektů frameworku *Ruby on Rails*.

Six[50] Filosofií této knihovny je docílení maximální jednoduchosti a to nejen jednoduchosti použití, ale i jednoduchosti vnitřní struktury a nezávislosti na vybraném frameworku.

Pro vývoj aplikace xCalc byla zvolena knihovna *Six* z důvodu celkové jednoduchosti a také faktu, že toto rozšíření používá i aplikace *Gitlab*. Knihovna *Cancancan* byla dlouhou dobu po vydání *Ruby on Rails* verze 4 velmi nestabilní. Autorům trvalo velmi dlouho, než upravili právě onu hlubokou integraci do frameworku, aby se rozšíření přizpůsobilo novým konceptům, jako například takzvané „silné parametry“ [27].

Způsob použití *Six* je inspirován zdrojovým kódem aplikace *Gitlab*. Definice struktury rolí se nachází v souboru *app/models/ability.rb*. Metoda *allowed* je vstupní bod při kontrole oprávnění. Vstupními parametry jsou uživatel a objekt, ke kterému se vztahuje dotaz na oprávnění. Pokud není předán žádný objekt, jedná se globální oprávnění. Podle typu objektu jsou volány funkce, které definují oprávnění uživatele.

```
def allowed(user, subject)
  return not_auth_abilities(user, subject) if user.nil?
  return [] unless user.kind_of?(User)
  case subject.class.name
    when "CourseInSemester" then
      course_in_semester_abilities user, subject
    when "TaskSubmit" then
```

```
        task_submit_abilities user, subject
      when "Task" then
        task_abilities user, subject
        ...
      else []
    end.concat(global_abilities(user))
end
```

Ve třídě jsou nadefinovány jednotlivé skupiny oprávnění. Výsledná množina oprávnění je složen pomocí množinových operací nad těmito skupinami.

6.7: Ukázka definice oprávnění pro entity *Task*

```
def task_abilities(user, task)
    rules = []
    rules.concat teacher_of_task if task.teacher? user
    rules
end

...

def teacher_of_task
    [
        :read_task,
        :read_task_as_teacher,
        :edit_task,
        :delete_task
    ]
end
```

Použití v aplikaci je následně umožněno pomocí implementace pomocné metody do hlavního aplikačního kontroleru (viz 6.8), kterou je možné použít jak v ostatních kontrolerech, tak i v šablonách.

6.8: Ukázka souboru *app/controllers/ApplicationController.rb*

```
class ApplicationController < ActionController::Base
  helper_method :abilities, :user_can?
  ...

  def abilities
    Ability.abilities
  end
  def user_can?(action, subject = nil)
    abilities.allowed?(current_user, action, subject)
  end
```

```
end
```

6.4.5 Studijní struktura

Z pohledu první iterace se vývoj týká pouze implementace importu.

Dalším krokem byla implementace klienta, který bude s rozhraním *KOSapi* komunikovat. V tomto směru existovaly pouze dvě možnosti: použití open-source *RUBY* klienta *cvut/kosapi_client.rb*[51] nebo implementovace vlastního řešení. Autor se rozhodl pro využití klienta *cvut/kosapi_client.rb*, neboť již obsahoval implementovanou strukturu pro zpracování odpovědí. Dalším argumentem byl fakt, že při implementaci chybějící funkcionality autor přispěje k rozvoji této knihovny.

Autor vytvořil „fork“ (rozštěp) [28] repozitáře[52], v kterém probíhala implementace nové funkcionality. V jeho vývoji je využíváno konceptu separátního vývoje funkcionalit ve vlastních větvích, což umožňuje jednodušší začlenění změn do původního repozitáře. Po dokončení implementace je vytvořen „pull request“ (žádost o spojení) zpět do původního repozitáře a zároveň je provedeno spojení do větve master, aby autor mohl tuto knihovnu i se svými změnami používat v rámci vývoje, dokud nejsou změny doladěny a spojeny s původním repozitářem.

Vyřešení konfliktu závislosti *Faraday* [53]První překážkou při použití knihovny *cvut/kosapi_client.rb* byla kolize požadavků pro závislosti. Knihovna pro vytváření HTTP požadavků *Faraday* je v klientovi shora omezena verzí *0.8.9*, protože knihovna pro testování komunikace s externími službami *VCR* dosud nepodporuje novou verzi. Naopak rozšíření používané při autentizaci *gitlab-omniauth* vyžaduje knihovnu *Faraday* alespoň ve verzi *0.9*. Z několika možných řešení vybral autor přesunutí definice závislosti knihovny ze souboru *kosapi_client.gemspec* do *Gemfile*, které umožňuje při zadání závislosti pomocí git repozitáře i použití specifikované větve. Jedná se totiž o závislost, která ovlivňuje pouze vývoj klienta a ne jeho použití. Navíc u knihovny *VCR* je řešení kompatibility již implementované, nicméně stále čeká na vydaní nové verze.

[51]Dostupný na `https://github.com/cvut/kosapi_client.rb`.

[52]Dostupný na `https://github.com/DraCzris/kosapi_client.rb`.

[53]Provedené změny jsou dostupné v commitu na `https://github.com/DraCzris/kosapi_client.rb/commit/316b4b8b3b9d6d459f5a95fa54b1378c114ae0c9` a výsledný pull request na `https://github.com/cvut/kosapi_client.rb/pull/9`.

Implementace procházení referenčních odkazů [54] Jednalo se o implementaci tiketu #3[55]. Rozhraní klienta umožňuje automatické stažení dalších referencovaných entit, při přístupu k dané promněné, resp. refereční odkaz je zaměněn za entitu, na kterou odkazuje. Původně KOSapi klient umožňoval pouze procházení stránkovacích referenčních linků. Pro implementaci této funkcionality bylo nutné zasáhnout do velké části knihovny. Nově je při mapování zpracovávaného stromu odpovědi předáván kontext, který v sobě nese http klienta, který je potřeba pro provedení dotazu na KOSapi. Po implementaci byly upraveny stávající testy a přidány testy pro novou funkcionalitu.

Přidání podpory entity a akcí zdroje semetr [56] Pro získání dat semestru bylo potřeba přidat nové mapování pro entitu *Semester*. Dále byla přidána podpora pro jednotlivé akce zdroje a implementovány testy pro přidanou funkcionalitu. V rámci této změny došlo také k nastavení hlavičky pro vyžádání formátu odpovědi[57]. Zároveň byla rozšířena dokumentace o návod na přidání dalších zdrojů.

Rozšíření akcí zdroje předmět a implementace entity kurz v semestru [58]. Aby bylo možné získat všechny vyučující pro akutální semestr, bylo nutné implementovat novou entitu „kurz v semestru“. Pro získání seznamu paralelek předmětu v daném semestru byla přidána podpora akce */courses/{code}/parallels*. Navíc byla doplněna možnost zadání parametru *sem*, který umožňuje filtrování výsledků v okruhu platnosti zadaného semestru[59].

Vlastní import je rozdělen do dvou vývojových iterací. V první je implementována úloha pro *Rake*(viz 6.2.2), kdy obě fáze importu spouští administrátor pomocí příkazového řadku přímo na serveru. V druhé iteraci je pro druhou fázi importu implementováno uživatelské rozhraní, které umožní manipulaci s daty před zpracováním získaných dat , například zrušení paralelky nebo přesun studenta.

[54] Provedené změny jsou dostupné na `https://github.com/DraCzris/kosapi_client.rb/commit/2e79897506b1c8f3ef2146e19026d722c5852244` a výsledný pull request na `https://github.com/cvut/kosap_client.rb/pull/10`.

[55] Dostupný na `https://github.com/cvut/kosapi_client.rb/issues/3`.

[56] Provedené změny dostupné na `https://github.com/DraCzris/kosapi_client.rb/commit/42c6700ff85b28cf4ec7393dd68cfef572c33b2d` a výsledný pull request na `https://github.com/cvut/kosap_client.rb/pull/11`.

[57] Úpravy dostupné na `https://github.com/DraCzris/kosapi_client.rb/commit/d48436f66c34486ca441308ddfb15278cbe473a3`.

[58] Provedené změny dostupné na `https://github.com/DraCzris/kosapi_client.rb/commit/08e35c3f11d0547813f870276d239dee7d0e1fd0d` a výsledný pull request na `https://github.com/cvut/kosap_client.rb/pull/12`.

[59] Dokumentace dostupná na `https://kosapi.fit.cvut.cz/projects/kosapi/wiki/URLParameters#sem`.

Importovací algoritmus se nachází v souboru *lib/study_structure/importer.rb*. Při první fázi importu je volána metoda *import_course_in_semester* a při druhé metoda *import_students_for_course_in_semester*.

Pro umožnění importu z různých datových zdrojů, například ze souboru importovací algoritmus nekomunikuje přímo s KOSapi, ale s rozhraním definovaným abstraktním datovým zdrojem. Následuje popis metod rozhraní:

semester Jejím argumentem je kód semestru a návratovou hodnotou jsou detailní údaje o semestru.

course Jejím argumentem je kód předmětu a návratovou hodnotou jsou detailní údaje o předmětu.

teachers Jejími argumenty jsou kód semestru a kód předmětu. Tato metoda vrací seznam vyučujících v předmětu včetně jejich rolí (zkoušející, garanti, cvičící a přednášející). Pokud má vyučující více rolí, je mu přiřazena ta nejvyšší.

parallels Jejími argumenty jsou kód semestru a kód předmětu. Volitelný parametrem je typ paralelky[60], který je aplikován jako filtr. Návratovou hodnotou jsou detailní údaje paralelky.

students Jejím argumentem je unikátní identifikátor paralelky. Návratovou hodnotu je pole studentů.

Knihovna je do přidána do seznamu závislostí příkazem:

```
gem 'kosapi_client', github: 'DraCzris/kosapi_client.rb'
```

6.4.5.1 Synchronizace

Následujícím krokem byla implementace synchronizace s *Gitlab* a *Gitlab CI*. Stejně jako u KOSapi je nutné vybrat klienta, pomocí kterého bude probíhat komunikace s cílovou službou. Komunikace probíhá prostřednictvím přístupového kódu uživatele *xcalc-app*, který je definován v nastavení aplikace.

Gitlab U aplikace *Gitlab* je oficiálně dostupný seznam kompatibilních knihoven a aplikací[61], který obsahuje i knihovnu *NARKOZ/gitlab*[62]. Tato knihovna implementuje všechny potřebné metody:

- vytvoření skupiny,

[60] Dokumentace dostupná na `https://kosapi.fit.cvut.cz/projects/kosapi/wiki/ParallelType`.

[61] Seznam je dostupný na `https://about.gitlab.com/applications/`.

[62] Dostupná na `https://github.com/NARKOZ/gitlab`.

- vytvoření repozitáře,
- správa uživatelů skupiny a repozitáře.

Gitlab CI Autorovi se pro tuto aplikaci nepodařilo najít žádného *RUBY* klienta. Navíc bude potřeba implementovat další akce pro komunikaci s rozšířenou funkcionalitou. Autor se tedy rozhodl pro implementace vlastního řešení. Struktura knihovny byla silně inspirována strukturou klienta *NARKOZ/gitlab*. Tato knihovna není implementována jako samostatný balíček, ale díky způsobu implementace je možná jednoduchá extrakce a vydání samostatné knihovny. Ze standardních metod je potřeba pouze vytvoření „CI projektu“, což je entita reprezentující repozitář v aplikaci *Gitlab*. Knihovna je umístěna ve složce *lib/gitlab_ci*.

Do všech entit, kterých se synchronizace týká, bylo přidáno pole vyjadřující stav synchronizace (zda proběhla, či ne), definice rozhraní v podobě metody *sync_with_gitlab* a další pomocné metody (viz ukázka zdrojového kódu 6.9). Tato rozšíření jsou implementována pomocí koncernů(viz **??**). Hodnota statusu je implementována pomocí enumerátoru a může nabývat dvou hodnot:

ready entita je synchronizována a připravena k použití,

awaiting_sync entita ještě nebyla synchronizována.

Pomocí metody *scope* je umožňuje jednoduché použití častých dotazů. Příkladem použití je scope *awaiting_sync*, který je ekvivalentem dotazu na všechny instance entity, která má synchronizační stav nastaven na „čeká na synchronizaci“.

6.9: Ukázka zdrojového kódu synchronizačního koncernu - *app/models/concerns/gitlab_syncable.rb*

```
module GitlabSyncable
  extend ActiveSupport::Concern
  included do
    enum state: [:ready, :awaiting_sync]

    scope :awaiting_sync, -> { where state: self.states[:awaiting_sync] }

    def self.sync_with_gitlab(client)
      raise "Sync with gitlab is not implemented in class #{self.class}"
    end
    ...
  end
end
```

Následuje příklad implementovaného rozhraní pro entitu kurz v semestru (ze zdrojového kódu byly odstraněny logující zprávy).

```
class CourseInSemester < ActiveRecord::Base
  include GitlabSyncable
  def self.sync_with_gitlab(logger, options)
    CourseInSemester.awaiting_sync.each do |course_in_semester|
      begin
        result = Gitlab.create_group(
          course_in_semester.group_name,
          course_in_semester.group_path
        )

        course_in_semester.update({
          group_id: result.id,
          state: CourseInSemester.states[:ready]
        })
      rescue => error
      ...
      end
    end
  end
```

Algoritmus, který provádí vlastní synchronizaci vypadá následovně:

```
module GitlabSynchronizer
  class Synchronizer
        ...
    def sync(entity)
      entity.sync_with_gitlab @logger, @options
    end

    def sync_all
      [
        CourseInSemester,
        UserTeachesCourse,
        UserHasParallel
      ].each do |entity|
        sync entity
      end
    end
  end
end
```

Synchronizace probíhá v následujícím pořadí:

CourseInSemester vytvoření skupiny,

UserTeachesCourse přiřazení vyučujících do skupiny,

UserHasParallel vytvoření repozitářů pro studenty a jejich přiřazení do repozitáře.

Při přiřazování uživatelů na začátku semestru se může stát, že uživatel je sice importován do aplikace *xCalc*, ale nemusí ještě existovat v *Gitlabu*, kde mu je uživatelský účet vytvořen až po prvním přihlášení, což by při synchronizaci způsobovalo chyby. Proto bylo zavedeno pravidlo, že synchronizace, pokud provádí přiřazování uživatelů v aplikaci *Gitlab*, se pro danou entitu provede až po prvním přihlášení uživatele do aplikaci *xCalc*.

Synchronizace je spouštěna pomocí vytvořené *rake* úlohy – *study_structure:sync_gitlab*. Tato je v první iteraci volána pomocí utility *cron*, která umožňuje periodicky opakované provádení příkazů. V druhé verzi bude implementována i možnost spuštění přímo z uživatelského rozhraní aplikace *xCalc*.

Při implementaci autor narazil na problém nekonzistence uživatelských práv aplikovaných při vytváření skupiny přes uživatelské rozhraní a pomocí API. Vytvoření bylo povoleno pouze uživatelům s rolí administrátora. Ve fakultním *Gitlabu* je chyba opravena „zápalatou“ (patchem) a v době tvorby práce čeká oprava, na jejímž vytvoření se autor podílel, na integraci do zdrojového kódu *Gitlabu* [29]. Díky této opravě nemusí mít uživatel *xcalc-app* v *Gitlabu* administrátorská práva, ale stačí pouze oprávnění k vytváření skupin, čímž se minimalizuje bezpečnostní riziko spojené s každým administrátorským účtem.

6.4.6 Správa úkolů

Tato podsekce popisuje implementační řešení vytváření složitých formulářů a editace množiny parametrů.

6.4.6.1 Knihovna pro tvorbu složitých formulářů

Pro implementaci složitého formuláře vytvoření úkolu a vytvoření výpočetní úlohy byla použita knihovna *Formtastic*[63]. Ta implementuje DSL pro vytváření formulářů v šablonách. Při vytváření vlastních formulářů určuje dynamicky cíl a typ akce. Například dokáže rozeznat dle entity předané jako datový zdroj formuláře, zda se jedná o formulář pro vytvoření nebo úpravu, a dle toho vybrat správnou cílovou akci i HTTP metodu. Umožňuje definici vlastních typů formulářových polí.

Pro zadávání parametrů pomocí formátu YAML byl implementován vlastní typ formulářového pole pro databázový typ *hstore*, aby knihovna *formtastic*

[63] Dostupná na `https://github.com/justinfrench/formtastic`.

mohla automaticky tento typ použít. K datům je přistupováno pomocí implementovaných metod, které při načtení data serializují do formátu YAML a při zpracování odeslaného formuláře tyto data naopak deserializují. Tato implementace je potenciálním kandidátem pro extrakci do koncernu a umožnění použití pomocí DSL, což by poskytovalo jednoduchou možnost modifikace vlastností entity uložené ve formátu *hstore*.

```
def parameters_plain
  psych_yaml = self.parameters.to_yaml
  psych_yaml.lines[2..-1].join
end

def parameters_plain=(content)
  self.parameters = YAML::load(content)
end
```

6.4.6.2 Implementace vykreslování a editace formátu *Asciidoc*

Editace zádání probíhá pomocí standardního textového pole. Pro vykreslování formátu byla použita knihovna *Asciidoctor*[64]. Použití v šabloně je umožněno implementací jednoduché pomocné funkce pro vlastní vygenerování HTML z dat uložených ve formátu *Asciidoc*.

```
def render_asciidoc(content)
  return "" if content.nil?

  Asciidoctor.convert content, safe: 'safe'
end
```

6.4.7 Komponenta pro zobrazování výsledků

Realizace této komponenty nebyla v době odevzdání této práce dokončena. Důvodem bylo zdržení z důvodu nutnosti přízpusobení změnám v API a procesech systémů *Gitlab* a *Gitlab CI*. Přizpůsobení předmětům MI-PAA a MI-ADM x je bohužel přímo závslé na implementaci komponenty pro zobrazování výsledků, a proto nebyla v době odevzdávání této práce realizována.

6.4.8 UI

Tato podkapitola nejprve popisuje pouřití vybraného formátu pro vytváření HTML šablon. Dále uvádí seznam a popis externích knihoven. Nakonec je popsána struktura frontendových zdrojových kódů.

[64]Dostupná na `http://asciidoctor.org`.

```
admin.rb ..... Typy pro sekci administrátora
shared.rb ..... Typy sdílené všemi uživateli
student.rb ..... Typy pro sekci studenta
teacher.rb ..... Typy pro sekci cvičího
```

6.4.9 HAML a haml-rails

Jak bylo již zmíněno v analýze (viz 6.2.5), jako šablonovací systém pro generování HTML byl vybrán *HAML*. Díky flexibilitě frameworku *Ruby on Rails* je po přidání vývojové závislosti rozšíření *haml-rails*[65] automaticky změněn výchozí formát z ERB právě na *HAML*. Tato knihovna obsahuje i utilitu pro převod šablon z formátu *ERB*.

6.4.10 Gretel

Knihovna *Gretel*[66] umožňuje jednoduchou implementaci a konfigurace drobečkové navigace. Konfigurace umožňuje i skládání z jednotlivých podsegmentů.

Vývojář definuje jednotlivé typy zobrazení v souborech ve složce *app/config/breadcrumbs*. Následuje popis struktury souborů s definicí typů drobečkové navigace, které jsou za účelem přehlednosti rozděleny do samostatných souborů dle role uživatele.

6.10: Ukázka definice typu drobečkové navigace

```
crumb :teacher_task do |task|
  link task.title, teacher_semester_course_task_path(*task.routerize)
  parent :course_in_semester, task.course_in_semester
end

crumb :teacher_edit_task do |task|
  link "Edit task"
  parent :teacher_task, task
end
...
```

Vykreslení v hlavní navigaci je provedeno pomocí příkazu = *breadcrumbs*. Na v jednotlivých šabonách je pak vybrán aktuální typ drobečkové navigace.

6.11: Příklad výběru typu drobečkové navigace

```
...
- breadcrumb :student_create_submit, @submit
...
```

[65] Dostupné na `https://github.com/indirect/haml-rails`.
[66] Dostupná na `https://github.com/lassebunk/gretel`.

6.4.11 Další použité knihovny

Pro implementaci uživatelského rozhraní bylo použito mnoho knihoven. Důvodem je jejich fragmentace a často velmi specifická funkcionalita.

6.4.11.1 Sass-rails

Sass[67] je takzvaný CSS preprocesor, který umožňuje vytváření vysoce znovupoužitelného kódu kaskádových stylů. Základní formát trpí například neustálou nutností opakování a jeho udržitelnost není vysoká. Preprocesor přidává asi jako nejdůležitější možnost používání proměnných a maker.

6.4.11.2 Coffee-rails

CoffeeScript[68] je programovací jazyk, který se kompiluje do javascriptu. Jeho předností je zjednodušení syntaxe, zvýšení čitelnosti a snížení délky zdrojového kódu.

6.4.11.3 Bootstrap-sass, twitter-bootstrap-rails a bootstrap-datepicker-rails

Tyto knihovny poskytují zdrojové kódy pro knihovnu *Twitter bootstrap* a navíc její použití v *RUBY* zjednodušují díky pomocným metodám pro vytvoření a konfiguraci komponent uživatelského rozhraní. Knihovna *bootstrap-datepicker-rails*[69] přidává formulářové pole pro výběr datumu.

6.4.11.4 Jquery-rails

Ačkoluv je tato knihovna dostupná pomocí balíčkovacího systému *Bundler*, jedná se pouze o zabalení javascriptové knihovny *JQuery*[70]. Ta umožňuje extrémně jednoduchou manipulaci se stromem HTML elementů a jejich argumenty.

6.4.11.5 Turbolinks a nprogress-rails

Tato knihovna zlepšuje a zrychluje pro uživatele procházení aplikací. Namísto obnovy celé stránky je odeslán na server požadavek pomocí technologie *AJAX* a při přijetí odpovědi nahradí aktuální obsah HTML elementu *body*. Standardně při přechodu z jedné stránky na jinou při odeslání požadavku dojde ke zmizení stránky a její načítání probíhá v pozadí. Naproti tomu při použití knihovny *turbolinks*[71] spolu s rozšířením *nprogress-rails*[72] zůstane stránka zob-

[67] Dostupný na `http://sass-lang.com`.
[68] Dostupný na `http://coffeescript.org`.
[69] Dostupná na `https://github.com/Nerian/bootstrap-datepicker-rails`.
[70] Dostupný na `https://jquery.com`.
[71] Dostupné na `https://github.com/rails/turbolinks`.
[72] Dostupné na `https://github.com/caarlos0/nprogress-rails`.

```
images ..................................... Obsahuje statické obrázky
javascripts ................................ Obsahuje frontendové skripty
    application.js ......... Základní soubor pro frontendové skripty
    home.js.coffee ....... Frontendové skripty pro určitou sekci aplikace
stylesheets ................................ Obsahuje zdroje pro kaskádové styly
    _variables.css.scss ............. Definice upravených proměnných
    application.css.scss ......... Základní soubor pro kaskádové styly
    home.css.scss ........... Kaskádový styl pro určitou sekci aplikace
```

razena, všechny její elementy jsou stále aktivní a navíc je zobrazen indikátor aktivity v horní části obrazovky.

6.4.11.6 Formtastic-bootstrap

Tato knihovna rozšiřuje základní formulářové typy v knihovně*formatastic*?? tak, aby vykreslování probíhalo pomocí elementů definovaných knihovnou *Twitter bootstrap*.

6.4.11.7 Font-awesome-rails

Stejně jako u *jquery-rails*6.4.11.4 se jedná o pro *Bundler* zabalenou knihovnu *Font Awesome*. Ta umožňuje použití vektrových ikon na webu a je rozšířena o pomocnou funkci pro jejich vykreslování do HTML pomocí *RUBY*.

6.4.12 Struktura frontendových zdrojů

Všechny kaskádové styly a frontendové skripty jsou umístěny ve složce *app/assets*. Spolu s nimi se zde nachází také použité obrázky. Následuje digram struktury této složky:

application.css.scss* a *application.js Tyto soubory zajisťují načítání skriptových závislostí a fungují také jako vstupní body pro daný kompilátor. Ten dle přípony buď soubor zkompiluje nebo ponechá nezměněný. *Ruby on Rails* rozšiřují standardní javascriptový formát o makra právě pro načítání závislostí, které nejsou přítomné přimo ve složce *assets*, ale jsou umístěny uvnitř nainstalovaných knihoven.

Ukázka načítání závislostí pro frontendové skripty

```
//= require bootstrap-datepicker
//= require_tree .
```

Ukázka načítání závislostí pro kaskádové styly

```
@import '_variables';
@import 'formtastic-bootstrap';
```

_variables.css.scss V tomto souboru se nachází konfigurace použitých frontendových knihoven pomocí úprav výchozích hodnot proměnných preprocesoru *SCSS*.

6.5 API

Aplikace obsahuje webové API, aby umožnila *FIT CI* efektivně aktualizovat stav výpočetních úloh. Implementace je inspirována zdrojovými kódy API aplikace *Gitlab CI*. V budoucnu bude možné jednoduše rozšířit toto API o další funkcionalitu, kterou by například studenti mohli využít pro vytvoření mobilních aplikací. Ty by mohly umožňovat provádění úkonů na aplikaci *xCalc* nebo upozorňovat uživatele na změny v systému pomocí push notifikací.

Implementace je založena na knihovně *Grape*, která funguje jako framework pro vytváření RESTful API. Celý blok obsahující všechny akce API je integrován přímo do směrovacího systému *Ruby on Rails* 6.12 a je možné si libovolně zvolit prefix těchto akcí. Standarně je prefixem řetězec */api*.

6.12: Ukázka integrace API do směrovacího systému aplikace (výřez souboru *config/routes.rb*)

```
Rails.application.routes.draw do
  ...
  mount API::API => '/api'
  ...
end
```

Autentizace aplikace *FIT CI* vůči *xCalc* API probíhá pomocí privátního, v konfiguraci definovaného klíče. Ten je předáván pomocí http hlavičky *HTTP_XCALC_TOKEN*. Tento způsob předávání je zabezpečen faktem, že obě aplikace běží na protokolu *HTTPS* a hlavička je tedy přenášena šifrovaným spojením.

Protože se jedná o potenciálně asynchronní komunikaci je k entitě výpočetní úlohy v aplikaci *xCalc* přidáno pole *last_updated_at*, které funguje jako ochrana proti prohození pořadí zpracování příchozích notifikací.

Následně byly implementovány základní metody. První zpracovává notifikaci o začátku procesování dané výpočetní úlohy a zároveň obnovu fronty ostatních čekajících výpočetních úloh. Druhá metoda je koncovým bodem pro zprávu o ukončení dané výpočetní úlohy (viz ukázka zdrojového kódu viz 6.13).

6.13: Zdrojový kód pro API akci pro zpracování zprávy o dokončení výpočetní úlohy

```
module API
  class Service < Grape::API
    before { authenticate_xcalc! }

    ...

    desc "Receive notification about task's finish."
    params do
      requires :time, type: DateTime, desc: "Notification timestamp."
    end
    put 'build-finished/:submit_sha' do
      status 200

      submit = TaskSubmit.find_by sha: params[:submit_sha]
      if submit
        submit.update({
          job_finished_date: params[:time],
          exit_status: params[:status].to_sym,
          exit_cause:
              (params[:cause].present? ? params[:cause].to_sym : nil),
          state: TaskSubmit.states[:finished],
          last_updated_at: params[:time]
        })
      else
        not_found!
      end
    end
  end
end
```

6.6 Testování

Z pohledu testování, flexibilita programovacího jazyka *RUBY* umožňuje hlavně díky jednoduché implementaci vlastních *DSL*, tvorbu velmi uživatelsky přívětivých definic testů. Pro *RUBY* existuje velké množství testovacích frameworků, ale asi nejpoužívanějšími jsou *minitest*, který je od verze 1.9 obsažen již přímo v *RUBY*, a knihovna *RSpec*[73]. Funkcionalitou jsou velmi podobné a liší se spíše ve formátování zdrojového kódu testů. Autor zvolil při implemen-

[73]Dokumentace a detailní informace dostupné na `http://rspec.info`.

taci knihovnu *RSpec* z důvodu, že s ní měl již v minulosti zkušenosti. Navíc tuto knihovnu používá i aplikace *Gitlab*.

6.6.1 Rspec

Testy jsou spouštěny pokocí rake úlohy *rake spec* a pomocí parametrů je možné spouštět jednotlivé soubory nebo i přímo určité testy. Zdrojvé kódy testů se nachází ve složce *spec*. Na začátku každého testu je importován soubor *rails_helper.rb*, který zajistí spuštění jádra frameworku *Ruby on Rails*. Pomocí knihovny *FactoryGirl* je možné vytvářet různě předkonfigurované objekty, se kterými následně testy operují.

```
FactoryGirl.define do
  factory :user do

    factory :student do
      nick { generate(:student_name) }
    end

    factory :teacher do
      nick { generate(:teacher_name) }
    end

    nick { name.downcase }
    email { "#{nick}@example.com".downcase }
    password Devise.friendly_token[0,20]
  end
end
```

6.7 Deployment

Pro deployment – volným překladem „nasazování“ – je použita knihovna *Capistrano*[74]. Verze upravená na míru frameworku *Ruby on Rails* se jmenuje *capistrano/rails*[75]. Při použití této knihovky není potřeba manuálně definovat celý postup deploymentu aplikace, který automaticky při instalaci knihovna vytvoří do souboru *config/deploy.rb*. Pro použití stačí pouze nakonfigurovat příslušné proměnné v souborech ve složce *app/config/deploy*. Struktura zdrojových kódů na produkci je znázorněna v diagramu 6.4. Následuje ukázka konfigurace pro deploy na produkční prostředí:

Migrační úloha zajistí stažení zdrojových kódů z repozitáře do nové složky v *releases* a nastaví symbolické odkazy uvnitř projektu na sdílené adresáře.

[74] Dostupné na `http://capistranorb.com`.
[75] Dostupná na `https://github.com/capistrano/rails/`.

```
set :rails_env, "production"

set :bundle_without, %w{test development}.join(' ')

server 'storm1.fit.cvut.cz',
  user: 'xcalc',
  roles: %w{web app db},
  ssh_options: {
    keys: %w(~/.ssh/id_rsa),s
    auth_methods: %w(publickey),
    port: <secret>
  }
```

```
current -> release/<version> ..... Odkaz na aktuální verzi
releases ..... Složky s deploynutými zdrojovými kódy
  20150410141655 ..... Zdrojový kód projektu unikátního deployementu
  20150421122556
  ...
revisions.log ..... Soubor obsahující záznamy o deploymentech
shared ..... Obsahuje adresáře sdílené napříč deploynutými verzemi
  bin
  bundle
  config
  log
  public
  tmp
  vendor
```

Diagram 6.4: Struktura zdrojových kódů pro deployment

Dále se postará o nainstalování nebo update závislostí, kompilaci statických souborů, spuštění migrace databáze a případně provedení dalších definovaných úloh. Nakonec je změněn symbolický odkaz aktuální verze zdrojového kódu a restartována samotná webová aplikace. Spouštění vlastního deploymentu je prováděn příkazem *cap production deploy*.

Závěr

V době odevzdání práce byla v aplikaci *xCalc* implementována funkcionalita, která umožňuje uživatelům systému odesílat parametrizované výpočetní úlohy vytvořené ze zdrojových kódů umístěných na portálu *ČVUT Gitlab* do *FIT CI*, kde se zařadí do fronty ostatních požadavků. *FIT CI* se následně postará o vlastní proces výpočtu. Uživatelé se do aplikace *xCalc* přihlašují pomocí účtu na *ČVUT Gitlab*, kde autentizace probíhá bezpečně pomocí fakultního systému *LDAP*. Dále aplikace *xCalc* umožňuje import struktury v podobě hierarchie předmětů v semestru, paralelek, včetně studentů i cvičících. Díky návrhu interní struktury není použití aplikace omezeno pouze na importovanou studijní strukturu, ale umožňuje i spouštění úloh například v rámci výzkumů, které z tohoto faktu mohou velmi benefitovat.

V rámci práce byly zpracovány funkční požadavky v analýze, na jejímž základě byl vytvořen návrh uživatelského rozhraní a struktury backendu. Součástí této analýzy bylo i definování vztahu a způsob komunikace s externími systémy při vykonávání jednotlivých aktivit.

Autorovi se podařilo navrhnout formát a způsob komunikace s externími službami tak, aby bylo možné využití aplikace *xCalc* co nejširší a vlastní použití nebylo omezeno pevně definovanými parametry úloh.

V průběhu implementace byl rozšířen open-source klient pro komunikaci s *KOSapi*. Za hlavní přínos lze označit rozšíření klienta o možnost dynamického procházení referencovaných entit. Tyto změny bohužel stále čekají na integraci.

Celkový rozsah implementace navržené funkcionality byl naneštěstí omezen hlavně z důvodu nutného přizpůsobení změnám při aktualizaci systémů *Gitlab* a *Gitlab Ci*. V obou případech se jednalo o velmi fundamentální změny a v případě aplikace *Gitlab CI* se jednalo dokonce o změnu celé koncepce vytváření výpočetních úloh. Celkově se však podařilo vytvořit flexibilní soubor systémů a komunikace, který umožňuje díky použitým knihovnám a přístupům jednoduché rozšíření v budoucnu.

Pro autora byla práce na tomto projektu vysoce přínosná. Přínosem je

jak získaná praxe s vývojem v programovacím jazyce *RUBY* a frameworku *Ruby on Rails*, tak i zkušenost s aktivním zapojením se do vývoje open-source projektu.

Literatura

[1] Nielsen, J.: 10 Usability Heuristics for User Interface Design. Dostupné z WWW: <http://www.nngroup.com/articles/ten-usability-heuristics/>, 2013.

[2] Cooper, A.: *The Inmates Are Running the Asylum*. Division of MacMillan Computer Publishing, 1999.

[3] Nielsen, J.: Breadcrumb Navigation Increasingly Useful. *Nielsen Norman Group*, 2007. Dostupné z: http://www.nngroup.com/articles/breadcrumb-navigation-useful/

[4] Kolektiv autorů: Viking Education Inc.: Approaching Complex Problems. Dostupné z WWW: <http://www.vikingcodeschool.com/software-engineering-basics/approaching-complex-problems>, 2015. Dostupné z: http://s3.amazonaws.com/viking_education/web_development/prep_engineering/facebook_status_bar_small.jpg

[5] Kolektiv autorů: The Daring Fireball Company LLC.: Markdown. Dostupné z WWW: <http://daringfireball.net/projects/markdown/>, 2004.

[6] Jakub Jirůtka: *Projekt KOSapi*. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki

[7] Jakub Jirůtka: *Projekt KOSapi - Semestry*. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki/Semesters

[8] Jakub Jirůtka: *Projekt KOSapi - Předměty*. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki/Courses

[9] Jakub Jirůtka: *Projekt KOSapi - Instance předmětu*. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki/Coursin

[10] Jakub Jirůtka: *Projekt KOSapi - Rozvrhové paralelky*. Dostupné z: https://kosapi.fit.cvut.cz/projects/kosapi/wiki/Parallels

[11] Kolektiv autorů: Gitlab: *Permissions*. [cit. 2015-04-29]. Dostupné z: https://github.com/gitlabhq/gitlabhq/blob/master/doc/permissions/permissions.md

[12] Kolektiv autorů: Gitlab: *Commits API - Create commit*. [cit. 2015-05-05]. Dostupné z: https://github.com/gitlabhq/gitlab-ci/blob/master/doc/api/commits.md#create-commit

[13] Oleg Bartunov, Teodor Sigaev a Andrew Gierth: *PostgreSQL: Documentation: 9.1: hstore*. Dostupné z: http://www.postgresql.org/docs/9.1/static/hstore.html

[14] Fowler, M.: *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002, ISBN 0321127420.

[15] Kolektiv autorů: Ruby on Rails: Getting Started with Rails — Ruby on Rails Guides: What is Rails? Dostupné z WWW: <http://guides.rubyonrails.org/getting_started.html#what-is-rails-questionmark>, 2015.

[16] Kolektiv autorů: Ruby on Rails: *The Rails Command Line — Ruby on Rails Guides: Rake*. Dostupné z: http://guides.rubyonrails.org/command_line.html#rake

[17] Kolektiv autorů: Ruby on Rails: *Layouts and Rendering in Rails — Ruby on Rails Guides*. Dostupné z: http://guides.rubyonrails.org/layouts_and_rendering.html

[18] Kolektiv autorů: Ruby on Rails: Ruby on Rails 4.2 Release Notes — Ruby on Rails Guides. Dostupné z WWW: <http://guides.rubyonrails.org/4_2_release_notes.html>, 2015.

[19] Vincent Driessen: A successful Git branching model. Dostupné z WWW: <http://nvie.com/posts/a-successful-git-branching-model/>, 2010. Dostupné z: http://nvie.com/img/git-model@2x.png

[20] Kolektiv autorů: Ruby on Rails: *Ruby on Rails 4.2.1: ActiveSupport::Concern*. Dostupné z: http://api.rubyonrails.org/classes/ActiveSupport/Concern.html

[21] Kolektiv autorů: Ruby on Rails: *Layouts and Rendering in Rails — Ruby on Rails Guides*. Dostupné z: http://guides.rubyonrails.org/routing.html#limits-to-nesting

[22] Kolektiv autorů: Rails 4 [Best practices] Nested resources and shallow: true. Dostupné z WWW: <https://stackoverflow.com/questions/21868544/rails-4-best-practices-nested-resources-and-shallow-true>, 2014.

[23] Kolektiv autorů: RailsCasts: Nested Resources. Dostupné z WWW: <http://railscasts.com/episodes/139-nested-resources?view=comments>, 2008.

[24] Kolektiv autorů: INTRIDEA Inc.: *List of Strategies*. Dostupné z: https://github.com/intridea/omniauth/wiki/List-of-Strategies

[25] Dick Hardt: The OAuth 2.0 Authorization Framework. Dostupné z: http://tools.ietf.org/html/rfc6749#section-1.2

[26] Kolektiv autorů: Gitlab: Gitlahhq: CHANGELOG. Dostupné z WWW: <https://github.com/gitlabhq/gitlabhq/blob/3ccdc02b89ab368e7701c53c38a2c9440c76fcdc/CHANGELOG#L352>.

[27] Kolektiv autorů: Ruby on Rails: *Action Controller Overview — Ruby on Rails Guides: Strong Parameters*.

[28] Kolektiv autorů: Git and Software Freedom Conservancy: *Hostování projektů Git*.

[29] Jakub Jirůtka: Fix #6417: users with group permission should be able to create groups via API. Dostupné z: https://github.com/gitlabhq/gitlabhq/pull/9066

Příloha **A**

Seznam použitých zkratek

API Aplication Interface

REST Representational State Transfer

HTTP Hypertext Transfer Protocol

UI User interface

CRUD Create Update Delete

DSL Domain-specific language

DRY Don't Repeat Yourself

CoC Convention Over Configuration

GUI Graphical user interface

XML Extensible markup language

AJAX Asynchronous JavaScript and XML

CSV Comma Seprated Values

JSON JavaScript Object Notation

WYSIWYG What You See Is What You Get

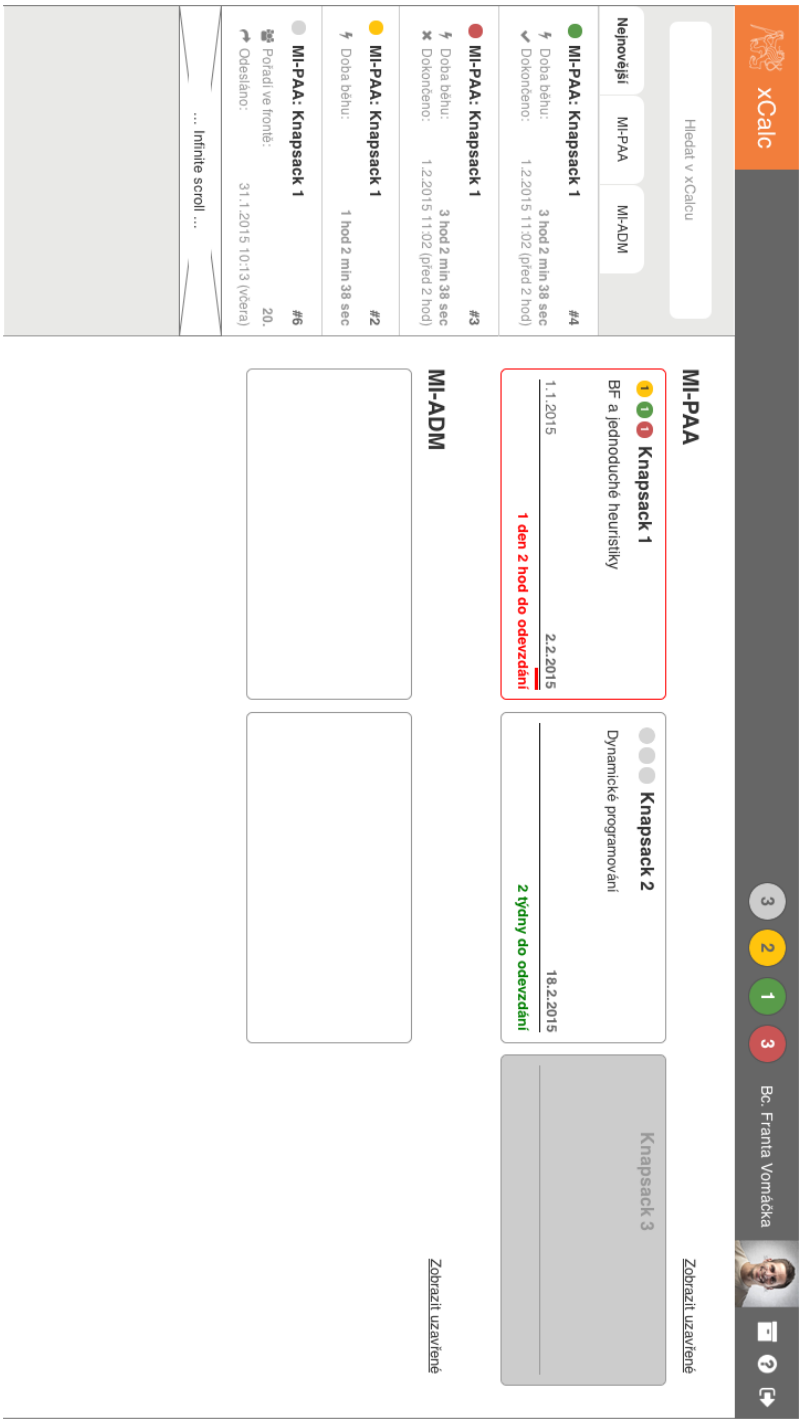
Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS

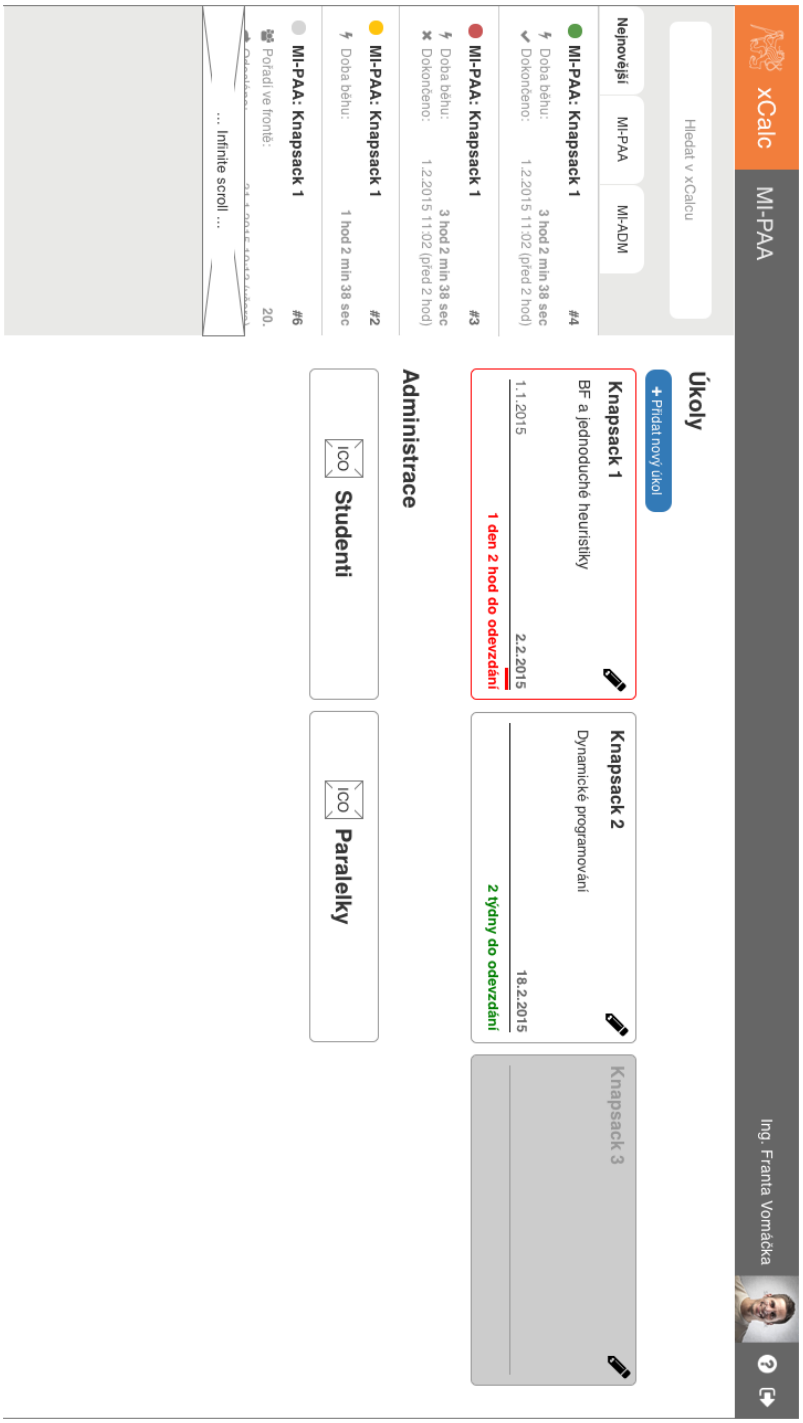
Příloha C

Ukázka obrazovek prototypu



Obrázek C.1: Návrh dashboardu z pohledu studenta

Obrázek C.2: Návrh formuláře pro výběr verze k vytvoření výpočetní úlohy



Obrázek C.3: Návrh detailu předmětu z pohledu cvičícího