Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

REST služba pro konverzi LaTeX souborů do PDF

TADEÁŠ KYRAL

Vedoucí: Ing. LUKÁŠ ZOUBEK
Obor: Softwarové inženýrství a technologie
Květen 2019

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kyral** Jméno: **Tadeáš** Osobní číslo: **466164**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

REST služba pro konverzi LaTeX souborů do PDF

Název bakalářské práce anglicky:

Creation of REST service for conversion from LaTeX to PDF

Pokyny pro vypracování:

Navrhněte webovou službu, která bude provádět konverzi LaTeX souborů do PDF a bude přijímat požadavky skrz REST rozhraní. Služba bude sloužit uživatelům Moodle a CourseWare pro jednodušší upravování LaTeX souborů, bez potřeby kompilace na jejich stroji. Uživatelé budou moci definovat parametry a na základě nich se daný soubor zkompiluje.
- Seznamte se s potřebnými technologiemi
- Analyzujte a definujte požadavky na službu
- Na základě požadavků vyhledejte existující řešení a analyzujte je
- Navrhněte vlastní řešení a porovnejte s existujícími
- Implementujete vlastní řešení
- Navrhněte testovací scénáře a otestujte funkčnost
- Nasaďte do produkčního prostředí

Seznam doporučené literatury:

Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, PhD Dissertation, University of California, Irvine, 2000.
The Java EE 6 Tutorial, https://docs.oracle.com/javaee/6/tutorial/doc/docinfo.html, 2013

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Lukáš Zoubek, Centrum znalostního managementu FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **24.01.2019** Termín odevzdání bakalářské práce: **24.05.2019**

Platnost zadání bakalářské práce: **20.09.2020**

Ing. Lukáš Zoubek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Poděkování

Především bych chtěl poděkovat svému vedoucímu Ing. Lukáši Zoubkovi za příjemnou spolupráci a cenné rady při našich konzultacích. Také bych chtěl poděkovat Bc. Jiřímu Fryčovi za poskytnutí informací a rad a v neposlední řadě mé rodině a kamarádům, kteří mě při psaní této práce podporovali.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze, 21. května 2019

Abstrakt

Práce se skládá z návrhu a implementace webové služby pro konverzi LaTeXu do PDF pro uživatele v Moodle a CourseWare. Služba má uživatelům usnadnit práci s LaTeXovými soubory přímo skrze zmíněné aplikace, bez potřeby stahování a kompilace na svém stroji.

Klíčová slova: LaTeX, PDF, Moodle, CourseWare, Webová služba, REST, Java EE

Vedoucí: Ing. LUKÁŠ ZOUBEK
Katedra ekonomiky, manažerství a humanitních věd

Abstract

Thesis consists of design and implementation of web service providing conversion of LaTeX to PDF. This service will be available for users of Moodle and CourseWare. It helps them to create PDF inside of web browser instead of compiling it on their computer.

Keywords: LaTeX, PDF, Moodle, CourseWare, Web service, REST, Java EE

Title translation: Creation of REST service for conversion from LaTeX to PDF

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

LaTeX patří v akademickém prostředí mezi velmi oblíbené typografické systémy a je hojně využíván ke psaní skript a odborných prací. Mnoho akademiků využívá LaTeX i k vytváření materiálů pro studenty, primárně v oblasti matematiky, jelikož nabízí jednoduché nástroje ke psaní vzorců. Hlavním zdrojem materiálů pro studenty jsou portály Moodle a CourseWare, kam vyučující dokumenty nahrávají a mohou je tam i upravovat. Nynější editor podporuje jenom řádkové příkazy a není schopen zpracovat celý LaTeX dokument natož s více zdrojovými soubory. Tento stav mnoha učitelům nevyhovuje, protože by chtěli svoje dokumenty upravovat a přímo kompilovat v prohlížeči bez potřeby je stahovat a následně zase nahrávat.

Práce se dělí na několik částí, které je potřeba splnit k úspěšnému dokončení projektu. Nejdříve proběhne seznámení s potřebnou teorií a technologiemi, dále budou specifikovány požadavky na službu, poté budou představena již podobná existující řešení a porovnají se s požadavky. Následně se přistoupí k návrhu samotné aplikace na základě něhož bude naimplementována. Nakonec bude služba otestována a zhodnocena.

1.1 Cíle práce

Hlavním cílem práce je poskytnout konverzi LaTeX souborů do PDF pro uživatele Moodle a CourseWare. V semestrálním projektu budou rozpracovány tyto dva dílčí cíle: analýza a návrh.

1.1.1 Analýza

V analýze jsou stanoveny tyto cíle:

- Získat a zformulovat požadavky na službu.
- Naleznout již existující řešení, zabývající se touto problematikou a porovnat je vůči požadavkům.
- Porovnat LaTeX kompilátory.

1.1.2 Návrh

V návrhu budou naplněny tyto cíle:

- Navrhnout řešení daného problému.
- Na základě požadavků stanovit technologie.

1.1.3 Implementace

V rámci implementace budou dosaženy tyto cíle:

- Naimplementovat funkční aplikaci.
- Otestovat aplikaci.

Kapitola 2

Teorie a technologie

V této části si přiblížíme technologie potřebné k návrhu a vývoji webové služby specifikované v zadání práce. Postupně budou vysvětleny všechny zásadní pojmy, které pomohou čtenáři doplnit znalosti v dané problematice.

2.1 REST

Representational state transfer (dále jen REST) je architektura pro komunikaci mezi distribuovanými systémy. Termín zavedl R. T. Fielding ve své disertační práci[4], kde toto rozhraní bylo taktéž popsáno a vymezeno. Mimo jiné stanovil i těchto 5 základních pravidel, která by měla být dodržena, aby se aplikace nazývala RESTful[6]:

- Klient-Server (Client-Server) - Toto omezení staví na principu oddělení zodpovědností (Separation of Concerns), nebo-li existuje klientská část starající se o uživatelské rozhraní a ta je oddělena od serverové části přistupující k databázi. Zlepšuje škálovatelnost sytému a zjednodušuje použitelnost na různých platformách.
- Bezestavovost (Stateless) - Každý požadavek musí přenášet všechna související data, server totiž neuchovává žádné informace o nynějším spojením a každý požadavek bere jako nový.
- Keš (Cache) - Data přenášená v odpovědi mohou být označená jako kešovatelná, tudíž si je klient může uložit a kdykoliv použít znova.
- Jednotné rozhraní (Unified interface) - Základem tohoto omezení je princip HATEOAS (Hypermedia As The Engine Of Application State), který říká, že klient nepotřebuje znát pravidla komunikace dopředu a data musí obsahovat odkazy na další data v aplikaci. Měla by být jasně definovaná adresa zdroje (např. URI), reprezentace přenášených dat (např. HTML) a typ média (např. JSON).
- Vrstvený systém (Layered system) - Přidáním vrstev se aplikace, kde každá vrstva je izolovaná a může komunikovat jenom se sousedícími vrstvami, zpřehlední a zlepší se její škálovatelnost.

Nejčastějším typem protokolu využívající tuto architekturu je Hypertext Transfer Protocol(HTTP). Pomocí čtyř hlavních metod GET, PUT, POST, DELETE v požadavku

poslaném z klienta server buď vrátí požadovaná data, přijme data poslaná v těle a uloží do databáze, nebo data vymaže.

2.2 Webová služba

Webová služba[8] představuje systém, který byl navrhnut pouze pro komunikaci s jiným strojem skrze internet. Samotná služba s nikým neinteraguje, pouze čeká na požadavky od ostatních systémů, které jí využívají. Komunikace probíhá nejčastěji přes HTTP, ale je možné používat i jiné protokoly. Informace bývají ve formátu XML, nebo častěji v JSON, který se za poslední roky stal velmi populární, díky čitelnosti a přehlednosti schématu. Dále je potřeba mít také stanovené rozhraní, přes které probíhá interakce, to může být například REST nebo SOAP(Simple Object Access Protocol). Hlavní smyslem služby je poskytování jejích zdrojů a funkcionalit ostatním systémům, které k ní přistupují.

2.3 PDF

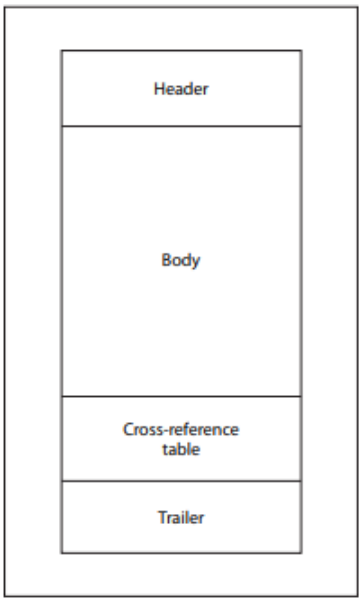
Portable Document Format (dále jen PDF), jak už název napovídá, představuje formát dokumentů, jejichž hlavním cílem je poskytování nezávislosti na platformě. Je velmi podobný programovacímu jazyku PostScript, ze kterého vzešel. Syntaxe souboru je nejlépe pochopitelná jako čtyři komponenty[1]:

- Objekty (Objects) - Základní jednotka celého souboru např. Pole, Čísla, Řetězce znaků. Jednotlivé objekty se popisují množinou znaků, která je definována lexikálními konvencemi.
- Struktura souboru (File Structure) - Samotný soubor se skládá ze čtyř částí, jak je i vidět na obrázku 2.1, ty si stručně popíšeme:
  - hlavička - identifikuje verzi PDF
  - tělo - obsahuje použité objekty, které reprezentují obsah dokumentu
  - tabulka odkazů - obsahuje odkazy na objekty v podobě počtu bytů od začátku souboru, kvůli náhodnému přístupu bez potřeby číst celý soubor
  - závěrečná sekce - udává pozici tabulky odkazů a speciálních objektů

  Tato struktura napomáhá náhodnému přístupu k jednotlivým částem a usnadňuje jejich aktualizaci.
- Struktura dokumentu (Document Structure) - Popisuje hierarchickou strukturu objektů v těle dokumentu.
- Proud obsahu (Content stream) - Objekt, ve kterém se nacházejí instrukce k vykreslování grafických elementů. Každá stránka má minimálně jeden a na rozdíl od ostatních objektu je procházen sekvenčně.

Vzhledem k povaze této práce je potřeba také zmínit systém práv a povolení. Existují dva typy práv, která jsou rozlišeny na základě poskytnutého hesla. Buď je zadáno uživatelské heslo, které umožní uživateli jenom otevřít daný soubor. Nebo heslo vlastníka, díky kterému je možno upravovat uživatelské heslo a jeho práva k manipulaci se souborem. Jde povolit tyto možnosti:

- Upravování obsahu dokumentu
- Kopírování textu a obrázků
- Vyplňování formulářu a polí
- Tisknutí dokumentu



Obrázek 2.1: Struktura objektů (převzato z [1])

2.4 LaTeX

Vychází z typografického sázecího systému TEX, který popisuje Pavel Satrapa ve své knize [7]: „*Patří do rodiny tak zvaných značkovacích jazyků (markup languages) a dal by se zjednodušeně charakterizovat jako programovací jazyk pro sazbu textů. Jeho základním vstupem je textový soubor, který obsahuje jak sázený dokument, tak příkazy ovlivňující sazbu. Určité znaky mají přiřazen speciální význam a jejich prostřednictvím jsou v textu odlišeny řídicí konstrukce. Typickým příkladem je zpětné lomítko, jímž začínají příkazy.*“

LATEX je rozšíření TEXu o balíček přednastavených řídicích konstrukcí. Hlavní cílem těchto systémů je jednoduchost při psaní matematických a jiných vzorců. Ovšem je také velmi oblíben kvůli možnosti jednoduše upravovat dokumenty podle potřeby, přestože prvotní seznámení je ve srovnání s jinými nástroji ke psaní náročnější.

Základní soubory mají příponu tex kromě souboru s nastavením, ten je ve formátu cls, neboli class file. Tyto soubory mohou být uspořádány do stromové struktury, kde v kořenu stromu je jeden hlavní soubor, do kterého jsou vnořovány další. To napomáhá přehlednosti velkých dokumentů a také používání již vytvořených.

Soubor se skládá z preambule, která obsahuje nastavení pro celý dokument a případně používané balíčky. Ve druhé části dokumentu se mimo jiné nacházejí kapitoly, sekce a podobně. Základní syntaxe může vypadat takto.

```
\documentclass{thesis}
\usepackage{csquotes}

\begin{document}
Hello world!
\end{document}
```

Výše zmíněné balíčky slouží k přidávání dalších předvytvořených řídicích konstrukcí. Tyto balíčky jsou buď nainstalovány zároveň s instalací kompilátoru, nebo je možné je ručně doinstalovávat podle potřeby.

Pro získání výsledného dokumentu, například ve formátu PDF, je potřeba vše zkompilovat. To provádí kompilátor.

Dalším podporovaným formátem je DVI (Device Independent file format), což je binární soubor, k jehož interpretaci je potřeba řadič. Je velmi podobný PDF, hlavní rozdíl je v tom, že DVI nefunguje jako úplný programovací jazyk a není tak univerzální a rozšířený napříč platformami.

2.5 Moodle a CourseWare

Ačkoliv tyto portály se v mnoha aspektech liší, v rámci této práce je můžeme považovat za ekvivalentní. Oba slouží pro podporu studia, a to hlavně v podobě poskytování informací o předmětech studentům, kteří je mají zapsané. Vyučující takto může předat informace o harmonogramu kurzu, přednášky a jiné materiály. Moodle je komplexnější open-source systém s možností upravovat ho k potřebě vzdělávacího zařízení, který navíc poskytuje prostředí pro vytváření a psaní testů, zapisování docházky, komunikaci se studenty a jiné.

2.6 FURPS+

Vychází z klasifikace požadavků FURPS(**F**unctionality, **U**sability, **R**eliability, **P**erformance, **S**upportability) se kterou přišel Robert Grady v roce 1992. Roku 1999 Jacobson at el rozšířili specifikaci o znaménko „+“, které přidává požadavky a omezení na návrh, implementaci, rozhraní a hardware[3].

- Funkční požadavky
  - Fukčnost (Functionality) - Požadavky popisující všechny hlavní prvky produktu i důležité aspekty z pohledu architektury např. lokalizovaný systém pro více jazyků.

- Nefunkční požadavky
  - Použitelnost (Usability) - Zaměřuje se na uživatelskou přívětivost nejenom samotné aplikace, ale i dokumentace týkající se estetiky a konzistence.
  - Spolehlivost (Reliability) - Spolehlivost systému v podobě doby běhu, správnosti fungování a četnosti výpadků.
  - Výkon (Performance) - Vypovídá o výkonnosti systému, jak rychle dokáže zpracovávat požadavky, spustit se atd.
  - Podporovatelnost (Supportability) - Popisuje testovatelnost, škálovatelnost, konfigurovatelnost...
  - Návrh (Design) - Omezení na návrh systému např. požadavek na relační databázi
  - Implementace (Implementation) - Specifikuje typ programovacího jazyku, platformu apod.
  - Rozhraní (Interface) - Komunikace s externími systémy.
  - Fyzické (Physical) - Definuje požadavky na hardware, na kterém daný software poběží, i co se týče fyzické velikosti.

Toto rozdělení nám pomáhá identifikovat požadavky. Přispívá k vyšší kvalitě systému a snižuje pravděpodobnost přehlédnutí funkcionality. Právě díky těmto vlastnostem je velmi oblíbené a využívané k vývoji jakéhokoliv software.

Kapitola 3
Analýza

3.1 Požadavky

Nyní představíme požadavky kladené na aplikaci. Ty jsou buď požadované zadávajícím, nebo odvozené z potřeb, ke kterým bude služba používána, ale i z omezení plynoucích z prostředí, v jakém bude provozována, v tomto případě z prostředí fakulty státní vysoké školy, jmenovitě Fakulty elektrotechnické, ČVUT.
Za pomoci metody FURPS+ rozdělíme požadavky a omezení na funkční a nefunkční.

3.1.1 Funkční požadavky

Typ	Požadavky

Funkčnost	• Webová služba s REST rozhraním pro komunikaci
	• Umět přijmout požadavky a soubory
	• Zkompilování LaTeX souborů s požadovaným nastavením do PDF
	• Uložení PDF a jeho poskytnutí
	• Uživatel může zvolit tyto nastavení:
	  • Přidat vodoznak na každou stranu PDF
	  • Nakonec souboru přidat stránku s předem stanoveným obsahem
	  • Výsledný soubor PDF bude chráněný
	  • Výsledný soubor PDF nebude kopírovatelný nebo tisknutelný

Tabulka 3.1: Funkční požadavky

Služba má jediný a specifický účel, z toho plyne menší množství funkcionalit.

3.1.2 Nefunkční požadavky

Typ	Požadavky
Použitelnost	• Dokumentace k rozhraní na swagger [1] • Přístup skrz REST Api, pouze s tokenem[2]
Spolehlivost	• Služba není kritická • Uptime 95% času • Ochrana proti špatnému vstupu • Autorizace pomocí tokenů
Výkon	• Zvládnutí obsloužení desítek požadavků najednou • Ukládání výsledných dokumentů po dobu jednoho měsíce • Maximální doba tvorby dokumentu 5 minut
Podporovatelnost	• Služba může být rozšířena o kompilování samotného TeXu a může podporovat vytváření i jiných formátů z LaTeXu
Implementace	• Platforma - Java EE • Komunikace - REST Api • Operační systém - Debian nebo CentOS
Rozhraní	• Komunikuje s Moodle a CourseWare, tyto portály posílají data
Fyzické	• Musí být provozováno na serverech ČVUT

Tabulka 3.2: Nefunkční požadavky

Důvody některých požadavků nemusejí být úplně jasné, proto je zmíníme.

- Platforma - Je požadováno prostředím vývoje, kde Java EE je hlavní platforma. Tudíž je zajištěna podpora.
- Operační systém - Také vyžadováno z pohledu podpory, zmíněné systémy jsou na serverech, kde bude služba nasazena nejčastěji používány a tudíž správci tyto systémy znají.
- Služba není kritická - Poskytuje funkčnost, která nijak neovlivňuje základní funkcionality určených portálů.
- Musí být provozováno na serverech ČVUT - Jelikož aplikace bude pracovat s dokumenty a popřípadě i informacemi, spojenými s pracovníky školy, je potřeba tato data chránit a uchovávat na vlastních serverech z důvodu GDPR.

[2] https:/swagger.io/

Z tabulky je vidět, že služba není nijak kritická a působí jenom jako doplněk výše zmíněných portálů. Musí ovšem splňovat vyšší bezpečnostní nároky související s prostředím, v jakém se bude používat.

3.2 Existující řešení

Na základě požadavků stanovených v minulé kapitole přejdeme k analýze již existujících řešení. Momentálně uživatelé Moodle mohou vkládat do svých souborů řádkové příkazy, což není úplně dostačující a nesplňuje to požadavky. Samozřejmě se dají používat pro vytváření PDF z LaTeX souborů kompilátory, které lze stáhnout a používat lokálně. To ovšem není předmětem této práce, a proto se podíváme na webové služby, které více odpovídají potřebám a požadavkům. Pár vybraných si představíme.

3.2.1 OverLeaf

Placená služba[3] pro tvorbu LaTeX dokumentů, kterou lze s některými omezeními používat i zdarma. Je velmi oblíbená hlavně kvůli přívětivému prostředí pro tvorbu dokumentů a jejich správu. Také nabízí výhody pro určité zájmové skupiny, nejzajímavější vzhledem k tématu této práce je předplatitelská služba OverLeaf Commons [4]. Ta poskytuje sdílené prostředí se všemi výhodami pro zaměstnance a studenty univerzity. Ale jako všechny ostatní služby je i tato placená, ovšem není jisté, jestli je poskytována všem univerzitám a ani za jakých podmínek.

3.2.2 BlueLaTeX

Open-source služba[5] pro kompilaci LateX souborů. Kompilovat lze na jejich serveru nebo nabízejí kód pro spuštění na vlastním. Kromě samotné serverové implementace je k dispozici i webový klient. Vše je zatím pouze v Beta verzi a samotní tvůrci varují před možnými nedostatky a problémy. Na jejich oficiálních stránkách je poslední aktivita z roku 2015 a zdá se, že tento projekt už není aktivní. Hlavním lákadlem je souběžná spolupráce více lidí na jednom dokumentu a také možnost provozovat server lokálně.

3.2.3 ScienceSoft

Starší služba[6], která mimo kompilace LaTeX souborů vložených přes webový prohlížeč poskytuje i jiné rozhraní pro poskytnutí souborů, a to jmenovitě REST Api a SOAP. Jak bylo řečeno, tak tato služba je starší, tudíž pro kompilaci používá zastaralý TexLive 2008 a její vývoj není aktivní.

[3] https://www.overleaf.com
[4] https://www.overleaf.com/for/universities
[5] http://www.bluelatex.org/
[6] http://sciencesoft.at/latex/index?lang=en

3.2.4 ShareLaTeX

Velmi podobný BlueLaTeXu, tedy open-source[7] nabízející jimi hostovanou verzi nebo lokální verzi pro vlastní potřebu, obě verze obsahují mnoho funkcí včetně grafického prostředí pro uživatele. Pod jménem ShareLaTeX se vyskytuje jenom výše zmíněné, ovšem společně s OverLeaf také spravují službu Pro[8], která je určená pro firmy, ale i univerzity. Ta se pyšní možností nasazení na vlastních serverech, správou uživatelů, podporou a zabezpečením.

3.2.5 Porovnání

Každé z těchto řešení má své problémy, ať už že je zastaralé a neaktualizované nebo nesplňující zanalyzované požadavky(sekce 3.1). Do první skupiny patří ScienceSoft a BlueLaTex, kde první z jmenovaných ani neposkytuje kód pro implementaci na lokálním serveru a druhý je v nedodělaném stavu, což může vést k nefunkčnosti a bezpečnostním rizikům. Služba Overleaf Commons nesplňuje stejný požadavek jako řešení od ScienceSoft, ale nabízí zajímavé prostředí pro vytváření a správu LaTeX dokumentů pro studenty i zaměstnance, o čemž by univerzita mohla popřemýšlet. Nejnadějnější možností se zdá být ShareLaTeX, který poskytuje k použití i jenom backendovou část pro kompilaci LaTeX a komunikaci, ale celý je implementovaný v CoffeeScriptu, což je v rozporu s požadavkem na implementaci, kde je Java EE.

3.3 Kompilátory

Nejdůležitější částí celé aplikace bude kompilátor, který bude provádět kompilaci LaTeX souborů, ale může poskytnout i podporu pro operace s výsledným PDF. Tyto operace vycházejí ze stanovených funkčních požadavků (sekce 3.1.1). Jelikož kompilace bude probíhat na serveru s Linuxovým operačním systémem, nebudeme zahrnovat do srovnání kompilátory pro Windows.

3.3.1 MikTeX

MikTeX [9] je velmi oblíbený hlavně na operačním systému Windows, a to díky příjemné instalaci a kvůli možnosti doinstalovávat balíčky „on-the-fly“, neboli za běhu. Ale je poskytován mimo jiné i pro Linux, a to například pro potřeby této práce v zajímavé verzi „Just enough TeX“. Tato instalace obsahuje jen to nejnutnější, tedy bez zbytečných balíčků navíc. Je vyvíjen jediným programátorem, který se stará o celou distribuci, což je do budoucna poněkud rizikové z pohledu udržitelnosti. Podporované Linuxové operační systémy jsou např. nejnovější Ubuntu a Debian.

[7] https://github.com/sharelatex/clsi-sharelatex
[8] https://www.overleaf.com/for/enterprises
[9] https://miktex.org

3.3.2 TeXLive

TeXLive je kompilátor spravovaný skupinou přispěvatelů, kteří ho udržují. Existují dvě různé cesty jak TeXLive[10] nainstalovat a od toho se odvíjí správa balíčků. Verze Native TeX Live a distribuce přímo pro daný operační systém. Liší se počtem balíčků po instalaci a způsobem jejich aktualizací. V Native verzi probíhají aktualizace pomocí TeX Live Manager neboli tlmgr, jinak se o to stará samotný operační systém. V případě verze Native je možno si i zvolit schéma, které určuje množství balíčků např. scheme-basic obsahuje jenom to nejnutnější. Podporuje skoro všechny Linuxové distribuce.

3.3.3 Porovnání

Oba kompilátory jsou si velmi podobné a liší se jenom v drobnostech, některé z nich už byly nastíněny v jejich popisech. Jedním z důležitých aspektů je velikost instalace, tedy i s balíčky, v této kategorii nabízejí oba to samé. K tomuto se váže práce s balíčky, kde už je jiná situace, a to kvůli jasné výhodě MikTeXu spočívající ve stahování balíčků za běhu. Něco podobného lze dosáhnout i v TexLive, ale je nutno použít externí skripty, což může mnohem více ovlivňovat rychlost kompilace. Rozdíl je také ve správě samotných kompilátorů, kde MikTeX je náchylnější k výpadkům aktualizací nebo celkovému zastavení vývoje, a to kvůli jedinému člověku, který se o něj stará. To může vést i k dalším problémům, co se týče bezpečnosti a podpory.

[10] https://www.tug.org/texlive/pkginstall.html

Kapitola 4

Návrh vlastního řešení

V této části navrhneme podobu vlastního řešení problému specifikovaného v předchozích kapitolách. Návrh by měl nabízet jednoduché a vyhovující řešení vycházející ze stanovených požadavků a cílů.

Cílem práce je poskytnout konverzi LaTeX souborů do PDF, čehož bude dosaženo pomocí webové služby. Tato služba bude poskytovat svoje rozhraní a funkcionalitu portálům Moodle a CourseWare. Jedná se o jednoduchou aplikaci s jediným účelem, tedy kompilace LaTeX souborů do PDF. Proto data budou ukládány do souborového systému.

4.1 Platforma

Na základě požadavků bude aplikace postavena na Java EE[2], což je platforma pro vývoj webových aplikací rozšiřující standardní Javu SE. Oproti ní poskytuje některé zásadní techniky navíc, jednu si tedy představme.

Vkládání závislostí (dependency injection) umožňuje objektu používat jiné objekty bez potřeby ho zatěžovat jejich vytvářením. Objekty, které můžeme takto vkládat, se nazývají beany a právě o jejich vytváření a zánik se stará Contexts and Dependency Injection (dále jen CDI) kontejner.

Dále je potřeba specifikovat aplikační server. Ten poskytuje pro webové aplikace běhové prostředí, tedy zajišťuje správu databázových spojení apod. Na základě zkušeností je vybrán open-source Payara, který staví na GlassFish, oproti němu poskytuje častější aktualizace a opravy chyb.

4.2 Architektura

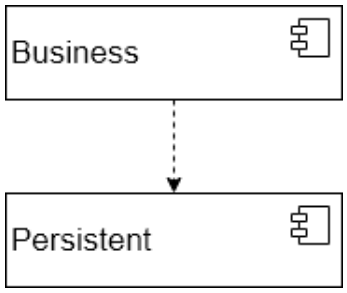
Architektura[5] popisuje, z jakých částí se aplikace skládá, jak mezi sebou tyto části komunikují a jakým způsobem procházejí informace a požadavky aplikací. Při navrhování architektury je potřeba zvážit několik věcí, například rozšířitelnost, modularizaci, složitost a pro jaký případ má sloužit.

Nejpoužívanější architekturou u webových aplikací je klient-server architektura s n vrstvami, která aplikaci rozděluje na n fyzických médií, a n vrstvá architektura, která

označuje n logických celků.[1] Nejčastěji jsou obě architektury 3 vrstvé, tedy rozdělení je následující: prezentační, aplikační a databázová vrstva. Ovšem služba, kterou se zabýváme v této práci, není plnohodnotná webová aplikace, tudíž nemá uživatelské rozhraní a vystavuje jenom určité API, odpadá tedy prezentační vrstva. Ani neobsahuje databázový server, ve výsledku zůstává jenom aplikační fyzická vrstva, což z naší služby dělá jedno-vrstvou klient-server aplikaci 4.1. Zbývá ještě vyřešit logickou architekturu, kde se budeme držet také zavedených postupů, ale zbavíme se prezentační vrstvy a zůstane nám 2-vrstvá architektura obsahující byznys logiku a přístup k databázi 4.2.

Obrázek 4.1: Klient server diagram

To znamená, že program přijíme požadavek od klienta, podle typu ho zpracuje v byznys vrstvě a případně předá datábázové pro persistentní uložení. Byznys vsrtva bude obsahovat celou logiku aplikace, mimojiné kompilaci a úpravu PDF.



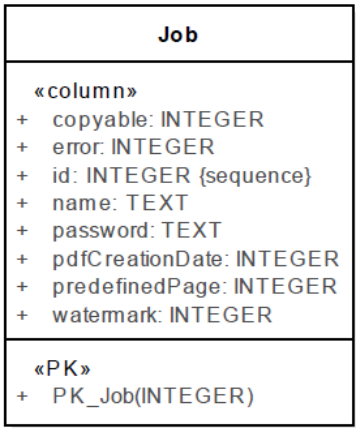
Obrázek 4.2: Diagram vrstev

4.3 Databáze a entitní model

Jak bylo řečeno, jedná se o jednoduchou aplikaci, tudíž není zapotřebí složité databáze s vlastním serverem. Vystačíme si s SQLite[2], což je velmi odlehčená verze klasických SQL databází, jejíž databáze je jenom jednoduchý soubor na disku, do kterého zapisuje napřímo. Na základě dostupných informací z požadavků a cílů služby víme, že bude stačit jediná tabulka, která bude držet hlavně informace o požadavcích klienta. Dále také jméno hlavního tex souboru, které je nutné pro kompilaci, a primarní klíč id, který slouží pro identifikaci. Mimo jiné také obsahuje vnitřní stavy průběhu zpracování LaTeX souborů. Všechny atributy jsou vypsány v diagramu 4.3, který je vyobrazen níže.

[1] V anglických zdrojích se používají výrazy N-Tier Client-Server architecture a N-Layered architecture, kde tier znamená to samé jako layer, tudíž vrstva

[2] https://www.sqlite.org/



Obrázek 4.3: Entitní diagram

4.4 Komunikace

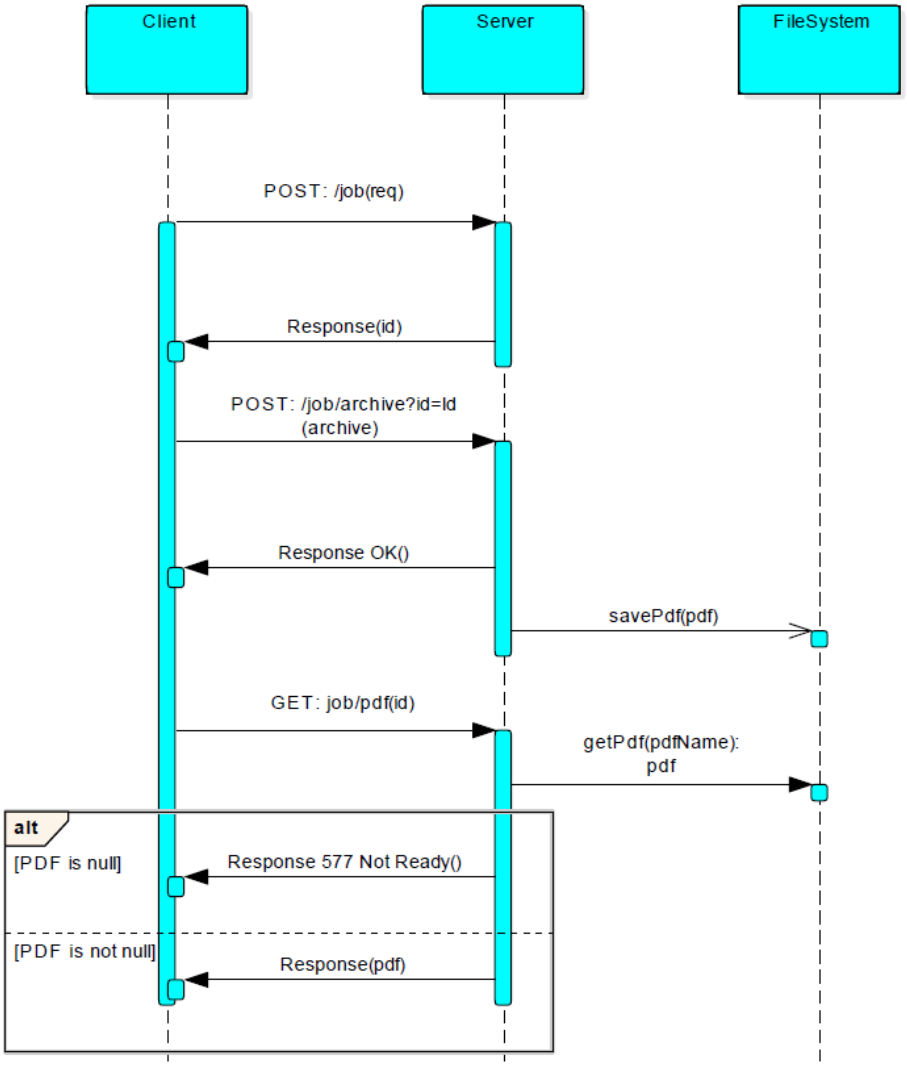
Pro komunikaci mezi klientem a serverem je vybráno REST Api. Poskytuje jednoduché rozhraní, se kterým je schopen komunikovat jakýkoliv systém pouze na základě znalosti struktury požadavků a odpovědí. Posílání zpráv bude postaveno nad protokolem HTTP pomocí GET a POST metod. Na obrázku 4.4 je zobrazena komunikace mezi serverem a klientem.

Kompletní dokumentace k rozhraní se nachází na swagger[3]. Na ukázku vypíšeme alespoň používané zdroje:

- **POST** `/job` - Slouží k předání požadavků na výsledné PDF.
- **POST** `/job/archive?id=<jobId>` - Musí obsahovat zip archiv s potřebnými soubory pro kompilaci. Jméno archivu musí být stejné jako hlavního texu.
- **GET** `/job/pdf?id=<jobId>` - Vrací výsledné PDF, pokud už je hotové.
- **HEAD** `/job/pdf?id=<jobId>` - Odpověď obsahuje pouze hlavičku, která informuje o stavu PDF.
- **GET** `/info` - Slouží k získání informací ohledně serveru, tedy verze kompilátoru, operačního systému, zbývajícího volného místa na disku.

Všechny tyto zdroje navíc obsahují parametr[4] token, který zajišťuje autentizaci. Ta je rozebrána v další sekci.

[3] https://app.swaggerhub.com/apis/Tadky/Thesis/1
[4] Typ parametru, který se posílá v samotné URL daného dotazu ve formě klíč=hodnota



Obrázek 4.4: Sekvenční diagram

4.5 Zabezpečení

Zabezpečení bude zavedeno jenom v minimální míře pomocí statických tokenů, které budou mít dané portály pro sebe k dispozici. Vzhledem k povaze dat není nutné používat OAuth[5] server, jehož implementace by byla značně nad rámec mé práce. Tokeny budou sloužit k autentizaci portálů, jímž budou tokeny vygenerovány a předány jejich správcům. Každá zpráva poslaná na server bude obsahovat token, který se bude ověřovat oproti tokenu uloženém v aplikačním serveru jako JNDI zdroj v podobě properties[6] souboru. Token bude posílán v parametru dotazu v otevřené formě.

[5] Protokol pro autentizaci a autorizaci aplikací.
[6] Soubor, v němž jsou data uspořádana klíč = hodnota

4.6 Kompilace

Pro vytvoření PDF je nutné příslušně zkompilovat celý LaTeX projekt. Pro úspěšnou kompilaci je potřeba, aby kompilátor měl k dispozici všechny balíčky, které jsou použity v projektu. Jsou tři způsoby, jak toho docílit.

1. Stáhnout všechny balíčky již při instalaci kompilátoru
2. Nainstalovat čistý kompilátor
   a. Stáhnout několik balíčků a jenom ty se budou používat
   b. Získávat balíčky podle potřeby za běhu

První možnost je zbytečná z důvodu mnoha nepoužívaných balíčků a velikosti na disku. Druhá možnost nabízí dvě podmožnosti, kde volba s přednastavenými balíčky vytváří omezení pro uživatele tudíž je může odrazovat od používání aplikace. Nejvíce vhodná se zdá poslední možnost, která eliminuje všechny neduhy předchozích, tím pádem je i zvolena.

Na základě porovnání kompilátorů (sekce 3.3.3) z předchozí kapitoly je zvolen MikTeX, který poskytuje vše potřebné a nabízí vhodné funkcionality pro téma této práce např. doinstalovávání balíčku za běhu. Bude nainstalován ve verzi „Just enough TeX“, tudíž nebude obsahovat žádné balíčky. Ty se právě budou doinstalovávat až na požadavek při kompilaci.

4.7 PDF úpravy

Jelikož jsou kladeny požadavky na výsledný soubor PDF a bohužel kompilátory tyto úpravy nepodporují, je potřeba použít jiný nástroj, který bude umožňovat měnit soubor podle požadavků. Nejvíce vhodný se zdá PDFtk[7], který je pod GPL[8] licencí. Nabízí mnoho možností úprav, pro naše použití jsou nejdůležitější tyto: spojední dvou PDF souborů do jednoho, přidání vodoznaku, zaheslování souboru a omezení práv na něm. Tedy po zkompilování do PDF se za pomoci výše zmíněné aplikace aplikují požadavky, které byly obdrženy od klienta.

4.8 Server

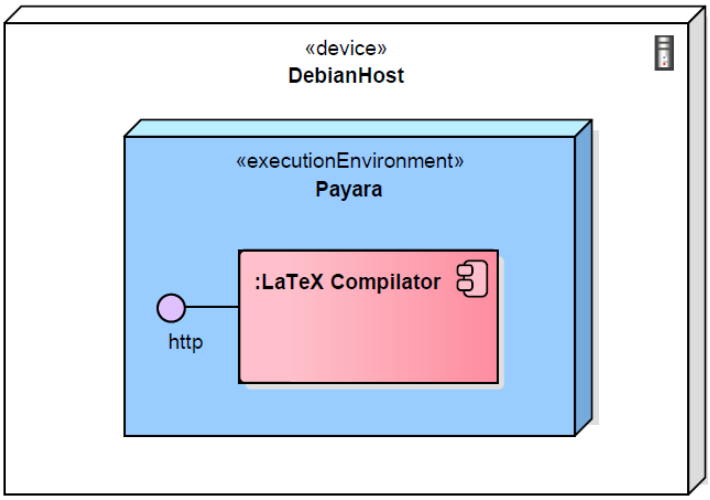
V této fázi návrhu máme už všechny potřebné informace, aby mohl být zvolen operační systém a stanoveny požadavky na server. Jsou známy dvě omezující podmínky kladené na systém. První vychází z požadavků: operační systém musí být Debian nebo CentOS. Druhou určuje kompilátor. Jelikož byl zvolen MikTeX, který je podporován na Debian a Ubuntu, volba je jednoznačná. Na základě průniku je vybrán nejnovější Debian, tedy verze 9 s označením „stretch“.

Je také potřeba stanovit velikost diskového pole potřebného pro fungování. Jelikož nejvíce místa budou zabírat balíčky a potažmo výsledné PDF, budeme vycházet hlavně

[7] https://www.pdflabs.com/tools/pdftk-server/
[8] Poskytuje svobodu šíření, provozování a upravování daného software.

z těchto údajů. Přibližná velikost všech balíčků obsažených v plné verzi kompilátoru je zhruba 4GB, což bylo zjištěno na základě testovací instalace. Dále je potřeba odhadnout velikost všech PDF, které musí server uchovávat po dobu jednoho měsíce, v jeden moment. Horní odhad pro počet vytvořených PDF za den je 30 a pro velikost souboru je 10MB. Tedy na konci měsíce se může očekávat velikost cca 10GB. Tedy celkové požadované místo je 15GB.

Na diagramu 4.5 je vidět, jaké HW a SW komponenty budou použity a jak na sobě závisí. Služba poběží na webovém serveru Payaře, která spravuje HTTP spojení. Samotná Payara bude spuštěn na linuxovém operačním systému Debian.



Obrázek 4.5: Diagram nassazení

Kapitola 5
Implementace

Kapitola implementace pojednává o praktické části, která navazuje na návrh a ztělesňuje ho do funkční podoby.

5.1 Projekt

Jak už bylo zmíněno v sekci architektura, aplikace nemá prezentační vrstvu a sestává se jenom z aplikačního serveru, který má na starosti komunikaci s klienty, byznys logiku a persistenci dat.

Pro sestavení projektu je využit Maven, který nabízí jednoduché inkludování externích balíčků a také pluginy pro sestavování a nasazování aplikace.

5.1.1 Java EE

Celý projekt je naimplementován v Javě EE verze 8. Ta obsahuje nové DateTime API, Streams API aj., dále například podporuje použití lambda výrazů, což zkracuje zápis a umožňuje například předávání funkcí. Tato funkcionalita byla využita při přijímání požadavků 5.1 a kontrolování tokenů 5.2. Dalším důležitým aspektem Javy EE jsou beany, které už byly představeny v sekci 4.1. Na tyto informace navážeme jejich anotacemi, které byly použity v rámci projektu. Většina bean je typu stateless, což znamená, že si nepamatují stav napříč požadavky jednoho klienta, jenom po dobu jednoho daného požadavku, poté je objekt vrácen do poolu a může ho využít jiný klient. Dále je použita request scope beana, která je vytvořena s každým novým http požadavkem a není zničena, dokud je požadavek aktivní. Objekt *Starter*, který je použit pro prvotní procesy po naběhnutí aplikace a objekt *PeriodicalCleaning*, který periodicky promazává nepotřebné soubory, jsou typu Singleton. Což je beana, která je vytvořena jenom jednou a je globálně sdílená pro celou aplikaci. Byly využiti i jiné anotace a funkce, ale pro představu tyto stačí.

5.1.2 Struktura projektu

Nyní si představíme strukturu projektu, tedy diagram balíčků, který zobrazuje tři hlavní balíčky. První se jmenuje Api, který slouží pro zpracování požadavků (detailněji popsáno v sekci 5.3), ten obsahuje tři další složky jmenovitě:

- Controllers - Třídy zdrojů obsahující metody pro přijímání požadavků

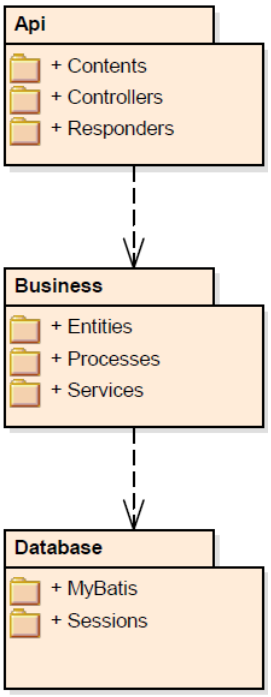
- Responders - Třídy, které se starají o sestavování odpovědí klientům
- Contents - Třídy představující objekty pro serializaci a deserializaci JSON formátu

Další balíček v pořadí je Business, ten představuje všechnu logiku, která je aplikována na data. Nejdůležitějším balíčkem je Processes, ale zase si popíšeme všechny.

- Services - Třídy obsahující hlavní logiku, tedy rozhodují, co dělat s příchozími daty
- Processes - Třídy pro kompilaci a úpravu PDF
- Entities - Třídy představující objekty používané v aplikaci

Poslední se jmenuje Database a obsahuje tyto složky:

- Sessions - Třídy pro komunikaci s databází
- MyBatis - Třídy pro mapování SQL dotazů a nastavování spojení

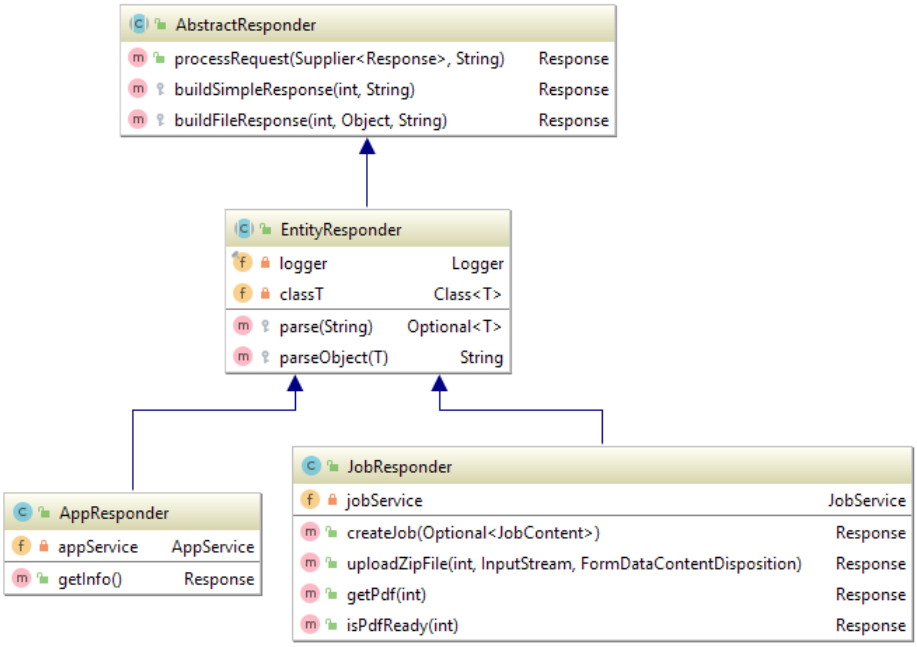


Obrázek 5.1: Diagram balíčků

Projekt obsahuje i další nevyobrazené balíčky, ale ty jsou spíše pomocné a tudíž nejsou tak důležité z pohledu funkčnosti a průchodu aplikací, např. Utils, Exceptions, Enums, atd.

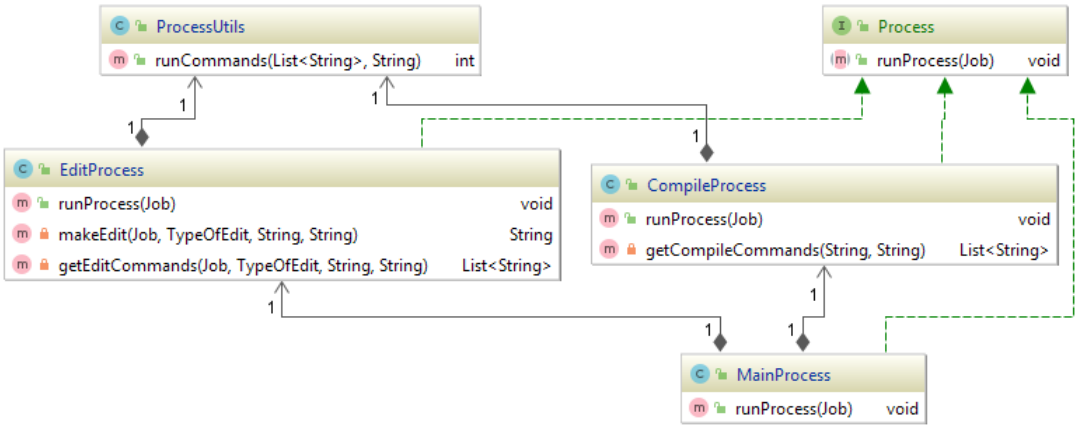
5.1.3 Diagram tříd

Na ukázku byly vytvořeny dva diagramy tříd, ty popisují jejich provázanost. První diagram 5.2 ukazuje hierarchii, kde úplně nahoře je obecná abstraktní třída, která má například metodu *processRequest*, která je popsána v sekci5.3. Pod ní je třída *EntityResponder*, která je také abstraktní a navíc generická, ta představuje už konkrétnější skupinu s metodami pro parsování JSON do objektů daných typem a naopak. Od ní dědí v našem případě dvě třídy, *JobResponder* vytváří odpovědi pro žádosti na adresu /job a *AppResponder* na adrese /app.



Obrázek 5.2: Diagram tříd balíčku Responders

Na druhém diagramu 5.3 je vidět, jaké třídy se hlavně starají o kompilaci a editaci. Všechny kromě *ProcessUtils*, která jenom poskytuje metodu pro samotnou exekuci příkazů, implementují rozhraní *Process* s metodou *runProcess*. Ta je jediná, která by měla být v jednotlivých třídách volána, představuje proběhnutí požadovaného procesu se vším všudy. V našem případě je nejdříve volána metoda z třídy *MainProcess* a ta následně zajistí kompilaci a úpravu za pomoci vyobrazených tříd *CompileProcess* a *EditProcess*.



Obrázek 5.3: Diagram tříd týkající se balíčku Processes

5.2 Aplikační server

V této sekci hlavně rozebereme nastavení serveru. Je použita Payara ve verzi 191 Full, ale dále zmiňované nastavení není nijak závislé na verzi. Bohužel jsem nebyl schopen nastavit JDBC pool v Payaře, což ale bylo vyřešeno jiným způsobem popsaným v sekci 5.5.1. Bylo přidáno pět JNDI zdrojů pro nastavení cesty k hlavnímu adresáři, cesty k souboru s databází, cest ke spustitelným souborům programů MikTeX a PDFtk a tokeny pro přístup v podobě properties. Nic jiného k úspěšnému spuštění aplikace není potřeba.

5.3 Zpracovávání požadavků

Požadavky klienta jsou přijímány skrze REST Api, jak můžeme vidět na ukázce 5.1, kde je vyobrazena funkce, která reaguje na htttp požadavek typu POST na základní adrese /job. Přijímá data ve formátu JSON, které obsahují požadavky na výsledné PDF, ale také token, používaný k autentizaci.

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
public Response postJob(@QueryParam("token") String token, String data) {
 return jobResponder.processRequest(()
                        -> jobResponder.createJob(data), token);
}
```

Ukázka kódu 5.1: Funkce pro přijímání HTTP požadavků

Token je zkontrolován ve funkci *processRequest* 5.2, ta kromě tokenu přijímá i parametr Supplier<Response>, který představuje obecnou funkci bez parametru s návratovou hodnotou typu Response. Tudíž se jako první zkontroluje daný token, pokud je nesprávný, je rovnou vrácena odpověď UNAUTHORIZED, v opačném případě je zavolána předaná funkce pomocí metody *get()*, kterou definuje interface Supplier.

```
public Response processRequest(Supplier<Response> request, String token) {
 Response response = null;
 if (TokenUtils.checkToken(token)) {
  response = request.get();
 } else {
  response = buildSimpleResponse(401, "Bad token");
 }
 return response;
}
```

Ukázka kódu 5.2: Funkce na ověření tokenu v požadavku

Následuje objekt typu Responder, která se stará o sestavování odpovědí klientovi na základě výjimek či výstupů ze servisní vrstvy. Ta obsahuje logiku aplikace, na základě typu požadavku, buď data persistentně uloží, nebo například započne hlavní proces služby, tedy kompilaci. O ukládání dat se starají Session objekty, které si získají spojení s databází a vykonají daný SQL dotaz.

5.4 Kompilace a úprava PDF

Jak bylo uvedeno v návrhu (sekce 4.6), pro kompilování LaTeX souborů je vybrán **MikTeX**. Ten je potřeba nejdříve nainstalovat v příslušné konfiguraci, která byla popsána v téže sekci, vybraná verze je 2.9.7. Samotná kompilace je prováděna pomocí PdfLatex, ten je spouštěn s těmito parametry:

- interaction = nonstopmode - program nečeká na vstup
- include-directory - slouží k určení adresáře, kde se nachází tex soubory
- output-directory - adresář pro výstupní PDF soubor
- aux-directory - adresář pro ostatní výstupní soubory, jako jsou logy
- jméno hlavního tex souboru

Výstupem je tedy soubor PDF, na který je potřeba ještě aplikovat požadavky od klienta. K tomu je použit program **PDFtk** verze 2.02. Pro požadované úpravy jsou potřeba tyto argumenty:

- output - určuje výstupní adresář
- cat - spojuje dva PDF soubory dohromady
- background - přidává vodoznak
- owner_pw a user_pw - první nastavuje heslo vlastníka a druhý heslo uživatele[1]
- allow copycontents - umožňuje kopírovat text
- allow printing - povoluje tisk

[1] Rozdíl je popsán v kapitole 2.3

Jelikož se argumenty kromě output, allow copycontents a printing nedají řetězit, je každá úprava vykonávána samostatně. Pro lepší představu následuje ukázka právě zaheslování a přidání práv pro kopírování a tisk.

```
commands.addAll(Arrays.asList("output", directoryPath + outputFileDir);
commands.addAll(Arrays.asList("owner_pw", job.getPassword()));
commands.addAll(Arrays.asList("user_pw", String.valueOf(job.hashCode())));
if (job.isCopyable() || job.isPrintable()) {
 commands.add("allow");
}
if (job.isCopyable()) {
 commands.add("copycontents");
}
if (job.isPrintable()) {
 commands.add("printing");
}
```

Ukázka kódu 5.3: Zaheslování a úprava práv souboru PDF

Můžeme vidět, že nejdříve se nastaví argument output, dále se nastavuje heslo vlastníka, které bylo přijato od klienta, naopak jako heslo uživatele se použije vygenerovaný hash, který zajišťuje dostatečnou bezpečnost a náhodnost. Jestliže si uživatel přál kopírovatelnost a tisknutelnost, je přidán argument allow a za ním dané povolení, buď jedno nebo obě.

5.4.1 Vykonávání příkazů

Java poskytuje rozhraní pro spouštění procesů a zacházení s nimi. Nejdříve je potřeba ProcessBilderu předat pole stringů, kde první je název nebo cesta ke spustitelnému souboru a ostatní jsou argumenty daného příkazu. Zavoláním metody start na právě vytvořeném objektu, získáme objekt Process, který už představuje běžící proces definovaný zadanými argumenty. Nyní je možné číst výstupy (normální i chybový) programu, ale je i umožněno psaní na vstup. Metoda *waitFor* čeká na dokončení procesu a její návratová hodnota udává, zda-li byl program ukončen chybou nebo normálním způsobem.

```
public int runCommands(List<String> commands, String logName) {
 Process process = new ProcessBuilder(commands).start();
 String logFile = directoryPath + DirUtils.LOG_DIR + logName + ".txt";
 try (
  InputStreamReader is = new InputStreamReader(process.getInputStream());
  BufferedReader reader = new BufferedReader(is);
  BufferedWriter log = new BufferedWriter(new FileWriter(logFile));
 ) {
  String line;
  while ((line = reader.readLine()) != null) {
   log.write(line + System.lineSeparator());
  }
 }
 return process.waitFor();
}
```

Ukázka kódu 5.4: Metoda pro spuštění procesu

Tato ukázka kódu představuje zmiňované zacházení s procesy používané pro kompilaci a úpravu do PDF, kde výstup programu je vypisován do textového souboru, pro budoucí použití. Pro přehlednost byly vynechány metodou vyhazované vyjímky.

5.5 Ukládání dat

Data jsou ukládána několika způsoby. Za prvé se využívá databáze, do které se ukládají požadavky klienta na výsledné PDF. Dále se používají již zmíněné JNDI zdroje, v kterých se nachází tokeny a cesty. A v neposlední řadě se všechny vygenerované PDF ukládají do klasického souborového systému.

```
/
├── app
│   └── log ... logy pro potřeby aplikace
├── archive ... zazipované archivy
├── pdf
│   ├── final ... výsledné PDF
│   ├── postcompile ... PDF před úpravami
│   └── temp ... pomocná složka pro úpravy
├── latex
│   └── aux ... ostatní výstupní soubory kompilace
└── log ... logy z kompilace a úprav
```

Obrázek 5.4: Adresářový strom

Na obrázku 5.4 můžeme vidět adresářovou strukturu s popisem k čemu dané složky slouží.

5.5.1 SQLite a MyBatis

SQLite. Tato databáze staví na klasickém SQL jazyku a stejně jako všechny ostatní nejznámější databáze je i tato relační. Ovšem na rozdíl od ostatních nepotřebuje ke svému běhu server, pouze vytváří databáze v podobě souborů na disku, ke kterým poskytuje rozhraní knihovna SQLite JDBC. Použitá verze JDBC v této práci je 3.27.2.1. Tato databáze má velmi omezený počet typů, například neobsahuje boolean, což bylo vyřešeno pomocí mapování frameworkem MyBatis[2], který je popsán o odstavec níže. Pomohl také k vyřešení problému ohledně JDBC poolu, který nešel nastavit v aplikačním serveru.

MyBatis. Je knihovna, která umožňuje mapování databáze na entity, předdefinovat si SQL dotazy a komunikovat se samotnou databází. Je použita ve verzi 3.5.0. Hlavní výhodou je velká customizace, která je potřeba při zacházení s SQLite. Jak už bylo napsáno výše, bylo vytvořeno mapování z typu boolean na typ integer a naopak. Dále dokáže spravovat spojení s databází, je možné nastavit pooling atd. Nastavení mapování entit a dotazů a samotná konfigurace se píše do XML souborů. Kde byl například nastaven již zmíněný JDBC pool, to vše je vidět v ukázce níže 5.5.

[2] http://www.mybatis.org/mybatis-3/

```
<environment id="development">
 <transactionManager type="JDBC"/>
 <dataSource type="POOLED">
  <property name="driver" value="org.sqlite.JDBC"/>
  <property name="url" value="${url}"/>
 </dataSource>
</environment>
```

Ukázka kódu 5.5: Konfigurace JDBC poolu

Samotné dotazy jsou napsány v xml souborech, ty jsou rozděleny podle tabulek, každý SELECT, INSERT atd. je označen jménem, vstupními a výstupními parametry. Jednotlivým dotazům odpovídají abstraktní metody v rozhraní entit nebo-li interface. Ty jsou vytaženy z instance SqlSession a následně je na nich zavolána daná metoda, která vykoná daný dotaz.

Kapitola 6
Testování

V této kapitole jsou navrženy jednoduché testovací scénáře, které prověřují aplikaci na různých úrovních, aby byl zajištěn bezproblémový chod služby. V dalších sekcích jsou rozebrány jedntolivé testovací postupy.

6.1 Testovací scénáře

Testovací scénáře jsou rozděleny do tří skupin, z čehož dvě se týkají integračních testů, které se dělí na automatizované 6.1 a manuální 6.2 a třetí skupinou jsou jednotkové testy 6.3. Tabulky obsahují tři sloupce, postupně z leva se jedná o název testu, dále typ neboli co je testované a v neposlední řadě očekáváný výsledek testu. Ten je nejčastěji v podobě stavového HTTP kódu a chybové hlášky v odpovědi a popřípadě stav aplikace.

Integrační pokrývají tři vrstvy: restové rozhraní, vytváření odpovědí, byznys logiku a u některých testů i přístup k databázi.

Název	**Typ**	**Očekávaný výsledek**
Nevalidní token	Rest požadavek	Kód 401
Nevalidní JSON	Rest požadavek	Kód 400
Špatný formát archivu	Rest požadavek	Kód 400
Nevalidní ID	Rest požadavek	Kód 404
Získání infa o serveru	Rest požadavek	Kód 200 a JSON obsahující atributy space, os, compiler
Příliš dlouhý název archivu	Rest požadavek	Kód 400

Tabulka 6.1: Automatizované integrační testy

Název	Typ	Očekávaný výsledek
Příliš velký archiv	Rest požadavek	Kód 400
Požadavek na job	Rest požadavek	Získané ID jobu a uložený záznam v DB
Nahrání archivu	Rest požadavek	Kód 200 a běžící proces kompilování
Získání PDF	Rest požadavek	Finální PDF v odpovědi
Špatný název archivu nebo neobsahuje LaTeX	Rest požadavek	Nejdříve 200 a na požadavek GET /pdf odpověď 400 Error during compiling or editing pdf
Získání logů	Rest požadavek	Archiv s logy

Tabulka 6.2: Manuální integrační testy

Jednotkové testy pokrývají pouze komunikaci s databází a to proto, že většina ostatních metod, u kterých by testování dávalo smysl, pracují se soubory. Tudíž je zapotřebí vstupní soubor a následně zkontrolovat výstupní, čehož testování je zbytečně časově náročné, jelikož jde to samé otestovat manuálními testy.

Název	Typ	Očekávaný výsledek
Zápis jobu do DB	Databázový dotaz	Záznam v DB
Úprava jobu v DB	Databázový dotaz	Správně změnený záznam v DB
Vrácení jobů vytvořených před datem	Databázový dotaz	Seznam jobů, jejichž datum kompilace je starší než zadané datum
Výběr jobu podle ID	Databázový dotaz	Správný záznam jobu

Tabulka 6.3: Jednotkové testy

6.2 Automatické integrační testy

K tomuto testování je použita knihovna Rest Assured, která nabízí jednoduché prostředí pro vytváření HTTP požadavků a kontrolování správnosti odpovědí. Testy byly vytvořeny pro všechny zdroje, ale ověřují pouze okrajové negativní případy, při kterých nejsou potřeba soubory.

```
@Test
void invalidToken(){
 String badToken = "asdfasdf";
 RestAssured.given().queryParam("token",badToken)
 .contentType(ContentType.JSON).body("test")
 .then().expect().statusCode(401).when().post(context);
}
```

Ukázka kódu 6.1: Požadavek se špatným tokenem

6.3 Jednotkové testy

Pro tyto testy je zapotřebí databáze, ale protože je nechtěné používat produkční a zbytečné vytvářet testovací, je proto použita H2 databáze[1]. Ta umožňuje vytvořit databázi přímo v paměti za běhu a tudíž není persistentní, což se mimojiné hodí k testovacím účelům. Jako testovací framework je použit JUnit[2] verze 5.

Celý proces probíhá tak, že je nastaveno připojení k databázi a vytvořeny objekty tříd zajišťujících komunikaci s ní. V ukázce níže 6.2 můžeme vidět metodu *createFactory*, která toto nastavení zajišťuje. Poté je vytvořen samotný objekt Job a ten je pomocí *create* v JobSession vložen do databáze. Nakonec dochází k porovnání záznamu z databáze a objektu vloženého dříve.

```
@Test
void createJob(){
 createFactory();
 Job job = getJob();
 jobSession.create(job);
 assertEquals(jobSession.getById(job.getId()),job);
}
```

Ukázka kódu 6.2: Vložení entity do databáze

6.4 Manuální integrační testy

Cílem těchto testů je zjistit, jestli je služba schopna správně vytvořit výsledné PDF na základě daných požadavků a souborů a také k otestování chybových stavů při manipulování se soubory. Všechny testy jsou vypsány v tabulce 6.2. K tomuto testování bude použit program Postman[3], který zvládá posílat všechny typy HTTP požadavků.

[1] https://www.h2database.com/html/main.html
[2] https://junit.org/junit5/
[3] https://www.getpostman.com/

6.5 Shrnutí

Byly zjištěny tyto chyby:

- U některých požadavků špatný návratový kód.
- Neošetřený požadavek s prázdným tokenem.

Dále bylo zjištěno, že některé LaTeX balíčky (např. DirtyTalk) MikTeX nepodporuje. Po opravení chyb všechny testy procházejí s kladným výsledkem. Tudíž se dá prohlásit, že aplikace reaguje správně na různé typy požadavků a je plně funkční.

Kapitola 7

Závěr

Cíle, které byly stanoveny v úvodu, byly dosaženy. Po seznámení s technologiemi a teorií relevantní pro téma této práce proběhla analýza.

V analýze byly pomocí metody FURPS+ určeny požadavky vyplývající z potřeb a omezení stanovených zadávajícím či prostředím. Dále byla zanalyzována vybraná existující řešení téhož problému, ovšem žádné z nich nesplnilo všechny požadavky. Také došlo na analýzu a porovnání vybraných kompilátorů, ze které vzešlo, že výběr je malý a rozdíly mezi nimi minimální.

Na analýzu je navázáno návrhem vlastního řešení. V této části je popsáno, jaké komponenty bude aplikace obsahovat a jak bude fungovat.

Platformou pro implementaci je Java EE s aplikačním serverem Payara. Byl navrhnut způsob komunikace a komunikační protokol mezi službou a klienty pomocí REST Api, včetně autentizace klientů, která bude prováděna na základě tokenů. Nejdůležitější částí aplikace je LaTeX kompilátor, přičemž byl zvolen MikTeX. Jeden z hlavních důvodu je možnost stahovat balíčky za běhu. Výsledné PDF soubory se uchovávají na serveru po dobu jednoho měsíce v chráněném souborovém systému.

Na základě zvoleného kompilátoru a požadavků byl vybrán operační systém Debian 9, který musí být přítomen na serveru pro správné fungování aplikace. Aby služba mohla vyhovět všem požadavkům a provozu, je požadováno 15GB volného místa na disku pro instalaci, balíčky a uchovávání výsledných PDF.

Na návrh bylo navázáno samotnou implementací, při které se nenarazilo na žádné výrazné problémy. Výsledkem je nasazená, plně funkční aplikace. Ta byla otestována jednotkovými i integračními testy a chyby jimi nalezené byly následně opraveny.

Služba je navržena a naimplementována tak, aby bylo možno navázat rozšiřováním funkcionalit, a to například i kompilováním čistého TeXu a podporou jiných výstupních formátů.

Literatura

[1] Document management - Portable document format - Part 1: PDF 1.7. https://www.adobe.com/content/dam/acom/en/devnet/pdf/PDF32000_2008.pdf, 2008. Online, Přístup 13.12.2018.

[2] The Java EE 6 Tutorial. https://docs.oracle.com/javaee/6/tutorial/doc/javaeetutorial6.pdf, 2013. Online, Přístup 5.4.2019.

[3] Bruegge, B., and Dutoit, A. H. *Object-Oriented Software Engineering Using UML, Patterns, and Java*. 2009. ISBN 0-13-606125-7.

[4] Fielding, R. T. Architectural styles and the design of network-based software architectures. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, 2000. Online, Přístup 17.11.2018.

[5] Fowler, M. *Patterns of Enterprise Application Architecture*. 2011. ISBN 0-321-12742-0.

[6] Hesam, Masne, Snehal, Harry, and Rajan. REST API Tutorial. https://restfulapi.net/. Online, Přístup 17.11.2018.

[7] Satrapa, P. Latex pro pragmatiky. http://www.nti.tul.cz/~satrapa/docs/latex/, 2011. Online, Přístup 13.12.2018.

[8] W3C. Web Service Architecture. https://www.w3.org/TR/ws-arch/wsa.pdf, 2004. Online, Přístup 4.4.2019.

Příloha A

Seznam zkratek

API – Application Programming Interface

CW – CourseWare

DB – Databáze

FURPS – Functionality, Usability, Reliability, Performance, Supportability

GB – Gigabyte

GPL – GNU General Public License

HTTP – Hypertext Transfer Protocol

JDBC – Java Database Connectivity

JNDI – Java Naming and Directory Interface

LaTeX – Lamport TeX - balík maker pro TeX

MB – Megabyte

OS – Operační systém

PDF – Portable Document Format

REST – Representional State Transfer

SQL – Structured Query Language

TeX – Typografický sázecí systém

URL – Uniforme Resource Locator

Příloha B

Zdrojový kód

- Odkaz na zdrojový kód - *https://gitlab.fel.cvut.cz/kyraltad/bachelor*