EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Evidenčné číslo:

Tvorba webovej aplikácie pomocou frameworku spring

Diplomová práca

2017 **Bc. Ivan Hlinka**

EKONOMICKÁ UNIVERZITA V BRATISLAVE
FAKULTA HOSPODÁRSKEJ INFORMATIKY

Tvorba webovej aplikácie pomocou frameworku spring

Diplomová práca

Študijný program: Informačný manažment
Študijný odbor: 3.3.24 Kvantitatívne metódy v ekonómi
Školiace pracovisko: Katedra aplikovanej informatiky
Vedúci diplomovej práce: Ing. Ján Pittner, PhD.

Bratislava 2017 **Bc. Ivan Hlinka**

3610294813

Ekonomická univerzita v Bratislave
Fakulta hospodárskej informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Ivan Hlinka
Študijný program: Informačný manažment (Jednoodborové štúdium, inžiniersky II. st., denná forma)
Študijný odbor: 3.3.24 Kvantitatívne metódy v ekonómii
Typ záverečnej práce: Inžinierska záverečná práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Tvorba webovej aplikácie pomocou frameworku Spring

Anotácia: V teoretickej časti sa diplomová práca zameriava na popis frameworku Spring. Praktická časť diplomovej práce sa zaoberá návrhom a realizáciou webovej aplikácie.

Vedúci: Ing. Ján Pittner, PhD.
Katedra: KAI FHI - Katedra aplikovanej informatiky FHI
Vedúci katedry: Ing. Mgr. Peter Schmidt, PhD.

Dátum zadania: 08.11.2015

Dátum schválenia: 10.05.2017

doc. Ing. Gabriela Kristová, CSc.
vedúci katedry

student

Vedúci

Čestné vyhlásenie

Čestne vyhlasujem, že diplomovú prácu som vypracovala samostatne a že som uviedol všetku použitú literatúru.

Dátum: **.....**

(podpis študenta)

ABSTRAKT

HLINKA, Ivan: Tvorba webovej aplikácie pomocou frameworku spring. – Ekonomická univerzita v Bratislave. Fakulta hospodárskej informatiky; Katedra aplikovanej informatiky. – Vedúci diplomovej práce: Ing. Ján Pittner, PhD. – Bratislava: FHI EU, 2017, 60s

Cieľom diplomovej práce je vytvorenie webovej aplikácie, ktorá bude študentom poskytovať spoľahlivé informácie o výučbe v danom semestri a to pomocou selektovania vopred uložených dát, ktoré budú do aplikáciou využívanej databázy vkladané osobou na to zodpovednou. Práca je rozdelená do dvoch kapitol. Obsahuje 1 tabuľku a 12 obrázkov. Prvá kapitola je venovaná popisu momentálneho stavu aplikácii používaných na univerzite. Druhá časť je venovaná cieľom tejto diplomovej práce a presnému popisu analýzy vstupných dát, návrhu aplikácie a implementácii, ku ktorej sme dospeli po naprogamovaní aplikácie.

Kľúčové slová:
spring, webová aplikácia, databáza, parser, framework, java

ABSTRACT

HLINKA, Ivan: Creation of a web application using spring framework. – The University of Economics in Bratislava. The Faculty of Business Informatics; Department of Applied Informatics. – Thesis supervisor: Ing. Ján Pittner, PhD. – Bratislava: FHI EU, 2017, 60p

The goal of thesis is to create a web application which will provide reliable data about school schedule to student in a given semester. It does that using selects on a database that was previously filled by an authorized person. Thesis is divided into two chapters. It contains one table and twelve images. First chapter describer current situation of used applications at the university. Second chapter describes goals of the thesis and describes analysis of the input data, design of the application and also its implementation that was achieved after programming the application.

Keyword:
spring, web application, database, parser, framework, java

Obsah

Úvod

Ako si všetci uvedomujeme informačné technológie sa vyvíjajú ohromujúcou rýchlosťou a to nie len po čo sa týka hardwarovej stránky, optimalizácii softwaru, ale aj čo sa týka samotného dizajnu, ktorý je zahrnutý v aplikáciách, či iných produktoch informačných technológii. Pri prvom stretnutí s momentálne používanou aplikáciou, ktorá študentom umožňuje prezerať si ich rozvrh som bol prirodzene ako väčšina študentov v rozpakoch ako sa v tejto aplikácii vôbec orientovať. Nehovorím o samotnom, už zobrazenom rozvrhu, ale o fakte, že zadávanie údajov, ktoré sú potrebné pre zobrazenie rozvrhu je prinajmenšom chaotické a a tomu nepomáha ani fakt, že sa tu objavuje problém s diakritikou. Avšak je dôležité vyzdvihnúť jeden fakt – táto aplikácia nie je ani zďaleka taká hrôzostrašná ako sa na prvý pohľad zdá. Po pár sekundách si človek uvedomí kde zadávať údaje a ako to celé vlastne funguje. Táto aplikácia však obsahuje pár obmedzení, ktoré sú v dnešnej dobe prinajmenšom nepohodlné, avšak v niektorých prípadoch sa dajú označiť až za mimoriadne problematické.

Ďalším dôvodom prečo som sa rozhodol pre tvorbu aplikácie je, že mojím cieľom je stať sa plnohodnotným programátorom a vzhľadom na rastúcu ponuku na trhu je pre každého jednotlivca jednoduchšie sa presadiť pokiaľ je jeho škála ovládaných programovacích, alebo iných, jazykov väčšia než je priemer na trhu. Vzhľadom na fakt, že počas bakalárskeho a ani inžinierkeho štúdia som sa na fakulte hospodárskej informatiky nestretol s predmetom, kde by sa programovací jazyk Java vyučoval som sa rozhodol pre samoštúdium a keďže sa s týmto jazykom stretávam v praxi už od tretieho ročníka kedy som začal písať bakalársku prácu a zamestnal sa vo firme, kde tento jazyk využívam takmer na každonnej úrovni povedal som si, že je najvyšší čas prispieť mojou troškou ku skvalitneniu života študentov Ekonomickej univerzity v Bratislave. Od môjho prvého pokusu o takéto skvalitnenie prebehlo pár rokov a skúsenosti, ktoré som v praxi nabral budú prirodzene len a len plusom ako pre mňa, tak aj pre študentov, ktorí aplikáciu budú používať, pevne dúfam ešte mnoho rokov po tom, čo ja už na škole nebudem pôsobiť. Je pravdou, že nová verzia rozvrhu je rozhodne viac užívateľsky príjemná než pôvodný Look In, avšak mojím názorom je, že vždy existuje niečo čo sa dá zlepšovať a vzhľadom na fakt, že Eubarozvrh bol naprogramovaný ako študentský projekt, pri testovaní aplikácie ako aj pri zhliadnutí samotného zdrojového kódu som dospel k rozhodnutiu, že je čas vytvoriť kód, ktorý bude jednoduchý na pochopenie a modifikáciu v prípade akýchkoľvek zmien,

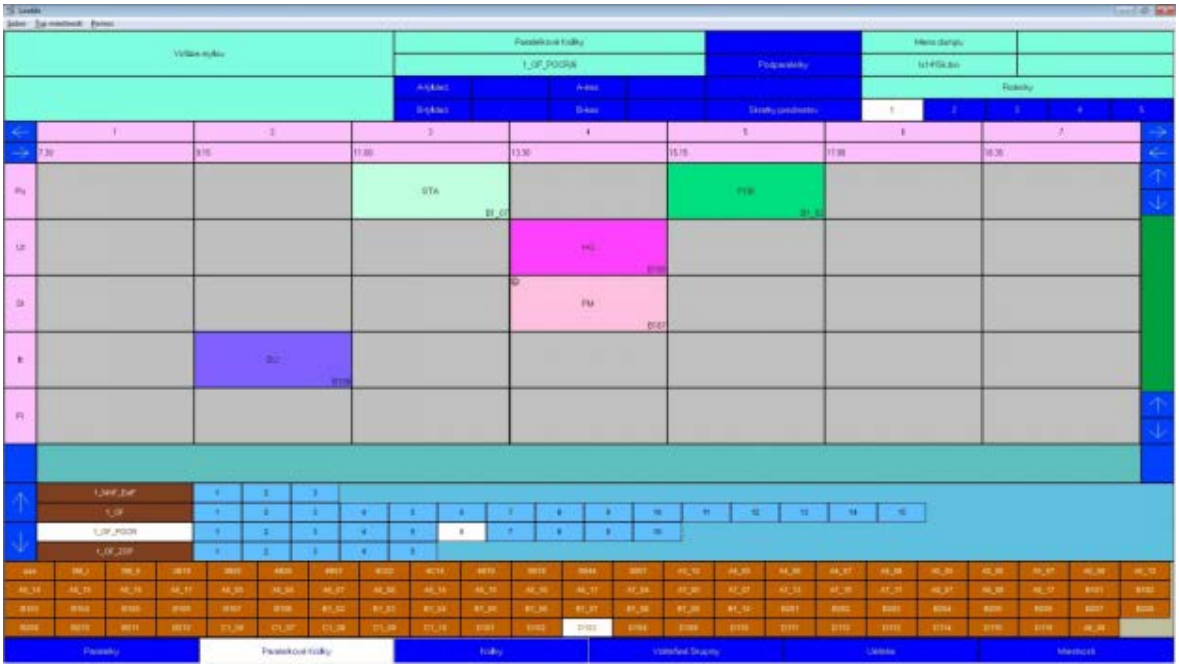
ktoré môžu v budúcnosti nastať. Taktiež by mal optimalizovať zobrazenie predmetov, keďže momentálne používaná aplikácia v niektorých prípadoch zobrazuje predmety, ktoré pre študenta nie sú nahlásené. Vo väčšine ide o voliteľné, alebo povinne voliteľné predmety, kde je mierny problém so špecifikáciou pre ktorého študenta tieto predmety sú a pre ktorého už nie.

1. Súčasný stav riešenej problematiky doma a zahraničí

V súčasnej dobe je možné zobrazovať rozvrh pomocou dvoch rôznych aplikácii, kde prvá z nich menom Look In je pôvodná aplikácia v desktopovej podobe funkčná výlučne na operačnom systéme Windows a druhá je relatívne nová verzia webovej aplikácie s pomenovaním Eubarozvrh. Poďme sa bližšie pozrieť na tieto dve aplikácie.

1.1.Look In

V súčasnej dobe sa na Ekonomickej univerzite v Bratislave používala aplikácia Look In, ktorú vytvoril J.Tvarožek, vo verzii 5.0. Táto aplikácia vyžaduje umiestnenie v špecifickej adresárovej štruktúre a funguje výlučne pre systém Windows. Tento fakt prirodzene vytvára komplikácie pre užívateľov, ktorí touto informáciou nedisponujú a snažia sa aplikáciu spustiť. Po spustení sa zobrazí okno, ktoré vyžaduje nájdeniu dump súboru, ktorý slúži ako zdroj údajov pre samotnú aplikáciu. Následne sa otvorí hlavné okno, v ktorom sú zahrnuté všetky funkčné prvky tejto aplikácie. Obrázok 1 vykresľuje okno, ktoré zobrazí rozvrh pre štandardný ročník, fakultu, odbor a krúžok. Následne študent na rôznych miestach v aplikácii tieto štandardne zadané údaje mení podľa toho, čo vyžaduje.



Obrázok 1 Podoba programu LookIn (zdroj : aplikácia Look In)

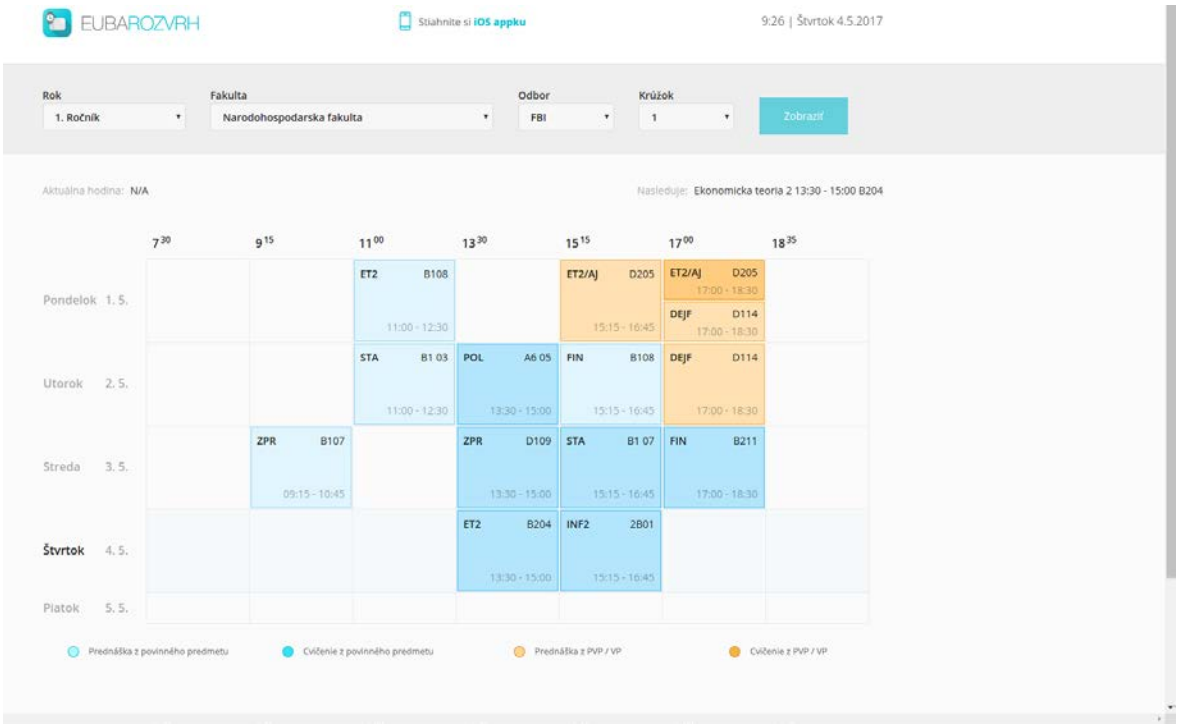
Táto aplikácia má širokú škálu možných zobrazení rozvrhu podľa toho, čo zadávateľ vyžaduje. Medzi tieto zobrazenia patrí vyhľadávanie rozvrhu pre učiteľov, rozvrh pre konkrétnu miestnosť a iné vyhľadávania, na ktoré sa sa autor zameral.

Z pohľadu užívateľa je zadávanie ročníka a ostatných potrebných údajov chaotické a nepraktické vzhľadom na to, že sú na rôznych miestach a je takmer vždy potrebné používať šípky, ktoré slúžia na zobrazovanie rôznych odborov daných fakúlt.

Táto aplikácia pozostáva z 2 častí, ktoré štandardne označujeme ako front end a back end. Front end je užívateľské rozhranie, ktorý dané dáta zobrazuje do podoby vhodnej pre užívateľa, zatiaľ čo back end zabezpečuje parsovanie súboru, ktorý obsahuje dáta o predmetoch, učiteľoch, miestnostiach a ďalšie informácie.

1.2.Eubarozvrh

Modernejšou verziou, ktorá je od nedávnej doby dostupná pre užívateľov a teda predovšetkým študentov je Eubarozvrh. Práve na tejto aplikácii bude táto diplomová práca stavať a zdokonaľovať ju ako po technickej, tak aj po užívateľskej stránke. Eubarozvrh je možné spustiť po prechode na príslušnú URL adresu a síce ttp://euba-rozvrh.herokuapp.com. Po zadaní údajov, konkrétne ročník, fakulta, odbor a krúžok sa zobrazí rozvrh podľa zadaných vstupných parametrov a teda používateľská skúsenosť je ďaleko príjemnejšia v porovnaní s Look In aplikáciu, avšak je potrebné mať prístup na internet vzhľadom na fakt, že sa jedná o webovú aplikáciu. Vzhľadom na komplexnosť subjektu akým je Ekonomická univerzita v Bratislave však Eubarozvrh nedokonale zobrazuje rozvrh a teda je možné, že sa používateľovi naskytne situácia kedy jemu zobrazený rozvrh obsahuje dáta, ktoré nie sú pre neho aktuálne. Príkladom je zobrazenie povinne voliteľných predmetov pre celý ročník aj v prípade, že študent nemá nahlásený žiadny povinne voliteľný predmet v danom semestri. Dôvodom takejto situácie študenta môže byť fakt, že si nahlásil viac povinne voliteľných predmetov v predchádzajúcom semestri, prípadne ich má viac nahlásených v ďalšom semestri jeho štúdia. Podobne ako Look In aj Eubarozvrh disponuje front endom a back endom, takže v tomto prípade už sú zdrojové kódy oddelené, čo zvyšuje prehľadnosť kódu a oddeľuje samostatné funkčné časti. Grafické rozhranie hlavného okna tejto aplikácie môžeme vidieť na obrázku 2.



Obrázok 2 Hlavné okno aplikácie Eubarozvrh (zdroj: aplikácia Eubarozvrh)

2. Cieľ práce, metodika práce a metódy skúmania

V nasledujúcej kapitole sa budeme venovať základným pojmom, s ktorými budeme pracovať v tejto diplomovej práci a predstavíme ciele práce, ktoré si stanovíme.

2.1. Základné pojmy

Aplikácia

V informačných technológiách je aplikácia použitím technológie, systému, alebo produktu. Pojem aplikácia je kratší ako jeho dlhá forma aplikačný program. Aplikačný program, alebo pogram navrhovaný na vykonávanie špecifickej funkcionality priamo pre užívateľa, alebo v niektorých prípadoch pre ďalší aplikačný program. Príkladom aplikácií sú slovné procesory, databázové programy, webové prehliadače, vývojové nástroje, nástroje na editáciu obrázkov a podobne. Aplikácie používajú servisy počítačového opreračného systému a iné podporné aplikácie. Formálne požiadavky a spôsoby komunikácie s inými programami, ktoré aplikácia využíva sa nazýva aplikačné programové rozhranie (API).[1]

Webová aplikácia

Webová aplikácia, skrátene web app je aplikačný program, ktorý je uchovávaný na vzdialenom serveri a posielaný cez internet pomocou rozhrania prehliadača. [2]

Parser

Parser je druh programu, ktorý je obyčajne súčasťou kompilera, ktorý dostane vstup vo forme vstupných dát a vstupných inštrukcií, rozdelí tieto dáta pomocou daných inštrukcií do menších častí(napríklad na slovesá, podstatné mená, prípadne podľa istého

[1]Margaret Rouse, Application [online]. Dostupné na internete: http://searchsoftwarequality.techtarget.com/definition/application (voľný preklad)

[2]Margaret Rouse, Web application [online]. Dostupné na internete: http://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app (voľný preklad)

znaku, resp. viacerých znakov). Avšak v tejto diplomovej práci budeme pojmom označovať časť funkcionality aplikácie, ktorá pretransformuje vstupné dáta do podoby vhodnej pre užívateľa aplikácie.[3]

Grafické rozhranie

„Grafické rozhranie (graphical user interface – GUI) je jednou zo základných častí každého súčasného operačného systému, je to vlastne to, čo používateľ „vidí“, keď zapne počítač a začne robiť za počítačom – pracovná plocha (desktop), ikony, okná…

Grafické rozhranie ako ho dnes poznáme, používajúce tzv. WIMP systém (window, icon, mouse, pointer – okno, ikona, myš, kurzor), vzniklo v 70. rokoch 20. storočia a jeho autorom bola firma Xerox. Takéto GUI prevzala spoločnosť Apple, ktorá ho použila pre Mac OS (ktorý bol zabudovaný do počítačov Lisa a Macintosh), a potom aj Microsoft (operačný systém Windows). Koncom 80. a začiatkom 90. rokov sa už GUI stalo štandardným prvkom každého operačného systému.“[4]

Objektovo-orientované programovanie

„Objektovo orientované programovanie je spôsob programovania, ktorý nás núti deliť programy na malé časti nazývané objekty. Nedá sa to celé vysvetliť v jednom článku, ale pokúsim sa na niekoľkých príkladoch ukázať pointu. (Zvyšok nájdete v učebniciach a na internete.) Aby boli príklady čo najjednoduchšie, predstavme si program, ktorý pracuje s jednoduchými geometrickými útvarmi, napríklad štvorcami. Každý štvorec má nejakú veľkosť, podľa ktorej môžeme vypočítať jeho obvod a plochu. Program si vytvorí nejaké štvorce a potom spočíta ich celkovú plochu.“[5]

---

[3]Margaret Rouse, Parser [online]. Dostupné na internete: http://searchmicroservices.techtarget.com/definition/parser (voľný preklad)

[4]Filip Filip, Grafické rozhranie (GUI) [online]. Dostupné na internete: http://bit.ly/2pykl1b

[5]Viliam Búr, Objektovo orientované programovanie [online]. Dostupné na internete: http://bur.sk/sk/2012/java-oop

Framework

Frameworky sú knižnice, ktoré majú uľahčiť prácu pri programovaní aplikácie. To znamená menej písania, prehľadnejší kód a rýchlejší vývoj. V prípade webových frameworkov sa často spomína MV, čo je architektúra rozdeľujúca aplikácie na nezávislé vrstvy.[6]

Spring

Vzhľadom na fakt, že nami vyvíjaná aplikácia využíva Spring je nutné tento prvok predstaviť a bližšie uviesť funkcionalitu a prvky, ktorými tento framework disponuje. Framework Spring je na Java platforme postavený komplexný nástroj s obsiahlou infraštruktúrou podpory na vývoj java aplikácií. Tento framework zabezpečuje infraštruktúru tak, aby sa vývojár mohol sústrediť na aplikáciu samotnú a stráviť menej času pri vytváraní dizajnových a funkcionálnych prvkov. Spring umožňuje postavenie aplikácie na POJO objektoch(plain old java object) a aplikovanie podnikových servisov na POJO objekty a to neinvazívnym spôsobom. Táto schopnosť sa vzťahuje na Java SE programovací model a na úplnú a čiastočnú Java EE.[7]

2.2. Cieľ práce

Cieľom projektu je vytvoriť webovú aplikáciu, ktorá bude na slúžiť na dekódovanie, ukladanie a poskytovanie dát určených pre študentov Ekonomickej univerzity v Bratislave. Tieto dáta predstavujú údaje o rozvrhu jednotlivých skupín do ktorých sú študenti zaradení, ktoré nazývame výučbové krúžky. A teda cieľom práce je vytvoriť aplikáciu prostredníctvom aplikačného frameworku s podporou inverzie kontroly zvaného Spring.

Pod poskytovaním rozumieme umožňovanie predom uložených dát v podobe vopred dohodnutej medzi žiadateľom dát a poskytovateľom. V našom prípade to bude formát JSON(JavaScript Object Notation), ktorý žiadateľ týchto dát spracuje a požadovateľovi rozvrhu zobrazí v užívateľsky prívetivej podobe.

---

[6] Peter Láng, Co je to framework [online]. Dostupné na internete: http://langi.cz/webarna/co-je-to-framework (voľný preklad)

[7] Spring Framework Reference Documentation [online] . Dostupné na internete: https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#overview (voľný preklad)

Jednou z funkcií je taktiež umožnenie osoby na to oprávnenej meniť obsah uložených dát prostredníctvom nahratia aktualizovaného súboru, ktorý aplikácia spracuje a uloží do databázy. Táto osoba je identifikovaná klasickou autentifikáciu v podobe užívateľského mena a hesla.

2.3. Analýza vstupných dát

Základom celej aplikácie je spracovať vstupné dáta poskytnuté v textovom dokumente, ktorý má štandardne príponu .par a uložiť tieto dáta do databázovej schémy na to prispôsobenej. Po vykonaní vyššie zmieneného slúži aplikácia ako poskytovateľ informácií, ktoré sú uložené v databáze pre žiadateľov. Pod žiadateľmi si môžeme predstaviť študentov, ktorí využívajú už existujúci front end v aplikácii Eubarozvrh. Avšak využitie front endu aplikácie Eubarozvrh nie je podmienkou a dostatočne zdatní žiadateľ dokáže požadovať dáta aj inou formou a to či už použitím príslušného URL odkazu, alebo inou aplikáciou, ktorú má k dispozícii.

Vstupné dáta sú zadávané v štandardizovanej forme, avšak vzhľadom na fakt, že tieto údaje sú ručne písané jedným človek a na rozsah týchto dát je potrebné brať do úvahy možnú chybu, ktorá sa môže vyskytovať v týchto dátach. Dáta obsiahnuté v jednom súbore sú špecifické pre jeden semester výučby.

Spomínaná forma je nasledovná. Dáta sú rozdelené do sekcií, pričom každá sekcia začína bodkou a nasleduje popis sekcie, ktorý je napísaný veľkými písmenami. V nasledujúcom riadku, prípadne viacerých riadkov môžu nasledovať dáta špecifické pre danú sekciu, alebo tu môže byť definovaná ďalšia sekcia a teda momentálna sekcia je prázdna, prípadne sa tu môže nachádzať prázdny riadok, alebo koniec dát.

Dáta v každej sekcii majú iný charakter a preto postupne analyzujeme sekcie, ktoré sú potrebné pre zobrazenie rozvrhu.

Sekcia Hodiny

Dáta v tejto sekcií označujú počet hodín, ktoré je možné odučiť počas jedného výučbového dňa a v momentálnej verzii pre letný semester akademického roka 2016/2017 sú nasledovné:

1 7.30 - 9.00

2 9.15 - 10.45
3 11.00 - 12.30
4 13.30 - 15.00
5 15.15 - 16.45
6 17.00 - 18.30
7 18.35 - 20.05

V prvom stĺpci je teda definované poradové číslo hodiny nasledované medzerou. Po tejto medzere nasleduje hodina a minúta, ktoré sú oddelené bodkou. Po čase začiatku hodiny nasleduje medzera, pomlčka a ďalšia medzera, čo v bežnom ľudskom ponímaní reprezentuje interval. Posledným údajom v jednom riadku je hodina a minúta oddelená bodkou, ktoré reprezentujú čas, kedy daná hodina končí. Ako je možné z dát vidieť, letný semester pozostával maximálne zo siedmich vyučovacích hodín počas jedného dňa a každej hodine prislúcha čas začiatku a čas konca.

Sekcia Katedry

Táto sekcia obsahuje údaje o katedrách, ktoré sú prítomné na škole v danom semestri a tieto údaje sú v nasledujúcej forme:

Neznama_katedra KAAA 728 0
Katedra_ekonomickej_teorie KET 675 9
Katedra_hospodarskej_politiky KHP 662 9

Po analýze dát v tejto sekcii môžeme dospieť k záveru, že prvým údajom je názov katedry, ktorý je v tvare začínajúcom veľkým písmenom nasledovaným ľubovoľným počtom malých písmen a v prípade, že katedra má viacslovný názov medzery v tomto názve sú reprezentované znakom podrthávania slova nazývaním podčiarkovník. Názov katedry je nasledovaný medzerou, po ktorej môžeme vidieť skratku katedry, ktorá môže obsahovať veľké a malé písmená. Po skončení skratky katedry nasleduje medzera a posledný pre nás zaujímavý údaj sa nachádza po tejto medzere. Toto číslo reprezentuje kód danej katedry.

Sekcia Miestnosti

V tejto sekcii môžeme nájsť, ako z názvy vyplýva údaje o miestnostiach, ktorými Ekonomická univerzita disponuje a sú určené na výučbu. Vzhľadom na množstvo týchto

miestností uvediem iba vybrané miestnosti, ktoré budú reprezentovať skupinu miestností im podobným.

U A9_17 20
U B101 24
L LAB5 20
S D206 26
I A9_06 15

Prvý údaj vo formáte veľkého písmena označuje kód typu miestnosti. Nasleduje medzera a po nej nájdeme údaj o názve miestnosti, ktorý my v ľudskom ponímaní chápeme ako označenie miestnosti. Ako možeme vidieť miestnosť, ktorá sa v škole nachádza pod označením A9.17 je tu reprezentovaná ako A9_17 a teda tento fakt by sme mali zohľadniť a pre minimalizáciu možnosti vzniku pomýlenia vymením znak podčiarkovníka za znak bodky. Po označení miestnosti nasleduje medzera a následne kapacita danej miestnosti.

Sekcia Učitelia

Táto sekcia obsahuje údaje o menách učiteľov, ich najvyššom dosiahnutom titule, ale aj ich príslušnosti ku katedre, pod ktorou pôsobia. Poďme sa pozrieť na štruktúru týchto dát. Obdobne ako pri sekcii miestností aj tu uvediem kvôli rozsahu dát iba vybrané údaje

Adamcova_Livia KJaT Prof : U
Adamcova_Silvia KJaT Mgr : U
Ahlrep_ KJaT Mgr : U
Alexy_Martin KF Ing : U
Antalova_Maria KSRP Ing : U
Antalova_Renata KUA Ing : U
Arendas_ KBMF PhD : U
Bacisin_Vladimir KET Mgr : U
Baculakova_ KMVHD PhD : U
Badura_Peter KPF PhD : U
Balaz_Peter KMO Doc : U

Ako zo štruktúry vyplýva je možné, že meno učiteľa obsahuje krstné meno a priezvisko, prípadne iba priezvisko učiteľa. V prvom prípade sú krstné meno a priezvisko oddelené podčiarkovníkom, ktorý v dôsledku štruktúry bežne využívaného jazyka budeme reprezentovať ako medzeru. V druhom prípade nájdeme iba priezvisko, ktoré je nasledované počiarkovníkom. Po údaji o mene učiteľa nasleduje medzera, po ktorej nájdeme údaj o katedre, pod ktorú daný učiteľ spadá. Nasleduje opäť medzera, po ktorej môžeme čítať údaj o titule, ktorý daný učiteľ dosiahol. Nie je však zjavné, či sa jedná o najvyššie dosiahnutý titul, alebo nie a preto ostaneme pri označení titul. Ďalšie údaje nie sú pre nás dôležité a preto ich preskočíme.

Sekcia Paralelky

Tu môžeme nájsť údaje o príslušností daných študijných odborov pod katedry a taktiež údaj o tom, pre ktorý ročník je daný odbor k dispozícii. Opäť uvediem len údaje pre 1 ročník, na ktorých demonštrujem akým spôsobom budeme pristupovať ku čítaní dát a ich transformácii na informácie pre užívateľa.

```
1_rocnik 1
 $ Narodohospodarska_fakulta NHF
   $$ Financie_bankovnictvo_a_investovanie FBI 212/1-9
   $$ Narodne_hospodarstvo NH 27/1
   $$ Ludske_zdroje_a_socialny_manazment LZSM 14/1
   $$ Ekonomia_a_pravo EaP 69/1-3
 $ Obchodna_fakulta OF
   $$ Podnikanie_v_obchode POB 70/1-3
   $$ Podnik_v_cestov_ruchu_a_sluz PCRS 88/1-4
   $$ Medzinarodne_podnikanie MP 64/1-3
2_rocnik 2
```

V prvom riadku vidíme označenie ročníka, vo formáte číslo nasledované podčiarkovníkom a slovom “rocnik”, po ktorom nasleduje medzera a poradové číslo ročníka. V ďalšom riadku nájdeme medzeru a symbol doláru, po ktorom opäť nasleduje medzera a názov fakulty. Tento názov začína veľkým písmenom nasledovaným ľubovoľným množstvom malých písmen. V prípade, že názov fakulty pozostáva z

viacerých slov, čo je bežný prípad, sú tieto slová oddelené podčiarkovníkom a jednotlivé slová pozostávajú z malých písmen. Nasleduje medzera a skratka fakulty, ktorá je reprezentovaná veľkými písmenami. Ďalší riadok začína viacerými medzerami, ktoré sú nasledované dvomi znakmi doláru a opäť medzerou. Po nich nájdeme názov odboru, ktorý pozostáva z kombinácie malých a veľkých písmen, pričom viacslovné názvy sú oddelené podčiarkovníkom. Pravidlom je, že aj tento názov začína veľkým písmenom. Po názve nasleduje medzera a skratka pre daný odbor, ktorá pozostáva z veľkých a malých písmen v ľubovoľnom množstve. Táto skratka je nasledovaná medzerou a údajom o počte študentov v danom študijnom odbore, ktorý je od ďalších údajov oddelený lomkou. Po lomke nasleduje údaj o čísle, ktoré reprezentuje prvý krúžok a v prípade, že odbor má viac študijných krúžkov je tento fakt reprezentovaný číslom prvého krúžku, pomlčkou a číslom posledného krúžku v danom odbore. Znaky doláru sú pre nás dôležité na porozumeniu, že daný odbor je v už definovovanej fakulte, alebo o tom, že práve definujeme ďalšiu fakultu, pre ktorú budú nasledovať študijné odbory, ktoré začínajú dvomi znakmi doláru. V prípade, že sme prečítali všetky riadky, ktoré prislúchajú k danému ročníku dostanem informáciu o fakte, že je potrebné začať s čítaním ďalšieho ročníka, ktorý je opäť definovaný rovnakou štruktúrou ako ten, ktorý sme práve prečítali.

Sekcia Predmety

Tu sa dostávame k hlavnej sekcii, ktorá obsahuje približne 4000 riadkov a teda je prirodzené, že aj napriek štandardizovanej forme, v ktorej sú údaje zadávané môžeme analyzovať viacero štandardov, ktoré sú v tejto sekcii používané.

```
3_FHI_UC 35290: KUA: P V | Bankove_operacie BO Ondrusova_Lucia 1/1 132 ;335
~ 727 1
$1 B1_05 1 4 0 Ondrusova_Lucia
&1 A7_15 1 5 1 0 Ondrusova_Lucia
&1 A6_17 1 2 2 0 Ondrusova_Lucia
&1 A7_05 2 4 3 0 Ondrusova_Lucia
&1 A7_15 1 6 4 0 Ondrusova_Lucia
&1 A7_15 2 5 5 0 Ondrusova_Lucia
&1 A7_05 2 3 6 0 Ondrusova_Lucia
```

Vyššie uvedený je útržok údajov zo sekcie predmety, na ktorom demonštrujeme najbežnejšie sa vyskytujúcu podobu údajov. Na prvom riadku môžeme vidieť číselné označenie ročníka, pre ktorý je daný predmet vyučovaný. Nasleduje znak podčiarkovníka, skratka fakulty pozostávajúca z veľkých písmen, opäť podčiarkovníkom a označenie skratka odboru, ktorá je reprezentovaná kombináciu veľkých a malých písmen spravidla začínajúcich veľkým písmenom. Nasleduje medzera a kód predmetu, ktorý na prvý pohľad vyzerá ako číselné označenie. Ďalší pre nás zaujímavý údaj nasleduje po znaku dvojbodky a medzery. Týmto údajom je skratka katedry, ktorá je kombináciou veľkých a malých písmen v ľubovoľnom množstve. Opäť nachádzame znak dvojbodka nasledovaný medzerou po ktorej sa pokúsime čítať všetky znaky až po dosiahnutí znaku predelovníka. Čítané znaky reprezentujú formu v akej sa daný predmet môže vyučovať a teda v našom prípade znak “P” označuje, že predmet môže byť povinný a znak “V”, že predmet môže byť voliteľným predmetom. Po už spomínanom znaku predelovník nasleduje medzera a názov predmetu, ktorý v prípade, že sa jedná o viacslovný názov, ako v našom prípade, je predelovaný podčiarkovníkom. Tento názov pozostáva z veľkých a malých písmen, pričom vždy začína veľkým písmenom. Nasleduje medzera a skratka predmetu, ktorá vyzerá byť slovom pozostávajúcim z veľkých písmen. Opäť nájdeme znak medzery, po ktorej môžeme prečítať meno vyučujúceho, ktorý daný predmet prednáša. V našom prípade sa stretávam s dvojslovným menom, pričom aj krstné meno, aj priezvisko začína veľkým písmenom a je oddelené podčiarkovníkom. Po prečítaní mena čítame znak medzery a nasleduje číslo, ktoré údáva údaj o pošte vyučovaných prednášok. Následne znak lomky a číslo označujúce počet cvičení daného predmetu. Opäť medzera, po ktorej nájdeme pre nás posledný dôležitý údaj v tomto riadku a to počet študentov, ktorý majú tento predmet absolvovať, alebo sa o to aspoň pokúsia. Ďalší riadok, ktorý začína symbolom vlnovky je pre nás nepodstatný a preto ho preskočíme.

Tu nájdeme znak doláru nasledovaný číslom jeden, čo špecifikuje, že ideme čítať údaje o prednáške. Nasleduje medzera a označenie miestnosti, v ktorej bude prednáška počas semestra prebiehať. Opäť sa stretávame s formátom miestnosti, ktorý sme už vyššie popisovali v sekcii miestnosti. Po údaji o miestnosti nasleduje, na prvý pohľad, náhodný počet čísel oddelených medzerou, avšak každé z nich špecifikuje pre nás potrebný údaj. Pusťme sa teda do dekódovania týchto čísel. Prvé z čísel špecifikuje poradové číslo dňa, kde Pondelok je prvý, Utorok druhý a tak ďalej. Ďalšie číslo je poradové číslo výučbovej hodiny, počas ktorej sa daná prednáška vyučuje. Tieto poradové čísla sme už bližšie rozoberali v sekcii hodiny. Nasleduje nula, ktorá nám potvrdí, že čítame riadok, ktorý

obsahuje dáta o prednáške. Po týchto číslach nasleduje meno prednášajúceho danej prednášky, ktorý je v tvare, ktorý sme už viackrát popisovali.

Ďalšie riadky obsahujú údaje o tom, kedy prebiehajú cvičenia pre daný predmet. Poďme sa na ne bližšie pozrieť. Prvým symbolom je anglický znak a známy ako uppersand nasledovaný jednotkou a medzerou. Opäť sa stretávame s označením miestnosti, v ktorej bude cvičenie prebiehať, po ktorom nasleduje medzera. Tu môžeme nájsť poradové číslo dňa, poradové číslo hodiny, číslo krúžku, pre ktorý je dané cvičenie a nula, ktorá zjavne reprezentuje koniec číselných údajov. Posledným údajom je meno učiteľa, ktorý daný predmet cvičí.

Výnimky

Vzhľadom na rozsah údajov a rozmanitosť predmetov sú vyššie demonštrované analýzy v niektorých prípadoch nepresné a dospeli by sme do bodu, kedy nám podľa tejto analýzy napísaný kód vyhodí chybu. Z tohto dôvodu pri písaní zdrojového kódu aplikácie budeme testovať vstupné dáta a spôsobom pokus omyl budeme zistené rozdiely implementovať do už existujúceho riešenia a tak spätne upravovať analýzu. Pre bližšie upresnenie takejto situácie uvedieme príklad.

```
2_FHI 11092/2: KET: V 1 V | Prenos_Ekonomicka_teor_2 P_ET2 AAA_ucitel 0/1 25 ;206
~ 675 9
&1 B1_09 2 6 1 0
```

Z vyššie uvedenej časti súboru reprezentujúceho vstupné dáta môžeme vidieť, že v porovnaní s analýzou vykonanou na jednom predmete sa tu vyskytujú rozdiely. Prvým je fakt, že nemáme uvedený odbor pre ktorý je tento predmet vyučovaný, ďalším je fakt, že namiesto mena učiteľa, ktorý predmet vyučuje je uvedené generické meno “AAA_ucitel” a teda v rámci zlepšenia užívateľskej skúsenosti meno vyučujúceho nebudeme uvádzať. Takisto môžeme pozorovať situáciu, kde nie je uvedené ani meno prednášajúceho. Nesmieme ani zabudnúť na fakt, že kód predmetu obsahuje znak lomky, ktorý sme vo vzorovom príklade predmetu nepozorovali.

Ďalším prípadom kedy by analýza len jedného predmetu nestačila je prípad, kedy je skratka predmetu v nami predtým nepozorovanom formáte, kde je použité číslo a nie len

kombinácia malých a veľkých písmen ako vo väčšine prípadov. Príklad takéhoto predmetu je uvedený nižšie.

```
2_FAJ 81945: KJaT: V 0 V | Treti_CJ_FJ 3FJ Rizekova_Iveta 1/0 13 ;241
~ 50 0
$1 E2_06 2 6 0 Rizekova_Iveta
```

Takýchto prípadov je viac než dosť a vzhľadom na počet riadkov, ktoré je potrebné analyzovať nám nami napísané testy pomôžu takéto prípady nájsť a zakomponovať do implementácie aplikácie. Pokiaľ je mne známe žiadne predchádzajúce riešenie, ktoré pracovalo s parsovaním vstupných dát a zobrazovaním týchto dát koncovému užívateľovi neobsahovalo testovanie a teda je dosť vysoká pravdepodobnosť, že tieto riešenia obsahujú chyby práve v takýchto extrémnych prípadoch. Tomu nasvedčuje aj fakt, že rozvrh výučby sa každý semester mení a môže nastať prípad kedy vopred dohodnuté štandardy vstupných dát nebudú postačovať a bude potrebné vytvoriť niečo nové, čo by mohlo v predchádzajúcich riešeniach parsovania spôsobiť problémy a v krajnom prípade aj pád aplikácie. To by malo za následok fakt, že sa študenti nedostanú k rozvrhu a teda chaos, ktorému sa chceme vyhnúť. Preto nesmieme ani my zabúdať na logovanie problémov v aplikácii, ktoré v prípade nastatia takejto situácie včasne a hlavne rýchlo napovie z akého dôvodu táto situácia nastala a ako kód aktualizovať tak, aby bral do úvahy nový štandard, ktorý bol použitý vo vstupných dátach.

2.4.Návrh

Vzhľadom na dokončenú analýzu vstupných dát sa môžeme pustiť do časti zaoberajúcej sa návrhom samotnej webovej aplikácie. Musíme tu brať do úvahy fakty a chyby, ktoré existujú v predchádzajúcich aplikáciách, ktoré sa zaoberali parsovaním, ukladaním a poskytovaním dát. Taktiež budeme brať do úvahy existujúce dizajnové vzory a štandardizované praktiky pri OOP. Pristúpime teda k samotnému návrhu databázovej schému aplikácie vzhľadom na fakt, že disponujeme informáciou o vstupných dátach a o tom, čo požaduje koncový používateľ. Vzhľadom na fakt, že nevieme kto bude v budúcnosti aktualizovať zdrojové kódy a akým jazykom bude táto osoba hovoriť, respektíve aké jazyky bude ovládať bude ako databázový model, tak aj samotný kód a jeho

časti v anglickom jazyku, ktorý poskytuje najväčšiu pravdepodobnosť, že táto osoba bude kódu a modelu rozumieť.

1.1.1. Databázový model

Predchádzajúca analýza ukázala, že aplikácia bude vyžadovať minimálne nasledujúce tabuľky: fakulta, katedra, odbor, miestnosť, vyučovacia hodina, predmet, výučba predmetu a učiteľ. V rámci dodržania jazykovej bariéry a štandardov databázového enginu, ktorým bude MySQL budú tabuľky v databáze skutočne nazvané nasledovne. Tabuľka fakulta ponesie meno faculty, tabuľka katedra sa bude nazývať department, odbor pomenujeme field_of_study, vyučovacia hodina bude v databáze niesť meno school_hour. Tabuľka predmet sa bude volať subject, výučba predmetu ponesie databázové pomenovanie subject_happening a tabuľku učiteľ nazveme teacher. Obdobné pomenovania použijeme aj v zdrojovom kóde písaného v jazyku Java.

Teraz si bližšie popíšeme atribúty jednotlivých tabuliek a ich typ, prípadne referenciu, ktorú predstavujú v rámci databázového modelu a taktiež vlastnosti, ktoré daným atribútom prislúchajú.

Faculty

id - štandardné identifikačné číslo typu bigint v maximálnom rozsahu 20 symbolov, unikátne, primárny kľúč, ne smie byť null

name - meno fakulty typu varchar v maximálnom rozsahu 255 symbolov, nesmie byť null

short_name - skratka predmetu typu varchar 255 symbolov, nesmie byť null

Department

id - štandardné identifikačné číslo typu bigint v maximálnom rozsahu 20 symbolov, unikátne, primárny kľúč, nesmie byť null

faculty_id - id fakulty, pod ktorú daná katedra spadá typu bigint v maximálnom rozsahu 20 symbolov, cudzí kľúč, smie byť null

code - kód katedry typu varchar v maximálnom rozsahu 255 symbolov, nesmie byť null

name - meno katedry typu varchar v maximálnom rozsahu 255 symbolov, nesmie byť null

short_name - skratka katedry v maximálnom rozsahu 255 symbolov, môže byť null

Field_of_study

id - štandardné identifikačné číslo typu bigint v maximálnom rozsahu 20 symbolov, unikátne, primárny kľúč, ne smie byť null

faculty_id - id fakulty, pod ktorú daný študijný odbor spadá typu bigint v maximálnom rozsahu 20 symbolov, cudzí kľúč, smie byť null

name - meno študijného odboru typu varchar v maximálnom rozsahu 255 symbolov, ne smie byť null

short_name - skratka študijného odboru typu varchar 255 symbolov, môže byť null

number_of_students - počet študentov, ktorí sú na predmet prihlásení typu int, môže byť null

number_of_first_parallel_class - cislo prveho krúžku typu int, môže byť null

number_of_last_parallel_class - cislo posledneho krúžku typu int, môže byť null

Subject

id - štandardné identifikačné číslo typu bigint v maximálnom rozsahu 20 symbolov, unikátne, primárny kľúč, ne smie byť null

field_of_study_id - id odboru, pre ktorý je predmet určený typu bigint v maximálnom rozsahu 20 symbolov, cudzí kľúč, smie byť null

teacher_id - id učiteľa, ktorý predmet zastrešuje typu bigint v maximálnom rozsahu 20 symbolov, cudzí kľúč, smie byť null

name - meno predmetu typu varchar v maximálnom rozsahu 255 symbolov, nesmie byť null

short_name - skratka predmetu typu varchar v maximálnom rozsahu 255 symbolov, môže byť null

number_of_students - počet študentov prihlásených na daný predmet typu int, môže byť null

year_of_study - rok štúdia, pre ktorý je daný predmet ponúkaný typu int, nesmie byť null

number_of_lectures - počet prednášok, ktoré musí každý študent týždenne absolvovať typu int, nesmie byť null

number_of_parallel_classes - počet cvičení, ktoré musí každý študent týždenne absolvovať typu int, smie byť null

code - kódové označenie predmetu typu varchar v maximálnom rozsahu 255 symbolov, nesmie byť null

mandatory - údaj o tom, či je predmet povinný typu bit, nesmie byť null

Subject_happening

id - štandardné identifikačné číslo typu bigint v maximálnom rozsahu 20 symbolov, unikátne, primárny kľúč, nesmie byť null

subject_id - id predmetu, ku ktorému sa táto výučba viaže typu bigint v maximálnom rozsahu 20 symbolov, cudzí kľúč, nesmie byť null

hour_id - id hodiny počas ktorej táto výučba prebieha typu bigint v maximálnom rozsahu 20 symbolov, cudzí kľúč, nesmie byť null

room_id - id miestnosti v ktorej ktorej táto výučba prebieha typu bigint v maximálnom rozsahu 20 symbolov, cudzí kľúč, nesmie byť null

type - typ predmetu a síce či ide o prednášku, alebo cvičenie typu int, nesmie byť null

day - poradové číslo dňa počas ktorého výučba prebieha typu int, nesmie byť null

number_of_parallel_class - číslo krúžku pre ktorý je daná výučba, prípadne null aj ide o prednášku, typ int, smie byť null

Teacher

id - štandardné identifikačné číslo typu bigint v maximálnom rozsahu 20 symbolov, unikátne, primárny kľúč, ne smie byť null

department_id - id katedry, pod ktorou daný učiteľ pôsobí typu bigint v maximálnom rozsahu 20 symbolov, cudzí kľúč, smie byť null

first_name - krstné meno učiteľa typu varchar v maximálnom rozsahu 255 symbolov, smie byť null

middle_name - stredné meno učiteľa typu varchar v maximálnom rozsahu 255 symbolov, smie byť null

last_name - priezvisko učiteľa typu varchar v maximálnom rozsahu 255 symbolov, nesmie byť null

title - titul učiteľa typu varchar v maximálnom rozsahu 255 symbolov, môže byť null

Room

id - štandardné identifikačné číslo typu bigint v maximálnom rozsahu 20 symbolov, unikátne, primárny kľúč, nesmie byť null

code - typ miestnosti typu varchar v maximálnom rozsahu 255 znakov, nesmie byť null

name - označenie miestnosti typu varchar v maximálnom rozsahu 255 znakov, nesmie byť null

capacity - kapacita miestnosti typu int, nesmie byť null

School_hour

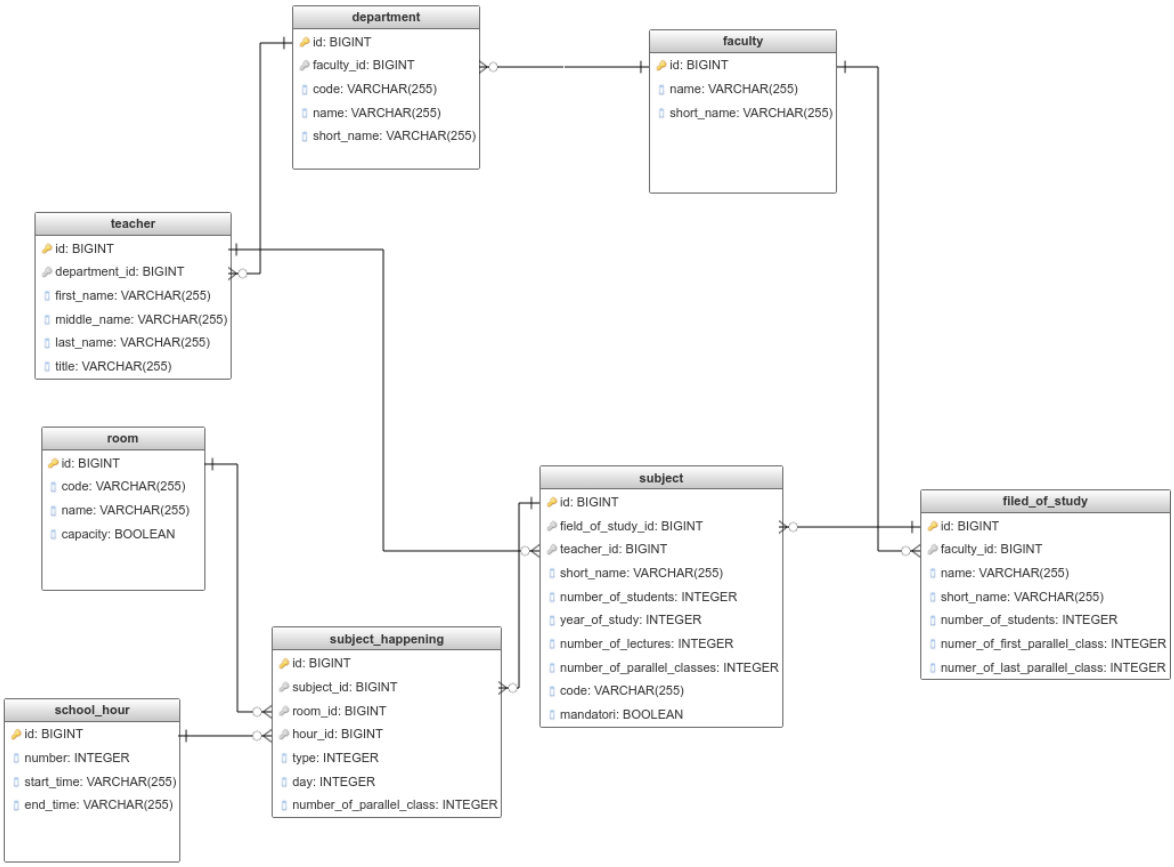
id - štandardné identifikačné číslo typu bigint v maximálnom rozsahu 20 symbolov, unikátne, primárny kľúč, nesmie byť null

number - poradové číslo hodiny typu int, nesmie byť null

start_time - čas začiatku hodiny typu varchar v maximálnom rozsahu 255 znakov, nesmie byť null

end_time - čas ukončenia hodiny typu varchar v maximálnom rozsahu 255 znakov, nesmie byť null

Po identifikovaní všetkých tabuliek a špecifikácii atribútov, ktorými disponujú možme pristúpiť k vyobrazeniu databázového modelu kvôli lepšej prehľadnosti. Toto vyobrazenie je na obrázku 3.



Obrázok 3 Databázový model aplikácie (zdroj: vlastné spracovanie)

1.1.2. User stories

V user stories budeme používať 2 rôzne pomyselné osoby, z ktorých jedna reprezentuje pracovníka školy, prípadne osobu na to poverenú, ktorá má na starosti nahrávanie dát do aplikácie. Túto osobu nazveme správca. Druhou osobou je študent, ktorý požaduje údaje o svojom školskom rozvrhu na daný semester. Táto osoba pre nás ponesie pomenovanie študent. Následne v tabuľke popíšem kto, čo a prečo. A teda kto požaduje danú funkcionality v aplikácii, za akým účelom je táto funkcionalita požadovaná a k čomu bude slúžiť.

Kto	**By chcel mať možnosť**	**Aby**
Správca	Prihlásiť sa do systému	Bol autorizovaný k nahrávaniu súboru za účelom aktualizácie rozvrhu
Správca	Nahrať aktualizovaný súbor do systému	Bola užívateľom poskytnutá aktuálna verzia s najnovšími dátami
Správca	Vidieť rozhranie nahrávania do systému v podobe user friendly UI	Sa nemusel zaťažovať pre neho komplikovaným nahrávaním dát cez API
Študent	Vidieť svoj rozvrh	Bol schopný zúčastniť sa jemu predpísaných hodín v čas a na mieste na to vopred určenom.

1.1.3. Návrh implementácie aplikácie

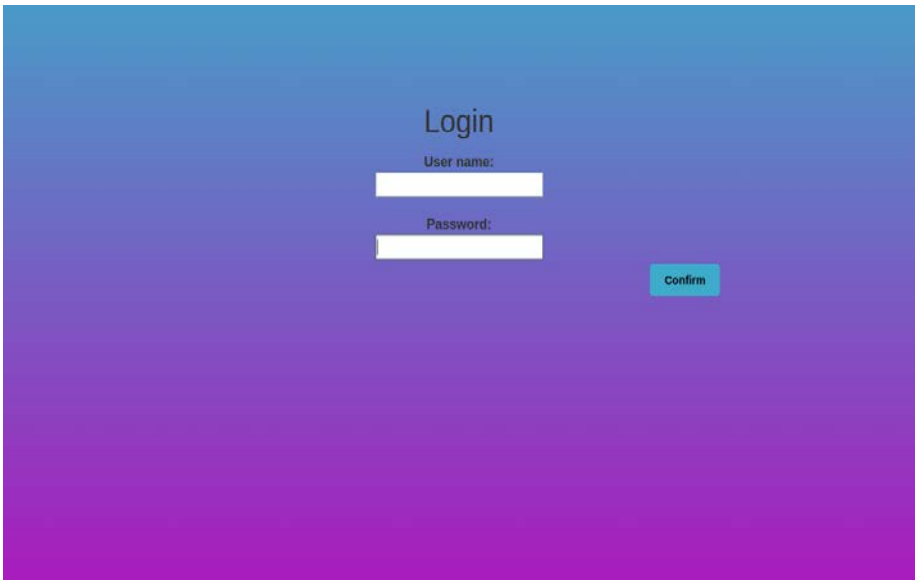
Vzhľadom na predchádzajúce analýzy a návrhy, ktoré sme už absolvovali je čas pustiť sa do návrhu implementácie našej aplikácie. Vzhľadom na požiadavky, ktoré sme definovali a fakt, že vytvárame webovú aplikáciu využijeme klasický model MVC(model view controller), ktorý sa v nasledujúcom odstavci pokúsime podrobne popísať.

Najskôr si predstavme akým spôsobom budeme k aplikácii pristupovať v rámci developerského prostredia, v ktorom bude aplikácia vyvíjaná a z tohto dôvodu budeme aj názorné príklady demonštrovať v tomto prostredí. Základnou URL pre nás bude adresa "http://localhost:8080". Z tejto URL môžeme vyčítať protokol, ktorým je štandardný "http", doménu, ktorou je pre nás localhost a port na ktorom aplikácia beží. Zvolený je štandardný port pre webové aplikácie a síce 8080. Prirodzene doména a port nie sú nemeniteľné a v prípade nasadenia aplikácie či už na testovacie, alebo produkčné prostredia sa bude meniť. V prípade portu ide o ľubovoľné číslo a je možné ho zmeniť podľa prípadných požiadaviek miesta, kde sa bude aplikácia nasadzovať, prípadne podľa obmedzení, ktoré prichádzajú s týmto prostredím.

Vstupný krok k akejkoľvek funkcionalite pre nás predstavuje controller. Controller si môžeme predstaviť ako časť zdrojového kódu, ktorá je zodpovedná za registrovanie requeste prichádzajúceho na špecifickú URL adresu, ktorú controller spravuje. V našom prípade budeme disponovať minimálne tromi rôznymi controllermi. Pričom každý z nich bude prvým krokom k prístupu k nejaké z funkcionalít ktorými aplikácia disponuje.

Login

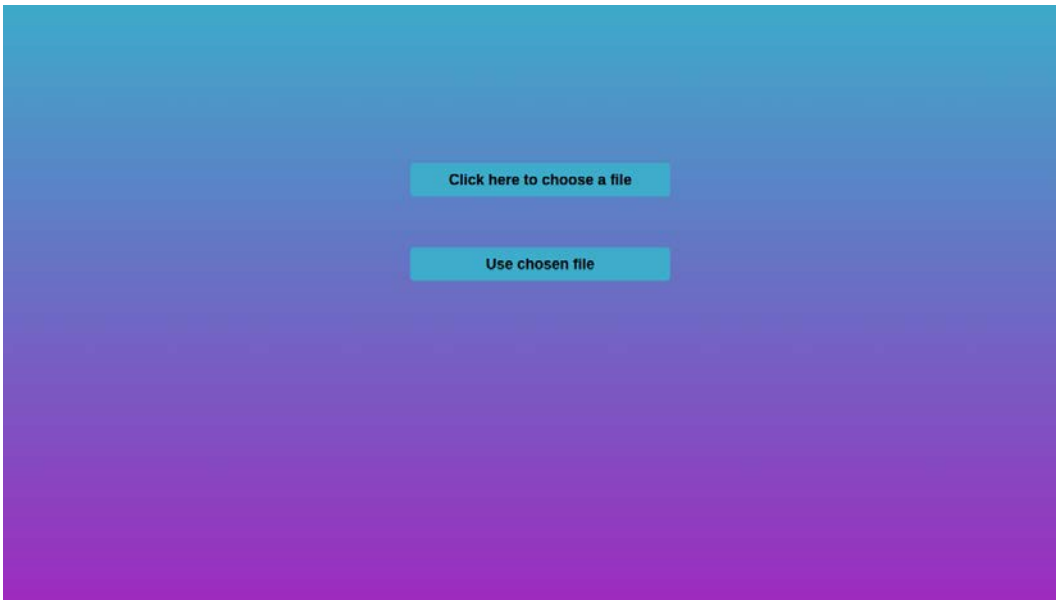
Prihlásenie sa do aplikácie je prvým krokom k prístupu do funkcionality určenej pre autorizované osoby. Pod prihlásením rozumieme sadu prihlasovacích údajov pozostávajúcich z prihlasovacieho mena a hesla. Táto funkcionalita je pre nás dostupná na adrese "http://localhost:8080/login". Tu by mal užívateľ nájsť UI, ktoré mu umožní zadať meno a heslo po zadaní ktorých aplikácia vyhodnotí zadané údaje a umožní užívateľovi ísť na výber súboru na nahrávanie dát, alebo mu oznámi, že zadal nesprávne údaje. Návrh grafiky tohto rozhrania je vidieť na obrázku 4.



Obrázok 4 Návrh dizajnu login obrazovky (zdroj: vlastné spracovanie)

Výber súboru

Po úspešnom prihlásení je užívateľovi umožnené vybrať súbor z lokálneho disku a použiť tento súbor na aktualizáciu dát pre rozvrh. Na túto akciu využije tlačidlo s označením „click here to choose a file“, ktoré možme vidieť na obrázku 5.



Obrázok 5 Tlačidlo pre výber súboru (zdroj: vlastné spracovanie)

V prípade, že užívateľ už daný súbor vybral je o tom informovaný prostredníctvom textu, ktorý sa zmenil v prvom tlačidle. V prípade, že užívateľ požaduje zmeniť prvotne vybraný súbor môže takú akciu vykonať pomocou opätovného stlačenia prvého tlačidla,

ktoré teraz už nesie názov súboru, ktorý prvotne vybral a vybrať súbor, ktorý chce použiť k aktualizácii. Na obrázku 6 je možné vidieť, že sme vybrali súbor s názvom “rozvrh.txt”.

Obrázok 6 Názov vybraného súboru (zdroj: vlastné spracovanie)

Po tom čo si užívateľ vyberie súbor na aktualizáciu dát a potvrdí výber tohto súbor stlačením druhého tlačidla začne prebiehať parsovanie tohto súboru a nahrávanie dát do databázy. Vzhľadom na už vykonanú analýzu bude postup tohto parsovania a nahrávania nasledovnvý. V prvom rade sa vymažú akékoľvek dáta, ktoré boli predtým obsiahnuté v databáze a tým sa pripraví databáza na vkladanie nových dát. Následne algoritmus začne prechádzať celým súborom, ktorý užívateľ vybral a vyhodnotí na akom type riadku sa momentálne nachádza. V prípade, že sa jedná o sekciu, ktorú máme záujem parsovať a vkladať do databázy je daný riadok vyhodnotený ako jeden z tých, ktoré sú pre nás zaujímavé a je priradené vhodný objekt, ktorý sa postará o parsovanie tejto sekcie a vkladanie dát do databázy.

1.1.4. Návrh štruktúry projektu

Vzhľadom na fakt, že každý projekt by mal udržiavať špecifickú a vhodnú štruktúru usporiadania súborov, ktorými disponuje sme sa rozhodli pre dodržanie istých pravidiel aj v tomto projekte. Súbory, ktoré majú generálny charakter sú umiestnené v hlavnom adresári projektu. Medzi takéto súbory patrí napríklad README file, ktorý obsahuje základné informácie o projekte, návod ako projekt spustiť a dodatočné

informácie, ktoré sú vhodné pre osobu, ktorá sa prvý krát stretne s týmto projektom. Všetky súbory, ktoré súvisia so zdrojovým kódom budú umiestnené v adresári “src”. Tento adresár rozdelíme na 2 rôzne časti a to “main” a “test”. Ako už z názvu týchto častí vyplýva adresár main bude určený na zdrojové súbory k hlavnej časti aplikácie, ktorá bude používaná užívateľmi. Adresár test je určený na písanie testov aplikácie, ktoré majú zabezpečovať požadovanú kvalitu funkcionality kódu, majú zabraňovať chybám pri aktualizácii kódu v časti main a tým zvyšovať šancu predikcie chýb, ktoré sú nežiadúce pri zobrazovaní rozvrhu študentom. Časť test rozdeľme ešte na zložku “resources” a zložku “java”, kde v zložke java budú umiestnené java súbory, ktoré obsahujú zdrojový kód testov a zložka resources bude obsahovať textové súbory, ktoré sa budú testovať. Tieto súbory sú časťami súboru, ktorý chce developer otestovať a preto je potrebné ich aktualizovať vždy keď sa mení súbor, ktorý obsahuje všetky dáta o rozvrhu študentov. Časť “main” bude pozostávať z troch rôznych častí, ktorými budú “java”, “resources” a “webapp”. Časť java obsahuje všetky java súbory, ktoré obsahujú zdrojový kód aplikácie. Časť resources obsahuje niektoré konfiguračné súbory, taktiež obsahuje changelog súbor, ktorý je používaný liquibase dependenciou, ktorú budeme používať pri inicializovaní tabuliek v databáze a teda obsahuje štruktúrovaná formu údajov potrebných k správnemu vytvoreniu databázového modelu. Časť webapp bude obsahovať jsp súbory, ktoré pre nás predstavujú FE časť aplikácie a teda obrazovky, s ktorými sa bude užívateľ stretávať.

2.5. Implementácia

Implementácia aplikácie prebiehala v niekoľkých týždňoch a vzhľadom na fakt, že mne osobne sa lepšie programuje bez toho, aby som sa zaťažoval dokumentáciou počas procesu programovania budem opisovať túto kapitolu retrospektívne. Uznávam, že tento spôsob ochudobňuje opis priebehu implementácie o chyby v analýze a návrhu, ktoré boli detegované, avšak na druhej strane sa tým vyhneme zdĺhavým popisom chybných krokov, ktoré by nepriniesli žiadnu dodatočnú informáciu o tom v akej podobe je aplikácia teraz.

V tejto časti prejdeme všetkými časťami kódu, ktorý aplikácia v momentálnej verzii obsahuje a popíšeme, čo dané časti znamenajú v “ľudskej reči” a v niektorých prípadoch odôvodníme prečo sme sa rozhodli práve tak.

V nasledujúcich odstavcoch budeme zdrojový kód v niektorých prípadoch zobrazovať ako text a nie ako obrázky priamo z vývojového prostredia. Dôvodom tohto rozhodnutia je fakt, že obrázky z vývojového prostredia sú značne širšie ako je A4 formát a

teda zdrojový kód by nebol čitateľný ani po zväčšení obrázkov. A taktiež fakt, že zdrojový kód je v niektorých prípadoch dlhší než jedna strana formátu A4 a teda by sme museli pristúpiť k skladaniu viacerých obrázkov, čo by prinieslo nie príliš príjemný dojem.

1.1.5. Výber súboru, parsovanie a vkladanie dát do databázy

Táto funkcionalita začína po tom, čo ImportController dostane dáta, ktoré používateľ vybral a teda bude zavolaná URL adresa "http://localhost:8080/import". V tomto controlleri je pre nás kľúčová metóda submit, ktorú môžeme vidieť na obrázku 7.

```
@Controller
public class ImportController {

    private final Parser parser;
    private final LiquibaseService liquibaseService;

    @Autowired
    public ImportController(final Parser parser, final LiquibaseService liquibaseService) {
        this.parser = parser;
        this.liquibaseService = liquibaseService;
    }

    @RequestMapping(value = "/import", method = RequestMethod.POST)
    public String submit(@RequestParam("file") MultipartFile multiPart,
                         HttpServletResponse httpServletResponse,
                         RedirectAttributes redirectAttrs)
        throws IOException
    {
        String jsp = "success";

        try {
            liquibaseService.redoMigrations();
            parser.parse(multiPart.getInputStream());
        } catch (FileNotFoundException e) {
            jsp = "chooseAFile";
            redirectAttrs.addAttribute("error", "file not found");
            httpServletResponse.sendRedirect( location: "/chooseFile");
        } catch (MigrationsFailedException e) {
            redirectAttrs.addAttribute("error", "migrations failed");
        }

        return jsp;
    }
```

Obrázok 7 ImportController (zdroj: vlastné spracovanie)

Ako môžeme vidieť anotácia requestMapping definuje, že metóda popisuje časť zdrojového kódu, ktorá bude vykonávaná po zachytení request, ktorý ma http typ POST a je cielený na adresu import. Metóda má prístup typu public a teda je globálne dostupná, má návratový typ string a nesie meno submit. Ako parametre očakáva MutliPart súbor menom file, ktorý je vlastne užívateľom súbor určený na aktualizáciu dát rozvrhu. Ďalšie atribúty sú httpServletResponse a redirectAttrs k popisu ktorých sa dostaneme o malú chvíľu. Ako môžeme vidieť metóda hádže IOException, čo je spôsobované faktom, že tu pristupujeme

k dátam súboru a taktiež tomu, že môže nastať situácia, kedy je problém s presmerovaním na chybovú adresu v prípade, že dôjde k problému s čítaním súboru.

Deklarácia String jsp je len deklarovanie návratovej premennej typu string, ktorú táto metóda vracia a pomocou ktorej spring rozhodne o tom, čo sa užívateľovi zobrazí. Tejto premennej rovno priradíme hodnotu "success", keďže to je meno jsp súboru, ktorý v ideálnom prípade chcem užívateľovi zobraziť.

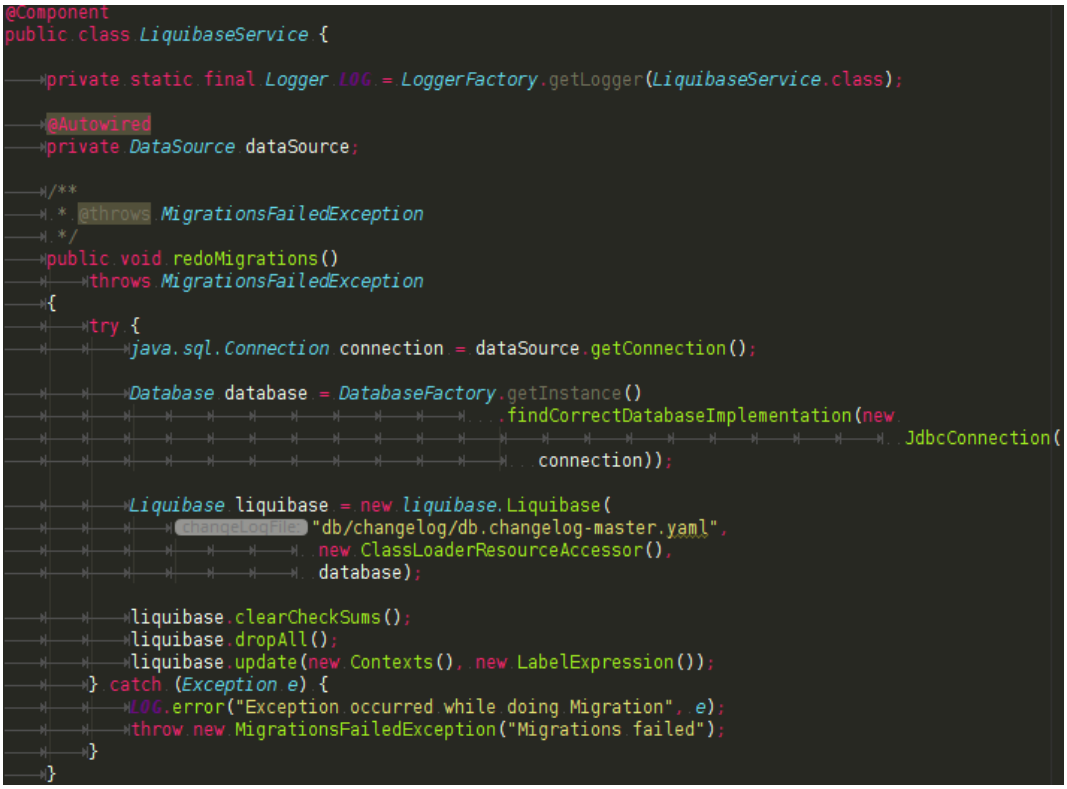
Následne začneme try-catch blok v ktorom zavoláme liquibaseService, ktorý má za úlohu vymazať dáta z databázových tabuliek, ktoré boli naplnené v predchádzajúcom aktualizovaní dát rozvrhu, alebo je to prvá inicializácia tabuliek v prípade, že aplikácia nebola ešte použitá.

V ďalšom riadku zavoláme parser, ktorý začne parameter multiPart parsovať, avšak vzhľadom na fakt, že MultiPart umožňuje prístup dátam iba vo forme vstupného dátového toku použijeme na parsovanie práve tento tok dát. V prípade, že počas parsovania došlo k chybe, ktorú sme nepredvídali je táto chyba zachytená práve pomocou try-catch bloku a následne sa návratová hodnota jsp premennej mení na "chooseAFile", čo je jsp určené na výber súboru. Na tejto stránke sa nám zobrazí taktiež chybová hláška o chybe ku ktorej došlo počas parsovania tak je užívateľ oboznámený s faktom, že ním vybraný súbor na túto akciu nie je vhodný a musí vybrať iný.

Vzhľadom na fakt, že sme použili niekoľko rôznych prvkov, ktorých kód sme ešte nepredstavili je najvyšší čas tak urobiť.

LiquibaseService

Tento servis je zodpovedný za znovu vykonanie migrácii, ktoré upravujú databázu. V našom prípade tento krok vykonávame za účelom vymazania dát rozvrhu, ktoré boli predtým do databázy nahrané. Servis pozostáva len z jednej relatívne jednoduchej metódy. Zdrojový kód tejto metódy vyobrazuje obrázok 8.



Obrázok 8 LiquibaseService (zdroj: vlastné spracovanie)

Ako môžeme vidieť metóda je typu public, čo zaručuje jej globálnu dostupnosť. Návratový typ je void vzhľadom na fakt, že požadujeme jedine vykonanie istých krokov a neočakávame žiadny typ údaju, ktorý by sa mal vrátiť. Meno metódy je redoMigrations, čo popisuje úkon, ktorý metóda vykonáva.

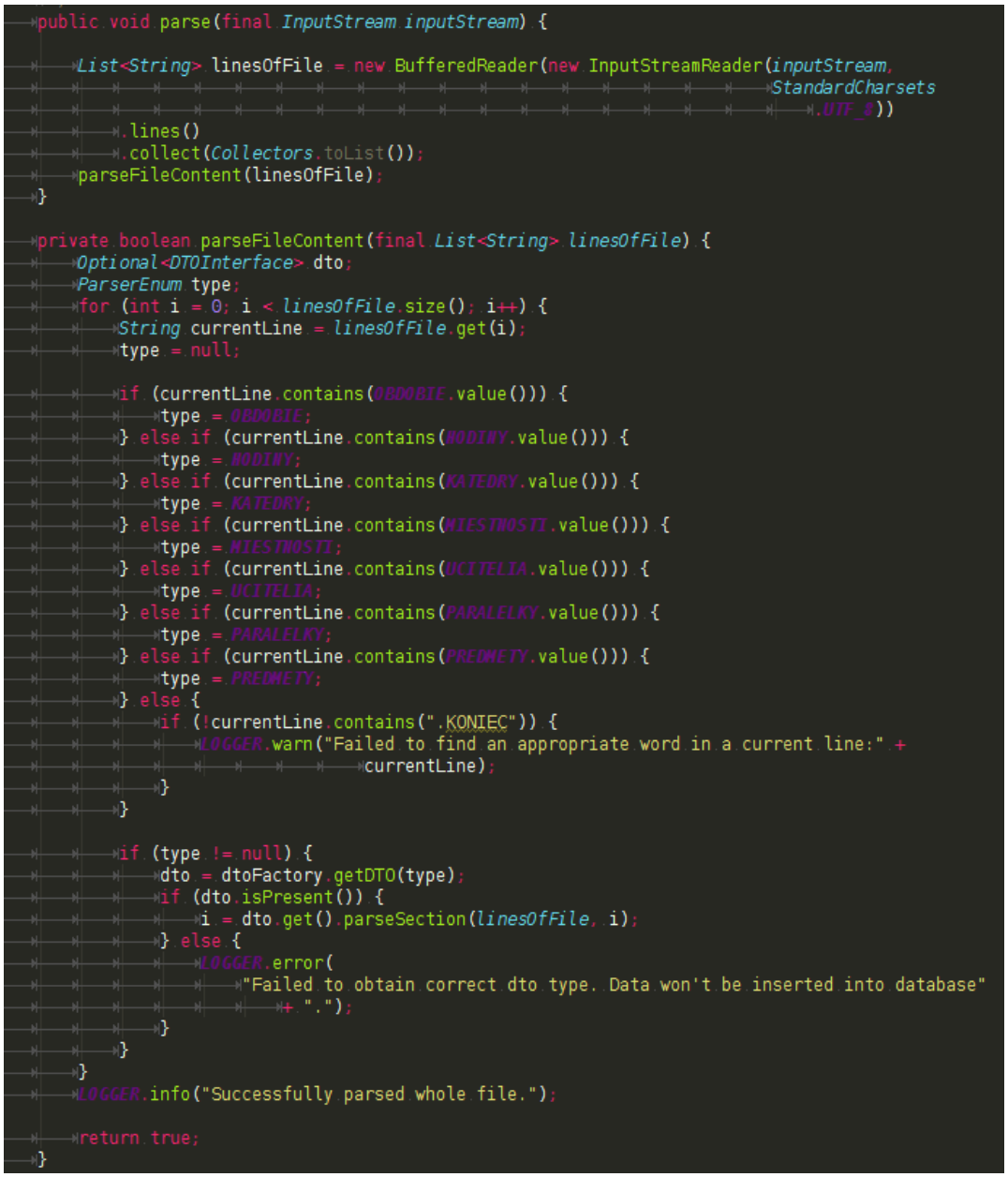
Vzhľadom na fakt, že akcie potrebné na vymazanie tabuliek môžu hodiť výnimku vykonávame ich v try-catch bloku. Prvotne deklarujeme a definujeme connection premennú, ktorá nesie informáciu o prepojení na databázu. Vďaka tejto premennej dokážeme nájsť databázu, ktorá je určená na vykonanie vymazania tabuliek a údajov, ktoré požadujeme. Následne pomocou premennej database dokážeme definovať objekt liquibase, ktorý nesie údaje o databáze a súboru, kde sú nami definované databázové migrácie. Táto knižnica tretej strany dokáže veľmi jednoducho vymazať všetky tabuľky v databáze a opätovne ich vytvoriť. Vzhľadom na fakt, že liquibase nesie informáciu o tom aké aktualizácie databázy už boli vykonané je potrebné vymazať checksums, ktoré práve tieto informácie nesú a pokiaľ by neboli vymazané nedokázali by sme opätovne vytvoriť pomocou tejto metódu tabuľky. V prípade, že počas vykonávania už zmienených operácií dôjde k výnimka je táto výnimka odchytená pomocou catch klauzuly a tento fakt zalogujeme s príslušnou správou ako aj o samotnou výnimkou, aby sme zjednodušili prácu

osoby, ktorá sa bude touto chybou zaoberať a v prípade potreby ju aj odstraňovať. Následne vyhodíme MigrationsFailedException, ktorá je odchytená v ImportControlleri.

Parser

Tento objekt pre nás predstavuje hlavný bod miesta, kde začína parsovanie už zvoleného súboru, ktorý prišiel aplikácii na určenej URL adrese. Základom je metóda parse, ktorú možme vidieť na obrázku 10.

Ako môžeme vidieť ide o krátku metódu s návratovou hodnotou typu void, ktorá nesie parameter inputStream, ktorý predstavuje vstupný tok dát obsiahnutých v súbore na parsovanie. Vykonáme vytvorenie zoznamu reťazcov s názvom linesOfFile, ktorý „pozbiera" všetky riadky vo vstupnom súbore a nastaví ich znakový typ na UTF-8, čo nám umožní čítať aj znaky špecifické pre slovenské písmo. Následne pošleme tento zoznam riadkov do metódy parseFileContent a to je miesto, kde sa nám to začína komplikovať. Túto metódu taktiež vyobrazuje obrázok 9.



```
public void parse(final InputStream inputStream) {

    List<String> linesOfFile = new BufferedReader(new InputStreamReader(inputStream,
                                                                        StandardCharsets
                                                                                .UTF_8))
            .lines()
            .collect(Collectors.toList());
    parseFileContent(linesOfFile);
}

private boolean parseFileContent(final List<String> linesOfFile) {
    Optional<DTOInterface> dto;
    ParserEnum type;
    for (int i = 0; i < linesOfFile.size(); i++) {
        String currentLine = linesOfFile.get(i);
        type = null;

        if (currentLine.contains(OBDOBIE.value())) {
            type = OBDOBIE;
        } else if (currentLine.contains(HODINY.value())) {
            type = HODINY;
        } else if (currentLine.contains(KATEDRY.value())) {
            type = KATEDRY;
        } else if (currentLine.contains(MIESTNOSTI.value())) {
            type = MIESTNOSTI;
        } else if (currentLine.contains(UCITELIA.value())) {
            type = UCITELIA;
        } else if (currentLine.contains(PARALELKY.value())) {
            type = PARALELKY;
        } else if (currentLine.contains(PREDMETY.value())) {
            type = PREDMETY;
        } else {
            if (!currentLine.contains(".KONIEC")) {
                LOGGER.warn("Failed to find an appropriate word in a current line:" +
                                    currentLine);
            }
        }

        if (type != null) {
            dto = dtoFactory.getDTO(type);
            if (dto.isPresent()) {
                i = dto.get().parseSection(linesOfFile, i);
            } else {
                LOGGER.error(
                        "Failed to obtain correct dto type. Data won't be inserted into database"
                                + ".");
            }
        }
    }
    LOGGER.info("Successfully parsed whole file.");

    return true;
}
```

Obrázok 9 Parser (zdroj: vlastné spracovanie)

Táto metóda je už typu private a teda prístup k nej máme len zvnútra objektu Parser. Ako parameter má zoznam riadkov, ktoré sme vyššie extrahovali zo vstupného toku súboru. Prvotne, ako to v jave chodí, definujeme premenné, ktoré budeme využívať. V našom prípade to je dto, ktorý predstavuje data transform object. Uznávam, že to nie je najšťastnejšie pomenovanie, vzhľadom na to, že štandardne sa pod touto skratkou skrýva data transfer object, ale žiaľ je to názov, ktorý vystihuje podstatu akcií prebiehajúcich v týchto objektoch. Následne tu máme premennú typ, ktorá je typu ParserEnum a slúži na inicializáciu dto pomocou factory patternu. Ako môžeme vidieť budeme prechádzať každým prvkom zoznamu riadkov a budeme v ňom hľadať slová, ktoré označujú sekcie.

V prípade, že sa nám nepodarí v danom riadku nájsť hľadané slovo a nejedná sa o koniec súboru zalogujeme tento úkaz na úrovni upozornenie, keďže nemusí nevyhnutne ísť o chybu, ale môže ísť len o časť súboru, ktorá pre nás nie je zaujímavá. V opačnom prípade nastavím premennú type na príslušný enumerátor triedy ParserEnum.

Ak sme teda nastavili premennú typ pokúsime sa o získanie správneho data transform objektu ako sme už spomínali pomocou factory patternu. V prípade, že sa nám nepodarilo nájsť príslušný data transform object zalogujeme túto chybu na úrovni chyby, keďže sa jedná o chybu, ktorá by nemala nastať. V opačnom prípade sme práve našli začiatok časti sekcie, ktorá je pre nás zaujímavá a teda začneme túto sekciu zdolávať po častiach, ktoré sme už predstavili v časti analýza. Vzhľadom na fakt, že factory pattern je určený, medzi iným, na schovávanie inicializácie za účelom obmeny inicializácie bez znakov vonkajšej zmeny uvedieme akým spôsobom funguje DTOFactory, pomocou ktorej sme inicializovali premennú dto.

```
public Optional<DTOInterface> getDTO(final ParserEnum parserEnum) {
    LOG.debug("Getting DTO for enum: " + parserEnum.toString());
    switch (parserEnum) {
    case OBDOBIE:
        return Optional.of(new ObdobieDTO());
    case HODINY:
        return Optional.of(new SchoolHoursDTO(schoolHourDAO));
    case KATEDRY:
        return Optional.of(new DepartmentSectionDTO(departmentDAO));
    case MIESTNOSTI:
        return Optional.of(new RoomSectionDTO(roomDAO));
    case UCITELIA:
        return Optional.of(new TeacherSectionDTO(teacherDAO));
    case PARALELKY:
        return Optional.of(new ParallelsSectionDTO(fieldOfStudyDAO, facultyDAO));
    case PREDMETY:
        return Optional.of(new SubjectDTO(subjectDAO, subjectHappeningDAO, departmentDAO));
    default:
        return Optional.empty();
    }
}
```

Obrázok 10 DTOFactory getDTO metóda (zdroj: vlastné spracovanie)

Ako môžeme na obrázku 10 vidieť na základe parametra parserEnum budeme vracať návratový typ DTOInterface, ktorý predstavuje rozhranie pre všetky typy data transform objektov. Vzhľadom na fakt, že môže nastať situácia kedy nenájdeme požadovaný data transform object budeme vracať Optional vzhľadom na to, že je táto časť Javy 8 lepšie definovaná čo sa týka hodnoty null. A teda na základe parserEnum parametra budeme vytvárať data transform objekty, ktoré vytvoríme s pre inicializáciu potrebnými data access objektami, ktoré budu v data transform objektoch využité za účelom vkladania dát do databázy. Avšak aby to nebolo také jednoduché každý data transfer object obsahuje parseSection metódu, ktorú sme volali v parseri. Preto sme sa rozhodli k extrakcii tejto

metódy do AbstractDTO triedy, ktorá je abstraktnou a každý data transform object ju dedí. Vzhľadom na to, že sme popísali náš facory pattern je čas vrátiť sa práve k tejto abstraktnej triede a popísať prvky, ktoré sú dôležité. Metóda teda na základe zoznamu riadkov obsiahnutých vo vstupnom súbore a indexu riadku na ktorom sa momentálne nachádzame identifikuje všetky riadky sekcie, v ktorej sa momentálne nachádzame. Vykonáva to pomocou DTOUtil triedy, ktorá práve tieto riadky identifikuje. Toto správanie môžeme vidieť na obrázku 11.

```
public FileContentHandler parseSection(final List<String> fileLines, int row) {
    LOG.debug("Section parsing started on row " + row);
    FileContentHandler fileContentHandler = new FileContentHandler();

    row++; //currently on line matching Pattern, time to move forward
    while (!matchesPattern(fileLines.get(row))) {
        if (!fileLines.get(row).isEmpty()) {
            LOG.debug("Adding line: " + fileLines.get(row));
            fileContentHandler.addLineToSectionLines(fileLines.get(row));
        }
        row++;
    }

    row--;
    fileContentHandler.setRow(row);

    LOG.debug("Section parsing done");
    return fileContentHandler;
}
```

Obrázok 11 DTOUtil parseSection metóda (zdroj: vlastné spracovanie)

Na začiatku sa posunieme o riadok nižšie, vzhľadom na fakt, že momentálne sa nachádzame na riadku, kde je definované meno sekcie. Pomocou cyklu while sa budeme posúvať nižšie a nižšie až kým nájdeme meno ďalšej sekcie. V tomto momente posunieme index o jedno číslo späť, aby sme túto sekciu v parseri mohli identifikovať a priradiť príslušný data transform object. V prípade, že sa stále nachádzame v nami parsovanej sekcii používame FileContentHandler triedu, ktorá je pre nás pomocnou a slúži na udržanie si údajov o riadkoch, ktoré sú v našej sekcii a taktiež o indexe posledného riadku. Tento index je vrátený parseru, ktorý odtiaľ začne znovu súbor prehľadávať a hľadať sekcie. Nás však budú teraz zaujímať riadky, ktoré sme práve prečítali a pošleme ich do metódy transformLinesOfSectionAndInsertThem. Táto metóda má anotáciu override v každej data tranform object triede, čo znamená, že jej pôvodné inštrukcie sú nahradené tými, ktoré majú práve túto anotáciu. V našom prípade bola pôvodná metóda abstraktná a teda nemala žiadne telo. Tu sa dostávame k náročnej časti, kedy budú riadky parsované podľa už

vykonanej analýzy. Každá z data transfer object tried teda obsahuje metódu transformLinesOfSectionAndInsertThem, ktorá volá privátnu metódu, ktorá má na starosti rozparsovanie riadku a následné volania data access objektov, ktoré vkladajú dáta do databázy.

Pred tým, než sa pustíme do popisu jednotlivých implementácií data transform objektov musíme vysvetliť jeden dôležitý pojem zvaný regulárny výraz, alebo skrátene regex. Regex je špeciálny vzor pre textové reťazce. Tento vzor je akási forma jazyka, pomocou ktorej dokážeme rozoznávať znaky, prípadne celé slová, alebo dokonca vety skrátenou formou a teda nemusíme vymenovať všetky znaky, ktoré hľadáme. Práve regulárne výrazy nám umožnia pracovať s časťami riadkov, ktoré sme už analyzovali.

V našom zdrojovom kóde budeme používať triedu Matcher, ktorá dokáže práve regulárne výrazy vyhodnocovať a spájať znaky do skupín, ktoré pre nás budu užitočné. Takéto spájanie do skupín je vykonávané pomocou znakov klasických zátvoriek, kde všetky znaky vo vnútri týchto zátvoriek tvoria jednu skupinu a teda v prípade, že by sme použili regex ([a-z]+)(0-9) dostaneme tak pomocou triedy Matcher 3 rôzne skupiny. Prvou skupinou je celá veta, slovo, prípadne znak, ktorý bol rozoznávaný. Druhou skupinou sú znaky malé a až malé z, ktoré sú zoradené za sebou v ASCII tabuľke. Bez ohľadu na to koľko takýchto znakov nájdeme v poradí za sebou budú umiestnené v druhej skupine. V tretej skupine bude jedno číslo od nuly po deväť. Pre názornú ukážku pri parsovaní vety abeceda1 by sme teda ako prvú skupina mali celé slovo abeceda1, druhá skupina by pozostávala zo slova abeceda a tretie skupina by bolo číslo 1. Prirodzene nás zaujímajú výlučne tie vety, v našom ponímaní riadky, ktoré vyhovujú nášmu regulárnemu výrazu a na to sa nám hodín ďalšia metóda triedy Matcher nazývaná matches. Táto nám po jej zavolaní vráti boolean hodnotu, ktorá reprezentuje či analyzovaná veta, respektíve riadok vyhovujú nášmu regexu a to hodnou true, v opačnom prípade nám vráti false.

Je potrebné podotknúť, že vzhľadom na kompilovanie javy niektoré znaky vyzerajú byť escapované viackrát, ale to je len dôsledok samotnej reprezentácie týchto znakov. Takýmto príkladom je napríklad regexový znak pre hocijaký biely znak “\s”, ktorý v java stringu obsahuje jednu lomku naviac, ktorá escapuje lomku pred znakom s.

Vzhľadom na to, že práca s regexmi je výpočtovo náročná regexy obsiahnuté v zdrojom kóde sú statické finálne premenné, ktoré sú skompilované iba raz a vzhľadom na fakt, že tieto regexy sú využívané jedine v triedach kde sú definované majú jedine private prístupový modifikátor a teda je možné k nim pristupovať len z triedy kde sú definované.

DepartmentSectionDTO

Táto trieda je pomerne jednoduchá a krátka, na začiatku definujeme Matcher, ktorá je porovnávaná s momentálnym riadkov. V prípade, že tento riadok nevyhovuje nášmu patternu nastala situácia, ktorá by nemala nastávať a síce sme našli riadok v časti katedra, ktorý nemá tvar katedry aký očakávame a preto vyhodíme výnimku PatternDoesntMatchException, ktorej správa bude momentálny riadok. V opačnom prípade definujeme novú katedru a to pomocou builder patternu, ktorý využívame pri vytváraní všetkých objektov, ktoré vkladáme do databázy. Ako môžeme vidieť meno katedry je skupina s indexom 1 nášho matchera, skratka katedry je uložená v skupine indexovanej číslom 2 a kód katedry číslom 3. Následne je tento objekt vytvorený pomocou metódy build, ktorá je súčasťou build patternu a takto vytvorený objekt je poslaný do DepartmentDAO. Časť pre data access objekty rozoberieme neskôr z dôvodu prehľadnosti a aby sme nekúskovali časť data transfer objektov. Zdrojový kód tejto metódy je uvedený nižšie.

```
private void transformLineIntoKatedra(final String line) {
        LOG.debug("Transforming line into department: " + line);
        Matcher matcher = PATTERN.matcher(line);
        if (!matcher.matches()) {
            throw new PatternDoesntMatchException(line);
        }
        Department department = Department.newDepartment()
            .name(matcher.group(1).replaceAll("_", " "))
            .shortName(matcher.group(2))
            .code(matcher.group(3))
            .build();
        departmentDAO.insertIntoDatabase(department);
        LOG.debug("Transformed into: " + department);
    }
```

Regulárny výraz katedry je len do regulárneho jazyka prepísaná analýza, ktorú sme už vykonali a premenná, ktorá sa v tomto data transfer objekte nachádza vyzerá nasledovne:

private static final Pattern PATTERN = Pattern.compile("([a-zA-Z_]+)\\s([a-zA-Z]+)\\s(\\d+)\\s(\\d).*");

ParallelsSectionDTO

Táto časť obsahuje nasledovné regulárne výrazy:

YEAR_PATTERN = Pattern.compile("\\s+([a-z_0-9]+)\\s(\\d).*");

FACULTY_PATTERN = Pattern.compile("\\s+\\$\\s([a-zA-Z_]+)\\s([A-Z]+).*");

FIELD_OF_STUDY_PATTERN=

Pattern.compile("\\s+\\${2}\\s([a-zA-Z_]+)\\s([a-zA-Z0-9]+)\\s(\\d+)/(\\d)-*(\\d*).*");

Práve tieto výrazy sú vyhodnocované v nasledujúcom zdrojovom kóde nám už známej metóde transformAndInsertLine. Tu sa už dostávame k dlhšej časti zdrojového kódu, kde na začiatku definujeme a vyhodnotíme premenné yearMatcher, facultyMatcher a fielOfStudyMatcher na základe už uvedených regulárnych výrazov. V prípade, že ani jedna z týchto premenných nie je vyhodnotená na základe metódy matches ako true hodíme PatternDoesntMatchException s momentálnym riadkom, na ktorom sa nachádzame, takýto prípad by totižto nemal nastať.

V ideálnom prípade je jedna z našich premenných vyhodnotená ako true. V prípade, že touto premennou je yearMatcher inicializujeme premennú triedy yearMatcher, ktorá nesie údaj o ročníku, ktorý momentálne načítavame. Vzhľadom na už obsiahly zdrojový kód práve špecifikovanej metódy sme niektoré iné metódy a premenné preskočili a práve toto je jedna z nich.

Ak nastane situácia, že ako true je vyhodnotená premenná facultyMatcher vytvoríme nový faculty objekt, ktorý inicializujeme pomocou builder patternu a to nasledovne. Meno fakulty je obsiahnuté v druhej skupine a síce s indexom 1 facultyMatchera. Vzhľadom na štruktúru mena fakulty vymeníme všetky znaky podčiarkovníka za znaky medzery pomocou metódy replaceAll. Skratka katedry je v skupine s indexom 2 a keďže fakulta obsahuje len tieto 2 údaje vytvoríme objekt pomocou metódy build. Následne tento objekt pošleme do FacultyDAO objektu, ktorý nám vráti už v databáze vytvorený faculty objekt a práve tento objekt uložíme do druhej a poslednej premennej našej triedy, nápodobne ako yearOfStudy.

Posledným možným prípadom je vyhodnotenie premennej fieldOfStudyMatcher ako true a v takomto prípade začneme s vytváraním inštancie objektu FieldOfStudy.Builder. Je tu malý rozdiel v tom, že tento objekt rovno nevytvoríme

pomocou metódy build a to v dôsledku toho, že FieldOfStudy obsahuje pole numberOfLastParallelClass, ktoré môže, ale nemusí byť definované. Najskôr teda vyplníme year buildera to premennou triedy yearOfStudy. Následne vyplníme meno fieldOfStudy a to tak, že vyberieme skupinu s indexom 1 premennej fieldOfStudyMatcher a nahradíme všetky nájdené znaky podčiarkovníka za medzery. Postupne vyplníme všetky atribúty príslušnými skupinami premennej fieldOfStudyMatcher. Tu sa dostávame do bodu, kedy musím zistiť, či daný riadok obsahuje údaj o poslednom krúžku. To zistíme jednoducho. Len pozrieme koľko skupín má premenná fieldOfStudyMatcher. Počet skupín tejto premennej dostaneme zavolaním metódy groupCount a v prípade, že sa tento počet rovná číslu 6 vyplníme tento údaj. Po vyplnení všetkých atribútov môžeme vytvoriť premennú fieldOfStudy, ktorú sme práve deklarovali a môžeme zavolať FieldOfStudyDAO objekt s použitím práve vytvoreného objektu na vloženie nového odboru do databázy. Vzhľadom na možno mierne chaotické popísanie zdrojového kódu uvedieme jej zdrojový kód:

```
private void transformAndInsertLine(final String line) {
        LOG.debug("Transforming line: " + line);
        Matcher yearMatcher = YEAR_PATTERN.matcher(line);
        Matcher facultyMatcher = FACULTY_PATTERN.matcher(line);
        Matcher fieldOfStudyMatcher =
FIELD_OF_STUDY_PATTERN.matcher(line);
        if (!yearMatcher.matches() && !facultyMatcher.matches() &&
!fieldOfStudyMatcher.matches()) {
            throw new PatternDoesntMatchException(line);
        }
        if (yearMatcher.matches()) {
            LOG.debug("Parsed line and found year of study = " +
yearMatcher.group(2));
            yearOfStudy = Integer.parseInt(yearMatcher.group(2));
        } else if (facultyMatcher.matches()) {
            faculty = Faculty.newFaculty()
            .name(facultyMatcher.group(1).replaceAll("_", " "))
            .shortName(facultyMatcher.group(2))
                .build();
```

```
                LOG.debug("Transformer into: " + faculty);
                faculty = facultyDAO.insertIntoDatabaseAndReturnSaved(faculty);
            } else if (fieldOfStudyMatcher.matches()) {
                FieldOfStudy.Builder fieldOfStudyBuilder =
FieldOfStudy.newFieldOfStudy()
                .year(yearOfStudy)
                .name(fieldOfStudyMatcher.group(1).replaceAll("_", " "))
                .shortName(fieldOfStudyMatcher.group(2))
                .numberOfStudents(Integer.parseInt(fieldOfStudyMatcher.group(3))
            )
                .numberOfFirstParallelClass(Integer.parseInt(fieldOfStudyMatcher.
            group(4)));
                        if(fieldOfStudyMatcher.groupCount() == 6) {
            fieldOfStudyBuilder.numberOfLastParallelClass(Integer.parseInt(fieldOfSt
            udyMatcher.group(5)));
                        }
                        final FieldOfStudy fieldOfStudy =
            fieldOfStudyBuilder.build();
                        LOG.debug("Transformed into: " + fieldOfStudy);
                        if(faculty != null) {

                fieldOfStudyDAO.insertIntoDatabase(fieldOfStudy,
            faculty.getName());
                        } else {
                                throw new
            MissingInitializationException("Faculty should have been initialized! For
            line " + line);
                        }
                    }
```

RoomSectionDTO

Táto trieda obsahuje len 1 regulárny výraz nakoľko všetky miestnosti majú len 1 tvar a týmto výrazom je premenná PATTERN, ktorá vyzerá nasledovne:

```
private static final Pattern PATTERN = Pattern.compile("([A-Z])\\s([A-Z_0-9]+)\\s([0-9]+)");
```

Podľa už nami používaného štandardu na začiatku vytvoríme a vyhodnotíme premennú matcher a v prípade, že sa na ňu zavolaná metóda matcher nerovná true vyhodíme nám už známu PatternDoesntMatchException s momentálnym riadkom. Prikladáme zdrojový kód pre lepšiu predstavu:

```
private void transformLineIntoRoom(final String line) {
        LOG.debug("Transforming line into room" + line);
        Matcher matcher = PATTERN.matcher(line);
        if (!matcher.matches()) {
                throw new PatternDoesntMatchException(line);
        }
        Room room = Room.newRoom()
                                .code(matcher.group(1))
                                .name(matcher.group(2).replaceAll("_", "."))
                                .capacity(Integer.parseInt(matcher.group(3)))
                                .build();
        roomDAO.insertIntoDatabase(room);
        LOG.debug("Transformed into: " + room);
}
```

V opačnom prípade začneme s vytváraním objektu room, kde kód miestnosti obsahuje skupina s indexom 1, meno miestnosti skupina indexovaná 2, kde nezabudneme na nahradenie znaku podčiarkovníka za znak medzery a kapacita miestnosti je obsiahnutá v skupine s indexom 3. Následne môžeme tento objekt vytvoriť zavolaním metódy build a poslať ho do príslušnej implementácie data access objektu. V tomto prípade to bude RoomDAO.

SchoolHoursDTO

Táto trieda je tiež jedna z jednoduchších ako na pochopenie, tak aj na naprogramovanie. Obsahuje jeden regulárny výraz, ktorý vyzerá nasledovne:

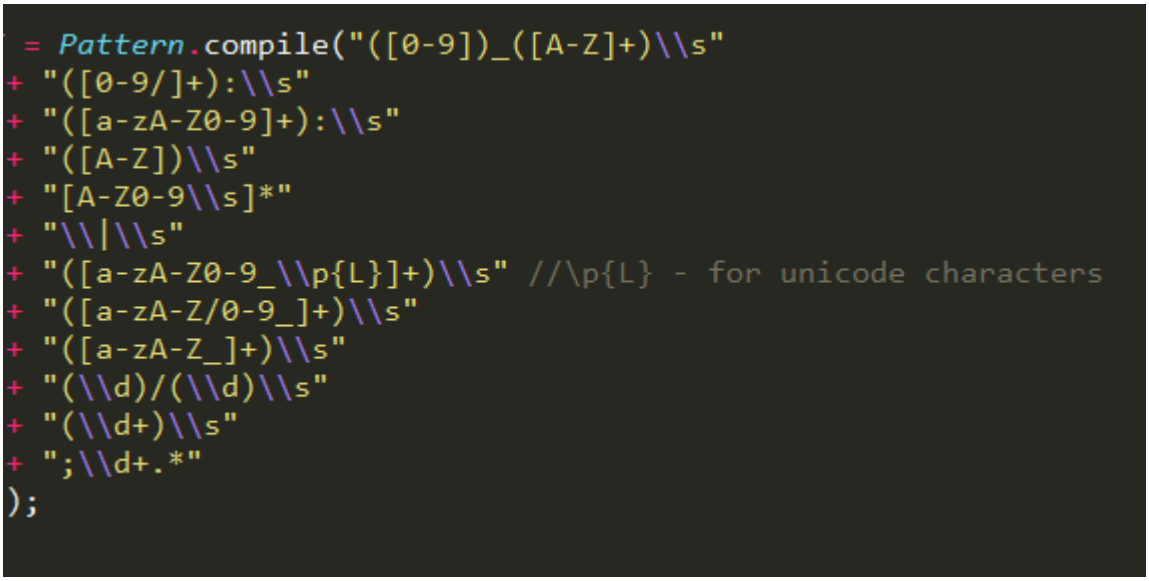
```
private static final Pattern PATTERN = Pattern.compile("(\\d)\\s(\\d+)\\.(\\d+)\\s-\\s(\\d+)\\.(\\d+)");
```

Metóda je implementovaním nám už známeho štandardu, ktorý používame pri data transform objektoch. Definujeme a vyhodnotíme premennú matcher. V prípade, že je na ňu zavolaná metóda matches hodnoty false vyhodíme PatternDoesntMatchException. Inak vytvoríme schoolHour inštanciu, kde nevyužívame builder pattern kvôli jednoduchosti a úplnosti údajov. A teda použitím parametrického konštruktora vytvoríme jednotlivé parametre, ktoré do tohto konštruktora pošleme. Prvým parametrom je druhá skupina s indexom 1 nášho matchera, ktorá je typu String a keďže konštruktor očakáva typ Integer pretypujeme tento String pomocou statickej metódy Integer.parseInt. Druhým parametrom je začiatočný čas hodiny, ktorý pozostáva s dvoch skupín premennej matcher. Použijeme teda skupiny s indexom 2 a 3, ktoré oddelíme znakom dvojbodky. Posledným parametrom je čas konca hodiny, kde použijeme skupiny 4 a 5, ktoré su taktiež oddelené znakom dvojbodky. Následne nami vytvorený objekt vložíme do databázy pomocou SchoolHourDAO triedy, konkrétne jej metódy insertIntoDatabase. Zdrojový kód tejto metódy vyzerá nasledovne:

```
private void transformIntoSchoolHour(final String line) {
		LOG.info("Transforming line into school hour: " + line);
		Matcher matcher = PATTERN.matcher(line);
		if (!matcher.matches()) {
			throw new PatternDoesntMatchException(line);
		}
		SchoolHour schoolHour = new
SchoolHour(Integer.parseInt(matcher.group(1)),
matcher.group(2) + ':' + matcher.group(3),
matcher.group(4) + ':' + matcher.group(5));
		LOG.debug("Transformed into: " + schoolHour);
		schoolHourDAO.insertIntoDatabase(schoolHour);
	}
```

SubjectDTO

Táto implementácia DTOInterface rozhrania je najdlhšia a teda pusťme sa do toho. Trieda disponuje 5 rôznymi regulárnymi výrazmi. Pre zjednodušenie uvedieme len jeden z týchto regulárnych výrazov a to ten, ktorý zodpovedá predmetu, ktorý nemá definovaná odbor. Tento regulárny výraz môžeme vidieť na obrázku 12.



Obrázok 12 Regulárny výraz pre predmet bez definvaného odboru (zdroj: vlastné spracovanie)

Metóda transformLinesOfSectionAndInsertThem, ktorá obyčajne odošle do privátnej metódy len 1 riadok tu pracuje a inak a do privátnej metódy odošle všetky riadky obsahujúce dáta o predmetoch. V tejto implementácii najprv definujeme premenné typu Matcher. Tieto premenné však hneď neiinicializujeme a to z dôvodu, že nebudeme potrebovať vyhodnotenie všetkých hneď naraz. Máme tu aj definovanie premennej subject, ktorá sa bude vzťahovať na viac vyučovacích hodín a teda bude pre nás ešte významná o chvíľku neskôr. Začneme teda prechádzať všetky riadky sekcie s vyučovacími hodinami pomocou for cyklu. Tu vyhodnotíme premenné subjectMatcherWithoutFieldOfStudy, subjectMatcherWitFieldOfStudy a parallelClassMatcher, ak nami čítaný riadok obsahuje jeden zo znakov, ktorými sa začínajú riadky, ktoré pre nás nemajú informatívnu hodnotu tieto riadky preskočíme, čo môžeme vidieť v prvej if podmienke for cyklu. Následne skontrolujeme, či aspoň jeden z vyhodnotených matcherov je true. Ak táto situácia nenastala vyhodíme PatternDoesntMatchException, ktorá zodpovedá tejto situácii. V opačnom prípade začneme hľadať ktorá z našich premenných nesie hodnotu true. Najskôr

zisťujeme hodnoty premenných, ktoré vyhodnocujú, či daný riadok zodpovedá formáte zadávania predmetu. Ak je to tak, začneme vytvárať objekt Subject. Vzhľadom na to, že v tomto objekte očakávame isté identifikátory a to konkrétne ID fakulty a ID učiteľa vytvoríme inštanciu pomocnej triedu SubjectWithoutIds, ktorá bude obsahovať namiesto týchto identifikačných prvkov obsahovať údaje potrebné na nájdenie ID fakulty a učiteľa, ale v jej kompozícii je aj trieda Subject. V našom prípade sú týmito údajmi plné meno učiteľa a skratka fakulty. Pri vytváraní tejto inštancie berieme do úvahy, či momentálny riadok obsahuje údaj o odbore pre ktorý je predmet vyučovaný, alebo nie. Ako môžeme vidieť využívame privátnu metódu createSubject, ktorá slúži na vytváranie už spomínanej inštancie za pomoci príslušného matchera. Po tom čo sme túto inštanciu vytvorili zavoláme SubjectDAO a to konkrétne jej metódu insertIntoDatabase s parametrami, ktoré sú nevyhnutné na vytvorenie objektu Subject v databáze. Vzhľadom na fakt, že v sekciách, ktoré sme už spracovali nebol údaj o tom, ktoré katedry spadajú pod aké fakulty aktualizujeme aj tento údaj pomocou metódy updateFacultyId triedy DepartmentDAO. Toto bol teda opísaný postup toho ako si počíname v prípade, že premenné zodpovedajúce riadku predmetu majú hodnotu true. V prípade, že to ale tak nie je ostáva iba jedna možnosť a to tá, že nami čítaný riadok predstavuje buď údaje o výučbe prednášky, alebo o výučbe cvičenia. Z tohto dôvodu vyhodnotíme premenné lectureMatcher a exerciseMatcher. Po vyhodnotení týchto premenných použitím príslušných regulárnych výrazom pomocou if else podmienky zistíme, či ide o prednášku, alebo cvičenie. V oboch prípadoch pošleme do SubjectHappeningDAO triedy údaje potrebné na vytvorenie záznamu v databáze. Jediným ešte nepredstaveným špecifikom je fakt, že číselná reprezentácia dňa je z hľadiska prehľadnosti v zdrojom kóde reprezentovaná enumeráciou DayEnum, ktorá obsahuje slovnú reprezentáciu poradového čísla dňa, avšak v databáze je uložená číselná reprezentácia. Dôvodom tejto zmeny je len uľahčenie prehľadnosti. Práve táto časť aplikácie je pre nás mimoriadne dôležitá a tak aj napriek rozsiahlej dĺžke zdrojového kódu ho tu vložíme, aby sa predišlo možných komplikáciám pri interpretácii tohto kódu:

Zdrojový kód metódy s najdôležitejšou časťou kódu vyzerá nasledovne:

```
private void transformAndInsert(List<String> sectionLines) {
                Matcher subjectMatcherWithoutFieldOfStudy,
subjectMatcherWithFieldOfStudy, parallelClassMatcher, lectureMatcher, exerciseMatcher;
                Subject subject = null;
```

```
                for (int i = 0; i < sectionLines.size(); i++) {
                        String currentLine = sectionLines.get(i);
                        subjectMatcherWithoutFieldOfStudy =
SUBJECT_PATTERN_WITHOUT_FIELD_OF_STUDY.matcher(currentLine);
                        subjectMatcherWithFieldOfStudy =
SUBJECT_PATTERN_WITH_FIELD_OF_STUDY.matcher(currentLine);
                        parallelClassMatcher =
PARALLEL_CLASS_PATTERN.matcher(currentLine);

        if(LINES_WITH_CHARACTERS_TO_SKIP.stream().filter(character ->
currentLine.contains(character)).findAny().isPresent()) {
                                continue;
                        } else if (!subjectMatcherWithoutFieldOfStudy.matches()
                                && !subjectMatcherWithFieldOfStudy.matches()
                                && !parallelClassMatcher.matches()
                                && !currentLine.isEmpty()) {
                                throw new PatternDoesntMatchException(currentLine);
                        } else if (subjectMatcherWithoutFieldOfStudy.matches() ||
subjectMatcherWithFieldOfStudy.matches()) {
                                LOG.debug("Transforming line into subject: " +
currentLine);
                                SubjectWithoutIds subjectWithoutIds;
                                if(subjectMatcherWithFieldOfStudy.matches()) {
                                        subjectWithoutIds =
createSubject(subjectMatcherWithFieldOfStudy, true);
                                } else {
                                        subjectWithoutIds =
createSubject(subjectMatcherWithoutFieldOfStudy, false);
                                }

        if(StringUtils.isEmpty(subjectWithoutIds.getFacultyShortName())) {
                                        LOG.error("Faculty short name is not set. Line: ",
currentLine);
                                }
```

```
                    LOG.debug("Transformed into: " + subjectWithoutIds);
                    subject =
subjectDAO.insertIntoDatabase(subjectWithoutIds.getSubject(),
subjectWithoutIds.getFacultyShortName(),
subjectWithoutIds.getFieldOfStudyShortName(),
subjectWithoutIds.getDepartmentShortName(), subjectWithoutIds.getTeacherFullName());

        departmentDAO.updateFacultyId(subjectWithoutIds.getFacultyShortName(),
subjectWithoutIds.getDepartmentShortName());
                }
                else if (parallelClassMatcher.matches()) {
                    lectureMatcher =
LECTURE_PATTERN.matcher(currentLine);
                    exerciseMatcher =
CVIKO_PATTERN.matcher(currentLine);
                    if (lectureMatcher.matches()) {
        subjectHappeningDAO.insertIntoDatabase(SubjectHappening.newSubjectHappenin
g()
.type(TypeOfClass.LECTURE)
.day(DayEnum.getByNumericalRepresentation(lectureMatcher.group(3))
                                        .build(),
                                        subject,
                                        lectureMatcher.group(4),
                                        lectureMatcher.group(2)
                                                );
                    } else if (exerciseMatcher.matches()) {
subjectHappeningDAO.insertIntoDatabase(SubjectHappening.newSubjectHappening()
                                        .type(TypeOfClass.PARALLEL_CLASS)
 .day(DayEnum.getByNumericalRepresentation(exerciseMatcher.group(3)))
        .numberOfParallelClass(Integer.parseInt(exerciseMatcher.group(5)))
                                        .build(),
                subject,
                exerciseMatcher.group(4),
                exerciseMatcher.group(2)
```

```
                );
                        }
                    }
                }
            }
```

TeacherSectionDTO

Posledná implementácia DTOInterface rozhrania predstavuje spracovanie údajov o učiteľoch. Táto trieda obsahuje len jeden regulárny výraz, ktorý vyzerá nasledovne:
private static final Pattern PATTERN = Pattern.compile("([a-zA-Z]+)_([a-zA-Z]*)_?([a-zA-Z]*)\\s([a-zA-Z]+)\\s([a-zA-Z]+)\\s:\\s[A-Z]")

Zdrojový kód je v štandardnej forme, keďže táto sekcia nie je náročná na logiku a teda generická transformLinesOfSectionAndInsertThem metóda prechádza každým riadkom tejto sekcie a posiela ho do privátnej metódy s názvom transformIntoTeacher.

V tejto implementácií sa stretávame so štandardnou definíciou premenných na začiatku metódy a taktiež s nám už známym vyhodením výnimky PatternDoesntMatchException. Následne pomocou switch klauzuly porovnávame počet skupín, ktoré obsahuje náš matcher. V prípade, že ide o štyri skupiny má učiteľ uvedené iba priezvisko a preto pomocou builder patternu inštancie triedy Teacher vytvoríme inštanciu Teacher, ktorá neobsahuje ani stredné meno, ani krstné meno učiteľa. Pokiaľ je počet skupín matchera rovný piatim učiteľ má definované krstné meno, ale aj priezvisko a teda vytvoríme inštanciu s týmito parametrami. V poslednom prípade môže mať náš matcher šesť skupín, kde by učiteľ mal definované krstné meno, stredné meno, ale aj priezvisko. Vytvoríme inštanciu s tromi prvkami mena. V každom z týchto prípadov ešte práve vytvorenú inštanciu učiteľ pošleme do metódy cleanUpTeacher, ktorá nahradí prázdne znaky v mene za null, aby prípadné selekty v databáze našli učiteľa. Táto ochrana je dôležitá vzhľadom na chybovosť súboru, ktorý spracovávame. Ten totiž vo viacerých prípadoch obsahoval namiesto krstného mena biely znak, prípadne znak, ktorý nezodpovedá krstnému menu. Takto už vyčistenú inštanciu pošleme do TeacherDAO, kde prebehne inzercia do databázy.

InsertDAOInterface

Implementácie tohto rozhrania pre nás predstavujú časť zdrojového kódu, ktorá je zodpovedná za prácu s databázou a to vrátane vkladania a čítania, prípadne aktualizovania dát v nej. V našej aplikácii pristupujeme k databáze dvoma rôznymi spôsobmi, jedným z nich je klasická cesta pomocou SQL príkazov, ktoré sme ručne napísali. Druhým spôsobom je využitie CRUD repozitárov, ktoré sú súčasťou frameworku spring. Tieto repozitáre mapujú jednotlivé stĺpce v databáze na java objekty, prípadne opačne. S ich pomocou teda nemusíme písať jednoduché SQL príkazy a tým ušetríme aplikáciu od kódu, ktorý nemá dodatočnú hodnotu. Tieto repozitáre umožňujú aj definovanie komplikovanejších SQL príkazov, ale pre takýto druh príkazov sme sa rozhodli využiť klasické SQL príkazy a JdbcTemplate objekt. Pusťme sa teda do popisu jednotlivých implementácií tohto rozhrania. Z dôvodu jednoduchosti preskočíme niektoré implementácie, ktoré pozostávajú len z jednoduchých volaní príslušného CRUD repozitára a uložia java objekt do databázy. Takýmito implementáciami sú RoomDAO a SchoolHourDAO.

DepartmentDAO

Operácie tejto implementácie sú vykonávané výlučne pomocou FacultyRepository a DepartmentRepository, ktoré sú CRUD repozitármi. Pre nás sú zaujímave dve metódy. Jednou z nich je insertIntoDatabase, s ktorou sa budeme stretávať častejšie, vzhľadom na to, že ju používame ako štandard pre vkladanie Java objektu do databázy.

Druhá metóda je o trochu komplikovanejšia, vzhľadom na fakt, že aktualizuje ID katedry za pomoci skratky fakulty a skratky katedry. Vykonáva to pomocou nájdenia fakulty v databáze a priradenia identifikačného čísla tejto fakulty ku katedre, ktorú našlo pomocou skratky. Následne nájdený objekt katedry uloží v databáze, čo je vzhľadom na fakt, že tento objekt sa už v databáze nachádza interpretované ako update.

FacultyDAO

Vzhľadom na fakt, že objekt, ktorý tu vytvoríme, prípadne aktualizujeme bude využitý v data transform objekte, ktorý sme už popísali má táto metóda návratový typ Faculty. Práve tento návratový typ je priradený premennej retVal, čo je skrátene return value a to tak, že sa najskôr pokúsi o nájdenie objektu, ktorý sa snažíme uložiť v databáze, pokiaľ takýto objekt v databáze nenájde uloží ho a vráti túto uloženú hodnotu.

FieldOfStudyDAO

Nájdeme tu na pochopenie jednoduchý zdrojový kód, kde sa pokúsime nájsť fakulty v databáze podľa mena fakulty. Robíme tak z dôvodu, že každý odbor je súčasťou nejakej fakulty a tento fakt chceme reprezentovať aj v našej databázovej schéme. Žiaľ existuje istá možnosť, že náš riadok na ktorom bol náš odbor definovaný mal zle chybne definovanú fakultu a z tohto dôvodu pomocou if zachytávame tento fakt a logujeme práve nastanú chybu. V prípade, že sa chyba nestala nastavíme inštancii triedy FielOfStudy, ktorú sa chystáme vložiť do databázy ID fakulty a tento objekt uložíme v databáze.

SubjectDAO

Subject data access object je jednou z komplikovanejších implementácií a pozostáva z jednej privátnej metódy určenej na nájdenie učiteľa a jednej verejnej metódy, ktorá je určená pre vkladanie inštancií triedy Subject do databázy. Privátna metóda robí veľmi jednoduchú vec. Hľadá v databáze učiteľa na základe celého mena učiteľa. Komplikáciou je fakt, že musíme zistiť z koľkých prvkov pozostáva plné meno učiteľa a to je dôvod implementácie tejto metódy.

Ako sme spomínali prebieha tu zisťovanie počtu prvkov celého mena učiteľa a to jednoduchým spôsobom. Do poľa reťazcov priradíme hodnotu učiteľovho plného mena rozdeleného pomocou znaku medzery. Následne odstránime prebytočné medzery a prázdne znaky a dostanem zoznam reťazcov, na základe ktorého dĺžky určíme aké parametre máme poslať do TeacherRepository. Ak sa stane, že náš zoznam má neočakávanú dĺžku tento fakt zalogujeme na úrovni chyby. V prípade, že počas hľadania učiteľa v databáze dôjde k chybe zachytíme túto skutočnosť a taktiež ju zalogujeme. Po tom ako sme sa pokúsili nájsť učiteľa postúpime k popisu metódy, ktorá ma za úlohu vložiť naše dáta do databázy.

Prvým krokom je využitie už spomínanej privátnej metódy na nájdenie učiteľa. Následne sa pokúsime nájsť v databáze odbor pre ktorý je tento predmet vyučovaný, ale to len v prípade, že na riadku, ktorý sme čítali bol odbor definovaný. Z tohto dôvodu tu môžeme pozorovať využitie ternárneho operátora, ktorý v prípade, že odbor nebol definovaný rovno priradí hodnotu null a nehľadá ho v databáze. Nasleduje časť kde vyhodnotíme, či sa daný odbor našiel, v prípade, že áno priradíme objektu, ktorý ideme vkladať ID tohto odboru. V opačnom prípade ešte vyhodnotíme, či bol odbor zadaný a nebol prázdny reťazec znakov. Ak sa tak stalo zalogujeme chybu. Následne vyhodnocujeme, či sme boli úspešní pri hľadaní učiteľa. Ak áno jednoducho priradíme ID

učiteľa nami vkladanému objektu. Inak zalogujeme chybu podľa toho o akú chybu ide. Možnosti sú, že učiteľ nebol definovaný, alebo nebol len nájdený v databáze a teda nebol definovaný v sekcii určenej pre učiteľov. Po vykonaní týchto krokov vložíme objekt do databázy pomocou SubjectRepository.

SubjectHappeningDAO

Metóda insertIntoDatabse, ktorá sa tu nachádza opäť zohráva kľúčovú a jedinú dôležitú úlohu, ktorá nás zaujíma. Na začiatku tejto metódy sa vyhodnotíme či objekt typu Subject, ktorý sme dostali ako parameter má definovaný odbor. Podľa toho nájdeme tento predmet v databáze nakoľko potrebujeme identifikačné číslo, ktoré nebolo súčasťou parametra. Následne nájdeme databázovú reprezentáciu školskej hodiny počas ktorej nami vkladaný objekt prebieha a taktiež miestnosť v ktorej sa má vykonávať. Vzhľadom na to, že mohli nastať situácie, kedy nebol predmet ktorý sme hľadali v databáze nájdený, alebo sme nenašli hodinu počas ktorej má výučba prebiehať, prípadne nebola špecifikovaná miestnosť tejto výučby musíme tieto situácie zalogovať pre prípadné odstraňovanie chýb, kde táto situácia nemala nastať. Pokiaľ jednotlivé situácie nenastali priradíme zodpovedajúce identifikačné čísla a pokúsime sa o vloženie objektu do databázy, ktoré obalíme try-catch blokom pre prípad, že by sa nepodarilo výučbu do databázy vložiť. Prípadnú chybu zalogujeme, keďže výučba v rozvrhu mala byť avšak došlo k chybe, ktorá nemá ovplyvniť ostatné predmety.

TeacherDAO

Poslednou pre nás zaujímavou implementáciou je práve TeacherDAO, kde prebieha vkladanie učiteľa do našej databázy. Táto implementácia je relatívne jednoduchá a pusťme sa do jej opisu.

Prvotne tu prebieha hľadanie katedry pod ktorou daný učiteľ funguje. Ak sa podľa skratky katedry táto katedra v databáze nenachádza zalogujeme chybu. V opačnom prípade priradíme učiteľovi ID tejto katedry a učiteľa vložíme do databázy pomocou TeacherRepository.

1.1.6. Zobrazovanie rozvrhu podľa zadaných požiadaviek

Keďže v predchádzajúcich častiach sme už opísali ako prebieha výber súboru s dátami, parsovanie dát a ich vkladanie do databázy môžeme predpokladať, že tieto dáta už sú v databáze a teda aplikácia je pripravená na poskytovanie týchto dát študentom, prípadne iným stranám, ktoré sa chcú informovať o rozvrhu pre daný krúžok.

Prvým krokom je teda využitie front end časti, ktorú naša aplikácia neposkytuje a bola už vytvorená v aplikácii EubaRozvrh. Alternatívou je využitie vlastného API, prípadne nástrojov na zasielanie requestov určených. Tieto requesty naša aplikácia prijíma a sú spracovávané pomocou ScheduleController-a, ktorý je načase popísať.

Začiatkom je teda definovanie adresy na ktorej controller pracuje. V tomto prípade je to základná URL adresa, bez akýchkoľvek prípon. Nasleduje anotácia, ktorá popisuje v akom tvare dostane požadovateľ requestu odpoveď - application/json. Ako bolo už skôr povedané rozhodli sme sa pre formát JSON, ktorý je považovaný za dostatočné dobre štuktúrovaný a zároveň jednoduchý. Nasleduje definícia metódy controllera, kde je evidentné, že vracia zoznam tried typu Schedule, ktorá pozostáva zo všetkých údajov potrebných pre osobu, ktorá požaduje informácie o výučbe. Medzi tieto údaje patrí meno učiteľa, miestnosť výučby, typ vyučovanej hodiny, deň, kedy výučba prebieha, čas výučby, meno predmetu a taktiež skratka predmetu. Táto metóda požaduje niekoľko parametrov, ktoré sú definované pomocu anotácie RequestParam, kde hodnota value predstavuje meno parametra. Za každou z týchto anotácií je meno parametra, ktorý bude poslaný do ScheduleService triedy, ktorú hneď predstavíme a ktorej volanie je jediným riadkom tela metódy getSchedule tohto controllera.

ScheduleService a ScheduleDAO

Vzhľadom na to, že ScheduleService obsahuje len jediný riadok, ktorý volá ScheduleDAO nebudeme uvádzať jej zdrojový kód a rovno predstavíme ScheduleDAO, ktorá je o niečo zaujímavejšia. Tá totiž pozostáva z manuálne napísaných SQL príkazov, RowMapper triedy, ktorá slúži na manuálne mapovanie stĺpcov databázovej tabuľky na java objekt, privátnej metódy, ktorá zloží stĺpce tabuľky učiteľ do celého mena učiteľa a finálne z public metódy, ktorá volá ďalšie privátne metódy, ktoré hneď popíšeme. Avšak pozrime sa najskôr na to, ako vyzerajú jednotlivé SQL príkazy, ktoré tu sú použité. Pre predstavu uvedieme len jeden z týchto SQL príkazov, ktorý slúži na nájdenie cvičení.

```
private String SELECT_MANDATORY = "SELECT s.name, s.short_name,
s.mandatory, sh.type, t.*, r.name as room_name, h.*, sh.day "
        + "FROM subject s "
        + "JOIN field_of_study fos ON s.field_of_study_id = fos.id "
        + "JOIN faculty f ON fos.faculty_id = f.id "
        + "JOIN subject_happening sh ON sh.subject_id = s.id "
        + "JOIN teacher t ON s.teacher_id = t.id "
        + "JOIN room r ON sh.room_id = r.id "
        + "JOIN school_hour h ON sh.hour_id = h.id "
        + "WHERE s.year_of_study = ? "
        + "AND f.name = ? "
        + "AND fos.short_name = ? "
        + "AND sh.number_of_parallel_class = ?"
        + "AND s.mandatory = 1";
```

Sú tu použité tri rôzne podobné SQL príkazy, kde jeden z nich je určený na nájdenie povinných častí výučby, ktoré majú definovaný krúžok, čo pre nás predstavuje cvičenia povinných predmetov, druhý je určený na nájdenie nepovinných predmetov a teda všetkých prednášok a cvičení určených pre nami definovaný ročník, ktoré nie sú povinné pre absolvovanie tohto ročníka. Tretí SQL príkaz je používaný na nájdenie všetkých prednášok, ktoré sú spojené s predmetmi v danom ročníku, ktoré musí študent tohto ročníka absolvovať. Pristúpime k popisu metódy, ktorá tieto SQL príkazy používa.

Na začiatku definujeme zoznam prvkov typu Schedule, ktorý zostane najskôr prázdny. Do tohto zoznamu pridáme všetky prednášky, ktoré sú špecifické pre predmety, ktoré študent musí absolvovať. Následne do zoznamu pridáme všetky cvičenia, ktoré k týmto prednáškam prislúchajú a v prípade, že v požiadavke bolo špecifikované, že chcem zobraziť aj nepovinné predmety, učiníme tak pridaním prvkov, ktoré sme dostali po zavolaní metódy getNonMandatory s príslušnými parametrami.

Záver

Cieľom tejto diplomovej práce bolo vytvoriť back end časť aplikácie na zobrazovanie rozvrhu pre študentov, prípadne iné osoby, ktoré o tieto informácie majú záujem.

Pre splnenie tohto cieľa bolo potrebné vykonať analýzu vstupných dát obsiahnutých v súbore, ktorý je vytvorený osobou na to zodpovednou a obsahuje všetky údaje o rozvrhu a dátach na jeho zobrazenie potrebných.

Ďalším krokom je návrh UI, ktoré bude osoba zodpovedná za nahrávanie dát do databázy využívať. Taktiež bolo potrebné navrhnúť databázový model, ktorý reprezentuje vstupné údaje v databázovom systéme a neposledným krokom je návrh implementácie aplikácie, ktorá bola vyvíjaná.

Záverečným krokom vývoja aplikácie bola implementácia podľa vykonanej analýzy a návrhu, tak, aby spĺňal všetky požiadavky

Domnievame sa, že splnením všetkých cieľov sme splnili aj hlavný cieľ tejto diplomovej práce. Jej prínos je podľa nášho názoru efektívnejšie a spoľahlivejšie zobrazovanie dát rozvrhu, jednoduchšie udržiavanie aplikácie osobou, ktorá na to bude určená a celkové vylepšenie prehľadnosti zdrojové kódu oproti jeho predchádzajúcim verziám.

Zoznam použitej literatúry

Ceruzzi, Paul E. (2000). A History of Modern Computing. Cambridge, Mass.: MIT Press. ISBN 0-262-03255-4,

Ulrich, William. "Application Package Software: The Promise Vs. Reality". Cutter Consortium,

Valums, Andrew (2010-02-10). "Web apps vs desktop apps". valums.com. Retrieved 2013-07-14,

Alex Chaffee (2000-08-17). "What is a web application (or "webapp")?". Retrieved 2008-07-27,

Riehle, Dirk (2000), Framework Design: A Role Modeling Approach (PDF), Swiss Federal Institute of Technology,

Hill, C; DeLuca, C; Balaji, V; Suarez, M; da Silva, A (2004), "Architecture of the Earth System Modeling Framework (ESMF)", Computing in Science and Engineering: 18–28,

Gachet, A (2003), "Software Frameworks for Developing Decision Support Systems – A New Component in the Classification of DSS Development Tools", Journal of Decision Systems, 12 (3): 271–281,

Binstock, Andrew (20 May 2015). "Java's 20 Years Of Innovation". Forbes. Retrieved 18 March 2016,

Gosling, James; McGilton, Henry (May 1996). "The Java Language Environment".Gosling, James; Joy, Bill; Steele, Guy; Bracha, Gilad. "The Java Language Specification, 2nd Edition",

Alexander, Christopher; Ishikawa, Sara; Silverstein, Murray; Jacobson, Max; Fiksdahl-King, Ingrid; Angel, Shlomo (1977). A Pattern Language: Towns, Buildings, Construction. New York: Oxford University Press. ISBN 978-0-19-501919-3,

Alur, Deepak; Crupi, John; Malks, Dan (May 2003). Core J2EE Patterns: Best Practices and Design Strategies (2nd Edition). Prentice Hall. ISBN 0-13-142246-4,

Fowler, Martin (1997). Analysis Patterns: Reusable Object Models. Addison-Wesley. ISBN 0-201-89542-0,

Fowler, Martin (2003). Patterns of Enterprise Application Architecture. Addison-Wesley. ISBN 978-0-321-12742-6,

Eloranta, Veli-Pekka; Koskinen, Johannes; Leppänen, Marko; Reijonen, Ville (2014). Designing Distributed Control Systems: A Pattern Language Approach. Wiley,

Internetové zdroje

Margaret Rouse, Application [online]. Dostupné na internete: http://searchsoftwarequality.techtarget.com/definition/application

Margaret Rouse, Web application [online]. Dostupné na internete: http://searchsoftwarequality.techtarget.com/definition/Web-application-Web-app

Margaret Rouse, Parser [online]. Dostupné na internete: http://searchmicroservices.techtarget.com/definition/parser

Filip Filip, Grafické rozhranie (GUI) [online]. Dostupné na internete:

http://www.vzlet.rs/graficke-rozhranie-gui/

Viliam Búr, Objektovo orientované programovanie [online]. Dostupné na internete: http://bur.sk/sk/2012/java-oop

Peter Láng, Co je to framework [online]. Dostupné na internete: http://langi.cz/webarna/co-je-to-framework https://www.quora.com/What-exactly-is-a-software-framework

Spring Framework Reference Documentation [online] . Dostupné na internete:

https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#overview