Jihočeská univerzita v Českých Budějovicích

Přírodovědecká fakulta

Využití jazyka Java Enterprise Edition a navazujících technologií při vývoji webové Quiz/Review aplikace

Bakalářská práce

Daniel Perník

Vedoucí práce: Ing. František Drdák, CSc.

České Budějovice 2015

Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

ZADÁVACÍ PROTOKOL BAKALÁŘSKÉ PRÁCE

Student: **Daniel Perník**
(jméno, příjmení, tituly)

Obor – zaměření studia: Aplikovaná informatika

Katedra: Ústav aplikované informatiky

Školitel: ing. František Drdák, CSc.
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Garant z PřF: .....
(jméno, příjmení, tituly, katedra – jen v případě externího školitele)

Školitel – specialista, konzultant: .....
(jméno, příjmení, tituly, u externího š. název a adresa pracoviště, telefon, fax, e-mail)

Téma bakalářské práce: **Využití jazyka Java Enterprise Edition a navazujících technologií při vývoji webové Quiz/Review aplikace.**

Cíle práce:

1. Navrhnout základní funkcionalitu webové aplikace typu Quiz/Review pro tvorbu a provádění znalostních testů.
2. Při výběru vhodných implementačních technologií a nástrojů vývojové platformy JEE se zaměřit zejména na použití technologií JSF, EJB, PrimeFaces, JPA a Apache Maven.
3. Realizovat vývojový proces jádra aplikace včetně základního otestování.
4. Shrnout nabyté zkušenosti s použitými technologiemi.

Základní doporučená literatura:

1. http://docs.oracle.com/javaee/7/tutorial/doc/
2. http://courses.coreservlets.com/Course-Materials/

Financování práce :.....

Vedoucí práce : ing. František Drdák, CSc podpis : .....

U externích vedoucích fakultní garant práce.....podpis : .....

Garant oboru bak.. studia (nepožaduje se u zaměření „příprava na mag. studium biologie)

..... podpis : .....

Vedoucí katedry RNDr. Libor Dostálek (v z. JMN PŘF) podpis .....

Případný souhlas vedoucího ústavu AV .....podpis : .....

V Českých Budějovicích dne 15. 9. 2014

Převzal/a dne..... 17. 9. 2014 ..... podpis : .....

Bibliografické údaje

Perník D., 2015: Využití jazyka Java Enterprise Edition a navazujících technologií při vývoji webové Quiz/Review aplikace.
[Utilization of Java Enterprise Edition and related technologies in development of a web Quiz/Review application. Bc. Thesis, in Czech.] – 40 p., Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Tato bakalářská práce se zabývá využitím moderních JEE technologií při vývoji webové aplikace, která poskytuje základní funkcionalitu na tvorbu a provádění uživatelských testů typu Quiz/Review, včetně vyhodnocování získaných výsledků. Další náplní práce je zhodnocení použitých technologií, při realizaci projektu takového typu.

In english

This bachelor thesis deals with implementation of modern JEE technologies in the development of Web application, whose goal is to enable application users to browse quizzes and then get evaluation of their process. The purpose of the work is a Quiz/Review web application with basic functionality and evaluation of the technologies, used in the development project of this type.

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích, dne 22. dubna 2015

Podpis: _____

Poděkování

Rád bych poděkoval panu Ing. Františku Drdákovi, CSc. za ochotu a odborné konzultace. Dále bych také rád poděkoval své rodině, přítelkyni, přátelům a kamarádům z práce za podporu a pevné nervy.

Obsah

1. Úvod

Po přijetí webu širokou veřejností zaznamenalo toto prostředí velice rychlý a rozsáhlý vývoj. Neplní již jen původní stroze informační poslání ve statickém pojetí, ale poskytuje také prostor pro interaktivní využití v mnoha dalších oblastech jako je podpora marketingu a obchodu, podpora sociálních sítí, ale i podpora vzdělávání a tak dále.

Důsledkem tohoto rozmachu je obrovské množství požadavků na vývoj webových aplikací, který s sebou přinesl také potřebu vyvíjet takové aplikace a systémy rychle, efektivně (a tím i levně) s možností jednoduché, centralizované údržby, možností škálovatelnosti a dalšími přínosnými vlastnostmi, ze kterých profitují jak uživatelé, tak samotní vývojáři těchto aplikací.

Z toho důvodu vznikla a dále vzniká celá řada technologií a frameworků, které se zmíněnou problematikou zabývají. Tato práce se věnuje platformě Java Enterprise Edition a navazujícím technologiím, které se v současné době těší velké popularitě.

Aby bylo možné ověřit a prozkoumat silné stránky této platformy, mělo být navrženo a implementováno jádro aplikace sloužící k procházení znalostními testy, které následně poskytne svým uživatelům evaluaci jejich postupu.

Výše zmíněný požadavek na implementaci byl v rámci vývoje rozšířen o uživatelské prostředí, čímž vznikla odlehčená testovací aplikace, která svým uživatelům kromě požadované funkcionality nabízí navíc ucelené uživatelsky přívětivé prostředí. Toto prostředí rozlišuje role přihlášených uživatelů, kterým je pak podle role umožněna jim určená funkcionalita.

Téma programované aplikace bylo inspirováno popularitou a rozšířením e-learningu samotného, nikoliv však konkrétní existující aplikací.

1.1. Motivace

Ověřit vlastnosti, které dělají z Javy EE jednu z nejrozsáhlejších technologií na trhu v oblasti server-side programovacích jazyků podnikových aplikací [1]. Tento proces vychází z nastudování v praxi tolik rozšířené platformy, relevantních technologií a komponent, což je navíc velice přínosné pro budoucí uplatnění se na trhu práce v oboru vývoje software.

Popularita e-learningu neustále roste a z průzkumů vyplývá jasněji než dříve, že bude tento trend i nadále pokračovat. Potřebu či význam testování a ověřování lidských vědomostí nalezneme ve velkém množství firem a institucí. Protože se největší pozornosti této problematice dostává až v posledních letech, stojí za to se jí zabývat [2]. Z odhadů pro rok 2011 vyplývá, že bylo po celém světě vynaloženo asi 35.6 miliard dolarů do oblasti e-learningu. V roce 2014 se tato hodnota zvýšila na 56.2 miliard a podle odhadů se v roce 2015 zdvojnásobí [2]. Takto rostoucí trend je velkou motivací pro zájem o tuto oblast.

Vytvořit aplikaci, která bude kromě výsledné funkcionality také charakteristická čitelností zdrojového kódu a přehledností adresářové struktury. Tento celek bude poté poskytnut jako vzorový příklad pro výuku kurzu Java EE.

1.2. Cíle

- Navrhnout základní funkcionalitu jádra webové quiz/review aplikace
- Při implementaci se zaměřit na platformu Java EE, zejména pak na JSF, EJB, JPA, PrimeFaces a Apache Maven
- Realizovat vývojový proces jádra aplikace včetně základního otestování
- Shrnout zkušenosti nabyté při vývoji této aplikace

1.3. Struktura

Tato práce se zabývá návrhem a implementací jádra webové aplikace určené k vědomostnímu testování uživatelů. Při implementaci je kladen důraz na využití zvolených technologií založených na platformě Java Enterprise Edition verze 7.

Úvodní kapitola stručně uvádí čtenáře do problematiky, předkládá motivaci, která vedla autora ke zvolení tohoto tématu a rekapituluje cíle této práce.

Druhá kapitola je analytická. Popisuje přístup k řešení této problematiky z pohledu metodiky vývoje, současný stav řešené problematiky, požadavky týkající se implementace a analýzu požadované funkcionality aplikace, která je zde navíc znázorněná formou UML diagramu. V této kapitole je navíc zmíněn a zhodnocen alternativní populární přístup k vývoji Java EE aplikací.

Další kapitolou je kapitola technologická. Jejím obsahem je přehled a rozbor prostředí, architektury, vybraných technologií platformy Java EE a technologií třetích stran, které byly využity při vývoji této aplikace. Cílem této kapitoly není popsat technologie do posledního detailu, nýbrž představit klíčové vlastnosti, které hrají roli v implementační části práce.

Implementační část popisuje jednotlivé úseky vývoje. Vedle obecnějšího popisu jsou zde také zmíněné konkrétní záležitosti, které jsou v kontextu vyvíjené aplikace důležité. Kapitola je zakončena informacemi o testování.

Poslední kapitolou je závěr. V závěru jsou v bodech shrnuty a zhodnoceny splněné cíle práce. Po tomto souhrnu následuje podkapitola týkající se možných směrů rozšiřování naprogramované aplikace.

V přílohách práce se nachází instalační a uživatelská příručka. Uživatelská je poněkud obsáhlejší a představuje čtenáři formou obrázků grafické rozhraní aplikace.

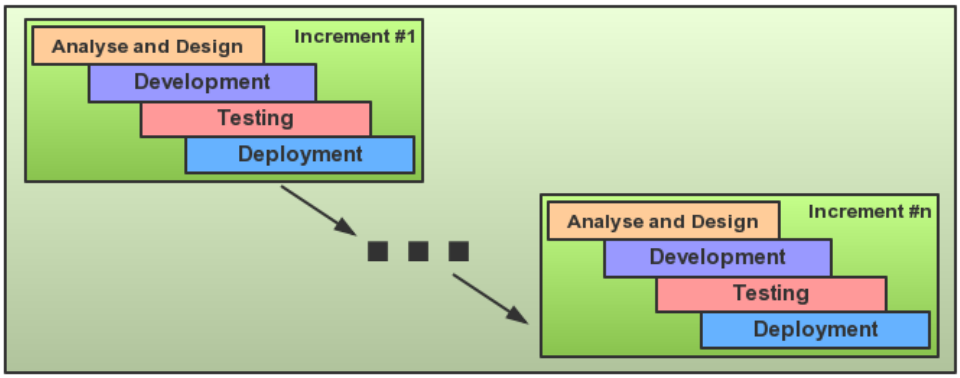
2. Analytická část

2.1. Metodika vývoje

Dodržování ověřené metodiky při vývoji přináší celou řadu výhod. V této podkapitole nejsou rozebírány globální výhody a nevýhody určitých modelů, nýbrž konkrétní přístup využitý při vývoji tohoto projektu.

Přestože byla základní funkcionalita vyvíjené aplikace stanovena již v počátcích vývoje, autor měl v úmyslu naprogramovat ucelenější rozhraní než jen samotné jádro. Původním plánem bylo využití vývojového modelu *Waterfall*. Ten však počítá s kompletní analýzou a designem již před zahájením vývoje. Po analýze modelů a metodik byla za nejvhodnější nakonec shledána metodika s inkrementálním (přírůstkovým) přístupem - **Incremental build model**.

Tento přístup kombinuje sekvenční a iterační přístupy. Projekt je rozdělen na menší podproblémy a tím je umožněno zavádět změny či nové požadavky během vývoje. Lze ho popsat jako sekvenci jednotlivých vodopádů, kde každý vodopád přísluší určitému podproblému (inkrementu). Na konci každého vodopádu vzniká nový prototyp softwaru. Tato sekvence končí, pokud je výsledný program v požadovaném stavu. Diagram modelu znázorňuje obrázek 1.



Obrázek 1: Znázornění využitého inkrementálního modelu

2.2. Požadavky týkající se implementace a současný stav problematiky

2.2.1. Současný stav zvolené platformy

Programovacích jazyků používaných pro programování na straně serveru je celá řada. Java v současné době patří v oblasti rozsáhlých enterprise aplikací k těm nejčastěji se vyskytujícím [3]. Tato skutečnost je patrná z níže umístěného obrázku, kterým je tabulka užitých technologií na back-endu úspěšných webových projektů.

Websites	ASP.NET	C	C++	Go	Java	JavaScript	PHP	Python	Ruby on Rails
Google	No	Yes	Yes	Yes	Yes	No	No	Yes	No
YouTube	No	Yes	Yes	Yes	Yes	No	No	Yes	No
Facebook	No	No	Yes	No	Yes	No	Yes	Yes	No
Yahoo	No	No	No	No	No	Yes	Yes	No	No
Amazon	No	No	Yes	No	Yes	No	No	No	No
Wikipedia	No	No	No	No	No	No	Yes	No	No
Twitter	No	No	Yes	No	Yes	No	No	No	Yes
Bing	Yes	No	No	No	No	No	No	No	No
eBay	No	No	No	No	Yes	Yes	No	No	No
MSN	Yes	No	No	No	No	No	No	No	No
Microsoft									
Linkedin	No	No	No	No	Yes	Yes	No	No	No
Pinterest									
Ask									
Wordpress	No	No	No	No	No	No	Yes	No	No

Obrázek 2: Přehled back-end technologií vybraných projektů. [3]

Jednou z výhod Javy je fakt, že pro ni existuje celá řada technologií, frameworků a komponent poskytovaných třetími stranami. Názory na jejich provedení jsou velice subjektivní a mnohdy se výrazně liší. Aby však výsledná aplikace nabídla svým uživatelům maximum, je zapotřebí mít o těchto technologiích povědomí. Mohou vývojáři usnadnit práci formou připravených komponent nebo nabídnout inovativní funkcionalitu.

Rodina Java EE je velmi rozsáhlá a obsahuje technologie, které se mohou v jistých záležitostech překrývat. Příkladem jsou JSP (Java Server Pages) a JSF (Java Server Faces) užité pro prezentační vrstvu aplikací. Je zapotřebí prozkoumat tyto technologie a zvolit tu, která svými rysy nejvíce vyhovuje konkrétnímu řešení.

2.2.2. Požadavky na implementaci

Při tvorbě zadání práce byl stanoven výběr technologií, na které má být kladen důraz a kterými se práce bude zabývat. To bylo naplněno následující selekcí:

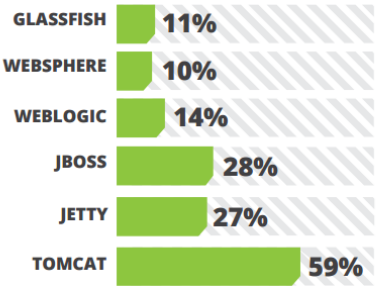
- **EJB** serverové komponenty využity jako objekty zajišťující přístup k datům z databáze
- **JPA** framework zajišťující objektově relační mapování

  (Jde pouze o specifikaci. Implementací je v tomto případě **Hibernate**.)
- **JSF** a framework **PrimeFaces** použity pro prezentační vrstvu aplikace
- **Apache Maven** použit pro řízení buildů a závislostí aplikace

Další technologie, které byly během vývoje shledány jako užitečné pro řešení dané problematiky:

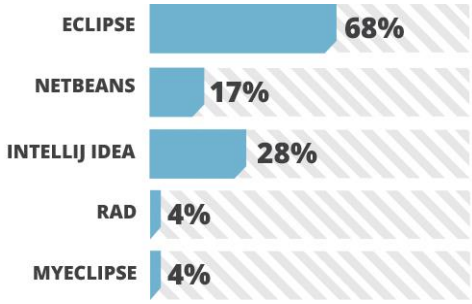
- **Apache Shiro** security framework pro řízení autentizace a autorizace
- **SLF4J** framework sloužící k vypisování logů v deployment čase

Jako aplikační server byl nejprve použit **Glassfish 4**. Při vývoji je nutné server často restartovat, aby proběhl tzv. *redeploy aplikace*. Takový restart Glassfishe trval přibližně 15 – 20 sec. Pro velkou časovou náročnost byla zahájena analýza aplikačních serverů. Podle průzkumu [4] je nejoblíbenějším serverem Tomcat. Ten však obsahuje pouze částečnou implementaci a není tedy zcela Java EE kompatibilní. Proto byl nakonec zvolen aplikační server **JBoss WildFly 8** snižující dobu na průměrných 10 sec.



Obrázek 3: Přehled popularity open-source AS. [4]

Výběr vývojového prostředí pro tento projekt nebyl zpočátku jasný. V úvahu připadaly dvě IDE a to konkrétně NetBeans a Eclipse. Průzkumem bylo zjištěno, že větší pozornosti vývojářů se dostává projektu **Eclipse** [5] a proto dostalo nakonec přednost.



Obrázek 4: Přehled popularity vývojových prostředí. [4]

Tato rozhodnutí nebyla však učiněna pouze na základě názoru převzatého z jednoho zdroje. Součástí průzkumu byla například analýza vykonaná pomocí webového nástroje Google Trends, kde byla porovnávána popularita jednotlivých klíčových slov v rozhraní světového vyhledavače.

Průzkum provedený na serverech *jobs.cz, práce.cz* a *indeed.com* ukázal, že při vyhledání poptávky po Java developerech pro práci na současných projektech v požadovaném technologickém sloupci převládaly právě ty, kterými se zabývá tato práce. Významnou četně zmiňovanou problematikou, kterou se však tato práce nezabývá, je *Spring framework.*

2.2.3. Odlišný přístup k řešení

V rámci průzkumu byly odhaleny dva oblíbené přístupy k vývoji Java-based webových aplikací. Prvním je implementace čistě pomocí technologií Java EE, kterou se řídí tato práce. Druhým je implementace s využitím **Spring frameworku.** Spring vznikl z důvodu usnadnění vývoje JEE aplikací, jenž byl velmi náročný. Dle autorova názoru je to spíše názor minulosti a vývoj těchto aplikací v Javě EE je od její sedmé verze dostatečně intuitivní. Nicméně stále platí skutečnost, že můžeme Spring vnímat jako jakousi nadstavbu. Ve většině projektů se využívá právě jeden z přístupů a bok po boku se s nimi proto většinou nesetkáme.

2.3. Analýza funkcionality aplikace

Za požadovanou základní funkcionalitu jádra této aplikace bylo označeno následující:

- **Vytváření kvízů**
- **Absolvování kvízů**
- **Evaluování kvízů**

Tato funkcionalita byla však v rámci vývoje navíc doplněna o řadu dalších záležitostí, počínaje nutností přihlášení / registrace uživatele před vstupem do rozhraní aplikace. Na základě autentizace jsou uživatelé separováni do dvou rolí, které nesou označení *student* a *mentor*.

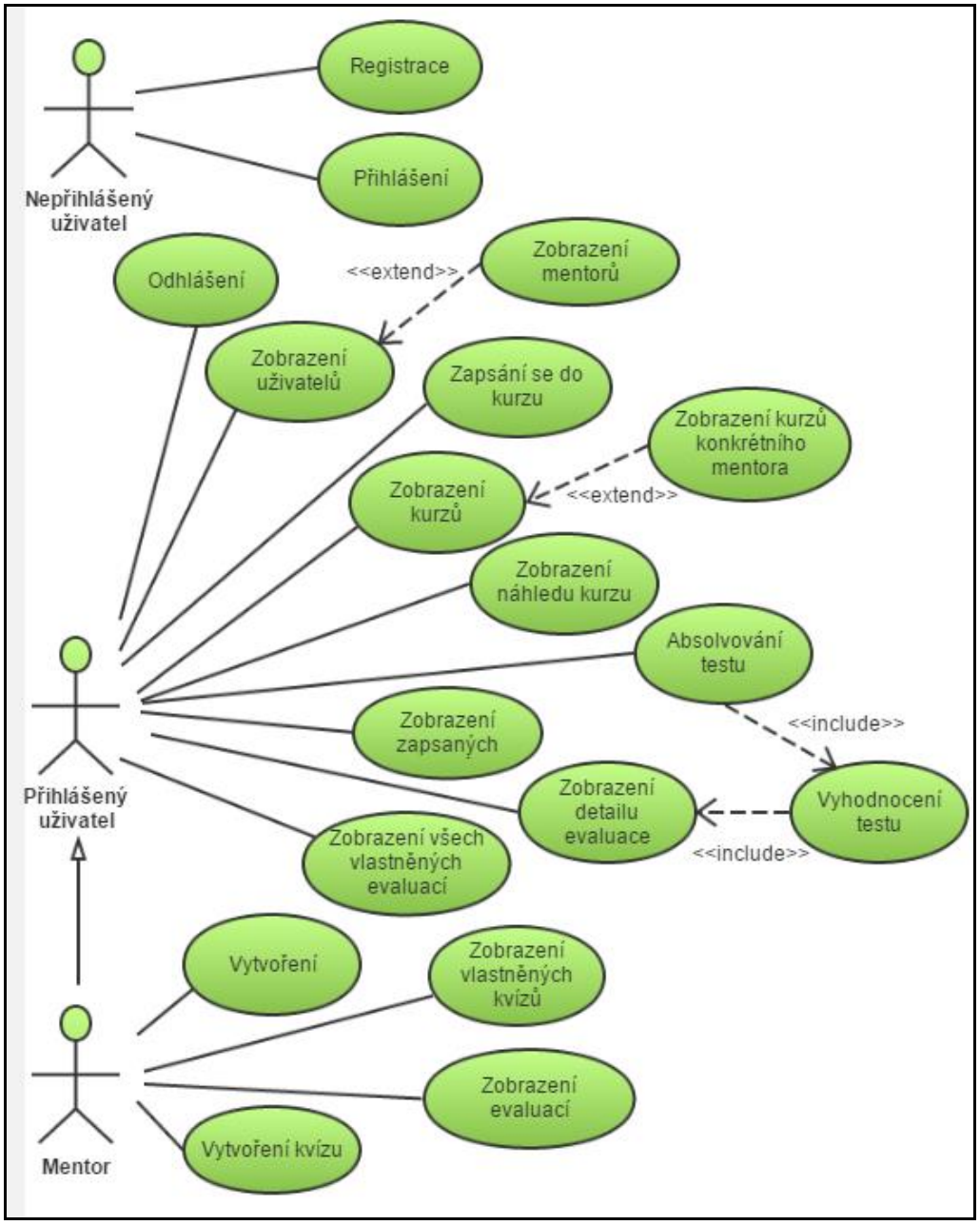
Pro lepší přehlednost obsahu a možnost třídění dle zájmu uživatele byl vytvořen objekt *kurz*. Ten zahrnuje určité informace o sobě samém a dále pak slouží jako struktura zastřešující vybrané kvízy. Uživatel se do těchto kurzů zapisuje a tím získává možnost přistupovat k jednotlivým kvízům.

Evaluace, které jsou uživateli generovány po absolvování kvízu, jsou uchovávány v databázi a to i s odkazy na nesprávně zodpovězené otázky. Existuje proto *view*, který uživatelům zobrazuje historii všech jejich evaluací.

Mentorský účet sdílí veškerou funkcionalitu se studenty, ale je mu v navigačním menu navíc zobrazena nabídka *„Mentor tools“*, jejímž prostřednictvím získává mentor dodatečnou funkcionalitu. Ta se skládá ze tří možností:

- **Vytvářet kurzy** – jako autor je uveden vždy aktuálně přihlášený uživatel
- **Vytvářet kvízy** – kvízy jsou vytvářeny pomocí formuláře, který je generován dynamicky podle požadavků a využívá Ajaxu
- **Zobrazovat statistiky** – každý mentor může zobrazovat seznamy evaluací, které jsou roztříděny podle mentorem vlastněných kurzů a ty jsou dále tříděny podle jím vytvořených kvízů

Pro lepší představu funkcionality aplikace a vymezení jejích hranic, se níže nachází UML diagram. Jde o diagram užití – Use Case – který zachycuje vnější pohled na podobu aplikace. V případě tohoto modelovaného systému rozlišujeme tři aktéry, kterými jsou Nepřihlášený uživatel, Přihlášený uživatel (student) a Mentor.



Obrázek 5: Use-case diagram vyvíjené aplikace.

K vytvoření diagramů byl využit online webový nástroj Processon, který je dostupný na adrese *www.processon.com*.

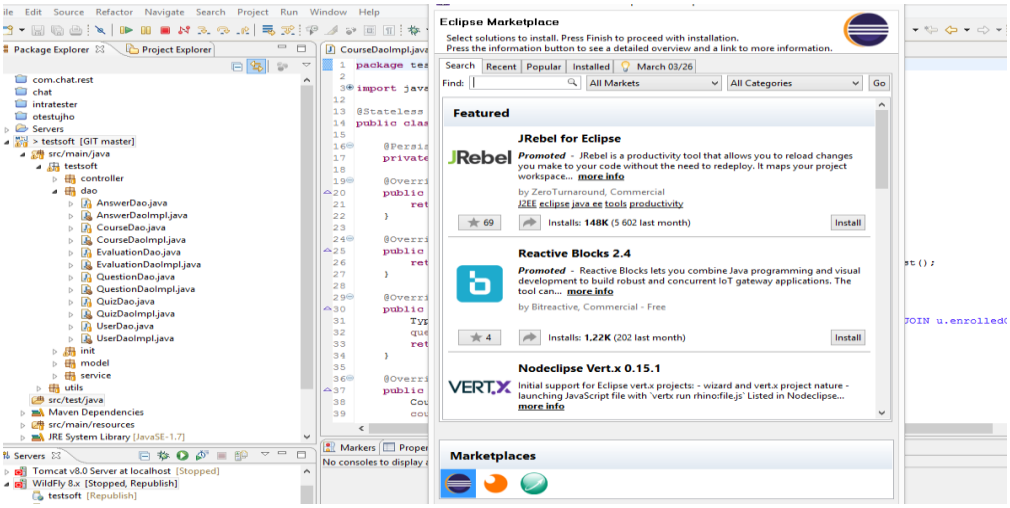
3. Technologická část

3.1. Prostředí a architektura

Eclipse Luna 4.4.2

Jedná se o multiplatformní open-source vývojové prostředí, které je primárně určeno programování v jazyce Java. Pomocí takzvaných *pluginů* však lze toto flexibilní IDE rozšířit tak, že jej lze využívat i pro další programovací jazyky, jako například C++ nebo PHP.

Síla Eclipse spočívá právě ve výše zmíněných pluginech. Nejrůznějších rozšíření existuje celá řada. Patří mezi ně Aplikační servery, nástroje pro tvorbu UML diagramů, integrace buildovacího nástroje Apache Maven, podpora týmového vývoje softwaru (SVN, GIT), nejrůznější editory, browsery, témata a tak podobně [5].



Obrázek 6: Eclipse Marketplace – přehled pluginů

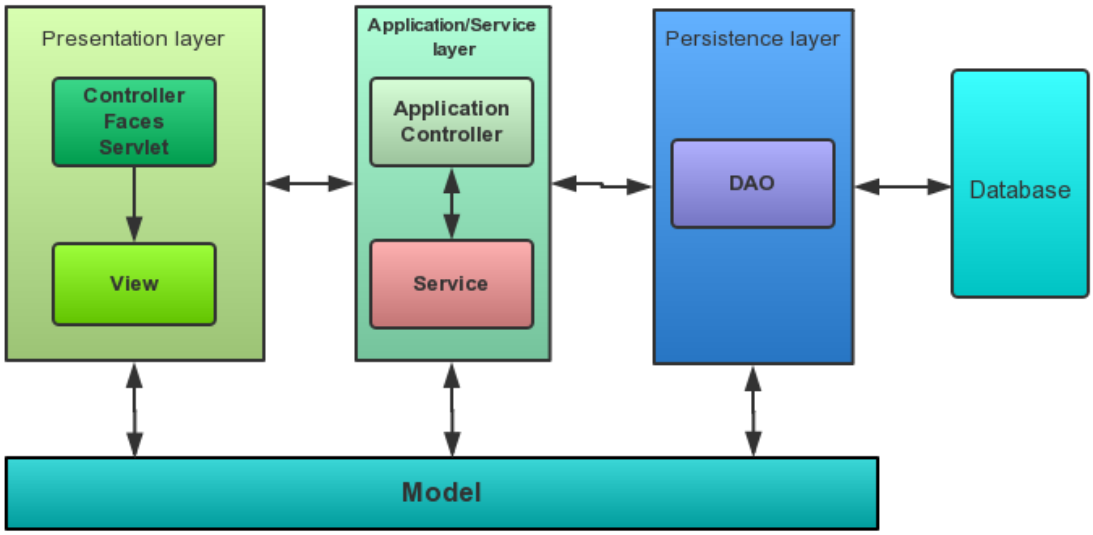
Tato otevřenost a velká možnost přizpůsobení je jedním z důvodu, proč projekt Eclipse shromáždil obrovskou uživatelskou základnu a velice aktivní komunitu. Základní instalace však obsahuje pouze nejdůležitější nástroje, jakými jsou třeba kompilátor či debugger. Nástroj pro grafickou tvorbu GUI, nebo aplikační servery v této instalaci nenajdeme.

WildFly 8

Jedná se o open-source aplikační server ze série *JBoss*. Je napsán v Javě a za jeho vývoj momentálně zodpovídá známá americká společnost poskytující open-source software, *RedHat*. Je výjimečně rychlý, obsahuje odlehčenou a výkonnou implementaci specifikace Java EE verze 7 a je jí plně kompatibilní [6].

Architektura aplikace

Aplikace je založená na vícevrstvé architektuře, jejíž struktura je znázorněna na obrázku číslo 7. Tento přístup k vývoji webových aplikací je velmi oblíbený z mnoha důvodů. Zdrojový kód jednotlivých logik je oddělený a proto dobře čitelný. Aplikace se tím stává flexibilnější, zásahy do kódu jsou rychlejší, chyby transparentnější a rozšiřitelnost mnohem pohodlnější. Takové aplikace jsou navíc v případě potřeb jednodušeji škálovatelné.



Obrázek 7: Vícevrstvá architektura vyvíjené aplikace

Datovou vrstvu aplikace zastává defaultní *embedded databáze* aplikačního serveru WildFly, kterou je **H2 Java SQL databáze**. H2 je velmi rychlou in memory databází, která využívá JDBC API [7]. Pro potřeby aplikace je tato volba dostačující. Pro produkční prostředí je doporučeno nasazení plnohodnotné databáze (Oracle Database, MySQL, PostgreSQL …).

To je z pohledu konfigurace JPA změna, která zabere nanejvýš několik sekund. Je potřeba pouze nastavit správné spojení v souboru *persistence.xml*, jak je vidět na obrázku číslo 8.

```
<property name="eclipselink.jdbc.url" value="jdbc:mysql://localhost:3306/intratester" />
<property name="eclipselink.jdbc.user" value="root" />
<property name="eclipselink.jdbc.password" value="fantasy" />
<property name="eclipselink.jdbc.driver" value="com.mysql.jdbc.Driver" />
<property name="eclipselink.logging.level" value="FINE" />
<property name="eclipselink.ddl-generation" value="none" />
<property name="javax.persistence.schema-generation.database.action" value="drop-and-create" />
<property name="javax.persistence.schema-generation.create.source" value="metadata" />
<property name="javax.persistence.schema-generation.drop.source" value="metadata" />
<property name="javax.persistence.sql-load-script-source" value="/META-INF/sql/data.sql" />
```

Obrázek 8: Záměna embedded H2 za MySQL v persistence.xml

Deployment aplikace zahrnuje všechny kroky a aktivity potřebné pro zprovoznění aplikace. Při tomto procesu jsou Java EE aplikační komponenty instalovány do takzvaných *Java EE kontejnerů*.

Java EE Aplikační Server poskytuje následující kontejnery.

- **EJB kontejner -** řídí provádění Enterprise Java Beanů.
- **Web kontejner –** řídí proces vykonávání webových stránek, servletů nějakých EJB komponent a dalších.
- **Kontejner klientských aplikací –** podporuje vykonávání klientských aplikací. Aplikační klient a jeho kontejner běží na lokálním počítači.

Tyto kontejnery jsou rozhraním mezi komponenty a low-level platformu specifikující funkcionalitou. Komponentu nelze vykonat, pokud nebyla nasazena do příslušného kontejneru. To zahrnuje specifikování nastavení kontejneru pro každou komponentu aplikace a pro Java EE aplikaci samotnou. Za tím stojí *Java EE security model*, *Java EE transaction model*, *Java EE remote connectivity model* a *JNDI lookup services*, které v podnikových aplikacích poskytují jednotné rozhraní pro přístup komponent k datům a objektům pomocí různých jmen [8].

3.2. Platforma Java Enterprise Edition

Java Enterprise Edition (dále Java EE, nebo JEE) je nadstavba platformy Java, uzpůsobená pro vývoj enterprise (podnikových) aplikací a informačních systémů. Je standardem v komunitou řízeném podnikovém software. Java EE je vyvíjena za pomoci Java Community Process, příspěvků odborníků z oboru, komerčních a open source organizací a bezpočtu jednotlivců. Každá její verze integruje řadu nových funkcí, které se ztotožňují s potřebami průmyslu, zlepšuje přenositelnost aplikací a zvyšuje produktivitu vývojářů. [9]

Pod platformou Java EE jsou aplikace vyvíjeny na základě API (Application Programming Interface) definovaných v jednotlivých specifikacích. Samozřejmostí je zde možnost integrace frameworků a technologií třetích stran.

Cílem této práce není rozbor celé platformy a z důvodu její robustnosti to ani není možné. V této části jsou však představeny klíčové záležitosti týkající se technologií Javy Enterprise Edition verze 7, které byly v aplikaci implementovány, nebo mají souvislost s řešenou problematikou.

3.2.1. Enterprise Java Beans

Enterprise beans (EJB) jsou Java EE komponenty, které běží v EJB kontejneru aplikačního serveru a implementují aplikační logiku (business logiku). EJB kontejner svým Enterprise Java Beans poskytuje služby ze systémové úrovně, jakými jsou třeba zabezpečení nebo transakční zpracování, a řídí jejich životní cyklus. Tyto služby umožňují rychle vytvářet třídy, které tvoří jádro transakčních Java enterprise aplikací [8].

Existují dva typy EJB.

- **Session bean –** Provádí klientské požadavky v rámci relace (session) konkrétního klienta.
- **Message-driven bean –** Zpravidla se chová jako posluchač pro určitý typ zpráv, jako například *Java Message Service API*.

Session Bean

Tento typ beanu obsahuje logiku, kterou lze vyvolat programově klientem a to lokálně, vzdáleně, nebo formou webových služeb [8]. Rozlišují se tři typy těchto beanů: *statefull, stateless* a *singleton*.

- **Stateless** – v průběhu životního cyklu aplikace udržuje AS proměnlivý počet instancí stateless session bean, které uchovává v takzvaných *poolech*. Když klient vyšle požadavek na tento bean, obslouží ho aktuálně volná instance. Není tedy udržován žádný vztah mezi instancí beanu a klientem. Jsou značeny anotací `@Stateless`, viz obrázek 9.
- **Statefull** – je úzce svázána s klientskou relací, kdy v rámci této relace je potřeba udržovat stav relace tj. výsledky zpracování předchozích požadavků zaslaných konkrétním klientem v průběhu této relace Každá její instance je vytvořena a připojena k určitému klientovi a zpracovává pouze jeho požadavky. To znamená, že mezi jednotlivými requesty existuje konverzační stav. Na konci životního cyklu je zavolána metoda *Remove* a instance je odstraněna. Značí se anotací `@Statefull`.
- **Singleton** – je instanciován pouze jednou za životní cyklus aplikace. Tento druh beanu byl navržen pro případy, kdy je potřeba mít právě jednu EJB instanci sdílenou pro všechny aktuálně připojené uživatele [8]. Odlišuje se anotací `@Singleton`.

```
@Stateless
public class UserDaoImpl implements UserDao {

    @PersistenceContext(unitName = "testsoft")
    private EntityManager entityManager;
```

Obrázek 9: Příklad stateless session beanu, využitého jako DAO.

Message driven Bean

Jedná se o enterprise beany, které umožňují aplikaci zpracovávat požadavky asynchronně. Může být použit jako JMS message listener podobný event listeneru, ale namísto eventů přijímá JMS messages [8].

3.2.2. Java Persistence API

JPA poskytuje vývojářům Java aplikací (Java SE i Java EE) možnosti objektově relačního mapování pro správu relačních dat ve svých aplikacích [8]. Jedná se však pouze o specifikaci. Implementacemi JPA jsou například *Hibernate*, *Eclipselink*, *Oracle Toplink* a další.

Entita je základním prvkem JPA. Jedná se o plain old java object (POJO), který typicky reprezentuje databázovou tabulku. Musí však splňovat určitá kritéria.

- entita musí nést anotaci `javax.persistence.Entity`
- musí obsahovat prázdný přístupný *konstruktor*
- nesmí obsahovat klíčové slovo `final`
- proměnné entity musejí být přístupné pouze prostřednictvím *accessor* metod entity (gettery a settery)

```
@Entity
@Table(name = "user")
public class User implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private long id;

    @Column(name = "login", unique = true)
    private String login;
    private String password;

    private String email;
    private String fullName;
    private String role;
    private String description;

    public User() {
    }

    //~ Getters and Setters
    public long getId() {
    public void setId(long id) {
```

Obrázek 10: Příklad entity reprezentující tabulku – user

Proměnné entitní třídy, které lze vidět na 10. obrázku, představují tedy sloupce tabulky a její jednotlivé instance jsou databázové řádky. Životní cyklus entity se skládá ze čtyř stavů a to: *New*, *Managed*, *Removed* a *Detached*.

Z pohledu kardinality Java Persistence API jsou rozlišovány 4 vztahy, které mohou mezi entitami nastat.

One To One – 1:1

Tento vztah nastává tehdy, když se každá instance entity vztahuje právě k jedné instanci jiné entity. Stejně jako v relačních databázích se nejedná o moc obvyklý vztah. Jeho zápis je znázorněn na obrázku číslo 11.

```
@OneToOne
private Quiz quiz;
```

Obrázek 11: Příklad zápisu vztahu 1:1

One To Many – 1:n

Jedná se o obvyklý vztah, který říká, že pro danou instanci entity existuje více instancí jiné entity. Příkladem je zde učitel, který je vlastníkem jednoho nebo více kurzů. Viz obrázek číslo 12.

```
@OneToMany(mappedBy = "owner")
private List<Course> ownedCourses;
```

Obrázek 12: Příklad zápisu vztahu 1:n

Many to One – n:1

Tento vztah je opakem předchozího vztahu *OneToMany* a vytváří se vždy na protějších entitách. Na existující vztah *1:n* (*uživatel:kurz*) se tedy z opačného pohledu hledí jako na *n:1*, čili konkrétní kurz má právě jednoho vlastníka.

```
@ManyToOne
@JoinColumn(name = "course_owner")
private User owner;
```

Obrázek 13: Příklad zápisu vztahu n:1

Many to Many – m:n

Instance jedné entity mohou mít vzájemnou vazbu s více instancemi entity druhé. Například studenti mohou mít zapsáno více kurzů a kurzy mohou mít několik zapsaných studentů. Na obrázku číslo 14 je znázorněné, jak programově provést připojení asociační tabulky a jak nastavit použití těch správných klíčů.

```
@ManyToMany
@JoinTable(name = "course_user",
    joinColumns = {@JoinColumn(name = "course_id", referencedColumnName = "id")},
    inverseJoinColumns = {@JoinColumn(name = "user_id", referencedColumnName = "id")})
private List<User> enrolledStudents;
```

Obrázek 14: Příklad vztahu m:n pro entitu Course

Obrázek číslo 15 na druhé straně pouze vytváří list využívající to samé mapování, které již bylo stanoveno u protější entity.

```
@ManyToMany(mappedBy = "enrolledStudents")
private List<Course> enrolledCourses;
```

Obrázek 15: Příklad vztahu m:n pro entitu User

V souvislosti s JPA je nutné zmínit objekt **Entity Manager (EM)**.

Objekt EM poskytuje programátorovi *interface*, který slouží k interakci s *persistence kontextem*. Jeho instance je s ním spojená a nabízí velké množství metod, které operují nad daty. *Entity Manager API* se používá při vytváření a mazání instancí perzistentních entit, k vyhledávání položek podle primárního klíče a k vytváření speciálních dotazů [10].

JPA Persistence Unit – persistence.xml

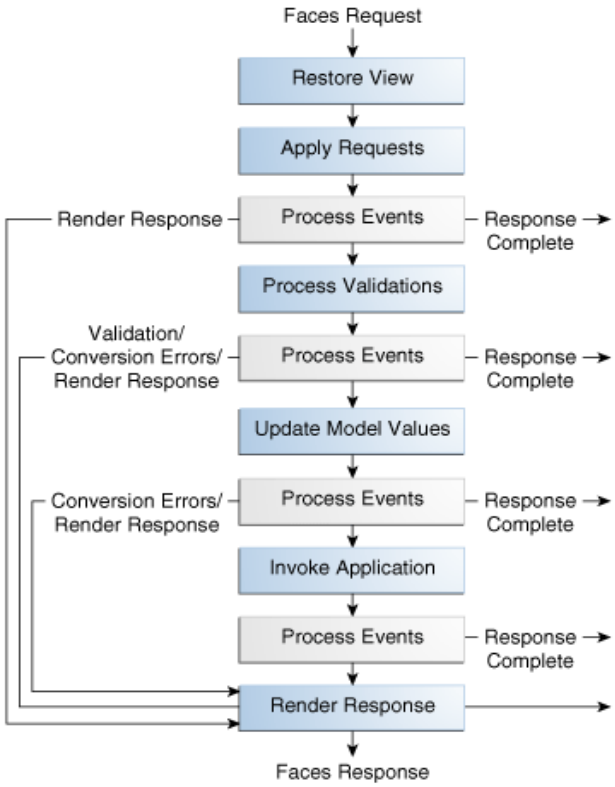
Jedná se o klíčový soubor pro konfiguraci JPA. Obsahuje definici tzv. persistentních jednotek pro uživatelem definované entity a jejich vazbu na konkrétní relační DB. Nastavuje se zde veškerá konfigurace ohledně databázového připojení, generování dat, načítání scriptů, logování a tak podobně.

3.2.3. Java Server Faces

JSF je *component-framework* fungující na straně serveru, který slouží k vytváření webových aplikací postavených na platformě Java. Je založený na architektuře MVC. *View* vrstva, reprezentována *xhtml* soubory, je tedy kompletně oddělena od controllerů. Tím mimo jiné dochází k výraznému zjednodušení vývoje a k mnohem lepší údržbě. Rozšiřitelnost těchto aplikací je také velice usnadněna.

Jeho základním stavebním prvkem je API zastřešující reprezentaci komponent, řízení jejich stavů, zpracování událostí, proces validací, konverzi dat, definici navigace a podporu internacionalizace. Neméně důležitým prvkem jsou knihovny, tzv. *Tag libraries*, které umožňují přímo do webových stránek přidávat komponenty a propojovat je s objekty, nejčastěji aplikačními controllery, na straně serveru [8].

Životní cyklus JSF začíná HTTP requestem klienta a končí tím, že server odpoví stránkou přeloženou do HTML. Mezitím však probíhá celá řada dalších činností, viz obrázek číslo 16 popisující životní cyklus JSF.



Obrázek 16: Standardní životní cyklus JSF Request-Response [8]

Facelets

Jedná se o odlehčený jazyk určený k vytváření JSF *views*, s využitím jazyka XHTML a knihoven tagů, které se do view promítají prostřednictvím XML *namespaces*. Facelets nabízí podporu pro *Expression Language* a možnost šablonování, které usnadní vývoj front-endu možností vytvoření tematických šablon a znovu-použitelností opakujících se částí kódu [8].

Starší technologií prezentační logiky jsou Java Server Pages. S příchodem JSF ve verzi 6 se stále hledělo na to, aby byly tyto technologie v určitých záležitostech sladěny. Právě Facelets od této myšlenky odstoupil a poskytl vysoce výkonnou technologii, která však byla mířená čistě na JSF. Současná situace je taková, že se vývojáři Javy EE snaží JSP nahrazovat JSF, nicméně v praxi se z důvodu existence velkého množství projektů postavených na JSP tento cíl naplňuje velmi pomalu.

Expression Language

EL je nepostradatelnou součástí JSF frameworku. Poskytuje totiž důležitý mechanismus, který umožňuje prezentační vrstvě komunikovat s aplikační logikou. EL není doménou pouze JSF, ale je využíván také například výše zmíněnými Java Server Pages a *Context and Dependency Injection (CDI)*. Může však být také využit samostatně [8].

Ve vyvíjené aplikaci je EL využit převážně k přístupu do instančních proměnných beanů a k provádění jejich metod. Toho je docíleno zapsáním jednoduchého výrazu v prostředí jsf stránky. Může jít o získání hodnoty, uložení hodnoty, provedení metody, nebo třeba o získání seznamu za účelem iterace.

Rozlišují se dva typy užití EL, které se syntakticky liší. Prvním typem je zápis, který provede vyhodnocení výrazu okamžitě. Takový výraz se zapisuje ve tvaru `${user.fullName}`. Druhý typ může být vyhodnocený v různém stádiu životního cyklu stránky a jeho syntaxe je znázorněna na obrázku číslo 17.

```
<div class="credentials">
<p:outputLabel value="#{loggedUser.loginName}" />
</div>
```

Obrázek 17: Příklad zápisu výrazu v EL

3.3. Primefaces

Jedná se o odlehčenou knihovnu UI komponent pro aplikace postavené na JSF frameworku. Slovo odlehčená v tomto případě neznamená, že obsahuje málo komponent, nýbrž to, že je naprosto oddělená od řešení třetích stran a ve výsledku jde o pouze jedinou *jar knihovnu* bez jakýchkoliv dalších závislostí [11].

Komponenty, které tento framework poskytuje, vývojářům skrývají veškerou složitost, ale jejich flexibilita zůstává zachována.

Mezi další výhody tohoto frameworku patří:

- Podpora pro *Ajax* s intuitivním xhtml zápisem
- Validace na straně clienta
- Framework pro vypisování dialogů
- Podpora pro *Push* gesta
- Kit pro vývoj mobilních aplikací
- Theme engine nabízející řadu témat zdarma
- Velmi dobře zpracovaná dokumentace
- Velká a aktivní uživatelská komunita, se kterou komunikují samotní vývojáři PrimeFaces

O celkové úspešnosti tohoto frameworku vypovídá obrázek číslo 18. Jedná se o anketu pořádanou společností *DevRates.com*, jejímž cílem bylo zjistit, jaký framework využívají vývojáři nejraději k tvorbě bohatých uživatelských prostředí v Javě.

Obrázek 18: Nejoblíbenější UI frameworky pro Java aplikace [11]

3.4. Apache Maven

Jedná se o nástroj, který se používá k „buildování“ a řízení Java projektů. Jeho hlavním cílem je umožnit vývojářům pracovat efektivně, transparentně a především snížit čas potřebný k vývoji [12].

Maven poskytuje jednotný buildovací systém. Know-how, které vývojář získá při práci s Mavenem na svém prvním projektu, si pak přenáší k projektům dalším. Pro jeho pochopení je však nutné vědět, jak takový buildovací proces vypadá.

Základem Mavenu je POM a jeho XML zápis uložený v konfiguračním souboru stejného názvu - *pom.xml*.

POM

Project Object Model, neboli POM, je v Mavenu základní pracovní jednotkou. Jedná se o XML soubor, který obsahuje informace o projektu a konfigurační detaily využívané Mavenem k vytváření projektů [13].

Jednou z nejdůležitějších částí POMu je správa závislostí projektu, která se umisťuje mezi tagy `<dependencies>`. Tím dochází k centralizaci všech stavebních částí aplikace. Klíčová je však schopnost Mavenu z těchto zápisů dependencí stahovat jednotlivé technologie a frameworky bez nutnosti další konfigurace. Dnešní IDE obsahují potřebné repozitáře, které znají většinu používaných závislostí. Namísto nejednotného hledání, stahování a přesouvání závislostí, stačí vložit závislost do POMu a „zbuildovat“ projekt. Na 19. obrázku je vidět příklad zápisu pro dependenci Java EE API. `Scope provided` však říká, že tato závislost bude poskytnuta Aplikačním Serverem a nemusí se tedy stahovat.

```
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>7.0</version>
        <scope>provided</scope>
    </dependency>
<dependencies>
```

Obrázek 19: Příklad zápisu dependence ze souboru pom.xml

3.4.1. Apache Shiro

Shiro je široce rozšířený jednoduchý *security framework*, který aplikaci poskytuje řadu potřebných bezpečnostních mechanismů. Je využíván jak pro zabezpečení jednoduchých mobilních aplikací, tak pro zabezpečení robustních enterprise systémů [14]. Veškerá jeho konfigurace je situovaná do jednoduchého textového souboru nesoucího název *shiro.ini*. Hlavními mechanismy, které Shiro poskytuje, jsou:

- Autentizace
- Autorizace
- Kryptografie
- Session management
- Webová integrace

3.4.2. SLF4J

Plným názvem – *Simple Logging Facade for Java* – slouží jako rámec pro různé logovací frameworky [15]. Při jeho nasazení umožňuje aplikace v deployment čase logovat potřebné události. To je pro administrátory, provozovatele aplikací a podobné role velmi důležité. V případě poruch a podobných problémů způsobených při provozu, se díky těmto logům odhalují jednodušeji postupy, které problém vyvolaly a může se na ně rychleji reagovat.

Jeho implementace není složitá. K získání zdrojových souborů lze využít například Maven a následný zápis je otázkou dvou řádků, viz obrázek číslo 20.

```
private final Logger logger = LoggerFactory.getLogger(getClass());
...
logger.info("User " + user.getFullName() + " has been enrolled");
...
```

Obrázek 20: Příklad užití Loggeru pro informování o zapsání studenta do kurzu

4. Implementační část

Postup při implementaci se dá rozdělit do několika specifických bloků. Po analýze a návrhu UML (Use-case) diagramu aplikace, jak již bylo zmíněno v **Analytické části**, následovala příprava prostředí pro vývoj. Byl vytvořen projekt a provedena základní konfigurace, která ho uvedla do spustitelného stavu.

Po vytvoření struktury projektu přišla na řadu databázová vrstva. Bylo navrženo schéma databáze, ze kterého se vycházelo při tvorbě modelu (entitních tříd) aplikace. Součástí toho kroku byla také konfigurace a navázání spojení s databází.

Jako další již mohla být zahájena implementace perzistentní vrstvy, *Data Access Objektů*, která se stará o veškeré přístupy do databáze a jejich transakční zpracování. Realizace DAO umožnila zahájení tvorby testovacích dat a následné ověření správnosti naprogramovaných vazeb entitních tříd.

Další úsek vývoje už poskytoval autorovi určitou transparentnost, díky které mohl být zahájen vývoj front-endu aplikace. Tato část byla zřejmě ze všech nejrozsáhlejší, protože během práce na *view* vrstvě probíhala paralelně také implementace aplikačních controllerů. Díky tomu mohla být do UI promítána již vytvořená testovací data namísto takzvaného *hardcodování* dodatečného obsahu.

Poslední etapa vývoje se týkala servisní vrstvy. Ta zahrnovala naprogramování servisních tříd, utilit a bezpečnostních záležitostí aplikace.

Přestože byla aplikace testována v průběhu vývoje, značného prostoru se dostalo také uživatelským testům. Ty jsou popsány v závěru této kapitoly.

4.1. Vytvoření projektu a jeho konfigurace

Po úspěšném nainstalování vývojového prostředí Eclipse Luna 4.4.x [4], bylo zapotřebí nainstalovat potřebné pluginy. Celkem se jednalo o dva balíčky. Prvním byl *JBoss tools*, který mimo jiné umožnil konektivitu s WildFly aplikačním serverem. Druhý plugin, *m2e*, poskytl integraci Mavenu do IDE.

Následovalo založení nového projektu. Projektu typu *Dynamic Web Project* byla provedena základní nastavení, jakými je například nastavení runtime JRE a byl vygenerován soubor *web.xml*. Tento projekt byl poté zkonvertován na *Maven project*, čímž vznikl klíčový soubor *pom.xml*. Protože Eclipse umožňuje předem definovat některé technologie na kterých je projekt postaven, bylo toho využito ve prospěch JSF a JPA. Tím došlo k vygenerování dalších konfiguračních souborů a to *faces-config.xml* a *persistence.xml*. Posledním konfiguračním souborem této aplikace je soubor *shiro.ini* patřící security frameworku Apache Shiro.

4.1.1. Konfigurace web.xml

Jedná se o základní konfigurační soubor Java Web Aplikací (The Deployment Descriptor). Aby byl Java servlet přístupný z prohlížeče, je třeba říci kontejneru, jaký servlet deployovat a na jaké URL ho mapovat. Viz obrázek číslo 21.

```
<!-- Servlet settings -->
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
</servlet-mapping>
```

Obrázek 21: Nastavení Servletu v souboru web.xml

4.1.2. Internacionalizace a konfigurace ve faces-config.xml

Tento XML soubor byl dříve využíván převážně pro nastavení navigace pro view vrstvu JSF. V současné době se největší popularity dostává nahrazování tohoto způsobu konfigurace anotacemi a to přímo na místech, kterých se konfigurace týká.

Tento soubor je však využit kvůli jeho druhé funkci, kterou je internacionalizace. Projekt je sice poskytnut pouze v jednom jazyce (angličtina), nicméně díky tomuto přístupu je rozšíření do více jazyků jednoduchým úkonem. Vytvoří se nový *properties* soubor, který bude obsahovat ten samý text jako výchozí soubor, ale přeložený do požadovaného jazyka. Následně přidá záznam o nové podpoře do zmiňovaného souboru, viz obrázek 22. Potom stačí, aby si uživatel zvolil jazykovou podporu jemu blízkou. Aplikace se postará o přepnutí mezi těmito soubory.

```
<application>
    <locale-config>
        <default-locale>en_US</default-locale>
    </locale-config>
    <message-bundle>
        l10n.messages
    </message-bundle>
    <resource-bundle>
        <base-name>l10n.messages</base-name>
        <var>msgs</var>
    </resource-bundle>
</application>
```

Obrázek 22: Konfigurace internacionalizace - faces-config.xml

V tomto konkrétním případě je pro dosažení požadovaného výsledku vytvořen soubor *.../l10n/messages_en.properties*, do kterého se texty z aplikace přidávají ve formátu `klíčové slovo=hodnota`, čili například `menu.home=Homepage`.

K požadovaným hodnotám se ve view přistupuje speciálním zápisem formou EL. Příklad je znázorněn na 23. obrázku.

```
<p:menuitem value="#{msgs['menu.homepage']}"
    icon="ui-icon-home" outcome="welcome" />
```

Obrázek 23: Získání properties hodnoty pro klíčové slovo menu.homepage v souboru default-menu.xhtml

4.1.3. Konfigurace Mavenu – pom.xml

Úvodní část POMu obsahuje informace o projektu. Mezi tyto záznamy patří id, jméno, verze, *packaging* (jar / war) a další. Tyto údaje byly vyplněny ve *wizardu* při vytváření projektu.

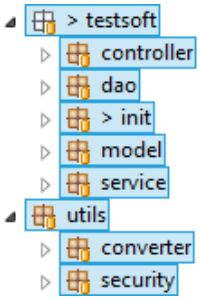
Následuje část závislostí. Vyhledání požadovaných závislostí proběhlo na základě zvolených technologií. V případě, že nebyla závislost nalezena samotným pluginem m2e v Eclipse, byla vyhledána na webu závislostí **mvnrepository.com**.

Následující dependence byly přidány: *javaee-api 7.0, joda-time 2.7, junit 4.11, slf4j-api 1.7.7, logback-classic 1.1.2, primefaces 5.1 + all-themes* a *Shiro-core 1.2.3 + shiro-web.*

Po stisknutí klávesové zkratky *Alt + F5* v prostředí Eclipse došlo k vyvolání Maven funkce *Upadate Project*, čímž proběhlo stažení všech výše zmíněných dependencí. Po aktualizaci projektu byly tyto závislosti připravené k použití.

Struktura projektu

Vychází z defaultního rozložení projektu typu *Dynamic Web Project*. Na cestě *src/main/java* se nacházejí dva *packages*, jejichž obsahem je veškerý java back-end aplikace. Tyto dva balíčky obsahují další názvem specifičtější balíčky, jak lze spatřit na obrázku číslo 24.



Obrázek 24: Struktura projektu z pohledu jednotlivých balíčků

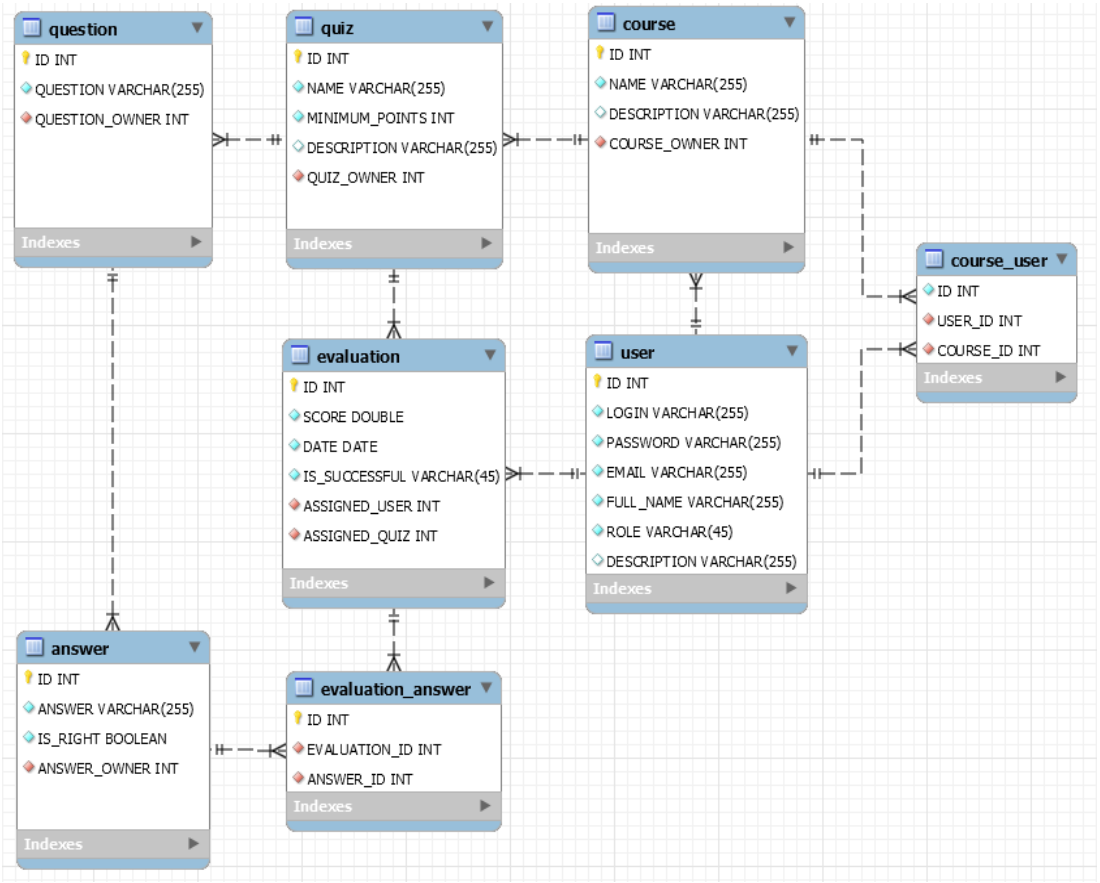
Front-end je umístěn ve složce *src/main/webapp*. V této složce se nacházejí *welcome-page* aplikace, kterou je *login.xhtml* a stránka pro registrace *registration.xhtml*. Dále jsou zde umístěny ostatní xhtml stránky, *resources* projektu (obrázky, styly) a složka WEB-INF obsahující konfigurační soubory.

4.2. Databázová vrstva

4.2.1. Struktura databáze

Po analýze funkcionality a vytvoření Use-case diagramu bylo patrné, jaké objekty budou v aplikaci figurovat. Jedná se o uživatele, kurz, kvíz a vyhodnocení, přičemž kvíz se skládá z otázek a odpovědí.

K vytvoření EER Diagramu posloužil open source nástroj umožňující grafickou tvorbu databázového schématu, MySQL Workbench 6.2 [16].



Obrázek 25: EER Diagram vyvíjené aplikace

Vytvářením výše zmíněného diagramu dostaly podobu všechny potřebné vazby a vztahy mezi entitami. Ačkoliv existuje mnoho automatizovaných nástrojů na generování entit z existující databáze, rozhodl se autor psát je ručně kvůli lepšímu pochopení problematiky.

Počátečním bodem je entita **user**. Obsahuje přihlašovací údaje uživatele, informace o jeho účtu a o uživateli samotném. Uživatel může být vlastníkem několika kurzů a proto mezi entitami *user* a *course* existuje vztah 1:n. Také však může být v kurzu zapsán v roli studenta. Takových kurzů může mít zapsaných libovolný počet, čímž vzniká vztah m:n, protože kurz může mít pochopitelně libovolný počet zapsaných studentů. Vzniká tedy asociační tabulka nazvaná podle konvencí *course_user*. Uživatel má také přímou vazbu na entitu *evaluation*. Protože může mít uživatel evaluací několik, jedná se opět o vztah 1:n.

Další entitou v pořadí je entita **course.** Tato entita obsahuje popisné informace a slouží jako obal pro vybrané kvízy. Jak vyplývá z předchozího odstavce, kurz je vlastněn jedním uživatelem a může v něm být více zapsaných studentů. Kvízů může vlastnit kurz libovolný počet, proto vztah 1:n.

Entita **quiz** je spolu s entitami **question** a **answer** v aplikaci prezentována jako celek. Vazba je taková, že *quiz* se skládá z otázek a ty se skládají z několika odpovědí. Každý *quiz*, stejně jako entita *user*, může obsahovat libovolný počet evaluací.

Entita **evaluation** je vázána na své vlastníky, kterými jsou vždy uživatel a kvíz. Dále obsahuje rekapitulující a vyhodnocující informace. Ve vztahu m:n přistupuje přes asociační tabulku *evaluation_answer* k nesprávně zvoleným odpovědím. Tímto je uživateli umožněno nahlédnout do jeho chybných rozhodnutí a poučit se tak v náhledu vyhodnocení ze svých chyb.

4.2.2. Implementace modelových objektů

Po ucelení pohledu na problematiku formou zmiňovaných diagramů následovala implementace modelové vrstvy aplikace. Všechny tabulky kromě asociačních, jsou v aplikaci reprezentovány formou entitních tříd. Každá z těchto tříd má dvě základní anotace pocházející z `javax.persistence.*`.

- `@Entity` – Určuje, že anotovaná třída je persistentní entitou.
- `@Table(name="test")` – Specifikuje tabulku, pod jménem test.

Sdíleným atributem napříč všemi entitami je atribut Id. Tomu jsou přiřazeny také dvě anotace. První z nich říká, že jde o privátní klíč. Druhá anotace zajišťuje automatické generování hodnot příslušného atributu (AI). Jejich zápis je vidět na následujícím obrázku.

```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private long id;
```

Obrázek 26: Zápis primárního klíče entit

Pokud je třeba zajistit unikátnost atributu na úrovni entitní třídy, docílí se toho způsobem znázorněném na obrázku 27. V tomto případě jde o zajištění unikátnosti *loginu* jednotlivých uživatelů.

```
@Column(name = "login", unique = true)
private String login;
```

Obrázek 27: Zápis požadavku pro unikátnost atributu

Další zajímavostí jsou atributy vznikající na základě vztahů mezi entitami. V případech 1:n se jedná vždy o kombinaci *objekt:seznam objektů*. Při vztazích m:n vznikají *ArrayListy* na obou stranách. Tyto atributy bylo nutné popsat vhodnými anotacemi, které již byly diskutované v **Technologické části – JPA**, proto zde budou zmíněny pouze ty, které jsou v práci využity. Jsou jimi:

- OneToMany `javax.persistence.OneToMany`
- ManyToOne `javax.persistence.ManyToOne`
- ManyToMany `javax.persistence.ManyToMany`

Ostatní atributy se nijak neliší od klasických atributů standardních Java tříd. Všechny atributy entity mají modifikátor přístupu nastavený jako `private` a přistupuje se k nim pouze prostřednictvím *accessor* metod (gettery a settery).

4.2.3. Nastavení a připojení databáze

V počátcích vývoje byla využívána databáze MySQL, nainstalovaná v linuxovém prostředí a ovládaná z *bashe*. Konfigurace této databáze je již zmíněna v **Technologické části**. Vzhledem k tomu, že vývoj aplikace později probíhal celkem na třech strojích, bylo využito možnosti *embedded in memory* databáze. To je ve vývojové fázi velmi přínosné, poněvadž odpadá potřeba instalace a konfigurace databázového serveru.

Veškerá nastavení týkající se zmiňované problematiky jsou umístěna v souboru *persistence.xml*. V tomto XML souboru se nachází takzvaná *persistence-unit*, ve které jsou zapsány všechny entitní třídy, které aplikace v kontextu této jednotky využívá. Zápis je ve tvaru: `<class>testsoft.model.User</class>`

```
<properties>
    <property name="javax.persistence.schema-generation.database.action" value="drop-and-create" />
    <property name="javax.persistence.schema-generation.create.source" value="metadata" />
    <property name="javax.persistence.schema-generation.drop.source" value="metadata" />
</properties>
```

Obrázek 28: Nastavení persistence-unit v persistence.xml

Výše vložená část kódu stanovuje, že pokud již existuje nějaké schéma, tak se nejprve smaže a poté znovu vytvoří a dále určuje, co se má využít při generacích schématu (metadata AS). Ostatní konfigurační záležitosti jsou umístěny v souboru `$WILDFLY_HOME/standalone/configuaration/standalone.xml`.

4.3. DAO vrstva

Na této vrstvě jsou naprogramované metody určené pro přístup k persistentním datům. Tím je zaručena transparentnost persistentní vrstvy, snížena komplexita kódu business objektů a jednodušší přenositelnost. DAO v aplikaci představuje rozhraní mezi zdroji dat, kterými jsou entitní třídy a konzumenty dat, kterými jsou aplikační Controllery.

Na každou entitu připadají dva DAO soubory. První nese jmennou konvenci `EntityNameDao` a druhý `EntityNameDaoImpl`. V případě prvního se jedná o `Interface`, který obsahuje deklarace metod spolu s jejich dokumentací.

Tím druhým je EJB třída implementující zmíněný interface. Všechny EJB implementace v projektu jsou typu `@Stateless` a sdílejí společný `@PersistenceContext` pro Entity Manager, který je pojmenován *testsoft*. Metody těchto tříd se dají rozdělit do dvou skupin a to na metody, které data ukládají a metody které data vyhledávají a vracejí.

Ukládání probíhá tak, že se na příslušném Entity Manageru volá metoda `merge()`, která záznam v případě jeho existence v databázi updatuje. V opačném případě uloží instanci entity do databáze.

Vyhledání jednotlivých entit probíhá voláním metody `find(class, id)`, přičemž se této metodě předá entitní třída, ve které se má vyhledávat a primární klíč hledaného záznamu.

V praxi je však potřeba nad databází konstruovat mnohem složitější dotazy. K tomu je v JPA využit *Java Persistence Query Language* (dále JPQL). Metodou využívající JPQL je například `findEnrolledCourses(long userId)` ze třídy UserDaoImpl, která vrací seznam kurzů, do kterých je zapsaný uživatel s primárním klíčem `userId`.

```
@Override
public List<Course> findEnrolledCourses(long userId) {
    TypedQuery<Course> query = entityManager.createQuery(
            "SELECT c FROM Course c JOIN c.enrolledStudents u WHERE u.id = :userId",
            Course.class);
    query.setParameter("userId", userId);
    return query.getResultList();
}
```

Obrázek 29: Příklad metody využívající JPQL.

Tvorba testovacích dat spadá pod soubor *init/DatabasePopulator*. Tato třída nese anotace `@Startup` a `@Singleton`. Obsahem těchto tříd jsou metody využívající DAO k vytváření instancí entit. Na konci se nachází `@PostConstruct` metoda, která volá všechny výše umístěné metody a pokud je to třeba, tak je mezi sebou propojuje. Tím je zajištěno, že vždy po startu aplikace dojde k vytvoření testovacích dat. Jedná se o opravdu rozsáhlou třídu a je v ní dobře vidět, jaký je rozdíl mezi vytvářením dat pomocí objektového přístupu / SQL dotazu.

4.4. Aplikační controllery

Jak již bylo zmíněno, vývoj controllerů a views probíhal paralelně. Pro zachování přehlednosti a dobré čitelnost práce budou tyto dva úseky však rozděleny do dvou podkapitol.

Na každou entitní třídu aplikace připadá jeden aplikační controller, který poskytuje její data jednotlivým views. Tyto controllery jsou anotované jako `@Named`. To znamená, že se jedná o CDI Beany, které jsou nyní upřednostňovány před JSF Managed Beans. Přednosti se jim dostává především proto, že poskytují širší Java EE Dependency Injection.

Dále je u controllerů nutné nastavit vhodný *scope* platnosti. V kontextu zmiňované aplikace jsou využity tři.

- `@RequestScoped` – vytvořen právě jednou pro každý request
- `@SessionScoped` – session kontext je sdílený mezi všemi servlet requesty, které patří té samé HTTP session
- `@ViewScoped` – přetrvává, dokud je zachováno to samé view

Controllery obsahují proměnné, *gettery* a *settery*, díky kterým se promítají perzistentní data do views, nebo je naopak z těchto views načítají. K tomu je však zapotřebí zajistit transakční zpracování. To je zajištěno v již implementované DAO vrstvě, která se proto do těchto tříd *injectuje*, viz obrázek 30.

```
@EJB
private QuizDao quizDao;
@EJB
private QuestionDao questionDao;
@EJB
private AnswerDao answerDao;
```

Obrázek 30: Příklad injectace DAO ve třídě QuestionController

Metody controllerů jsou různými variacemi *accessor* metod. Nejčastěji se vyskytujícím typem je metoda předávající parametr primárního klíče hledané hodnoty. Tento parametr je nejčastěji získáván z URL. To znamená zápis URL ve formátu například *testsoft/welcome.jsf?ownerId=1*.

Obrázek číslo 31 představuje metodu vracející seznam kurzů, které vlastní uživatel s primárním klíčem `ownerId`. Utilita JSFUtils je popsána v kapitole zabývající se **servisní vrstvou**.

```
public List<Course> getOwnedCourses() {
    if (ownedCourses == null && JsfUtils.getRequestParameterAsLong("ownerId") != null) {
        ownedCourses = userDao.findOwndedCourses(JsfUtils.getRequestParameterAsLong("ownerId"));
    }
    return ownedCourses;
}
```

Obrázek 31: Příklad metody UserController#getOwnedCourses

Sofistikovanější metodou nacházející se na controlleru je například metoda `doStep()`, která se v rámci session kontextu třídy `QuestionController` stará o navigaci mezi jednotlivými otázkami konkrétního kvízu, viz obrázek číslo 32. Dále je zde nastaveno, že pokud je otázka otázkou poslední, nastaví se příznak na *true* a na view je zajištěna záměna tlačítka z `Next` na `Evaluate quiz`.

```
public void doStep() {
    if (questionIndex < numberOfQuestions && getSelectedAnswerId() != null) {
        selectedAnswers.add(answerDao.findById(getSelectedAnswerId()));
        if (questionIndex < numberOfQuestions - 1) {
            questionIndex++;
        }
        if (questionIndex == numberOfQuestions - 1) {
            setLastQuestion(true);
        }
    }
    setSelectedAnswerId(null);
   log.info("Selected:" + selectedAnswers.size());
}
```

Obrázek 32: Zápis metody QuestionController#doStep

4.5. JSF views

JSF umožňuje *templating*, čehož bylo v prezentační vrstvě využito. Hlavním souborem hierarchie je *pages/templates/template.xhtml*. Tento soubor obsahuje základní html tagy, kterými jsou <*head*>, <*body*> a tak dále. Primárními stavebními prvky této šablony jsou tagy **`<ui:insert>`**. Tím dojde k deklarování přepisovatelné oblasti. Template aplikace se skládá z odhlašovací lišty, hlavičky s logem, navigačního menu, obsahového prostoru a zápatí.

Tyto části jsou v jednotlivých views definovány pomocí tagu **<ui:define>**, nebo dědí defaultní obsah ze souboru *template*. I přesto, že je obsah generován dynamicky, bylo implementováno přes dvacet xhtml-view souborů.

Výsledné UI se skládá z komponent, výrazů a klasických html elementů. Komponenty jsou do jednotlivých `ui:compositions` promítnuty prostřednictvím *namespaces*. Všechny *namespaces* prezentační vrstvy jsou k vidění na obrázku 33.

```
xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets"
xmlns:h="http://xmlns.jcp.org/jsf/html"
xmlns:p="http://primefaces.org/ui"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:c="http://xmlns.jcp.org/jsp/jstl/core"
template="templates/template.xhtml">
```

Obrázek 33: Namespaces spolu s šablonou využité ve views

Komponenty primefaces stojí za UI aplikace. Při jejich implementaci bylo nahlíženo na *PF-showcase* [17], ze kterého byl výběr vhodných komponent velice pohodlný. Konfigurace designu primefaces uživatelského prostředí byla dodatečně provedena v souboru *web.xml*. Bylo zvoleno téma nazvané *Afternoon*.

Pro iterace přes seznamy byl využit *facelets* tag `<ui:repeat>`. Vyhodnocení podmínek a větvení na front-endu aplikace podléhalo užití JSTL. Na obrázku 34 je vidět způsob zápisu nacházejícího se na view *quiz-evaluation.xhtml*.

```
<c:choose>
<c:when test="#{evalService.succeed(evaluationController.viewedEvaluation)}">
    <p:outputLabel value="- You have been successful! -" styleClass="eval_success" />
</c:when>
<c:otherwise>
    <p:outputLabel value="- You have failed! -" styleClass="eval_fail" />
</c:otherwise>
</c:choose>
```

Obrázek 34: JSTL c:choose tag - quiz-evaluation.xhtml.

Navigaci a volání metod obstarávají především tyto komponenty:

- `p:button` – pro navigaci v těle obsahuje *outcome="String value"*
- `p:commandButton` – jako první se provede *actionListener* a poté samotná *action*, která může také vracet *String* určený pro navigaci

Podoba výše zmiňovaného `commandButtonu` je vidět na obrázku číslo 35. Tento příklad navíc obsahuje *Ajaxový* update pro formulář s id `quizCreation`.

```
<p:commandButton value="Reset all"
    actionListener="#{quizCreationService.reset()}"
    update="quizCreation" immediate="true" action="quiz-creation" />
```

Obrázek 35: Resetující commandButton #quiz-creation.xhtml.

Za zmínku stojí také implementace „Mentor tools“. Jedná se o položku v menu, která umožňuje mentorům vytvářet kvízy a zobrazovat jejich evaluace. Tato položka je zobrazena pouze v případě, že je role přihlášeného uživatele mentor. To je ověřeno pomocí tagu `<c:if>`, také ze sady JSTL.

```
<c:if test="#{userController.isLoggedUserMentor()}">
    <p:submenu label="Mentor tools" icon="ui-icon-wrench"
        ...
```

Obrázek 36: Restartovací commandButton #default-menu.xhtml.

Nejsofistikovanějším view projektu je *quiz-creation.xhtml*. Na tomto místě je mentorům umožněno vytvářet libovolně dlouhé kvízy. Formulář pro otázky a odpovědi je generován dynamicky s využitím Ajaxu. Bylo zde třeba vyřešit logiku přepočítávání odpovědí a otázek, zachování posloupnosti, vlastnictví a určení správnosti zvolené odpovědi. To vše bylo nakonec implementováno na straně view a stejně jako ostatní vstupy aplikace je i zde zajištěna validace.

Příklad validace u komponenty primefaces je k vidění na obrázku 37. Nejprve je určeno, že je hodnota vstupu povinná a poté je komponentě předložena hláška, která se zobrazí v případě nedodržení této potřeby.

```
<tr>
    <td><p:inputText placeholder="Full name" required="true"
            requiredMessage="#{msgs['registration.error.fullName']}"
            value="#{userController.user.fullName}" /></td>
</tr>
```

Obrázek 37: Příklad validace PF komponenty #registration.xhtml.

Validace na front-endu jsou velmi výhodné. Například v tomto případě by byla neoprávněná registrace zamítnuta dříve, než by ji musel začít vyhodnocovat server.

4.6. Services

Na této vrstvě se nachází aplikační a business logika. Tím, že jsou toho ušetřeny controllery, je zajištěná lepší čitelnost kódu a jednodušší detekce chyb. Jedná se o CDI beany, do kterých jsou injectovány potřebné controllery a DAO. Tyto třídy obsahují kromě accessor metod k vlastněným proměnným velmi důležité metody pro aplikaci. Existují dvě hlavní třídy:

- `EvaluationService` – Stará se kompletně o celé vyhodnocování postupu uživatele a ukládá evaluaci do databáze.
- `QuizCreationService` – Jedná se o *session scoped* bean, který zajišťuje dynamické generování view při vytváření kvízu. Dále obsahuje metody potřebné k uložení nového kvízu do databáze.

4.7. Utility

Utility jsou třídy, jejichž úkolem je nějakým způsobem pomáhat při vývoji, provozu, nebo chodu aplikace. Ve zmiňovaném projektu existují tři takové utility.

`utils/security/ShiroLoginBean` – Jedná se o CDI bean s anotací *viewScoped*. Jak už její název napovídá, jedná se o třídu zajišťující login a logout uživatelů. Přihlášený uživatel, neboli subject, je ukládán i vyvoláván z kontextu třídy `org.apache.shiro.SecurityUtils`. Potřebná nastavení autentizace a autorizace jsou provedena v souboru *WEB-INF/shiro.ini*. Zavedení filtru do aplikace se provádí v souboru *WEB-INF/web.xml*. Při nastavování tohoto frameworku autor vycházel z přehledně psané dokumentace [18]. Ačkoliv se tato problematika může na první pohled zdát jako obtížná, celá konfigurace nezabrala více než 15 řádků. Jedná se o velmi dobře a intuitivně navržený framework.

Provedenou konfigurací bylo docíleno toho, že se uživatel nedostane k žádné jiné stránce než je *login.xhtml*, pokud aktuální session neobsahuje informace o správně provedeném přihlášení. Tato session navíc udržuje v konverzačním stavu referenci na objekt aktuálně přihlášeného uživatele. Konverzační stav končí odhlášením nebo timeoutem.

utils/JsfUtils – Zde je implementována statická metoda, která vyhledá request parametr s požadovaným ID a vrátí jeho hodnotu v textové podobě. Toho je využíváno především na controllerech kde je potřeba vkládat do DAO primární klíče hledaných záznamů. Zmiňované PK je však nutné přetypovat na datový typ *long*, což je také řešeno v těle této třídy. Zmiňovaná metoda je vidět na obrázku 38.

```
public static String getRequestParameter(String paramName) {
    String param = FacesContext.getCurrentInstance().getExternalContext()
            .getRequestParameterMap().get(paramName);
    if (param == null) {
        return null;
    }
    return param;
}
```

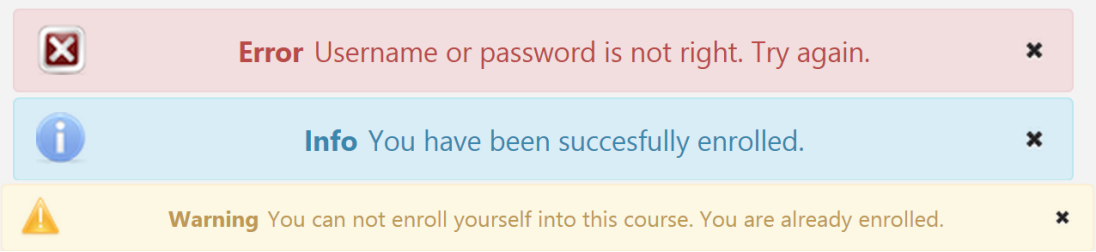
Obrázek 38: Statická metoda JsfUtils#getRequestParameter

utils/MessageUtils – Tato třída umožňuje jednodušší vytváření informačních zpráv, viz obrázek 39. Jejím zavedením byla odstraněna redundance kódu. `FacesMessage` rozlišuje tři typy zpráv. Jsou jimi INFO, WARN a ERROR.

```
public static void addInfoMessage(String message) {
    FacesContext.getCurrentInstance().addMessage(null,
            new FacesMessage(FacesMessage.SEVERITY_INFO, "Info", message));
}
```

Obrázek 39:Tvorba INFO message MessageUtils#addInfoMessage

Konkrétní zobrazení všech tří typů informačních zpráv v UI aplikace je vidět na obrázku číslo 40. Zprávy jsou většinou vypisovány s využitím JSF Ajax frameworku.



Obrázek 40: Příklady zobrazení zpráv uživateli.

4.8. Testování

Aplikace byla testována v průběhu vývoje samotným vývojářem. Po implementaci každé funkcionality byl vždy zapnut *debugovací mód* a provedeny testovací scénáře. Všechny odhalené nedostatky byly ihned odstraněny a v případě, že se žádné další neobjevily, implementace pokračovala.

Nad aplikací byly také vytvořeny Unit testy. Pro tento typ projektu se však jejich použití úplně nehodí, proto jich bylo vytvořeno jen několik a to na místech, kde mají smysl. Verze *junit* dependence v Mavenu je 4.11.

Největší nedostatek byl odhalen při provozních uživatelských testech. Aplikace byla spuštěna pro více uživatelů. Konkrétně se jednalo o čtyři osoby, které prováděli ty samé úkony ve stejný čas. Při tomto *multiuser* testu byla zjištěna špatná vazba na modelu, kdy při vyhodnocování testu došlo k přerušení chodu aplikace výjimkou. Vazba mezi evaluacemi a na ně se vázajícími špatně zvolenými odpověďmi byla 1:n. Při zvolení té samé špatné odpovědi u více uživatelů naráz je samozřejmě vyžadována vazba m:n. Model byl upraven, mezi entitami byla vytvořena asociační tabulka a tím byl problém odstraněn. Žádné další chyby zjištěny nebyly.

Vzhledem k tomu, že zadáním práce bylo navržení a implementace jádra aplikace bez požadavku na okamžité nasazení do provozu, nebyly prováděny například zátěžové testy a testy podobného charakteru.

5. Závěr

Na základě cílů práce byly provedeny následující kroky:

1. Analýza problematiky e-learningu, požadavků na aplikaci a následný návrh její funkcionality formou Use-case diagramu, který je současně viditelným vymezením možností a hranic vyvíjené aplikace.
2. Výběr technologií, které nejlépe doplňovaly předem stanovené technologické jádro poskytnuté platformou Java Enterprise Edition (JSF, EJB, JPA). Výběr vhodného aplikačního serveru (JBoss WildFly) a konfigurace vývojového prostředí Eclipse. Následovalo založení projektu a integrace vybraných technologií pomocí Apache Mavenu. Integrovány nakonec byly výše zmíněné Java EE technologie + PrimeFaces, Apache Shiro a SLF4J.
3. Realizace vývoje jádra aplikace, které bylo v rámci implementace navíc doplněno o další záležitosti přispívající k ucelenosti a použitelnosti výsledné aplikace. Při implementaci byly plně využity všechny požadované technologie. Otestování aplikace odhalilo, že stav, ve kterém se aplikace nachází, umožňuje její nasazení jakožto prototypu určeného pro lokální testování uživatelů. Příkladem může být testování studentů, kdy učitel založí v aplikaci kurz, vytvoří test a stanoví požadavky. Studenti se do aplikace zaregistrují a následně se zapíší do kurzu. Studenti absolvují pod dohledem učitele test a nechají aplikaci vytvořit evaluaci. Učitel si následně může zobrazit statistiky konkrétního kurzu, kde uvidí evaluace všech uživatelů. Takový proces by mohl programátorovi přinést cennou zpětnou vazbu od samotných uživatelů a dát mu tak potřebné směrnice dalšího rozvoje vedoucího k vylepšení aplikace.
4. Proces přípravy, vytvoření a konfigurace projektu, který využívá zvolené technologie, byl dosti obtížný. To se ovšem vzhledem k robustnosti Java EE platformy dalo předpokládat. Z tohoto důvodu lze potvrdit fakt, že se tato platforma hodí především na tvorbu rozsáhlejších projektů. Se zavedením jednotlivých technologií do projektu a s jejich následným propojení velmi pomohla správa závislostí konfigurovaná Apache Mavenem.

Vývoj probíhal plynule a veškeré součásti aplikace zachovávaly i při narůstající složitosti transparentnost a flexibilitu. Vybrané technologie, kterými byly řešeny jednotlivé vrstvy aplikace, se v implementační části prokázaly jako velice dobře zvolené. Vlastnosti, úlohy a výhody těchto technologií jsou podrobněji popsány v předchozích kapitolách (v Technologické a Implementační). Aplikace je v současném stavu navíc velmi dobře připravena reagovat na potřebu škálování svých vrstev na více fyzických uzlů, čímž by došlo k rozdělení zátěže a umožnění oddělené správy.

Zkušenosti získané při vývoji této bakalářské práce za využití Javy EE chtěl autor využít při vstupu do praxe. Ten mu byl přibližně v polovině vývojového procesu umožněn. Cíl lze tedy rovněž považovat za splněný.

5.1. Možnosti rozšíření aplikace

Aplikace se nachází ve stavu, ze kterého ji lze rozšiřovat několika směry. Jedním z nich by mohlo být rozšíření možnosti testování o možnost vzdělávání. V aplikaci již existují kurzy, které by kromě kvízů mohly navíc zastřešovat studijní materiály nebo stránky psané autorem kurzu.

Uvažovaná plná verze by také mohla obsahovat třetího aktéra, kterým by byl *System Administrator*. Ten by měl na starost správu uživatelských účtů a obsahovou část sdíleného webového rozhraní aplikace.

Potřebným rozšířením je také bezpečnostní otázka. V úvahu připadá například spouštění testů po předložení generovaného klíče, otestování nestandartních scénářů a tak podobně.

Největších výsledků v oblasti rozšiřování by však bylo dosaženo nasazením aplikace do vhodného provozního prostředí, kde by na tuto problematiku mohli reagovat samotní uživatelé a jejich individuální potřeby. Analýza jejich připomínek by vývojáři odhalila rizika a nedostatky, které je třeba zpracovat do další verze aplikace.

6. Seznam použité literatury

[1] Programming Language Popularity. Langpop [online]. 2013 [cit. 2015-04-17]. Dostupné z: http://langpop.com/

[2] Top 10 e-Learning Statistics for 2014 You Need To Know. ELearning industry [online]. 2014 [cit. 2015-04-17]. Dostupné z: http://elearningindustry.com/top-10-e-learning-statistics-for-2014-you-need-to-know

[3] Programming languages used in most popular websites. Wikipedia [online]. 2014 [cit. 2015-04-17]. Dostupné z: http://en.wikipedia.org/wiki/Programming_languages_used_in_most_popular_websites

[4] DEVELOPER PRODUCTIVITY REPORT. Zeroturnaround: rebellabs [online]. 2012 [cit. 2015-04-17]. Dostupné z: http://zeroturnaround.com/rebellabs/download/?token=abe74eedfe59be0d4482bc6c1b270f139f39cb07

[5] About the Eclipse. Eclipse [online]. 2014 [cit. 2015-04-17]. Dostupné z: https://eclipse.org/org/

[6] Getting Started Guide. JBossDeveloper [online]. 2014 [cit. 2015-04-17]. Dostupné z: https://docs.jboss.org/author/display/WFLY8/Getting+Started+Guide?_sscc=t

[7] H2 Database Engine. H2database [online]. 2014 [cit. 2015-04-17]. Dostupné z: http://www.h2database.com/html/main.html

[8] Java Platform, Enterprise Edition: The Java EE Tutorial. Oracle Java doc [online]. 2014 [cit. 2015-04-17]. Dostupné z: https://docs.oracle.com/javaee/7/tutorial/

[9] Java EE at a Glance. Oracle [online]. 2015 [cit. 2015-04-17]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/overview/index.html

[10] Interface EntityManager. Java EE 7 Specification API [online]. 2014 [cit. 2015-04-17]. Dostupné z: http://docs.oracle.com/javaee/7/api/javax/persistence/EntityManager.html

[11] Why PrimeFace. Primefaces [online]. 2014 [cit. 2015-04-17]. Dostupné z: http://www.primefaces.org/whyprimefaces

[12] Maven's Objectives. Apache Maven Project [online]. 2014 [cit. 2015-04-17]. Dostupné z: https://maven.apache.org/what-is-maven.html

[13] Introduction to the POM. Apache Maven Project [online]. 2014 [cit. 2015-04-17]. Dostupné z: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

[14] Welcome to Apache Shiro. Apache Shiro [online]. 2014 [cit. 2015-04-17]. Dostupné z: http://shiro.apache.org/

[15] Simple Logging Facade for Java. SLF4J [online]. 2014 [cit. 2015-04-17]. Dostupné z: http://www.slf4j.org/

[16] MySQL Workbench. MySQL [online]. 2015 [cit. 2015-04-17]. Dostupné z: https://www.mysql.com/products/workbench/

[17] PrimeFaces Showcase. Primefaces [online]. 2015 [cit. 2015-04-17]. Dostupné z: http://www.primefaces.org/showcase/

[18] Get Started with Apache Shiro. Apache Shiro [online]. 2014 [cit. 2015-04-17]. Dostupné z: http://shiro.apache.org/get-started.html

7. Obrázky

8. Přílohy

8.1. Instalační příručka

Vzhledem k tomu, že bylo využito embedded databáze poskytované aplikačním serverem, jak již bylo řečeno v **požadavcích na implementaci**, není zprovoznění aplikace vůbec obtížné. Vše je nakonfigurováno, automatizováno a připraveno k použití. Ke spuštění aplikace s defaultní in-memory databází jsou zapotřebí:

- WAR – webový archiv aplikace
- Aplikační server – JBoss WildFly verze 8

Po instalaci aplikačního serveru se provede klasický deployment webového archivu a uživatel se může přihlásit / registrovat. Pokud systém využívá defaultního nastavení, je aplikace dostupná na adrese ***localhost:8080/testsoft***.

Na ostrý provoz aplikace je přísně doporučeno využít klasického databázového serveru. V případě embedded databáze totiž nedojde k trvalému uložení dat. Při každém restartu aplikace dojde k jejich smazání a opětovnému nahrání testovacích dat ze souboru *init/DatabasePopulator*.

8.2. Uživatelská příručka

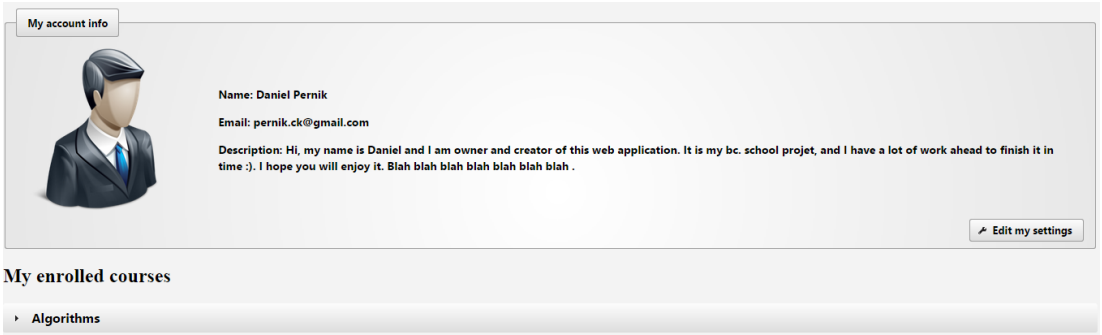
Před vstupem do hlavního rozhraní aplikace je nutné se přihlásit. V případě, že uživatel ještě nemá účet, klikne na *create an account*, čímž dojde k přesměrování na registrační formulář.

Obrázek 1: Přihlašovací formulář

V aplikaci existují dva action bary. První z nich je v horní části aplikace. Poskytuje informaci o aktuálně přihlášeném uživateli a možnost odhlášení. Druhý bar slouží k navigaci a nachází se pod logem aplikace.

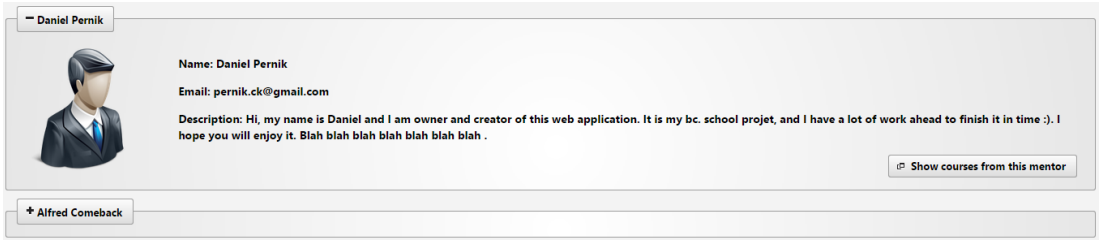
Obrázek 2: Navigační menu

Stránky *Homepage* a *About Project* jsou pouze informační a obsahují jen statický obsah. Na stránce my account nalezne uživatel informace o svém profilu a také kurzy, do kterých je zapsaný.



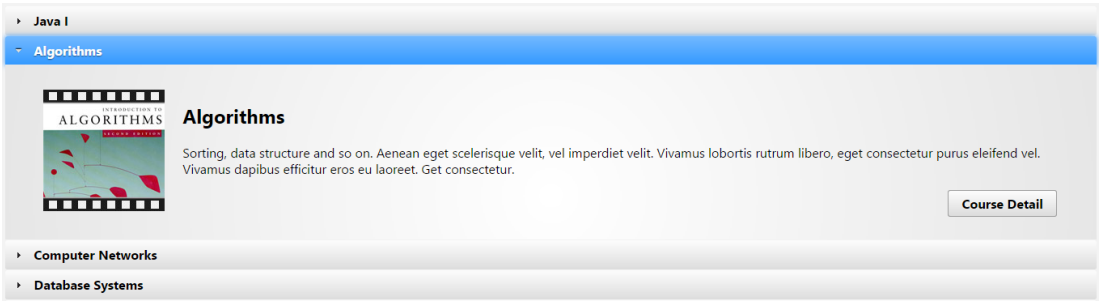
Obrázek 3: Stránka My account

Na stránce *Users* lze zobrazit všechny uživatele a mentory evidované v aplikaci. Je to řešeno efektní formou dropdown seznamu. V seznamu mentorů lze přejít ke kurzům konkrétního mentora.



Obrázek 4:Stránka Users > Mentors

Pod položkou *Courses* se nachází seznam všech existujících kurzů, nebo seznam všech kurzů, zapsaných aktuálně přihlášeným uživatelem. Ty jsou v kontextu designu řešeny odlišným dropdown seznamem, jak lze vidět na obrázku číslo 5 na následující straně příručky.

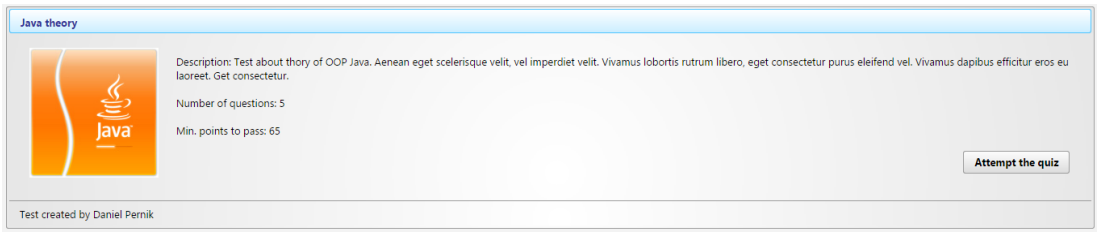


Obrázek 5:Stránka Courses

Z jednotlivých kurzů jde dále přistoupit na jejich detail. V detailu kurzu se může uživatel zapsat, nebo si třeba zobrazit uživatele zapsané do tohoto kurzu.

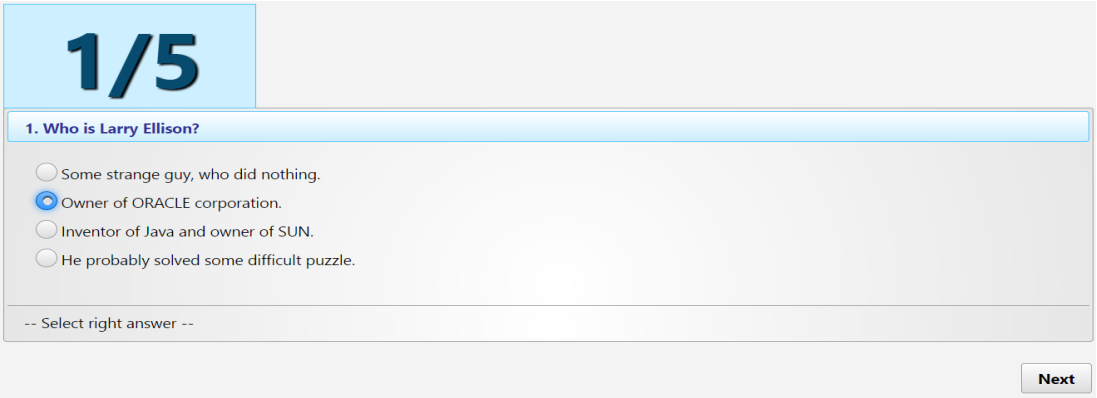
Obrázek 6:Stránka Course detail

Po zaspání se do kurzu může uživatel v menu *Courses* > *My enolled courses* přistoupit k *view* kurzu, ve kterém jsou kromě informací také kvízy vztahující se k danému kurzu. Podoba Kvízu je k vidění na obrázku číslo 7.



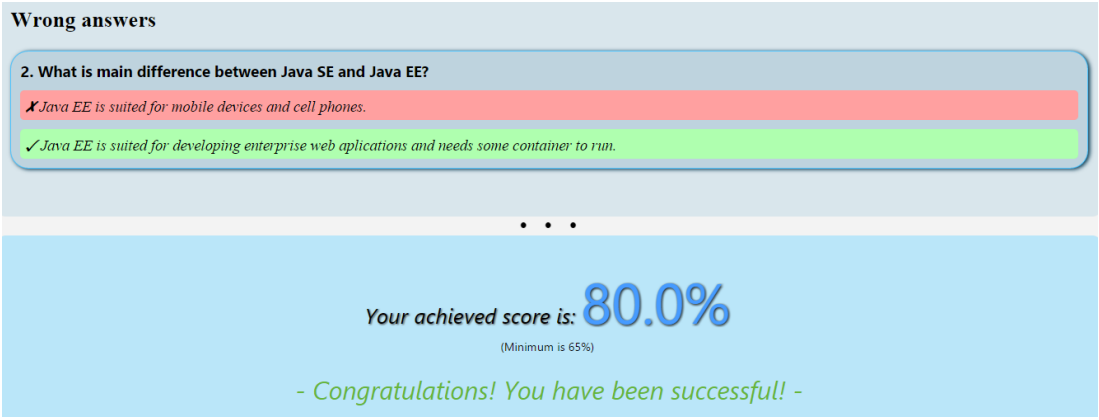
Obrázek 7: Kvíz Java theory

Kvízy se vždy skládají z otázek, kdy každé otázce náleží 4 odpovědi a právě jedna je správná. Uživatel se k předchozím otázkám nemůže vracet, a proto by měl vždy pečlivě zvážit svou odpověď. Podoba kvízové otázky je vidět na obrázku číslo 8 následující stránky.



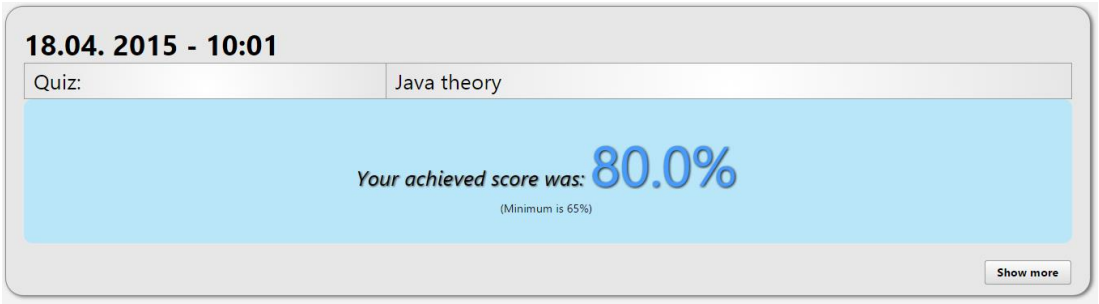
Obrázek 8: Kvíz Java theory > první otázka

Následuje vyhodnocení kvízu. To obsahuje informace o uživateli, který jej vykonával, datum a čas, název kvízu, ale hlavně špatně zodpovězené otázky a samotné vyhodnocení, viz obrázek číslo 9.



Obrázek 9: Vyhodnocení kvízu Java theory

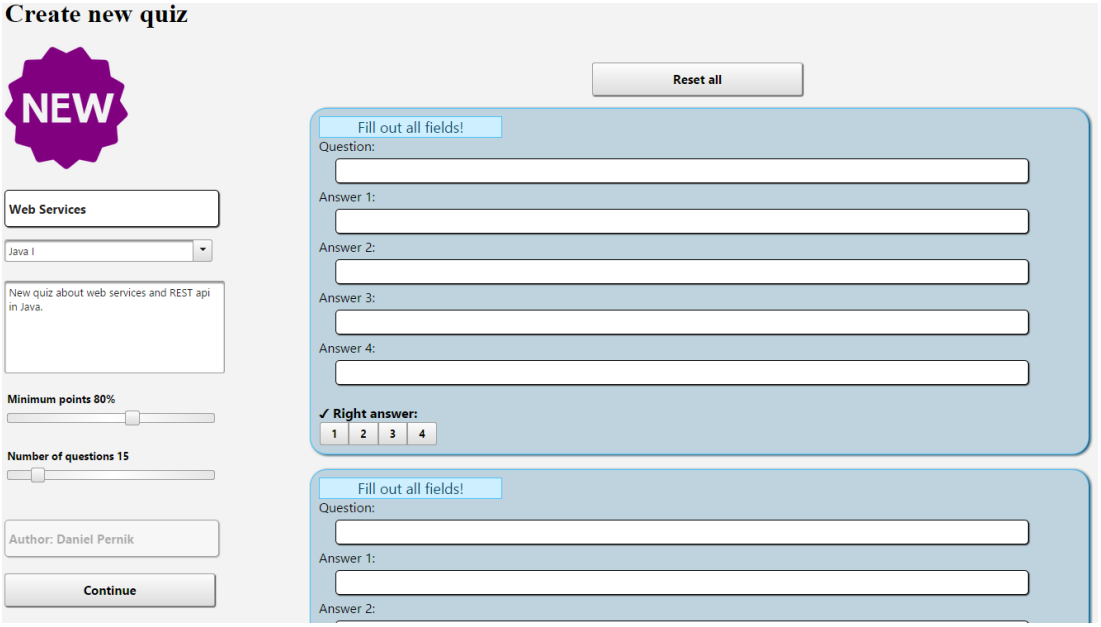
Ke svým evaluacím se může uživatel vracet v položce *My achievements*. Tam může dále pokračovat tlačítkem *Show more* pro zobrazení více detailů.



Obrázek 10: Zobrazení evaluací na stránce My achievements

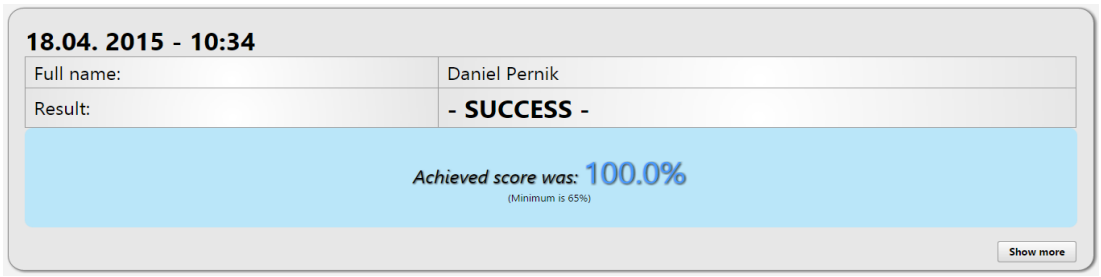
Položka *Mentor tools* je zobrazena pouze v případě, že je přihlášeným uživatelem mentor. Mohou zde být vytvořeny nový kurz a kvíz, nebo zde mohou být zobrazeny statistiky konkrétního vlastněného kvízu.

Pro vytvoření kurzu je připravený formulář s předvyplněným autorem. Podléhá validacím a po úspěšném vytvoření je uživatel přesměrován na jeho detail. Vytvoření kvízu je o něco složitější. Uživatel vyplní základní údaje a stiskne tlačítko *Continue*. Tím dojde k dynamickému vygenerování formulářů pro jednotlivé otázky a jejich odpovědi.



Obrázek 11: Vytváření nového kvízu

K zobrazení statistik je zapotřebí si nejprve zvolit kvíz, kterého se statistiky týkají. Poté mentor klikne na tlačítko *Show evaluations* a je přesměrován. Zobrazení jednotlivých výsledků je vidět na obrázku číslo 12.



Obrázek 12: Zobrazení výsledku konkrétního studenta

8.3. Disk (CD)

Obsah disku

- Text bakalářské práce (PDF)
- Build aplikace (WAR soubor)
- Eclipse project (zdrojové kódy + dokumentace)
- README.txt (popis struktury disku)