Mendelova univerzita v Brně
Provozně ekonomická fakulta

Webová aplikácia pre podporu adaptívneho testovania

Bakalárska práca

Vedúci práce:

Ing. Jan Kolomazník

Ľuboš Maxina

Brno 2015

Rád by som sa poďakoval vedúcemu mojej bakalárskej práce, pánovi Janovi Kolomazníkovi, za odborné rady a cenné pripomienky, a taktiež za ochotu a čas, ktorý mi venoval pri vypracovávaní tejto práce.

Čestné prehlásenie

Prehlasujem, že som túto prácu: **Webová aplikácia pre podporu adaptívneho testovania,** vypracoval samostatne a všetky použité pramene a informácie sú uvedené v zozname použitej literatúry. Súhlasím, aby moja práca bola zverejnená v súlade s §47b zákona č. 111/1998 Sb., o vysokých školách v znení neskorších predpisov, a v súlade s platnou *Smernicou o zverejňovaní vysokoškolských záverečných prác*.

Som si vedomý, že sa na moju prácu vzťahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brne má právo na uzatvorenie licenčnej zmluvy a použitie tejto práce ako školského diela podľa §60 odst. 1 autorského zákona.

Ďalej sa zaväzujem, že pred spísaním licenčnej zmluvy o použití diela inou osobou (subjektom) si vyžiadam písomné stanovisko univerzity o tom, že predmetná licenčná zmluva nie je v rozpore s oprávnenými záujmami univerzity, a zaväzujem sa uhradiť prípadný príspevok na úhradu nákladov spojených so vznikom diela, a to až do ich skutočnej výšky.

V Brne dňa 5. januára 2015 _____

Abstract

Maxina, Ľ. Web application with adaptive testing support. Bachelor thesis. Brno: Mendel University in Brno, 2015.

This thesis focuses on the implementation of application for adaptive testing. The main objective is to implement application with ability to create relations between exam items. This application will allow basic operations for exam and user account management.

Keywords

Adaptive testing, Spring Framework, Hibernate

Abstrakt

Maxina, Ľ. Webová aplikácia pre podporu adaptívneho testovania. Bakalárska práca. Brno: Mendelova univerzita v Brně, 2015.

Táto práca je zameraná na implementáciu aplikácie s podporou adaptívneho testovania. Cieľ práce je vytvoriť aplikáciu, ktorá umožňuje vytvárať vzťahy medzi jednotlivými položkami testu. Aplikácia bude umožňovať základné operácie pre správu testov a používateľského účtu.

Kľúčové slová

Adaptívne testovanie, Spring Framework, Hibernate

Obsah

Zoznam tabuliek

Zoznam obrázkov

Zoznam použitých skratiek

AJAX – Asynchronous JavaScript and XML

CAT – Computer Adaptive Testing

CRUD – operácie Create, Read, Update, Delete

CSS – kaskádové štýly

DAO – Data Access Object

DI – Dependency Injection

ERD – Entity Relational Diagram

HQL – Hibernate Query Language

HTML – HyperText Markup Language

IoC – Inversion of Controll

IRT – Item Response Teory

J2EE – Java Enterprise Edition

JPA – Java Persistance API

JRE – Java Runtime Enviroment

JSP – Java Server Pages

JVM – Java Virtual Machine

MVC – Model View Controller

ORM – Object Relational Mapping

POJO – Plain Old Java Object

SQL – Structured Query Language

UML – Unified Modeling Language

XML – Extensible Markup Language

1 ÚVOD

V súčasnosti sa s testami nestretávame len na školách, kurzoch alebo pri inej vzdelávacej činnosti. Testovanie vedomostí sa stáva stále bežnejším nástrojom používaným napr. pri pracovných pohovoroch. Klasický test s preddefinovanými otázkami má mnoho nedostatkov, ktoré môžu skresliť výsledok testu. Medzi hlavný nedostatok klasického testovania patria položky testu, ktoré sú nesystematicky zvolené. Môžu medzi nimi existovať vzťahy, vďaka čomu niektorí testovaní odpovedajú správne, aj keď nemajú požadovanú vedomosti. V neposlednom rade patrí medzi faktory skresľujúce výsledok, aj stres spôsobený obmedzeným časom na vypracovanie alebo aj hádanie možností.

Nedostatky klasického testovania odstraňuje do určitej miery adaptívny test. Tento test je založený na odhadovaní schopnosti testovaného prispôsobovaním úrovne náročnosti otázok. Medzi najväčšie výhody adaptívnych testov patrí ich rýchlosť. Adaptívny test je označovaný ako „šitý na mieru“.ˇ

2 CIEĽ PRÁCE A METODIKA

2.1 Cieľ práce

Cieľ práce je vytvoriť aplikáciu, ktorá umožňuje vytvárať vzťahy medzi jednotlivými položkami adaptívneho testu. Podobne Aplikácia bude umožňovať základné operácie pre správu testov a používateľského účtu.

2.2 Metodika

2.2.1 Analýza

Na základe preštudovania teórie adaptívnych testov budú zvolené vhodná technológie pre implementáciu aplikácie s podporou testovania vedomostí. Ďalej bude spracovaná teória zvolenéných technológii, na základe ktorej bude aplikácia implementovaná.

2.2.2 Implementácia a dokumentácia

Implementácia aplikácie realizovaná v tejto práci, bude priblížená v príslušných kapitolách. Následne bude spísaná dokumentácia práce s aplikáciou.

2.2.3 Zhodnotenie riešenia

Implementované riešenie bude v závere práce zhodnotené spolu z návrhom možných vylepšení aplikácie do budúcnosti.

3 ADAPTÍVNE TESTOVANIE ZNALOSTÍ

Adaptívny test na rozdiel od klasického nemá pevne stanovený počet, ktoré budú testovanému položené. Počet položiek v adaptívnom teste je závislý na odpovediach, podľa ktorých sa test snaží prispôsobiť schopnostiam testovaného. Adaptívny test za ideálnych podmienok skončí, až keď je naplnené ukončovacie kritérium. Položková banka musí byť dostatočne veľká, aby neboli vyčerpané všetky položky skôr ako bude určená úroveň vedomostí. (Weiss, 2004).

Prvý adaptívny test vytvoril Alfred Binet v roku 1905, ktorý sa písal na papier. Zvolil otázky, na ktoré odpovedali testovaný približne s 50% úspešnosťou a zoradil ich podľa náročnosti do úrovní. Počiatočnú úroveň otázok odhadoval. Test skončil, keď našiel skupiny, v ktorej testovaný odpovedal na všetky položky správne a skupinu, kde odpovedal na všetky položky nesprávne. (International Association for Computerized Adaptive Testing, 2015)

Adaptívne testovanie meria úroveň vedomostí testovaného na rozdiel od klasického testovania, ktoré meria úspešnosť odpovedí. Tento proces je podobný ako pri ústnom preskúšaní, kde je však vyžadovaná účasť vyučujúceho, ktorý sa snaží zistiť úroveň vedomostí skúšaného. Takýto proces je síce dostatočný pre určenie vedomostí skúšaného, ale vyžaduje obrovské množstvo času už pri niekoľkých študentoch. Tento problém rieši CAT, ktorý nahradí skúšajúceho výpočtami počítača. (Weiss, 2004).

3.1 CAT

Počítačové adaptívne testovanie (CAT) je založené na meraní schopností testovaného s využitím psychologických prístupov. CAT vyžaduje veľkú predkalibrovanú položkovú banku, pretože presnosť meraní vedomostí môže byť skreslená pri nevhodne zvolených položkách alebo sa predĺži dĺžka testu. Plne adaptívny CAT by mal umožňovať zvoliť počiatočnú úroveň, využíva určitú definovanú hodnotiacu funkciu, existuje pravidlo, ktoré určuje ďalšiu generovanú položku a tiež existuje kritérium pre ukončenie testu. (Weiss, 2004).

V súčasnosti je teória odpovedi na položku (IRT) jednou z najpoužívanejších teórii v CAT. Využíva informačnú funkciu založenú na bayesovskej pravdepodobnosti, vďaka ktorej určuje skóre testovaného. IRT používa jeden až tri parametre (obtiažnosť, diskriminačný parameter a parameter uhádnuteľnosti) pre výpočet informačnej funkcie, podľa počtu parametrov sa jedná o jednoparametrový, dvojparametrový alebo trojparametrový model. (Weiss, 2004).

3.2 Pojmové mapy

Pojmová mapa ako nástroj pre tvorbu adaptívnych testov, zobrazuje položky testu v grafovej štruktúre. Existuje počiatočný uzol, na ktorom sú závislé všetky ostatné uzly testu. Obsahuje základné vedomosti potrebné pre daný typ testovaných vedomostí. Položky sú prepojené orientovanými hranami, ktoré zobrazujú vzťah medzi uzlami.

Adaptívnosť je zabezpečená na základe prepojení uzlov testu. Uzol z hlbšej vrstvy vyžadaje úspešné splnenie jeho naduzlov, inak položka nebude testovanému ponúknutá rovnako ako jej ďaľšie poduzly. (Dlabolová, Rybička, 2013).

4 VÝBER TECHNOLÓGII

Požiadavky pre implementáciu aplikácie vyžadujú výber vhodných technológii. Aplikácia je určená k testovaniu znalostí mnohých používateľov, preto sa bude jednať o webovú aplikáciu. Technológii pre implementáciu webovej aplikácie je viacero, preto treba zobrať v úvahu aj ďaľšie kritéria. Medzi ďaľšie požiadavky by rozhodne mali patriť prehľadnosť kódu, znovupoužiteľnosť kódu, jednoduchá rozšíriteľnosť a tiež platformová nezávislosť. Z dôvodu rozvrstvenia aplikácie a prehľadnejšieho kódu bude aplikácia realizovaná v návrhovom vzore MVC.

Tieto požiadavky spĺňa Spring Framework, ktorý je založený na programovaciom jazyku Java. Jeho súčasťou je Spring Web MVC, ktorý je určený k vytváraniu webových aplikácii podľa návrhového vzoru MVC.

4.1 Java

Java je programovací jazyk vyvinutý firmou Sun Microsystems. V súčasnosti je vlastníkom tohto programovacieho jazyka firma Oracle. Oracle sa stal novým vlastníkom po kúpení firmy Sun Microsystems v roku 2010.

Java je nezávislá na platforme vďaka JVM, ktorý prekladá Java kód do strojového kódu. Preto je Java ľahko prenositeľná medzi rôznymi platformami a zariadeniami. V súčasnosti je používaná viac ako 9 miliónmi vývojármi po celom svete.

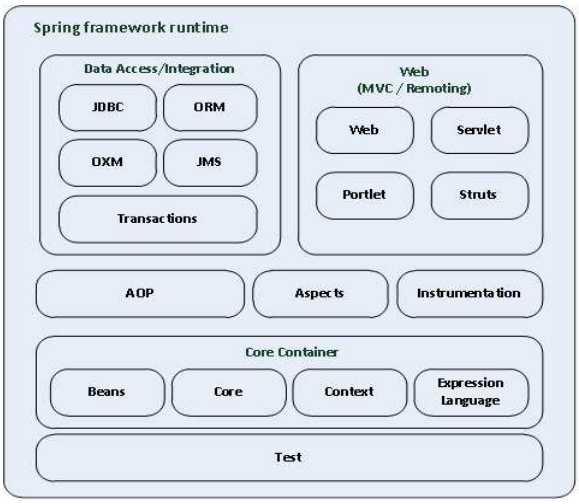
4.2 Spring Framework

Spring Framework je otvorený voľne šíriteľný aplikačný framework vyvinutý v roku 2002, ktorý je určený predovšetkým pre zjednodušenie vývoja J2EE aplikácii. Má viacero modulov a služieb postavených na IoC kontajnery, vďaka čomu poskytuje vysokú mieru abstrakcie pri vývoji aplikácii. (Walls, 2011)

Hlavná črta tohto aplikačného rámca je IoC. Vďaka IoC je konfigurácia jednoduchšia a konzistentnejšia. Je to koncepcia založená na princípe „nevolaj mi, ja ti zavolám“, kde objekty namiesto volania svojích požadovaných závislostí priamo dostanú potrebné závislosti od kontajneru. V Spring Frameworku sa táto koncepcia nazýva Dependency Injection (DI). Vďaka DI je oddelená konfigurácia od služieb a je ľahko modifikovateľná. (Johnson, 2005)

Zostavovanie aplikácie je spravované nástrojom Apache Maven. Okrem zostavovania aplikácie umožňuje spravovať závislosti z jedného miesta tzv. Project Object Model (súbor pom.xml), preto je migrovanie na nové technológie jednoduché. (Apache Maven Project, c2001)

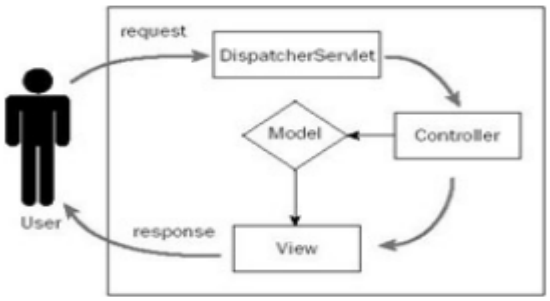
Architektúra Spring Frameworku je rozdelená do viacerých modulov.



Obr. 1 Spring Framework moduly (zdroj: http://javacodebook.com/ (2013))

4.2.1 Spring Web MVC

Spring Web MVC je implementácia Spring Frameworku pre webové aplikácie napísané v návrhovom vzore MVC. Základné typy objektov sú Model, View a Controller. DispatcherServlet je vpodstate dispečér webových požiadaviek, ktorý má za úlohu priradiť k požiadavke Controller, s ktorého prevezme údaje o pohľade a modely, a následne sa postará o vykreslenie odpovede. (Johnson, 2005)



Obr. 2 Spracovanie požiadavku (JOHNSON, 2005)

4.3 Apache Tomcat

Apache Tomcat je otvorený voľne šíriteľný web server a servlet[1] kontajner vyvinutý organizáciou Apache Software Foundation. Umožňuje spúšťať servlety a JSP aplikácie. (Apache, c1999)

Aplikácie bola testovaná na servlet kontajnery Apache Tomcat 7.0.

[1] Servlet je program v jazyku Java, ktorý spracováva požiadavky pre aplikáciu od webového serveru.

4.4 PostgreSQL

PostgreSQL je otvorený voľne šíriteľný objektovo-relačný databázový systém od firmy PostgreSQL Global Development Group. Podporuje všetky bežné operačné systémy t.j. Linux, Unix (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), Windows. Okrem toho má zabudovanú natívnu podporu rozhraní pre programovacie jazyky (C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC a iné). PostgreSQL vyhrala niekoľko odcenení od používateľov aj z podnikovej oblasti. (PostgreSQL, 2015)

4.5 Hibernate ORM

Hibernate ORM[2] je framework od firmy Red Hat určený pre generovanie databáze priamo z kódu aplikácie mapovaním tried na databázové entity. Okrem ušetrenia práce s návrhom databáze poskytuje aj vyšší výkon pri databázových operáciach. Umožňuje využiť okrem SQL aj HQL, ktoré poskytuje viac možností ako klasické SQL. Hibernate je možné detailne konfigurovať a rozširovať o ďalšie technológie.

V aplikácii je Hibernate ORM konfigurovaný cez JPA anotácie nad POJO triedami.

4.6 Apache Tiles

Apache Tiles je *open-source* šablónovací framework založený na skladaní vopred nadefinovaných fragmentov, vďaka čomu je možné znovu použiť kód a vytvoriť konzistentné používateľské prostredie. Tiles je ľahko konfigurovateľný cez XML definičný súbor. (Apache Tiles, c2001)

Implementovaná aplikácia používa Apache Tiles 3.0 a definuje fragmenty pre hlavičku, pätu a menu.

4.7 HTML, CSS a Javascript

Pri implementácii webovej aplikácie sa dá len ťažko zaobísť bez týchto technológii. V aplikácii sú okrem týchto technológii použité aj javascriptové knižnice jQuery a jQueryUI.

HTML je štrukturovací jazyk pre webové stránky, ktorý umožňuje vytvoriť nadpisy, tabuľky, zoznamy a pod. Tieto elementy sú vytvárané pomocou značkovacích prvkov. CSS je jazyk pre definovanie vzhľadu aplikácie, ktorý zahŕňa napr. farby, písmo, okraje a pod. (JAVASCRIPT WEB APIS, 2015)

Javascript je najbežnejší skriptovací jazyk. Umožňuje reagovať na udalosti vyvolané interakciou používateľa s webovou stránkou. Javascript umožňuje vytvárať dynamickejšie webové stránky. (W3C, 2015)

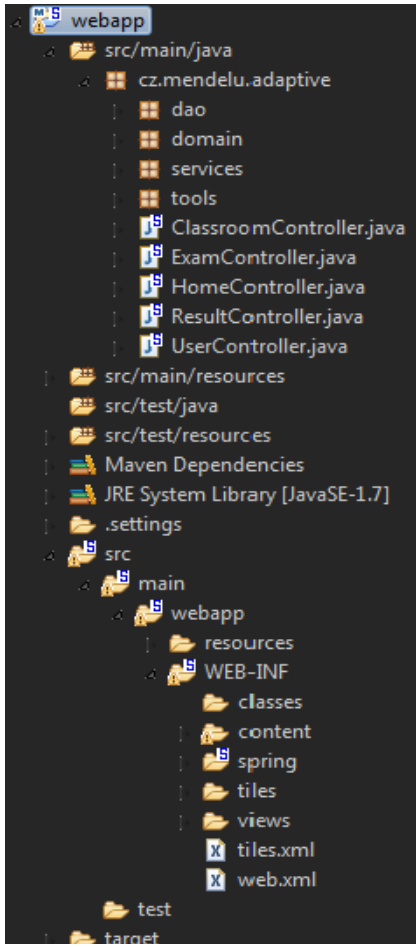
[2] ORM je technika, ktorá umožňuje mapovať objekty na databázové entity.

Knižnica jQuery zjednodušuje písanie javascriptu. Riadi sa heslom „Write less, do more.“, čo pomerne dobre vystihuje jej funkcionalitu. Využíva selektory, ktoré nahradzujú cykly pri manipulácii s elementami. Tiež prináša veľa nových funkcii pre animácie, udalosti, AJAX a pod.

Knižnica jQueryUI je nadstavbou jQuery, ktorá obsahuje preddefinované efekty, témy, prvky a pod.

5 ADRESÁROVÁ ŠTRUKTÚRA APLIKÁCIE

Štruktúra aplikácie sa skladá z niekoľkých adresárov, tak ako sú zobrazené na obrázku. Prvý adresár *„src/main/java“* obsahuje java triedy a rozhrania rozdelené do jednotlivých balíkov a podbalíkov. Ďalej sú v adresáry *„src/main/resources“* umiestné zdroje pre java triedy ako napr. konfiguračné súbory. Pre testovanie java kódu a zdroje k tomu určené, sú vyhradené adresáry *„src/test/java“* a *„src/test/resources“*.



Obr. 3 Štruktúra projektu

„Maven Dependencies“ obsahuje knižnice získané definovaním potrebných artifaktov v konfiguračnom súbore nástroja maven t.j. *„pom.xml“*. V *„JRE System library“* sú umiestnené knižnice pre JRE. Adresár *„src/main/webapp“* obsahuje dva podadresáre *„resources“* a *„WEB-INF“*.

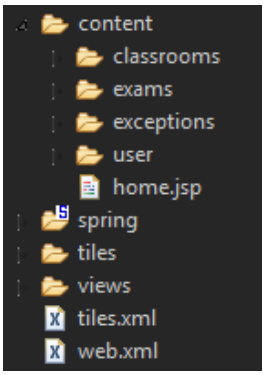
Prvý je určený pre webové zdroje, napr. javascript, kaskádové štýly, obrázky a iné. Druhý adresár obsahuje okrem pohľadov (JSP súborov) aj konfiguračné súbory xml. Adresár *„.settings“* je určený pre konfiguračné súbory komponent Spring frameworku. Posledný adresár *„target“* obsahuje zkompilované java triedy, ktoré sú vytvorené pri kompilovaní.

5.1 Štruktúra koreňového balíku

Koreňový balík *„cz.mendelu.adaptive“* obsahuje niekoľko podbalíkov a triedy reprezentujúce *controller*. V podbalíku *„dao“* sa nachádzajú DAO rozhrania a ich implementácie sú v ďalšom podbalíku *„impl“*. Triedy reprezentujúce doménový model sú umiestnené v podbalíku *„domain“*. Balík *„services“* je určený pre služby aplikácie. Pre nástroje ako implementácia emailového klienta alebo repozitár písaných testov slúži balík *„tools“*.

5.2 Štruktúra adresára „WEB-INF“

V koreňovom adresáry sú konfiguračné súbory pre Apache Tiles a Spring. Obsah webových stránok je uložený v adresáry *„content“* a je rozdelený do jednotlivých kategórii. Šablony meny, hlavičky a päty sú v adresáry *„tiles“*. Pohľady v adresáry *„views“* odkazujú na konkrétny webový obsah. Posledný adresár *„spring“* obsahuje konfiguračné súbory kontextu aplikácie.



Obr. 4 WEB-INF štruktúra

5.3 Štruktúra závislostí

Na správu projektu je použitý nástroj Maven. Aby bolo možné použiť ďalšie nástroje ako napr. Hibernate alebo vybraný databázový server je treba najprv defenovať závisloti pre Maven. Závislosti sú definované v súbore *pom.xml.*

Tab. 1 POM definície pre Maven

Význam závislosti	XML definícia závislosti
Hibernate závislosti	<dependency> <groupId>org.hibernate</groupId> <artifactId>hibernate-core</artifactId> </dependency>
	<dependency> <groupId>org.hibernate</groupId> <artifactId>hibernate-validator</artifactId> </dependency>
	<dependency> <groupId>org.hibernate.javax.persistence</groupId> <artifactId>hibernate-jpa-2.0-api</artifactId> </dependency>
Závislosť pre PostgreSQL databázový server	<dependency> <groupId>org.postgresql</groupId> <artifactId>postgresql</artifactId> </dependency>
Spring ORM závislosť	<dependency> <groupId>org.springframework</groupId> <artifactId>spring-orm</artifactId> </dependency>
Apache Tiles závislosť	<dependency> <groupId>org.apache.tiles</groupId> <artifactId>tiles-extras</artifactId> </dependency>
Spring Security závislosti	<dependency> <groupId>org.springframework.security</groupId> <artifactId>spring-security-web</artifactId> </dependency>
	<dependency> <groupId>org.springframework.security</groupId> <artifactId>spring-security-config</artifactId> </dependency>
	<dependency> <groupId>org.springframework.security</groupId> <artifactId>spring-security-taglibs</artifactId> </dependency>
Závislosť pre validáciu objektov	<dependency> <groupId>javax.validation</groupId> <artifactId>validation-api</artifactId> </dependency>

6 DOMÉNOVÝ MODEL

Doménový model tvoria hlavne triedy používateľ, test a výsledky testu. Ostatné doménové triedy sú prevažne modelmi vzťahov medzi týmito triedami. Napríklad vzťah medzi používateľmi zobrazujú triedy, ktoré umožňujú zoskupiť určitú skupiny používateľov a umožniť im písať určité testy.

Všetky používateľské účty s vínimkou administrátora majú dostupné rovnaké funkcie. To znamená, že každý používateľsi môže vytvoriť vlasné testy a tiež písať verejné testy. Používatelia môžu vytvárať triedy a pridávať si do nich ďaľších používateľov. Triedy umožňujú zobraziť test len používateľom v danej triede.

Bolo potrebné vytvoriť test, ktorý obsahuje grafovú štruktúru kapitôl, s otázkami. Preto každá kapitola musí poznať svojho predka aj potomka, ale aj otázky patriace do danej kapitoly. Každá kapitola zároveň patrí práve jednému testu a každý test pozná svoju počiatočnú (koreňovú) kapitolu.

Tab. 2 Doménové triedy

Trieda	Implementuje
User	Používateľa
Profile	Používateľský profil
UserRoles	Bezpečnostné role používateľov
UserType	Profilový atribút typu používateľa
Classroom	Používateľské triedy
Exam	Test
ExamType	Testový atribút typ testu
GraphChapterNode	Testovú kapitolu
Question	Otázku v kapitole
Option	Možnosti otázky
ExamResult	Výsledok testu
ChapterResult	Výsledok kapitoly
Answer	Odpoveď na otázku

6.1 Implementácia testových kapitôl

Každá kapitola v teste potrebuje poznať svojho predkov a potomkov, aby kapitoly tvorili grafovú štruktúru. Testu postačí poznať koreňovú kapitolu, navyše test bez kapitoly by nedával žiadny zmysel, preto je medzi kapitolou a testom kompozičná väzba.

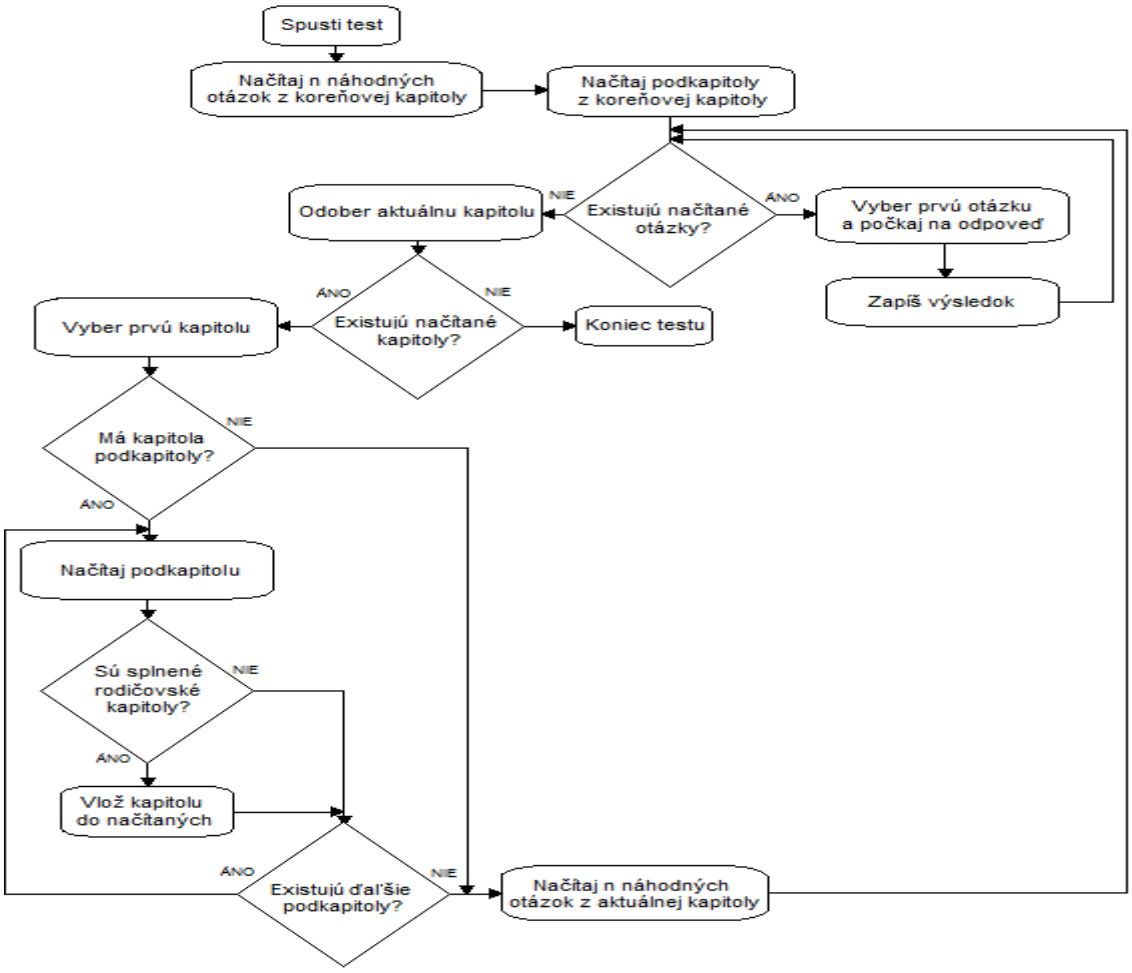
Každá kapitola môže obsahovať niekoľko otázok. Otázky majú testový charakter, kde každá otázka má niekoľko možností. Otázky v každej kapitole (ideálne v celom teste) by mali byť unikátne.

Vo väčšine prípadov by postačovala stromová štruktúra kapitôl, ale v špeciálnych prípadoch je treba umožniť kapitole mať viacerých predkov. Tento fakt je možno využiť pri generovaní otázok z kapitôl, tak že každý predok (rodičovská kapitola) reprezentuje závislosť potrebnú pre prístup k svojim potomkom.

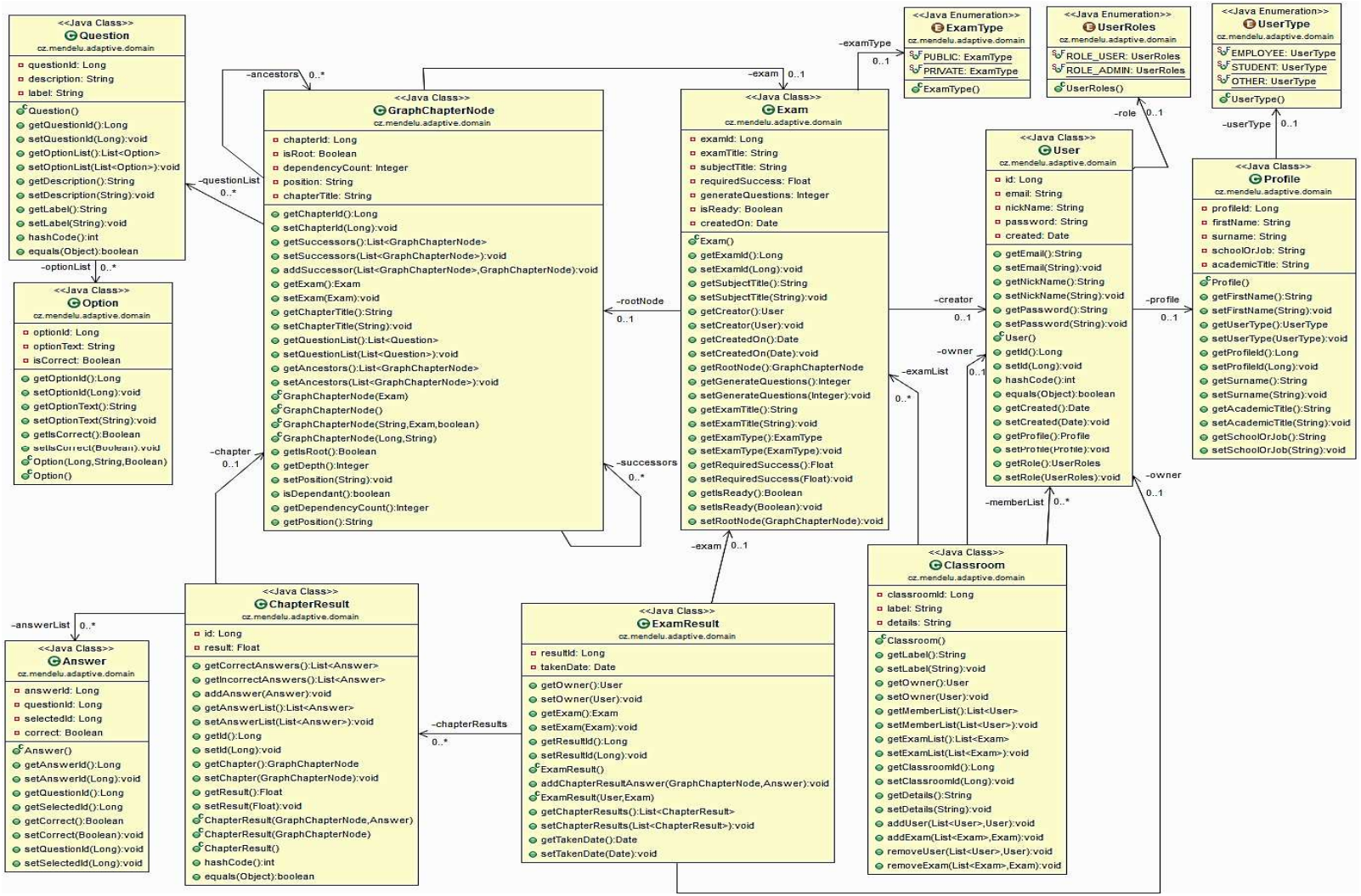
6.2 Generovanie kapitôl a otázok

Každý test vyžaduje správne zodpovedanie koreňovej kapitoly testu, aby používateľ mohol pokračovať v riešení otázok z následujúcich podkapitôl. Správne zodpovedanie kapitoly rovnako ako počet generovaných otázok závisí od konfigurácie vlastností testu. Test umožňuje globálne nastaviť počet náhodne generovaných otázok na kapitolu v teste. Obdobne je možné nastaviť percento vyžadovaných správnych odpovedí pre kapitoly testu, aby bola kapitola považovaná za správne zodpovedanú. Algoritmus generovania kapitôl je závislí od štruktúry kapitôl v teste, ktorú vlastník testu definuje pri tvorbe testu.

Obr. 5 Algoritmus generovania kapitôl a otázok



Obr. 6 UML


<<Java Class>>
Question
cz.mendelu.adaptive.domain
questionId: Long
description: String
label: String
Question()
getQuestionId():Long
setQuestionId(Long):void
getOptionList():List<Option>
setOptionList(List<Option>):void
getDescription():String
setDescription(String):void
getLabel():String
setLabel(String):void
hashCode():int
equals(Object):boolean
<<Java Class>>
Option
cz.mendelu.adaptive.domain
optionId: Long
optionText: String
isCorrect: Boolean
getOptionId():Long
setOptionId(Long):void
getOptionText():String
setOptionText(String):void
getIsCorrect():Boolean
setIsCorrect(Boolean):void
Option(Long,String,Boolean)
Option()
<<Java Class>>
GraphChapterNode
cz.mendelu.adaptive.domain
chapterId: Long
isRoot: Boolean
dependencyCount: Integer
position: String
chapterTitle: String
getChapterId():Long
setChapterId(Long):void
getSuccessors():List<GraphChapterNode>
setSuccessors(List<GraphChapterNode>):void
addSuccessor(List<GraphChapterNode>,GraphChapterNode):void
getExam():Exam
setExam(Exam):void
getChapterTitle():String
setChapterTitle(String):void
getQuestionList():List<Question>
setQuestionList(List<Question>):void
getAncestors():List<GraphChapterNode>
setAncestors(List<GraphChapterNode>):void
GraphChapterNode(Exam)
GraphChapterNode()
GraphChapterNode(String,Exam,boolean)
GraphChapterNode(Long,String)
getIsRoot():Boolean
getDepth():Integer
setPosition(String):void
isDependant():boolean
getDependencyCount():Integer
getPosition():String
<<Java Class>>
Exam
cz.mendelu.adaptive.domain
examId: Long
examTitle: String
subjectTitle: String
requiredSuccess: Float
generateQuestions: Integer
isReady: Boolean
createdOn: Date
Exam()
getExamId():Long
setExamId(Long):void
getSubjectTitle():String
setSubjectTitle(String):void
getCreator():User
setCreator(User):void
getCreatedOn():Date
setCreatedOn(Date):void
getRootNode():GraphChapterNode
getGenerateQuestions():Integer
setGenerateQuestions(Integer):void
getExamTitle():String
setExamTitle(String):void
getExamType():ExamType
setExamType(ExamType):void
getRequiredSuccess():Float
setRequiredSuccess(Float):void
getIsReady():Boolean
setIsReady(Boolean):void
setRootNode(GraphChapterNode):void
<<Java Enumeration>>
ExamType
cz.mendelu.adaptive.domain
PUBLIC: ExamType
PRIVATE: ExamType
ExamType()
<<Java Enumeration>>
UserRoles
cz.mendelu.adaptive.domain
ROLE_USER: UserRoles
ROLE_ADMIN: UserRoles
UserRoles()
<<Java Enumeration>>
UserType
cz.mendelu.adaptive.domain
EMPLOYEE: UserType
STUDENT: UserType
OTHER: UserType
UserType()
<<Java Class>>
User
cz.mendelu.adaptive.domain
id: Long
email: String
nickName: String
password: String
created: Date
getEmail():String
setEmail(String):void
getNickName():String
setNickName(String):void
getPassword():String
setPassword(String):void
User()
getId():Long
setId(Long):void
hashCode():int
equals(Object):boolean
getCreated():Date
setCreated(Date):void
getProfile():Profile
setProfile(Profile):void
getRole():UserRoles
setRole(UserRoles):void
<<Java Class>>
Profile
cz.mendelu.adaptive.domain
profileId: Long
firstName: String
surname: String
schoolOrJob: String
academicTitle: String
Profile()
getFirstName():String
setFirstName(String):void
getUserType():UserType
setUserType(UserType):void
getProfileId():Long
setProfileId(Long):void
getSurname():String
setSurname(String):void
getAcademicTitle():String
setAcademicTitle(String):void
getSchoolOrJob():String
setSchoolOrJob(String):void
<<Java Class>>
ChapterResult
cz.mendelu.adaptive.domain
id: Long
result: Float
getCorrectAnswers():List<Answer>
getIncorrectAnswers():List<Answer>
addAnswer(Answer):void
getAnswerList():List<Answer>
setAnswerList(List<Answer>):void
getId():Long
setId(Long):void
getChapter():GraphChapterNode
setChapter(GraphChapterNode):void
getResult():Float
setResult(Float):void
ChapterResult(GraphChapterNode,Answer)
ChapterResult(GraphChapterNode)
ChapterResult()
hashCode():int
equals(Object):boolean
<<Java Class>>
Answer
cz.mendelu.adaptive.domain
answerId: Long
questionId: Long
selectedId: Long
correct: Boolean
Answer()
getAnswerId():Long
setAnswerId(Long):void
getQuestionId():Long
getSelectedId():Long
getCorrect():Boolean
setCorrect(Boolean):void
setQuestionId(Long):void
setSelectedId(Long):void
<<Java Class>>
ExamResult
cz.mendelu.adaptive.domain
resultId: Long
takenDate: Date
getOwner():User
setOwner(User):void
getExam():Exam
setExam(Exam):void
getResultId():Long
setResultId(Long):void
ExamResult()
addChapterResultAnswer(GraphChapterNode,Answer):void
ExamResult(User,Exam)
getChapterResults():List<ChapterResult>
setChapterResults(List<ChapterResult>):void
getTakenDate():Date
setTakenDate(Date):void
<<Java Class>>
Classroom
cz.mendelu.adaptive.domain
classroomId: Long
label: String
details: String
Classroom()
getLabel():String
setLabel(String):void
getOwner():User
setOwner(User):void
getMemberList():List<User>
setMemberList(List<User>):void
getExamList():List<Exam>
setExamList(List<Exam>):void
getClassroomId():Long
setClassroomId(Long):void
getDetails():String
setDetails(String):void
addUser(List<User>,User):void
addExam(List<Exam>,Exam):void
removeUser(List<User>,User):void
removeExam(List<Exam>,Exam):void
-ancestors 0..*
-exam 0..1
-examType 0..1
-role 0..1
-userType 0..1
-questionList 0..*
-optionList 0..*
-rootNode 0..1
-creator 0..1
-profile 0..1
-owner 0..1
-examList 0..*
-chapter 0..1
-successors 0..*
-exam 0..1
-memberList 0..*
-owner 0..1
-answerList 0..*
-chapterResults 0..*

7 PERZISTENCIA DÁT

Všetky informácie v aplikácii nestačí iba uložiť do premenných, ale treba aj zabezpečiť aby boli dostupné aj po páde, či úmyselnom reštartovaní serveru. Pre perzistenciu dát bola zvolená voľne distribuovaná objektovo-relačná databáza PostgreSQL. Do tejto databáze sú vkladané údaje pomocou Hibernate frameworku.

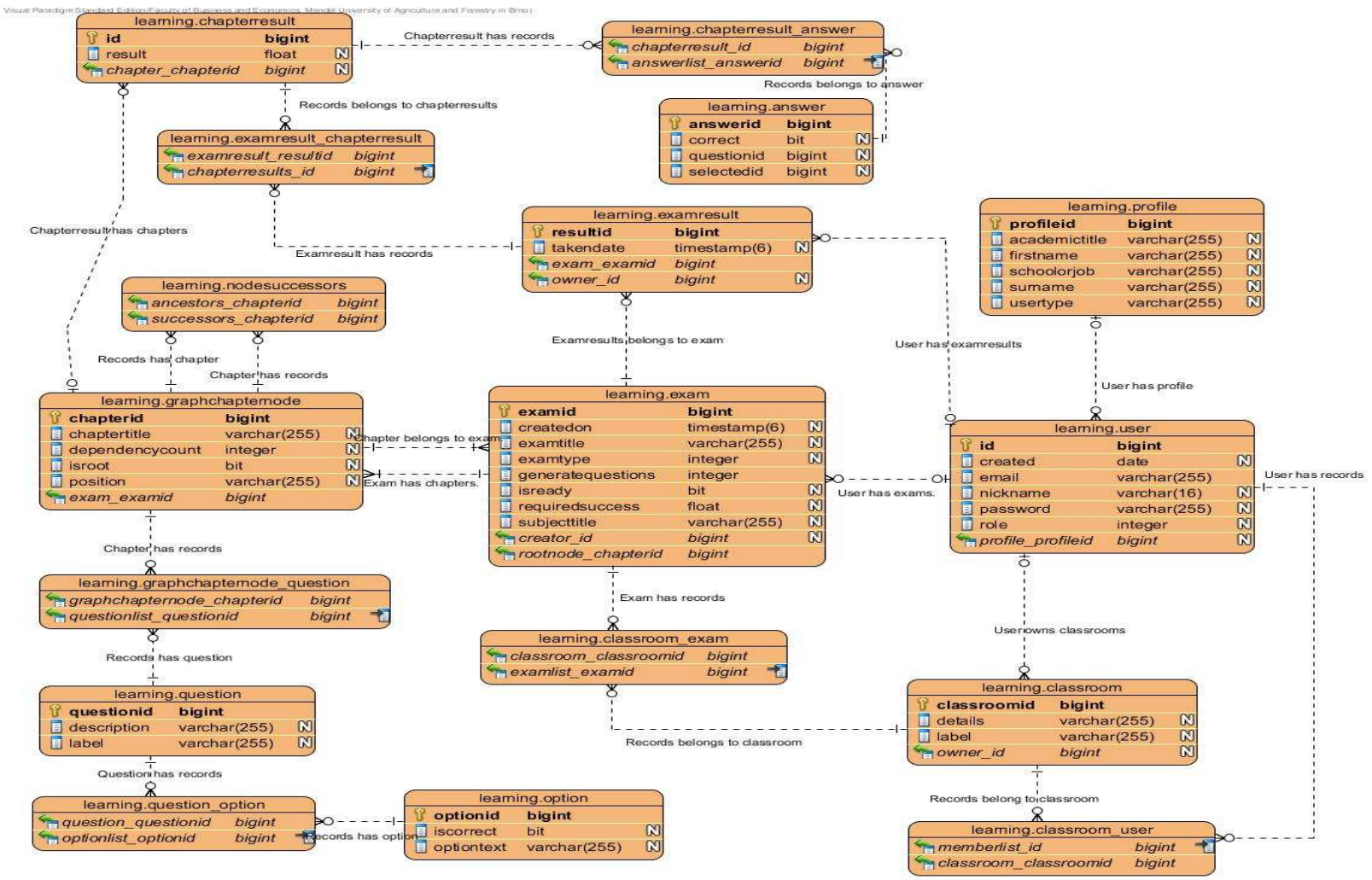
7.1 Hibernate

Databázové entity sú vytvorené automaticky mapovaním java tried. Mapovanie tried je realizované pomocou JPA anotácii nad POJO triedami. Aby bolo možné využiť Hibernate je treba nakonfigurovať aplikačný kontext. Vložením XML konfigurácii do aplikačného kontextu sa umožní pripojenie do databáze cez „*sessionFactory*“ objekt. Cez tento objekt sú ďalej realizované implementácie DAO vrsty.

Obr. 7 Konfigurácia Hibernate

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.postgresql.Driver" />
        <property name="url" value="jdbc:postgresql://localhost:5432/learning" />
        <property name="username" value="postgres" />
        <property name="password" value="******" />
</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
   <property name="dataSource" ref="dataSource" />
   <property name="packagesToScan" value="cz.mendelu.adaptive.domain" />
   <property name="hibernateProperties">
      <props>
         <prop key="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</prop>
         <prop key="hibernate.show_sql">true</prop>
         <prop key="hibernate.hbm2ddl.auto">create-update</prop>
         <prop key="hibernate.default_schema">learning</prop>
         <prop key="connection.pool_size">2</prop>
         <prop key="connection.autocommit">true</prop>
      </props>
   </property>
</bean>

<bean id="transactionManager" class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

Obr. 8 ERD


learning.chapterresult
id bigint
result float N
chapter_chapterid bigint N
Chapterresult has records
learning.chapterresult_answer
chapterresult_id bigint
answerlist_answerid bigint
Records belongs to answer
learning.answer
answerid bigint
correct bit N
questionid bigint N
selectedid bigint N
Records belongs to chapterresults
learning.examresult_chapterresult
examresult_resultid bigint
chapterresults_id bigint
Chapterresult has chapters
Examresult has records
learning.examresult
resultid bigint
takendate timestamp(6) N
exam_examid bigint
owner_id bigint N
learning.profile
profileid bigint
academictitle varchar(255) N
firstname varchar(255) N
schoolorjob varchar(255) N
surname varchar(255) N
usertype varchar(255) N
learning.nodesuccessors
ancestors_chapterid bigint
successors_chapterid bigint
Records has chapter
Chapter has records
Examresults belongs to exam
User has examresults
User has profile
learning.graphchaptermode
chapterid bigint
chaptertitle varchar(255) N
dependencycount integer N
isroot bit N
position varchar(255) N
exam_examid bigint
Chapter belongs to exam
Exam has chapters.
learning.exam
examid bigint
createdon timestamp(6) N
examtitle varchar(255) N
examtype integer N
generatequestions integer
isready bit N
requiredsuccess float N
subjecttitle varchar(255) N
creator_id bigint N
rootnode_chapterid bigint
User has exams.
learning.user
id bigint
created date N
email varchar(255)
nickname varchar(16) N
password varchar(255) N
role integer N
profile_profileid bigint N
User has records
Chapter has records
learning.graphchaptermode_question
graphchaptermode_chapterid bigint
questionlist_questionid bigint
Exam has records
Records has question
learning.classroom_exam
classroom_classroomid bigint
examlist_examid bigint
User owns classrooms
learning.question
questionid bigint
description varchar(255) N
label varchar(255) N
Question has records
Records belongs to classroom
learning.classroom
classroomid bigint
details varchar(255) N
label varchar(255) N
owner_id bigint N
Records belong to classroom
learning.question_option
question_questionid bigint
optionlist_optionid bigint
Records has option
learning.option
optionid bigint
iscorrect bit N
optiontext varchar(255) N
learning.classroom_user
memberlist_id bigint
classroom_classroomid bigint

7.2 DAO vrstva

DAO vrstva obsahuje okrem jednotlivých implementácii v HQL a SQL jazyku aj rozhrania pre CRUD operácie. Rozhrania DAO vrstvy ďalej tvoria DAO služby, ktoré zapúzdrujú databázové operácie a automatizujú správu databázových sedení. DAO služby sú komponenty aplikácie, preto sa k ním dá ľahko pristúpiť vďaka IoC.

Tab. 3 DAO vrstva

DAO služba	DAO rozhranie	DAO implementácia
ExamDaoService	ExamDao	ExamDaoImpl
	ChapterDao	ChapterDaoImpl
	QuestionDao	QuestionDaoImpl
	ResultDao	ResultDaoImpl
UserDaoService	UserDao	UserDaoImpl
ClassroomDaoService	ClassroomDao	ClassroomDaoImpl

8 PREZENTAČNÁ VRSTVA A BEZPEČNOSŤ

Zobrazovanie údajov v aplikácii je riešené generovaním webových stránok s použitím JSP. Controller aplikácie zabezpečuje dostupnosť modelu v JSP pohľadoch. Používateľské prostredie je rozdelené na niekoľko častí, ktoré sú spojené pomocou šablónovacieho nástroja Apache Tiles. Jednotlivý obsah a používateľské rozhranie webových stránok je definované značkovacím jazykom HTML a naštýlované pomocou kaskádových štýlou. Ďalej používateľské rozhranie využíva javascript a javascriptovú knižnicu jQuery. Šablona pre grafické rozhranie bola vytvorená cez jQuery UI.

Bezpečnosť aplikácie je realizovaná cez Spring Security. Pridaním Spring Security do kontextu aplikácie vznikne filter, ktorý je spravovaný podľa nastavení definovaných v XML súbore *root-context-security.xml*. V konfigurácii treba nastaviť registračnú, odhlasovaciu stránku bez zabezpečenia, aby bolo možné sa prihlásiť alebo registrovať. A tiež je treba nastaviť odkaz na triedu implementujúcu službu pre načítanie používateľských údajov z databázy. Pokiaľ je treba používať inú ako automaticky generovanú prihľasovaciu stránku, je nutné ju tiež definovať.

Obr. 9 Konfigurácia Spring Security

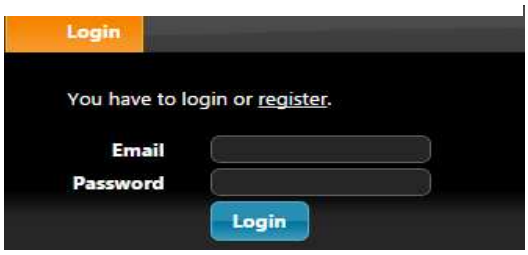
```
<http use-expressions="true" security="none" pattern="/login" />
<http use-expressions="true" security="none" pattern="/register" />
<http use-expressions="true" security="none" pattern="/resources/**"/>

<http use-expressions="true" auto-config="false">
    <form-login login-page="/login" authentication-failure-url="/login?err=1"
        username-parameter="username" password-parameter="password" />
    <intercept-url pattern="/**" access="hasRole('ROLE_USER') or hasRole('ROLE_ADMIN')"/>
    <logout logout-url="/logout" logout-success-url="/login"/>
</http>
<beans:bean id="userDetailsService"
    class="cz.mendelu.adaptive.services.UserSecurityService"></beans:bean>
    <authentication-manager>
        <authentication-provider user-service-ref="userDetailsService" >
            <password-encoder hash="sha"></password-encoder>
        </authentication-provider>
    </authentication-manager>
    <global-method-security secured-annotations="enabled"/>
</beans:beans>
```

9 DOKUMENTÁCIA

9.1 Registrácia a prihlasovanie

Po načítaní webovej adresy aplikácie je potrebné sa prihlásiť na svoj účet zadaním emailu a hesla. Ak používateľ nevlastní účet môže si účet vytvoriť po kliknutí na odkaz *„register“*. Registračný formulár požaduje vložiť prezívku, email a heslo. Po zaregistrovaní je poslaný email s prihlasovacími údajmi na zadaný email. Po úspešnom prihlásení sa zobrazí uvítacia správa, inak sa zobrazí chybová správa s výzvou o opätovné zadanie prihlasovacích údajov.

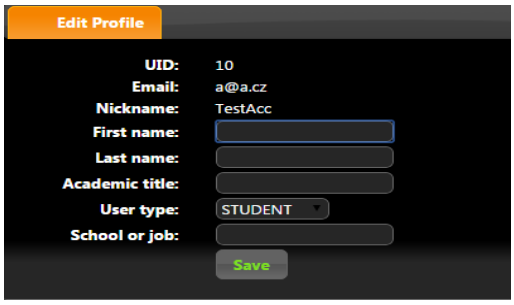
Obr. 10 Prihlasovací formulár



9.2 Profil

Pri prvom prihlásení je vhodné vložiť údaje do profilu, aby bolo jednoduchšie rozpoznať identitu používateľa. Profil je možné upravovať po kliknutí na tlačitlo *„My Profile“* v menu.

Obr. 11 Používateľský profil



9.3 Vyhľadávanie používateľa

V menu *„Find user“* sa dajú vyhľadávať používatelia pomocov údajov *UID*, prezívky, emailu alebo školy. Nájdení používatelia sú zobrazení na karte *„Result“* a po kliknutí na používateľovu prezívku je zobrazený profil používateľa.

Obr. 12 Vyhľadávanie používateľa

9.4 Testy

Pod položkou menu „*My Exams*“ je možné vytvárať, spravovať, odstraňovať vytvorené testy, tiež je možné zobrazovať výsledky vlastnených testov, ktoré písali ostatný používatelia.

9.4.1 Vytváranie testov

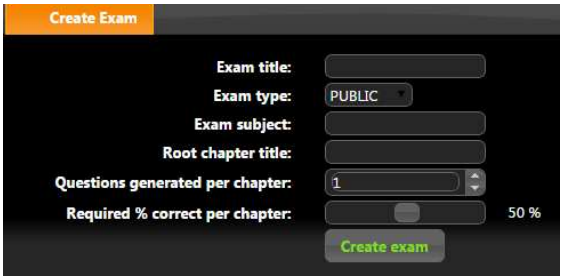
Test je možné vytvoriť kliknuťím na položku v podmenu „*Create new exam*“. Zobrazí sa formulár, ktorý pozostáva z položiek:

Obr. 13 Položky pri vytváraní testu

Názov	Hodnoty	Význam
Exam title	text	Názov testu
Exam type	public	Verejný test môže písať každý používateľ
	private	Privátny môže písať len používateľ z triedy
Exam subject	text	Názov predmetu
Root chapter title	text	Názov koreňovej kapitoly
Questions generated per chapter	číslo	Počet vygenerovaných otázok z každej kapitoly
Required % correct per chapter	číslo	Požadované percento správnych odpovedí na kapitolu

Po potvrdení formuláru nastane presmerovanie na správu testu a položku podmenu *„Add exam data“*.

Obr. 14 Vytváranie testu



9.4.2 Vkladanie otázok

Na karte *„Add question“* je možné vložiť otázku do súčasne prezeranej kapitoly. Treba vyplniť názov, popis otázky a vložiť požadovaný počet možností s ich znením. Možností je možné pridávať alebo odoberať stlačením tlačidla „+“ alebo „-“.

9.4.3 Detail, úprava a odoberanie otázok

Na karte *„Questions in <názov kapitoly>“* je možné vidieť vložené otázky v danej kapitole. Pri každej vloženej otázke sú zobrazené tri ikony, ktoré po kliknutí realizujú príslušnú udalosť. Prvá zobrazí detail otázky, druhá umožnuje upraviť otázku a tretia zmaže otázku.

9.4.4 Vkladanie kapitol

Do každej kapitoly je možné vložiť podkapitolu na záložke *„Add sub-chapter“*. Podkapitola je vložená do aktuálne prehliadanej kapitoly. Je treba vyplniť iba názov vkladanej kapitoly, ktorý musí byť dlhší ako štyri znaky.

9.4.5 Navigácia medzi kapitolami

Aktuálne prehliadaná kapitola sa dá zmeniť cez položku podmenu *„Chapter navigation“*. Ak existuje viac kapitol, tak na tejto stránke možno vidieť najbližšie nadradené kapitoly a podkapitoly aktuálne prehliadanej kapitoly. Pre zmenu prehliadanej kapitoly je možné kliknúť buď na niektorú zo všetkých kapitol testu v ponuke *„Go to chapter“* alebo na niektorú z najbližších podkapitol v stromovom zobrazení pod nápisom *„Go to child chapter“*. Po kliknutí na niektorú z alternatív dôjde k presmerovaniu na pôvodnú stránku pre vkladanie otázok a kapitol, ale už pre zvolenú kapitolu.

9.4.6 Úprava a odstránenie kapitoly

Prehliadanú kapitolu je možné premenovať cez podmenu *„Edit chapter"* na karte *„Edit chapter title"*. Na karte *„Delete chapter"* sa dá odstrániť aktuálna kapitola so všetkými jej podkapitolami. Ak je však na tejto kapitole závislá aj iná kapitola, bude aj táto kapitola odstránena spolu s jej podkapitolami. Preto je vhodné pred zmazaním kapitoly vhodné odstrániť všetky vytvorené závislosti na tejto kapitole.

9.4.7 Vytváranie závislostí medzi kapitolami

Na poslednej záložke *„Chapter dependencies"* v podmenu *„Edit chapter"* sa dajú pridávať závislosti medzi kapitolami, ktoré nie sú priamymi následovníkmi alebo potomkami aktuálnej kapitoly. Kapitola, ktorej sa priradí závislosť na inej kapitole, bude pri písaní testu vygenerovaná len ak budú úspešne zodpovedané všetky závislé kapitoly a jej rodičovská kapitola. Ak je kapitola závislá na nejakých kapitoliach, sú tieto kapitoly zobrazené na tejto karte. Existujúce závislosti je možné odstrániť po kliknutí na ikonu *„x"* vedľa závislosti, ktorá sa má odstrániť.

9.4.8 Úprava a odstránenie testu

Vlastnosti testu testu je možné upravovať v menu *„My exams"*, kde sú zobrazené testy vlastnené prihláseným používateľom. Úpravy testov sa vykonávajú cez prvú ikonu pri teste (ikona ceruzky). Každý test, ktorý je pripravený na písanie treba upraviť a zaškrtnúť možnosť *„ready to take"*. Po tomto úkone je verejný test možné začať písať, privátny test je ešte potrebné najprv priradiť do niektorej z vlastnených tried. Prostredná ikona pri teste otvorí stránku na pridávanie kapitôl a otázok. Posledná ikona umožňuje zmazať celý test.

9.5 Triedy

Triedy slúžia pre zadávanie privátnych testov pre určitú skupinu používateľov. Triedy sú prístupné cez položku v menu *„Classrooms"*. Vlastník triedy má možnosť pridávať a odoberať používateľov a privátne testy do triedy. Tiež si môže prezerať výsledky používateľov v triede. Používatelia v triede si možu zobraziť detail triedy, ktorý pozostáva so spolužiakov a tiež prístupných testov.

9.5.1 Priradené triedy

Na prvej položke podmenu *„Show classrooms"* je zobrazený zoznam tried, ku ktorým je používateľ priradený. Každá trieda má dve ikony akcie. Ikona lupy zobrazí detaily triedy, druhá ikona odobere používateľa z danej triedy.

9.5.2 Vytvorenie a správa triedy

Triedy je možné vytvárať cez podmenu *„Create new classroom“*. Po vyplnení názvu a detailov triedy je používateľ presmerovaný na správu triedy. Privátne testy sa pridávajú na karte *„Add exam“* zo zoznamu testov, ktoré používateľ vlastní. Používateľov je možné pridávať na karte *„Add user“* cez *UID* alebo vyhľadať *UID* podľa prezívky. Pridané testy a používatelia sa dajú z triedy odobrať cez ikonu pri danom teste alebo používateľovi. Na druhej požke podmenu *„Classroom exam result“* sa zobrazia výsledky testov, ktoré písali používatelia v triede.

9.5.3 Úprava a odstránenie triedy

V menu *„Classrooms“* v prostrednej položke podmenu *„Show owend classroom“* sa zobrazia triedy, ktoré používateľ vytvoril a vlastní ich. Ku každej sú dostupné tri akcie cez ikony. Úprava názvu a detailov triedy, správa triedy a posledná ikona zmaže triedu.

9.6 Písanie testu

Testy je možné písať cez menu *„Take exam“*. Na prvej položke podmenu *„Available exams“* sú dostupné dve karty prvá *„Classroom exams“*, ktorá zobrazuje dostupné testy zo všetkých tried, ku ktorým je používateľ priradený. Druhá karta *„Browse public exams“* zobrazuje všetky dostupné verejné testy.

9.6.1 Písanie testu

Po kliknutí na nápis *„Start exam“* sa zobrazí detail testu. Stlačením tlačidla begin exam sa test spustí vygenerovaním otázky z koreňovej kapitoly. Po skončení testu je používateľ presmerovaný na stránku s výsledkom testu.

9.6.2 Systém generovania kapitol a otázok

Otázky v teste sú generované z kapitôl vo vopred definovanom počte zobrazenom v detaile testu. Postupnosť kapitôl, z ktorých sú generované otázky, začína koreňovou kapitolou a pokračuje najbližšiou podkapitolou ak používateľ úspešne splnil jej rodičovskú kapitolu. Ak používateľ nesplní požadované percento úspešnosti pri niektorej z kapitôl budú mu vygenerované otázky z najbližšej susednej kapitoly. Test skončí keď už nezostanú žiadne dostupné kapitoly.

9.7 Výsledky testu

Na druhej položke podmenu *„Exam results“* v menu *„Take exam“* sú zobrazené všetky výsledky testov, ktoré používateľ písal. Výsledky z testov, ktoré vlastní používateľ sa dajú zobraziť cez menu *„My exams“* pod položkou *„Browse exam results“*.

9.7.1 Zobrazenie výsledkov testu

Po kliknutí na ikonu pri výsledku z testu sa zobrazí úspešnosť pri jednotlivých kapitolách v poradí, v ktorom na ne používatelia odpovedali.

9.7.2 Detail výsledku

Detail výsledku pre používateľa, ktorý písal test, zobrazuje každá kapitola iba poradie otázok a úspešnosť . Preto testovaný nemá možnosť získať znenie otázky z výsledku testu. Vlastník testu si pri zobrazení detailu výsledku testu, ktorý vlastní uvidí navyše aj zadanie otázky s možnosťami.

10 ZÁVER

Na záver možno zhodnotiť, že sa podarilo vytvoriť aplikáciu s podporou adaptívneho testovania vedomostí. Aplikácia je napísaná v programovaciom jazyku Java, konkrétne pomocou Spring Framework MVC, vďaka čomu je nezávislá na platforme a jednoducho modifikovateľná. Medzi hlavné črty aplikácie patrí vlastnosť, že každý používateľ môže vytvárať testy. Používateľ musí vytvoriť test tak, aby otázky a kapitoly zodpovedali požiadavkám na položkovú banku adaptívneho testovania ak má výsledok znázorňovať reálne vedomosti testovaného. Štruktúra kapitol a jednotlivé otázky je možné jednoducho upravovať, pridávať a zmazávať. Medzi ďaľšie implementované funkcie aplikácie patrí aj používateľské triedy, vyhľadávanie a zobrazovanie profilu používateľa.

Medzi nevýhody implementácie možno zahrnúť fakt, že aplikácia nie je založená na IRT, ale hodnotí používateľove výsledky na základe správnosti jeho odpovedí. Aj napriek tomu je však možné zistiť úroveň testovaného, keďže kapitoly testu sú previazané zobrazujú určitú vedomostnú úroveň.

10.1 Budúcnosť aplikácie

V budúcnosti by bolo vhodné rozšíriť aplikáciu o IRT skórovanie odpovedí, čo vyžaduje matematické a štatistické funkcie výpočtu a implementáciu kalibračných funkcí pre štruktúru testu. Aplikácia by mala byť internacionalizovaná vo viacerých svetových jazykoch. Výsledky testov by mohli byť exportovatelné do fotmátu pdf a tiež by sa mali dať otázky vkladať do aplikácie v určitom súborovom formáte.

11 ZOZNAM LITERATÚRY

[1]. JOHNSON, Rod. *Professional Java development with the Spring framework*. Indianapolis: Wiley Publishing, 2005, xxviii, 644 s. ISBN 07-645-7483-3.

[2]. WALLS, Craig. *Spring in action*. 3rd ed. Shelter Island: Manning, c2011, xxiii, 400 p. ISBN 19-351-8235-8.

[3]. WEISS, D. J., 2004: Computerized Adaptive Testing for Effective and Efficient Measurement in Counseling and Education. Measurement and Evaluation in Counseling and Development. 37, 2: 69–84. ISSN-0748-1756.

[4]. DLABOLOVÁ, D., RYBIČKA, J. Study maps as tool for the adaptive tests construction. Acta Universitasis Agriculturae et Silviculturae Mendeliane Brunesis. 2013. Sv. 61, č. 7, s 2045-2054. ISSN 1211-8516

[5]. Apache Tomcat. *Tomcat* [online]. c1999 [cit. 2015-01-01]. Dostupné z: http://tomcat.apache.org/

[6]. *International Association for Computerized Adaptive Testing* [online]. 2015 [cit. 2015-01-01]. Dostupné z: http://iacat.org/

[7]. About. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL* [online]. 2015 [cit. 2015-01-01]. Dostupné z: http://www.postgresql.org/about/

[8]. JAVASCRIPT WEB APIS. *W3C* [online]. 2015 [cit. 2015-01-01]. Dostupné z: http://www.w3.org/standards/webdesign/script

[9]. Apache Maven Project. *Welcome to Apache Maven* [online]. c2001 [cit. 2015-01-01]. Dostupné z: http://maven.apache.org/