TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

# SQL TESTER

## Bakalářská práce

*Studijní program:* B2646 – Informační technologie
*Studijní obor:* 1802R007 – Informační technologie

*Autor práce:* **Jakub Venglář**
*Vedoucí práce:* Ing. Jana Vitvarová, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

# SQL TESTER

## Bachelor thesis

*Study programme:* B2646 – Information Technology
*Study branch:* 1802R007 – Information Technology

*Author:* **Jakub Venglář**
*Supervisor:* Ing. Jana Vitvarová, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
**Fakulta mechatroniky, informatiky a mezioborových studií**
Akademický rok: **2014/2015**

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Jakub Venglář** |
| Osobní číslo: | **M12000189** |
| Studijní program: | **B2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Název tématu: | **SQL Tester** |
| Zadávající katedra: | **Ústav mechatroniky a technické informatiky** |

Zásady pro vypracování:

1. V novém iteračním kroku upravte návrh prototypu webové aplikace pro testování SQL z bakalářskeho projektu podle finálních požadavků zadavatele tj. vedoucího práce.
2. Seznamte se s existujícími Java frameworky a vybrané implementujte. Aplikaci postavte nad Oracle databází.
3. Popište možnosti autentizace a autorizace interních a externích uživatelů. Vyberte a implementujte nejvhodnější variantu.
4. Ošetřete aplikaci z bezpečnostního hlediska.
5. Vytvořte vzorové testovací scénáře a aplikaci otestujte.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:


[1] **Oracle Academy: Materiály k programu Advanced Computer Science [online]. [vid. 9. 10. 2014]. Dostupné (pod heslem) z: https://academy.oracle.com/oa-web-advancedcs-curriculum.html**

[2] **Pokorný J., Valenta M.: Databázové systémy, Praha: Nakladatelství ČVUT, 2013**

[3] **Venglář J.: Webová aplikace pro testování SQL - bakalářský projekt, Technická univerzita v Liberci, 2014**


Vedoucí bakalářské práce: **Ing. Jana Vitvarová, Ph.D.**
Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2014**

Termín odevzdání bakalářské práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2014

# Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem. Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 15.5. 2015

Podpis: Věnglář

# Abstrakt

První část práce popisuje kompletní funkcionalitu aplikace SQL Tester a také návrh datového modelu. SQL Tester je aplikace pro podporu předmětu Databázové sytémy na Technické univerzitě v Liberci. Automatizuje kotrolu SQL dotazů a tím šetří učitelům čas a studentům nabízí možnost testovat své znalosti. Druhá fáze zabývající se implementací kombinuje popis technologií s jejich konkrétním využitím v aplikaci. Vzhledem k tomu, že je aplikace postavena nad Oracle databází a Java EE frameworky, jsou zde uvedeny například technologie Spring, Java Persistence API, JavaServer Faces, PrimeFaces a mnoho dalších. Rovněž je zde popis zvolené architektury a zabezpečení aplikace. V poslední třetí fázi práce seznamuje s testováním aplikace v reálném provozu a s funkčními testy.



# Klíčová slova

webová aplikace, SQL, JPA, Spring, PrimeFaces



# Abstract

The first part of the work introduces the complete functionality of the SQL Tester application and its data model. SQL Tester supports the Database Systems course at the Technical University of Liberec. It automatically checks SQL queries. The second implementation part combines the general descriptions of chosen technologies with the explanation of their particular use in the SQL Tester application. As the application is built on Oracle database and Java EE frameworks we discuss here, among other, technologies such as Spring, Java Persistence API, JavaServer Faces, PrimeFaces as well as the overall architecture and security issues. The functional testing and real traffic tests are described in the last third section.



# Keywords

web application, SQL, JPA, Spring, PrimeFaces

# Obsah

**1 Úvod** **12**
1.1 Cíle práce . . . 12

**2 Návrh aplikace** **13**
2.1 Jaký problém aplikace řeší? . . . 13
2.2 Základní požadavky zadavatele . . . 13
2.3 UseCase diagram a relační datový model . . . 13
2.4 Základní funkce aplikace . . . 13
2.4.1 Role student . . . 14
2.4.2 Role administrátor . . . 14
2.4.3 Role učitel . . . 14
2.5 Návrhový vzor MVC a architektura aplikace . . . 15

**3 Příprava před vývojem** **17**
3.1 Webový server Apache Tomcat 8 . . . 17
3.2 Databázový server Oracle XE 11g . . . 17
3.3 Vývojové prostředí NetBeans IDE 8 . . . 17
3.4 Vytvoření Maven projektu . . . 18
3.5 Verze knihoven . . . 18
3.6 Systém balíčků . . . 19

**4 Model** **20**
4.1 Domain . . . 20
4.1.1 ORM . . . 20
4.2 Repository . . . 21
4.2.1 Hibernate . . . 21
4.2.2 JPA . . . 23
4.2.3 Spring Data JPA . . . 24
4.3 Service . . . 25
4.3.1 Spring . . . 25
4.3.2 Validace a Verifikace SQL dotazu . . . 26

**5 Controller** **28**

**6 View** **29**
6.1 JSF . . . 29
6.2 PrimeFaces . . . 30
6.3 PrettyFaces . . . 31
6.4 Šablona . . . 31
6.5 Chybové stránky . . . 32
6.6 Resources . . . 32
6.7 Lokalizace . . . 33

**7 Zabezpečení** **35**
7.1 HTTPS . . . 35
7.1.1 Certifikát . . . 35
7.2 Spring security . . . 36
7.3 Autentizace uživatelů . . . 37
7.4 Shibboleth . . . 38
7.5 Spring SAML . . . 39
7.6 Ochrana proti 10 nejčastějším rizikům podle OWASP . . . 39
7.6.1 SQL Injection . . . 39
7.6.2 Odcizení hesel a správa sessions . . . 40
7.6.3 XSS . . . 40
7.6.4 Nezabezpečené přímé odkazy na objekty . . . 41
7.6.5 Nebezpečná konfigurace . . . 41
7.6.6 Zveřejnění citlivých dat . . . 41
7.6.7 Chyby v řízení úrovní přístupů . . . 42
7.6.8 CSRF . . . 42
7.6.9 Použití známých zranitelných komponent . . . 42
7.6.10 Neošetřené přesměrování a předávání . . . 42
7.6.11 Clickjacking . . . 42
7.7 Validace vstupů . . . 43
7.8 Logování . . . 43

**8 Testování** **44**

**9 Závěr** **45**

9.1 Splnění cílů . . . . . 45

**Použitá literatura** **46**

**Přílohy** **49**

**A Obsah CD** **49**

**B Use Case diagram** **50**

**C Relační datový model** **51**

**D Procedura v PL/SQL vracející XML** **52**

**E Administrace uživatelů** **53**

**F Řešení úloh** **54**

**G Dialog editace kroužku** **55**

**H Správa kroužků** **56**

**I Spouštění SQL dotazů** **57**

**J Chybová stránka** **58**

# Seznam obrázků

1 Architektura aplikace SQL Tester . . . . . 15
2 Systém balíčků . . . . . 19
3 ORM schéma . . . . . 20
4 Databázové schéma pro ORM . . . . . 21
5 Schéma JPA . . . . . 24
6 Tabulka uživatelů v PrimeFaces . . . . . 30
7 Záhlaví stránky s logem a menu . . . . . 32
8 Zápatí stránky s vlaječkami . . . . . 33

# Seznam zkratek

**AJAX**
Asynchronous JavaScript and XML

**AOP**
Aspect Oriented Programming

**API**
Application Programming Interface

**CESNET**
Czech Education and Scientific NETwork

**CRUD**
Create Read Update Delete

**CSRF**
Cross-Site Request Forgery

**CSS**
Cascading Style Sheets

**CSV**
Comma Separated Values

**DAO**
Data Access Object

**EJB**
Enterprise Java Beans

**EL**
Expression Language

**GUI**
Graphical User Interface

**HTML**
HyperText Markup Language

**HTTP**
Hypertext Transfer Protocol

**HTTPS**
Hypertext Transfer Protocol Secure

**IDE**
Integrated Development Environment

**IdP**
Identity Provider

**IoC**
Inversion of Control

**Java EE**
Java Enterprise Edition

**JDBC**
Java Database Connectivity

**JKS**
Java KeyStore

**JNDI**
Java Naming and Directory Interface

**JPA**
Java Persistence API

**JPQL**
Java Persistence Query Language

**JSF**
JavaServer Faces

**JSP**
JavaServer Pages

**JTA**
Java Transaction API

**LDAP**
Lightweight Directory Access Protocol

**MVC**
Model View Controller

**ORM**
Objektově-relační mapování

**OWASP**
Open Web Application Security Project
**PDF**
Portable Document Format
**PEM**
Privacy Enhanced Mail
**PHP**
Hypertext Preprocessor
**PKCS**
Public Key Cryptographic Standards
**PL/SQL**
Procedural Language/Structured Query Language
**SAML**
Security Assertion Markup Language
**SEO**
Search Engine Optimization
**SLF4J**
Simple Logging Facade for Java
**SP**
Service Provider
**SQL**
Structured Query Language
**SSO**
Single Sign-On
**STAG**
Studijní agenda
**TLS**
Transport Layer Security
**TUL**
Technická univerzita v Liberci
**URI**
Uniform Resource Identifier
**URL**
Uniform Resource Locator
**UTF-8**
Universal Character Set Transformation Format
**XE**
Express Edition
**XHTML**
Extensible Hypertext Markup Language
**XML**
Extensible Markup Language
**XSLT**
Extensible Stylesheet Language Transformations
**XSS**
Cross-site scripting

# 1 Úvod

Svou bakalářskou prací navazuji na ročníkový projekt, ve kterém jsem se seznámil s databází Oracle a základními prvky jazyka Java EE, jako jsou Java Servlets a JSP (JavaServer Pages), za účelem vytvoření aplikace SQL Tester. Také jsem se zabýval možnostmi validace a verifikace SQL dotazu. Technologie Java Servlets a JSP se ovšem v praxi už jistou dobu nevyužívají. Během posledních let vzniklo nad těmito technologiemi několik frameworků, bez nichž se dnes téměř neprogramuje. Proto jsem se ve své bakalářské práci zaměřil zejména na ně. Nicméně si myslím, že je dobré znát i technologie pod těmito frameworky.

Využil jsem tedy tuto práci, abych obohatil své znalosti o moderní technologie, které se v praxi využívají a v budoucnu bych se jimi rád zabýval. Druhá motivace je vytvoření užitečné aplikace, která pomůže studentům a usnadní práci učitelům. Podobnou aplikaci lze nalézt na stránkách [9], ovšem je integrována v portále s online vzdělávacími kurzy.

Při vývoji aplikace jsem využil inkrementálního přístupu metodiky vývoje softwaru. Tento způsob vývoje spočívá v definování požadavků, návrhu struktury a vývoji samostatných funkčních částí aplikace. Po vytvoření každé části jsem danou funkcionalitu konzultoval s vedoucí práce. Výhodou tohoto přístupu je jednodušší reakce na případné změny.

Práci jsem rozdělil do tří částí. Do první části jsem zařadil návrh aplikace a zvolenou architekturu. V druhé části se zabývám implementací aplikace. Zde každou kapitolu začínám popisem technologií a končím jejich konkrétním využitím v aplikaci. Ve třetí části pak popisuji způsob testování funkčnosti aplikace. U závěrečných prací, které se týkají informačních technologií a zejména vývoje softwaru, je běžné psát práci kombinovaně. Takto strukturovaná práce se i lépe čte, protože po seznámení s technologií následuje její konkrétní využití. K tomuto způsobu členění práce jsem dospěl po konzultaci s vedoucí práce.

## 1.1 Cíle práce

- Seznámit se s frameworky pro vývoj webových aplikací v Java EE.
- Vytvořit aplikaci pro testování SQL dotazů.
- Zabezpečit aplikaci.
- Zvolit vhodný způsob autentizace a autorizace uživatelů.
- Otestovat funkčnost aplikace.

# 2 Návrh aplikace

## 2.1 Jaký problém aplikace řeší?

Učitelé často na cvičeních a u zkoušek v rámci předmětu Databázové systémy vyžadují po studentech psaní SQL dotazů. SQL je jazyk pro komunikaci s relačními databázemi. Ve chvíli, kdy jsou dotazy složitější a studentů je hodně, je pro učitele manuální kontrola správnosti dotazu poměrně časově náročná. Současný stav dokonce vypadá tak, že učitelé některé dotazy ručně přepisují a spouštějí na vlastní databázi. Aplikace tedy tuto kontrolu automatizuje a usnadní učitelům práci. Z malé rešerše vyplynulo, že každý semestr ušetří učitelům přes 20 hodin. Navíc umožní procvičovat SQL dotazy se studenty častěji.

## 2.2 Základní požadavky zadavatele

Zadavatel má na aplikaci několik základních požadavků. Aplikace má umožnit učitelům vytvářet v databázích tabulky a plnit je daty. Nad těmito databázemi mohou učitelé vytvářet testovací otázky. Studenti mohou tyto otázky řešit a učitelé uvidí jejich úspěšnost. S kompletní funkcionalitou nás seznámí kapitola 2.4.

## 2.3 UseCase diagram a relační datový model

Jelikož jsem se návrhem aplikace zabýval již v ročníkovém projektu, zmíním pouze UseCase diagram (zvaný i diagram případů užití) a relační datový model v příloze B a C. V ročníkovém projektu jsem specifikoval základní požadavky, vytvořil jsem UseCase diagram, sekvenční diagramy, logický a relační datový model. Protože se měnily požadavky na aplikaci, bylo nutné provést změny i v modelu. UseCase diagram je kvůli přehlednosti zjednodušený. Jednotlivé případy užití by bylo ještě možné rozdělit a více konkretizovat. V této práci bych se rád více zabýval architekturou a implementací aplikace.

## 2.4 Základní funkce aplikace

Aplikace obsahuje 3 uživatelské role. Konkrétně se jedná o administrátora, učitele a studenta.

### 2.4.1 Role student

- Může řešit úlohy. Úloha obsahuje sadu testovacích otázek. Úlohy se dělí do kategorií volné úlohy, zkouškové úlohy a domácí úkoly. Liší se v počtu pokusů, které má student k dispozici pro řešení každé otázky.
- Může sledovat statistiky své úspěšnosti.

### 2.4.2 Role administrátor

- Spravuje sekci často kladených otázek.
- Spravuje uživatele a přiděluje uživatelské role. Administrátor z informačního systému STAG získá seznam studentů a učitelů v souboru aplikace Excel. Tento soubor předá aplikaci a ta vytvoří vlastní databázi uživatelů, kterým přidělí roli studenta. Původně jsem chtěl využít školní databáze, ale bohužel webové služby systému STAG zatím nemají vhodné a volně dostupné rozhraní. Nicméně i mnou zvolené řešení je velice jednoduché a uživatelsky nenáročné.

### 2.4.3 Role učitel

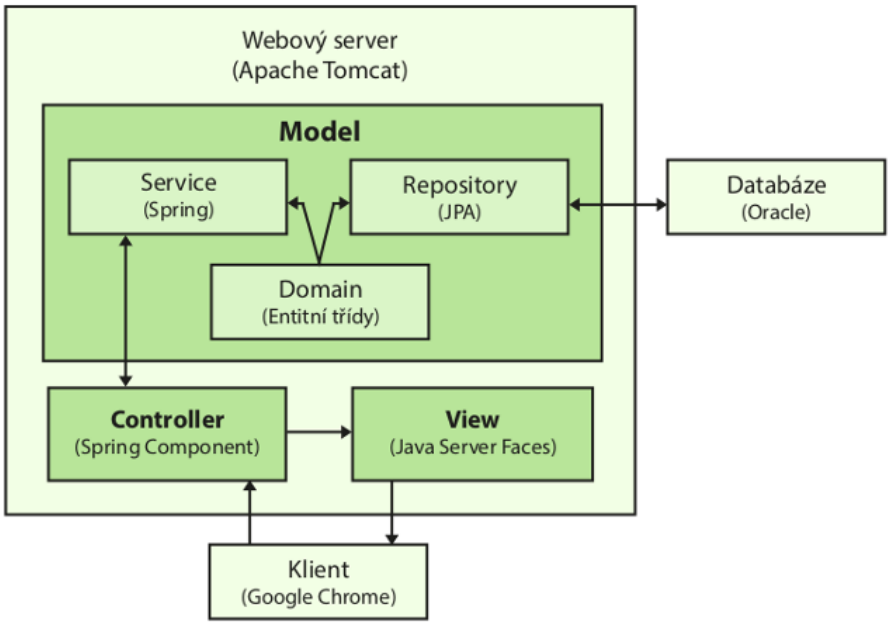
- Spravuje testovací databáze. Každá databáze obsahuje specifikaci, seznam tabulek a obrázky schémat. Pro tvorbu specifikace a relací je k dispozici HTML editor.
- Spravuje testovací otázky. Každá otázka obsahuje zadání a správné řešení. K tomu se navíc otázky řadí do 3 skupin podle obtížnosti. Tyto skupiny jsou pak barevně odlišeny.
- Spravuje úlohy. Ke každé úloze lze přiřadit libovolný počet otázek z jedné databáze. Úlohy je pak možné řešit v nastaveném čase.
- Sleduje úspěšnost studentů pomocí statistik. Tabulky lze vyexportovat do formátu PDF, XML, XLS a CSV. Grafy lze vyexportovat jako obrázek.
- Spravuje kroužky. Učitelé si třídí studenty do kroužků. Kroužek může představovat cvičení nebo zkoušku.
- Využívá nástroj pro spouštění SQL dotazů. Umožňuje mu spouštět v testovacích databázích SQL dotazy a sledovat výstup z databáze.

## 2.5 Návrhový vzor MVC a architektura aplikace

Návrhový vzor MVC (Model View Controller) je velmi rozšířený a velice dobře se uplatňuje ve webových aplikacích. Jeho hlavní myšlenkou je oddělit logickou část aplikace od jejího grafického výstupu. Aplikaci rozdělíme na 3 komponenty:

- Model - Obsahuje logiku pro práci s daty.
- View - Vykresluje data z modelu.
- Controller - Sbírá požadavky od uživatele a zprostředkovává komunikaci mezi Modelem a View.

Ovšem jak se MVC konkrétně implementuje, záleží na použité technologii. Například implementace v jazyce PHP je odlišná od implementace v jazyce Java. Často se také používá v kombinaci s jinými návrhovými vzory, případně existují jeho různé modifikace.




Obrázek 1: Architektura aplikace SQL Tester

Nyní se podíváme na architekturu aplikace SQL Tester. Držel jsem se používaného návrhu pro zvolené technologie. Základní schéma a princip komunikace mezi jednotlivými komponentami jsem zachytil na obrázku 1. Některé odborné termíny jsem ponechal v

anglickém jazyce, tak jak se běžně používají, protože jejich vynucený překlad by mohl být matoucí.

Vidíme zde webový server, na kterém je umístěna aplikace. Aplikaci podle návrhového vzoru MVC tvoří 3 základní komponenty. Model je dále členěn na další 3 vrstvy. Požadavky uživatele, které uživatel zašle prostřednictvím webového prohlížeče, zpracovává Controller a na jejich základě požádá Model o potřebná data. Konkrétně komunikuje s vrstvou zvanou Service. Ta obsahuje veškerou logiku aplikace. Pokud je možné vyřídit požadavek bez přístupu k databázi, pak jej Service zpracuje sama a vrátí Controlleru odpověď. V opačném případě je potřeba využít vrstvy Repository, která má na starost právě komunikaci s databází. Někdy se také Repository označuje jako DAO (Data Access Object) vrstva. Zde se také potkáme se třetí vrstvou Modelu, které se říká Domain. Jedná se o entitní třídy, které odrážejí schéma databáze. Pomocí těchto tříd právě komunikují vrstvy Service a Repository. Jak komunikace přesně probíhá, uvidíme v kapitole 4. Ve chvíli kdy má Controller od Service potřebná data, tak je předá View, které je zobrazí uživateli v prohlížeči.

# 3 Příprava před vývojem

K vývoji webové aplikace v jazyce Java je třeba nejprve zajistit server a vhodné vývojové prostředí. Od školy jsem dostal k dispozici přístup na virtuální server *rysy.mti.tul.cz* s operačním systémem Oracle Linux, kde jsem vše potřebné nainstaloval a zprovoznil. Nainstaloval jsem Javu 8 (Java Runtime Environment 8) a webový server, který jsem nakonfiguroval. Dále jsem na serveru nakonfiguroval firewall kvůli zpřístupnění a přesměrování potřebných portů. Databázový server jsem měl sice k dispozici již nainstalovaný, ale rovněž jsem provedl některé konfigurace jako přidělování práv nebo založení nových účtů pro testovací databáze.

## 3.1 Webový server Apache Tomcat 8

Jako webový server jsem zvolil Apache Tomcat 8. Jedná se o open-source implementaci technologií Java Servlets a Java Server Pages, které spadají do rozhraní Java EE. Hlavním úkolem webového serveru je zpracovávat HTTP požadavky od klienta. Zásadním konfiguračním souborem pro Apache Tomcat je soubor *server.xml*, který se nachází ve složce *conf*. Všechny potřebné informace o konfiguraci nalezneme v dokumentaci [4]. Alternativou mohou být aplikační servery JBoss Application Server, GlassFish nebo WebLogic.

## 3.2 Databázový server Oracle XE 11g

Databázový server se skládá z databáze a databázového řídícího systému. Pod pojmem databáze si lze představit konkrétní data uložená specifickým způsobem na databázovém serveru a databázový řídící systém je aplikace, která umožňuje s těmito daty nakládat. Někdy se lze setkat s termínem systém řízení báze dat. Pro svou práci jsem využil databázový server Oracle XE 11g, což je bezplatná alternativa verze pro komerční užití. Jiné databázové servery jsou například MySQL, Microsoft SQL Server nebo PostgreSQL.

## 3.3 Vývojové prostředí NetBeans IDE 8

IDE (Integrated Development Environment) je software, který umožňuje programátorům jednoduše psát a ladit aplikace. Česky jej lze označit jako vývojové prostředí. Pro svou práci jsem si vybral vývojové prostředí NetBeans IDE 8, na které jsem zvyklý ze školní výuky a programuje se mi v něm nejlépe. Jako open-source projekt je zcela zdarma a navíc

je multiplatformní. Lze jej tedy nainstalovat na operační systémy Windows, Linux nebo Mac OS X. Sice se jedná o prostředí primárně určené pro vývoj Java aplikací, ale s několika rozšířeními jej lze využít i pro jiné programovací jazyky. Alternativou mohou být například prostředí Eclipse nebo IntelliJ.

## 3.4 Vytvoření Maven projektu

Pro začátečníky je běžné vytvářet ve vývojovém prostředí základní Java projekty. Takové projekty jsou často na daném prostředí závislé a obsahují přebytečné soubory. Navíc je nepohodlné do nich přidávat nové knihovny. Proto se v praxi využívají nástroje, které tyto problémy řeší. Pro svůj projekt jsem se rozhodl využít velice oblíbený Maven, který kromě nezávislosti na vývojovém prostředí nabízí spoustu dalších užitečných nástrojů.

Apache Maven je software pro správu projektů a výsledné sestavování aplikací. Za jeho největší přednost se považuje správa závislostí. V reálných aplikacích je často potřeba využít několik desítek externích knihoven. Aplikace SQL Tester například obsahuje kolem 100 externích knihoven. Tyto knihovny se nachází v externích nebo lokálních repozitářích. Při využití nástroje Maven pak stačí v konfiguračním souboru *pom.xml* zadat název a verzi knihovny, kterou chceme do projektu a výsledné aplikace přidat a nemusíme ji stahovat manuálně. Ale Maven toho umí mnohem více. Proces sestavování aplikace může zahrnovat různé fáze. Například je možné vytvořit dokumentaci, spustit testy nebo nahrát aplikaci na server. Pro spuštění těchto činností slouží tzv. pluginy. Jeden z pluginů může například zajistit kompilaci aplikace, jiný zase spuštění testů. Seznam hlavních funkcionalit můžeme nalézt na oficiálních stránkách [13]. Jednoduché ovládání zajišťuje integrace Mavenu v běžných vývojových prostředích. Alternativními nástroji mohou být například Ant nebo Gradle.

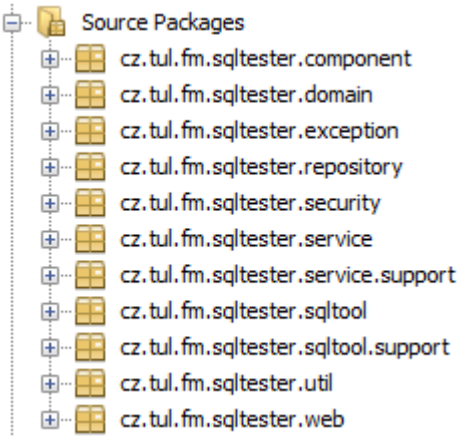
## 3.5 Verze knihoven

Mezi nejdůležitější knihovny, které jsem v práci použil patří:

- JSF - 2.2.7
- Spring - 4.1.0
- JPA - 2.1

- Hibernate - 4.3.6
- Spring Data JPA - 1.7.0
- SLF4J - 1.7.7
- Spring Security 3.2.5
- PrimeFaces - 5.0

## 3.6 Systém balíčků




Obrázek 2: Systém balíčků

V Javě se aplikace strukturuje pomocí takzvaných balíčků. Balíček je v podstatě složka na disku. Na obrázku 2 je systém balíčků aplikace SQL Tester. Základní název balíčků je *cz.tul.fm.sqltester* a poté následuje členění. Balíček *component* obsahuje Controllery, *domain* obsahuje entitní třídy, *exception* obsahuje vlastní vyjímky, *repository* obsahuje třídy pro práci s databází, *security* obsahuje bezpečnost, *service* obsahuje logiku, *sqltoo*l obsahuje zpracování SQL dotazů od uživatelů, *util* obsahuje pomocné nástroje a *web* obsahuje vše co se týká webu. Balíček *support* obsahuje podpůrné třídy, například implementace rozhraní.

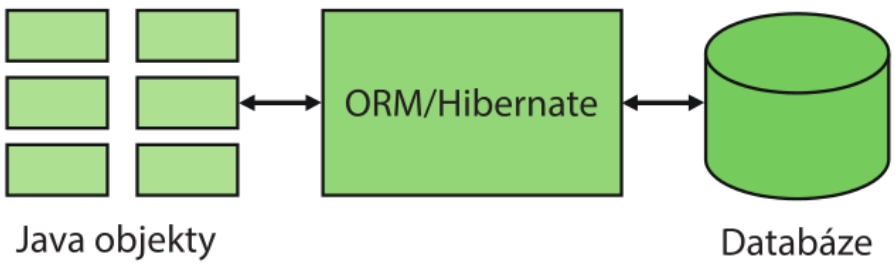
# 4 Model

## 4.1 Domain

Vrstva Modelu Domain obsahuje entitní třídy. Jedná se o třídy, které odráží schéma databáze. Tyto třídy se využívají napříč celou aplikací. Nejzásadnější jsou ovšem pro práci s databází. Objekty entitních tříd lze ukládat a načítat z databáze pomocí objektově-relačního mapování ORM (Object Relational Mapping).

### 4.1.1 ORM

Pro práci s databází v Javě slouží rozhraní JDBC (Java Database Connectivity). Umožňuje z aplikace spouštět SQL dotazy v relačních databázích. Proč tedy používat objektově-relační mapování? Programovací jazyk Java je objektový, kdežto relační databáze s objekty nepracuje. Jinými slovy Java ukládá data v objektech, ale relační databáze v tabulkách. S tímto problémem nám pomáhá ORM. Jedná se o způsob převodu dat mezi relační databází a objektově orientovaným programovacím jazykem. V programu poté místo s databázovými tabulkami pracujeme s objekty. V podstatě se jedná o abstrakci nad JDBC, kdy nemusíme psát přímo SQL dotazy a můžeme mít kód nezávislý na databázi. Další výhodou je mnohem kratší kód než při použití klasického JDBC. To umožňuje mnohem rychlejší vývoj a přehlednější kód. Na obrázku 3 vidíme znázorněné grafické schéma procesu, kde ORM vrstva zajišťuje převod mezi objekty v aplikaci a tabulkami v databázi. Popis ORM můžeme nalézt na stránkách jednoho z ORM nástrojů pro Javu Hibernate [27].



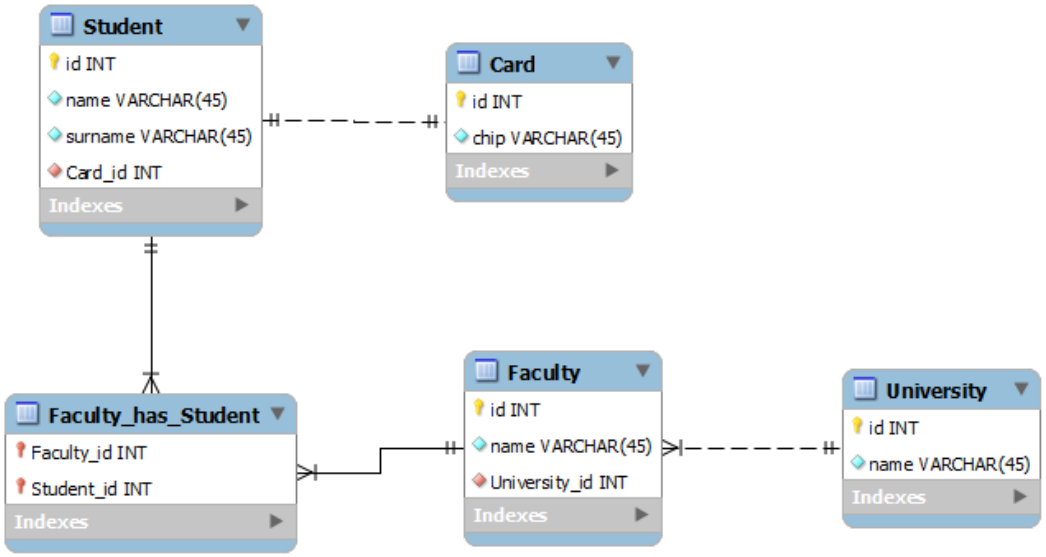

Obrázek 3: ORM schéma

## 4.2 Repository

Vrstva Modelu Repository má za úkol pracovat s databází. Je tedy třeba využít ORM řešení, které za nás bude mapovat entitní třídy na tabulky a naopak. Jednou z možností je Hibernate.

### 4.2.1 Hibernate

Hibernate je asi nejznámější ORM řešení pro Javu. V podstatě nabízí vše, co od ORM řešení očekáváme, například CRUD operace (vytvoření, přečtení, aktualizace a smazání záznamu v databázi) nebo transakční zpracování. Vše potřebné nalezneme v dokumentaci [8]. Pro lepší pochopení jak mapování na objekty funguje, jsem vytvořil jednoduchý ukázkový příklad se všemi druhy kardinalit relací. Kardinalita je četnost vztahu tedy 1:1, 1:N a M:N. Na obrázku 4 vidíme, že student vlastní právě jednu kartu a karta je vlastněna právě jedním studentem. Student může studovat na více fakultách a na fakultě může studovat více studentů. Fakulta pak spadá pod jednu univerzitu, ale univerzita může mít fakult více. Níže vidíme namapované třídy.




Obrázek 4: Databázové schéma pro ORM

```
public class Student {
    Integer id;
    String name;
    String surname;
    Card card;
    List<Faculty> facultyList;
}
```

Třída *Student* obsahuje atributy tabulky *Student* a zároveň referenci na objekt *Card* a kolekci objektů *Faculty*.

```
public class Card {
    Integer id;
    String chip;
    Student student;
}
```

Třída *Card* obsahuje atributy tabulky *Card* a referenci na objekt *Student*.

```
public class Faculty {
    Integer id;
    String name;
    List<Student> studentList;
    University university;
}
```

Třda *Faculty* obsahuje atributy tabulky *Faculty*, referenci na objekt *University* a kolekci objektů *Student*.

```
public class University {
    Integer id;
    String name;
    List<Faculty> facultyList;
}
```

Třída *University* obsahuje atributy tabulky *University* a kolekci objektů *Faculty*.

Vidíme, že relace 1:1 se mapuje na každé straně po jedné instanci objektu, relace 1:N má na jedné straně jednu instanci a na druhé kolekci objektů, no a relace M:N má na obou stranách kolekci objektů. Na první pohled si můžeme všimnout rozdílu v relacích mezi tabulkami a objekty. Zatímco relace mezi tabulkami jsou jednosměrné, tedy cizí klíč je pouze v jedné tabulce, tak u objektů jsou relace obousměrné a reference na objekt nebo kolekci objektů je na obou stranách. Tento fakt je poměrně zásadní při ukládání objektů do databáze.

Samozřejmě je nutné dodat další informace, například k jaké tabulce se entitní třída váže a k jakým sloupcům se váží jednotlivé atributy nebo jaké mají sloupce vlastnosti a vztahy. Jedna možnost je definovat tyto vlastnosti v XML souboru, ale dnes je mnohem častější využívat anotace. Já jsem ze svého příkladu záměrně anotace vypustil kvůli přehlednosti. Entitní třídy v aplikaci SQL Tester anotace obsahují.

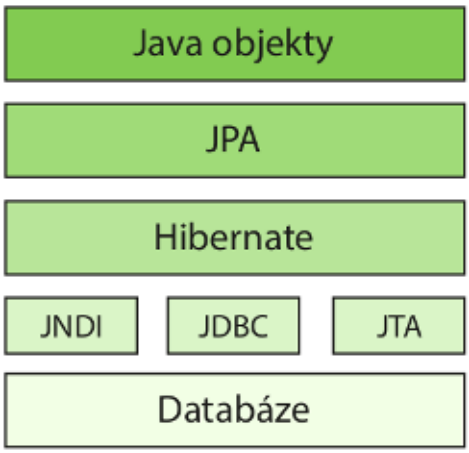
Hibernate je rovněž plná implementace rozhraní JPA (JavaPersistence API). K tomuto účelu jsem jej použil i já. Alternativní implementace je například také EclipseLink.

### 4.2.2 JPA

JPA slouží zejména k vytváření kódů nezávislých na konkrétní implementaci a je součástí specifikace Java EE. Ve výsledku tedy programátor využívá pouze třídy a metody z rozhraní JPA. Jak jsem již uvedl v kapitole předchozí, tak jsem jako implementaci zvolil Hibernate. Ale implementací může existovat více a některá může být v budoucnu například rychlejší. Pokud máme kód napsaný v JPA, pak v případě změny implementace vůbec nemusíme zasahovat do samotné aplikace, ale pouze přidáme knihovnu s jinou implementací a změníme konfigurační soubor pro JPA. V kódu je ovšem důležité neporušit architekturu programování přes rozhraní a nesmíme tedy využít žádnou třídu ani metodu z implementace, tedy například z Hibernate. Na obrázku 5 vidíme schéma, na kterém je rozhraní JPA nadřazeno Hibernate. Hibernate pak dále pracuje s API JDBC, JTA a JNDI. Více informací o JPA nalezneme například v tutoriálu [10].

#### 4.2.2.1 JPQL

Abychom mohli kromě standardních CRUD operací vytvářet také vlastní dotazy, slouží nám k tomu jazyk JPQL (JavaPersistence Query Language). Programátor tedy píše dotaz v jazyce JPQL, který je podobný jazyku SQL a implementace JPA převede JPQL právě na SQL, které se spustí v databázi. Tím máme zajištěnu nezávislost na druhu databáze.




Obrázek 5: Schéma JPA

JPQL samozřejmě nepracuje s tabulkami, ale s objekty. Příklad konkrétního JPQL dotazu uvidíme v kapitole 4.2.3, kde předvedu svojí vlastní třídu. Jak uvidíme, nepsal jsem kód přímo v JPA, ale využil jsem framework Spring Data JPA.

### 4.2.3 Spring Data JPA

Když programátor využívá JPA, tak aby si ušetřil práci a šetřil kódem, píše si takzvané generické třídy, které implementují CRUD operace pro libovolný objekt. Tyto generické třídy nabízí framework zvaný Spring Data JPA, který jsem pro svou aplikaci využil. Využití frameworku Spring Data JPA je vhodné zejména v případě, že programátor má jíž předchozí zkušenost s technologií JPA. V opačném případě může být jeho použití naopak přítěží. Já jsem se s JPA již v praxi setkal a proto jsem si řekl, že se naučím pracovat se Spring Data JPA, který mi uspořil spoustu řádků kódu. Jako hlavní zdroj informací jsem využil oficiální dokumentaci [21].

Ve vrstvě Repository se nachází pouze rozhraní, které implementuje Spring Data JPA. Tím, že rozšíříme rozhraní *JpaRepository* získáme CRUD operace. Můžeme rovněž využít vlastní JPQL dotaz pomocí anotace *@Query* a framework se za nás postará o implementaci metody. Rovněž existují určité názvy metod, ze kterých Spring Data JPA pozná, jakou operaci chceme provést. Zde je ukázka Repository objektu pro entitní třídu *Question*.

```
@Transactional(readOnly = true)
public interface QuestionRepository extends JpaRepository<Question, Integer> {

    List<Question> findByDatabase(Database database);

    @Query("SELECT q FROM Question q WHERE q.database.iddatabase = :iddatabase
        AND q NOT IN (SELECT q FROM Question q LEFT JOIN q.taskList t WHERE
        t.idtask = :idtask)")
    List<Question> findByDatabaseAndTaskNot(@Param("iddatabase") Integer
        iddatabase, @Param("idtask") Integer idtask);

}
```

Vidíme zde například metodu *findByDatabase*, ze které je patrné, že budeme hledat otázky podle databáze, ke které patří. Naproti tomu metoda *findByDatabaseAndTaskNot* využívá vlastního JPQL dotazu, do kterého se dosadí pojmenované parametry. Také vidíme, že metody jsou transakčně zpracované díky anotaci *@Transactional*.

## 4.3 Service

Vrstva Modelu Service obsahuje logiku aplikace a komunikuje s Controllerem. Controller přistupuje k této vrstvě přes rozhraní, stejně jako Service přistupuje přes rozhraní k vrstvě Repository. V této vrstvě se nám zásadněji začne projevovat framework Spring.

### 4.3.1 Spring

Spring je open-source projekt a jedná se o jeden z nejpopulárnějších frameworků pro vývoj Java EE aplikací. Můžeme díky němu psát přehledné a jednoduše testovatelné kódy. Pro běh aplikace ve Springu navíc stačí vlastnit webový server jako je například Apache Tomcat a nevyžaduje tedy využití aplikačního serveru jako je JBoss. Spring se dělí na mnoho částí. Všechny nalezneme na oficiálních stránkách [19]. Základní Spring může být využit pro tvorbu libovolné aplikace a díky dalším přídavným součástem je možné jej využít i pro aplikace webové a mobilní. Také ulehčuje implementaci návrhového vzoru MVC a nabízí transakční zpracování. Alternativní technologií může být EJB (Enterprise Java Beans).

#### 4.3.1.1 Dependency Injection

Asi nejznámější a nejvýraznější funkcionalitou Springu je využití návrhového vzoru IoC (Inversion of Control). Konkrétní provedení se nazývá Dependency Injection. V aplikaci máme často spoustu objektů, které jsou vzájemně provázány referencemi. Abychom se nemuseli manuálně starat o inicializaci objektů v konstruktoru a řešit jakou instanci vytvoříme, můžeme tuto činnost přenechat Springu, který díky tomu, že budeme používat rozhraní, za běhu zajistí inicializaci správné implementace. To zajistí například i znovupoužitelnost kódu při změně implementace. Základem Springu je kontejner, který se právě stará o vytváření a spojování objektů. Spravuje životní cyklus těchto objektů od jejich vytvoření až po jejich zánik. Objekty spravované Springem se nazývají Spring Beans. Teoreticky je tato část velmi dobře popsána v tutoriálu [22], ze kterého jsem i já vycházel.

#### 4.3.1.2 Aspektově orientované programování

Další výraznou součástí Springu je AOP (Aspect Oriented Programming). Zatím co OOP (Object Oriented Programming) člení aplikaci na objekty, AOP jí člení na moduly zvané aspekty. Díky aspektům můžeme například přidávat metodám nějakou funkcionalitu navíc před nebo po jejich vykonání. Nejčastěji se vyžívá anotací, kterými označíme metodu, která má být obohacena o externí funkcionalitu. Například můžeme zajistit logování volání metody. Výhodou tedy je, že máme funkcionalitu na jednom místě a využijeme ji na více místech. Detailnější popis AOP nabízí [3].

### 4.3.2 Validace a Verifikace SQL dotazu

Možnostmi validace a verifikace SQL dotazu jsem se zabýval již v ročníkovém projektu. Podíváme se tedy pouze na zvolené řešení. Validace je kontrola syntaxe dotazu a verifikace je kontrola jeho funkcionality. Pro validaci dotazu jsem využil knihovnu General SQL Parser, kterou jsem doplnil o vlastní funkcionality. Validátor zahrnuje následující funkce:

- Kontrola syntaxe.
- Kontrola typu dotazu. Například vytvoření tabulky, čtení dat, modifikace dat apod.
- Kontrola nepovolených slov. Vytvořil jsem seznam slov, které nesmí dotaz obsahovat.
- Kontrola SQL Injection. To znamená, že dotaz pro čtení dat čte pouze data a nic jiného.

Další třídou je spouštěč SQL dotazů. Jelikož se frameworky nehodí pro spouštění dotazů zadaných uživatelem, využil jsem rozhraní JDBC. Každá testovací databáze představuje jeden účet v databázi s omezenými právy. Není tedy možné například číst nebo modifikovat jiné databáze. Níže vidíme rozhraní s metodami, které spouští dotazy pro tvorbu tabulek, čtení dat a modifikaci dat.

```
public interface SQLRunner extends Serializable {
    public String runSelect(String sql) throws NoConnectionException,
        SQLException;
    public boolean hasMoreThanTwoResults(String sql) throws
        NoConnectionException, SQLException;
    public int runDML(List<String> sql) throws NoConnectionException,
        SQLException;
    public boolean runDDL(String sql) throws NoConnectionException,
        SQLException;
}
```

Verifikátor poté obsahuje jedinou metodu, kdy se porovná studentův dotaz se správným řešením. Využil jsem faktu, že dotazy vrací shodná data za předpokladu, že výsledkem dotazu níže je prázdná množina.

```
String query = "(" + studentSolution + " MINUS " + teacherSolution
                   + ") UNION ("
                   + teacherSolution + " MINUS " + studentSolution + ")";
```

V relační algebře zapsáno rovnicí $(R1 - R2) \cup (R2 - R1) = \emptyset$. Při splnění rovnice jsou relace R1 a R2 shodné. Před spuštěním tohoto dotazu ovšem nejprve proběhne kontrola počtu sloupců a jejich datových typů.

Poslední část této kapitoly se týká výpisu výsledků z databáze. První krok je procedura v PL/SQL, což je procedurální nadstavba jazyka SQL pro Oracle databázi. Můžeme ji nalézt v příloze D. Vrací výsledek dotazu ve formátu XML (Extensible Markup Language). Nicméně výsledek je nutné zobrazit v prohlížeči jako XHTML. K tomu slouží technologie XSLT (Extensible Stylesheet Language Transformations), která dokáže transformovat XML na XHTML. Napsal jsem si tedy vlastní univerzální transformaci pro zobrazení výsledků z databáze ve formě XHTML tabulky.

# 5 Controller

Jako Controller slouží Spring Beans, které mají tzv. scope specifický pro webové aplikace. Scope určuje počet vytvořených instancí a dobu platnosti instance objektu. Ve Springu může být scope následujícího druhu:

- Singleton - Existuje pouze jedna instance, jedná se o výchozí nastavení.
- Prototype - Existuje více instancí.
- Request - Obdoba HTTP dotazu, instance spolu s dotazem zaniká.
- Session - Obdoba HTTP session, instance spolu se session zaniká.
- Global Session - Obdoba globální HTTP session, instance spolu se session zaniká.

Já jsem si vytvořil i vlastní scope implementací rozhraní *Scope*. Konkrétně se jednalo o scope zvaný *View*. Ten má platnost po dobu zobrazení stránky v prohlížeči. Další Scope by mohl být například *Flash*, který má platnost 2 HTTP dotazy, ale rovněž bychom si jej museli implemetovat sami. Nyní se podíváme na definici konkrétního Beanu.

```
@Component("solutionBean")
@Scope("session")
public class SolutionBean implements Serializable {

}
```

Vidíme, že komponentu jsme pojmenovali *solutionBean* a má scope session. Název nám bude později sloužit pro přístup z View (komponenta MVC). V této třídě poté už můžeme mít metody přístupné z View, které budou pracovat s vrstvou Service nebo budou předávat View data pro zobrazení. Metody by samozřejmě měly být spustitelné pouze uživatelem s příslušným oprávněním. O bezpečnosti se dozvíme více v kapitole 7. Někdy se tyto Beany označují jako Model, protože udržují data k zobrazení, ale myslím si, že pokud data získávají z Modelu a neobsahují žádnou logiku, jedná se o Controllery. Jak jsme již četli v kapitole 2.5, neexistuje žádná definice jak implementovat MVC a vždy záleží na použitých technologiích a přístupu. Navíc vývojová prostředí často umožňují vygenerování kódu a NetBeans rovněž tyto objekty nazývá Controllery. S View můžeme pracovat také z Controlleru pomocí tříd *RequestContext* a *FacesContext*, viz dokumentace [20] a [6].

# 6 View

## 6.1 JSF

JSF (JavaServer Faces) je framework pro zjednodušení tvorby uživatelského rozhraní ve webových aplikacích. Dále také usnadňuje použití návrhového vzoru MVC a vytvoření šablonovacího systému. Dříve se používala technologie JSP, ale dnes převládá technologie JSF, ve které programátor píše tzv. facelety. Základní nastavení se nachází v souborech *faces-config.xml* a *web.xml*. Nyní se podíváme na konkrétní příklad JSF tagu.

```
<h:outputText value="Hello World!" />
```

Tento příklad by měl vést k vyrendrování HTML.

```
<span>Hello World!</span>
```

Ve výchozím nastavení tento tag zajišťuje escapování speciálních znaků. Pro zajištění uživatelsky přívětivého rozhraní je také důležitá podpora technologie AJAX. Ve faceletu se komunikuje s Controllerem pomocí EL (Expression Language) výrazů. Předchozí příklad bychom mohli upravit následovně.

```
<h:outputText value="#{solutionBean.question}" />
```

Životní cyklus JSF aplikace obsahuje 6 částí:

- JSF po obdržení HTTP dotazu začne vytvářet komponenty a přidělí jim handlery a validátory. Poté je uloží do FacesContextu.
- Aktualizuje hodnoty komponent na základě parametrů v dotazu.
- Hodnoty validuje vůči validátorům, které jsou ke komponentám přiřazeny.
- Uloží hodnoty do proměnných v Controlleru. Samozřejmě za předpokladu, že validace proběhne úspěšně.
- Zpracuje požadavek od uživatele, například zpracování odeslaného formuláře příslušnou metodou.
- Vykreslí odpovědi včetně případných zpráv o chybách.

Vycházel jsem z výukového materiálu [12] a tutoriálu [11].

## 6.2 PrimeFaces

Primefaces je open-source sada JSF komponent s mnoha rozšířeními. Dalo by se říci, že se jedná o JSF framework. Díky PrimeFaces komponentám můžeme vytvářet tzv. rich GUI neboli uživatelsky bohaté rozhraní, které je uživatelsky přívětivé. Může obsahovat například HTLM editor, dialogy nebo grafy. Existuje také velké množství přednastavených vzhledů a je možné si díky aplikaci [25] vytvořit vlastní. Nebo můžeme upravit CSS třídy, což jsem udělal ve své aplikaci i já. Jako příklad PrimeFaces slouží obrázek 6. Konkrétně se jedná o tabulku uživatelů, která nabízí funkce stránkování, řazení a filtrování. Více obrázků nalezneme v přílohách E, F,G, H a I.


Obrázek 6: Tabulka uživatelů v PrimeFaces

Alternativou k PrimeFaces mohou být RichFaces nebo IceFaces. V dnešní době jsou suverénně nejpopulárnější PrimeFaces a obsahují také nejvíce komponent. Porovnání využití těchto frameworků nabízí například [2]. Z firem, které používají PrimeFaces bych vyjmenoval Siemens, Cisco, Intel, Ford, Mercedes-Benz, Volvo, Boeing, eBay, Dell, Xerox nebo HP. Seznam firem lze nalézt na oficiálních stránkách [16]. Samotní vývojáři Springu doporučují jako JSF framework v kombinaci se Springem právě PrimeFaces, jak uvádí [17]. Informace jsem čerpal především z oficiální dokumentace [18], která je velice pěkně zpracovaná. Také jsem využil přídavnou knihovnu PrimeFaces Extensions. Příkladem PrimeFaces je následující kód, který se postará o vytvoření dialogového okna.

```
<p:dialog closeOnEscape="true" widgetVar="createDialog" modal="true"
    showEffect="slide" hideEffect="slide" resizable="false" >

</p:dialog>
```

## 6.3 PrettyFaces

PrettyFaces je open-source řešení pro přepisování URL v JSF aplikacích. Přepisování na hezky čitelnou URL se hodí především pro SEO (Search Engine Optimization), tedy optimalizaci pro vyhledávače. Konfigurace se nachází v souboru *pretty-config.xml*. Příkladem přepsání URL může být změna adresy z */sqltester/public/login/form.xhtml* na */sqltester/login*. Docílíme toho pomocí následující konfigurace.

```
<url-mapping id="login">
    <pattern value="/login" />
    <view-id value="/public/login/form.xhtml" />
</url-mapping>
```

Další způsoby využití můžeme nalézt na oficiálních stránkách [15].

## 6.4 Šablona

Pro moderní dynamické aplikace je samozřejmostí využití šablon. Do globální šablony v aplikaci spadá záhlaví s menu, část pro obsah a zápatí. Do hlavičky se vždy vkládá titulek a do obsahu se vkládá celé tělo stránky. V šabloně s názvem *template.xhtml* se vkládá obsah následovně.

```
<ui:insert name="content" >Content</ui:insert>
```

Poté vytvoříme stránku, která tuto šablonu bude využívat.

```
<ui:composition template="/WEB-INF/layout/template.xhtml">
```

A nakonec vložíme obsah na patřičné místo.

```
<ui:define name="content" ></ui:define>
```

Nižší úroveň šablon je dle mého názoru samotné dynamické vkládání obsahu z databáze tedy například následující kód.

```
<h:outputText value="#{user.surname}" />
```

## 6.5 Chybové stránky

K profesionální aplikaci určitě patří vytvoření vlastních chybových stránek, které mají uživatele informovat o chybě a nabídnout mu cestu na funkční stránku. Chyba vždy obsahuje stavový kód protokolu HTTP a popis stavu. Kódy začínající číslicí 1 jsou informačního charakteru, 2 znamená úspěšné zpracování požadavku, 3 přesměrování, 4 klientské chyby a 5 serverové chyby. Často se vytváří vlastní stránky alespoň pro následující kódy a i já jsem je vytvořil. Ukázka stránky se nachází v příloze J.

- 401 – Selhání autentizace nebo autorizace, například špatné přihlašovací údaje.
- 403 – Přístup zamítnut, uživatel například nemá dostatečná oprávnění zobrazit obsah.
- 404 – Stránka nenalezena, zadaná URL neodkazuje na existující zdroj.
- 500 – Interní chyba serveru, neočekávaná chyba při zpracování dotazu na straně serveru.
- 503 – Nedostupná služba, například je server v údržbě.

## 6.6 Resources

Ke každé webové aplikaci patří kromě HTML také další soubory jako CSS, javascript, obrázky apod. Pro uložení těchto souborů slouží složka *resources*. Já osobně dělím tuto složku dále ještě na složky *css*, *js*, *images*, *icons* a *fonts*. Mezi CSS mám kromě části vlastního vzhledu i CSS knihovnu Twitter Bootstrap. S její pomocí jsem vytvořil například i záhlaví stránky, ve kterém se nachází hlavní menu. Ukázka záhlaví je na obrázku 7.

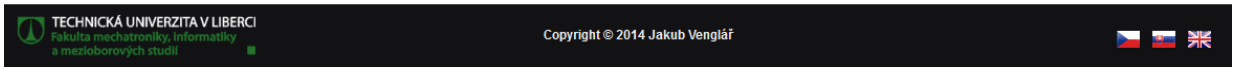

Obrázek 7: Záhlaví stránky s logem a menu

Ve složce *js* se nachází javascriptové soubory. Musel jsem v některých případech trochu poupravit chování JSF a PrimeFaces komponent. Jedná se o frameworky, které musí být univerzální a tudíž ne vždy nám musí vyhovovat jejich základní funkcionalita. Dále jsem

vytvořil tlačítko pro rychlý návrat na začátek stránky a také jsem pomocí javascriptu vyřešil lokalizaci chybových stránek, protože odeslání nového dotazu pro změnu jazyka by smazalo předchozí chybový stav. V aplikaci se nachází několik obrázků. K vytvoření favikony jsem využil generátor [7]. K vytvoření loga jsem využil nástroj [5], přičemž jsem jej upravil v grafickém editoru Gimp a pro textovou část loga jsem využil bezplatného fontu Roboto. Vlaječky pro změnu jazyka jsem převzal ze zdroje [1]. Ikonky pro export statistik jsem vytvořil sám. Poslední obrázek se nachází ve spouštění dotazů a představuje databázi. Najdeme jej i v příloze I. Ten mi vytvořila šikovná grafička Monika Jarolímová, která mi rovněž dala několik cenných rad ohledně celkového vzhledu a pomohla mi vytvořit schémata do této zprávy. Za to jí tímto děkuji. Všechny obrázky je možné využít pro soukromé i komerční účely bez nutnosti uvádět původ obrázků.

## 6.7 Lokalizace

Do aplikace jsem integroval možnost volby jazyka. Jako primární jazyk je nastavená čeština. Dále je možné rozhraní přepnout do slovenštiny a angličtiny. Jsou to jazyky, do kterých jsem byl schopný český text přeložit. Nicméně díky již hotové implementaci přepínání jazyků je přidání dalšího jazyka poměrně jednoduché. Tato možnost by se mohla využít zejména, pokud bychom se rozhodli v budoucnu aplikaci zpřístupnit veřejnosti. Na obrázku 8 vidíme v zápatí stránky vlaječky. Podíváme se, jak se taková lokalizace v JSF vytváří.




Obrázek 8: Zápatí stránky s vlaječkami

Vytvoříme ve složce *resources* takzvaný message bundle. Jedná se o soubory s názvy *messages_cs.properties*, *messages_sk.properties* a *messages_en.properties*. Samozřejmě se jedná o jinou složku *resources*, než se kterou jsme se setkali v kapitole 6.6. V každém souboru je uveden klíč a k němu hodnota. V aplikaci jako takové potom použijeme klíč, za který se dosadí hodnota z příslušného souboru podle právě zvoleného jazyka.

```
MENU_TOOLS=Nastavení
MENU_TOOLS=Nastavenie
MENU_TOOLS=Tools
```

Message bundle je potřeba zaregistrovat v konfiguračním souboru *faces-config.xml* a vytvořit session objekt pro udržování a změnu aktuálního jazyka. Zde je ukázka konfiguračního souboru.

```
<locale-config>
    <default-locale>cs</default-locale>
    <supported-locale>cs</supported-locale>
    <supported-locale>en</supported-locale>
    <supported-locale>sk</supported-locale>
</locale-config>
<message-bundle>cz.tul.fm.sqltester.locale.messages</message-bundle>
<resource-bundle>
    <base-name>cz.tul.fm.sqltester.locale.messages</base-name>
    <var>msg</var>
</resource-bundle>
```

Metoda v objektu pro změnu jazyka do slovenštiny může vypadat například takto.

```
public void changeLocaleToSk() {
    locale = "sk";
    FacesContext.getCurrentInstance().getViewRoot().setLocale(new
        Locale(locale));
}
```

# 7 Zabezpečení

## 7.1 HTTPS

HTTPS je protokol, který je podobný protokolu HTTP s tím rozdílem, že přenášená data šifruje. V běžném HTTP se data odesílají v textově čitelné podobě včetně hesel. Samotné šifrování v HTTPS zajišťuje protokol TLS (Transport Layer Security). Jedná se o asymetrickou šifru, kdy každá strana vlastní privátní a veřejný klíč. U serveru je samozřejmě nutné ověřit certifikát. Certifikát je ověřený certifikační autoritou a ta je ověřena jinou certifikační autoritou. Konečná autorita se nazývá kořenová certifikační autorita a její klíč je uložen v uložišti, tedy například v prohlížeči.

### 7.1.1 Certifikát

Pro využití HTTPS je nutné vlastnit certifikát. Jedná se o veřejný klíč, který je podepsaný certifikační autoritou. Podíváme se na postup vytvoření certifikátu pro školní server, který jsem zajistil. Nejprve je nutné vygenerovat klíč privátní a k němu žádost o elektronický podpis. Poté je nutné vytvořit konfigurační soubor pro vygenerování privátního klíče a žádosti o certifikát.

```
default_bits = 2048
distinguished_name = req_distinguished_name
string_mask = nombstr
prompt = no
req_extensions = req_ext
[req_distinguished_name]
countryName = CZ
organizationName = Technicka univerzita v Liberci
commonName = rysy.mti.tul.cz
[req_ext]
subjectAltName = @san
[san]
DNS.0 = rysy.mti.tul.cz
```

Tento soubor nazveme například *server-req.cfg* a bude nám sloužit jako vsutp pro funkci knihovny openssl.

```
openssl req -new -keyout serverkey.pem -out serverreq.pem -config server-req.cfg
```

Tímto příkazem získáme privátní klíč *serverkey.pem* a žádost o certifikát *serverreq.pem*, kterou odešleme CESNETU pomocí formuláře na stránce [28]. CESNET zařídí certifikát, který si budeme moci poté stáhnout. Jelikož používáme Javu, je nutné privátní klíč spolu s certifikátem převést do formátu pro Javu. Jedná se o formát JKS. Přímý převod z formátu PEM nefunguje. Je tedy nutné převést formát PEM na formát PKCS12 a ten posléze na formát JKS. Využijeme opět knihovnu openssl. Následujícím příkazem získáme formát PKCS12.

```
openssl pkcs12 -export -inkey serverkey.pem -in servercert.pem -name nazev -out
    keystore.p12
```

Nyní můžeme náš *keystore.p12* převést na požadovaný JKS. K tomu slouží Java nástroj KeyTool. Nachází se v běžné instalaci Javy. Následující příkaz převod zajistí.

```
keytool -importkeystore -srckeystore neystore.p12 -srcstoretype pkcs12
 -srcalias nazev -destkeystore keystore.jks
 -deststoretype jks -deststorepass heslo -destalias nazev
```

Návodů jak převést formát PEM na JKS je mnoho, nicméně po několika nejrůznějších pokusech jsem zjistil, že výše uvedený postup je jediný skutečně funkční. Soubor nám poslouží pro zprovoznění HTTPS na webovém serveru Tomcat a zároveň pro zprovoznění Shibbolethu. V souboru *server.xml* je nutné poupravit údaje pro HTTPS. Stačí odkomentovat již existující konfiguraci a přidat cestu k souboru *keystore.jks* a heslo.

## 7.2 Spring security

Spring Security je přídavná součást frameworku Spring. Poskytuje komplexní zabezpečení pro Java EE aplikace. Zabezpečení je konfigurovatelné a tak si každý může zvolit specifickou konfiguraci pro svůj systém. Nabízí například funkce pro autentizaci a autorizaci uživatelů nebo ochranu před různými útoky. Autentizace je ověření uživatele. Ověření probíhá nejčastěji prostřednictvím jména a hesla. Autorizace je ověření práv uživatele. Například zda může provést požadovanou akci. K tomu slouží nejčastěji uživatelské role. Jako hlavní zdroj informací jsem opět využil oficiální dokumentaci, viz [23].

## 7.3 Autentizace uživatelů

Mezi autentizační mechanismy patří například následující možnosti:

- Anonymní ověření - Ověření anonymního uživatele žádnou autentizaci nevyžaduje. Některé stránky v aplikaci je možné nechat zobrazit libovolnému uživateli.
- HTTP Basic - Jedná se metodu, kdy se jméno a heslo uživatele uloží do cache paměti a odešle se v hlavičce každého HTTP dotazu. Implementace je zpravidla velice rychlá, protože prohlížeče obsahují vlastní formulář pro přihlášení uživatele, nicméně je nutné předpokládat zabezpečenou komunikaci mezi klientem a serverem. Tento způsob autentizace se využívá například u nastavení síťových prvků jako jsou routery.
- Formulářové ověření - Lepší variantou je vytvořit vlastní přihlašovací formulář, data od uživatele zpracovat a informaci o jeho přihlášení udržovat na serveru. Pro zajištění bezpečnosti je nutné využít protokol HTTPS.
- LDAP - Jedná se o databázi se stromovou strukturou. Uživatele a informace o nich ukládá do adresářové struktury. Ověření uživatele vůči LDAP databázi je poměrně běžné a přívětivé. Škola sice LDAP databázi vlastní, ale od nějaké doby k ní nedává přístup, i když si myslím, že by to pro mě byla poměrně dobrá varianta autentizace uživatelů. Konfigurace je ve Spring Security poměrně jednoduchá.
- OpenID - Umožňuje nechat autentizaci uživatele na externí aplikaci. Externí aplikací může být například Google nebo mojeID. Uživatel se tak přihlásí ke svému Google účtu a tím potvrdí svou autenticitu.
- SSO (Single Sign-On) - Jedná se o službu jednotného přihlášení. Uživatel se přihlásí na jedné jediné službě, čímž získá přihlášení na všech službách, které tento mechanismus využívají. SSO tvoří poskytovatel identity a poskytovatel služeb. Příkladem může být Shibboleth.
- Zapamatování uživatele - Informace o posledním úspěšném přihlášení se uloží na lokálním počítači uživatele do tzv. cookies. Poté již stačí přečíst cookie a zjistit, zda má uživatel platné přihlášení.

Pro aplikaci SQL tester jsem pro některé stránky zvolil přístup anonymního uživatele, pro administrátorský účet využívám vlastní formulář a pro ostatní uživatele Shibboleth. Přes

Shibboleth se mohou přihlásit studenti a učitelé. Pokud by se aplikace v budoucnu otevřela veřejnosti, přidal bych i OpenID.

## 7.4 Shibboleth

Shibboleth je open-source projekt, který nabízí službu SSO. Umožňuje tedy uživateli pomocí jednoho přihlášení přístup k více aplikacím. Mezi organizace, které využívají Shibboleth patří především univerzity. Patří mezi ně i TUL. SSO systém obsahuje celkem 4 prvky.

- Webový prohlížeč – Představuje uživatele v rámci procesu SSO.
- Zdrojová data – Obsahují chráněný obsah, který chce uživatel zobrazit.
- Identity Provider (IdP) – Poskytovatel ověření, ověřuje uživatele. Například Shibboleth.
- Service Provider (SP) – Poskytovatel služby, dotazuje se IdP na ověření. Například SQL Tester.

Komunikace a výměna dat mezi IdP a SP probíhá prostřednictvím zpráv v jazyce SAML (Security Assertion Markup Language). Údaje o uživatelích jsou předávány ve formě metadat, což je XML soubor splňující právě SAML standard. Ke komunikaci je rovněž potřeba protokol TLS a metadata musí obsahovat veřejný klíč. K tomu využijeme náš certifikát, který zajišťuje i HTTPS. Samotný SSO proces pak probíhá v 5 krocích:

1. Uživatel přistupuje ke zdroji - Pokud je v bezpečnostním kontextu aplikace informace o již přihlášeném uživateli, potom je uživateli přístup umožněn. V opačném případě je požadavek na autentizaci přesměrován na server IdP s cílem zahájit SSO.
2. Přihlášení uživatele u IdP - Pokud má uživatel u IdP existující relaci, pokračuje se dalším krokem. V opačném případě ověří IdP uživatele prostřednictvím jména a hesla.
3. IdP připravuje autentizační odpověď - Po identifikaci uživatele IdP připraví autentizační odpověď ve formě SAML zprávy a odešle ji. Uživatele odešle zpět na SP.

4. SP kontroluje autentizační odpověď - SP ověří odpověď a změní informace o aktuálně přihlášeném uživateli.

5. Zdroj vrací obsah - Stejně jako v kroku 1 je uživatel ověřen vůči bezpečnostnímu kontextu. Tentokrát už je ovšem přihlášený a je mu chráněný obsah k dispozici.

Rád bych zmínil, že jsem 25. ledna 2015 zveřejnil na stránkách Wikipedie český článek o Shibbolethu, viz [26]. Článek jsem přidal pod účtem JakubVenglar. Sepsat článek bylo součástí zápočtu z předmětu Počítačová bezpečnost. Jelikož jsem se v bakalářské práci tou dobou zabýval Shibbolethem, použil jsem podobný text i pro vytvoření článku na Wikipedii. Článek na Wikipedii tedy vznikl na základě této práce.

## 7.5 Spring SAML

V kapitole 7.4 jsem zmínil, že Shibboleth komunikuje prostřednictvím protokolu SAML. Abychom s ním mohli komunikovat, využijeme knihovnu Spring SAML, která spolupracuje se Spring Security. Zvládá připojení k IdP, generování metadat, získání atributů uživatele a vše potřebné k implementaci SSO. V konfiguračním souboru je nutné nastavit cestu a heslo k certifikátu plus metadata IdP, což je v našem případě Shibboleth. Využil jsem konfiguračního souboru z ukázkové aplikace [24], který jsem poupravil dle vlastních požadavků.

## 7.6 Ochrana proti 10 nejčastějším rizikům podle OWASP

V této kapitole se pokusím nastínit 10 nejznámějších bezpečnostních rizik pro webové aplikace. U každého rizika rovněž uvádím druhy ochran v aplikaci SQL Tester. Vycházel jsem ze seznamu organizace OWASP [14].

### 7.6.1 SQL Injection

SQL Injection je technika, kdy útočník pomocí formuláře modifikuje vykonávaný SQL dotaz a přidá do něj vlastní SQL příkaz. Například bychom na serveru měli následující SQL dotaz.

```
sql = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Tento příkaz by měl vrátit maximálně jednoho uživatele. Předpokládáme, že *UserId* je unikátní identifikátor uživatele. Parametr *txtUserId* převezmeme od uživatele pomocí formulářového pole. Uživatel zadá například *125 or 1=1*. Nyní se podíváme na příkaz, který se vykoná.

```
SELECT * FROM Users WHERE UserId = 125 or 1=1;
```

Takový dotaz už získá všechny uživatele z databáze. Proti SQL Injection se dá bránit parametrizovanými dotazy. Dotaz se tedy nespojuje pomocí řetězců, ale vloží se do něj parametry, které budou parsovány jako hodnoty. Parametrizované dotazy využívám i v aplikaci SQL Tester. Níže je ukázka kódu, kde je dotazu předán parametr *role*.

```
@Query("SELECT u FROM User u WHERE u.role.name = :role")
public List<User> findByRoleName(@Param("role") String role);
```

### 7.6.2 Odcizení hesel a správa sessions

Útočník se pokouší například dešifrovat získané heslo nebo jej vidí v čitelné podobě. Proto je v aplikaci SQL Tester heslo pro administrátorský účet vytvořeno funkcí *bcrypt*. Heslo je tedy zašifrované a je k němu přidána tzv. sůl. To je řetězec, který se připojí k heslu před samotným šifrováním. Je tedy velice obtížné původní heslo dešifrovat. Ostatní uživatelé se autentizují pomocí Shibbolethu, takže zde moje aplikace nehraje roli. Komunikace s Shibbolethem probíhá přes HTTPS. Podvržení autorizace či autentizace uživatele lze provést získáním session ID. Některé aplikace jej zasílají v URL. Na serveru jsem tuto možnost zamezil a lze session ID předávat pouze pomocí cookies. Navíc při novém přihlášení uživatele se všechny sessions vytvoří znovu a staré se zneplatní. Toho jsem dosáhl prostřednictvím konfigurace Spring Security. Poslední ochrana proti této zranitelnosti je platnost sessions po dobu 30 minut. Cookies je také možné zasílat pouze prostřednictvím HTTPS. Nelze je tedy odcizit například javascriptem.

### 7.6.3 XSS

XSS (Cross-site scripting) je útok, při němž na stránce zobrazíme nežádoucí data, která nám dříve zadal uživatel. Mezi nežádoucí data řadíme například HTML nebo v horším případě javascript, který například zcela změní vzhled a funkcionalitu stránky. V aplikaci

zajišťuje ochranu proti XSS escapování speciálních znaků při výpisu.

```
<h:outputText value="#{taskBean.question}" />
```

Jedinou výjimku tvoří rozhraní, kde učitel zadává specifikaci databáze. Tady má přichystaný HTML editor, v němž není možné měnit zdrojový kód přímo. Případné speciální znaky tedy rovněž budou escapovány. Ukázka pro výpis z HTML editoru.

```
<h:outputText value="#{solutionBean.database.specification}" escape="false" />
```

### 7.6.4 Nezabezpečené přímé odkazy na objekty

Uživatel se může například pomocí URI dostat k souborům, ke kterým by neměl mít nikdo přístup. Může to být například konfigurace databáze. Já mám všechny neveřejné zdroje ve složce s názvem *WEB-INF*, která je pro uživatele nepřístupná. Také zde mám facelety, které se vkládají do šablony. To je moje osobní vylepšení návrhu oproti těm, které jsem měl na internetu možnost vidět. Některé URL jsou rovněž zabezpečeny na základě uživatelských rolí. Další možností pro útočníka je pozměnit parametr a získat data, která mu nepatří. Já načítám do paměti pouze objekty, ke kterým uživatel přístup má a jiné modifikovat nelze. Pokud by se objekt nenacházel v paměti, pak neprojde validace vstupu.

### 7.6.5 Nebezpečná konfigurace

Útočník může využít aktuální přednastavené konfigurace, která bývá často nebezpečná. Proto jsem konfiguraci pozměnil. Vytvořil jsem také vlastní chybové stránky a aplikace je v režimu ostrého nasazení a nevypisuje tedy případné chyby. Na serveru ani neexistují žádné přednastavené účty.

### 7.6.6 Zveřejnění citlivých dat

Útočník může využít přenášených dat v čitelné textové podobě, slabých zastaralých šifer apod. Na serveru jsem nastavil komunikaci výhradně prostřednictvím protokolu HTTPS. Heslo administrátora je uloženo pomocí funkce *bcrypt* a komunikace s Shibbolethem probíhá rovněž prostřednictvím HTTPS. Na některých formulářích, kde je to vhodné, rovněž zakazuji automatické doplňování na základě předchozích relací.

### 7.6.7 Chyby v řízení úrovní přístupů

Přístup k URL je řízen na základě uživatelských rolí a nepovolený uživatel si tedy nemůže zobrazit obsah, který mu není určený. Nicméně nic by mu nebránilo zavolat funkci na serveru. Proto každá funkce v Controlleru je opatřena anotací *@Secured*. Ta zajistí spuštění metody pouze uživatelem se správným oprávněním. Jelikož anotace funguje pro Spring Web MVC, musel jsem její chování poupravit, aby fungovala i v kombinaci s JSF. Využil jsem k tomu AOP.

### 7.6.8 CSRF

CSRF (Cross-Site Request Forgery) je metoda, při níž se snaží útočník odeslat formulář z cizího zdroje a vydávat se přitom za autorizovaného uživatele. Formulář z cizího zdroje v aplikaci SQL Tester odeslat zabraňuji. V aplikaci využívám takřka výhradně metody POST nikoli GET. Metody POST znesnadňují provedení CSRF, ale rozhodně mu nezabrání. K tomu slouží token. Jedná se o náhodně vygenerovanou hodnotu, která se vygeneruje při zobrazení formuláře na originální stránce. Bez tohoto tokenu nelze formulář odeslat. JSF a Spring Security v aplikaci implementují CSRF ochranu pomocí tokenu automaticky.

### 7.6.9 Použití známých zranitelných komponent

Pro svoji aplikaci jsem se snažil vybírat známé open-source projekty, které mají dobrý vývoj a dokumentaci. Zároveň jsem využil nejnovějších verzí a komponenty jsou tak aktuální.

### 7.6.10 Neošetřené přesměrování a předávání

Jelikož žádné přesměrování nevyužívám, tak je aplikace proti tomuto druhu zranitelnosti chráněna. Jediné přesměrování může být na chybové stránky.

### 7.6.11 Clickjacking

Clickjacking mezi top 10 rizik podle OWASP nepatří, ale i tak jej zmíním. Jedná se o techniku, kdy se po kliknutí na nějaký prvek na stránce, například tlačítko, provede jiná akce než očekáváme. Tlačítko může být třeba překryto neviditelným odkazem. Spring Security zajišťuje automatickou ochranu proti tomuto útoku.

## 7.7 Validace vstupů

Ke každému vstupu uživatele je přidělen validátor, který kontroluje správný datový typ a délku či rozmezí vstupu. Pro správné fungování je přidán converter, který před validací na každý vstup aplikuje funkci *trim*, která odstraní mezery na začátku a na konci vstupu. Formuláře podporují znakovou sadu UTF-8. Některé validátory jsou v JSF již vytvořené a stačí je použít, nicméně někdy je nutné si napsat vlastní validátor. Níže přikládám ukázku validátoru z JSF. Kontroluje délku vstupu v rozmezí 1 až 100.

```
<p:inputText id="questionInput" value="#{faqBean.selectedQuestion.question}"
   required="true" maxlength="100" autocomplete="off" size="50" >

    <f:validateLength minimum="1" maximum="100" />

</p:inputText>
```

## 7.8 Logování

Velice důležitou součástí aplikací je logování. Umožňuje nám sledovat chybové stavy a akce uživatelů. K logování jsem využil rozhraní SLF4J, které nabízí nezávislost na konkrétní implementaci. Asi nejznámější implementací je log4J, ale já jsem se rozhodl využít novější a rychlejší knihovnu Logback, která je od autorů knihovny log4J. Logy se zapisují do databáze. Základem je konfigurační soubor *logback.xml*.

# 8 Testování

Testování je nedílnou součástí vývoje softwaru. I já jsem jej měl jako jeden z bodů zadání. Nicméně existuje mnoho druhů testování software, a tak jsem od vedoucí musel zjistit, které testování konkrétně mám provést. Nakonec jsme se dohodli na funkčních testech a nasazení v reálném provozu. Existují například i testy jednotkové, kdy se testuje každá metoda, testy uživatelského rozhraní, testy bezpečnosti nebo testy optimalizace. Pro reálné nasazení jsme aplikaci vyzkoušeli na cvičení na větším vzorku studentů. U funkčních testů jsem zvolil následující testy:

- Vytvoření, editace a smazání často kladené otázky.
- Vytvoření uživatelů ze souboru a přidělování práv.
- Vytvoření, editace a smazání kroužků. Plus editace uživatelů v kroužku.
- Editace cvičných databází a nahrávání souborů.
- Spouštění různých druhů SQL dotazů a jejich kombinace.
- Vytvoření, editace a smazání otázek.
- Vytvoření, editace a smazání úloh.
- Možnost řešení úloh.
- Sledování statistik.

Otestoval i validátory, přístupová práva uživatelů a zkoušel jsem odeslat data, ke kterým by uživatel neměl mít přístup. V ideálním případě by bylo dobré otestovat přes 80 % všech metod, to ale není z časových důvodů v této práci možné. Tento postup se využívá spíše v praxi, kde je časová dotace na projekt mnohonásobně vyšší. Pro případné chybové stavy využívá aplikace logování a neměl by tedy být problém je případně v budoucnu dohledat a opravit.

# 9 Závěr

Během své bakalářské práce jsem se seznámil s moderními technologiemi pro vývoj webových aplikací. Konkrétně se jednalo o databázi Oracle a velice populární frameworky Spring, Java Persistence API, JavaServer Faces a PrimeFaces, které zjednodušují vývoj aplikací v programovacím jazyce Java EE. Tyto technologie se využívají zejména pro vývoj bankovních a informačních systémů. V praxi se tyto technologie velmi často objevují a využívají je i velké světové firmy. Kupříkladu UniCredit, Intel, Siemens, Dell, Cisco a mnoho dalších.

Myslím si, že práce je technologicky velice náročná díky mnoha zcela novým technologiím a také díky tomu, že jsem musel konfigurovat server, což nebylo součástí zadání. Aplikace je navíc velice nestandardní tím, že přijímá od uživatelů celé SQL dotazy a musel jsem mnohdy přijít s vlastním řešením problému.

Výsledkem práce je moderní webová aplikace pro podporu výuky databází v předmětu Databázové systémy. Automatizuje kontrolu SQL dotazů a ušetří tak učitelům spoustu času nejen na cvičeních, ale i u zkoušek. Snažil jsem se vytvořit kvalitní návrh a čistý kód, aby bylo možné aplikaci do budoucna rozšiřovat, například je možnost zpřístupnit některé testovací úlohy veřejnosti nebo i jiným univerzitám. Uživatelské rozhraní nabízí možnost volby českého, slovenského a anglického jazyka. Celkově je dle mého názoru aplikace uživatelsky velice přívětivá a působí profesionálním dojmem. Nabízí například filtrování, řazení a stránkování tabulek, export statistik do 4 různých formátů, umí zobrazovat grafy a podporuje Drag & Drop. Aplikace je velice dobře zabezpečená proti nejrůznějším útokům. Vycházel jsem ze seznamu 10 nejčastějších zranitelností podle organizace OWASP. Pro profesionálnější výstup této písemné zprávy jsem využil systému na sázení textu LaTeX.

## 9.1 Splnění cílů

- Seznámil jsem se s frameworky pro vývoj webových aplikací v Java EE. ✓
- Vytvořil jsem aplikaci pro testování SQL dotazů. ✓
- Aplikaci jsem zabezpečil. ✓
- Zvolil jsem vhodný způsob autentizace a autorizace uživatelů. ✓
- Otestoval jsem funkčnost aplikace. ✓

# Použitá literatura

[1] 2600 Flag Icon Set - Resources from GoSquared. *Real-time user-level analytics - GoSquared* [online]. [cit. 2015-05-04]. Dostupné z: `https://www.gosquared.com/resources/flag-icons/`

[2] A Blog about Liferay, JSF, Primefaces and Bootstrap: Comparing Primefaces, ICEFaces and Richfaces with Google Trend. *A Blog about Liferay, JSF, Primefaces and Bootstrap* [online]. [cit. 2015-05-04]. Dostupné z: `http://liferay-blogging.blogspot.cz/2014/01/comparing-primefaces-icefaces-and.html`

[3] AOP with Spring Framework. *Tutorials for F#, Anger Management, Social Media Marketing, AIML, Artificial Intelligence, RESTful, Swift, Node.js, LinQ, Drools, Content Marketing, SIP, Pay per Click, Accounting, Sqoop* [online]. [cit. 2015-05-04]. Dostupné z: `http://www.tutorialspoint.com/spring/aop_with_spring.htm`

[4] Apache Tomcat 8 (8.0.21) - Documentation Index. *Apache Tomcat - Welcome!* [online]. [cit. 2015-05-04]. Dostupné z: `http://tomcat.apache.org/tomcat-8.0-doc/index.html`

[5] Cool Text: Logo and Graphics Generator. *Cool Text: Logo and Graphics Generator* [online]. [cit. 2015-05-04]. Dostupné z: `http://cooltext.com/`

[6] FacesContext (JavaServer Faces API (2.0)). *Oracle Help Center* [online]. [cit. 2015-05-04]. Dostupné z: `http://docs.oracle.com/cd/E17802_01/j2ee/javaee/javaserverfaces/2.0/docs/api/javax/faces/context/FacesContext.html`

[7] Favicon.ico Generator. *Favicon.ico Generator* [online]. [cit. 2015-05-04]. Dostupné z: `http://www.favicon.cc/`

[8] Hibernate ORM documentation - Hibernate ORM. *Hibernate. Everything data. - Hibernate* [online]. [cit. 2015-05-04]. Dostupné z: `http://hibernate.org/orm/documentation/`

[9] Introduction to Databases - Stanford University — Coursera. *Coursera - Free Online Courses From Top Universities* [online]. [cit. 2015-05-04]. Dostupné z: `https://www.coursera.org/course/db`

[10] JPA Tutorial. *Tutorials for F#, Anger Management, Social Media Marketing, AIML, Artificial Intelligence, RESTful, Swift, Node.js, LinQ, Drools, Content Marketing, SIP, Pay per Click, Accounting, Sqoop* [online]. [cit. 2015-05-04]. Dostupné z: `http://www.tutorialspoint.com/jpa/`

[11] JSF - Life Cycle. *Tutorials for F#, Anger Management, Social Media Marketing, AIML, Artificial Intelligence, RESTful, Swift, Node.js, LinQ, Drools, Content Marketing, SIP, Pay per Click, Accounting, Sqoop* [online]. [cit. 2015-05-04]. Dostupné z: `http://www.tutorialspoint.com/jsf/jsf_life_cycle.htm`

[12] Kapitola 7. Životní cyklus JSF aplikace. *Java.vse.cz : Výuka programování a softwarového inženýrství na KIT VŠE* [online]. [cit. 2015-05-04]. Dostupné z: `http://java.vse.cz/jsf/chunks/ch07.html`

[13] Maven - Maven Features. *Maven - Welcome to Apache Maven* [online]. [cit. 2015-05-04]. Dostupné z: `https://maven.apache.org/maven-features.html`

[14] OWASP Top 10 - 2013 Deset nejkritičtějších bezpečnostních rizik webových aplikací. *OWASP* [online]. [cit. 2015-05-04]. Dostupné z: `https://www.owasp.org/images/f/f3/OWASP_Top_10_-_2013_Final_-_Czech_V1.1.pdf`

[15] Pretty URLs for JavaServer Faces and Java Application Servers — OCPsoft. *Simple Software — OCPsoft* [online]. [cit. 2015-05-04]. Dostupné z: `http://www.ocpsoft.org/prettyfaces/`

[16] PrimeFaces. *PrimeFaces* [online]. [cit. 2015-05-04]. Dostupné z: `http://primefaces.org/whouses`

[17] PrimeFaces. *PrimeFaces* [online]. [cit. 2015-05-04]. Dostupné z: `http://primefaces.org/whyprimefaces`

[18] PrimeFaces User's Guide. *PrimeFaces* [online]. [cit. 2015-05-04]. Dostupné z: `http://www.primefaces.org/docs/guide/primefaces_user_guide_5_0.pdf`

[19] Projects. *Spring* [online]. [cit. 2015-05-04]. Dostupné z: `https://spring.io/projects`

[20] RequestContext (primefaces 4.0 API). *PrimeFaces* [online]. [cit. 2015-05-04]. Dostupné z: `http://www.primefaces.org/docs/api/4.0/org/primefaces/context/RequestContext.html`

[21] Spring Data JPA - Reference Documentation. *Spring* [online]. [cit. 2015-05-04]. Dostupné z: `http://docs.spring.io/spring-data/jpa/docs/current/reference/html/`

[22] Spring IoC Containers. *Tutorials for F#, Anger Management, Social Media Marketing, AIML, Artificial Intelligence, RESTful, Swift, Node.js, LinQ, Drools, Content Marketing, SIP, Pay per Click, Accounting, Sqoop* [online]. [cit. 2015-05-04]. Dostupné z: `http://www.tutorialspoint.com/spring/spring_ioc_containers.htm`

[23] Spring Security Reference. *Spring* [online]. [cit. 2015-05-04]. Dostupné z: `http://docs.spring.io/spring-security/site/docs/3.2.7.RELEASE/reference/htmlsingle/`

[24] Spring-security-saml/sample at master • spring-projects/spring-security-saml • GitHub. *GitHub • Build software better, together.* [online]. [cit. 2015-05-04]. Dostupné z: `https://github.com/spring-projects/spring-security-saml/tree/master/sample`

[25] ThemeRoller — jQuery UI. *JQuery UI* [online]. [cit. 2015-05-04]. Dostupné z: `http://jqueryui.com/themeroller/`

[26] VENGLÁŘ, Jakub. Shibboleth – Wikipedie. In: *Wikipedia: the free encyclopedia* [online]. 2015 [cit. 2015-05-04]. Dostupné z: `http://cs.wikipedia.org/wiki/Shibboleth`

[27] What is Object/Relational Mapping? - Hibernate ORM. *Hibernate. Everything data. - Hibernate* [online]. [cit. 2015-05-04]. Dostupné z: `http://hibernate.org/orm/what-is-an-orm/`

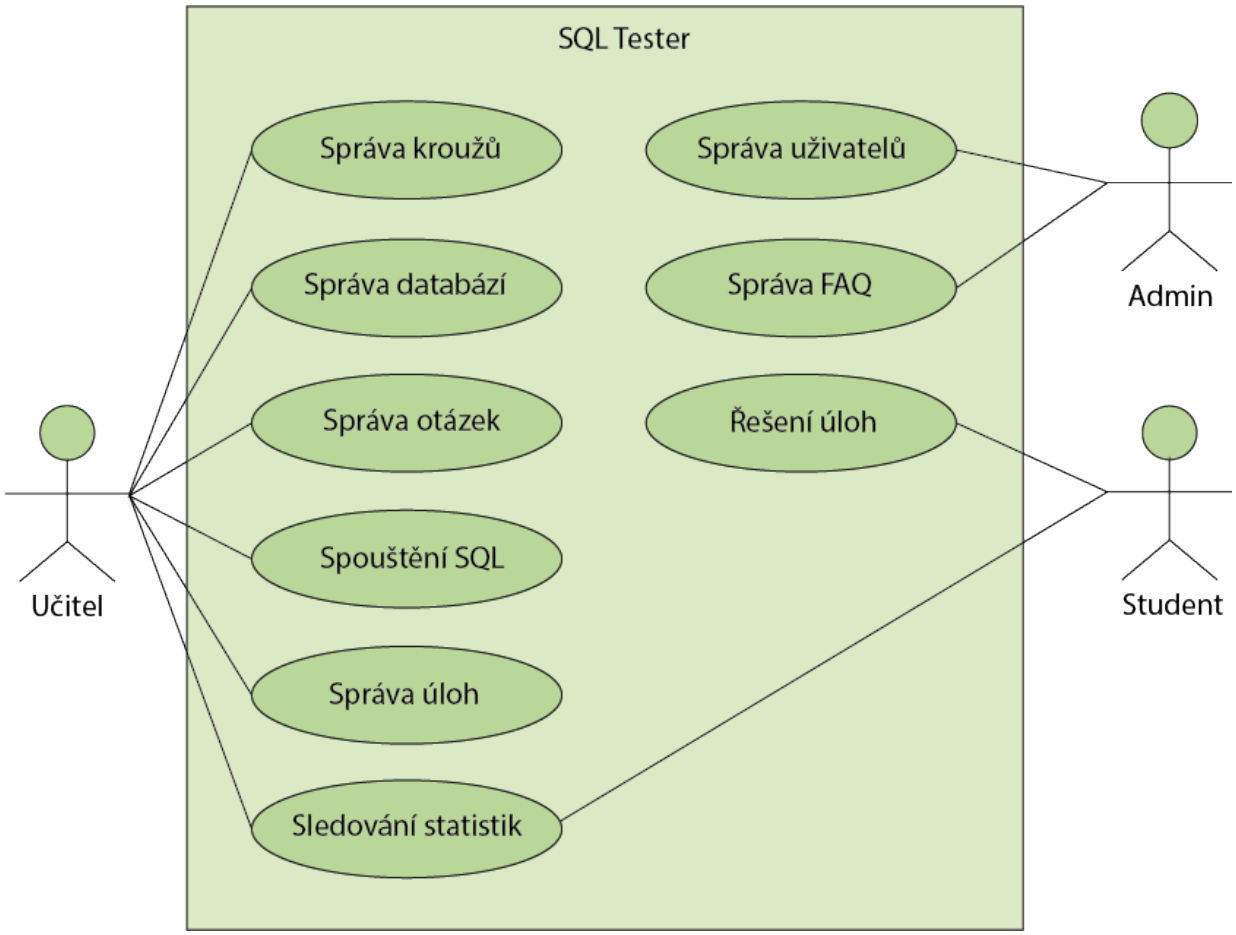
[28] Žádost o TCS serverový certifikát. *Žádost o TCS serverový certifikát* [online]. [cit. 2015-05-04]. Dostupné z: `https://tcs.cesnet.cz/req/`

# Přílohy

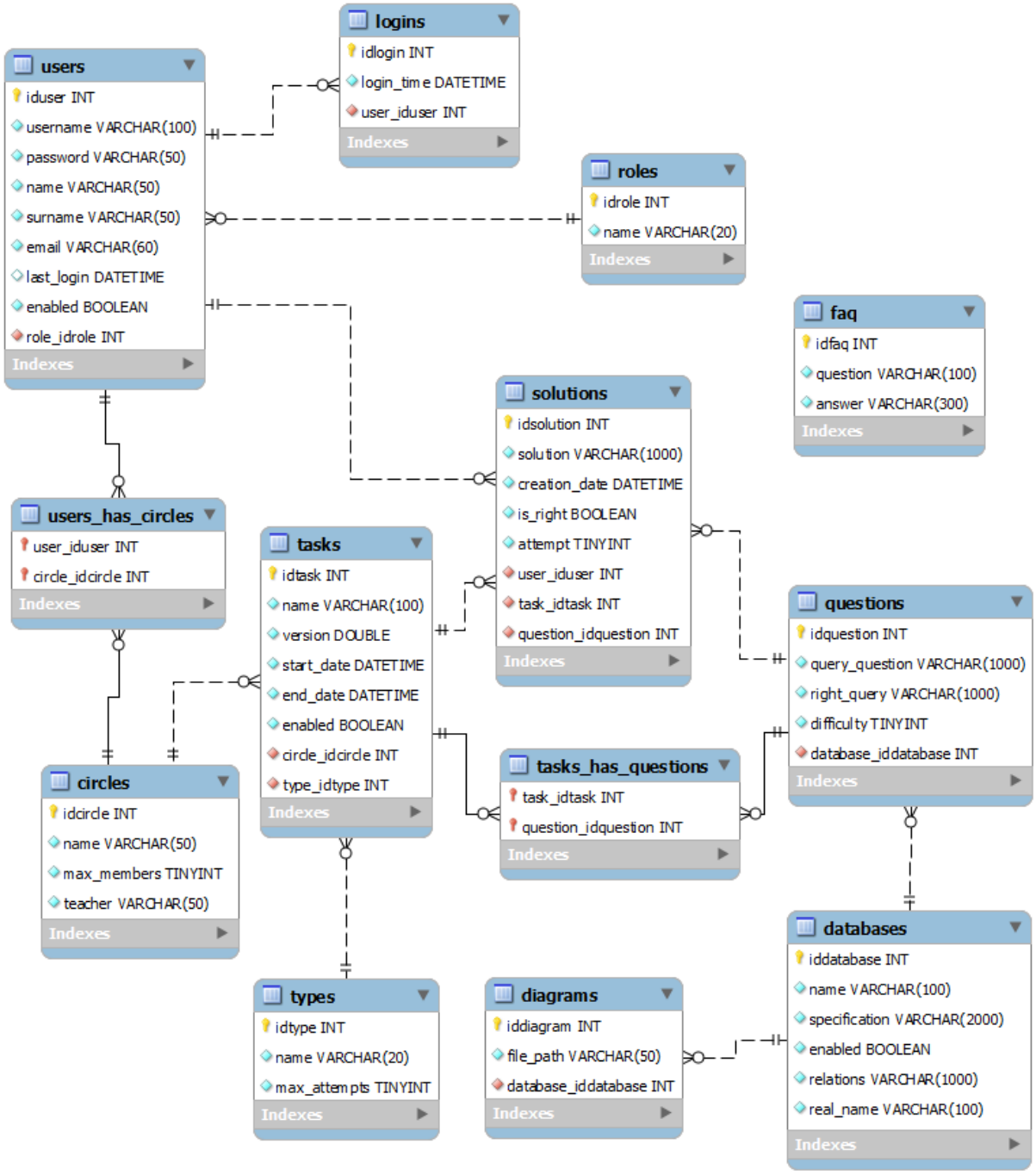
## A Obsah CD

- bp-jakub-venglar-2015.pdf (elektronická verze této práce)

# B Use Case diagram



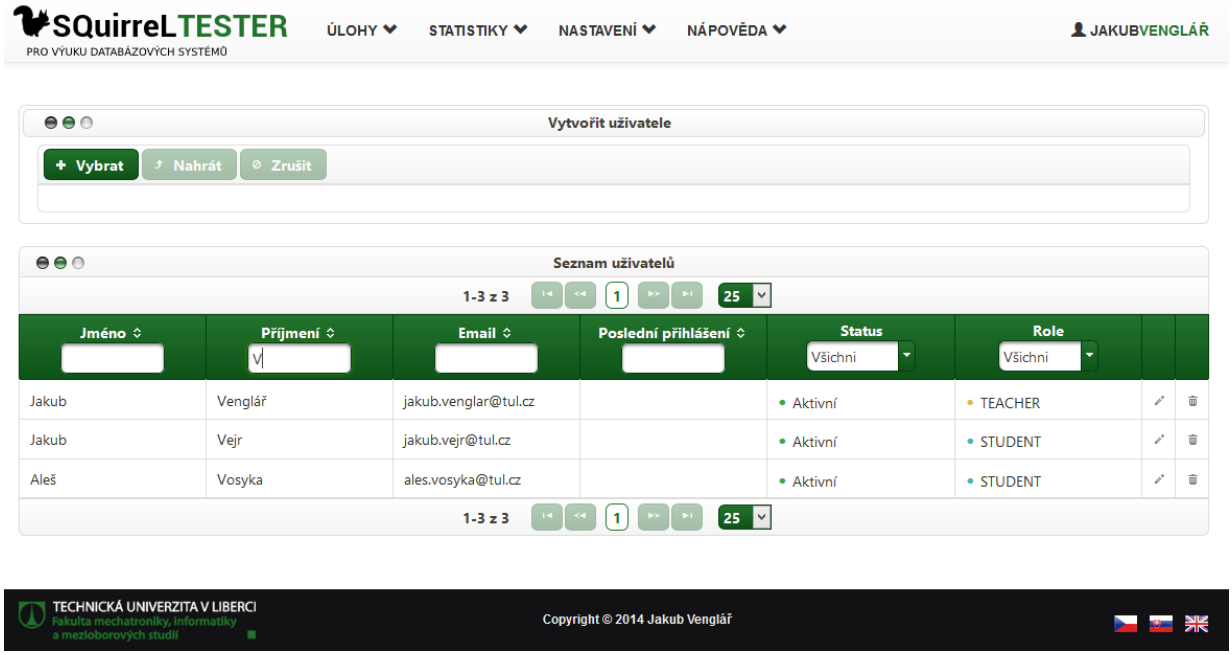
# C Relační datový model



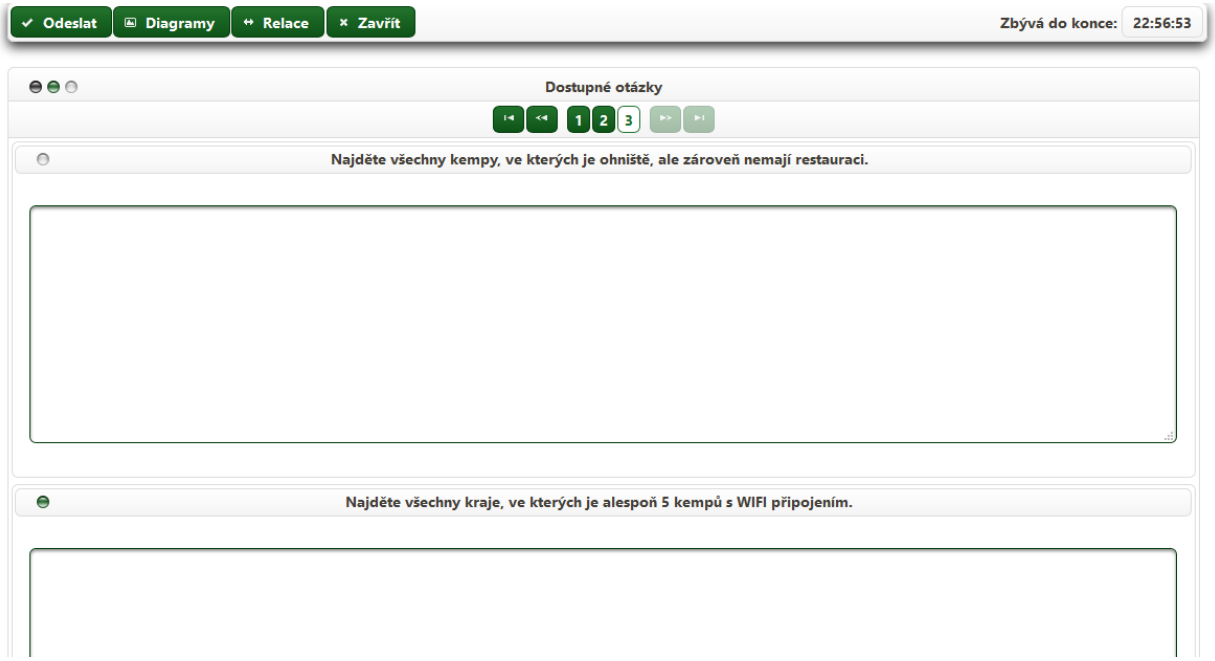
# D Procedura v PL/SQL vracející XML

```
CREATE OR REPLACE PROCEDURE SP_GET_RESULT_AS_XML(input_query IN NVARCHAR2,
    output_result OUT NVARCHAR2) AS

     BEGIN
          declare
          qryCtx DBMS_XMLGEN.ctxHandle;

               BEGIN
                    qryCtx := DBMS_XMLGEN.newContext(input_query);
                    DBMS_XMLGEN.setRowTag(qryCtx, 'SQLTESTER');
                    output_result := DBMS_XMLGEN.getXML(qryCtx);
                    DBMS_XMLGEN.closeContext(qryCtx);
                    exception
                         when others then
                         DBMS_XMLGEN.closeContext(qryCtx);
                         output_result := SQLERRM;
               END;

     END;
```

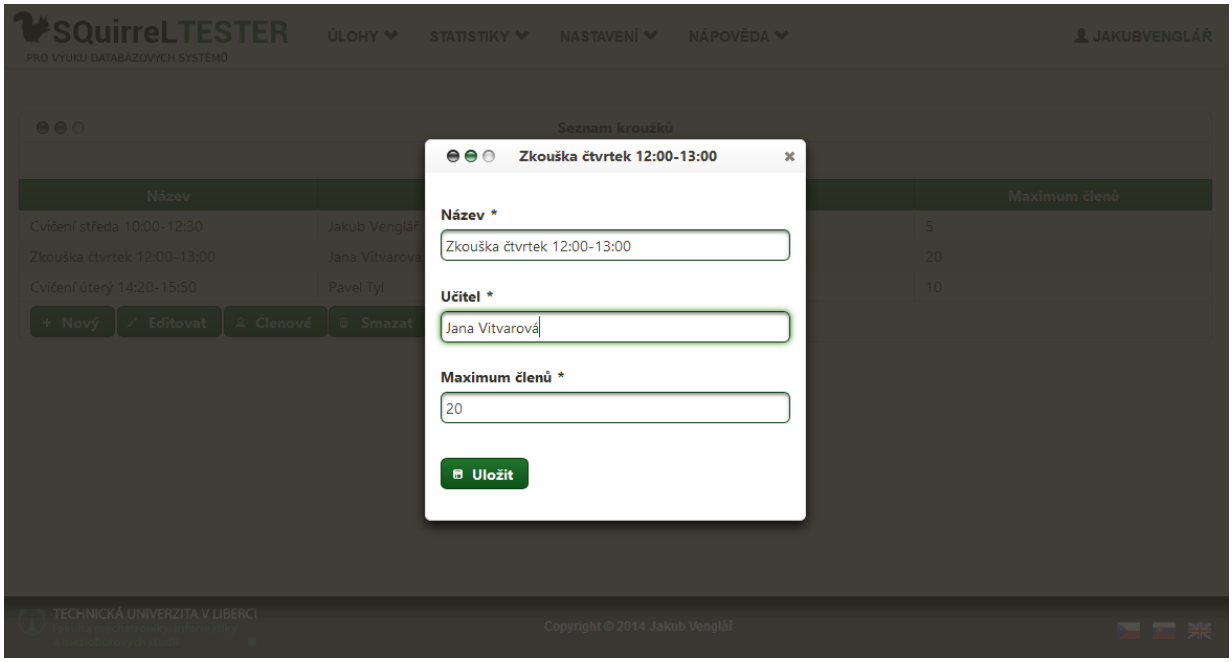
# E Administrace uživatelů



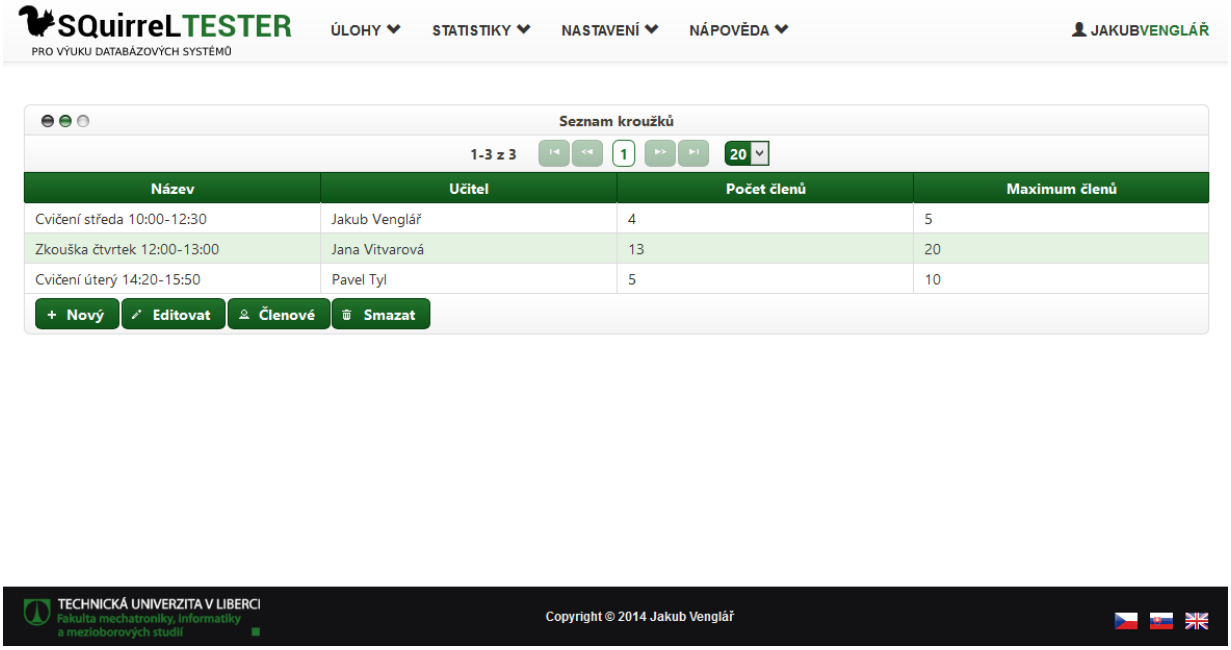
# F Řešení úloh



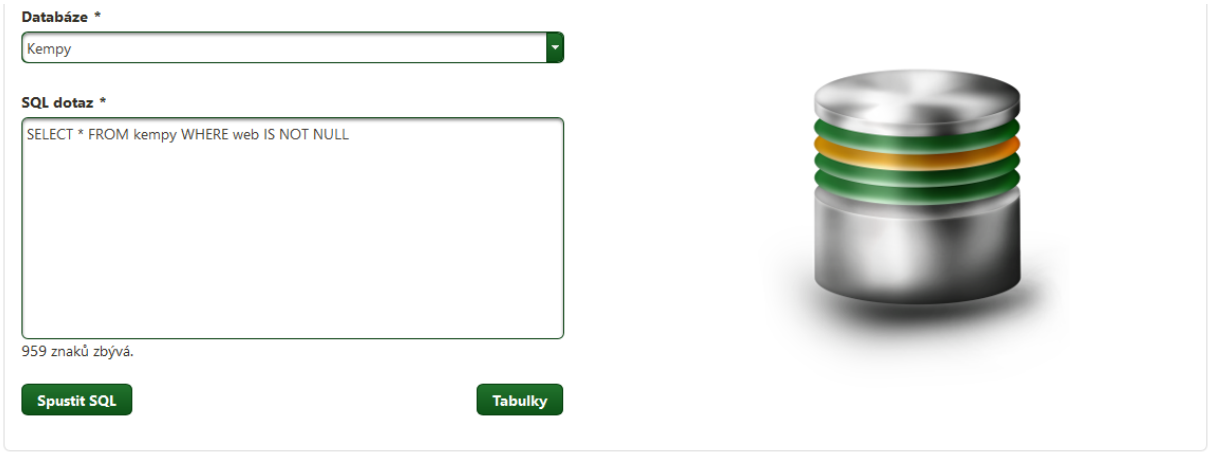
# G Dialog editace kroužku



# H Správa kroužků



# I Spouštění SQL dotazů




Výsledky dotazu a zprávy z databáze

| ID_KEMPU | ID_MESTA | NAZEV_KEMPU | WEB |
|---|---|---|---|
| 1 | 1 | Bucek | www.bucek.com |
| 2 | 2 | Visnova | www.visnova.com |
| 6 | 2 | Bucek | www.bucek.com |

# J Chybová stránka