Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Diplomová práce

Vývoj webové aplikace pomocí moderních technologií

Bc. Ahliddin Ibragimov

! ! !

Místo této strany vložíte zadání diplomové práce.

(Do jedné vazby originál a do druhé kopii)

! ! !

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Vývoj webové aplikace pomocí moderních technologií" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání _____

Poděkování

Chtěl bych poděkovat všem, kteří mi jakkoli pomohli při tvorbě mé diplomové práce, ať již morálně, či radou. Jmenovitě bych pak chtěl poděkovat svému vedoucímu, panu Ing. Martinu Havránkovi, Ph.D., rodičům a manželce za podporu.

Vývoj webové aplikace pomocí moderních technologií

Web application development with modern technologies

Souhrn

Tato práce řeší vývoj webové aplikace použitím moderních programovacích jazyků a technologií. Cílem této práce je studium a analýza moderních technologií, zvolení jednotlivých programovacích jazyků, frameworků, databázi a následující vývoj webové aplikace použitím zvolených technik.

Na základě analýzy moderních trendů zvolil jsem následující technologie: vývoj serverové části použitím populárního frameworku Spring, založeného na Java EE, implementace klientské části využitím nejpopulárnějšího skriptovacího jazyku JavaScript a konkrétně framework AngularJS vyvinutého společností Google. Pro ukládání dat jsem využil NoSQL databáze MongoDB.

Detailně jsem zdokumentoval jednotlivé implementační kroky, zahrnující definici projektu, analýzu požadavků, návrh, implementaci a testování.

Summary

This thesis covers the development of web applications with modern programming languages and technologies. The objective of this thesis is the research and analysis of modern technologies and subsequent selection of particular programming languages, frameworks and database for development of web application applying those techniques.

Based on analysis of modern trends I chose the following technologies – backend development using popular Spring framework based on Java EE, frontend implementation using one of the most widespread scripting languages JavaScript and its known framework AngularJS developed by Google. For data persistence I chose NoSQL database MongoDB.

I provided detailed documentation of each implementation step – project definition, requirements analysis, design, implementation and testing.

Klíčová slova: webová aplikace, Java, Java EE, Spring, JavaScript, AngularJS, NoSQL, MongoDB, aplikační server, Tomcat.

Keywords: web application, Java, Java EE, Spring, JavaScript, AngularJS, NoSQL, MongoDB, application server, Tomcat

Obsah

1 Úvod 6

1.1 Cíle práce 6

1.2 Metodika 7

2 Teoretická východiska 8

2.1 Aplikace na straně serveru 8

2.2 Databáze 8

2.2.1 NoSQL 8

2.2.2 Dokumentově orientovaná databáze 9

2.2.3 MongoDB 9

2.3 Java EE 10

2.4 Spring framework 10

2.4.1 Motivace pro Spring Framework 10

2.4.2 Představení Spring Frameworku 11

2.4.3 Spring jako komponentová technologie 13

2.4.4 Spring Context 14

2.4.5 DAO a objektově-relační mapping 15

2.4.6 Spring Web 15

2.5 Aplikace na straně klienta 16

2.6 JavaScript 16

2.7 AngularJS framework 17

2.7.1 Hlavní rysy AngularJS 17

2.8 Testování 19

2.8.1 Testování serverové části 19

2.8.2 Testování klientské části 21

3 Zahájení projektu 22

3.1 Výběr příkladu webové aplikace pro praktickou část práce 22

3.2 Účastníci projektu 22

3.3 Řízení projektu 23

3.4 Analýza trendů ve sféře vývoje webových aplikací 24

3.4.1 Serverová část 25

3.4.2 Klientská část 26

3.4.3 Databáze 27

4 Analýza požadavků 31

4.1 Funkční požadavky 31

4.1.1 Přihlášení a registrace uživatelů 31

4.1.2 Hledání slov a překladů 31

4.1.3 Nastavení osobního účtu 31

4.1.4 Administrace 31

4.2 Nefunkční (technické) požadavky 31

4.2.1 Serverová strana aplikace 32

4.2.2 Klientská strana aplikace 32

4.2.3 Databáze 32

4.2.4 Spouštění aplikace 32

5 Architektura webové aplikace 33

5.1 Scénář zpracování požadavku – technický pohled 34

5.2 Databázový model webové aplikace 35

5.2.1 Vnořený datový model 35

5.2.2 Normalizovaný datový model 36

5.2.3 Vnořený datový model webové aplikace 37

5.2.4 Normalizovaný datový model webové aplikace 38

5.3 Případy užití webové aplikace 40

6 Vývoj 42

6.1 Vývoj klientské části aplikace 42

6.1.1 Design a vzhled 42

6.1.2 Řídící logika klientské části 48

6.2 Vývoj serverové části aplikace 52

6.2.1 Konfigurace projektu 52

6.2.2 Konfigurace Spring Framework 54

6.2.3 Řídící logika serverové části 56

6.3 Konfigurace databáze 58

6.3.1 Instalace MongoDB 58

6.3.2 Nastavení databáze 58

6.3.3 Importování dat 59

7 Testování 60

7.1 Testování na straně serveru 60
7.2 Testování na straně klienta 60
8 Závěr 63
8.1 Náměty na další rozvoj 63
Seznam použitých zdrojů 64
Seznam obrázků 66
Seznam tabulek 67
Obsah přiloženého CD 68

1 Úvod

12. srpna 1981 představila společnost IBM hardware platformu osobního počítače (PC) běžícího na operačním systému od Microsoft MS-DOS, který se ovládal pomocí textové konzole. V 90. letech však byla textová konzole nahrazena grafickým uživatelským rozhraním v operačních systémech Microsoft Windows. Poté v roce 1991 byla představena celosvětová síť WWW a v roce 1993 první webový prohlížeč Mosaic. Tyto historické milníky byly základem pro prudký vývoj informačních technologií označovaný jako *technologický boom*, který pokračuje i v současné době.

I když prvních deset let počet uživatelů internetu stoupal velice rychle (podle Internet World Stats počet uživatelů od roku 1995 do 2005 vzrostl ze 16 milionů na 1 018 miliard [1]), webové prohlížeče sloužily téměř výhradně k pasivnímu prohlížení webového obsahu. V posledních letech s růstem rychlostí připojení k internetu a prudkým rozvojem webových technologií a cloud-computingu ovšem z webu vykrystalizovala jakási univerzální platforma dostupná na širokém spektru zařízení.

Současné webové aplikace buď již nahradily úplně, anebo jsou častou alternativou tradičním desktopovým aplikacím pro běžné uživatelské potřeby (například Spotify a YouTube místo Winamp, Google Translate a The Free Dictionary místo ABBYY Lingvo, Google Docs a Office 365 místo desktopového Microsoft Office). Jelikož se rozšiřují i do prostředí firem a finančních sfér, rostou také požadavky na nepřetržitost, stabilitu a odolnost webových aplikací, což působí na vývoj nových frameworků s podporou webu pro jazyky programování, které si zasloužily důvěru programátorů ve sféře desktopových aplikací (Java, C#).

1.1 Cíle práce

Cílem této diplomové práce je vývoj webové aplikace použitím moderních programovacích jazyků a technologií. Vývoj serverové části použitím populárního frameworku Spring, založeného na Java EE, implementace uživatelské části využitím nejpopulárnějšího skriptovacího jazyku JavaScript a konkrétně framework AngularJS, vyvinutého společností Google. Pro ukládání dat byla zvolena NoSQL databáze

MongoDB. V této práci bude zdokumentován vývojový cyklus aplikace zahrnující definici projektu, analýzu požadavků, návrh, implementaci a testování.

1.2 Metodika

V rešeršní části této práci budou uvedeny definice projektu, požadavky a základní teoretická východiska zvolených programovacích jazyků a technologií, jež jsou použity v praktické části. Srovnání s konkurujícími technologiemi, pozorování trendů, velikost open-source komunity, známé silné a slabé stránky jednotlivých programovacích jazyků, frameworků a databází. Rešerše možností testování serverových a klientských strán aplikace.

Praktická část se bude skládat z detailního popisu jednotlivých částí aplikace a potřebných implementačních kroků: proces návrhu architektury aplikace a databázového modelu, základní komponenty systému podle návrhového vzoru MVC (Model, View, Controller), možné problémy a komplikace při integraci frameworků a při procesu sestavení a nasazení kódu. Praktická část bude s ohledem na testování zahrnovat metodiku automatizace testů uživatelského rozhraní a serverové části.

2 Teoretická východiska

V této sekci popíši základní teoretická východiska, která jsou nezbytná pro vývoj aplikace a pochopení její architektury.

2.1 Aplikace na straně serveru

Serverová část neboli *aplikace na straně serveru* slouží pro dotazování do databáze, hledání a zpracování informací, provedení různých výpočtů a celkovou podporu takzvané business logiky aplikace.

Serverová část – klientská část (v angličtině více známé jako *backend-frontend*) je nejčastějším schématem pro oddělení zodpovědností (anglický Separation of Concerns – SoC), podle kterého se dnes staví webové aplikace s dynamickým obsahem.

2.2 Databáze

Databáze je nenahraditelnou součástí moderních webových aplikací. Umožňuje ukládání uživatelských a aplikačních dat, které se používají pro autentizaci a autorizaci, podporu dynamického obsahu a rychlé vyhledávání potřebných informací.

2.2.1 NoSQL

Termín *NoSQL* poprvé použil Carlo Strozzi pro pojmenování souborové databáze, kterou sám vyvinul v roce 1998 [2]. Byla to právě relační databáze, jenom nepoužívala pro dotazování nad daty jazyk SQL. Termín byl použit znovu v roce 2009, ale už pro pojmenování technologií nerelačních databází, jež jsou stále populárnější.

Za vznikem tohoto nového přístupu k databázím stojí potřeba ukládání a zpracování enormně rostoucí množství dat především pro velké hráče ve sféře informačních technologií jako Google, Facebook, Amazon, což je vedlo k potřebě vyřešit především dva problémy. A to jak data rozložit z důvodu uložení a dostupnosti a zároveň jak nad touto nepředstavitelně velkou množinou dat vykonávat dotazy, na které by byl systém schopen odpovědět v desítkách či stovkách milisekund. Tedy tak rychle, aby byl uživatel u počítače po odeslání požadavku vůbec ochoten čekat na výsledek jeho zpracování.

Největším průkopníkem v této oblasti byla společnost Google, jež v samostatných pracích publikovala v říjnu 2003 principy Google File System (GFS), v prosinci roku 2004 princip široce aplikovaného dotazovacího modelu MapReduce a v listopadu 2006 představila na těchto základech postavený systém distribuovaného úložiště BigTable, ze kterého dále vychází valná většina sloupcově orientovaných (column-oriented) NoSQL systémů. Na její práci navázala v roce 2007 také společnost Amazon se svým projektem distribuovaného datového úložiště Dynamo [3].

2.2.2 Dokumentově orientovaná databáze

Dokumentově orientované databáze logicky navazují na systémy typu klíč-hodnota. Základním prvkem pro ukládání dat je zde dokument, kterým je míněn objekt s jedinečným identifikátorem a množinou párů klíč-hodnota, jež mohou být do sebe dále libovolně vnořovány. Na rozdíl od systémů typu klíč-hodnota většinou umožňují také sekundární indexování atributů a co do funkcionality jsou robustnější. Pro ukládání dat užívají JSON, XML nebo podobné strukturované formáty [3].

2.2.3 MongoDB

MongoDB je dokumentově orientovaná databáze, která pro ukládání dat užívá JSON format. MongoDB patří mezi dokumentově orientované databáze, které nemají schéma a přejímají mnoho konceptů ze světa relačních databází. Pro lidi zvyklé na svět SQL je MongoDB pravděpodobně nejsnadnější způsob, jak nahlédnout do světa dokumentových databází.

MongoDB funguje jako klasický databázový server, ke kterému se klienti připojují přes síť (obvykle na portu 27017) a komunikují s ním speciálním protokolem. MongoDB má i jednoduché HTTP rozhraní obvykle běžící na portu 28017. Součástí instalace je knihovna pro C++, jako součásti projektu jsou ale vyvíjeny a podporovány i knihovny pro Javu, Python, Ruby, PHP a další jazyky. Pro účely administrace a dotazování má instalace MongoDB javascriptovou konzoli. Pokud byl databázový server spuštěn s implicitním nastavením, stačí pro otevření příkazové konzole spustit příkaz *mongo* bez parametrů [4].

Základní příkazy MongoDB:

`show dbs` – seznam databází

`use <název_databáze>` – nastaví danou db na <název_databáze>

`show collections` – seznam kolekcí v dané databázi

`show users` – seznam uživatelů dané databáze

2.3 Java EE

Podniková neboli enterprise aplikace je softwarová aplikace, kterou firma vyvinula, zakoupila nebo převzala pro účely poskytování strategických služeb dané firmě. Může se jednat o libovolné podnikové procesy (účetnictví, sklad), ERP (Enterprise Resource Planning – plánování a správa výrobního procesu), CRM (Customer Relationship Management – správa vztahu se zákazníky). Dříve byly tyto aplikace úzce svázány s mainframy a například jazykem Fortran.

Java Enterprise Edition (neboli **Java EE**, dříve označovaná jako *Java 2 Enterprise Edition* nebo *J2EE*) je součást platformy Java určená pro vývoj a provoz podnikových aplikací a informačních systémů.

Základem pro platformu Java EE je platforma Java SE, nad ní jsou definovány součásti tvořící Java EE.

Součástí platformy Java EE jsou především specifikace pro:

- vývoj webových aplikací – Java Servlets, Java Server Pages (JSP), JavaServer Faces (JSF),
- Contexts and Dependency Injection – vkládání závislostí,
- přístup k relačním databázím – Java Persistence API,
- vývoj sdílené business logiky – Enterprise Java Beans (EJB),
- přístup k legacy systémům – Java Connector Architecture (JCA),
- přístup ke zprávovému middleware – Java Messaging Services (JMS),
- komponenty zajišťující integraci webových aplikací a portálů – Portlety,

podpora technologií Webových služeb [5].

2.4 Spring framework

2.4.1 Motivace pro Spring Framework

Spring umožňuje **snadné věci dělat jednoduše**, avšak při zachování vysoké škálovatelnosti a rozšiřitelnosti. Dále celé rozhraní klade velký důraz na dobré praktiky v programování (bestpractices) – například možnosti v testování. Spring je stále ve vývoji a rychle reaguje na nové techniky ve vývoji obchodních aplikací, které třeba jinde ještě nejsou aplikovány.

Možnosti jednoduchých zásahů do samotného frameworku jsou dalším kritériem, které by mohlo rozhodovat při volbě mezi J2EE a Springem. Spring je nesmírně modulární, jedná se o sadu rozhraní s mnoha různými implementacemi. V neposlední řadě je možné zasahovat přímo do zdrojového kódu (Spring jako takový je open-source softwarem), včetně možnosti vytvořit si vlastní aplikační rámec nad Springem a dále jej distribuovat, nebo se přímo účastnit vývoje a participovat se na dalších verzích.

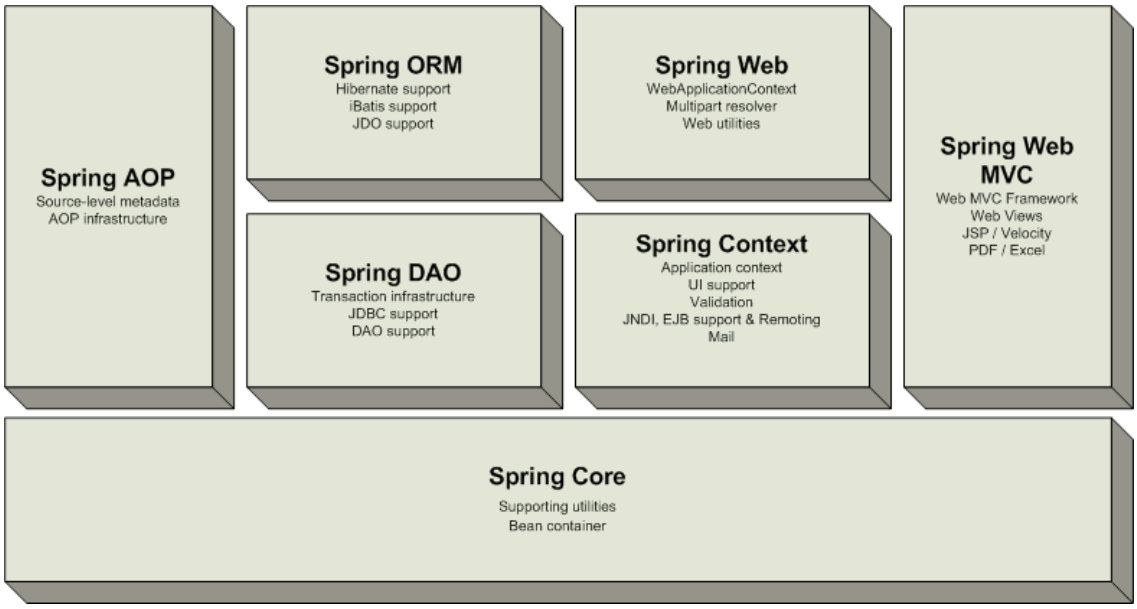
Spring nevyžaduje žádné závislosti. Ačkoli má distribuce Springu kolem 100 MB, vlastní jádro není závislé na žádné knihovně a až s postupem projektu vývojový tým přidává nové závislosti. Není potřeba žádného aplikačního serveru, avšak ta možnost tady je.

Díky vpíchnutí závislostí aplikace nejsou přímo závislé na Springu a nic programátorovi tak nebrání aplikaci zprovoznit nad jinými aplikačními rozhraními. V neposlední řadě nabízí Spring jednotnou konfiguraci, která je také snadno pochopitelná [6].

2.4.2 Představení Spring Frameworku

Hlavním cílem Spring Framework je vytvoření jednoduché, ale účinné komponentové technologie založené na moderních postupech. Spring se snaží pokrýt veškeré oblasti spojené s vývojem obchodních aplikací s důrazem na databáze, web a aspektově orientovaného programování.

Základnímy komponenty Spring jsou kontext aplikace (podpora konfigurace, prostředků, UI, validace, adresářových služeb či zasílání pošty), databázová vrstva (DAO, transakce), O/R mapování a webové programování (modely, pohledy, způsob propojení modelů). K tomu všemu ještě Spring přidává AOP.



Obrázek 2.1: Komponenty Spring Frameworku. Zdroj: [6]

Jak je vidět na obrázku 2.3, klíčovou vrstvou je jádro nazvané Spring Core, které představuje vlastní komponentovou technologii (paralela s EJB), dále kontejner pro tyto komponenty a potřebné nástroje. K tomuto účelu se využívá popsaná technologie vpíchnutí závislostí. Nad tímto jádrem se nacházejí balíky popsané detailjněji níže.

Nejdůležitějším je Spring Context, který obsahuje instancované komponenty (služby) pro email, JNDI, validaci a internacionalizaci a poskytuje informace pro nastavení aplikace během runtime.

Spring DAO (Data Access Object) je cílen pro usnadnění práci s technologie pro přistup k datům jako JDBC, která je zavislá na konkretním databázovém řešení. Spring DAO právě nabízí abstrakci JDBC a umožňuje snadné přepínání mezi databázové řešení.

Nad Spring DAO stojí ORM (Object Relational Mapping). Spring ORM poskytuje integraci s nástroje pro objektově relační mapování jako Hibernate a EclipseLink.

Spring AOP modul nabízí napojení na aspektově orientované programování v Javě, která umožňuje přidávání dalších vlastností softwarový komponentě (metoda, třída). AOP jde ruku v ruce právě s vpíchnutím závislostí a je základním stavebním kamenem Spring Framework [6].

Balíky Spring Web a Web MVC společně zobecňují webové aplikační rozhraní. Spring Web poskytuje bázové pomocné třídy pro webové aplikace a také slouži jako

integrační bod pro implementaci Struts a JSF. Spring MVC je nadstavbou Spring Web poskytující alternativu pro jiné MVC řešení.

2.4.3 Spring jako komponentová technologie

Komponenta je ve Springu obyčejná javovská třída (POJO – Plain Old Java Object – starý dobrý obyčejný objekt). Díky dostatečné podpoře platformy Java (reflexe, RMI, atd.) není nutno zavádět žádná specifika a jelikož je celý framework určen právě pro tuto platformu Java, komponenty nazývají názvem Beans (resp. dlouze: Java Beans). Jako Bean se totiž označuje libovolný java objekt, který má jednu nebo více vlastností.

Jak již bylo zmíněno, Spring využívá vpíchnutí závislostí. Závislosti komponent se načítají z XML souboru nebo pomocí Java Configuration anotací. Zde je krátký příklad definování komponent pomocí XML:

```
<bean id="exampleBean" class="examples.ExampleBean">
  <property name="database"><ref bean="db"/></property>
  <property name="numConnections"><value>32</value></property>
</bean>

<bean id="db" class="examples.DatabaseBean"/> [6]
```

Stejný příklad, ale pomocí Java Configuration:

```
@Component
public class TestBean {
      private int numberOfConnections;

      @Autowired //instance of DatabaseBean is injected
      private DatabaseBean database;

      @PostConstruct   //executed after instance created
      public void init () {
            this.numberOfConnections = 32;
      }
}

@Component
```

```
public class DatabaseBean {
      ...
}
```

2.4.4 Spring Context

Součástí frameworku je kontext aplikace usnadňující některé operace, které musí programátor často řešit. Prvním je API pro jednotný přístup k prostředkům (soubor na disku, v kontextu webové aplikace, na síti nebo z *classpath*). Podpora snadné lokalizace je také součástí tohoto balíku.

Podpora validace vstupů jde ruku v ruce s navázáním vstupních hodnot na doménové objekty. K těmto účelům se používají rozhraní *DataBinder* a *Validator*. K těmto účelům se využívá standardního API definovaného přímo firmou Sun pro JavaBeans, pomocí kterého si Spring *hlídá* změny v datových objektech a tyto hodnoty validuje pomocí Validator API.

Validace probíhá tak, že v konfiguračném XML souboru nebo pomocí Java Configuration anotací stačí u vlastnosti zadat daný validátor a ten implementovat. V něm se dá použít předdefinované validátory pro jednotlivé vlastnosti objektu (JavaBeanu) . Pokud některý z vestavěných (například celé číslo, řetězec o minimální délce, nebo regulární výraz) nebude vyhovovat, můžeme definovat vlastní, jak uvedeno níže.

```
public class PersonValidator implements Validator {
        public boolean supports(Class clzz) {
                return Person.class.equals(clzz);
        }
        public void validate(Object obj, Errors e) {
                ValidationUtils.rejectIfEmpty(e, "name", "name.empty");
                Person p = (Person)obj;
                if (p.getAge() < 0) {
                        e.rejectValue("age", "negativevalue");
                } else if (p.getAge() > 110) {
                        e.rejectValue("age", "tooold");
                }
        }
} [6].
```

2.4.5 DAO a objektově-relační mapping

Najde se jen málo obchodních aplikací bez databází, proto Spring nabízí několik metod pro přístup k nim. Použitá technika DAO je nezávislá na přístupu. Spring se nesnaží jako J2EE vytvořit vlastní vrstvu pro ukládání objektových dat, ale používá k tomu špičkové open-source projekty, které jsou ověřené trhem i časem.

Ve Springu lze používat přímo rozhraní JDBC, přičemž se Spring snaží zastřešit a sjednotit některé věci. Jednak je to vlastní přístup k databázi (otevření spojení) a dále obsluha výjimek. Spring také definuje API zastřešující SQL dotazy pro zjednodušení vytváření mapujících tříd, což jsou třídy převádějící výsledky volání JDBC vrstvy na databázové objekty.

Hlavní síla Springu je bezesporu ve výborné integraci s knihovnami Hibernate, Oracle TopLink, EclipseLink. Poskytují spojení mezi doménovými objekty a relační databází. Princip spočívá v tom, že JavaBeans komponenty a jejich vlastnosti jsou namapovány (pomocí konfiguračních souborů) do databázových tabulek. Knihovna samotná se postará o jejich správné vyzvednutí, ošetřuje kolekce (spojování dotazů), výjimky a cachování [6].

2.4.6 Spring Web

Webová část Spring frameworku zastřešuje kompletní MVC (Model, Vew, Controller) přístup pro tvorbu webových aplikací. Ačkoli programátorovi nic nebrání použít jiný nebo vlastní aplikační rámec (Struts, Tapestry, Velocity/Freemaker, XLST, Tiles, JasperReports), použití implementace ve Springu má řadu výhod:

- *čistá separace všech rolí* – v mnoha MVC rámcích se slučují pojmy jako form a model, validator a model a podobně,
- *jednoduchá konfigurace* – Spring používá samozřejmě na všechno JavaBeans komponenty a s tím je spojena řada dalších výhod,
- *znovupoužití business tříd* – není potřeba vytvářet další objekty typu Form, ve Springu jsou na to přímo business komponenty,
- *nezávislé mapování* – handlery a pohledy jsou nezávislé a lze použít více přístupů.

Spring obsahuje přímou podporu JSP a také podporu technologii JSF.

Všechny klíčové komponenty (Controller, Model, View) jsou součástí Springu a ten navíc obsahuje mnoho připravených implementací. Typická aplikace pomocí Spring MVC je nakonfigurovaná tak, aby na straně J2EE kontejneru předávala všechny dotazy na jeden jediný servlet. Tento servlet pak načítá vlastní konfiguraci a mapování z XML souborů nebo pomocí Java Configuration [6].

2.5 Aplikace na straně klienta

Klientská část neboli *frontend* pochází z oblasti programování webových aplikací, kde slouží k označení části webu viditelné běžným návštěvníkům, jinými slovy je to uživatelské rozhraní webové aplikace.

Na rozdíl od serverové části bývá klientská část většinou mnohem lépe propracována po všech stránkách, zejména z hlediska přístupnosti, použitelnosti a vzhledu.

I když mezi programátory je dobře známé, že při vytvoření webové aplikace vyžaduje vývoj serverové strany více času a energie a mnohem složitější výpočty a algoritmy, pro koncového uživatele je však důležitější to, co vidí před sebou, což je souhrn designu, intuitivního ovládání a rychlosti.

2.6 JavaScript

JavaScript je objektově orientovaný programovací jazyk využívaný při tvorbě webových stránek. JavaScript představuje implementaci standardizovaného skriptovacího jazyka ECMAScript, který je normovaný neziskovou organizací *ECMA International* podle specifikace ISO/IEC 16262:2011 [7].

Na rozdíl od serverových programovacích jazyků (například PHP), sloužících ke generování kódu samotné stránky, JavaScript běží na straně klienta, tedy v prohlížeči až po stažení do počítače.

JavaScript se používá především pro vytváření interaktivních webových stránek. Příkladem použití mohou být nejrůznější kontroly správného vyplnění formulářů, obrázky měnící se po přejetí myší, rozbalovací menu atd. JavaScript se také často používá k měření statistik návštěvnosti.

Společně s jazykem HTML (informační kostrou stránky) a CSS (formátováním vzhledu stránky) je JavaScript součástí DHTML – souboru technik a postupů zaměřených na zlepšení uživatelského rozhraní a zvýšení prožitku z používání stránek. K tomu

JavaScript využívá tzv. DOM – rozhraní umožňující přistupovat k jednotlivým prvkům stránky.

JavaScript byl vyvinut v roce 1995 společnosti Netscape. Následně byl v roce 1998 standardizován organizací ISO.

JavaScript je také možné použít pro programování backendu. Existuje několik implementací JavaScriptu na straně serverů. David Flanagan popsal ve své knize *JavaScript The Definitive Guide 6th edition* implementace Rhino, Node.js.

2.7 AngularJS framework

V této podkapitole stručně uvedu a popíšu hlavní rysy frameworku AngularJS. Jak je vidět na obrázku 2.3, AngularJS je v momentu psaní této práce nejpopulárnějším JavaScript frameworkem pro vývoj frontendu.

AngularJS je *open-source* framework, který je založený na architektuře MVC (Model-View-Controller), skládá se tedy z následujících tří částí: *Model* reprezentuje data a business logiku, *View* zobrazuje uživatelské rozhraní a *Controller* má na starosti události v aplikaci a obecně aplikační logiku.

I když AngularJS si říká Model-View-Whatever framework, zásadní rozdíl oproti MVC u něj není. Vše, co není aplikační logika a není View (šablony), je právě to *Whatever*. V podstatě ve Whatever jde (stejně jako v Controlleru) jen o to zavolat nějaké API aplikační logiky a aktualizovaná data poslat zpět do View [8].

2.7.1 Hlavní rysy AngularJS

Two Way Data-Binding je koncept, který řeší synchronizaci stavů mezi Modelem a View. Jedná se o automatické propojení hodnoty například <input> a javascriptové proměnné bez potřeby psaní ani řádky kódu navíc. Mezi programátory je považován za nejužitečnější vlastnost frameworku.

Níže je uveden příklad HTML stránky, kde se využívá *Two Way Data-Binding.* Hodnota *yourName,* která se načítá z pole vstupu, se bude automatický vypisovat za běhu v tagu <h1> místo {{yourName}}.

```
<!doctype html>
<html ng-app>
  <head>
```

```
  <script src="http://code.angularjs.org/angular-
     1.0.0rc10.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="yourName"
placeholder="Enter a name here">
      <hr>
      <h1>Hello, {{yourName}}!</h1>
    </div>
  </body>
</html>
```

Šablony v AngularJS jsou jednoduché HTML stránky, takzvané *Plain-Old-HTML.* Prohlížeč parsuje HTML šablonu do objektového modelu dokumentu (DOM), který pak slouží jako vstup pro kompilátor AngularJS. AngularJS zprocesuje vstupní DOM a vyhledá instrukce pro rendering neboli *direktivy,* jež jsou podrobněji popsány níže.
Je důležité si uvědomit, že AngularJS nezpracovává šablonu jako řetězce (string). Vstupem je DOM, který vygeneruje prohlížeč, a tenhle princip je právě největším rozdílem mezi AngularJS a jinými frameworky, protože to umožňuje rozšířit knihovnu HTML direktiv a vyvinout vlastní direktivy.

Dependency Injection neboli „vpíchnutí“ závislostí je vestavěným (*built-in*) rysem frameworku, který usnadňuje vývoj, testování a čtení kódu pro vývojáře.

Dependency Injection umožňuje snadné poptávání závislostí jednoduchým přidáním určité služby do parametru funkce, kterou framework najde sám a „vloží“ do daného parametru. Služby jsou v Angularu *singletony* a *lazily instantiated* (instance se vytvoří až po prvním vyžádání).

Například v níže uvedené definici AngularJS funkci, pří jejím volání, se framework sám postará o vložení instancí *$scope, $http,* a *$routeParams* do parametrů.

```
function EditCtrl($scope, $http, $routeParams) {
    //kód pro edit...
```

```
}
```

Direktivy umožňují rozšíření HTML o nové možnosti, ať již formou nových HTML elementů jako například nebo <input>, nebo přidáním atributů ke stávajícím elementům.
Níže je jednoduchý příklad definování direktivy v AngularJS:

```
myModule.directive('myComponent', function(mySharedService) {
    return {
        restrict: 'E',
        controller: function($scope, $attrs, mySharedService)
        {
            $scope.$on('handleBroadcast', function() {
                $scope.message = 'Directive: ' +
                    mySharedService.message;
            });
        },
        replace: true,
        template: '<input>'
    };
});
```

Takhle se direktiva pak využívá v HTML kódu:

```
<my-component ng-model="message"></my-component>
```

2.8 Testování

2.8.1 Testování serverové části

Pro automatizaci testů webových aplikací založených na Spring MVC existuje od stejné komunity framework *Spring MVC Test*, který poskytuje prvotřídní JUnit podporu pro testování Spring MVC aplikaci. Spring MVC Test vždy používá instanci *Dispatcher Servlet* pro zpracování požadavků a má vlastní *TestContext*, do které nahrává aktuální

konfigurace aplikace, čímž maximálně napodobuje reálný proces a umožňuje integrační testy.

Níže je příklad testu pomocí Spring MVC Test frameworku.

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration("test-servlet-context.xml")
public class ExampleTests {
    @Autowired
    private WebApplicationContext wac;
    private MockMvc mockMvc;
    @Before
    public void setup() {
        this.mockMvc = MockMvcBuilders.
                    webAppContextSetup(this.wac).build();
    }

    @Test
    public void getAccount() throws Exception {
        this.mockMvc.perform(get("/accounts/1")
.accept(MediaType.parseMediaType("application/json;charset=UT
F-8")))
        .andExpect(status().isOk())
        .andExpect(content().contentType("application/json"))
        .andExpect(jsonPath("$.name").value("Lee"));
    }
} [9]
```

Pomocí anotace *@RunWith* se nastaví speciální *ClassRunner* od JUnit *SpringJUnit4ClassRunner,* poskytující funkcionalitu Spring TestContext. Pomocí anotace *@ContextConfiguration* se předá buď XML soubor, nebo třída obsahující deklaraci konfiguračních beanů, které se při spouštění aplikace nainstalují a potom se díky funkcionalitě vpíchnutí závislostí přidělují proměnným anotovaným *@Autowired.*

2.8.2 Testování klientské části

Jedním z hlavních cílů autorů AngularJS bylo umožnit snadné testování. Při generování skeletonu Angular projektu pomocí angular-seed se vygeneruje také nakonfigurovaný runner pro *end-to-end* a *unit* testy. Pro funkcionální testování uživatelského rozhraní webových aplikací existují takzvané *record/playback* nástroje, které umožňují nahrávání nějakých akcí proti webové stránce (např. klikání, vyplňování formulářů atd.) a přehrávání těchto akcí v budoucnu jako regresní test. Nejznámější z těch nástrojů jsou *Selenium WebDriver* a *HP QuickTest Professional.* Hlavní nevýhodou HP QTP v rámci této práce je to, že HP QTP používá jenom programovací jazyk Visual Basic. Na rozdíl od něj Selenium WebDriver podporuje programovací jazyky jako *Java, C#, Python, Ruby, PHP, Perl* a *JavaScript.* Stačí stáhnout příslušnou distribuci pro určitý programovací jazyk. Po nahrání akcí Selenium vygeneruje kód, který po spuštění provede úplně stejné akce, které byly nahrány.

3 Zahájení projektu

3.1 Výběr příkladu webové aplikace pro praktickou část práce

Hlavním cílem práce je analýza, srovnání moderních technologií a následující vývoj webové aplikace, v níž budou využity zvolené technologie.

Jako příklad webové aplikace jsem zvolil sociální slovník, který umožňuje uživatelům editování slov a jejich překladů, přidávání nových slov, ukládání slov do osobního slovníku a administraci uživatelských požadavků.

Důvody pro výběr daného příklad webové aplikace jsou následující:

1. Tato aplikace zahrnuje většinu případů, kdy běžný uživatel použije webové stránky nebo aplikaci, které se skládají z vytvoření účtu (registrace), přihlašování, změny obsahu webové stránky, nastavení osobního účtu, odhlašování. To je dostačující pro základní analýzu výhod a nevýhod užívání určitých technik pro vytvoření webové aplikace.
2. Kombinace akademického přínosu výsledků práce spolu se sociálním přínosem. Tádžický jazyk je stejný s perštinou, jediný rozdíl je v písmech. V době, kdy byl Tádžikistán součástí Sovětského svazu, byla abeceda státu změněna z arabských písmen na cyrilici, což způsobilo to, že vznikla informační bariéra mezi dvěma státy mluvícími stejným jazykem až na to, že většina Tádžiků nemůže využít například perský slovník v Google Translate, jenž je již v docela zralém stavu s velkou databází slov. Implementovaný slovník v rámci této práce bude podporovat tádžičtinu, pro kterou v době psaní práce zatím neexistuje online slovník s velkou databází slov a možností přidávání a úpravy překladů. Nejpopulárnější online slovník Google Translate začal podporovat tádžičtinu v prosinci 2014 [10], ale slovník je stále ještě daleko od dokonalosti.

3.2 Účastníci projektu

Základním kamenem slovníku je kvalitně přeložená slovní databáze. Vzhledem k tomu jsem projekt rozšířil o jednoho člena zodpovědného za přepisování fyzických kopií slovníků do jejich elektronických podob, které budou importovány do databáze webové aplikace.

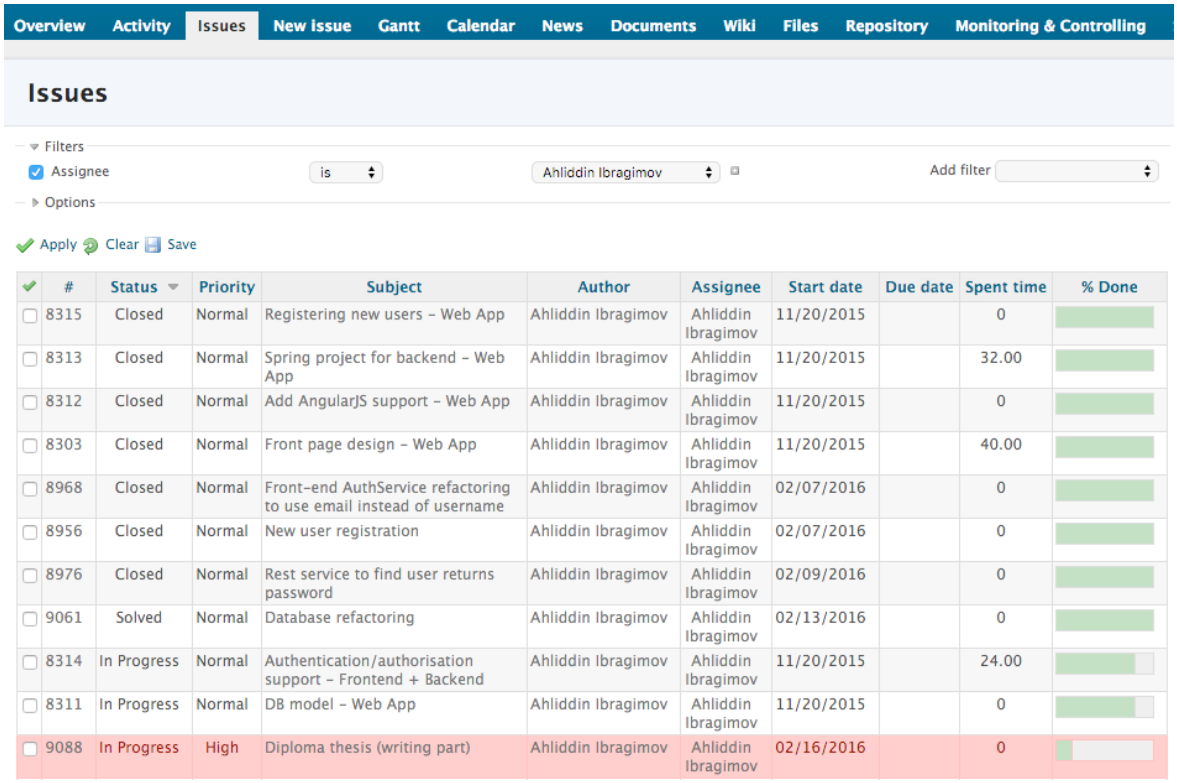
3.3 Řízení projektu

Nástroje pro řízení projektů jsou v dnešní době nezbytnou součástí každého významného projektu. Pro řízení projektů v IT a konkrétně ve sféře vývoje software existuje několik nástrojů: Jira, Redmine, Trac, Bugzilla atd.

Pro řízení procesu vývoje dané práce, sledování a řešení jednotlivých úkolů jsem zvolil nástroj Redmine.

Redmine má hostovanou verzi pro volné využití pro registrované uživatele s následujícími vlastnostmi:

- podpora více projektů,
- flexibilní systém řízení přístupů založený na rolích,
- flexibilní systém pro správu úkolů,
- Ganttův diagram a kalendář,
- novinky, správa dokumentů a souborů,
- oznámení pomocí e-mailu a zdrojů (feeds),
- wiki pro každý projekt,
- fórum pro každý projekt,
- snadné sledování času u jednotlivých projektů,
- uživatelská pole pro úkoly, časové vstupy, projekty a uživatele,
- integrace SCM (SVN, CVS, Git, Mercurial, Bazaar a Darcs),
- podpora vícenásobné LDAP autentizace,
- uživatelé se mohou sami registrovat do systému,
- podpora více jazyků,
- podpora více databází [11].



#	Status	Priority	Subject	Author	Assignee	Start date	Due date	Spent time	% Done
8315	Closed	Normal	Registering new users – Web App	Ahliddin Ibragimov	Ahliddin Ibragimov	11/20/2015		0	
8313	Closed	Normal	Spring project for backend – Web App	Ahliddin Ibragimov	Ahliddin Ibragimov	11/20/2015		32.00	
8312	Closed	Normal	Add AngularJS support – Web App	Ahliddin Ibragimov	Ahliddin Ibragimov	11/20/2015		0	
8303	Closed	Normal	Front page design – Web App	Ahliddin Ibragimov	Ahliddin Ibragimov	11/20/2015		40.00	
8968	Closed	Normal	Front-end AuthService refactoring to use email instead of username	Ahliddin Ibragimov	Ahliddin Ibragimov	02/07/2016		0	
8956	Closed	Normal	New user registration	Ahliddin Ibragimov	Ahliddin Ibragimov	02/07/2016		0	
8976	Closed	Normal	Rest service to find user returns password	Ahliddin Ibragimov	Ahliddin Ibragimov	02/09/2016		0	
9061	Solved	Normal	Database refactoring	Ahliddin Ibragimov	Ahliddin Ibragimov	02/13/2016		0	
8314	In Progress	Normal	Authentication/authorisation support – Frontend + Backend	Ahliddin Ibragimov	Ahliddin Ibragimov	11/20/2015		24.00	
8311	In Progress	Normal	DB model – Web App	Ahliddin Ibragimov	Ahliddin Ibragimov	11/20/2015		0	
9088	In Progress	High	Diploma thesis (writing part)	Ahliddin Ibragimov	Ahliddin Ibragimov	02/16/2016		0	

Obrázek 3.1: Seznam úkolů, softwarových chyb a jejích stav v Redmine. Zdroj: vlastní projekt na Redmine

3.4 Analýza trendů ve sféře vývoje webových aplikací

Tato podkapitola slouží pro analýzu a srovnání trendů konkurujících technologií pro vývoj webové aplikace.

Webová aplikace se skládá ze tří součástí:

1. klientská část – reprezentuje prezentační vrstvu webové aplikace,
2. serverová část – slouží pro provedení různých výpočtů a přístup k databázi,
3. databáze – slouží pro dotazování a ukládání generovaných dat a informací.

Ve většině případů jsou klientská a serverová část vyvinuty použitím sady různých technologií.

Pro srovnání určitých programovacích jazyků, frameworků a databází jsem zvolil následující kritéria:

- trend popularity odpovídajících klíčových slov podle Google Trends;
- počet dotazů použitím odpovídajících klíčových slov ve Stack Overflow;
- velikost komunity.

Pro analýzu trendů jsem využil Google Trends, který na základě analýzy uživatelských vstupů a počtu hledaných výrazů buduje diagram popularity neboli trendu určitého slova či výrazu.

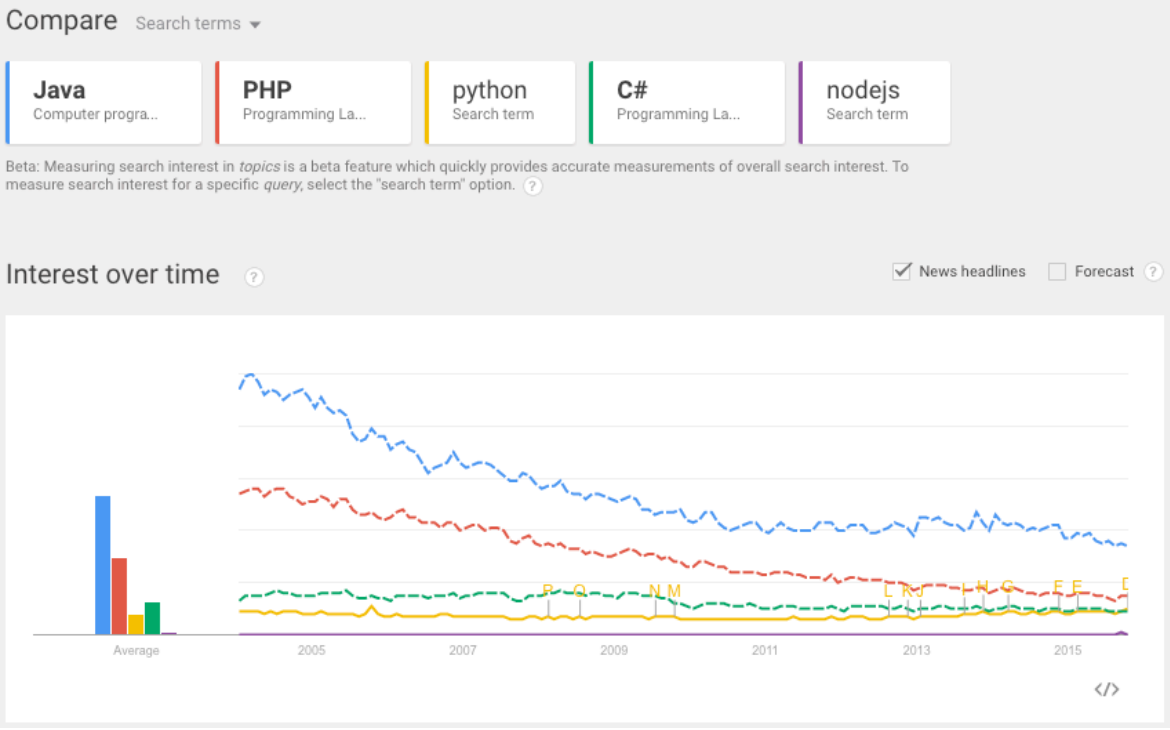
Zjistit počet otázek použitím určitých klíčových slov na stránce Stack Overflow lze jednoduše udělat nastavením filtrů pomocí tagů.

Při porovnání velikosti komunity programovacího jazyka jsem se setkal s problémem, že v oficiálních zdrojích se to neuvádí, proto jsem použil počet projektů na Github hledáním určitého jazyka nebo frameworku, které jsou v korelaci s velikostí komunity vývojářů daného programovacího jazyka.

3.4.1 Serverová část

Pro serverovou část byly porovnány nejznámější programovací jazyky – PHP, Java, Python, C# a NodeJS.

Výsledky z Google Trends jsou uvedené na obrázku 3.2.



Obrázek 3.2: Trend programovacích jazyků pro vývoj serverové části webové aplikace. Zdroj: vlastní průzkum na Google Trends

Velikost komunit vyjádřená počtem veřejných repozitářů na Github a počtem dotazů ve Stack Overflow je uvedená v tabulce 3.1.

	Java	C#	PHP	Python	Node.js
Počet dotazů na StackOverflow	1 017 679	912 948	879 112	537 132	111 061
Počet repozitářů na Github	274 725	12 657	173 451	242 308	46 065

Tabulka 3.1: Počet dotazů na Stack Overflow a počet veřejných repozitářů na Github pro programovací jazyky na straně serveru. Zdroj: vlastní průzkum na Github a Stack Overflow

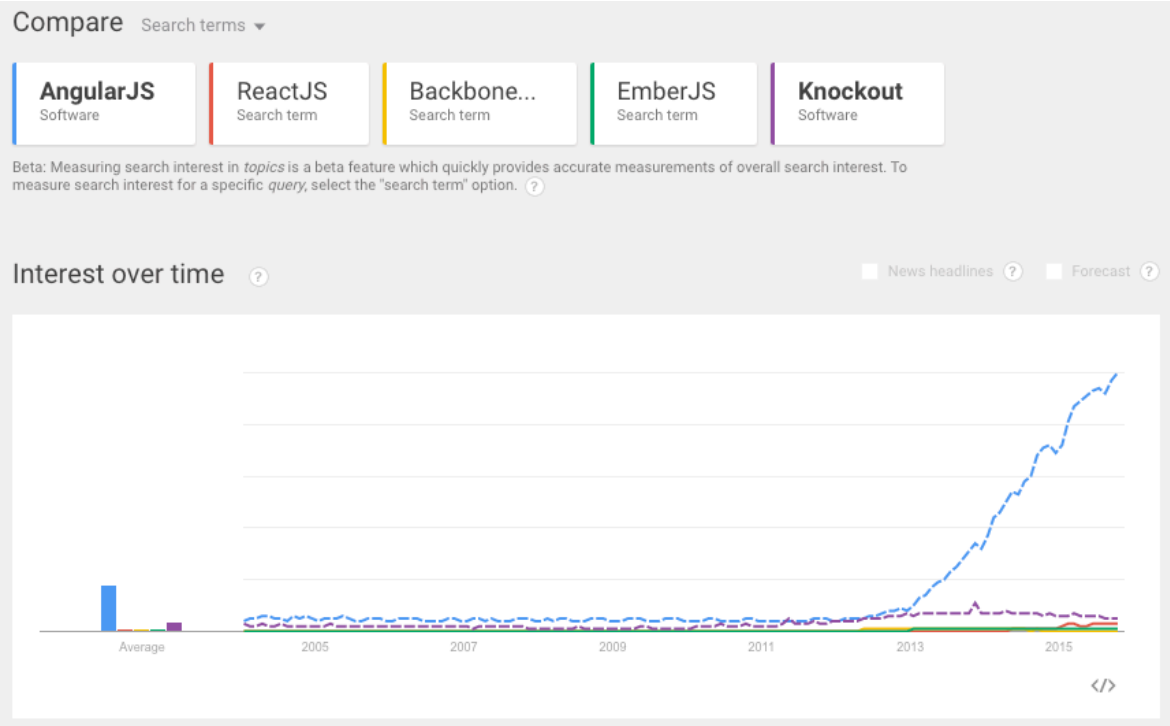
3.4.2 Klientská část

Pro klientskou část je výběr skriptovacích jazyků a frameworků také široký – Java applets, JSP, CoffeScript, ActionScript, JavaScript – však v porovnání s konkurenty je jistým lídrem JavaScript. Proto byly stejným způsobem pomocí Google Trends porovnány nejpopulárnější JavaScript frameworky – AngularJS, ReactJS, Backbone.js, Ember.js a Knockout.

Výsledky porovnání v Google Trends jsou uvedené na obrázku 3.3.
Velikost komunit vyjádřená počtem veřejných repozitářů na Github a počtem dotazů ve Stack Overflow je uvedená v tabulce 3.2.

	AngularJS	ReactJS	Backbone.js	Ember.js	Knockout
Počet dotazů na StackOverflow	154 125	10 830	19 408	18 069	15 597
Počet repozitářů na Github	105 369	54 301	20 907	16 931	3 603

Tabulka 3.2: Počet dotazů na Stack Overflow a počet veřejných repozitářů na Github pro frameworkové řešení na klientské straně. Zdroj: vlastní průzkum na Github a Stack Overflow



Obrázek 3.3: Trend JavaScript frameworků pro vývoj uživatelského rozhraní webové aplikace. Zdroj: vlastní průzkum na Google Trends

3.4.3 Databáze

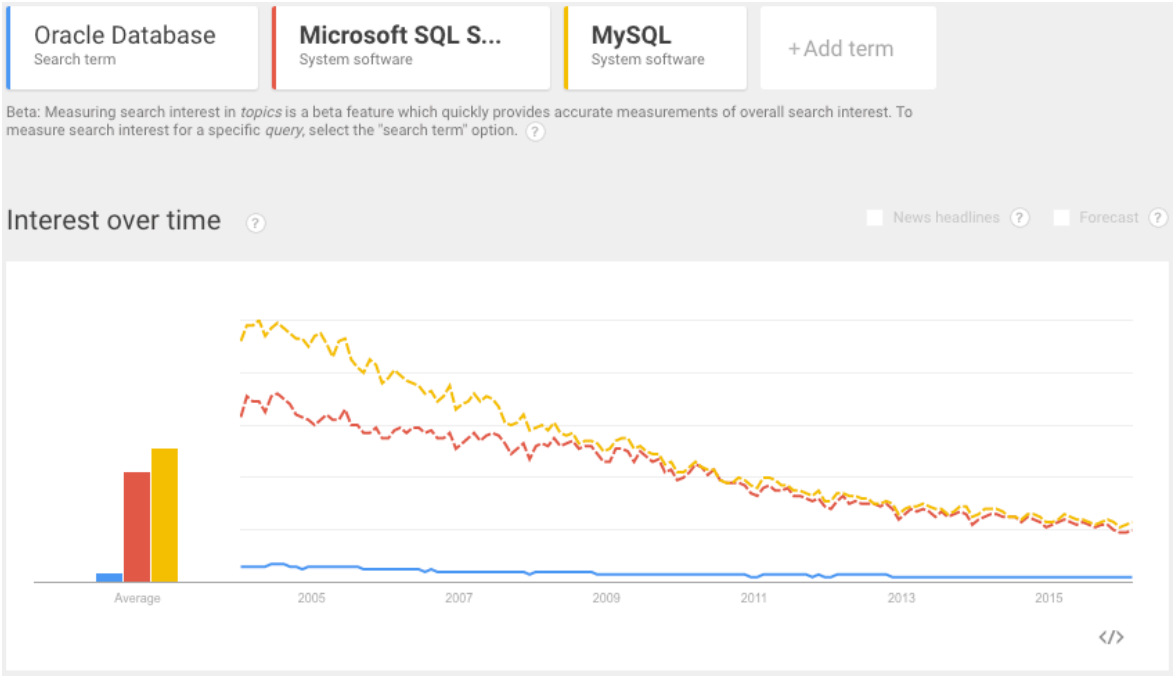
Z hlediska způsobu ukládání dat a vazeb mezi nimi se databáze dělí do následujících typů:

- hierarchická databáze,
- síťová databáze,
- relační databáze,
- objektová databáze,
- objektově relační databáze,
- dokumentově orientovaná databáze (také známá jako nerelační databáze).

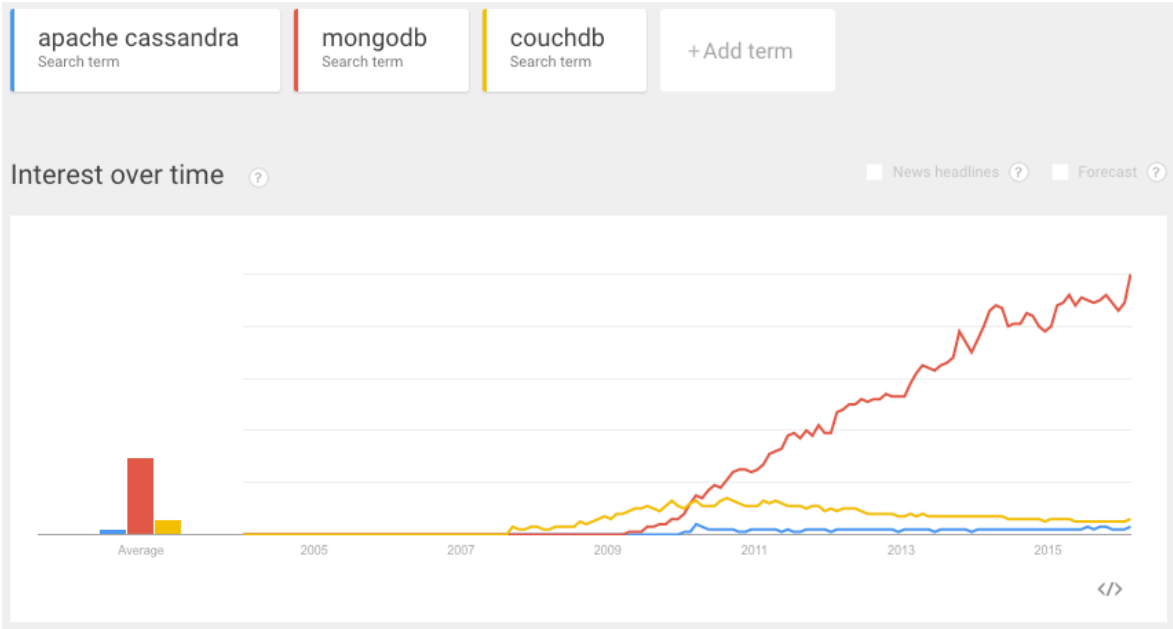
V rámci této práce však budou zvažovány relační a nerelační nebo NoSQL databáze, protože pro vývoj webových aplikaci se jiné typy databází běžně nepoužívají.

Pro porovnání popularity databázových řešení jsem si vybral tři známé relační databáze – Oracle, Microsoft SQL Server, MySQL a tři nejvíce známé NoSQL databáze – MongoDB, CouchDB a Cassandra.

Výsledky Google Trends pro relační databáze jsou uvedené na obrázku 3.4., pro nerelační databáze jsou na obrázku 3.5.



Obrázek 3.4: Trend relačních databázových řešení. Zdroj: vlastní průzkum na Google Trends



Obrázek 3.5: Trend nerelačních (NoSQL) databázových řešení. Zdroj: vlastní průzkum na Google Trends

Je vidět, že lídry jsou MySQL a MongoDB. Zajímavé je také to, že všechny tři nerelační databáze mají před rokem 2008 nulový trend, což právě odpovídá době vzniku nerelačních databází.

V následující tabulce jsou uvedené výsledky porovnání popularity databázových řešení na Stack Overflow a Github.

	Oracle DB	MS SQL Server	MySQL	Cassandra	Mongo DB	Couch DB
Počet dotazů na StackOverflow	66 510	156 311	377 964	8 534	58 661	4 058
Počet repozitářů na Github	7 341	814[1]	31 791	4 051	24 171	3 774

Tabulka 3.3: Počet dotazů na Stack Overflow a počet repozitářů na Github souvisejících s databázovymi řešeními. Zdroj: vlastní průzkum na Github a Stack Overflow

Je vidět, že jsou výsledky ze stránek Stack Overflow a Github a Google Trends v souladu, až na relační databázi MS SQL Server a NoSQL databázi Cassandra. MS SQL Server má nejmémě veřejných repozitářů na Github a Cassandra má nejhorší výsledky na Google Trends, i když veřejných repozitářů na Github je více než CouchDB.

Možné vysvětlení je to, že MS SQL Server nemá jednotně známou zkratku, kterou by šlo použít jako klíčové slovo pro hledání repozitářů na Github. Dalším důvodem může být to, že MS SQL Server je komerční databáze a hlavně běží jenom na operačním systému od Microsoftu. S ohledem na to, že všechny veřejné repozitáře na Github jsou OpenSource (s otevřením zdrojovým kódem), je pochopitelné, proč je projektů na Github používajících MS SQL Server nejméně.

[1] Microsoft SQL Server se těžko hledá, protože má několik zkratek, např. *sql server, ms sql server, mssql* atd. V daném příkladu jsem použil klíčové slovo s nejlepším výsledkem – *ms sql.*

Nesoulad ve výsledcích Google Trends a Github pro databázi Cassandra lze vysvětlit tím, že v Google Trends jsem použil klíčová slova *Apache Cassandra*, jelikož použití samotného slova *Cassandra* by způsobilo nepřesné výsledky, protože uvedené slovo nesouvisí jenom s databází.

Jak již bylo zmíněno v rámci této diplomové práce, pro vývoj serverové části webové aplikace jsem zvolil programovací jazyk Java a na něm postavený framework s největší komunitou – Spring. Pro implementaci klientské části pak scriptovací jazyk JavaScript a nejpopulárnější framework od Google – AngularJS. Pro ukládání dat jsem zvolil nerelační databázi MongoDB.

4 Analýza požadavků

V této sekci popisuji funkční a nefunkční požadavky, tedy požadavky na funkcionalitu aplikace z hlediska uživatelského rozhraní a požadavky na sadu technologií (technology stack).

4.1 Funkční požadavky

V této podkapitole jsou uvedené požadavky na funkcionalitu jednotlivých částí webové aplikace.

4.1.1 Přihlášení a registrace uživatelů

- registrace uživatele pomocí registračního formuláře,
- přihlášení uživatele,
- odhlášení uživatele.

4.1.2 Hledání slov a překladů

- přepínání jazyků,
- hledání slov,
- virtuální klávesa pro zadávání slov s podporou jiných abeced,
- úprava překladů.

4.1.3 Nastavení osobního účtu

- přehled upravených překladů,
- editace a nastavení osobních údajů.

4.1.4 Administrace

- přehled uživatelů a změna jejích rolí,
- potvrzení nebo zamítnutí úprav překladů.

4.2 Nefunkční (technické) požadavky

Nefunkční požadavky nepopisují funkcionalitu cílové aplikace, ale definují požadavky na technologie pro vývoj webové aplikace.

4.2.1 Serverová strana aplikace

- Serverová strana aplikace bude napsaná v jazyce Java 8 použitím frameworku Spring 4.0.

4.2.2 Klientská strana aplikace

V závislosti na daném problému budou pro implementaci klientské strany aplikace využité následující knihovny a frameworky:

- AngularJS 1.4.8 – primární framework pro klientskou část,
- jQuery 1.8.0 – pro řešení ad hoc specifických problému,
- jQueryUI 1.11.4 – potřebné zásuvné moduly (pluginy).

4.2.3 Databáze

- Ukládání dat bude probíhat pomocí NoSQL databáze MongoDB 3.0.

4.2.4 Spouštění aplikace

Webové aplikace napsané v jazyce Java Enterprise nemohou byt spouštěné na klasickém webovém serveru (jako například Apache HTTP serveru) z mnoha důvodů, které jsou uvedené v kapitole Teoretická východiska.
Webová aplikace musí byt spouštěná na aplikačním serveru nebo servlet kontejneru.
Z toho plyne následující požadavek:

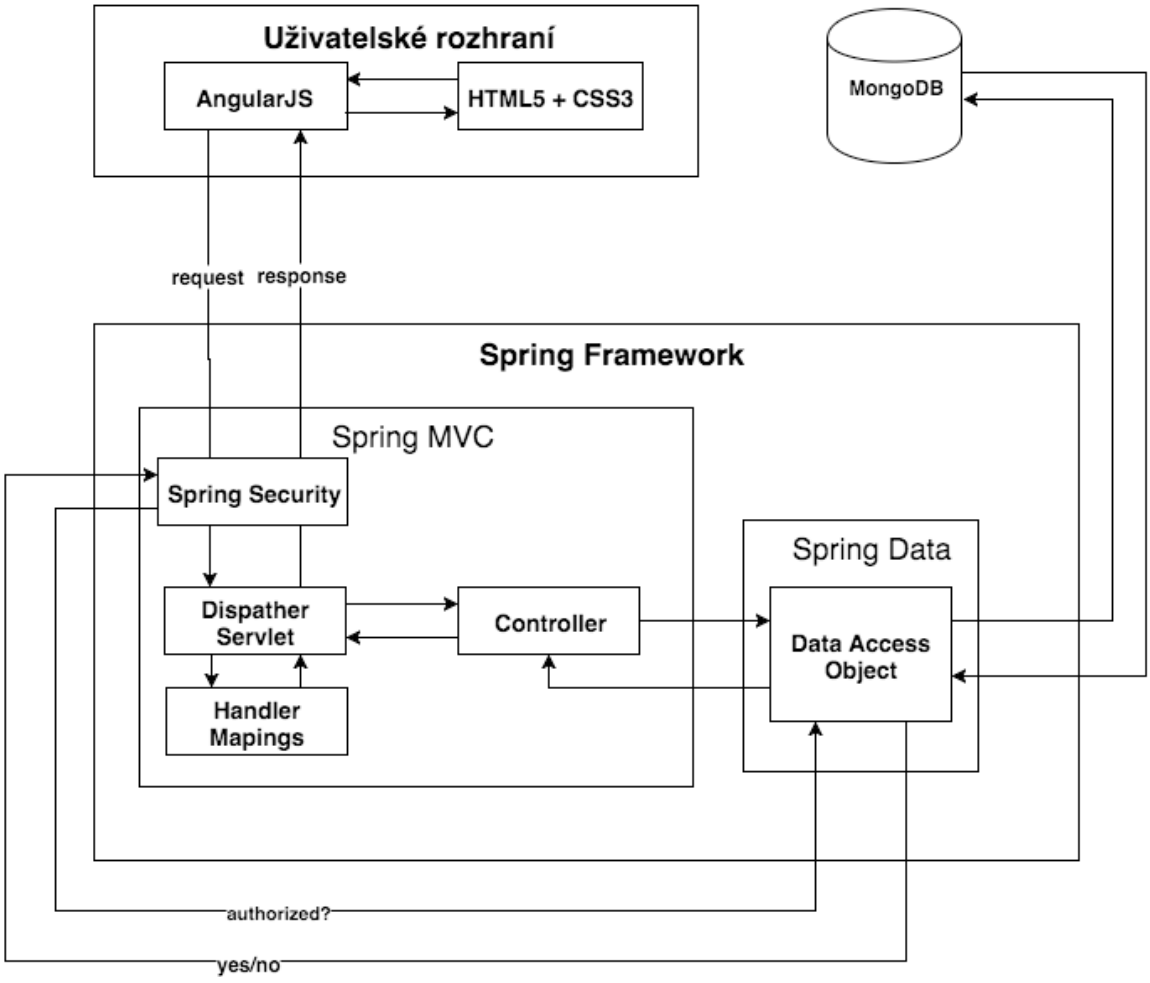
- webová aplikace bude běžet na aplikačním serveru Tomcat 8.0.

5 Architektura webové aplikace

Aplikace je sestavená z následujících tří částí:

- uživatelské rozhraní,
- serverová část,
- databáze.

Návrh architektury aplikace je vyjádřen na obrázku 5.1.



Obrázek 5.1: Architektura webové aplikace. Zdroj: autor

Uživatelské rozhraní se skládá z následujících dvou částí:

- HTML5 a CSS3 – zodpovědné za vzhled rozhraní,
- AngularJS – zodpovědný za řídící logiku na straně klienta.

Serverová část je plně řízena frameworkem Spring, konkrétně v rámci této práce jsou využity komponenty Spring Core (obsahuje základní rysy jako Spring kontext, vpíchnutí závislostí), Spring Web MVC (možnost mapovaní kombinací URL a typu HTTP požadavku na Java metodu) a Spring Data (zodpovědný za komunikaci s databází, mapování Java tříd na databázové tabulky nebo kolekce v případě NoSQL).

MongoDB databáze slouží pro ukládání aplikačních a uživatelských dat.

5.1 Scénář zpracování požadavku – technický pohled

Na následujících řádcích uvádím příklad zpracování klientského požadavku.

- Uživatel provede nějakou akci (například zmačkne tlačítko pro hledání).
- AngularJS Controller reaguje na akci uživatele a pošle HTTP dotaz na server.
- Každý HTTP požadavek se nejdříve dostane do Spring Security Filteru, který provede autorizaci (kontrolu oprávnění) požadavku. Pokud je požadavek neautorizovaný, je zamítnut a na klientskou stranu se pošle HTTP odpověď s kódem 400. Pokud je požadavek autorizovaný, tak se přepošle na Dispatcher Servlet.
- Dispatcher Servlet zachycuje a analyzuje HTTP požadavky a vysílá je odpovídajícím kontrolerům. Pro zjištění odpovídajícího kontroleru pro určitý HTTP požadavek se Dispatcher Servlet podívá do Handler Mappings, který obsahuje mapování požadavků (na základě URL definovaného pomocí *@RequestMapping* anotací) a handlerů (metoda v kontroleru vyplňující business logiku).
- Kontroler dostane požadavek od Dispatcher Servletu, a když je potřeba, pošle požadavek do databáze prostřednictvím abstrakcí Data Access Objects (objekty pro přístup k datům).
- Spring Data zkonvertuje požadavek do jazyku MongoDB a pošle jej. Odpověď ve formátu String se pak deserializuje do Java objektu a pošle zpátky kontroleru.
- Kontroler jej pak vrátí servlet dispečerovi (Dispatcher Servlet).
- Servlet Dispatcher jej zase serializuje do JSON stringové podoby a pošle klientovi.
- Na straně klienta AngularJS dostane JSON string a zase jej deserializuje do javaskriptového objektu a zobrazí jej v prohlížeči konečnému uživateli.

5.2 Databázový model webové aplikace

5.2.1 Vnořený datový model

Při návrhu struktury nerelační databáze je důležité si rozhodnout, zda používat reference pro logické spojení dokumentů nebo *přístup vnořených dokumentů*.

V MongoDB je možné vnořovat související dokumenty, databázový model používající tento přístup je známý jako *denormalizovaný* model. Příklad denormalizovaného modelu je uveden v diagramu 5.2.

Obrázek 5.2: Diagram předvádějící přístup vnořených dokumentů. Zdroj: [12]

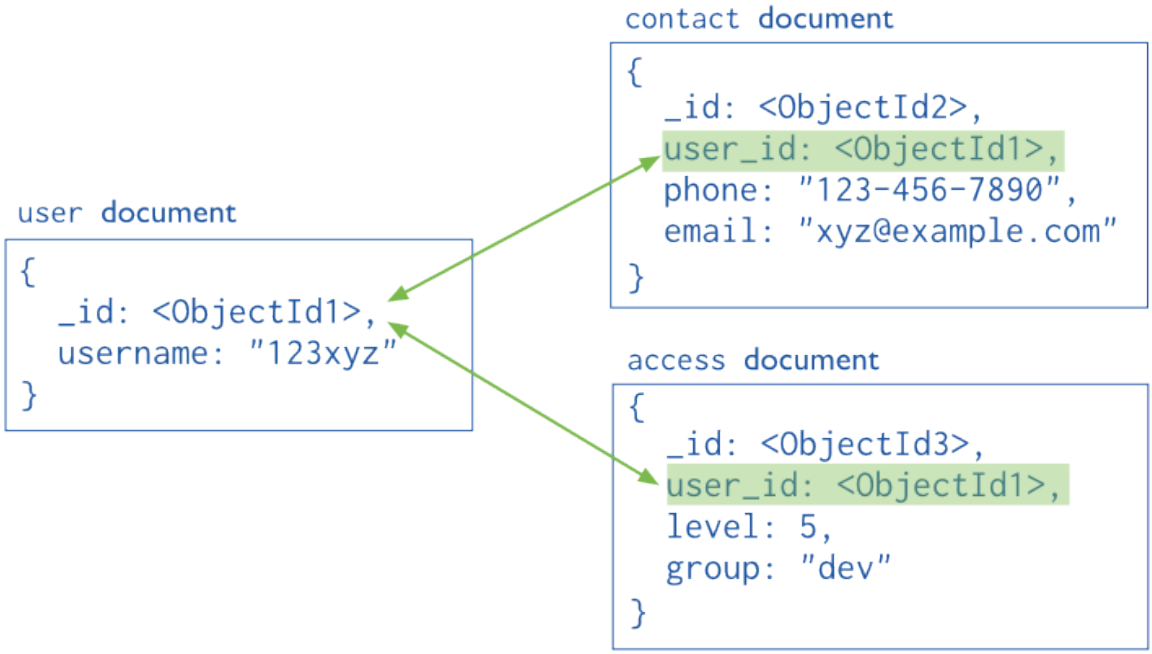
Vnořené datové modely umožňují ukládat související části informace ve stejném databázovém záznamu. Díky tomu se může snížit počet dotazů a aktualizací databáze pro provedení běžných úkolů.

Obecně vnořování dokumentů způsobuje lepší výkon operací čtení a také vyhledávání a získání souvisejících dat v jediné databázové operaci. Vnořené datové modely umožňují aktualizovat související data v rámci jedné atomické operace [12].

Při použití vnořeného modelu však může docházet k duplicitnímu uložení dat a růstu velikostí dokumentů. Růst velikostí dokumentů může ovlivnit výkon pro operace zápisu a také způsobit fragmentaci dat [12].

5.2.2 Normalizovaný datový model

Normalizované datové modely vyjadřují vztahy mezi dokumenty pomocí *referencí.* Příklad normalizované verze diagramu uvedeného na obrázku 5.2 je vidět na obrázku 5.3 [12].



Obrázek 5.3: Diagram normalizovaného datového modelu v MongoDB. Zdroj: [12]

Taková struktura dat odpovídá normalizovanému schématu dat známému z relačních databází. Toto schéma je flexibilnější a používáme jej většinou v situacích, kdy by použití vnořených dokumentů vedlo k duplicitnímu uložení dat, tedy pro modelování vztahů *N:M*. Zjevnou nevýhodou je nutnost získávání provázaných záznamů pomocí několika operací vzhledem k tomu, že dokumentované databáze umožňují jen zřídka propojování několika záznamů v rámci jednoho dotazu. Současné systémy tohoto typu také většinou neposkytují podporu pro automatickou kontrolu existence referencovaného primárního klíče [13].

Obecně by se normalizované datové modely měly používat, pokud:

- by použití vnořeného modelu vedlo k duplicitnímu uložení dat, ale bez dostatečného zvýšení výkonu, které by převážilo důsledky duplicity dat,
- se jedná o komplexní *many-to-many* vztahy,
- je potřeba modelovat složité hierarchické datové množiny.

5.2.3 Vnořený datový model webové aplikace

Při návrhu datového modelu aplikace jsem nejdřív uvažoval o vnořeném modelu.

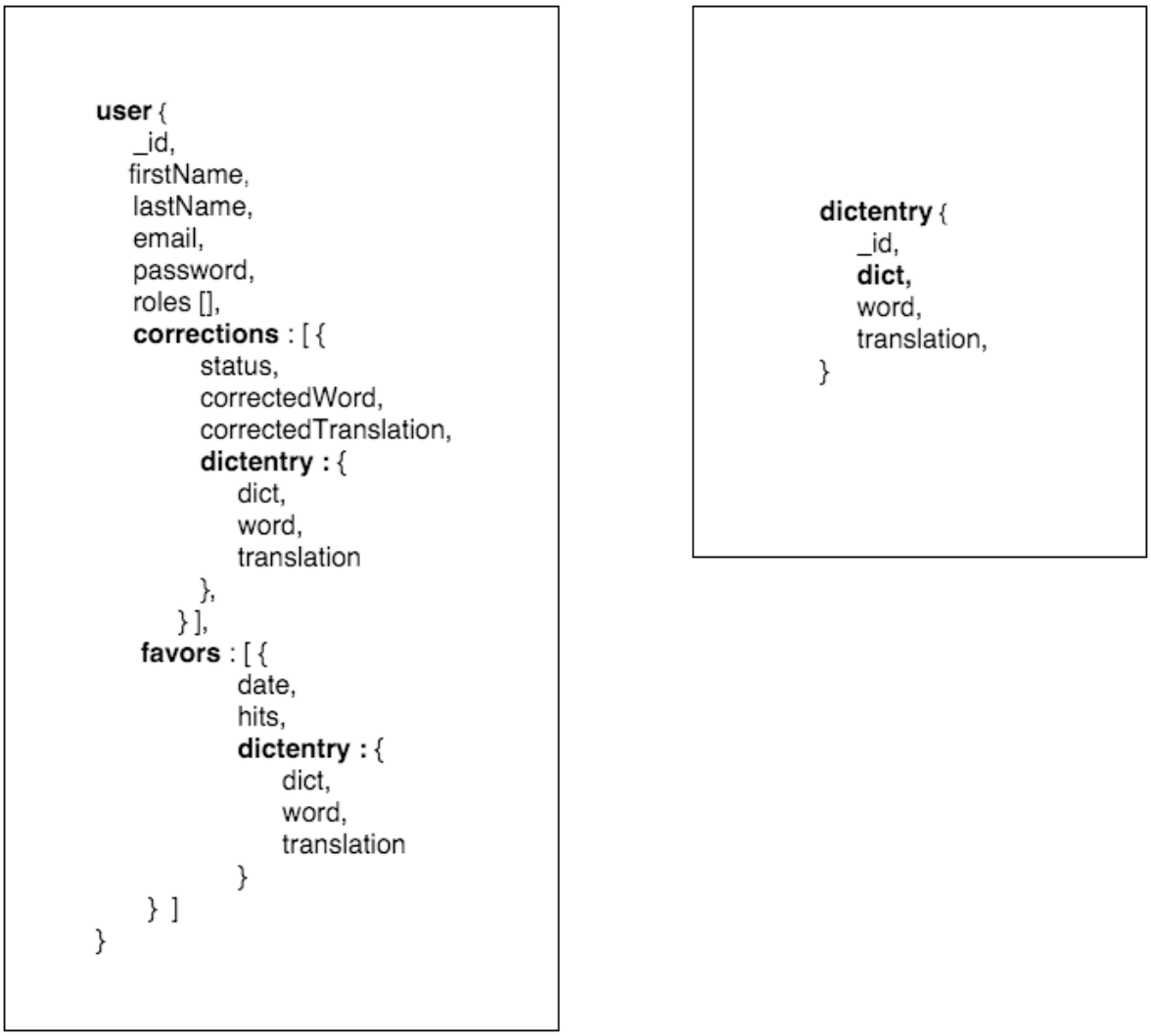
Vnořený model nerelační databáze může být navrhnut různými způsoby, proto je při modelování vnořeného modelu potřeba vycházet ze seznamu otázek, na které má databáze odpovídat.

Pro naši aplikaci jsou tyto otázky následující:

- Kdo je uživatelem pro daný e-mail?
- Jaký je překlad daného slova?
- Pro jaký slovník (dict) je daný překlad?
- Jaký je seznam všech úprav od všech uživatelů?
- Jaký je eznam všech úprav (corrections) daného uživatele?
- Jaký je seznam oblíbených slov (favors) daného uživatele?
- Pro jaký pár slovo-překlad (dictentry) je daná úprava?

Z těchto otázek vychází vnořený datový model uvedený na obrázku 5.4, který je sestaven ze dvou kolekcí *user* a *dictentry*.

Kolekce *user* obsahuje veškerá data související s uživatelskými úpravy a osobním slovníkem. Problémem však je, že *dictentry* se duplikuje až třikrát – *dictentry, user.corrections.dictentry* a *user.favors.dictentry*. Pokud by byly prvky kolekce *dictentry* konstantní, duplicita by mohla byt zanedbatelná. Ale jelikož v aplikaci mohou uživatelé upravovat záznamy v kolekci *dictentry*, po každé úpravě by bylo potřeba procházet každý prvek *dictentry* v *user.favors* a *user.corrections* a aktualizovat jej, což má velký dopad na výkon aplikace.



Obrázek 5.4: Schéma vnořeného databázového modelu aplikace. Zdroj: autor

5.2.4 Normalizovaný datový model webové aplikace

Kvůli důvodům uvedeným v předchozí sekci jsem si pro webovou aplikaci zvolil normalizovaný datový model použitím referencí.

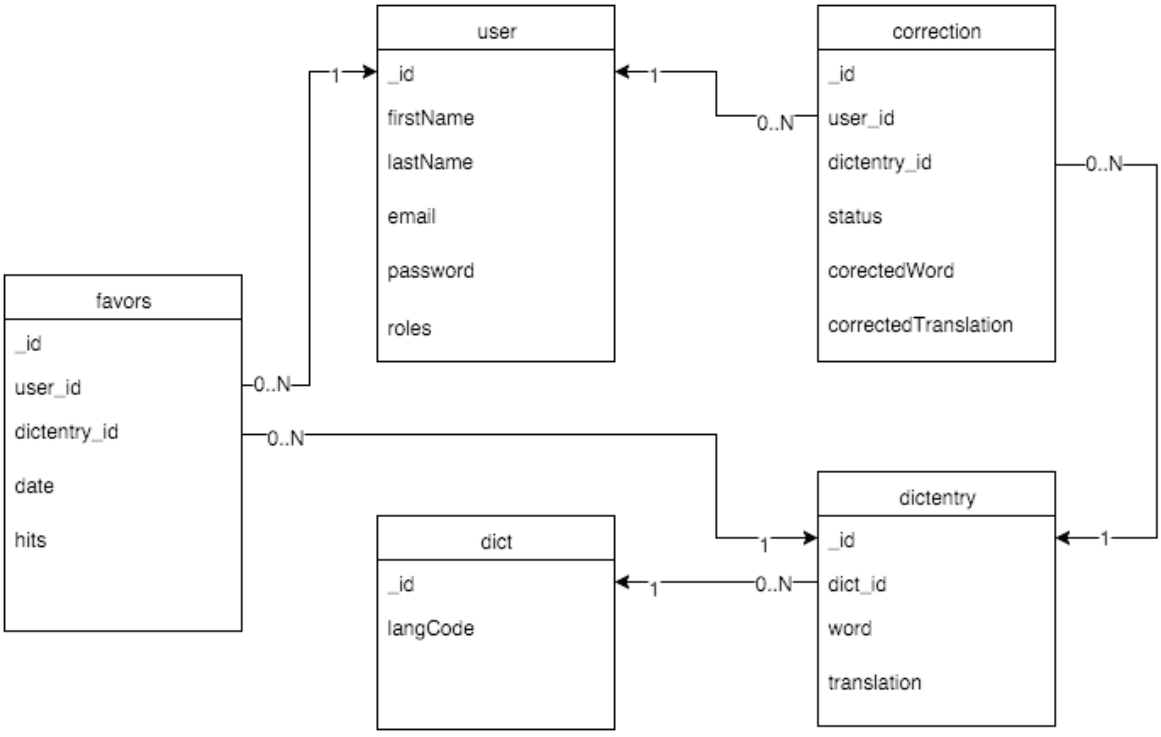
Model je sestaven z pěti kolekcí dokumentů pojmenovaných *user, correction, dict, dictentry, favors.*

- *User* [user][2] – kolekce pro ukládání uživatelských dat jako jméno, příjmení, e-mail, heslo apod.

[2] V hranatých závorkách jsou uvedené skutečné názvy kolekcí v databázi.

- *Correction* [correction] – kolekce pro ukládání požadavků pro úpravy překladů jednotlivých slov.
- *Dictionary* [dict] – kolekce obsahující kód slovníku, který odpovídá směru překladů, například ENTJ definuje slovník anglicko-tádžický.
- *DictionaryEntry* [dictentry] – kolekce obsahující nejdůležitější data, což je pár slovo-překlad.
- *Favourites* [favors] – kolekce pro osobní slovník oblíbených slov uživatele.

Jelikož jsem si zvolil normalizovaný model použitím referencí, diagram databázového modelu je podobný ER (Entity-Relation) modelu relační databáze (obrázek 5.5).



Obrázek 5.5: Schéma normalizovaného databázového modelu aplikace použitím referencí (cizích klíčů). Zdroj: autor

Použitím tohoto modelu jsme se zbavili duplicity, ale už musíme manuálně spravovat cizí klíče a kaskádové mazání dokumentů, protože MongoDB to nepodporuje.

5.3 Případy užití webové aplikace

Webová aplikace podporuje následující tři základní uživatelské role:

- anonymní uživatel,
- přihlášený uživatel,
- administrátor.

Obrázek

5.6: Případy užití webové aplikace na základě rolí. Zdroj: autor

Anonymní nebo nepřihlášený uživatel má jenom základní práva pro:

- hledání slov,

- přihlášení,
- vytvoření účtu – registraci.

Pokud se zaregistrovaný uživatel přihlásí, rozšíří se mu práva pro:

- upravování překladu slov,
- přidání slova do oblíbených slov (osobního slovníku),
- nastavení účtu,
- odhlášení.

Pokud má přihlášený uživatel roli administrátora, tak má veškerá práva pro:

- hledání slov,
- upravování překladu slov,
- přidání slova do oblíbených slov (osobního slovníku),
- nastavení osobního účtu,
- nastavení účtu uživatelů (změna jejích rolí, deaktivace účtu apod.),
- potvrzování nebo zamítnutí úpravy překladu,
- odhlášení.

6 Vývoj

V této kapitole uvedu popis procesu vývoje webové aplikace.

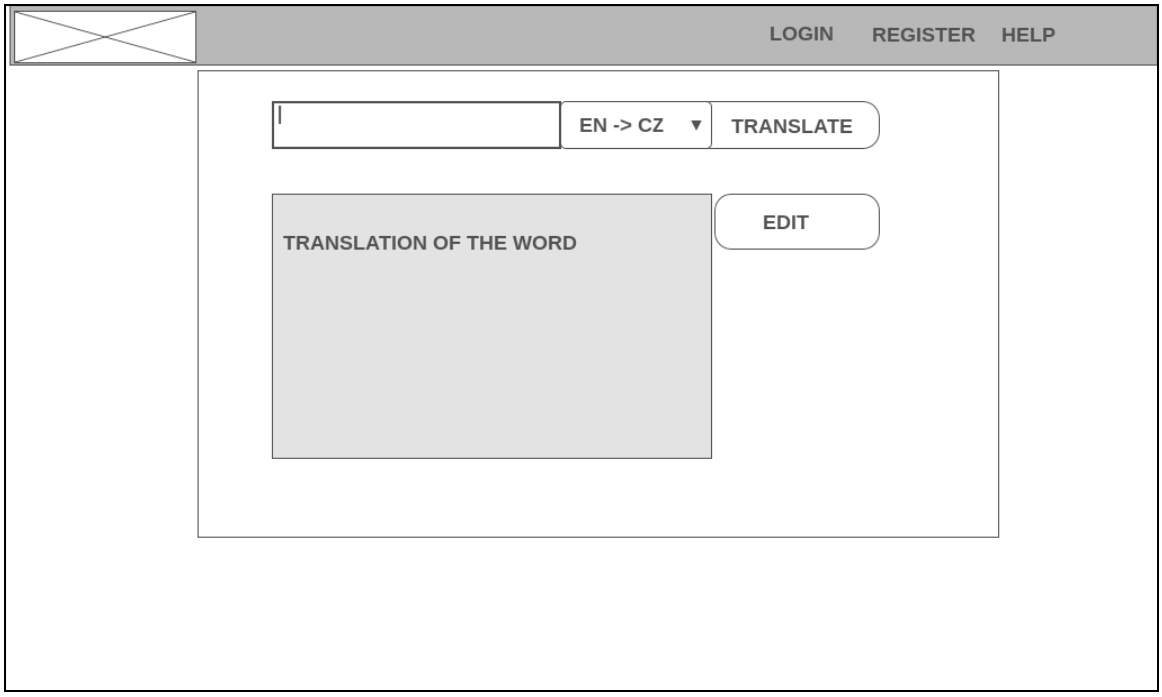
6.1 Vývoj klientské části aplikace

6.1.1 Design a vzhled

Vývoj klientské části webové aplikace začíná nejdříve návrhem vzhledu nebo designu uživatelského rozhraní. Pro návrh vzhledu uživatelského rozhraní existuje několik způsobů . Pokud se jedná o složité rozhraní s velkým počtem prvků, doporučuje se pro lepší představu a náhled obsahu webových stránek používat wireframe webu (*drátěný model*). Wireframe webu je obzvlášť užitečný pro lepší komunikaci v projektech, na kterých pracuje více lidí a tým je rozdělen dle rolí (např. designer, programátor, vedoucí týmu, zákazník).

Pro návrh uživatelského rozhraní webové aplikace jsem použil online nástroj pro kreslení wireframu[3]. Pro nepřihlášeného uživatele by hlavní stránka webové aplikace měla vypadat tak, jak je ukázáno na obrázku 6.1.

[3] Online nástroj pro kreslení wireframu webových stránek www.wireframe.cc



Obrázek 6.1: Wireframe (drátěný model) hlavní stránky webové aplikace. Zdroj: autor

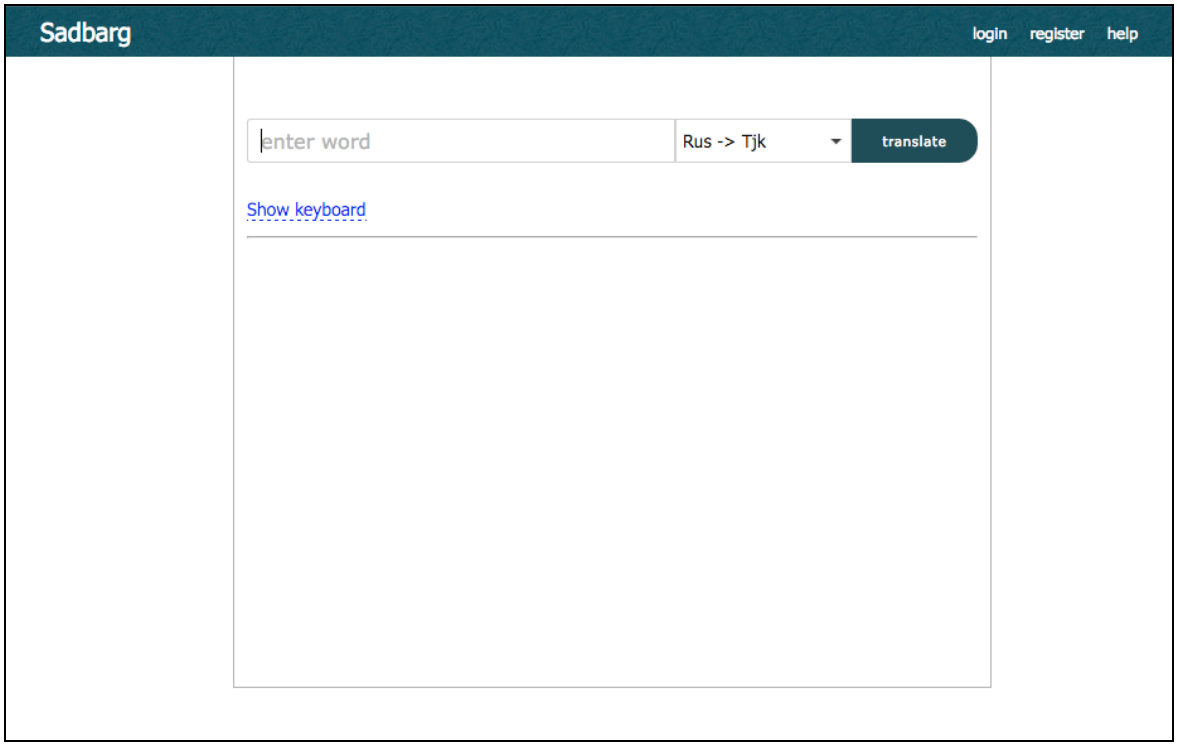
Jednu z hlavních rolí při návrhu vzhledu webové stránky hraje výběr primárních barev a stylů prvků webové stránky (písmo/font, tlačítka, prvky pro vstup / input elements apod.) a konzistence, což znamemá, že všechny prvky webové aplikace musejí mít podobný styl. Vzhledem k výše uvedeným důvodům jsem při návrhu designu uživatelského rozhraní pomocí CSS3 na začátku stylesheet dokumentu definoval základní styly pro obecné prvky stránky – font, tlačítka, barvy a styl chybových hlášek.

```
/* defining the default font and colors across all pages */
* {
    font-family: Tahoma, Arial, sans-serif;
    color: black;
}

/* style for predefined parts of web (translation part) */
pre {
    white-space: pre-wrap; /* CSS 3 */
    white-space: -moz-pre-wrap; /* Mozilla, since 1999 */
    white-space: -pre-wrap; /* Opera 4-6 */
```

```
    white-space: -o-pre-wrap; /* Opera 7 */
    word-wrap: break-word; /* Internet Explorer 5.5+ */
}

/* no default decoration is needed for links */
a:link {
    text-decoration: none;
}

/* cursor to change to pointer when hovering these elements*/
a, button, select, input[type="submit"] {
    cursor: pointer;
}

/* style for all error messages (field validation, etc.) */
.input-error {
    color: red;
    width: 150px;
}

/* defining default style for buttons in app */
.btn {
    border: 0 none;
    letter-spacing: 1px;
    background-color: #1D3E48;
    color: #ffffff;
    border-radius: 2px;
    outline: 0 none;
    box-shadow: none;
}
/* color when cursor is hovering a button */
.btn:hover {
    background: #1d6361;
}
```

```
/* color when button is actived (being clicked) */
.btn:active {
    background: #007299;
}

/* sizes of extra small button */
.btn-extra-small {
    width: 45px;
    height: 25px;
}

/* sizes for small button */
.btn-small {
    width: 60px;
    height: 25px;
}

/* sizes for medium buttons */
.btn-medium {
    width: 90px;
    height: 35px;
}

/* size for large button */
.btn-large {
    width: 120px;
    height: 45px;
}

}
```

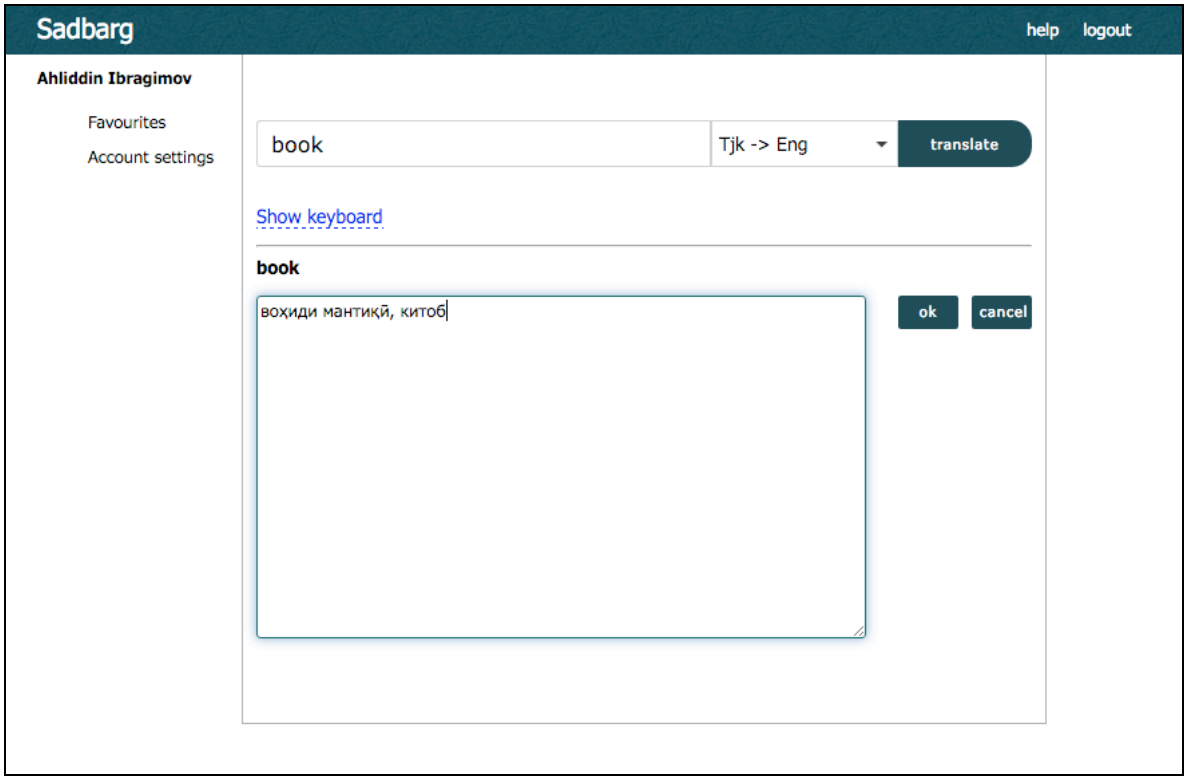
Vzhled skutečně implementované hlavní stránky je vidět na obrázku 6.2.



Obrázek 6.2: Hlavní stránka webové aplikace pro nepřihlášeného uživatele. Zdroj: autor

Po přihlášení se uživateli nabídne více možností, které jsou uvedené v diagramu případu užití (viz obrázek 5.5).

Na obrázku 6.3 je vidět uživatelské rozhraní pro přihlášeného uživatele, který nemá administrátorská práva.

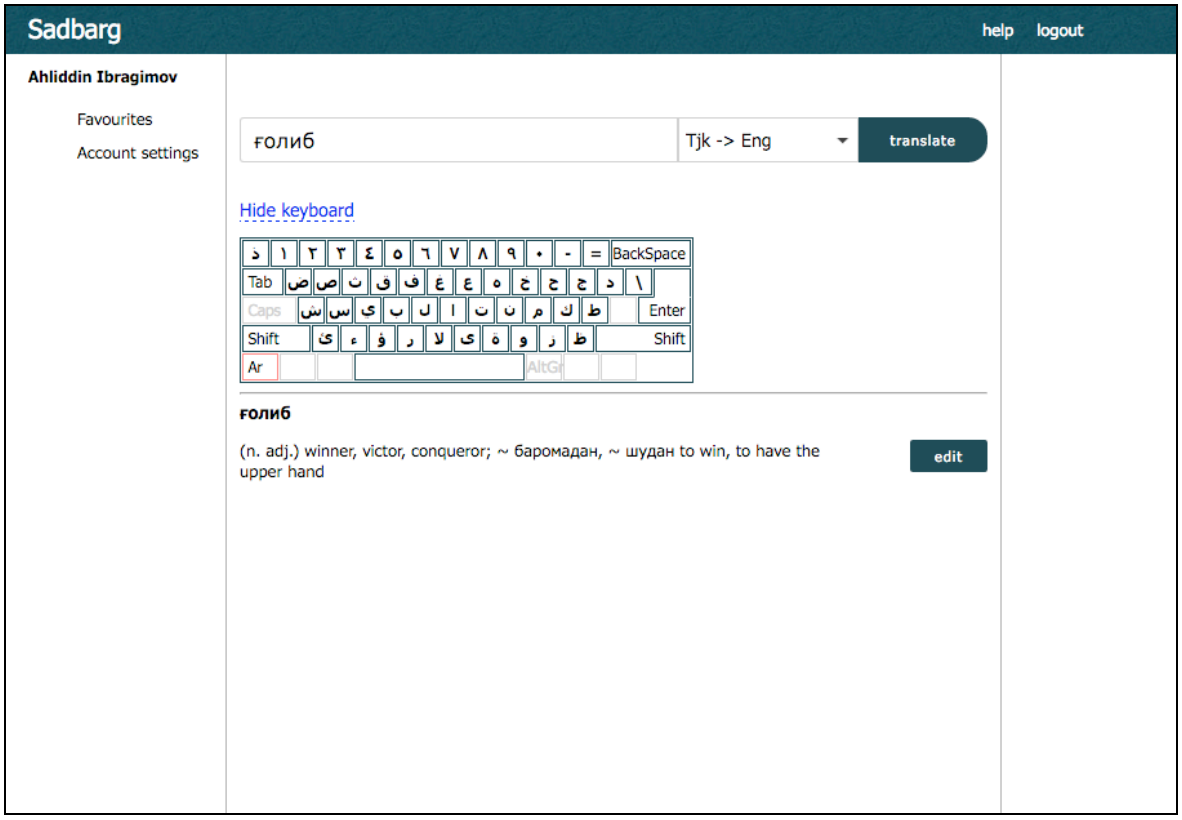


Obrázek 6.3: Příklad úpravy překladu pro přihlášeného uživatele. Zdroj: autor

Po vyhledání slova ve slovníku se přihlášenému uživateli nabídne další možnost úpravy překladu. Po úpravě a zmáčknutí tlačítka *„ok“* se požadavek pošle na server a uloží se v kolekci dokumentů *corrections*. Následně bude zobrazen administrátorovi buď pro potvrzení, nebo zamítnutí úpravy.

Z obrázku 6.3 je vidět, že i když je směr překladu Tjk → Eng, slovník dokázal přeložit slovo i v opačném směru. To je kvůli tomu, že když aplikace nenajde překlad pro uvedené slovo ve slovníku pro daný směr překladu, tak potom začíná hledat v jiné kolekci *dictentry* odpovídající opačnému směru překladu.

Dalším užitečným prvkem uživatelského rozhraní je pro usnadnění zadávání slov v cizích jazycích virtuální klávesnice, která podporuje různé jazyky včetně tádžičtiny (viz obrázek 6.4). Virtuální klávesnice byla do aplikace integrována od externího open-source projektu.



Obrázek 6.4: Virtuální klávesnice pro zadávání slov v cizích jazycích. Zdroj: autor

6.1.2 Řídící logika klientské části

Logika uživatelského rozhraní je plně řízená frameworkem AngularJS a jenom částečně jQuery a jQueryUI.

Pro celou aplikaci jsem vytvořil jeden hlavní Angular modul ***sadbarg*** a pro každou stránku zvlášť jeden Angular controller.

Zdrojový kód klientské části jsem rozdělil do následujících balíčků:

- *constants* – balík pro ukládání Angular konstant,
- *controllers* – balík obsahující Angular controllers,
- *services* – balík obsahující Angular services (služby),
- *other* – složka obsahující externí knihovny nebo nástroje, konkrétně virtuální klávesnici a autocomplete-widget.

Všechny konstanty používané v klientské části jsou deklarovány v **constants.AppConstants.** Tyto konstanty se používají pro hlášení chyb při přihlašování a

registraci uživatele. Konstanta *USERS.currentUser* se používá jako klíč pro ukládání uživatelských dat do mapy LocalStorage.

```
angular.module("sadbarg")
    .constant("AUTH_EVENTS", {
        loginSuccess: 'auth-login-success',
        loginFailed: 'auth-login-failed',
        logoutSuccess: 'auth-logout-success',
        sessionTimeout: 'auth-session-timeout',
        notAuthenticated: 'auth-not-authenticated',
        notAuthorized: 'auth-not-authorized'
    })
    .constant("REGISTER_EVENTS", {
        registerSuccess: 'register-success',
        invalidFirstName : 'invalid-firstname',
        invalidLastName: 'invalid-lastname',
        invalidEmail: 'invalid-email',
        emailExistsError: 'email-exists-error',
        invalidPassword: 'invalid-password',
        invalidBirthdate: 'invalid-birthdate'
    })
    .constant ('USERS', {
        currentUser: 'currentUser'
    });
```

Důležitým modulem je **services.AuthService,** který je zodpovědný za řízení procesu autentizace uživatelů. Níže je uveden zkrácený výpis zdrojového kódu tohoto modulu.

```
angular.module("sadbarg", ['spring-security-csrf-token-
interceptor', 'LocalStorageModule'])
    .factory("AuthService", [function ( /*závislostí*/ ) {
          return {
            preparePostData: function (credentials) {
```

```
            ...
        },

        login: function (credentials) {
            ...
        },

        fetchUser: function (username) {
            ...
        },

        logout: function () {
            Session.destroy();
            localStorageService
                .remove(USERS.currentUser);
            ...
        }
    }
}]);
```

Nejdůležitějším modulem v Angular projektu je však *controller*. Jelikož všechny controllery této aplikace mají podobnou strukturu, pro pochopení logiky dostačuje ukázka výpisu zdrojového kódu jedné z nich – **controllers.IndexController.**

IndexController je controllerem hlavní stránky webové aplikace a řídí základní funkcionality jako *hledání překladu slov, připravování a posílání požadavku pro úpravu překladu, odhlášení a zrušení session.*

```
'use strict';
angular.module('sadbarg')
    .controller('IndexController',
        function (... /*function dependencies */) {
          var main = this;
```

```
    (function init() {
          ... // self-invoking function to initilize
              // variables
      }());

      main.showEditArea = function () {
          ... // enable user to edit the translation
      };

      main.translate = function () {
          ... // translate given word
      };

      main.sendTranCorrection = function (correction) {
         ... // sending user correction to server
      };

      main.logout = function () {
          AuthService.logout();
      };

      function clearTransField() {
          ... // clearing input fields
      }

      function isAdmin(authorities) {
          ... // check if logged in user is admin
      }

      function updateCurrentUser() {
          ... // updating data of current user
      }
  });
```

6.2 Vývoj serverové části aplikace

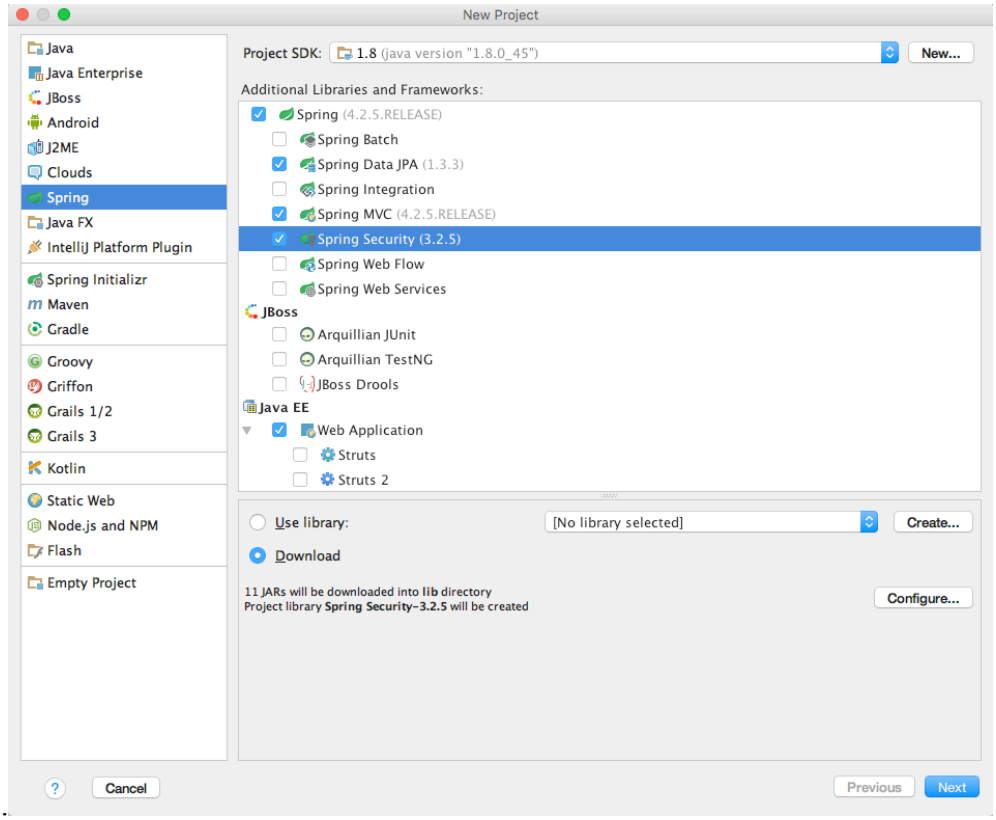
Pro vývoj serverové části aplikace jsem použil vývojové prostředí nebo IDE (Integrated Development Environment) *IntelliJ IDEA 15*.

Serverová strana aplikace je plně řízena Spring MVC. Jak již bylo zmíněno v rešeršní části práce, aplikace Spring může být konfigurována buď pomocí XML souboru, nebo pomocí Java Configuration. Zvolil jsem použití Java Configuration.

Pro správu, řízení a automatizaci procesu *sestavování* (buildování) zdrojového kódu a řešení závislostí externích knihoven jsem použil nástroj *Apache Maven*. Hlavní soubor s názvem *pom.xml*, kde se definují závislosti a potřebné konfigurace Mavenu, musí být umístěn v kořenovém adresáři projektu (aplikace).

6.2.1 Konfigurace projektu

Nejdříve je potřeba založit nový Spring MVC projekt v IntelliJ IDEA (dále jenom „IntelliJ“ nebo „vývojové prostředí“). Při vytvoření nového projektu nabízí IntelliJ možnost definování frameworku a potřebných knihoven projektu.



Obrázek 6.5: Založení projektu a základní nastavení. Zdroj: autor

Na obrázku 6.5 je vidět, že pro daný projekt jsem nastavil Java SDK verze 1.8.0, framework Spring a dodatečné knihovny Spring Data JPA, Spring MVC a Spring Security.
Spring Data JPA – obsahuje třídy a rozhraní sloužící pro snadnou implementaci repozitářů založených na JPA pro komunikaci s databázi.
Spring MVC – obsahuje třídy a rozhraní vyplňující základní operace pro webovou aplikaci.
Spring Security – obsahuje třídy a rozhraní pro konfiguraci bezpečnostních politik webové aplikace.

Po zmáčknutí tlačítka *next* nabídne vývojové prostředí uvedení názvu projektu. Potom se po zmáčknutí tlačítka *ok* vytvoří maven projekt s defaultní adresářovou strukturou pro Spring MVC. V konfiguračním souboru *pom.xml* bude uvedené základní nastavení jako *groupId, artifactId, version, packaging* a definovány závislosti.

Při konfiguraci závislých knihoven v maven je velmi důležité správně uvádět verze knihoven. Určitá verze jedné knihovny může fungovat spolu jenom s určitou verzí jiné knihovny. Například Spring Security 4.0 nemůže správně fungovat s knihovnou Spring MVC 3.0. Správa verzí závislostí může být hodně časově náročná, proto vyvinula komunita Spring další knihovnu *Spring IO platform,* která slouží právě pro řešení verzí závislosti, protože Spring IO platform poskytuje verze různých Spring knihoven a jejich závislostí.

Bez Spring IO platform by definování závislostí v mavenu vypadalo následovně:

```
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>4.2.3.RELEASE</version>
</dependency>
<dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-core</artifactId>
      <version>4.0.3.RELEASE</version>
</dependency>
```

Pomocí Spring IO platform stačí jenom do souboru pom.xml přidat následující:

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.spring.platform</groupId>
```

```
            <artifactId>platform-bom</artifactId>
            <version>2.0.0.RELEASE</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

A dále se už nemusíme starat o verzi a jenom stačí uvést závislou knihovnu bez definování verzí a Spring IO nastaví správné a příslušné verze sám:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
</dependency>
```

6.2.2 Konfigurace Spring Framework

Pro spouštění Spring aplikace je potřeba ji nahrát na aplikační server, jímž je v našem případě Tomcat 8.0. Vstupním bodem je pro aplikační server do nahrané Spring aplikace implementace základní třídy *WebApplicationInitializer,* ve které se musí přepsat metoda *onStartup (ServletContext servletContext)*. Metoda *onStartup* slouží pro nastavení ServletContext. ServletContext udržuje informaci o všech instancích v dané aplikaci. Každá aplikace může mít právě jeden Servlet Context [14]. Do ServletContext vytvoříme a přidáme *DispatcherServlet,* který je nastavený na prohlížení určených javovských balíků a vytváření instancí tříd, které mají jednu z anotací *@Service, @Repository, @Controller, @Component, @Configuration.*

Pro zajištění bezpečnosti je potřeba implementovat třídu *AbstractSecurityWebApplicationInitializer.* Pro nastavení bezpečnostních politik, jako jsou autentizace (potvrzení pravosti) a autorizace (kontrola oprávnění), je potřeba implementovat třídu *WebSecurityConfigurerAdapter.* V implementované třídě pak přepíšeme metodu *configure (HttpSecurity httpSecurity)* a nastavíme základní bezpečnostní pravidla:

```
protected void configure(HttpSecurity http) throws Exception {
        CsrfTokenResponseHeaderBindingFilter csrfTokenFilter = new
                            CsrfTokenResponseHeaderBindingFilter();
        http.addFilterAfter(csrfTokenFilter, CsrfFilter.class);
        http.authorizeRequests()
                .antMatchers("/css/**").permitAll()
                .antMatchers("/img/**").permitAll()
                .antMatchers("/lib/**").permitAll()
                .antMatchers("/js/**").permitAll()
                .antMatchers("/admin/**").hasRole("ADMIN")
                .and()
                .formLogin()
                .defaultSuccessUrl("/")
                .loginPage("/login")
                .loginProcessingUrl("/authenticate")
                .usernameParameter("username")
                .passwordParameter("password")
                .successHandler(
           new AjaxAuthenticationSuccessHandler(
           new SavedRequestAwareAuthenticationSuccessHandler()))
                .and()
                .rememberMe()
                .tokenValiditySeconds(60*60*24*14)//two-week
                                                  //validation span
                .key("TemporaryRememberMeSolutionKey")
                .and()
                .logout()
                .logoutUrl ("/logout")
                .logoutSuccessUrl("/")
                .permitAll();
}
```

Důležitými metodami pro konfiguraci bezpečnosti jsou:

- *antMatchers* – slouží pro nastavení přístupových práv. Z výše uvedené konfigurace vychází, že do adresářů */css, /img, /lib, /js* můžou přistupovat všechni uživatelé (přihlášení a anonymní) webové stránky. To je logické, protože v těchto adresářích jsou uložené veřejné zdroje nezbytné pro správné zobrazení webové aplikace. Pro

přístup do adresáře */admin* už musí uživatel mít právý *ADMIN*, což se kontroluje metodou *hasRole („ADMIN“).* Spring Security sám zajistí dotazováním do databáze kontrolu role uživatele.

- *formLogin* – určuje, že autentizace bude probíhat pomocí HTTP formy. Pomocí metody *loginPage* se nastaví URL pro zobrazení přihlašovací formy. Pokud nebude loginPage nastavený, tak Spring Security vygeneruje defaultní stránku s přihlašovací HTTP formou.
- *loginProcessingUrl* – slouží pro nastavení adresy, na kterou má klientská část poslat vyplněnou formu s uživatelskými přihlašovacími údaji.
- *usernameParameter* a *passwordParameter* – říkají, v jakých parametrech mají HTTP formy hledat uživatelské jméno a heslo.
- *rememberMe* – umožňuje uchování session uživatele na dobu uvedenou v *tokenValiditySeconds.*
- *logout* – určuje možnost odhlašování, která se aktivuje při navigaci na adresu uvedenou v *logoutUrl.* Spring Security se také postará o invalidaci session a tokenu.
- *logoutSuccessfulUrl* – určuje adresu, na kterou bude uživatel po odhlášení přesměrován.

6.2.3 Řídící logika serverové části

Za logiku serverové části jsou zodpovědné instance tříd, které jsou anotované pomocí @Controller, @Repository nebo @Document.

Controllery (třídy anotované @Controller) slouží pro mapování URL a metody, která se spouští při dotazování do příslušné URL adresy.

```
@RequestMapping(path = "/register", method =
RequestMethod.GET)
public String registerPage() {
     logger.info("mapping '/register'");
     return "registerpage.html";
}
```

Výše je uvedená metoda třídy controller. Tato metoda je mapována na URL adresu *{hostname}/register* pro HTTP požadavky typu GET. Pokud tento požadavek přijde na server, pak se zvolí metoda *registerPage( ),* která vrátí stránku *registerpage.html.*

Repozitář (rozhraní anotované @Repository a rozšiřující (extending) *MongoRepository*) představuje mechanismus pro zapouzdření procesu ukládání, vyvolávání a hledání dat. Repozitář napodobuje kolekci objektů. Pro každou tabulku v databázi, v případě MongoDB pro každou kolekci, je potřeba definovat samostatný repozitář.

```
@Repository
public interface UserRepository extends MongoRepository<User,
String> {
    /**
     * Search for user using id
      */
    public User findUserById (String id);

    /**
     * Searches for user using email
     */
    public User findUserByEmail (String email);

    /**
     * Saves a given entity.
     */
    public User save (User user);
}
```

Lze si všimnout, že to je jenom Java rozhraní a metody nejsou nikde implementovány. Spring Data se sám postará o vytvoření mongo příkazů parsováním názvu metod tohoto rozhraní, které musejí dodržovat správný formát [15].

Model nebo dokument (třída anotovaná @Document) představuje mongo dokument, tedy záznam v mongo kolekci.

```
@Document
public class User implements Serializable {
    @Id
    private String id;
    private String firstName;
    private String lastName;

    @Indexed (unique = true)
    private String email;

    @Enumerated(value = EnumType.STRING)
    private List<RoleEnum> roles =
Arrays.asList(RoleEnum.ROLE_USER);
    private String password;
    // getters and setters...
}
```

6.3 Konfigurace databáze

6.3.1 Instalace MongoDB

Praktická část této práce byla provedena na operačním systému *Mac OS X El Capitan*. Pro instalaci MongoDB na Mac OS jsem využil balíčkovací systém *Homebrew,* který usnadňuje instalování a odinstalování aplikací přes konzoli.

Pro instalaci MongoDB stačí z příkazové řádky vyplnit následující příkaz:

```
brew install mongodb
```

6.3.2 Nastavení databáze

Po spouštění MongoDB příkazem *mongodb* bez parametrů dle standartních konfigurací MongoDB běží na portu *27017*. Pokud chceme spustit MongoDB na jiném portu, je potřeba přidat parametr *--port <číslo portu>*. Pro administraci databáze je potřeba se přihlásit do *mongo Shell*. Stačí vyplnit příkaz *mongo* z příkazové řádky a naběhne mongo Shell.

Pro vytvoření (nebo přepínání do již existující) databáze existuje následující příkaz:

```
use <databáze>
```

Pro danou webovou aplikaci jsem vytvořil databázi s názvem *sadbarg*. V databázi sadbarg jsem vytvořil tři základní kolekce:

- *dict* – kolekce pro ukládání směru překladů (například *entj* pro anglicko-tádžický slovník),
- *dictentry* – kolekce pro ukládání slov a odpovídajících překladů,
- *user* – kolekce obsahující data uživatelů.

6.3.3 Importování dat

Do MongoDB lze importovat data ve formátu JSON, TSV (Tab Separated Values) nebo CSV (Comma Separated Values). Při importování dat do databáze jsem se setkal s následujícími dvěma problémy:

- Seznam slov a překladů jsem dostal ve formátu Excel, který není podporován MongoDB.
- Špatné kódování textu v tádžičtině při exportování Excel dokumentů do CSV nebo TSV.

Implementoval jsem na to řešení skládající se z následujících kroků:

1. Uložit Excel dokument jako Unicode textový soubor (ve formátu TSV).
2. Identifikovat kódování souboru příkazem (pro Mac OS):
   ```
   file -I <soubor>
   ```
3. Převést kódování souboru z UCS-2 (zjistil jsem to pomocí příkazu z bodu 2.) do UTF-8 příkazem:
   ```
   iconv --from-code=UCS-2 --to-code=UTF-8 <vstupni_soubor>
   > <vychozi_soubor>
   ```
4. Importovat TSV soubor v UTF-8 kódování do MongoDB pomocí příkazu:
   ```
   mongoimport -c <nazev_kolekce> -d <nazev_databaze>
   --type tsv --file <nazev_souboru> --headerline
   ```

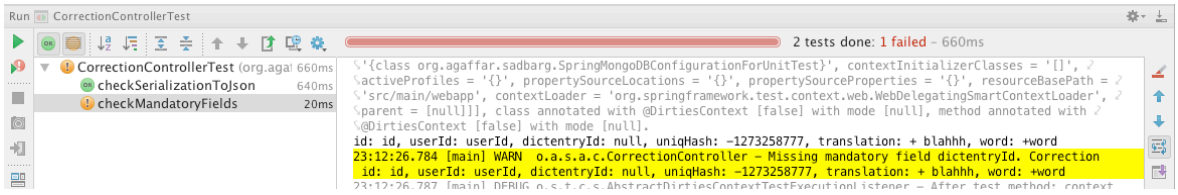
7 Testování

7.1 Testování na straně serveru

Pro testování a automatizaci testů serverové části webové aplikace jsem využil Spring MVC Test framework. Důležitější částí kódu zodpovídající za funkcionalitu aplikace jsou controllery. S ohledem na to jsem pro každý controller vytvořil zvlášť *unit* test ověřující funkcionalitu daného controlleru jako vracení správné HTML stránky a správné vyplňování business logiky jako generování seznamů slov začínajících určitými písmeny nebo přidávání slova do seznamu oblíbených slov.

Všechny vytvořené unit testy se spouští příkazem *maven test.* Při buildování projektu podle standardních nastavení maven automaticky spouští nejdříve všechny unit testy projektu a jenom po úspěšném provedení testů se zbuilduje sám zdrojový kód, což zaručuje, že změna zdrojového kódu (například kvůli refactoringu nebo přidání nových rysů) neporušila již existující logiku aplikace.

Na obrázku 7.1 je ukázka příkladu proběhnutí unit testu pro controller zodpovědný za provedení úprav překladu. Test obsahuje jednu metodu ověřující správné konvertování objektu Correction do řetězce ve formátu JSON, které proběhlo úspěšně, a další metodu ověřující všechny povinné parametry objektu, které neuspěly kvůli chybějícímu parametru *dictentryId.*



Obrázek 7.1: Příklad výsledku unit testu pro CorrectionController. Controller zodpovídající za provedení úprav překladů. Zdroj: autor

7.2 Testování na straně klienta

Pro funkcionální testování uživatelského rozhraní jsem využil knihovnu Selenium 2 (je známá také jako Selenium WebDriver). Knihovna Selenium se jednoduše přídává do maven projektu definicí závislosti v pom.xml souboru:

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
```

```
    <artifactId>selenium-java</artifactId>
    <version>2.53.0</version>
</dependency>
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-firefox-driver</artifactId>
    <version>2.42.2</version>
</dependency>
```

Tím se knihovna Selenium přidá do classpath projektu a můžeme ji využívat při implementování testů.

Pro nahrávání uživatelských akcí a generování testů jsem využil Selenium IDE plugin do FireFox, jenž obsahuje Selenium *recorder*, který nahrává veškeré operace provedené na webové stránce a může potom nahrané operace přehrát ve stejném pořadí.

Níže je uveden částečný příklad vygenerovaného unit testu pomocí Selenium IDE:

```
@Test
public void invalidCredentialsErrorTest() throws Exception {
  driver.get(baseUrl + "/login");
  driver.findElement(By.linkText("login")).click();
  driver.findElement(By.id("username")).clear();
  driver.findElement(By.id("username"))
        .sendKeys("test@test.com");
  driver.findElement(By.id("password")).clear();
  driver.findElement(By.id("password")).sendKeys("wrongPwd");
  driver.findElement(By.cssSelector("button.btn.btn-small"))
        .click();
  String errorMsg = "Incorrect email or password";
  String foundErrorMsg = driver
        .findElement(By.className("error-messages"))
        .getText();
  Assert.assertTrue("Error message wasn't shown: " +
        foundErrorMsg, foundErrorMsg.contains(errorMsg));
}
```

Tento test spouští FireFox prohlížeč a přejde do přihlašovací stránky webové aplikace a pokusí se přihlásit pomocí nesprávných údajů. Následně zkontroluje, zda se objeví v sekci HTML stránky označené CSS třídou *error-messages* chybová hláška *„Incorrect email or password“*. Po proběhnutí testu se okno prohlížeče zavře.

Výše uvedeným způsobem byly vytvořeny testy pro ověření klíčových funkcionalit webové aplikace.

8 Závěr

Cílem této diplomové práce byla rešerše současných technologických trendů ve sféře vývoje webových aplikací a implementace webové aplikace použitím technologií jako Spring pro vývoj serverové části, AngularJS pro vývoj klientské části a využití nerelační databáze MongoDB pro ukládání dat.

V rámci této práce jsem udělal analýzu požadavků na aplikaci a podle toho navrhnul architekturu aplikace a datový model. Detailně jsem zdokumentoval vývojový cyklus webové aplikace skládající se z implementace designu uživatelského rozhraní, návrhu architektury a datového modelu a vývoje serverové a klientské části webové aplikace.

Finálním výsledkem této práce je funkční webová aplikace implementovaná integrací modernějších technologií pro vývoj serverové části, klientské části a ukládání dat.

Dalším výsledkem je samotná aplikace ve formě sociálního slovníku, který byl vyvinut během implementace praktické části dané práce a může být hostován a využit veřejností.

8.1 Náměty na další rozvoj

Webová aplikace je stále ve vývoji a plánuji přidat další možnosti, jako přihlašování přes sociální sítě Facebook a Twitter a hodnocení důvěryhodnosti přispívajících uživatelů, kteří by časem na základě svých hodnocení mohli být sami oprávněni ověřovat úpravy překladů od jiných uživatelů. Následně plánuji přeložit webovou aplikaci do více jazyků a hostovat ji pro veřejnost.

Seznam použitých zdrojů

[1] Internet World Stats [online]. [cit. 2016-03-20]. Dostupné z: http://www.internetworldstats.com/emarketing.htm

[2] KNUT Haugen [online]. [cit. 2016-03-20]. Dostupné z: http://blog.knuthaugen.no/2010/03/a-brief-history-of-nosql.html

[3] REGNER, Tomáš. Dokumentově orientované open source databázové systémy. [cit. 2016-02-24]. Dostupné z: http://www.vse.cz/vskp/eid/32955

[4] HOWS, David, Peter MEMBREY a Eelco PLUGGE. MongoDB basics. New York, New York: Apress, 2014. Expert's voice in open source. ISBN 9781484208960.

[5] Wikipedia [online]. Java Enterirpse Edition. [cit-2016-03-20]. Doctupné z: https://cs.wikipedia.org/wiki/Java_EE

[6] ZAPLETAL Lukáš [online]. Spring Framework. [cit. 2016-03-20]. Dostupné z: http://static.zapletalovi.com/papers/spring/

[7] International Standard ISO [online]. ISO/IEC 16262:2011(en). [cit. 2016-03-20]. Dostupné z: https://www.iso.org/obp/ui/#iso:std:iso-iec:16262:ed-3:v1:en

[8] HonzIT [online]. První projekt s AngularJS. [cit. 2016-03-20]. Dostupné z: http://honzit.blogspot.cz/2013/08/blog-post.html

[9] Spring [online]. Spring Documentation. [cit. 2016-01-01]. Dostupné z: http://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-refe rence/htmlsingle/#spring-mvc-test-framework

[10] News TJ [online]. [cit. 2016-02-01]. Dostupné z: http://news.tj/en/news/google-translate-adds-tajik-language

[11] Redmine [online]. Official web page of Redmine. [cit. 2016-03-20]. Dostupné z: http://www.redmine.org

[12] MongoDB [online]. Official web page of MongoDB. [cit. 2016-03-20]. Dostupné z: https://docs.mongodb.org/manual/core/data-model-design

[13] HOLUBOVÁ, Irena, Jiří KOSEK, Karel MINAŘÍK a David NOVÁK. *Big Data a NoSQL databáze*. První vydání. Praha: Grada, 2015. Profesionál. ISBN 978-80-247-5466-6.

[14] Oracle [online]. Official web page of Oracle. [cit. 2016-03-20]. Dostupné z:

https://docs.oracle.com/cd/B14099_19/web.1012/b14017/overview.htm

[15] Spring [online]. Official web page. [cit. 2016-03-20]. Dostupné z: http://docs.spring.io/spring-data/jpa/docs/1.10.0.M1/reference/html/#repositories.query-methods.query-creation

Seznam obrázků

Obrázek 2.1: Komponenty Spring Frameworku 12
Obrázek 3.1: Seznam úkolů, softwarových chyb a jejích stav v Redmine 24
Obrázek 3.2: Trend programovacích jazyků pro vývoj serverové části webové aplikace 25
Obrázek 3.3: Trend JavaScript frameworků pro vývoj uživatelského rozhraní webové aplikace 27
Obrázek 3.4: Trend relačních databázových řešení 28
Obrázek 3.5: Trend nerelačních (NoSQL) databázových řešení 28
Obrázek 5.1: Architektura webové aplikace 33
Obrázek 5.2: Diagram předvádějící přístup vnořených dokumentů 35
Obrázek 5.3: Diagram normalizovaného datového modelu v MongoDB 36
Obrázek 5.4: Schéma vnořeného databázového modelu aplikace 38
Obrázek 5.5: Schéma normalizovaného databázového modelu aplikace použitím referencí (cizích klíčů) 39
Obrázek 6.1: Wireframe (drátěný model) hlavní stránky webové aplikace 43
Obrázek 6.2: Hlavní stránka webové aplikace pro nepřihlášeného uživatele 46
Obrázek 6.3: Příklad úpravy překladu pro přihlášeného uživatele 47
Obrázek 6.4: Virtuální klávesnice pro zadávání slov v cizích jazycích 48
Obrázek 6.5: Založení projektu a základní nastavení 52
Obrázek 7.1: Příklad výsledku unit testu pro CorrectionController. Controller zodpovídající za provedení úprav překladů 60

Seznam tabulek

Tabulka 3.1: Počet dotazů na Stack Overflow a počet veřejných repozitářů na Github pro programovací jazyky na straně serveru 26
Tabulka 3.2: Počet dotazů na Stack Overflow a počet veřejných repozitářů na Github pro frameworkové řešení na klientské straně 26
Tabulka 3.3: Počet dotazů na Stack Overflow a počet repozitárů na Github souvisejících s databázovymi řešeními 29

Obsah přiloženého CD

text/

xibra001_DP.pdf .....text práce ve formátu PDF

xibra001_DP.doc.....text práce ve formátu MS Word

xibra001_T.pdf.....text téze ve formátu PDF

xibra001_T.doc.....text téze ve formátu MS Word

diagramy/

…….soubor diagramů

obrázky/

…….soubor obrázků