VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Bakalářská práce - Hra Port Royal a moderní vývoj softwaru

Bachelor thesis - Game Port Royal and modern software development

2017 Jan Čech

Zadání bakalářské práce

Jiří Dvorský

Ukázka sazby diplomové nebo bakalářské práce

Diploma Thesis Typesetting Demo

+++

Podpis vedoucího katedry

+++

Podpis děkana fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 1. dubna 2016 .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 20. dubna 2017 .....

Rád bych na tomto místě poděkoval Ing. Davidovi Ježkovi, Ph.D., za pomoc a vedení u této práce.

Abstrakt

reklama - kratsi uvod - par vet

Klíčová slova: typografie, LaTeX, diplomová práce

Abstract

This is English abstract. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Fusce tellus odio, dapibus id fermentum quis, suscipit id erat. Aenean placerat. Vivamus ac leo pretium faucibus. Duis risus. Fusce consectetuer risus a nunc. Duis ante orci, molestie vitae vehicula venenatis, tincidunt ac pede. Aliquam erat volutpat. Donec vitae arcu. Nullam lectus justo, vulputate eget mollis sed, tempor sed magna. Curabitur ligula sapien, pulvinar a vestibulum quis, facilisis vel sapien. Vestibulum fermentum tortor id mi. Etiam bibendum elit eget erat. Pellentesque pretium lectus id turpis. Nulla quis diam.

Key Words: typography, LaTeX, master thesis

Obsah

Seznam použitých zkratek a symbolů

SPA	–	Single Page Application
REST	–	REpresentational State Transfer
WWW	–	World Wide Web
HTML	–	Hyper Text Markup Language
W3C	–	World Wide Web Consortium
XML	–	Extensible Markup Language
JSON	–	JavaScript Object Notation
J2EE	–	Java 2 Platform, Enterprise Edition
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
TDD	–	Test-driven development

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Jako vše v oblasti informačních technologii i oblasti vývoje webových stránek se objevuje spousta nových trendů. Tyto trendy se týkají jak nových technologii, jenž usnadňují vývoj. Tak také návrhu celé architektury aplikace, která vývoj zpřehledňuje a usnadňuje rozšířitelnost aplikace do budoucna. Také se do popředí dostává testování software, jenž byl kdysi okrajovou záležitostí. V dnešní době se však testování software, stalo standardní součástí vývoje aplikace. Cíle této práce bylo všechny tyto trendy zachytit. Práce by se dala shrnout do následujících bodu. Všechny body níže byly splněny.

1. Nastudovat moderní webové technologie Spring, AngulaJS a REST architekturu
2. Nastudovat principy testování a vybrat SW pro testování výše zmíněných technologii.
3. Navrhnout a implementovat hru Port Royal.
4. Zprovoznit testovací technologie a napsat v nich testy.

1.1 Stručný obsah jednotlivých kapitol

bude navazovat na horni cast

2 Technologie

2.1 REST webove služby

Webové služby jsou typ architektury pro komunikaci na principu server - klient přes World Wide Web (WWW) HyperText Transfer Protocol (HTTP). Tak jak jej popisuje World Wide Web Consortium (W3C). [6]

REpresentational **S**tate **T**ransfer neboli REST je webová služba, jenž by měla plnit následující podmínky díky nimž je rychlá a jednoduchá:

- Dotazy by měly být sebe popisující - Tímto se myslí, že dotaz obsahuje všechny informace k jeho zpracování. Tyto informace lze zjistit například z URI či hlavičky dotazu.
- Bezstavovost - díky tomu, že dotazy nemají stav jsou mezi sebou nezávislé. Pěkný test této podmínky je například restart serveru, po němž by se na interakci mezi serverem a klientem nemělo nic změnit.
- Uložitelnost do krátkodobé paměti - Jelikož jsou dotazy bezstavové a obsahují všechny informace k jejich zpracování, jsou výsledky dotazů téměř vždy stejné. Tudíž je zde velký prostor pro ukládaní dotazu do krátkodobé paměti. Tímto se sníží objem přenesených dat a urychlí komunikace mezi serverem a klientem, při opakovaném dotazu.
- Interface - rest služby mají předepsaný interface, jenž je popsán v následující podkapitole.

RESTy nemají upřesněný formát dotazu. Tudíž RESTový dotaz může být ve formátu XML, JSON, ale také například i ve formátu PDF. [7] [8]

2.1.1 Vzorové REST rozhraní

RESTové rozhraní je založeno na typech HTTP dotazů a jejich URI. Použít se standardní HTTP metody. POST pro vytvářená záznamu, GET pro získání záznamu, PUT pro úpravu záznamů a DELETE pro mazání. Dají se použít i další metody, tyto jsou ovšem nejpoužívanější. Následující tabulka ukazuje vzorové rozhraní, podobné tomu které jsem použil. [9]

Tabulka 1: Vzorové REST rozhraní

HTTP metoda	**URI**	**Operace**
GET	/administrace/uzivatele	Vrátí list všech uživatelů
GET	/administrace/uzivatele/1	Vrátí data uživatele s ID 1
POST	/administrace/uzivatele	Vloží nového uživatele
PUT	/administrace/uzivatele/1	Upraví údaje uživatele s ID 1
DELETE	/administrace/uzivatele/1	Smaže uživatele s ID 1

2.2 AngularJs

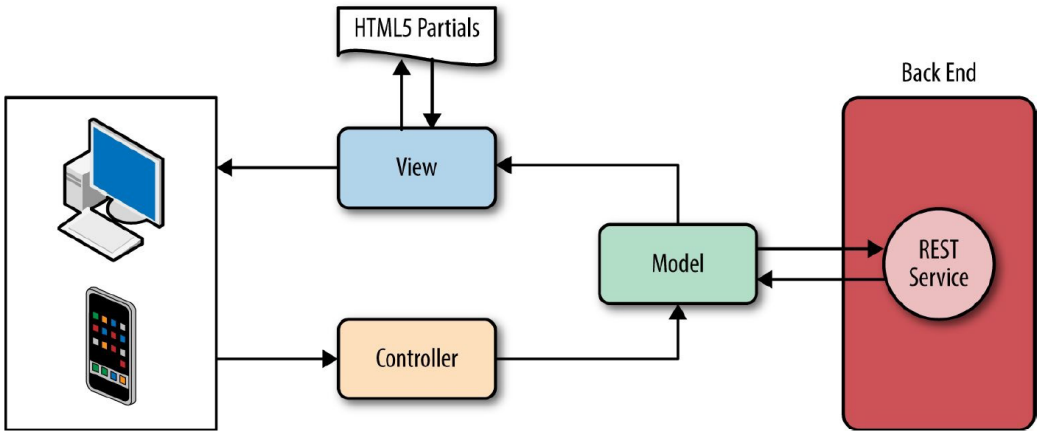
AngularJS je javascriptový rámec založen na model-view-controller architektůře, určený pro vývoj jednostránkových aplikací. AngularJS rozšiřuje HTML direktivy, používá dvoucestný data-binding, dependency injection a u něj vysoká znovupoužitelnost komponent. Celkově tak usnadňuje a urychluje vývoj webové částni aplikace. Pro přístup na backend AngularJS využívá webové služby. [2]

2.2.1 Jednostránková aplikace

Jednostránková aplikace je taková aplikace, která běží uvnitř prohlížeče na straně uživaetle a nevyžaduje znovunačtení stránky během používáni. Funguje to tak, že při vstupu na stránku si prohlížeč stáhne celou aplikaci. Následně se nestahují HTTP šablony, ale pouze data, která zpracovává aplikace. Práce s takovouto stránkou je pak mnohem rychlejší, jelikož se přenáší mnohen nižší objem informaci. [3]

2.2.2 Model–View–Controller architektura

Model–View–Controller architektura dělí aplikaci na 3 části. Model, pohled a kontrolér. Uživateli je zobrazen pohled, pokud provede nějakou akci spustí funkci v kontroléru. Ten aktualizuje model, buď to svými daty, nebo se doptá se serverové strany přes REST. Aktualizovaný model se následně zobrazí v pohledu. Krásně je to zobrazeno v následujícím obrázku.



Obrázek 1: MVC architektura v AngularJS

2.2.3 Dvoucestný databinding

Jedna z výhod frameworku AngularJS kterou programátoři bezpochyby ocení je dvoucestný databinding. Jedná se o automatickou synchoronizaci mezi modelem a pohledem. Což umožňuje rychnou a pro prográmátora celkem nenáročnou synchronizaci mezi modelem v javascriptu a

pohledem v HTML šabloně. Což se děje v například pokud dorazí nová data nebo pokud uživatel provede nějakou akci. [5]

2.2.4 Dependency injection v AngularJS

Dependency injection je návrhový vzor, ve kterém jsou závislosti definovány jako součást konfigurace aplikace. Díky této vlastnosti nemusí programátor ručně vytvářet použité závislosti. AngularJS načte a inicializuje všechny závislosti při spuštění aplikace a následně se stará o celý jejich životní cyklus. Programátor si tak jen napíše, kterou závislost chce použít a pak už volá její metody.

Díky snadnosti vložení závislostí Dependency injection také usnadňuje testování, jelikož se snadno dají místo skutečných závislostí použít podvržené objekty.

Dependency injection je také použit ve rámci Springu, v němž je jednou z klíčových součástí. Spring bude zmíněn v dalších kapitolách. [10]

2.2.5 Základní proměnné

V angularJS se pro pojmenovávání objektů používá camelCase. Neboli způsob psaní víceslovných názvů, kdy se pro oddělení slov nepoužívají mezery, ale místo toho každé další slovo začíná velkým písmenem.

Proměnné od AngulárJS začínají $ nebo $$. Ty jenž začínají dvěma dolary jsou soukromé a programátor by k nim neměl vůbec přistupovat. Ty jenž začínají jedním dolarem se běžně používají. Níže je uvedeno pár hlavních.

$location - poskytuje informace o aktuální URI na jednostránkové aplikaci, také svýma funkcemi umožňuje přechod mezi stránkami.
$http - slouží k volání http metod,
$scope - propisuje proměnné do vybrané šablony.
$rootscope - propisuje proměnné do všech šablon.

2.2.6 Ukázka základního kontroléru

Zde je ukázka kódu základního kontroléru, jenž zavolá http metodu get na url http://test.com. Odpověď pak uloží do modelu na podstránce /game.

```
1  var portRoyalApp = angular.module('portRoyalApp', [
2      'rzModule',
3      'portRoyalApp.gameModule'
4      'portRoyalApp.jakykolivDalsiModul',]);
5
6  var gameCtrl = angular.module('portRoyalApp.gameModule', ['ngRoute']);
7
8  gameCtrl.controller('gameControler', function ($scope, $http) {
9          $http.get('http://test.com').then
```

```
10          (function (odpoved) {
11              $scope.promennaDoSablony = odpoved.data;
12          });
13      })
14      .config(['$routeProvider', function ($routeProvider) {
15          $routeProvider.when('/game', {
16              templateUrl: 'components/templates/game.html',
17              controller: 'gameControler'
18          });
19      };
```

Výpis 1: Ukázka kontroléru v AngularJS

Na 1. řádku se vytváří hlavní angulaří modul.
Na 2. - 4. řádku se do něj vkládají další moduly. Například rzModule je modul stahlý pomocí Node.js, přidávající posuvník a portRoyalApp.gameModule je modul vytvořený níže.
Na 6. řádku se vytváří modul pro hru.
Na 8. řádku se pro tento modul vytváří kontrolér, jenž si po pomocí Dependency Injection přidá 2 závislosti. $scope - do této závislosti se ukládají proměnné jenž se pak dále zobrazují v šabloně. $http - závislost pro volání http metod.
Na 9. řádku se volá funkce get nad URL http://test.com poskytnutá $http. Jedná se o asynchronní metodu tudíž je její návratová hodnota slib zachycen funkci then. Tato funkce má jako vstupní parametr funkci jenž zpracuje slib v momentě jeho naplnění
Na 10. - 12. řádku se zpracovává odpověď na get. Tato odpověď má hlavičku, http status a dále aktuálně jsou podstatná její data. Ty se uloží do $scope tak, že v ní vytvoříme nový objekt promennaDoSablony do nějž přiřadíme data.
Na 14. - 19. řádku $routeProvider konfiguruje pohyb na stránce. Je zde určeno, že pokud uživatel přejde na URI /game zobrazí se mu šablona game.html, jenž má na starost gameControler. Ten se okamžitě při přechodu na tuto URI inicializuje. Tudíž spustí get metodu.

2.3 Další technologie

2.3.1 Node.js

Většinu problémů které řeší programátor v javascriptu řešil už někdo před ním. Z tohoto důvodu by nebylo rozumné, aby programátor psal všechnu funkcionalitu sám. Je rozumnější podívat se zda už tato funkcionalita někde neexistuje. Proto vznikl Node.js. Node.js umožňuje programátorovi najít si funkcionalitu, a přidat ji do konfiguračního souboru. Po následném spuštění Node.js instalace se stáhnou závislosti, jenž programátor požaduje. Ty pak stačí pouze přidat pomocí dependency injection.

Node.js také slouží jako spouštěč úloh což znamená, že dokáže spouštět různé testovací ramce, či fungovat jako server.

2.3.2 Spring

Spring je populární open-source rámec pro vývoj J2EE aplikací. Jeho první verze vyšla v říjnu 2003 od té doby značně nabyl na popularitě.

Spring je označován jako kontejner jelikož je modulární. V základu tudíž neumí téměř nic, až s přidáváním modulů získává další funkcionalitu. Ačkoli přidávání dalších modulů komplikuje přípravu prostředí je to výhoda, jelikož následně na serveru běží pouze to co je opravdu potřeba. Závislosti se přidávají přes Maven. V aplikaci Port Royal jsou použity například tyto moduly:

- Hibernate - toto je implementace Java Pesistance Api, která slouží k namapování javovských entit na relační databázi. Následně pak zajišťuje komunikaci mezi databází a Springem.
- Spring Security - Spring Security poskytuje autentizaci a autorizaci uživatele
- Tomcat plugin - Tomcat je open source webový server sloužící k nasazení aplikace. Implementuje většinu specifikace Java EE API. Tomcat je v této práci použit jako mavenovský plugin. Díky toho není potřeba instalovat Tomcat server. Stačí pouze vytvořit spustitelný war soubor a následně pustit tento plugin mavenovským příkazem
- AspectJ - Tento modul slouží k vytváření aspektu. Aspekty jsou metody, které se volají nad větší skupinou různých tříť u nichž je potřeba stejná funkcionalita. Například autentizaci nebo logování příchozího dotazu, které se nemusí psát pro všechny třídy zvlášť.
- spring websocket - Ve hře Port Royal, kterou hraje více hráčů najednou je potřeba informovat hráče o akcích spoluhráčku. Každý hráč by měl tuto informaci dostat právě jednou a to co nejdříve. Toto zařídí websockety, které mají dvě URL. Jednu na niž se přihlásí všichni hráči ve hře. A druhou na kterou hráči zasílají své akce. Pokud některý zašle svou akci na druhou URL budou o této akci informování všichni hráči kteří poslouchají na první.

2.3.3 Maven

Maven slouží ke kompilaci aplikace, ovšem nejen to. Umí spouštět testy či různé pluginy například výše zmíněný Tomcat plugin. Také slouží k přidávání závislostí ty se stejně jako pluginy definují do do souboru pom.xml. Maven tyto závislostí nebo pluginy pak sám stáhne. Toto jednak usnadňuje přidávání závislostí, jelikož programátor nemusí složitě stahovat různé soubory stačí je pouze vypsat. Ale také usnadňuje přenos programu, jelikož stačí pouze přeposlat zdrojové kódy o kompilaci a závislosti se již dále postará maven.

2.3.4 GIT

git

3 Typy testování

3.1 Smysl testování software

Testování dnes hraje při vývoji software důležitou roli. Organizace i vývojáři pochopili výhody testování hlavně u velkých aplikací, které se vyvíjejí a pak udržují mnoho let. U takovýchto aplikací mnohdy nelze s jistotou vědět kde a jak se změny v kódu projeví. Pokud je ovšem určitá funkcionalita pokryta testy může se říci, že je tato funkcionalita splněna. A nejen to pokud se bude kód upravovat ví se, že testy pohlídají aby byla původní funkcionalita zachována. Takto se předejde vytvoření chyb.

Ve výsledku tedy testy nejsou něco co by programátora zdržovalo. Právě naopak testy zabraňují výskytu chyb a ohlídají, že má aplikace požadovanou funkcionalitu. Takto testování šetří čas programátorů a peníze organizacím.

3.2 Manuální oproti automatickým testům

Testy se dají rozdělit na manuální a automatické. Manuální testování je když se tester vžije do role uživatele a ručně otestuje danou funkcionalitu oproti specifikaci. Tento typ testování je rychlý a jednoduchý tudíž tester ani nemusí mít větší technické znalosti. Tyto testy jsou vhodné pro menší aplikace, které se nebudou měnit.

Problém nastává u větších a složitějších aplikací. U nichž nelze snadno a rychle zkontrolovat všechnu funkcionalitu manuálně. Této funkcionality začne být hodně. Zde jsou již potřeba automatické testy. Tyto testy sice trvá déle napsat, ovšem při několikanásobném opakování testu, jenž je potřeba po úpravách v aplikaci. Jsou již automatické testy časově výhodnější.

Tato práce se zabývá automatickými testy.

3.3 Test-driven development

Klasický vývojový cyklus byl takový, že programátor dostal zadání to nastudoval. Následně napsal kód, trochu to lidově řečeno proklikal, pokud se mu vše zdálo funkční kód odevzdal. Pak bylo dále na testerovi, aby kód pořádně otestoval.

V dnešní době se však do popředí dostává Test-driven development neboli zkráceně už jen TDD. Tento přístup k vývoji předpokládá krátký vývojový cyklus, neboli psaní software po menších částech. Vývojář praktikující TDD po obdržení a nastudování zadání nezačne psát implementaci zadání, ale nejprve vytvoří testy na požadovanou funkcionalitu. Tyto testy samozřejmě bez implementace neprojdou. Vytvoření požadované funkcionality je až další krok. Až v momentě kdy všechny testy projdou by měla být požadovaná funkcionalita vytvořena.

3.4 Úrovně testů

Testy se také dají rozdělit na více úrovní podle toho jak velký objem kódu testují. Programátor většinou při vývoji pouští pouze své testy. Jelikož u větších systémů proběhnutí všech testů zabere několik minut. Všechny testy se tudíž většinou pouštějí až po dokončení práce pro potvrzení, že původní funkcionalita nebyla narušena.

U všech testů platí, že by se navzájem neměly ovlivňovat.

3.4.1 Jednotkové testy

Jednotkové testy jsou zaměřené na otestování funkcionality základních jednotek kódu. Většinou jedné funkce jedné třídy. Výhodou je, že tyto testy jsou rychlé a snadno znovu spustitelné. Po napsání by měly zajistit, že daná část kódu plní svou funkcionalitu. Jedna funkcionalita může mít více testů pro různé vstupní parametry.

Při unit testech v AngularuJS a Springu se využívá Dependency injection. Komponenty, které využívá testovaný kód se nahrazují podvrženými komponentami. Díky čemuž není potřeba žádných závislostí a můžeme si nasimulovat různé situace.

3.4.2 Integrační testy

Integrační testy propojují více komponent. Testují zda jsou správně propojené a plní očekávanou funkcionalitu. Nevýhodou těchto testů je, složitější odhalení chyby pokud test neprojde. Jelikož je třeba debugovat více komponent.

3.4.3 End-To-End testy

End-To-End testy simulují chování koncového uživatele v aplikaci. Tyto testy kontrolují například zda se na stránce objeví určitý prvek, či zda se po kliknutí na něj provede očekávaná akce.

3.5 Testovací technologie

3.5.1 Jasmine

Jasmine open source testovaní je framework pro Javascript. Má za cíl být nezávislý na ostatních frameworcích či vývojářských prostředích. Snaží se také o snadno čitelnou syntaxi. Využívá se pro unit testy nejen v prostředí angularJS. Před každým testem se nejprve injectuje testovaný modul, ten má spoustu závislostí. Pro tyto závislosti se vytvoří podvržené moduly. Takto se zajistí jednak, že se opravdu testuje pouze daný modul, ale také se takto dají nasimulovat různé vstupní data. Například určením hodnot, které budou vracet http odpovědi. Jasmine také umožňuje vytváření takzvaných spy objektů, které kontrolují zda byla metoda zavolána, případně s jakýma hodnotami.

3.5.2 Protractor

Protractor je end-to-end testovací framework pro AngularJS. Protractor spouští testy oproti skutečné aplikaci, která je již spuštěná na serveru. Spustí si prohlížeč, v němž simuluje chování skutečného uživatele. Protractor se na stránce naviguje pomocí http šablony, zde si nalezne elementy podle id, css stylu či angulářího modelu. S těmito elementy pak provádí akce nebo ověření jejich hodnoty oproti očekávané hodnotě. Protractor také umí počkat na načtené stránky tudíž programátor nemusí dávat pevnou dobu čekáni. Což by jednak prodlužovalo běh testu, ale také by při spoždění odpovědi serveru mohlo způsobit zahlášení selhání testu, jenž by ovšem bez mimořádného zpoždění prošel. [11]

3.5.3 Karma

Karma je spouštěč testu v Node.js. Využívá ho Jasmine i Protractor. Spouští prohlížeč, v němž běží testy.

3.5.4 Jersey

Jersey je testovací framework vytvořen pro ověření správnosti serverových komponent. Testuje RESTové služby, funguje to tak že si Jersey rozjede svůj vlastní aplikační server s testovanými REST funkcemi. V této bakalářské práce Jetty ovšem dá se použít i jiný server. Následně se volají tyto resty, odpovědi serveru se nakonec porovnají s očekávanými. Pokud vývojář dostane například zadáni aby vytvořil REST, jenž vrací nějakou hodnotu. Snadno vytvoří test, kde pouze zavolá REST podle zadání a porovná navrácenou hodnotu. U těchto testů je pouze zdlouhavější nastavení prostředí následné testy se píší rychle a snadno.

4 Obsah a funkcionalita aplikace

Hra uvítá nově příchozího hráče uvítací stránkou na níž má možnost podívat se na statistiky hráčů či se přehlásit nebo registrovat. Po přihlášení přibude možnost vytvořit hru do níž se mohou přihlašovat další hráči nebo se přihlásit do již vytvořené hry. Také zde přibude možnost přejít do administrace hráčů pokud má hráč roli uživatele.

4.1 Zkrácená pravidla hry

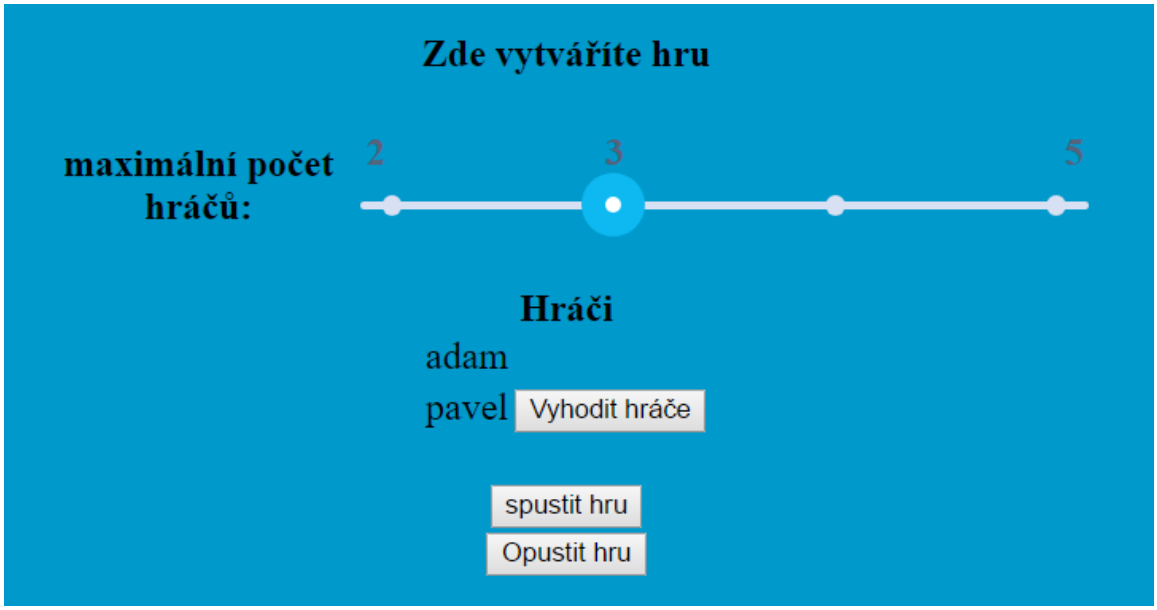
Hra se dělí na dvě fáze. Nejprve aktivní hráč otáčí karty z kupky. Na stole se může objevit více druhů karet. Karty lodi za tyto dostane hráč mince. Karty postav jenž poskytují zvláštní schopnosti. Tyto karty si může hráč koupit pouze pokud má dost mincí. Pak zde jsou Expedice, ty se po otočení přesunou na zvláštní hromádku. Pokud má hráč postavy jenž umožňují provést expedici může je kdykoliv kdy je aktivní vyměnit. Pak zde jsou ještě karty daní, při jejich otočení může hráč splňující určitou podmínku dostat mince. Hráči mající 12 nebo více mincí musejí zaplatit polovinu svých mincí. Točí tak dlouho dokud se aktivní hráč nerozhodne jednu z otočených karet vzít nebo dokud neotočí dva typy stejné lodi.

Následně začne druhá fáze v níž postupně může každý ze zbývajících hráčů kupovat karty ze stolu, pokud zde nějaké jsou. Ovšem při tomto nákupu musí hráči jenž karty otočil zaplatit minci. Jakmile tato fáze skončí posouvá se aktivní hráč. Celá pravidla je možno samozřejmě stáhnout ve hře.

4.2 Obsah a implementace stránky na vytváření hry

Na stránce vytváření hry je použit posuvník k určení maximálního počtu hráčů. Při změně hodnoty na posuvníku se také změní hodnota angulářího modelu uvadajícího maximální počet hráčů. Nad tímto modelem je použit $watch, ten při pohybu posuvníku informuje zadní část aplikace o změně.

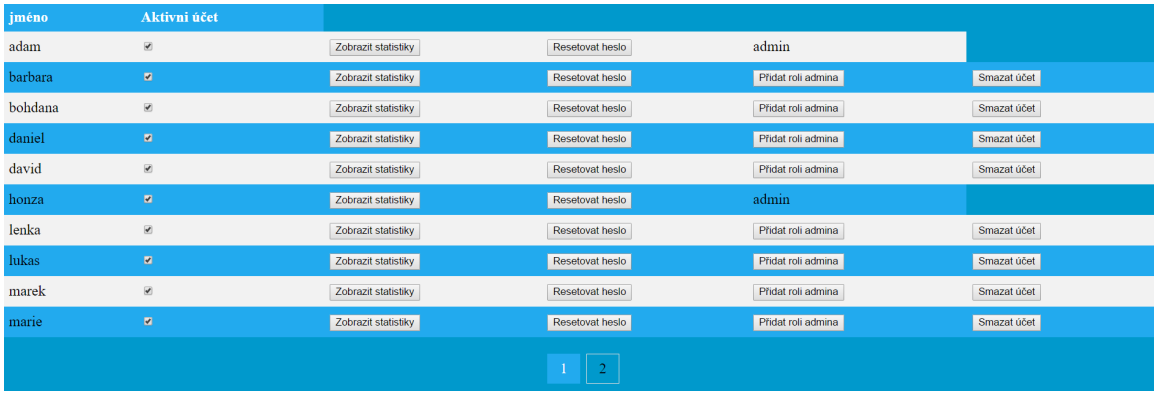
Podobná kontrola probíhá při změně každé proměnné. Nazývá se takzvaný dirty checking. Zajišťuje propsání změn mezi mezi modelem a šablonou, případně mezi modely. Proto se doporučuje, že by počet proměnných na stránce neměl přesáhnout 5000.



Obrázek 2: Stránka na vytváření hry

4.3 Obsah a implementace stránky s administraci

Tato stránka je dostupná pouze uživatelům s rolí administrátora. Pokud by se na ní uživatel nějak dostal Spring Security stejně zablokuje všechny resty začínající /useradministration což jsou resty provádějící akce na této stránce. Tudíž mu nedovolí provést žádnou změnu. Stránka obsahuje seznam uživatelů seřazen podle loginu. Zde použit stránkování, kdy se na jednu stránku načte pouze deset uživatelů.

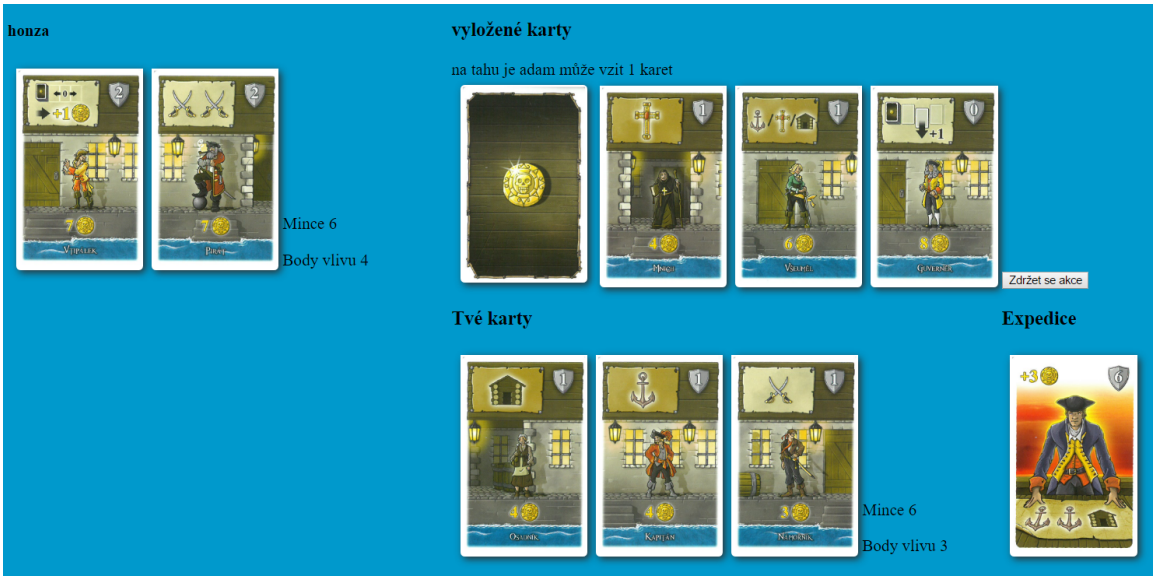


Obrázek 3: Stránka administrace uživatelů

4.4 Obsah a implementace stránky s hrou

Hra je uložena v objektu, jenž obsahuje více stavů hry, které se postupně zobrazují aby ukázaly akci aktivního hráče. Proto by by doptávání na server v určitých intervalech nebylo jen zbytečně náročné na přenos dat a se zpožděním, ale také by se nevědělo kdy byla akce provedena tudíž

by se mohla zobrazit několikrát. Proto zde jsou použity websockety ty po zaslání akce na server aktualizují hráčům ve hře objekt se stavy hry právě jednou.



Obrázek 4: Stránka na které se hraje hra

5 Analýza použitých technologii

Aplikace je vyvinuta ve Springu, jediné co potřebuje je připojení do databáze v níž si sám vytvoří tabulky pomocí Hibernate. Spring kontejner v sobě má dvě části.

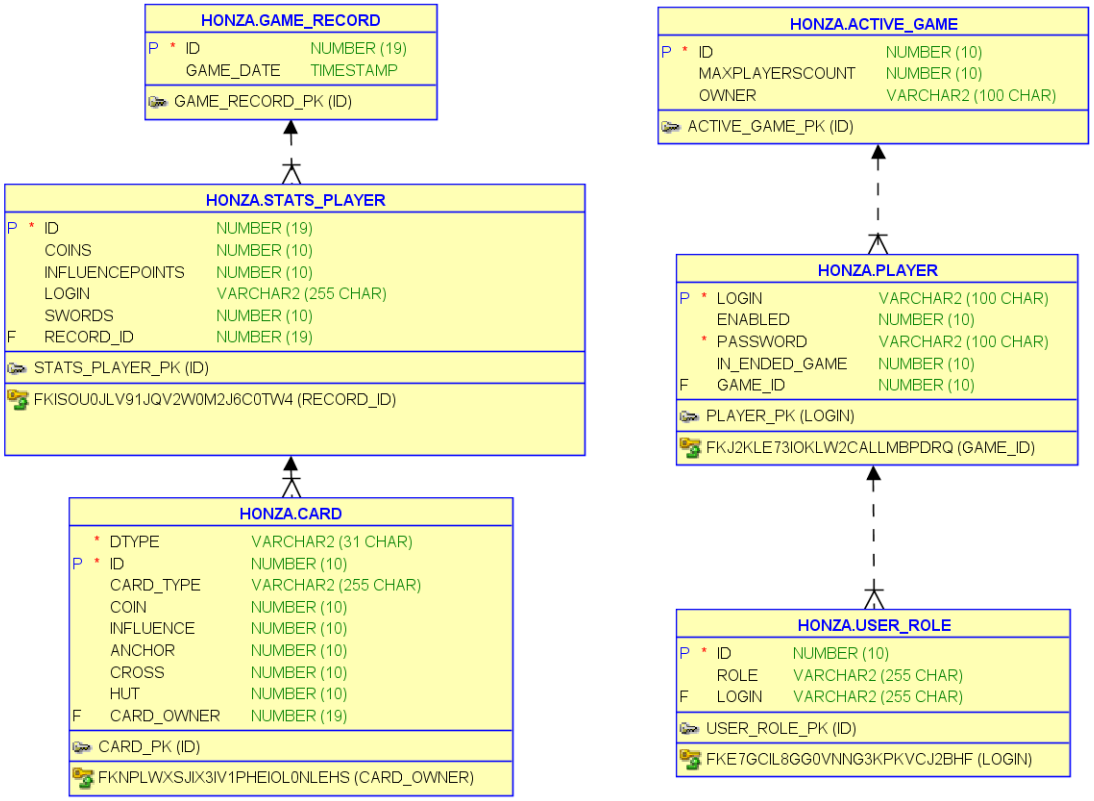
Zadní napsané v Javě jenž se stará o logiku aplikace, přistupující do databáze. Po nasazení na server obsluhuje REST URI. Tato část se nachází ve složce java.

Přední část ve složce webapp tato část obsahuje AngularJS. Při uživatelově vstupu do webové aplikace se celá tato část načte do prohlížeče. Po provedení akce pak tato část volá zadní z níž si bere informace pomocí REST přístupových bodů.

Obě tyto části by se daly snadno oddělit do dvou serveru.

5.1 Schema DB

Hibernate po připojení do databáze vytvoří následující schéma.



Obrázek 5: Databázový model

5.2 Node.js

Pokud chce programátor přidat závislost přes Node.js, Například posuvník. Je nejprve potřeba vyhledat závislost v tomto případě "angularjs-slider": "6.0.1", tato závislost se přidá do souboru

bower.json, konkrétně do objektu dependencies. pak se ve složce s tímto souborem spustí příkaz "bower install". Tento příkaz do nakonfigurované složky stáhne nově přidanou závislost. Následně stačí pouze přidat nově stažený script do indexu a jeho závislost do modulu AngularJS.

5.3 Spring Security a Bcrypt

Bezpečnost aplikace je řešená pomocí Spring Security, jenž po nakonfigurování blokuje URI na jenž uživatel nemá přístup.

Hesla v databázi nejsou samozřejmě uloženy pouze jako text. Pro heslo se při registraci uživatele vygeneruje hash pomocí Bcryptu. Tento hash je pak v databázi uložen místo hesla samotného. Při přihlašování Spring Security kontroluje zda přihlašovací heslo odpovídá vygenerovanému hashi.

5.4 Aspekty

Spring podporuje aspekty, jenž se volají před každým zavoláním vybraných metod. Níže je ukázka aspektu, který před každým zavoláním funkce z balíčku resource nastaví uživatele v beaně UsersProvider. Rozsáhlejší verze tohoto aspektu je použita v aplikaci. Pro beany v rámci Spring je přednastaveno, že je jedna pro celý běh aplikace. U beany UsersProvider je to změněno, vytváří se zvlášť pro každý http dotaz. Jelikož na server přistupuje několik uživatelů zároveň.

```
1  @Aspect
2  public class UserSaver {
3
4      @Inject
5      private UsersProvider usersProvider;
6
7      @Before("execution (* vsb.cec0094.bachelorProject.resource.*.*(..))" +
8              "&& !execution(* vsb.cec0094.bachelorProject.resource.StatsResource.*(..))")
9      public void setUser() {
10         String login = SecurityContextHolder.getContext().getAuthentication().getName();
11             usersProvider.prepareUser(login);
12     }
13 }
```

Výpis 2: Ukázka aspektu

Na 1. řádku se anotacemi určí, že se jedná o Aspekt
Na 4. řádku je použita anotace Inject, touto anotací se říká rámci Spring aby do proměnné níže automaticky přiřadil beanu této třídy.
Na řádcích 7 až 9 Se určí, že se funkce setUser má zavolat před každým voláním funkce třídy jenž je v balíčku resource. Mimo třídu StatsResource jenž beanu userProvider nepoužívá.

Na 10 řádku se pak z kontextu vyčte jméno přihlášeného uživatele.
Na 11. řádku se nastaví beana userProvider.

5.5 Komunikace mezi Spring a AngularJS

Komunikaci mezi zadní stranou aplikace v rámci Spring a AngularJS vždy začíná AngularJS zavoláním určitého koncového bodu, neboli URL přístupného klientským aplikacím běžícím ve Springu. To se děje v případě uživatelovy akce, jenž je třeba uložit na zadní část aplikace. Také v pozadí běží kontrola zda se uživatel nachází na správné URI v rámci aplikace. Jenž se v pravidelných intervalech ptá na jaké URI by se měl uživatel nacházet a kontroluje zda na ní opravdu je. To se děje pro případ, že by uživatel ve hře přešel třeba tlačítkem zpět do statistik či vypnul prohlížeč. Po znovu spuštění aplikace je ho potřeba přesměrovat na URI s hrou. Na uvítací stránce a na stránce s hrou pak běží websockety, pro chat a aktualizování hry.

5.6 Websockets

Zde je ukázka nastavení websocketu pro chat jenž je na úvodní stránce. Websockety je nutno nastavit jak na straně serveru (Springu) tak na straně klienta (AngularuJS).

5.6.1 Strana rámce Spring

Na straně rámce Spring se nastavuje serverová část websocketu. Je zde nutno přidat závislosti ze skupiny "org.springframework" konkrétně "spring-messaging" a "spring-websocket"

V XML konfiguraci se nastaví, že se uživatelé mohou připojovat k websocketům na URI "/chat". Na URI "/messages" půjde následně odebírat zprávy.

V Javě je nutno vytvořit funkci jenž zpracovává příchozí právy. Na prvním řádku se určí URI pro příchozí zprávy. Na druhém řádku se určí URI na které jsou odběratelé, jenž všichni dostanou zpracovanou zprávu. Dále je jen zpracování a odeslání zprávy.

```
1 <websocket:message-broker application-destination-prefix="/port-royal/">
2     <websocket:stomp-endpoint path="/chat">
3         <websocket:sockjs/>
4     </websocket:stomp-endpoint>
5     <websocket:simple-broker prefix="/messages"/>
6 </websocket:message-broker>
```

Výpis 3: XML konfigurace websocketu

```
1 @MessageMapping("/sendMessage")
2 @SendTo("/messages")
3 public Message receive(Message message) {
4     return message;
5 }
```

Výpis 4: Implementace websocketu v javě

5.6.2 Strana AngularJS

V AngularJS je klientská část aplikace. Níže je ukázka použiti, pro níž je nutno mít závislosti "sockjs"a "stomp-websocket" které se přidávají přes Node.js.

Jsou zde 3 asynchronní funkce. Nejprve je nutno se připojit na serverový koncový bod funkci connect. Konkrétně se zde připojuje na URI "/port-royal/chat".

Pak je možno začít odebírat zprávy funkcí subscribe, jenž vypíše do konzole všechny příchozí zprávy. Také je možné odeslat zprávu funkcí send.

```
1  var client;
2  function connect() {
3      var socket = new SockJS('/port-royal/chat');
4      client = Stomp.over(socket);
5      client.connect();
6  };
7  function subscribe() {
8      client.subscribe("/messages", function (message) {
9          var body = JSON.parse(message.body);
10         console.log('message was recived', body);
11     });
12 };
13 function send () {
14     client.send("/port-royal/sendMessage", {}, JSON.stringify(
15         {'text': 'test message',
16         'author': 'Honza Cech'}
17     ));
18 };
```

Výpis 5: Použití websocketu na straně klienta

5.7 Návrh aplikace Angular / Spring

Jak to nakreslit, popr slovne ?

ulm tridni diagram

6 Analýza a implementace testu

6.1 Jasmine

6.1.1 Konfigurace a pouštění

Jasmine testy se pouštění přes Node.js příkazem "npm test"ve složce se souborem package.json. V němž je uvedena cesta ke konfiguračnímu souboru karmy v tomto souboru jsou uvedeny cesty k testům a pluginy pro spuštění prohlížeče v němž se testy jeden po jednom spouštějí.

Samozřejmě, že také existuje software jako například Karma plugin pro InteliJ IDEA jenž umožňují spustit testy jedním kliknutím ve vývojovém prostředí.

6.1.2 Ukázkový test

Zde je ukázkový test jenž ověří, zda se po zavolání funkce login od loginService, zkutečně zavolá backendGateway jenž komunikuje na ven z aplikace a zda vrátí login nově přihlášeného uživatele.

```
1  describe('example test for portRoyalApp.loginService', function () {
2      var $q, $rootScope, loginService, backendGateway;
3
4      beforeEach(function () {
5          module('portRoyalApp.loginService');
6          module({
7              'backendGateway': jasmine.createSpyObj('backendGateway', ['get', 'post'])
8          });
9
10         inject(['$q', '$rootScope', 'loginService', 'backendGateway',
11             function (_$q_, _$rootScope_, _loginService_, _backendGateway_) {
12                 $q = _$q_;
13                 $rootScope = _$rootScope_;
14                 loginService = _loginService_;
15                 backendGateway = _backendGateway_;
16             }]);
17     });
18
19     it('should login user and return his login when login is called', function () {
20         //prepare
21         var loggedUser;
22         var userName = 'MOCKED_USERNAME';
23         var password = 'MOCKED_PASSWOED';
24         var expectedConfig = {
25             params: {
26                 username: userName,
27                 password: password
28             },
29             ignoreAuthModule: 'ignoreAuthModule'
30         };
```

```
31          backendGateway.get.and.returnValue($q.resolve({data: 'MOCKED_USER_FROM_BACKEND'}));
32          backendGateway.post.and.returnValue($q.resolve());
33          //test
34          loggedUser = loginService.login(userName, password);
35          //validation
36          $rootScope.$digest();
37          expect(backendGateway.post).toHaveBeenCalledWith('LOGIN_URL', '', expectedConfig,
                false, true);
38          expect(loggedUser).toEqual($q.resolve('MOCKED_USER_FROM_BACKEND'));
39      });
40  });
```

Výpis 6: Ukázka testu pomocí Jasmine

Na 1. řádku se určuje, že se jedná o test pomocí describe, jenž shlukuje Jasmine testy. První parametr je popis skupiny testů, druhý je funkce se skupinou testů.
Na 2. řádku se vytváří proměnné jenž se budou využívat. Na 4. až 17. řádku je funkce beforeEach jenž se spouští před každým testem v této skupině. V této funkci se nejprve přidává testovaný modul. Následně jeho závislosti, v tomto případě se jedná pouze backendGateway. Zde se nepřidává skutečný modul, ale pouze podvržený objekt. Jenž má dvě funkce get a post, obě bez implementace. Po přidání modulů se na řádcích 10 až 15 vytáhnou z modulů objekty jenž se budou používat.
Na 19. řádu začíná test. To udává funkce it. Ta 2 vstupní parametry, popis jehož jmenná konvence vypadá takto: it(should "očekávané chování"when "co se děje/jaké jsou vstupní podmínky"). Pak následuje funkce s testem.
Na 21. až 30. řádku se připravují pomocné proměnné.
Na 31. a 32. řádku se určují návratové hodnoty funkcí podvržených objektů. Zde je možno také napsat celou novou funkci.
na 34. řádku se zavolá testované funkce.
Na 36. řádku se vyvolá dirty checking zmíněn v kapilone 4.2 . Ten je použit hlavně pro vyhodnocení asynchronních metod.
Na 37. řádku se zkontroluje zda byla opravdu zavolána backendGateway se správnými parametry pro přihlášení.
Na 38. řádku se provádí kontrola vrácené hodnoty.

6.2 Protractor

6.2.1 Konfigurace a pouštění

Protractor se pouští stejně jako Jasmine přes Node.js ve složce se souborem package.json. Ovšem příkazem "npm protractor run". V souboru package.json. Je cesta ke konfiguračnímu souboru

Protractoru, v tom se nastavuje na jaké URL je spuštěn testovaný server nebo v jakém prohlížeči se mají testy spouštět.

6.2.2 ukazkovy test

```
1  describe('examle login test', function () {
2      it('should login user', function () {
3          browser.get('http://localhost:8090/port-royal/#!/login');
4          browser.ignoreSynchronization = true;
5
6          var pageTittle = element(by.id('loginLabel'));
7          expect(pageTittle.getText()).toMatch('Zde se muzete prihlasit');
8
9          element(by.model('userName')).sendKeys('honza');
10         element(by.model('password')).sendKeys('heslo');
11         element(by.id('loginButton')).click();
12         browser.sleep(2500);
13
14         expect(browser.getLocationAbsUrl()).toMatch("/welcome");
15         pageTittle = element(by.css('.welcomeLabel'));
16         expect(pageTittle.getText()).toMatch('Vitej pirate honza');
17     });
18 });
```

Výpis 7: Ukázkový Protractor test

6.3 Jersey

6.3.1 Konfigurace a pouštění

Jersey testy se spouštějí automaticky při kompilaci Jersey testy jsou naming

6.3.2 ukazkovy test

kod s komentari

7 Závěr

a

7.1 Největší komplikace při vývoji

Při vývoji jsem narazil na několik problémů

7.2 Co bych dělal jinak

Při vývoji jsem byly použity konfigurace aplikace pomocí XML souborů. to se nakonec ukázalo jako nevhodné jelikož v dnešní době se do popředí dostává konfigurování pomocí anotací. Tudíž většina návodů byla pro tuto aplikaci nevhodná.

7.3 Jak by se dala aplikace vylepšit

a

Literatura

[1] Ferda Marvenec: Kdesi cosi.

[2] **angularjs** [cit. 2017-01-04]. *Dostupne z: https://docs.angularjs.org/guide/introduction*

[3] **nevim** [cit. 2017-01-04]. *Dostupne z: https://neoteric.eu/single-page-application-vs-multiple-page-application*

[4] **taka sablona** [cit. 2017-01-04]. *Dostupne z: https://www.zdrojak.cz/clanky/zaciname-s-angularjs/*

[5] **angularJS.org** [cit. 2017-02-04]. *Dostupne z: https://docs.angularjs.org/guide/databinding*

[6] **Oracle documentation** [cit. 2017-06-04]. *Dostupne z: https://docs.oracle.com/javaee/7/tutorial/webservices-intro001.htm#GIJVH*

[7] **Oracle documentation** [cit. 2017-06-04]. *Dostupne z: https://docs.oracle.com/javaee/7/tutorial/webservices-intro002.htm#GIQSX*

[8] **Oracle documentation** [cit. 2017-06-04]. *Dostupne z: http://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html*

[9] **Tutorialspoint** [cit. 2017-06-04]. *Dostupne z: https://www.tutorialspoint.com/restful/restful_introducti*

[10] **Williamson, Ken.** [cit. 2017-07-04]. *Dostupne z: Learning Angularjs: A Guide to Angularjs Development. Sebastopol, CA: O, 2015.*

[11] **Jesse Palmer** [cit. 2017-08-04]. *Dostupne z: Testing Angular Appliacction Cover Angular 2. 2016*

A Instalace aplikace

Jak nainsatlovat maven a npm ?