Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Návrh a vývoj internetovej aplikácie pre správu rezervácií

Bakalárska práca

2012

Jakub Uhrík

Univerzita Komenského, Bratislava
Fakulta Matematiky, Fyziky a Informatiky

Návrh a vývoj internetovej aplikácie pre správu rezervácií

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra Informatiky
Školiteľ:	RNDr. Tomáš Kulich PhD.

Bratislava, 2012
Jakub Uhrík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jakub Uhrík
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Návrh a vývoj internetovej aplikácie pre správu rezervácií

Cieľ: Návrh a vývoj internetovej aplikácie pre správu rezervácií

Vedúci: RNDr. Tomáš Kulich, PhD.
Katedra: FMFI.KI - Katedra informatiky

Spôsob sprístupnenia elektronickej verzie práce:
bez obmedzenia

Dátum zadania: 06.10.2011

Dátum schválenia: 10.10.2011

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Ďakujem vedúcemu bakalárskej práce RNDr. Tomášovi Kulichovi PhD. za cenné rady a pripomienky, rodine a blízkym priateľom za pomoc a morálnu podporu.

Jakub Uhrík

Abstrakt

Naša bakalárska práca sa zaoberá návrhom a vývojom internetovej aplikácie pre správu rezervácií. Zamerali sme sa na dôležité časti návrhu aplikácie: ciele aplikácie, skupiny základných používateľov, tvorbu obsahovej štruktúry a navigácie. Vysvetlili sme podstatné časti vybranej architektúry a použitých technológií.

Následne sme popísali použitie technológií pri vývoji aplikácie. Do podrobna sme rozpracovali fungovanie prehliadačovej aplikácie ako aj serverového jadra poskytujúceho API.

Kľúčové slová: internetová aplikácia, rezervácie, portál, MVC

Abstract

Our project talks about the proces of functional design and development of internet application for managing making an appointment proces. We described primary parts of application design like goals of application, essential users, creation of content structure including navigation and basic parts of choosen architecture and selection of technologies.

In the second part we described the use of selected technologies in the development process. In detail we described work of browser application and server core offering API.

Key words: internet application, booking, reservations, portal, MVC

Obsah

Zoznam obrázkov

Úvod

S postupom času sa čím ďalej, tým viac užitočných aplikácií presúva z operačných systémov do prostredia internetového prehliadača. Pôvodne profesionálne nástroje získavajú plne funkčné, prostredníctvom internetu ľahko dostupné a v nezanedbateľnej miere lacnejšie alternatívy.

Z uvedeného dôvodu sme sa rozhodli pre prácu spočívajúcu v podrobnom popise návrhu a vývoja interaktívnej internetovej aplikácie. Primárne sa budeme venovať podrobnému popisu návrhu a tvorby aplikácie, preto vynecháme popis základných princípov tvorby webových stránok, ktorý je už na viacerých miestach kvalitne spracovaný. Ako podklad použijeme vývoj internetovej aplikácie na správu rezervácií. Prejdeme pritom všetky podstatné časti návrhu a vývoja samotnej aplikácie. Zoznámime čitateľa so základnými výhodami aj nevýhodami vybraného postupu a použitých technológií.

V prvej kapitole Špecifikácia projektu predstavíme návrh aplikácie, základné skupiny predpokladaných používateľov, tvorbu obsahovej štruktúry a navigácie. Na tomto mieste uvedieme aj základné prvky použiteľnosti softwaru.

V nasledujúcej kapitole Návrh a architektúra aplikácie podáme a zdôvodníme základné rozhodnutia týkajúce sa architkektúry projektu a použitých technológií. Načrtneme tiež dôvody, ktoré nás viedli k výberu frameworkov a programovacieho jazyka.

Tretia kapitola Aplikácia v prehliadači sa zaoberá tvorbou prehliadačovej aplikácie pomocou vybraných technológií. Vysvetlíme základné prvky fungovania zvoleného nástroja a ukážeme ich použitie v aplikácii.

V štvrtej kapitole Serverová API popíšeme všetky podstatné časti servera poskytujúceho API. Z dôvodov vyplývajúcich z návrhu architektúry aplikácie je táto časť stručnejšia oproti tretej kapitole.

Kapitola 1

Špecifikácie projektu

Ako sme spomenuli v úvode, veľa profesionálnych softwareových riešení sa presúva na internet. Microsoft Word, Excel, Project, ako aj AutoCad, CorelDRAW či Adobe Photoshop majú svoje alternatívy dostupné v internetovom prehliadači.

Aplikácie sprostredkovávané cez internet majú tiež veľkú výhodu v bezpečnosti a zabezpečení proti softwarovému pirátstvu. Tieto aplikácie totiž nikto nestiahne a necrackne.

Nasledujúca kapitola stručne pojednáva o špecifikácii aplikácie a základných rozhodnutiach, ktoré sú potrebné pred začatím vývoja aplikácie.

1.1 Základný popis a ciele

Navrhovaná aplikácia má slúžiť na objednávanie služieb prostredníctvom internetu. Jej cieľom je vytvoriť na internete priestor, v ktorom môžu poskytovatelia ponúkať svoje služby a zákazníci majú možnosť sa na tieto služby objednávať prostredníctvom internetového prehliadača. Aplikácia by mala poskytovateľovi ponúknuť možnosť vytvoriť služby, nastaviť ich otváracie hodiny a typický časový interval. Následne sa na tieto služby môže zarezervovať zákazník prostredníctvom prehľadného kalendára.

Vyhľadávanie konkrétneho poskytovateľa by malo byť jednoduché, a preto je žiaduce aby sa na hlavnej stránke vyskytovalo dominantné vyhľadávacie okno. Toto okno by malo sprístupňovať všetkých poskytovateľov v aplikácii. Z dôvodu návyku ľudí na vyhľadávač google by vyhľadávanie na stránke malo pracovať spoľahlivo, ponúkať pravdepodobne vyhľadávané možnosti priamo počas písania a automaticky dopĺňať vyhľadávaný text.

Aplikácia by mala ponúkať prehľad rezervácií prihláseného používateľa, možnosť zmeniť ich a zrušiť. Poskytovateľovi by mala umožňovať ľubovoľne manipolovať s rezerváciami na ním ponúkané služby a prideľovať svojim zamestnancom práva na

správu týchto služieb a k nim prislúchajúcich rezervácií.

Základné informácie by mali byť obsiahnuté priamo hlavnou stránkou. Doplňujúce by mali byť ľahko dostupné a dobre viditeľné, nemali by však narúšať funkcionalitu portálu alebo zasahovať do aktívnej zóny aplikácie.

Hlavná časť aplikácie by mala byť odľahčená o zbytočnú navigáciu. Tá by mala byť tak umiestnená, aby nerušila majoritnú funkcionalitu.

1.2 Návštevníci a používatelia

Aplikácia je tvorená so zameraním na jednoduché používanie väčšinovým používateľom. Z toho dôvodu boli vytvorené dva základné profily používateľov aplikácie. Prvým je profil používateľa, ktorým je bežný človek prichádzajúci na stránku s cieľom objednať sa na termín napríklad k svojej zubárke. Druhým profilom je poskytovateľ služieb, ktorý chce v aplikácii prezentovať svoje služby a ponúkať ich na objednávanie sa online. Obidva profily zahŕňajú väčšinu návštevníkov aplikácie, preto v nej musia byť zohľadnené ich potreby a požiadavky.

Používateľ

Používateľ je bežný človek. Zameriavame sa na dve skupiny: mladí ľudia (18 - 30 rokov) a ľudia v produktívnom veku (28 - 45 rokov). Mladí ľudia trávia veľa času na sociálnych sieťach. Na vybavovanie podstatných vecí majú málo času, preto privítajú možnosť rýchlo a jednoducho sa objednať na rôzne služby. Ľuďom v produktívnom veku zaberá podstatnú časť dní zamestnanie. Ich prímárnymi komunikačnými prostriedkami sú telefón a e-mail. Táto skupina ľudí máva už svoje rodiny, ktorým sa snaží venovať čo najviac času. Preto je pre nich prínosom vybaviť nutné objednávanie sa rýchlo a jednoducho.

Pre používateľov má byť preto jednoduchým vykonanie nasledujúcich úkonov:

- prihlásiť sa, optimálne by si mala aplikácia toto prihlásenie pamätať,
- vyhľadať svojho pokytovateľa a otvoriť správnu službu,
- ľahko sa orientovať v kalendári a nájsť potrebný termín,
- vytvoriť rezerváciu, v prípade, že používateľ nie je prihlásený, povoliť ju s požiadavkou základných údajov,
- zobraziť vlastné rezervácie,
- upraviť čas rezervácie podľa potreby,
- zrušiť rezerváciu,
- vytvoriť poskytovateľa.

Poskytovateľ

Poskytovateľmi sú obvykle kadernícke a kozmetické salóny, stomatologické ambulancie, súkromné kliniky, športové a relaxačné zariadenia. Pre túto kategóriu je podstatné mať prehľad vo svojich rezerváciách, vedieť ľahko zistiť, kedy byť pripravený na zákazníka. Dôležitým faktom je, že táto aplikácia u nich môže byť otvorená dlhší čas a preto sa musí automaticky obnovovať, alebo ponúknuť možnosť znovu načítať rezervácie. Potrebná je tiež možnosť pridávania a upravovania rezervácií, ako v normálnych otváracích hodinách, tak aj mimo nich.

Podstatné úkony kategórie poskytovateľ sa dajú stručne zhrnúť do nasledujúceho zoznamu:

- jednoduché zaregistrovanie poskytovateľa, nastavenie základných parametrov,
- prihlásenie sa ako poskytovateľ,
- vytvárenie rezervácií,
- prehľadné usporiadanie svojich rezervácií,
- upravovenie rezervácií,
- rušenie rezervácií,
- automatické upozornenie zarezervovanej osoby na zrušenie alebo zmenenie rezervácie,
- automatické upozornenie poskytovateľa formou e-mailu o novej, zmenenej alebo zrušenej rezervácii,
- exportovanie rezervácií do externého kalendára,
- upozornenie zarezervovanej osoby pomocou sms správy.

Pre oba profily návštevníkov je potrebné mať k dispozícii zrozumiteľné materiály, z ktorých sa dá jednoducho dozvedieť, ako služba funguje. Je žiadúce, aby boli do značnej miery reprezentované formou krátkych zaujímavých videí, z ktorých by jasne vyplývala funkcionalita portálu. Z pohľadu používateľa je potrebné zaistiť čo najlepšiu odozvu a podporu prehliadačov. Preto by mala aplikácia reagovať najneskôr do 2 sekúnd.
Podporované prehliadače by mali byť nasledovné:

- Internet Explorer 6 a vyššie,
- Firefox 3.0 a vyššie,
- Opera 9.X a vyššie,

- Google Chrome 3.X a vyššie.

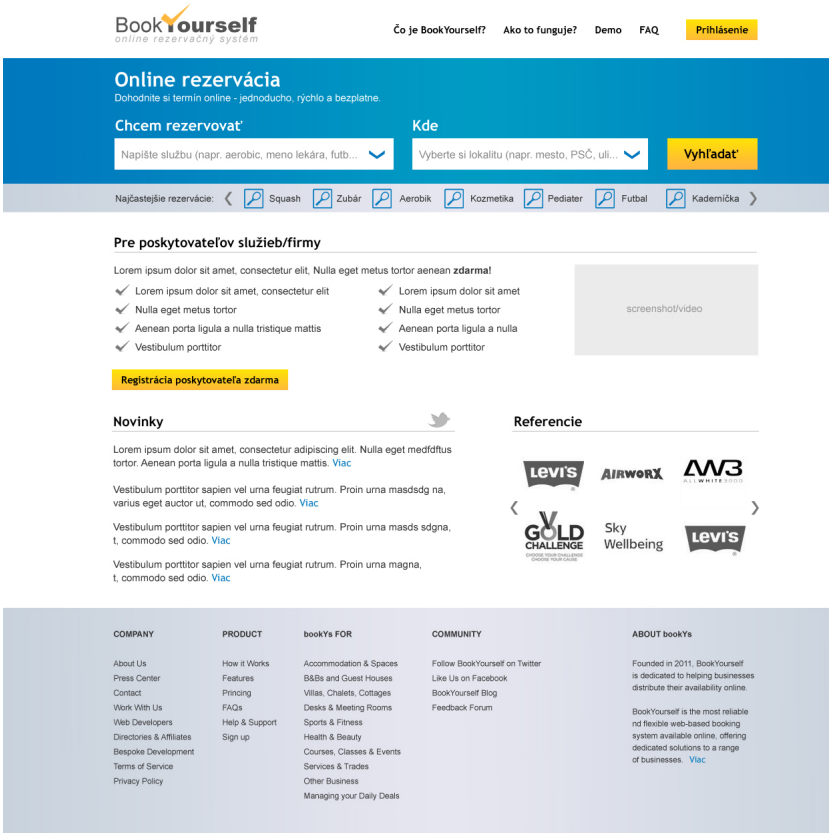
Aplikácia by mala fungovať korektne, v ktoromkoľvek z uvedených prehliadačov nezávisle od operačného systému.

1.3 Obsah a štruktúra

Hlavným obsahom aplikácie je vyhľadávanie poskytovateľov a objednávací kalendár. Ostatné časti aplikácie majú druhotný alebo informačný význam. Preto sa v návrhu sústreďujeme na postup: vyhľadanie poskytovateľa → rezervácia termínu. Obsah aplikácie je rozdelený medzi desať základných typov podstránok. Každá z nich má špecifickú pozíciu v aplikácii a je istým spôsobom nezastupiteľná.

1. Hlavná stránka

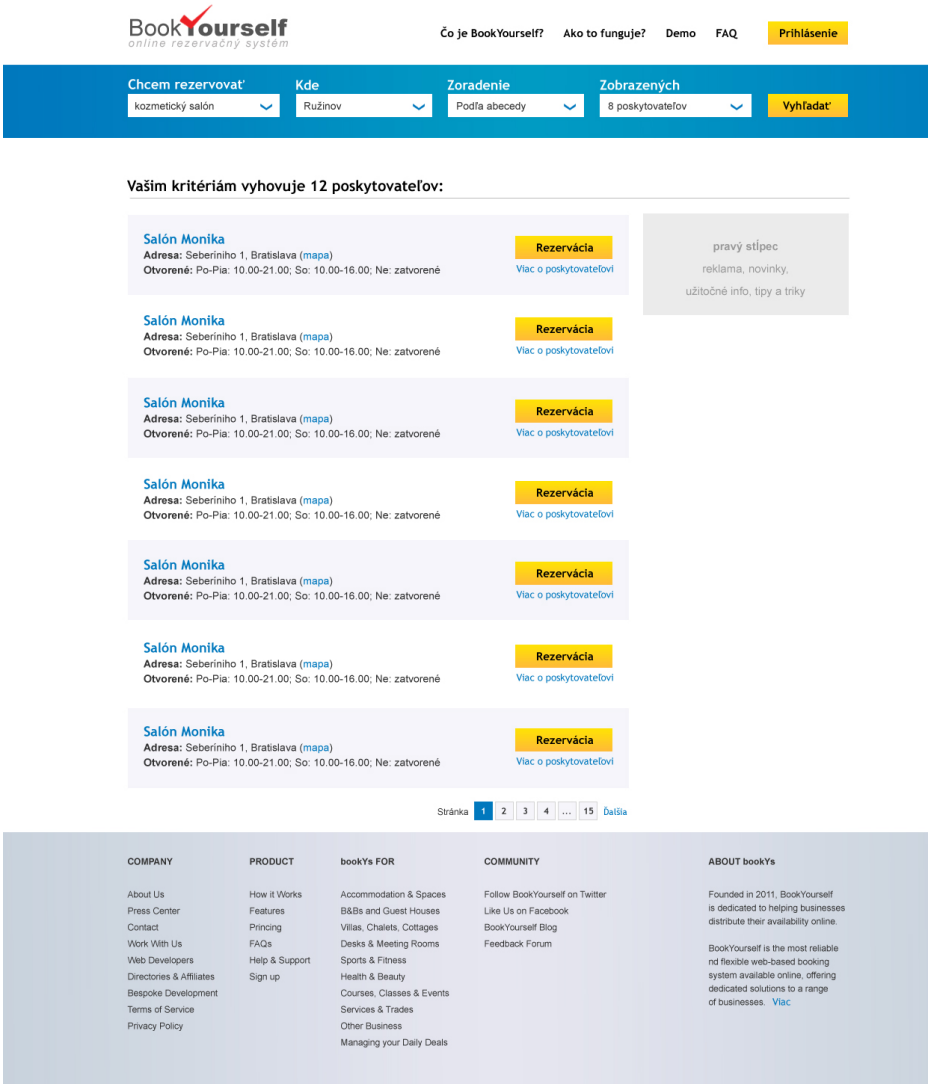
Slúži ako brána do celej aplikácie. Je v nej umiestnený veľký vyhľadávací panel, ako aj stručný popis funkcionality a informačné video. Obsahuje tiež panel noviniek, ktoré sú získavané zo sociálnej siete twitter.com. Je tu umiestnený aj panel s referenciami, ktorý prezentuje významných klientov.



Obr. 1.1: Domovská stránka

2. Stránka vyhľadávania

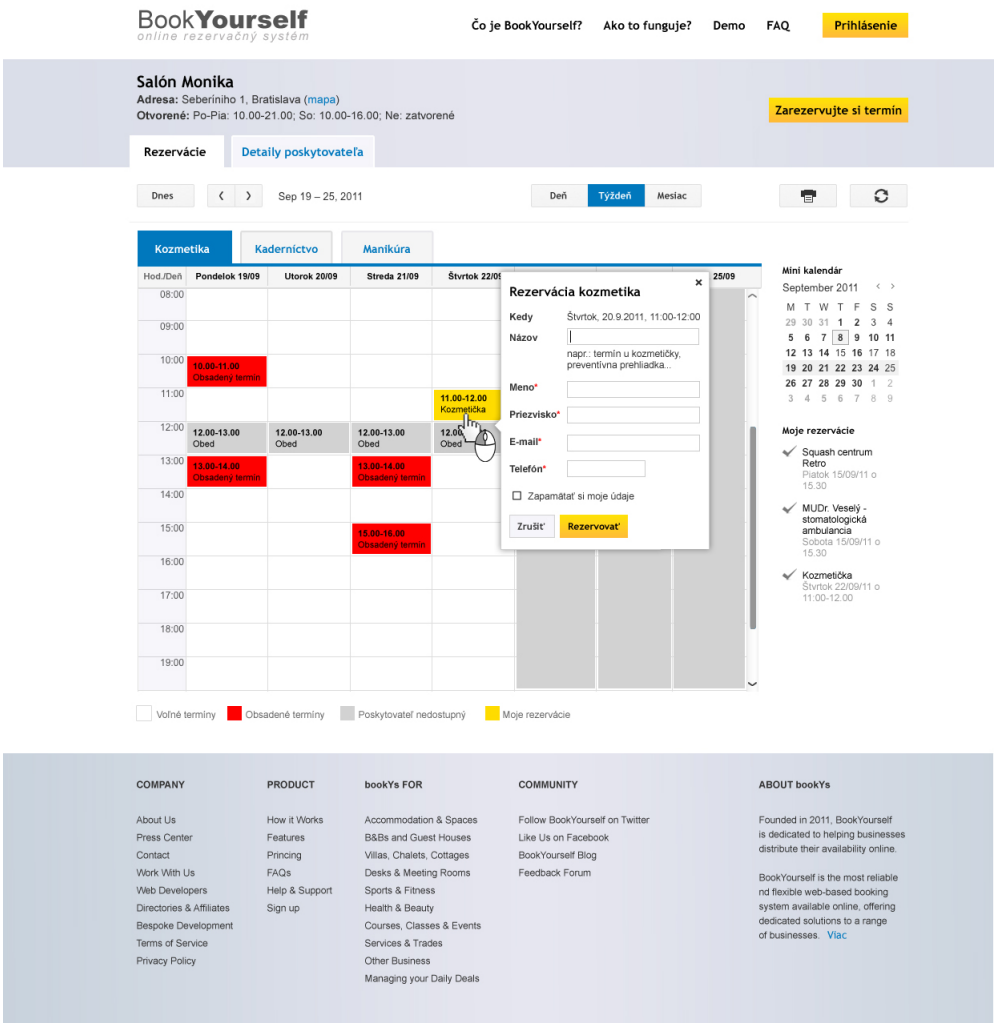
Na stránku s vyhľadávaním sa obvykle prechádza z hlavnej stránky použitím vyhľadávacieho panela. Vyskytujú sa tu podrobnejší vyhľadávací panel s možnosťami zoradenia alebo určenia počtu poskytovateľov na stranu a výsledky vyhľadávania so základnými informáciami o poskytovateľoch. Nachádza sa tu aj otvorená možnosť pre reklamný panel, novinky alebo podobné informácie.



Obr. 1.2: Stránka vyhľadávania

3. Stránka služieb a rezervácií poskytovateľa

Je to veľmi podstatná časť aplikácie. Vykonáva najpodstatnejší úkon celej aplikácie - rezervácia termínu na konkrétnu službu. Obsahuje záložky prepínajúce medzi rezerváciam a detailami poskytovateľa, ovládacie prvky kalendára, záložky so službami, ako aj orientačný kalendárik a najbližšie rezervácie. Podstatná súčasť obrazovky rezervácií je dialógové okno. Otvorí sa po kliknutí na voľný termín a ponúka možnosť tento termín rezervovať.



Obr. 1.3: Stránka rezervácií s dialógovým oknom

4. Detaily poskytovateľa

Na tomto mieste sa vykresľujú fotografie poskytovateľa a jeho popis. Je to miesto, kde prihlásený poskytovateľ môže vstúpiť do administrátorského prostredia a upravovať svoje nastavenia.

5. Môj panel

V mojom paneli sa ukladajú rezervácie vykonané prihláseným používateľom. Môže si ich prehliadať vo svojom kalendári, meniť ich termín pomocou kliknutia na ne a použitia následne otvoreného dialógového okna. Nachádzajú sa tu aj záložky, pomocou ktorých sa dá orientovať v časti "Môj panel", ktorá ponúka možnosť vyhľadávania aj zobrazovania obľúbených poskytovateľov.

6. Prihlásenie

Obrazovka prihlásenie obsahuje jednoduchý prihlasovací formulár a možnosť prejsť na registráciu používateľa alebo poskytovateľa.

7. Registrácia poskytovateľa

Registrácia poskytovateľa je v aplikácii klúčová, predovšetkým z dôvodu, že pre fungovanie aplikácie sú potrební poskytovatelia ponúkajúci svoje služby. Registrácia je rozdelená do štyroch krokov. V prvom si poskytovateľ vyberie druh služieb, ktoré ponúka. V druhom kroku pridá služby a navolí parametre, ktoré potrebuje. V treťom vyplní údaje o zodpovednej osobe a poskytovateľovi: obchodné meno, IČO, DIČ, atď. V prípade, že je už prihláseným používateľom, predmetné informácie sa predvyplnia. Posledným krokom je zhrnutie, kde sa všetky informácie prehľadne zobrazia a je možnosť ich následného potvrdenia, upravenia alebo zrušenia.

8. Registrácia používateľa

Registrácia používateľa je jednoduchá stránka s poľami krstného mena, priezviska, e-mailu, telefónu, hesla a zopakovania hesla. Vstupy sú kontrolované priamo pri vypĺňaní a tak má používateľ okamžitú spätnú väzbu.

9. Úprava poskytovateľa

V tejto časti aplikácie môže používateľ v úlohe poskytovateľa upravovať svoje nastavenia, nastavenia služieb, pridávať, odoberať a upravovať fotografie, ako aj prideľovať práva iným používateľom.

10. Statické stránky

Majú informačný charakter a ich princíp spočíva v čisto textovej forme, bez akejkoľvek interakcie s používateľom, okrem sekundárnej a terciálnej navigácie.

1.4 Navigácia v aplikácii

V aplikácii sme použili tri úrovne navigácie.

Primárna navigácia - vyhľadávanie

Hlavným navigačným prvkom je vyhľadávanie. Pomocou neho je možné dopracovať sa k ľubovoľnému poskytovateľovi v aplikácii. Je výrazným prvkom hlavnej stránky, ponúka automatické dopĺňanie vyhľadávaného výrazu. Má dve vyhľadávacie polia, a to: čo vyhľadávame a kde to vyhľadávame s jasným ohraničením geografického územia. Vyhľadávanie musí byť výrazné tak, aby každého hneď upútalo a motivovalo k použitiu.

Sekundárna navigácia - rýchle informačné linky v hlavičke

Sekundárnou navigáciou sú odkazy na stránky: Čo je to BookYourself?, Demo a FAQ. Účelom tejto navigácie je usmerniť používateľa v jeho prípadných

rozpakoch, spojených s fungovaním aplikácie. Nachádza sa medzi logom a tlačidlom prihlásenia sa, prípadne miesta zobrazenia aktuálne prihláseného používateľa. Umiestnili sme ju tak, aby bola ťažko prehliadnuteľná a zároveň nenarúšala obsah, čo považujeme pri sekundárnej navigácii za podstatné.

Terciálna navigácia - obsah v päte

Terciálna navigácia sa nachádza v päte stránky. Je to takzvaný "megafooter", ktorý čiastočne zastupuje mapu stránky. Je to miesto na odkazy na statické súbory akými sú podmienky, spoplatňovanie, pracovné príležitosti, o spoločnosti, ... Taktiež sa tu vyskytujú často vyhľadávané kategórie alebo výrazy, ako aj návody a pomôcky.

Dôležitá je aj z pohľadu optimalizácie obsahu pre vyhľadávače. Prepája stránku "skrz naskrz", čo dopomáha zvýšeniu PageRanku[1].

Vyššie uvedené úrovne navigácie sú absolútne nezávislé, pracujú na odlišných princípoch. Pre aplikáciu sme vybrali vyhľadávanie ako primárnu navigáciu z dôvodu najlepšieho prepojenia ponúkaných služieb, ako aj z dôvodu zvyku ľudí vyhľadávať cez službu Google. Sekundárnu navigáciu sme navrhli ako prvú pomoc v prípade potreby niečo vysvetliť. Terciálna navigácia dopĺňa možnosť pridať na stránku aj iný ako základný obsah vyplývajúci zo zamerania aplikácie.

1.5 Použiteľnosť

Návrh aplikácie sme vytvorili s ohľadom na použiteľnosť. V dnešnej dobe je potrebné pri tvorbe internetovej aplikácie alebo iného ľubovoľného softwaru myslieť na finálneho zákazníka. Platí totiž jednoduché pravidlo, že ľudia trávia viac času na internete mimo vašej aplikácie ako v nej. Preto je potrebné poskytnúť používateľovi čo najväčšie zadosťučinenie a spokojnosť s použitím aplikácie, aby sa vrátil a použil ju znova.

Prečo je použiteľnosť dôležitá?

- Ak je aplikácia zložitá na používanie, ľudia odchádzajú.
- Ak sa na hlavnej stránke ľudia nedozvedia, čo spoločnosť ponúka a čo môžu urobiť na tejto stránke, odchádzajú.
- Ak sa ľudia stratia na stránke, odchádzajú.

[1] PageRank je algoritmus hodnotiaci stránky na základe spätných odkazov. Viac na http://sk.wikipedia.org/wiki/PageRank

- Ak informácie na webstránke sú ťažko čitateľné, alebo nevedia poskytnúť odpovede na základné otázky, ľudia odchádzajú.[Nie03]

Základným kameňom použiteľného softwaru je teda vyhnúť sa spomenutým negatívnym okolnostiam. To však vo väčšine prípadov nebýva jednoduché kvôli komplexnosti a zložitosti jednotlivých aplikácií. Podstatná je tiež skutočnosť, že programátor, ktorý pracuje na tvorbe projektu, ho vidí pravidelne a vie, ako má fungovať. Považuje ju za jednoduchú a ľahko použiteľnú. Z týchto dôvodov si menusí plne uvedomiť, ako samotná aplikácia pôsobí na konečného používateľa. Práve preto sme považovali za podstatné vypracovať návrh aplikácie pred samotným začiatkom tvorby projektu. Možnosťou je tiež využiť vypracovanie návrhu aplikácie inou osobou ako programátorom, ktorá nie je výrazne zainteresovaná do funkcionality projektu.

Použiteľnosť je "merateľná"v niekoľkých základných aspektoch:

Naučiteľnosť - ako ľahko používatelia splnia jednoduchú úlohu pri prvom stretnutí s prostredím?

Efektivita - keď sa raz používateľ naučí pracovať s prostredím, ako rýchlo vie vykonávať úlohy?

Zapamätateľnosť - keď sa používatelia vrátia po dlhšom časovom intervale nepoužívania aplikácie, aké ľahké je pre nich obnovenie skúseností?

Chyby - koľko chýb používatelia urobia, ako závažné sú tieto chyby a aké ťažké je tieto chyby napraviť?

Zadosťučinenie - ako príjemne sa používa aplikácia? [Nie03]

Kapitola 2

Architektúra aplikácie a použité technológie

2.1 Dátový model a majoritné entity

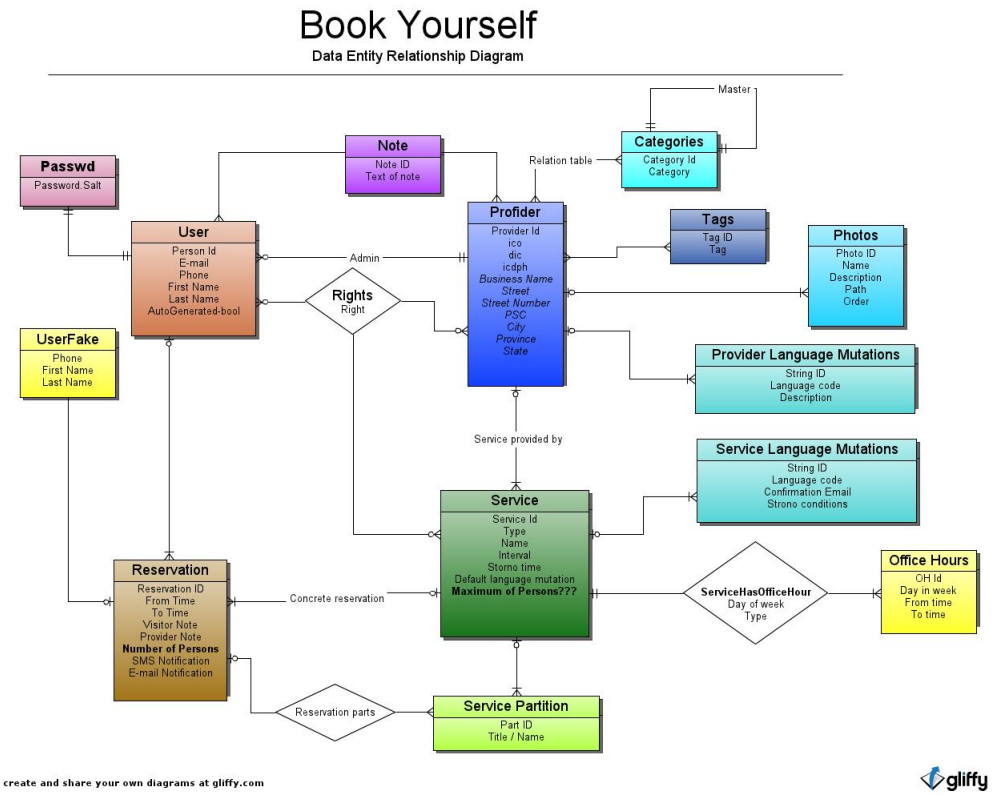
Dátový model zahŕňa komplexnú štruktúru informácií týkajúcich sa samotnej aplikácie. Obsahuje entity používateľa, poskytovateľa, služieb a rezervácií, ako aj "falošného používateľa", fotografie prislúchajúce k poskytovateľom, jazykové mutácie neidentifikujúcich textových častí poskytovateľa a služieb, kategórie a značky (tagy) poskytovateľov, otváracie hodiny a heslá. Celý dátový model je prehľadne zobrazený v entitno-relačnom diagrame na obrázku 2.1.

Používateľ

Entita používateľa je kľúčová v celej aplikácii. Reprezentuje osobu manipulujúcu s používateľským rozhraním. Pod touto entitou je možné sa v aplikácii registrovať, prihlásiť, spravovať rezervácie a vytvárať poskytovateľov a služby.

K danej entite sa viaže tabuľka Passwd obsahujúca heslá používateľov zreťazené s takzvaným salt, ktorý sa z bezpečnostného hľadiska pridáva k heslu pred hashovacou funkciou.

Ďalšou entitou je UserFake, ktorá vzniká pri vytváraní rezervácie poskytovateľom v prípade, že tento zadá odlišné neidentifikujúce údaje pre vytvorenie rezervácie, ale e-mail sa zhoduje s už existujúcim používateľom. Predmetné informácie majú totiž výpovednú hodnotu (viac ľudí používa jeden e-mail, poskytovateľ si môže zapamätať meno) a preto by nemali byť vymazané z databázy. UserFake je tiež vytvorený v prípade, že neprihlásený používateľ zadá e-mail, ktorý už v databáze existuje, informácie s ním spojené však nemôžu byť takýmto prístupom zmenené, a zároveň



Obr. 2.1: Dátový model v ERD diagrame

by sa údaje získané od rezervujúceho používateľa nemali stratiť.

Poskytovateľ

Entita poskytovateľa sa jednoznačne viaže na používateľa reláciou Many-To-One [1], ktorá je povinná. Tabuľka používateľa obsahuje stĺpec user_id typu foreign key [2] ktorý jednoznačne priraďuje používateľa k danému poskytovateľovi. Tento používateľ je takzvaný administrátor poskytovateľa, môže upravovať všetky detaily poskytovateľa, jeho služieb ako aj rezervácií.

K entite poskytovateľa sa viažu aj tabuľky s dodatočnými informáciami, napríklad kategórie, značky a fotografie. Podávajú bližšie informácie o zameraní poskytovateľa a umožňujú lepšie vyhľadávanie poskytovateľa v databáze.

Podstatnou tabuľkou priradenou k poskytovateľovi je tabuľka jazykových mutácií poskytovateľa, ktorá rieši viacjazyčnosť aplikácie. Prostredníctvom nej si poskytovateľ môže prekladať neidentifikujúce atribúty, ako napríklad popis.

[1] Relácia Many-To-One (viac ku jednej) reprezentuje vzťah medzi entitami, kde ku entine na strane One môže byť priradených viac entít na strane Many, avšak k entine na strane Many môže byť priradená len jedna entita na strane One.

[2] Cudzí kľúč je identifikátor v entite, ktorý danej inštancii entity jednoznačne priraďuje inštanciu obvykle inej entity.

Ďalšou tabuľkou prepájajúcou poskytovateľa a používateľa je tabuľka práv. Zaznamenáva práva priradené iným používateľom ako administrátorovi k danému poskytovateľovi. Práva majú dva možné ciele. Sú nimi poskytovateľ, alebo služba prislúchajúca tomuto poskytovateľovi. Cieľ je určený nepovinným parametrom práva, ktorým je ID služby. Vytvorili sme dva rôzne typy práv: EDIT a MANAGECHILDREN. EDIT znamená upravovanie cieľa a MANAGECHILDREN pridávanie a upravovanie potomka. Vzťah predok → potomok je definovaný postupnosťou: poskytovateľ → služba → rezervácia. Tým sme dosiahli zastúpenie podstatnej väčšiny možných priradení práv, napríklad umožnenie spravovať všetky rezervácie, ale neupravovať žiadne služby, možnosť poskytnúť právo upravovať detaily konkrétnej služby za účelom možnosti zmeny otváracích hodín a podobne.

K entite poskytovateľa sa viaže aj tabuľka poznámok, ktorá reprezentuje poznámky poskytovateľa k používateľom. Napríklad: "Chodí neskoro", "Pravidelný zákazník, zľava".

Služba

Služba je entita priradená k danému poskytovateľovi vzťahom Many-To-One. Daný poskytovateľ môže mať viacero služieb. Služby môžu byť niekoľko typov:

individuálne - služba, na ktorú sa môže v jednom čase zarezervovať maximálne jeden používateľ, ako napríklad zubár, kaderníctvo, atď.,

skupinové - služby, na ktoré sa môže v danom čase prihlásiť väčšie množstvo ľudí, ktorých rezervácie nie sú ničím špecifikované, napríklad zumba, plávanie, koncert, atď.,

výberové - služba obsahujúca podslužby s obmedzenou kapacitou, napríklad bowlingové dráhy ako podslužby a bowling ako množinová služba, čiže viaceré služby toho istého druhu zastrešené jednou službou.

K službe sa viaže tabuľka jazykových mutácií služieb, v ktorej sú textové časti služby okrem názvu, a to potvrdzujúci e-mail a storno podmienky.

Každá služba má svoje otváracie hodiny, počas ktorých je možné sa na ňu objednávať. Sú uložené v osobitnej tabuľke, v ktorej sú uvedené všetky možné intervaly počas dňa. Tieto sú previazané so službou pomocou relačnej tabuľky, v ktorej sú dodatočné informácie: deň v týždni, ku ktorému sa vzťahujú tieto otváracie hodiny, typ týchto otváracích hodín.
Typy otváracích hodín:

voľné - otváracie hodiny, počas ktorých sa môže prihlásiť ktokoľvek,

obed - obedová prestávka.

Rezervácia

Rezervácia je nosnou časťou celej aplikácie. Reprezentuje objednávku na konkrétnu službu vykonanú konkrétnym používateľom, vzťahujúcu sa na konkrétny časový interval. K danej rezervácii môžu byť priradené špecifické podslužby v prípade, že služba, ku ktorej sa táto rezervácia vzťahuje je výberová.

K rezervácii môže byť vytvorený tzv. falošný používateľ. Nastupuje v prípade, že rezervácia je vytvorená poskytovateľom alebo neprihláseným používateľom a parametre zadávaného používateľa sa nezhodujú s existujúcou inštanciou v tabuľke s rovnakým e-mailom.

2.2 Prístup oddelených komponentov komunikujúcich pomocou API

Pre základnú architektúru aplikácie sme zvolili prístup oddelených komponentov na serveri a v prehliadači. V prehliadači beží samostatná aplikácia naprogramovaná v JavaScripte, získavajúca zo servera len informácie týkajúce sa biznis logiky samotnej aplikácie. Tým sme odľahčili server od potreby vykresľovania a odstránili sme zbytočné znovunačítavanie takmer identickej stránky.

2.2.1 Výhody

Hlavnou výhodou zvoleného prístupu je značne odľahčený server a od začiatku existujúca API, poskytnuteľná tretej strane. Značne zjednodušuje dodatočný vývoj aplikácií pre mobilné zariadenia aj potencionálnych desktopových aplikácií. Umožňuje tiež lepšie cache-ovanie dát v prehliadači z dôvodu, že celé užívateľské rozhranie je možné obnoviť z offline-ových zdrojov a len potrebné údaje, týkajúce sa databázy, sa získavajú zo servera poskytujúceho API.

2.2.2 Nevýhody

Nevýhodami sú napríklad odlišnosti interpretácie JavaScriptu v prehliadačoch a vyššia náročnosť na výpočtový výkon na strane užívateľa. Značnou je tiež nevýhoda takzvanej dvojitej práce, keďže prístup k zdrojom, validácia a overovanie právomocí musí byť naprogramované na dvoch miestach, v dvoch rôznych programovacích jazykoch.

2.3 Použité technológie

Pri výbere techológií sme zohľadňovali základný návrh aplikácie. V záujme komplexnosti projektu sme sa rozhodli pre framework s návrhovým vzorom Model-View-Controller. Na serveri sme vzali do úvahu ORM pre jednoduchšie dopyty na databázu a zabezpečenie pred SQL injection. Pri voľbe frameworkov bola podstatná dostupnosť programovateľného testovania.

2.3.1 Serverové jadro

Na strane servera sme vybrali programovací jazyk Java a framework Play. Z dôvodu kompilovania zdrojového kódu, a teda vysokej rýchlosti servera, ako aj z dôvodu lepšej výkonnosti aplikácie pri väčších množstvách dát sme sa rozhodli pre programovací jazyk Java. Nevýhodou tvorby internetovej aplikácie v jazyku Java je horšia dostupnosť webhostingu v rámci Slovenska. Tento problém je možné vyriešiť dvoma spôsobmi: webhostingom v zahraničí alebo virtuálnym serverom v rámci cloud-u.

Pre framework Play sme sa rozhodli kvôli dobrej implementácii MVC návrhového vzoru, REST architektúry, ORM systému, JPA, podpory pre formát JSON a integrovanému testovaciemu prostrediu pre testovanie, ako samotných tried tak aj kompletnej funkcionality stránky pomocou zabudovaného prehliadača selenium. Play nám poskytol aj priamo implementovanú možnosť asynchrónneho spracovávania požiadaviek, dobrý systém pre produkciu a jednoduchú konfiguráciu IDE.

Podrobnejšie dôvody pre výber frameworku Play sú dobre popísané na http://entjavastuff.blogspot.com/2012/01/java-web-frameworks-discussed.html.

2.3.2 Aplikácia v prehliadači

Pre aplikáciu na strane prehliadača sme zvolili framework JavaScriptMVC postavený na jQuery. K výberu daného frameworku nás viedli nasledujúce dôvody:

- MVC návrhový vzor,
- dobré rozčlenenie aplikácie do súborového systému,
- dobrý systém produkcie,
- priamo implementovaná a jednoducho upraviteľná komunikácia so serverom vo formáte JSON,
- Views vo formáte EJS, ktoré môžu byť aj dynamicky načítavané,
- jednoduché nastavenie reagovania na udalosti,

- implementované testy a tvorba dokumentácie aplikácie,
- jednoduché reagovanie na udalosti modelu v kontroleroch.

JavaScriptMVC má taktiež veľmi dobrú dokumentáciu a podporu vo forme rýchlych odpovedí na fórach.

2.3.3 Použitá databáza

Ako databázu sme vybrali PostgreSQL. Dôvodov bolo niekoľko. Patria medzi ne podpora transakcií, parametrizovaných dotazov, stabilita databázy, a tiež skutočnosť, že je navrhnutá na spravovanie veľkého objemu dát, ktoré sú v našej aplikácii predpokladané. PostgreSQL odporuje uložené procedúry, ktoré napomáhajú prevencii pred SQL injection.

2.4 Komunikácia medzi prehliadačom a serverom

Komunikácia medzi serverom a prehliadačom je kľúčová pre správne fungovanie aplikácie. Rozhodli sme sa pre formát JSON, ktorý je v dnešnej dobe najpoužívanejším formátom pre tento druh komunikácie. Zvolili sme používanie REST architektúry - objekty na serveri mapujeme na striktne zadefinované URL adresy a manipulujeme s nimi pomocou rôznych stavov HTTP requestov na rozdielne účely.

GET - sa používa na získavanie informácií zo servera, napríklad zoznam rezervácií pre danú službu.

PUSH - je určené na vytváranie informácií na strane servera, napríklad vytvorenie novej rezervácie, zaregistrovanie nového používateľa.

PUT - sa aplikuje pri upravovaní existujúcej inštancie nejakej entity, napríklad úprava profilu poskytovateľa.

DELETE - sa používa na vymazávanie inštancií zo servera, napríklad zrušenie rezervácie.

Entitu, na ktorej robíme úpravy adresujeme pomocou URL nasledovným spôsobom:

GET /entities.json - získanie zoznamu entít,

POST /entities.json - vytvorenie novej entity,

GET /entities/:id.json - získanie inštancie entity s ID == :id,

PUT /entities/:id.json - upravenie inštancie entity,

DELETE /entities/:id.json - vymazanie inštancie entity.

Prenos informácií je realizovaný klasickým JSON. Napríklad pri prenášaní objektu používateľa posielame informácie v trave:

```
{
    "firstName": "Krstnemeno",
    "lastName": "Priezvisko",
    "email": "e-mail@example.com",
    "phone": "0901111111"
}
```

Pri prenose poľa informácií, napríklad výsledkov vyhľadávania poskytovateľa, používame objekt, v ktorom je parameter "count"vyjadrujúci celkový počet inštancií na serveri a parameter "data", v ktorom je pole posielané serverom.

```
{
  "data": [
      {
         "id" : 1,
         "fromUserId": 921,
         "text": "Hello World",
         "createdAt" : 1024324214123
      },
      {
         "id" : 2,
         "fromUserId": 923,
         "text": "Goodnight World",
         "createdAt" : 23524365346543
      },
      ...
  ],
  "count": 100
}
```

Keď žiadame server o nejaký zoznam, máme možnosť zadať základné parametre:

limit - maximálny počet vrátených položiek,

offset - od koľkého prvku v poradí má zoznam začínať,

order - pole stringov "MENO SMERTRIEDENIA".

REST architektúra a formát JSON umožňujú mazať a upravovať aj množiny dát. Tento doplnok sme nepotrebovali, preto sme sa ho rozhodli neimplementovať.

Keď chceme upraviť rezerváciu s ID = 47, vykonáme dopyt na /reservations/47.json metódou PUT s telom

```
{
    "firstName": "Janko",
    "lastName": "Hrasko",
    "email": "janko.hrasko@gmail.com",
    "phone": "09005265652"
}
```

Kapitola 3

Aplikácia v prehliadači

Pre prehliadačovú časť sme zvolili framework JavaScriptMVC, založený na jQuery frameworku. Pri niektorých ovládacích prvkoch sme použili framework jQueryUI. Aplikáciu v prehliadači sme rozdelili na niekoľko základných častí. Ako názov použitého frameworku napovedá, ide o JavaScriptový framework, ktorý v sebe implementuje návrhový vzor Model-View-Controller.

View časť frameworku je zabezpečená pomocou embedded JavaScript, v skratke EJS. Funguje na základe vložených JavaScriptových výrazov do špeciálnych tagov v HTML kontexte. Značnú časť funkcionality formátu EJS veľmi dobre vystihuje nasledujúci príklad:

```
<ul>
<% for(var i=0; i<supplies.length; i++) {%>
   <li><%= supplies[i] %></li>
<% } %>
</ul>
```

Templaty v sebe obsahujú tri základné typy tagov: `<% javascript %>` - obsahuje príkazy ako `for` a podobne, ktoré sa vykonávajú, ale nič nevykresľujú, `<%= javascript %>` - vkladajú “odescapeovaný"text do HTML, `<%# javascript %>` - komentáre a `<%== javascript %>` - vkladajú reťazec priamo do HTML, čiže umožňujú vložiť aj ďalšie elementy. To môže byť aj nebezpečné kvôli potenciálnej hrozbe XSR útokov. Preto sme túto formu vykresľovania použili len na vykresľovanie jazykových mutácií, ktoré popisujeme v časti 3.3.1 Trieda jazykových mutácií.

Za podstatnú zložku frameworku sme pokladali aj správcu závislostí `Steal.js`. Funguje tak, že po načítaní zdrojov uvedených vo formáte znakového reťazca zavolá funkciu, ktorú dostane ako argument. Ak sa cesta vo forme reťazca končí príponou súboru, načíta tento súbor. Ak sa touto príponou nekončí, zoberie poslednú časť cesty

a pridá ju na koniec cesty s príponou `.js`. Napríklad ak cesta je `"cesta/k/balicku"`, tak posledná časť je `"balicku"`, čím vznikne `"cesta/k/balicku/balicku.js"`.

```
steal(
        "nejaka/cesta/k/balicku",
        "nejaka/cesta/k/suboru.js",
        function(){
                coSaMaVykonatPotom();
        }).then(function(){
                totoSaVykonaAzPoVsetkomPredtym();
        })
```

Uvedený príklad zahŕňa všetky podstatné možnosti systému `Steal.js`. V tomto príkaze sa najprv načítajú prvé dva balíčky a to konkrétne: `"nejaka/cesta/k/balicku/balicku.js"` a `"nejaka/cesta/k/suboru.js"`, následne systém zavolá anonymnú funkciu. Keď sa predchádzajúca dovykonáva, zavolá sa anonymná funkcia, ktorá je v parametri funkcie `then`. Takto je možné reťaziť vykonávania funkcií a načítavania zdrojových súborov podľa potreby.

Ďalšie podstatné komponenty JavaScriptMVC - Model a Controller popíšeme v nasledujúcich častiach spolu s využitím týchto prvkov.

3.1 Modely

JavaScriptMVC vytvára triedu `$.Class`, z ktorej dedí trieda `$.Controller` aj trieda `$.Model`. `$.Model` má v aplikácii dvojaký účel.

Nosiče stavu

Inštancia triedy `$.Model` môže obsahovať atribúty, ktoré sa dajú meniť pomocou metódy `attr("meno-atributu", hodnota)` a získavať pomocou metódy `attr("meno-atributu")`. Pri zmene atribútu sa vyvolá udalosť popisujúca túto zmenu, čo umožňuje nastaviť reakcie na danú udalosť.

Komunikácia so serverom

Trieda zdedená od `$.Model` môže obsahovať statické metódy `findOne`, `findAll`, `create`, `update` a `destroy`. Môžu byť popísané reťazcom vo forme `"GET /messages.json"`, popísanej vyššie, framework ich preloží na príslušné AJAXové volania. Uvedené metódy sa používajú na komunikáciu so serverom. `$.Model` tiež vytvára udalosti pri zostrojení, upravení a zmazaní inštancie, pričom iné časti aplikácie môžu na tieto udalosti reagovať.

V aplikácii sme využili obidva prínosy triedy `$.Model`. Pre takmer každú entitu v dátovom modeli sme vytvorili triedu dediacu od triedy `$.Model`, ktorá v sebe obsahuje statické metódy určené na komunikáciu so serverom. Vytvorili sme tiež prototype metódy, ktoré asynchrónne načítavajú závislosti konkrétnej inštancie danej entity. Tieto metódy majú parameter `callback`, obsahujúci funkciu, ktorá je zavolaná so získanou závislosťou ako argumentom a vracajú inštanciu triedy `$.Deferred`.

Trieda `$.Deferred` slúži na pamätanie si stavu riešenia nejakého problému, ktorý a nachádza v stave nevyriešeného problému, vyriešeného problému alebo zlyhania problému. Môžu byť na nej zavolané metódy: `resolve` a `reject` - menia stav, priraďujú riešenie alebo zamietnutie a `then` a `fail` - ako argumenty majú funkcie, ktoré sú zavolané, keď je stav triedy zmenený. Ich parametrom je riešenie problému, alebo jeho zamietnutie, ktoré vzniká zavolaním `resolve` a `reject`.

V modeloch sme nastavili aj validáciu údajov zadaných používateľom, ako povinnosť niektorých údajov, tak aj formát údajov. Na overovania formátu sú používané regulárne výrazy. Formát kontrolujeme najmä pri e-mailoch a telefónnych číslach.

Ďalej nastavujeme konverziu údajov zo servera na inštancie tried v JavaScripte, napríklad časových známok [1] na triedu `Date` a údajov naviazaných reláciami na konkrétne triedy danej entity.

V modeli Rola nastavujeme aj užívateľské práva pre používateľa s danou rolou a vytvárame metódy na ich zistenie. Základné metódy sú `canEditProvider`, `canEditService`, `catAddService`, `canAddPhoto` a `canEditPhoto`. Všetky vracajú `boolean` a z nich metódy zisťujúce právo na upravovanie entity majú navyše parameter `id`, prislúchajúci inštancii tejto entity.

3.2 Kontrolery

Vo frameworku JavaScirptMVC sú kontrolery dediace od hlavnej triedy `$.Controller`. Viažu sa na HTML elementy, napĺňajú ich obsahom a odchytávajú udalosti, ktoré sa udejú vo vnútri elementu, aj mimo neho. Základné odchytávanie udalostí vo vnútri elementu sa definuje ako prototype atribút vyjadrený textovým reťazcom, pozostávajúcim z jQuery selektora a typu udalosti oddelenými medzerou.

Kontrolery týmto spôsobom odchytávajú aj udalosti súvisiace s modelmi, napríklad `created`, `updated` alebo `destroyed`.

V našej aplikácii sú kontrolery umiestnené v priečinku `by/controllers` a patria do namespace-u `By.Controllers`. Kontroler využívaný najmä v tele iného kontroleru, je vytvorený ako jeho atribút. Čiže, ak máme hlavný kontroler

[1] používame UNIX timestamp na prenos časových údajov

`HlavnyKontroler` a primárne v ňom používaný `PodKontroler`, umiestnime podkontroler následne: `...HlavnyKontroler.PodKontroler`. V súborovej štruktúre sa priečinok podkontrolera nachádza v priečinku hlavného kontrolera.

Náš hlavný kontroler je `By.Controllers.Url`. Reaguje na zmeny URL, na základe ktorých posiela dáta získané z adresy častiam stránky. V prípade, ak ide o stránku poskytovateľa alebo služby, získava zo servera základné dáta použitím príslušných modelov. Obsah aplikácie má tri hlavné komponenty.

Hlavička - `By.Controllers.Header` - obsahuje hlavné logo, sekundárnu informačnú navigáciu a prihlasovanie.

Hlavný obsah - `By.Controllers.Content` - obsahuje podstatné funkčné časti aplikácie: vyhľadávacie okno, kalendár rezervácií a podobne.

Päta - `By.Controllers.Footer` - takzvaný MegaFooter, obsahuje terciálnu navigáciu, v ktorej sa nachádzajú odkazy na statické časti aplikácie ako aj odkazy na najčastejšie vyhľadávané kategórie.

Účelom hlavičky je ponúknuť používateľom v každom momente základné informácie o aktuálne prihlásenom používateľovi / poskytovateľovi, odkazy na informačné stránky, napríklad demo, často kladené otázky či "Ako to funguje?". Poskytuje možnosť vrátiť sa na hlavnú stránku pomocou kliknutia na logo a ponúka aj odkaz na prihlasovaciu stránku.

Päta má význam ako z pohľadu SEO, tak aj z pohľadu zamerania obsahu na funkcionalitu a nepridávania zbytočnej navigácie do podstatných častí stránky. Tu sú zhrnuté všetky odkazy na stránky, nie priamo zahrnuté vo funkcionalite, ale potrebné pre informovanosť používateľov. Medzi takéto informácie patria obchodné podmienky, informácie o spoločnosti, kontakt, ponúkaná práca, atď.

Hlavný obsah stránky je miesto, kde sa deje prevažná väčšina funkcionality stránky. Tento kontroler rozlišuje, v akej časti aplikácie sa nachádzame a na základe toho spravuje svoj obsah. V závislosti od informácií, ktoré mu poskytne URL-kontroler zvolí, akú časť stránky vykresliť a aký podkontroler zavolať.

Predmetné stránky sú šiestich rôznych typov.

Hlavná stránka - `By.Controllers.Content.Home` - je úvodná stránka celej aplikácie. Nachádza sa na nej veľké vyhľadávacie pole s možnosťou automatického dopĺňania hľadaného výrazu, a tiež základné informácie o aplikácii a novinky zo siete Twitter.

Stránka vyhľadávania - `By.Controllers.Content.Search` - zobrazuje výsledky

vyhľadávania z hlavnej stránky, ako aj možnosť doplniť, či zmeniť toto vyhľadávanie.

Stránka poskytovateľa
- `By.Controllers.Content.Provider` - obsahuje informácie o poskytovateľovi a kalendár so službami a rezerváciami.

Stránka prihlásenia a registrovania - `By.Controllers.Content.Login` - zahŕňa možnosť prihlásenia a registrácie.

Môj panel - `By.Controllers.Content.MyPanel` - je kalendár s rezerváciami prihláseného používateľa a zoznam jeho obľúbených služieb.

Statický obsah - `By.Controllers.Content.StaticContent` - slúži na vykresľovanie statického obsahu.

Informácie od URL-kontrolera sa delegujú ďalej, čím je zabezpečené, že každý kontroler, ktorý ich potrebuje, má k nim prístup.

Najdôležitejší prvok aplikácie je kalendár s rezerváciami a voľnými termínmi. Tento kalendár obsluhuje kontroler `By.Controllers.Reservations`. Má päť základných častí: kontrolný panel, záložky so službami, orientačný kalendárik, najbližšie služby a takzvanú mriežku. Záložky služieb sa zobrazujú len v prípade, že ide o rezervácie u poskytovateľa a nie v pohľade "moje rezervácie".

Mriežka vykresľuje rezervácie z pohľadu týždňa štandardne, dni horizontálne a hodiny vertikálne. Z dôvodu možných odlišností v otváracích hodinách a teda aj konkrétnych voľných časových intervaloch sme nepoužili tabuľku, ale absolútne umiestňovanie v rámci dní. Vykresľujú sa tu aj vykonané rezervácie, tie však nie sú závislé od časových intervalov. Postytovateľ teda môže zarezervovať ľubovoľný termín, aj mimo otváracích hodín, ktorý sa vykreslí správne.

Rezervácie sa vykresľujú dvoma spôsobmi: cudzie rezervácie - červenou farbou a moje rezervácie - žltou farbou. Na moje rezervácie sa dá kliknúť a v dialógovom okne sa dá zmeniť čas a dátum rezervácie, ako aj poznámka používateľa k tejto rezervácii.

Na voľný termín v mriežke, čiže taký, do ktorého nezasahuje žiadna existujúca rezervácia, je možné kliknúť a objednať sa. Objednáva sa pomocou dialógového okna, v ktorom je nutné vyplniť potrebné údaje. V prípade, že je používateľ prihlásený, tieto údaje vyplní aplikácia automaticky a znemožní ich priame zmenenie.

Kontrolný panel poskytuje používateľovi možnosť pohybovania sa v týždňoch pomocou posúvacích šípok, vrátenia sa na aktuálny dátum, znovuvykreslenia rezervácie a vytlačenia mriežky.

V najbližších rezerváciách sa zobrazujú najaktuálnejšie termíny pre daného používateľa / poskytovateľa. Na konkrétne rezervácie je možné kliknúť, čo používateľa presunie na danú službu v danom časovom období.

Zaujímavou je kombinácia modelu a kontrolera `By.Controllers.Content.StaticContent` a `By.Models.StaticContent`, ktoré spolu zabezpečujú zobrazovanie statického obsahu v aplikácii. Za týmto účelom sme vytvorili externú inštaláciu redakčného systému Wordpress, v ktorej je možné upravovať obsah v pohodlnom WYSIWYG[2] editore. Systém je následne previazaný s aplikáciou pomocou rozšírenia poskytujúceho JSON API, ktorá je dopytovaná aplikáciou s parametrami `slug` [3] a `lang`, jednoznačne identifikujúcimi článok a jeho jazykovú mutáciu. Text získaný z redakčného systému je následne vložený do elementu, na ktorý je aplikovaný kontroler. Výhodou tohto prístupu je jednoduché a dobre funkčné rozhranie na upravovanie textov s podporou pre jazykové mutácie a príjemným textovým editorom.

3.3 Globálne premenné a triedy

Globálne premenné zohrávajú v našej aplikácii dôležitú úlohu. Ukladajú sa do nich centrálne nastavenia, podpora viacjazyčnosti stránky, správa aktuálne prihláseného používateľa a atomicita operácií. Nachádza sa tu trieda spravujúca názov stránky zobrazovaný v záložkách prehliadača, ktorá prijíma objekt popisujúci názov stránky pomocou módu stránky, čiže jej typu a parametrami. Následne nastaví príslušný názov.

Abstrakcia od priameho prideľovania názvu stránke napomáha ľahšej neskoršej úprave detailov tvorby príslušného textu.

Ďalej sa tu nachádza trieda `By.Global.Flash`, slúžiaca na výstup vo forme jednoduchej notifikácie, napríklad úspešného uloženia nastavení alebo zlyhania komunikácie so serverom. Má dve základné metódy `flash` a `error`, ktoré slúžia na vytvorenie notifikácie. Trieda `By.Global.Flash` je inštanciou triedy `$.Model`, čiže je schopná generovať udalosti vzťahujúce sa k nej. Pri každom volaní spomínaných metód vytvorí udalosť `flashed`, na ktorú reaguje kontroler, zobrazujúci tieto notifikácie. Trieda Flash si taktiež pamätá všetky volané upozornenia a vyhlásené chyby.

[2] WhatYouSeeIsWhatYouGet je typ textového editora, v ktorom je možné upravovať obsah aj po vizuálnej stránke, pridávať odstavce, nadpisy rôznych úrovní, atď. http://sk.wikipedia.org/wiki/WYSIWYG

[3] slug, v preklade známka, je skrátená alebo zjednodušená verzia názvu článku, používaná na identifikovanie v URL alebo na iných miestach, kde je nutnosť absencie diakritiky a špeciálnych znakov. http://www.cybercoded.net/permalinks-slugs-and-seo/

3.3.1 Trieda jazykových mutácií

Trieda jazykových mutácií `By.Global.Languages` je inštancia triedy `$.Model`. Slúži na spravovanie aktuálnej jazykovej mutácie a jej menenie. Obsahuje metódy na zmenu jazyka, načítanie jazykového modulu, ako aj na získanie konkrétneho textového reťazca v danom jazyku, reprezentujúceho konkrétny text uživateľského rozhrania. Medzi jej základné metódy patria:

`get(identifikátor-textového-reťazca)`

Vráti daný textový reťazec viažuci sa na identifikátor poskytnutý ako argument funkcie. Textový reťazec vráti v aktuálnom jazyku obalený v HTML elemente SPAN s triedou nastavenou ako identifikácia konkrétneho reťazca pomocou príslušného identifikátora.

V prípade, že daný text nie je definovaný v danej jazykovej mutácii, vráti SPAN element s triedou `error` a textom `MISSING TRANSLATION`, aby samotný výzor aplikácie upozornil na časti jazykovej mutácie, ktoré treba dorobiť.

`getPure(identifikátor-textového-reťazca)`

Má rovnaký účel ako predchádzejúca metóda, ale neobaluje reťazec do elementu SPAN. Je použitá v prípade, keď je toto obalenie nežiaduce.

`add([cesta, poleciest])`

Metóda add má nepovinné parametre. Pomocou nej je možné pridať jednu alebo viac ciest pomocou poľa, prípadne cestu prislúchajúcu k aktuálne vykonávanému JavaScriptovému súboru. Ak cesta nekončí `/mutations/` , metóda si to doplní a v príslušnom adresári hľadá súbor v tvare `kodAktualnehoJazyka.json` . Ak cesta končí `/nieco.js` , túto časť metóda odstráni a doplní `/mutations/` .

Ak teda zavoláme metódu `add()` bez parametrov zo súboru `by/models/languages/languages.js`, metóda túto cestu upraví na `by/models/languages/mutations/`, uloží si ju do svojho zoznamu ciest a v prípade, že aktuálna jazyková mutácia je slovenská, načíta z nej súbor `sk.json` .

`change(kodJazykovejMutacie)`

Slúži na zmenu jazykovej mutácie. Táto zmena overí, či je požadovaná jazyková mutácia dostupná. Ak áno, načíta zo všetkých uložených ciest príslušné `.json` súbory a uloží si aktuálnu jazykovú mutáciu do pamäte. Sama na sebe vytvorí udalosť `changed` z dôvodu, aby časti aplikácie, ktoré potrebovali "čistú"verziu textového reťazca mohli túto verziu zmeniť. Následne zavolá metódu `updateTexts()`.

`updateTexts()`

Nájde všetky príslušné texty obalené SPAN elementom s konkrétnou triedou a zmení ich obsah na text v aktuálnom jazyku. V prípade, že tento text v danom jazyku nie je definovaný, označí ho za chybný, aby aplikácia ukazovala, čo je potrebné preložiť.

V produkčnom móde sa prepisuje metóda add tak, že nič nevykonáva. Tento stav nastáva preto, že všetky moduly jazykov sú zhrnuté do jedného súboru pre každú mutáciu, a ten je načítaný naraz pri inicializovaní triedy `Languages`. Zabraňuje to zbytočnému množstvu requestov na server, čo by mohlo spomaľovať samotnú aplikáciu.

Vo vývojovom prostredí môžeme mať konkrétne jazykové segmenty umiestnené v priečinkoch prislúchajúcich náležitým častiam aplikácie, čo umožňuje lepšiu orientáciu.

3.3.2 Atomicita operácií

Nakoľko celá aplikácia je vykonávaná na strane prehliadača v JavaScripte, nastáva tu potenciálny problém rýchlej užívateľskej aktivity. Môže nastať z dôvodu, že udalosti sú odchytávané asynchrónne, čo je pozitívne pre rýchlosť aplikácie, Nastáva však riziko narušenia jej konzistencie v prípade, že sa nejaká reakcia nestihne dovykonávať kým používateľ opäť zasiahne. Tomuto problému sa dá vyhnúť nasledovným jednoduchým spôsobom.

Pri začatí akcie, ktorá by nemala byť prerušená úžívateľom, akcia zavolá príslušnú metódu. Tá zablokuje uživateľské rozhranie. Pri ukončení akcie zavolá metódu na odblokovanie uživateľského rozhrania.

Princíp je implementovaný globálnou triedou `By.Global.Lock` a kontrolerom `By.Controllers.BlockUI`, ktoré spolu komunikujú jednoduchou cestou. Celá funkcia kontolera je reagovať na udalosti triedy `Lock` a na ich podnet zamykať a odomykať užívateľské rozhranie prostredníctvom jQuery rozšírenia jQuery.BlockUI.

Trieda `Lock` má dve základné metódy.

`lock`

`By.Global.Lock.lock()` vracia integer identifikujúci vytvorený zámok, ktorý sa ukladá to interného poľa triedy. Ak bol tento zámok prvý, je vytvorené udalosť `locked`.

`unlock(integer)`

Metóda `unlock` berie ako parameter `integer` identifikujúci konkrétny zámok. Ak sa tento zámok nachádza v poli zámkov, je odtiaľ odstránený. Ak bol tento zámok posledný, je vytvorená udalosť `unlocked`.

3.3.3 Prihlásený používateľ

Na niektoré úkony na stránke je potrebný autorizovaný prístup. O správu tohto prístupu sa stará trieda `By.Global.ActualUser`, ktorá je inštanciou triedy `$.Model` s účelom generovania udalostí. Jej základnými metódami sú `login(meno, heslo)`, `logout()` a `loadActualUser()`. Označenie prvých dvoch je samovysvetľujúce. Úlohou tretej je pýtať sa servera, či je nejaký užívateľ aktuálne prihlásený, čo má význam v dvoch prípadoch. Prvým je, že ak si používateľ otvorí novú záložku v prehliadači, je požadované, aby bol naďalej prihlásený. Druhým je zapamätanie si prihlásenia s cieľom odľahčenia používateľa od potreby opätovného prihlasovania z rovnakého počítača alebo mobilného zariadenia. Pomocou atribútu `loggedIn` získateľného prostredníctvom metódy `attr("loggedIn")` je možné zistiť, či je nejaký používateľ aktuálne prihlásený.

Táto trieda spravuje zároveň aj role daného používateľa. Atribút `roles` obsahuje pole používateľových rolí, ktoré sú načítateľné zo servera pomocou metódy `loadRoles`. Tá funguje anologicky ako podobné funkcie v modeloch. Metóda `changeRole(integer index)` zmení rolu na `roles[index]`, ak takáto rola v poli existuje. Ak `index == -1`, aktuálny používateľ je nastavený na nepoužívanie žiadnej zo svojich rolí, vystupuje sám za seba.

Pomocou rolí aktuálneho používateľa sa tiež zisťujú jeho aktuálne práva. Atribút `role` je inštancia modelu role, ktorú má používateľ nastavenú na aktuálne používanie. Pomocou nej vieme zistiť, aké má práva.

3.4 Pomocné funkcie a triedy

Framework JavaScriptMVC síce poskytuje možnosť nastaviť kontrolovanie atribútov modelov, ale pre naše potreby ich bolo nutné mierne poupraviť. Z dôvodu predchádzania opakovania kódu na viacerých miestach sme vytvorili pomocné funkcie pre kontrolovanie prítomnosti a tvaru konkrétnych parametrov. Regulárne výrazy na kontrolovanie správneho tvaru jednotlivých atribútov sú uložené v globálnych nastaveniach. Tieto pomocné funkcie vytvoria kontroly definované vo frameworku. Ako správu prislúchajúcu nesplnenej kontrole sme nepriradili textový reťazec, ale funkciu, ktorá dynamicky získava jazykovú mutáciu danej chybovej hlášky a vrácia ju ako svoj výsledok. Tým vieme zabezpečiť, že aj po zmene jazyka sa zobrazujú správne texty chybové hlášky. Kontrolujúce funkcie dostávajú ako parameter pole reťazcov identifikujúcich kontrolované atribúty. Funkcie sú konkrétne tri: kontrola prítomnosti atribútu, kontrola tvaru atribútu pomocou regulárneho výrazu a kombinovaná kontrola tvaru aj prítomnosti. Vytvorením uvedených funkcií sme zabránili vzniku zbytočného množstva opakujúceho sa kódu.

Vypracovali sme tiež pomocnú triedu Interval, ktorá zabezpečuje spravovanie času a časových posunov. V konštruktore berie objekt, v ktorom sú atribúty od, do, časový posun a relatívny časový posun letného času. Táto trieda má metódu `applyToDate`, ktorá aplikuje časový interval na konkrétny dátum a vráti inštanciu tej istej triedy s konkrétnym časovým horizontom. Trieda Interval zastupuje entitu otváracích hodín. Je použitá v modeli služby, kde atribút `officeHours` pozostáva z poľa dní v týždni, pričom každý z nich je poľom intervalov. Otváracie hodiny môžu byť následne aplikované na konkrétny dátum, ako sme uviedli v predchádzejúcom texte.

Pomocnú funkciu `cookie` sme vytvorili z dôvodu jednoduchšieho centrálneho nastavenia dĺžky expirácie. Vytvorili sme aj funkciu, ktorá odstraňuje všetku diakritiku z použitého reťazca. Je použitá pri vyhľadávaní.

Zaujímavé a inovatívne vytvorené pomocné funkcie sú metódy rozširujúce základné JavaScriptové triedy ako `Date` a `Number`. Triede Number sme pridali metódu `pad(int length)`, ktorá dopĺňa potrebný počet núl na začiatku čísla, aby malo dĺžku `lenght`. Výsledok vracia vo forme textového reťazca. V prípade, že dĺžka v desatinnej sústave je dlhšia ako argument funkcie, vráti reťazec obsahujúci neupravené pôvodné číslo. Triede `Date` sme pridali statické metódy `getSummerTimeStartByYear` a `getSummerTimeEndByYear`, ktoré vracajú dátum zmeny z letného času a naspäť. Tento údaj je algoritmicky vypočítateľný, lebo sa riadi presnými pravidlami. Pridali sme aj prototype metódy:

`integer getWeek()` - vracia týždeň v roku, do ktorého patrí inštancia Date, na ktorej je metóda zavolaná,

`integer getDOY()` - getDayOfYear vracia deň v roku,

`boolean useSummerTime()` - overí, či inštancia dátumu patrí do intervalu, v ktorom sa používa letný čas.

Kapitola 4

Serverové jadro a API

V podkapitole 2.3 Použité technológie sme uviedli a zdôvodnili výber programovacieho jazyka Java a frameworku Play ako technológie na strane servera. Play v sebe zahŕňa kompletný návrhový vzor Model-View-Controller. Časť View návrhového vzoru MVC je sprostredkovaná pomocou kvalitného templatovacieho nástroja založeného na jazyku Scala, inšpirovaného ASP.NET Razor-om. Vzhľadom na skutočnosť, že sme zvolili prístup serverovej API, časť View sme v našom projekte nevyužili.

4.1 Modely

Rovnako ako v prehliadačovej časti, aj tu modely reprezentujú entity, vyskytujúce sa v dátovom modeli. Sú zastúpené triedami, dediacimi od triedy `play.mvc.Model`, ktorá v sebe zahŕňa perzistenciu v databáze. Pridáva triede základné metódy save, update a destroy.

Mapovanie do databázy je spostredkované anotáciami. Sú využívané na označenie triedy ako perzistentnej aj na definovanie vzťahov medzi triedami. Medzi základné anotácie patrí `@Entity` - označuje triedu perzistentnú v databáze, `@Id` - označuje parameter použitý ako jednoznačný identifikátor a `@ManyToOne`, `@OneToMany`, `@OneToOne` a `@ManyToMany` - relačné anotácie. Na komunikáciu s databázou sa používa inštancia triedy `Finder`. Obsahuje metódy získavajúce konkrétnu inštanciu na základe parametra `ID` a metódy na vyhľadávanie zoznamu položiek. Vo frameworku je použitý Ebean ORM systém. Jeho značnou výhodou je prehľadná takzvaná bodková syntax.

Z nášho pohľadu za najzaujímavejší model pokladáme triedu používateľa, ktorá spravuje registráciu, prihlasovanie a ďalšie akcie týkajúce sa používateľa. Nastavuje `session` tak, aby si pamätala prihláseného používateľa a ponúka metódu, ktorá vracia daného používateľa, alebo iba jeho `ID`. Spravuje tiež práva používateľa, reprezentovaného inštanciou triedy, na úkony súviciace s aplikáciou. Všetky modely

v aplikácii obsahujú metódu `toJson`, ktorá vytvorí objekt `JsonNode`, reprezentujúci danú inštanciu modela.

4.2 Kontrolery

Kontrolery spracovávajú dopyty na server. Ako parameter dostávajú informácie z URL adresy a pomocou funkcie `request()` získavajú informácie o aktuálne spracovávanej požiadavke. Metóda `request()` vracia inštanciu triedy `Http.Request`, ktorá obsahuje všetky informácie o dopyte, napríklad telo, hlavičku, atď.

V našej aplikácii sme vytvorili kontrolery priradené konkrétnym modelom. Napríklad modelu User sme vytvorili kontroler Users. Spracovávajú požiadavky na konkrétne entity, a to najmä na vytvorenie a upravenie inštancie, vrátenie jednej inštancie alebo zoznamu a zmazanie inštancie z databázy. Pre niektoré modely riešia aj požiadavky na vyhľadávanie a automatické dopĺňanie textu.

Kontrolery v aplikácii dedia od triedy `play.mvc.Controller`. Jednotlivé akcie odpovedajúce na dopyty sú reprezentované statickými funkciami, ktoré vracajú inštanciu triedy Result. V prípade asynchrónneho spracovávania požiadaviek vrátia inštanciu triedy `Promise<Result>`, prísľub výsledku.

Jedoduchá tvorba výsledku je zabezpečená statickými metódami `ok()`, `badRequest()`, `forbidden()`, atď., ktoré nastavujú obsah, hlavičku a chybový kód odpovede podľa ich argumentu a významu. Pomocou nich je možné generovať HTML stránky prostredníctvom `Views`, v našej aplikácii sme túto funkcionalitu nevyužili.

4.2.1 URL mapovanie

Je realizované pomocou smerovacieho konfiguračného súboru `conf/routes`. Obsahuje zoznam všetkých URL adries používaných aplikáciou. Každý riadok obsahuje HTTP metódu, vzor URI a k nemu priradenú akciu.

Pre každý kontroler mapujeme päť základných smerovacích pravidiel.

```
GET     /services.json          controllers.Services.items()
POST    /services.json          controllers.Services.newItem()
GET     /services/:id.json      controllers.Services.item(id: Long)
PUT     /services/:id.json      controllers.Services.editItem(id: Long)
DELETE  /services/:id.json      controllers.Services.deleteItem(id: Long)
```

`GET /entities.json` slúži na získanie zoznamu entít zo servera. V tele dopytu je možné zadávať parametre, napríklad `providerId`, `what` a `where` pri vyhľadávaní.

`POST /entities.json` je URL určená na vytváranie nových inštancií entít. Parametre novej inštancie sa nachádzajú v tele požiadavky vo formáte JSON. `/services/:id.json` je miesto, kde sú mapované akcie s konkrétnou inštanciou na serveri. `:id` označuje miesto pre voliteľnú časť URL, ktorá je následne použitá v akcii prislúchajúcej danému smerovaniu. V aplikácii sme využili metódy GET, PUT a DELETE na získanie, upravenie a vymazanie inštancie zo servera.

Na základe smerovacieho konfiguračného súboru je možné vo frameworku Play spätné zisťovanie URL adresy prislúchajúcej ku konkrétnej akcii. V našej aplikácii sme túto možnosť nevyužili, nakoľko server nevykresľuje žiadne časti HTML.

V smerovacom konfiguračnom súbore sa nastavuje aj prístup ku statickým súborom.

```
GET     /assets/*file     controllers.Assets.at(path="/public", file)
```

kde `*file` označuje cestu vrátane lomítok, ktorá je následne použitá pri hľadaní súboru v priečinku `/public`.

4.3 Správa prístupových práv

V aplikácii sa nachádzajú základné práva na prístup a úpravu jednotlivých poskytovateľov, služieb a rezervácií. Sú definované vzťahom `Many-To-One` medzi poskytovateľom a používateľom. Každý poskytovateľ má jedného používateľa, kotrý je jeho administrátorom. Ten má právo vykonávať všetky úkony spojené s daným poskytovateľom.

V prípade väčších poskytovateľov je potrebné umožniť prideľovanie práv aj iným používateľom ako administrátorovi. Na tento účel sme navrhli jednoduchý systém zahŕňajúci všetky dôležité alternatívy priraďovania práv. Medzi základné možnosti určujúce požiadavky na návrh systému práv patria:

- právo upravovať detaily poskytovateľa bez možnosti akejkoľvek manipulácie so službami a rezerváciami,
- právo pridávať, odoberať a upravovať všetky rezervácie konkrétnej služby bez možnosti inej manipulácie s poskytovateľom,
- právo spravovať služby bez možnosti zasahovania do rezervácií,
- právo upravovať konkrétnu službu a pridávať, odoberať a upravovať rezervácie k nej prislúchajúce,
- spravovať všetky rezervácie bez možnosti ľubovoľných iných úprav.

Pre tento systém sme navrhli dva základné druhy práv. Prvým je právo upravovať detaily cieľa a druhým právo pridávať, odoberať a upravovať potomkov cieľa.

Práva sú uložené v tabuľke `permissions`, ktorá tvorí perzistenciu pre model `Permission`. Majú voliteľný parameter `service_id`, ktorý určuje, či cieľom práva je poskytovateľ alebo jeho služba s `ID service_id`. Entity sú vo vzťahu predok → potomok nasledovne: poskytovateľ → služba → rezervácia. Pridelením práva upravovať potomkov konkrétnej služby určenej nepovinným parametrom `service_id`, vzniká možnosť pridávať, odoberať a upravovať rezervácie prislúchajúce k tejto službe.

Predmetné typy práv sú reprezentované typom `enum`. Právo upravovať reprezentuje `EDIT`, ukladané v databáze znakom `E` a právo spravovať potomkov reprezentuje `MANAGECHILDREN`, v databáze znakom `M`.

4.4 Autentifikácia

Autentifikácia je realizovaná metódami kontrolera `Users` `login`, `logout` a `actualUser`, ktoré sú v smerovacom konfiguračnom súbore mapované na `POST /login.json`, `GET /actual_user.json` a `POST /logout.json`. Login prijíma parametre e-mailu, hesla a zapamätania prihlásenia vo formáte JSON.

Na overenie prihlásenia metódy používajú modely `User` a `Passwd`, na overenie hesla externú triedu. Po realizácii overenia je zavolaná metóda `User.login`, ktorá uloží `ID` prihláseného používateľa do `session`. Ak sa v prihlasovacích údajoch vyskytoval parameter `remember` s priradenou hodnotou `true`, zavolá sa metóda `User.remember()`, ktorá uloží ID aktuálneho používateľa do `cookie` a označí ju `secret` kľúčom.

Záver

V práci sme objasnili všetky podstatné časti návrhu a vývoja interaktívnej internetovej aplikácie. Definovali sme základných používateľov a vytvorili obsahovú štruktúru a navigáciu aplikácie. Popísali sme základný návrh, výber technológií v prehliadači a na serveri a na komunikáciu medzi nimi.

Vysvetlili sme všetky postupy pri praktickom využívaní vybraných technológií pri tvorbe internetovej aplikácie na správu rezervácií.

Aplikácia, ktorú sme popísali, v sebe obsahuje všetku základnú funkcionalitu objednávacieho portálu. V našom návrhu vidíme aj potenciál ďalšieho rozširovania, ktoré je možné rozdeliť do dvoch základných skupín:

1. **technické vylepšenia** - spočívajúce v možnostiach zvyšovania rýchlosti aplikácie a jej spoľahlivosti. Napríklad využitie asynchrónneho spracovávania dopytov na server pomocou systému Akka, priamo zakomponovaného vo frameworku Play a vypracovanie rozsiahlejších testov aplikácie na strane servera aj v prehliadači. Vylepšenia aplikácie sú možné aj na strane prehliadača v oblasti cachovania dát a skrátenia kódu.

   Do tejto oblasti spadá tiež optimalizácia pre vyhľadávače z dôvodu, že užívateľ nezaznamená žiadny rozdiel vo fungovaní aplikácie, ale zlepší sa jej pozícia vo výsledkoch vyhľadávania.

2. **rozšírenia funkcionality** - zahŕňajúce vylepšenia vyhľadávacieho algoritmu, poskytnutie väčšieho výberu typov služieb a zakomponovanie sociálnych sietí do aplikácie.

   Patrí sem aj optimalizácia aplikácie pre mobilné zariadenia, prípadne vytvorenie aplikácie pre Android Market a Apple Store.

Veríme, že naša práca oboznámila čitateľa so základnými postupmi pri návrhu aplikácie ako aj s postupmi pri jej vývoji.

Literatúra

[Aja10] AjaxPatterns.org Wiki. *RESTful Service*, 2010.
http://ajaxpatterns.org/RESTful_Service.

[jav12] *Java web frameworks discussed*, 2012.
http://entjavastuff.blogspot.com/2012/01/java-web-frameworks-discussed.html.

[JQU12] JQUERY FOUNDATION AND THE JQUERY UI TEAM. *jQueryUI Demos & Documentation*, 2012.
http://jqueryui.com/demos/.

[Mic] Microsoft Developer Network. *Model-View-Controller*.
http://msdn.microsoft.com/en-us/library/ff649643.aspx.

[Nie03] Jakob Nielsen. *Usability 101: Introduction to Usability*, 2003.
http://www.useit.com/alertbox/20030825.html.

[Ste07] Stefan Tilkov. *A Brief Introduction to REST*, 2007.
http://www.infoq.com/articles/rest-introduction.

[Sun02] Sun Microsystems, Inc. All Rights Reserved. *Java BluePrints: Model-View-Controller*, 2002.
http://java.sun.com/blueprints/patterns/MVC-detailed.html.

[TS] Jupiter Consulting JavaScriptMVC Training and Support. *JavaScriptMVC Documentation*.
http://javascriptmvc.com/docs.html.

[zen] zenexity & Typesafe. *Play 2.0 documentation*.
http://www.playframework.org/documentation/2.0.1/Home.