Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového Inženýrství

Diplomová práce

HTML 5 a JavaScript pro vývoj a simulaci uživatelského rozhraní automobilu

Bc. Martin Doubek

Vedoucí práce: Ing. Martin Půlpitel

30. dubna 2015

Poděkování

Rád bych poděkoval Ing. Martinu Půlpitlovi za vedení mé práce a Ing. Miloši Purkertovi za to, že mi umožnil pracovat na tomto zajímavém projektu a také za jeho cenné rady. Poděkování patří i mé rodině za podporu během celé doby studia.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 30. dubna 2015 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Doubek, Martin. *HTML 5 a JavaScript pro vývoj a simulaci uživatelského rozhraní automobilu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstract

The main goal of this master thesis is to implement new parts to already existing application which simulates user interface of onboard computer in car. New context of Phone, Info and DAB radio will be added to Tuner, Media and Weather that already exists. Phone is used to simulate incoming and outgoing phone calls, Info shows information about the car state and DAB radio can play digital radio.

Keywords User interface, HTML 5, JavaScript, AngularJS

Abstrakt

Hlavním cílem této práce je implementace nových kontextů do již existující aplikace, která simuluje uživatelské rozhraní palubního počítače automobilu. K existujícím uživatelským rozhraním Tuneru, Medií a Počasí budou přidána rozhraní Telefon, Info a DAB radio. Telefon umožňuje simulaci odchozích a příchozích hovorů, Info zobrazuje informace o stavu automobilu a DAB radio slouží k přehrávání digitálního rozhlasu.

Klíčová slova Uživatelské rozhraní, HTML 5, JavaScript, AngularJS

Obsah

Seznam obrázků

Úvod

Tato práce navazuje na diplomovou práci HTML 5 pro uživatelské rozhraní automobilu[1] Ing. Jana Václavíka. Ta zkoumá možnosti využití webových technologií v palubních počítačích automobilů. Jednou z hlavních motivací pro použití webových technologií je snadná možnost změny vzhledu aplikace, při zachování její struktury. To je velmi důležitá vlastnost, neboť k těmto změnám dochází poměrně často. Použití těchto technologií zjednoduší také testování a prezentaci aplikace, neboť tyto úkony je nyní možné provádět ve webovém prohlížeči libovolného počítače, tabletu či dokonce mobilního telefonu.

Další výhodou použití webových technologií je jejich stoupající popularita a vysoký počet webových vývojářů. Z tohoto důvodu lze také předpokládat méně nákladný vývoj, než kdyby byly použity jiné, méně rozvinuté technologie. Velmi rychle také přibývá nástrojů, které usnadňují práci vývojářům, čímž se tvorba aplikací ještě dále zjednodušuje.

Jedním z cílů této práce je původní aplikaci rozšířit o nové části uživatelského rozhraní. K původním rozhraním pro Tuner, Media a Počasí budou nově přidány rozhraní pro Telefon, Info a DAB radio. K existujícímu rozhraní pro Media bude také přidáno rozhraní pro prohlížení úvodních obrázků médií.

Dalším cílem je implementace grafického rozhraní serveru tak, aby z něj mohly být jednoduše odesílány příkazy na klientskou část. Práce také prozkoumá možnosti některých nástrojů, které by mohly usnadnit další vývoj aplikace. Jedná se například o automatizované generování nových částí aplikace podle šablon či tvorbu vlastních HTML komponent. Výsledná aplikace bude otestována pomocí nástrojů Karma a Protractor.

Kapitola 1

Cíle práce

Tato kapitola shrnuje stav aplikace před zahájením této práce a popisuje důvody vedoucí k jejímu pokračování.

1.1 Současný stav aplikace

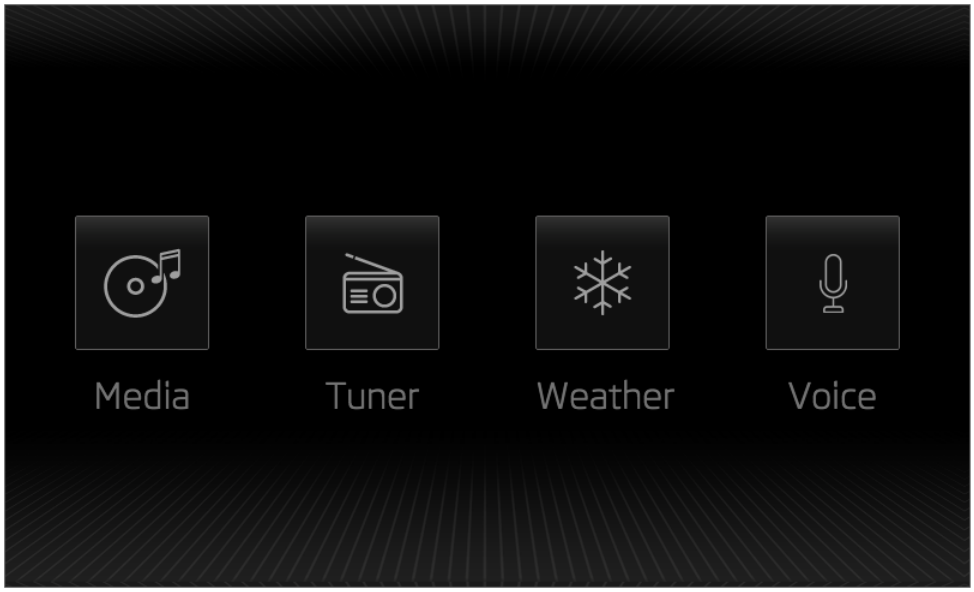
V současné době existuje aplikace realizující část uživatelského rozhraní palubního počítače automobilu za použití webových technologií. Aplikace je realizována pomocí klient-server architektury. Klientská část je tvořena webovou aplikací, která simuluje uživatelské rozhraní palubního počítače v automobilu. Tato aplikace komunikuje se serverovou částí, který funguje jako simulace funkční jednotky automobilu. Komunikace mezi klientskou a serverovou částí probíhá prostřednictvím technologie WebSockets. Hlavní obrazovku klientské části této aplikace ukazuje obrázek 1.1.

Volba vhodných webových technologií probíhala po předchozí analýze jednotlivých možností. Další podrobnosti o realizaci a o tom, na základě čeho byly zvoleny jednotlivé technologie, jsou uvedeny v diplomové práci Ing. Václavíka [1].

1.2 Motivace pro pokračování

Současná verze aplikace demonstruje výhody použití webových aplikací na uživatelském rozhraní pro Tuner, Media a Počasí. Jak ukazuje obrázek 1.2, do budoucna se počítá s rozšířením aplikace o mnoho dalších funkcí, používaných v současných automobilech. Některé z těchto funkcí proto budou implementovány v této práci.

Kromě implementace nových funkcí do klientské části je zde také požadavek na úpravu serveru. Ten slouží jako simulace funkční jednotky automobilu a v současné verzi funguje pouze pasivně, tedy čeká na zprávy od klientské části a zasílá na ně odpovědi. Jedním z hlavních cílů této práce je proto upra-



Obrázek 1.1: Hlavní menu původní aplikace

vit server tak, aby z něj bylo možné aktivně posílat příkazy na klientskou část aplikace. Díky tomu se velmi zjednoduší testování některých částí uživatelského rozhraní.

Původní aplikace také obsahuje několik menších chyb, které je nutné opravit. Vhodné je upravit taktéž některé správně fungující části, jež nejsou implementovány zcela čistým způsobem podle obecně doporučovaných pravidel a zvyklostí.

Obrázek 1.2: Návrh hlavního menu možné budoucí aplikace

Kapitola 2

Analýza a návrh

2.1 Funkční požadavky

- Klientská část aplikace umožňuje spravovat oblíbené kontakty.
- Klientská část aplikace umožňuje zadávat vstup pomocí virtuální klávesnice.
- Klientská část aplikace umožňuje zobrazit seznam telefonních kontaktů.
- Klientská část aplikace umožňuje simulovat hovory.
- Klientská část aplikace umožňuje změnit vzhled aplikace za běhu.
- Klientská část aplikace zobrazuje statistiky o používání vozidla.
- Klientská část aplikace graficky upozorňuje na části vozidla, které vyžadují pozornost.
- Klientská část aplikace umožňuje přehrávat digitální rozhlas.
- Klientská část aplikace umožňuje prohlížet úvodní obrázky přehrávaných skladeb.
- Serverová část aplikace umožňuje spravovat seznam telefonních kontaktů.
- Serverová část aplikace umožňuje přijímat simulované hovory z klientské části.
- Serverová část aplikace umožňuje zahajovat simulované hovory na klientskou část.
- Serverová část aplikace umožňuje spravovat seznam digitálních rádií.
- Serverová část aplikace umožňuje posílat klientské části zprávy o stavu vozidla.

2.2 Nefunkční požadavky

- Klientská část aplikace je optimalizována pro rozlišení 800 x 480 pixelů
- Klientská část aplikace je lokalizována do angličtiny, němčiny a češtiny.
- Klientská část aplikace má dlouhé seznamy optimalizované pro plynulé listování.
- Serverová část aplikace má grafické uživatelské rozhraní běžící v prostředí node-webkit.
- Serverová část aplikace používá databázi MongoDB pro uchování telefonních kontaktů.
- Celé vývojové prostředí včetně simulačního serveru musí fungovat ve Windows7.

2.3 Případy užití

- **Přidání telefonního čísla mezi oblíbené**
  Uživatel otevře menu *telefon - nastavení telefonu - upravit oblíbené kontakty*, vybere volnou pozici, kam chce kontakt uložit a stiskne tlačítko *přidat oblíbený kontakt*. Aplikace zobrazí telefonní seznam, ze kterého uživatel zvolí požadované číslo. Po vybrání čísla aplikace zobrazí daný kontakt mezi pěti oblíbenými čísly na úvodní obrazovce telefonu.
- **Vytočení libovolného čísla**
  Uživatel v hlavním menu zvolí položku *Telefon* a stiskne tlačítko *Klávesnice*. Na zobrazené virtuální klávesnici uživatel zadá požadované číslo. Aplikace po každém stisknutí klávesy zobrazí v pravé polovině obrazovky seznam kontaktů z adresáře, jejichž telefonní čísla odpovídají zadanému vstupu. Uživatel stiskne tlačítko *Telefon*, čímž dojde k vytočení zadaného čísla. Uživatel může také kliknout na kontakt v pravé části obrazovky a zahájit tak vytáčení příslušného čísla. Systém zobrazí vyskakovací okno s informacemi o probíhajícím hovoru.
- **Příchozí hovor**
  Uživatel v serverovém rozhraní pro telefon vyplní číslo, ze kterého má být hovor simulován a stiskne tlačítko *Volat*. Klientská část zobrazí modální překryvné okno s informací o příchozím hovoru bez ohledu na to, která část rozhraní je právě aktivní. Uživatel má možnost hovor přijmout či odmítnout. Během doby hovoru je pozastaveno přehrávání médií.
- **Změna jazyka aplikace**
  Uživatel otevře menu *nastavení - jazyk* a zvolí požadovaný jazyk ze zobrazeného seznamu dostupných jazyků. Aplikace ihned zobrazuje všechny texty v požadovaném jazyce.

- **Změna vzhledu aplikace**
  Uživatel otevře menu *Nastavení - Vzhled* a zvolí jednu z nabízených možností vzhledu aplikace. Aplikace ihned změní svůj vzhled podle dané volby.
- **Přehrání digitálního rozhlasu**
  Uživatel v hlavním menu zvolí položku *DAB* a klikne na tlačítko *list*. Aplikace zobrazí seznam dostupných digitálních rádií. Uživatel zvolí požadované rádio. Aplikace spustí přehrávání požadovaného rádia.
- **Prohlížení úvodních obrázků médií**
  Uživatel v hlavním menu zvolí položku *Obrázky alb*. Aplikace načte a zobrazí úvodní obrázek alba právě přehrávané skladby. Spolu s ním zobrazí po stranách také úvodní obrázky tří předchozích a tří následujících skladeb. Při přechodu na další skladbu dojde k plynulému posunutí všech obrázků o jednu pozici, odstranění krajního obrázku a načtení nového.
- **Vizualizace porouchaných částí vozidla**
  Uživatel pomocí tlačítek v uživatelském rozhraní serverové části simuluje porouchání různých částí vozidla. Klientská část na toto okamžitě reaguje zvýrazněním daných částí na obrázku automobilu přístupném pod menu *Auto - Stav vozidla*.

2.4 Návrh uživatelského rozhraní

Uživatelské rozhraní klientské části aplikace vychází ze skutečného vzhledu palubních počítačů v automobilech Škoda. Grafické podklady pro tuto část byly dodány společností Škoda.

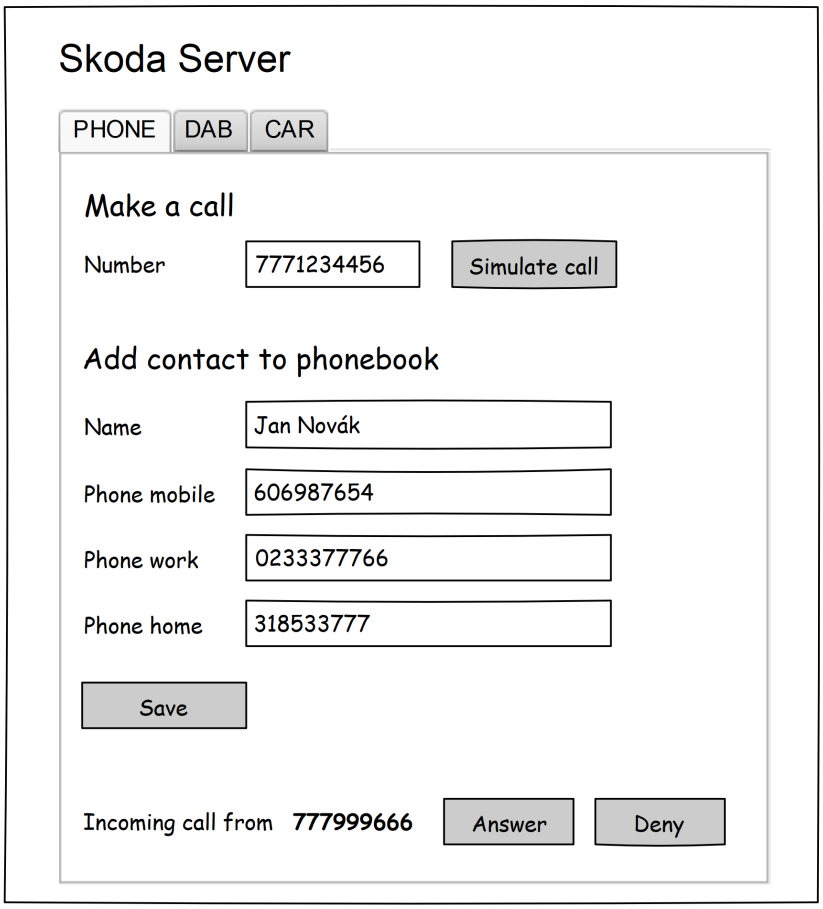
Některé obrázky obsažené v původní aplikaci bylo nutné vyexportovat ve vyšším rozlišení, neboť jsou nově zobrazovány ve větších rozměrech a původní rozlišení nepostačuje [1].

Grafickou podobu uživatelského rozhraní pro serverovou část bylo nutné navrhnout, neboť k ní žádné grafické podklady neexistují. Výsledný návrh, vytvořený v programu Pencil[2], je zobrazen na obrázku 2.1. Hotové uživatelské rozhraní serveru je pak vidět na obrázku 4.1 na straně 27.

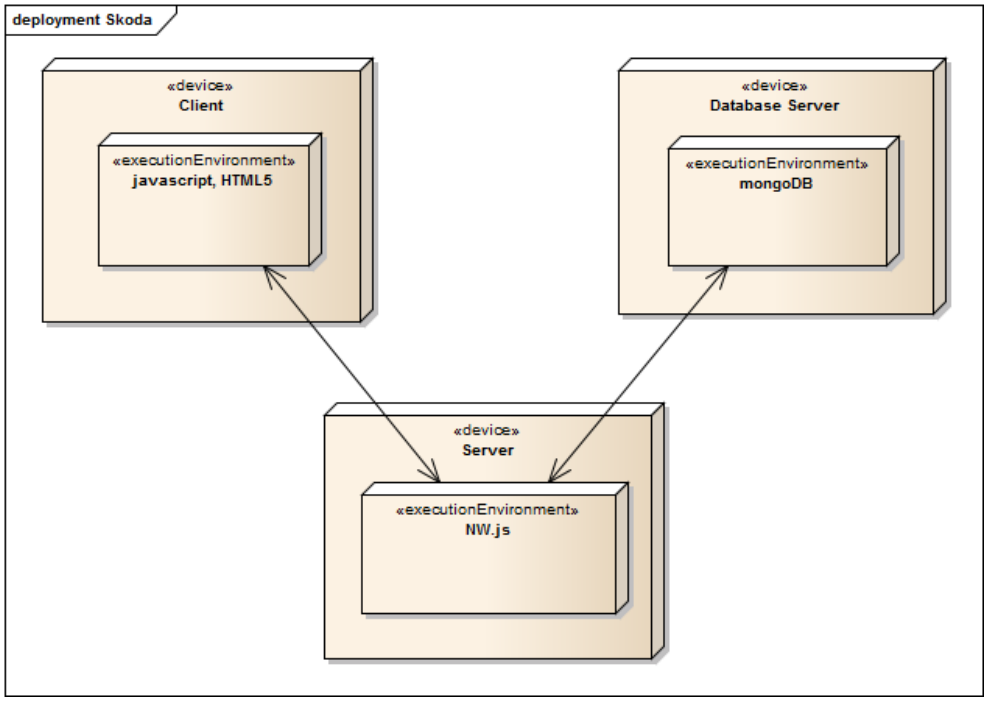
2.5 Diagram nasazení

Obrázek 2.2 zobrazuje diagram nasazení výsledné aplikace. Oproti původní verzi aplikace došlo k rozšíření o databázový server. Ten uchovává důležitá data aplikace a poskytuje je serverové části. Ta je dále zpracovává a posílá klientské části, která je prezentuje uživateli.

[1] To se týká především hlavního otočného menu.
[2] http://pencil.evolus.vn



Obrázek 2.1: Návrh uživatelského rozhraní pro server



Obrázek 2.2: Diagram nasazení aplikace

Kapitola 3

Použité nástroje a technologie

Tato kapitola popisuje nástroje, technologie a postupy, které byly použity pro vytvoření klientské a serverové části aplikace a souvisejících nástrojů. Nejsou zde znovu popsány nástroje a technologie, které již byly použity a popsány v původní aplikaci.

3.1 AngularJS

Klientská i serverová část aplikace jsou postaveny na frameworku AngularJS. Důvody pro výběr právě této knihovny jsou podrobně popsány v diplomové práci Ing. Václavíka [1] v kapitole 3.6 na straně 27.

V současné době je aktuální stabilní verze Angularu 1.3.15, ale očekává se blízký příchod verze 2.0[3]. Ta nebude ani tak velkou aktualizací jako spíše úplným přepsáním celé knihovny a bude obsahovat řadu radikálních, zpětně nekompatibilních změn. Podporovány by měly být pouze nejnovější verze prohlížečů Chrome, Firefox, Opera, Safari a Internet Explorer (IE). Motivací pro novou verzi je zvýšení rychlosti a zlepšení logiky vývoje.

Nová verze by měla být modulární. Některé moduly budou odstraněny z jádra Angularu, čímž bude dosaženo vyšší výkonnosti. Dané moduly budou načteny pouze budou-li právě potřeba.

Změn dozná také základní struktura frameworku a syntaxe zápisu. Měly by se používat kulaté závorky pro volání funkcí a hranaté závorky pro atributy. Nový styl zápisu do šablony pak mohl vypadat následovně:

```
<div>
 <input type="text" [value]="newTodoTitle">
 <button (click)="addTodo()">+</button>
 <tab-container>
  <tab-pane title="Good␣kids">
   <div [ng-repeat|todo]="todosOf('good')">
   <input type="checkbox" [checked]="todo.done">
```

[3] Vydání verze 2.0 je očekáváno koncem roku 2015.

```
    {{todo.title}}
    <button (click)="deleteTodo(todo)">
     X
    </button>
   </div>
  </tab-pane>
</tab-container>
</div>
```

Nová verze Angularu bude vytvořena s ohledem na vývoj mobilních aplikací. Tento přístup vychází z předpokladu, že je jednoduché vytvořit verzi pro stolní počítače ve chvíli, kdy jsou vyřešeny hlavní nedostatky mobilní verze (výkonnost, doba načítání, apod.).

Názory na to, zda je tato komplexní změna krok správným směrem, jsou mezi uživateli Angularu různé. Zatímco zastánci s nadšením očekávají vyšší výkonnost, které by nebylo možné dosáhnout mírnější úpravou knihovny, odpůrci vyjadřují své obavy kvůli chybějícímu plánu pro migraci existujících aplikací na novou verzi Angularu. [17] [18]

3.2 MongoDB databáze

MongoDB patří mezi dokumentově orientované Not only SQL (NoSQL) databáze, které nemají schéma. To umožňuje jednoduše ukládat libovolně strukturovaná data ve formátu JavaScript Object Notation (JSON), přičemž tuto strukturu je možné dynamicky měnit. Vnitřně jsou data reprezentována ve formátu Binary JSON (BSON), přičemž velikost dokumentu je omezena na 4 MB. Jednotlivé databáze není potřeba vytvářet, vzniknou při prvním zápisu. Totéž platí i pro kolekce, což je obdoba tabulek v Structured Query Language (SQL). Všem ukládaným dokumentům je automaticky přiřazen unikátní identifikátor Universally Unique Identifier (UUID). Databázi MongoDB využívá například známý server SourceForge[4]. [3] V aplikaci databáze slouží k uchování telefonních kontaktů, oblíbených kontaktů a digitálních rádií.

3.2.1 Mongoose

Mongoose je knihovna pro Node.js, která umožňuje mapování MongoDB objektů do prostředí Node.js. To znamená, že překládá data z databáze do javascriptových objektů, aby je bylo možné použít v aplikaci a obráceně. Je to tedy jakási obdoba Object-Relational Mapping (ORM), známá z jiných programovacích jazyků. V případě mongoose se nazývá Object Data Mapping (ODM). [5]

Aby bylo možné mongoose používat, je nutné nejdříve přidat tuto knihovnu do Node.js projektu příkazem

```
npm install mongoose -save.
```

[4] http://sourceforge.net

3.2.1.1 Připojení k databázi

```
    var mongoose = require('mongoose');

    var db = mongoose.connection;

db.on('error', console.error);
db.once('open', function() {
  // Create your schemas and models here.
});

mongoose.connect('mongodb://localhost/skoda');
```

Řetězec *skoda* na konci výše uvedeného příkladu označuje databázi, ke které se připojujeme. Tato databáze, jak již bylo dříve zmíněno, je automaticky vytvořena při prvním zápisu.

3.2.1.2 Schémata a modely

Schémata definují strukturu dokumentů uvnitř kolekcí (obdoba tabulek), zatímco modely se používají k vytvoření instancí dat, která budou uložena v dokumentech. Schéma objektu pro uchovaní informací o jedné osobě z telefonního kontaktu může vypadat následovně:

```
var personSchema = mongoose.Schema({
    name: String,
    phoneMobile: Number,
    phoneWork: Number,
    phoneHome: Number
});
```

Zavoláním příkazu níže dojde ke kompilaci modelu *Person*, který bude mít strukturu dříve definovaného schématu.

```
var Person = mongoose.model('Person', personSchema);
```

3.2.1.3 Ukládání dokumentů

Ukládání dokumentů do ModgoDB databáze pomocí mongoose je velmi jednoduché. Stačí vytvořit instanci dříve zkompilovaného modelu a následně zavolat její funkci *save()*.

```
var person = new Person({
    name: "Jan Novak",
    phoneMobile: 777123456,
    phoneWork: 606111222,
    phoneHome: 777222333
});

person.save(function (err) {
    if (err){
        return console.error(err);
    }
```

```
});
```

3.3 Web components

Web Components je soubor standardů, umožňujících tvorbu widgetů[5], které jsou znovupoužitelné. Hlavní výhodou těchto widgetů je, že jejich HyperText Markup Language (HTML) a Cascading Style Sheets (CSS) styly jsou plně odstíněny od zbytku dokumentu. Díky tomu máme jistotu, že vzhled widgetu nebude nechtěně ovlivněn CSS styly dokumentu, do kterého tento widget vkládáme. Stejným způsobem je HTML widgetu chráněno před změněním externím javascriptem. Další výhodou je možnost uložení kódu widgetu do šablon. Ty mohou být použity opakovaně, čímž nedochází k duplikaci kódu a případné změny stačí provést pouze na jednom místě. Web Components také umožňují definovat vlastní HTML elementy. Tyto elementy musí ve svém názvu obsahovat pomlčku (např. <my-element><my-element>). [8] [9]

3.3.1 Polymer

Polymer je jednou z knihoven, která ulehčuje práci se standardy Web Components. Mezi další knihovny patří například X-tag nebo Bosonic.

3.3.1.1 Instalace

Polymer je doporučeno nainstalovat příkazem

```
bower install -save Polymer/polymer#^0.5
```

Další možností je stáhnout zip archiv s potřebnými soubory z Internetu [6] nebo naklonovat projekt na serveru GitHub.

3.3.1.2 Použití existujících elementů

Pro použití již vytvořených elementů stačí do HTML stránky vložit knihovnu Web Components a importovat danou komponentu.

```
<script src="webcomponents.min.js"></script>
 <link rel="import" href="google-map.html">
```

Poté již můžeme tento element ve stránce využívat stejným způsobem jako standardní HTML elementy.

```
<google-map lat="37.790" long="-122.390"></google-map>
```

[5] Widget je základní ovládací prvek pro interakci uživatele s programem (např. tlačítko, textové pole, ...)

[6] https://www.polymer-project.org

3.3.1.3 Vytvoření nového elementu

Do nově vytvořeného elementu musí být vloženo jádro polymeru (`polymer.html`) a musí být definován vlastní element pomocí tagu `<polymer-element>`. Takto vytvořený element pak můžeme v aplikaci používat stylem popsaným v předchozí kapitole.

```
<link rel="import" href="../polymer/polymer.html">

<polymer-element name="my-element" noscript>
  <template>
    <span>This is <b>my-element</b>. </span>
  </template>
</polymer-element>
```

3.4 Angular direktivy

Angular direktivy, stejně jako Web components, umožňují tvorbu vlastních HTML elementů a v mnoha ohledech jsou si tyto dva přístupy podobné. Vzhledem k tomu, že komponenty, které mají být vytvořeny, jsou určeny pouze pro použití v této aplikaci, je vhodnější použití Angular direktiv místo Web components.

Vytvořené šablony pro angular direktivy jsou v tomto projektu umístěny ve složce `templates/components`.

3.4.1 Příklad vytvořené direktivy

Následující direktiva umožňuje snadnou tvorbu obrazovek skládajících se z nadpisu, tlačítka zpět a seznamu položek, kterým je možné se posouvat v případě, že jich je více, než je možné zobrazit najednou. Příkladem takové obrazovky je nastavení jazyka aplikace zobrazené na obrázku 5.11 na straně 45. Tento typ obrazovky se v aplikaci velmi často opakuje a díky této direktivě je výsledný kód významně kratší a přehlednější.

```
app.directive('myScrollScreen', function () {
  return {
    restrict: 'E',
    transclude: true,
    scope: {
      heading: "@",
      back: "@"
    },
    templateUrl: 'templates/components/scroll-screen.html',
    compile: function (element, attributes) {
      return {
        post: function (scope, element, attributes, controller,
            transcludeFn) {
          baseCtrl = new BaseCtrl();
        }
```

```
      };
    }
  };
});
```

Direktiva má název `myScrollScreen`. Použití prefixů před názvem direktivy je vhodné z toho důvodu, aby nedošlo ke kolizi názvů v případě, že by například budoucí verze HTML zavedla element nazvaný `ScrollScreen`.

Pomocí hodnoty `restrict: 'E'` je nastaveno, že daná direktiva bude použita pouze při shodě s názvem elementu. Toto nastavení se používá, pokud je vytvářena nová komponenta. Další možností je `restrict: 'A'` pro použití v případě shody s názvem atributu. Toho se využívá při obohacování elementu o novou funkcionalitu. Možné je také použití při shodě s názvem třídy a to pomocí `restrict: 'C'`. Jednotlivé možnosti je také možné kombinovat, např. `restrict: 'AEC'`.

Možnost `transclude: true` označuje, že daná direktiva může obalovat libovolný HTML obsah.

Následující řádky říkají, že direktiva bude přijímat parametry `heading` a `back` pod stejnými názvy. Parametr `heading` slouží k nastavení nadpisu obrazovky a `back` určuje, na kterou obrazovku se má přejít v případě stisknutí tlačítka zpět.

Hodnota `templateUrl` udává cestu k šabloně, která má být použita pro tuto direktivu a `post` označuje funkci, která má být spuštěna po kompilaci šablony. V tomto případě slouží k tomu, aby došlo k aktualizaci posuvníku po načtení obsahu.

3.4.2 Šablona direktivy

Níže je uveden zkrácený kód šablony pro direktivu `myScrollScreen`, ukazující její nejpodstatnější části.

```
  <section class="buttons">
    ...
      <h1>{{heading| translate}}</h1>
    ...<a href="{{back}}" class="nav-link"></a>
    ...
</section>

<section class="buttons unordered-list scrollbar-box">
   ...
      <ul class="overview" ng-transclude>
      </ul>
   ...
</section>
```

V první sekci šablony, která definuje záhlaví obrazovky, můžeme vidět vložení předaných parametrů `heading` a `back`. Druhá sekce definuje samotný obsah obrazovky. Za povšimnutí stojí atribut `ng-transclude` označující místo, na které má být vložen obsah uzavřený do tagu `<my-scroll-screen>`.

3.4.3 Použití direktivy

Níže uvedený kód demonstruje jednoduché použití dříve vytvořené direktivy. Výslednou obrazovku vytvořenou pomocí této direktivy ukazuje obrázek 5.11 na straně 45.

```
<my-scroll-screen heading="LANGUAGE" back="setup">
  <li class="both"><a ng-click="setLanguage('cs')">{{'CZECH'|
      translate}}</a></li>
  <li class="both"><a ng-click="setLanguage('en')">{{'ENGLISH'|
      translate}}</a></li>
  <li class="both"><a ng-click="setLanguage('de')">{{'GERMAN'|
      translate}}</a></li>
</my-scroll-screen>
```

3.5 Optimalizace dlouhých seznamů

Ať už se jedná o seznam telefonních kontaktů, seznam dostupných audio souborů či jakýkoliv jiný seznam v aplikaci, je nutné brát v potaz, že může obsahovat několik stovek až tisíc položek. To samozřejmě může mít špatný vliv na plynulost listování takovým seznamem, zvláště pak běží-li aplikace na palubním počítači automobilu, který může mít nižší výkon, než na jaký jsme zvyklí u stolních počítačů. Tato kapitola popisuje různé způsoby, jakými je možné docílit plynulého listování velmi dlouhých seznamů v aplikaci postavené na frameworku AngularJS.

3.5.1 Stránkování

Stránkování je základní možností, jak omezit velikost zobrazovaného seznamu a tím urychlit jeho zobrazení. Stránkování je ve webových aplikacích velmi rozšířené, příkladem budiž stránkování výsledků při vyhledávání na serveru Google.

AngularJS poskytuje filtr `limitTo:pageSize`, snižující počet zobrazovaných položek seznamu na hodnotu danou proměnnou `pageSize`. Samotného stránkování dosáhneme kombinací tohoto filtru s vlastním filtrem `startFrom`, který zajistí, aby zkrácený seznam začínal správným prvkem. [14]

```
angular.module('app').filter('startFrom', function() {
  return function(input, start) {
    return input.slice(start);
};
```

Pro docílení stránkování pak můžeme tyto filtry v HTML šabloně použít následujícím způsobem:

```
<tr ng-repeat="item␣in␣displayedItemsList␣|␣startFrom:␣currentPage␣*
    ␣pageSize␣|␣limitTo:pageSize" /tr>
```

3.5.2 Nekonečné rolování

Při nekonečném rolování (infinites scrolling, endless scrolling) dochází k automatickému načtení obsahu a jeho dynamickému připojení na konec stránky ve chvíli, kdy se uživatel blíží konci dosud načteného obsahu. Tento přístup je používán například na sociální síti Facebook. Pro snadnou implementaci nekonečného rolování v AngularJS aplikaci je možné použít knihovnu *ngInfiniteScroll*. Její použití je jednoduché:

```
<div infinite-scroll="myPagingFunction()" infinite-scroll-distance="
    3"></div>
```

Atribut `infinite-scroll` udává, která funkce má být zavolána, pokud se uživatel přiblíží ke konci stránky. Volitelným atributem `infinite-scroll-distance` je možné nastavit, jak blízko konci stránky se musí uživatel přiblížit před zavoláním výše uvedené funkce. Dalším volitelným atributem je `infinite-scroll-disabled`. Je-li jeho hodnota *true*, je nekonečné rolování dočasně zakázáno. [15]

3.5.3 Bindonce

AngularJS nabízí jednoduchou synchronizaci dat mezi modelem a view (pohledem) pomocí tzv. *Two-Way Data Binding*. Kdykoliv jsou data na kterémkoliv místě upravena, jsou automaticky aktualizována i na všech ostatních místech. To značně usnadňuje vývoj, ovšem může to přinášet i výkonnostní problémy, neboť hlídat, zda se některá z mnoha proměnných nezměnila, stojí odpovídající systémové prostředky.

Knihovna bindonce nabízí přístup, kdy jsou data z controleru načtena pouze jednou a dále se jejich změna nesleduje. To nemusí být vhodné vždy, ale pokud chceme daný list pouze vypsat a neočekáváme v něm žádné změny, může to značně zvýšit výkon aplikace. [16]

3.5.4 Direktiva ng-if

Další úsporu výkonu může přinést nahrazení direktivy *ng-show* direktivou *ng-if* v případech, kdy jsou dodatečné informace zobrazovány například až po kliknutí na danou položku. Direktiva *ng-if* (narozdíl od *ng-show*) zabraňuje renderování daného obsahu. Dané elementy jsou tak připojeny k Document Object Model (DOM) až ve chvíli, kdy jsou zobrazovány. Stejně tak je odloženo vyhodnocení případných vnořených provázání mezi modelem a view.

3.6 Yeoman generátor

Yeoman je nástroj, který umožňuje generovat kód podle předpřipravených šablon. S jeho pomocí je tak možné rychle vytvořit základ nové aplikace, pří-

padně přidávat nové části k již existujícímu projektu. Je možné využít některý z mnoha již existujících generátorů[7], nebo vytvořit generátor vlastní.

Yeoman je navržen tak, aby se vhodně doplňoval s nástroji Grunt[8] a Bower[9], nicméně tyto nástroje jsou vyvíjeny odděleně a je možné každý z nich použít i samostatně.

3.6.1 Tvorba generátoru

Yeoman je nejprve nutné nainstalovat příkazem `npm install --global yo`. Poté je třeba vytvořit složku, ve které bude generátor umístěn. Ta musí být pojmenována ve tvaru `generator-název`, kde *název* označuje jméno generátoru. V této složce buď ručně vytvoříme manifestační soubor `package.json`[10], nebo použijeme příkaz `npm init`.

Každý Yeoman generátor může obsahovat jeden hlavní a několik vedlejších generátorů. Hlavní generátor je vždy umístěn ve složce `app/`. Vedlejší generátory jsou umístěny ve vlastních složkách, pojmenovaných shodně s názvy vedlejších generátorů. Logika jednotlivých generátorů je pak vždy umístěna v souboru `index.js` v příslušné složce.

Nově vytvořený generátor je před spuštěním nutné registrovat do systému pomocí příkazu `npm link`. Samotné spuštění generátoru se provede příkazem `yo název_generátoru` v případě hlavního generátoru, resp. příkazem `yo název_generátoru:název_vedlejšího_generátoru` v případě generátoru vedlejšího.[19]

3.6.1.1 Interakce s uživatelem

Výstupy generátoru mohou být parametrizovány pomocí vstupu od uživatele. Ten může mít mnoho podob, například zadání libovolného textu, výběr jedné z nabídnutých hodnot apod.

Níže je uvedena ukázka interakce s uživatelem při generování základu pro nový překlad aplikace. Uživateli jsou položeny dvě otázky, přičemž jsou mu poskytnuty výchozí hodnoty, které může jednoduše potvrdit stisknutím klávesy enter. Zadané hodnoty jsou následně uloženy tak, aby byly přístupné i ostatním funkcím generátoru.

```
promting: function () {
  var done = this.async();

  var prompts = [{
```

[7] http://yeoman.io/generators/

[8] Grunt slouží k automatizaci kompilace CSS preprocesorů, spouštění testů, generování dokumentace, minifikaci souborů pro produkční nasazení a desítky další úkonů.

[9] Bower je balíčkovací systém, který usnadňuje instalaci knihoven a zásuvných modulů, spravuje jejich vzájemné závislosti a zjednodušuje aktualizace.

[10] Každý Yeoman generátor je v podstatě Node.js modul a musí tedy obsahovat manifestační soubor.

```
    name: 'languageName',
    message: 'What␣is␣your␣language\'s␣name?',
    default: 'slovak'
  }, {
     name: 'languageCode',
     message: 'What␣is␣your␣language\'s␣code?',
     default: 'sk'
   }];

  this.prompt(prompts, function (props) {
    this.languageName = props.languageName;
    this.languageCode = props.languageCode;
    done();
  }.bind(this));
}
```

3.6.1.2 Manipulace se soubory

Před samotnou manipulací se soubory je důležité nastavit výchozí zdrojovou a cílovou složku. K tomu slouží příkazy `this.sourceRoot('path')` a `this.destinationRoot('path')`.

Kopírování složky

V aplikaci je použita níže uvedená funkce, sloužící k duplikaci celé složky `sass/skin1`. Ta je rekurzivně zkopírována do nové složky, jejíž název závisí na vstupu od uživatele. Podtržítko před názvem funkce označuje, že tato funkce není spouštěna automaticky, ale spouští ji jiná funkce generátoru.

```
_copySass: function () {
  this.directory('sass/skin1', 'sass/skin' + this.styleName);
}
```

Zápis do existujícího souboru

O něco složitější situace nastává, chceme-li vkládat určitý obsah na konkrétní pozici v již existujícím souboru. V takovém případě je nutné si tuto pozici dopředu nějakým způsobem označit. K tomuto účelu se obvykle používá speciální unikátní komentář. Ten může vypadat například takto:

```
<!-------- yeoman hook language ======-->
<!-- NB! The above line is required for yeoman generator and should
    not be changed.-->
```

Při zápisu je pozice tohoto komentáře v souboru vyhledána a požadovaný kód je zapsán na toto místo.

3.6.2 Vytvořené generátory

V rámci této práce byly vytvořeny dva generátory. Jeden je určen pro tvorbu nových překladů aplikace a druhý pro tvorbu nových vzhledů. Lze očekávat, že obě tyto akce se při dalším vývoji aplikace budou mnohokrát opakovat. Adresářovou strukturu vytvořených generátorů zobrazuje obrázek 3.1.

```
generator-skoda ..... kořenová složka generátoru
├── package.json ..... manifestační soubor generátoru
├── node_modules/ ..... složka s Node.js moduly
├── style/
│   └── index.js ..... generátor nových vzhledů
└── translation/
    └── index.js ..... generátor nových překladů
```

Obrázek 3.1: Adresářová struktura vytvořeného Yeoman generátoru

3.6.2.1 Generátor nových překladů

Pokud bychom chtěli přidat nový překlad do aplikace ručně, museli bychom provést změny v následujících souborech:

- **javascripts/translations.js** Zkopírování existujícího překladu, úprava názvu jazyka, překlad textů.
- **javascripts/scripts-unify.js** Zaregistrování nového jazyka do služby zajišťující překlad.
- **templates/setup.html** Přidání nového jazyka do menu *nastavení - jazyk*.

Vytvořený generátor může tyto úlohy (kromě samotného překladu textů) vykonat za nás. Generování nového překladu můžeme spustit pomocí příkazu `yo skoda:language`. Po spuštění se nás generátor zeptá na název a kódové označení jazyka, který chceme přidat. Poté již generátor začne konat svoji práci a od uživatele pouze požaduje potvrzení, že má dojít k přepsání existujících souborů.

Jediné, co musí uživatel udělat ručně po skončení práce generátoru, je samotný překlad jednotlivých textů v souboru `javascripts/translations.js`. Použití tohoto generátoru je zobrazeno na obrázku 3.2.

3.6.2.2 Generátor nových vzhledů

Tento generátor usnadňuje vytvoření nového vzhledu aplikace. Po jeho spuštění je uživatel vyzván k zadání názvu nově vytvářeného stylu. Podle zadaného

```
C:\xampp\htdocs\skoda\client\generator-skoda>yo skoda:translation
New language generator
? What is your language's name? slovak
? What is your language's code? sk
 conflict templates\setup.html
? Overwrite templates\setup.html? overwrite
    force templates\setup.html
 conflict javascripts\scripts-unify.js
? Overwrite javascripts\scripts-unify.js? overwrite
    force javascripts\scripts-unify.js
 conflict javascripts\translations.js
? Overwrite javascripts\translations.js? overwrite
    force javascripts\translations.js
```

Obrázek 3.2: Yeoman generátor usnadňující lokalizaci do nového jazyka

názvu dojde k vytvoření nové složky, do které jsou zkopírovány potřebné soubory. Ty uživatel upraví podle potřeb. Obdobně jako v předchozím případě dojde také k automatickému přidání nové položky do menu *nastavení - vzhled*. Generátor se spouští příkazem `yo skoda:style`.

Kapitola 4

Realizace serverové části

Serverová část aplikace slouží k simulaci funkční jednotky automobilu. Oproti původní verzi aplikace obsahuje server grafické uživatelské rozhraní, umožňující zasílání různých signálů klientské části jednoduše stisknutím tlačítka. Díky tomu je možné testovat chování klientské části bez nutnosti napojení na skutečný automobil. Serverová část je nyní také napojena na databázi, dochází tak tedy k uchování dat i při vypnutí serveru.

4.1 Struktura serveru

```
images/ ..... složka s obrázky
javascripts/
    _config.js ..... nastavení serveru
    angular.js .... javascriptová knihovna definující strukturu projektu
    jquery-1.10.2.min.js ..... knihovna pro práci s javascriptem
    jquery-ui-1.11.3.js ..... knihovna grafických komponent
    scripts-server.js ..... controller serveru
    server-*.js ..... controllery jednotlivých částí serveru
node_modules/ ..... složka s Node.js moduly
sass/
    server.sass ..... zdroj souboru servers.css
stylesheets/
    server.css ..... kaskádové styly určující vzhled serveru
index.html ..... úvodní stránka serveru
package.json ..... Node-Webkit manifest
server.js ..... aplikační logika serveru
```

4.2 Node-Webkit

Node-Webkit (nově NW.js) je aplikační prostředí založené na Chromiu a platformě Node.js, které umožňuje vytvářet nativní aplikace v HTML a JavaScriptu. [6]

Každá Node-Webkit aplikace by měla obsahovat manifest uložený v souboru `package.json`. Tento manifest určuje, jak se má aplikace spustit a jak se bude chovat. [2] Manifest může vypadat například takto:

```
{
  "name": "skoda-server",
  "main": "index.html",
  "window": {
    "title": "Skoda␣Server",
    "toolbar": false
  }
}
```

Význam jednotlivých prvků je následující:

- **name:** Název balíčku. Měl by být globálně unikátní, neboť node-webkit do takto pojmenované složky ukládá aplikační data.
- **main:** Stránka, která má být spuštěna při startu aplikace.
- **window:** Určuje, jak bude vypadat hlavní okno aplikace.
- **title:** Popisek okna zobrazovaný v záhlaví aplikace. Užitečný zejména při startu aplikace, než dojde k načtení popisku z HTML tagu `<title>` úvodní stránky aplikace.
- **toolbar:** Určuje, zda má být zobrazen ovládací panel. Ten mimo jiné obsahuje adresní řádek.

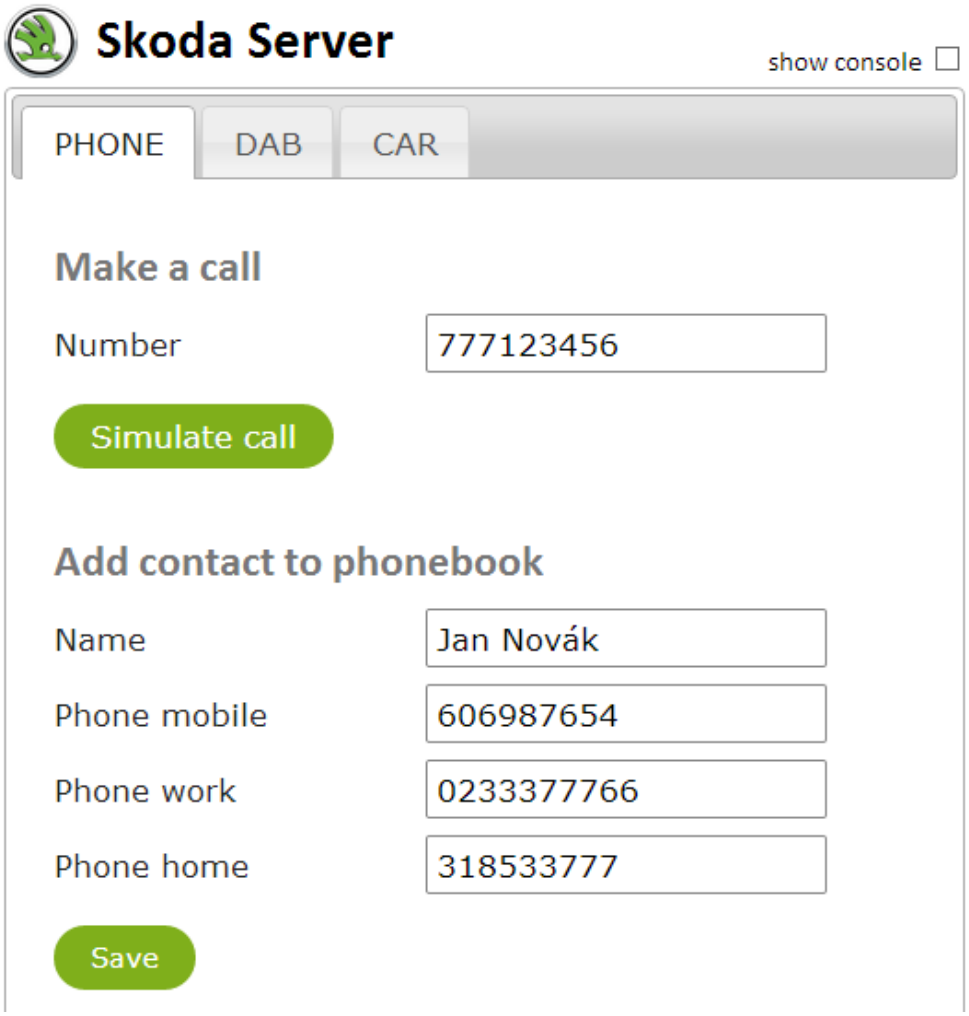
Pole *main* a *name* jsou povinná. Nastavení neumožňuje aplikaci automaticky spustit maximalizovanou. Maximalizování aplikace je dosaženo pomocí příkazů:

```
    var nwin = ngui.Window.get();
    nwin.maximize();
```

4.3 Kontext Telefon

Uživatelské rozhraní serveru pro telefon umožňuje přidávání telefonních kontaktů do databáze pomocí jednoduchého formuláře. Po uložení nového kontaktu je na klientskou část odeslán aktualizovaný telefonní seznam, čímž dojde k okamžitému projevení provedené změny v klientské části.

Další funkcí je simulace hovorů a to jak příchozích, tak odchozích. Chcemeli na klientské části simulovat příchozí hovor, vyplníme na serveru telefonní

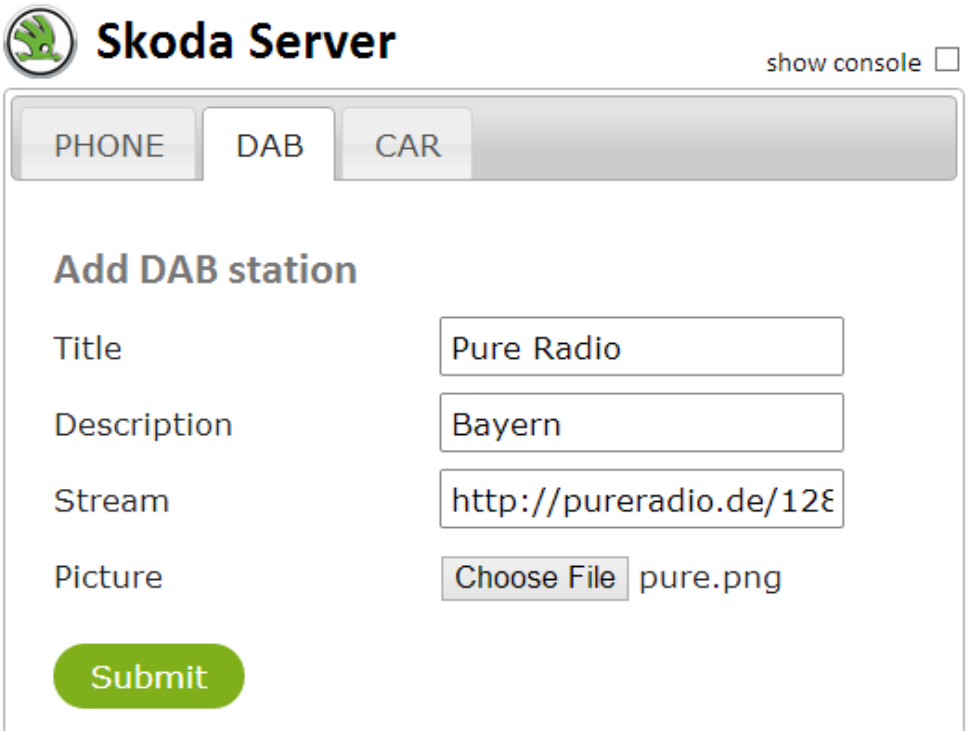


Obrázek 4.1: Uživatelské rozhraní serveru pro ovládání telefonu

číslo, ze kterého má být hovor proveden a stiskneme tlačítko *simulovat hovor*. Tím dojde k odeslání příkazu na klientskou část, která zobrazí modální okno s informací o příchozím hovoru a má možnost daný hovor přijmout či odmítnout. Aktivní hovor může být ukončen jak z klientské tak i ze serverové části. Rozhraní serveru slouží také k přijímání odchozích hovorů z klientské části.

4.4 Kontext DAB

Serverové rozhraní pro DAB slouží ke správě stanic digitálního rozhlasu. Ty je možné přidávat pomocí formuláře, který obsahuje pole pro název stanice, popisek, stream a obrázek stanice. Formulář pro ukládání stanic je zobrazen na obrázku 4.2.



Obrázek 4.2: Uživatelské rozhraní pro ukládání stanic digitálního rozhlasu

Pole *stream* slouží k uložení internetové adresy, ze které má být rádio přehráváno. Všechna tato data jsou po odeslání formuláře uložena do databáze a také odeslána na klientskou část, aby došlo k okamžitému projevení provedených změn.

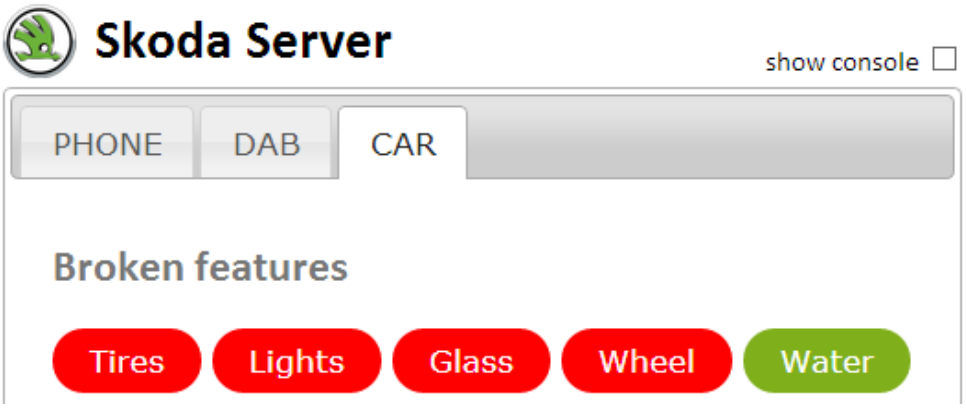
Obrázek stanice je ukládán a přenášen v kódování Base64. K nahrání obrázku a převedení do toho kódování je použito direktivy *baseSixtyFourInput*[11]. Ta je uložena v souboru `javascripts/angular-base64-upload.js`.

Server dále poskytuje informace o umělci a názvu právě přehrávané skladby. Jelikož tyto informace nelze odnikud automaticky načíst, je na serveru umožněno jejich ruční vyplnění a odeslání klientské části.

4.5 Kontext Info

Prostřednictvím uživatelského rozhraní serverové části pro kontext Info je možné vyslat zprávu o závadě na pneumatikách, světlech, čelním skle, volantu a také zprávu o nedostatku vody do ostřikovačů. Jsou tím simulovány zprávy, které jinak v automobilech přicházejí do řídící jednotky z různých čidel ve vozidle.

[11] https://github.com/adonespitogo/angular-base64-upload



Obrázek 4.3: Tlačítka nastavující které části vozu vyžadují kontrolu

Uživatelské rozhraní pro ovládání výše uvedené funkce je zobrazeno na obrázku 4.3. Zeleně podbarvená tlačítka zde označují části vozu, které jsou v pořádku, a červená značí, že daná část automobilu vyžaduje kontrolu. Při stisku tlačítka dojde k jeho přebarvení a vyslání patřičné zprávy na klientskou část. Odpovídající obrázek klientské části pro tento stav je zobrazen na obrázku 5.8 na straně 43.

Kromě zasílání zpráv o částech vozu, které vyžadují kontrolu, umožňuje tato část rozhraní také nastavovat informace o jízdě vozidla. Jedná se o délku jízdy, počet ujetých kilometrů, průměrnou rychlost a průměrnou spotřebu vozidla. Nastavit je také možné informaci o odhadovaném počtu kilometrů, které vozidlo ujede se současným stavem nádrže.

4.6 Komunikační protokol

Tato kapitola popisuje nové části komunikačního protokolu mezi klientskou částí a serverem. Symbolem „>“ je označena zpráva odeslaná z klienta na server, symbol „<“ značí odpověď serveru.

4.6.1 Telefon

- **Seznam telefonních kontaktů**

  Po otevření telefonního seznamu klient posílá žádost o seznam telefonních kontaktů. Server odpovídá zasláním celého telefonního seznamu.

```
> {type: "phone", action: "list"}
< {"type":"phone","action":"list","value":[
{"name":"Eugen␣Korda","phoneMobile":420776346092,"phoneWork"
    :24354789454,"phoneHome":31834534783},
{"name":"Alexander␣","phoneMobile":777547209},
{"name":"Jan␣Novak","phoneMobile":606408776,"phoneWork"
    :776123456}
```

```
]}
```

- **Seznam oblíbených telefonních kontaktů**
  Po spuštění aplikace klient žádá o seznam oblíbených telefonních čísel.

```
> {"type":"phone","action":"getFavs"}
< {"type":"phone","action":"favs","phoneFavouriteContacts":[
{"name":"Jan␣Novak","phone":24354789454,"phoneType":"work","
    position":0}
]}
```

- **Přidání oblíbeného kontaktu**
  Klient posílá žádost o přidání telefonního čísla na danou pozici v seznamu oblíbených kontaktů. Server odpovídá zasláním aktualizovaného seznamu oblíbených kontaktů.

```
> {"type":"phone","action":"addFav","name":"Jan␣Novak","phone"
    :777547209,"phoneType":"mobile","position":3}
< {"type":"phone","action":"favs","phoneFavouriteContacts":[
{"name":"Jan␣Novak","phone":24354789454,"phoneType":"work","
    position":0},
{"name":"Jan␣Novak","phone":777547209,"phoneType":"mobile","
    position":3}
]}
```

- **Odebrání oblíbeného kontaktu**
  Klient posílá žádost o odebrání telefonního čísla na dané pozici v seznamu oblíbených kontaktů. Server odpovídá zasláním aktualizovaného seznamu oblíbených kontaktů.

```
> {"type":"phone","action":"removeFav","index":3}
< {"type":"phone","action":"favs","phoneFavouriteContacts":[
{"name":"Jan␣Novak","phone":24354789454,"phoneType":"work","
    position":0}
]}
```

- **Zahájení hovoru**
  Strana, která zahajuje hovor[12], posílá příkaz *startCall*. V případě, že hovor zahajuje klient, položka *number* označuje, na které číslo je voláno. V zahájení hovoru serverem položka *number* značí, ze kterého čísla je na klient voláno. Druhá strana zprávu nepotvrzuje.

```
{"type":"phone","action":"startCall","number":777547209}
```

- **Přijmutí hovoru**
  Strana, která přijímá hovor, posílá příkaz *answerCall*. Druhá strana zprávu nepotvrzuje.

```
{"type":"phone","action":"answerCall"}
```

[12]Klient nebo server

- **Ukončení / odmítnutí hovoru**
  Strana, která ukončuje (nebo odmítá) hovor, posílá příkaz *endCall*. Druhá strana zprávu nepotvrzuje.

```
{"type":"phone","action":"endCall"}
```

4.6.2 Info

- **Statistiky využití vozidla** Po zahájení spojení a při každé změně zasílá server klientské části statistiky využití vozidla.

```
>{"type":"car","action":"init"}
<{"type":"car","action":"stats","value":{
    "name":"LONG_TERM","distance":"3␣536",
    "avgSpeed":54,"duration":"65:30","avgConsumption":"7,4"}}
```

- **Stav vozidla** Po zahájení spojení a při každé změně zasílá server klientské části seznam částí vozidla, které vyžadují kontrolu.

```
<{"type":"car","action":"brokenFeatures","value":[2,3,5]}
```

4.6.3 DAB

- **Následující rádio** Klient zasílá žádost o následující rádio v pořadí. Server posílá zpět odkaz na rádio, které má být přehráváno, včetně dodatečných informací jako popisku rádia či obrázku ve formátu base64.

```
>{"type":"dab","action":"nextRadio"}
<{"type":"dab","action":"play","value":{"title":"Evropa␣2","
    description":"Maximum␣muziky","stream":"http://icecast3.
    play.cz:80/evropa2-128.mp3","coverImage":{"filetype":"image
    /png","filename":"galaxy.png","filesize":9914,"base64":"
    iVBORw0KGgoAAAANSUhEUgAAAT..."}}}
```

- **Předchozí rádio** Obdobně jako v předchozím případě, pouze klient zasílá příkaz *prevRadio*.

- **Seznam DAB rádií** Klient žádá o seznam všech dostupných digitálních rádií. Server odpovídá zprávou obsahující tento seznam.

```
>{"type":"dab","action":"list"}
<{"type":"dab","action":"list","radiolist":[{"title":"Evropa␣2"
    ,"description":"Maximum␣muziky","stream":"http://icecast3.
    play.cz:80/evropa2-128.mp3","coverImage":{"filetype":"image
    /png","filename":"galaxy.png","filesize":9914,"base64":"
    iVBORw0KGgoAAAANSUhEUgAAAT..."}},...]}
```

4.6.4 Prohlížeč úvodních obrázků medií

- **Všechny obrázky** Klient žádá o seznam devíti skladeb, pro které následně načte a zobrazí úvodní obrázky. Server vrací tento seznam s ohledem na právě přehrávanou skladbu.

```
>{"type":"media","action":"getAllCovers"}
<{"type":"media","action":"allCovers","value":["media/Golden\
    _Red.mp3","media/LukHash␣-␣PIXELOVE.mp3","media/
    Stellardrone␣-␣Aurora.mp3","media/01␣Sign␣of␣the␣Times.mp3"
    ,"media/3.Alexander_Boyes-Februum.mp3","media/Golden_Red.
    mp3",...]}
```

- **Poslední obrázek** Při přechodu na následující skladbu není nutné načítat všech devět obrázků znovu (bylo by to zbytečně náročné na systémové zdroje). Místo toho stačí načíst jediný obrázek na poslední pozici, která se uvolní posunutím všech obrázků o jedna vlevo. Klient posílá žádost o název této skladby, aby poté mohl načíst daný obrázek. Server odpovídá vrácením názvu této skladby.

```
>{"type":"media","action":"getLastCover"}
<{"type":"media","action":"lastCover","value":"media/01-Black␣
    Eyed␣Peas␣-␣The␣Time␣(Dirty␣Bit).mp3"}
```

- **První obrázek** Obdobně jako v předchozím případě, pouze klient posílá příkaz *getFirstCover* a server odpovídá zprávou *firstCover*.

4.7 Konfigurace serveru

Konfigurační soubor `server/javascripts/_config.js` umožňuje uživateli provést některé základní změny v nastavení serveru.

```
var conf = {
  //zobrazeni konzole s vypisem komunikace
  console: true,
  //nazev databaze
  db_name: "skoda",
  //porty, na kterych bezi jednotlive servery
  port_media: "8000",
  port_tuner: "8001",
  port_phone: "8002",
  port_dab: "8003",
  port_car: "8004"
};
```

4.8 Nasazení serverové části

Kvůli přidání databáze pro ukládání některých záznamů a kvůli předělání serveru do podoby grafické aplikace došlo ke změně způsobu nasazení serverové

části. To nyní probíhá následovně:

1. Instalace databáze MongoDB[13]
2. Spuštění databáze (`C:/mongodb/mongod.exe`)
3. Instalace platformy Node.js z repozitářů na Internetu[14]
4. Zkopírování adresáře `src/impl/server` z přiloženého CD na server
5. Instalace komponenty Web Sockets příkazem `npm install ws`
6. Stažení platformy node-webkit [15]
7. Spuštění serverové části v prostředí node-webkit příkazem `C:/node-webkit/nw.exe C:/xampp/htdocs/skoda/server`, kde první část příkazu je cesta k node-webkit a druhá část příkazu je cesta k serveru

[13] Zdarma ke stažení na https://www.mongodb.org/downloads
[14] http://nodejs.org
[15] http://nwjs.io (nebo pomocí příkazu `npm install nw`

Kapitola 5

Realizace klientské části

5.1 Hlavní otočné menu

Jedním z úkolů této práce je předělání původního dlaždicového menu do formy otočného menu. Dlaždicové menu původní aplikace je zobrazeno na obrázku 1.1 na straně 4.

Jako základ pro vytvoření otočného menu byl použit plugin *jQuery Circular Carousel*[16]. Ten bylo nutné upravit tak, aby vzhledem i funkčností odpovídal našim požadavkům. Zejména tedy aby jednotlivé prvky nebyly rozmístěny po celém obvodu kružnice, ale pouze na její přední polovině. Dále bylo nutné zajistit, aby byly jednotlivé prvky tím menší, čím vzálenější jsou od středu. Ke zmenšení prvků je použito CSS transformace *scale*.

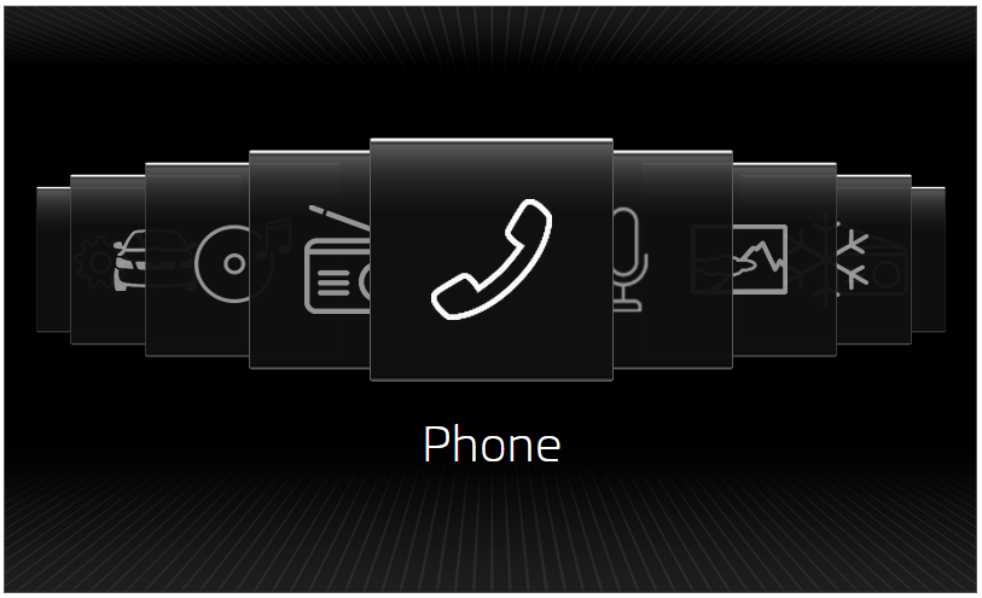
Níže je uveden úryvek HTML šablony vytvářející otočné menu:

```
<ul class="carousel">
  <li class="item␣active">
    <a class="nav-link␣button-phone" href="phone"></a>
    <span>{{ 'PHONE' | translate }}</span>
  </li>
  <li class="item">
    <a class="nav-link␣button-media" href="media"></a>
    <span>{{ 'MEDIA' | translate }}</span>
  </li>
  <li class="item">
    <a class="nav-link␣button-tuner" href="tuner"></a>
    <span>{{ 'TUNER' | translate }}</span>
  </li>
  ...
</ul>
```

Otočné menu neobsahuje žádné ovládací prvky, otáčení je ovládáno výhradně pomocí „swipe gest“. Uživatel přidrží prst na dotykovém displeji a tažením vlevo či vpravo dojde k otočení menu v požadovaném směru.

Zavoláním příkazu

[16] https://github.com/samuelgbrown/jquery.circular-carousel



Obrázek 5.1: Hlavní otočené menu

```
carousel = $('.carousel').CircularCarousel(options);,
```

kde `options` označuje možná nastavení, dojde k inicializaci menu a počátečnímu rozmístění jednotlivých prvků.

Soubor `scripts-home.js` poskytuje následující nastavení otočného menu:

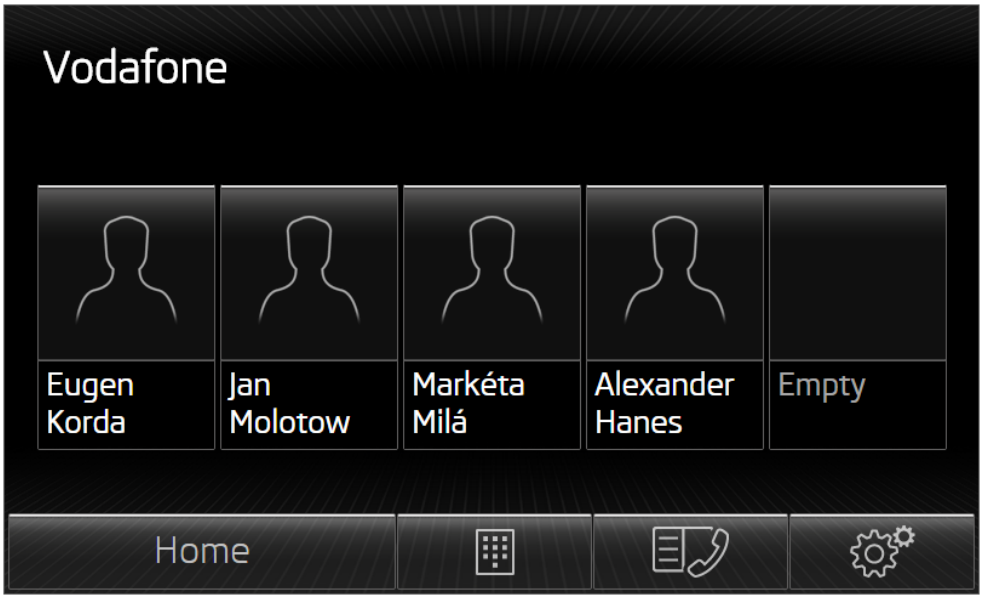
- **width:** Šířka otočného menu,
- **offsetX:** Posunutí menu v ose X,
- **offsetY:** Posunutí menu v ose Y,
- **activeItem:** Index prvku, který má být aktivní po spuštění aplikace,
- **duration:** Doba, kterou trvá pootočení menu z jednoho prvku na druhý (v ms),
- **className:** Název třídy, kterou jsou označeny jednotlivé prvky menu.

Logika otočného menu je umístěna v souboru `jquery.circular-carousel.js`. Výsledná podoba otočného menu je zobrazena na obrázku 5.1.

5.2 Telefon

Držení mobilního telefonu za jízdy je v České republice zakázáno. Telefonovat je ovšem povoleno, a to za pomoci tzv. „hands free“.

Moderní automobily často umožňují spárování s mobilním telefonem řidiče pomocí technologie Bluetooth. Po spárování je možné ovládat některé funkce



Obrázek 5.2: Oblíbené telefonní kontakty

telefonu pomocí dotykového displeje palubního počítače automobilu či ovládacích prvků blízko volantu. Díky tomu se řidič může lépe soustředit na řízení a nedochází ke zbytečnému odvádění jeho pozornosti.

5.2.1 Oblíbené kontakty

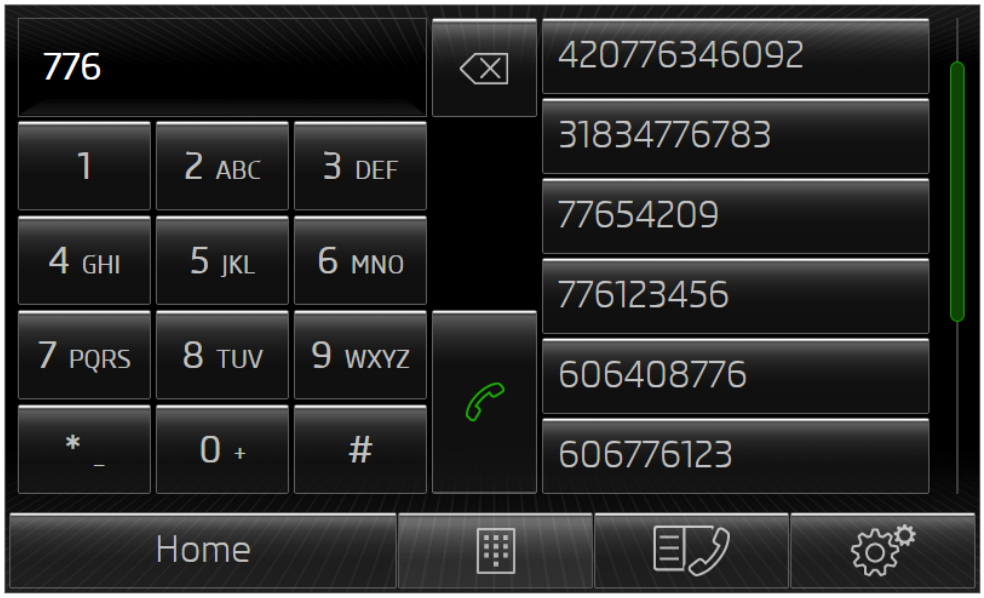
Oblíbené kontakty poskytují rychlý přístup k pěti nejpoužívanějším telefonním číslům. Kliknutím na oblíbený kontakt je ihned zahájena simulace volání na dané číslo viz kapitola 5.2.4.

5.2.1.1 Úprava oblíbených kontaktů

Úprava oblíbených kontaktů probíhá z menu *telefon - nastavení - upravit oblíbené*. Kliknutím na ikonku koše v pravé horní části kontaktu je na server odeslán požadavek o smazání kontaktu z oblíbených. Server odstraní daný záznam z databáze a pošle zpět aktualizovaný seznam oblíbených kontaktů.

5.2.2 Klávesnice

Pomocí virtuální klávesnice je možné zadat libovolné telefonní číslo a následně simulovat odchozí hovor na toto číslo. Během zadávání vstupu dochází také k filtrování kontaktů z telefonního seznamu. Čísla ze seznamu, která odpovídají právě zadanému vstupu, jsou zobrazována v pravé části obrazovky. Vyhledávání kontaktů pomocí vstupu z virtuální klávesnice názorně ukazuje obrázek 5.3.



Obrázek 5.3: Vyhledávání v telefonních kontaktech pomocí vstupu z virtuální klávesnice

5.2.3 Seznam telefonních kontaktů

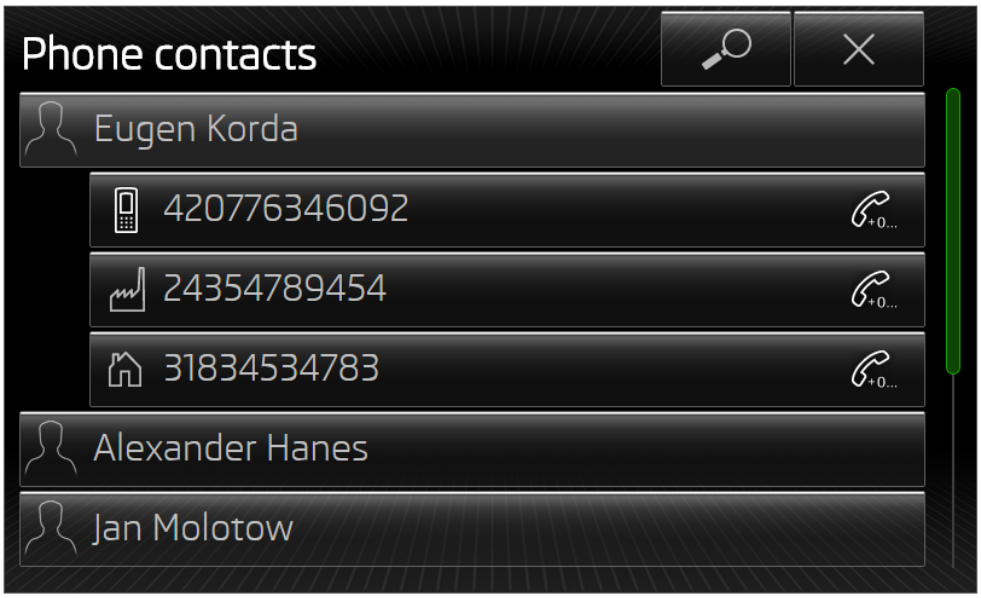
Telefonní kontakty jsou zobrazeny formou seznamu, který zpočátku zobrazuje jen jména kontaktů. Po kliknutí na jméno dojde k zobrazení / skrytí dostupných telefonních čísel daného kontaktu. U jednotlivých čísel je graficky znázorněno, o jaký typ telefonního čísla se jedná. Kliknutím na položku s číslem dojde k zahájení simulace hovoru, viz kapitola 5.2.4.

5.2.4 Odchozí hovor

Simulace odchozího hovoru může být zahájena kliknutím na kontakt v seznamu oblíbených, zadáním telefonního čísla pomocí virtuální klávesnice či vyhledáním kontaktu v telefonním seznamu. V případě, že je právě přehrávána hudba či rádio, dojde před začátkem hovoru k jejímu pozastavení. Informace o stavu hovoru jsou zobrazeny ve vyskakovacím okně, jak ukazuje obrázek 5.5. Po ukončení hovoru dojde k obnovení přehrávání hudby / rádia.

5.2.5 Příchozí hovor

Ve chvíli, kdy klient přijme od serveru zprávu o příchozím hovoru, je aktuální obsah aplikace, ať už uživatel dělá cokoliv, překryt vyskakovacím oknem s informacemi o volajícím. Stejně jako v případě odchozího hovoru dojde k přerušení přehrávání hudby / rádia.



Obrázek 5.4: Seznam telefonních kontaktů

5.3 DAB radio

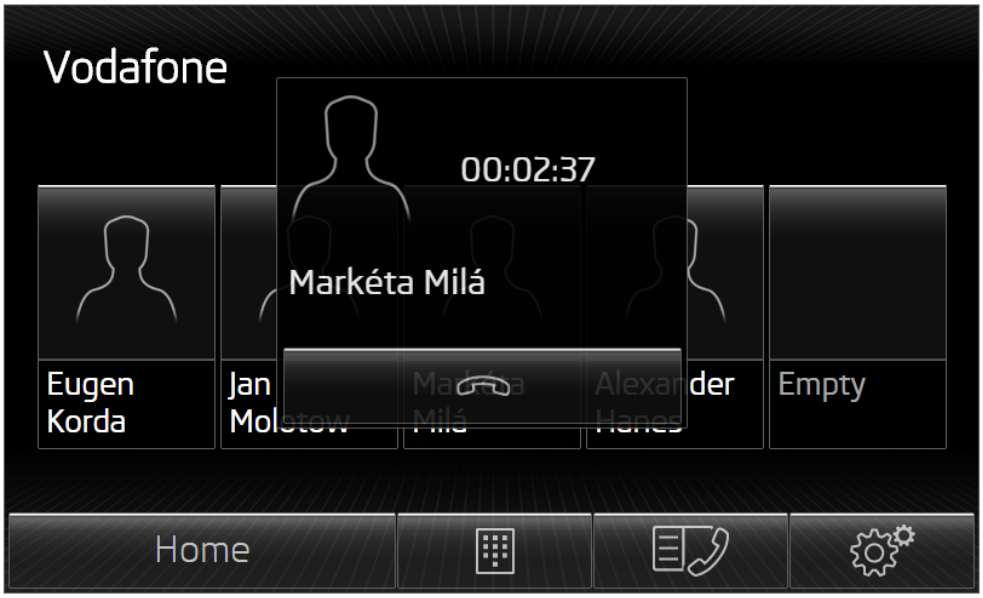
DAB je digitální rozhlasové vysílání, které by v budoucnu mohlo nahradit stávající VKV-FM vysílání. Mezi výhody DAB patří například kvalitnější zvuk, větší dosah, kvalitní příjem i v členitých oblastech a možnost přenosu různých doplňkových dat jako jsou texty či obrázky. [13]

Tato aplikace simuluje přehrávání digitálního rozhlasu pomocí přehrávání internetových rádií. Informace o názvech rádií a adresách, na kterých vysílají, jsou načítány ze serverové části aplikace. Jak je vidět na obrázku 5.6, server zasílá též doplňující data jako logo rádia či jméno interpreta a název přehrávané skladby.

5.4 Info

Tato část uživatelského rozhraní slouží k zobrazování informací o automobilu a je rozdělena na dvě základní části. První část, zobrazená na obrázku 5.7, zahrnuje statistiky o počtu najetých kilometrů, průměrné rychlosti, době jízdy a průměrné spotřebě vozidla. Je možno zobrazit statistiky dlouhodobé, od posledního načerpání paliva nebo od nastartování. Zobrazena je také odhadovaná dojezdová vzdálenost vozidla vzhledem k aktuálnímu množství paliva v nádrži a průměrné spotřebě.

Druhá část informuje o stavu stavu jednotlivých dílů automobilu. V případě, že jsou všechny díly v pořádku, je zobrazen obrázek bílého automobilu Škoda. Pokud je zjištěna závada na některém dílu, obrázek automobilu ze-



Obrázek 5.5: Modální okno aktivního hovoru

šedne, zprůhlední a daný díl je v obrázku vozidla červeně označen. Obrázek 5.8 zobrazuje stav, kdy je zjištěna závada na pneumatikách, světlech, volantu a čelním skle.

Zvýrazňování jednotlivých dílů auta je realizováno tak, že přes obrázek automobilu jsou na přesné pozici zobrazovány průhledné obrázky daných dílů.

5.5 Prohlížeč úvodních obrázků médií

Jak již bylo zmíněno dříve, jednou z částí původní aplikace je přehrávač médií. Ten umožňuje procházení složek s hudbou, přehrávání písniček ve formátech mp3 a ogg a zobrazení obrázku právě přehrávané skladby.

Tato práce rozšiřuje přehrávač o funkci prohlížení úvodních obrázků. Ta funguje tak, že uprostřed obrazovky je zobrazen obrázek aktuálně přehrávané skladby a po stranách jsou na každé straně vidět další tři obrázky. Ty odpovídají třem předchozím a třem následujícím skladbám a jsou částečně natočené směrem ke středu. Při přechodu na další písničku [17] dojde k animovanému posunutí všech obrázků o jednu pozici vlevo.

Pro realizaci animovaného posunutí je využito pluginu jQuery Circular Carousel stejně jako v případě hlavního otočného menu, viz. kapitola 5.1 na straně 35. V tomto případě však dochází s každým přepnutím skladby ke změně animovaného obsahu. Z výkonnostních důvodů není možné načítat obrázky všech skladeb ve složce najednou, načítá se tedy vždy jen to, co je nutné.

[17] ať už stiskem tlačítka, gestem či automaticky po dohrání aktuální skladby

Obrázek 5.6: Digitální rozhlas (DAB)

Při inicializaci se jedná o sedm právě zobrazovaných obrázků plus jeden na každém konci seznamu. Tyto dva krajní obrázky jsou skryté a slouží k tomu, aby mohly být rychle zobrazeny ve chvíli, kdy dojde k posunu na další / předchozí skladbu. V té chvíli jsou všechny obrázky o jednu pozici posunuty a na poslední (skrytou) pozici je načten nový obrázek. V případě, že hudební soubor neobsahuje obrázek alba, je zobrazen obecný obrázek jako u třetí skladby zleva na obrázku 5.9.

K částečnému natočení obrázků směrem ke středu je využito následující CSS transformace:

```
transform: perspective(800px) rotateY(40deg);
          transform-origin: 0% 100%;
```

5.6 Změna vzhledu

Pomocí menu `nastavení - vzhled` může uživatel zvolit požadovaný vzhled aplikace. Jedná se o funkci, která především demonstruje jednoduchost takové změny při použití webových technologií.

V kódu tuto změnu zajišťuje následující jednoduchá funkce:

```
$scope.changeStyle = function (style) {
  $("link").attr("href", "stylesheets/skin" + style + "/screen-skin.
      css");
};
```

Obrázek 5.7: Statistika využití vozidla od nastartování

Tato funkce pomocí knihovny jQuery vyhledá ve zdrojovém kódu stránky tag *link* a obsah jeho atributu *href* nahradí cestou k požadovanému vzhledu aplikace. Webový prohlížeč ihned po této změně aplikaci sám automaticky překreslí do nově nastaveného vzhledu.

Obrázek 5.10 ukazuje hlavní otočné menu po nastavení alternativního vzhledu aplikace. K usnadnění tvorby nových vzhledů je možné použít Yeoman generátor popsaný v kapitole 3.6.2 na straně 23.

5.7 Lokalizace

Klientská část aplikace umožňuje zvolit jazyk, ve kterém mají být zobrazeny všechny texty v aplikaci. V současnosti je na výběr mezi angličtinou, němčinou a češtinou, přičemž je snadné doplnit další jazyky.

Volba jazyka probíhá z menu *nastavení* - *jazyk*, zobrazeném na obrázku 5.11.

Ihned po kliknutí na jeden z nabízených jazyků dojde v celé aplikaci k přeložení všech zobrazovaných textů do požadovaného jazyka.

Pro lokalizaci je nezbytné vložení souboru `angular-translate.min.js` a inicializace pomocí následujícího kódu:

```
var app = angular.module('app', ['pascalprecht.translate'],
  ['$translateProvider', function ($translateProvider) {
  $translateProvider
          .translations('en', english)
          .translations('de', german)
```

Obrázek 5.8: Grafické znázornění stavu vozidla

```
          .translations('cs', czech);
  $translateProvider.preferredLanguage('en');
}]);
```

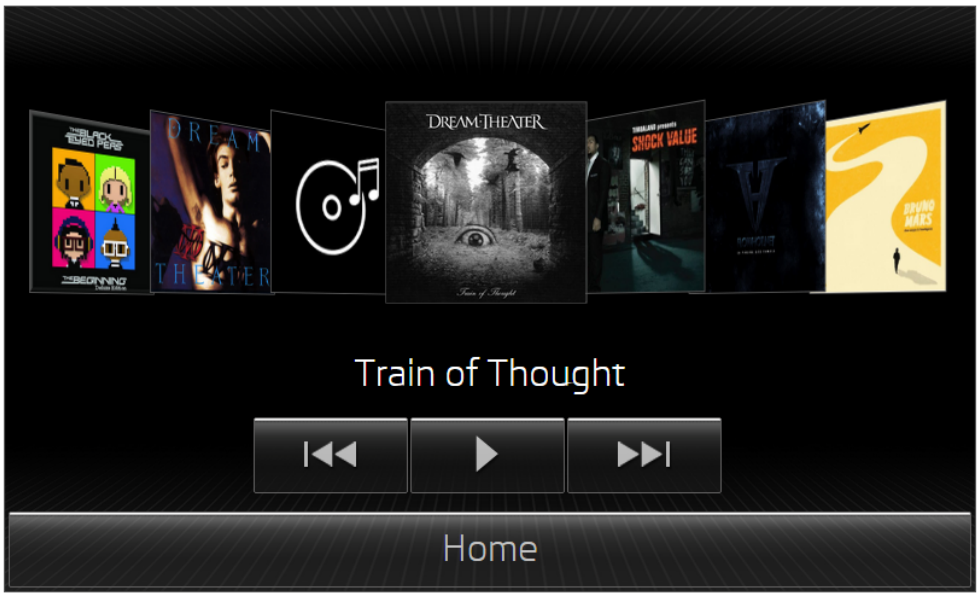
Příkazy *translations* nastavují překlady pro konkrétní jazyky, přičemž první argument je označení daného jazyka a druhý argument jsou samotné překlady. Ty jsou zde načítány ze souboru `translations.js`, kde jsou uvedeny ve tvaru *klíč : hodnota*. Překlad hlavního menu do českého jazyka tedy může vypadat třeba následovně:

```
var czech = {
  // Home screen
  MEDIA: 'Hudba',
  PHONE: 'Telefon',
  TUNER: 'Radio',
  VOICE: 'Hlas',
  CAR: 'Auto',
  ...
};
```

Příkazem `$translateProvider.preferredLanguage('en');` je nastaven výchozí jazyk, v tomto případě angličtina.

Samotné použití v HTML šabloně je již velmi jednoduché. Stačí daný řetězec uzavřít do dvojitých složených závorek a použít filtr *translate*:

```
<span>{{ 'MEDIA' | translate }}</span>.
```



Obrázek 5.9: Prohlížeč úvodních obrázků médií

Chybné zobrazení českých znaků

Při nastavení jazyka aplikace na češtinu dochází k chybnému zobrazování písmen s háčky. To je způsobeno tím, že používané fonty `skodapro_medium` a `skodapro_regular` dodané společností Škoda tyto znaky neobsahují. Pro tyto znaky je tak automaticky použit výchozí font prohlížeče. Řešením je použít font, který tyto české znaky obsahuje. To je demonstrováno v alternativním stylu 2, kde je použit font `Roboto`, který české znaky obsahuje.

5.8 Nasazení klientské části

Ve způsobu nasazení klientské části aplikace nedošlo oproti původní verzi aplikace k žádným změnám a probíhá tedy tak, jak je popsáno v práci Ing. Václavíka [1] v kapitole 4.9.2 na straně 82.

Obrázek 5.10: Alternativní vzhled hlavního otočného menu

Obrázek 5.11: Menu nastavení jazyka

Kapitola 6

Testování

Testování je velmi důležitou součástí vývoje každého softwaru. Čím dříve jsou chyby odhaleny, tím menší je jejich dopad. Javascript patří mezi dynamicky typované jazyky a je tak snazší udělat některé chyby, na které by u jiných jazyků upozornilo vývojové prostředí nebo kompilátor. Toho jsou si dobře vědomi vývojáři Angularu, a proto jej navrhli tak, aby bylo možné výsledné aplikace snadno testovat. [10]

Testování může probíhat buď na úrovni jednotlivých funkcí (Unit testy) nebo na úrovni celé aplikace (End-to-end testy). Oba typy testů byly v aplikaci realizovány pomocí testovacího frameworku Jasmine[18]. Otestovány byly jak nově vytvořené, tak i dříve existující části aplikace.

Struktura testů

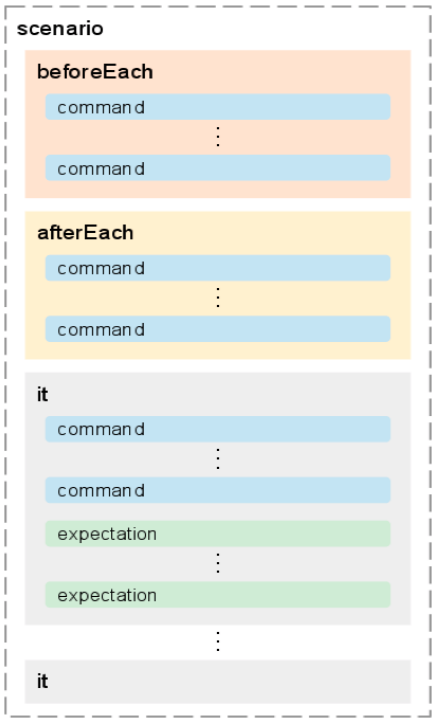
Struktura testů je zobrazena na obrázku 6.1 a je shodná pro oba druhy testování. Testovací soubory obsahují jeden nebo více „it“ bloků, které popisují požadavky na aplikaci. Každý „it“ blok je složen z příkazů (*commands*) a očekávání (*expectations*). Příkazy říkají, jaké akce mají být provedeny (např. stisk tlačítka) a očekávání ověřují správný stav aplikace po provedení příkazů (například hodnotu určitého pole). Dojde-li k selhání kteréhokoliv očekávání, je celý blok označen jako neúspěšný a testování pokračuje dalším blokem.

Testovací soubory mohou taktéž obsahovat bloky funkcí, které jsou spuštěny před (*beforeEach*) či po (*afterEach*) každém „it“ bloku. Tyto bloky jsou spouštěny vždy, bez ohledu na výsledek testu.

6.1 Unit testy

Unit testy (jednotkové testy) ověřují správné fungování jednotlivých částí (jednotek) zdrojového kódu. Těmito jednotkami jsou obvykle samostatné funkce

[18] https://github.com/jasmine/jasmine



Obrázek 6.1: Struktura testovacího scénáře [12]

či třídy. Jednotlivé testy by na sobě měly být vzájemně nezávislé.

6.1.1 Karma

Spouštění unit testů probíhá pomocí nástroje Karma [19]. Karma je jednoduchý nástroj umožňující automatizované spouštění javascriptových testů ve skutečných prohlížečích [20]. Pro napsání testů, které Karma spouští, si je možné vybrat z mnoha různých testovacích frameworků (mimo jiných například Jasmine, Mocha nebo QUnit). [4]

6.1.1.1 Instalace Karmy

Instalace Karmy se spustí příkazem

```
npm install karma –save-dev,
```

a pomocí příkazu

```
npm install karma-jasmine karma-chrome-launcher
                    –save-dev
```

[19] Dříve známá pod názvem Testacular.
[20] Včetně prohlížečů v tabletech a telefonech.

dojde k instalaci potřebných zásuvných modulů.

6.1.1.2 Konfigurace Karmy

Konfigurační soubor může být vytvořen manuálně nebo automaticky příkazem `karma init my.conf.js`.

V případě této aplikace je konfigurační soubor Karmy umístěn v souboru `javascripts/tests/karma.conf.js` a jeho (zkrácený) obsah je následující:

```
module.exports = function (config) {
  config.set({
    basePath: 'javascripts/',
    frameworks: ['jasmine'],
    files: [
      'angular-full.js',
      'angular-mocks.js',
      'angular-translate.min.js',
      'jquery-1.10.2.js',
      '_config.js',
      'translations.js',
      'scripts-unify.js',
       ...
      'tests/*.js'
    ],
    exclude: [
    ],
    preprocessors: {
    },
    reporters: ['progress'],
    port: 9876,
    colors: true,
    logLevel: config.LOG_INFO,
    autoWatch: true,
    browsers: ['Chrome'],
    singleRun: false
  });
};
```

Význam nejdůležitějších položek konfigurace:

- **frameworks:** Seznam použitých frameworků.
- **files:** Seznam souborů, které mají být načteny prohlížečem[21].
- **exclude:** Seznam souborů, které mají být z testování vyloučeny.
- **autoWatch:** Určuje, zda mají být testy automaticky opakovány při změně libovolného ze zahrnutých souborů.

[21] Pořadí souborů je důležité. Při použití vzoru (např. `*.js`), kterému odpovídá více souborů, jsou tyto soubory vloženy v abecedním pořadí. Každý soubor je vložen právě jednou a to na místě prvního výskytu.[11]

- **browsers:** Seznam webových prohlížečů, ve kterých mají být testy spuštěny.

6.1.2 Příklad unit testu

Následující test slouží k ověření správného fungování souboru `javascripts/scripts-unify.js`.

```
describe('Unit: UnifyCtrl', function () {
  // Load the module with MainController
  beforeEach(module('app'));

  var ctrl, scope;
  // inject the $controller and $rootScope services
  // in the beforeEach block
  beforeEach(inject(function ($controller, $rootScope) {
    // Create a new scope that's a child of the $rootScope
    scope = $rootScope.$new();
    // Create the controller
    ctrl = $controller('UnifyCtrl', {
      $scope: scope
    });
  }));

  it("should have a controller", function () {
    expect(ctrl).toBeDefined();
  });

  it('should load all widgets specified in config', function () {
    expect(scope.displayedWidgets.length).toBe(conf.modules.length);
  });

  it('should load the landing page', function () {
    expect(scope.displayedWidgets).toContain(conf.templates_dir +
        conf.landing_page + ".html");
  });
});
```

Tento test ověřuje:

- přítomnost controlleru,
- načtení všech rozhraní definovaných v konfiguraci,
- načtení úvodní stránky.

6.1.3 Spuštění unit testů

Spuštění unit testů probíhá příkazem `karma start` ze složky, ve které je umístěn konfigurační soubor Karmy (tj. `javascripts/tests`). Tím dojde k nastartování předdefinovaného prohlížeče (Chrome), ve kterém vzápětí proběhnou testy uvedené v konfiguračním souboru. Výsledek testů je vypsán do konzole a může vypadat přibližně takto:

```
INFO [karma]: Karma v0.12.31 server started at http://localhost
    :9876/
INFO [launcher]: Starting browser Chrome
INFO [Chrome 42.0.2311 (Windows 8.1)]: Connected on socket
    rMd0EaX1AtIlcW8frEZo with id 94461931
Chrome 42.0.2311 (Windows 8.1): Executed 7 of 7 SUCCESS (0.179 secs
    / 0.155 secs )
```

Podle nastavení budou testy opakovány při každé změně testovaných souborů. Necháme-li tedy okno prohlížeče s testy otevřené během vývoje aplikace, můžeme po každém uložení práce ihned vidět počet splněných / nesplněných testů.

6.2 End-to-end testy

S narůstajícím množstvím kódu je nemyslitelné testovat správné fungování všech nových komponent manuálně. Na rozdíl od unit testů, end-to-end testy ověřují správnou spolupráci mezi jednotlivými komponentami a funkčnost aplikace jako celku, včetně například komunikace se serverem nebo databází. [12]

6.2.1 Protractor

Spouštění end-to-end testů probíhá pomocí nástroje Protractor. Ten umožňuje automaticky spustit webový prohlížeč, přejít na zadanou adresu a na ní simulovat chování uživatele například klikáním na tlačítka či zadáváním textového vstupu. Následně pak ověřuje, zda se stránka nachází v požadovaném stavu - tj. obsahuje správné elementy, texty patřičných hodnot, pořadí a podobně. Pro své testy používá Protractor syntaxi frameworku Jasmine.

6.2.1.1 Instalace Protractoru

Protractor se nainstaluje příkazem

```
npm install -g protractor
```

a pomocí příkazů níže dojde ke stažení nezbytných souborů a nastartování Selenium serveru[22] Ten slouží k ovládání prohlížeče, ve kterém běží testy a musí běžet po celou dobu end-to-end testování.

```
webdriver-manager update
webdriver-manager start
```

[22]K jeho spuštění je potřeba mít nainstalováno Java Development Kit (JDK).

6.2.1.2 Konfigurace Protractoru

Konfigurace Protractoru je uložena v souboru `javascripts/tests/protractor.conf.js` a obsahuje následující volby:

```
exports.config = {
  seleniumAddress: 'http://localhost:4444/wd/hub',
  specs: ['e2e/*.js'],
  capabilities: {
    browserName: 'chrome'
  },
  jasmineNodeOpts: {
    showColors: true,
    defaultTimeoutInterval: 50000,
    isVerbose: true,
    includeStackTrace: true
  }
};
```

Význam nejdůležitějších položek nastavení:

- **seleniumAddress:** Adresa, na které běží Selenium server.
- **specs:** Seznam souborů, které obsahují testy.
- **browserName:** Webový prohlížeč, ve kterém mají být testy spuštěny.
- **jasmineNodeOpts:** Nastavení předávané Jasmine frameworku.
- **defaultTimeoutInterval:** Maximální doba trvání testu (v ms).

6.2.2 Spuštění end-to-end testů

Pro spuštění testů je nutné, aby běžel selenium server (příkaz `webdriver-manager start`, viz výše). Samotné testování se pak spustí příkazem `protractor protractor.conf.js` ze složky `client/javascripts/tests/`.

6.2.3 Příklad end-to-end testu

Níže uvedený test kontroluje správné fungování funkce přidání oblíbeného telefonního kontaktu.

```
describe('Phone', function () {
  var favButtons = element(by.id('screen-phone')).all(by.repeater('
      fav in favContacts'));
  var favButtonsEdit = element(by.id('screen-edit-favorites')).all(
      by.repeater('fav in favContacts'));

  beforeEach(function () {
    browser.get('http://localhost/skoda/client/unify.html');
    $('.button-phone').click();
  });

  it('should add fav contact', function () {
```

```
    expect(favButtons.count()).toEqual(5);

    $('a[href="phone-settings"]').click();
    $('a[href="edit-favorites"]').click();

    expect(favButtonsEdit.count()).toEqual(5);

    var emptyFavs = element.all(by.css('.edit-favorites␣.fav-phone-
        buttons␣a.empty'));
    expect(emptyFavs.count()).toEqual(5);

    //expect the last fav to be empty for this test
    expect(emptyFavs.last().isDisplayed()).toBeTruthy();

    emptyFavs.last().click();

    element.all(by.css('.add-fav␣.contact-name')).first().click();
    element.all(by.css('.add-fav␣.phone-mobile')).first().click();

    expect(emptyFavs.last().isDisplayed()).toBeFalsy();
  });
});
```

Na začátku testu jsou definovány elementy *favButtons* a *favButtonsEdit*[23] tak, aby je bylo možné opakovaně použít ve všech testech. Funkce *beforeEach* je automaticky spuštěna před každým jednotlivým testem. V tomto případě otevře v testovacím prohlížeči výchozí stránku aplikace a simuluje kliknutí na tlačítko telefonu v hlavním menu. Následně je zkontrolováno, zda se zobrazuje pět polí pro oblíbené telefonní kontakty [24]. Poté je otevřeno menu *Nastavení telefonu* a *Úprava oblíbených kontaktů*. Zkontroluje se, zda polí pro oblíbené kontakty je správný počet a zda poslední pole je volné. To je nutný předpoklad tohoto testu, v opačném případě by nový kontakt nebylo kam uložit a test by nemohl proběhnout. Pokud je toto splněno, je simulováno kliknutí na tlačítko pro přidání nového kontaktu. Tím dojde k zobrazení seznamu telefonních kontaktů. Mobilní číslo prvního člověka v seznamu je vybráno a uloženo do oblíbených. Na závěr testu je zkontrolováno, zda byl kontakt opravdu přidán a dané pole tedy již není prázdné. K odstranění tohoto oblíbeného kontaktu dojde při následujícím testu, který ověřuje správné fungování funkce mazání kontaktů z oblíbených.

6.2.4 Vytvořené testy

Obdobně jako výše uvedený test přidání telefonního kontaktu do oblíbených byly vytvořeny další testy pro ověření správného fungování všech částí aplikace. Jedná se o:

- Smazání kontaktu z oblíbených

[23] Reprezentují pole pro oblíbené kontakty, resp. jejich úpravu.
[24] Nezáleží na tom, jestli jsou prázdné nebo ne.

- Vytočení telefonního čísla pomocí virtuální klávesnice
- Změna jazyka aplikace
- Změna vzhledu aplikace
- Přehrávání rádia
- Přehrávání digitálního rádia
- Přehrání skladby v rozhraní Média
- Načtení všech modulů aplikace

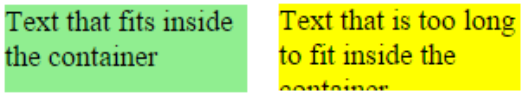
6.3 Další možnosti testování

Kromě výše uvedených testů, které ověřují především správnou reakci klientské části na příkazy zaslané serverem a kontrolují přítomnost patřičných prvků na stránce, by bylo také možné kontrolovat správné umístění těchto prvků. Zejména pak to, zda v případě změny jazyka aplikace nedojde k tomu, že některý přeložený text by byl příliš dlouhý a nevešel by se do určené oblasti. Tato kontrola probíhá v současné době ve společnosti Škoda ručně, což je náročné především kvůli množství dostupných jazyků, počtu stavů, ve kterých se aplikace může nacházet a častým změnám.

Při použití webových technologií by toto bylo možné kontrolovat následujícím způsobem. Mějme ohraničující box s id *element*, který má pevně dané rozměry. Uvnitř se nachází text, u kterého chceme zjistit, zda se do ohraničujícího boxu vejde.

```
<html>
  <head>
    <style>
      #element{
        width:150px;
        height:40px;
        overflow:hidden;
      }
    </style>
  </head>
  <body>
    <div id="element">
      <span>Text that fits inside the container</span>
    </div>
  </body>
</html>
```

Následující kód ověří, zda nedochází k „přetečení“ textu mimo ohraničující element a to jak horizontálně, tak vertikálně. Pro větší zřetelnost je elementu nastaveno zelené pozadí v případě, že k přetečení nedošlo a žluté, pokud text přetekl mimo danou hranici.



(a) Bez přetečení (b) S přetečením

Obrázek 6.2: Detekce přetečení textu mimo ohraničující element

```
var element = document.querySelector('#element');
if((element.offsetHeight < element.scrollHeight) ||
 (element.offsetWidth < element.scrollWidth)){
   //element with overflow
  element.style.background = "yellow";
}
else{
  //element without overflow
  element.style.background = "lightgreen";
}
```

Změní-li se obsah textu na delší (například změnou jazyka, zvýšením číselné hodnoty apod.), dojde k detekci přetečení a změně barvy pozadí na žlutou jak ukazuje obrázek 6.2b.

```
<div id="element">
  <span>Text that is too long to fit inside the container</span>
</div>
```

Závěr

Tato práce rozšířila klientskou část původní aplikace o nová rozhraní Telefon, Auto a DAB radio. Aplikace byla lokalizována do angličtiny, němčiny a češtiny. Navrženo a implementováno bylo také grafické rozhraní serveru simulujícího funkční jednotku automobilu. Server nyní ukládá důležitá data do databáze, je tedy možná jejich jednoduchá editace a jsou uchována i po restartování. Výsledný kód byl otestován jak na úrovni jednotlivých funkcí, tak na úrovni celé aplikace. Analyzovány byly různé možnosti výkonové optimalizace dlouhých seznamů a vhodné z nich byly implementovány. V práci byly také popsány možnosti ulehčení dalšího vývoje aplikace za použití navrženého Yeoman generátoru a vytvořených vlastních komponent. Všechny body zadání tedy byly splněny.

V návaznosti na tuto práci by bylo možné aplikaci rozšířit o nová rozhraní, která jsou obsažena v současných automobilech. V souvislosti s tím by samozřejmě mělo dojít i odpovídajícímu rozšíření serverové části. V úvahu je třeba vzít příchod nové verze 2.0 frameworku AngularJS a vyhodnotit její možný přínos. Implementováno by také mohlo být automatické kontrolování, zda některý z textů po přeložení do cizího jazyka není příliš dlouhý a „nepřetéká“ přes vyhrazenou oblast. To je ve webových technologiích možné kontrolovat způsobem popsaným v poslední kapitole této práce.

Literatura

[1] Václavík, Jan. *HTML 5 pro uživatelské rozhraní automobilu.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.

[2] Node Webkit Manifest format. [online], cit 2015-03-19. Dostupné z WWW <https://github.com/nwjs/nw.js/wiki/Manifest-format>

[3] Thon, Ladislav. *Lehký úvod do MongoDB.* [online], cit 2015-03-19. Dostupné z WWW <http://www.abclinuxu.cz/clanky/programovani/lehky-uvod-do-mongodb>

[4] Boaventura, Raoni. *Your First AngularJS App Tutorial Part 2: Tools for Scaffolding, Building, and Testing.* [online], cit 2015-03-19. Dostupné z WWW <http://www.toptal.com/angular-js/your-first-angularjs-app-part-2-scaffolding-building-and-testing>

[5] Hernandez, Matt. *NODE.JS AND MONGODB - GETTING STARTED WITH MONGOOSE.* [online], cit 2015-03-19. Dostupné z WWW <http://blog.modulus.io/getting-started-with-mongoose>

[6] Ravulavaru, Arvind. *Node Webkit – Build Desktop Apps with Node and Web Technologies.* [online], cit 2015-03-19. Dostupné z WWW <http://thejackalofjavascript.com/getting-started-with-node-webkit-apps>

[7] Object Modeling in Node.js with Mongoose. [online], cit 2015-03-24. Dostupné z WWW <https://devcenter.heroku.com/articles/nodejs-mongoose>

[8] Cooney, Dominic. *Shadow DOM 101.* [online], cit 2015-03-21. Dostupné z WWW <http://www.html5rocks.com/en/tutorials/webcomponents/shadowdom>

[9] Dodson, Rob. *A Guide to Web Components.* [online], cit 2015-03-21. Dostupné z WWW <https://css-tricks.com/modular-future-web-components>

[10] AngularJS: Developer Guide: Unit Testing. [online], cit 2015-03-21. Dostupné z WWW <https://docs.angularjs.org/guide/unit-testing>

[11] Karma: files. [online], cit 2015-03-21. Dostupné z WWW <http://karma-runner.github.io/0.8/config/files.html>

[12] AngularJS: Developer Guide: E2E Testing. [online], cit 2015-03-21. Dostupné z WWW <https://docs.angularjs.org/guide/e2e-testing>

[13] DAB - digitální rozhlas v ČR. [online], cit 2015-04-07. Dostupné z WWW <http://pureradio.cz/eshop-info-www/dab-digitalni-rozhlas-v-Cr>

[14] Fröstl, Sebastian. *AngularJS Performance Tuning for Long Lists.* [online], cit 2015-04-13. Dostupné z WWW <http://tech.small-improvements.com/2013/09/10/angularjs-performance-with-large-lists>

[15] *ngInfiniteScroll - Infinite Scrolling for AngularJS.* [online], cit 2015-04-13. Dostupné z WWW <http://binarymuse.github.io/ngInfiniteScroll>

[16] Vazzana, Pasquale. *Bindonce.* [online], cit 2015-04-13. Dostupné z WWW <https://github.com/Pasvaz/bindonce>

[17] Panda, Preetish. *What's New in AngularJS 2.0.* [online], cit 2015-04-14. Dostupné z WWW <http://www.sitepoint.com/whats-new-in-angularjs-2>

[18] Krátká, Nikola. *Radikální změny v AngularJS.* [online], cit 2015-04-14. Dostupné z WWW <http://www.itnetwork.cz/angularjs-2>

[19] *Writing your own Yeoman generator.* [online], cit 2015-04-22. Dostupné z WWW <http://yeoman.io/authoring/index.html>

Příloha **A**

Seznam použitých zkratek

BSON Binary JSON 14
CSS Cascading Style Sheets 16
DAB Digital Audio Broadcasting xiii
DOM Document Object Model 20
HTML HyperText Markup Language 16
IE Internet Explorer 13
JDK Java Development Kit 51
JSON JavaScript Object Notation 14
NoSQL Not only SQL 14
ODM Object Data Mapping 14
ORM Object-Relational Mapping 14
SQL Structured Query Language 14
UUID Universally Unique Identifier 14

Příloha **B**

Obsah přiloženého CD

```
readme.txt ..... stručný popis obsahu CD
src
    impl ..... zdrojové kódy implementace
    thesis ..... zdrojová forma práce ve formátu LaTeX
text ..... text práce
    DP_Doubek_Martin_2015.pdf ..... text práce ve formátu PDF
```