VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

HTML5 APLIKACE PRO SMARTPHONE A TABLET UMOŽŇUJÍCÍ ZNAČKOVÁNÍ MÍST V MAPĚ

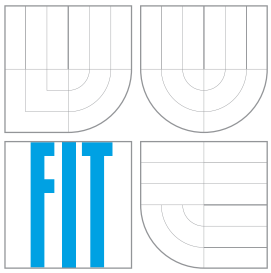
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VALTER KASPER

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

HTML5 APLIKACE PRO SMARTPHONE A TABLET UMOŽŇUJÍCÍ ZNAČKOVÁNÍ MÍST V MAPĚ

HTML5 APPLICATION FOR SMARTPHONE AND TABLET ALLOWING MARKING PLACES ON A MAP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

VALTER KASPER

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. BORIS PROCHÁZKA

BRNO 2013

Abstrakt

Rozvoj webových technologiím na čele s HTML5 standartem, dovoluje vývoj pokročilých mobilních aplikací jen s použitím HTML, CSS a JavaScriptu. V mnoha případech je tento přístup lepší, jako vytvářet nativní mobilní aplikaci zvlášť pro každou platformu. Představíme možný postup při vývoji reálné aplikace. Aplikace bude přistupovat k periferiím zařízení jen za pomoci HTML5.

Abstract

The development of web technologies headed by the HTML5 standard, has allowed creating of advanced mobile applications using only HTML, CSS and Javascript. In many cases, this approach is better, than creating native mobile application separately for each mobile platform. We will show you a possible approach to develop real applications in this way. The application will access peripheral devices only with the help HTML5.

Klíčová slova

mobilní aplikacie, hybridní aplikacie, HTML5, JavaScript, geolokace, smartfón, tablet, REST

Keywords

web app, mobile application, hybrid application, HTML5, JavaScript, geolacation, smartphone, tablet, REST

Citace

Valter Kasper: HTML5 aplikace pro smartphone a tablet umožňující značkování míst v mapě, bakalářská práce, Brno, FIT VUT v Brně, 2013

HTML5 aplikace pro smartphone a tablet umožňující značkování míst v mapě

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Procházka.

.....
Valter Kasper
12. května 2013

Poděkování

Chcel by som sa poďakovať vedúcemu práce za poskytnutú pomoc a Ústavu počítačovej grafiky a multimédií za zapožičanie mobilných zariadení pre účely testovania.

Obsah

Úvod

Moderné mobilné aplikácie sa do značnej miery líšia zložitosťou, môže sa jednať o jednoduchú kalkulačka alebo aj aplikáciu využívajúcu širokú škálu hardwaru zariadenia. Práve užitočné funkcie, prístup k perifériám zariadenia a v neposlednom rade príjemne pôsobiaci vzhľad, robí mobilné aplikácie zaujímavými.

Nástupom nového HTML štandardu nastal čas, keď webové aplikácie doháňajú možnosti tých natívnych. HTML5 sa nemusí použiť iba na tvorbu webových stránok, ale môže sa zúžitkovať aj jeho podpora pre prístup k perifériám. Na programovanie sa používa JavaScript, HTML a CSS, ktoré sú webovým vývojárom známe a tým odpadá učenie sa nových programovacích jazykov a prístupov. Aktuálnym trendom je vytvoriť jednu HTML5 aplikáciu, ktorá pracuje na rôznych typoch zariadení, či sa jedná o iPad, iPhone, zariadenia Android, BlackBerry alebo desktopové prehliadače.

V kapitole 1 Vám ukážem niektoré prvky HTML5, ktoré sa dajú využiť pri vývoji mobilných aplikácii. Budú predstavené nové HTML elementy a nové JavaScriptové aplikačno-programové rozhrania.

Kapitola 2 priblíži vývoj aplikácii s využitím HTML5 ako platformy. S explóziou mobilných aplikácii, čo do počtu a rozmanitosti, rastie aj potreba zlepšiť ich architektúru a kvalitu. JavaScript nebol navrhnutý ako jazyk pre všeobecné výpočty a aj preto boli vyvinuté prístupy, knižnice a frameworky, ktoré to umožňujú.

Ako demonštráciu prístupu, kde je aplikácia napísaná v JavaScripte a používa HTML5 technológie, bude popísaná tvorba takejto aplikácie v kapitole 3. Táto aplikácia siaha po fotoaparáte zariadenia, umožňuje značkovanie miest na mape a mnoho iného.

Vytvorená aplikácia bude otestovaná v kapitole 4, kde využijem niekoľko rôznych mobilných zariadení, smartfónov a tabletov, a urobím užívateľský test. Na konci kapitoly bude nasledovať zhodnotenie aplikácie i celého prístupu.

Kapitola 1

HTML5

V tejto kapitole bude predstavený značkovací jazyk HTML vo verzií 5. Na začiatku bude stručne popísaný vznik štandardu HTML5. Neskôr bude ozrejmené, čo sa za pojmom HTML5 skrýva. Potom budú popísané nové značkovacie elementy, aplikačno–programové rozhranie a iné HTML5 technológie.

Pri niektorých technológiách sa nachádza upozornenie, že existuje príklad na priloženom CD. Alternatívne sa dajú príklady pozrieť na adrese:
http://www.stud.fit.vutbr.cz/~xkaspe01/examples/

1.1 História

HTML bolo primárne určené ako jazyk pre sémantický popis vedeckých dokumentov, napriek tomu jeho všeobecný návrh v priebehu rokov dovolil, aby sa HTML použilo aj na popis iných typov dokumentov.

Počas rokov 1990-1995, HTML prešlo cez množstvo revízii a bolo vyskúšaných množstvo rozšírení. Počas týchto prvých piatich rokov prebiehal vývoj v CERN-e a neskôr na IETF.

Keď sa vytvorilo **W3C**, vývoj HTML znovu zmenil miesto. Prvé pokusy o rozšírenie HTML boli v roku 1995 známe pod označením HTML 3.0, nasledované verziou 3.2, ktorá bola dokončená v roku 1997. Verzia 4 nasledovala ešte v ten istý rok.

Nasledujúci rok sa W3C rozhodlo nepokračovať vo vývoji HTML a namiesto toho pracovať na jeho XML ekvivalente nazvanom XHTML. Táto snaha začala s formuláciou HTML 4 v XML, nazvanou XHTML 1.0 a bola dokončená v roku 2000.

Okolo roku 1998, keď sa vývoj vo W3C zastavil, časť z aplikačno-programového rozhrania pre nové HTML, vyvíjaného výrobcami prehliadačov, bolo špecifikované a vydané ako **DOM 1**, **DOM Level 2 Core** a **DOM Level 2 HTML**. Špecifikácia DOM Level 3 bola vydaná v roku 2004, ale pracovná skupina bola uzavretá skôr, ako boli Level 3 návrhy kompletné.

Návrh, že by bolo dobré pokračovať vo vývoji HTML, bol kladne prijatý na W3C workshope v roku 2004. Na tomto mieste boli prezentované niektoré z princípov, ktoré boli základom pre prácu na HTML5. Tento návrh bol ale zamietnutý zo strany zamestnancov a členov W3C na základe toho, že návrh bol v konflikte s predchádzajúcim vybraným smerom pre vývoj webu. Preto hlasovali pre pokračovanie v XML nástupcovi.

Krátko nato, Apple, Mozilla a Opera spoločne ohlási svoj zámer, pokračovať na práci pod záštitou novej spoločnosti nazvanej **WHATWG**. Bol vytvorený verejný *mailing list* a návrh bol presunutý na stránky WHATWG.

WHATWG prístup bol založený na niekoľkých základných princípoch a to konkrétne:

- Technológie musia byť spätne kompatibilné.
- Špecifikácia a implementácia musia súhlasiť a to i vtedy, keby to znamenalo, že sa musí zmeniť špecifikácia namiesto implementácie.
- Špecifikácia musí byť natoľko detailná, že implementácia môže dosiahnuť úplnej kompatibility bez vzájomného spätného inžinierstva.

V roku 2006 W3C naznačila záujem, zúčastniť sa napriek všetkému na vývoji HTML5 a následne v roku 2007 vytvorila skupinu, ktorá mala spolupracovať s WHATWG na vývoji HTML5 špecifikácie[1].

1.1.1 WHATWG, W3C HTML5 špecifikácia a súvisiace technológie

HTML5 nie je jedna monolitická vec, ale skôr súbor funkcii, technológii a aplikačno-programových rozhraní. Dokopy tieto technológie dovoľujú vytvárať komplexné aplikácie, ktoré mohli byť v minulosti vytvárané iba na desktopové platformy.

Pojem HTML5 je často používaný ako označenie pre moderné webové technológie. Veľa z týchto technológii je popísaných vo **W3C a WHATWG špecifikáciach**. W3C špecifikácia je podmnožinou WHATWG špecifikácie[2] a v ďalších častiach práce pod pojmom HTML5 špecifikácia bude myslená práve WHATWG špecifikácia. Vzťahy medzi špecifikáciami možno pozorovať na obrázku 1.1.

Existujú príbuzné technológie, ktoré nie sú súčasťou ani jednej zo spomínaných špecifikácii. Tieto technológie vydáva W3C s vlastnými špecifikáciami. Príbuzné HTML5 technológie sú napríklad tieto:

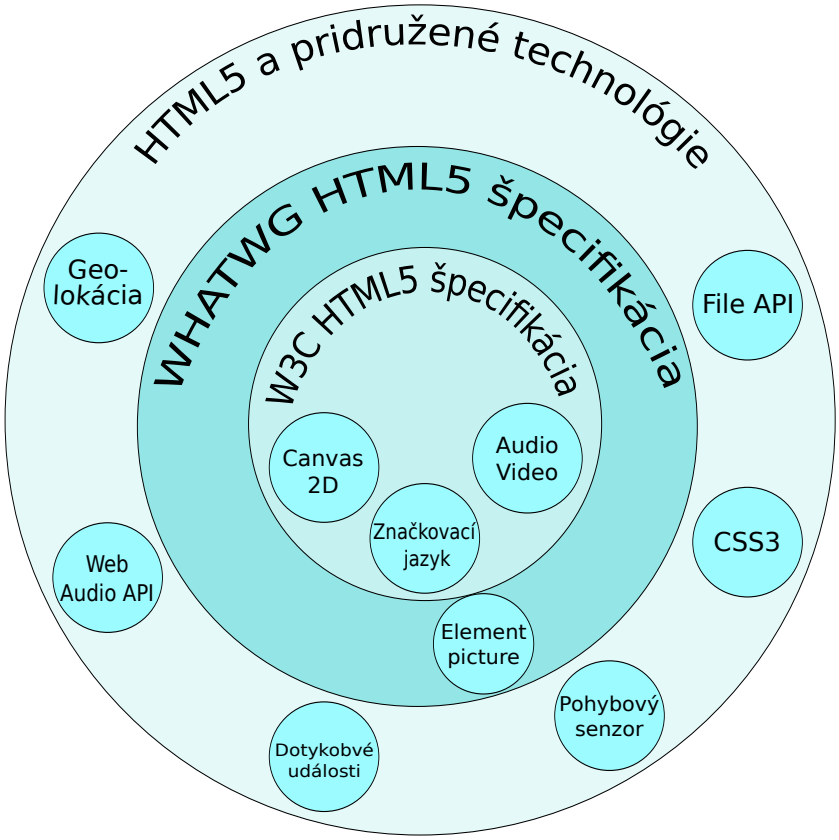
- geolokácia,
- Web SQL databáza,
- indexovaná databáza,
- HTML5 File API,
- File Writer,
- Web Audio API.

V nasledujúcich sekciách budú popísanie niektoré HTML5 technológie z WHATWG špecifikácie a niektoré súvisiace technológie.

1.2 Značkovací jazyk

HTML5 špecifikácia upravuje okrem iného aj sémantický význam niektorých starších HTML značiek a k tomu zavádza niekoľko nových značiek. Niektoré z nich sú náhradou za bežné používanie značky `<div>` alebo *inline* značky `<span>` novými značkami, ako je napríklad `<article>` a `<nav>`. Tieto značky majú za úlohu sémanticky popísať rôzne časti dokumentu.

Viaceré elementy v HTML5 ako je `<article>`, `<aside>`, `<nav>` a `<section>` majú špeciálny význam — rozdeľujú obsah stránky na sekcie. Každá taká sekcia môže mať vlastný `<header>` a `<footer>`[3].



Obrázok 1.1: Vzťahy medzi jednotlivými špecifikáciami a technológiami. V obrázku nie sú zahrnuté všetky technológie a technológie môžu svoju pozíciu v grafe časom meniť.

Deklarácia typu dokumentu

Deklarácia typu dokumentu — `<!DOCTYPE>` nie je HTML značkou. HTML5 upravuje *Doctype*, ktorý pôvodne označoval používanú verziu HTML a obsahoval cestu, kde nájsť **DTD**[1], pretože HTML 4.1 bolo postavené na **SGML**. Teraz sa uvádza iba `<!DOCTYPE html>`, pretože HTML5 nie je postavené na SGML a preto nevyžaduje referenciu na DTD[4].

Section

Element `<section>` označuje všeobecný dokument alebo časť aplikácie. V podstate je to nejaký typ obsahu, ktorý by sa mohol uložiť ako záznam do databázy. Nemal by sa používať, ak pre danú sekciu neexistuje prirodzený nadpis a nemal by sa používať ako element, ktorý na seba viaže štýly, poprípade skripty, eventuálne sa autorom odporúča namiesto toho použiť značku `<div>`. Značka `<section>` je najvšeobecnejšia zo všetkých značiek sekcií a preto sa odporúča používať značku `<article>` namiesto `<section>` vždy, keď to pre obsah elementu dáva zmysel[5].

[1] DTD špecifikuje pravidlá pre značkovací jazyk, aby prehliadač mohol vykresliť korektne obsah.

Article

Značka `<article>` reprezentuje ucelenú časť dokumentu, stránky alebo aplikácie a je ju možné nezávisle opakovane použiť. Touto časťou môže byť diskusný príspevok, časopisecký článok, novinový článok, príspevok na blogu, komentár odoslaný užívateľom, interaktívny widget — gadget alebo iný nezávislý prvok obsahu. Nezávislá časť obsahu, ktorá je vhodná na vloženie do elementu <article>, musí dávať sama o sebe zmysel. [6]

Elementy `<article>` môžu byť zanorené. Vnútorný element `<article>` reprezentuje článok, ktorý v princípe tvorí súčasť vonkajšieho článku. Napríklad jednotlivé komentáre odoslané užívateľmi, sú obalené elementom `<article>` a tie sú zanorené v elemente `<article>` príspevku na blogu[7].

Time

Jednotlivé elementy `<article>` môžu obsahovať element `<time>`. Medzi otváracou a uzatváracou značkou obsahuje čitateľný text a v atribúte **datetime** element `<time>` obsahuje dátum, čas alebo trvanie v platnom formáte. [8][9]

```
1    <article>
2      <header>
3        <h3>Pridal: Jan Novak</h3>
4        <p><time pubdate datetime="2009-10-10T19:15-08:00">Pred 1
             hodinou</time></p>
5      </header>
6      <p>Urgh, jablka!? mali by ste radsej pisat o~pomarancoch!!!</p>
7    </article>
```

Ukážka kódu 1.1: Príklad použitia <article> a <time>.

Aside

Element `<aside>` slúži na obalenie sekundárneho obsahu. Rozlišujú sa dva druhy použitia:

- Zanorený v elemente `<article>`. Ak je `<aside>` začlenený do `<article>` elementu, tak by sa mal konkrétne týkať článku, napríklad glosár.
- Mimo elementu `<article>`. Keď je použitý mimo `<article>`, tak obsah by mal byť venovaný celej stránke, napríklad zoznam blogov, ďalšia navigácia alebo reklama[10].

Nav

Nový element `<nav>` poskytuje možnosť sémanticky vyčleniť časť dokumentu, ktorá bude združovať odkazy na iné stránky alebo iné časti aktuálnej stránky. Nie všetky skupiny odkazov na stránke musia byť zoskupené v elemente `<nav>`, týka sa to iba hlavnej navigácie. Napríklad je bežné, že pätička obsahuje odkazy na rôzne časti stránky, ale `<footer>` element je v takomto prípade vhodnejší obaľovací element ako `<nav>`[11].

Header

Element `<header>` označuje úvodnú časť pre jeho priameho predka. Môže obsahovať nadpis sekcie, elementy `<h1>` – `<h6>`, obsah sekcie, vyhľadávanie alebo logo. Na stránke sa

môže použiť viacero sekcii `<header>`, jedna pre celú stránku a ďalšie pre je ostatné sekcie stránky[12].

Footer

Použitie elementu `<footer>` je veľmi podobné ako použitie elementu `<header>`. Element `<footer>` reprezentuje pätičku k jeho najbližšiemu predkovi. `<footer>` typicky obsahuje informácie o jeho sekcii, ako je meno autora, odkazy na súvisiace dokumenty alebo autorské práva[13].

Figure a figcaption

Element `<figure>` reprezentuje časť obsahu dokumentu, voliteľne s popiskom `<figcaption>`, ktorý je sebestačný a mohol by byť teoretický vybratý z hlavného dokumentu bez jeho ovplyvnenia. `<figcaption>` môže byť umiestnený pred alebo za obsah v elemente `<figure>`.

Ako bolo spomínané v sekcii 1.2, element `<aside>` sa požíva na oddelenie sekundárneho obsahu. Je potrebné vedieť rozdiel medzi týmito dvoma elementami:

- Ak je obsah vo vzťahu k hlavného obsahu, ale nie je podstatný, používa sa `<aside>`.
- Ak je obsah podstatný ale jeho pozícia v dokumente v toku dokumentu nie je dôležitá, tak sa používa `<figure>`.

Obsah vhodný pre `<figure>` element môže byť: videá, obrázky, diagramy alebo tabuľky[14].

```
1 <figure>
2   <img src="/yosemite.jpg" alt="Yosemite Valley">
3   <figcaption>Znama fotografia Yosemitskej doliny. Znimku vytvoril <a
        href="http://www.anseladams.com/">Ansel Adams</a></figcaption>
4 </figure>
```

Ukážka kódu 1.2: Príklad použitia <figure>.

Video

Novým elementom v HTML5 je aj `<video>`. Umožňuje prehrávať, podobne ako element `<audio>`, multimédia bez použitia doplnkov. Používa sa nastavením niekoľkých atribútov, ktorými sa ovplyvňuje jeho chovanie:

- **autoplay** — umožňuje prehrať video po načítaní stránky.
- **download** — umožňuje stiahnúť video. Vhodné v prípade, ak video nie je možné prehrať alebo sa vyskytla nejaká chyba.
- **autobuffer** — atribút sa použije v prípade, ak autoplay nie nastavené, ale vlastník stránky si myslí, že sa video prehrá. Video je stiahnuté v pozadí, takže ak sa ho užívateľ rozhodne prehrať, tak bude prednačítané. Je dôležité podotknúť, že prehliadače sťahujú video dopredu nezávisle na tom, či sa atribút autobuffer použije.
- **poster** — pri použití tohto atribútu, sa použije jeho hodnota ako adresa obrázku, ktorý sa vykreslí v prípade, že sa z nejakého dôvodu video nenačítalo.

- **controls** — pridanie tohto atribútu znamená, že použijete vlastné ovládacie prvky.
- **subtitles** — v budúcnosti budeme schopný vďaka tomuto atribútu dodať do videa titulky bez použitia JavaScriptu[15].

Zmeny v sémantike existujúcich elementov

Zatiaľ čo väčšina HTML4 elementov prešla do HTML5 bez zmeny, tak niekoľko prezentačných elementov zmenilo svoj sémantický význam. Medzi tieto elementy patrí:

- `<i>` — znamenal kurzívu, po novom označuje slová, ktoré by mali byť čítané inak ako okolitý text. Napríklad cudzie slová.
- `<b>` — pôvodne tučné písmo, teraz označuje slová, ktoré sú štylisticky odlišné od okolitého textu. Napríklad kľúčové slová alebo produktové názvy[16].
- `<em>` — časť textu, na ktorú by ste mali pri čítaní dať dôraz.
- `<strong>` — časť textu, ktorá je dôležitá.
- `<small>` — pôvodne označoval text, ktorý prehliadače vykreslovali menším písmom. Teraz dostal aj sémantický význam: označuje poznámky k textu. Často sa používa pre legálne poznámky, ako napríklad odkazy na licencie obrázkov alebo licencie použitých fontov[3].

`<i>` a `<b>` elementy mali v HTML4 význam pre štylizovanie fontu, teraz majú sémantický význam a štylizovanie sa môže zmeniť cez CSS. Napríklad `<b>` element nemusí obsahovať tučné písmo[3].

1.3 Aplikačno-programové rozhranie

V nasledujúcich sekciach predstavíme aplikačno-programové rozhrania, ktoré je možne využívať pri písaní HTML5 aplikácii. Väčšina z nich je dostupná už teraz, no niektoré ešte len čakajú na implementáciu v prehliadačoch.

Je škoda, že HTML5 v súčastnosti nedokáže komunikovať s **GSM modulom**. Mohlo by to umožniť napr. písanie SMS správ priamo z prostredia webovej stránky alebo spustenie aplikácie pre telefónny hovor so zvoleným telefónnym číslom. Možno sa v budúcnosti dočkáme aj takejto podpory.

1.3.1 Webové úložisko

Webové úložisko (angl. web storage) umožňuje ukladať údaje o relácii v podobe kľúč — hodnota. Dáta sú uložené v prehliadači a pri požiadavku sa neprenášajú na server, na rozdiel od *cookies*. Dáta môžeme ukladať dvoma spôsobmi:

- **sessionStorage** — používa sa na ukladanie krátkodobých dát. Dáta ostávajú uložené iba po dobu relácie prehliadača.
- **localStorage** — dáta ostávajú v prehliadači aj po skončení relácie (nezávisí na životnosti karty/okna prehliadača). Môže sa použiť ako efektívna náhrada cookies.

Webové úložisko poskytuje oveľa väčší priestor pre ukladanie dát ako cookies. Doporučená veľkosť pre každý zdroj (doménu) je **5MB**. V súčasnej dobe je lokálne úložisko priateľnou formou pre ukladanie dát na strane klienta. Je vhodné hlavne pri webových aplikáciach, ktoré sú schopné pracovať bez internetového pripojenia[17]. Príklad použitia je na priloženom CD (/examples/web_storage.php).

```
1  // nastavi par kluc-hodnota
2  localStorage.setItem("meno", "Jan Novak");
3
4  // ziska hodnotu
5  // vrati "Jan Novak"
6  localStorage.getItem('meno');
7
8  // odstrani par kluc-hodnota
9  localStorage.removeItem('meno'); // odstrani meno
10
11 // vycistenie ulozista
12 localStorage.clear();
```

Ukážka kódu 1.3: Príklad použitia lokálneho úložišťa.

1.3.2 Dotykové udalosti

Na zariadeniach, ponukajúcich ovládanie dotykom, používajú webové aplikácie typicky interpretované udalosti vyvolané myšou, na prístup k interaktívnym webovým aplikáciam. Týmto spôsobom nie je možné obslúžiť viacnásobný vstup týchto udalostí, kvôli obmedzeniam udalosti myši.

Dotykové udalosti poskytujú riešenie tohto problému. Existujú tri základné dotykové udalosti a to: **touchstart** — prst je položený DOM elemente, **touchmove** — prst sa posunul po DOM elemente a **touchend** — prst sa zdvihol z DOM elementu.

Každá udalosť zahrňuje nasledujúce zoznamy dotykov:

- **touches** je zoznam dotykových bodov, ktoré sa aktuálne dotýkajú obrazovky,
- **targetTouches** je zoznam dotykových bodov, ktoré sa dotýkajú obrazovky a súčasne začali sa elemente, ktorý je cieľom udalosti,
- **changedTouches** je zoznam dotykových bodov, ktoré prispeli k vyvolaniu udalosti.

Každý dotyk obsahuje nasledujúce atribúty:

- **identifier** je identifikačné číslo každého dotykového bodu. Keď dotykový bod začne byť aktívny, tak sa mu musí priradiť jedinečný identifikátor. Pokiaľ je bod aktívny, tak všetky udalosti ktoré naň ukazujú, musia mať prístup k rovnakému identifikátoru.
- **target** je cieľový element udalosti, na ktorom dotykový bod začal a to dokonca v prípade, keď sa dotykový bod pohol mimo oblasť elementu.
- **coordinates** je pozícia, kde sa udalosť odohrala. Obsahuje tri druhy koordinátov: koordináty prehliadača, stránky a obrazovky.
- **polomer koordinátov a uhol rotácie.**

Dotykové udalosti poskytujú bohatú množinu funkcií pre podporu interakcie užívateľa pomocou dotyku, napr. multi — dotykové gestá, ako je priblíženie či rotácia [18][19]. Príklad použitia je na priloženom CD (/examples/touch.php).

1.3.3 Geolokačne rozhranie

Toto rozhranie popisuje jednoduchú službu pre webové aplikácie, ktorá umožňuje pracovať s aktuálnou polohou zariadenia. Po povolení užívateľom, môže webová aplikácia jednorázovo alebo opakovanie pristupovať k dátam o polohe, ktoré boli získané z GPS senzoru, mobilných sietí, wifi a podobne.

Úlohou geolokačného rozhrania nie je samotná lokalizácia. Rozhranie len zaobstaráva cestu pre získanie informácií o polohe, nemá teda na starosti ich samotné získanie. To robí zariadenie, na ktorom beží prehliadač a zachytáva žiadosti o poskytnutie polohy. Informácie o polohe sú vrátené v bežnom desatinnom formáte, ktorý vyjadruje zemepisné súradnice.

K lokalizačným informáciám sa pristupuje pomocou JavaScriptu s využitím metódy `getCurrentPosition()` objektu `navigator.geolocation`.

```
1 navigator.geolocation.getCurrentPosition(showLocation, errHandler,
      options);
```

Ukážka kódu 1.4: Volanie metódy getCurrentPosition.

Popis parametrov funkcie `getCurrentPosition`:

- **showLocation** — určuje callback funkciu, ktorá je zavolaná po získaní informácie o polohe. Tejto funkcii sú predané parametre koordinátov a časové razítko.
- **errorHandler** — callback funkcia, ktorá sa zavolá v okamihu výskytu chyby. Predajú sa jej informácie o chybe.
- **options** — určuje nastavenia získavania polohy.
  - **enableHighAccurancy** — ak je táto možnosť nastavená, tak sa získa najpresnejšia možná poloha.
  - **timeout** — nastavuje časový limit na získanie polohy. V prípade vypršania limitu sa vyvolá errorHandler. Hodnota je v milisekundách.
  - **maximumAge** — určuje aký najstarší údaj môže prehliadač vybrať z medzipamäti. Hodnota je v milisekundách.

Nasleduje popis atribútov objektu koordinátov, ktorý je predaný callback funkcii v `getCurrentPosition`:

- **latitude** — zemepisná šírka,
- **longitude** — zemepisná dĺžka,
- **accuracy** — presnosť zemepisnej dĺžky a výšky v metroch,
- **altitude** — nadmorská výška v metroch,
- **altitudeAccuracy** — presnosť nadmorskej výšky v metroch,
- **speed** — aktuálna rýchlosť zariadenia v metroch za sekundu,

- **heading** — smer pohybu zariadenia v stupňoch, v smere hodinových ručičiek.

Základné údaje (latitude, longitude a accurancy) sú poskytnuté vždy. Zatiaľ čo ostatné údaje nemusia byť dostupné, v takom prípade majú hodnotu NULL[20]. Rozsiahlejší príklad použitia je na priloženom CD (/examples/geolocation.php).

```
navigator.geolocation.getCurrentPosition (showLocation);
// callback funkcia pre ziskanie polohy
function showLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    alert("Zemepisna sirka: " + latitude + " dlzka: " + longitude);
}
```

Ukážka kódu 1.5: Príklad použitia získania polohy.

Opakovane získavaná poloha

Ako sme si ukázali, k lokalizačným údajom sa jednorázovo pristupujeme pomocou metódy `getCurrentPosition()`. Ak požadujeme periodicky obnovovať informácie o polohe, tak je možné využiť metódu `watchPosition()`. Jej použitie je jednoduché, má rovnakú syntax ako `getCurrentPosition()`, no jediným rozdielom je to, že metóda vracia identifikátor požiadavku, ktorý sa môže použiť na zrušenie sledovania polohy[21].

1.3.4 Canvas

Canvas je nový element HTML5, ktorý dovoľuje dynamicky vykreslovať 2D tvary a bitmapové obrázky. Značky `<canvas>` používajú `context`, s ktorým sa môže pomocou JavaScriptu kresliť do canvasu. Prehliadače môžu implementovať viacero contextov s rozličnými rozhraniami. Väčšina prehliadačov implementovala 2D context canvasu[22].

Príklad použitia je na priloženom CD (/examples/canvas.php).

1.3.5 Zachytávanie audia a videa

V priebehu vývoja HTML5 sa vyvinulo niekoľko variant rozhraní pre zachytávanie médií. Jednou možnosťou ako toho dosiahnuť, je formulárový prvok `<input>`, ktorý po kliknutí automaticky spustí aplikáciu pereférie, umožňujúcu záznam daného média. Druh média sa špecifikuje cez atribút *accept* a jeho povolené hodnoty sú *image/**, *audio/** alebo *video/**. Ďalším atribútom je *capture*. Je to atribút pomocný, určujúci vstup pre dané médium. Implicitnou hodnotou je súborový systém a prípustné hodnoty sú nasledovné: *camera, camcoder, microphone, filesystem*[23].

```
<form enctype="multipart/form-data" method="post">
  <input type="file" accept="image/*" capture="camera">
</form>
```

Ukážka kódu 1.6: Získanie fotografie.

Mocnejšou a komplexnejšou možnosťou zachytávania medií je metóda `getUserMedia()`. So spomínanou metódou možno získať živý *stream*, ktorý sa dá modifikovať, nahrávať alebo streamovať ďalej. Jej použitie je jednoduché, vyžaduje dva povinné argumenty a jeden voliteľný: Prvým je JavaScriptový objekt, ktorý určuje druh média. Napríklad `audio: true,`

`video: true` pre záznam audia aj videa. Druhý je callback, ktorý sa zavolá v prípade úspechu a tretím a voliteľným parametrom je callback zavolaný v prípade chyby[24].

```
1 navigator.getUserMedia({video: true}, onSuccess, onError);
2
3 function onSuccess(stream) { // uspech
4 }
5
6 function onError(){ // neuspech
7 }
```

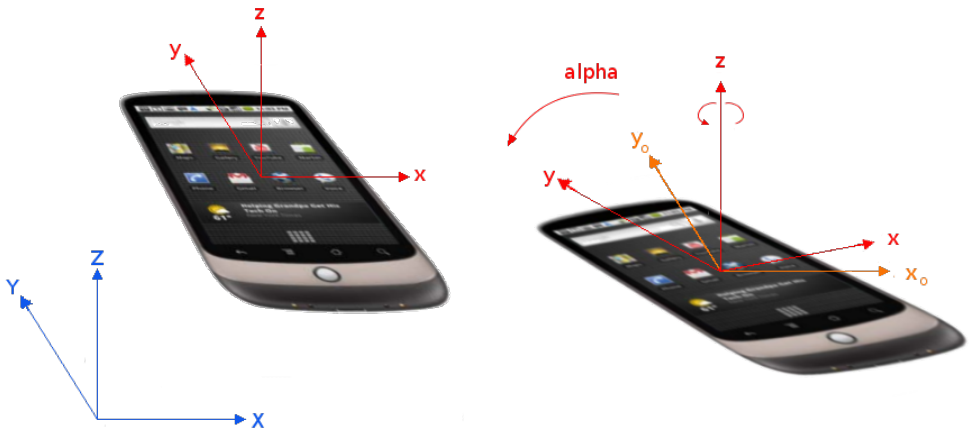
Ukážka kódu 1.7: Použitie getUserMedia na zachytenie videa.

1.3.6 Spolupráca s pohybovými senzormi

V dnešnej dobe veľa nových zariadení s prístupom na internet disponuje perifériami, umožňujúcimi získavanie informácii o orientácii. Takýmito perifériami sú akcelometer, gyroskop alebo kompas. S výskytom týchto periférii sa zlepšili schopnosti prehliadačov s nimi pracovať. Príkladom toho sú nové udalosti, ktoré poskytujú vývojárovi informácie o orientácii a pohybe: *DeviceOrientation* a *DeviceMotion*.

Informácie sú poskytované v podobe hodnôt prislúchajúcich jednotlivým osiam x, y, z zariadenia (viď. 1.2). Ak sa jedná o informácie o pohybe, hodnoty vyjadrujú zrýchlenie pozdĺž jednotlivých osí v metroch za sekundu na druhú $[m/s^2]$. Vzhľadom k tomu, že niektoré zariadenia nemusia mať hardware schopný vylúčiť vplyv gravitačného zrýchlenia, tak udalosť poskytuje dve vlastnosti: *accelerationIncludingGravity* a *acceleration*. Ak zariadenie nedokáže vylúčiť vplyv gravitácie, tak *acceleration* má hodnotu *NULL*. Informácie o rotácii sú vyjadrené ako rozdiel súčasnej polohy a normálnej polohy zariadenia v eulerových uhloch[25].

Rozsiahlejší príklad použitia je na priloženom CD (/examples/orientation.php).



Obrázok 1.2: Počiatočná pozícia zariadenia a rotácia pozdĺž osy z. Obrázok je prevzatý z http://dev.w3.org/geo/api/spec-source-orientation

Príklady užitia:

- Webová hra kde pohyb v prostredí hry môže byť ovládaný naklonením zariadenia.
- Pokročilé gestá, ktoré využívajú akceleráciu zariadenia. Môže sa jednať napríklad o vymazanie webového formuláru pri zatrasení zariadením.
- Mapová webová aplikácia využívajúca otočenie zariadenia na zarovnanie mapy podľa reality[26].

1.3.7 Zistenie stavu batérie zariadenia

Programové rozhranie pre HTML5 bude poskytovať možnosť získať informácie o stave batérie zariadenia. V súčastnosti je podpora prehliadačov pre spomínané rozhranie slabá. Získaný poznatok o stave batérie umožní vývojárovi napríklad uspôsobiť obsah stránky, aby sa znížila/zvýšila spotreba energie. Povedzme, ak webová aplikácia v pravidelných intervaloch synchronizuje väčší objem dát so serverom, tak frekvencia synchronizácie sa môže znížiť, ak zariadenie nie je nabíjané zo siete.

Objekt `navigator.battery` obsahuje informácie o napájaní a má nasledujúce vlastnosti:

- **charging** reprezentuje status, či sa zariadenie nabíja,
- **chargingTime** ostávajúci čas v sekundách, kým sa batéria plne nabije,
- **dischargingTime** ostávajúci čas v sekundách, kým sa batéria vybije a zariadenie sa vypne,
- **level** udáva aktuálnu úroveň batérie v rozsahu od 0 do 1.0[27].

Rozsiahlejší príklad použitia je na priloženom CD (/examples/battery.php).

1.3.8 Offline prístup

Webové HTML5 aplikácie umožňujú fungovanie, aj keď nie je dostupné pripojenie k internetu. Offline webová aplikácia obsahuje zoznam URL adries na HTML, CSS, JavaScriptové, obrázkové alebo iné súbory. Hlavná stránka webovej aplikácie odkazuje na súbor, ktorý obsahuje tieto informácie. Tento súbor, nazývaný manifest, je obyčajný textový súbor umiestnený na webovom serveri. Webový prehliadač stiahne súbory zo zadaných URL adries obsiahnutých v manifeste a uloží ich lokálne v prehliadači. Keď sa pokúsite pristúpiť na stránku bez internetového pripojenia, webový prehliadač automaticky použije lokálne uložené súbory.

```
1 <!DOCTYPE HTML>
2 <html manifest="/cache.manifest">
3     <body>
4     ...
5     </body>
6 </html>
```

Ukážka kódu 1.8: Pridanie atribútu manifest do elementu HTML.

```
1 CACHE MANIFEST
2 /style.css
```

```
3 /app.js
4 /app_logo.jpg
```

Ukážka kódu 1.9: Príklad validného súboru manifest.

V objektovom modeli prehliadača, je vlastnosť `navigator.onLine`, pomocou ktorej sa dá detekovať, či je prehliadač *online* alebo *offline*, prípadne existujú udalosti **online** a **offline**, ktoré oznamujú zmenu stavu pripojenia. Pre ukladanie stavu aplikácie v stave offline, je vhodné použiť webové úložisko 1.3.1 a potom zosynchronizovať so serverom, keď je znovu dostupné pripojenie[28].

1.3.9 Podpora HTML5 v prehliadačoch mobilných zariadení

V tabuľke 1.1 je prehľadovo zachytená podpora HTML5 v mobilných prehliadačoch.

Názov technológie	**iOS Safary prehliadač**	**Chrome pre Android**	**Android prehliadač**	**Opera Mobile**	**BlackBerry preh. pre mobily**	**Windows Phone 7.5 prehliadač**	**Internet Explorer 10**
Offline prístup	X	X	X	X	X	-	X
Webové úložisko	X	X	X	X	X	X	X
Webové SQL databáza	X	X	X	-	X	-	-
Audio a video prehrávač	X	X	X	X	X	X	X
Canvas API	X	X	X	X	X	X	X
Pohybový senzor	X	X	X	X	-	-	-
Geolokácia	X	X	X	X	X	X	X
Web Sockets	X	X	-	X	X	-	X
SVG formát	X	X	X	X	X	X	X
Web Workers	X	X	-	X	X	-	X
Dotykové udalosti	X	X	X	X	X	-	X
WebGL (3d canvas)	-	-	-	X	-	-	-
XMLHttpRequest 2.0	X	X	X	X	-	-	X
File API	X	X	X	X	-	-	X
Záznam médii	X	X	-	-	-	-	X
getUserMedia	-	-	-	X	-	-	-

Tabuľka 1.1: Prehľadová tabuľka podpory jednotlivých HTML5 technológii v mobilných prehliadačoch. Do úvahy sa berie iba najvyššia verzia daného prehladača. V tabuľke je zahrnutý aj prehliadač Internet Explorer 10, pretože sa nachádza v systéme Windows 8, ktorý je možné nainštalovať na tablety. Údaje prevzaté z http://mobilehtml5.org/.

Kapitola 2

Vývoj mobilných aplikácii

Pred niekoľkými rokmi boli mobilné aplikácie iba jednoduché programy s nekomplikovaným užívateľským rozhraním, obsahujúcim väčšinou iba text, odkazy, prípadne obrázky. Ovládaniu chýbala podpora pre dotykové gestá a aplikácie nemohli pristupovať k perifériám telefónu. Neskôr prichádza na trh nový typ inteligentného telefónu iPhone, nasledovaný zariadeniami postavenými na platforme Android. Tieto zariadenia disponujú veľkými obrazovkami, sú prispôsobené na ovládanie dotykom a majú mnohé periférne zariadenia ako fotoaparát, GPS modul, elektronický kompas alebo pohybový senzor, ktoré sú dostupné pre vývojárov aplikácii. Tvorcovia nových platforiem poskytujú týmto vývojárom nástroje na vývoj pre danú platformu, tzv. *SDK*. Sú to zostavovacie systémy, aplikačno-programové rozhrania, ladiace nástroje, emulátory či simulátory zariadení.

Samozrejme aplikácie sa píšu v programovacom jazyku danej platformy a používajú sa vlastné nástroje na vývoj. Ďalej sa musia zohľadniť rôzne možnosti jednotlivých platforiem. Takže pri požiadavke na vývoj aplikácie pre rôzne mobilné platformy, sú vývojári nútení vyvíjať zvlášť pre každý operačný systém.

2.1 Paradigma mobilných aplikácii

Pre vývojára je dôležité vedieť, ktoré paradigma je vhodné pre jeho typ aplikácie a typ používania. V tejto sekcii budú predstavené populárne prístupy pre tvorbu mobilných aplikácii podľa [29].

Natívne aplikácie
: Mobilná natívna aplikácia je špecificky vytvorená tak, aby fungovala na jednej platforme. Nemôže byť použitá na zariadeniach, ktoré používajú inú platformu. Pre väčšinu moderných mobilných platforiem je bežné, že aplikácie sa získavajú a inštalujú z obchodu aplikácii danej platformy, napr. App Store v systéme iOS alebo Google Play v platforme android. S týmto prístupom sa dajú vytvárať robusné, užívateľsky prívetivé a relatívne rýchle mobilné aplikácie.

Mobilné widgety
: Mobilné widgety sú špecifické aplikácie, ktoré zobrazujú webový obsah a využívajú pritom webové technológie ako HTML, CSS alebo JavaScript. Tieto aplikácie bežia v behovom prostredí danej platformy napr. Opera, Nokia WRT, Samsung TouchWiz a Yahoo!Blueprint.

Mobilná webová aplikácia
Mobilné webové aplikácie sú hlavne vhodné pre poskytovanie informácii, ktoré sú uložené na webovom servery. Väčšinou využívajú architektúru, kde stránka zobrazená na mobilnom zariadení, prekresľuje svoj obsah pri komunikácií so serverom, ktorý zas získava dáta z databázy. Pomocou tohto paradigmu je ťažké vytvoriť mobilnú aplikáciu s pokročilými funkciami.

HTML5 mobilné aplikácie
HTML5 mobilná aplikácia sa líši od predchádzajúceho typu tým, že technológie ktoré používa, sú obsiahnuté v HTML5 a v pridružených technológiách, takže nemusí využívať pluginy a doplnky ako je Adobe Flash, Java Oracle FX. HTML5 aplikácie sa možnosťami približujú natívnym aplikáciam.

Hybridná mobilná aplikácia
Hybridná mobilná aplikácia využíva možnosti HTML5 a natívnych aplikácii. Podobne ako natívna aplikácia, hybridná aplikácia beží v mobilnom zariadení — zdrojové súbory sú stiahnuté v zariadení, ale aplikácia je napísaná pomocou HTML5, CSS a JavaScriptu. Tento prístup spája výhody oboch prístupov, a to využitie možností zariadenia a prenositeľnosť. Príkladom frameworkov, ktoré takýto prístup umožňujú sú PhoneGap, Titanium alebo Appspresso.

Bližšie porovnanie spomínaných paradigm z rôznych pohľadov je možné nájsť v [29].

2.2 Vývoj mobilných aplikácií s použitím webových technológií

Nástup technológií zoskupených okolo štandardu HTML5, prispel novými možnosťami pri písaní webových stránok. Čo predtým bolo možné iba za pomoci proprietárnych doplnkov, teraz je možné iba za pomoci HTML5. Podpora HTML5 štandardu v moderných mobilných prehliadačoch je na veľmi dobrej úrovni.

Využitie HTML5 ako platformy pre vývoj aplikácii, umožnili hlavne tieto technológie: offline prístup, webové úložisko, canvas a dotykové udalosti. Vďaka ním sa webové aplikácie priblížili možnostiam desktopových aplikácii, v mnohých oblastiach ich dohnali a v oblasti prenositeľnosti medzi platformami ich predbehli.

2.2.1 Single-page application

Single-page application, ďalej SPA, je webová aplikácia, ktorá pozostáva iba z jednej stránky, za účelom poskytnúť plynulejšiu užívateľskú skúsenosť podobnú s desktopovými aplikáciami alebo natívnymi mobilnými aplikáciami.

Pri narastajúcom počte komponent na stránke, narastá počet stavov, ktoré je potrebné uchovávať. Vykresľovanie stránok na strane servera je veľmi ťažko implementovateľné pre všetky tieto nezávislé stavy — tieto malé stavy je ťažké namapovať do URL adries. Tradičné webové aplikácie pracujú pomocou prekreslenia celej webovej stránky, po interakcii užívateľa s užívateľským rozhraním, napríklad po kliknutí na hypertextový odkaz.

Z užívateľského hľadiska, neustále prekresľovanie ruší užívateľa, pretože latencie siete nemôžu byť pred ním skryté. Aj stabilné prvky užívateľského rozhrania, ako je navigácia, neustále miznú a znovu sa objavujú. Z pohľadu výkonu, opätovné prekreslenie stránky, po interakcii užívateľa so stránkou znamená, zbytočné prenesie dát.

Oproti tomu SPA funguje bez toho, aby sa pri každej zmene stavu komponenty musel dotazovať server o jej prekreslenie alebo o prekleslenie celej stránky a následného zaslania HTML kódu. SPA toto dosahujú oddelením dát od prezentácie pomocou oddelených vrstiev — modelovej vrstvy, ktorá sa stará o data a prezentačnej vrstvy, ktorá číta dáta z modelov a tie ďalej vykresluje. Dáta sa prenášajú na pozadí cez **XMLHttpRequest** objekt. Niektoré princípy SPA:

- **Do DOM-u sa iba zapisuje**. Do DOM-u sa neukladajú dáta/stavy. Aplikácia pracuje s HTML kódom, pracuje nad elementami dokumentu, ale nikdy sa z DOM-u nenačítajú dáta, pretože ukladanie dát do DOMu sa rýchlo stane ťažko spravovateľné. Lepšie je mať jedno miesto, kde sú dáta uložené.
- **Dáta uložené v modeloch**. Namiesto ukladania dát v DOM-e alebo rôznych objektoch je lepšie použiť modely, ktoré reprezentujú dáta aplikácie.
- **Pohľady sledujú zmeny na modeloch**. Je výhodné využiť systém udalostí, pomocou ktorého sú pohľady oboznámené o zmene na modely. Viacero pohľadov môže byť oboznámených o jednej zmene modelu a podľa toho sa korektne vykresliť [30].

2.2.2 Backbone

Backbone je knižnica, ktorá by mala dať štruktúru celej aplikácii, resp. jej JavaScriptovej časti, s využitím návrhového vzoru MVC (Model View Controller).

S knižnicou Backbone sú dáta reprezentované ako **modely**, ktoré môžu byť vytvorené, validované, vymazané alebo odoslané na server. Kedykoľvek je nejaký atribút modelu zmenený z užívateľského rozhrania, tak model spustí udalosť *change* a ostatné pohľady (angl. **views**), ktoré zobrazujú daný model, môžu byť o danej zmene informované a teda sú schopné reagovať, napríklad sa môžu prekresliť a tak zobraziť novú informáciu[31]. Pohľad je HTML reprezentáciou časti aplikácie, napríklad jedného modelu. Dokáže reagovať na zmeny a adekvátne prekresliť svoj HTML kód.

Usporiadaný zoznam modelov sa v Backbone aplikácii nazýva kolekcia (angl. *collection*). Zvyčajne obsahuje jeden typ modelu, ale nemusí to tak byť.

Ďalšou dôležitou komponentovou Backbone aplikácie je smerovač (angl. *Router*). Slúži na smerovanie URL adries našej aplikácie a používa pritom mriežku #. Smerovač mapuje adresy za mriežkou na odpovedajúce metódy. Tak ako statické adresy, tak isto dokáže smerovať aj dynamické adresy, napríklad ak chceme získať článok s id 12, tak takáto URL by mohla vyzerať nasledovne: *http://example.com/#/posts/12* a definícia smerovača takto:

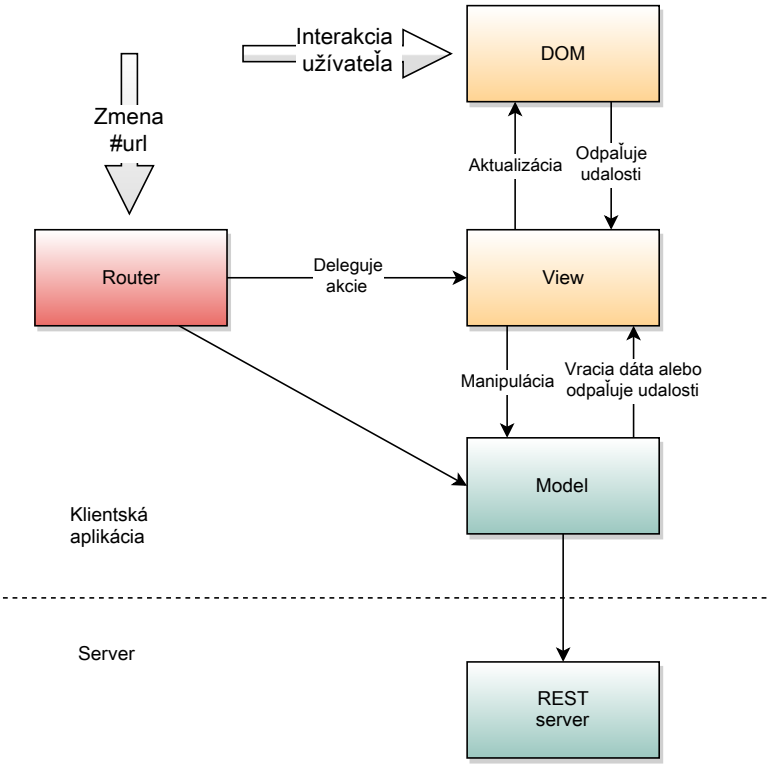
```
routes: {
  "posts/:id": "getPost",
}
```

Ukážka kódu 2.1: Definícia jednoduchého smerovača.

Backbone kolekcie a modely dovoľujú synchronizáciu dát so serverom. Pri synchronizácii sa využíva implicitne REST rozhranie a dáta sú prenášané pomocou XMLHttpRequest objektu a vo formáte JSON[32].

Marionette

Pri písaní väčšej aplikácie s využitím knižnice Backbone, dochádza často k opakovanému písaniu rovnakého kódu, čo môže viesť k chybám. Preto je vhodné v takomto prípade,



Obrázok 2.1: Architektúra typickej Backbone aplikácie

siahnuť po knižnici Marionette. Marionette je kolekcia komponentov pre Backbone, obsahujúca bežné návrhy a implementáciu vzorov, ktoré sú bežné pri písaní Backbone aplikácie.

Ako bolo spomínané, Backbone aplikácie vo veľkom množstve využívajú naväzovanie akcií pohľadov na udalosti modelov. Častým problémom, ktorý musia vývojári riešiť je odstraňovanie naviazaní pri odstránení pohľadu. Ak sa naviazanie neodstráni, tak objekt pohľadu ostáva v pamäti, čo môže spôsobiť rôzne problémy, ako napríklad *memory leak* alebo viacnásobné vyvolanie akcií.

Tento problém rieši Marionette manažmentom životného cyklu pohľadov. Okrem toho umožňuje:

- modulárnu udalosťami riadenú aplikáciu,
- redukovanie opakovania kódu,
- zoskupovanie pohľadov do Layoutov,
- zanorovanie pohľádov a Layoutov[33].

2.2.3 RequireJS

Ako sa aplikácia rozrastá, tak často narazíme na problém ako spravovať zdrojové súbory. Typické problémy sú:

- Pri načítavaní JavaScriptového súboru, ktorý je pripojený k HTML stránke, dochádza k blokovaniu prehliadača, kým sa nenačíta daný súbor.

- Ak sa do stránky vkladá veľa zdrojových súborov, tak rapídne klesá prehľad o závislostiach medzi modulmi (súbormi).

Tieto problémy umožňuje riešiť knižnica — *script loader* **RequireJS**, ktorá implementuje návrhový vzor AMD[34].

```
1 define(
2   ["foo", "bar"],
3   function(foo, bar {
4     // definicia modulu bude tu
5   }
6 );
```

Ukážka kódu 2.2: AMD modul závislý na moduloch *foo* a *bar*.

Asynchronous Module Definition (AMD)

AMD dovoľuje písať modulárne aplikácie v JavaScripte už dnes, nakoľko JavaScript také niečo v súčasnej dobe neumožňuje. AMD je formát ktorý umožňuje definovať moduly v aplikácii a okrem toho definuje ich závislosti na iné moduly.

AMD prináša dve podstatné výhody. Jednou je asynchrónne načítavanie súborov, kde AMD vie aké súbory má načítať, aby sa splnili závislosti, ale s načítaním počká až do okamihu, keď si ich aplikácia, prípadne modul vyžiada. Druhou výhodou je jednoduchšia správa závislostí modulov, kde modul na začiatku definuje moduly, na ktorých je závislý[35].

2.2.4 jQuery

jQuery je JavaScriptová knižnica, ktorá zjednodušuje interakciu medzi JavaScriptom a HTML kódom. Syntax jQuery je navrhnutá tak, aby zjednodušila navigáciu v dokumente, vyber DOM elementov, prácu s udalosťami, tvorbu animácii a AJAX-ovú komunikáciu so serverom. Ďalej umožňuje lepšie oddeliť chovanie od štruktúry HTML. Na manipuláciu s DOM elementami sa používa knižnica *Sizzle*. Veľkou výhodou tejto knižnice je, že kód napísaný v jQuery je kompatibilný naprieč majoritnými prehliadačmi (jQuery prekrýva rozdiely medzi prehliadačmi).

Knižnica jQuery sa používa dvoma spôsobmi:

- Cez funkciu `$`, čo je továrenská metóda pre jQuery objekt. Tieto metódy sa dajú reťaziť a vracajú jQuery objekt. Typicky sú tieto metódy volané cez CSS selectorový reťazec.
- Cez `$.` — predponové funkcie. Sú to pomocné funkcie, ktoré zväčša nevracajú jQuery objekt.

```
1 $("div.test").add("p.quote").addClass("blue").slideDown("slow");
```

Ukážka kódu 2.3: Príklad použitia jQuery.

Zepto

Zepto je odľahčená JavaScriptová knižnica pre moderné prehliadače s aplikačným rozhraním kompatibilným s jQuery. Cieľom vytvorenia tejto knižne bola 5-10kb modulárna knižnica, ktorá sa sťahuje a vykonáva rýchlo. Na rozdiel od jQuery, nerieši spätnú kompatibilitu u niektorých prehliadačov, no kompatibilita s mobilnými prehliadačmi je veľmi dobrá a je vhodná pre vývoj natívnych webevých aplikácii, napríklad pomocou frameworku **Phonegap**[36].

2.2.5 Underscore

Underscore je nástrojový pás funkcií pre JavaScript, ktorý poskytuje podporu pre funkcionálne programovanie bez toho, aby rozširoval JavaScriptové objekty. Táto knižnica poskytuje funkcie pre funkcionálny prístup pomocou funkcií **map**, **select**, **invoce** atď. Ďalej poskytuje aj iné špecializované pomocné funkcie ako je templatovanie v JavaScripte alebo hlboké porovnávanie objektov.

Tieto metódy sa volajú z objektu tejto knižnice, typicky nazvanej podržítkom _.

```
1 \_.each([1,2,3], alert);
```

Ukážka kódu 2.4: Volanie vypíše alert pre každé číslo.

Knižnica Underscore je nutnou závislosťou pre knižnicu Backbone[37].

2.2.6 XMLHttpRequest uroveň 2

XMLHttpRequest (ďalej iba XHR) na úrovni 2 ponúka oproti svojmu predchodcovi nové prvky, ktoré skvalitňujú vývoj webových aplikácií. Spomínanými prvkami sú:

- Funkcia *cross-origin requests* umožňuje AJAXové požiadavky medzi rôznymi doménami. Ak chceme pristupovať na inú doménu, tak na tej doméne musíte nastaviť hlavičku **Access-Control-Allow-Origin** vložením povolenej domény (z ktorej je možné pristupovať). Potom pristúpenie z povolenej domény, sa ničím nelíši od normálneho XHR požiadavku.
- Je vytvorená podpora pre monitorovanie priebehu nahrávania súboru.
- Asi najdôležitejšou súčasťou je schopnosť nahrávania a sťahovania binárnych súborov. Po novom je možné nastaviť XHR objektu vlastnosť **responseType** a priradiť jej typ prijímaných alebo odosielaných dát. Podporované hodnoty sú *text*, *arraybuffer*, *blob* alebo *document*[38].

```
1 function sendBinaryData(data) {
2   var xhr = new XMLHttpRequest();
3   xhr.open('POST', '/server', true);
4   xhr.responseType = 'blob';
5
6   xhr.onload = function(e) {
7     if (e.status == 200) {
8       console.log(this.response);
9     }
```

```
10    };
11
12    xhr.send(data);
13 }
```

Ukážka kódu 2.5: Odoslanie binárnych dát na server.

2.2.7 Využitie Google Maps API

Google maps je online mapová služba spoločnosti Google. Okrem webového rozhrania ponúka vývojárom aplikačno-programové rozhranie umožňujúce vkladanie interaktívnych mapových podkladov do vlastných stránok.

2.2.8 REST komunikačné rozhranie

REST *Representational State Transfer* je typ rozhrania, cez ktoré sa pristupuje k dátam. Rozhranie REST určuje akým spôsobom sa k týmto dátam pristupuje. REST definuje štyri základné metódy ako sa so zdrojmi pracuje. Tieto metódy sa súhrnne označujú **CRUD** (*create*, *retrieve*, *update* a *delete*) a sú implementované pomocou metód HTTP protokolu: **POST**, **GET**, **UPDATE** a **DELETE**[39]. Príklad použitia REST rozhrania je v sekcii 3.3.10.

2.3 PhoneGap

PhoneGap je opensource framework pre vytváranie natívnych medziplatformových mobilných aplikácii za požitia HTML, JavaScriptu a CSS.

Pri vytváraní aplikácii pre každú mobilnú platformu — Android, iPhone, Windows Mobile atď., sú potrebné rôzne jazyky a frameworky. PhoneGap používa existujúce webové technológie na premostenie webových aplikacii a mobilných zariadení. Užívateľské rozhranie Phonegap aplikácie tvorí internetový prehliadač, ktorý zaberá 100 % obrazovky mobilného zariadenia.

PhoneGap poskytuje programovo aplikačné rozhranie, ktoré dovoľuje prístup k natívnej funkcionalite operačného systému pomocou JavaScriptu. Teda logika aplikácie je napísaná v JavaScripte a PhoneGap zabezpečuje komunikáciu s operačným systémom zariadenia.

Výsledná aplikácia je nakoniec skompilovaná ako nativná aplikácia a ďalej je ju možné distribuovať cez ekosystém danej platformy (iTunes, Google Play, ...)[40].

2.3.1 Štruktúra aplikácie

Špecifická architektúra aplikácie sa samozrejme líši od prípadu k prípadu, no napriek tomu typická aplikácia závislá na dátach má takúto základnú architektúru: PhoneGap aplikácia sa správa k užívateľovi ako klient a komunikuje s aplikačným serverom pre získanie dát. Tento aplikačný server je normálny webový sever a používa serverový skriptovací jazyk (PHP, Python, Ruby ...). Aplikačný server zabezpečuje biznis logiku, prijíma a uchováva dáta v nejakom úložišti napríklad v relačnej databáze.

Komunikácia PhoneGap aplikácie a aplikačného serva môže prebiehať nad protokolom HTTP a prenos dát môže byť vykonaný vo formáte JSON či XML pomocou metódy REST.

Klientská PhoneGap aplikácia je typicky napísaná ako Single-page aplikácia. Stránka sa nikdy nevymazáva z pamäte a dáta sa zobrazujú zmenou HTML DOM-u[40].

Framework Phonegap dokáže pracovať na väčšine mobilných operačných systémov. Týmito systémami sú: iPhone, Android, Blackberry, WebOS, Windows Phone, Symbian a Bada[41].

2.3.2 Spolupráca s perifériami zariadenia

Framework Phonegap zlepšuje spoluprácu s perifériami zariadení a dovoľuje preklenúť rozdielnu podporu v jednotlivých mobilných zariadeniach. Ak daný prehliadač (aplikácia Phonegap beží v okne prehlidača) nepodporuje prácu s nejakým zariadením, cez JavaScriptové HTML5 programové rozhranie, tak sa využije implementácia Phonegapu[42]. Príklady spolupráce s perifériami:

Zaznamenávanie médií
Ak chceme z PhoneGap aplikácie pristupovať k nahrávacím možnostiam zariadenia, môžme využiť objekt `capture` priradený do objektu `navigator.device`. Objekt `capture` obsahuje metódy ako `captureAudio`, `captureImage` alebo `captureVideo`. Tieto metódy umožňujú asynchronné nahratie média pomocou základnej aplikácie na zaznamenávanie daného média[42].

Pohybový senzor
Programové rozhranie pre snímanie pohybu obsahuje objekt `accelerometer` v objekte `navigator` a obsahuje metódu `getCurrentAcceleration` na získanie aktuálnej akcelerácie a metódu `watchAcceleration` pre sledovanie akcelerácie v pravidelných intervaloch. Uvedené metódy v prípade úspechu vracajú objekt `Acceleration`, ktorý obsahuje dáta v špecifickom čase. Podobne ako bolo uvedené v sekcii 1.3.6, tak aj tu obsahuje dáta o akcelerácii pozdĺž jednotlivých osí, ale v tom to prípade, vždy s vplyvom gravitačného zrýchlenia[42].

Geolokačné rozhranie
Implementácia geolokačného rozhrania sa zhoduje s W3C implementáciou (sekcia 1.3.3). Pre zariadenia, ktoré nemajú podporu pre geolokáciu, Phonegap nahrádza W3C implementáciu vlastnou implementáciou[42].

Vibrácie
Aplikácie využívajúce framework PhoneGap majú možnosť upozorniť užívateľa na vznik udalosti aj vibrovaním. Metóda `vibrate` objektu `navigator.notification` vibruje zariadenie na špecifikovaný čas v milisekundách[42].

```
1 || navigator.notification.vibrate(2500);
```

Ukážka kódu 2.6: Spustenie vibrovania na 2.5 sekundy.

Kompas
Ďalšou informáciou, ktorú dokáže Phonegap zo zariadenia získať je smer, kam je zariadenie nasmerované. Túto funkcionalitu poskytuje objekt `compass` pomocou metód `getCurrentHeading` a `watchHeading`, ktoré asynchrónne vracajú dáta o smere v stupňoch zo senzora v rozsahu 0 až 359.99[42].

Kapitola 3

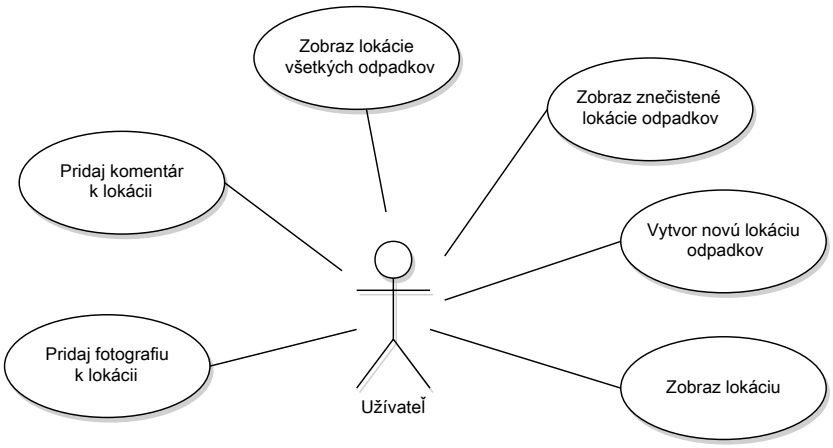
Návrh aplikácie a implementácia

Nasledujúca kapitola bude popisovať návrh a implementáciu aplikácie, ktorá využíva HTML5 technológie. Najprv bude priblížená špecifikácia a budú popísané základné funkcie. Potom sa prezradí spôsob implementácie aplikácie, jej komponent a komunikačný protokol. Zároveň bude predstavený návrh užívateľského rozhrania a postup pri jeho vytváraní. Nakoniec uvedieme, ako sme postupovali pri vytváraní hybridnej aplikácie s frameworkom Phonegap a čo sa zmenilo oproti pôvodnej aplikácií.

3.1 Špecifikácia aplikácie

Naša HTML5 aplikácia bude správca lokácii, na ktorých sa vyskytujú odpadky. Je určená pre zariadenia typu smartfón a tablet. Jej primárnym cieľom je udržovať informácie o ich polohe, pre smetiarov alebo dobrovoľníkov, ktorí ich budú chcieť vyzbierať. Každý užívateľ aplikácie má možnosť pridať novú lokáciu odpadkov, pridať k nej popis a prípadne odfotiť miesto. K lokácii sa dajú pripisovať komentáre a vyčistená lokácia sa dá označiť za vyzbieranú.

Aplikácia beží v okne prehliadača alebo sa inštaluje ako natívna aplikácia pre platformu android. Tieto dve aplikácie sú skoro zhodné, android aplikácia pridáva funkcionalitu navyše ako zistenie stavu batérie, ikonu aplikácie alebo úvodnú obrazovku — (*splash screen*). Funkcie, ktoré aplikácia ponúka sú znázornené na use case diagrame 3.1.

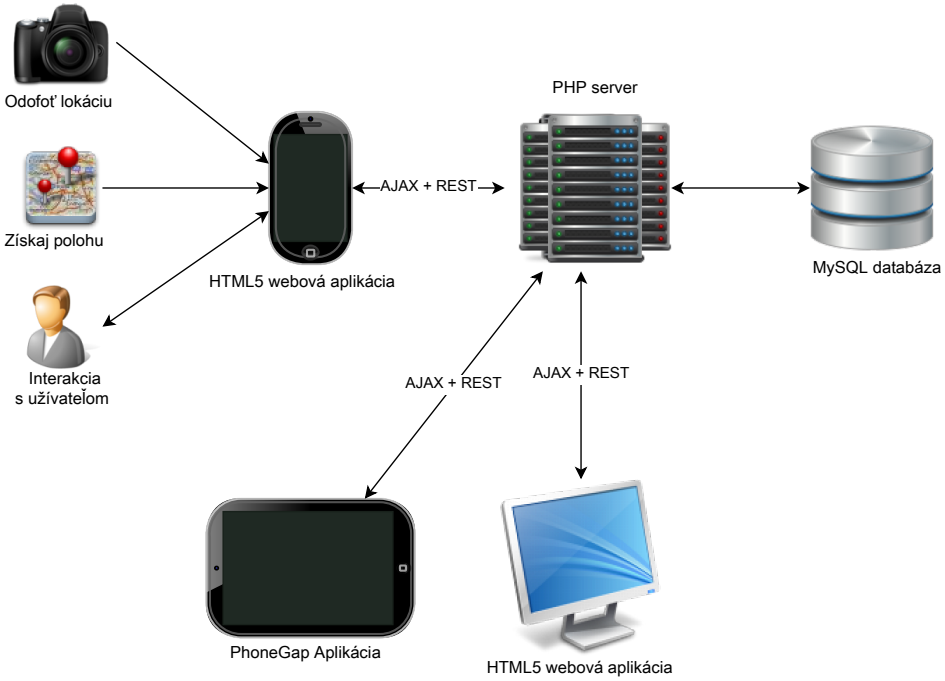


Obrázok 3.1: Use case diagram

3.2 Návrh aplikácie

3.2.1 Hlavné komponenty aplikácie

Hlavnou časťou bude **klientská aplikácia**, cez ktorú bude užívateľ pracovať. Klientská aplikácia komunikuje cez komunikačný protokol so **serverovou aplikáciou**. Tá sa stará o ukladanie, získavanie, upravovanie a mazanie dát. Tieto dáta sú trvalo uložené v **databáze**. Prepojenie komponent na najvyššej úrovni je zobrazené na obrázku 3.2.



Obrázok 3.2: Top level architektúra

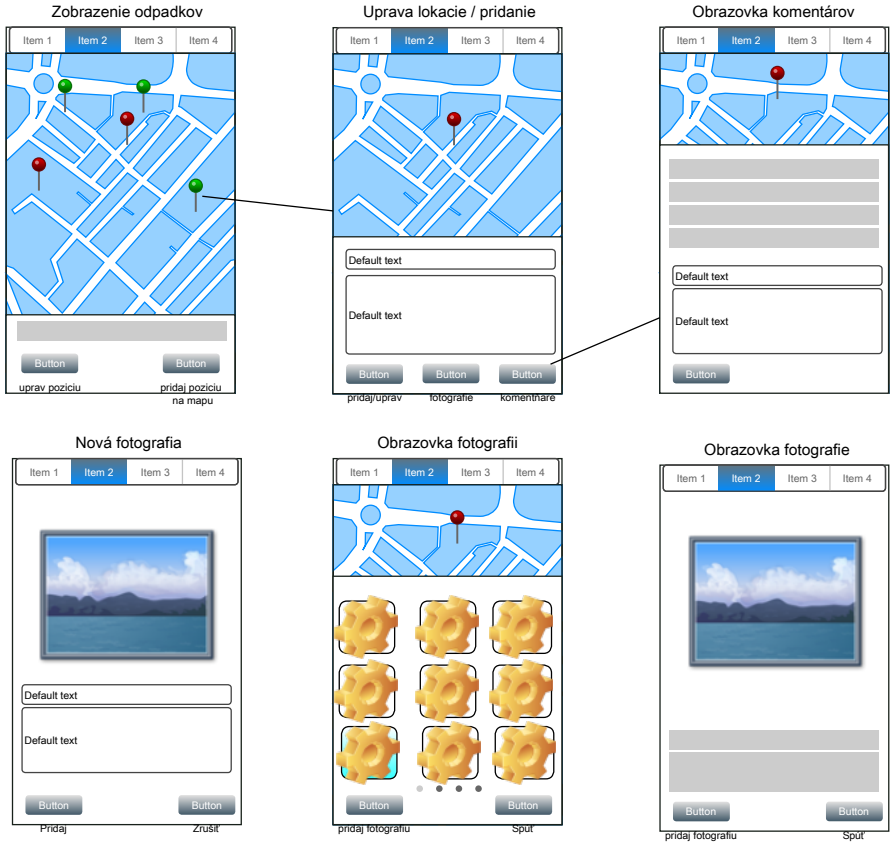
3.2.2 Návrh užívateľského rozhrania

Užívateľské rozhranie je zamerané na jednoduchosť a dostupnosť jednotlivých funkcií aplikácie, teda užívateľ by mal byť schopný pracovať s aplikáciu aj bez vysvetlenia.

Užívateľské rozhranie by sa malo prispôsobiť veľkosti obrazovky viď. návrhy obrazoviek pre mobil a tablet 3.3 a D.1. Rozhranie pre tablet by malo rozumne využiť zväčšený priestor, v našom prípade sa osamostatní menu s pridanou funkcionalitou, zmení sa rozloženie akčných tlačidiel pri zobrazenej lokácii. Veľkosť prvkov užívateľského rozhrania je podriadená cieľovým mobilným zariadeniam.

Aplikácia by mala reagovať na tlačítko späť v prehliadači mobilného zariadenia, poprípade na hardwarové tlačidlo, napr. v zariadeniach android. Po vykonaní spomínanej akcie by sa mala aplikácia vrátiť na predchádzajúcu obrazovku — stav.

Návrhy užívateľského rozhrania pre smartfón a tablet je vidieť na obrázkoch 3.3 a D.1 Ako možno pozorovať, tak mapa tvorí dominantný prvok na väčšine obrazoviek. Typy lokácii na mape sú farebne odlíšené. Vyzbierané lokácie sú vyznačené zelenou farbou a znečistené červenou. Tak isto aj pri zobrazení popisu lokácie sa farebne odlíšia. V pravom hornom rohu



Obrázok 3.3: Návrh užívateľského rozhrania pre užšie obrazovky.

sú stavové ikony, pre stav siete a stav batérie. Rozšírené menu pre tablet obsahuje doplnkové informácie o počte všetkých lokácii a o počte znečistených lokácii.

V sekcii *O aplikácii* bude stručný popis aplikácie a návod na použitie.

3.2.3 Popis fungovania webovej aplikácie

Ako bolo spomínané v 3.2.2, tak mapa tvorí dominantnú časť aplikácie na väčšine obrazoviek. Umožňuje zobrazenie lokácii a po kliknutí/dotyku na zvolenú lokáciu sa vykreslí jej detail spolu s tlačidlami na vykonanie akcii a dané miesto na mape sa vycentruje. Ďalšou funkciu ktorú umožňuje, je pripnutie špendlíka pri vytváraní novej lokácie. Samozrejme mapa podporuje bežné akcie ako priblíženie/oddialenie, či zmenu mapového podkladu.

Pri pridávaní novej lokácie, sa vyberá miesto sa mape pomocou špendlíka alebo sa použije aktuálna poloha poskytnutá zariadením. Okrem toho je potrebné zadať názov, prípadne vyplniť popis miesta.

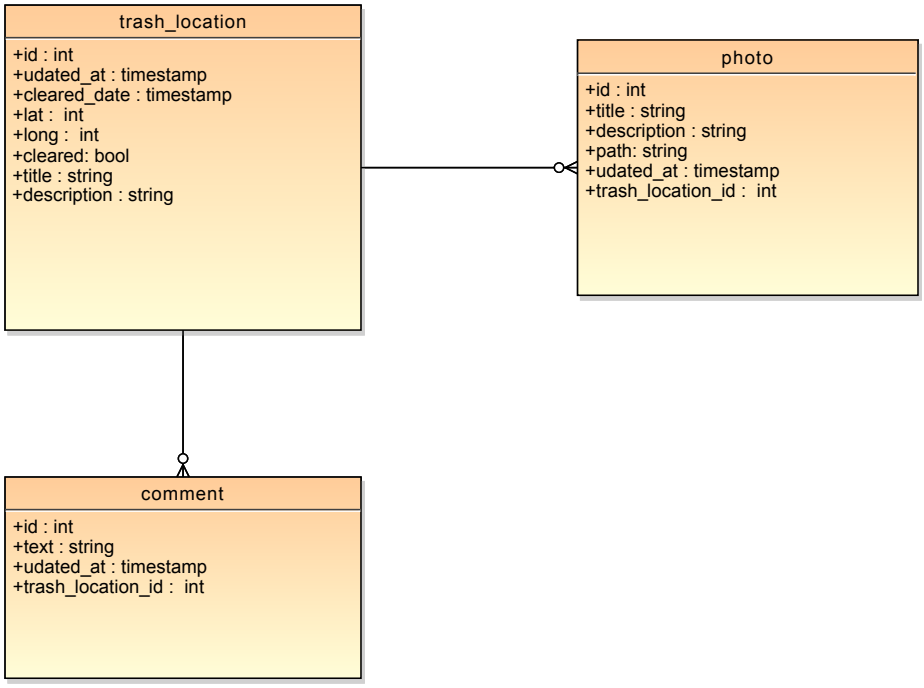
Každé miesto môže mať priradené fotografie. Fotografie sa zobrazujú v galérii a po výbere sa zobrazí nadhľad fotografie. Fotografie sa dajú nahrávať priamo z mobilného zariadenia. Ak to zariadenie podporuje, tak sa použije aplikácia zariadenia na fotografovanie, ak nie, tak sa vyberie fotografia zo súborového systému. Po vložení fotografie sa fotografia odosiela na server, kde sa zmenší na vhodnú veľkosť. Pridávanie fotografii je možné iba s pripojením k internetu.

Pri prvom načítaní aplikácie sa nahrajú všetky dáta zo serverovej aplikácie a potom sa

s týmito dátami pracuje. Prijaté dáta sú uložené v modeloch (dátové štruktúry programu). Na server sa odosielajú nové dáta, zmeny dát a informácie o mazaní dát.

3.2.4 Dátový model aplikácie

Aplikácia používa tri dátové entity: **lokácia odpadkov**, **komentár** a **fotografia**. Komentáre a fotografie odkazujú cudzími kľúčmi na lokácie. Jednotlivé entity a ich prepojenie sú zobrazené v ER diagrame 3.4. Mapovanie modelov v Backbone klientskej aplikácie a databázových tabuliek, ku ktorým pristupuje serverová aplikácia, odpovedá v pomere 1:1.



Obrázok 3.4: ER diagram aplikácie na lokalizáciu odpadkov

3.3 Implementácia jednotlivých funkcii

Nasledujúce sekcie bližšie predstavia implementáciu jednotlivých funkcií a komponent aplikácie, ktoré sú určitým spôsobom zaujímavé.

3.3.1 Serverová aplikácia

Serverová časť obsluhuje REST požiadavky a podáva dotazy do databázy. Je naprogramovaná v jazyku PHP a využíva framework **Codeigniter**. Výhodou tohto frameworku je, že má v sebe zabudovanú podporu pre pohodlné nahrávanie súborov (v našom prípade fotografií), zmenu veľkosti obrázkov[1], prístup k databáze pomocou funkcií ktoré implementujú návrhový vzor *Active record*[2] a iné.

[1] Pôvodne sme použili zmenšovanie na severy, ale teraz sa zmenšujú obrázky na strane klienta.

[2] Tento návrhový vzor pracuje s dátami v relačnej databáze. Active record ako taký, je objekt, ktorý obaľuje riadok tabuľky alebo pohľadu, zapúzdruje prístup do databázy a pridáva vlastnú logiku pre tieto dáta.

Framework Codeigniter nemá priamo zabudovanú podporu pre REST požiadavky, tak sme ju museli implementovať sami. Preto vznikol Controller[3] `MY_controller`, ktorý dokáže rozpoznať rôzne typy HTTP požiadavkov (GET, POST, PUT a DELETE) a tieto požiadavky predať nato určeným metódam. Tieto metódy pracujú s modelmi, ktoré sú inštanciami tried modelov, dediacich z triedy `MY_Model`. Z tried MY_Controller a MY_Model boli odvodené triedy `Loc, Photo, Comments` resp. `Location_model, Photo_model a Comment_model`. Názvy controlerov odpovedajú adresám v komunikačnom protokole 3.3.10.

Dôležitým prvkom, ktorý sme museli implementovať bola podpora pre CORS[4], pretože inak by prehliadač zariadenia zakázal pristupovať na server, ktorý nie je na rovnakej doméne ako stránka. To by sa mohlo stať v prípade Phonegap aplikácie, ktorá beží na inej doméne ako serverová aplikácia. Preto sa ku každej odpovede na požiadavok, ktorú odošleme zo serveru, pridá hlavička **Access-Control-Allow-Origin: ***.

3.3.2 Štruktúra klientskej aplikácie

Klientskú aplikáciu sme sa rozhodli implementovať v jazyku **JavaScript**, pretože ho dokáže interpretovať každý moderný prehliadač v mobilnom zariadení. Pre JavaScript existuje množstvo knižníc, ktoré umožňujú písanie rozsiahlejších aplikácii tým, že im dávajú štruktúru. V našom prípade sme použili knižnicu **Backbone** a jej nadstavbu **Marionette**.

S narastajúcou dĺžkou kódu nastáva potreba lepšie organizovať zdrojový kód. My sme sa rozhodli rozdeliť definície objektov, ktoré vznikajú použitím Backbone, do modulov. Preto voľba padla na knižnicu **Requirejs**.

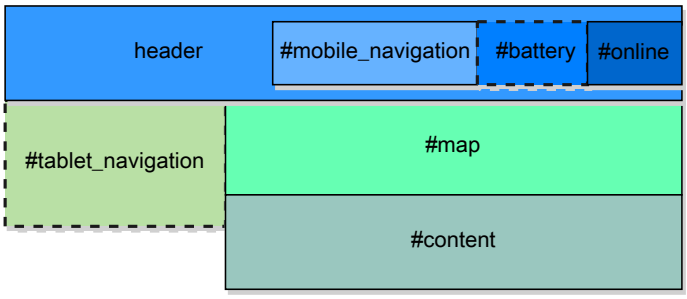
Pri implementácii sme sa snažili využiť komunikáciu medzi komponentami cez udalosti. Využili sme návrhový vzor **Mediator**, vďaka ktorému sa rieši napríklad komunikácia s objektom, ktorý pracuje s mapou. Ďalej je užitá podpora pre udalosti knižnice Backbone, ktorá pracuje s udalosťami nad užívateľským rozhraním a modelmi.

Jadrom aplikácie je objekt **router** — smerovač. Jeho úlohou je obsluhovať zmeny url a na základe zmien správne vykreslovať pohľady. Okrem toho obsahuje objekty kolekcií, ktoré predáva jednotlivým pohľadom. Ďalším dôležitým objektom je inštancia triedy `Marionette.Application` **GeoApp**, ktorá obstaráva inicializáciu aplikácie: vytvára regióny aplikácie, vytvára a spúšťa router, vytvára mediátora a pred-načíta lokácie odpadkov. Regióny aplikácie sú časti užívateľského rozhrania, ktoré sa počas behu aplikácie menia. Rozloženie regiónov je vyobrazené na obrázku 3.5:

- **header**, hlavička — je zaujímavá tým, že obsahuje zanorené regióny zoskupené v **layoute**. Týmito regiónmi sú: #**mobile_navigation**, #**battery** a #**online** — mobilná navigácia, stav batérie a stav pripojenia.
- #**content** je hlavné telo aplikácie. Router do tohto regiónu vykresluje pohľady.
- #**tablet_navigation** je menu ktoré sa zobrazí v prípade väčšej šírky obrazovky.
- #**map**.

[3] Codeigniter je MVC framework podobne ako Backbone, ale tu logiku aplikácie, zväčša obsahujú controlery, nie pohľady ako v Backbone.

[4] CORS je mechanizmus, ktorý dovoľuje webovej stránke urobiť XHR na inú doménu. Funguje tým spôsobom, že sa pridáva do odpovede nová hlavička, v ktorej sa uvedie povolená doména.



Obrázok 3.5: Názvy regiónov sú zároveň CSS selektory v DOM-e aplikácie.

Jednotlivé pohľady vykreslujú HTML kód, reagujú na interakciu užívateľa s užívateľským rozhraním alebo na zmeny modelov. HTML kód, ktorý vykreslia sa nachádza v súboroch — *templatoch*, ktoré okrem HTML kódu obsahujú aj templatovací jazyk z knižnice **Underscore**.

3.3.3 Pridanie fotografie

Jednou z požadovaných vlastností aplikácie je pridávanie fotografií miesta. V našom prípade sme použili formulárový prvok **input** s atribútom **accept="image/*"**. Tento prvok spustí aplikáciu pre prácu s fotoaparátom a v zariadeniach, ktoré takúto možnosť nemajú, dovolí nahratie fotografie zo súborového systému. Fotografie sa odosielajú na server cez XHR objekt.

Pre mobilné zariadenia je nevhodné, aby odosielali fotografie v plnom rozlíšení na server, pretože často používajú mobilné pripojenie k internetu. Preto sme sa rozhodli zapracovať funkcionalitu, ktorá dovolí zmenšenie fotografie na strane klienta. V klientskej aplikácii vznikol objektový typ, ktorý sa stará o zmenšenie obrázku — `CanvasResizer`. Inštancia tohoto typu dokáže nahrať do pamäti súbor obrázku zo súborového systému. Potom obrázok vloží do elementu `<canvas>` a nastaví mu požadovanú maximálnu výšku a šírku. Takýto zmenšený obrázok sa dá extrahovať pomocou metódy canvasu `toDataUrl()`, tým získame dáta vo formáte **BASE64**. Pred odoslaním na server cez XHR však potrebujeme dáta vo formáte **blob**. O konverziu sa stará metóda `dataURLtoBlob`. Okrem zmenšovania obrázkov, sa CanvasResizer stará aj o vykreslenie zmenšeného obrázka užívateľovi — ako jeden z parametrov konštruktoru sa mu zadá CSS selector elementu, do ktorého sa má obrázok vykresliť.

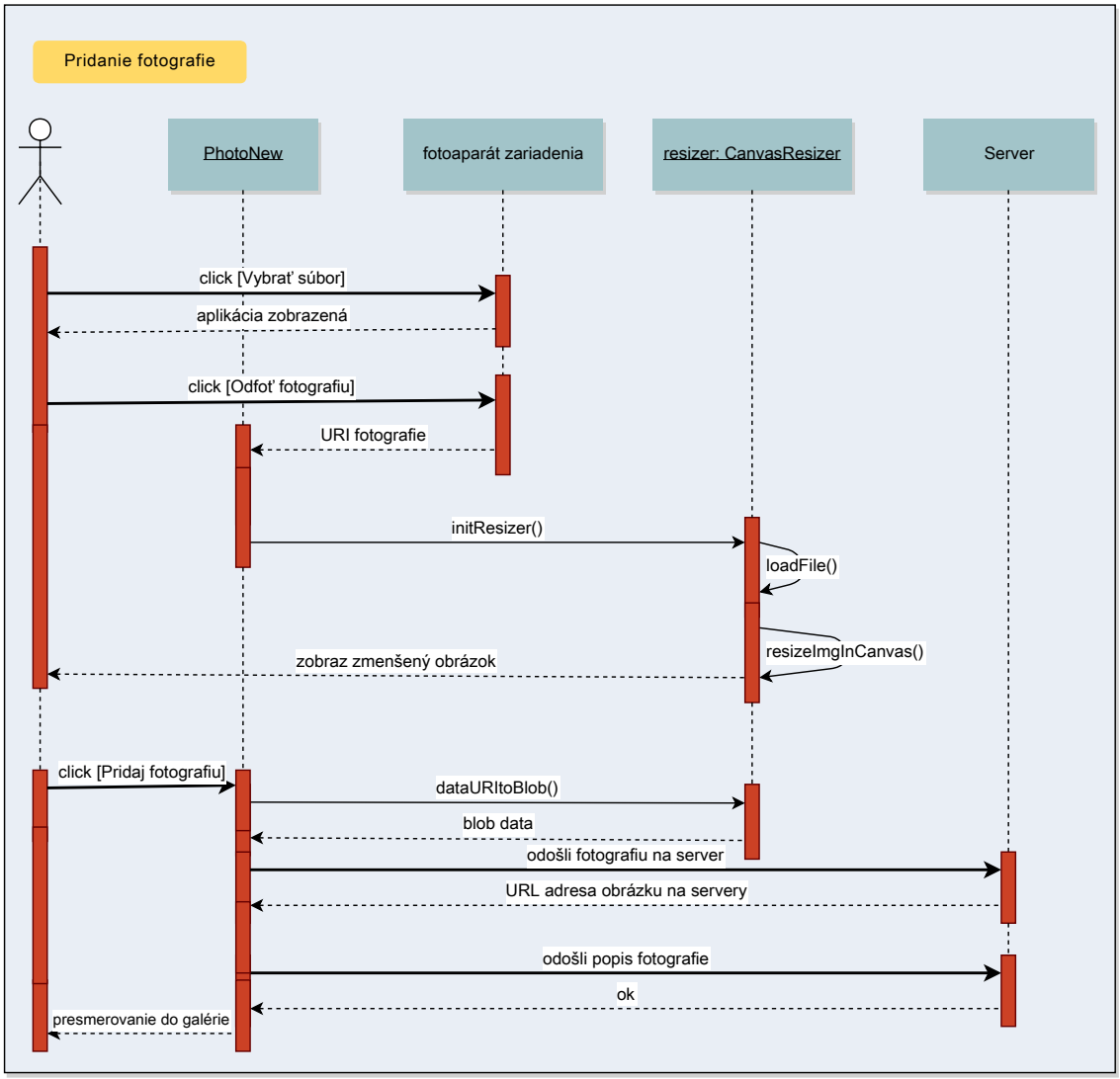
Pre lepšiu ilustráciu ako pridávanie obrázkov funguje, si môžte prezrieť obrázok 3.6.

3.3.4 Práca s mapou

Zobrazenie mapy zabezpečuje knižnica **Google Maps API v3**, ktorá sa sťahuje asynchrónne pri spustení aplikácie.

Od spustenia aplikácie je vytvorená jedna inštancia máp, ktorá je obalená do pohľadu **Map**. Jednotlivé pohľady a router komunikujú s pohľadom mapy nepriamo cez objekt **mediatora**. Tak isto aj mapa upozorňuje na svoje udalosti ostatné časti aplikácie cez tento objekt.

Pohľad je možné podľa potreby skrývať/zobrazovať, napríklad pri pridávaní novej fotografie k polohe, sa mapa nezobrazuje. Pred zobrazením pohľadu pre pridanie novej fotografie, router vyvolá udalosť na mediatorovi *map:hide* a pohľad mapy túto udalosť zachytí a skryje sa.



Obrázok 3.6: Obrázok zobrazuje sekvenčný diagram pridania fotografie. Je znázornená interakcia užívateľa so zariadením, keď najprv užívateľ vyberá fotografiu (spustí sa aplikácia fotoaparátu) a potom, keď je fotografia vytvorená, odosiela ju na server kliknutím na *Pridaj fotografiu*. PhotoNew je typ objektu pohľadu, ktorý sa stará o vykreslenie užívateľského rozhrania pre pridanie fotografie, interakciu s užívateľom, aplikačnú logiku a komunikáciu so serverom.

Keď sa vytvára nová poloha odpadkov, tak pohľad mapy slúži na pripnutie špendlíka na znečistené miesto na mape. Okrem toho je schopný presunúť špendlík na aktuálnu polohu zariadenia pomocou geolokačného rozhrania.

Pohľad mapy umožňuje zobrazenie všetkých polôh odpadkov na mape. Jednotlivé polohy, znečistené a vyzbierané, sú farebne odlíšené. Pohľad obsahuje pole, ktoré uchováva zobrazené lokácie. Jednotlivé prvky poľa sú objektmi typu `google.maps.Marker`. Na tieto objekty sa viažu poslucháči udaloti *click*, ktorý vyvolá na mediatorovi udalosť *map:position-clicked*. Táto udalosť spôsobí, že sa zobrazí detail danej lokácie. Pred zobrazením polôh na mape, je ich možné filtrovať podľa toho, či sú vyzbierané alebo nie. Ak aplikácia zobrazuje detail polohy, tak mape sa vyobrazí iba daná poloha.

3.3.5 Zobrazenie stavu pripojenia k internetu

V pravom hornom rohu je stavová ikona, ktorá zobrazuje, či je zariadené pripojené k internetu. Na prvotné vykreslenie používa hodnotu premennej `onLine` objektu `navigator`. Získanie spomínanej hodnoty je zaobalené do funkcie `areWeOnline()`. Pri neskoršej zmene stavu pripojenia sa zachytávajú udalosti online a offline, a následne sa znova prekresľuje stavová ikona.

3.3.6 Zobrazenie stavu batérie zariadenia

Ak prehliadač podporuje rozhranie pre poskytnutie stavu batérie, tak sa vedľa ikony stavu pripojenia zobrazí jej stav, ktorý pozostáva z percentuálnej hodnoty a stavovej ikony. Okrem toho pohľad, ktorý sa stará a vykreslenie stavu, zobrazuje aj výstražnú hlášku pri poklese stavu batérie pod kritickú hodnotu[5].

V súčastnosti, rozhranie pre prácu s mapu nefunguje v žiadnom mobilnom prehliadači. Stav siete sa zobrazí iba v našej hybridnej aplikácii, pretože framework Phonegap takéto rozhranie poskytuje.

3.3.7 Vykreslenie navigácie

Keďže aplikácia obsahuje dve navigácie, mobilnú a tabletovú, a tie zobrazujú rovnaké položky, tak sme sa im rozhodli vytvoriť spoločný pohľad ako predka. Tento pohľad sa nazýva `Navigation` a z konfiguračného súboru **config.js** načítava pole položiek do menu, čo sú vlastne dvojice názov a adresa. Týmto spôsobom sa dá jednoducho pridať nová položka do menu bez toho, aby sa musel meniť kód pohľadov a šablón.

Navigácie sa líšia spôsobom, akým vykresľujú svoje položky, zatiaľ čo tabletová navigácia sa vykresľuje do nečíslovaného zoznamu, tak mobilná sa vykresľuje do formulárového prvku `<select>`. Tabletová navigácia zobrazuje počet všetkých a vyzbieraných lokácií. Ak nastane zmena v kolekcií lokácií, tak prekreslí ich počty.

3.3.8 Prispôsobenie šírke obrazovky

Zmena užívateľského rozhrania sa vykonáva, ak šírka obrazovky prekročí hranicu 800px. Pri prekročení tejto hranice, sa zobrazí rozšírené menu pre tablety a skryje sa menu pre mobily v hlavičke. Ďalej sa preskupia tlačítka s akciami lokácie — pri šírke menšej ako 800px sú tlačítka pod sebou, pri väčšej sú vedľa seba.

[5]Úroveň kritickej hodnoty sa líši od typu prehliadača a jej prekročenie je oznámené udalosťou **battery-critical**.

Na dosiahnutie tejto funkcionality sme použili **CSS3 media queries** pre minimálnu šírku zariadenia.[6]

3.3.9 Manipulácia s DOM

K manipulácii s DOM-om je využitá knižnica **jQuery**. Manipulácia s DOM-om je ale minimalizovaná použitím knižnice Backbone, ktorá pracuje s jQuery interne. Tým sa sprehľadnil zdrojový kód a znížil sa počet callbackov, ktoré sú typické pri používaní jQuery.

Pôvodne sme chceli použiť knižnicu **Zepto** ako náhradu za jQuery, pre jej zameranie na moderné prehliadače, ale nakoniec sme pri testovaní zistili, že Zepto nepodporuje prehliadače **Internet Explorer**.

3.3.10 Komunikačný protokol

Komunikáciu medzi klientskou aplikáciou a serverovou aplikáciou sme sa rozhodoli implementovať cez REST rozhraniem pretože knižnica Backbone má preň výbornú podporu. Vo svojej implementácii komunikuje so serverom práve týmto spôsobom. REST rozhranie nám dovoľuje vykonať všetky potrebné požiadavky na server:

- stiahnutie všetkých lokácií, komentárov a informácií o fotografiách,
- uloženie novej lokácie, komentáru a informácie o fotografií,
- označenie lokácie za vyzbieranú/znečistenú,
- vymazanie lokácie.

Komunikačný protokol pre výmenu dát o lokácií je znázornený v tabuľke 3.1.

Zdroj	**GET**	**PUT**	**POST**	**DELETE**
/index.php/loc/	**Získa** zoznam všetkých lokácii.	-	**Vytvorí** novú lokáciu. ID novej lokácie je pridané automaticky a vrátené naspäť.	-
/index.php/loc/ID	-	**Upraví** danú lokáciu. Napríklad ju označí ako vyzbieranú.	-	**Vymaže** danú lokáciu.

Tabuľka 3.1: Príklad REST rozhrania pre manipuláciu s informáciami o lokáciach. Ďalšie REST rozhrania pre komentáre a informácie o fotografiách vyzerajú analogicky.

Odoslanie fotografie prebieha trochu iným spôsobom. Po kliknutí na *"Pridať fotografiu"* sa ako prvé, asynchrónne odošle súbor fotografie na server pomocou XHR objektu. V prípade úspechu, je navrátená adresa uloženia na servery. Potom sa už postupuje normálnym spôsobom, teda informácie o fotografií sa odošlú cez REST rozhranie na server.

[6] CSS3 media queries sú pravidlá v CSS súbore, ktoré sa uplatňujú na základe istých vlastností zariadenia. Typicky minimálnej, maximálnej šírky zariadenia alebo orientácie.

Formát prenášaných dát

Po úvahe sme sa rozhodli kódovať prenášané dáta do formátu JSON. Dôvodom bolo to, že Backbone implicitne komunikuje cez tento formát a na strane servovej aplikácie je v použitom frameworku tiež dobrá podpora pre zakódovanie a rozkódovanie JSON formátu z a do asociatívneho poľa jazyka PHP.

3.3.11 Optimalizácia

I keď dnešné mobily a tablety sú na tom výkonnostne dobre, stále je vhodné optimalizovať, napríklad zlepšenie odozvy, rýchlosť prvotného spustenia, či množstvo prenesených dát.

V našej aplikácií sme zaviedli niekoľko optimalizácií.Napríklad užívateľské rozhranie obsahuje niekoľko ikoniek, ktoré by sa normálne načítali vlastným HTTP požiadavkom. My sme však tie obrázky zakódovali ako URI dáta[7] a tieto dáta vložili do CSS súboru.

3.3.12 Použitie frameworku Phonegap

Okrem webovej aplikácie, ktorá je dostupná cez internetový prehliadač, sme sa rozhodli vytvoriť vytvoriť aplikáciu pre platformu **Android**. Jej základ tvorí naša webová aplikácia, ale navyše má tie možnosti:

- Aplikácia teraz zobrazuje stav batérie. Je zobrazená stavová ikona a okrem toho je zobrazený stav v percentách.
- Po nainštalovaní sa vytvorí ikona, cez ktorú sa aplikácia spúšťa.
- Počas inicializácie sa zobrazí úvodná obrazovka – *splashscreen*.
- Veľkou výhodou je, že aplikácia by sa dala distribuovať cez obchod aplikácii platformi Android — **Google Play**.

Vývoj cez Phonegap sa líši od vývoja webovej stránky a natívnej aplikácie. Do značnej miery sú obmedzené možnosti pre ladenie aplikácie — *debuggovanie*. Zatiaľ čo vo webovej stránke sa dá sledovať vykonávanie javascriptu cez plugin v prehliadači, napr. **firebug** vo Firefoxe alebo v natívnej aplikácii priamo vo vývojovom rozhraní, tak Phonegap takúto možnosť priamo neponúka.

Jednou možnosťou, ktorá sa dá využiť pre vývoj takejto hybridnej aplikácie je vzdialené debuggovanie. Do tela stránky sa vloží skript, ktorý komunikuje so vzdialeným serverom a ten vykresľuje, i keď značne obmedzené, ladiace informácie. Príkladom takejto služby je **http://debug.phonegap.com/**.

Ďalším spôsobom je vyvíjať takúto aplikáciu v prehliadači tak, ako iné webové stránky. Trik spočíva v tom, že sa pred ladením odstránia všetky závislosti, ktoré má aplikácia na frameworku Phonegap. Základný princíp je, že sa všetky Phonegap volania nahradia nahradia niečím iným. Týmto sa oddiali ladenie na zariadení a spohodlní vývoj.

Aplikácia bola primárne vyvíjaná na Android verziu 4.0 a vyššie. Pre testovanie funkcionality sa použil emulátor systému android, neskôr aj reálne zariadenia.

[7]URI dáta sú podporované väčšinou moderných prehliadačov a sú užitočným spôsobom ako zakódovať malé obrázky do URL. Malou nevýhodou je, že URI dáta môžu zväčšiť veľkosť pôvodných dát.

Pri písaní Phonegap verzie aplikácie musela byť zavedená globálna premenná `PHONEGAP_APP` určujúca, či sa vykonáva webová aplikácia alebo Phonegap aplikácia. Na základe toho sa vetvia niektoré časti programu. Vo funckií `areWeOnline` sa pristupuje k premennej `navigator.connection.type` namiesto `navigator.onLine`, pretože sme zistili jej nefunkčnosť v Phonegap aplikácií.

Kapitola 4

Testovanie a vyhodnotenie

V nasledujúcej kapitole sa zameriame na testovanie a vyhodnotenie webovej aplikácie. Použijú sa testy, ktoré overia funkčnosť aplikácie na rôznych typoch mobilných zariadení.

4.1 Metodika testovania

Implementovaná aplikácia bude testovaná na niekoľkých mobilných zariadeniach, ktoré sa môžu líšiť platformou, jej verziou, použitým internetovým prehliadačom, ponukou periférii či veľkosťou obrazovky. Základné rozdelenie je podľa typu zariadenia — **smartfón** a **tablet**. Testované zariadenia možno vidieť v tabuľke 4.1.

Množina testov (4.2) vychádza z prípadov užitia 3.1, zo špecifikácie 3.1 a z grafického návrhu. Testy sa robili v základnom prehliadači daného zariadenia.

4.2 Testované zariadenia

Všetky testované zariadenia disponujú fotoaparátom, GPS modulom a pohybovým senzorom.

Pri prvom testovaní obstáli zariadenia s operačným systémom Android a iPhone veľmi dobre, no na systéme Windows aplikácia nefungovala vôbec. Chyba spočívala v použitej knižnici Zepto, ktorá je nekompatibilná s prehliadačmi Internet Explorer. Preto došlo k náhrade Zepto za jQurey 2.0 a tým k vyriešeniu problému zariadenia ASUS VivoTab RT (Windows 8 RT). Potom nasledovalo opakované testovanie.

Názov	**Platforma**	**Prehliadač**	**typ zariadenia**
Google Nexus 7	Android 4.2.2	Google Chrome (webkit)	tablet
LG Optimus One	Android 2.3.3	Android prehliadač (webkit)	smartfón
Lenovo ThinkPad	Android 4.0	Android prehliadač (webkit)	tablet
ASUS VivoTab RT	Windows 8 RT	Internet Explorer 10 (trident)	tablet
Nokia Lumia 800	Windows 7.5	Internet Explorer 9 (trident)	smartfón
iPhone 5	IOS 6	Safari (webkit)	smartfón
Samsung Galaxy S II	Android 4.0	Android prehliadač (webkit)	smartfón

Tabuľka 4.1: Výpis testovaných zariadení. V zátvorke pri názve prehliadača je uvedené vykreslovacie jadro.

Názov funkcie	Nexus 7	Lenovo Thinkpad	iPhone 5	LG Optimus One	Samsung S 2	ASUS VivoTab RT	Nokia Lumia 800
Zobrazenie všetkých lokácii	X	X	X	X	X	X	-
Zobrazenie znečistených lokácii	X	X	X	X	X	X	-
Označenie lokácie za vyzbieranú	X	X	X	X	X	X	-
Pridanie novej lokácie	X	X	X	X	X	X	-
Pridanie komentáru	X	X	X	X	X	X	-
Pridanie fotografie	X	X	X	-	X	X	-
Práca offline – zmena stavovej ikony	X	X	X	X	X	X	-
Práca offline – zakázanie fotografovania	X	X	X	X	X	X	-
Zmena užív. rozhrania podľa šírky obrazovky	X	X	X	X	X	X	-
Zobrazenie fotografie/fotografií	X	X	X	X	X	X	-
Konzistentnosť užívateľského rozhrania	X	X	X	X	X	X	-
Geolokácia	X	X	X	X	X	X	-
Rýchlosť štartu v sekundách	5	6	4	9	11	6	-

Tabuľka 4.2: Výsledky testov zariadení. V stĺpcoch sú názvy zariadení a v riadkoch testovaná funkcia. Časy sú len orientačné, zaokrúhlené na celé sekundy.

4.3 Výsledky testov

Testované zariadenia sa vysporiadali s testmi vcelku dobre. Malé odchýlky sa vyskytli pri vykresľovaní užívateľského rozhrania. Príkladom je posunuté tabletové menu na zariadení Lenovo Thinkpad. Problémy boli aj na smartfóne LG Optimus One. Toto zariadenie malo najmenšiu obrazovku zo všetkých testovaných (320x480px) a v dôsledku toho došlo k čiastočnému zakrývaniu menu a ťažšej práci na šírku, pretože mapa zakrývala celú obrazovku. Vcelku dobre dopadlo aj zariadenie s prehliadačom Internet Explorer 10, ktorý na rozdiel od predchodcov má oveľa lepšiu podporu pre moderné webové technológie. Pri teste na zariadení Nokia Lumia 800, sa aplikáciu nepodarilo vôbec spustiť, čo je pravdepodobne dôsledok toho, že používa Internet Explorer 9 a jeho nekompatility s HTML5 a pridruženými technológiami, viď 1.1.

V našom teste sa vyskytli zariadenia s prehliadačmi s vykresľovacím jadrom **trident** a **webkit**. Jadro webkit dosiahlo veľmi dobré výsledky a u jadra trident úspech závisel na verzií. Výsledky testov sú do značnej miery závislé na type prehliadača a jeho podpore moderných webových technológii.

4.4 Testovanie Phonegap aplikácie

Phonegap aplikácia bola testovaná iba na platforme Android, na zariadeniach LG Optimus One a Nexus 7. Množina testov obsahovala testy z tabuľky 4.2 a navyše boli pridané nasledovné testy:

- zobrazenie stavu batérie,
- zobrazenie úvodnej obrazovky.

V oboch testoch zariadenia obstáli. V pôvodných testoch dopadli rovnako ako je uvedené v sekcií 4.3, jedine s tým rozdielom, že začalo fungovať fotografovanie miest odpadkov na zariadení LG Optimus One, pretože sme použili rozhranie PhoneGap na spustenie aplikácie fotoaparátu namiesto prvku `<input type='file'>`.

4.5 Zhodnotenie testov

Vytvoriť aplikáciu, ktorá by fungovala rovnako na rozdielnych typoch mobilných zariadení nie je jednoduché. Vývoj komplikuje rozdielna podpora moderných webových technológii a preto je potrebné sa rozhodnúť, na ktoré zariadenia zameriame vývoj a tomu podriadiť výber technológii.

Testy ukázali, že súčasné mobilné zariadenia, či sa už jedná o tablety alebo smartfóny, si veľmi dobre poradili s modernými prvkami, ktoré boli implementované do aplikácie. Predpokladám, že v budúcnosti sa bude podpora pre moderné technológie v prehliadačoch ešte zlepšovať.

Zároveň bola preverená funkčnosť aplikácie, ktorá využíva framework Phonegap. S jeho využitím, aplikácie môžu konkurovať natívnym aplikáciam pre mobilne platformy.

Záver

V mojej práci ste sa dozvedeli o nových prvkoch v HTML — ukázal som nové značky, aplikačno-programové rozhrania a príklady ich použitia ste si mohli prezrieť a vyskúšať na priloženom CD.

Neskôr som ukázal možný výber návrhu a knižníc na vytvorenie mobilnej aplikácie — tento postup samozrejme nie je jediný možný, dajú sa použiť aj iné alternatívy. Predstavil som napríklad knižnicu Backbone, ktorá pomáha vytvárať lepšiu štruktúru aplikácie, knižnicu Requirejs pre modularitu kódu alebo návrhový vzor SPA pre plynulejšie a pohodlnejšie ovládanie aplikácie.

Motiváciou pre písanie webových mobilných aplikácii je ich prenositeľnosť a to, že sa na vývoj používajú známe technológie — HTML, CSS a JavaScript. Natívnu aplikáciu by som odporučil v prípade, že má aplikácia vyššie požiadavky na výkon alebo chceme zachovať základný vzhľad prvkov užívateľského prostredia danej platformy.

Pre demonštráciu spomínaného prístupu bola vytvorená aplikácia umožňujúca značkovanie miest na mape. Aplikácia používa interaktívnu mapu, niektoré HTML5 prvky uvedené v kapitole 1 a postupy z kapitoly 2. Pre overenie funkčnosti som aplikáciu otestoval pomocou užívateľských testov, ktoré preverili jednotlivé funkcie aplikácie. Testovanie bolo prevedené na štyroch mobilných telefónoch (smartfónoch) a troch tabletoch.

Pri vytváraní mobilnej aplikácie, je vhodné sa najprv zamyslieť nad cieľmi aplikácie a výberom cieľových platforiem. Ak plánujeme vyvíjať pre viac platforiem, tak stojí za zváženie, či nevytvoriť jednu jednu webovú aplikáciu namiesto niekoľkých natívnych. Samozrejme je tu ešte cesta hybridnej aplikácie, ktorá spája dva prístupy — pridáva webovej aplikácií výhody natívnej. V kapitole 3 som takto navyše vytvoril aplikáciu pre platformu android, ktorá vychádza z pôvodnej HTML5 aplikácie.

Aplikácia by sa dala rozšíriť rôznymi spôsobmi. Bolo by vhodné zapracovať autentizáciu uživateľov, aby mohli obsah upravovať iba prihlásený užívatelia. Pekným rozšírením by bolo pridanie prechodových efektov pri prepínaní obrazoviek, ktoré využívajú natívne aplikácie. Takúto funkcionalitu by nám poskytlo CSS3. Ďalej by bolo dobré zapracovať, aby aplikácia fungovala kompletne offline. Napríklad by boli mapové podklady načítané vopred, zdrojové súbory by boli uložené lokálne s využitím súboru manifest, dáta by sa ukladali pomocou lokálneho úložiska alebo by sa fotografie sťahovali od lokálneho súborového systému.

Literatúra

[1] World Wide Web Consortium. Html5 - w3c candidate recommendation. www.w3.org/TR/html5/introduction.html, 2012. [cit. 2012-12-17].

[2] HTML living standart. Html living standart. http://www.whatwg.org/specs/web-apps/current-work/multipage/introduction.html, 2013. [cit. 2013-3-11].

[3] Jan Sládek. Webdesignérův průvodce po html5 – nová sémantika. http://www.zdrojak.cz/clanky/webdesigneruv-pruvodce-po-html5-nova-semantika/, 2013. [cit. 2013-3-22].

[4] w3schools. Html doctype declaration. http://www.w3schools.com/tags/tag_doctype.asp, 2013. [cit. 2013-3-25].

[5] Bruce Lawson. The section element. http://html5doctor.com/the-section-element/, 2013. [cit. 2013-3-22].

[6] Tom Leadbetter. The article element. http://html5doctor.com/the-article-element/, 2013. [cit. 2013-3-23].

[7] w3. A vocabulary and associated apis for html and xhtml. http://www.w3.org/TR/html5/sections.html#the-article-element, 2013. [cit. 2013-3-23].

[8] Bruce Lawson. The time element (and microformats). http://html5doctor.com/the-time-element/, 2013. [cit. 2013-3-23].

[9] Bruce Lawson. The best of <time>s. http://www.brucelawson.co.uk/2012/best-of-time/, 2013. [cit. 2013-3-23].

[10] Mike Robinson. Aside revisited. http://html5doctor.com/aside-revisited/, 2013. [cit. 2013-3-27].

[11] Tom Leadbetter. Semantic navigation with the nav element. http://html5doctor.com/nav-element/, 2013. [cit. 2013-3-27].

[12] Richard Clark. The header element. http://html5doctor.com/the-header-element/, 2013. [cit. 2013-3-28].

[13] Jack Osborne. The footer element update. http://html5doctor.com/the-footer-element-update/, 2013. [cit. 2013-3-27].

[14] Richard Clark. The figure & figcaption elements. http://html5doctor.com/the-figure-figcaption-elements/, 2013. [cit. 2013-3-25].

[15] Tom Leadbetter. The video element. http://html5doctor.com/the-video-element/, 2013. [cit. 2013-3-25].

[16] Html: The markup language (an html language reference). http://dev.w3.org/html5/markup/b.html, 2013. [cit. 2013-3-27].

[17] w3schools. The hgroup element. http://www.w3schools.com/html/html5_webstorage.asp, 2013. [cit. 2013-3-23].

[18] Boris Smus. Multi-touch web development. http://www.html5rocks.com/en/mobile/touch/, 2013. [cit. 2013-3-29].

[19] W3C. Touch events version 1. https://dvcs.w3.org/hg/webevents/raw-file/tip/touchevents.html, 2013. [cit. 2013-3-29].

[20] W3C. Geolocation api specification. http://dev.w3.org/geo/api/spec-source.html, 2013. [cit. 2013-3-29].

[21] Jiří Šťastný. Html5 - geolokacni rozhrani. http://programujte.com/clanek/2011052400-html5-geolokacni-rozhrani/, 2013. [cit. 2013-3-29].

[22] Martin Hassman. Martin hassman. http://www.zdrojak.cz/clanky/zaciname-z-html5-canvasem/, 2013. [cit. 2013-3-23].

[23] W3C. Html media capture. http://www.w3.org/TR/2011/WD-html-media-capture-20110414/#captureparam, 2013. [cit. 2013-4-2].

[24] Bruce Lawson. It's curtains for marital strife thanks to getusermedia. http://html5doctor.com/getusermedia/, 2013. [cit. 2013-3-29].

[25] W3C. Deviceorientation event specification. http://dev.w3.org/geo/api/spec-source-orientation, 2013. [cit. 2013-4-2].

[26] Pete LePage. This end up: Using device orientation. http://www.html5rocks.com/en/tutorials/device/orientation/, 2013. [cit. 2013-4-3].

[27] W3C. Battery status api. http://www.w3.org/TR/battery-status/, 2013. [cit. 2013-4-3].

[28] MARK PILGRIM. Dive into html5. http://kniha.html5.cz/, 2013. [cit. 2013-4-3].

[29] Ngu Phuc Huy and Do vanThanh. Evaluation of mobile app paradigms. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*, MoMM '12, pages 25–30, New York, NY, USA, 2012. ACM.

[30] Mikito Takada. Single page apps in depth. http://singlepageappbook.com/goal.html, 2013. [cit. 2013-4-3].

[31] backbone. Backbone.js. http://backbonejs.org/, 2013. [cit. 2013-4-3].

[32] Thomas Davis. What is a router? http://backbonetutorials.com/what-is-a-router/, 2013. [cit. 2013-4-4].

[33] marionette. Marionette.js. http://marionettejs.com/, 2013. [cit. 2013-4-4].

[34] requirejs. Require.js. http://requirejs.org/, 2013. [cit. 2013-4-4].

[35] adam. Javascript modulárne s amd & requirejs. http://www.sven.sk/blog/javascript-modularne-s-amd-requirejs, 2013. [cit. 2013-4-4].

[36] zepto. Zepto.js. http://zeptojs.com/, 2013. [cit. 2013-4-5].

[37] underscore.js. underscore.js. http://underscorejs.org/, 2013. [cit. 2013-4-5].

[38] Eric Bidelman. New tricks in xmlhttprequest2. http://www.html5rocks.com/en/tutorials/file/xhr2/, 2013. [cit. 2013-4-16].

[39] Martin Malý. Rest: architektura pro webové api. http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/, 2013. [cit. 2013-4-16].

[40] atrice. Phonegap explained visually. http://phonegap.com/2012/05/02/phonegap-explained-visually/, 2013. [cit. 2013-4-5].

[41] Phonegap. http://phonegap.com/, 2013. [cit. 2013-4-5].

[42] Api reference. http://docs.phonegap.com/en/2.6.0, 2013. [cit. 2013-4-5].

Zoznam použitých skratiek a symbolov

skratka	**celý názov**	**vysvetlenie**
CERN	European Organization for Nuclear Research	Európska organizácia pre jadrový výskum
IETF	Internet Engineering Task Force	vytvára a propaguje internetové štandardy
SDK	Software development kit	sada nástrojov pre vývoj softvéru
API	Application Programming Interface	rozhranie pre programovanie aplikácii
W3C	World Wide Web Consortium	konzorcium produkujúce slobodné štandardy pre World Wide Web
SGML	Standardized General Markup Language	metajazyk, v ktorom sa definujú značkovacie jazyky
DTD	Document Type Definition	definícia typu dokumentu
CORS	Cross-origin resource sharing	zdieľanie zdrojov medzi doménami

Zoznam príloh

Príloha A

Obsah CD

Na priloženom CD sú zdrojové kódy k webovej aplikácii v zložke /**web_app**. V tejto zložke sú dve podzložky. Jedna obsahuje klientskú aplikáciu a druhá serverovú. Klientská obsahuje nasledujúce zložky a súbory:

- **js** — hlavná zložka klientskej aplikácie, obsahuje javascriptové zdrojové kódy,
- **style** — CSS súbory a písma,
- **img** — obrázky aplikácie,
- **tpl** — templatovacie súbory klientskej aplikácie,
- **app.html** — súbor, ktorý spúšťa klientskú aplikáciu.

Serverová aplikácia obsahuje nasledujúce zložky a súbory:

- **application** — severová aplikácia. Dôležité sú podadresáre: **controllers**, **models**, **core** a **config**,
- **system** — zdrojové kódy frameworku Codeigniter, ktoré používa serverová aplikácia,
- **uploads** — zložka, kde sa ukladajú fotografie lokácii,
- **sql** — obsahuje sql dotaz na vytvorenie databázových tabuliek,
- **index.php** — súbor, ktorý spúšťa serverovú aplikáciu.

Na priloženom CD v zložke /**phonegap_app** je Phonegap aplikácia. Obsahuje súbor **geo.apk**, čo je inštalačný súbor na platformu android. Potom obsahuje zložku /**source_codes**, v ktorej sú najdôležitejšie priečinky:

- **res** — obsahuje ikonu a splashscreen obrázok pre rôzve veľkosti obrazoviek,
- **assets/www** — zdrojové kódy, do značnej miery rovnaké, ako vo webovej aplikácii,
- **src/com/geoApp** — obsahuje zavádzaciu aktivitu, ktorá spúšťa našu hybridnú aplikáciu.

Poslednou zložkou v koreni priloženého CD, je zložka /**examples**, ktorá obsahuje príklady použitia HTML5 technológii.

Príloha B

Manuál

B.1 Postup inštalácie klientskej a serverovej aplikácie

1. Obe zložky /client a /server je potrebné umiestniť na PHP server.
2. Nastavte práva pre zápis pre zložky /server/uploads, /server/application/logs a ./server/application/cache.
3. Nainštalujete databázové tabuľky z /server/sql/tables.sql a nastavte prístup do databázy v configuračnom súbore /server/application/config/database.php.
4. Nastavte klientskú aplikáciu, aby pristupovala práve k Vášej serverovej aplikácii v súbore /client/js/config.js. Vložte adresu serveru do premennej BASE.
5. Spustite aplikáciu cez /client/app.html

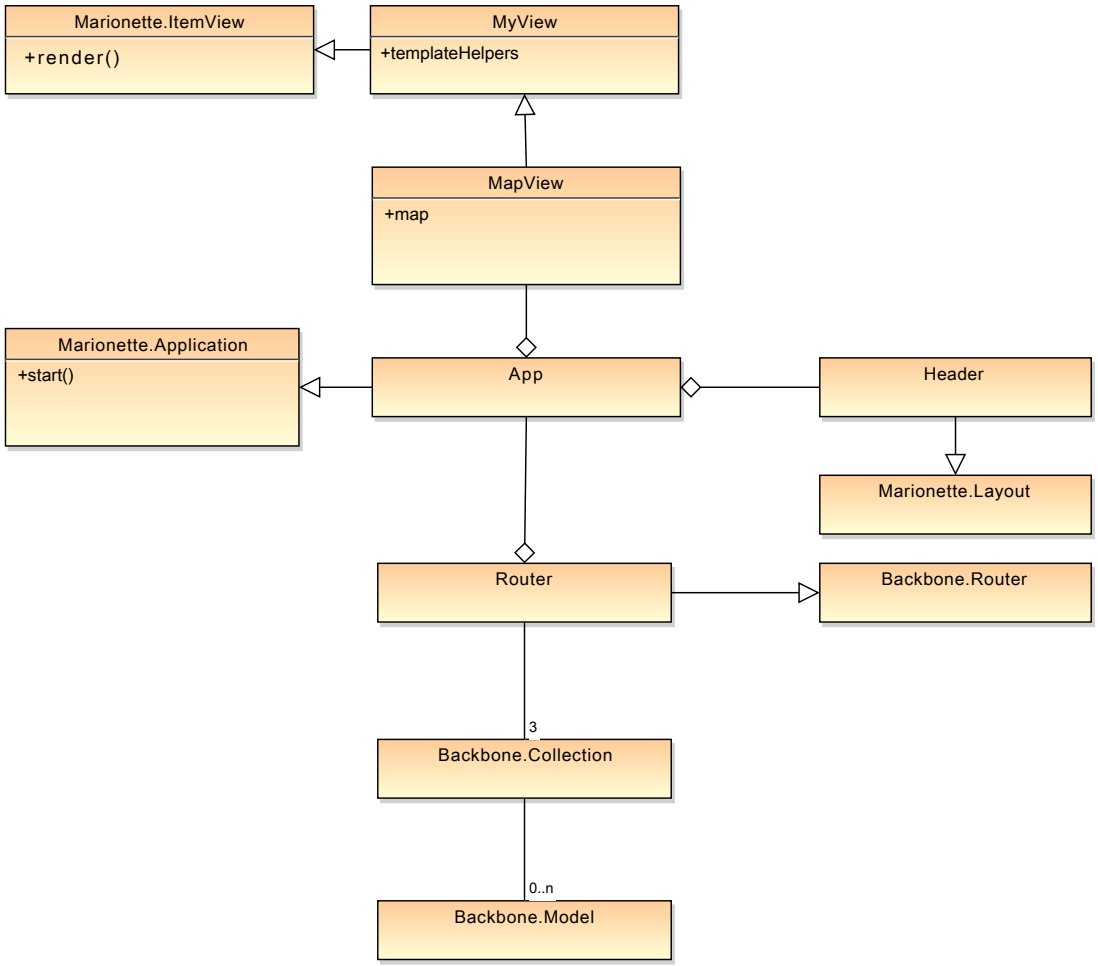
Aplikáciu je možné vyskúšať aj bez inštalácie na adrese:
http://www.stud.fit.vutbr.cz/~xkaspe01/geo/client/app.html

B.2 Preklad Phonegap aplikácie

V zložke /phonegap_app spustite príkaz `ant release`. Preložená aplikácia sa nachádza v priešinku /phonegap_app/bin.

Príloha C

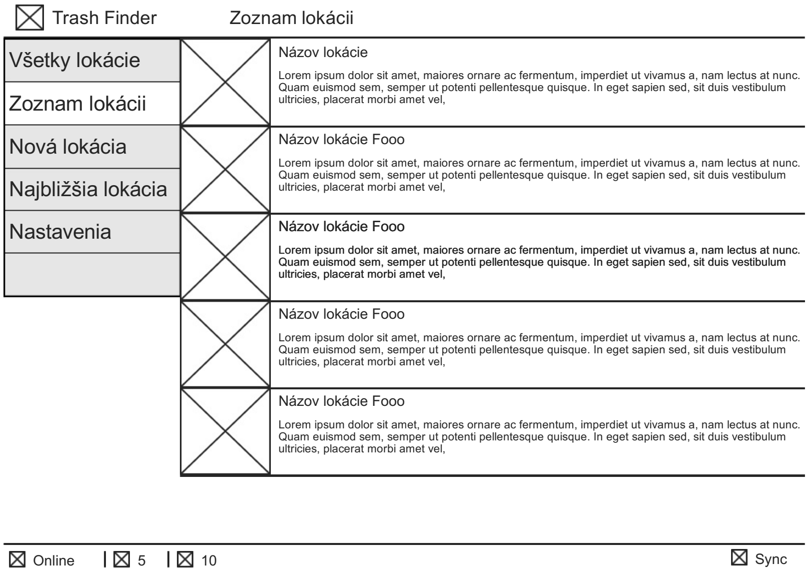
Diagram tried



Obrázok C.1: Diagram tried aplikácie. Neobsahuje všetky triedy – chýbajú skoro všetky pohľady a zobrazená je iba jedna kolekcia.

Príloha D

Návrhy užívateľského rozhrania



Obrázok D.1: Návrhy užívateľského rozhrania pre širšie obrazovky, napr. tablety.