TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

MOBILNÍ KLIENT PRO EXISTUJÍCÍ WEB

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Petr Žďárský**
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.

Liberec 2015

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

MOBILE CLIENT OF AN EXISTING WEB

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Petr Žďárský**
Supervisor: Mgr. Jiří Vraný, Ph.D.

Liberec 2015

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: 2014/2015

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Petr Žďárský**
Osobní číslo:	**M12000195**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Mobilní klient pro existující web**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se problematikou programování webových služeb s podporou REST api. Dále proveďte rešerši dostupných řešení pro programování mobilních aplikací za využití HTML5.
2. Na základě znalostí získaných v bodě 1 vytvořte návrh konverze existujcí webové aplikace na webovou službu. Rozhraní navržené služby by mělo odpovídat principům REST. K této webové službě navrhněte mobilního klienta, včetně způsobu jeho komunikace se službou.
3. Oba návrhy implementujte a vytvořte webovou službu i mobilní aplikaci. Otestujte funkčnost výsledného řešení.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **FIELDING, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures [online]. University of California, 2000 [cit. 2013-10-08]. Dostupné z: http://www.ics.uci.edu/-fielding/pubs/dissertation/top.htm**
[2] **RICHARDSON, Leonard a Sam RUBY. RESTful web services. 1st ed. Sebastopol: O'Reilly, 2007, xxiv, 419 s. ISBN 978-0-596-52926-0.**
[3] **OSMANI, Addy. Learning JavaScript design patterns. 1st ed. Sebastopol, CA: O'Reilly Media, 2012, xii, 235 p. ISBN 14-493-3181-5.**
[4] **WEYL, Estelle. Mobile HTML5. 1st ed. O'Reilly Media, 2013, xxvi, 450 pages. ISBN 14-493-1141-5.**

Vedoucí bakalářské práce: **Mgr. Jiří Vraný, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání bakalářské práce: **20. října 2014**
Termín odevzdání bakalářské práce: **15. května 2015**

prof. Ing. Václav Kopecký, CSc.
děkan

Technická univerzita v Liberci - 14 -
L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 20. října 2014

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 14. 5. 2015

Podpis:

Poděkování

Na tomto místě bych rád poděkoval Mgr. Jiřímu Vranému, Ph.D. za jeho odborné vedení mé bakalářské práce. A také bych chtěl poděkovat svému kamarádovi a spoluautorovi webových stránek SerialTracker.cz Tomáši Zenknerovi za jeho připomínky a konzultace řešení.

Abstrakt

Tato bakalářská práce se věnuje vývoji aplikačního rozhraní na existující webové stránce a dále vývojem webového mobilního klienta, který s touto službou komunikuje. První část práce se zabývá principy použité architektury rozhraní REST a srovnání vůči jiným typům architektur. Dále obsahuje rešerši o možnostech vývoje mobilních aplikací s využitím jazyka HTML5. Ve druhé části práce je popis implementace architektury REST na platformě ASP .NET. Práce také seznamuje s knihovnou Polymer, která byla použita pro vývoj mobilního klienta, a samotným vývojem tohoto klienta. Výsledkem práce je rozšíření stávající webové aplikace na webovou službu a vytvoření mobilního webového klienta, jehož design se přizpůsobuje uhlopříčce displeje použitého mobilního zařízení.

Klíčová slova:

HTML5, JavaScript, REST, Polymer, ASP .NET MVC, ASP .NET Web API

Abstract

This bachelor thesis is focused on development of application programming interface on existing web site and further on development of mobile web application that communicates with this service. The first part covers the principles of used interface architecture REST, and its comparision to other types of architectures. It also includes a search about capabilities of development mobile application using HTML5. The second part describes implementation of REST architecture on ASP .NET platform. There is also an introduction of Polymer library which was used for development of mobile client, and development of client itself. The result of this work is creating of a web service on existing web site and creating a mobile web client which has design that adapts to the screen size of mobile device.

Key words:

HTML5, JavaScript, REST, Polymer, ASP .NET MVC, ASP .NET Web API

Obsah

Seznam zdrojových kódů

Seznam obrázků

Cizí slova a zkratky

JS	JavaScript
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
REST	Representational State Transfer
HATEOAS	Hypermedia As The Engine Of Application State
CORS	Cross-Origin Resource Sharing
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
API	Application Programming Interface, aplikační rozhraní
cache	Dočasná paměť
applet	Softwarová komponenta
JSON	JavaScript Object Notation, zápis objektu v JS
XML	Extensible Markup Language
HAL	Hypertext Application Language
XML	Extensible Markup Language
XML-RPC	Extensible Markup Language – Remote Procedure Call
SOAP	Simple Object Access Protocol
SDK	Software Development Kit – sada nástrojů pro vývojáře
AJAX	Asynchronous JavaScript and XML

1 Úvod

Cílem této bakalářské práce je vývoj webové mobilní aplikace za využití HTML5 pro zájmovou webovou stránku SerialTracker.cz. Tato aplikace by měla mít přívětivé a intuitivní rozhraní pro uživatele, které bude responzivní.

Projekt SerialTracker.cz slouží jako informační kanál pro lidi, kteří pravidelně sledují seriály a chtějí mít přehled o seriálech a jejich nově odvysílaných epizodách. Spoluautorem této webové stránky je autor této práce a Tomáš Zenkner.

Nezbytnou součástí práce je také rozšíření stávající webové stránky na webovou službu, která bude splňovat požadavky architektury rozhraní REST. Mobilní aplikace bude skrze toto aplikační rozhraní získávat data webové stránky.

Tato bakalářská je rozdělena na dvě části: první část této práce pojednává o principech architektury rozhraní REST a vývoji aplikací za pomoci HTML5. Druhá část je zaměřena na implementaci rozhraní na webové stránce a dále o použitém řešení při vývoji webové aplikace.

2 Webové rozhraní REST

Architektura rozhraní REST – Representational State Transfer je softwarová architektura skládající se z principů a pravidel pro vytváření snadno škálovatelných webových služeb. Tyto principy byly poprvé zmíněny v dizertační práci [1] Roye Fieldinga.

Skládá se z šesti základních principů, které budou detailněji popsány v podkapitolách 2.1 až 2.6:

- Uniform interface
- Stateless
- Cacheable
- Client-Server
- Layered system
- Code on Demand

V této kapitole budou použity zejména anglické názvy, neboť jsou ustálené i v českém prostředí a nepřekládají se.

2.1 Uniform Interface

Jednotné rozhraní je základním prvkem architektury, kterým se liší od ostatním návrhů webových služeb. Celá architektura je zjednodušena a rozdělena do menších částí, které se mohou nezávisle rozvíjet a je tím také zlepšena přehlednost možností rozhraní. Jednotné rozhraní definují 4 pravidla [2] popsaná v kapitolách níže.

2.1.1 Resource-Based

V REST architektuře jsou zdroje rozhraní popsány pomocí jejich URI. Zdroje mohou být reprezentací záznamu v databázi, která jim však plně nemusí odpovídat, doplněnou o další informace v různých webových

formátech jako je HTML, JSON nebo XML. Zdroje však také mohou být i skripty (viz 2.6 Code on Demand), tedy kód, který si klient ze serveru stáhne.

```
http://mojestranka.cz/api/filmy/
```

Zdrojový kód 1 – Příklad URI ke zdroji Filmy

Při dotazování zdrojů webové služby se používají standartní metody [3] požadavků protokolu HTTP. Odpovědi také nesou tzv. HTTP stavový kód [4], který je číselný a odpovídá všeobecně dané informaci o výsledku dotazu.

HTTP metody lze rozdělit na bezpečné a idempotentní. Bezpečné metody jsou takové metody, které nemohou nijak ovlivnit data uložená na serveru, a to buď už cíleně, nebo omylem. Při provádění těchto dotazů může docházet k logování či analýze, ale není to považováno za ovlivnění dat. Mezi bezpečné metody patří GET, HEAD a OPTIONS. Metody jsou idempotentní, pokud jejich několikanásobné provedení má stále stejný účinek, mezi ně patří metody PUT, DELETE a v některých případech i POST.

GET

Metoda GET slouží k získání zdroje na dotazované URI. Data zdroje budou poté serverem navrácena v těle odpovědi. Zdrojem však nemusí být jenom datová reprezentace, ale může se jednat i například o skript či obrázek.

Metoda umožňuje vytvoření tzv. podmíněného požadavku, který v hlavičce nese informaci o podmínce pro dotaz na zdroj. Zdroj poté je vrácen podle těchto podmínek. Nejčastější kritériem je datum modifikace (*If-Modified-Since, If-Unmodified-Since*), nebo shoda (*If-Match, If-None-Match*). Podmínky slouží, ke zmenšení datového přenosu protože server případně vrátí prázdné tělo odpovědi a místo toho je použita reprezentace v dočasném úložišti klienta. Při použití hlavičky *If-Range*, lze získat pouze částečnou reprezentaci zdroje, která vyhovuje časovému rozsahu daného touto hlavičkou.

HEAD

Metoda HEAD je identická s metodou GET, avšak server nevrací tělo odpovědi. Hlavičky přijaté v odpovědi od serveru, by měli být naprosto stejné jako v případě použití metody GET. Tato metoda se používá k získání metadat (informace obsažené v hlavičce) o zdroji, testování jeho dostupnosti nebo zjištění nedávných změn.

OPTIONS

Tato metoda umožňuje klientu získat informace o požadavcích, povolených metodách nad zdrojem nebo možnostech serveru bez nutnosti získání či modifikace nějakého zdroje. Dalším použitím je, že prohlížeče mají povinost při mezidoménových dotazech provádět nejdříve ověření zda je tento dotaz možné provést, více v kapitole 2.7 Cross-Origin Resource Sharing.

```
OPTIONS * HTTP/1.1
Host: mojestranka.cz/api/hudba/
```

Zdrojový kód 2 – Příklad odchozích hlaviček na server

```
HTTP/1.1 200 OK
Allow: GET,POST,OPTIONS,DELETE
```

Zdrojový kód 3 – Příklad odpovědi od serveru

Pokud je jako cíl zadána „*“ informace se týká celého serveru, nikoli konkrétního zdroje. Toto však nemá žádné reálné použití, kromě testu zda je server dostupný tzv. ping.

POST

Metoda POST slouží k předání entity serveru, která má přímou souvislost se zdrojem, k jehož URI je dotaz odeslán. Například při odeslání entity na zdroj *Diskuze*, by tento dotaz měl vést k vytvoření nového příspěvku v diskuzi.

```
http://mojestranka.cz/api/diskuze/
Allow: GET,HEAD,POST,OPTIONS,TRACE
```

Zdrojový kód 4 – Příklad URI ke zdroji Diskuze

Pokud na serverem byla entita přijata kladně, může odpovědět několika způsoby, které jsou rozdílné podle stavových kódů:

200 – entita byla přijata (mohlo dojít k například vytvoření záznamu v databázi) a v těle odpovědi je související reprezentace této entity,
201 – entita byla přijata, a byl vytvořen nový zdroj, který může být dotázán pomocí URI; tělo odpovědi obsahuje související vytvořenou reprezentaci zdroje,
204 – entita byla přijata, avšak server nevrátil žádná data.

Pokud však server tuto entitu nepřijal z nějakého důvodu, vrací stavový kód z rodiny kódu 4xx, které jsou určeny pro hlášení chyb.

PUT

Metoda PUT se používá pro uložení entity na serveru obdobně jako metoda POST, avšak pokud již na serveru existuje reprezentace, která by měla být shodná s touto entitou, dojde k jejímu nahrazení.

DELETE

DELETE slouží k odstranění zdroje ze serveru.

2.1.2 Manipulation of Resources Throught Representatitions

Pokud klient již získal reprezentaci nějakého zdroje, tak má všechny potřebné informace k tomu jak zdroj modifikovat, smazat či získat rozšířené informace.

2.1.3 Self-descriptive Messages

Každá odpověď serveru obsahuje informaci o tom jak odpověď zpracovat. Toto bývá specifikováno v hlavičce odpovědi pomocí hodnoty Content-Type.

V odpovědi je také informace o možnosti dočasného uložení odpovědi (vytvoření tzv. cache, viz 2.3 Cacheable).

2.1.4 HyperMedia as the Engine of Application State

Hypermédia jako aplikační stav, je dalším důležitým principem architektury, který umožňuje objevovat dostupné akce nad zdrojem pomocí hypermédia, konkrétně hypertextového odkazu. Klient tak nepotřebuje dopředu vědět, jaké akce jsou dostupné, protože jsou všechny obsaženy již v samotném zdroji.

```
{
  "jmeno":"Jan",
  "prijmeni":"Novák"
  "links":[
    {
      "rel":"self",
      "href":"http://stranka.cz/lide/1/"
    },
    {
      "rel":"alba",
      "href":"http://stranka.cz/lide/1/alba/"
    }
  ]
}
```

Zdrojový kód 5 – Příklad reprezentace s odkazy ve formátu JSON

Viz Zdrojový kód 5 je příkladem reprezentace informací o uživateli jménem Jan Novák, a tato reprezentace má v sobě uloženy dva odkazy:

- odkaz sám na sebe (*self*), tedy na tuto reprezentaci,
- odkaz na alba uživetele.

Pokud by tento uživatel neměl k sobě přiřazena žádná alba, poté by dle definice HATEOAS[1] nebyl odkaz na vedoucí k získání alb uveden. Klíčovým

[1] What is HATEOAS and why is it important for my REST API?. *The RESTful cookbook* [online]. © 2012-2014 [cit. 2015-04-1]. Dostupné z: http://restcookbook.com/Basics/hateoas/

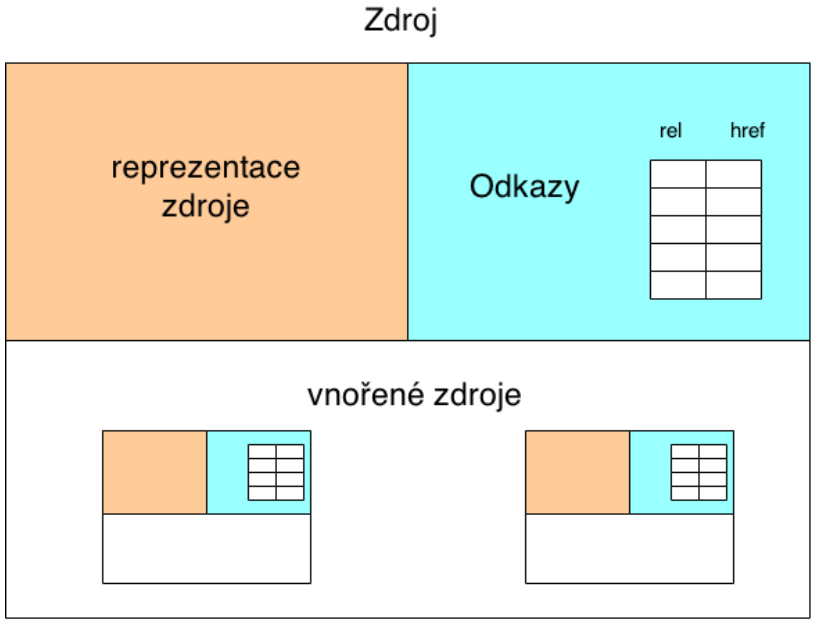
slovem *rel* je definován vztah odkazu neboli jeho jméno, *href* je poté samotný odkaz.

Napříč webovými službami, se ustálilo několik formátů implementujících HATEOAS v reprezentacích. Tyto formáty jsou blíže popsány v podkapitolách níže, včetně jejich srovnání.

HAL

HAL [5] je zkratkou pro Hypertext Application Language jehož autorem je britský softwarový inženýr Mike Kelly vlastnící agenturu poskytující konzultace při vytváření API.

HAL specifikuje, jakým způsobem mají být odkazy v reprezentaci uvedené. Umožňuje popisovat jak formát JSON tak i XML. Základní objektový model popsaný v HAL se skládá z reprezentace zdroje a jeho odkazů.



Obrázek 1 – Model objektu v HAL

Odkazy jsou ve zdroji uloženy pod klíčovou položkou *_links* a vnořené zdroje pod položkou *_embedded*.

```
{
  …
  "_links":{
    "self":{
      "href":http://stranka.cz/lide/1
    }
  },
  "_embedded":{
    "fotoaparat":{
      //vnořené zdroje reprezentující fotoaparáty
    }
  }
}
```

Zdrojový kód 6 – Příklad reprezentace zdroje v HAL

HAL také umožňuje přikládat odkaz na dokumentaci popisující vztah odkazu ke zdroji pomocí tzv. CURIE (Compact URI, kompaktní URI), umožňující šablonování a tím snadnější vytváření odkazů.

JSON-LD

JSON-LD [6] je zkratkou pro JavaScript Object Notation for Linked Data a jedná popisný formát zdroje v jazyce JavaScript podporované mezinárodním konsorciem W3C. JSON-LD přidává do objektu několik klíčových slov, které rozvíjejí a definují informace obsažené ve zdroji reprezentovaném v klasickém JSON objektu.

```
{
  "@context":{
    "jmeno":"http://stranka.cz/schema/jmeno",
    "avatar":{
      "@id":"http://stranka.cz/schema/avatar",
      "@type":"http://stranka.cz/schema/obrazek"
    },
    "fotoaparat":{
      "@container":"@set"
    }
  },
  "@id":"http://stranka.cz/lide/1",
  "jmeno":"Jan Novák"
  "avatar":"http://stranka.cz/lide/1/avatar.png"
}
```

Zdrojový kód 7 – Příklad reprezentace zdroje v JSON-LD

Viz Zdrojový kód 7 je jednoduchým příkladem popsaného zdroje v JSON-LD obsahující několik klíčových slov tohoto formátu. Několik vybraných klíčových slov je popsáno níže:

- @context – určuje kontext obsažených dat ve zdroji a jeho odkazy vedou na popis v online dokumentaci těchto dat, například pro položku *jmeno* bude na uvedeném odkaze detailněji popsáno, jestli se jedná o křestní jméno, příjmení či kombinaci obojího,
- @id – jedná se unikátní globální identifikátor daného zdroje,
- @type – odkazuje na datový typ informace,
- @container – definuje, že datový typ *fotoaparat* je poli,
- @set – definuje, že data v poli *fotoaparat* nejsou seřazena.

Collection+JSON

Collection+JSON [7] je další popisný formát založený na JSON objektech, autorem tohoto formátu je Mike Amundsen. Tento formát je určen zejména pro data, která lze reprezentovat v kolekci/poli avšak lze v něm reprezentovat i jeden datový objekt.

```
{
  "collection":{
    "version":"1.0",
    "href":"http://stranka.cz/lide/",
    "items":[
      {
        "href":"http://stranka.cz/lide/1",
        "data":[
          {
            "name":"jmeno",
            "value":"Jan",
            "promt":"Jméno"
          },
          …
        "links": [ … ]
        ]
      }
    "links":[
      {
        "rel":"avatar",
        "href":"http://stranka.cz/lide/1/avatar.png",
        "prompt":"Avatar"
      }
    ],
    "queries":[ … ],
    "template":[
      "data": { … }
      ]
    ]
  }
}
```

Zdrojový kód 8 – Příklad reprezentace zdroje v Collection+JSON

Viz Zdrojový kód 8 je příkladem kompletního objektu popsaném v Collection+JSON. Minimální objekt musí obsahovat objekt *collection* a v něm vnořené atributy *version* (verze formátu), *href* (odkaz na tuto reprezentaci) a poté vnořenou kolekci dat *items*. Jednotlivé položky kolekce poté mají vlastní odkaz k jejich zdroji, datovou reprezentaci a případně další odkazy. Celá reprezentace poté může být rozšířena o odkazy na celou kolekcí, dále *queries* což je informace o dostupných dotazech nad kolekcí, jejich formátu a způsobu dotazování. V neposlední řadě může reprezentace obsahovat *template* což je šablona objektu, který je serverem vyžadován při požadavcích pomocí metod POST či PUT.

SIREN

SIREN [8] je dalším formátem, který umožňuje prezentovat data jako pomocí JSON objektů tak i v XML, jehož autorem je Kevin Swiber.

Reprezentace je vyjádřena podobně jako objekty vyšších programovacích jazyků. Je definován typ zdroje pomocí klíčového slova *class*. V položce *properties* jsou poté vlastnosti tohoto zdroje. Reprezentace dále obsahuje odkazy (*links*), dostupné akce nad zdrojem (*actions*) a vnořené zdroje (*entities*).

```
{
  "class": "uzivatel",
  "properties": {
    "name":"Jan Novák",
    "avatar": "http://stranka.cz/lide/1/avatar.png"
  },
  "links":[
    {
      "rel": ["self"],
      "href":http://stranka.cz/lide/1
    }
  ]
  "actions": [ … ],
  "entities": [ … ]
}
```

Zdrojový kód 9 – Příklad reprezentace zdroje v SIREN

Srovnání

Ve výše zmíněných kapitolách byl přehled asi nejznámějších [9] formátů pro popisování reprezentací. Avšak formátů existuje daleko větší množství a pro potřeby dané služby si lze vytvořit i vlastní specifikaci.

Pro jednoduchý popis reprezentace je vhodné použít formáty SIREN nebo HAL, pro kolekce dat stejně dobře poslouží Collection+JSON. Další výhoda u formátu SIREN je, že odděluje dostupné akce na zdrojem od odkazů, stejně tak jako další formát HYDRA, který je postaven na JSON-LD. JSON-LD se pak více hodí pro komplexnější popis dat a vztahů. Nelze tak přesně určit, který formát je nejlepší, každý se hodí na něco jiného.

2.2 Stateless

Dalším pravidlem architektury je bezstavovost rozhraní, tzn., že všechny informace týkající se stavu klienta musí být obsaženy v dotazu prováděném na server. Není tak použita běžná praktika, kdy stav aplikace je uchováván serverem ať už v databázi či v *session*, ale všechny informace jsou tak uloženy v klientu. Klient tato data posílá na místě určené rozhraním. Můžou být například: součást URL v cestě nebo parametru, v těle nebo hlavičce dotazu.

Bezstavovost má několik výhod i nevýhod:

+ server má všechna data týkající se kontextu (např. autentizační token) obsažena v aktuálním dotazu,
+ jelikož jsou data obsažena v dotazu, server je méně náročný na zdroje, protože nemusí uchovávat data všech klientů,
+ stabilita celého systému je zlepšena, protože nelze ovlivnit data ostatních klientů,
– režie a vyšší datový přenos po síti jelikož se kontextová data posílají s každým dotazem.

2.3 Cacheable

Kvůli zvýšení síťové efektivity, je možnost dočasného uložení příchozích odpovědí od serveru, vytvoření tedy tzv. *cache*. Server tedy musí v odpovědi přidat i informaci možnosti dočasného uložení, zda lze odpověď uložit a případně na jak dlouho. Díky tomu je možné částečně či plně eliminovat zbytečné dotazy na server a tak snížit objem přenášených dat, zvýšit efektivitu a rychlost klienta. Nevýhodou je, že může nastat stav, kdy se liší data uložená v klientu a na serveru, i když z principu by se to stát nemělo, avšak kvůli neočekávanému zásahu do serverových dat mohou být změněna.

2.4 Client-Server

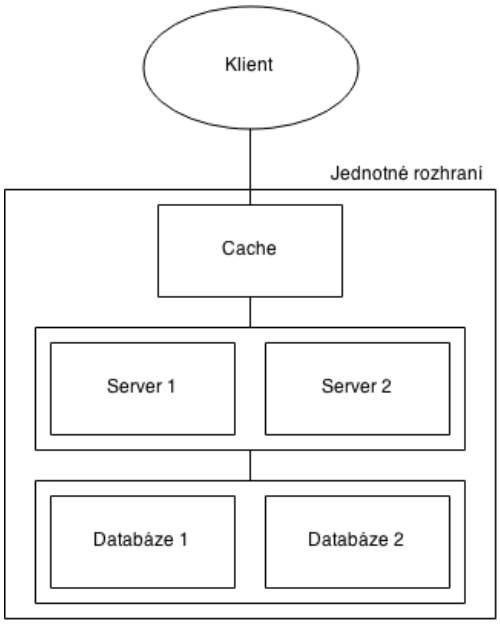
Jedním z nejdůležitějších pravidel je, že architektura je klient - server viz Obrázek 2. Principem této architektury je, že existují dvě aplikace, které komunikují přes jednotné rozhraní.

Obrázek 2 – Architektura klient – server

Klientská aplikace tak neví a nestará se o to, jak jsou data na serveru zpracovávána, a serverová aplikace je nezávislá na uživatelském rozhraní a jejím stavu. Přenositelnost a udržitelnost zdrojového kódu je tak o mnoho jednodušší, obě aplikace mohou být vyvíjené nezávislé na sobě, dokud dodrží předem dané rozhraní.

2.5 Layered System

Architektura umožňuje vytvářet vrstvy, tím je umožněna větší škálovatelnost serverové aplikace a rozložení zátěže mezi více serverů či poskytování sdílené mezipaměti. Klient se pouze dotazuje na rozhraní a neví, se kterou vrstvou komunikuje.



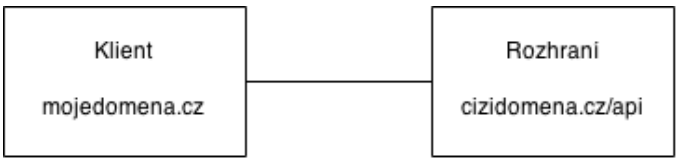
Obrázek 3 – Vrstvy serverové části

2.6 Code on Demand

Server umožňuje kromě zdrojů také poskytovat kód, který klient může použít k rozšíření své funkčnosti, například Java applety nebo kód v JavaScriptu. Tím jsou sníženy nároky na implementaci částí klientské aplikace. Server může tak poskytovat své zdroje v médiu (specifickém formátu), kterému klient nemusí rozumět, ale díky stažené knihovně (kódu) je schopen tomuto médiu porozumět.

2.7 Cross-Origin Resource Sharing

Nezbytnou součástí, kterou serverová aplikace při poskytování svého rozhraní musí implementovat je povolení dotazů na server z jiných domén, než je doména, na které je spuštěno rozhraní, pokud chce umožnit dotazování webovým klientům.



Obrázek 4 – Mezi-domění dotaz

Takové to dotazy bez povolení dotazovaného serveru není možné v prohlížeči provést. Moderní prohlížeče toto respektují a nelze tak pomocí HTTP dotazu získat zdroje z jiného serveru a to kvůli bezpečnosti.

Dotazovaný server tak musí do svých odpovědí přidat speciální hlavičky [10], které nesou informaci o tom, zda bude dotaz povolený. Prohlížeč před skutečným dotazem provede nejdříve zkušební dotaz pomocí metody OPTIONS, tzv. *preflight request* ve kterém zkontroluje hlavičky a případně skutečný dotaz zakáže.

Hlavičky, o které je odesílaný dotaz z klienta rozšířen:

- Origin – doména, ze které bude dotaz odeslán,
- Access-Control-Request-Method – HTTP metoda, která bude použita při skutečném dotazu,
- Access-Control-Request-Headers – seznam hlaviček, které budou v odpovědi přijaty a akceptovány.

Hlavičky, o které je odpověď ze serveru rozšířena:

- Access-Control-Allow-Origin – seznam domén, ze kterých jsou dotazy akceptovány, případně „*“ pro všechny domény,
- Access-Control-Allow-Credentials – hlavička nesoucí informaci o tom zda server bude akceptovat odesílané cookies z klienta,
- Access-Control-Expose-Headers – seznam hlaviček, které jsou prohlížeči dostupné,
- Access-Control-Max-Age – hlavička nesoucí informaci o tom jak dlouho je možné mít dočasně uložený výsledek zkušebního dotazu,
- Access-Control-Allow-Methods – seznam metod, které bude server akceptovat při skutečném dotazu,
- Access-Control-Allow-Headers – seznam hlaviček, které budou akceptovány při skutečném dotazu.

2.8 Další možnosti stavby webových služeb

Tato kapitola se zabývá srovnáním dalších možných způsobů jak vytvářet aplikační rozhraní webových služeb v architektuře klient-server. Konkrétně byly vybrány protokoly, kterou jsou opakem REST architektury a využívají volání procedur místo dotazování se přímo na zdroje.

2.8.1 XML-RPC

XML-RPC [11] je protokol pro provádění volání procedur na vzdáleném serveru pomocí klasického HTTP dotazu. Autorem je Dave Winer, zakladatel firmy UserLand, který protokol navrhl a specifikoval pro produkt Frontier v roce 1998.

Klientská aplikace odesílá dotaz na server, v jehož těle je formátovaná zpráva ve značkovacím jazyce XML. Dotaz je vždy odeslán pomocí metody POST.

Formát dotazu

V dotazu musí být přítomno několik specifických hlaviček pro splnění specifikací protokolu:

- User-Agent – jméno klientské aplikace,
- Host – adresa klientské aplikace, ze které je požadavek odesílán,
- Content-Type – hlavička vždy musí obsahovat hodnotu *text/xml*, jelikož protokol formátuje data do XML souboru,
- Content-Length – přesná velikost těla požadavku v bytech, která slouží k ověření správného přenosu všech dat.

Tělo požadavku musí mít přesně daný formát. Jednoduchý příklad těla dotazu je zobrazeno v Zdrojový kód 10.

Veškerá data dotazu musí být obalena značkou *methodCall.* V této značce jsou už poté data určující jméno metody, která se bude volat (značka *methodName*) a poté její parametry (značka *params*). Parametry metody mohou mít několik základních datových typů: *int*, *double*, *string*, *Boolean*,

base64 a *dateTime.iso8601* což je datum a čas zapsané v normě ISO 8601. Tyto datové typy lze poté použít při reprezentací polí (*array*) či struktur (*struct*).

```
<?xml version="1.0"?>
<methodCall>
  <methodName>jmenoTridy.jmenoMetody</methodName>
  <params>
    <param>
      <value>
        <int>41</int>
      </value>
    </param>
    </params>
</methodCall>
```

Zdrojový kód 10 – Příklad XML-RPC dotazu

Formát odpovědi

Všechny odpovědi od serveru vrací, mají HTTP stavový kód o hodnotě 200 – OK, a to i v případě, že dotaz skončil chybně. Tato chyba je případně popsána v těle odpovědi. Pokud však chyba vznikla na nižší úrovni než je aplikace, server vrací standardní kódy. Odpověď nese obdobné hlavičky, jako ty které jsou přítomné v dotazu.

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <string>Text odpovědi</string>
      </value>
    </param>
  </params>
</methodResponse>
```

Zdrojový kód 11 – Příklad odpovědi na XML-RPC dotaz

Výsledek dotazu je uchován v jednoduché struktuře, která je obalena značkami *methodResponse* (odpověď) a *params* (hodnoty). Datové typy jsou stejné jako pro dotaz.

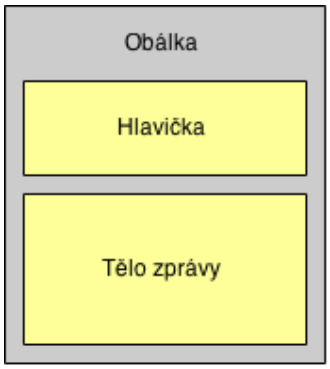
Srovnání vůči architektuře REST

XML-RPC je velice jednoduchý protokol, který umožňuje volání procedur na vzdáleném serveru a získávat tak potřebná data. Nevýhodou tohoto protokolu je neobjevitelnost dostupných metod na aplikačním rozhraní. Protokol nevyžaduje, ani nepodporuje popis rozhraní pomocí jazyka WSDL, který slouží pro popis webových služeb. Aplikační rozhraní tak musí být zdokumentováno jinak. Dále je pro vývoj webových klientů nevhodný formát dat XML, avšak existuje i alternativa JSON-RPC.

2.8.2 SOAP

SOAP [12] je protokol s otevřeným standardem, který je nyní pod záštitou konsorcia W3C. Za návrhem protokolu stojí David Winer, Don Box, Bob Atkinson a Mohsen Al-Ghosein, kteří protokol specifikovali v roce 1998 za podpory firmy Microsoft.

Vychází z protokolu XML-RPC a tak jako jeho předchůdce používá pro výměnu zpráv značkovací jazyk XML.



Obrázek 5 – Struktura SOAP zprávy

SOAP umožňuje zasílat zprávy po nejrůznějších síťových protokolech, ale nejčastěji je používán na protokolu HTTP. V tomto protokolu jsou zprávy opět posílány pomocí metody POST, jako tomu bylo i u XML-RPC. Zpráva má pevně danou strukturu (viz Obrázek 5) a je uzavřena v obálce (*Envelope*), v níž se poté volitelně nachází hlavička (*Header*), která nese informace o tom jak zprávu zpracovat. Následně poté je samotné tělo zprávy (*Body*) obsahující posílaná data.

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ziskejDetailProduktu xmlns="http://stranka.cz/api">
      <id>123456</id>
    </ziskejDetailProduktu >
  </soap:Body>
</soap:Envelope>
```

Zdrojový kód 12 – Příklad SOAP dotazu

V ukázce Zdrojový kód 12 je uveden příklad dotazu na webovou službu, pomocí kterého chce klient získat detailnější informace o produktu s daným id.

```
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ziskejDetailProduktuResponse
xmlns="http://stranka.cz/api">
      <ziskejDetailProduktuResult>
        <jmeno>Prací prášek</jmeno>
        <id>123456</productID>
        <cena>159</cena>
      </ziskejDetailProduktuResult >
    </ziskejDetailProduktuResponse >
  </soap:Body>
</soap:Envelope>
```

Zdrojový kód 13 – Příklad SOAP odpovědi

V ukázce Zdrojový kód 13 je uveden příklad odpovědi na dotaz, obsahující detailnější informace o produktu.

Srovnání vůči architektuře REST

Jelikož je protokol založen na protokolu XML-RPC má tak i stejné vlastnosti. Výhodou tohoto protokolu je použitelnost na více síťových protokolech, kde REST funguje pouze přes protokol HTTP. Avšak formát zpráv a samotný jazyka XML je velice těžkopádný a je tak přenášeno mnohem více dat nežli u jednodušších formátu u REST architektury. Objevitelnost rozhraní je u tohoto protokolu vylepšena, tím že webová služba bývá z pravidla popsána pomocí jazyka WSDL, toto ale vyžaduje udržovat aktuální dokumentaci vůči rozhraní, kdežto u REST architektury jsou objevitelné akce součásti rozhraní.

3 Vývoj aplikací v HTML5

V současnosti je trendem vyvíjet webové aplikace nežli vytvářet klasické aplikace pro různé operační systémy. Důvodem je přístupnost aplikace na všech operačních systémech, jelikož si uživatel vystačí pouze s webovým prohlížečem, který je dostupný na všech platformách a to i těch mobilních. V případě, že taková aplikace nepotřebuje speciální přístup k hardware zařízení nebo velký výpočetní či grafický výkon, jak je tomu například u her (i když v dnešní době lze za pomoci moderních technologií jako je WebGL dosáhnout dobrých výsledků i v prohlížeči), lze takovou aplikaci nahradit pomocí webové aplikace. Další nespornou výhodou je údržba aplikace, protože není nutné se starat o podporu různých verzí aplikace, jelikož každý uživatel používá stejnou nejnovější verzi aplikace na oné webové stránce.

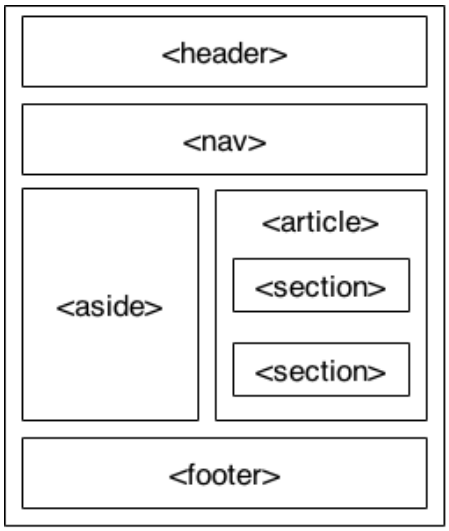
3.1 Možnosti HTML5

Jelikož je tento standard poměrně rozsáhlý a obsahuje spoustu nových vlastností oproti předchozím verzím, bude se následující kapitola zaobírat pouze vybranými vlastnostmi [13] HTML5, a to jak z pohledu samotných HTML elementů, tak následně z pohledu nových možností programovatelnosti webové aplikace spadající pod HTML5.

3.1.1 HTML5 elementy

Rozdělení stránky na sekce

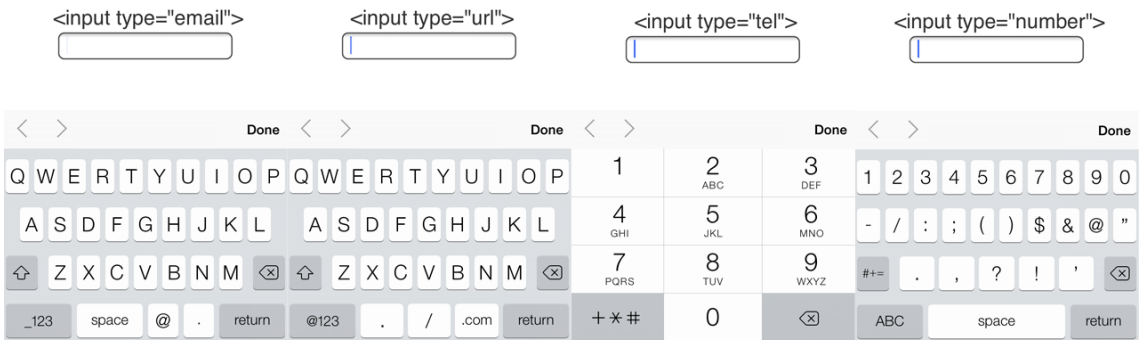
Nově byly do HTML5 přidány sémantické elementy, které rozdělují stránku na jednotlivé sekce stránky (viz Obrázek 6). Takto rozdělená stránka nejenže lépe vystihuje obsah jednotlivých částí, ale také usnadňuje strojové zpracování stránky, například obrazovým čtečkám hendikepovaným uživatelům.



Obrázek 6 – Příklad sémantického rozložení stránky

Typy vstupních prvků

V předešlé verzi HTML mohl mít vstupní prvek *<input>* pouze několik málo typů, jednalo se například o políčko pro text, heslo či odesílací tlačítko. Do specifikace bylo přidáno 13 dalších typů, které pokrývají velkou škálu vstupních typů dat od uživatele, konkrétně se jedná o tyto typy: *search, tel, url, email, datetime, date, month, week, time, datetime-local, number, range, color.* Jednotlivé typy disponují také jednoduchou formou validace elementu, takže například v typu *email* bude akceptován pouze email ve správném formátu a v typu *number* pouze číslo.

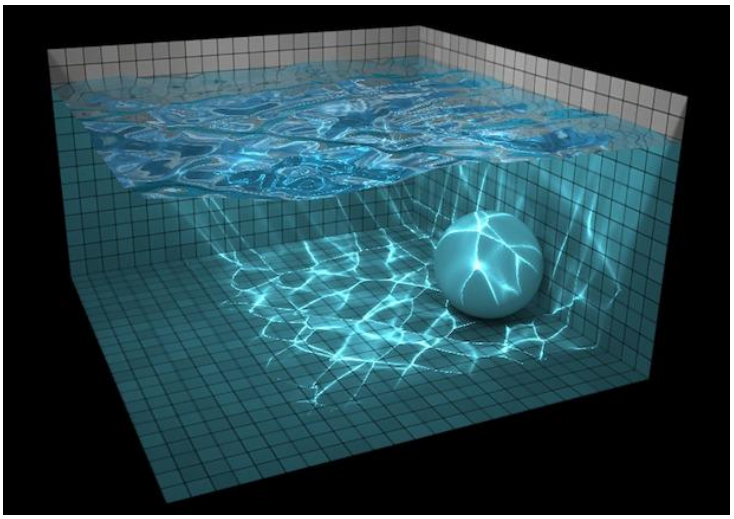


Obrázek 7 – Příklad různých klávesnic pro vstupní typy [14]

Největší uplatnění těchto typů však bude na mobilních zařízeních, která mají softwarové klávesnice a pro každý typ se tak přizpůsobí (viz Obrázek 7) a nabídnou uživateli relevantní klávesy.

Plátno

Plátno neboli *<canvas>* je velmi důležitá a oblíbená značka, která se uplatňuje při vykreslování různé grafiky, a to buď ve 2D nebo 3D (WebGL, viz Obrázek 8). Lze jej například využít pro vykreslení grafů, herní nebo jiné grafiky. Aplikační rozhraní plátna umožňuje kreslit jednoduché tvary, text, obrázek, avšak dnes již existující pokročilé frameworky v jazyce JavaScript, které mají vlastnosti profesionálních frameworků pro tvorbu počítačové grafiky.



Obrázek 8 – Aplikace s pokročilou simulací vody ve WebGL [15]

Multimédia

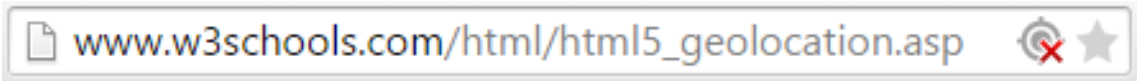
Specifikace přidává dva nové elementy umožňující nativně přehrávat audio (*<audio>*) či video (*<video>*) v prohlížeči. Toto bylo možné i předtím, avšak pouze za pomoci pluginů do prohlížeče jako je Adobe Flash nebo Apple QuickTime

3.1.2 Programové funkce

Tato kapitola pojednává o vybraných vlastnostech a funkcích aplikačního rozhraní jazyka HTML5. Některé zmíněné funkce nejsou přímou součástí standardu HTML5, avšak termín se v současnosti stal spíše synonymem pro moderní webové technologie, a proto jsou tyto techniky zde uvedeny.

Geolokace

Geolokace je jednou z nových funkcí standardu HTML5 a lze pomocí této technologie zjistit poměrně přesnou polohu uživatele. Určování polohy probíhá několika způsoby podle dostupných údajů ze zařízení. Nejméně přesné je určování polohy pomocí IP adresy, avšak u mobilních zařízení, která disponují GPS, je možné určit polohu s přesností na metry. Jelikož získání polohy uživatele zasahuje do jeho soukromí, je nutné získání polohy pro danou stránku nejprve povolit (viz Obrázek 9).



Obrázek 9 – Zamítnutá geolokace v adresním řádku prohlížeče

Úložiště

Lokální úložiště (*local storage*) a dočasné úložiště (*session storage*) je velice podobné *cookies,* informace se ukládají jako klíč – hodnota. Avšak jeho výhodou je, že tato data nejsou odesílány na server společně s každým požadavkem a velikost tohoto úložiště je také několika násobně větší. Maximální velikost *cookies* je 4 kB, lokální úložiště má velikost mezi 2-10 MB a dočasné úložiště bývá zpravidla neomezené.

Aplikační mezipaměť

Aplikační mezipaměť (*AppCache*) je způsob jak vytvářet webové aplikace, které fungují i bez připojení k internetu. Je toho docíleno tím, že se do stránky přidá odkaz na manifest soubor (viz Zdrojový kód 16), který prohlížeči sdělí, které soubory se mají uložit do mezipaměti. Tyto soubory se pak při příštím spuštění stránky načtou z mezipaměti a je tak i snížena datová náročnost aplikace.

```
CACHE MANIFEST
CACHE:
index.html
css/site.css
scripts/app.js
```

Zdrojový kód 14 – Příklad manifestu aplikační mezipaměti

Paralelní běh funkcí

Nově standard umožňuje spouštět více vláken v JavaScriptu pomocí WebWorkers, do této doby všechen napsaný kód blokoval hlavní a jediné vlákno, ve kterém vše probíhalo, to u špatně napsaných aplikacích či chybách mohlo způsobit zamrznutí webové stránky. Kód v dalším vlákně však nemá přístup k DOM objektu stránky, a proto se tato vlákna uplatňují výhradně pro výpočetně náročné aplikace. Dále musí být kód každého vlákna oddělen do vlastního souboru. Příklad vytvoření nového vlákna je uveden v Zdrojový kód 15.

```
var worker = new Worker('worker.js');
```

Zdrojový kód 15 – Příklad vytvoření nového vlákna

Databáze

V prohlížečích je také možné vytvářet lokální databáze, ty umožňují ukládat data v komplikovanější struktuře, než které poskytuje lokální úložiště. Prohlížeče disponují dvěma navzájem nekompatibilními databázemi: IndexedDB a WebSQL. V současné době se však nedoporučuje používat WebSQL, a to z toho důvodu, že se jedná pouze o jakýsi obal nad SQLite databází, vytvoření plnohodnotného standardu by tak bylo složité. Rozdíl mezi těmito databázemi je v jejich typu, WebSQL je klasická relační databáze, kdežto IndexedDB je objektové úložiště. Pro JavaScriptové vývojáře je tak mnohem jednodušší použití IndexedDB, jelikož k databázi přistupují jako k objektu a nemusejí se učit dotazovacímu jazyku SQL, který je použit při práci s WebSQL.

3.2 Aplikace pro mobilní zařízení

Aplikace napsané v HTML5 samozřejmě lze otevřít v prohlížeči na mobilních zařízeních. Takové aplikace by tak měly mít responzivní design tzn., že zobrazení stránky bude optimalizováno pro všechny druhy nejrůznějších zařízení od mobilů až po velké monitory (viz Obrázek 10). Tento pojem poprvé definoval Ethan Marcotte ve svém článku [16] pro webový magazín A List Apart.

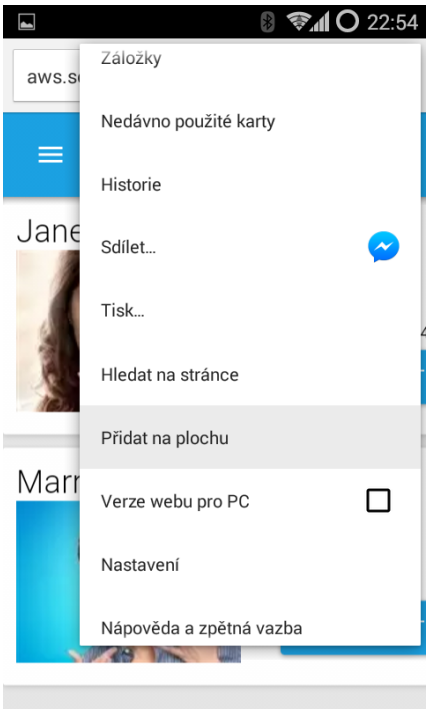
Obrázek 10 – Ukázka responzivního designu [17]

Webové aplikace dokáží bez problému nahradit nativní aplikace pro mobilní zařízení. V následujících kapitolách je popsáno, jak lze vytvářet nativní aplikaci v HTML5, nebo alespoň působila takovým dojem.

3.2.1 Skoro nativní aplikace

V moderních mobilních prohlížečích, jako je Chrome od firmy Google, je podpora tzv. manifestu webové aplikace [18] (*Manifest for a web application*). Jedná se o soubor ve formátu JSON, který dává autorovi webové aplikace kontrolu nad tím, jak jeho aplikace bude vypadat například na domovské obrazovce zařízení a také to, jak bude spuštěna. Díky tomu může webová aplikace působit jako nativní.

Prohlížeč Chrome má podporu manifestu od října roku 2014 a díky kombinaci s možností přidání rychlého odkazu na plochu (viz Obrázek 11) lze vytvořit plný dojem nativní aplikace.



Obrázek 11 – Přidání odkazu na plochu v prohlížeči Chrome

Struktura manifestu

```
{
  "name": "Moje aplikace",
  "icons": [{
      "src": "ikona.png",
      "sizes": "96x96",
      "type": "image/png"
  }],
  "start_url": "/index.html",
  "display": "standalone",
  "orientation": "landscape",
  "theme_color": "blue"
}
```

Zdrojový kód 16 – Příklad struktury manifestu webové aplikace

Manifest může obsahovat několik klíčových hodnot pro specifikování chování aplikace, minimalistická struktura je uvedena v příkladu Zdrojový kód 16.

Popis hodnot z výše uvedeného příkladu:

- *name* – jméno aplikace, pod kterým bude vytvořen zástupce na ploše,
- *icons* – pole ikon aplikace o různých velikost
- *start_url* – URL, ze které se má aplikace z odkazu spouštět,
- *display* – určuje, jak bude aplikace spuštěna,
    - *fullscreen* – aplikace se spustí přes celou obrazovku,
    - *standalone* – aplikace bude spuštěna jako klasická nativní aplikace,
    - *minimal-ui* – aplikace se spustí přes celou obrazovku s viditelnými ovládacími prvky systému,
    - *browser* – aplikace se otevře klasicky ve webovém prohlížeči,
- *orientation* – orientace zařízení (na výšku / na šířku),
- *theme_color* – definuje barevné téma aplikace, kterým se například zabarví systémový status bar.

Další možné hodnoty manifestu lze najít v jeho specifikaci[18].

Manifest je vkládán do webové stránky jako klasický odkaz na soubor v jazyce HTML (viz Zdrojový kód 17).

```
<link rel="manifest" href="/manifest.json">
```

Zdrojový kód 17 – Příklad odkazu na soubor s manifestem

Nevýhodou takovýchto aplikací je závislost na připojení k internetu, i když lze pomocí technik v jazyce JavaScript vytvořit lokální mezipaměť.

3.2.2 Nativní aplikace v HTML5

Tvorba nativních aplikací pro každou platformu je velice náročná a drahá záležitost. Je však možné vytvořit jednu webovou aplikaci za pomoci HTML a JavaScriptu, která se zabalí do nativní aplikace díky jednomu z dostupných frameworků. Takto obalená aplikace dokáže daleko lépe suplovat nativní aplikaci, jelikož může fungovat bez problémů offline a může mít přístup

k hardwaru telefonu, tedy např. k senzorům, mikrofonu, úložišti souborů, fotoaparátu atp. Tento hardware je poté dostupný pomocí aplikačního rozhraní daného frameworku přímo v JavaScriptu. Několik nejznámějších frameworků je popsáno v kapitolách níže.

Apache Cordova / PhoneGap

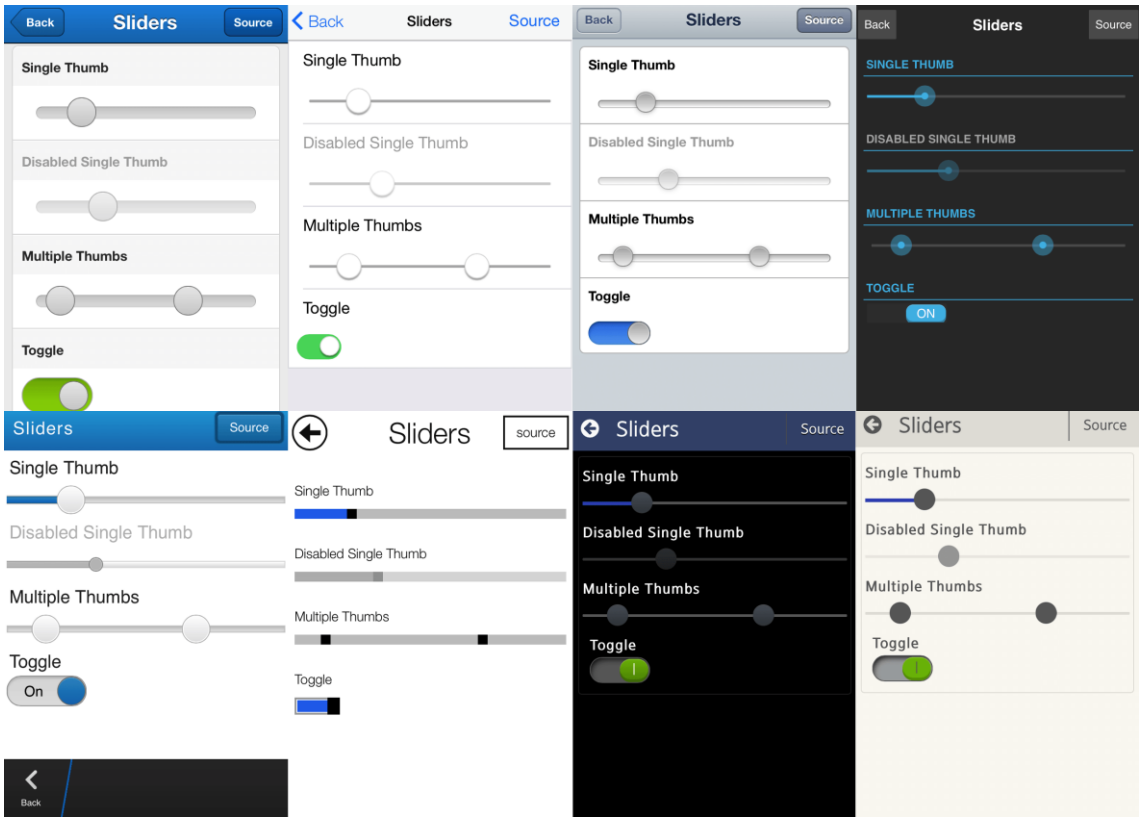
PhoneGap [19] je framework vyvinutý firmou Nitobi, který umožňuje tvorbu nativních aplikací v HTML5, JavaScriptu a CSS3. Nitobi bylo později koupené firmou Adobe a zdrojové kódy frameworku věnovány organizaci Apache Software Foundation, která zaštituje open source projekty. Tak vznikl druhý framework Apache Cordova, který je vyvíjen komunitně a ze kterého je spolu s dalšími nástroji sestaven PhoneGap. Frameworky jsou si velice podobné, jelikož se jedná pouze o jinou distribuci toho druhého.

Adobe rozšířilo PhoneGap o další zajímavé nástroje pro vývojáře, například o editaci aplikace bez nutnosti jejího opětovného kompilování či sestavování aplikací na jejich vzdáleném serveru pomocí služby Adobe PhoneGap Build. Tím odpadá nutnost instalování SDK pro jednotlivé platformy na počítači vývojáře. Tato služba je však dostačující, a to i v omezené verzi zdarma, samozřejmě je tu i možnost zakoupení licence pro plné využití služby.

Zpřístupnění funkcí systémů (senzory, systémové prvky, úložiště, databáze, …) je zprostředkováno pomocí pluginů, které zabalují nativní funkce do JavaScriptu. Pluginy jsou napsány vždy pro konkrétní platformu a je možné je stáhnout z veřejných repositářů jednotlivých frameworků či si napsat vlastní. Pluginy v těchto úložištích poskytují další rozšiřující možnosti aplikace, jako jsou například dialogová okna, ovládací prvky a další. Samotné frameworky nabízejí základní sadu nejdůležitějších pluginů napříč platformami, přehled těchto pluginů pro framework PhoneGap je uveden v příloze (viz Příloha A).

Sencha Touch

Sencha Touch framework je určený spíše pro tvorbu uživatelského prostředí, a to díky tomu, že je možné vytvářet jednu aplikaci, která na daných zařízeních použije ovládací prvky, které se podobají nativním (viz Obrázek 12). Tento framework také umožňuje zabalení webové aplikace do té nativní, avšak není to tak používané z důvodu nedostatku dostupných systémových funkcí aplikačního rozhraní, a proto je vhodnější kombinovat tento framework pro tvorbu uživatelského rozhraní a samotnou aplikaci zabalit pomocí vyspělejších frameworků jako je PhoneGap či Apache Cordova.



Obrázek 12 – Design aplikací napříč platformami v Sencha Touch[20]

4 Vývoj aplikačního rozhraní v ASP .NET Web API

Aplikace SerialTracker.cz byla napsána na frameworku ASP .NET [21] a aplikační rozhraní bylo plánované provozovat na stejném hostingovém serveru, a to z finančních ale i technologických důvodů. Bylo tedy nutné pro vývoj rozhraní použít také tento framework. Následující kapitoly pojednávají o implementaci rozhraní aplikace na tomto frameworku.

4.1 Web API

Pro implementaci aplikačního rozhraní jsou ve frameworku ASP .NET k dispozici dvě technologie: ASP .NET MVC (ve které je napsána samotná webová aplikace) a ASP .NET Web API. Pro vývoj rozhraní bylo použito Web API, hlavní důvody pro výběr právě této technologie jsou popsány v podkapitolách níže.

4.1.1 Formát odpovědi

Velkou výhodou Web API je, že automaticky serializuje data do formátu, který je požadován v příchozím požadavku, avšak formát musí být rozhraním podporován.

```
public class EpisodessController : Controller {
  [HttpGet]
  public ActionResult Index() {
    return Json(Episodes.GetAll(),
      JsonRequestBehavior.AllowGet);
  }
}
```

Zdrojový kód 18 – Ukázka kódu aplikačního rozhraní v ASP .NET Web MVC

```
public class EpisodesController : ApiController {
  public List<Episode> Get() {
    return Episodes.GetAll();
  }
}
```

Zdrojový kód 19 – Ukázka kódu aplikačního rozhraní v ASP .NET Web API

Výše uvedené ukázky zdrojových kódu (viz Zdrojový kód 18, Zdrojový kód 19) vykonávají prakticky stejnou funkci, avšak kód z první ukázky neumožňuje měnit návratový formát dat, který bude vždy JSON. Ve druhé ukázce je návratovým typem pouze kolekce objektů nesoucí požadované informace a o serializaci dat do konkrétního žádaného formátu se už stará samotná třída. Díky tomu je možné jednoduše podporovat více formátů a případné změny nevyžadují zásah do všech metod aplikačního rozhraní.

4.1.2 Jednoduché nastavení

Další nespornou výhodou Web API je, že konfigurace celého rozhraní je soustředěna do jedné třídy (souboru).

```
public static class WebApiConfig
{
  public static void Register(HttpConfiguration config)
  {
    //povoleni CORS
    config.EnableCors();

    //odstraneni klasického xml formateru
    config.Formatters.Remove(
      config.Formatters.XmlFormatter);

    //formatery pro HAL
    config.Formatters.Add(
      new JsonHalMediaTypeFormatter());
    config.Formatters.Add(
      new XmlHalMediaTypeFormatter());

    //handler na prihlaseni uzivatele pomocí hlavicky
    config.MessageHandlers.Add(
      new ApiAuthorizeHandler());
  }
}
```

Zdrojový kód 20 – Reálná ukázka konfigurační třídy rozhraní

V ukázce (viz Zdrojový kód 20) je část konfigurační třídy aplikačního rozhraní této aplikace. V třídě jsou nastavovány pro rozhraní následující věci:

- povolení mezi-doméních dotazů (viz 2.7Cross-Origin Resource Sharing),
- odebrání podpory serializace do klasického formátu XML,
- přidání dalších formáterů pro serializaci do formátu XML a JSON v hypermédiu HAL,
- přidání třídy, která umožňuje předzpracovat dotaz a získává autentizační hlavičku z dotazu, a vytvoří tak v aplikaci kontext přihlášeného uživatele.

4.1.3 Oddělení rozhraní od aplikace

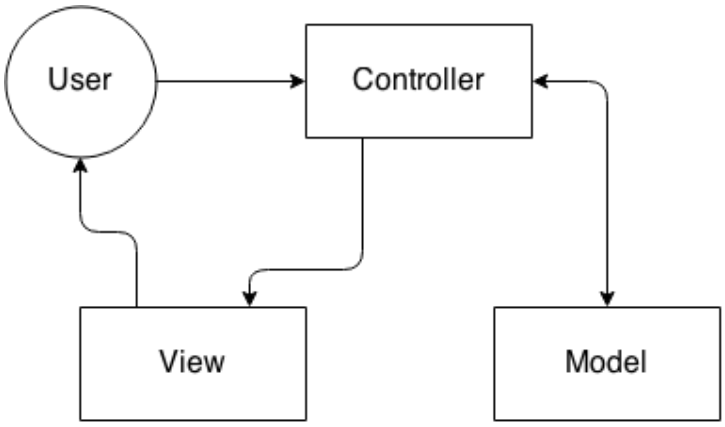
Rozhraní není závislé na přítomnosti dalších frameworků, např. ASP .NET MVC, lze tedy od webové aplikace jednoduše oddělit. To umožňuje lepší rozdělení zátěže, jelikož rozhraní a aplikace může běžet na oddělených serverech. Rozhraní je také daleko kompaktnější, jelikož Web API technologie je k tomu určená, kdežto MVC je zaměřeno spíše na tvorbu vizuálních stránek nežli na poskytování dat.

4.2 Struktura rozhraní

Tvorba Web API rozhraní je velice podobná tvorbě klasické MVC aplikace. Zkratka MVC znamená Model – View – Controller, což je softwarový návrhový vzor (viz Obrázek 13). Avšak v případě tvorby rozhraní odpadá ta část architektury, která zprostředkovává vizuální komunikaci s uživatelem, tedy View, i když jako View by se dala považovat data reprezentovaná v určitém formátu. Tvorba rozhraní se tak skládá ze dvou částí:

- Controller – servíruje data na rozhraní,
- Model – data.

Implementace jednotlivých částí je popsána v následujících podkapitolách.



Obrázek 13 – MVC architektura

4.2.1 Model

Rozhraní bylo navrženo tak, aby poskytovalo co nejvíce dat, která jsou uživateli k dispozici ve webové aplikaci. Rozhraní poskytuje přístup ke všem uživatelsky důležitým záznamům databáze (viz Příloha D), ale také k informacím, které jsou z těchto dat dynamicky počítány, například statistiky sledování uživatele nebo celého webu.

Objekty pro přenos dat

V aplikaci je použit pro přístup k databázi Entity framework [22], který slouží jako objektová vrstva mezi aplikací a databází. Vývojář tak nemusí mít žádné znalosti dotazovacího jazyka SQL, který se používá pro získávání dat z databáze, ale manipuluje přímo s daty na úrovni objektů, které jsou mu dobře známé. To však přináší nevýhodu při serializaci objektů, která je v případě rozhraní nutná. Jelikož objekty jsou navzájem propojené, serializování tak není možné, protože z objektu A se dá dostat přes jeho vlastnosti do objektu B, ovšem objekt B má vazbu na A, a dojde tak k zacyklení. Hloubka serializace se dá nastavit, avšak pro lepší kontrolu je vhodné objekt mapovat na vlastní DTO objekt. DTO je zkratkou pro Data Transfer Object, nejedná se o nic jiného než objekt sloužící k přenosu dat. To také umožňuje sestavit objekt na míru dané potřebě a například vynechat citlivé informace či naopak doplnit dynamicky počítané hodnoty.

DTO objekty v tomto rozhraní jsou odvozeny z objektu *Representation*, který poskytuje knihovna WebApi.HAL [23], jejímž autorem je Jake Ginnivan. Tato knihovna rozšiřuje objekty o nezbytná hypermédia dle specifikace architektury REST. Již z názvu knihovny vypovídá, že jako hypermédium tohoto rozhraní byla zvolena specifikace HAL. Tato knihovna rozšiřuje objekty o odkazy na sebe, odkazy na další akce nad daty a případně další vnořené objekty (viz Zdrojový kód 21).

```
//odkaz na sebe
public override string Href
{ … }

//jmeno odkazu na sebe
public override string Rel
{ … }

//dalsi odkazy
protected override void CreateHypermedia()
{ … }
```

Zdrojový kód 21 – Rozšířené atributy DTO objektu

4.2.2 Controller

Controllery jsou v rozhraní rozděleny podle toho, jaký zdroj dat poskytují, tedy například data epizod a seriálů mají oddělené controllery. Všechny controllery jsou odvozeny od základního bázového controlleru (viz Příloha C), který poskytuje přístup k databázi či aktuálně přihlášenému uživateli.

```
[ParameterValidationActionFilter]
[ApiRoutePrefix("Api")]
[ApiKeyAuthorize]
[EnableCors(origins: "*", headers: "*", methods: "*")]
public abstract class ApiBaseController : ApiController
{ … }
```

Zdrojový kód 22 – Atributy bázového controlleru

Bázový controller je odvozen od ApiControlleru, který je součástí Web API a umožňuje tak například vyjednávání formátu serializace

dat (viz 4.1.1 Formát odpovědi). Controller má definováno základní chování pro odvozené controllery, které je nastaveno pomocí atributů (viz Zdrojový kód 22). Tyto atributy nastavují následující:

- validaci parametrů akcí,
- prefix URL adresy aplikačního rozhraní,
- nutnost autentizace klientské aplikace pomocí aplikačního klíče,
- povolení mezi-doménich požadavků (viz 2.7 Cross-Origin Resource Sharing).

Každý odvozený controller poté má vlastní další atributy, z pravidla jde o atributy nastavující nutnost přihlášení uživatele k přístupu a prefix URL adresy controlleru.

Samotné akce uvnitř controlleru také mají atributy nastavující jejich vlastnosti (viz Zdrojový kód 23):

- nastavení typu HTTP metody,
- URL adresa akce, jež umožňuje i jednoduchou formu validace specifikováním datových typů, které se mají doplnit do šablony,
- návratový typ dat.

```
[HttpGet]
[Route("")]
[Route("page-{page:int}")]
[ResponseType(typeof(EpisodesPagedDto))]
```

Zdrojový kód 23 – Ukázka atributů akce

URL adresy akcí jsou skládány automaticky dle prefixu bázového controlleru, prefixu controlleru a šablony adresy dané akce. Výše zmíněná ukázka je součástí controlleru pro epizody a může mít 2 URL adresy, protože je obsah stránkovatelný: adresu s výchozí stránkou (první stránka) a adresu s číslem stránky. Proto následující URL adresy (viz Zdrojový kód 24) vedou ke stejným datům.

```
http://st-api.aspone.cz/api/episodes/
http://st-api.aspone.cz/api/episodes/page-1
```

Zdrojový kód 24 – Ukázka URL adresy na akci rozhraní

Samotný kód všech akcí poté není nijak složitý, jedná se pouze o získání dat z databáze a jejich přemapování do DTO objektů. Rozhraní v současné době disponuje 41 akcemi, které pokrývají prakticky všechna různá data, která se v aplikaci vyskytují.

4.3 Monitorování rozhraní

V aplikačním rozhraní je implementováno nutné autentizování klientské aplikace. Hlavním důvodem je získání přehledu o vytíženosti rozhraní, případně zjištění, které klientské aplikace jsou populární a používají se.

Autentizaci webových aplikací by šlo vyřešit pomocí jednoduché kombinace přiděleného aplikačního klíče a domény, ze které je dotaz odeslán. Klientské aplikace nemusejí být pouze webové a může se jednat například o nativní aplikaci pro operační systémy. U těchto aplikací by poté v dotazu nebylo možné určit doménu, oproti tomu je u nich výhodou, že lze ve zdrojovém kódu aplikace skrýt autentizační údaje aplikace, což u webových aplikací napsaných v HTML nelze. Vytvořit tak jednotnou a bezpečnou multiplatformní autentizaci by nebylo možné, proto je autentizace vyřešena pouze pomocí aplikačního klíče, který slouží spíše jako identifikátor aplikace nežli skryté heslo.

4.3.1 Formát klíče

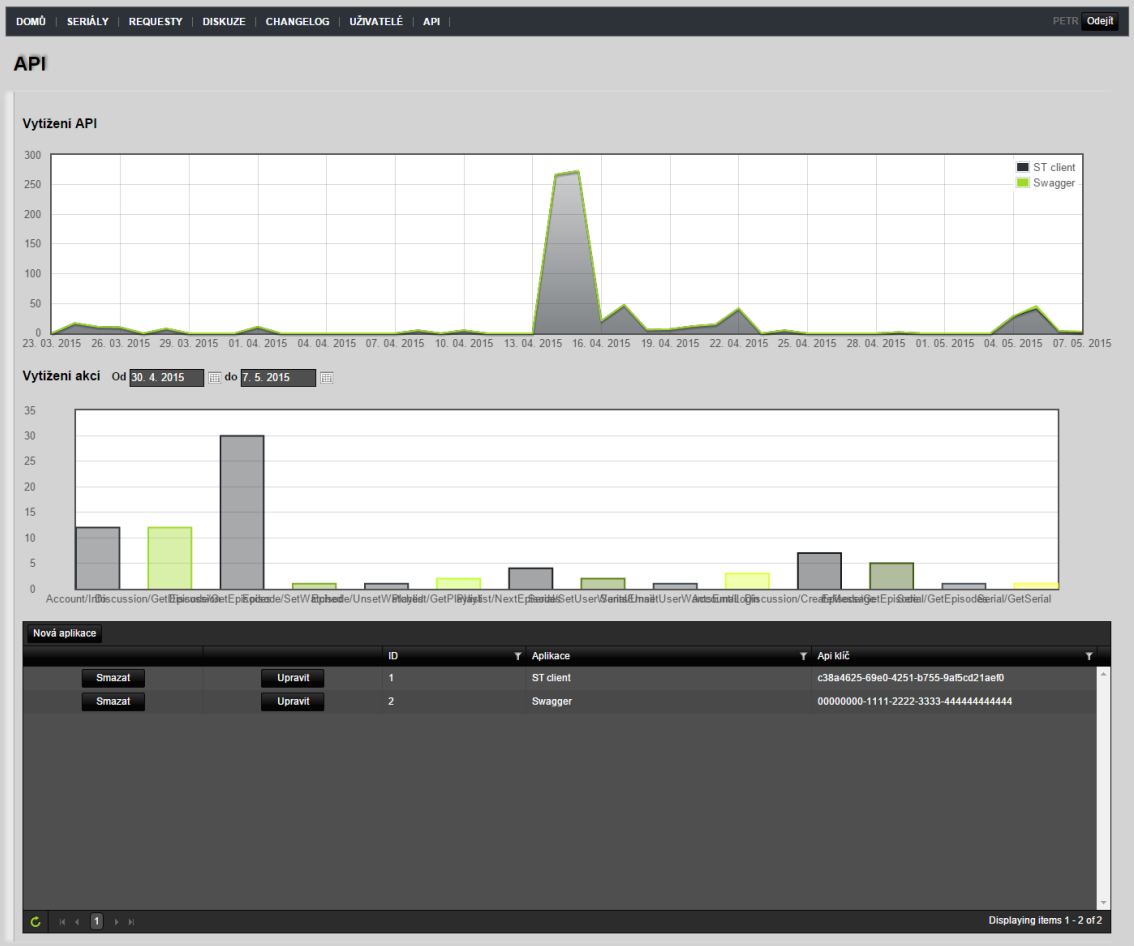
Jako formát klíče byl zvolen GUID – Globally Unique Identifier, jedná se o 128b číslo umožňující jednoznačnou identifikaci. Klíč je posílán aplikací v hlavičce dotazu o jméně *X-ApiKey*.

```
c61f721b-58e4-4ce0-bd0a-9a8df59c5b50
```

Zdrojový kód 25 – Ukázka aplikačního klíče

4.3.2 Monitoring rozhraní

Při každém dotazu klientské aplikace dochází k inkrementování záznamu v databázi, a je tak přesně určen počet dotazů a cíl akce. Tyto údaje jsou v přehledné formě interaktivních grafů zobrazeny v administračním rozhraní aplikace (viz Obrázek 14), kde mimo jiné lze registrovat novou aplikaci.



Obrázek 14 – Náhled vytíženosti rozhraní v administraci

4.4 Dokumentace rozhraní

Aplikační rozhraní architektury REST nevyžaduje přímo dokumentaci či popis funkcí rozhraní, pro větší komfort vývojářů, kteří budou chtít rozhraní používat, byla vytvořena dokumentace v nástroji Swagger UI [24]. Tento nástroj také velice ulehčil testovatelnost rozhraní při jeho vývoji.

4.4.1 Swagger UI

Swagger UI je nástroj, který tvoří součást frameworku Swagger [25], jenž byl navržen speciálně pro rozhraní architektury REST a umožňuje jejich popis, tvorbu, používání a v neposlední řadě vizualizaci rozhraní. Je tak možné vytvářet rozhraní, které je dobře srozumitelné jak pro člověka tak i pro počítač.

Framework obsahuje několik nástrojů, mezi něž patří Swagger UI, Swagger Editor – umožňující návrh rozhraní, a SDK CodeGen, který umožňuje z návrhu rozhraní vygenerovat zdrojový kód rozhraní pro různé programovací jazyky.

Integrace do aplikace

Dokumentaci rozhraní by bylo možné vytvořit ručně, to by ale bylo velice náročné na údržbu při změnách rozhraní. Proto existují komunitně vyvíjené nástroje či knihovny pro různé programovací jazyky, jež dokumentaci generují automaticky přímo ze zdrojového kódu. Tím je zaručeno, že dokumentace je vždy aktuální a přesně odpovídá rozhraní.

Pro generování Swagger dokumentace na frameworku .NET je dostupných několik knihoven, přičemž pro toto rozhraní byla vybrána knihovna Swashbuckle [26], která nabízí největší množství nastavení a je také neustále aktualizována a vyvíjena. Hlavním autorem knihovny je Richard Morris.

Integrace knihovny do aplikace je velice jednoduchá, lze ji totiž nainstalovat jako balíček z balíčkovacího manažeru Nuget [27], jenž je používán na platformě .NET. Po instalaci balíčku (viz Zdrojový kód 26) je automaticky do aplikace integrována a stačí provést minimální konfiguraci, která obsahuje nastavení cesty k XML souboru obsahující dokumentaci

```
Install-Package Swashbuckle
```

Zdrojový kód 26 – Instalace knihovny Swashbuckle

projektu (tento soubor je automaticky generován vývojovým prostředím Visual Studio [28]).

Nastavení knihovny

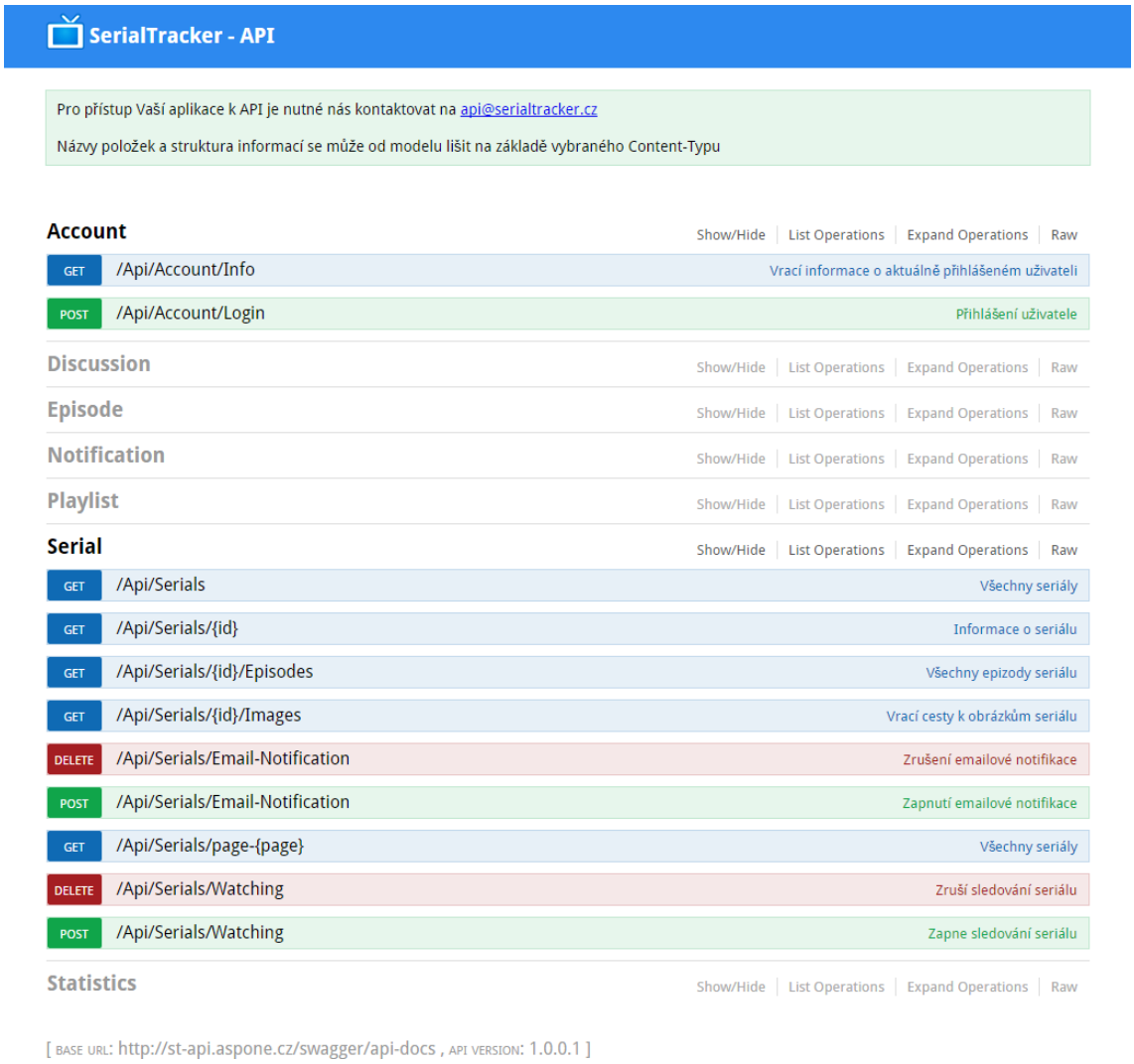
V ukázce níže (viz Zdrojový kód 27) je část konfiguračního souboru knihovny použité v aplikaci. Konfigurace je velice jednoduchá, je rozdělena do dvou částí: nastavení generování dokumentace a nastavení samotné vizuální aplikace Swagger UI.

V první části je definována cesta ke generovanému souboru obsahující dokumentaci rozhraní. Poté je přidán filtr, který vynechává vlastnosti objektů nesoucí atribut pro ignorování při serializaci. Dále už je pouze z formálního hlediska nastavena verze rozhraní odpovídající verzi projektu.

Ve druhé části je nastavena podpora odesílání hlaviček. Poté je už jenom nastavena cesta k souborům obsahující vlastní design aplikace Swagger UI (viz Obrázek 15), jenž je laděn do celkového vzhledu webové aplikace. Kompletní vyexportovaná dokumentace je součástí přiloženého média (viz Příloha E).

```
/// <summary>
/// Zavaděč swaggeru do aplikace
/// </summary>
public class SwaggerConfig
{
  public static void Register()
  {
    Swashbuckle.Bootstrapper.Init(
      GlobalConfiguration.Configuration);

    SwaggerSpecConfig.Customize(c =>
    {
      //načtení xml komentářů
      c.IncludeXmlComments(GetXmlCommentsPath());

      //filtr na JsonIgnore atribut
      c.ModelFilter<
      OmitJsonIgnorePropertiesModelFilter>();

      //nastavení verze API z assembly
      c.ApiVersion(Assembly.GetExecutingAssembly()
        .GetCustomAttribute<AssemblyApiVersion>()
        .ApiVersion);
      });

      SwaggerUiConfig.Customize(c =>
      {
        c.SupportHeaderParams = true;

        c.InjectStylesheet(
          typeof(SwaggerConfig).Assembly,
          "SerialTracker.Web.Api.SwaggerUI.Api.css");

        c.CustomRoute("index.html",
          Assembly.GetExecutingAssembly(),
          "SerialTracker.Web.Api.SwaggerUI.Api.html");
      });
    }
  }
}
```

Zdrojový kód 27 – Ukázka konfiguračního souboru knihovny Swagger



Obrázek 15 – Design aplikace Swagger UI

Vlastnosti

Aplikace zobrazuje dostupné akce rozhraní ve velice příjemné a interaktivní formě a díky generování dokumentace z komentářů a zdrojového kódu poskytuje všechny informace o dané akci.

V ukázce zdrojového kódu níže (viz Zdrojový kód 28) je uveden příklad správné definice akce, která popisuje následující:

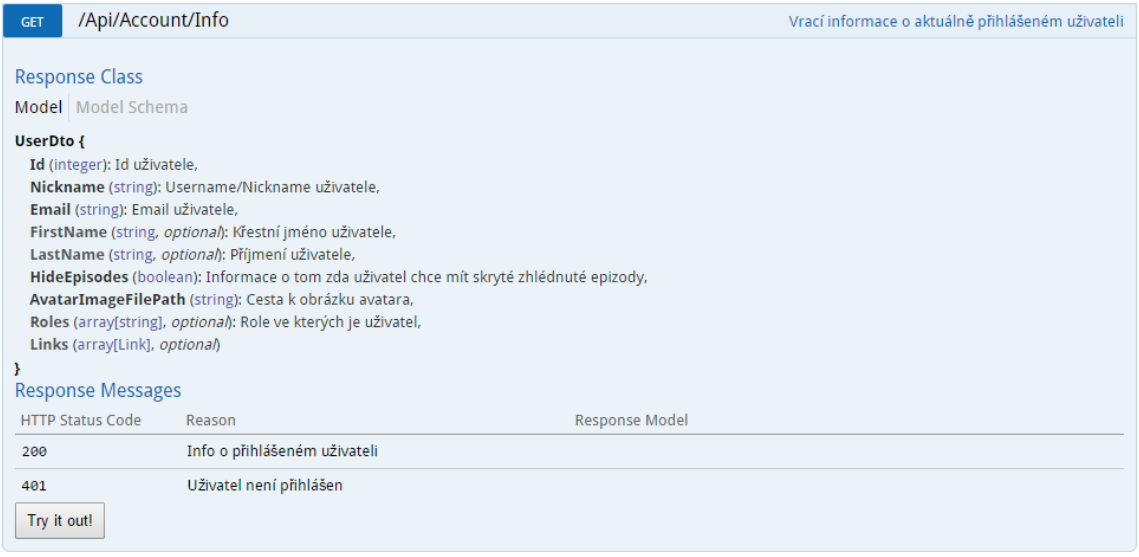
- informace, co akce dělá,
- HTTP stavové kódy, co akce může vracet,
- typ HTTP metody,
- URL adresa akce,
- návratový typ akce.

```
/// <summary>
/// Vrací informace o aktuálně přihlášeném uživateli
/// </summary>
/// <response code="200">Info o přihlášeném
uživateli</response>
/// <response code="401">Uživatel není
přihlášen</response>
/// <returns></returns>
[HttpGet]
[Authorize]
[Route("Info")]
[ResponseType(typeof(UserDto))]
public HttpResponseMessage Info()
{ … }
```

Zdrojový kód 28 – Ukázka dokumentace akce rozhraní

Pokud návratový typ (v tomto případě DTO objekt informací o uživateli) obsahuje komentáře, jsou zobrazeny vedle položek, ke kterým patřím – vše je tedy pro uživatele velice přehledné (viz Obrázek 16).

Všechny akce jsou v aplikaci testovatelné, lze tak vidět odpovědi rozhraní včetně hlaviček a stavových kódů.



Obrázek 16 – Dokumentace akce

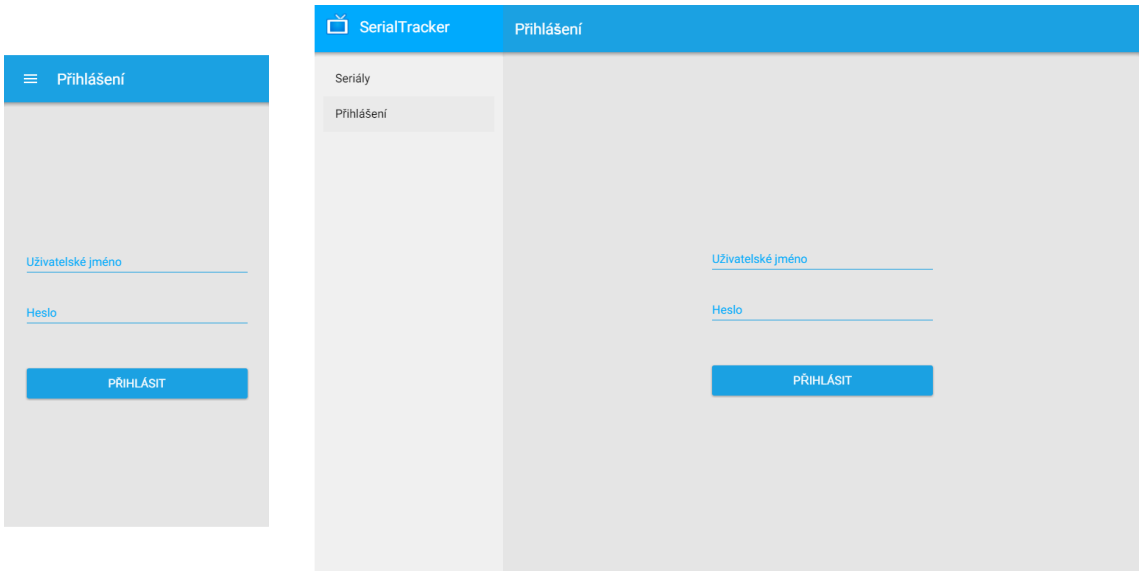
5 Mobilní webová aplikace

Tato kapitola se zabývá popisem a vývojem webového klienta v HTML5 za využití knihovny Polymer [29]. Pro vývoj aplikace byla použita verze knihovny 0.5, avšak k datu 27. 3. 2015 byla uvolněna verze 0.8, která přináší radikální změny ve vývoji aplikací v této knihovně. Proto se následující podkapitoly budou zabývat pouze vývojem v použité verzi knihovny.

5.1 Představení aplikace

V této kapitole budou představeny jednotlivé části aplikace spolu s náhledy ze dvou mobilních zařízení, jednoho s menším displejem – telefon, a druhého s větším displejem – tablet, a to z důvodu prezentování přizpůsobivého designu aplikace.

5.1.1 Přihlášení



Obrázek 17 – Náhled přihlašovací obrazovky

Na výše uvedeném obrázku (viz Obrázek 17) je náhled přihlašovací obrazovky aplikace. Přihlašovací formulář obsahuje i jednoduchou validaci polí, takže nedojde k odeslání formuláře při nevyplněných polích. Po odeslání formuláře je uživatelské jméno a heslo odesláno HTTP metodou POST na aplikační rozhraní, které, pokud jsou údaje správné, vrací aplikaci autentizační tiket, jenž je poté použit pro všechny následující požadavky na

rozhraní. Autentizační tiket je v aplikaci nezbytný, jelikož většina dat má spojitost s uživatelem.

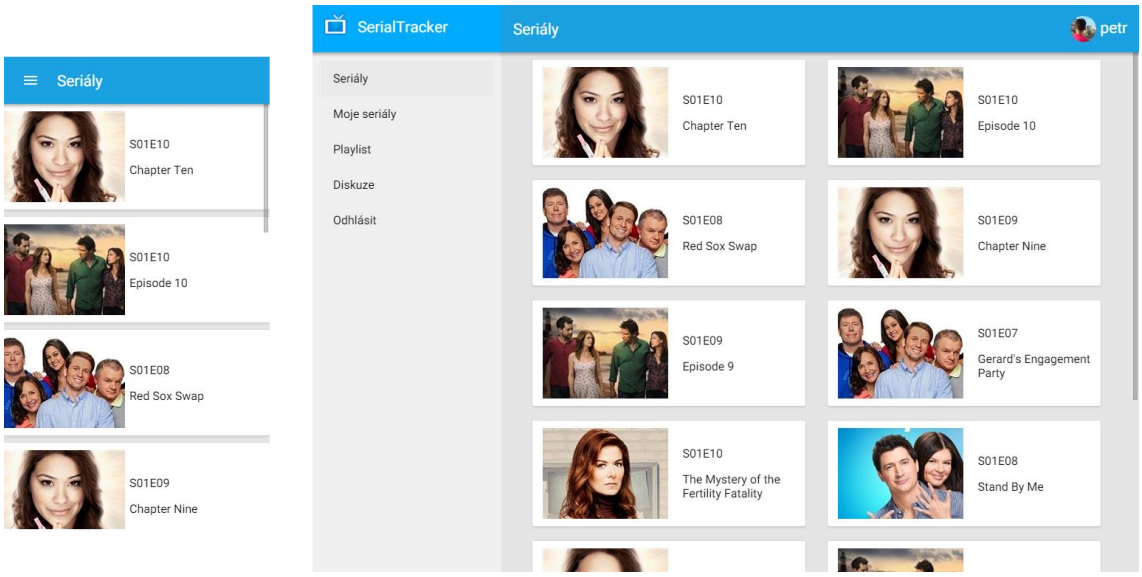
Stránky dostupné bez ověření uživatele:

- Přihlášení – přihlašovací obrazovka,
- Seriály – přehled všech nově odvysílaných epizod.

Stránky dostupné po přihlášení uživatele:

- Seriály – přehled všech nově odvysílaných epizod,
- Moje seriály – přehled nově odvysílaných epizod oblíbených seriálů konkrétního uživatele,
- Playlist – jednoduchý přehled následujících epizod ke sledování,
- Diskuze – diskuze na SerialTrackeru umožňující diskutovat o seriálech a jiných tématech,
- Odhlášení – odhlášení uživatele.

5.1.2 Odvysílané epizody



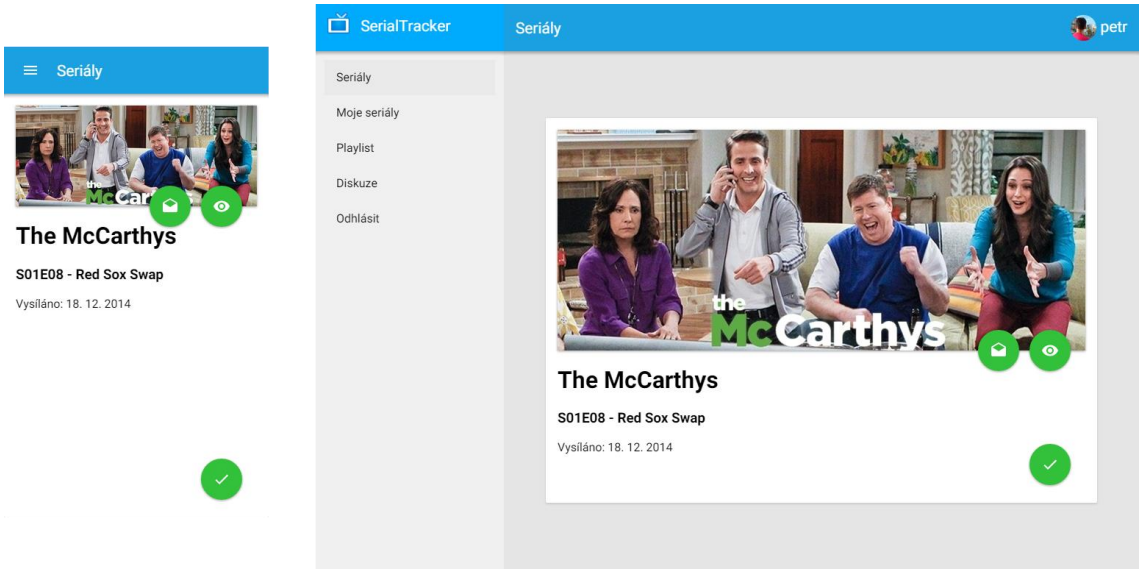
Obrázek 18 – Náhled obrazovky s odvysílanými epizody

Na výše uvedeném obrázku (viz Obrázek 18) je náhled obrazovky s odvysílanými epizodami. Aplikace disponuje také filtrovaným přehledem

odvysílaných epizod oblíbených seriálů právě přihlášeného uživatele. Tento přehled má naprosto stejný design i funkci, pouze nabízí filtrované epizody.

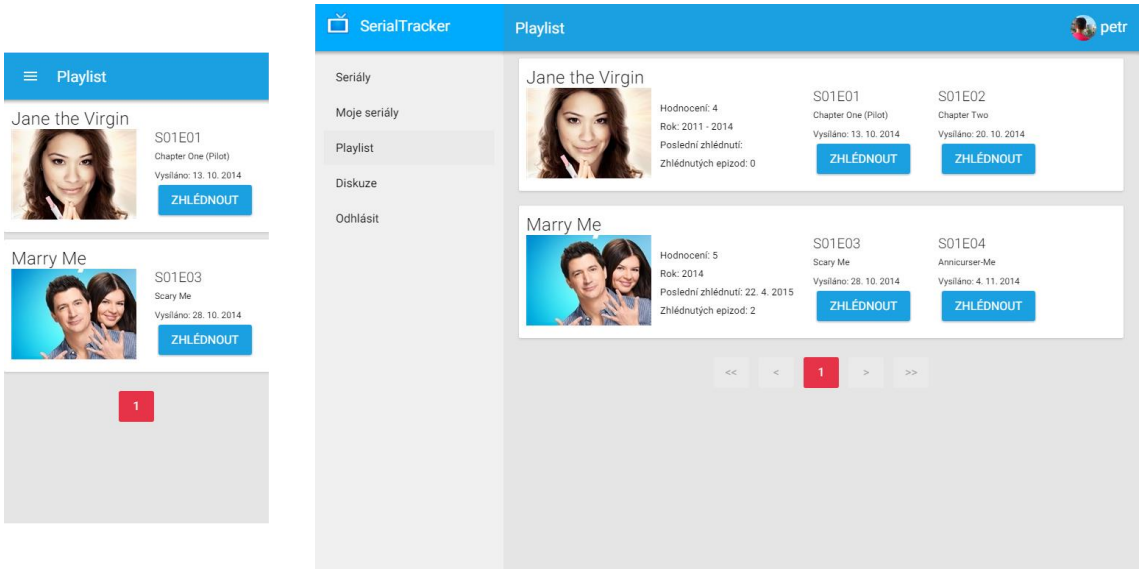
5.1.3 Detail epizody

Dlaždice s epizodami, které byly zobrazeny na předchozím obrázku (viz Obrázek 18), jsou interaktivní a po kliknutí na ně se otevře dialogové okno, které lze vidět na následujícím obrázku (viz Obrázek 19). Pokud je uživatel přihlášen, jsou zde k dispozici tři tlačítka. Dvě horní tlačítka slouží k nastavení emailových notifikací seriálu a k přidání mezi oblíbené seriály. Spodní tlačítko poté slouží k nastavení epizody jako zhlédnuté. Pokud má uživatel nastaveno ve svém profilu, že chce mít skryté zhlédnuté epizody, daná epizoda se již v přehledu nezobrazuje.



Obrázek 19 – Náhled na detail epizody

5.1.4 Playlist

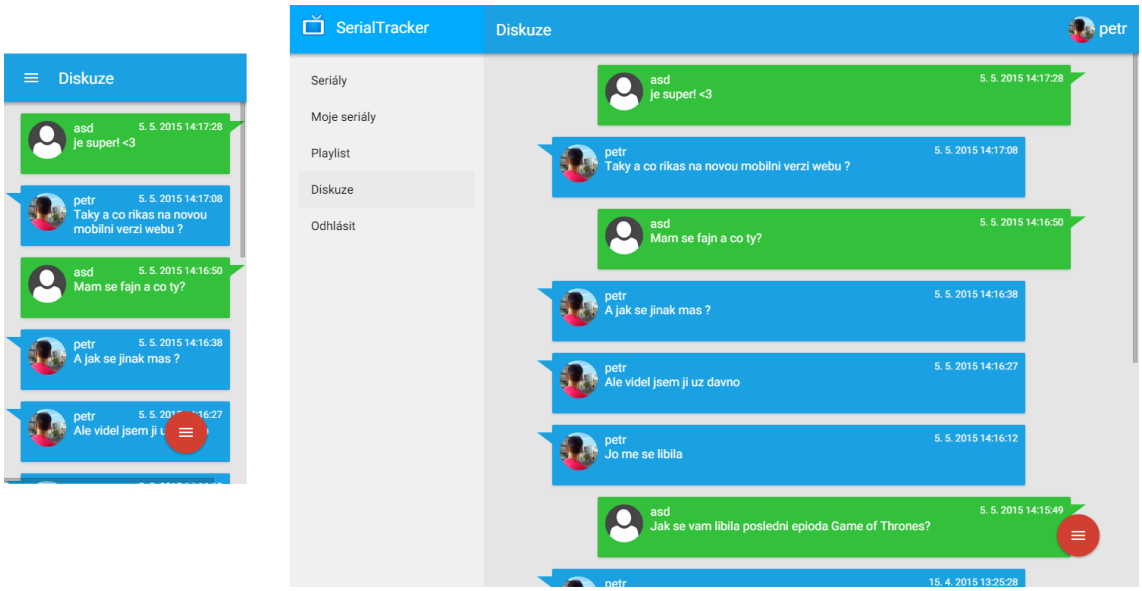


Obrázek 20 – Náhled playlistu

Na výše uvedeném obrázku (viz Obrázek 20) je náhled na playlist v aplikaci. Tato stránka uživateli nabízí rychlý přehled, na kterou další epizodu daného seriálu se má podívat. Funkce je velice praktická, pokud uživatel sleduje více seriálů najednou – ve filtrovaném přehledu nových epizod může být nepřehledné hledat následující epizodu konkrétního seriálu.

5.1.5 Diskuze

Na této stránce (viz Obrázek 21) je uživatelům k dispozici diskuze, kde mohou volně diskutovat o seriálech či psát různé připomínky k webu.



Obrázek 21 – Náhled diskuze

Barvy zpráv se liší podle toho, jakou mají uživatelé roli. Pokud jsou moderátoři či adminové, mohou přidávat modrou zprávu, jako formu odpovědi. Ostatní uživatelé mohou přidávat pouze klasickou tedy zelenou zprávu. Přidání zprávy je umožněno po kliknutí na plovoucí akční tlačítko v pravém dolním rohu stránky. Toto tlačítko se při vertikálním posouvání stránky dolů skrývá a směrem nahoru zase odkrývá, aby byla umožněna čitelnost zpráv i za tímto tlačítkem. Pokud uživatel nemá žádnou roli, tlačítko vyvolá rovnou dialogové okno pro vytvoření zprávy, pro role, které umožňují přidání odpovědi, je po kliknutí na tlačítko k dispozici jednoduchá nabídka (viz Obrázek 22).

Obrázek 22 – Náhled na tlačítka pro přidání zprávy do diskuze. Vlevo pro moderátory, vpravo pro běžné uživatele

5.2 Knihovna Polymer

Pro vývoj aplikace byla použita knihovna Polymer, jak již bylo zmíněno výše. Knihovna je produktem firmy Google a je zveřejněna pod BSD licencí pro svobodný software. V současnosti je knihovna ještě stále ve fázi vývoje, avšak termín produkční verze je stanoven na léto roku 2015. Tato knihovna umožňuje vytvářet vlastní HTML elementy za využití standardu HTML5 – Web Components. Tyto vlastní elementy mají stejné vlastnosti jako klasické elementy:

- prohlížeč ví, jak má elementy interpretovat ve webové stránce,
- jsou znovupoužitelné kdekoliv na stránce,
- lze s nimi manipulovat pomocí JavaScriptu,
- jsou zapouzdřené, tak že neovlivní okolní elementy,
- jsou konfigurovatelné pomocí HTML atributů,
- jsou programovatelné pomocí JavaScriptu,
- generují vlastní události,
- lze je do sebe různě zanořit.

Tvorbou vlastních elementů lze jednoduše a sémanticky web rozdělit. Při klasickém způsobu kódování webové stránky, kdy jednotlivé části stránky jsou z pravidla tvořeny velkým množstvím elementů (které dohromady nemají žádný význam), lze tyto části nahradit jediným elementem, z jehož jména vyplývá i jeho funkce, a to vede ke zlepšení čitelnosti zdrojového kódu. Element může být různě veliký, od malých tlačítek přes formuláře až po celou aplikaci, také nemusí být vůbec viditelný a má pouze programovou funkci.

5.2.1 Podpora

Ne všechny prohlížeče však v současné době podporují standard HTML5 – Web Components (viz Příloha D), v takovém případě je nutné do stránky přidat tzv. *polyfill* či *polyfiller*. Jedná se o kód, jenž supluje funkce, které nejsou nativně implementovány v prohlížeči.

```
if ("registerElement" in document
  && "createShadowRoot" in HTMLElement.prototype
  && "import" in document.createElement("link")
  && "content" in document.createElement("template")) {
  // prohlizec ma nativni podporu WebComponents
} else {
  document.write('<script
src="bower_components/webcomponentsjs/webcomponents.js">
<\/script>');
}
```

Zdrojový kód 29 – Kontrola nativní podpory Web Components

Ve výše uvedeném zdrojovém kódu (viz Zdrojový kód 29) je testováno, zda prohlížeč tento standard podporuje. V podmínce je kontrolováno, zda prohlížeč má nativní podporu klíčových vlastností standardu:

- registrování nového elementu,
- vytvoření vnořeného DOM objektu,
- importování souborů,
- vytváření HTML šablon.

Pokud není podmínka splněna, tedy prohlížeč nepodporuje všechny vlastnosti, bude do stránky přidán kód pro *polyfill*.

5.3 Tvorba elementu v knihovně Polymer

Každý element je popsán ve svém vlastním souboru pomocí dalších HTML elementů a volitelně JavaScriptového kódu, který dává elementu dynamičnost.

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

<polymer-element name="hello-world">
  <template>
    <span>Hello world!</span>
  </template>
  <script>
    Polymer({ });
  </script>
</polymer-element>
```

Zdrojový kód 30 – Příklad jednoduchého elementu

Na začátku souboru daného elementu by měl být uveden odkaz na knihovnu, aby bylo možné využít funkce knihovny. Ve výše uvedené ukázce (viz Zdrojový kód 30) je definován velice jednoduchý element, který dokáže na stránku pouze vypsat pozdrav „Hello world!“.

5.4 Použité elementy v aplikaci

Tvůrci knihovny Polymer vytvořili poměrně obsáhlou kolekci HTML elementů, které jsou rozděleny do dvou skupin. První skupina jsou tzv. *core* elementy, které mají jak vizuální tak i programovou funkci. Druhou skupinou jsou *paper* elementy, jedná se o vizuální elementy, které splňují požadavky Material designu[2], do něhož jsou v současné době migrovány Google služby i jeho operační systém Android. Díky těmto elementům tak lze jednoduše vytvořit webovou aplikaci, která je sladěna s designem operačního systému a působí nativnějším dojmem.

[2] *Material design - Google design guidelines* [online]. 2015 [cit. 2015-05-01]. Dostupné z: http://www.google.com/design/spec/material-design/introduction.html

V aplikaci byly použity elementy z obou skupin, dále elementy vyvinuté dalšími vývojáři a elementy vlastní. Vybrané elementy budou popsány níže v této kapitole.

5.4.1 Repositář elementů

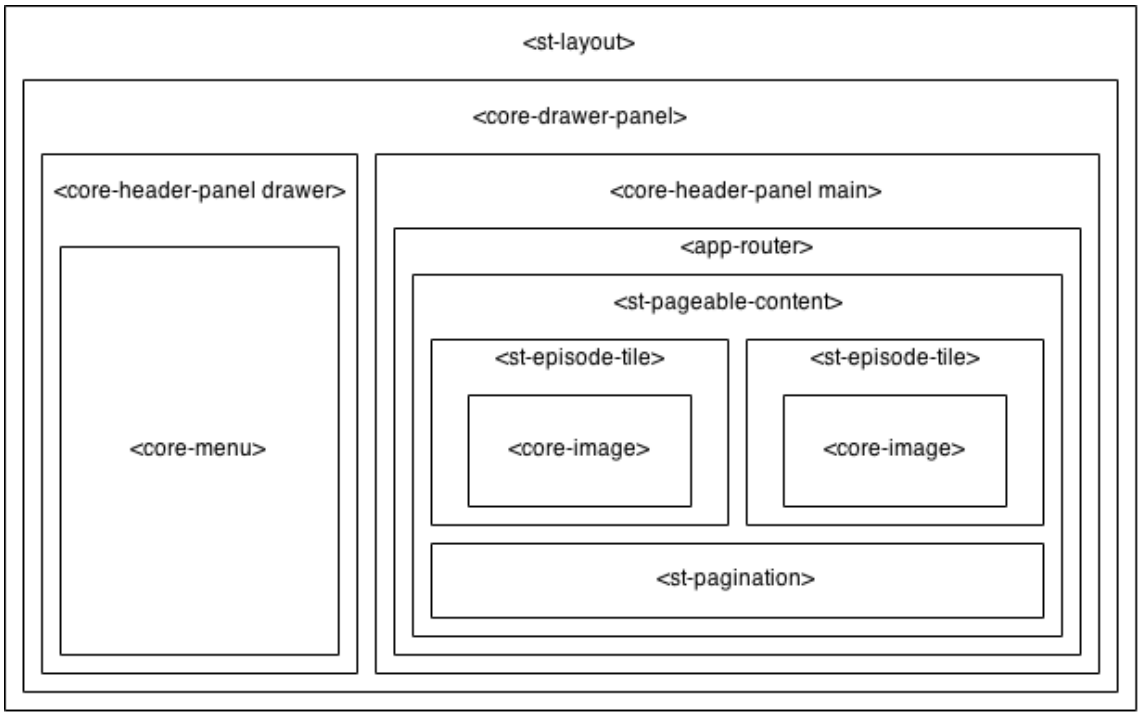
Pro tvorbu mobilního klienta bylo použito hned několik elementů od dalších vývojářů. Zdrojové kódy elementů je možné stáhnout jako klasické soubory z veřejného repositáře GitHub [30], nebo je nainstalovat (viz Zdrojový kód 31) do své aplikace pomocí balíčkovacího manažeru Bower [31]. Bower zdrojové kódy také stahuje z výše zmíněného repositáře, avšak v aplikaci vytváří soubor, který nese informace o nainstalovaných balíčcích, jejich verzích a závislostech na další balíčcích. Pokud tedy nějaký element potřebuje ke své funkcí další elementy, manažer je stáhne také. Další výhodou je snadná hromadná aktualizace těchto balíčků, či získání jiné konkrétní verze.

```
bower install --save Polymer/core-ajax
```

Zdrojový kód 31 – Příklad instalace elementu pomocí Bower

5.4.2 Použité elementy

Celá aplikace klienta je složena z velkého množství různých elementů, které dohromady dávají celistvou aplikaci. Zjednodušená struktura vizuálních elementů na hlavní stránce aplikace je vyobrazena na následujícím obrázku (viz Obrázek 23) a odpovídá vzhledu stránky (viz 5.1.2 Odvysílané epizody). Následující podkapitoly se budou týkat vybraných vizuálních a programových elementů použitých v aplikaci. Názvy podkapitol budou odpovídat skutečným názvům elementů.



Obrázek 23 – Zjednodušená struktura aplikace na hlavní stránce

st-layout

Celá webová aplikace je zabalena do elementu, který definuje základní rozložení stránky, jako je například umístěn menu na levé straně.

Pro rozdělení stránky na dvě části – navigační a obsahovou část – byl použit element *core-drawer-panel*, který byl vytvořen autory knihovny a který umožňuje skrytí navigační části při nízké velikosti displeje, která může být vysunuta z kraje obrazovky gestem či programově.

app-router

Tento element [32] umožňuje navigaci napříč aplikací bez obnovení stránky prohlížečem a jeho autorem je Erik Ringsmuth. Umožňuje definovat

(viz Zdrojový kód 32) vzory URL adres, pro které bude do stránky dynamicky načten obsah.

```
<app-router>
  <app-route path="/serials/:currentPage"
    import="pages/st-serials.html" data-title="Seriály"
    element="st-serials">
  </app-route>
</app-router>
```

Zdrojový kód 32 – Definování URL adresy pro obsah Seriály

st-pageable-content

Tento element je použit (vit Zdrojový kód 33) napříč celou aplikací, jelikož umožňuje automaticky stahovat a stránkovat data z aplikačního rozhraní a vykreslovat je.

```
<st-pageable-content
  urltemplate="/Episodes/Page-{page}"
  appurltemplate="/serials/{page}"
  currentpage="{{currentPage}}"
  dataname="episode"
  grid>
    <template>
      <st-episode-tile data="{{model}}">
      </st-episode-tile>
    </template>
</st-pageable-content>
```

Zdrojový kód 33 – Použití stránkovacího elementu

Elementu jsou zadány následující atributy:

- *urltemplate* – šablona URL adresy, na které se má element dotazovat aplikačního rozhraní na data,
- *appurltemplate* – šablona URL adresy, která má být použita při navigaci v aplikaci,
- *currentpage* – číslo stránky,
- *dataname* – jméno položky, pod kterým jsou v odpovědi od serveru uložena data,

- grid – pokud je přítomen tento atribut, jednotlivé položky budou zobrazeny v mřížce místo pod sebou.

Uvnitř elementu je umístěna šablona, která definuje, jak má být jedna konkrétní položka zobrazena na stránce. Celý element na základě informací z odpovědi od serveru doplní pod obsah také stránkovací tlačítka.

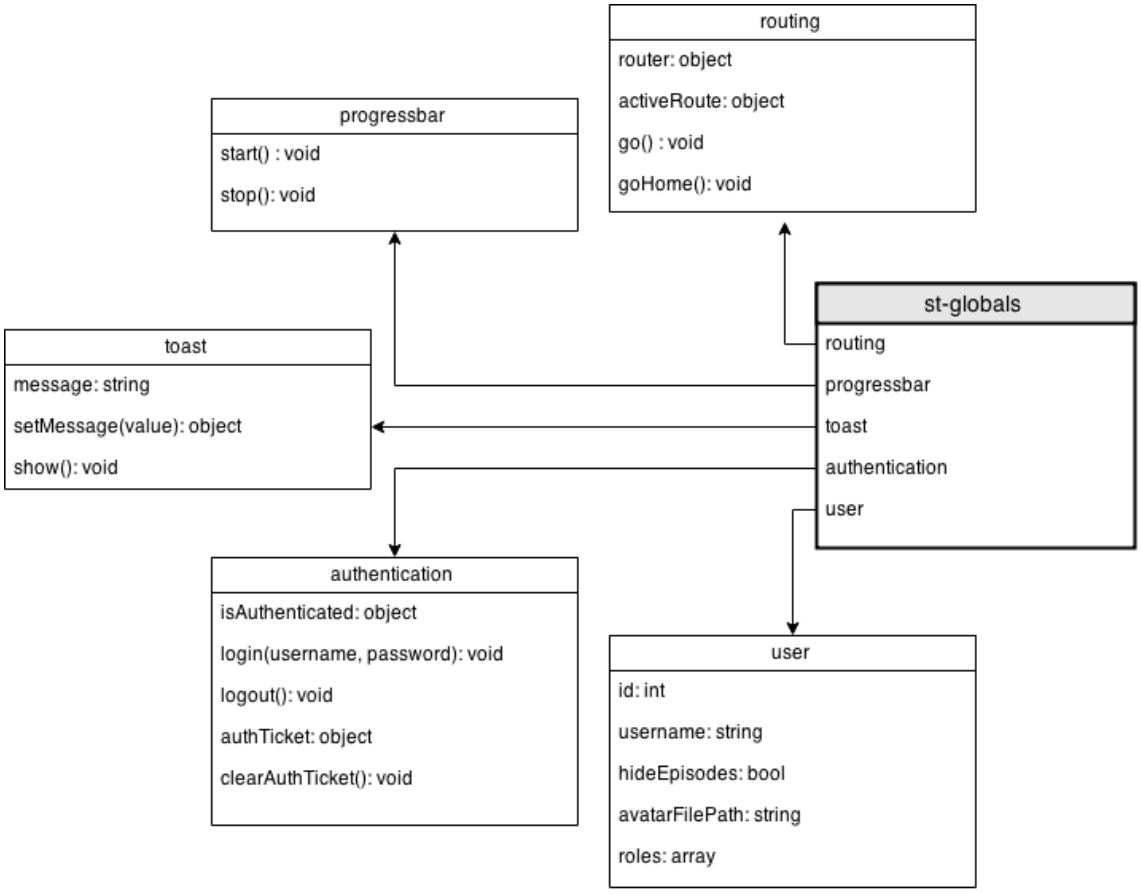
Tento element byl odvozen z *core-list* elementu, který umožňuje zobrazovat seznamy dlouhé i několik tisíc položek. Takovéto množství HTML kódu by mohlo být příliš náročné pro některé prohlížeče, a to obzvláště pro ty mobilní. Tento element však negeneruje kód pro všechny jeho položky, ale pouze nezbytný počet elementů k zaplnění viditelné stránky. Posouvání stránky je tedy pouze virtuální a dochází pouze ke změně dat uvnitř šablon položek. Díky tomu je dosaženo velice plynulého posouvání až 60 snímků za vteřinu. Další vlastností je, že pokud šablony obsahují element pro obrázek (*core-image*), pak se tyto obrázky stahují až v tom případě, kdy jsou viditelné na stránce a ne pro celý seznam. Takto je velice efektivně uspořen datový přenos aplikace.

st-ajax

Ačkoli jsou AJAX dotazy klasicky tvořeny v JavaScriptu, myšlenkou standardu je: „Všechno je element“, a tak i pro tvorbu dotazu je vytvořen element. *St-ajax* byl odvozen od *core-ajax* elementu a rozšiřuje ho o nové události, automatickou konverzi JavaScriptových objektů do JSON řetězce a dále o pevně dané HTTP hlavičky nutné pro komunikaci s aplikačním serverem.

st-globals

Tento element slouží ke globálnímu ovládání aplikace. Poskytuje přístup například k routeru pro navigování v aplikaci, poskytnutí informací o uživateli, vyvolání informační zprávy či lze použít k ovládání progressbaru. Element má pouze programovou funkci, a tak není na stránce nijak zobrazen. Zjednodušená objektová struktura elementu je znázorněna na obrázku níže (viz Obrázek 24).



Obrázek 24 – Objektová struktura elementu st-globals

Pro návrh objektu byl použit návrhový vzor Monostate pattern (viz Zdrojový kód 34), který umožňuje při vytváření nových instancí manipulovat stále se stejnými daty. Ačkoli je element instanční, jeho atributy jsou statické napříč všemi instancemi. Tak je možné mít při několikanásobném použití elementu kdekoliv na stránce přístup k již vytvořeným datům, jako jsou informace o uživateli či DOM objekty elementů.

```
(function() {
  //staticke atributy
  var userModel = {
    id: 0, //id
    username: "", //uzivatelske jmeno
    hideEpisodes: false, //skryt zhlednute epizody
    avatarFilePath: "", //cesta k avatarovi
    roles: [] //role
  };

  Polymer({
    user: userModel, //staticky atribut do instacniho
  });
})();
```

Zdrojový kód 34 – Monostate patter v jazyce JavaScript

Závěr

Prvním výsledkem této práce je rozšíření exitující webové aplikace na webovou službu. Rozhraní aplikace pokrývá prakticky všechna data, která aplikace uživatelům nabízí. Přínosem je, že je možné pro aplikaci vyvinout další návazné aplikace, které tato data využívají. V současné době existuje minimálně jedna aplikace, která tato data těží parsováním přímo z HTML, rozhraní tak ulehčí a urychlí vývoj.

Druhým výsledkem práce je vytvoření webového mobilního klienta této aplikace. Díky tomu, že se jedná o webové řešení, je tak mobilní aplikace multiplatformní. Při vývoji klienta došlo také k zapojení do komunitního vývoje některých použitých elementů, a byly tak opraveny jejich nedostatky, chyby či přidáno nějaké rozšíření.

Klient i rozhraní je připravené na nasazení do produkční verze a oba projekty budou podléhat dalšímu vývoji a úpravám podle ohlasů uživatelů.

Zdroje

[1] FIELDING, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. [online]. *University of California, 2000* [cit. 2015-03-15]. Dostupné z:https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[2] REST API Resources. *RESTful Web Services Resources* [online]. 2013 [cit. 2015-03-16]. Dostupné z: http://www.restapitutorial.com/resources.html

[3] Method Definitions. *Hypertext Transfer Protocol -- HTTP/1.1* [online]. [cit. 2015-03-20]. Dostupné z: http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[4] Status Code Definitions. *Hypertext Transfer Protocol -- HTTP/1.1* [online]. 2004 [cit. 2015-03-22]. Dostupné z: http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[5] HAL - Hypertext Application Language. *Stateless API Consulting* [online]. 2011, 2013 [cit. 2015-03-20]. Dostupné z: http://stateless.co/hal_specification.html

[6] JSON-LD 1.0. *World Wide Web Consortium (W3C)* [online]. 2014 [cit. 2015-03-20]. Dostupné z: http://www.w3.org/TR/json-ld/

[7] Collection+JSON - Hypermedia Type : Media Types. *Amundsen.com* [online]. 2010, 24.2.2013 [cit. 2015-03-20]. Dostupné z: http://amundsen.com/media-types/collection/

[8] Siren: a hypermedia specification for representing entities. *GitHub* [online]. 2014 [cit. 2015-03-23]. Dostupné z: https://github.com/kevinswiber/siren

[9] On choosing a hypermedia type for your API - HAL, JSON-LD, Collection+JSON, SIREN, Oh My!. *Kevin Sookocheff : sookocheff.com* [online]. 2014, 11.3.2014 [cit. 2015-03-24].

Dostupné z: http://sookocheff.com/posts/2014-03-11-on-choosing-a-hypermedia-format/

[10] Cross-Origin Resource Sharing. *World Wide Web Consortium (W3C)* [online]. 2014 [cit. 2015-03-20]. Dostupné z: http://www.w3.org/TR/cors/

[11] XML-RPC Specification. *XML-RPC.Com* [online]. © 1998-2011 [cit. 2015-04-14]. Dostupné z: http://xmlrpc.scripting.com/spec.html

[12] SOAP Version 1.2 Part 0: Primer (Second Edition). *World Wide Web Consortium (W3C)* [online]. 2007 [cit. 2015-04-14]. Dostupné z: http://www.w3.org/TR/2007/REC-soap12-part0-20070427/

[13] WEYL, Estelle. *Mobile HTML5*. 1st ed. O'Reilly Media, 2013, xxvi, 450 pages. ISBN 1449311415.

[14] Quick Fix: Use New HTML5 Input Types to Simplify Forms. *UXcellence* [online]. 2014 [cit. 2015-04-20]. Dostupné z: http://uxcellence.com/2014/01/11/quick-fix-use-new-html5-input-types-to-simplify-forms/

[15] WebGL Water. *Projects - Made by Evan* [online]. 2011 [cit. 2015-04-20]. Dostupné z: http://madebyevan.com/webgl-water/

[16] Responsive Web Design. *A List Apart: For People Who Make Websites* [online]. 2010 [cit. 2015-04-19]. Dostupné z: http://alistapart.com/article/responsive-web-design/

[17] Do Modern Responsive Websites benefits Google ranking?. In: *Vinaora - Free Templates, Extensions and Tutorials* [online]. 2014 [cit. 2015-04-20]. Dostupné z: http://vinaora.com/2014/08/do-modern-responsive-websites-benefits-google-ranking/

[18] Manifest for a web application. *World Wide Web Consortium (W3C)* [online]. 2015 [cit. 2015-04-20]. Dostupné z: https://w3c.github.io/manifest/

[19] PhoneGap, Cordova, and what's in a name?. *PhoneGap* [online]. 2012 [cit. 2015-04-20]. Dostupné z: http://phonegap.com/2012/03/19/phonegap-cordova-and-what%E2%80%99s-in-a-name/

[20] 5 Best Mobile Web App Frameworks: Sencha Touch. *Modus Create | HTML5 Application Development & Training* [online]. 2014 [cit. 2015-04-21]. Dostupné z: http://moduscreate.com/5-best-mobile-web-app-frameworks-sencha-touch/

[21] *ASP.NET | The ASP.NET Site* [online]. 2015. [cit. 2015-05-02]. Dostupné z: http://www.asp.net/

[22] *Entity Framework | The ASP.NET Site* [online]. 2015. [cit. 2015-05-02]. Dostupné z: https://msdn.microsoft.com/cs-cz/data/ef

[23] WebApi.Hal. 2015. *GitHub* [online]. [cit. 2015-05-02]. Dostupné z: https://github.com/JakeGinnivan/WebApi.Hal

[24] Swagger UI. 2015. *GitHub* [online]. [cit. 2015-05-02]. Dostupné z: https://github.com/swagger-api/swagger-ui

[25] *Swagger | The World's Most Popular Framework for APIs.* [online]. 2015. [cit. 2015-05-02]. Dostupné z: http://swagger.io/

[26] Swashbuckle. 2015. *GitHub* [online]. [cit. 2015-05-02]. Dostupné z: https://github.com/domaindrivendev/Swashbuckle

[27] *NuGet Gallery* [online]. 2015. [cit. 2015-05-02]. Dostupné z: https://www.nuget.org/

[28] *Visual Studio - Microsoft Developer Tools* [online]. 2015. [cit. 2015-05-02]. Dostupné z: https://www.visualstudio.com/

[29] *Polymer* [online]. 2015 [cit. 2015-05-01]. Dostupné z: https://www.polymer-project.org/0.5/

[30] *GitHub* [online]. 2015 [cit. 2015-05-01]. Dostupné z: https://github.com/

[31] *Bower* [online]. 2012 [cit. 2015-05-01]. Dostupné z: http://bower.io/

[32] Router for Web Components. 2015. *GitHub* [online]. [cit. 2015-05-01]. Dostupné z: https://github.com/erikringsmuth/app-router

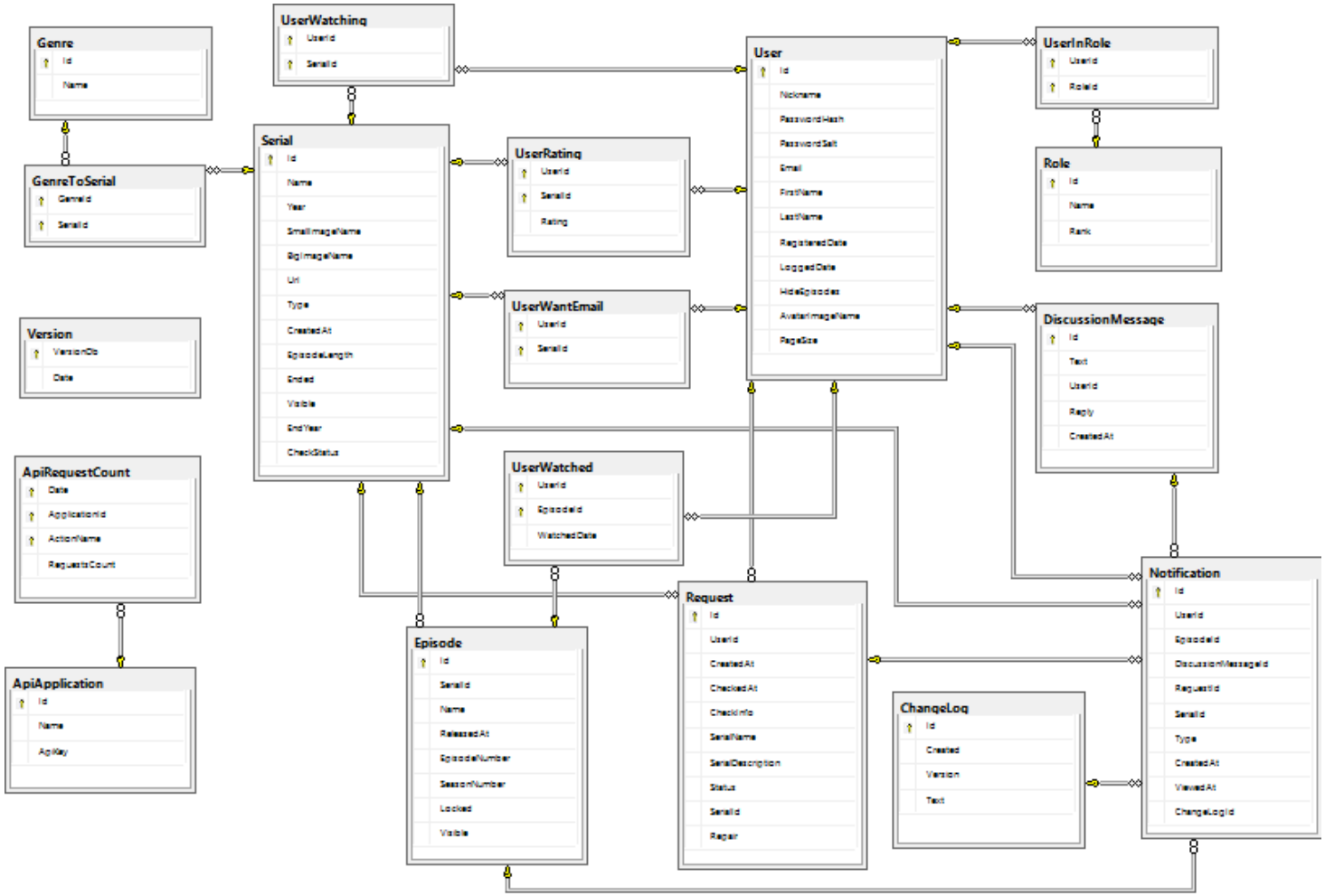
Příloha A **Podporované funkce ve Frameworku PhoneGap**

	iPhone / iPhone 3G	iPhone 3GS a novější	Android	Blackberry OS 6.0+	Blackberry 10	Windows Phone 8	Ubuntu	Firefox OS
Akcelerometr	✓	✓	✓	✓	✓	✓	✓	✓
Fotoaparát	✓	✓	✓	✓	✓	✓	✓	✓
Kompas	X	✓	✓	X	✓	✓	✓	✓
Kontakty	✓	✓	✓	✓	✓	✓	✓	✓
Soubory	✓	✓	✓	✓	✓	✓	✓	X
Geolokace	✓	✓	✓	✓	✓	✓	✓	✓
Média	✓	✓	✓	X	✓	✓	✓	X
Síť	✓	✓	✓	✓	✓	✓	✓	✓
Notifikace	✓	✓	✓	✓	✓	✓	✓	✓
Úložiště	✓	✓	✓	✓	✓	✓	✓	✓

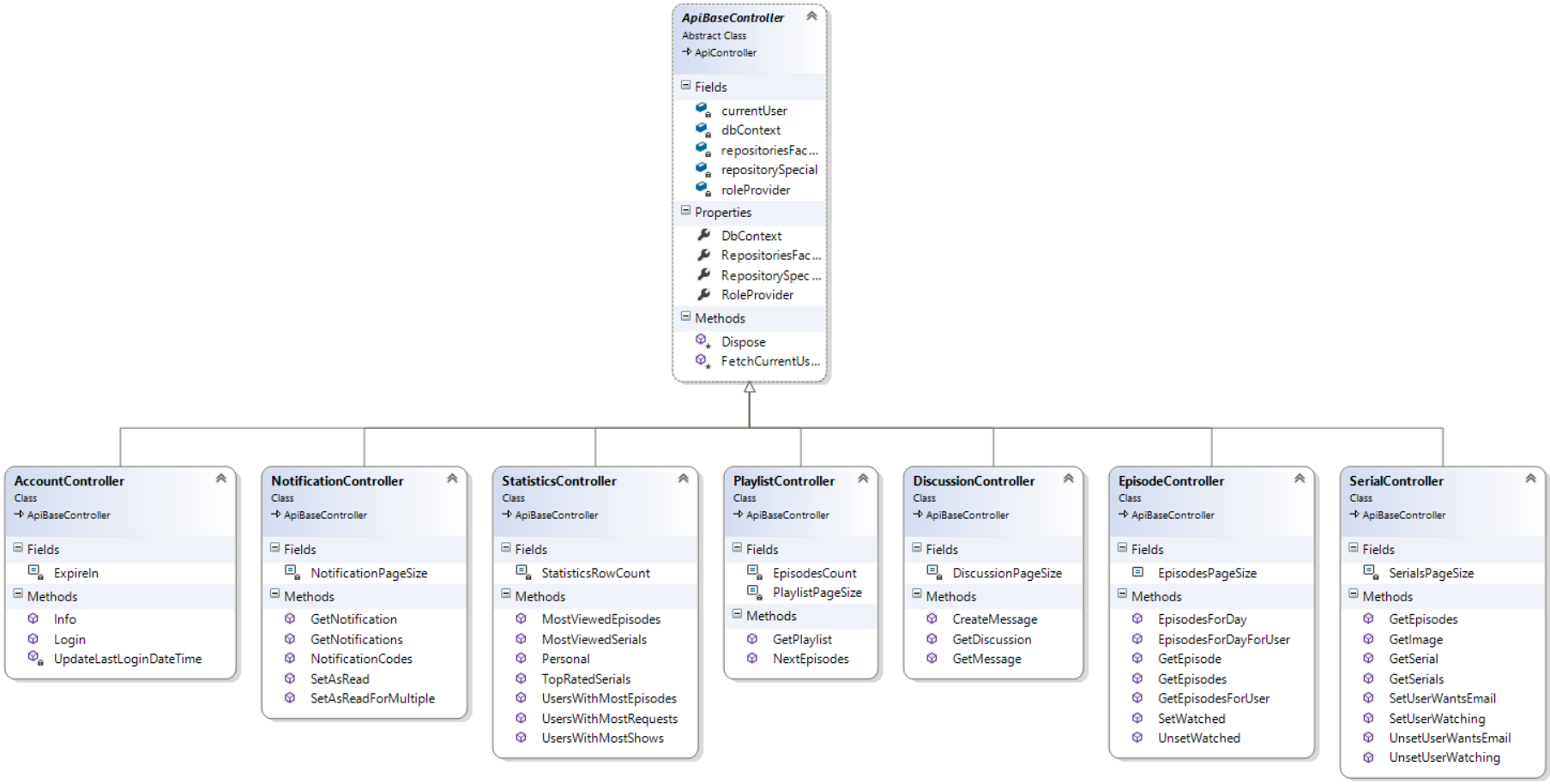
Tabulka byla převzata z oficiálních stránek[3] frameworku PhoneGap.

[3] Supported Features. *PhoneGap* [online]. © 2015 [cit. 2015-04-18]. Dostupné z: http://phonegap.com/about/feature/

Příloha B **Diagram databáze**



Příloha C **Class diagram controllerů rozhraní**



Příloha D **Podpora Web Components napříč prohlížeči**

	Chrome	Firefox	Opera	IE10+	Safari8+	Chrome (Android)	Safari (iOS 8.1)
HTML šablony	✓	✓	✓	X	✓	✓	✓
Importování souborů	✓	X	✓	X	X	✓	X
Registrace elementů	✓	X*	✓	X	X	✓	X
Vnořený DOM	✓	X*	✓	X	X	✓	X

*Ve výchozím stavu je funkce vypnuta

Tabulka byla převzata z informací na webu *Can I Use*[4], která poskytuje informace o podpoře nových technologií napříč prohlížeči.

[4] *Can I use... Support tables for HTML5, CSS3, etc* [online]. 2015 [cit. 2015-05-01]. Dostupné z: http://caniuse.com/

Příloha E **Obsah přiloženého média**

- Zdrojové kódy webové aplikace SerialTracker včetně aplikačního rozhraní.
- Zdrojové kódy webového klienta.
- Kompletní dokumentace aplikačního rozhraní v PDF.
- Bakalářská práce *Mobilní klient pro existující web* v PDF.