VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

NATIVNÍ APLIKACE V JAVASCRIPTU

NATIVE APPLICATION WITH JAVASCRIPT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **CYRIL URBAN**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. RADEK BURGET, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Urban Cyril**
Program: Informační technologie
Název: **Nativní aplikace v JavaScriptu**
Native JavaScript Applications
Kategorie: Informační systémy

Zadání:

1. Seznamte se s problematikou návrhu a implementace aplikací klient-server na platformě JavaScript.
2. Prostudujte existující technologie pro tvorbu mobilních klientských aplikací přenositelných na různé platformy. Zaměřte se zejména na technologii NativeScript.
3. Po dohodě s vedoucím navrhněte architekturu demonstrační aplikace klient-server za účelem praktického testování vybraných serverových i klientských technologií.
4. Implementujte navrženou aplikaci.
5. Proveďte testování vytvořené aplikace a ověřte přenositelnost řešení.
6. Zhodnoťte dosažené výsledky.

Literatura:

- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Brno: Computer Press, 2015
- Anderson, N. J.: Getting Started with NativeScript. Birmingham: Packt Publishing, 2016

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Burget Radek, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 30. října 2018

Abstrakt

Tato bakalářská práce zkoumá existující technologie pro tvorbu mobilních aplikací typu klient-server přenositelných na různé platformy mobilního operačního systému. Zvýšená pozornost je věnovaná především nejnovějším trendům v podobě využití technologií pro tvorbu nativních, multiplatformních aplikací v jazyku JavaScript, které jsou za běhu interpretované do nativního kódu, zejména pak framework NativeScript. Praktická část se věnuje návrhu a implementaci ukázkové aplikace, která demonstruje zvolené technologie.

Abstract

This bachelor thesis examines existing technologies for the creation of client-server mobile applications that can be used in different platforms of mobile operating systems. It is focused especially on the latest trends in the use of technologies for creating native, cross-platform JavaScript applications that are interpreted into native code, especially the NativeScript framework. The practical part deals with the design and implementation of a sample application that demonstrates the chosen technologies.

Klíčová slova

multiplatformní mobilní aplikace, klient-server, JavaScript, NativeScript, Node.js, Express.js, ES6

Keywords

cross-platform mobile applications, client-server, JavaScript, NativeScript, Node.js, Express.js, ES6

Citace

URBAN, Cyril. *Nativní aplikace v JavaScriptu.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Burget, Ph.D.

Nativní aplikace v JavaScriptu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Burgeta Ph.D. a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Cyril Urban
24. března 2019

Poděkování

Rád bych poděkoval panu doktoru Burgetovi za vedení práce a za odborné rady. Dále bych chtěl poděkovat všem, kteří se podíleli na testování výsledné mobilní aplikace.

Obsah

Kapitola 1

Úvod

Používání chytrých mobilních telefonů se v posledních letech neustále zvyšuje [9]. S touto skutečností jde ruku v ruce také vývoj mobilních aplikací. Je důležité uvědomit si, že na současném trhu existuje více než jedna platforma operačního systému mobilního telefonu, pro kterou je možné aplikace vyvíjet. Právě tento fakt výrazně přispěl k novým, moderním metodám, jak vyvíjet mobilní, multiplatformní aplikace a prostudování těchto moderních přístupů, obzvláště těch využívající jazyk *JavaScript* jsou cílem této bakalářské práce.

V kapitole 2 budou probrány všechny běžné možnosti, jak vyvíjet mobilní aplikace a to od původních, konzervativnějších přístupů až po nejmodernější trendy, které pomocí jazyka JavaScript dokáží vytvořit skutečně nativní, multiplatformní mobilní aplikace. Existují dva významní zástupci reprezentující tento styl vývoje – *NativeScript*[1] a *React Native*[2]. Součástí této kapitoly je tedy také jejich detailnější popis a vzájemné porovnání. Zajímavým trendem v oblasti vývoje mobilních aplikací je také to, rozdělit vývoj na klientskou a serverovou část. Tento přístup má řadu výhod, které také dostanou prostor k vysvětlení, včetně popisu konkrétních technologií jako je serverový *Node.js*[3].

Zadání demonstrační aplikace, na které jsou prakticky ukázány použité technologie z předešlé kapitoly, je popsáno v kapitole 3. Nechybí ani porovnání s již existujícími aplikacemi a dále návrh klientské mobilní aplikace, serverové části včetně databázového modelu a taktéž navržení komunikace mezi klientem a serverem komunikujícím přes aplikační rozhraní. Kapitola 4 pojednává o samotné implementaci navržené mobilní aplikace. Zde jsou popsány nejzajímavější a nejnáročnější poznatky z vývoje a to jak ze serverové tak z klientské části. V kapitole 5 je popsáno testování, které v dnešní době tvoří důležitý faktor vývoje. Bude zde ukázáno, jak probíhalo testování během vývoje a také finální testování reálných uživatelů. Poslední, 6. kapitola je tradičně věnována závěru, kde je shrnuta studie o technologiích a implementaci mobilní aplikace. Dále je zde zmíněna potencionální možnost rozšíření.

[1] NativeScript – https://nativescript.org/
[2] React Native – http://reactnative.com/
[3] Node.js – https://nodejs.org/

Kapitola 2

Technologie aplikací na platformě JavaScript

V této kapitole budou probrány technologie a popis architektury aplikací *klient-server*. V posledních letech se začíná projevovat zajímavý potenciál využití jazyka *JavaScript* a to i v aplikacích typu klient-server a proto zde tento jazyk bude blíže popsán.

Největší část této kapitoly pojednává o různých přístupech vývoje klientských mobilních aplikací s převážným zaměřením na nové trendy, které představují *NativeScript* a *React Native*. Tyto dvě technologie jsou blíže popsány a porovnány.

V závěru této kapitoly jsou popsány zvolené technologie pro následný vývoj demonstrační aplikace, zde jsou blíže vysvětleny technologie jako *ECMAScript 2015*[1], serverový *Node.js* či knihovna na propojení s různými databázemi *Knex.js*[2].

2.1 Architektura klient-server na platformě JavaScript

Některé typy mobilních aplikací nepotřebují pro svůj plnohodnotný chod připojení k dalším serverům za účelem přístupu k datům či synchronizaci, takovou architekturu lze nazívat jako osamocené mobilní aplikace. Nicméně velká část mobilních aplikací pro své fungování musí komunikovat se vzdálneým serverem a to například za účelem sdílení či získávání dat. Na serverovou část je kromě dat možné umístit i logiku aplikace. Tato architektura se nazývá klient-server.

Jazyk JavaScript v posledních letech prošel velkou proměnou a značnou rozšířitelností jeho použití. Z původního nástroje pro animace webů se dostal až k jazyku pro tvorbu složitých webových aplikací, ale také se přesunul na jiné klientské platformy jako jsou mobilní či desktopové aplikace. Prostřednictvím jazyka JavaScript je dále možné plnohodnotně napsat i serverovou část aplikace. To z tohoto jazyka tvoří dobrého kandidáta na vývoj aplikací typu klient-server.

2.1.1 Architektura klient-server

Klient-server je síťová architektura, která odděluje klientskou a serverovou část komunikují přes počítačovou síť. Tento model je známý především ze síťových technologií, kde ho

[1] ECMAScript 2015 – https://www.ecma-international.org/ecma-262/6.0/
[2] Knex.js – http://knexjs.org/

využívají internetové protokoly jako *HTTP*[3] či *DNS*[4] nebo z databázových systémů, kde probíhá komunikace pomocí strukturovaného dotazovacího jazyka. Původně se tato architektura využívala k propojování terminálů a sálového počítače, avšak s rozvojem osobních počítačů a zrychlením sítě Internet se teprve nedávno tato architektura [11] plnohodnotně rozšířila do oblasti informačních systémů.

Nosná myšlenka této architektury, jak již z názvu vypovídá, je rozdělení klientské a serverové části na dva oddělené celky, které mohou býti vyvíjeny dvěma různými technologiemi, což může využít potenciál jednotlivých programovacích jazyků a také vyvíjena dvěma vývojovými týmy, což může přinést snadné a efektivní rozdělení práce. Klient a server spolu komunikují pomocí určeného aplikačního rozhraní, přes které si zasílají zprávy. Typickým příkladem klientské části je webový prohlížeč, který prostřednictvím webové aplikace generuje požadavky a zasílá je ve tvaru vhodném pro specifické aplikační rozhraní na server. Serverová část tento požadavek zpracuje a na klientskou část odešle potřebnou odpověď (data), na základě které klientská část interpretuje uživateli nový výsledek. Výhodou v kombinaci s technologií *Ajax*[5] je, že není třeba vždy překreslovat celou stránku, ale pouze její část, která byla asynchronně změněna odpovědí serveru z požadavku klienta.

Mezi příklady serverové části lze zařadit webové, databázové či e-mailové servery. Mezi další zástupce klientské části, kromě zmíněného webového prohlížeče obvykle patří mobilní nebo desktopové aplikace. Právě variabilita více klientů může být klíčovým faktorem pro zvolení architektury klient-server, jelikož je možné napsat logiku aplikace jenom jednou na stranu serveru a k němu se dotazovat prostřednictvím několika klientů v podobě například webové a mobilní aplikace. Zvláště výhodné je poté využít tzv. *tenkého klienta*, který na rozdíl od toho *tlustého* na klientské části využívá jen nezbytné minimum algoritmů a naprostá většina funkcionality systému je ukryta na straně serveru.

Výhody

- Snadné rozšíření na více platforem (webové, mobilní, desktopové a další aplikace)
- Možnost použití více technologií
- Snadnější údržba, například je možné modernizovat či nahradit server bez ovlivnění klientů
- Mnohdy i vyšší bezpečnost, jelikož je většina dat uchovaných na serverové části, která je zpravidla více zabezpečená, než klientská část

Nevýhody

- Může dojít k přetížení serverů, když dochází k častým požadavkům klientů současně
- Menší robustnost, protože kdyby server selhal, klientské požadavky by nemohly být splněny a celý systém by se stal nefunkčním

[3] Hypertext Trasfer Protocol – aplikační protokol pro přenos obsahu, základ pro funkci World Wide Web
[4] Domain Name System – je hierarchický systém doménových jmen
[5] Asynchronous JavaScript and XML – technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich kompletního znovunačítání za pomoci asynchronního zpracování

2.1.2 Jazyk JavaScript a jeho využití

Jazyku *JavaScript* se v poslední době věnuje nemalá pozornost. JavaScript je multiplatformní, objektově orientovaný, funkcionální jazyk. Jeho syntaxe vychází z jazyka C, některá standardní rozhraní se podobají Javě, objektový a funkcionální model je inspirován jazyky Scheme a Self [12]. Do nedávna se nejednalo o nijak zvláště zajímavý jazyk, který se používal zpravidla jako interpretovaný programovací jazyk pro WWW[6] stránky, mnohdy vkládaný přímo do HTML[7] kódu, kde se používal primárně pro ovládání grafického uživatelského rozhraní nebo pro animace a efekty obrázků. V posledních několika letech však postupně nabírá na oblibě.

Velkou část popularity JavaScriptu přinesl příchod standardu *ECMAScript 2015*, který mimo jiné zavedl novou definici tříd, dědičností či zavedení Promises. Celkově se dá říci, že se jazyk zmodernizoval, podrobněji bude tato problematika probrána v sekci 2.7.2. Rozvoj využití JavaScriptu v oblasti webu se zvyšuje taktéž proto, že zprvu jednoduché animace webových prezentací přecházejí do čím dál složitějších webových aplikací. Značnou oblibu a snadnější použití JavaScriptu ve webových aplikací přinesl příchod nejrůznějších knihoven a frameworků[8], mezi ty nejznámější patří *React*[9], *jQuery*[10] nebo *Angular*[11].

O zajímavý rozmach tohoto jazyka se postaralo vydání *enginu V8*[12], který přinesl možnost použít JavaScript také na serveru a nahradit tak tradičnější serverové jazyky jako Javu či PHP. Mezi nejrozšířenější platformu pro serverový JavaScript patří bezesporu *Node.js*, díky kterému začala popularita JavaScriptu růst [8]. Mezi další serverové platformy v JavaScriptu patří například *Vert.x*[13].

Z Node.js později vznikly i zajímavé frameworky, které přenesly JavaScript i do oblasti vývoje desktopových aplikací, mezi příklady takovýchto frameworků patří *Electron*[14] nebo *NW.js*[15]. Jejich princip je, že využívají JavaScript, HTML a CSS[16] podobně jako na webu, pro tvorbu multiplatformních desktopových aplikací, které jsou kompatibilní s desktopovými operačními systémy *Windows*, *Mac* i *Linux*.

JavaScript se dále objevuje taktéž v aplikacích chytrých mobilních telefonů. Zde se používají dva přístupy, jak vyvíjet pomocí JavaScriptu. První a původní přístup v podstatě simuluje okno webového prohlížeče přes celou obrazovku, toho využívají frameworky jako *PhoneGap*[17] či *Ionic*[18]. Druhým, modernějším přístupem je využití frameworků pro tvorbu nativních aplikací v JavaScriptu, které jsou za běhu interpretované do nativního kódu, mezi hlavní zástupce patří *NativeScirpt* a *React Native*. Podrobněji bude tato problematika probrána v sekci 2.3.

[6] World Wide Web – označení pro systém prohlížení, ukládání a odkazování dokumentů nacházejících se v Internetu.

[7] Hypertext Markup Language – značkovací jazyk pro tvorbu webových stránek

[8] Framework – softwarová struktura, která slouží jako podpora při programování a vývoji za účelem vyřešení typických problémů, proto aby se vývojáři mohli soustředit pouze na své zadání

[9] React – https://reactjs.org/

[10] jQuery – https://jquery.com/

[11] Angular – https://angular.io/

[12] V8 – interpret jazyka JavaScript vyvinutý firmou *Google* pro jejich webový prohlížeč *Chrome*

[13] Vert.x – https://vertx.io/

[14] Electron – https://electronjs.org/

[15] NW.js – https://nwjs.io/

[16] Cascading Style Sheets – kaskádové styly, používají se pro nastavení způsobu zobrazení elementů na stránkách vytvořených v jazycích (X)HTML nebo XML

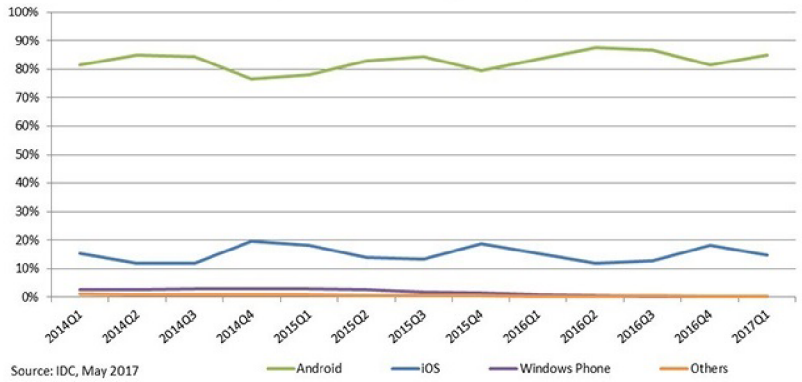
[17] PhoneGap – https://phonegap.com/

[18] Ionic – https://ionicframework.com/

2.2 Možné přístupy vývoje mobilních aplikací

S příchodem chytrých telefonů vzniklo nové odvětví v informačních technologiích a to vývoj mobilních aplikací. Je dobré si uvědomit, že je na trhu řada platforem mobilních operačních systémů, ale ne pro všechny se vyplatí vyvíjet. Podle *International Data Corporation* je aktuální odhadovaný podíl na trhu mobilních telefonů následující [3]:

- 85% Android
- 14,7% iOS
- 0,1% Windows Phone
- 0,2% ostatní



Obrázek 2.1: Vývoj podílu platforem mobilních operačních systémů na trhu

Z těchto dat vyplývá, že Android a iOS pokrývají zhruba 97,7% aktuálního trhu a proto je velmi důležité podporovat obě tyto platformy. Co se týče Windows Phone, tak zde už vývoj nových aplikací pomalu přestává dávat smysl, jelikož je jeho podíl velmi nízký a hlavně rok od roku nabírá velmi sestupnou tendenci, jak lze vidět na obrázku 2.1. Podíl ostatních platforem smysl příliš nedává.

Ač má toto odvětví krátkou historii, i tak stihlo projít zajímavým vývojem. Původní tvorba *nativních aplikací* pro každou platformu se postupem času tlakem na multiplatformnost rozrostla o několik dalších přístupů a to sice *tvorbou založenou na webu*, na *překládání z jiného programovacího jazyka* a konečně na přístupu pro *tvorbu nativních aplikací v JavaScriptu*, které jsou za běhu interpretované do nativního kódu. Nicméně pořád se nedá přímo říci, že je některý přístup lepší, než jiný, každý ma své výhody i nevýhody a může se hodit na jiné projekty.

2.2.1 Nativní aplikace

První a nejstarší metodou, jak vytvářet mobilní aplikace je vytvořit aplikaci nativně, zvlášť pro každou platformu. Toto řešení není sice přenositelné na více platforem, ale za to přináší nejpřímočařejší vývoj pro každou platformu. Je zde velká řada výhod, jako maximální využití funkcí a výkonu každé platformy. Výsledné aplikace pak budou nejrychlejší a nejvíce uživatelsky přívětivé. Tento přístup je taktéž vhodný na tvorbu složitějších aplikací, např. těch s využitím 3D grafiky.

Na druhou stranu je tu zásadní nevýhoda, a to nepřenositelnost na více platforem, což znamená, že pokud chce být aplikace podporovaná na N platformách je nebytné vytvořit N aplikací. To je hlavním důvodem, proč vznikly nové přístupy vývoje mobilních aplikací.

Programovací jazyky využívané na jednotlivých platformách pro nativní vývoj tohoto typu jsou pro Android Java nebo nově Kotlin[19], pro iOS Swift/ObjectiveC a pro Windows Phone C#/C++.

Výhody

- Nejvyšší možný výkon
- Podpora nejnovějšího aplikačního rozhraní pro ovládání telefonu
- Nativní UI prvky platformy

Nevýhody

Jediná, ale za to velmi zásadní nevýhoda je nepřenositelnost na více platforem, z které vyplývají problémy:

- Vysoká cena, protože je nezbytné vyvíjet pro každou platformu zvlášť
- Nároky na schopnosti programátorů, neboť není jednoduché najít programátora, který by ovládal hned několik programovacích jazyků
- Náročné rozšíření a opravy aplikací, jelikož je nutné editovat aplikaci na každé platformě

2.2.2 Multiplatformní aplikace založené na webových technologiích

Tento přístup je nejjednodušší způsob, jak vytvořit mobilní aplikaci pro více platforem. Celé to funguje na principu, že aplikace využívá optimalizované a zjednodušené (bez URL baru a navigačních tlačítek) okno webového prohlížeče, ve kterém jsou zobrazeny veškeré komponenty aplikace [10].

Největší výhoda tohoto přístupu je multiplatformnost a snadné použití. Vývojář v podstatě používá stejné technologie jako při vývoji webu - *JavaScript*, *HTML* a *CSS*. Nicméně nelze využít nativních UI komponent (tlačítka, textová pole) jednotlivých platforem a je na programátorovi, aby je vytvořil do podoby více podobné nativním platformám, avšak tento problém řeší za programátory celá řada frameworků. Dalším nedostatkem zůstává větší náročnost na výkon telefonu.

Je zde několik frameworků, založené na této technologii, mezi ty nejznámější patří *PhoneGap*, *Ionic* nebo *Apache Cordova*[20].

Výhody

- Multiplatformnost
- Snadný vývoj a jeho testování přímo ve webovém prohlížeči počítače
- Využití webových technologií, které jsou zpravidla lehčí na naučení oproti jazykům pro nativní vývoj

[19] Kotlin – https://kotlinlang.org/
[20] Apache Cordova – https://cordova.apache.org/

Nevýhody

- Chybějí nativní UI prvky, nicméně frameworky se snaží řešit jejich imitaci za nás
- Vyšší požadavky na hardware mobilního telefonu
- Možná nedostupnost některých aplikačních rozhraní telefonu

2.2.3 Multiplatformní aplikace překládané do nativního kódu

Za pomocí tohoto stylu vývoje jsou mobilní aplikace napsány v jednom programovacím jazyku a potom je zdrojový kód překládán do jednotlivých platforem [4]. Představitelem této technologie je *Xamarin*[21].

Výhodou těchto aplikací je to, že jsou skutečně nativní, to znamená, že všechny elementy UI, dialogy a podobně jsou na každé použité platformě skutečně reálné, což je rozdíl oproti aplikacím založených na webu, kde dochází pouze k jejich imitování. Výsledné aplikace mohou být taktéž plně výkonné a zároveň méně náročné na hardwarové vybavení telefonu.

K nevýhodám patří trochu méně příjemnější vývoj, protože je nutné po každé změně znovu přeložit celou aplikaci, což může být zdlouhavé. Další z nevýhod je ta, že se zde programátor mnohdy nevyhne použití programovacího jazyka cílové platformy za účelem doladění převážně grafického uživatelského rozhraní. Často se pak stává, že sdílena část aplikace v rámci platforem je pouze ta s logikou (algoritmy, manipulace s daty atd.).

Výhody

- Multiplatformnost a uživatelský pocit téměř jako kdyby to byla nativní aplikace
- Vysoký výkon, nižší nároky na hardware

Nevýhody

- Ne všechen zdrojový kód, lze sdílet mezi platformami, proto tedy nutnost pracovat s jazyky cílových platforem
- Částečně znepříjemněný vývoj, jelikož je třeba po každé změně znovu překládat celou aplikaci

2.2.4 Multiplatformní aplikace interpretované za běhu

Nejnovější směr ve vývoji mobilních aplikací přináší tvorba multiplatformních aplikací interpretovaných za běhu, tento směr je trochu podobný jako u překládaných aplikací, avšak v některých věcech se liší. Aplikace napsaná v jednom jazyku – *JavaScriptu* je pomocí frameworku za běhu interpretovaná do cílových platforem a za běhu taktéž volá jejich aplikační rozhraní [7]. Tyto rozdílná aplikační rozhraní jednotlivých platforem spojí do svého jednoho. Pak je tedy možné psát aplikaci v jednom jazyku a výsledkem je nativní aplikace.

Výhodou tedy je, že jsou k dispozici nativní UI komponenty jednotlivých platforem a není příliš potřeba zasahovat do úpravy grafického uživatelského prostředí jednotlivých platforem. Další výhodou je, že během vývoje, není po každé změně nutné přeložit celou aplikaci, ale pouze přenést tu část kódu, která se změnila a to díky povaze interpretovaného jazyka JavaScript.

[21] Xamarin – https://visualstudio.microsoft.com/xamarin/

Nevýhodou je to, že jsou tyto technologie velmi nové a pod stále velkým vývojem, což znamená, že ještě nejsou vyladěny všechny detaily a některé problémy nejdou vyřešit, tak přímočaře, jak to praví základní filozofie tohoto přístupu.

Frameworky pracující tímto stylem jsou *NativeScript*, *React Native* a méně známější *Titanium*[22].

Výhody

- Multiplatformnost a uživatelský pocit téměř jako kdyby to byla nativní aplikace
- Snadnější vývoj, synchronizace jen změněného kusu kódu
- Nativní vývoj v jazyku JavaScript, což může být pro řadu vývojářů přijatelnější, než například Swift nebo Java

Nevýhody

- Ne vše je zcela vyladěno a funguje tak, jak by mělo
- Pomalejší start aplikace a vyšší nároky na výkon hardwaru

2.3 Multiplatformní aplikace interpretované za běhu v JavaScriptu

Následující sekce se zaměří na dva v současnosti nejpoužívanější frameworky pracující na tomto principu a to sice *NativeScript* a *ReactNative*. I když toho mají hodně společné v řadě věcí se liší, budou tedy probrány jejich společné rysy, ale také jejich odlišnosti v detailnějším rozboru.

2.3.1 Princip funkcionality

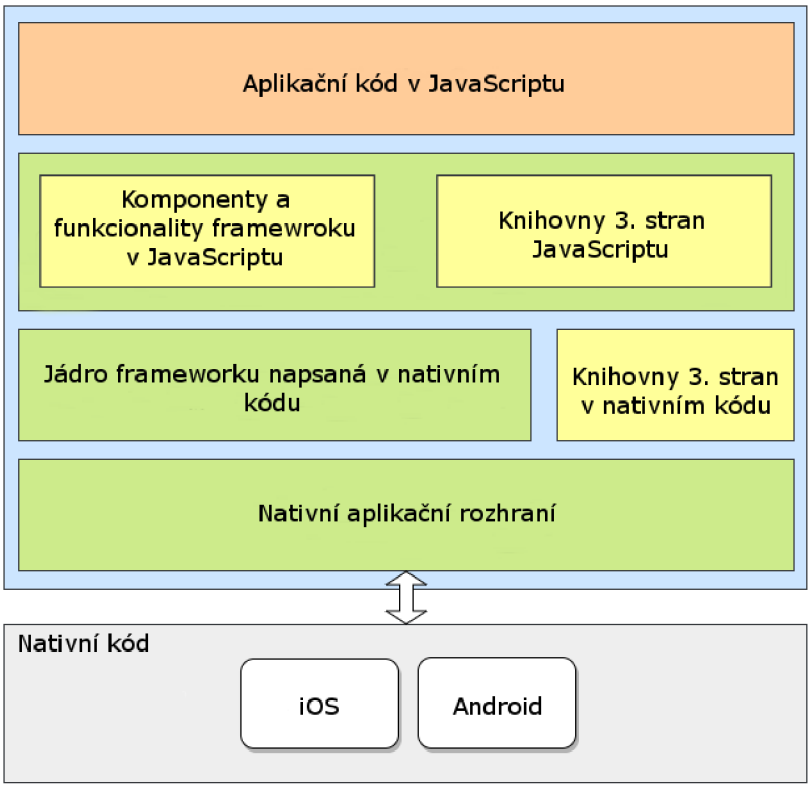
Základní architektura, kterou si je možné prohlédnout na obrázku 2.2 je stejná pro oba frameworky. Máme zde dva typy kódu a to nativní kód, který je rozdílný pro každou cílovou platformu a dále pak sdílený kód napsaný v JavaScriptu.

2.3.2 Nativní a sdílený JavaScriptový kód

Nativní kód je Objectiv-C pro iOS a Java pro Android, případně C#/C++ pro Windows Phone. Při kompilaci je aplikace překládána jako klasická nativní aplikace, kde nativní část tvoří především jádro frameworku, to tvoří základní funkcionalitu jako běh aplikace a také vytvoří kontejner pro JavaScriptový kód, který funguje jako komunikátor mezi JavaScriptovým a nativním kódem. Poté je možné volat veškerou funkcionalitu z JavaScriptu, za čímž si lze představit funkce frameworků, nativní aplikační kód a rozhraní či nativní knihovny 3. stran.

JavaScriptový kód je sdílen mezi všemi platformami a je za běhu interpretovaný s dosahem až k cílovým platformám. Pomocí JavaScriptu je tedy možné psát logiku uživatelské aplikace. Avšak JavaScript neslouží jen pro vývoj koncových uživatelů, taktéž velká část zmiňovaných frameworků NativeScript a React Native je napsána v tomto jazyku.

[22]Titanium – https://www.appcelerator.org/#titanium



Obrázek 2.2: Základní architektura porovnávaných frameworků

JavaScript s nativním kódem komunikuje po celou dobu běhu aplikace a to oběma směry. Při volání požadavků je JavaScriptový kód přeložen do nativního kódu a následná odpověď cílové platformy je z nativního kódu přeložena zpět do toho JavaScriptového. Výhodou při vývoji taktéž je, že při změně některého souboru, není potřeba překládat celou aplikaci, ale pouze ten jediný změněný soubor, což je velmi rychlé. Dokonce se o to vůbec nemusíme starat, jelikož to zvládnou frameworky za nás, které změnu detekují automaticky po uložení souboru.

2.3.3 Přístup k nativním funkcím

Oba porovnávané frameworky umožňují přistupovat k aplikačnímu rozhraní nativní platformy skrze JavaScript. V podstatě to funguje na principu, že v rámci JavaScriptu definuji například element s textovým vstupem a tento element je skrze volání nativního aplikačního rozhraní vykreslen rozdílně do každé cílové platformy. Může se však stát, že některá komponenta, specifický dialog a tak dále není takto pohodlně připraven frameworkem, nicméně ještě to není důvod k obavám, protože je tu možnost vytvořit si takovou komponentu sám pro všechny cílové platformy a pak jí spojit do jedné v JavaScriptu. Nejedná se až zas tak o nic nedosažitelného, protože jde zpravidla jen o již existující komponenty z každé nativní platformy a pak dochází pouze ke sjednocení aplikační rozhraní v rámci JavaScriptu.

Taktéž stojí za zmínku, že do frameworků existuje celá řada doplňujících modulů 3. stran, z nichž se velká část zabývá právě tímto problémem. Celkově je tento přístup obrovskou výhodou, protože zde není téměř nic nemožné, maximálně jen náročnější.

2.3.4 Využití frameworku nad JavaScriptem

Trendem dnešní doby je použití frameworku napříč spektrem programovacích jazyků. Framework je v podstatě sada nástrojů a základních funkcí, které jsou při vývoji aplikaci běžně potřebné a snaží se je dělat za koncového vývojáře.

Vzhledem k tomu, že zmiňovaný *NativeScript* a *React Native* poskytují spíše zakládaní funkcionalitu pro spuštění a běh sdíleného kódu v JavaScriptu, než podporu ulehčení některých výše úrovňových funkcí, tak nabízejí možnost integrace některých moderních frameworků, jakožto nadstavbu nad svůj JavaScript. Je tedy možné kromě psaní v čistém JavaScriptu zapojit frameworky jako *Angular*, *Vue*[23] či *React*.

2.3.5 Open-source

Poslední společnou vlastností je, že oba porovnávané frameworky jsou open-source. To znamená, že i přesto, že oba byly vytvořeny a nyní udržovány a rozšiřovány velkými společnostmi, jejich zdrojový kód je volně dostupný a zároveň otevřený pro editaci a přispívání do projektu.

Důležitým faktorem open-source kódu je také to, že je zdarma. Stojí za zmínku, že NativeScript měl sekci komponentů ve verzi premium, které byly placené, avšak *3.11.2017* se společnost *Telerik*[24], která NativeScript vyvíjí rozhodla nechat tyto komponenty zcela zdarma[25].

2.4 NativeScript

NativeScript je framework pro multiplatformní vývoj nativních mobilních aplikací pomocí JavaScriptu, CSS a XML (nebo HTML). NativeScript byl poprvé vydán v roce 2014 firmou *Telerik*, která ho i nadále rozšiřuje.

Pomocí NativeScriptu lze vyvíjet aplikace pro platformu Android a iOS. Windows Phone zatím není podporován [1].

Vývoj mobilních aplikací pro Android je možný ze všech běžných desktopových platforem jako je Windows, Linux a Mac. S vývojem pro platformu iOS je to trochu komplikovanější, protože k přeložení aplikace je potřebný XCode a ten je dostupný pouze na platformě Mac. Avšak NativeScript vytvořil vývojové prostředí *NativeScript Sidekick*[26], které umí mimojiné provést kompilici aplikace v cloudu. Tento cloud běží na stroji Mac Pro. Díky tomuto vývojovému prostředí je tak možné vyvíjet aplikaci v NativeScriptu pro iOS bez nutnosti vlastnit zařízení Mac, tedy například ve Windowsu či Linuxu.

2.4.1 Vývojové možnosti

Tvorba aplikací pomocí NativeScriptu umožňuje jistou variabilitu vývoje. Lze vyvíjet buď pouze prostřednictvím čistého JavaScriptu (včetně podpory ECMAScript 2015) nebo pomocí *TypeScriptu*[27], což je nadstavba nad jazykem JavaScript, která tento jazyk rozšiřuje o statické typování a další atributy z objektově orientovaného programování.

[23] Vue – https://vuejs.org/

[24] Telerik – https://www.telerik.com/

[25] NativeScript premium komponenty zdarma – https://www.nativescript.org/blog/nativescript-ui-is-now-free-here-s-how-to-get-started/

[26] Vývojové prostředí NativeScript Sidekick – https://www.nativescript.org/nativescript-sidekick/

[27] TypeScript – https://www.typescriptlang.org/

Kromě TypeScriptu lze NativeScript rozšířit i o frameworky *Angular* (verze 2) a nově[28] také o *Vue*. Pokud je vyvíjeno pomocí Angularu soubory typy XML[29] jsou nahrazeny soubory typu HTML. Angular lze také zkombinovat s TypeScriptem.

Původně však bylo možné vyvíjet pouze pomocí čistého JavaScriptu, což ukazuje, jak dobře se NativeScript dokáže přizpůsobit novým trendům.

2.4.2 Stylovaní a animace

NativeScript podporuje stylování pomocí CSS. CSS pochází z webových technologií, avšak díky jeho známým a snadno použitelným principům se dostává i do dalších technologií, včetně NativeScriptu. Výhodou je, že ho již řada vývojářů zná a existuje celá řada návodů a článků, jak jej použít. K další výhodě patří, že je možné zapojit i dva známé pre-procesory CSS, které proces stylování ještě zjednoduší. Zmíněnými pre-procesory, které své soubory následně přeloží do CSS jsou SASS[30] a LESS[31].

Je však nutné zmínit, že nelze použít všechna pravidla CSS jako při vývoji webu. Další nevýhodou oproti webu je, že po úpravě CSS souboru se změny neprojeví okamžitě, ale NativeScript musí nahrát změněný soubor, což trvá o něco déle oproti vývoji webu.

Pomocí NativeScriptu je možné provádět celou řadu animací pro zpestření uživatelského zážitku. Mezi animace patří *posunutí*, *rotace*, *změna měřítka*, *průhlednost* či *změna barvy*. U jednotlivých animací lze nastavit atributy jako *délka*, *zpoždění*, *směr animace* a další. Animace lze tvořit buď deklarativně pomocí CSS nebo také imperativně skrze JavaScript.

Do oblasti stylování patří také uživatelské fonty, i tuto možnost NativeScript povoluje, je tedy možné snadno nahrát externí font, který se umístí do složky */fonts*. Zajímavým benefitem této možnosti je i to, že pomocí fontů lze používat ikonky, které jsou zobrazeny pomocí fontů a tím pádem jsou vektorové. Mezi takové speciální fonty patří například *Font Awesome* [32].

2.4.3 Popularita a velikost komunity

Není lehké posuzovat popularitu a rozšířenost nějaké technologie, nicméně existuje pár veřejně dostupných prostředků, z kterých se dají vzít nějaká číselná data. Patří sem, počet hvězd, chyb a přispívatelů na *GitHubu*, počet položených otázek na *StackOverflow* nebo počet pluginů na *npm*. Konkrétní data vypadají následovně:

- GitHub: počet hvězd[33] – 13299
- GitHub: celkový počet chyb – 3757 (již uzavřených 3399)
- GitHub: počet přispívatelů – 112
- StackOverflow: počet položených otázek[34] – 2820

[28] Vue pro NativeScript – https://www.nativescript.org/blog/a-new-vue-for-nativescript/
[29] EXtensible Markup Language – obecný značkovací jazyk
[30] SASS – https://www.npmjs.com/package/nativescript-sass
[31] LESS – https://www.npmjs.com/package/nativescript-dev-less
[32] Font Awesome – https://fontawesome.com/
[33] Zdroj počtu hvězd, chyb a přispěvovatelů NativeScriptu na GitHubu – https://github.com/NativeScript/NativeScript/
[34] Zdroj počtu položených otázek NativeScriptu na Stack Overflow – https://stackoverflow.com/questions/tagged/nativescript/

- npm: počet pluginů[35] – 700

Výhody

- Možnost zkombinovat NativeScript s Angularem, Vue či TypeScriptem
- Přístup k nativnímu aplikačnímu rozhraní obou platforem z JavaScriptu
- Dobře zpracované tutoriály a ukázky aplikací přímo na oficiálních stránkách NativeScriptu[36]
- Stylování pomocí CSS s možností nadstavby SASS nebo LESS pre-procesoru
- Možné vyvíjet také pouze pomocí JavaScriptu samotného, což může znamenat velmi nízkou učící křivku

Nevýhody

- Některá dokumentace aplikačního rozhraní neobsahuje detailnější popis složitějšího použití
- NativeScript je stále pod velkým vývojem a bohužel obsahuje nějaké chyby
- Základní NativeScript ještě neobsahuje příliš velké množství komponent, avšak existuje zde velká řada přídavných modulů 3. stran, které většinu chybějících věcí nahrazují, nicméně nutno dodat, že ne vždy vše funguje na obou platformách, tak jak by mělo

2.5 React Native

React Native je framework pro multiplatformní vývoj nativních, mobilních aplikací pomocí JavaScriptu (včetně ES2015), frameworku React a HTML. První zmínka o frameworku React Native byla poprvé na mezinárodním *Facebook hackathonu* v roce 2013 a poprvé byl vydán v roce 2015[37] firmou Facebook[38].

Vývoj mobilních aplikací pomocí React Native byl z počátků možný jen pro iOS, později se přidala podpora vývoje pro platformu Android a nově také pro Windows Phone[39].

Vývoj mobilních aplikací pro Android a Windows Phone je možný ze všech běžných desktopových platforem jako je Windows, Linux a Mac. S vývojem pro platformu iOS je to stejné jako u NativeScriptu – kvůli potřebnému XCode je potřebná platforma Mac. Nicméně je možné přeložit aplikaci v cloudu přes vývojové prostředí *Expo*[40]. Pomocí tohoto vývojového prostředí lze vyvýjet pro iOS také z Windowsu či Linuxu.

[35] Zdroj počtu pluginů NativeScriptu na npm – https://www.npmjs.com/search?q=keywords:NativeScript/

[36] NativeScript návody – https://docs.nativescript.org/tutorial/chapter-1/

[37] Historie React Native – https://medium.com/react-native-development/a-brief-history-of-react-native-aae11f4ca39/

[38] Facebook – https://www.facebook.com/

[39] React Native na Windows Phone – https://blogs.windows.com/buildingapps/2016/04/13/react-native-on-the-universal-windows-platform/#2I8GPdUYZokY82Q3.97/

[40] Vývojové prostředí Expo – https://expo.io/

2.5.1 Vývojové možnosti

React Native nenabízí tolik variability možnosti vývoje jako NativeScript, nabízí pouze vývoj pomocí webového frameworku React, který je rovněž od firmy Facebook. Pohled na tuto věc může být nevýhoda ale i výhoda.

Nevýhodou je, že pro vývojáře, který neznal React bude pravděpodobně učící křivka o něco delší, než u NativeScriptu. Na druhou stranu popularita společnosti Facebook a hlavně samotného frameworku React, který má obrovskou komunitu byla bez větší snahy předána i na React Native. Což mimo jiné znamená kvalitní dokumentaci, řadu návodů a článků a také přenesení zajímavých principů samotného Reactu mezi které patří využití všeho jako komponent, deklarativní přístup, efektivním renderování a další.

2.5.2 Stylovaní a animace

React Native nepoužívá ke stylování tradičnější CSS jako NativeScript, ale svůj speciální přístup, který částečně vychází z principů CSS, nicméně jeho syntaxe je trochu odlišná a psaná v JavaScriptu. Avšak není zde použito klasické aplikování stylů pomocí tříd nebo id. Za zmínku stojí, že ale existují rozšiřující moduly[41], které umožňují CSS soubory nahrát přímo do React Nativu, ale možnost použití pravidel klasického CSS je značně omezena a celkově tento přístup není příliš doporučován.

Podobně jako v NativeScriptu je pomocí React Nativu možné využít celá řada animačních efektů, ke kterým patří *posunutí*, *rotace*, *změna měřítka* či *průhlednost*. Animace lze použít v různých režimech jako režim s *postupně klesající rychlostí*, *chvěním na konci*, či nastavitelnou *dobou trvání*.

Stejně jako v NativeScriptu i zde je možné použít externí fonty a to včetně těch s ikonami. Je nutné podotknout, že zde je přidávání fontů pro každou platformu odlišné a trochu pracnější, pro platformu iOS je nutné fonty nahrát přes XCode.

2.5.3 Popularita a velikost komunity

Jsou použita stejná kritéria jako u posuzování popularity NativeScriptu v sekci 2.4.3. Z dat lze vidět, že React Native má citelně větší komunitu, částečně se dá přisuzovat věhlasností Reactu či Facebooku. Konkrétní data z *GitHubu* [**?**], *StackOverflow* [**?**] a *npm* [**?**] vypadají následovně:

- GitHub: počet hvězd[42] – 62920
- GitHub: celkový počet chyb – 12084 (již uzavřených 11531)
- GitHub: počet přispívatelů – 1650
- StackOverflow: počet položených otázek[43] – 29860
- npm: počet pluginů[44] – 5873

[41] React Native CSS – https://facebook.github.io/react-native/docs/style.html/

[42] Zdroj počtu hvězd, chyb a přispěvovatelů React Native na GitHubu – https://github.com/facebook/react-native/

[43] Zdroj počtu položených otázek React Native na Stack Overflow – https://stackoverflow.com/questions/tagged/react-native/

[44] Zdroj počtu pluginů NativeScriptu na npm – https://www.npmjs.com/search?q=keywords:react-native/

Výhody

- Obrovská popularita a komunita s čímž přichází velké množství rozšiřujících komponent, návodů atd.
- Podpora vývoje pro platformu Windows Phone
- Kvalitní dokumentace, včetně online emulátoru mobilu, který zobrazuje ukázkové kódy, které je dokonce možné editovat a zobrazit změny [45]
- Přístup k nativnímu aplikačnímu rozhraní obou platforem z JavaScriptu

Nevýhody

- Nemožnost zkombinovat React Native s dalšími frameworky či nadstavbami JavaScriptu
- Pro vývojáře bez znalosti Reactu delší učící křivka
- Odlišný přístup stylování než CSS
- Ne všechny komponenty fungují na všech platformách – React Native byl původně pouze pro platformu iOS. Android a Windows Phone byly přidány až opožděně, a tak některé komponenty fungují pouze na iOS či Androidu a některé jenom na iOS

2.6 Srovnání NativeScript vs React Native

Oba porovnávané frameworky mají mnoho společného a jejich hlavní výhodou je, že umožňují přistupovat k aplikačnímu rozhraní dvou nejvýznamnějších mobilních platforem Android a iOS. Plusové body získává React Native za svoji podporu platformy Windows Phone (ač zatím v omezené dostupnosti funkcionality), na druhou stranu Windows Phone tvoří 0,1% aktuálního trhu s dlouhodobým klesajícím trendem, a tak vyvstává otázka jestli se vývoj pro tuto platformu skutečně vyplatí.

Oba frameworky nabízí dobré podmínky pro tvorbu animací či nahrání vlastních fontů. Co se týče stylování obecně, tak zde si nese výhodu NativeScript se svojí přímočarou podporou CSS a snadnou integrací pre-procesorů jako SASS a LESS.

Nutné podotknout citelně výraznější komunitu frameworku React Native. Tato výhoda má za následky větší počet dostupných komponentů, modulů třetích stran, návodů, internetových diskuzí a podobně. Na druhou stranu komunita NativeScriptu není zdaleka zanedbatelná v porovnání s jinými frameworky na vývoj mobilních aplikací.

Co pro řadu vývojářů může být klíčovým faktorem je použití rozšiřujícího frameworku z pohledu nadstavby nad JavaScriptem. U Reactu Native je jediná, ale za to velmi lukrativní volba v podobě Reactu, díky kterému si získal řadu příznivců. U NativeScriptu je variabilita o poznání větší – lze použít Angular či Vue. Navíc je možné využít nadstavbu jazyka JavaScript – TypeScript. Zajímavým faktorem NativeScriptu také je, že není nutné žádný rozšiřující framework využít, v takovém případě probíhá vývoj pouze pomocí čistého JavaScriptu, což pro některé vývojáře přináší velmi přívětivou učící křivku.

Porovnání lze shrnout tak, že React Native má za sebou obrovskou komunitu, z které řádně těží. NativeScript je zase flexibilnější a přizpůsobivější novým trendům. Celkově výběr ani jednoho z těchto frameworků nebude krok špatným směrem, důležitý je samotný koncept tohoto přístupu vývoje, který je velmi moderní a má vysoký potenciál.

[45] React Native návody – https://facebook.github.io/react-native/docs/tutorial.html/

2.7 Popis zvolených technologií

V této části budou podrobně popsány technologie, které jsou použity v demonstrační aplikaci.

2.7.1 NativeScript

Pro výběr klientské mobilní aplikace je zvolena technologie *NativeScript*. Detailnější popis této technologie spolu s porovnáním konkurenční, podobné technologie React Native je popsán v sekci 2.4 a celý směr vývoje multiplatformních aplikací interpretovaných za běhu v JavaScriptu pak v sekci 2.3.

Některé z důvodů této volby jsou například možnosti použití stylování pomocí CSS a nadstavby SASS pre-procesoru. Dalším zajímavým faktorem, který hrál roli při výběru byl fakt, že není nutné zapojit framework jako nadstavbu JavaScriptu a je tedy možné vyvíjet bez nutnosti použití další technologie, což nelze u React Native, kde je nutné pracovat s frameworkem React.

2.7.2 EcmaScript 2015 (ES6)

Jazyk JavaScript je implementací standardu ECMAScript od společnosti *European Computed Manufactures Association* (ECMA). Tento standard je vydáván ve verzích. Verze JavaScriptu se odvíjí od verze ECMAScriptu. Velká řada důležitých změn přišla s verzí EcmaScript 2015 (také nazývaná ES6). Je nutné říci, že již existují i vyšší verze JavaScriptu, nicméně funkce, které přibyly již nejsou natolik zásadní a hlavně současná verze NativeScriptu má zatím podporu do verze ES6, proto se v následující částí budeme zabývat právě popisem ES6, respektive novinkami, které přinesla.

Nahrazení var za let a const

Příjemnou změnu přineslo nahrazení klíčového slova *var*, které definuje proměnou za klíčové slovo *let* a *const*. Toto nahrazení odstraňuje nepříjemný hoisting[46] a pak je tedy definování proměnné pouze v rámci bloku. V případě použití *const* pak nastává definice konstantní proměnné, která nemůže být změněna.

Snadnější definice funkcí

Zajímavou novinkou jsou *arrow funkce*, které jsou funkční zkratkou pro definici funkcí pomocí syntaxe `=>`. Arrow funkce jsou podobné prvku z jazyků C#, Java 8 nebo CoffeeScript. Lze je zkombinovat i s asynchronním *Promise*, což je prezentováno na zdrojovém kódu 2.1.

Promise namísto callback

S příchodem ES6 přišel nový zajímavý způsob, jak pracovat s asynchronními funkcemi. Do dřívější verze bylo nutné používat tzv. callbacky[47], které z většího počtu asynchronních

[46] Hoisting – je chování jazyka JavaScript, které přesune všechny deklarace proměnných na horní část skriptu nebo aktuální funkce bez ohledu na to, kde byly programátorem vytvořeny. Detailnější popis problematiky: https://www.w3schools.com/js/js_hoisting.asp

[47] Princip callback funkce – https://developer.mozilla.org/en-US/docs/Glossary/Callback_function/

funkcí, které se navzájem volaly udělaly nečitelný kód. Použití *Promise*[48] nám dává způsob, jak provést asynchronní zpracování více čitelným kódem podobající se synchronnímu přístupu. Výhodou také je, že Promise podporuje *catch* blok, do kterého se běh programu dostane v případě selhání operace.

```
ziskejData()
.then(ziskana_data => {
    return zpracujData(ziskana_data);
})
.then(zpracovana_data => {
    console.log(zpracovana_data);
})
.catch(err => {
    // odchyceni chyby vznikle funkci ziskejData nebo zpracujData
    console.error(err);
});
```

Výpis 2.1: Ukázka práce s asynchronními funkcemi pomocí Promise a také využití *arrow funkcí*

Nová definice tříd

Další změnou je nová definice tříd. Dříve bylo v JavaScriptu možné vytvářet objekty stejného typu pomocí prototypu, nyní je možné využít pohodlnější a známější způsob pomocí klíčového slova *class*. Stejně tak je možné využít třídní dědičnosti. Avšak je nutné říci, že se jedná spíše o tzv. *syntactic sugar* a nejedná se o klasické třídy jako v jazycích Java či C++, jde spíše o speciální, tovární funkce a JavaScript v podstatě pouze simuluje principy a vlastnosti třídově založených mechanismů [6].

Další zajímavé funkce

ECMAScript 2015 přináší celou řadu zajímavých změn a nových konstruktů mezi které patří například snadnější exportování a načítání modulů, práce s textovými literály, požívání výchozích hodnot a mnoho dalšího. Avšak detailnější popis všech funkcí by vystačil na celý dokument.

2.7.3 CSS pre-procesor SASS pro NativeScript

Mezi dostupnými CSS pre-procesory pro framework NativeScript jsou aktuálně dostupné *SASS* a *LESS*. Zvolil jsem pre-procesor *SASS* a to hlavně díky jeho většímu počtu stažení na *npm* a také díky předešlé zkušennosti z webových technologií.

CSS pre-procesor

CSS pre-procesor je nástroj, který ze zdrojového kódu psaného ve své vlastní syntaxi interpretuje nebo kompiluje validní CSS soubor. Dostupné pre-procesory pro NativeScript berou navíc ohled na omezené množství pravidel CSS, které jsou validní pro NativeScript.

Důvodem vzniku pre-procesorů byla omezená funkcionalita samotného CSS, která u větších projektů způsobovala duplicitu, špatnou modifikovatelnost či nečitelnost kódu.

[48] Detailnější popis Promise – https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/Promise/

Pre-procesor SASS

SASS je pre-procesorový skriptovací jazyk, který generuje validní CSS soubory. O proti běžnému CSS přináší celou řadu nových funkcionalit a konstruktů, které značně usnadňují vývoj. Patří sem například zavedení:

- Proměnné
- Vnořování
- Mixin[49]
- Dědičnost

SASS podporuje 2 druhy syntaxe. Původní syntax, který vycházel ze systému *Haml*[50] používá odsazení k definici bloků a nové řádky pro oddělení pravidel. Pokud chce vývojář využít tuto syntaxi, je nutné zdrojový kód zapisovat do souboru s příponou *.sass*, SASS preprocesor pak vygeneruje ekvivalentní soubor s příponou *.css*.

Novější verze používá syntax podobnější CSS, k definici bloků používá složené závorky a pro oddělení jednotlivých pravidel je použit středník. Soubory s touto syntaxí musí mít přípony *.scss*, SASS pre-procesor taktéž tyto soubory převede na ty s příponou *.css*.

SASS pro NativeScript

SASS pro NativeScript je rozšiřující modul do NativeScriptu, podporuje oba možné typy souborů s příponami typu *.sass* i *.scss*. Instalace přidá automatizované skripty do složky *hooks*.

SASS pre-procesor přeloží všechny *.sass* a *.scss* soubory ze složky *app* do souborů *.css*. Výhodou je, že automatizované skripty dokáží v průběhu vývoje aplikace rozeznat změnu editovaného souboru (po uložení souboru *.sass* nebo *.scss*), přeložit soubor do formátu *.css* a za běhu přenést jen potřebné změny. To znamená, že není nutné znovu překládat celou aplikaci.

2.7.4 Node.js a serverové technologie

Výběr serverové technologie patří jednoznačně technologii *Node.js*, který v poslední době nabírá na velké popularitě a na platformě JavaScript nemá na serveru významnějšího konkurenta podobných rozměrů. Výhodou *Node.js* je také to, že nabízí snadnou instalaci a správu rozšiřujících knihoven a frameworků přes *npm*. Jedním takovýmto použitým frameworkem je i *Express.js*[51], který bude také v demonstrační aplikaci použit.

Node.js

Node.js je velmi výkonné, událostmi řízené prostředí pro JavaScript. Základem Node.js je JavaScriptový *interpret V8* od společnosti *Google*[52]. Nad ním je tenká vrstva kódu v *C++* poskytující potřebné zázemí jako je *event-loop*[53] vyhodnocující *příchozí události*, obsluhu *Input/Output bufferů* a jiné. Avšak je navržen tak, aby vývojáře odstínil od nízkoúrovňových

[49] Mixin – třída, která obsahuje metody z několika dalších tříd
[50] Haml – http://haml.info/
[51] Express.js – https://expressjs.com/
[52] Google – https://www.google.cz/
[53] Event-loop je programovací konstrukt, který čeká a odesílá události v programu

problémů. Node.js je čistý JavaScript běžící v jediném vlákně, fungujícím velmi rychle a škálovatelně [5].

Programy pro Node.js hojně využívají *asynchronní operace* pro minimalizaci režie procesoru a maximalizaci výkonu. Asynchronní znamenají neblokující, pod tím si lze představit například asynchronní čtení z databáze. Během načítání není čekáno na dokončení operace, ale okamžitě se pokračuje na další dostupnou operaci. Po načtení dat se přechází zpět na návratové místo asynchronní operace.

Výraznou popularitu lze také vyčíst z toho, kdo z velkých společností Node.js využívá. Patří sem takový giganti jako je *LinkedIn*[54], *Netflix*[55], *PayPal*[56], *Uber*[57], *eBay*[58] a další[59].

Npm

Npm nebo také *Node Package Manager* je správce balíčků pro JavaScript. Pro Node.js je výchozím správcem balíčků, instalujícím se společně s Node.js. Npm se stará o automatický proces instalace, odstranění, upgrade a hlídání kompatibility používaných balíčků v rámci projektu. Pod balíčky si lze představit rozšiřující knihovny, ale i celé frameworky.

Npm se skládá z klientského programu v příkazové řádce a online databáze věřených balíčků které jsou zdarma. Navíc je zde i možnost vytvoření placeného účtu, s kterým přichází eventualita mít v npm databázi balíčky jako soukromé a k dispozici jen v rámci týmu.

Zajímavou funkcí npm online databáze je také to, že umožňuje snadné vyhledávání, zobrazení popisu a přehledu o počtech stažení jednotlivých balíčků a to jak z dlouhodobého hlediska, tak i z aktuálního týdne. To uživateli dokáže snadno pomoci při výběru, právě toho balíčku, který potřebuje a že je rozhodně z čeho vybírat, na npm je už více než *650 000* balíčků, což z něho dělá suverénně největšího správce balíčků [2].

Express.js

Express.js nebo jednoduše Express je open-source framework pro Node.js, primárně zaměřený na tvorbu aplikací typu *klient-server*. Express se pomáhá vypořádat s typickými úlohami související s tímto typem aplikací jako je tvorba aplikačního rozhraní, rout, rozbor HTTP požadavků, vyjmutí parametrů z URL[60], práce s cookies, sezením a další. Nicméně přináší i trochu netypické funkce jako je *middleware*.

Middleware je funkce, která má přístup k *požadavku (request)* a *odpovědi (response)*, dále má pak ukazatel na další funkci typu middleware. Série těchto funkcí se spouští automaticky s příchodem nového požadavku na server, obsahem těchto funkcí může být vykonání jakéhokoliv kódu, provedení změny v požadavku či odpovědi, ukončení cyklu odpověď-požadavek nebo zavolání další funkce typu middleware.

Jako příkladem využití může být middleware, který se stará o autentizaci. Při novém požadavku je nejdříve spuštěna middleware funkce, která ověří přihlašovací údaje uživatele

[54] LinkedIn - https://www.linkedin.com/
[55] Netflix – https://www.netflix.com/
[56] PayPal – https://www.paypal.com/
[57] Uber – https://www.uber.com/
[58] eBay – https://www.ebay.com/
[59] Seznam používanosti Node.js v korporacích – http://www.tothenew.com/blog/how-are-10-global-companies-using-node-js-in-production/
[60] Uniform Resource Locator - řetězec znaků s definovanou strukturou, který slouží k jednoznačné identifikaci zdrojů informací

(případně jeho sezení). V případě úspěchu se zavolá další middleware na zpracování požadavku. V případě neúspěchu middleware posílá na klientskou část odpověď o tom, že je neautorizován a uzavírá tak dotaz.

2.7.5 Databázové technologie

Součástí demonstrační aplikace bude taktéž práce s daty. K jejich ukládání jsem se rozhodl využít tradiční přístup v podobě relační SQL databáze. Relačních databází existuje celá řada, některé jsou dobré pro svoji jednoduchost, snadné zprovoznění a také proto, že jsou zdarma. Avšak při větší zátěži většinou zaostávají v rychlosti od těch pokročilejších zpravidla placenějších.

Při tvorbě nových projektů nikdo neví jakou zátěž bude mít jeho databáze a pak tedy není úplně jednoduché zvolit vhodnou databázi. Nevýhodou při přechodu na nový typ databáze pak je, že existuje trochu odlišná syntaxe pro každou databázi, což znamená přepisování SQL dotazů. Touto problematikou se zabývá framework do Node.js s názve *Knex.js*.

Knex.js

Knex.js je framework pro Node.js určený pro flexibilní práci s databázemi. Knex.js používá svoji syntax SQL dotazů, která je velmi podobná těm z klasického SQL, s tím rozdílem, že se zapisuje v JavaScriptu. Pomocí této syntaxe je pak možné připojit se do celé řady databází, do kterých Knex.js provede překlad SQL dotazů. Mezi dostupné databáze patří Postgres[61], MSSQL[62], MySQL[63], MariaDB[64], SQLite3[65], Oracle[66] a Amazon Redshift[67].

Pomocí JavaScriptu je tak možné napsat SQL dotazy a pak připojit jakoukoliv z výše zmíněných databází. Případně později změnit databázi bez nutnosti psaní nových dotazů.

Knex.js provádí čtení z databáze v asynchronním, neblokujícím režimu. Pracovat s těmito příkazy jde pak snadno pomocí Promise nebo je zde i podpora starších callbacků.

Databáze Sqlite3

Databáze *Sqlite3* je databáze, která je založena na souborovém systému a veškerá data databáze jsou uložena pouze pomocí souboru, což je hlavní rozdíl oproti typickým relačním databázím, které jsou zpravidla uloženy na vzdáleném serveru a namísto přistupování ke společnému souboru pracují na principu založeném na procesech.

To, že se k datům Sqlite3 databáze přistupuje takovýmto způsobem může přinést velkou rychlost, protože se nemusí připojovat na vzdálený server přes síť Internet a stejnou cestou dostat výsledek. Avšak takových to zajímavých výsledků je docíleno pouze při nízkém počtů dotazů zároveň, jelikož hlavní nevýhoda této databáze je ta, že lze do databáze – souboru zapisovat pouze jednou ve stejný čas.

Do demonstrační aplikace byla tato databáze vybrána z několika důvodů. Jedním faktorem je to, že samotné databázové technologie nebyly hlavním cílem této práce, a tak převládla snaha o snadnou použitelnost bez nutnosti připojování se na další databázový

[61] Postrgres – https://www.postgresql.org/
[62] MSSQL – https://www.microsoft.com/en-us/sql-server/default.aspx
[63] MysSQL – https://www.mysql.com/
[64] MariaDB – https://mariadb.org/
[65] SQLite3 – https://www.sqlite.org/
[66] Oracle – http://www.oracle.com/technetwork/database/enterprise-edition/overview/index.html
[67] Amazon Redshift – aws.amazon.com/redshift

server. Další výhoda, která pramení z povahy této databáze je ta, že se snadno testuje. Je velmi jednoduché vytvořit integrační testy během, kterých se vytvoří nová databáze a po skončení se zase odstraní. Poslední zajímavou výhodou je, že Sqlite3 lze umístit i přímo do mobilního telefonu, toho by se dalo využít v případě rozšíření aplikace, kde by tato databáze sekundovala hlavní databázi na serveru, třeba pro účely částečného offline[68] provozu či pro *caschování*[69] dat.

Hlavní sílu zvolených databázových technologií však vidím v použitém frameworku *Knex.js*, díky kterému bude případně možné rychle databázi vyměnit bez nutnosti přepisovat databázové dotazy.

[68] Offline – bez připojení k síti Internet
[69] Cashování – ukládání dat do mezipaměti

Kapitola 3

Návrh aplikace

V následující kapitole je probrán návrh demonstrační aplikace, na které jsou prakticky vyzkoušeny zvolené technologie a jejich vzájemná integrace.

Nejprve je popsáno *zadání aplikace*, které slouží jako podklad pro implementaci, dále pak *návrh serverové části* aplikace, s čímž souvisí hlavně návrh relační databáze a komunikačního protokolu. Závěr této kapitoly pak patří *návrhu samotné klientské části* mobilní aplikace.

3.1 Zadání aplikace

Cílem práce je vytvořit **systém pro rezervaci a správu firemních prostředků** v rámci interního chodu firmy. Jedna z klíčových vlastností je, aby systém byl univerzální, ne tedy navržen na míru. Pod firemními prostředky si tedy lze představit celou řadu věcí jako jsou například automobily, nemovitosti, nářadí a tak dále.

Rezervace je možná prostřednictvím připravených *akcí*, které usnadňují a doporučují rezervaci více relevancích prostředků současně. U jednotlivých prostředků lze také vybrat dostupné *atributy* pro bližší specifikaci vyhledávaného prostředku určeného k rezervaci.

Se systémem pracují zaměstnanci firmy, je zde tedy řešeno jejich *přihlašování* do aplikace a *variabilita práv* jednotlivých funkcí systému. Jednotlivý uživatelé mají přehled o svých rezervovaných událostech prostřednictvím přehledného *kalendáře* v různých možnostech zobrazení.

V rámci nové rezervace je možné přizvat *účastníky* či *pořadatele*. Rezervace se může opakovat a to velmi variabilně. K rezervaci je možné připojit soubor jakožto přílohu a také nastavit e-mailové *upozornění*.

3.1.1 Typ a objekt prostředku

Typ prostředku je zastřešení skupiny pro konkrétní objekty prostředků. Nejlépe si to lze představit na konkrétním příkladě. Typ prostředku může být *místnost* a konkrétní objekt je *A101 – zasedací místnost*.

Jednotlivým typům lze nastavit, zdali jsou *všechny jejich objekty dostupné* pro rezervaci. Taktéž je možné zvolit, zdali je *dostupný pro rezervaci konkrétní objekt* a případně lze dokonce nastavit v *jaké dny či hodiny je dostupný*. Kupříkladu lze nastavit, že místnost *A101* je k dispozici jen ve dnech *pondělí až čtvrtek od 8 do 17 hodin*.

Objekty mohou mít také několik atributů, což je výhodné pro specifické vyhledávání. Lze tedy zadat například, že chci zobrazit pouze ty *místnosti*, které mají atributy *zvukotechnika* a *projektor*.

3.1.2 Akce

Rezervace probíhají prostřednictvím akcí. Akce může mít *nezbytné a doporučené typy* k rezervaci. To si lze představit na příkladu akce *školení*, kde je její nezbytný typ prostředku *místnost* a doporučený pak *notebook*. Akce si také nese některé předdefinované výchozí údaje sloužící jakožto našeptávač pro uživatele. Mezi tyto výchozí informace patří *délka rezervace*, *zpráva pro e-mailové upozornění* a *barva do kalendáře*.

Akci lze nastavit, zdali je možné rezervaci *potvrdit automaticky* či je nutné *potvrzení administrátora*. Za každou akci zodpovídá alespoň jeden administrátor, kterému je umožněno detailní nahlížení do jednotlivých rezervací a potvrzování či mazání rezervací.

3.1.3 Vyhledávání volných prostředků

Proces samotného vyhledávání by měl být pro uživatele přívětivý. Uživatel nemusí například zadávat konkrétní čas ani konkrétní objekt, zvolí kupříkladu jen akci a délku trvání, systém mu pak bude nabízet dostupné výsledky. Avšak své vyhledávání může zpřesňovat, třeba v podobě zadání konkrétního času rezervace, či konkrétního objektu.

3.1.4 Rezervace

Po vybrání volných dostupných prostředků lze k samotné rezervaci přizvat další uživatele v systému – *účastníky*, případně i další *pořadatele*. Pozvaným účastníkům se pak bude tato událost zobrazovat v jejich kalendáři a pořadatelé ji navíc mohou i editovat.

K rezervaci lze také přidat *přílohu*, zvolit *barvu do kalendáře* a také nastavit *upozornění* v podobě e-mailu. Proces nastavení upozornění by měl být uživatelsky přívětivý a upozornění může být hned několik. Mimo uživatele systému je možné přidat i e-maily dalších lidí.

3.1.5 Opakování

Pokud jsou vybrané prostředky volné na všechny opakované termíny, lze také zvolit opakování rezervace. Opakovat rezervaci lze několika způsoby: *denně, týdně, měsíčně*. Ukončení opakování lze nastavit buď v podobě *počtu opakování* nebo podle *data konce opakování*.

V týdenním režimu lze vybrat jaké dny v týdnu rezervaci opakovat a zda opakovat každý týden či každý *Ntý* týden. V měsíčním režimu je možné vybrat jaké dny v měsíci opakovat, či kombinace například první pondělky v měsíci.

3.1.6 Zobrazení rezervací

Rezervace jsou v aplikaci zobrazeny prostřednictvím *kalendáře*. Každý uživatel má přístup ke svým pořádaným rezervacím či k rezervacím, do kterých je pozván. Je tu tedy možnost zobrazení *osobního kalendáře*, kde má daný uživatel pro něho relevantní rezervace.

Mimo tento kalendář má uživatel také možnost nahlédnout na konkrétní *kalendáře objektu*, kde si může prohlížet vytíženost (volnou dostupnost) těchto objektů, avšak nemůže vidět detaily rezervací, do kterých není pozván.

Výchozí zobrazení kalendáře je *měsíční*. Nicméně lze přepnout i do *týdenního* režimu s denním hodinovým přehledem či do *ročního*, který slouží jako rychlý přepínač mezi měsíci.

3.2 Podobné aplikace

Na trhu je mnoho softwarů podporujících tvorbu rezervačních systému, řada z nich má k dipozici i mobilní aplikaci pro Android a iOS nebo při nejmenším responzivní webové aplikace. Některé z nich jsou dokonce (s omezeným počtem funkcí) zdarma. Nicméně tyto rezervační systémy slouží pro rezervaci služeb pro zákazníky nikoli pro interní potřeby firem.

Pak zde existují aplikace přes které je možné v rámci interního chodu firmy plánovat schůzky, avšak ne už pak rezervovat konkrétní prostředky dané firmy.

Do jisté míry lze za podobné aplikace označit i klasické kalendářové aplikace přes které už dnes lze plánovat události včetně přizvání dalších uživatelů. Výhodou je, že jsou zdarma a dostupné i jako mobilní aplikace.

3.2.1 My Time

Systém *My Time* nabízí univerzální rezervační systém pro firmy, které chtějí zákazníkům nabízet určitý typ služeb, který je možné rezervovat, příkladem takové firmy může být kadeřnický salón. Umožňuje zobrazení uživatelsky přívětivého procesu rezervace pro zákazníky a dobrou přehledovou podporu pro majitele firem a to i prostřednictvím interaktivního kalendáře. Zákazníkům lze také automaticky zaslat upozorňující e-mail o rezervaci. Ze systému lze dokonce zaplatit službu platební kartou. Výhodou je, že je k dispozici nejen webová aplikace, ale také mobilní aplikace pro Android i iOS.

Podobný rezervační systém je i *Reservio*, jehož výhoda je ta, že je i v českém jazyce. Bohužel však neobsahuje mobilní aplikaci, nicméně lze využít pouze serverové části systému a skrze dané aplikační rozhraní vytvořit libovolného klienta, tedy i multiplatformní mobilní aplikaci.

3.2.2 Hub Spot

Hub Spot je webová aplikace, která umožňuje plánování schůzek v rámci interního prostředí firem. Výhodou je, že lze importovat kontakty z *Google účtu* či *Outlooku*. Lze nastavit datum a délku naplánované schůzky, napsat popis, přizvat účastníky nebo přiložit soubor. Taktéž je dostupná funkce na e-mailové upozornění plánované schůzky.

Velmi podobnou funkcionalitu nabízí i *Outlook* samotný přes svoji funkci *Meeting*.

3.2.3 Google kalendář

Google kalendář je aplikace skrze kterou lze vytvářet události a spravovat jejich zobrazení prostřednictví kalendáře. Událost je možné sdílet i s jinými uživateli, kterým se daná událost zobrazí v e-mailu nebo pokud mají Google účet, tak i v jejich kalendáři. Dále je zde podpora opakování události, přiložení souboru nebo také místa. Již klasická funkcionalita je upozornění a to v podobě notifikace či e-mailu. Nespornou výhodou Google kalendáře je to, že je dostupný, jak v podobě multiplatformní mobilní, tak i webové aplikace a to hned v několika jazycích.

Mezi podobné aplikace lze zařadit opět *Outlook* se svojí funkcí *sdílení kalendáře* anebo *iCalendar* od platformy iOS, nicméně ten je nedostupný jako mobilní aplikace pro Android.

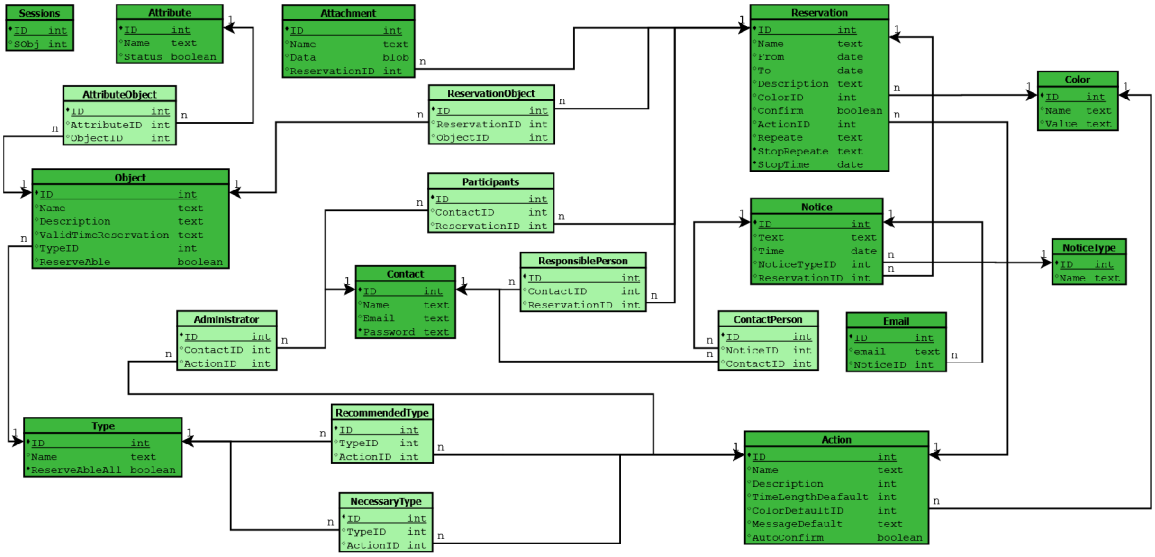
3.3 Serverová část

Nedílnou součásti návrhu serverové části systému je vytvoření *databázového modelu*, což vytvoří dobrou rozvahu ohledně ukládání a pracování s perzistentními daty.

Důležitým prvkem návrhu serverové části je také nepochybně návrh *aplikačního rozhraní*. Prostřednictvím aplikačního rozhraní bude následně klientská část získávat data ze serveru a také je na server posílat. Navržené aplikační rozhraní vytvoří ideální plán na postupnou implementaci.

3.3.1 Databázový model

Při návrhu systému je namístě vytvořit model relační databáze za pomoci Entity Relationship Diagramu, zkráceně pak ERD. ERD slouží pro vizualizaci jednotlivých entit relační databáze a jejich vzájemných vztahů.



Obrázek 3.1: ERD diagram navrhnuté relační databáze

Hlavním prvkem navrženého systému je tabulka *Reservation*. Jeden řádek v této tabulce představuje jednu rezervaci, jak lze vidět na obrázku 3.1 s ERD, tak k této tabulce je navázaných několik dalších tabulek. Tyto tabulky konkrétně reprezentují *přílohu*, *barvu* v kalendáři, *akci* přes kterou probíhala rezervace a *upozornění* pro rezervaci. Dále jsou zde vazby na tabulky *objekt* a *kontakt*. Vazba na tabulku představující objekt slouží pro uložení objektů, na které bude provedena rezervace. Jelikož rezervujících objektů může být více a zároveň jeden objekt může být rezervován několika rezervacemi je zde nezbytná 3. pomocná tabulka (*ReservationObject*). Podobně je vyřešena vazba na tabulku reprezentující kontakt, zde jsou pomocné tabulky představující *účastníky* a *zodpovědné osoby* (pořadatele rezervace).

Za zmínku stojí problematika ohledně tabulek pro *akci* a *typ*. Mezi těmito tabulkami jsou taktéž třetí pomocné tabulky pro *nezbytný* a *doporučený typ*. Podobné je to u entity pro *objekt*, která představuje konkrétní prostředek a u tabulky představující *atributy* těchto prostředků. Na zmíněnou tabulku s akcí je taktéž navázaná entita pro *administrátory* dané akce, administrátoři pak mohou mimo jiné potvrzovat rezervace daných akcí.

Zajímavá je také situace kolem *upozornění* pro rezervaci. Tato entita je také přes pomocnou tabulku navázána na *kontakt*, proto aby byla zajištěna možnost přidat k upozornění jednotlivé kontakty ze systému. Nicméně v návrhu je i tabulka *email*, sloužící pro přidání e-mailových adres kontaktů, které nejsou součásti systému. Dále je k entitě upozornění navázána tabulka pro *typ upozornění*. Dalo by se říci, že je něco takového zbytečné, protože je navržen pouze jeden typ upozornění prostřednictví e-mailu, nicméně je to prostor pro **snadné rozšíření**, například v podobě upozornění přes sms.

Poslední zmínka patří tabulce *Session* neboli sezení. Tato tabulka v systému slouží pro ukládání Cookies, z kterých lze vyčíst o jakého uživatele se jedná a není nutné, aby si vždy přihlašoval.

3.3.2 Návrh aplikačního rozhraní

Pro komunikaci mezi klientskou a serverovou částí je využit tzv. *Representational State Tranfer Application Programming Interface*, neboli více známý zkrácený tvar *REST API*. REST API nabízí jednoduchý způsob, jak vytvořit, číst, editovat nebo smazat informace ze serveru pomocí jednoduchých HTTP volání.

REST API se skládá z jednotlivých přístupových bodů, kde každý takovýto přístupový bod musí mít definovanou:

- svoji jednoznačnou URL
- standardní HTTP metodu (GET, POST, DELETE, PUT...)
- určený typ internetového média (text/plain, application/json...)

Po analýze zadaní aplikace byla navržena série přístupových bodů, která pokryla veškeré potřebné operace k získání a posílání dat potřebných k procesu vytvoření a správě rezervací. Celkově se návrh systému a s tím související komunikační protokol koncipoval do filozofie tenkého klienta, což znamená, že se na serverovou část ukryla celá logika a klientská část tuto logiku využívala prostřednictvím zasílání požadavků na jednotlivé přístupové body.

V demonstrační aplikaci je aplikační rozhraní tvořeno vždy *URL*, která jasně definuje přístupový bod, dále pak HTTP metodami, z kterých je využito *GET* a *POST* a jako typ internetového média je všude využit *application/json*, což je na platformě JavaScript extrémně výhodné, jelikož se tento formát automaticky převede na JavaScript objekty.

3.4 Klientská část

Tato sekce se zabývá návrhem klientské mobilní aplikace v NativeScriptu. S tím souvisí navržení práce s daty a navržením jednotlivého rozložení stránek v rámci aplikace.

Pro snadnější promyšlení návrhu bylo využitu tzv. mockupu[1]. Nejprve v podobě obrázků na papíře a po dospění ke konkrétnějším výsledkům v podobě elektronického zobrazení v rámci open source programu *Gimp*[2].

3.4.1 Návrh práce s daty – MVVM

NativeScript je postaven na použití návrhového vzoru *model, view, view model* neboli zkráceně MVVM. Jednotlivá slova tohoto návrhového vzoru znamenají:

[1] Mockup – návrh obrazovek sloužící pro hodnocení návrhu, propagaci a další
[2] Gimp – https://www.gimp.org/

- **Model:** *Model* definuje a reprezentuje data. Oddělení modelu od *view* (pohledů), které ho mohou použít, umožňuje opětovné použití kódu.
- **View:** *View* (pohled) představuje uživatelské rozhraní, které je v jazyce NativeScript napsáno ve formátu XML. *View* je často datově vázané na *model* tak, aby změny provedené v modelu okamžitě vyvolaly vizuální změny komponent uživatelského rozhraní.
- **View Model:** *View model* drží stav aplikace a obsahuje její logiku (často včetně *modelu*) a zobrazuje data do *view*. NativeScript obsahuje speciální *view model* s názvem *Observable*, který automaticky při změně aktualizuje uživatelské rozhraní.

Výhodou tohoto návrhového vzoru je, že lze využít tzv. *two-way data binding* neboli obousměrnou vazbu dat. To znamená, že změny od uživatele ve *view* se projeví do *modelu* a naopak.

V demonstrační aplikaci je tak tento návrhový vzor využit. Jednotlivé *view modely* jsou v oddělených souborech a jsou mnohdy typu *Observable*, pro dosažení automatizované aktualizace dat.

Jednotlivé *view modely* taktéž potřebují získat data, které budou držet. Tyto data získají jako odpověď ze serveru zaslaným požadavkem, který odpovídá navrhnutému komunikačnímu rozhraní. Veškerá komunikace se serverovou částí tedy probíhá v rámci *view modelů*.

3.4.2 Návrh grafického uživatelského rozhraní

Aplikace bude mít 3 hlavní sekce: *nová rezervace*, *můj kalendář*, *kalendář objektu* a 4. sekci dostupnou pouze pro administrátory – *potvrzení rezervací*. Pro rychlé přepínaní mezi jednotlivými sekcemi by mělo být dostupné *vysouvací menu*.

Kromě dostupných sekcí bude také k dispozici jednoduchá stránka určená pro *přihlašování*. Po úspěšném přihlášení bude uživatel přesměrován na stránku s jeho kalendářem. V menu bude taktéž tlačítko na odhlášení uživatele.

Nová rezervace

Sekce pro novou rezervaci by měla obsahovat několik stránek, které uživatele provedou kompletním procesem tvorby nové rezervace. To by mělo zahrnovat *vybrání akce*, *zvolení termínu*, *volných prostředků*, *účastníků*, *zodpovědných osob*, *upozornění*, *příloh* a *barvy do kalendáře*.

Můj kalendář

Tato sekce by měla sloužit pro zobrazení relevantních událostí přihlášeného uživatele. To znamená události, které daný uživatel buď *založil* nebo ty na které je pozván, jakožto *účastník*. Rezervace bude možné prohlížet buď v *týdenním*, *měsíčním* nebo *ročním módu*.

Po kliknutí na vybranou rezervaci budou zobrazeny detailnější informace o dané rezervaci a pokud uživatel rezervaci založil bude ji moci i *editovat*.

Kalendář objektu

V této sekci bude moci uživatel najít jakýkoliv objekt v systému a podívat se na jeho obsazenost. Z toho vyplývá, že bude nezbytné, aby tato sekce kromě kalendáře měla i stránku s *vyhledáváním prostředků*.

Kalendářová část bude stejná jako u sekce můj kalendář, avšak s tím rozdílem, že zde půjde zobrazit detaily pouze těch rezervací, které bude pořádat či budu veden jako účastník. U ostatních rezervací uvidí pouze jejich termín a název.

Potvrzení rezervací

Pokud bude přihlášený uživatel administrátor pro některou akci bude moci kromě ostatních funkcí systému taktéž potvrzovat rezervace. V této sekci by měl být pro administrátora *přehledný výpis* s jednotlivými rezervacemi k potvrzení.

Součástí potvrzovacího procesu bude možné zobrazit *plné detaily rezervace*. Dále bude také možné k potvrzení (či nepotvrzení) přiložit *zprávu*, která bude zaslána účastníkům a/anebo zodpovědným osobám.

Kapitola 4

Implementace řešení

Tato kapitola pozvolna navazuje na předešlou kapitolu zabývající se návrhem demonstrační aplikace. V této kapitole je popsán samotný proces implementace demonstrační aplikace. Implementace se dělí na dvě základní části – *implementace serveru* a *implementace klienta*. V každé části jsou popsány nejzajímavější problémy, které bylo nutné v rámci vývoje navržené aplikace řešit.

4.1 Server

Jako první část systému demonstrační aplikace byla implementována serverová část, která byla napsána v Node.js. Bylo nutné řešit řadu problémů, v této sekci budou probrány ty nejklíčovější a nejzajímavější jako je *autentizace*, *vyhledávání volných prostředků*, *opakování rezervace* či *plánování e-mailů*. Výhodou implementace v Node.js je jeho velká popularita a extrémně velká škála rozšiřujících modulů, které byly použity velkým množstvím vývojářů a troufám si říci, že jsou spolehlivé. Téměř v každé klíčové části implementace byl některý z těchto rozšiřujících modulů využit.

4.1.1 Autentizace a sezení

Autentizace slouží k jednoznačnému určení uživatele, který přistupuje k systému. Cílem autentizace je zajistit, že systém přesně ví, s jakým uživatelem komunikuje, kdo to je.

Sezení je síťové spojení mezi klientem a serverem. Nastavené sezení přispívá k automatickému procesu autentizace a autorizace uživatele. Sezení je v HTTP předáváno dvěma nejrozšířenějšími způsoby:

- jako HTTP cookie
- v URL přístupového bodu – jako její proměnná

Ukládání hesel

Hesla uživatelů jsou v databázi z bezpečnostních důvodů uloženy ne jako prostý text, ale šifrovaně. K procesu šifrování a dešifrování hesel na straně serveru v Node.js je využit rozšiřující modul *bcrypt*[1].

[1] Bcrypt – https://www.npmjs.com/package/bcrypt

Proces přihlášení

Když se chce uživatel přihlásit je nutné, aby z klientské aplikace zaslal na daný přístupový bod aplikačního rozhraní serveru svůj *e-mail* a *heslo*. E-mail a heslo jsou zadány pomocí jednoduché klientské stránky a odeslány požadavkem přes typ internetového média *application/json*. Tento typ internetového média není sám o sobě šifrován, takže by se takovéto posílání mohlo jevit jakožto bezpečnostní hrozba, avšak je počítáno s tím, že server bude s klientem komunikovat přes HTTPS protokol, z čehož vyplývá, že komunikace bude šifrovaná a bezpečná.

Po obdržení těchto přihlašovacích údajů na server je spuštěn proces autentizace. Tento proces spočívá v tom, že je nejdříve vyzkoušeno, zdali někdo s danou e-mailovou adresou existuje. Pokud neexistuje, tak je na klientskou část odeslána chybová zpráva s HTTP status kódem *401*. Pokud však existuje je z databáze vybráno i uživatelské (zašifrované) heslo. Toto zašifrované heslo je přes modul *bcrypt* porovnáno s heslem zaslaným od klienta. V případě neúspěchu je odeslána úplně stejná chybová hláška jako v případě chybného e-mailu.

Dále je pak zjištěno jestli je přihlášený uživatel zároveň administrátor pro některé akce. Tuto informaci a uživatelovo ID, celé jméno a e-mail jsou zaslány jako odpověď na přihlášení spolu s HTTP status kódem *200*.

Sezení

Zároveň s úspěšnou odpovědí o přihlášení je vytvořeno tzv. *session* neboli sezení, což je síťové spojení mezi klientem a serverem. Sezení je předáváno pomocí *HTTP cookie*. Pomocí cookie se předává ID sezení a další nastavující atributy jako doba expirace. V databázi je vytvořena tabulka *Session*, která má pod tímto ID sezení zapsané informace o uživateli, jako jeho ID či zdali je administrátor. Celý tento proces pomáhá zjednodušit modul *express-session*[2].

Klient ke každému svému HTTP požadavku přiloží do hlavičky cookies a server následně z tohoto cookies zjistí o jakého uživatele se jedná, případně zdali má právo na danou operaci. Tato autentizace pak probíhá přes middleware, který se spustí vždy při přijetí požadavku.

Na klientské straně NativeScript automaticky po přihlášení přidává tento typ cookies do dalších svých požadavků. Nicméně, rozhodl jsem se aplikaci udělat tak, aby se uživatel po každém startu aplikace nemusel přihlašovat. A tak je implementována funkcionalita, která po prvním přihlášení uživatele uloží cookies do interní paměti mobilu. A pokud uživatel znovu zapne aplikaci, tak je načteno toto cookie a je explicitně přidáno do požadavků. Pokud by došlo k chybě, například kvůli vypršené expiraci sezení, tak by byl uživatel jednoduše přesměrován na přihlašovací stránku.

Pokud se uživatel rozhodne *odhlásit*, cookies je smazáno – sezení je ukončeno. Taktéž je smazáno cookies z interní paměti telefonu a uživatel aplikace je přesměrován na přihlašovací stránku.

4.1.2 Vyhledávání volných prostředků

Stěžejní část systému je tvořena vyhledáváním volných prostředků, které je možné rezervovat. Samotný proces vyhledávání je velmi variabilní. Nejprve si uživatel vybere z jakých typů chce prostředky vybírat, případně jaké atributy by prostředek měl mít. Pak je na uživateli zdali dá rovnou vyhledat a nechá si nabídnout nějaké alternativy či začne upřesňovat

[2]Express-session – https://www.npmjs.com/package/express-session

výsledky pomocí konkrétnějšího času nebo některých konkrétních prostředků.
Lze tedy zadat některý nebo kombinaci více těchto časových údajů:

- čas nerozhoduje
- datum
- čas od
- čas do
- délka rezervace
- opakování

V rámci typu prostředku lze zvolit buď jakýkoliv nebo začít vybírat konkrétní prostředek, nicméně pokud už je vybrán nějaký časový údaj, budou nabízeny jen volné prostředky v těchto termínech.

Vyhledávací algoritmy pak hledají podle všech dostupných kritérií: čas, typ prostředků a atributy. Pokud se nalezne prostředek, který je k dispozici v daný čas, tak je přidán do seznamu výsledků. Aby byl prostředek v určitý čas k dispozici musí splňovat několik požadavků:

- Jeho nadřazený typ musí mít povolené, že lze všechny jeho prostředky rezervovat, pokud ne, tak musí mít konkrétní prostředek povolené, že je dostupný pro rezervaci.
- Čas ve který by byla rezervace musí být uvnitř doby, kdy je prostředek dostupný pro rezervaci (lze nastavit, že je konkrétní prostředek dostupný pouze od 8:00 do 17:00).
- Prostředek se časově nepřekrývá s jinou rezervací v daném čase.

Navržené aplikační rozhraní přijímá v rámci těla požadavku variabilní kombinace jednotlivých údajů a jako odpověď vrátí seznam výsledků s termíny a konkrétními prostředky. Čím více je výsledek obecný, tím více výsledků algoritmus vygeneruje. Lze nastavit limit výsledků pro jednu odpověď a také lze nastavit offset výsledků. Některé prostředky mohou mít u odpovědi příznak *general* neboli obecný, to znamená, že kromě daného prostředku pro stejný čas existují i další dostupné alternativy.

Pokud není zadán vůbec žádný časový údaj, je použita výchozí délka akce. Prohledávání začne od aktuálního data a hodiny a bude se posouvat po 30 minutách.

4.1.3 Opakování rezervace

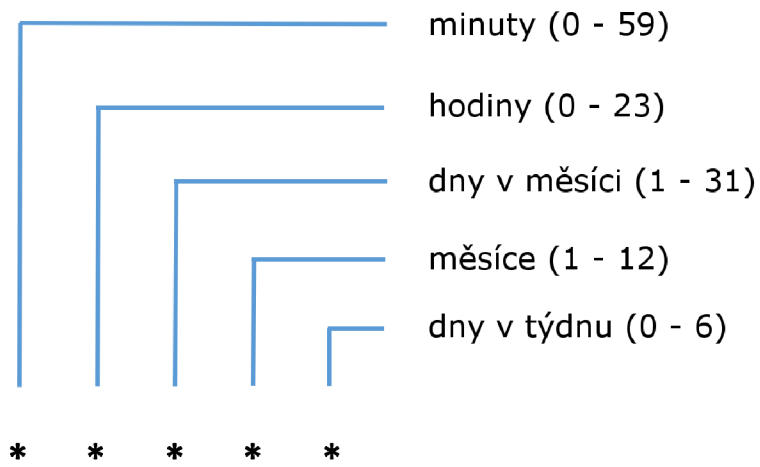
Dle návrhu lze rezervaci opakovat a to hned v několika režimech, které jsou ukončeny buď počtem opakování nebo datem. Konkrétní režimy, jak opakovat jsou:

- *Denně.*
- *Týdně*, kde lze zvolit v jaké dny v týdnu opakovat. Dále je možné zvolit, zda opakovat každý týden nebo každý Ntý týden.
- *Měsíčně*, kde lze zvolit v jaké měsíce a jaké dny v měsíci opakovat. Případně lze zvolit kombinace typu každý první pátek v měsíci.

Plánování podle CRON formátu

I když se bude rezervace opakovat, bude stále v databázi uložena pouze jednou a podle specifických atributů bude možné odvodit opakující se termíny rezervací. Z popisu problematiky je možné pozorovat, že variabilita opakování je vcelku rozsáhlá. Jedna ze zajímavých možností, jak univerzálně definovat opakování pomocí krátkého textového řetězce je CRON.

CRON je původem softwarová funkcionalita používaná v UNIXových systémech na plánování úloh, nicméně princip zápisu opakování lze využít i na jiných místech.



Obrázek 4.1: Popis zápisu ve tvaru CRON

Jak lze vidět na obrázku 4.1, zápis se skládá z pěti[3] části oddělených mezerou. Každá část představuje časový atribut (minuta, hodina...). Každý časový atribut může být pevně stanoven *číslicí* a nebo může být zapsán pomocí symbolu *hvězdičky*, který pak představuje to, že se bude událost opakovat v každou jednotku tohoto časového atributu. Pomocí *pomlčky* lze nastavit i rozsah konkrétních hodnot (9-15). Prostřednictvím *čárek* jen vybrané hodnoty z dané části (10,11,19). A za pomoci lomítka a čísla (N) je možné nastavit frekvenci platnosti – každých N jednotek daného časového atributu (*/3). Pomocí tohoto formátu lze tedy zapsat (téměř[4]) všechny potřebné kombinace pro navržené opakování.
Například:

30 11 * * 1,5 – opakovat v 11:30 každé pondělí a pátek

Generování termínů na základě CRON formátu

Rozšiřující modul pro Node.js s názvem *later*[5] umožňuje mimo jiné generování platných termínů na základě řetězce v CRON formátu.

Pak už tedy není, tak náročné vytvořit algoritmus, který na základě CRON formátu a počtu opakování (či ukončujícího data) vytvoří pole termínů, které představují opakování.

[3] CRON formát někdy bývá složen i z 6 složek, kde je na začátek přidána část s vteřinami
[4] Jisté omezení CRON formát má, jak byl nedostatek vyřešen je popsáno v podsekci níže
[5] Later – https://www.npmjs.com/package/later

Omezení CRON formátu a jeho řešení

Bohužel na řešení, které bylo navrženo CRON formát v jednom ohledu nestačí. Konkrétně pomocí CRON formátu nelze zapsat, aby se rezervace opakovala každý Ntý týden. A to z důvodu, že CRON formát neobsahuje sekci s týdny v roce.

Řešení tohoto problému je, že byl klasický CRON formát rozšířen o novou sekci. Tato sekce vyjadřuje právě ony chybějící týdny a je přidána úplně napravo a místo hvězdičky je zde použito T.
Nejlépe to lze vidět na příkladu:

30 11 * * 1,5 T/2 – opakovat v 11:30 v pondělí a v pátek *každý druhý týden*

Algoritmy pak vychází z původního plánování, akorát při takovémto formátu je poslední část nejprve odstraněna. Následně pak vygenerované celé pole termínů a z toho pak ručně vyfiltrovány odpovídající termíny.

4.1.4 Upozornění prostřednictvím e-mailu

V rámci rezervace je také možné vytvořit upozornění, které uživatelům umožňuje připomenutí rezervace nebo jim poskytuje informace o potvrzení (nepotvrzení) rezervace. Jako forma upozornění bylo navrženo e-mailové upozornění. K této skutečnosti jsou zapotřebí dvě věci – *zasílání e-mailů* a *plánování úloh*.

Odesílání emailů

Pro zasílání e-mailů byla založena demonstrační e-mailová schránka na portálu Gmail[6]. Pro zasílání zpráv přes tuto schránku prostřednictvím jiné aplikace je nutné v nastavení Gmailu tuto funkci povolit[7].

Samotná práce s e-maily v rámci Node.js je velmi jednoduchá přes rozšiřující moduly *nodemailer*[8], který se stará o samotné zasílání zpráv a *nodemailer-smtp-transport*[9], který slouží jako transportní protokol pro zasílání prostřednictvím *nodemaileru*. Pouze se zadají přihlašovací údaje vytvořené schránky spolu s hostitelským serverem a názvem služby. Dále už je zcela jednoduše možné posílat e-maily. V e-mailu lze nastavit text zprávy, předmět a příjemce. Příjemců může být klidně několik, v takovémto případě se příjemci vkládají jako pole e-mailových adres.

Plánování úloh

Jelikož samotné e-maily nechtějí být odesílány ihned, ale v určený čas, bylo potřebné implementovat plánovač úloh. Ovšem, jak je v Node.js již zvykem i na toto je zde rozšiřující modul s názvem *node-schedule*[10]. Naplánování nové úlohy je pak extrémně jednoduché zvolí se čas a jaká funkce se má vykonat.

Při tvorbě nové rezervace, která obsahuje i upozornění a je automaticky potvrzená je naplánováno odesílání e-mailu (nebo e-mailů). Dále je e-mail naplánován tehdy pokud ad-

[6] Gmail – https://www.google.com/gmail/

[7] Tato funkce je ve výchozím nastavením vypnuta kvůli vyšší bezpečnosti. Změnit toto nastavení Gmailu lze přímo na této adrese – https://myaccount.google.com/lesssecureapps/

[8] Nodemailer – https://www.npmjs.com/package/nodemailer/

[9] Nodemailer-smtp-transport – https://www.npmjs.com/package/nodemailer-smtp-transport/

[10] Node-schedule – https://www.npmjs.com/package/node-schedule/

ministrátor potvrdí rezervaci či pokud administrátor v rámci svého potvrzení přidá nějakou novou zprávu.

Kromě takovéhoto plánování e-mailů jsou také sekundárně vytvořeny kopie těchto e-mailů v databázi. To je primárně z důvodu, kdyby se restartoval či dočasně zastavil server, protože data naplánované prostřednictvím plánovače úloh nejsou perzistentní. Pokud se tedy zapne server, vždy naplánuje všechny dostupné e-maily, které byly naplánované na čas pozdější, než čas zapnutí serveru (ještě neodeslané e-maily).

4.2 Klient

Druhá část této kapitoly je věnována implementaci klientské části, která probíhala v NativeScriptu. Jsou zde popsány nejdůležitější pasáže, které nastaly během vývoje. Patří sem využití *ukázkových aplikací*, používání *nativních komponent* včetně *kalendáře* nebo implementace *intuitivního vyhledávání*. V závěru jsou taktéž zmíněny *nedostatky technologie NativeScript*.

Popis problematiky je zde mnohdy doplněn obrázky, které lépe vystihnou daný popis. Obrázek se skládá ze dvou částí. Vlevo je vždy snímek obrazovky platformy *Android* a vpravo je zobrazen snímek obrazovky vytvořené na platformě *iOS*.

4.2.1 Ukázkové aplikace

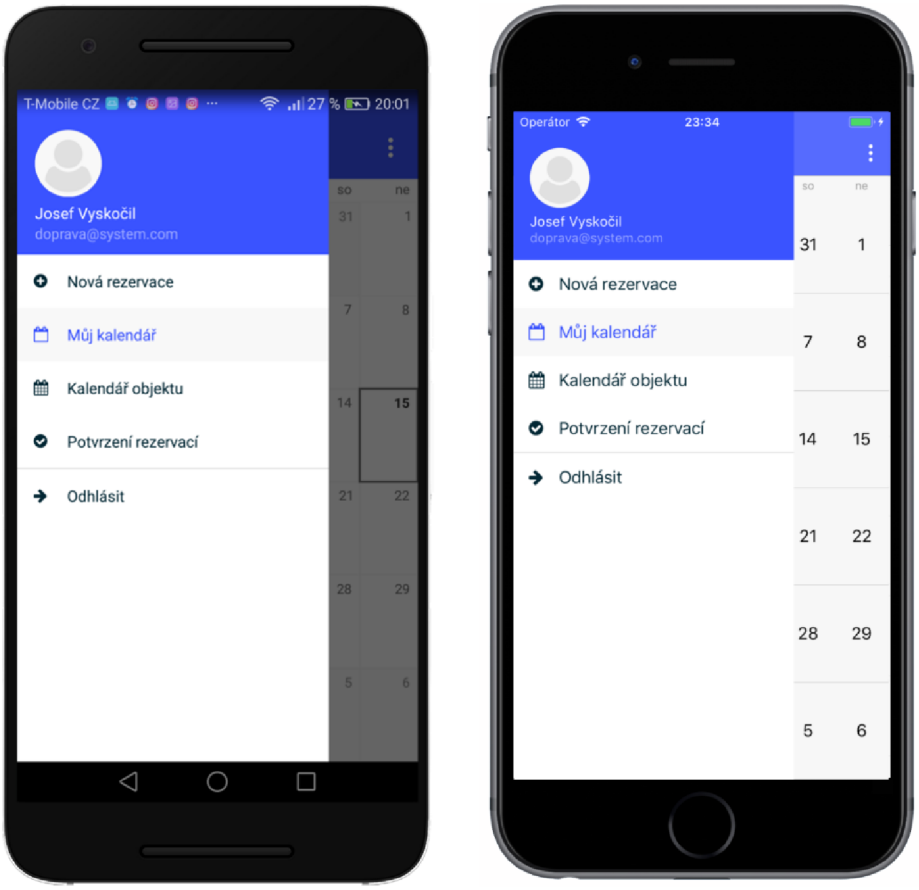
Na oficiálních stránkách NativeScriptu je sekce[11] s ukázkovými aplikacemi. Lze tu nalézt řadu jednoduchých aplikací, které demonstrují základní funkcionalitu jako jsou přechody mezi stránkami, mezi záložkami v rámci stránky, používání seznamů nebo ukázka vysouvacího menu.

Co je ale dle mého názoru hlavní, tak obsahují již vytvořený kód, který se řídí nejlepšími praktikami, má dobře poskládanou strukturu zdrojového kódu a demonstruje ideální práci s uživatelským rozhraním.

Pro vývoj demonstrační aplikace jsem se tedy rozhodl jednu z těchto šablon využít. Konkrétně je využito šablony, která demonstrovala použití menu a přepínaní mezi jednotlivými stránkami. Kromě výhod zmíněných v předchozím odstavci oceňuji i dobrý grafický základ. V aplikaci už byl nainstalován SASS pre-procesor a vytvořeny nějaké základní styly, což bylo vcelku užitečné.

Nebylo tedy vůbec obtížné začít a menu přizpůsobit dle návrhu. Na obrázku 4.2 je ukázáno, jak vytvořené vysouvací menu vypadá. Vlevo je snímek obrazovky platformy Android a vpravo snímek obrazovky platformy iOS.

[11] Ukázkové aplikace v NativeScriptu – https://docs.nativescript.org/tooling/app-templates



Obrázek 4.2: Ukázka vysouvacího menu na platformách Android a iOS

4.2.2 Nativní komponenty

Nesčetnou výhodou vývoje aplikace v NativeScriptu je její snadná tvorba multiplatformních nativních komponentů. Při implementaci bylo potřebné na řadě míst přidat komponentu představující *textové pole*. Tuto komponentu je potřebné přidat do XML souboru, který představuje rozložení uživatelského rozhraní. Výsledná komponenta je vykreslena nativně v Androidu a nativně v iOS.

Odlišným, ale stejně jednoduchým způsobem lze vyvolat přímo z JavaScriptu například komponentu představující *dialog*. Dialog může být ve více podobách jako je prostý oznamovací dialog, ale také třeba dialog s výběrem možností. Dialog se pak opět zobrazí odlišně na jednotlivé platformy, což je uživatelsky přívětivé. Při implementaci byly dialogy hojně využívány, třeba k tomu, aby uživateli oznámily nějakou zprávu či chybu nebo mu umožnili vybrat z více možností.

Některé části nešlo udělat nativně automaticky, ale bylo nutné komponentu přizpůsobit jinak na Android a jinak na iOS. Příkladem může být *Action Bar*[12], který je odlišný v Androidu a v iOS. Uživatelé Androidu jsou zvyklí na vše využívat ikonky, v iOS je zase běžné více používat popisky. Například funkce na návrat zpět je v Androidu využita přes *šipku doleva* a na iOS popiskem *Zpět*. Při této problematice bylo nutné na jedné platformě

[12] Action Bar – Lišta navrchu obrazovky, která obsahuje název stránky, tlačítka atd. pro podporu navigace a zvýraznění důležitých funkcí.

skrýt tlačítko a zobrazit popisek a na druhé naopak. Výhodou je, že v rámci XML dokumentu jsou podobné úkony dobře podporované. Aplikace při startu zjistí v jaké platformě je spuštěna a komponentám lze přidat atribut pro skrytí komponenty jen za předpokladu, že jsou spuštěny v dané platformě. Výsledek AcrionBaru lze vidět na obrázku 4.3.

Jsou zde i komponenty, které v rámci NativeScriptu nejsou přímo vytvořené, ale je možné přidat je přes rozšiřující modul. Jedním z takovýchto potřebných modulů je *nativescript-timedatepicker*[13]. Což je modul pro uživatelsky přívětivé zadávání času a data. Toto zadávání je rozdílně pro obě platformy a tento rozšiřující modul si s tím umí hravě poradit. Výsledkem je tak rozdílný způsob, jak od uživatele sbírat data, ale stejný způsob používání komponenty v rámci NativeScriptu. Modul lze v praxi vidět na obrázku 4.3, kde je použita jeho část na zadávání času.

Obrázek 4.3: Ukázka rozdílného Action Baru a odlišného přístupu k zadávání času platforem Android a iOS

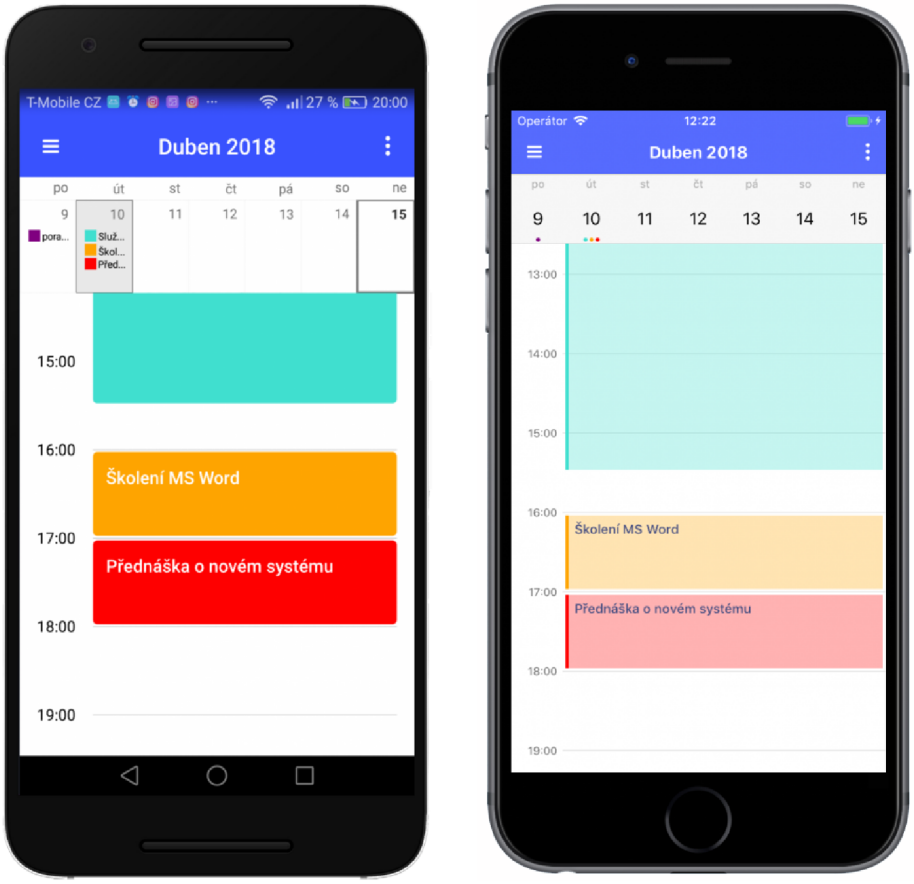
4.2.3 Kalendář

Jedna z dominantních částí demonstrační aplikace je kalendář. Dle návrhu je potřeba kalendář, který má k dispozici týdenní, měsíční a roční mód. Všechny tyto vlastnosti po vizuální stránce dokonale splňuje kalendář od tvůrců NativeScriptu. Výhodou je, že je jeho zobrazení opět rozdílné pro obě platformy. Zde nedokáži s jistotou říci jestli tvůrci využili

[13]Nativescript-timedatepicker – https://www.npmjs.com/package/nativescript-timedatepicker

nějakým způsobem nativní aplikační rozhraní pro zobrazení kalendáře či jestli vytvořili imitaci. Každopádně výsledek je velmi povedený. U Androidu se kalendář z velké části podobá *Google kalendáři* a u iOS se kalendář podobá systémovému *iCalendar*. Což je pro konečného uživatele příjemné, neboť je na takový kalendář zvyklý.

Pro uživatele relevantní události jsou staženy ze serveru a přidány do kalendáře. Do NativeScript kalendáře lze přidávat tzv. události, které se skládají z *času*, *názvu* a *barvy*. Uživatel vidí události buď v měsíčním režimu, kde se po kliknutí na konkrétní den objeví panel ze seznamem událostí, dále pak v ročním režimu, který slouží spíše jako rychlé přepínaní mezi měsíci a poslední, podle mého názoru nejzajímavější režim, je týdenní režim. V týdenním režimu, jak lze vidět na obrázku 4.4, je možné vidět dominantní hodinovou osu a nahoře lze přepínat mezi jednotlivými dny v týdnu.



Obrázek 4.4: Ukázka kalendáře v týdenním režimu na obou platformách

Po kliknutí na událost je uživatel přesměrován na jednoduchou stránku, kde jsou prezentovány detailnější informace o rezervaci. Pokud je uživatel pořadatel rezervace, může rezervaci dokonce editovat. U kalendáře, kde uživatel nahlíží na konkrétní objekty může tyto detailní informace vidět, jen za předpokladu, že je účastník nebo pořadatel rezervace.

Navržené aplikační rozhraní serveru umožňuje stahovat rezervace buď v rámci týdne nebo v rámci měsíce a s možným ofsetem. Na demonstrační aplikaci stahuji rezervace vždy v rámci měsíce. Rezervace se stahují a přidávají *asynchronně*, což znamená, že během stahování lze normálně kalendář používat. Po spuštění kalendáře jsou nejdříve staženy události na aktuální měsíc. Hned po té se začnou stahovat události na následující a předešlý měsíc.

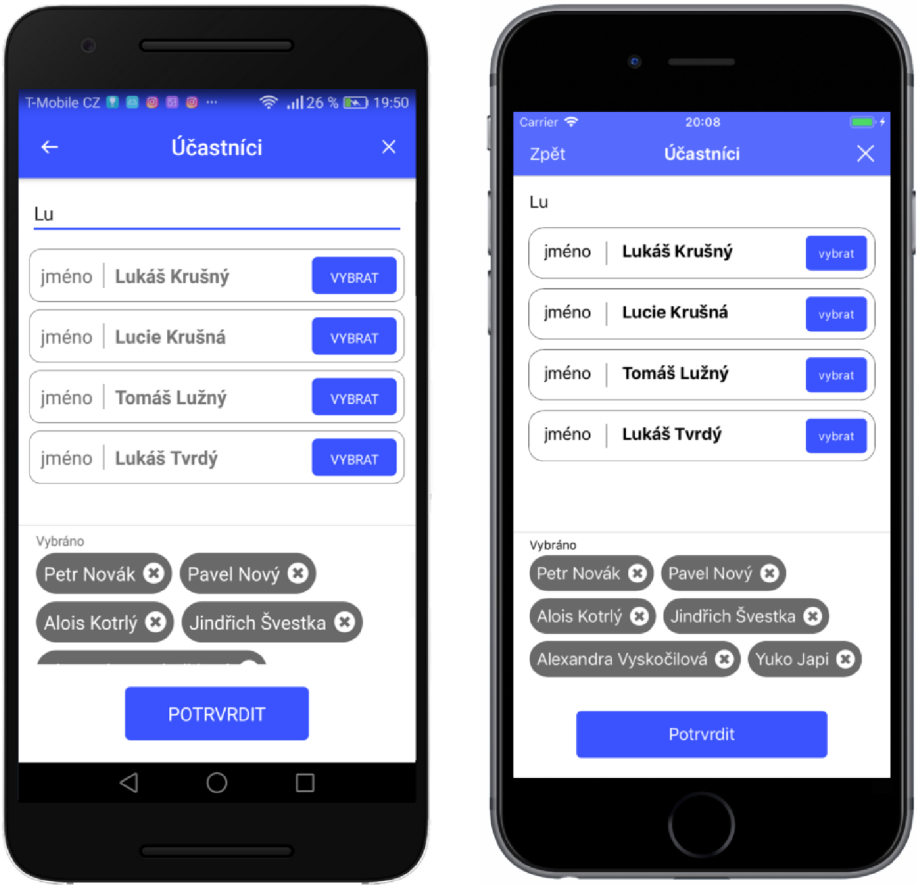
Pokud se uživatel posune na další měsíc začnou se automaticky stahovat události na měsíc o jeden dál v daném směru. Tato skutečnost umožní uživateli plynulé stahování, které ani nepostřehne. Dále je zde implementován mechanizmus, který před zasláním požadavku o stažení nejdříve zkontroluje, jestli už náhodou daný měsíc není stažený.

4.2.4 Intuitivní vyhledávání osob

V aplikaci je na více místech nutné vybrat osoby (přidání účastníků, zodpovědných osob či dalších kontaktů). Byla tedy implementována funkcionalita, která pomůže uživateli jednoduše přidávat a vyhledávat osoby.

Ze serverové části jsou staženy všechny kontakty. Následně jsou přidány do *seznamu* stránky. Kromě tohoto seznamu jsou na stránce i další dvě části. Dole je oblast, kde jsou zobrazeny již *vybrané kontakty* a úplně nahoře je komponenta pro vstupní text, který slouží jako *vyhledávač*.

Pokud uživatel začne psát text do vyhledávače je zachycena událost na změnu textu a tento text se zkusí porovnat se jmény v seznamu. Hledá se shoda a to nejen se jménem, ale i s příjmením. Shodné výsledky jsou pak zobrazeny v seznamu, který je už filtrovaný. Čím delší slovo uživatel napíše, tím přesnější bude mít výsledky.



Obrázek 4.5: Ukázka intuitivního vyhledávání osob

Pokud uživatel klikne na tlačítko *vybrat* u některé položky seznamu, tak je tato položka – *jméno* přidána do spodní části stránky – do sekce *vybráno*. Zároveň je z původního seznamu

vyřazena (aby nemohl uživatel vyhledat jednu osobu vícekrát). V sekci vybráno, jak si je možné všimnout na obrázku 4.5, je také u každého jména křížek. Po kliknutí na křížek je ze seznamu s vybranými osobami položka odstraněna a naopak je zpět přidána do původního, vyhledávacího seznamu.

Tento vyhledávací mechanizmus je také použit na vyhledávání konkrétních objektů, což je stránka, která předchází stránce s *kalendářem objektu*.

4.2.5 Nedostatky technologie NativeScript

Implementaci klientské části aplikace hodnotím úspěšně, všechny základní aspekty návrhu se podařilo splnit a tím správně otestovat vybrané technologie, nicméně narazil jsem i na nějaké drobné nedostatky o kterých je fér se zmínit.

U kalendáře v měsíčním režimu, po kliknutí na detail dne je vypsán seznam událostí. Událost je popsaná názvem a časem. Bohužel čas je pouze ve 12 hodinovém formátu, což není žádoucí a nelze to nijak změnit. Řešením problému by bylo použít jiný kalendář, avšak reálně lepší není k dispozici. Dalším řešením by mohlo být vytvořit si svůj kalendář nebo deaktivovat zobrazení detailu dne. Nicméně pozitivní je, že jsem během pátrání o tom, jak to vyřešit, nalezl na fóru NativeScriptu[14], že přidání 24 hodinového formátu je označeno jako nová funkce, která bude implementována.

Druhý a poslední nedostatek, který jsem během vývoje zaznamenal byl v problematice nahrávání příloh. V rámci Androidu vše funguje perfektně, podařilo se udělat funkcionalitu, která otevře souborový systém, uživatel vybere soubor v libovolném formátu například pdf. Soubor se nahraje a pošle na server. U iOS toto bohužel nejde. Do jisté míry to funguje, ale lze takto ze zařízení nahrát pouze fotografie z galerie. Nicméně zjistil jsem[15], že uživatelé iOS nemůžou souborovým systém (kromě galerie) procházet ani v jiných aplikacích a to z bezpečnostních důvodů. Není to tedy ani tak chyba NativeScriptu jakožto spíš jiná filozofie iOS. A jak uživatelé iOS přikládají například pdf soubor ze souborového systému k e-mailu? Ze souborového systému to není možné, ale je to řešeno tak, že lze soubor nahrát z externího uložiště jako je *iCloud*[16] nebo *Dropbox*[17]. Pátral jsem tedy, zdali je v rámci NativeScriptu možné napojit se na externí uložiště a získat z něho data. Ovšem NativeScript ani žádný rozšiřující modul to nyní nepodporuje. Nabízelo se prověřit, jak je na tom konkurenční React Native. Výsledek pátraní je, že existuje rozšiřující modul[18], který by toto napojení na externí uložiště měl podporovat.

[14]Plán přidání funkce 24 hodinového formátu – https://github.com/telerik/nativescript-ui-feedback/issues/394/

[15]Problematika přidání příloh k e-mailu u iPhone – https://www.imore.com/how-add-email-attachments-mail-iphone-and-ipad

[16]iCloud – https://www.icloud.com/

[17]Dropbox – https://www.dropbox.com/

[18]Rozšiřující modul pro React Native na napojení externího uložiště – https://www.npmjs.com/package/react-native-document-picker/

Kapitola 5

Testování

Testování je nedílnou součástí vývoje softwaru. Tato kapitola je rozdělena na tři části, které popisují celý proces testování. První část se věnuje popisu *testování serverové části* pomocí integračních testů, další část popisuje *testování klientské části* pomocí jednotkových testů a poslední sekce patří *testování od uživatelů* na různých zařízeních.

5.1 Serverové testování

V rámci testování serverové části bylo navrženo *integrační testování aplikačního rozhraní*. To znamená, že se testují jednotlivé koncové body aplikačního rozhraní přímo zasláním požadavků a porovnává se správnost těla odpovědi a status kódu. Na testování v rámci Node.js (nebo i JavaScriptu) obecně je velmi rozšířený testovací framework s názvem *chai*. Tento framework obsahuje klasické testovací funkce jako je podpora očekávajících funkcí (*expect*), rozdělení testů do *sekcí*, *bloků* či vytvoření *speciálních bloků*, které se provedou před testy nebo po testech. Pro testování aplikačního rozhraní serveru existuje rozšíření frameworku chai s názvem *chai-http*. Toto rozšíření podporuje jednoduchou práci s posíláním požadavků a přijímání odpovědí se snadným rozborem.

Aby bylo možné otestovat konkrétní data odpovědi bylo nutné vytvořit *testovací data*. Aby bylo možné pracovat pouze s testovacími daty, které neovlivní skutečnou databázi demonstrační aplikace, tak bylo použito řešení v podobě vytvoření *nové testovací databáze*.

Celý proces tedy vypadá tak, že se spustí testovací modul. Vytvoří se testovací databáze, do které se vzápětí vloží testovací data. Spustí se série testů. Výsledky testů se vyhodnotí a vypíšou v rámci konzole. A na závěr se tato testovací databáze odstraní.

Testuje se většina koncových bodů aplikačního rozhraní. Každý tento bod je otestován řadou testů, které otestují jak běžné použití, tak i to které se záměrně snaží zadávat nečekané požadavky. Celý testovací modul se spouští jednoduše v příkazové řádce příkazem: *npm test*.

5.2 Klientské testování

Jelikož byla většina logiky ukryta na serverové části, tak na klientské části bylo množství věcí, které by se daly testovat o poznání menší. V rámci mobilní aplikace jsem se tedy rozhodl pro *jednotkové testy*. To znamená, že se testuje vždy jen jedna funkce. K testování byl opět použit testovací framework *chai*.

Testují se všechny funkce, které pomáhaly aplikaci s její logickou funkcionalitou. Většinou jde o pomocné funkce, které pracují s časem či úpravou textů. Testy aplikace se spouštějí stejným způsobem jako na serverové části – v příkazové řádce stačí zadat: *npm test*.

5.3 Uživatelské testování

Poslední částí testování systému bylo samotné testování od konečných uživatelů. Cílem tohoto testování bylo zjistit jestli uživatelé dokáží s aplikací pracovat, případně zjistit, co jim dělá problémy nebo co by na aplikaci změnili. Na základě této zpětné vazby bylo drobně změněno uživatelské rozhraní, ale nejednalo se o nic zásadního. Do testování se zapojilo celkem 9 lidí, z čehož 5 osob byli vývojáři softwaru a zbývající 4 byli nepoučení uživatelé, avšak plně znalí v používání chytrých telefonů.

Druhý cíl tohoto testování byl ověřit aplikaci v rámci více zařízení. Android aplikace byla testována na mobilních zařízení *Honor 7, Huawei Nova 3, Huawei P9 lite, Huawei P10, Samsung Galaxy J3* a na tabletu *Samsung Galaxy tab 2*. U iOS bylo testováno hardwarově pouze na *iPhone 6*. Více iPhone telefonů bohužel nebylo k dispozici a tak jsem testoval na emulátoru mobilního telefonu na počítači, v kterém byly odsimulované iPhone telefony verze *7, 7 plus, 8, 8 plus* a *iPhone X*. Aplikace byla všude plně funkční, avšak na iPhone byly lehce měněn stylový předpis, jednalo se většinou o úpravu odsazení či velikosti. Žádné větší zásahy nebyly nutné.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo seznámení s možnými metodami, jak vyvíjet *multiplatformní, nativní aplikace v JavaScriptu* na architektuře klient-server. V rámci textu práce byly popsány jednotlivé přístupy, jak vyvíjet mobilní aplikace a to od těch starších, konzervativnějších přístupů až po ty moderní, kde největší prostor dostal přístup, ve kterém jsou aplikace vyvíjeny pomocí JavaScriptu a za běhu interpretované do nativního kódu více platforem. Hlavními zástupci tohoto přístupu jsou technologie *NativeScript* a *React Native*, a tak proběhla jejich hlubší analýza a vzájemné porovnání. Práce dále poskytla *popis technologií*, které souvisí s problematikou aplikací na architektuře klient-server prostřednictvím JavaScriptu. Zde byl popis rozšířen o technologie jako je serverový Node.js, knihovna pro komunikaci s databázemi Knex.js či CSS pre-procesor SASS. Prostor pro popis dostala také jedna z přelomových verzí JavaScriptu – EcmaScript 2015 (ES6).

Bakalářská práce dále obsahuje popis návrhu a implementace *demonstrační aplikace*, která slouží jako prostředek na prezentování integrace jednotlivých technologií. Pro tuto aplikaci byla na serverovou část zvolena technologie *Node.js* a na klientskou část *NativeScript*. V práci jsou probrány nejzajímavější a nejklíčovější problémy, které nastaly během vývoje.

S vývojem serverové části v Node.js jsem byl velice spokojený. Je znát, že se tato technologie stává hodně populární, což se odráží na velkém množství kvalitní dokumentace a hlavně na obrovském počtu rozšiřujících modulů. V rámci vývoje byl na každý závažnější problém nějaký rozšiřující modul využit a nebylo nutné zabývat se rutinními záležitostmi, ale spíše se věnovat samotné logice, což zpětně velmi oceňuji.

Nejdůležitější část vývoje bylo bezesporu použití samotné technologie NativeScript. Tuto technologii hodnotím kladně, podařilo se vyvinout (až na drobné detaily) to, co bylo navrženo a hlavně se podařilo úspěšně implementovat nativní aplikaci pro platformy Android a iOS za použití jednoho zdrojového kódu. Na druhou stranu, na technologii bylo částečně znát, že je ještě relativně nová a není k dispozici například tolik rozšiřujících modulů. Avšak troufám si říci, že situace bude do budoucna pozitivní a to hlavně kvůli možné integraci NativeScriptu s technologiemi jako Angular, Vue či TypeScript. Věřím, že tato integrace způsobí příliv nových vývojářů. Celkově si však myslím, že implementace mobilních aplikací tímto stylem vývoje má obrovský potenciál a v budoucnu bude hojně používána.

6.1 Rozšíření práce

Zajímavým námětem na rozšíření bakalářské práce by bylo *implementovat úplně stejnou demonstrační aplikaci* prostřednictvím technologie React Native. Studie technologií by pak mohla být více obsáhlejší. Daly by se jednak porovnávat samotné poznatky z vývoje, ale také třeba velikost aplikace, nativní vzhled, nároky na hardware nebo porovnání rychlosti odezvy, spuštění či kompilace.

Z hlediska rozšíření implementované mobilní aplikace vidím přímočarou příležitost v podobě *vývoje webové aplikace*. Výhodou systému je, že drtivá většina jeho funkcionality je ukryta na serverové části, tudíž by přidání další klientské části nebylo tak obtížné.

Literatura

[1] Anderson, M. J.: *Getting Started with NativeScript*. Packt Publishing, 2016, ISBN 978-17-858-8865-6.

[2] Debill, E.: *Module Counts*. [Online; navštíveno 10.1.2018].
URL http://www.modulecounts.com/

[3] International Data Corporation: *Worldwide Smartphone OS Market Share*. [Online; navštíveno 6.11.2017].
URL https://www.idc.com/promo/smartphone-market-share/os/

[4] Microsoft: *Understanding the Xamarin Mobile Platform*. [Online; navštíveno 2.11.2017].
URL https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/understanding-the-xamarin-mobile-platform/

[5] Nguyen, D.: *Node.js: Okamžitě*. Computer Press, první vydání, 2016, ISBN 978-80-251-4820-4.

[6] Prusty, N.: *Learning ECMAScript 6*. Packt Publishing, 2015, ISBN 978-17-858-8653-9.

[7] Quora: *How does React Native work?* [Online; navštíveno 6.11.2017].
URL https://www.quora.com/How-does-React-Native-work/

[8] Quora: *How popular is Node.js in 2017?* [Online; navštíveno 21.10.2017].
URL https://www.quora.com/How-popular-is-Node-js-in-2017/

[9] Statista: *Number of smartphone users worldwide from 2014 to 2020*. [Online; navštíveno 10.10.2017].
URL https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[10] Telerik Developer Network: *What is a WebView?* [Online; navštíveno 2.11.2017].
URL https://developer.telerik.com/featured/what-is-a-webview/

[11] Wikibooks: *Client/Server Evolution*. [Online; navštíveno 16.10.2017].
URL https://en.wikibooks.org/wiki/A_Bit_History_of_Internet/Chapter_5_:_Client-Server#Client.2FServer_Evolution/

[12] Žára, O.: *JavaScript: Programátorské techniky a webové technologie*. Computer Press, první vydání, 2015, ISBN 978-80-251-4573-9.

Příloha A

Obsah DVD

- `readme.txt` – Soubor ve kterém jsou pokyny, jak nainstalovat a spustit systém. A to včetně klientské i serverové části. Dále jsou zde pokyny, jak spustit databázový skript či testování systému.
- `/client` – Adresář se zdrojovými kódy klientské části a vygenerovaná dokumentace.
- `/server` – Adresář se zdrojovými kódy serverové části a vygenerovaná dokumentace.
- `/thesis_pdf` – Adresář s písemnou zprávou bakalářské práce ve formátu *Portable Document Format* neboli pdf.
- `/thesis_latex` – Adresář s písemnou zprávou bakalářské práce ve formátu LaTeX.