

Využití realtime databází v Ionic frameworku

Bc. Michal Třaskalík

Diplomová práce
2018



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2017/2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Michal Třaskalík**
Osobní číslo: **A16141**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Informační technologie**
Forma studia: **prezenční**

Téma práce: **Využití realtime databází v Ionic frameworku**
Téma anglicky: **Using Real-time Databases in the Ionic Framework**

Zásady pro vypracování:

1. **Nastudujte problematiku tvorby mobilních aplikací pomocí vývojového frameworku Ionic.**
2. **Zaměřte se jak na komerční realtime databáze, především Firebase, tak na open-source databáze a vybrané zástupce těchto kategorií stručně popište.**
3. **Srovnajte možnosti použití komerční databáze Firebase s open-source řešeními, zejména v oblasti integrace s vývojovým frameworkem Ionic.**
4. **Vytvořte ukázkové implementace mobilních aplikací, z nichž jedna bude využívat komerční řešení založené na Firebase a druhá na některém z open-source realtime databázových řešení.**
5. **Srovnajte způsoby implementace a dostupnost vývojových nástrojů.**
6. **Otestujte aplikace na reálném mobilním zařízení.**

Rozsah diplomové práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **Ionic Framework Cookbook.** Birmingham, Spojené království: Packt Publishing, 2015. ISBN 1785287974.
2. **CHENG, Fu. Build Mobile Apps with Ionic 2 and Firebase: Hybrid Mobile App Development.** New York, USA: Apress, 2017. ISBN 1484227379.
3. **Hybrid Mobile Development with Ionic.** Birmingham, Spojené království: Packt Publishing, 2017. ISBN 1785280287.
4. **Getting Started with RethinkDB.** Birmingham, Spojené království: Packt Publishing, 2016. ISBN 1785884468.
5. **Learning TypeScript.** Birmingham, Spojené království: Packt Publishing, 2015. ISBN 1783985550.
6. **Web Standards: Mastering HTML5, CSS3, and XML. 2.** New York, USA: Apress, 2014. ISBN 1484208838.

Vedoucí diplomové práce:

Ing. Radek Vala, Ph.D.

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

1. prosince 2017

Termín odevzdání diplomové práce:

16. května 2018

Ve Zlíně dne 11. prosince 2017



doc. Mgr. Milan Adámek, Ph.D.
děkan



prof. Mgr. Roman Jašek, Ph.D.
garant oboru

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 14. 5. 2018


.....
podpis diplomanta

ABSTRAKT

Diplomová práce se zabývá realtime databázemi a jejich využití v rámci vývojového frameworku Ionic. Hlavním cílem této práce je srovnání možností komerční realtime platformy Firebase s jedním zástupcem z open-source řešení. Součástí práce jsou dvě mobilní aplikace, které slouží jako ukázka práce s danými databázemi. V teoretické části této práce je stručně popsán framework Ionic a technologie, které pod něj spadají. Následně jsou popsány jednotlivé součásti většiny realtime databázových systémů. Poté jsou stručně popsány jednotliví zástupci databází, a to jak z oblasti komerční, tak i ze světa open-source. V praktické části této práce jsou popsány ukázkové aplikace, které byly vytvořeny za pomoci vývojového frameworku Ionic. Nakonec této práce jsou zástupci jednotlivých databází zhodnoceni a porovnání.

Klíčová slova: Realtime databáze, Framework Ionic, Firebase, MongoDB, RethinkDB, CloudBoost

ABSTRACT

This master's thesis deals with the real-time databases and their use within the Ionic framework. The main objective of this thesis is comparing options of real-time commercial platform Firebase with one representative from an open-source solution. Part of thesis is two mobile applications, which serve as example of working with these databases. In the theoretical part of this thesis is briefly described the Ionic framework and the technology that it uses. Subsequently, the components of the real-time database systems are described. After that, individual representatives of databases are briefly described. In the practical part of this thesis are described sample applications, which were created by using the Ionic framework. Finally, the representatives of individual databases are evaluated and compared.

Keywords: Realtime database, Framework Ionic, Firebase, MongoDB, RethinkDB, CloudBoost

Děkuji vedoucímu mé diplomové práce, panu Ing. Radkovi Valovi, Ph.D. za pomoc, cenné rady a připomínky, poskytnuté materiály a čas, který mi během práce poskytl.

OBSAH

ÚVOD	8
I TEORETICKÁ ČÁST	9
1 FRAMEWORK IONIC	10
1.1 APACHE CORDOVA.....	11
1.2 ANGULAR.....	13
1.2.1 Moduly	15
1.2.2 Komponenty	15
1.2.3 Šablony, direktivy a data-binding	15
1.2.4 Služby a Dependency Injection.....	16
1.2.5 Routing.....	17
1.3 TYPESCRIPT	18
1.3.1 Typová anotace	19
1.3.2 Deklarované soubory	19
2 REALTIME DATABÁZE A JEJICH POROVNÁNÍ S SQL DATABÁZEMI	20
2.1 BAAS (BACKEND AS A SERVICE).....	22
2.2 TECHNOLOGIE VYUŽÍVANÉ V REALTIME DATABÁZOVÝCH SYSTÉMECH.....	23
2.2.1 NoSQL databáze	23
2.2.2 Serverová aplikace	24
2.2.3 Websocketové připojení.....	26
2.3 KOMERČNÍ REALTIME DATABÁZE.....	27
2.3.1 Firebase	27
2.3.2 CloudBoost.....	31
2.4 OPEN-SOURCE REALTIME DATABÁZE.....	32
2.4.1 RethinkDB.....	32
2.4.2 MongoDB.....	33
II PRAKTICKÁ ČÁST	35
3 CHATOVACÍ APLIKACE	36
3.1 FUNKČNÍ A NEFUNKČNÍ POŽADAVKY	36
3.1.1 Funkční požadavky	36
3.1.2 Nefunkční požadavky.....	37
3.2 STRUKTURA APLIKACE	37
3.3 IMPLEMENTACE	38
3.3.1 Zvolené technologie	39
3.3.2 Komunikace s Firebase databází	40
3.3.3 Základní funkčnost a způsob použití.....	45
4 APLIKACE NA SDÍLENÍ ZAJÍMAVÝCH BODŮ NA MAPĚ	52
4.1 FUNKČNÍ A NEFUNKČNÍ POŽADAVKY.....	52
4.1.1 Funkční požadavky	52
4.1.2 Nefunkční požadavky.....	52

4.2	STRUKTURA APLIKACE	52
4.3	IMPLEMENTACE	54
4.3.1	Zvolené technologie	54
4.3.2	Komunikace s MongoDB.....	55
4.3.3	Základní funkčnost a způsob použití.....	56
5	POROVNÁNÍ IMPLEMENTOVANÝCH REALTIME DATABÁZÍ.....	59
5.1	INTEGRACE S VÝVOJOVÝM FRAMEWORKEM IONIC	59
5.2	DOSTUPNOST VÝVOJOVÝCH NÁSTROJŮ	59
5.3	STRUKTURA DATABÁZE.....	60
5.4	CELKOVÉ POROVNÁNÍ	60
	ZÁVĚR	62
	SEZNAM POUŽITÉ LITERATURY.....	63
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	66
	SEZNAM OBRÁZKŮ	67
	SEZNAM TABULEK.....	68
	SEZNAM ZDROJOVÝCH KÓDŮ	69
	SEZNAM PŘÍLOH.....	70

ÚVOD

V dnešní době jsou mobilní zařízení používána stále častěji jako komunikační nástroje, či dokonce nástroje pro ovládání nebo monitorování reálných zařízení. S tímto faktem jsou tedy kladeny stále větší a větší požadavky na mobilní aplikace, které zobrazují nebo zpracovávají data z datových úložišť. Konkrétně se jedná o rychlost komunikace mobilních zařízení mezi serverovou částí a koncovou mobilní aplikací. Ve stále více případech je nutné tuto komunikaci zajistit v reálném čase.

Díky tomu začaly růst nároky na schopnost databází efektivně ukládat a zpracovávat velké množství dat, s tímto problémem se začaly potýkat hlavně tradiční relační databáze. Především pokud jde o aplikaci velkého rozsahu, která obsluhuje mnoho uživatelů, kteří operují se stejnými daty ve stejný čas. V tomto případě se začaly stávat tradiční relační databáze neadekvátní až nepoužitelné pro tyto případy. Toto vedlo k rozvoji tzv. NoSQL databází, jedná se o druh databáze, která nevyužívá jazyka SQL pro manipulaci s daty, ale využívá jiný mechanismus ukládání dat než relační způsob.

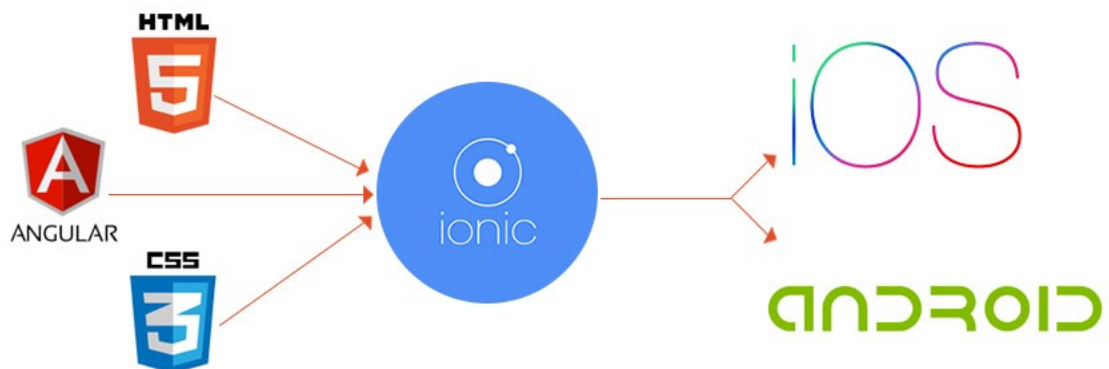
Na trhu se již dnes pohybuje spousta poskytovatelů realtime databází, hlavně takzvaných BaaS (Backend as a Service), kteří nabízejí celou řadu služeb od databází až po autentizaci. Jejich cílem je také ulehčit vývojářům práci ohledně serverové části, jehož celou logiku (backend) přebírá poskytovatel, který se stará o veškerá data, hostuje databázi, spravuje autentizaci uživatelů a vývojářům nabízí pohodlný management. Mezi takové poskytovatele patří i cloudová služba Firebase.

V této práci budou vytvořeny dvě mobilní aplikace za pomoci vývojového frameworku Ionic, z nichž jedna bude využívat služeb již zmíněného Firebase a druhá aplikace bude využívat jednoho ze zástupců open-source databází.

I. TEORETICKÁ ČÁST

1 FRAMEWORK IONIC

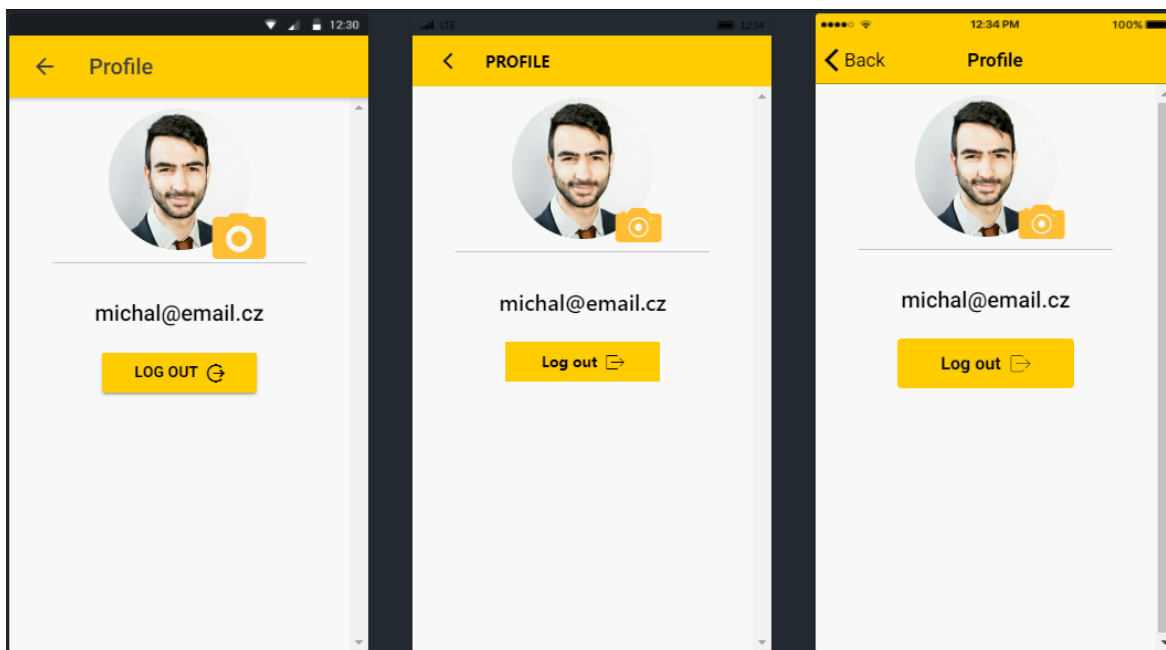
Ionic je vývojový framework pro vývoj mobilních aplikací zaměřený na budování hybridních aplikací. Hybridní aplikace jsou malé webové stránky spuštěné v prostředí prohlížeče, které mají přístup k nativním prvkům dané platformy. Ionic využívá několik dalších frameworků a svým vlastním způsobem je všechny zastřešuje, využívá jejich potenciálů a funkcí, a tak jako celek vytváří framework s otevřeným zdrojovým kódem umožňující vývoj hybridních mobilních aplikací. Tento framework je též někdy označován jako framework frameworků, a to dosti oprávněně. Ionic byl vydán v roce 2013 a tehdy byl postaven pouze na frameworkích AngularJS a Apache Cordova. [1, 2]



Obrázek 1 – Využívané technologie frameworkem Ionic [3]

Framework Ionic poskytuje nástroje a služby pro vývoj hybridních mobilních aplikací, a to za pomoci webových technologií jako jsou HTML5, CSS, nebo Sass. Takto vytvořený projekt se sestaví a díky právě frameworku Apache Cordova se vytvoří instalační balíček podle toho na které platformy se daná aplikace bude distribuovat. [2, 3]

Díky Ionic frameworku mají aplikace nativní styl mobilního uživatelského rozhraní a rozvržení. Prvky uživatelského rozhraní, jako jsou tlačítka, vstupní pole pro text nebo dokonce i ikonky, automaticky přizpůsobuje dané platformě. Díky tomuto bude výstupní aplikace vypadat odlišně jak na iOS, tak i na Androidu. Dané prvky budou tedy přizpůsobeny té dané platformě. [1]



Obrázek 2 – Rozdíl v interaktivních prvcích a rozložení na jednotlivých platformách (zleva Android, Windows Phone a iOS)

1.1 Apache Cordova

Apache Cordova je framework pro vývoj mobilních aplikací původně vytvořený firmou Nitobi, kterou v roce 2011 odkoupila společnost Adobe Systems. Apache Cordova umožňuje softwarovým programátorům vytvářet aplikace pro mobilní zařízení pomocí programovacích jazyků CSS3, HTML5 a JavaScript místo toho, aby se spoléhalo na API specifické pro platformu jako jsou například Android, iOS, nebo Windows Phone. Tento framework umožňuje zabalení kódu vytvořeného v těchto programovacích jazycích v závislosti na platformě daného zařízení. Výsledné aplikace jsou hybridní, což znamená, že nejsou skutečně, i když na první pohled nelze poznat rozdíl, nativní mobilní aplikace, protože veškeré rozvržené vykreslování se provádí prostřednictvím webového zobrazení namísto nativního UI frameworku, Nejedná se pouze o webové aplikace, jsou totiž zabaleny jako aplikace pro distribuci a mají přístup k API nativního zařízení. [4]

Apache Cordova je open-source software umožňující obalení jiným frameworkem jako jsou například Monaca, TACO, App Builder, Framework7, nebo již zmíněným Ionic frameworkem. [4]

Jádro aplikací Apache Cordova využívá CSS3 a HTML5 pro jejich vykreslování a JavaScript pro jejich logiku. HTML5 poskytuje přístup k základnímu hardwaru jako je

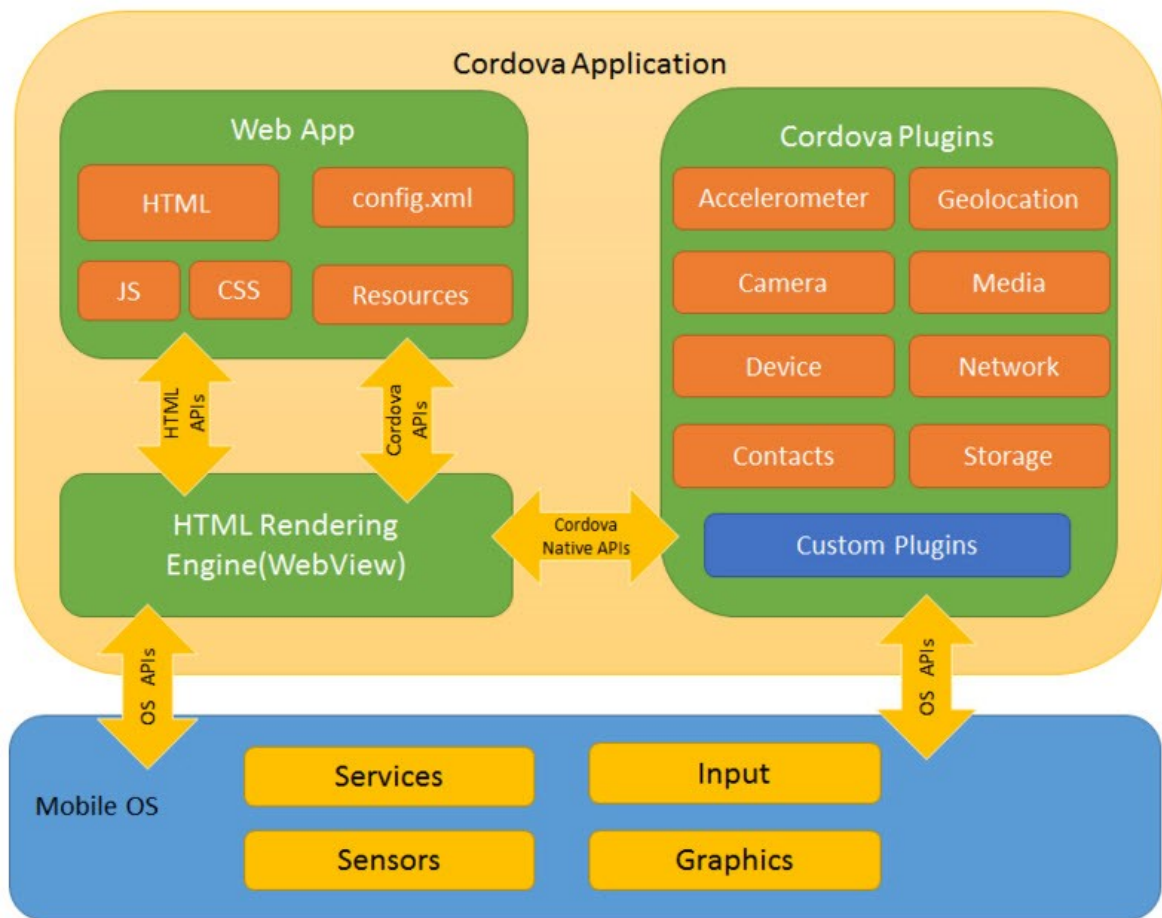
akcelerometr, fotoaparát nebo i GPS. Nicméně podpora prohlížečů pro přístup k těmto prvkům není konzistentní, zejména ve starších verzích systému Android. K překonání těchto omezení Apache Cordova vkládá HTML5 kód do nativního WebView na zařízení pomocí FFI¹ (Foreign function interface) pro přístup k jeho původním zdrojům. Níže uvedená tabulka obsahuje seznam podporovaných funkcí v uvedených operačních systémech. [4]

Vlastnosti	Android	Apple iPhone 3G	Apple iPhone 3GS a novější	BlackBerry 10 a PlayBook OS	BlackBerry OS 4.6-4.7	BlackBerry OS 5.0-6.0	Firefox OS	Windows Phone
Akcelerometr	Ano	Ano	Ano	Ano	N / A	Ano	Ano	Ano
Fotoaparát	Ano	Ano	Ano	Ano	N / A	Ano	Ano	Ano
Kompas	Ano	N / A	Ano	Ano	N / A	N / A	Ano	Ano
Kontakty	Ano	Ano	Ano	Ano	N / A	Ano	Ano	Ano
Soubor	Ano	Ano	Ano	Ano	N / A	Ano	N / A	Ano
Geolokace	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
Média	Ano	Ano	Ano	Ano	N / A	N / A	N / A	Ano
Síť	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
Oznámení	Ano	Ano	Ano	Ano	Ano	Ano	Ano	Ano
Úložný prostor	Ano	Ano	Ano	Ano	N / A	Ano	Ano	Ano

Tabulka 1 – Seznam podporovaných funkcí v daných operačních systémech [4]

Apache Cordova má v základu několik pluginů, které nabízí přístup k některým základním službám jako jsou například kontakty, kamera a tak podobně. Navíc lze rozšířit o nativní pluginy, což vývojářům přidává další funkce, které lze volat z jazyka JavaScript, tyto pluginy umožňují přístup k akcelerometru, fotoaparátu, kompasu, souborovému systému, mikrofonu a k dalším. [2, 4]

¹ FFI (Foreign function interface) – mechanismus, kterým může program napsaný v jednom programovacím jazyce volat rutiny nebo využívat služby napsané v jiném programovacím jazyce.



Obrázek 3 – Blokové schéma funkčnosti Apache Cordova [3]

1.2 Angular

Angular je webový MVC (Model View Controller) framework a platforma pro vytváření klientských aplikací v jazycích HTML a TypeScript. Celý Angular je sám napsán v TypeScriptu a tím se hlavně odlišuje od svého předchůdce AngularJS. Je tedy zřejmé, že veškeré knihovny implementované v jazyce TypeScript je možné využívat i v Angularu. Angular je vyvinutý týmem společnosti Google a komunitou jednotlivců a společností. V době psaní této práce se Angular nachází již ve verzi 5.2 a zanedlouho má být vydána verze 6.0. [7]

Základním stavebním kamenem každé Angular aplikace jsou NgModules, ty poskytují kompilační kontext pro komponenty. Angular aplikace je definována sadou NgModulů. Aplikace má vždy alespoň kořenový modul, který umožňuje bootstrapping a obvykle obsahuje mnoho dalších modulů. [2, 7]

Aplikace vytvořené v tomto frameworku jsou tvořeny z komponent. Komponenty jsou nejdůležitější stavební bloky uživatelského rozhraní (UI) v aplikacích Angularu. Každá aplikace má alespoň kořenovou komponentu. Komponenty mohou využívat služby, které poskytují specifické funkce, ale které přímo nemusí souviset se zobrazením. Poskytovatelé služeb mohou být přidávány jako závislosti, čímž se stává kód modulární, opakovaně použitelný i efektivní. Komponenty jsou podmnožinou direktiv. Na rozdíl od direktiv, má komponenta vždy HTML šablonu a pouze jedna komponenta může vytvořit instanci na jeden element v šabloně. Každá komponenta musí být součástí takzvaného NgModulu, aby mohla být použita jinou komponentou nebo aplikací. Komponenta je tedy jednoduchá třída s dekorátorem „@Component“, kde tento dekorátor obsahuje několik metadat, které sdělují jednotlivé informace o komponentě. Můžeme zde nalézt například:

- „template“: šablona, HTML kód pro zobrazení,
- „selector“: pokud se v rodičovském zobrazení vyskytne stejnojmenná HTML značka, Angular vytvoří a vloží instanci dané komponenty mezi tyto HTML tagy,
- „templateUrl“: odkaz na externí soubor, který obsahuje zobrazení pro danou komponentu,
- „styles“: CSS styly, které budou aplikovány v zobrazení dané komponenty,
- „animations“: seznam animací, které budou použity v komponentě,
- „directives“: pole komponentů nebo direktiv, které daná šablona bude využívat. [2, 5]

```
1  @Component({
2    selector: 'animation-cmp',
3    templateUrl: 'animation-cmp.html',
4    animations: [
5      trigger('myTriggerName', [
6        state('on', style({ opacity: 1 })),
7        state('off', style({ opacity: 0 })),
8        transition('on => off', [
9          animate("1s")
10       ])
11     ])
12   ]
13 })
```

Zdrojový kód 1 – Ukázka nastavení metadat pro komponentu [5]

Šablona pro konkrétní komponentu poté kombinuje běžné HTML s Angular direktivy a se značkami pro data-binding, které umožňují modifikovat HTML bez nutnosti znovu načtení stránky. [5, 7]

1.2.1 Moduly

Každá Angular aplikace má kořenový modul, běžně je nazýván „AppModule“. Modul poskytuje mechanismus bootstrap, který spouští aplikaci. Aplikace obvykle obsahuje mnoho dalších funkčních modulů. Stejně jako moduly jazyka JavaScript mohou NgModuly importovat funkce z jiných NgModulů a umožnit, aby jejich vlastní funkce byly exportovány a používány jinými NgModuly. Například pokud byste potřebovali použít službu směrovače ve vaší aplikaci, nainportovali byste si Router NgModul. Takovéto uspořádání kódu do odlišných modulů pomáhá řídit vývoj složitých aplikací. Kromě toho tato technika umožňuje využívat takzvaný lazy-loading, což je načítání modulů na vyžádání, aby se minimalizovalo množství kódu, které je třeba načíst při spuštění aplikace. [8]

1.2.2 Komponenty

Každá Angular aplikace má alespoň jednu komponentu, a to takzvanou kořenovou komponentu, která spojuje hierarchii komponentů s DOM (Document Object Model). Každá komponenta definuje třídu obsahující aplikační data a logiku, je přidružená k šabloně HTML, která definuje pohled, jenž má být zobrazen v cílovém prostředí. Dekorátor „@Component“ identifikuje třídu bezprostředně pod ním jako komponentu a poskytuje šablonu a související metadata specifická pro komponentu. [7, 8]

1.2.3 Šablony, direktivy a data-binding

Šablony kombinují HTML s Angular značkami, které mohou upravit prvky HTML ještě před jejich zobrazením. Direktivy v šablonách poskytují programovou logiku. Značky pro bindování (dvojitě složené závorky, mezi nimiž je název proměnné například: {{username}}) propojují aplikační data a model DOM.

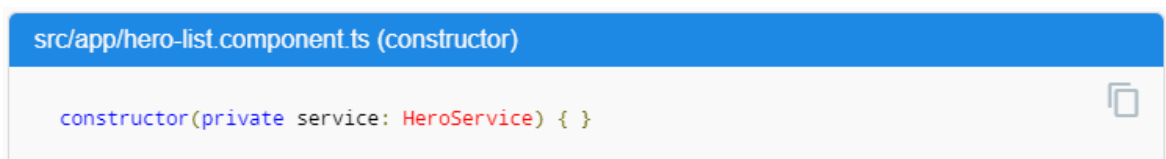
Předtím než je View zobrazeno, Angular vyhodnotí direktivy a vyřeší binding syntaxi v šabloně pro modifikování prvků HTML a modelu DOM. Angular podporuje dvoucestný data-binding, což znamená, že změny v modelu DOM se také mohou odrazit do programových dat. [8]

Šablony mohou také používat takzvané pipes ke zlepšení uživatelského prostředí tím, že transformují hodnoty pro zobrazení. Pomocí nich lze zobrazit hodnoty měn takovým způsobem, že budou odpovídat místnímu nastavení uživatele. Angular poskytuje několik takovým předem definovaných pipes pro běžné transformace, nebo lze nadefinovat vlastní. [8, 9]

1.2.4 Služby a Dependency Injection

U logiky nebo dat, které nejsou přidruženy k určitému View a které chcete sdílet mezi komponentami, lze vytvořit třídu pro službu. Definici této třídy musí bezprostředně předcházet dekorátor „@Injectable“. Tento dekorátor poskytuje metadata, která umožňují, aby byla služba vložena do klientských komponent jako závislost.

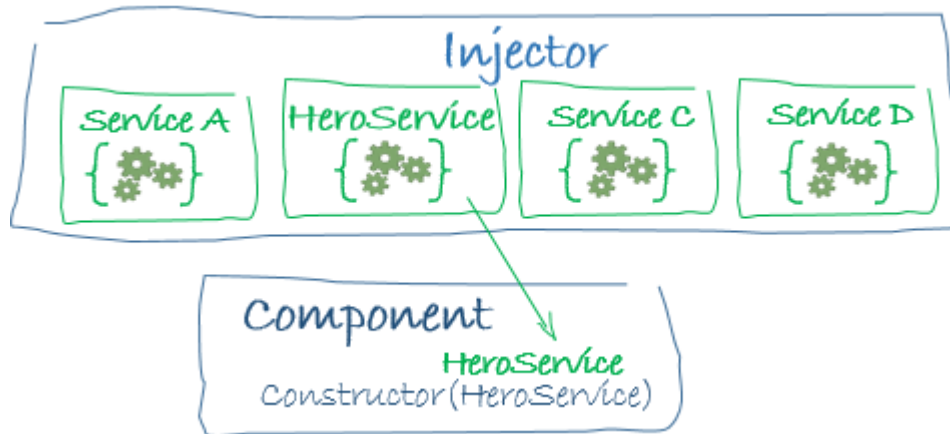
Dependency Injection je používáno všude, aby poskytoval komponenty se službami nebo jinými potřebami. Injektor je hlavní mechanismus Angularu. Injektory se nemusí vytvářet, ty totiž vytváří samotný Angular v průběhu zavádění v celém rozsahu aplikace. Injektor si udržuje zásobník všech závislostí, které již vytvořil a pokud je to možné, tak je znovu použije. Když Angular vytvoří novou instanci komponentní třídy, určí, kterou službu nebo jinou závislost tato komponenta potřebuje, tím že se podívá na typy parametrů v konstruktoru. Například konstruktor komponentní třídy „HeroListComponent“ potřebuje službu „HeroService“. [7, 10]



```
src/app/hero-list.component.ts (constructor)
constructor(private service: HeroService) { }
```

Obrázek 4 – Konstruktor komponentní třídy „HeroListComponent“ [10]

Když Angular zjistí, že komponenta závisí na službě, nejprve zkontroluje, zda injektor již obsahuje nějaké existující instance této služby. Pokud požadovaná instance služby dosud neexistuje, injektor jednu vytvoří pomocí registrovaného provideru a přidá ji k injektoru ještě před tím, než vrátí službu do Angularu. Když byly všechny požadované služby vyřešeny a vráceny, tak Angular může vyvolat konstruktor komponenty s těmito službami jako argumenty. Proces injektoru služby „HeroService“ vypadá následovně:



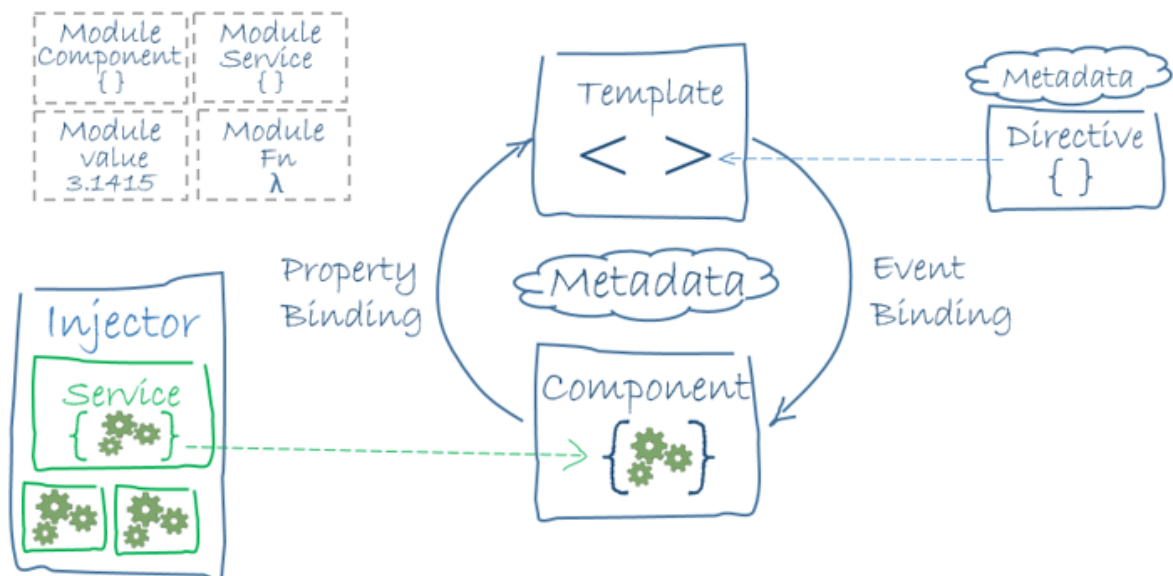
Obrázek 5 – Injektor služby „HeroService“ [10]

1.2.5 Routing

NgModul Router poskytuje službu, která umožňuje definovat navigační cestu mezi různými stavy aplikací a hierarchiemi Views v aplikaci. Router mapuje cesty typu URL na views namísto stránek. Když uživatel provede nějakou akci, například klepne na odkaz, který načte novou stránku do prohlížeče, router zachycuje chování prohlížeče a zobrazuje nebo skrývá hierarchie views.

Pokud router zjistí, že aktuální stav aplikace vyžaduje zvláštní funkcionalitu a modul, který ji definuje, nebyl načten, může router tento modul na vyžádání načíst. Router implementuje adresu URL odkazu podle navigačních pravidel zobrazení a stavu dat aplikace. Je možno přejít na nové view, když uživatel klepne na tlačítko, vybere položku z dropdown boxu nebo v reakci na nějaký jiný podnět z jakéhokoli zdroje. Router zaznamenává aktivitu v historii prohlížeče, takže tlačítka zpět a dopředu fungují bez problému.

K definování navigačních pravidel stačí propojit navigační cesty ke komponentám. Navigační cesty používají syntaxi podobnou adrese URL. Potom je možno aplikovat programovou logiku pro výběr view, které se zobrazí nebo skryje v závislosti na vstupu uživatele a vlastních pravidlech přístupu. [8]



Obrázek 6 – Blokový diagram Angularu [8]

Komponenta a šablona společně definují View. Dekorátor komponentní třídy přidává veškerá metadata včetně ukazatele na přidruženou šablonu. Direktivy a značky pro data-binding v šabloně upravují views na základě programových dat a logiky. Dependency injector poskytuje služby komponentě, jako například router službu, díky níž je umožněno definovat navigaci mezi views. [8]

1.3 TypeScript

TypeScript je programovací jazyk s otevřeným zdrojovým kódem vytvořený a spravovaný firmou Microsoft. Byl vyvinut v roce 2012 jako nástavba jazyka JavaScript, jenž tento jazyk rozšiřuje například o volitelný statický typ, třídy, moduly a další atributy z objektově orientovaného programování. V době psaní této práce se TypeScript nachází ve verzi 2.8. Veškerý kód napsaný v TypeScriptu se následně kompiluje do JavaScriptu, tudíž je každý JavaScriptový kód automaticky validním TypeScriptovým kódem.

TypeScript podporuje hlavičkové soubory, které mohou obsahovat typové informace z již existujících JavaScriptových knihoven. To umožňuje ostatním programům používat hodnoty definované v souborech, jako by byly staticky napsané entity TypeScriptu. Již nyní existují hlavičkové soubory třetích stran pro populární knihovny jako jsou jQuery, MongoDB nebo třeba D3.js a Node.js. [11]

1.3.1 Typová anotace

Jedna z výhod TypeScriptu je určitě jeho typová kontrola. Tento programovací jazyk poskytuje statické typování pomocí anotací, které umožňují kontrolu typu při kompilaci. Anotace je volitelná a lze ji ignorovat, místo ní lze použít dynamické typování. Anotace pro nejzákladnější statické typy jsou number, boolean a string. Pro dynamicky typovanou strukturu lze použít anotace any.

Anotace typů lze exportovat do samostatného souboru deklarácí, což umožní používat již existující JavaScriptové knihovny uvnitř TypeScriptového kódu. [11, 12]

1.3.2 Deklarované soubory

Když dochází ke kompilaci TypeScriptového scriptu, existuje zde možnost generovat deklarovaný soubor, který má příponu .d.ts a funguje jako rozhraní ke komponentám v kompilovaném JavaScriptu. V tomto procesu kompilátor odřízne všechny funkce a těla metod a ponechá pouze signatury exportovaných typů. Výsledný soubor deklarácí lze poté použít k popisu exportovaných virtuálních TypeScript typů JavaScriptových knihoven nebo modulů, pokud je programátor třetích stran konzumuje z TypeScriptu. [12]

2 REALTIME DATABÁZE A JEJICH POROVNÁNÍ S SQL DATABÁZEMI

Realtime databáze jsou databázové systémy, které používají realtime processing ke zpracování úloh, jejichž stav se neustále mění. Databázový systém v reálném čase je databázový systém, který poskytuje veškeré funkce tradičního databázového systému, jako je nezávislost dat a řízení souběžnosti a současně vynucuje omezení v reálném čase. Realtime databáze je databáze, která má časové omezení pro vykonání příkazu či požadavku. Tyto databáze se běžně používají v aplikacích, které vyžadují velmi rychlý přístup k datům, ale definice rychlého přístupu není kvantifikována, pro některé aplikace to jsou milisekundu a pro jiné zase minuty. [13]

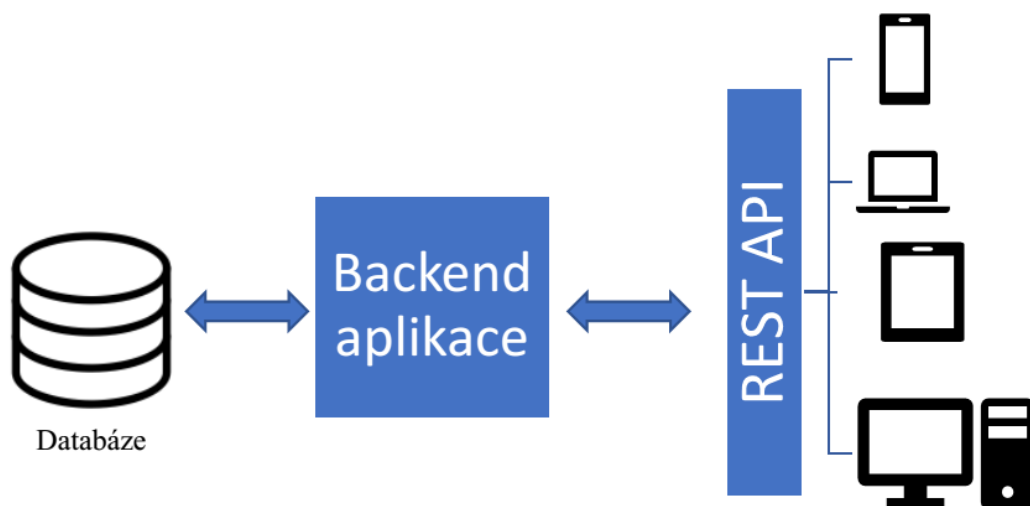
Tradiční databázové systémy se od realtime databázových systémech liší v mnoha aspektech. Mezi nejdůležitější patří, že každá z nich má odlišné cíle výkonu, kritéria správnosti a předpoklady o aplikacích. Realtime databáze se poté hodnotí na základě několika kritérií, například jak často transakce přesáhne termín deadline, podle průměrného zpoždění, nákladů vzniklých při transakcích, které překročili svůj deadline, externí konzistence dat a časové konzistence dat.

Aby se využil největší potenciál těchto databází, měla by být realtime databáze umístěna na cloudovém serveru a měla by umožňovat ukládání a synchronizaci dat mezi uživateli v reálném čase, což znamená, že po změně jakýchkoliv dat v databázi, by měli být všichni uživatelé, kteří mají v danou chvíli v aplikaci zobrazena daná data, informováni o změnách v těchto datech. [14]

Architektura komunikace databází nebo dokonce i použité technologie nejsou nikterak striktně stanovené. Naopak některá jak už z komerčních nebo tak i z open-source řešení se od sebe velmi technologicky odlišují. Ve většině případů jde o kombinaci tří technologií, a to použití NoSQL realtime databáze, serverové aplikace a websocketového připojení s klientskou aplikací. NoSQL databáze má za úkol uchovávat uložená data v cloudovém úložišti. Úkol serverové aplikace spočívá v obsluhování příchozích požadavků, komunikací s databází a zajišťování komunikace mezi serverovou aplikací a klientskými aplikacemi uživatelů. Klientská aplikace může být například desktopová aplikace, webová backendová či frontendová aplikace, aplikace pro chytrá nositelná zařízení nebo dokonce i drobná IoT zařízení. V této práci se bude klientskou aplikací rozumět mobilní aplikace pro smartphony a tablety. Jedinými omezeními pro připojení k datům a jejich následnému

synchronizování je technologická schopnost zařízení navázat websocketové připojení a mít dostupné internetové připojení. Kromě těchto omezení nehraje platforma, operační systém nebo povaha zařízení žádnou roli. [13, 14]

V případě mobilních zařízení je v dnešní době standardně používána klient-server architektura REST, která je stále nejrozšířenějším způsobem klient-server komunikace, avšak by v jistých případech mohla být zcela zastoupena realtime databázemi, které umožňují synchronizaci dat mezi jednotlivými mobilními zařízeními. Pokud tedy serverová část mobilní aplikace slouží pouze pro zprostředkování komunikace s databází, pak lze tuto serverovou aplikaci nahradit realtime databází. Tím by se odstranily náklady na vývoj a testování serverové aplikace. Při implementaci náročnějších aplikací, kde funkce serverové aplikace již není jen zprostředkování komunikace s databází, by absence serverové aplikace nadělo problémy, jelikož klientská aplikace bez serverové aplikace by musela vykonávat veškerou výpočetní logiku, včetně výpočetně náročných operací. Pokud by se jednalo o zařízení, které hostuje klientskou aplikaci, mobilní telefon, nebo i dokonce IoT zařízení, které má velmi omezený výpočetní výkon, výdrž baterie i v některých případech nestálé internetové připojení, v tom případě by absence serverové aplikace nebyla vhodným řešením. Další problémy by nastaly z hlediska bezpečnosti, jelikož například validace zapisovaných dat do databáze nebo zachování konzistence databáze a další zodpovědnosti by neměly být prováděny na straně klientských aplikací. [14]



Obrázek 7 – Topologie REST architektury [14]

Kromě tedy jednoduchých aplikací je absence serverové aplikace nepoužitelným a nepraktickým řešením. Je tedy vhodné použít řešení, které zahrnuje i serverovou aplikaci.

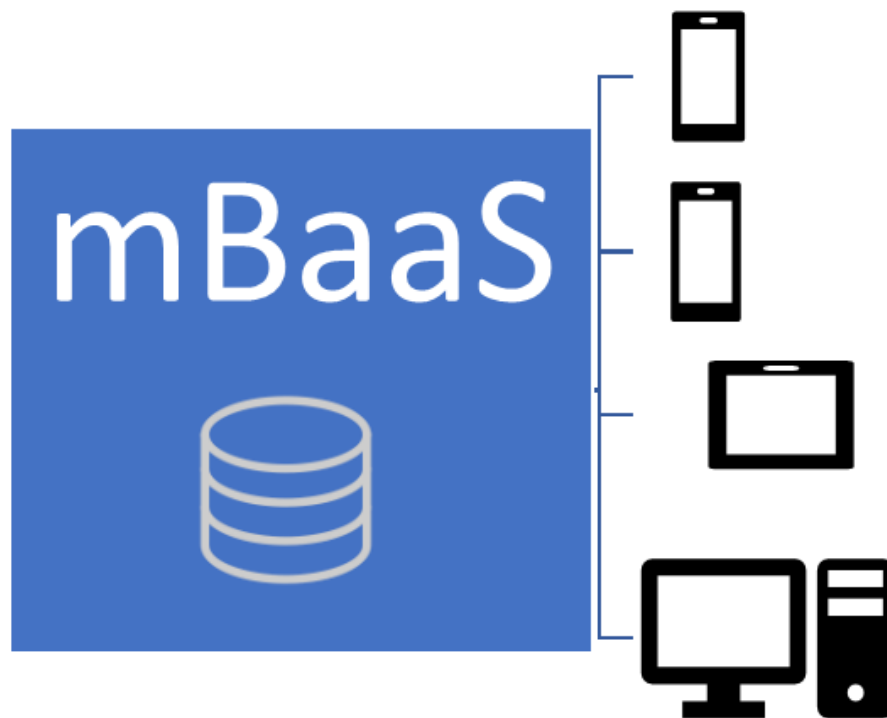
Tímto ale odpadá původní záměr: úspora nákladů a urychlení vývoje díky tomu, že by šla obejít implementace serverové aplikace, ale komunikace mezi realtime databází a klientskou aplikací může být nadále prováděna přímo bez nutnosti zprostředkování komunikace skrz REST API. [13]

Jednou z dalších možností řešení nabízejí mnoho komerčních i nekomerčních poskytovatelů realtime databází. Jedná se o celistvé platformy označované jako BaaS (respektive mBaaS), kde realtime databáze je ve většině případech pouze jen jedna z mnoha nabízených funkcí. Mezi další nabízené funkce patří například možnost pomocí předpřipravených funkcí konfigurovat nebo i manuálně jednoduše naprogramovat serverovou logiku. Pomocí BaaS a všech vymožeností, která jednotlivá řešení poskytují, lze tedy zcela nahradit serverové backend aplikace. [14]

2.1 BaaS (Backend as a Service)

Jedná se o poskytování hotového řešení pro vytvoření, používání a správy aplikačního cloudového backendu pro mobilní a webové aplikace jako služby. Existuje také termín mBaaS neboli mobile Backend as a Service a ten se odlišuje od BaaS, tím, že slouží primárně pro mobilní aplikace, na rozdíl od BaaS, který slouží primárně pro webové aplikace. Jde ale spíše jen o marketingový tah, než aby se tyto dva typy výrazně lišily. Jelikož spousta poskytovatelů BaaS nabízí jednotnou službu umožňující propojení souvisejících aplikací na webu, tak i na mobilních zařízeních. [14, 15]

Tyto služby v sobě zahrnují mnoho funkcionalit, jako jsou například databáze, správa uživatelů, datové úložiště, odesílání push notifikací a integraci se sociálními sítěmi. Všechny tyto služby poskytuje většina komerčně dostupných mBaaS řešení. Jednotlivé služby se ale od sebe odlišují v množství a povaze dalších služeb, přičemž jde o velké odlišnosti, jelikož se na trhu objevují začínající start-upové společnosti na jedné straně a oproti nim jsou tady společnosti jako je Google nebo Microsoft na straně druhé. Tyto korporátní společnosti mají oproti všem ostatním znatelnou konkurenční výhodu na trhu. Například společnost Google dovedla svoji platformu Firebase do takových rozměrů a kvalit, čemuž menší společnosti dokáží jen těžko konkurovat. [15]



Obrázek 8 – Topologie mBaaS architektury [14]

2.2 Technologie využívané v realtime databázových systémech

Konkrétní architektura a použité technologie nejsou v realtime databázových systémech striktně stanovené a jednotlivá nabízená řešení se mohou technologicky odlišovat. Ve většině případů se však jedná o kombinaci těchto tří technologií: NoSQL databáze, serverová aplikace a websocketové připojení s klientskou aplikací. [14]

2.2.1 NoSQL databáze

Se zvyšujícími nároky na schopnost databází efektivně ukládat a zpracovávat velké množství dat se tradiční relační databáze začaly potýkat s novými výzvami. Především pokud jde o aplikaci velkého rozsahu, která obsluhuje mnoho uživatelů, kteří operují se stejnými daty ve stejný čas. V tomto případě se začaly stávat tradiční relační databáze neadekvátními až nepoužitelnými pro tyto případy. Toto vedlo k rozvoji tzv. NoSQL databází. Jedná se o druh databáze, který nevyužívá jazyka SQL pro manipulaci s daty, a také využívá jiný mechanismus ukládání dat než relační způsob. [14, 16]

NoSQL databáze nejsou v IT světě žádnou novinkou a pro komerční využití se využívají již několik let, ale v porovnání s tradičními relačními databázemi, které mají svůj standard na poli databází již několik desítek let, se NoSQL databáze stále mohou označovat za novou

technologii, což je jedním z problémů těchto databází. Relační databáze mají oproti NoSQL databází své pevné místo a mnoho dalších připojených technologií, reportovacích a modelovacích nástrojů a spoustu dalšího softwaru, který je pevně spojen s těmito databázemi. Proto převod z relačních databází na novou technologii, by pro velké, průmyslové či bankovní systémy byl velkou finanční investicí a také by došlo k vystavení se obrovskému riziku z nevydařené transformace. Na druhou stranu uplatnění NoSQL databází se uplatní u nově vznikajících projektů, jelikož důvodů pro jejich využití je mnoho. Hlavní výhody NoSQL databází se začínají objevovat v oblastech, kde relační databáze začaly být pro novou dobu a nové technologie nevyhovující. [15]

„NoSQL databáze jsou stavěny tak, aby byly schopné rychle a efektivně číst i zapisovat data, ačkoliv databáze již velké množství dat obsahuje.“ [14]

Jedním z příkladů NoSQL databází je MongoDB, která je technologicky na pomezí relačních a nerelačních databází. Během selekce dat tato databáze dosahuje až desetinásobek rychlosti relační databáze MySQL, a to při velikosti databáze 50 GB.

Další výhody a také jisté důvody pro používat NoSQL databáze je snazší možnost škálování, dostupnost databáze a snížení nákladů na provoz a správu. Na trhu již existuje spousta variant řešení NoSQL databází, a to jak i z komerčního tak i z nekomerčního řešení, proto je velmi snadné vybrat nejvhodnější variantu pro konkrétní projekt podle dostupných parametrů a poskytovaných funkcí. [14, 16]

2.2.2 Serverová aplikace

Serverová aplikace slouží pro navázání spojení s klientskou aplikací. Touto aplikací může být například mobilní aplikace anebo také aplikace pro chytré televizory, hodinky, a dokonce i automobily. Dalším úkolem serverové aplikace je zajištění komunikace s databází, selekce, nebo editace dat na základě příchozích požadavků z klientských aplikací.

Serverová aplikace má také na starost bezpečnost dat. Od klientské aplikace, při navázání spojení, může požadovat autorizaci aplikace nebo autentizaci konkrétního uživatele, a to z důvodu ověření práv přístupu k datům. Tato kontrola bývá prováděna průběžně s každým dalším požadavkem. Serverová aplikace také kontroluje, zda při dalším požadavku nedochází ke kolizi se stanovenými restrikcemi přístupu, to znamená, zdali uživatel požaduje editaci dat, ale ta je povolena jen uživatelům s přiděleným vyšším stupněm oprávnění, jako například administrátorovi nebo dokonce jen správci databáze. Dále je

kontrolována validita příchozích dat u požadavků, zda jsou obsaženy všechny povinné parametry, nebo zda dodržují daný datový typ a rozsah délky a také se kontroluje, zda neobsahují škodlivý kód či nepovolené znaky a také jestli nedojde k porušení databázové integrity při editačních a destruktivních požadavcích. [14]

Serverová aplikace také provádí výpočetně náročné operace, nebo procedury, jelikož serverová část disponuje větším výpočetním výkonem než klientské zařízení. Provádí se operace, které mají totožný průběh pro všechny komunikující zařízení – klientské aplikace. Tímto lze zamezit zbytečné implementaci a testování dané funkcionality samostatně v každé aplikaci a místo toho toto implementovat pouze na jednom místě, a to na straně serveru.

Pokud se jedná již o hotové mBaaS řešení je serverová aplikace ve většině případů již hotová a je součástí nabízeného řešení společně s databází, včetně možnosti přístupu ke konfigurační konzoli, a to přes webové rozhraní. Pomocí konzole lze zprostředkovaně přistupovat k databázi, vytvářet skripty výpočetně náročných operací pro backend a také konfigurovat ostatní nabízené služby dané mBaaS.

Existuje mnoho programovacích jazyků a technologií díky nimž lze implementovat backendovou službu pro mobilní aplikace. Pokud podporují komunikaci prostřednictvím websocketového připojení nebo umí používat standardní komunikaci rozhraní REST, pak nezáleží na použité platformě. Implementace backendové služby lze provést díky těmto programovacím jazykům: Ruby, Python, Pearl, Java, JavaScript a mnoho dalších. V této práci je pro demonstraci open-source řešení použit softwarový systém Node.js, ve kterém se webový server píše v programovacím jazyce JavaScript. [14]

Node.js

Node.js je softwarový systém speciálně navržený pro psaní internetových aplikací, a to především webových serverů. Programy vyvíjené pod Node.js jsou psány v jazyce JavaScript. Většina z nich hodně využívá model událostí a asynchronní I/O operace pro maximalizaci výkonu a minimalizaci režie procesoru. Node.js je složen z několika standardních knihoven a také z V8 JavaScript engine od společnosti Google. Node.js byl vytvořen v roce 2009. Tento systém se od většiny JavaScriptových programů liší v tom, že není spuštěn v internetovém prohlížeči, ale běží na straně serveru. Node.js implementuje některé části ze specifikace CommonJS a v roce 2012 byl oceněn webem InfoWorld jako nejlepší technologie roku. [17, 18]

2.2.3 Websocketové připojení

V současné době pro komunikaci klient-server, v případě vývoje mobilních aplikací, je dominantou architektura REST, která má několik problematických vlastností pro použití na mobilních zařízeních. Jedná se o bezstavovost komunikace. Každý požadavek na server musí na svém počátku obsahovat HTTP hlavičku, která může obsahovat velké množství informací, například o klientském zařízení a prostředí z kterého se požadavek odesílá, nebo také i informace o požadavku a jeho očekávané odpovědi. Tímto se tento druh komunikace pro realtime komunikaci stává problematický, jelikož výměna těchto informací je pomalá, datově, energeticky i výpočetně náročná pro mobilní zařízení, a hlavně je jednosměrná. [14]

```

▼ General
  Request URL: https://www.google.com/
  Request Method: GET
  Status Code: 200
  Remote Address: 216.58.201.68:443
  Referrer Policy: no-referrer-when-downgrade
► Response Headers (17)
▼ Request Headers
  :authority: www.google.com
  :method: GET
  :path: /
  :scheme: https
  accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
  accept-encoding: gzip, deflate, br
  accept-language: cs-CZ,cs;q=0.8
  cache-control: no-cache
  cookie: CONSENT=WP.2693ae; SID=wAV50yIPexoa880VtyDCKZDYj7eVAE7P-ORb1s0M1x7QGL50gzr1RZsBe_t_XqEgezXZow.; HSID=AnsH7ZknbQ0E
  LPqUs; SSID=Ax0INnvzRHEI2YFv; APISID=g1weGFzHTM9c-hq/_Aat-9fJvacS9K0Y1R; SAPIID=PpAebKhe3ZmmIMrX/AI14UZ03Iqd9sZqo0; OG
  PC=845686784-21;; OGP=-845686784;; S=billing-ui-v3=v37N8vWceRe-4D5YJbOM9jFIaIoB0kbX:billing-ui-v3-efe=v37N8vWceRe-4D5YJb
  OM9jFIaIoB0kbX; NID=129=Uv6dYGHcY0Ac1dLkNnU2xoqeR2pszg5fC15WqGrgDSjUZuWH3pEF92fzXc6pVsEUcz460SYfYGlBxS9kebiA1aysRapN-o
  g7CVZ8W8V0cEwqEdZUU6L0cmiI92tDD1fYqMH0AHGNFai-oE1TofwSDm8cJCTAppG6QjijWT5G8W7yp1dIfTm_I1g9KmaGgDgRew6k1WzNYY3kWeqQRyqyG0
  rmJLroZ7Vin1i2zM8n3-GgPprbjv-W54sGjCbA; DV=k1k_VkAIHs9IUPBfaeE_xvNkZJV0MRbW5xWlyBWS90AEAAFD-SYvu5vSIR2MBAIB6m10JdwpE1kAA
  A; 1P_JAR=2018-04-30-16; SIDCC=AEfoLeYRDoTmq5iJ4iOg13Xdy6r8qjJrTrxHiEfXSjF7hFuynz9RYYdA76Z-Eb3us2-oL9MiiA
  dnt: 1
  pragma: no-cache
  upgrade-insecure-requests: 1
  user-agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.105 Safari/537.3
  6 Viv/1.92.917.43

```

Obrázek 9 – Ukázka HTTP hlavičky ze serveru google.com

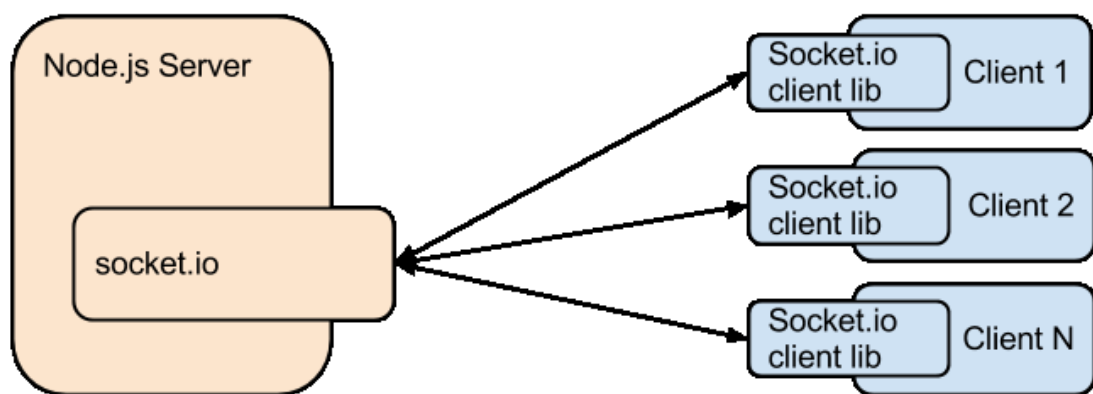
Pro získání nových dat ze serveru musí klientská aplikace neustále odesílat nové požadavky a zpracovávat příchozí odpovědi. Standardní http komunikaci by šlo použít v případech, kdy není nutné, aby klientská aplikace měla vždy dostupná nejnovější data, ale postačí, když je stažení dat provedeno pouze po spuštění aplikace, nebo v nějakých delších pravidelných intervalech či pouze na základě vykonané akce uživatelem v aplikaci, která by vedla k nutnosti aktualizovat data.

Pomocí websocketového připojení lze nároky internetové komunikace do značné míry redukovat, jelikož umožňuje udržovat obousměrné připojení se serverem. Při každém

požadavku již není nutné odesílat obsáhlou http hlavičku, ale pouze data prostřednictvím již navázaného spojení. [14]

Socket.io

Socket.io je JavaScriptová knihovna pro realtime webové aplikace. Knihovna poskytuje realtime, obousměrnou komunikaci mezi klientskou aplikací a serverem. Skládá se ze dvou částí: klientské knihovny, která běží v prohlížečích na klientských zařízeních a serverovou částí knihovny pro Node.js. Obě části mají velmi podobné API. Socket.io je stejně jako Node.js řízené událostmi. Knihovna primárně používá WebSocketový protokol. [19, 20]



Obrázek 10 – Ukázka navázaného spojení se serverem přes socket.io s N klienty [20]

2.3 Komerční realtime databáze

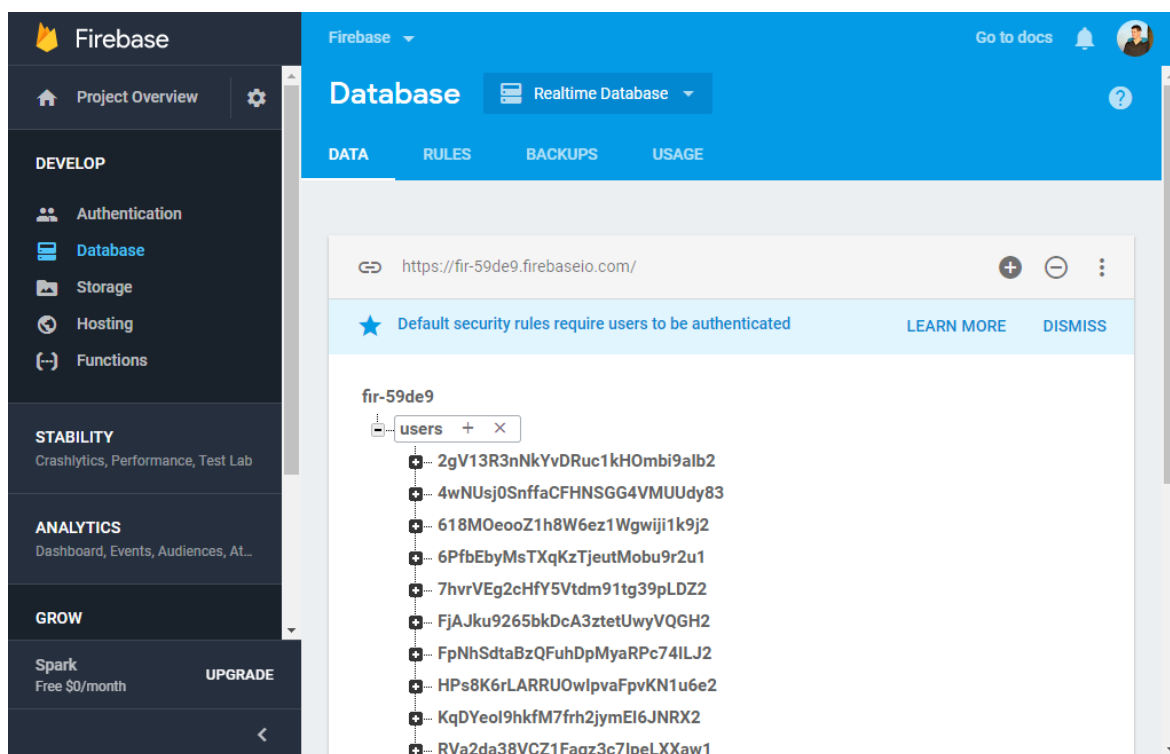
Na trhu realtime databázi existuje spousta z nich, které patří do skupiny komerčních databází, mezi sebou se liší hlavně počtem a kvalitou nabízených služeb, ale také i cenou. Některé z komerčních databází patří do obchodní modelu Freemium, to znamená, že své základní funkce nabízejí zcela zdarma, ale rozšířené možnosti, například větší kapacitu databáze, jsou již zpoplatněny. Do takovéto skupiny patří i Firebase. [21, 27]

2.3.1 Firebase

Jedná se o jednu z nejvyužívanějších mobilních ale i webových BaaS řešení na celém trhu. Firebase vznikl v roce 2011 a byl vyvinut stejnojmennou firmou (Firebase, Inc.). Původně byl tento projekt pouhá realtime služba, která synchronizovala data pro chatovací místnost napříč platformami a fyzickými zařízeními. Následně byl tento systém přetvořen tak, aby byl schopen v reálném čase synchronizovat i jiné datové struktury než jen zprávy pro chat. V roce 2014 došlo k akvizici společnosti a od tohoto roku je Firebase ve vlastnictví

společnosti Google, který tento projekt propojil s množstvím dalších IT služeb, kterými již v té době společnost Google disponovala, a tak z Firebase udělala svoji hlavní mobilní backend platformu. Přestože je Firebase primárně určena mobilním vývojářům, je rovněž velmi oblíbená i u webových vývojářů, jelikož lze její služby využívat ve frontendových ale i v backendových webových aplikacích. [14, 15, 21]

Firestore instance je vytvořena na serveru vlastněného společností Google. Pro nastavení služeb, správu uživatelů a databáze nebo odeslání push notifikací je dostupná webová administrační konzole, ke které je přístup přímo z webového rozhraní. [14, 15]



Obrázek 11 – Webová konzole pro správu veškerých služeb Firebase

Společnost Google zaručuje, v případě velkého množství připojených uživatelů k databázi, téměř neomezenou škálovatelnost instance. Velikost úložného místa nebo rozsah využití jednotlivých služeb se samozřejmě promítá do ceny za poskytnuté služby. Zde je na výběr ze tří různých tarifů. Základní tarif je zdarma a vzhledem k jeho restrikcím je vhodný pouze pro vývoj menších interních aplikací s méně než 100 aktivními uživateli. Další dva tarify jsou již placené, první z nich nese název Flame, zde je cena fixní a to 25 dolarů měsíčně, tento tarif se hodí již pro aplikace malého až středního rozsahu. Posledním z tarifů je tarif Blaze a je realizován metodou *Pay-As-You-Go* a veškeré služby jsou měsíčně účtovány dle ceníku a jejich rozsahu využití v daném měsíci. [14, 21]

Autentifikace uživatelů

Firestore nabízí službu autentifikaci uživatelů, která slouží jako možnost registrace a přihlášení uživatelů do aplikace, a to pomocí emailu a hesla, nebo také existujících účtů sociální a jiných sítí, jako jsou například Google, Facebook, GitHub, Twitter a další. Tuto službu lze integrovat také do již stávajícího autentizačního procesu na vlastním serveru. Firestore udělí každému uživateli přístupový token, díky němuž uživatel získá přístup do realtime databáze. Tuto nabízenou autentizaci uživatelů lze poté využít pro zavedení svých vlastních bezpečnostních pravidel pro přístup daného uživatele či skupiny uživatelů do databáze.

Každému registrovanému uživateli, který se do aplikace zaregistroval pomocí *Firestore Auth*, se přiřadí unikátní identifikátor, který se poté společně s emailovou adresou a dalšími nepovinnými údaji, jako například uživatelským jménem nebo profilovým obrázkem, uloží na server Firestore k příslušnému projektu. [14, 21]

Realtime databáze

Jedná se o nejhlavnější službu celého Firestore. Umožňuje realtime synchronizaci dat napříč webovými i mobilními aplikacemi. Celá databáze v podstatě představuje jeden rozsáhlý a snadno editovatelný soubor ve formátu JSON. [14]

Firestore nabízí k dispozici zdarma SDK pro mobilní platformy Android, iOS a také pro jazyk JavaScript pro podporu hybridního vývoje pro více platforem. To usnadňuje zajištění spojení aplikace s databází a její používání, dále také umožňuje na mobilních zařízeních zajistit i off-line podporu, tedy pokud je funkce aktivována, pak stažená data z databáze jsou na mobilním zařízení perzistentní a lze k nim přistoupit, i když zařízení nemá dostupné internetové připojení, aniž by bylo nutné manuálně ukládat data do lokálního úložiště. Bez přístupu na internet lze i data zapisovat. Tyto změny se uloží do lokálního úložiště a při získání internetového připojení se změny automaticky zapíší i do cloudové databáze. [15]

Spojení s databází je zajištěno pomocí websocketového kontinuálního připojení. To je vytvořeno pouze na počátku komunikace a není nutné jej navazovat při každém požadavku, tak jak je to v případě komunikace prostřednictvím rozhraní REST. Tímto je dosaženo velkých rychlostí při synchronizaci dat. Změny v cloudové databázi se projevují v řádech milisekund, a proto je tato technologie vhodná i pro použití v multiplayerových hrách. Při kontinuálním připojení je značnou výhodou velká úspora objemu přenášených dat, což je velkým plusem pro zařízení, která mají datové limity na internetovém připojení. [14]

Firestore přistupuje k databázi přímo, což přináší jistá bezpečnostní rizika. Jelikož se nijak nekontroluje integrita databáze, může to mít za následek narušení databázové struktury a tím pádem i výskyt závažných chyb v klientských aplikacích. Dalším problémem je to, že každá klientská aplikace má přístup k celé databázi. Ve Firestore lze nastavit bezpečnostní pravidla, čímž lze ochránit integritu dat a také zde lze nastavit i pravidla, za jakých smí nebo nesmí daný uživatel přistoupit k daným datům v databázi. [14, 21, 22]

Souborové úložiště

Firestore nabízí také souborové úložiště, což lze označit za zástupnou službu FTP serveru. Veškeré nahrané soubory, jimiž mohou být nejrůznější multimediální soubory od obrázků až po videa a spoustu dalších, lze třídit do složek. Tak jako u databáze lze i zde definovat práva přístupu k jednotlivým složkám a souborům, a to pro uživatele nebo skupiny uživatelů. Data jsou při nahrávání šifrována. Je-li na zařízení nainstalováno již zmíněné SDK a dojde-li při nahrávání rozměrného souboru ke ztrátě internetového připojení, pak toto nahrávání není přerušeno, nýbrž pozastaveno. Při opětovném přístupu k internetovému připojení dojde k obnovení tohoto nahrávání. [14, 15]

Cloudové funkce

Jak již bylo zmíněno v kapitole 2.2.2, tak všechny výpočetně náročné operace by se měli provádět na straně serverové, nikoliv na straně klienta. Proto Firestore nabízí relativně nové cloudové funkce, které umožňují programovat výpočetně náročné funkce přímo ve webové cloud konzoli, a to za pomoci upravené syntaxe jazyka JavaScript. Tyto funkce mají přímý přístup do realtime databáze a také do datového úložiště, a tedy mohou tak snadno reagovat například na přidání fotografie do úložiště, a tuto fotografii různě upravovat a měnit, následně například odeslat push notifikace nebo email. Lze nastavit i vlastní funkci, která bude cenzurovat nevhodné a vulgární výrazy ve zprávách, které se budou ukládat do databáze. [14, 21]

Testovací laboratoř

Testování mobilních aplikací je obtížné. Je nutné brát v potaz mnoho hraničních případů, které mobilní zařízení mají. Jsou to například stavy, jako je přerušované nebo slabé či dokonce žádné internetové připojení, nebo také nízký stav baterie, příchozí hovor během provádění jisté funkce v aplikaci, horizontální natočení displeje a mnoho dalších stavových a konfiguračních parametrů a samozřejmě jejich kombinace.

Testování na mobilních zařízeních přináší další úskalí, a to různorodost zařízení obzvláště u platform s operačním systémem Android. Tento operační systém totiž využívají někteří výrobci mobilních zařízení a každý výrobce produkuje několik desítek dalších různých modelů mobilních zařízení, kde se jejich tvary mnohdy liší, mají jiné velikosti obrazovek, liší se ve výkonových parametrech, a dokonce i ve verzích operačního systému. Verzí operačního systému je mnoho, a proto nelze zajistit kompatibilitu aplikace s úplně všemi verzemi operačního systému.

Firestore nabízí možnost provést test aplikace alespoň na několika tisících nejrozličnějších reálných Android zařízeních, a plně automatizovaně simulovat interakci uživatele s aplikací. Výsledkem je podrobný report ke každému zařízení, průběžné snímky obrazovky, video z testování aplikace a výpisy zpráv o případném selhání aplikace. [14, 15]

Monitoring

Jde o podobnou službu jako testovací laboratoř s tím rozdílem, že služba již netestuje aplikaci, ale sbírá data již z publikovaných aplikací. Konkrétně měří výkon, dobu odezvy aplikace při spouštění aplikace, stahování a nahrávání souborů a dalších procedurách. Tyto hodnoty lze poté roztřídit například podle modelu mobilního zařízení, verze operačního systému a díky tomuto nalézt nejvhodnější variantu, jak řešit problémy s výkonem a také s uživatelskou přívětivostí aplikace.

Sbírají se i data o případném selhání aplikace a poté se tato data poskytují vývojářům, jedná se nejčastěji o data typu: na jakém zařízení a s jakou verzí operačního systému a také za jakých podmínek a okolností došlo k pádu aplikace. Data obsahují dokonce i hierarchický seznam metod, které byly volány před pádem aplikace. Nalezneme zde i počet pádů zapříčiněné stejnou chybou a také kolika unikátním uživatelům aplikace se daná chyba objevila. Díky těmto službám je možnost identifikovat chyby v aplikaci a určit tak podle závažnosti priority oprav chyb. [14, 15, 21]

2.3.2 CloudBoost

Platforma CloudBoost obsahuje vesměs podobné funkce a služby jako Firestore. Včetně autentifikace uživatelů, zaslání push notifikací, datové i multimediální úložiště, a dokonce také i off-line přístup k datům na klientském zařízení. CloudBoost se také zařazuje mezi mBaaS a poskytuje ukládání a synchronizaci dat s NoSQL databází. Tato platforma poskytuje klientské knihovny, které umožňují integraci s Androidem, iOS, JavaScriptem,

Javou, Objective-C a Node.js aplikacemi. Poskytována databáze je také přístupná pomocí rozhraní REST API a pomocí bindování pro několik JavaScriptových frameworků jako například Angular, React, Ember.js a Backbone.js. Vývojáři, kteří používají databázi CloudBoost, mohou svá data zabezpečit pomocí bezpečnostních pravidel na straně serveru.

CloudBoost disponuje, na rozdíl od jiných platform, možností fulltextového vyhledávání s mnoha volitelnými parametry, které velmi pomáhají a zrychlují vyhledávání záznamů v databázi. Lze nastavovat jazyk vyhledávání a také lze vyhledávat bez ohledu na diakritiku anebo velikost písmen. Dále je zde možnost negativního vyhledávání, což znamená, že klíčová slova, která byla zadána, se v hledaném textu vyskytovat nemají.

CloudBoost je možné vyzkoušet na 30 dní zadarmo, poté se budete muset rozhodnout, jaký z placených tarifů vyberete. Na výběr je základní tarif „Basic“ za 79 dolarů měsíčně, který je vhodný pro začínající projekty, nabízí možnost uložení 50 000 záznamů, 1 GB úložiště a 100 000 API požadavků, dále je na výběr tarif s názvem „Pro“ za 249 dolarů měsíčně, který je určen již pro středně velké projekty a nabízí možnost uložení 250 000 záznamů, 10 GB úložiště, a jeden milión API požadavků. Dalším tarifem je „Pro+“ za 599 dolarů měsíčně, který je určen pro korporátní společnosti a jejich projekty a nabízí již neomezený počet záznamů, 100 GB úložiště a 10 miliónů API požadavků. Nejdražším tarifem je „Enterprise“, který stojí 2500 dolarů měsíčně a je určen pro velmi obsáhlé projekty a nabízí již neomezené počty všech záznamů, velikosti úložiště a API požadavků. [14, 23]

2.4 Open-source realtime databáze

Pojem open-source databáze označuje databáze s otevřeným zdrojovým kódem, kde daná otevřenost znamená jak technickou dostupnost kódu, tak i zároveň jejich licenci. U open-source realtime databází patří mezi důležité aspekty mimo jiné i rozsah vývojářské komunity. Je zřejmé, že s databází, se kterou nepracuje moc vývojářů, ať už z jakéhokoliv důvodu, bude vývoj zdlouhavý a obtížný. Proto při výběru realtime databáze z open-source řešení je nutné brát ohled i na tento aspekt. [28]

2.4.1 RethinkDB

RethinkDB je volně šiřitelná open-source NoSQL databáze vyvinutá v roce 2009 stejnojmennou společností. V době psaní této práce je vydána stabilní verze číslo 2.3. Tato databáze ukládá JSON dokumenty s dynamickými schématy a je navržena tak, aby usnadňovala posílat zaznamenávané změny v datech do aplikací, a to v reálném čase.

Databáze využívá nového přístupového modelu – namísto dotazování na změny v databázi, mohou vývojáři říci databázi, aby nepřetržitě posílala aktualizované výsledky dotazu do aplikací v reálném čase. Toto dramaticky snižuje čas a úsilí potřebné pro vytváření škálovatelných realtime aplikací. Kromě toho, že je RethinkDB od základů navržena pro realtime aplikace, nabízí flexibilní dotazovací jazyk, intuitivní operace monitorování API. Tato databáze byla implementována v C++ od nuly, a byla vyvinuta v průběhu 5 let týmem databázových odborníků a také s pomocí stovek přispěvatelů z celého světa. [24]

RethinkDB se zásadně liší od platform jako je Firebase, PubNub nebo Pusher, a to třemi důležitými způsoby:

- Všechny tyto vyjmenované platformy jsou cloudové služby a na rozdíl od nich je RethinkDB open-source projektem. RethinkDB spolupracuje s Compose.io a s Amazon AWS a díky těmto společnostem se také může stát cloudovou databází a může tak být využita podle potřeb vývojářů.
- RethinkDB je obecný databázový systém. Díky němuž lze spouštět libovolné dotazy včetně propojení tabulek, poddotazů, geoprostorových dotazů anebo agregace. Na rozdíl tedy od vyjmenovaných platform, které mají mnohem omezenější možnosti dotazování.
- RethinkDB je navržen tak, aby byl přístupný z aplikačního serveru, podobně jako tradiční databáze. To sice vyžaduje více nastavovacího programovacího kódu, ale umožňuje to spoustu flexibility, jelikož se aplikace stává sofistikovanější. [24]

2.4.2 MongoDB

MongoDB je multiplatformní open-sourcový databázový program, klasifikován jako NoSQL. Využívá dokumenty typu JSON. MongoDB byla vyvinuta v roce 2009 stejnojmennou společností a v době psaní této práce je ve verzi 3.6. MongoDB umožňuje využívat data a technologie pro maximalizaci konkurenčních výhod, snížení rizik při kritickém nasazení, dramaticky snížit celkové náklady. [25, 26]

MongoDB je orientován na dokumenty, místo toho, aby byl subjekt rozdělen do množství relačních struktur, tak tato databáze ukládá subject v minimálním množství dokumentů. Například aby se neukládaly informace o názvu knihy a autorovi ve dvou relačních strukturách, může se název i autor spolu s ostatními informacemi o dané knize uložit do jednoho dokumentu s názvem Kniha, tento přístup je více intuitivní a lépe se s ním pracuje. MongoDB podporuje hledání podle pole, rozsahové dotazy a hledání podle regulárních

výrazů. Dotazy mohou vracet specifická pole dokumentů a také obsahovat uživatelsky definované JavaScriptové funkce. Kterékoliv pole v MongoDB dokumentu lze indexovat pomocí primárních a sekundárních indexů, které jsou koncepčně stejné jako ty v relačních databázích. MongoDB poskytuje vysokou dostupnost s množinami replik. Sada replik se skládá ze dvou nebo více kopií dat. Každý člen z množiny replik může kdykoliv zastávat funkci primární nebo sekundární repliky. Sekundární repliky udržují kopii dat primární repliky. Při selhání primární repliky, provede sada replik automatický volební proces, při kterém určí, která sekundární replika by měla být primární. MongoDB může běžet na více serverech a vyvažovat tak zátěž nebo duplikovat data, aby nedošlo k pádu systému, pokud by byl problém v hardwaru. [25]

II. PRAKTICKÁ ČÁST

3 CHATOVACÍ APLIKACE

Pro ukázkou práce s realtime databázemi byly vytvořeny dvě aplikace, z nichž ta první pracuje s realtime databázovým systémem Firebase. Pro ukázkou využívání této technologie byla vytvořena, za pomoci frameworku Ionic, chatovací aplikace. Tato aplikace funguje na velmi podobném principu jako již několik existujících aplikací, jako jsou například Messenger nebo také WhatsApp, akorát nenabízí tolik vymožeností, jako jsou video hovory, posílání dokumentů a spoustu dalších. Vše, co tato aplikace naopak nabízí je popsáno v této kapitole níže. Nejdříve je potřeba se zaměřit na požadavky, a to jak funkční, tak i nefunkční.

3.1 Funkční a nefunkční požadavky

V následujících kapitolách budou stanoveny funkční i nefunkční požadavky vytvořené aplikace.

3.1.1 Funkční požadavky

- Aplikace bude umožňovat vytvoření uživatelského účtu díky registraci. Mezi registrační údaje bude patřit emailová adresa a heslo.
- Díky tomuto vytvořenému uživatelskému účtu bude možné provést přihlášení do aplikace ke svému účtu.
- Aplikace bude umožňovat každému přihlášenému uživateli vytvořit a poslat novou zprávu, a to vybranému uživateli, který má vytvořený uživatelský účet v této aplikaci.
- V aplikaci bude zobrazen seznam uživatelů, spolu s poslední zprávou a datem, se kterými si daný uživatel v minulosti psal.
- Seznam těchto zpráv bude v aplikaci seřazen podle času a to sestupně.
- Nové příchozí zprávy, které daný uživatel ještě nepřečetl, budou v aplikaci zobrazeny tučným písmem.
- Aplikace bude umožňovat otevření chatovacího okna se zprávami s daným vybraným uživatelem. V tomto okně budou zobrazeny všechny zprávy, které si kdy přihlášený uživatel s daným uživatelem poslal, a to vzestupně podle času odeslání.
- Aplikace bude také umožňovat změnu profilového obrázku přihlášeného uživatele.
- Aplikace umožní přihlášenému uživateli se odhlásit.

3.1.2 Nefunkční požadavky

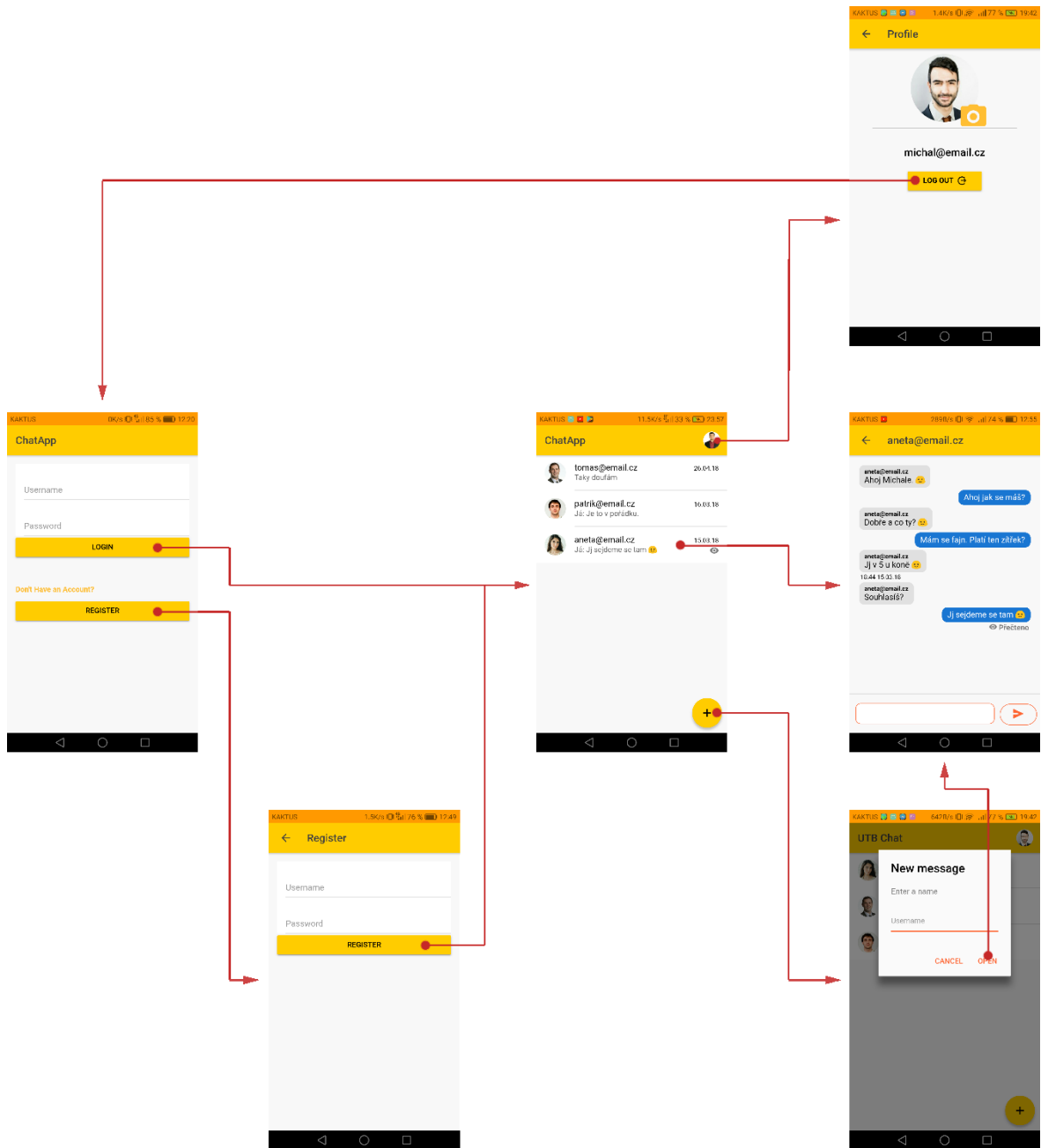
- Registrace a přihlášení uživatelů bude prováděno přes Firebase Authentication.
- Aplikace bude mít oddělenou klientskou a serverovou část.
- Aplikace bude dostupná pro platformu Android a také pro platformu iOS.
- Aplikace musí být udržovatelná a rozšiřitelná.

3.2 Struktura aplikace

Při prvním spuštění bude aplikace zobrazovat přihlašovací obrazovku, kde má uživatel možnost vyplnit dvě položky, a to emailovou adresu a heslo. Zadá-li uživatel špatné údaje aplikace jej upozorní pomocí upozorňovacího okénka na to, že zadal špatné přihlašovací údaje. Jestliže uživatel nemá ještě založený uživatelský účet v této aplikaci, může přejít z přihlašovací obrazovky do obrazovky registrace. Zde uživatel zadává emailovou adresu, heslo a jeho potvrzení. Pokud uživatel zadá emailovou adresu ve špatném formátu, pak jej aplikace opět upozorní na tuto chybu. Heslo má pouze jedno omezení a to takové, že jeho délka musí být minimálně šest znaků.

Po přihlášení anebo registraci uživatele si aplikace uloží do interní paměti emailovou adresu uživatele a po dalším spuštění aplikace se přihlášení provede automaticky, poté se aplikace dostane do své hlavní stránky, kterou je seznam všech zpráv s ostatními uživateli. Zde má uživatel možnost přejít na stránku svého osobního profilu, kde se naskýtá možnost změnění profilového obrázku, nebo odhlášení z aplikace. Při odhlášení se aplikace dostane na přihlašovací obrazovku.

Na hlavní stránce má uživatel možnost také přejít do chatovací stránky, a to pomocí klepnutí na určitou zprávu od určitého uživatele, nebo je zde možnost vytvoření nové zprávy po klepnutí na tlačítko označené znakem plusu, po jeho interakci vyskočí upozorňovací okénko, kde uživatel má možnost zadat emailovou adresu konkrétního uživatele, po jeho potvrzení se také otevře chatovací stránka. V této stránce má uživatel možnost přejít zpět na hlavní stránku nebo odeslat zprávu pomocí vstupního textového pole a tlačítka ve spodní části obrazovky. Zprávy se zobrazují pod hlavičkou stránky a to tak, že jsou seřazeny vzestupně dle času odeslání zprávy. Zprávy od přihlášeného uživatele se zobrazují na modrém pozadí a jsou zarovnané k pravému okraji obrazovky. Zprávy od uživatele, se kterým si přihlášený uživatel píše, se zobrazují zarovnané k levému okraji obrazovky a mají šedé pozadí. Velmi podobný styl lze nalézt i v aplikaci Messenger.



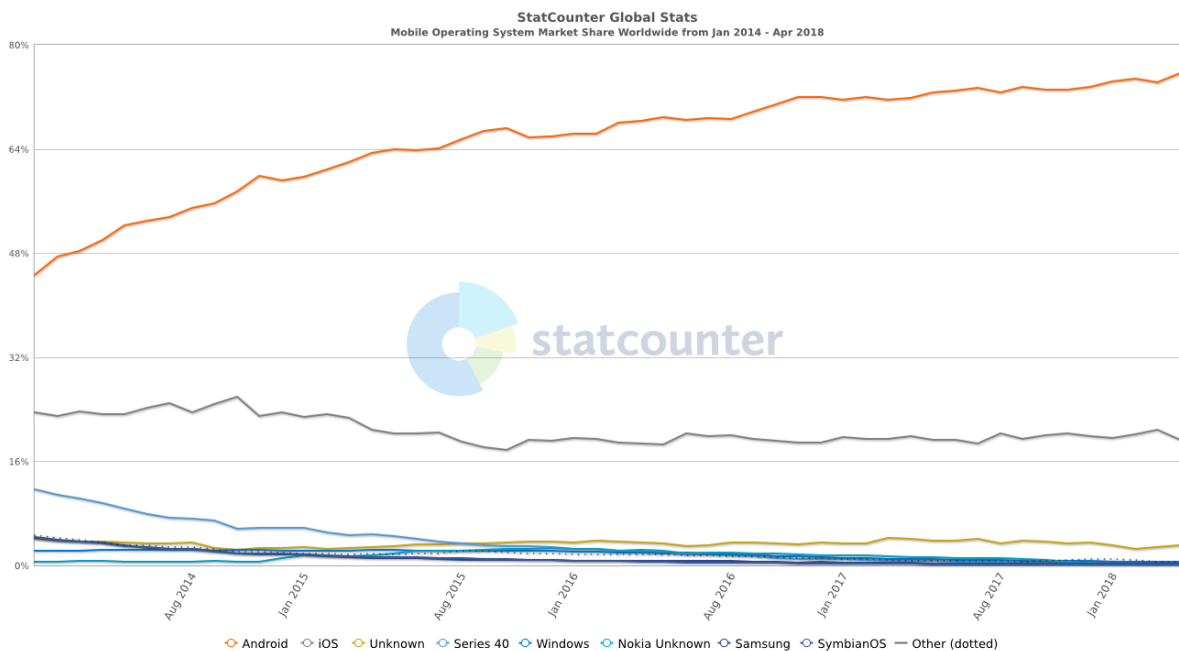
Obrázek 12 – Struktura chatovací aplikace a přechody mezi jednotlivými stránkami

3.3 Implementace

Tato kapitola je zaměřena hlavně na použité technologie, které se využívaly při vývoji aplikace a také zde bude dopodrobna rozebrána funkčnost a struktura dané aplikace.

3.3.1 Zvolené technologie

Jedním z cílů této práce je vytvoření ukázkové implementace mobilní aplikace využívající právě framework Ionic. Tato mobilní aplikace byla tedy vytvořena za pomoci tohoto frameworku, který využívá webových technologií jako jsou HTML, Sass, Angular a další. Nejdůležitější součástí tohoto frameworku je určitě Apache Cordova, který umožňuje multiplatformní vývoj aplikací. Cordova nabízí hybridní vývoj, což ušetří spoustu času a práce oproti nativnímu vývoje aplikace. Jelikož v době psaní této práce, je nejvyužívanějším mobilním operačním systémem Android (75,66%) a na druhém místě je iOS (19,23%), tak tato aplikace bude dostupná pouze pro tyto dva mobilní operační systémy, a to z toho důvodu, že ostatní mobilní operační systémy, jako například Windows Phone, mají na trhu operačních systému již velmi malé zastoupení, z hlediska jejich využití. [6]



Obrázek 13 – Graf představující podíl používaných mobilní operačních systémů na trhu [6]

Chatovací aplikace využívá službu Firebase, která je popsána v teoretické části této práce, a to konkrétně v kapitole 2.3.1. Aby byla možnost využít služeb Firebase, je nutné mít uživatelský účet u společnosti Google. Poté je již velmi snadné založit projekt přímo v konzoli Firebase. K té se dostaneme tak, že přejdeme na webovou stránku www.console.firebase.google.com, zde vybereme možnost „Add project“. Poté budeme tázáni, jak se má budoucí projekt jmenovat, nebo zdali má být přiřazen k již existujícímu projektu. Po vyplnění země, či regionu, kde se nachází organizace nebo firma, pod kterou

by případná aplikace spadala a potvrzení celého formuláře, se nám vytvoří Firebase projekt. Nyní lze spravovat veškeré funkce, které Firebase nabízí, z nichž nejdůležitější je tedy realtime databáze. Připojení k této databázi a práce s ní bude vysvětlena v následující kapitole.

3.3.2 Komunikace s Firebase databází

Aby bylo možné komunikovat s databází ve vytvořeném projektu, je nejprve nutné zkopírovat z Firebase konzole pár důležitých nastavení a ty poté vložit do Ionic projektu. Tato nastavení lze nalézt v přehledu projektu, zde zvolíme možnost „Add Firebase to your web app“. Zobrazí se nám vyskakovací okno, ve kterém lze nalézt veškerá potřebná nastavení pro aplikaci, tak aby mohla navázat spojení s Firebase.

```
<script src="https://www.gstatic.com/firebasejs/4.13.0/firebase.js"></script>
<script>
  // Initialize Firebase
  var config = {
    apiKey: "AIzaSyCaQzrAEUr_AG7QZMQGtr534xtB_R9eN6A",
    authDomain: "test-8943f.firebaseio.com",
    databaseURL: "https://test-8943f.firebaseio.com",
    projectId: "test-8943f",
    storageBucket: "test-8943f.appspot.com",
    messagingSenderId: "83271659075"
  };
  firebase.initializeApp(config);
</script>
```



Obrázek 14 – Ukázka nastavovacího kódu pro Firebase projekt

Pro komunikaci s Firebase databází lze použít již existující různé knihovny. V této práci byla použita knihovna `angularfire2`, která je oficiální knihovnou pro framework Angular, tedy lze použít i v Ionic projektech. Aby bylo možné danou knihovnu použít je nutné ji přiřadit k importům v hlavním `NgModulu` celé aplikace. Nastavení pro připojení k Firebase databázi je nutno implementovat, tak jak je to v následujícím zdrojovém kódu.

```
1 import { AngularFireModule } from 'angularfire2';
2
3 var config = {
4   apiKey: "AIzaSyCaQzrAEUr_AG7QZMQGtr534xtB_R9eN6A",
5   authDomain: "test-8943f.firebaseio.com",
6   databaseURL: "https://test-8943f.firebaseio.com",
7   projectId: "test-8943f",
8   storageBucket: "test-8943f.appspot.com",
```

```
9     messagingSenderId: "83271659075"
10 };
11
12 @NgModule({
13   declarations: [
14     // Vynecháno několik řádků
15   ],
16   imports: [
17     // Vynecháno několik řádků
18     AngularFireModule.initializeApp(config),
19     // Vynecháno několik řádků
20   ],
21   // Vynecháno několik řádků
22 })
23 export class AppModule {}
```

Zdrojový kód 2 – Ukázka nastavení knihovny angularfire2

To nám tedy zajistí propojení aplikace s vytvořeným Firebase projektem. Nyní můžeme využívat služeb, které Firebase nabízí. Jedna z těchto funkcí je i autentizace uživatelů. Tu lze provést díky třídě AngularFireAuth z knihovny angularfire2. Následující zdrojový kód je ukázkou toho, jak lze daného uživatele zaregistrovat.

```
1 import { AngularFireAuth } from 'angularfire2/auth';
2 import { User } from '../models/user';
3
4 @IonicPage()
5 @Component({
6   selector: 'page-register',
7   templateUrl: 'register.html',
8 })
9 export class RegisterPage {
10
11   constructor(private afAuth: AngularFireAuth) { }
12
13   async register(user: User) {
14     try {
15       await this.afAuth.auth.createUserWithEmailAndPassword(
16         user.username, user.password);
17     } catch (e) {
```

```
18     alert("Error! " + e.message);
19   }
20 }
21 }
```

Zdrojový kód 3 – Registrace uživatele

Díky této třídě lze i ověřovat totožnost uživatele, a to například díky emailové adrese a hesla, tak jak to zobrazuje následující zdrojový kód.

```
1  import { AngularFireAuth } from 'angularfire2/auth';
2  import { User } from '../models/user';
3
4  @IonicPage()
5  @Component({
6    selector: 'page-login',
7    templateUrl: 'login.html',
8  })
9  export class LoginPage {
10
11    constructor(private afAuth: AngularFireAuth) { }
12
13    async login(user: User) {
14      try {
15        this.afAuth.auth.signInWithEmailAndPassword(
16          user.username, user.password)
17          .then((data) => {
18            console.log('Success!');
19          }).catch(() => {
20            alert('Error! Wrong username or password!');
21          });
22      } catch (e) {
23        alert("Error! " + e.message);
24      }
25    }
26  }
```

Zdrojový kód 4 – Přihlášení uživatele

Firestore nabízí autentizaci uživatelů pomocí nejrůznějších účtů, v předešlých případech se používala autentizace pomocí emailové adresy a hesla. Další možností autentizace jsou například pomocí telefonního čísla, Google, Facebook, Twitter, a dokonce i GitHub účtu.

Práce s databází

U databáze lze nastavit nejrůznější pravidla pro čtení a zápis do databáze. Lze zde nastavit, kteří konkrétní uživatelé nebo skupiny uživatelů mohou manipulovat s daty v databázi. V této práci jsou nastaveny pravidla pro čtení a zápis dle následujícího obrázku.

```
{  
  "rules": {  
    ".read": "auth != null",  
    ".write": "auth != null"  
  }  
}
```

Obrázek 15 – Pravidla pro Firebase databázi

Číst a zapisovat data do databáze mohou tedy jen autorizovaní uživatelé. Pravidla read a write lze nastavit i na hodnotu true, tudíž by měli přístup k databázi a možnost provádět změny všichni uživatelé, což se nedoporučuje, jelikož to přináší bezpečnostní rizika. Na druhou stranu lze těmto pravidlům nastavit hodnotu false, tudíž nikdo nebude mít přístup k databázi.

Firestore je NoSQL, neobsahuje žádné tabulky, namísto toho do ní lze vkládat stromově strukturovaná data, podobná těm, které lze nalézt ve formátu JSON. Rozdílem mezi JSONem a Firestore databází je absence polí, pokud tedy je nutné pracovat s posloupností záznamů, vloží se tyto záznamy pod unikátními a seřazenými klíči do běžné struktury. Při čtení je poté garantované pořadí.



Obrázek 16 – Příklad struktury uložených dat ve
 Firebase databázi

K ukládání dat a jejich realtime synchronizaci lze opět použít jednu ze tříd knihovny angularfire2. Tentokrát se třída jmenuje AngularFireDatabase. Ukládat data do databáze lze následujícím způsobem.

```

1  import { AngularFireDatabase } from 'angularfire2/database';
2  @IonicPage()
3  @Component({
4    selector: 'page-chat',
5    templateUrl: 'chat.html'
6  })
7  export class ChatPage {
8    constructor(public db: AngularFireDatabase) { }
9    sendMessage() {
10     this.db.list('users/').push({
11       username: 'username',
12       message: 'First message',
13       time: new Date()
14     });
15   }
  
```

16 }

Zdrojový kód 5 – Ukázka vkládání dat do Firebase databáze

Data lze získat z daného uzlu podle následujícího zdrojového kódu. Výhoda tohoto kódu je, že metoda `valueChanges()` se provede vždy když se změní data v daném uzlu.

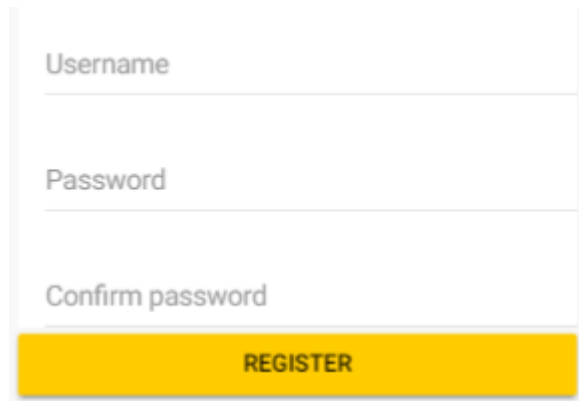
```
1 import { AngularFireDatabase } from 'angularfire2/database';
2 @IonicPage()
3 @Component({
4   selector: 'page-chats-history',
5   templateUrl: 'chats-history.html'
6 })
7 export class ChatsHistoryPage {
8
9   constructor(public db: AngularFireDatabase) {
10     this.db.object('users').valueChanges().subscribe(users => {
11       console.log(users);
12     });
13   }
14 }
```

Zdrojový kód 6 – Ukázka synchronizace realtime dat z databáze Firebase

3.3.3 Základní funkčnost a způsob použití

Registrace

Pokud nemá uživatel již založený účet, má možnost si jej založit pomocí registrace. K zaregistrování mu bude stačit emailová adresa a heslo. Tyto dvě položky uživatel vyplní na registrační stránce. Poté se tyto údaje přepošlou na Firebase autentizaci a pokud se zadaný email dosud nenachází mezi registrovanými uživateli a heslo má více než šest znaků, pak registrace proběhne úspěšně. Jeli tomu tak, pak se emailová adresa spolu s id uživatele, které vrátí Firebase autentizace, zapíše do lokální paměti. Následně se do databázové struktury vloží nový uživatel pod větev „users“ a nastaví se klíč s názvem „username“ na emailovou adresu a také klíč s názvem „avatar“ na defaultní cestu profilového obrázku, který je součástí aplikace. Po úspěšném zaregistrování uživatele, přejde aplikace na hlavní stránku. Naskytne-li nějaký problém ze strany Firebase autentizace, pak se zobrazí upozorňovací okénko se zprávou tohoto problému.

A registration form with three input fields: 'Username', 'Password', and 'Confirm password'. Below the fields is a prominent yellow button labeled 'REGISTER'.

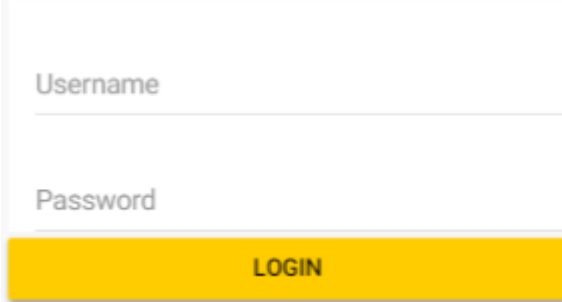
Obrázek 17 – Registrační formulář

```
1  async register(user: User) {
2    if(user.password !== this.confPassword) {
3      this.showAlert("Error!", "Password does not match the confirm
4        password.");
5    }
6    else {
7      try {
8        const result = await
9          this.afAuth.auth.createUserWithEmailAndPassword(
10             user.username, user.password);
11        if (result) {
12          this.storage.set('username', this.user.username);
13          this.storage.set('uid', result.uid);
14          this.db.object(`users/${result.uid}/
15            username`).set(this.user.username);
16          this.db.object(`users/${result.uid}/avatar`).set(
17             "assets/imgs/avatar.png");
18
19          this.navCtrl.setRoot('ChatsHistoryPage', {
20             username: this.user.username
21           });
22         }
23       }
24     }
25     catch (e) {
26       console.error(e);
27       this.showAlert("Error!", e.message);
28     }
29   }
30 }
```

Zdrojový kód 7 – Registrační asynchronní funkce

Přihlášení

Uživatel se přihlásí ke svému uživatelskému účtu opět pomocí svých přihlašovacích údajů, tedy emailové adresy a hesla. Tyto údaje se odešlou na Firebase autentizaci, pokud jsou špatně zadaná, například nebyl nalezen uživatel se zadanou emailovou adresou, pak aplikace otevře upozorňovací okénko, ve kterém uživateli sdělí, že zadal špatnou emailovou adresu nebo heslo. Pokud autentizace proběhne úspěšně, aplikace si opět uloží emailovou adresu a id uživatele do lokální paměti a poté přejde na hlavní stránku tedy seznam všech zpráv.



The image shows a login form with two input fields: 'Username' and 'Password'. Below the fields is a yellow button labeled 'LOGIN'.

Obrázek 18 – Přihlašovací formulář


```
1  async loginUser(user: User) {
2    try {
3      if(user.username == undefined || user.password == undefined) {
4        this.showAlert('Error!', 'Wrong username or password!');
5      }
6      else {
7        const result = await this.afAuth.auth
          .signInWithEmailAndPassword(
            user.username, user.password)
8        .then((data) => {
9          this.storage.set('uid', data.uid);
10         this.storage.set('username', this.user.username);
11         this.navCtrl.setRoot('ChatsHistoryPage', {
12           username: this.user.username
13         });
14       }).catch(()=>{
15         this.showAlert('Error!', 'Wrong username or password!');
16       });
17     }
18   } catch (e) {
19     console.error(e);
20   }
21 }
```

Zdrojový kód 8 – Přihlašovací asynchronní funkce

Hlavní stránka

Hlavní stránka aplikace obsahuje seznam všech zpráv, které kdy byly uskutečněny s jednotlivými uživateli. Každá položka tohoto seznamu obsahuje:

- profilový obrázek daného uživatele,
- poslední zprávu, která byla napsaná mezi těmito uživateli, pokud poslední zprávu poslal přihlášený uživatel, doplní se k této zprávě na její začátek označení „Me: “,
- datum poslední zprávy,
- ikonku oka, která představuje, že poslední zpráva odeslána přihlášeným uživatelem byla přečtena.

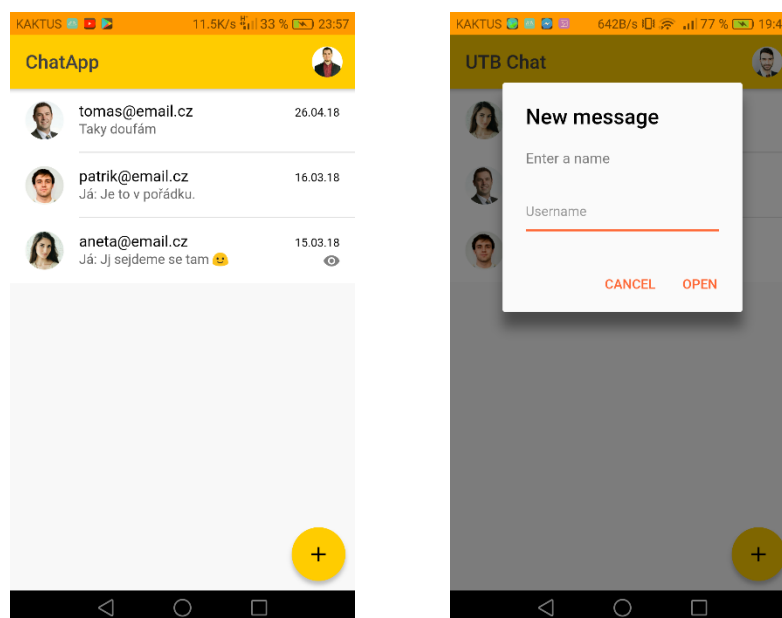
Každá položka seznamu je zároveň posunovací, která po posunutí vlevo odhalí dvě tlačítka, a to pro smazání všech zpráv s konkrétním uživatelem, anebo blokaci daného uživatele. Smazání zpráv se provede pouze na straně přihlášeného uživatele. Pokud uživatel zablokuje

jiného uživatele, tak se všechny zprávy mezi těmito uživateli vymažou, a to na obou stranách a zároveň se daný uživatel zapíše do blokováných uživatelů. Každá položka seznamu je zároveň odkazem a po jeho interakci aplikace zobrazí stránku zpráv s konkrétním uživatelem.

Hlavní stránka obsahuje také tlačítko na vytvoření nové zprávy a profilový obrázek přihlášeného uživatele, po jehož interakci aplikace zobrazí stránku profilu.

Vytvoření nové zprávy

Novou zprávu lze vytvořit pomocí tlačítka označeného plusem, nacházejícího se v dolní části hlavní stránky aplikace. Po klepnutí na toto tlačítko aplikace zobrazí upozorňovací okénko, ve kterém se nachází jedno zadávací vstupní pole, kde uživatel zadá emailovou adresu osoby, které chce poslat zprávu. Po potvrzení dialogu, aplikace zjišťuje, zda je zadaný uživatel registrován v aplikaci, pokud tomu tak není, je dialog uzavřen a následně je zobrazena zpráva o tom, že se nezdařilo nalézt daného uživatele. Pokud uživatel byl nalezen, pak aplikace kontroluje, zda tento uživatel nemá v seznamu blokováných uživatelů právě osobu, která se pokouší tomuto uživateli poslat zprávu. Pokud je přihlášený uživatel v seznamu blokováných uživatelů, pak je zobrazena zpráva o tom, že daný uživatel je blokován. Jestliže aplikace nezjistí ani jeden z těchto případů, pak zobrazí stránku zpráv s daným uživatelem.



Obrázek 19 – Vytvořené nové zprávy

Profilová stránka

Na profilové stránce má uživatel možnost změnit si svůj profilový obrázek nebo se z aplikace odhlásit. Změna profilového obrázku lze díky nativnímu pluginu Camera. Tento plugin je v aplikaci nastaven tak, aby bylo možné vybrat obrázek, který je už na daném zařízení uložen. Po vybrání obrázku je nahrán také do Firebase storage a po úspěšném uploadu je v databázi aktualizována hodnota položky „avatar“ u daného uživatele na adresu nahraného obrázku.

```
1  async changePic() {
2    const options: CameraOptions = {
3      quality: 100,
4      targetWidth: 256,
5      targetHeight: 256,
6      allowEdit: true,
7      correctOrientation: true,
8      destinationType: this.camera.DestinationType.DATA_URL,
9      encodingType: this.camera.EncodingType.JPEG,
10     mediaType: this.camera.MediaType.PICTURE,
11     sourceType: this.camera.PictureSourceType.PHOTOLIBRARY
12   };
13   const result = await this.camera.getPicture(options);
14   this.showLoader = true;
15   const image = `data:image/jpeg;base64,${result}`;
16   const pictures = this.firestore.ref('avatars/'+this.user);
17   pictures.putString(image, 'data_url').then(data => {
18     const ref = this.firestore.ref('avatars/'+this.user);
19     this.profileUrl = ref.getDownloadURL();
20     this.profileUrl.subscribe(data => {
21       this.profileUrlString = data.toString();
22       this.db.object('users/'+this.user+'/avatar').set(data);
23       this.showLoader = false;
24     }, err => {
25       this.profileUrlString = "assets/imgs/avatar.png";
26     });
27   });
28 }
```

Zdrojový kód 9 – Ukázka kódu pro změnu profilového obrázku

Stránka zpráv

Tato stránka zobrazuje veškeré zprávy mezi přihlášeným uživatelem a jiným konkrétním uživatelem. Všechny zprávy přihlášeného uživatele jsou zarovnány k pravému okraji obrazovky, mají bílou barvu textu a modré pozadí, kdežto zprávy od uživatele, se kterým probíhá konverzace jsou zarovnány k pravému okraji, mají černou barvu textu a šedé pozadí. Pokud uživatel klepne na jednotlivou zprávu, zobrazí se pod ní čas a datum odeslání této zprávy. Tento čas se ztratí, když uživatel klepne na stejnou zprávu znovu nebo klepne na jinou zprávu, v tomto případě by se čas a datum zobrazil pod zprávou, na kterou by bylo klepnuto.

V dolní části obrazovky se nachází jedno vstupní pole pro zadání textu samotné zprávy a odesílací tlačítko. Výška vstupního pole je flexibilní, má pouze omezení pro maximální výšku a to 150 pixelů. Pokud poslední zprávou bude zpráva od přihlášeného uživatele a bude přečtena uživatelem, kterému tato zpráva patřila, pak se pod touto zprávou zobrazí ikonka oka a text: „Seen“. Po odeslání zprávy se uloží informace o tom, kdo zprávu poslal, text zprávy a čas kdy byla odeslána, a to jak k uživateli, který tuto zprávu poslal tak i k uživateli, který ji přijal. Následující zdrojový kód zobrazuje, která data si aplikace do databáze ukládá a také na která místa – uzly.



Obrázek 20 – Struktura uložené zprávy ve Firebase databázi

4 APLIKACE NA SDÍLENÍ ZAJÍMAVÝCH BODŮ NA MAPĚ

Jako druhá aplikace pro ukázkou využití open-source realtime databáze byla vytvořena aplikace na sdílení zajímavých bodů na mapě. Tato aplikace byla vytvořena tak jako předešlá aplikace ve frameworku Ionic. Tato aplikace bude umožňovat uživatelům vytvořit svoje vlastní zajímavé body, přidat k nim svůj vlastní komentář a poté tyto body umístit na mapu. Nejprve je nutné zmínit funkční i nefunkční požadavky.

4.1 Funkční a nefunkční požadavky

V následujících kapitolách budou stanoveny funkční i nefunkční požadavky této vytvořené aplikace.

4.1.1 Funkční požadavky

- Aplikace bude umožňovat vytvoření uživatelského účtu díky registraci. Mezi registrační údaje bude patřit emailová adresa, přezdívka a heslo.
- Díky tomuto vytvořenému uživatelskému účtu bude možné provést přihlášení do aplikace ke svému účtu.
- Aplikace bude umožňovat každému přihlášenému uživateli vytvořit nový zajímavý bod a umístit jej kdekoliv na mapě.
- Aplikace nabídne minimálně dvě kategorie, do kterých bude zajímavý bod spadat.
- Aplikace umožní přidat uživatelův komentář ke každému bodu.
- Každá kategorie bude vlastnit svou ikonku pro body na mapě.

4.1.2 Nefunkční požadavky

- Aplikace bude mít oddělenou klientskou a serverovou část.
- Aplikace bude dostupná pro platformu Android a také pro platformu iOS.
- Aplikace musí být udržovatelná a rozšiřitelná.

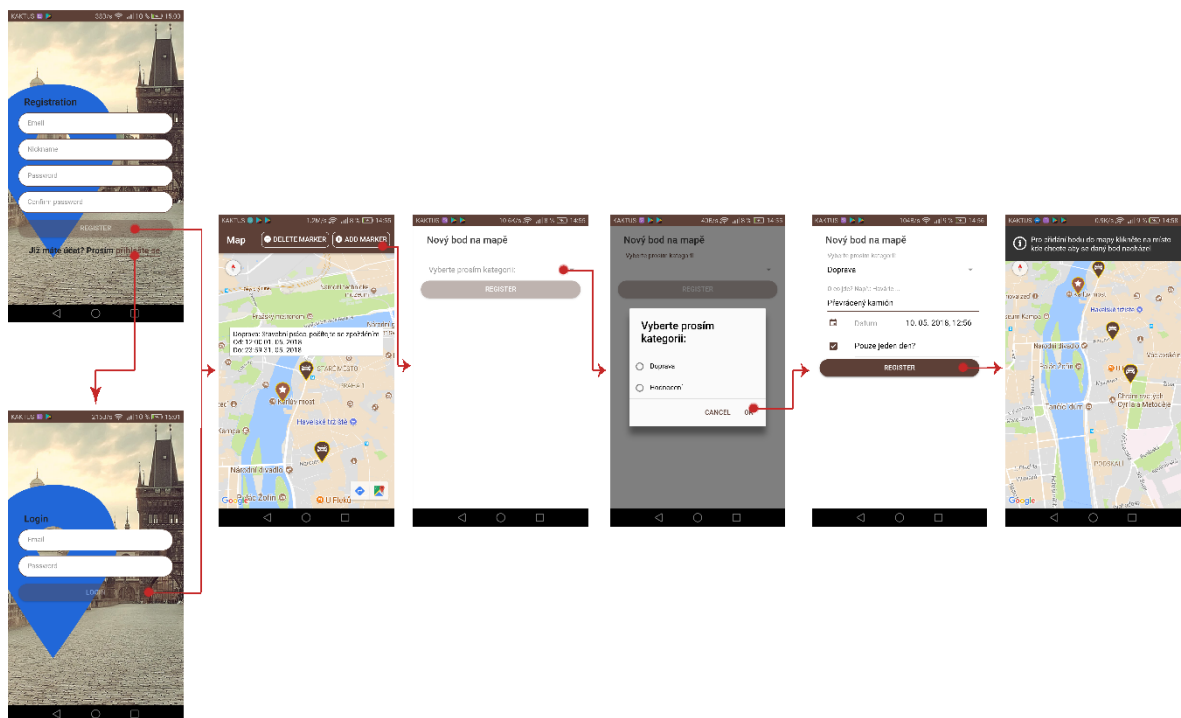
4.2 Struktura aplikace

Při prvním spuštění bude aplikace zobrazovat registrační obrazovku, kde uživatel má možnost vyplnit údaje jako jsou: emailová adresa, přezdívka, hesla a jeho potvrzení. Pokud uživatel zadá emailovou adresu, která je již registrována, tak jej aplikace upozorní, že uživatel s touto emailovou adresou již existuje. Následně aplikace ověřuje, zda heslo a jeho

potvrzení jsou shodné údaje, pokud tomu tak není, aplikace opět upozorní uživatele na nesprávnost těchto údajů.

Z registrační obrazovky se lze pomocí odkazu přesunout na obrazovku přihlašovací, ta obsahuje dvě vstupní pole, a to pro emailovou adresu uživatele a heslo k jeho účtu. Pokud se snaží přihlásit uživatel, který není registrován, aplikace uživatele na tento fakt upozorní. Pokud uživatel zadá správně svoji emailovou adresu, ale zadá špatně své heslo, aplikace opět upozorní uživatele na tuto chybu.

Pokud proběhne registrace nebo přihlášení úspěšně, pak aplikace přejde do hlavního okna, které obsahuje Google mapu a tlačítka pro práci se zajímavými body na mapě. Jedno z nich je pro přidání nového bodu. Po jeho interakci, přejde aplikace na modální stránku, která obsahuje několik interaktivních prvků jako je například selekce kategorie, do které bude nový bod patřit, nebo vstupní pole, které slouží jako komentář k danému bodu. Na této stránce se vyskytuje i tlačítko, čímž se potvrdí veškerá vyplněná pole a aplikace přejde zpět na stránku s mapou. Po tomto přechodu aplikace zobrazí uživateli informační panel v horní části obrazovky, který oznamuje, že nový bod se přidá až po klepnutí na určité místo v mapě. Pokud uživatel klepne na některý ze svých přidanych bodů, které budou označeny zlatým okrajem, zobrazí se tlačítko, které umožní vymazat daný bod.



Obrázek 21 – Struktura aplikace pro sdílení zajímavých bodů na mapě

4.3 Implementace

Tato kapitola je zaměřena hlavně na použité technologie, které se využívaly při vývoji aplikace a také zde bude dopodrobna rozebrána funkčnost a struktura dané aplikace.

4.3.1 Zvolené technologie

Jedním z cílů této práce je vytvořit dvě ukázkové mobilní aplikace naprogramované prostřednictvím vývojového frameworku Ionic, které budou pracovat s realtime databázemi. První aplikace využívající zástupce z komerčních řešení, konkrétně Firebase, je popsána v předešlých kapitolách. Aplikace pro sdílení zajímavých bodů na mapě je tedy druhou aplikací, která tedy využívá realtime databázi spadající do open-source řešení. Pro tuto aplikaci byla vybrána databáze MongoDB, která je již popsána v teoretické části této práce.

Aby bylo možné k databázi přistoupit odkudkoliv, bylo nutné vytvořit serverovou část a tu následně nahrát na cloud, a to nejlépe takový, který nabízí své služby zdarma. Pro tuto aplikaci byl zvolen Heroku, jelikož součástí projektu v něm vytvořeného může být i již zmíněná databáze MongoDB. Jako softwarový systém pro webový server byl vybrán Node.js, jelikož spadá také do řešení, které je open-source, ale také je i jednoduchý na implementaci a vysoce výkonný. Pro realtime WebSocketovou komunikaci bude využívána knihovna Socket.IO, jejíž největší výhodou je oboustranná komunikace mezi klientem a serverem.

Serverová část

Nejprve je nutné mít na daném zařízení, kde se bude vytvářet serverová část, nainstalované Node.js. To lze stáhnout z oficiálních stránek www.nodejs.org. Vytvoření serverové části se provede pomocí spuštění příkazu „npm init“ ve složce, kde chceme serverovou část vytvořit. Po vyplnění potřebných atributů mezi které patří: název serveru, jeho popis, autor, soubor, ve kterém se bude nacházet veškerá serverová logika a další, se vytvoří soubor package.json, který obsahuje všechny tyto údaje a také veškeré dependencies, které budou využity a v případě Heroku po buildu kódu také nainstalovány. Veškerá programová logika serverové části bude uložena v JavaScriptovém souboru, který je nutné ve stejné složce vytvořit. Jeho název se volil při vykonávání příkazu „npm init“. Jelikož je v této práci využívána databáze MongoDB a knihovna Socket.IO, je nutné doplnit dependencies do souboru package.json.

Heroku

Aby bylo možné nahrát serverovou část na cloud Heroku, je nejdříve nutné provést několik úkonů k jeho nastavení a poté nahrání serverové části. Nejprve je nutné se registrovat na www.heroku.com, poté stáhnout a nainstalovat Heroku CLI. Po úspěšné instalaci se přihlásit ke svému účtu pomocí příkazu „heroku login“, kde se vyplní přihlašovací údaje k Heroku účtu. Následně je nutné provést příkaz „heroku create“ ve složce, kde je vytvořena serverová část aplikace. Tímto se vytvoří Heroku projekt se všemi náležitostmi a přiřadí se mu jeho náhodný název. Nahrávání serverové části se provádí pomocí Gitu, tudíž je nutné jej mít také nainstalovaný a s doporučením autora této práce jej mít aktualizovaný na nejnovější dostupnou verzi. Nyní lze serverová část nahrát na Heroku a tím ji tak zpřístupnit, a to pomocí příkazu „git push heroku master“ ve složce, kde se nachází serverová část.

4.3.2 Komunikace s MongoDB

Heroku nabízí také možnost přidat k projektu databázi MongoDB, to lze provést buď přes Heroku Dashboard, který je dostupný na adrese <https://dashboard.heroku.com>, nebo pomocí příkazového řádku. V Dashboardu lze tedy najít doplněk s názvem „mLab MongoDB“ a jednoduše jej přidat do požadovaného projektu. Nyní je nutné zjistit takzvaný „connection URI“, který potřebujeme k tomu, abychom se k dané databázi připojili. To lze provést pomocí příkazu „heroku config:get MONGODB_URI“. Nyní je již možné se připojit k databázi. V souboru serverové logiky (většinou index.js nebo server.js), lze tedy psát:

```
1  var uristring = "mon-  
   godb://<dbuser>:<dbpassword>@ds247619.mlab.com:47619/heroku_85h  
   jtm4j";  
  
2  const mongo = require('mongodb').MongoClient;  
3  
4  mongo.connect(uristring, function(err, client) {  
5    if(err) {  
6      throw err;  
7    }  
8    console.log("MongoDB connected ...");  
9  });
```

Zdrojový kód 10 – Ukázka kódu – připojení k MongoDB

Abychom mohli ukládat data, je nutné vytvořit kolekci v databázi. To lze provést v mLabu, do kterého lze přejít z Heroku Dashboard poklepáním na již přidáný doplněk. Pokud již je k dispozici kolekce, lze do ní vkládat data nebo je i mazat.

```
1  mongo.connect(uristring, function(err, client) {
2    console.log("MongoDB connected ...");
3
4    let users = client.db('heroku_85hjtm4j').collection('users');
5
6    // Vložení nového uživatele do kolekce 'users'
7    users.insert({"username": "Admin", "password": "admin"},
8      function() {
9        console.log("User added.");
10     });
11
12    // Odebrání uživatele
13    users.remove({"username": "Admin", "password": "admin"},
14      true);
15  });
```

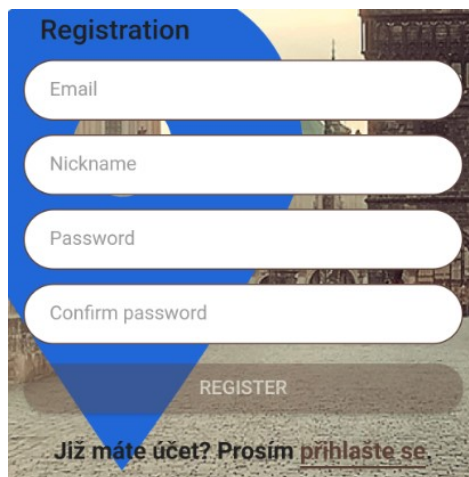
Zdrojový kód 11 – Vkládání a mazání dat z MongoDB kolekce

Aby byla možná realtime klient-server komunikace, využila se v této práci JavaScriptová knihovna Socket.IO. Ta se nachází jak na straně serveru, tak i na straně klienta.

4.3.3 Základní funkčnost a způsob použití

Registrace

Při prvním spuštění se aplikace bude nacházet na registrační stránce, zde uživatel, který zatím nevlastní uživatelský účet, vyplní čtyři povinné údaje jako jsou: emailová adresa, přezdívka, heslo a jeho potvrzení. Pokud uživatel zadá emailovou adresu, která je již registrovaná, pak jej aplikace na tento problém upozorní. Jestliže uživatel zadá heslo a poté jeho potvrzení a tyto dvě položky nebudou shodné, pak aplikace opět uživatele upozorní. Po úspěšném zadání všech údajů aplikace zaregistruje uživatele tím, že si zapíše do databáze jeho emailovou adresu a 512 bitový hash, který byl vypočten z hesla, které uživatel zadal při registraci. Po správném zapsání uživatele do databáze, přejde aplikace na hlavní stránku a uživatel si uloží do lokální paměti.

The image shows a registration form titled "Registration". It features four input fields: "Email", "Nickname", "Password", and "Confirm password". Below these fields is a "REGISTER" button. At the bottom, there is a link that says "Již máte účet? Prosím přihlaste se." The form is set against a background image of a city street.

Obrázek 22 – Registrační formulář

Přihlášení

Pokud uživatel má již uživatelský účet, může přejít z registrační stránky na stránku přihlašovací. Zde zadá pouze svoji emailovou adresu a heslo. Aplikace kontroluje, zda zadaná emailová adresa se nachází v databázi uživatelů, pokud tomu tak není, pak uživatele upozorní na neexistující emailovou adresu. Pokud se emailová adresa v databázi nachází, pak aplikace vypočte 512 bitový hash zadaného hesla a porovná jej s hodnotou hashe, který získala z databáze. Jsou-li hashe rozdílné, pak aplikace uživatele upozorní na nesprávně zadané heslo. Shodují-li se, pak je provedeno přihlášení, aplikace přejde na svoji hlavní stránku a uživatele si uloží do lokální paměti, aby při příštím spuštění aplikace jej mohla přihlásit automaticky.

Hlavní stránka

Na hlavní stránce se nachází Google mapa, do které lze vkládat zajímavé body, dále se tam nacházejí i dva tlačítka, z nichž jedno je pro přidání nového bodu do mapy a druhé je pro mazání bodu, které do aplikace přidal přihlášený uživatel, toto tlačítko je defaultně skryto.

Po klepnutí na tlačítko pro přidání nového bodu se aplikace dostane na stránku, ve které se vyplňují veškeré údaje o bodu, který chce uživatel přidat do mapy. V aplikaci jsou vytvořeny dvě kategorie, ke kterým lze nový bod přiřadit, a to Doprava a Hodnocení. Pokud uživatel vybere kategorii Doprava, pak se zobrazí vstupní pole, do kterého uživatel může zadat, o co by mělo na daném místě jít. Může to být například havárie, kolona a další. Spolu s tímto polem se zobrazí také kolonka pro výběr data, kdy se daná událost stala. Pokud událost trvá déle než jeden den, má uživatel možnost odškrtnout checkbox, který odkryje další kolonku pro výběr data kdy by měla daná událost končit. Pokud uživatel vybere druhou

kategorii, kterou je Hodnocení, pak se zobrazí dvě vstupní textová pole, kde do prvního může uživatel zadat o které místo se jedná a do druhého pole může uživatel přidat svůj vlastní komentář. Po klepnutí na tlačítko „Register“ se aplikace dostane na hlavní stránku s tím, že ve vrchní části obrazovky se zobrazí informační panel, který říká, že bod se do mapy přidá až poté, co uživatel klepne na konkrétní místo v mapě. Tento bod se následně uloží do databáze spolu s emailovou adresou autora tohoto bodu.

5 POROVNÁNÍ IMPLEMENTOVANÝCH REALTIME DATABÁZÍ

V této kapitole budou rozebrány rozdíly mezi použitými realtime databázemi tedy mezi Firebase a MongoDB.

5.1 Integrace s vývojovým frameworkem Ionic

Jak už bylo v této práci zmíněno, tak framework Ionic zastřešuje několik frameworků jako jsou Angular nebo třeba Apache Cordova. Tudíž pokud je vytvořena jistá knihovna pro tyto frameworky, je zřejmé, že půjde využít i ve frameworku Ionic. A to je případem knihovny „angularfire2“, která je sice oficiální knihovnou pro Angular, ale není žádný problém ji použít i v projektech vytvořených pod Ionicem. Toto nese velmi značnou výhodu. Díky připojení této knihovny do Ionic projektu se usnadní veškerá práce nejen s realtime databází, kterou Firebase nabízí. Práce s touto knihovnou je velmi jednoduchá a intuitivní. [21]

Společnost MongoDB nabízí svoji NoSQL databázi jako službu v cloudu pomocí MongoDB Atlas. Tato společnost nabízí také svůj BaaS, který poskytuje podobné služby jako Firebase, s názvem MongoDB Slitch, který je ale v době psaní této práce stále ještě v Beta verzi. Bohužel neexistuje pro tuto databázi žádná podobná knihovna, tedy alespoň ne tak propracována, jako je „angularfire2“. [25]

5.2 Dostupnost vývojových nástrojů

Jak již bylo zmíněno v teoretické části této práce, Firebase je především mBaaS neboli „mobile backend as a service“, což znamená že poskytuje backend hlavně pro mobilní platformu, ale nejen pro ni. Pro vývoj mobilních aplikací na operační systém Android poskytuje Firebase asistenta, který se dá jednoduše připojit k Android Studiu. Firebase nabízí celou řadu klientských knihoven, které jsou dostupné jak pro mobilní platformy, tak i pro web a také existuje rozhraní REST. Dále existuje i spousta neoficiálních knihoven pro další programovací jazyky jako například Python, Ruby a další. [22]

Taktéž MongoDB nabízí SDK pro Android Studio, a to konkrétně MongoDB Stitch SDK. Na rozdíl od Firebase podporuje MongoDB spoustu programovacích jazyků, mezi které patří: C, C++, C#, Erlang, Java, JavaScript, Perl, PHP, Python, Ruby, Scala a spoustu dalších. [25, 29]

5.3 Struktura databáze

Firestore je NoSQL databáze, která neobsahuje žádné tabulky. Namísto toho ukládá stromově strukturovaná data. Jde o velmi podobná data, která jsou ukládána jako JSON, s tím rozdílem, že je zde absence polí. Posloupnost záznamů se tedy vkládá pod unikátními a seřazenými klíči do běžné struktury. Databáze Firestore je dostupná pouze na cloudu, nelze ji tedy stáhnout a pracovat s ní lokálně. Tato databáze nabízí pomocí pravidel pouze limitní funkcionalitu skriptování na straně serveru. [21]

MongoDB je také NoSQL databází, využívající datového formátu BSON. Tento formát je postaven na JSON formátu a je označován za binární JSON. Každý prvek se skládá z názvu pole, typu a hodnoty, kde názvy jsou typu string a typy mohou například být string, integer, double, datum, binární pole, nebo dokonce i JavaScriptový kód. Databázi MongoDB lze stáhnout, a tak s ní lze pracovat na většině operačních systémů. Pokud se databáze využije na straně serveru, pak je zde možnost skriptování v programovacím jazyce JavaScript, díky němu lze nastavit nejrozličnější pravidla pro přístup k databázi a spoustu dalších funkcí. [25, 29]

5.4 Celkové porovnání

Následující tabulka shrnuje, v čem se dané databáze liší, nebo naopak, které vlastnosti mají společné.

	Firestore databáze	MongoDB
Vlastník	Google	MongoDB, Inc
Rok vydání	2012	2009
Licence	Komerční	Open-source
Dostupná pouze jako cloud	Ano	Ne
Jazyk implementace		C++
Podpora předdefinovaných typů dat (float, date)	Ano	Ano

Sekundární indexy	Ano	Ano
Podpora zajištění integrity dat po neatomické manipulaci s daty	Ano	Ne
Podporované programovací jazyky	Java, JavaScript, Objective-C	Actionscript, C, C#, C++, Clojure, ColdFusion, D, Dart, Delphi, Erlang, Go, Groovy, Haskell, Java, JavaScript, Lisp, Lua, MatLab, Perl, PHP, PowerShell, Prolog, Python, R, Ruby, Scala, Smalltalk
Řízení přístupu	Ano, založeno na autentizaci a databázových pravidel	Ano, uživatelé mají přístupová práva a role

Tabulka 2 – Srovnání Firebase databáze s MongoDB [29]

ZÁVĚR

V této práci byly vytvořeny dvě aplikace využívající realtime databáze. Tyto aplikace byly v praktické části této práce dopodrobna popsány jak už z hlediska struktury, tak i z hlediska jejich funkčnosti. První aplikace fungující jako chat, využívá komerční řešení Firebase od společnosti Google. Firebase je Freemium, což znamená, že jeho základní služby jsou poskytovány vývojářům zdarma. To v této práci, na ukázkou, jak s touto platformou pracovat, stačilo. Realtime databáze od Firebase má hned několik výhod, první z nich je již zmíněná knihovna pro framework Angular. Díky této knihovně byla práce s touto platformou velmi snadná, a tak se mohlo investovat více prostředků do vymožeností, které obsahuje vytvořená chatovací aplikace. Díky tomu, že Firebase nabízí i autentizaci uživatelů, u kterých lze rovnou vytvořit pravidla pro přístup k databázi, se také ušetří spousta námahy a prostředků. Jednou z nevýhod Firebase je špatná možnost migrace databáze.

Druhá aplikace, která slouží k vytváření zajímavých bodů na mapě, využívá databázi MongoDB. Aby byla aplikace publikovatelná, muselo být zajištěno, aby tato databáze byla přístupná z kteréhokoliv místa připojeného k Internetu. Je tedy na vývojáři, zda zvolí již některou z existující cloudových služeb jako je například Amazon AWS, který také poskytuje databázi MongoDB. Tato platforma sice také nabízí počáteční služby zdarma, ale při registraci již požaduje údaje z kreditní karty. Proto v této práci bylo zvoleno jiné východisko, a to za pomoci cloudové platformy Heroku. Tato platforma nenabízí mnoho služeb, které by se mohly platformě Firebase rovnat. Výhodou této platformy je, že serverová část aplikace může být vytvořena podle vlastních potřeb a během několika sekund vydána na Heroku server, což bylo pro tuto práci přínosem, jelikož při vytváření aplikace od úplného začátku, serverová část aplikace neustále narůstala o nový programový kód.

SEZNAM POUŽITÉ LITERATURY

- [1] Welcome to Ionic. Ionic Framework [online]. [cit. 2018-04-26]. Dostupné z: <https://ionicframework.com/docs/v1/guide/preface.html>
- [2] KADLEC, Šimon. Mobilní aplikace pro vzdálené ovládání chytrého termostatu se zónovým vytápěním. Praha, 2017. Bakalářská práce. České vysoké učení technické v Praze. Vedoucí práce Ing. Michal Vaňkát.
- [3] What is Ionic?. PrsCreative Blog [online]. 2016 [cit. 2018-04-26]. Dostupné z: <http://blog.prscreative.com/what-is-ionic/>
- [4] Apache Cordova. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-04-26]. Dostupné z: https://en.wikipedia.org/wiki/Apache_Cordova
- [5] Angular [online]. [cit. 2018-04-27]. Dostupné z: <https://angular.io>
- [6] Mobile Operating System Market Share Worldwide. StatCounter [online]. [cit. 2018-05-01]. Dostupné z: <http://gs.statcounter.com/os-market-share/mobile/worldwide/#monthly-201401-201804>
- [7] Angular [online]. [cit. 2018-05-11]. Dostupné z: <https://angular.io>
- [8] Angular architektura. Angular.io [online]. [cit. 2018-05-11]. Dostupné z: <https://angular.io/guide/architecture>
- [9] Angular Pipes. Angular.io [online]. [cit. 2018-05-11]. Dostupné z: <https://angular.io/guide/pipes>
- [10] Introduction to services and dependency injection. Angular.io [online]. [cit. 2018-05-11]. Dostupné z: <https://angular.io/guide/architecture-services>
- [11] Documentation of TypeScript [online]. [cit. 2018-05-11]. Dostupné z: <https://www.typescriptlang.org/docs/home.html>
- [12] TypeScript Language Specification. GitHub [online]. 2016 [cit. 2018-05-11]. Dostupné z: <https://github.com/Microsoft/TypeScript/blob/730f18955dc17068be33691f0fb0e0285ebbf9f5/doc/spec.md>
- [13] Real Time Database Systems. Wiley Encyclopedia of Computer Science and Engineering [online]. 2009, s. 36 [cit. 2018-05-11]. ISBN 9780470050118. Dostupné z: <https://onlinelibrary.wiley.com/doi/book/10.1002/9780470050118>

- [14] HAUBERT, Marek. Realtime synchronizace dat mobilních aplikací [online]. Praha, 2018 [cit. 2018-05-11]. Dostupné z: <https://vskp.vse.cz/id/1341739>. Diplomová práce. Vysoká škola ekonomická v Praze.
- [15] KŘEČEK, Daniel. Metody distribuce zpráv mezi mobilními zařízeními [online]. Brno, 2017 [cit. 2018-05-11]. Dostupné z: https://theses.cz/id/ueuou9/zaverecna_prace.pdf. Bakalářská práce. Mendelova univerzita v Brně.
- [16] Návrh databáze – NoSQL vs SQL. Zdrojak.cz [online]. 2010 [cit. 2018-05-11]. Dostupné z: <https://www.zdrojak.cz/clanky/navrh-databaze-nosql-vs-sql/>
- [17] Node.js Intro. W3 School [online]. [cit. 2018-05-11]. Dostupné z: https://www.w3schools.com/nodejs/nodejs_intro.asp
- [18] Node.js [online]. [cit. 2018-05-11]. Dostupné z: nodejs.org
- [19] Socket.IO [online]. [cit. 2018-05-11]. Dostupné z: <https://socket.io>
- [20] Socket.IO and Realtime Applications with Guillermo Rauch. Softwareengineering-daily [online]. 2016 [cit. 2018-05-11]. Dostupné z: <https://softwareengineering-daily.com/2016/03/03/socket-io-and-realtime-applications-with-guillermo-rauch/>
- [21] Firebase [online]. [cit. 2018-05-11]. Dostupné z: <https://firebase.google.com>
- [22] Firebase: krátké seznámení. Zdrojak.cz [online]. [cit. 2018-05-11]. Dostupné z: <https://www.zdrojak.cz/clanky/firebase-kratke-seznameni/>
- [23] CloudBoost [online]. HackerBay [cit. 2018-05-11]. Dostupné z: <https://www.cloudboost.io>
- [24] RethinkDB Documentation. RethinkDB [online]. [cit. 2018-05-11]. Dostupné z: <https://www.rethinkdb.com/docs/>
- [25] MongoDB Docs. MongoDB [online]. [cit. 2018-05-11]. Dostupné z: <https://docs.mongodb.com>
- [26] MongoDB. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-05-11]. Dostupné z: <https://en.wikipedia.org/wiki/MongoDB>
- [27] Freemium. Management Mania [online]. 2011 [cit. 2018-05-12]. Dostupné z: <https://managementmania.com/cs/freemium>

- [28] ŠTOLC, Robin. Porovnání komerčních a open source nástrojů pro testování softwaru [online]. Praha, 2011 [cit. 2018-05-12]. Dostupné z: <https://vskp.vse.cz/id/1098080>. Diplomová práce. Vysoká škola ekonomická v Praze.
- [29] System Properties Comparison Firebase Realtime Database vs. MongoDB. DB-Engines [online]. [cit. 2018-05-14]. Dostupné z: <https://db-engines.com/en/system/Firebase+Realtime+Database%3BMongoDB>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

API	Application Programming Interface
AWS	Amazon Web Services
BaaS	Backend as a Service
CLI	Command-line interface
CSS	Cascading Style Sheets
DOM	Document Object Model
FFI	Foreign function interface
FTP	File Transfer Protocol
GPS	Global Positioning System
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
mBaaS	Mobile Backend as a Service
MVC	Model View Controller
REST	Representational State Transfer
Sass	Syntactically Awesome Style Sheets
SDK	Software development kit
UI	User Interface
URL	Uniform Resource Locator

SEZNAM OBRÁZKŮ

Obrázek 1 – Využívané technologie frameworkem Ionic	10
Obrázek 2 – Rozdíl v interaktivních prvcích a rozložení na jednotlivých platformách (zleva Android, Windows Phone a iOS)	11
Obrázek 3 – Blokové schéma funkčnosti Apache Cordova	13
Obrázek 4 – Konstruktor komponentní třídy „HeroListComponent“	16
Obrázek 5 – Injektor služby „HeroService“	17
Obrázek 6 – Blokový diagram Angularu	18
Obrázek 7 – Topologie REST architektury	21
Obrázek 8 – Topologie mBaaS architektury	23
Obrázek 9 – Ukázka HTTP hlavičky ze serveru google.com	26
Obrázek 10 – Ukázka navázaného spojení se serverem přes socket.io s N klienty	27
Obrázek 11 – Webová konzole pro správu veškerých služeb Firebase	28
Obrázek 12 – Struktura chatovací aplikace a přechody mezi jednotlivými stránkami	38
Obrázek 13 – Graf představující podíl používaných mobilní operačních systémů na trhu.....	39
Obrázek 14 – Ukázka nastavovacího kódu pro Firebase projekt.....	40
Obrázek 15 – Pravidla pro Firebase databázi	43
Obrázek 16 – Příklad struktury uložených dat ve Firebase databázi	44
Obrázek 17 – Registrační formulář.....	46
Obrázek 18 – Přihlašovací formulář	47
Obrázek 19 – Vytvořené nové zprávy	49
Obrázek 20 – Struktura uložené zprávy ve Firebase databázi	51
Obrázek 21 – Struktura aplikace pro sdílení zajímavých bodů na mapě.....	53
Obrázek 22 – Registrační formulář.....	57

SEZNAM TABULEK

Tabulka 1 – Seznam podporovaných funkcí v daných operačních systémech.....12

Tabulka 2 – Srovnání Firebase databáze s MongoDB.....61

SEZNAM ZDROJOVÝCH KÓDŮ

Zdrojový kód 1 – Ukázka nastavení metadat pro komponentu	14
Zdrojový kód 2 – Ukázka nastavení knihovny angularfire2.....	41
Zdrojový kód 3 – Registrace uživatele	42
Zdrojový kód 4 – Přihlášení uživatele	42
Zdrojový kód 5 – Ukázka vkládání dat do Firebase databáze	45
Zdrojový kód 6 – Ukázka synchronizace realtime dat z databáze Firebase	45
Zdrojový kód 7 – Registrační asynchronní funkce	46
Zdrojový kód 8 – Přihlašovací asynchronní funkce	48
Zdrojový kód 9 – Ukázka kódu pro změnu profilového obrázku.....	50
Zdrojový kód 10 – Ukázka kódu – připojení k MongoDB.....	55
Zdrojový kód 11 – Vkládání a mazání dat z MongoDB kolekce	56

SEZNAM PŘÍLOH

Příloha 1 CD

PŘÍLOHA P I: CD