Realtime aplikace na platformě iOS

Roman Turna

Bakalářská práce
2018

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

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

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Roman Turna**
Osobní číslo:	**A15606**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Informační a řídicí technologie**
Forma studia:	**kombinovaná**
Téma práce:	**Realtime aplikace na platformě iOS**
Téma anglicky:	**Real-time Applications on the iOS Platform**

Zásady pro vypracování:

1. **Prozkoumejte a popište nejpopulárnější open-source a komerční realtime databáze používané při vývoji mobilních aplikací.**
2. **Stručně popište nástroje pro vývoj mobilních aplikací na platformě iOS a možnosti nasazení realtime databáze.**
3. **Vyberte si některou realtime databázovou technologii z komerčního a open-source světa, popište ji a prozkoumejte možnosti jejího praktického nasazení do reálného projektu vývoje mobilní aplikace pro iOS.**
4. **Navrhněte demonstrační aplikaci a popište, jakým způsobem demonstruje specifické vlastnosti a možnosti realtime databáze.**
5. **Aplikaci implementujte a popište důležité kroky a úskalí vývojové fáze.**

Rozsah bakalářské práce:

Rozsah příloh:

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

Seznam odborné literatury:

1. **NEUBURG, Matt. iOS 11 programming fundamentals with Swift: Swift, Xcode, and Cocoa Basics. Sebastopol, CA : OReilly Media, Incorporated, 2017. ISBN 9781491999318.**
2. **MANNING, Jon, BUTTFIELD-ADDISON, Paris and NUGENT, Tim. Learning Swift: building apps for macOS, iOS, and beyond. Sebastopol, CA : OReilly, 2017. ISBN 9781491967065.**
3. **NEUBURG, Matt. Programming iOS 10 Dive Deep into Views, View Controllers, and Frameworks. s.l. : OReilly UK Ltd, 2016. ISBN 9781491970164.**
4. **CHENG, Fu. Build mobile apps with Ionic 2 and Firebase: hybrid mobile app development. United States : Apress, 2017. ISBN 9781484227374.**
5. **TIEPOLO, Gianluca. Getting started with rethinkdb. Packt Publishing Limited, 2016. ISBN 9781785884467.**
6. **Cloud Firestore | Firebase. Google [online]. [cit. 24. 11. 2017]. Dostupné z: https://firebase.google.com/docs/firestore/**

Vedoucí bakalářské práce: **Ing. Radek Vala, Ph.D.**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **15. prosince 2017**

Termín odevzdání bakalářské práce: **25. května 2018**

Ve Zlíně dne 15. prosince 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské 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/bakalářská 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/bakalářskou 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/bakalářskou 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/bakalářské 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/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské 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/bakalářské 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

.....
podpis diplomanta

ABSTRAKT

Tato bakalářská práce se zabývá využitím realtime databázových systémů v mobilních aplikacích určených pro platformu iOS od firmy Apple. Práce je rozdělena do dvou částí. V teoretické části popisuje obecně princip fungování realtime databází, možnosti vývoje aplikací pro platformu iOS a srovnání a výběr realtime databází pro praktickou část. Ta poté obsahuje popis demonstrační aplikace, postup realizace a příklady zdrojových kódů.

Klíčová slova:

iOS, vývoj mobilních aplikací, realtime databáze

ABSTRACT

This Bachelor's thesis is focused on usage of realtime database systems in mobile applications developed for the Apple iOS platform. The thesis is divided into two parts. The theoretical part describes realtime databases in more broad terms, options of development for the iOS platform and comparison and selection of realtime databases used in practical part. The practical part then contains description of the demonstration app, implementation description and samples of source code.

Keywords:

iOS, mobile app development, realtime databases

Tímto bych rád poděkoval Ing. Radku Valovi, Ph.D. za cenné rady, vstřícný přístup a čas věnovaný konzultacím a pomoci při realizaci.

Prohlašuji, že odevzdaná verze bakalářské/diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD **9**
I TEORETICKÁ ČÁST **10**
1 REALTIME DATABÁZE **11**
1.1 STRUČNÝ POPIS REALTIME DATABÁZÍ 11
1.2 REALTIME DATABÁZE PRO PLATFORMU IOS 12
1.2.1 Komerční RTDBS 12
1.2.1.1 Google Firebase 12
1.2.1.2 Google Firestore 13
1.2.2 Open source RTDBS 14
1.2.2.1 RethinkDB 15
1.2.2.2 Realm Database 15
1.2.3 Běžně používané databáze v prostředí iOS 16
1.2.3.1 UserDefaults 16
1.2.3.2 SQLite 16
1.2.3.3 Core Data 17
1.2.4 Podíl a popularita realtime databází na trhu 17
2 VÝVOJ MOBILNÍCH APLIKACÍ PRO PLATFORMU IOS **19**
2.1 HISTORIE, SOUČASNOST A BUDOUCNOST 19
2.1.1 Vývojové prostředí 20
2.1.2 Programovací jazyky 22
2.1.2.1 Objective-C 22
2.1.2.2 Swift 23
2.1.3 Používané architektury 24
2.1.3.1 MVC 24
2.1.3.2 MVVM 25
2.1.3.3 VIPER 26
2.1.4 Nejnovější trendy na poli vývoje pro iOS 27
2.1.4.1 Multiplatformní vývoj 27
2.1.4.2 Rozšířená a virtuální realita 28
2.1.4.3 Strojové učení a umělá inteligence 28
II PRAKTICKÁ ČÁST **29**
3 POŽADAVKY NA APLIKACI **30**
3.1 FUNKČNÍ POŽADAVKY 30
3.2 NEFUNKČNÍ POŽADAVKY 30
3.2.1 Podporovaný operační systém 31
3.2.2 Programovací jazyk 31
3.2.3 Podpora hardwaru 31
3.2.4 Bezpečnost 31
3.2.5 Odezva a výkon 31
3.2.6 Uživatelské rozhraní a použitelnost 31
3.2.7 Lokalizace 32
3.2.8 Use case diagram 32
4 GRAFICKÉ ROZHRANÍ APLIKACE **33**
4.1 POPIS JEDNOTLIVÝCH OBRAZOVEK 34
4.1.1 Obrazovka pro přihlášení k databázi Firestore 34

4.1.2 Obrazovka s kompasem 36
4.1.3 Obrázek kompasu 40
4.1.4 Obrazovka s orientací zařízení v prostoru 41
4.1.5 Datové modely 43
4.1.5.1 Datová vrstva pro aplikaci s Google Firestore 43
4.1.6 Správa dat v databázích 44
4.1.6.1 Konzole Firebase pro Google Firestore 45
4.1.6.2 Realm Studio 45
4.2 STRUKTURA PROJEKTŮ 46
ZÁVĚR 48
SEZNAM POUŽITÉ LITERATURY 49
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK 51
SEZNAM OBRÁZKŮ 52
SEZNAM PŘÍLOH 53

ÚVOD

Obrovský nárůst oblíbenosti chytrých telefonů a neustále se zvětšující nároky na přenos a synchronizaci dat s okolím kladou rostoucí nároky na s nimi spojené databázové systémy. A díky stále větším, někdy dokonce neomezeným, datovým tarifům už není třeba až tak řešit přenos každého kB směrem k uživateli nebo od něj. Dnešní uživatel je také zvyklý mít všechna svá data neustále při sobě a vědět okamžitě o každé příchozí zprávě nebo žhavé novince ve světě.

To samozřejmě klade velké nároky na databázové a synchronizační systémy spojené s mobilními aplikacemi a jejich serverovými protějšky. Databázové systémy dnes masivně používané například ve firemních prostředích a webových serverech nebyly na tyto požadavky stavěny a díky tomu je vývoj aplikací, které se snaží o navození dojmu provozu v reálném čase především pro programátory někdy velkým oříškem.

Na trhu se tak v posledních letech začaly objevovat tzv. realtime databáze speciálně určené pro tyto účely, kdy programátor, a především uživatel nemusí řešit samotnou manuální synchronizaci dat mezi mobilním telefonem a vzdálenou databází. Databázový systém se o neustálou aktualizaci dat postará sám a tím všem zúčastněným stranám usnadní práci.

Tato práce má tedy za cíl v teoretické části stručně popsat samotné principy realtime databází, prozkoumat aktuální stav trhu a v části praktické poté vybrat několik vhodných kandidátů a na nich postavit demonstrační aplikaci ukazující jejich možnosti. Samotná aplikace je postavena na platformě iOS pro mobilní telefony společnosti Apple.

I. TEORETICKÁ ČÁST

1 REALTIME DATABÁZE

V následujících kapitolách jsou obecně popsány realtime databáze a jejich využití na platformě iOS. Také jsou uvedeny nejčastěji používané komerční a open source varianty spolu s jejich přednostmi a negativy.

1.1 Stručný popis realtime databází

Realtime databázový systém (někdy také označovaný zkratkou RTDBS [7]) je databázový systém, který poskytuje většinu funkcí tradičních databázových systémů, jako je například nezávislost dat, a zároveň k nim přidává časová omezení konkrétní aplikace.

Fungují tedy jako efektivní úložiště a umožňují získávání a zapisování dat, stejně jako známé databázové systémy. Navrch ale přidávají časové omezení pro úlohy s daty tak, aby byly zajištěny požadavky konkrétní aplikace, jež je zrovna využívají. RTDBS je tak databází, která má konečný termín platnosti dat, jinak řečeno časové omezení validity dat. Toto časové omezení není pevně stanoveno. Záleží na konkrétní situaci a požadovaných parametrech. Pro některé aplikace jde o milisekundy, jiným stačí validita v rámci minut, nebo i déle. Hlavním rozdílem je tak právě snaha o co nejlepší dodržení tohoto časového omezení.

Dalším rozdílem jsou požadavky na výkon, správnost dat a předpoklady aplikace. RTDBS tak mohou být na rozdíl od svých tradičnějších verzí posuzovány například podle toho, jak často nesplní nastavený časový limit platnosti informace, průměrného zpoždění, náklady způsobenými zpožděním nebo konzistencí externích dat.

Často uváděným příkladem využití RTDBS jsou aplikace pro burzovní systémy. Kótovaná cena se velmi rychle mění, hlavně na velkých burzách, a co nejrychlejší aktualizace dat je kritická pro všechny zúčastněné strany. I přes změny v datech v řádech milisekund musí celý systém fungovat, pokud možno s co nejmenším zpožděním, minimálními výpadky a veškeré změny musí být archivovány pro pozdější využití.

Jiným příkladem mohou být nejrůznějších "chatovací" aplikace hodně populární právě mezi uživateli chytrých mobilních telefonů. Zde již není absolutní prioritou co nejmenší latence mezi odeslanou a přijatou informací. Požadavek na spolehlivou archivaci ale zůstává a přidává se k němu bezchybný, a hlavně rychlý přístup do historie konverzace. Také se mění povaha dat, neukládají se pouze textové informace, ale také například fotografie a zvuky.

Jedním z hlavních poznatků, které by si měl čtenář odnést, je fakt, že slovo realtime neznamená pouze v co nejkratším časovém měřítku. Také nejde o časová měřítka v řádech nano – nebo mikrosekund. Realtime znamená splnit jasně daná časová kritéria předvídatelným způsobem.

1.2 Realtime databáze pro platformu iOS

Výběrem konkrétních RTDBS pro tuto bakalářskou práci se zabývají další kapitoly této práce. Zde jde pouze o přehled některých dostupných řešení pro platformu iOS.

1.2.1 Komerční RTDBS

Komerční realtime databáze v této práci zastupují dvě řešení. Obě jsou od společnosti Google, která je na tomto poli lídrem na trhu. Firebase je několik let zavedeným řešením využívaným velkými společnostmi i malými aplikacemi. Firestore je žhavá novinka a velmi rychle se stává mezi vývojáři populární.

1.2.1.1 Google Firebase

Pravděpodobně nejpoužívanějším systémem je Firebase od společnosti Google. Jde o nejrychleji rostoucí službu na trhu a mezi vývojáři si získala velkou popularitu. Systém je multiplatformní, podporuje nejen iOS, ale samozřejmě i operační systém od Googlu Android. Dále podporuje webové aplikace napsané v Javascriptu, lze jej použít jako REST API endpoint, v aplikaci napsané v jazyce C++ nebo využívající grafický engine Unity.

Synchronizaci a ukládání dat řeší přes NoSQL cloudový server na všechny klienty a lze ji používat i v offline režimu [9]. Samotná data jsou pak ukládána jako JSON (Javascript object notation), což značně ulehčuje práci právě pro vývojáře využívající webové technologie a mobilní aplikace.

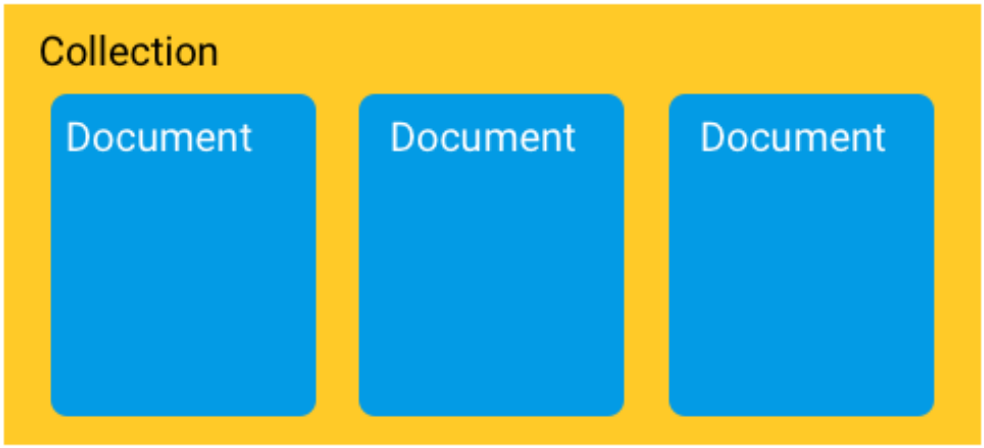
Definice toho, kdo má k jakým datům přístup pomocí zápisu a čtení se určuje pomocí pravidel nazvaných Firebase Realtime Database Rules [4]. Tyto pravidla jsou uložena na serveru a jsou automaticky aplikována ve všech případech.

Velkou výhodou hlavně pro jednotlivé vývojáře je možnost provozovat Firebase jako takzvaný BaaS (Backend as a service), tedy bez nutnosti spravovat vlastní serverový systém. Základní balík služeb je také dostupný zdarma s omezením na určitou velikost databáze a množství přenesených dat či počet dotazů na databázový server. Každý tak má možnost si službu vyzkoušet zdarma s minimálním úsilím při prvotním nastavování.

1.2.1.2 Google Firestore

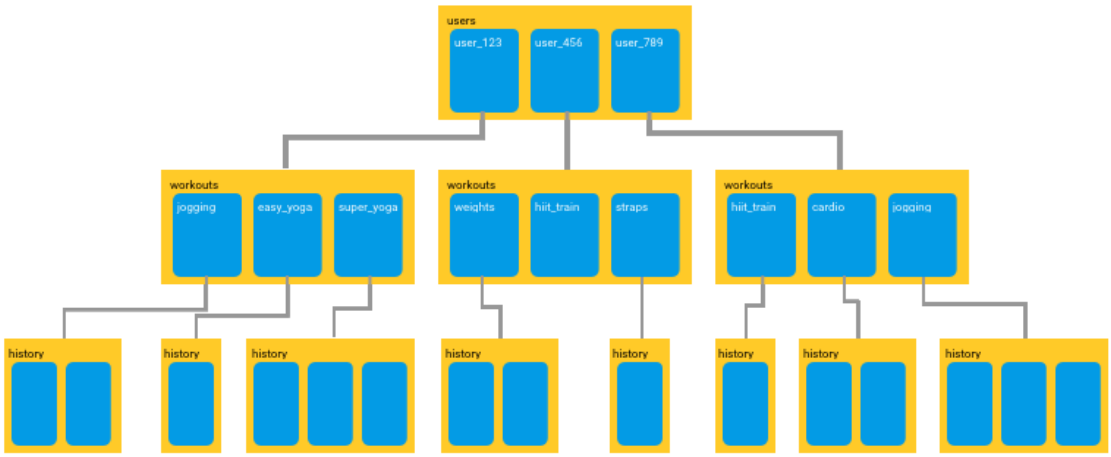
Firestore je produkt velmi podobný výše zmíněnému Firebase. Také pochází od Googlu a také jde o cloudové NoSQL řešení. Jde o velmi mladou technologii představenou v říjnu roku 2017 [8]. Stejně jako Firebase podporuje platformy Android, iOS a má své SDK také pro Javascript a REST API. Navrch přidává podporu pro Node.js, Python, Javu a GO.

Na rozdíl od služby Firebase nabízí Firestore lepší strukturování dat v databázi. Firebase je v podstatě jeden obří JSON, zatímco Firestore takzvaná "document-model" databáze [6]. To znamená, že veškerá data jsou ukládána v objektech zvaných dokumenty (documents), které se skládají z párů klíč-hodnota. Tyto hodnoty mohou obsahovat různá data, od textových řetězců přes čísla s plovoucí čárkou (float), binární data až po data podobná JSONu, která vývojáři Googlu nazývají mapy (maps). A tyto dokumenty jsou nazývány kolekce (collections).



Obrázek 1 Schéma kolekce dokumentů uložených v databázi Firestore [8]

Firestore databáze pak může obsahovat těchto kolekcí několik, propojených přes dokumenty, které odkazují na další subkolekce. Tímto způsobem pak vzniká stromová struktura dat, jak je uvedeno na dalším obrázku.



Obrázek 2 Stromová struktura dat v databázi Firestore [8]

Tato nová struktura má několik výhod v oblasti dotazů nad uloženými daty. Všechny dotazy jsou například "plytké" (shallow), což znamená, že si lze z databáze vyžádat pouze jeden dokument, a není třeba draze stahovat žádné další údaje ze sub-kolekcí. To umožňuje ukládat data hierarchicky podle jasné logiky bez starostí o zbytečné načítání dat.

Firebase má také problémy s velkými databázemi. Umí sice měnit jejich velikost podle potřeby služby kterou obsluhuje, jakmile ale velikost dat překročí určitou hranici nebo se začne připojovat větší množství uživatelů najednou, začne mít Firebase problémy s obsluhou. Firestore je postaven na proprietární cloudové infrastruktuře, která pohání některé z nejnáročnějších řešení, které využívají služby Googlu. Inženýři pracující na vývoji tohoto systému tak slibují podstatně lepší přizpůsobení a kapacitu.

Liší se také účtování za služby. Zatímco Firebase určuje cenu hlavně za objem přenesených a uložených dat, Firestore se zaměřuje především na počet dotazů nad databází. To s sebou přináší lepší odlišení podle potřeb zákazníka a služby kterou se nad těmito databázemi plánuje postavit.

1.2.2 Open source RTDBS

Open source databáze jsou také zastoupeny dvěma řešeními. RethinkDB nemá tak dobrou integraci pro mobilní řešení jako například databáze od společnosti Google výše, Realm Database oproti ní ale s těmito platformami problémy nemá a otevřeným řešením dělá čest.

1.2.2.1 RethinkDB

Stejně jako Firebase je RethinkDB postaveno na datovém modelu JSON a NoSQL [10]. Na rozdíl od něj ale jde o open source řešení, a tak se na vývoji může podílet v podstatě kdokoli. Pro implementaci není uživatel nucen spoléhat se na cloudové řešení a může využít vlastní server k nasazení služby. Pokud ale vlastní server nemá, nebo se nechce zabývat jeho zprovozněním, jsou k dispozici řešení třetích stran, například Amazon se svým AWS.

RethinkDB má také širší škálu dotazů oproti realtime řešení jako je Firebase [5]. Umí tak například posílat různé příkazy typu spojování tabulek, pod-dotazy (subqueries), agregaci nebo třeba funkce map-reduce známé z funkcionálního programování.

Nakonec tento systém umí flexibilněji reagovat na rostoucí nároky aplikace. Prvotní nastavení je sice o něco složitější, následný růst požadavků je ale mnohem snazší. RethinkDB je navržen pro přístup z aplikačního serveru, podobně jako tradiční databáze.

RethinkDB se nehodí pro výpočetně náročné analytické nástroje, zde je lepší použít například Hadoop. Také není příliš vhodný pro řešení požadující příliš velký počet zápisů do databáze v krátkém čase. Upřednostňuje především konzistenci dat.

1.2.2.2 Realm Database

Realm Database je také open source projektem, který původně vznikl pouze pro mobilní platformy Android a iOS. Postupně se pak rozšířil do multiplatformních prostředí Xamarin, React Native a dalších, včetně desktopových aplikací na Windows. Je distribuován pod licencí Apache.

Jde o jednou ze dvou součástí platformy The Real Platform. Druhou součástí je Realm Object Server. Na něm jsou vytvářeny takzvané Realmy (Realms), které mají na starosti uložení dat a jejich následnou synchronizaci do dalších zařízení [11]. Každé zařízení má lokální kopii tohoto Realmu a změny v něm jsou automaticky synchronizovány přes Realm Server.

Data jsou v Realmu uložena jako objekty. Čtení a zápis dat je vždy nejprve prováděn do lokální kopie v mobilním zařízení. Až poté dojde k jejich synchronizaci mimo zařízení, a to v okamžiku, kdy je k dispozici připojení k internetu. Trvalé připojení tak není vyžadováno.

Pokud vývojář nepotřebuje data synchronizovat s kopií v internetu, může si databázi vytvořit čistě lokálně. Dokonce ani nemusí být uložena na disku, stačí-li dočasné úložiště v

operační paměti zařízení. Tímto se Realm Database podobá nativním databázím z prostředí iOS, například SQLite nebo Core Data.

1.2.3 Běžně používané databáze v prostředí iOS

Platforma má několik databázových řešení přímo integrovaných v sobě. Pokrývají potřeby všech druhů aplikací, od těch nejjednodušší po ty velmi rozsáhlé.

1.2.3.1 UserDefaults

Nejjednodušším databázovým systémem na platformě iOS je nativní úložiště nazvané UserDefaults. Jde o třídu, pomocí níž lze přes instanci typu singleton přistoupit k jednoduché databázi typu klíč-hodnota [1]. Ta umí ukládat jednak přímo datové typy jako celá čísla, čísla s desetinnou čárkou, textové řetězce, data, nebo kolekce jež tyto datové typy obsahují. Pokud programátor potřebuje uložit jiný typ dat, je možno je zakódovat do datového typu NSData (Data v jazyce Swift) a pak je uložit.

Práce s databází UserDefaults je velmi jednoduchá a rychlá a veškeré zápisy a čtení z ní jsou také takzvaně thread-safe. To znamená, že nemůže dojít k nechtěnému přepisování právě zapisovaných dat a programátor tak nemusí složitě tyto problémy řešit.

Celá UserDefaults databáze je uložena lokálně v jednom zařízení a zároveň je zálohována, pokud má uživatel zálohy systému a nastavení aplikací povoleny. K jedné instanci UserDefaults má přístup pouze ta aplikace, ke které patří. Výjimkou je sdílení mezi hlavní aplikací a rozšířeními, které k ní patří. Příkladem může být Today widget nebo aplikace pro hodinky Apple Watch. Díky tomu je tak sdílení nastavení a drobných dat mezi takto jinak oddělenýmkódem mnohem snadnější.

1.2.3.2 SQLite

Jde o open-source relační databázový systém široce používaný nejen na platformách iOS a MacOS od Applu, ale i například na Androidu, Linuxu a ve Windows. Nejde o tak výkonnou databázi, jakou jsou například sofistikovanější MySQL nebo SQL Server, na rozdíl od nich je ale její používání o něco jednodušší, nevyžaduje samostatný server a pro použití na mobilních platformách je o mnoho vhodnější.

Jak bylo uvedeno výše, SQLite nevyžaduje samostatný serverový proces, data totiž ukládá přímo na disk zařízení do jednoho souboru. Tvůrci sami uvádějí, že nejde o náhradu systémů jako je Oracle, ale spíše C funkce fopen().

1.2.3.3 Core Data

Core Data je nativní knihovnou platformy iOS a MacOS pro řešení modelové vrstvy aplikace [1]. Apple uvádí, že využitím této technologie programátor ušetří 50-70 % kódu. Core Data má mnoho funkcí, která vývojáři usnadní práci a zamezí zbytečnému duplikování kódu. Níže je uvedeno několik z nich pro přehled.

- Zabudované sledování změn v uložených datech a management změn s historií krokování zpět a vpřed
- Údržba propagace změn v hierarchii dat a hlídání konzistence vztahů mezi daty
- Lazy nahrávání dat (až v okamžiku potřeby) a vytvoření kopie a zápis dat až v případě potřeby
- Možnost propojení datové vrstvy přes controller až do UI vrstvy a propagace změn v datech
- Shlukování, filtrování a organizace dat v paměti a v uživatelském rozhraní
- Automatická podpora pro ukládání objektů v externích datových úložištích
- Velmi efektivní propojení s nástroji v systémech iOS a MacOS

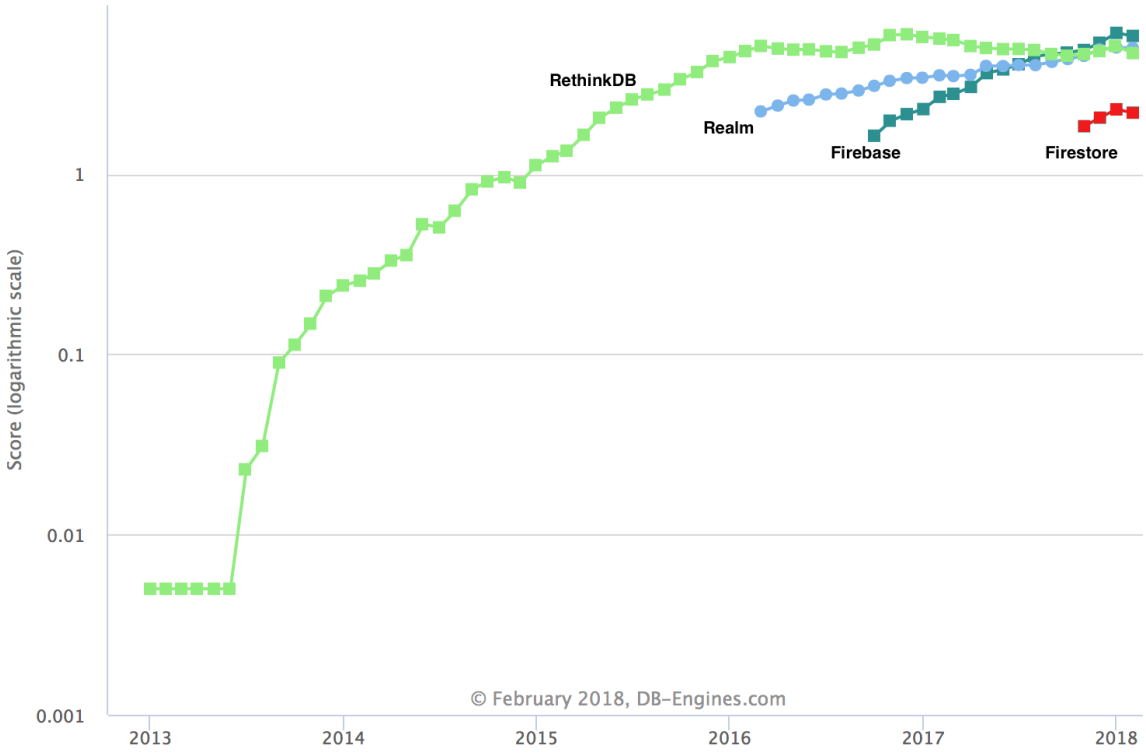
Core Data může data serializovat do formátu XML, do binárního souboru nebo do výše zmíněného SQLite. Core Data pak komunikují napřímo se SQLite, programátor tak nemusí využívat přímo jazyk SQL.

1.2.4 Podíl a popularita realtime databází na trhu

K poměrnému porovnání bylo využito zdrojů webu [12]. Na grafu níže je vidět poměrné zastoupení čtyř výše zmíněných databází a jejich vývoj popularity v čase. Popularita je měřena pomocí několika kritérií:

- Počet zmínek o systému na internetových stránkách. Počítají se počty výskytů ve vyhledávačích Google, Bing a Yandex. Pro větší relevanci se vyhledávají výrazy ve tvaru <název systému> s výrazem database, tedy například Oracle database.
- Obecný zájem o systém. Pro toto měření se používá frekvence výskytu hledání ve službě Google Trends.
- Frekvence výskytu technických diskuzí o systému. Počítá se výskyt otázek na systém na populárních dotazovacích webech, jako jsou Stack Overflow a DBA Stack Exchange.

- Počet nabídek práce, ve kterých je systém zmíněn. Počítá se výskyt nabídek ve vyhledávačích pracovních pozic Indeed a Simply Hired.
- Počet profilů na profesních stránkách, kde je systém zmíněn. Využívá se nejpoužívanějších mezinárodních sítí LinkedIn a Upwork.
- Relevance na sociálních sítích. Počítá se celkový počet tweetů na sociální síti Twitter.



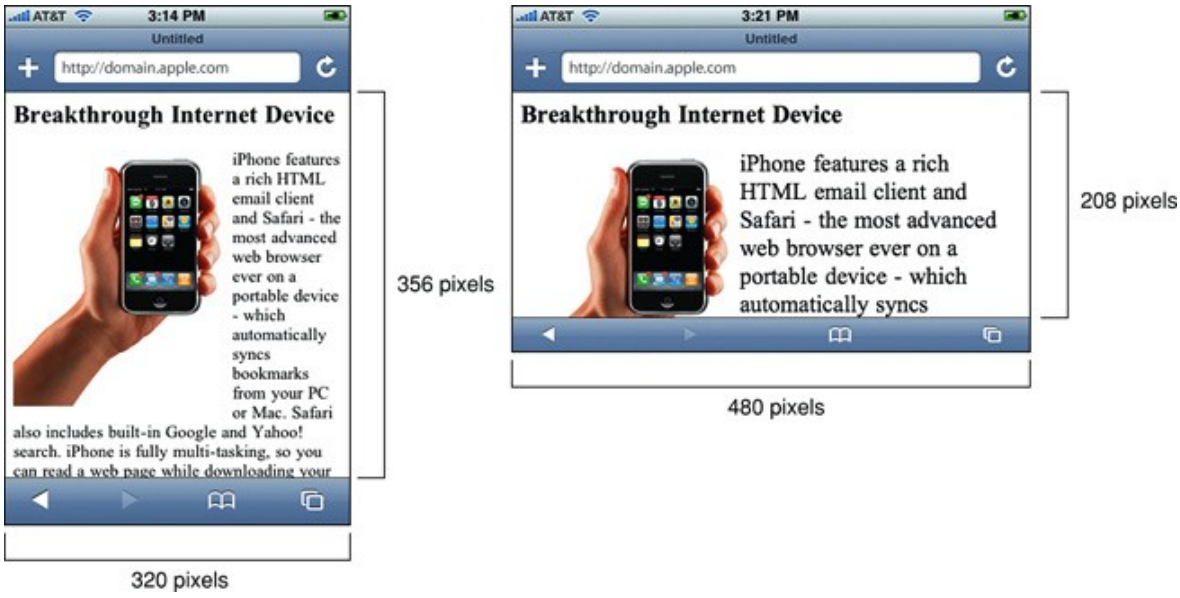
Obrázek 3 Popularita jednotlivých databází [12]

2 VÝVOJ MOBILNÍCH APLIKACÍ PRO PLATFORMU IOS

V této kapitole je popsána historie, současnost a budoucnost vývoje pro mobilní platformu iOS, používané vývojové prostředí, architektury a programovací jazyky.

2.1 Historie, současnost a budoucnost

Operační systém iOS byl světu představen v lednu roku 2007 pod názvem iPhone OS [1]. Původně šlo o uzavřený systém a externí vývojáři neměli mít možnost vyvíjet přímo nativní aplikace. Myšlenka byla taková, že budou vznikat webové aplikace napsané v Javascriptu, které budou mít přístup k systémovým API přes prohlížeč Safari.



Obrázek 4 Obrázek původního prostředí iPhone OS [13]

V říjnu téhož roku bylo ale oznámeno, že Apple přijde s nativním SDK v únoru roku 2008. Nakonec se datum posunulo o měsíc a 6. března byla uspořádána akce, na které byl vývojářům oficiálně představen iPhone SDK. iOS App Store, obchod s aplikacemi pro koncové uživatele, byl otevřen 10. července téhož roku a v okamžiku spuštění na něm bylo k dispozici přes 500 aplikací. Jejich počet se rychle zvětšoval a již v září překročil počet 3 000. V současné době jejich počet překračuje 2 200 000.

Dalšími milníky ve vývoji iOS aplikací jsou září 2007, kdy byl představen iPod Touch a leden 2010, kdy světlo světa spatřil tablet iPad. V červnu roku 2010 také došlo k přeznačení z iPhone OS na iOS, aby jeho název lépe reflektoval univerzálnost zařízení, na kterých běží.

V současné době tedy systém iOS běží na třech druzích zařízení – iPhone, iPod Touch a iPad. Jednou ročně vycházejí bezplatné aktualizace na kompletně novou verzi operačního systému a v průběhu roku poté menší aktualizace s opravami chyb a novými malými funkcemi. V současnosti (začátek roku 2018) je nejaktuálnější verzi iOS 11. Apple je znám podporou i poměrně starých zařízení, nejnovější verzi tak lze nainstalovat i například na iPhone 5S z roku 2013, což je velkou výhodou oproti Android, kde jsou aktualizace podstatně střídmější.

Zvláštní podkapitolou jsou operační systémy Watch OS a Tv OS. Watch OS byl na trh uveden v dubnu 2015 a je určen pouze pro hodinky Apple Watch. Běží na ořezaném jádře iOS a od verze Watch OS 2.0 běží kód aplikací nativně přímo v hodinkách. Za zmínku ještě stojí, že aplikace na hodinky nelze distribuovat samostatně, musí být vždy součástí aplikace pro systém iOS. Tv OS byl oznámen v září 2015 pro čtvrtou generaci zařízení Apple TV a stejně jako Watch OS i on běží na ořezaném jádře iOS. Vyvíjejí se pro něj úplně samostatné aplikace s odlišným UI a běží pouze na zařízení Apple TV.

O budoucím vývoji Apple tradičně nic veřejnosti nesděluje. Jedinou výjimkou jsou každoroční konference pro vývojáře. Na těch inženýři Applu vždy představí nové verze operačních systémů iOS, Mac OS, Tv OS a Watch OS, aktualizovanou verzi vývojářského programu Xcode a novinky v SDK. V poslední době se ale například objevují spekulace, že Apple chystá pro nové verze operačních systémů možnost psát univerzální aplikace, které potom poběží jak na iOS zařízeních, tak na MacOS. Půjde tak pravděpodobně o odezvu na podobnou technologii, kterou již nyní podporují Android a Windows.

2.1.1 Vývojové prostředí

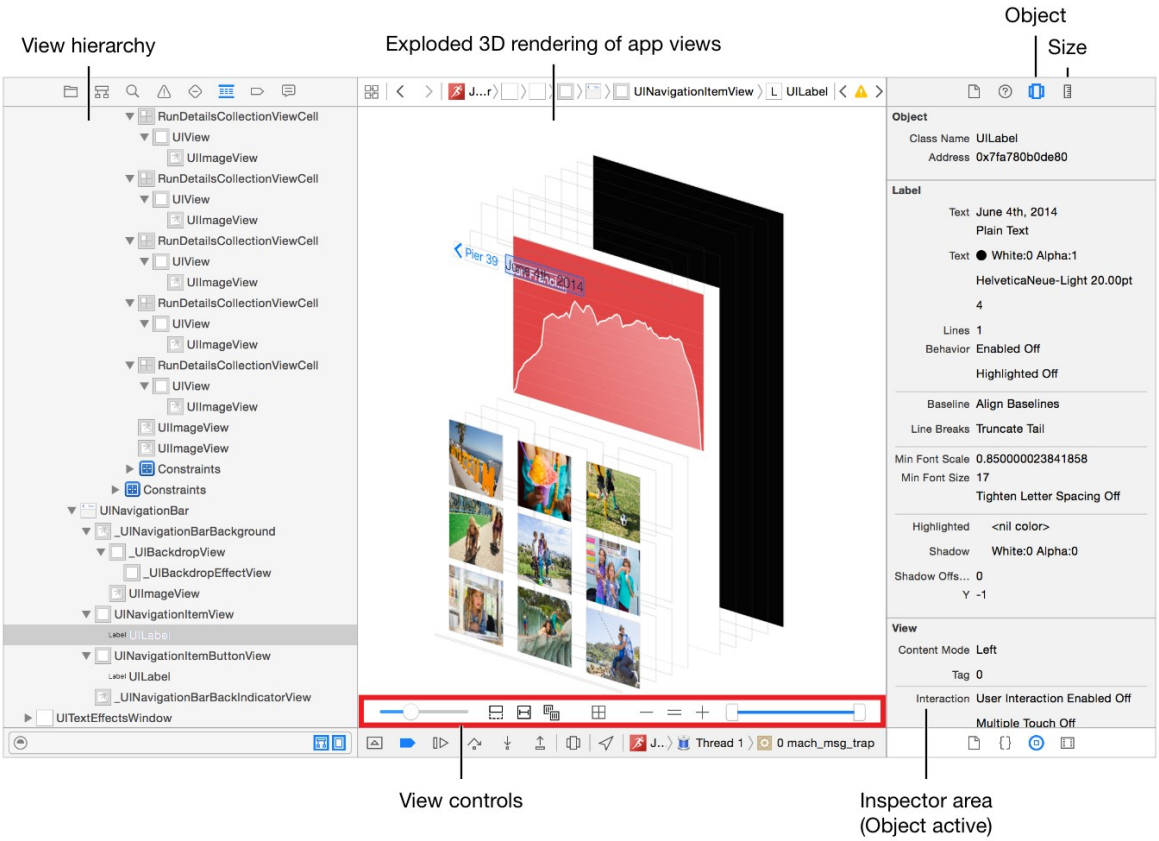
Vývoj nativních aplikací pro platformu iOS probíhá přes program Xcode, který je dostupný pouze pro operační systém MacOS. Xcode je k dispozici zdarma ke stažení buď přes obchod s aplikacemi App Store, nebo přímo na stránkách firmy Apple.

Dále je pro vývoj potřeba vývojářský účet spojený s Apple ID. Ten lze zdarma vytvořit také na stránkách Apple. Naopak není potřeba pořizovat si fyzické zařízení (iPhone nebo iPad), alespoň ne v začátcích vývoje. Pro pozdější testování je to ale určitě dobrá volba. Ne všechny funkcionality lze vyzkoušet v integrovaném simulátoru a ten se ne vždy chová tak, jako fyzické zařízení.

Pokud chce vývojář umístit vyvinutou aplikaci do Apple App Store, bude k tomu potřebovat placený účet. Existují ale výjimky z tohoto pravidla. Apple nedávno představil program pro neziskové a vzdělávací instituce, které mohou být od tohoto poplatku osvobozeny.

Program Xcode je plnohodnotné vývojové prostředí, ve kterém lze vyvinout kompletní aplikaci od návrhu, přes realizaci grafického prostředí a zdrojového kódu až po ladění, diagnostiku a samotné umístění na App Store [3]. Pro návrh uživatelského prostředí slouží takzvaný Interface Builder, kde lze pomocí techniky WYSIWYG (What you see is what you get) vybudovat kompletní grafickou reprezentaci aplikace. To lze ale učinit i čistě pomocí kódu a některými programátory je tento způsob stále preferován.

Xcode také obsahuje velmi užitečné nástroje pro diagnostiku vyvíjené aplikace. Najdeme zde například utility pro hledání úniků paměti, ladění hierarchie uživatelského prostředí, diagnostiku síťové vrstvy nebo diagnostiku zatížení procesoru zařízení či prostředí pro psaní testů. Samozřejmostí je také integrovaná správa verzování kódu například přes populární systém Git.



Obrázek 5 Rozhraní pro ladění uživatelského prostředí [20]

2.1.2 Programovací jazyky

Pro vývoj iOS aplikací se používají především dva jazyky – Objective-C a Swift. Lze ale použít i další, například Objective-C++ nebo Javascript.

2.1.2.1 Objective-C

Programovací jazyk Objective-C byl vyvinut začátkem osmdesátých let Brademem Coxem a Tomem Lovem v jejich společnosti Stepstone. Jde o víceúčelový objektový programovací jazyk postavený na jazyce C s rozšířením pomocí posílání zpráv podobným jazyku Smalltalk. V roce 1988 si společnost NeXT licencovala použití tohoto jazyka pro jejich systém NeXTSTEP a rozšířila kompilátor GCC o jeho podporu. NeXTu se sice nedařilo výrazněji prosadit s jejich pracovními stanicemi, nástroje využívající Objective-C a jejich knihovny AppKit a Foundation Kit ale byly mezi vývojáři velmi oblíbené. NeXT se tak zbavil hardwarové divize a zaměřil se čistě na softwarové nástroje.

Když společnost Apple v roce 1996 koupila NeXT, využila právě jazyk Objective-C, vývojový nástroj Project Builder a nástroj pro tvorbu uživatelského prostředí Interface Builder a integrovala je do jejich nástroje Xcode. Díky tomu tak má dodnes spousta knihoven Cocoa API předponu NS, jde o zkratku slova NeXTSTEP.

Objective-C je striktní nadmnožinou jazyka C, to znamená že aplikace psaná v Objective-C může obsahovat čistý C kód a kompilátor s tím nebude mít žádný problém [1]. Jak již bylo zmíněno výše, Objective-C je založen na Smalltalku a objektový kód tak značně přebírá jeho syntaxi podobnou posílání zpráv. Na rozdíl od jiných jazyků tak programátor "nevolá" metodu, ale "posílá zprávu" instanci třídy. Příklad níže:

```
1  - (UIImage*)generateGradientForView:(UIView*)view
2  {
3      CAGradientLayer *gradient = [CAGradientLayer layer];
4      gradient.frame = view.frame;
5      UIColor *topColor = [UIColor colorWithRed:0.71 green:0.00
   blue:0.00 alpha:1.0];
6      UIColor *endColor = [UIColor colorWithRed:0.90 green:0.00
   blue:0.00 alpha:1.0];
7      gradient.colors = [NSArray arrayWithObjects:(id)
   topColor.CGColor, (id)endColor.CGColor, nil];
8      gradient.locations = @[@(0.0), @(1.0)];
9      return [self generateHeaderImage:NO gradient:gradient];
10 }
```

Krok v jiných jazycích, jako je například C++, nazývaný deklarace třídy, probíhá u Objective-C v takzvaném hlavičkovém souboru s příponou .h. Jde tak o podobný koncept jako u jazyka C. Implementace pak probíhá v implementačním souboru s koncovkou .m.

Jazyk Objective-C je ale již dnes na ústupu. V roce 2014 totiž Apple představil jazyk Swift.

2.1.2.2 Swift

Po téměř dvaceti letech Apple představil úplně nový jazyk míněný jako náhradu za zastarávající Objective-C [2]. Jeho vývoj byl zahájen v roce 2010 a světu byl představen 2. června 2014. Verze 1.0 spatřila světlo světa 9. září téhož roku. Od verze 2.2 je Swift vyvíjen jako open-source a kterýkoli vývojář tak může navrhovat jeho změny a hlásit chyby v implementaci. V letošním roce (2018) je plánováno vydání verze 5.0, která má mimo jiné přinést plnou ABI kompatibilitu. To kromě jiného také znamená, že k aplikacím psaným v tomto jazyce již nebude potřeba přibalovat kompletní Swift runtime knihovny. Ty budou již součástí systému a díky tomu se značně zmenší velikost aplikačního balíčku.

Apple popisuje Swift jako Objective-C bez C. Má mnohem jednodušší syntaxi, kód je lépe čitelný pro programátory a některými prvky připomíná populární skriptovací jazyky jako Javascript nebo Python. Výchozí vlastností Swiftu je na rozdíl od Objective-C skrývání ukazatelů a nebezpečných členských přístupových metod (unsafe accessor). Místo hranatých závorek a "posílání zpráv" jako u svého předchůdce je využívána tečková notace a jmenné prostory, podobně jako u Javy nebo C#.

Ve velkém se také s představováním nativních datových typů upouští od zbytečných NS předložek a s tím také přichází další zjednodušení syntaxe. Přibývají též nové datové typy a operace nad nimi. Nově tak máme k dispozici například kolekci Tuple a více návratových hodnot funkce, generika nebo rychlou iteraci nad kolekcemi pomocí funkcí map(), filter() nebo reduce(). Nejde samozřejmě o všechny novinky jazyka. Kompletní dokumentaci a informace o jazyce lze nalézt na adrese swift.org.

```
11      override func viewDidLoad() {
12          super.viewDidLoad()
13
14          title = "DATA_NETWORK_GUARANTEE".localized()
15          screenType = .DNG
```

```
16          displaysMSISDNSelector = false
17          showsHelpButton = false
18          analyticsPageName = AnalyticDngReportPageName + "result"
19          view.backgroundColor = UIColor.vodafoneBackgroundGray()
20      }
```

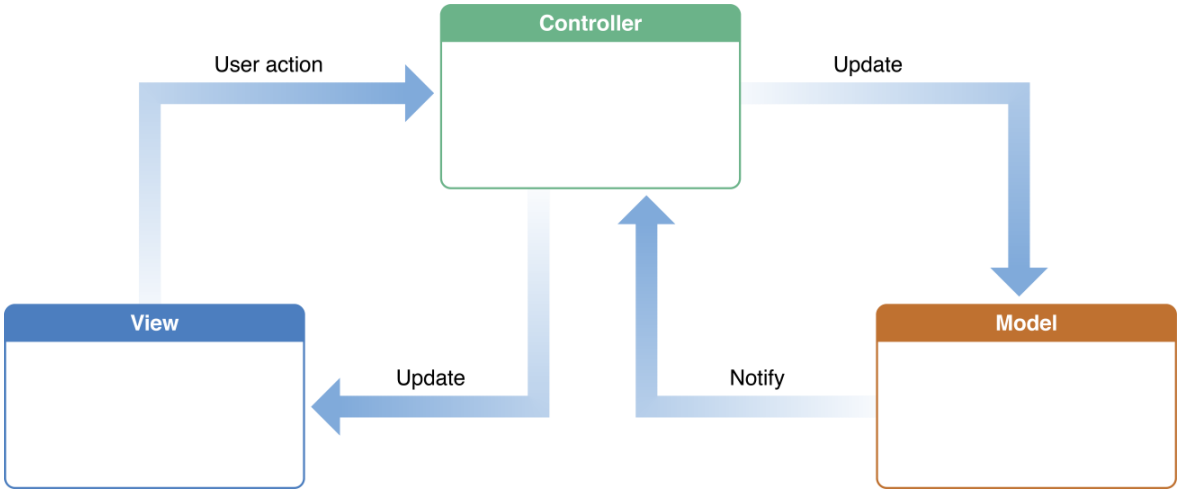
Apple s představením Swiftu pomalu opouští podporu Objective-C. Úplně stáhl veškeré výukové materiály, příklady nových zdrojových kódů už zveřejňuje pouze ve Swiftu a i samotnou implementaci systémů MacOS, iOS, WatchOS a TvOS postupně přepisuje do tohoto nového jazyka. Oficiální ukončení podpory Objective-C ale ještě nebylo zveřejněno a pravděpodobně k tomu ještě dlouho nedojde. Kódová báze Objective-C je přeci jenom za těch téměř dvacet let používání obrovská, a to nejenom u Applu. Zejména u korporátní sféry je přechod na Swift velmi pozvolný a tato oblast trhu se drží hesla "Co není pokažené, neopravuj". Objective-C tak zde s námi bude ještě řadu let existovat.

2.1.3 Používané architektury

Na platformě iOS, stejně jako jinde, programátoři používají několik architektur aplikací. Ta nejoblíbenější a zároveň doporučované Apple je architektura s názvem MVC.

2.1.3.1 MVC

MVC je zkratka pro Model View Controller [14]. Ta zkráceně říká, že program by měl logiku rozdělovat do třech typů tříd.



Obrázek 6 Schéma architektury MVC [14]

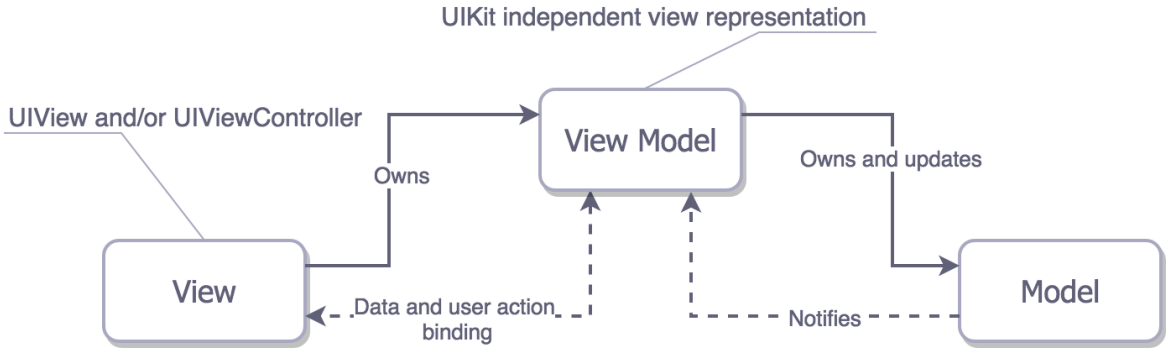
Třídy typu Model slouží pro datovou vrstvu programu. Zde jsou ukládána data nahraná nebo vložená do programu a jsou nad nimi prováděny další operace. Například u osobního kontaktu jsou zde uložena veškerá osobní data vztahující se k tomuto kontaktu. Objekt typu model může mít vztah one-to-one nebo one-to-many s jiným objektem typu model. Modely by ideálně neměly vědět o tom, jak jsou jejich data prezentována uživateli ani odkud ona data přišla. Jejich prací je pouze data uložit a případně nějak zpracovat.

View (pohled) objekty slouží čistě jako grafické prvky na obrazovce zařízení, která zobrazují data získaná původně z modelové vrstvy. Vše, co uživatel programu vidí, je instancí objektu typu View. Tuto instanci nezajímá, odkud data přišla a ideálně by je neměla ani nijak upravovat, prostě je jenom zobrazí uživateli.

Controller (řadič) je takovým prostředníkem mezi objekty typu model a view. Zajišťuje mezi nimi přesouvání dat a také řeší interakci uživatele s programem. Controller neukládá žádná data ani je nezobrazuje, tyto úkoly řeší obě dříve zmíněné vrstvy.

2.1.3.2 MVVM

Nejde ani zdaleka o tak rozšířenou architekturu, jako například u platformy Windows. V poslední době ale začíná nabírat na popularitě, hlavně s příchodem programátorů z jiných platforem.



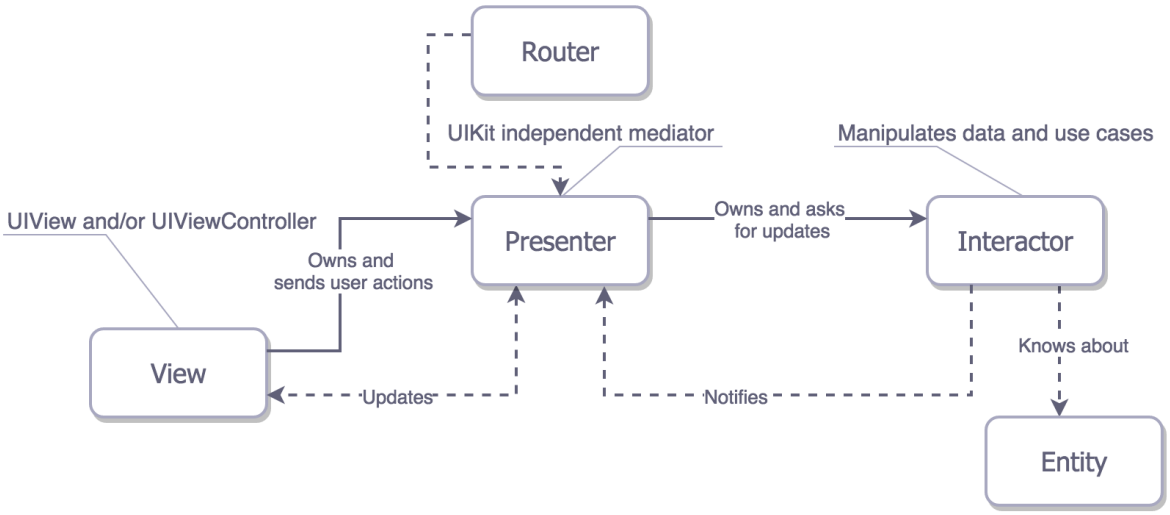
Obrázek 7 Schéma architektury MVVM [15]

Vrstva model je v podstatě stejná jako u architektury MVC, ostatní dvě jsou ale trochu jiné. Mezi vrstvou model a view model existuje takzvaný binding, který zajišťuje notifikaci a změnu dat v modelové vrstvě [15]. Jakmile dojde k aktualizaci dat, je vyslána notifikace do vrstvy view model a ta následně notifikuje vrstvu view, která pak aktualizovaná data zobrazí uživateli.

U této architektury dochází k rozdělení povinností controlleru z MVC do vrstev view a view model. Následkem je lepší testovatelnost kódu a menší náchylnost k nekontrolovatelnému "bujení" kódu v controlleru u MVC.

2.1.3.3 VIPER

VIPER je poměrně novou a zajímavou alternativou k předchozím dvěma popsanými architekturám. Na první pohled se může zdát zbytečně komplikovaným řešením, po bližším prozkoumání se ale jeví pro některé typy projektů jako ideální kandidát. Hlavně u větších softwarových produktů s velkým důrazem na testovatelnost a znovu použitelnost kódu jde o jedno z nejlepších řešení. Skládá se z pěti částí – View, Interactor, Presenter, Entity a Routeru [15].



Obrázek 8 Schéma architektury VIPER [15]

Interactor obsahuje takzvanou business logiku, která obsluhuje data získaná z datové vrstvy, jež je v případě VIPER architektury nazvaná Entitou. Interactor požádá Entitu o splnění určitého úkolu a výstupem z ní jsou data, která pak interactor dále zpracuje a pošle dál. Jde o čistý objekt a jako takový jde snadno testovat. Také by měl být nezávislý na uživatelském rozhraní, lze jej tak snadno použít i v jiných aplikací, a to i napříč platformami.

Entita je datovou vrstvou aplikace. Komunikuje pouze s interactorem a žádnou jinou vrstvou. Často jde pouze o jednoduché struktury, a i zde jde snadno napsat testy pro kontrolu funkčnosti.

Presenter obsahuje logiku pro zpracování dat určených k zobrazení uživateli. Získává je z interactoru a po zpracování posílá dále. Také má na starosti obsluhu interakcí ze strany

uživatele, například při stlačení tlačítka. Tuto informaci zpracuje a buď hned zareaguje například aktualizací uživatelského rozhraní, nebo si vyžádá víc informací z interaktoru. Presenter nikdy nedostane přímo instanci entity, dostává pouze čistá data.

View je čistě pasivní prvek. Pouze čeká, až dostane z presenteru nějaká data a pak je zobrazí. Nikdy si o ně aktivně neříká. Presenter také nezná vnitřní strukturu view, pouze mu předá data a ta se již zobrazí na správných místech správným způsobem.

Posledním, ale neméně důležitým prvkem VIPERu je router. Ten v sobě obsahuje celou strukturu aplikace a tím pádem i vnitřní navigaci mezi jejími obrazovkami. Interakci má na starosti presenter, ten tedy ví, kdy se chce uživatel přesunout jinam a tuto informaci předá routeru. Ten pak vytvoří novou obrazovku a zobrazí ji uživateli.

2.1.4 Nejnovější trendy na poli vývoje pro iOS

Platforma iOS se neustále vyvíjí a ani jí se nevyhýbají nové trendy, ať už jde o rozšířenou realitu, strojové učení nebo multiplatformní vývoj.

2.1.4.1 Multiplatformní vývoj

V poslední době se začíná čím dál tím více prosazovat jednotný vývoj pro všechny mobilní platformy. Tento trend je viditelný především u korporátních aplikací, kde již existuje webové řešení a je pouze potřeba přenést jej na mobilní platformu. Často jde v podstatě pouze o překlopení péče o zákazníka. Typickými příklady jsou mobilní bankovnictví nebo aplikace mobilních operátorů. Ty již mají vybudovaný backend systém a mobilní aplikace tak může často využít stávající API a na straně uživatele data pouze naformátovat a zobrazit.

Vývoj pro několik platforem samostatně tak z pohledu zákazníka nedává moc smysl, platí dvakrát za tu samou věc. Další využití multiplatformního vývoje je z úplně opačného konce mobilní sféry. Vývojáři malých aplikací často nemají potřebné zkušenosti z více platforem, popřípadě nativní mobilní vývoj vůbec neovládají a nemají čas nebo chuť se jej učit. Pokud ale mají zkušenosti s webovým vývojem, existují na trhu nástroje, které jim pomohou splnit jejich cíl.

Nejznámějšími nástroji jsou Xamarin [16] a React Native [17]. Xamarin, patřící společnosti Microsoft, využívá ke stavbě aplikací jazyk C#. Lze pomocí něj programovat nejen pro Android a iOS, ale i vytvářet nativní aplikace pro Windows. Pokud tak má vývojář zkušenost s Microsoft prostředím, nebude pro něj přechod nijak obtížný. Oproti Xamarinu je

React Native zaměřený více na webové vývojáře. Vývoj probíhá v jazyce Javascript a celý nástroj udržuje a rozvíjí jeho majitel, společnost Facebook.

Na první pohled se může zdát, že multiplatformní vývoj je naprosto jasné řešení a nemá v podstatě smysl vyvíjet samostatné aplikace pro jednotlivé platformy. Není tomu ale tak, v některých případech je opravdu lepší držet se nativních nástrojů. Pokud chce například vývojář používat nejnovější technologie dostupné pro tu kterou platformu, nezbývá mu nic jiného než se multiplatformnímu vývoji vyhnout. Podobných problémů existuje více, jsou ale mimo téma této bakalářské práce.

2.1.4.2 Rozšířená a virtuální realita

Společnost Apple na loňské konferenci WWDC pro vývojáře představila knihovnu ARKit [18], určený pro vývoj aplikací využívajících takzvanou rozšířenou realitu. Jde tak jiným směrem než platforma Android, která dříve vsadila spíše na realitu virtuální. Přímo na této konferenci pak bylo představeno i několik demonstračních aplikací a krátce po vypuštění knihovny mezi ostatní vývojáře se jich začalo objevovat velké množství. Do budoucna půjde o velmi zajímavý trh a díky velkému pokroku v oblasti senzorů a kamer do mobilních zařízení budou vznikat čím dál zajímavější a užitečnější řešení.

2.1.4.3 Strojové učení a umělá inteligence

Díky nové knihovně Core ML [19] vstoupil Apple na čím dál zajímavější půdu strojového učení na mobilních platformách. Na pár řádcích kódu tak může vývojář vytvářet inteligentní aplikace, které by dříve zabraly měsíce pokusů a vývoje. Strojové učení a umělá inteligence je již protkána celým systémem iOS. Za všechny případy můžeme zmínit například asistentku Siri, rozpoznávání tváře pomocí Face ID nebo čtení otisků prstů pomocí Touch ID. Ve všech těchto případech se díky zpětné vazbě a dalším algoritmům tyto technologie neustále učí a přizpůsobují novým podmínkám. Například Face ID neustále kontroluje nově pořízené snímky vaší tváře a postupně upravuje datový model tak, aby bylo možno uživatele rozeznat i s pomalu rostoucími vousy, vlasy, s čepicí na hlavě na brýlemi na nose.

II. PRAKTICKÁ ČÁST

3 POŽADAVKY NA APLIKACI

V praktické části této práce bude popsán vývoj dvou aplikací pro platformu iOS, které využívají realtime databáze pro přenos dat. Jedna z nich využívá technologii Realm Database, druhá je postavena nové databázi Google Firestore. Grafické rozhraní aplikací je totožné, až na přihlašovací obrazovku u verze aplikace využívající databázi Firestore. Níže jsou popsány funkční a nefunkční požadavky na tuto aplikaci.

3.1 Funkční požadavky

Funkční požadavky specifikují jednotlivé funkcionality aplikaci. Jde tedy o konkrétní úkoly, které aplikace řeší a které může případně uživatel ovlivnit. V zadání práce nebyly tyto požadavky konkrétně specifikovány, byly tedy po konzultaci s vedoucím práce vybrány takto:

- Aplikace na straně uživatele bude pomocí hardwarových senzorů snímat náklon a rotaci mobilního zařízení
- Tyto informace budou pomocí API realtime databáze synchronizovány na server v cloudu
- Odtud budou dále synchronizovány na všechna zařízení připojená k tomuto serveru
- Cílová zařízení budou graficky znázorňovat aktualizovaná data okamžitě po přijetí ze serveru
- Dvě separátní obrazovky demonstrující přijímaná data:
    - První obrazovka znázorňující kompas s daty z HW kompasu zařízení
    - Druhá obrazovka znázorňující orientaci zařízení v prostoru
- U jedné z aplikací nastavit přístupová práva pro přístup k databázi
- U druhé aplikace nechat databázi otevřenou bez nutnosti se přihlašovat
- Obousměrná komunikace v reálném čase – zařízení mohou data zároveň číst a zapisovat
- Minimalistické uživatelské rozhraní bez nutnosti cokoli nastavovat

3.2 Nefunkční požadavky

Nefunkčními požadavky jsou takové, které nemůže přímo uživatel ovlivnit a určují princip fungování aplikace a její vlastnosti. Jde tak například o výkon, podporované verze operač-

ního systému nebo zabezpečení aplikace. Nefunkční požadavky opět nebyly přesně specifikovány v zadání, byly ale nicméně vybrány takto:

3.2.1 Podporovaný operační systém

Pro co nejsnazší vývoj byla zvolena nejnovější verze operačního systému iOS 11. Nabízí nejnovější API pro vývoj a v dnešní době je nainstalována na naprosté většině mobilních zařízení od společnosti Apple.

3.2.2 Programovací jazyk

Programovacím jazykem pro vývoj aplikace byl vybrán Swift ve verzi 4.0.3. Apple samotný již jakékoli nové ukázky kódu poskytované programátorům uvádí pouze v tomto jazyce a prakticky všichni nový programátoři začínají pouze s tímto jazykem. Objective-C jako druhá varianta je již na ústupu a pro vývoj nových aplikací se již téměř nevyužívá.

3.2.3 Podpora hardwaru

Aplikace je psána jako univerzální, to znamená jak pro iPhone, tak iPad. Díky technologiím jako je Autolayout a s využitím technologie Size class nemá podpora tabletů prakticky žádný dopad na náročnost vývoje.

3.2.4 Bezpečnost

Kromě obvyklých zásad při psaní softwaru nejsou nároky na zabezpečení nijak neobvyklé. Nepřenáší se žádná citlivá data, pouze anonymní údaje z hardwarových senzorů zařízení.

3.2.5 Odezva a výkon

Aplikace sama o sobě má velmi jednoduché rozhraní a objem přenášených dat není velký, žádné specifické kroky k optimalizaci aplikace tak nejsou potřeba aplikovat.

3.2.6 Uživatelské rozhraní a použitelnost

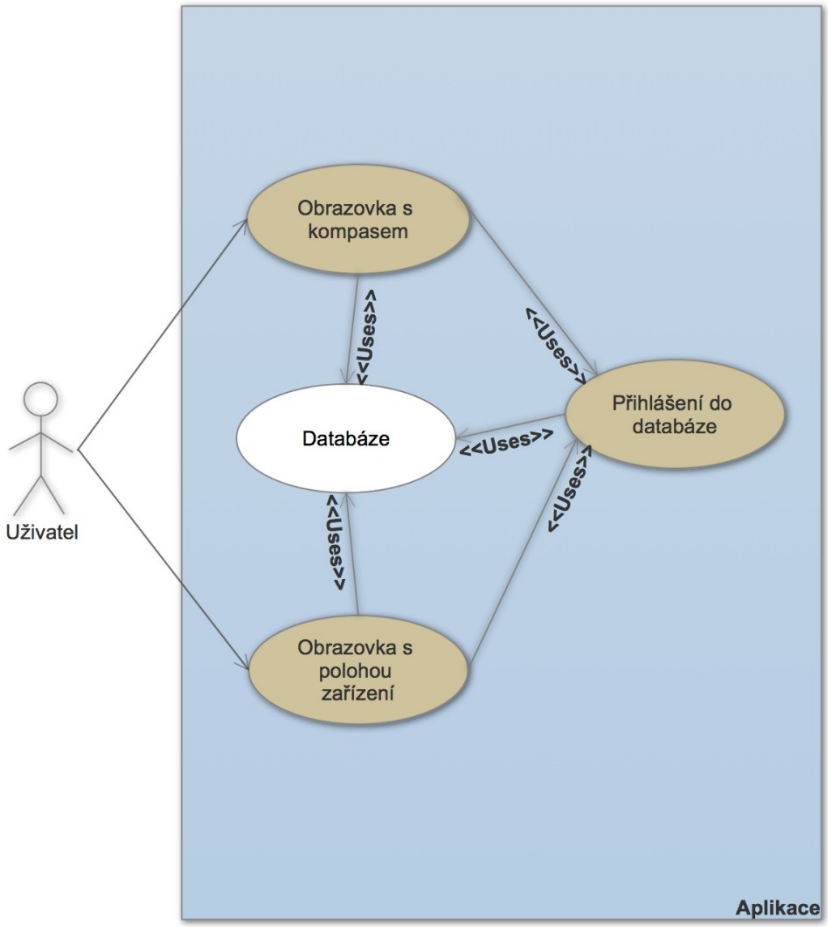
Aplikace využívá standardní grafické a ovládací prvky z knihovny UIKit poskytované přímo společností Apple. Není tak potřeba instalovat ani vyvíjet žádné specifické komponenty

3.2.7 Lokalizace

Aplikace je určena pro demonstrační účely u obhajoby bakalářské práce. Celá je tak lokalizována pouze do češtiny a pro využití v České republice.

3.2.8 Use case diagram

Grafické znázornění funkčních požadavků lze vidět na následujícím grafu. Aplikace má jednoho aktéra, a to aktivního uživatele. Ten si vybírá aktivní obrazovku a následně manipuluje se zařízením, které pomocí hardwarových senzorů snímá polohu. Získané údaje se okamžitě synchronizují do databáze, odkud jsou dále šířena na všechna přihlášená zařízení.



Obrázek 9 Use case diagram aplikace

4 GRAFICKÉ ROZHRANÍ APLIKACE

Téměř celé grafické rozhraní aplikace bylo vytvořeno pomocí aplikace Interface Builder, která je součástí vývojového prostředí Xcode. V něm vytvořený Storyboard představuje digram všech obrazovek aplikace a jejich vzájemné propojení přes menu. Díky tomu je vývoj graficky jednoduchých aplikací jako je tato přístupný všem programátorům bez ohledu na jejich zkušenosti s prostředím iOS. Náhled na tvorbu uživatelského rozhraní je vidět na obrázku níže.

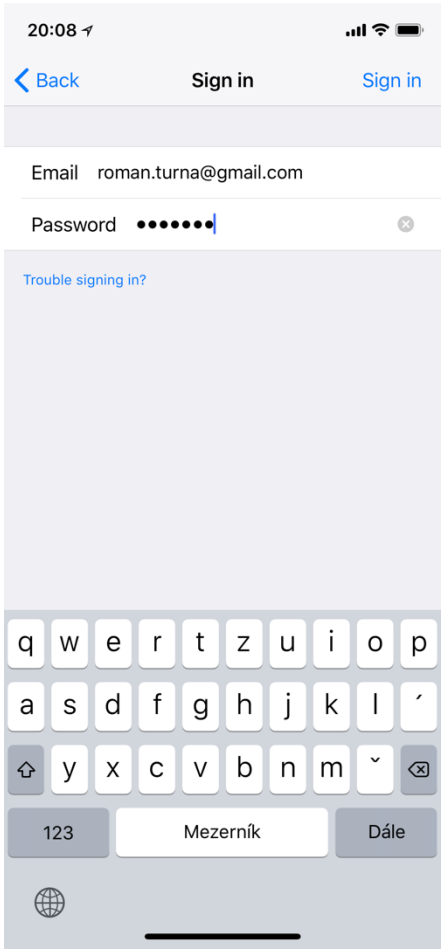
Obrázek 10 Schéma aplikace převzaté z nástroje Interface Builder

4.1 Popis jednotlivých obrazovek

Aplikace se skládá ze dvou obrazovek znázorňujících aktuální data z databáze v grafické podobě, mezi nimiž se přepíná pomocí tzv. Tab Baru umístěného ve spodní části aplikace. Ten se při procházení aplikací nemění, pouze zvýrazňuje právě vybranou obrazovku. Po spuštění aplikace se uživatel dostane na první obrazovku v pořadí s obrázkem kompasu. U aplikace napojené na databázi Firestore se ještě může objevit obrazovka pro přihlášení k databázi.

4.1.1 Obrazovka pro přihlášení k databázi Firestore

Pokud uživatel ještě není přihlášen do databáze, objeví se okno pro zadání přihlašovacího emailu a hesla. Po jejich zadání a klepnutí na Sign In dojde v případě správně zadaných údajů k přihlášení k databázi a okamžitému zahájení přenosu dat z právě aktivní obrazovky. Samotná přihlašovací obrazovka je součástí knihovny Firebase a programátor tak musí pouze zavolat příslušné metody pro její zobrazení.



Obrázek 11 Přihlašovací obrazovka do databáze Firestore

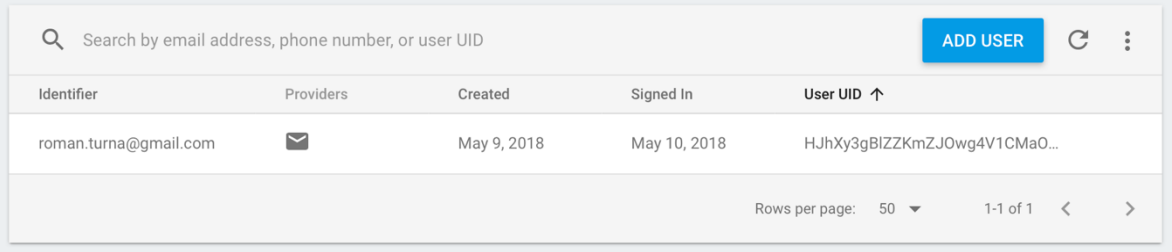
Níže je ukázka zdrojového kódu pro její vyvolání. Na řádku 25 se kontroluje, jestli je již uživatel přihlášen. Pokud ne, zavolá se na řádku 27 metoda present(), která zobrazí výchozí obrazovku pro přihlášení uživatele, která je vidět na obrázku 11 výše.

```
21      /**
22       Shows authorization screen in case the user is not logged in.
23       */
24      func authorizeUser() {
25          let auth = FUIAuth.defaultAuthUI()!
26          if auth.auth?.currentUser == nil {
27              present(auth.authViewController(), animated: true, com-
    pletion: nil)
28          }
29      }
```

Ve webovém rozhraní administrace Firestore databáze jsem se pro tuto demonstrační aplikaci rozhodl nastavit jednoduché pravidlo pro přístup uživatelů k databázi. Mají k ní přístup všichni, kdo mají zřízený uživatelský účet. Pravidla jsou vidět níže:

```
30 service cloud.firestore {
31   match /databases/{database}/documents {
32     match /{document=**} {
33       allow read, write: if request.auth.uid != null;
34     }
35   }
36 }
```

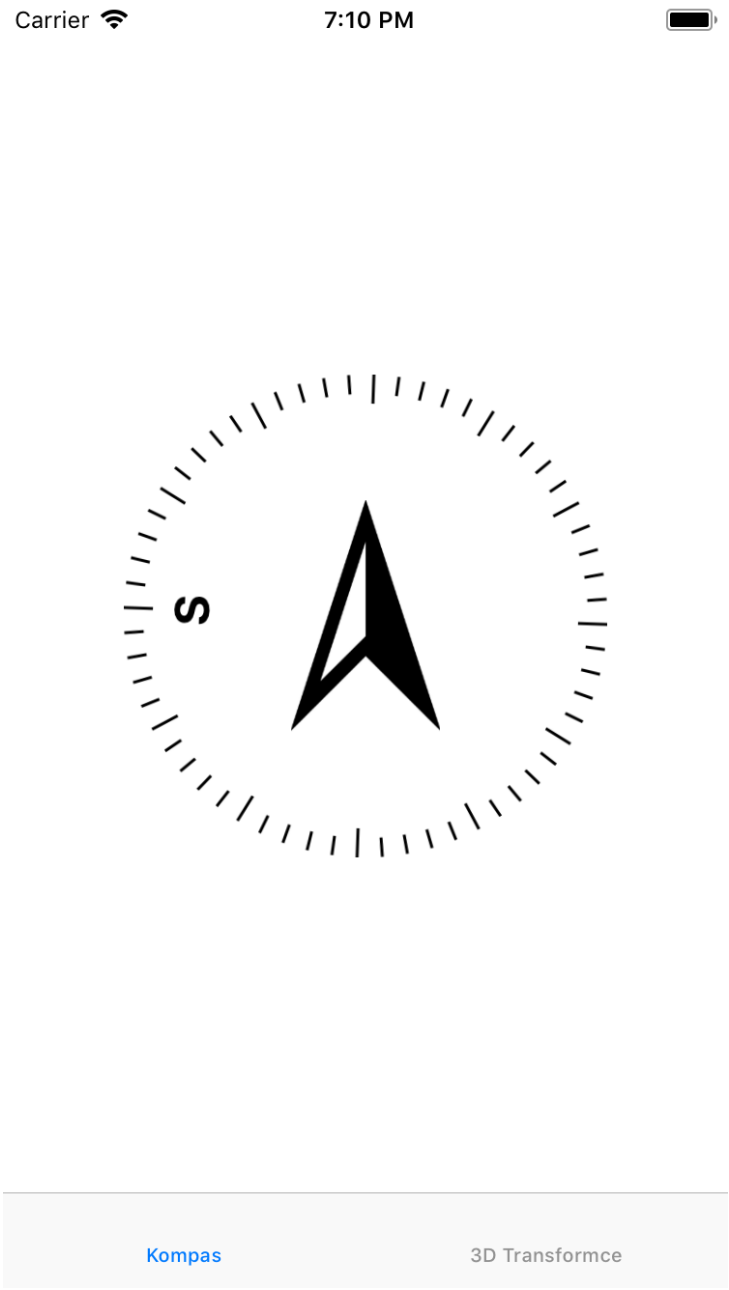
Jak je vidět, pravidlo je velmi jednoduché a čitelné i pro člověka se základním porozuměním angličtiny. Přístup pro čtení i zápis mají všichni, kdo jsou autentifikovaní. Správa uživatelů probíhá také přes webové rozhraní, pod záložkou Authenitication/Users je kompletní seznam vytvořených a přihlášených uživatelů. Lze zde například u každého uživatele resetovat heslo, deaktivovat a smazat účet.



Obrázek 12 Webová konfigurace uživatelů

4.1.2 Obrazovka s kompasem

První výchozí obrazovkou po spuštění aplikace je obrazovka s kompasem. Na lze přehledně vidět graficky zpracovaný kompas, který ukazuje polohu podle dat načtených z databáze. Pokud je uživatel přihlášen k databázi, bude aplikace načítat data z hardwarových senzorů. Ty pak zpracuje a pokud se liší od dat aktuálně uložených v databázi, aktualizuje je v příslušném datovém modelu a synchronizuje do databáze. Odtud si je automaticky stáhnou všichni přihlášení uživatelé a s minimální prodlevou se jim aktualizuje natočení kompasu vyobrazeného na obrazovce.



Obrázek 13 Obrazovka s kompasem

Samotný proces získávání dat, jejich následná kontrola a případné nahrání do databáze je patrný v následující ukázce zdrojového kódu. Nejprve situace pro Google Firestore:

```
37     func locationManager(_ manager: CLLocationManager, didUpdate-
   Heading newHeading: CLHeading) {
38         guard let _ = Auth.auth().currentUser else { return }
39         let compassHeading = Int(-newHeading.magneticHeading)
40         if let currentHeading = position?.compassHeading, cur-
   rentHeading != compassHeading {
```

```
41              let positionDoc = Constants.positionDocument
42              let roll = position?.roll ?? 0
43              let pitch = position?.pitch ?? 0
44              let yaw = position?.yaw ?? 0
45              positionDoc.setData([Constants.rollName: roll,
46                                   Constants.pitchName: pitch,
47                                   Constants.yawName: yaw,
48                                   Constants.compassHeadingName: com-
passHeading])
49          }
50      }
```

Nejdříve se na řádku 31 zkontroluje, jestli je uživatel přihlášen. Pokud ano, uloží se do proměnné compassHeading na řádku 32 aktuální stav natočení zařízení směrem k magnetickému severu jako datový typ Int. Dále se na řádku 33 zkontroluje, jestli je současná nejnovější poloha načtená z databáze v proměnné currentHeading jiná, než hodnota načtený z hardwarových senzorů a pokud ano, získá se na řádku 34 reference na dokument uložený v databázi. Na řádcích 38 až 41 se pak nahraje nový údaj o orientaci kompasu do databáze. Proměnné roll, pitch a yaw nejsou na této obrazovce potřeba, jejich hodnota tak není měněna. Pouze pokud by nebyla inicializována, jsou nastavena do výchozí hodnoty 0.

Zajímavá je ještě část s přidáním tzv. listeneru na referenci dokumentu uloženého v databázi. Zdrojový kód metody addListenerTo(_ query: DocumentReference) je popisuje podrobně:

```
51      func addListenerTo(_ query: DocumentReference) {
52          listener = query.addSnapshotListener { [weak self] (snap-
shot, error) in
53              guard let `self` = self else { return }
54              guard let data = snapshot?.data() else {
55                  print("Data can not be loaded: \(error!)")
56                  return
57              }
58              guard let position = DevicePosition(dictionary: data)
else { return }
59              self.position = position
60              self.rotateCompassTo(position.compassHeading)
61          }
62      }
```

Na řádku 45 vidíme samotné přiřazení listeneru, který má asociovaný blok kódu (closure), jež se zavolá pokaždé, když se ve sledovaném souboru v databázi změní nějaká hodnota. Zde se nejprve kontroluje validita získaných dat v proměnné snapshot. Pokud není něco v pořádku, vypíše se do konzole chybová hláška a metoda skončí. Pokud jsou data validní, uloží se do instanční proměnné aktuální stav na řádku 52 a na řádku 53 se volá metoda rotateCompassTo(), která následně natočí obrázek kompasu do správné polohy.

```
63      func rotateCompassTo(_ heading: Int) {
64          let radians = degreesToRadians(degrees: heading)
65          self.compassView.transform = CGAffineTrans-
form(rotationAngle: radians)
66      }
```

V metodě rotateCompassTo() se nejprve převedou stupně na radiány na řádku 57 a poté se pomocí proměnné transform natočí samotný obrázek.

Databáze Realm má trochu jiný přístup. Krátce po spuštění aplikace se volá metoda startObserving(), která přes notifikační token získává aktualizace z databáze. V samotné closure s kódem se kontroluje stav změny. Nás zajímá tzv. update, který v sobě nese změněná data.

```
67      func startObserving() {
68          notificationToken = position?.observe { [weak self]
(changes) in
69              guard let `self` = self else { return }
70              switch changes {
71              case .initial:
72                  print(changes)
73              case .update(let results, _, _, let modifications):
74                  modifications.forEach({ (index) in
75                      let radians = degreesToRadians(degrees: re-
sults[index].compassHeading)
76                      self.compassView.transform = CGAffineTrans-
form(rotationAngle: radians)
77                  })
78              case .error(let error):
79                  fatalError("\(error)")
80              }
81          }
```

```
82      }
```

Pro každou získanou modifikaci se nejprve získá přepočet ze stupňů na radiány, a poté se natočí samotné view s kompasem.

Data také nejsou manuálně nahrávána do databáze jako v případě Google Firestore, ale jsou ukládána do lokální instanční proměnné, která je propojena s databází a sama si hlídá uložené změny a případně je hned nahrává do databáze. Pokud dojde k výpadku připojení na internet, nic se neděje a data se ukládají lokálně. Jakmile se spojení obnoví, veškerá neaktualizovaná data se automaticky synchronizují s databází.

4.1.3 Obrázek kompasu

Obrázek kompasu je naprogramován jako samostatné view čistě programově. Díky skvělé podpoře systému iOS je programové vykreslování grafických objektů velmi jednoduché a opravdu rychlé. V tomto případě je využito API Core Graphics a doslova na pár řádcích je možno vykreslit přesně to, co programátor zamýšlí. Není tak potřeba importovat žádné veliké obrázky, což šetří operační paměť a úložiště na disku zařízení.

```
83      func addDegreeMarkers() {
84          guard let ctx = UIGraphicsGetCurrentContext() else { return
    }
85          let radius = bounds.width / 2.0
86
87          for i in 0...59 {
88              ctx.saveGState()
89              ctx.translateBy(x: bounds.midX, y: bounds.midY)
90              ctx.rotate(by: degreesToRadians(degrees: i * 6))
91              if i % 5 == 0 {
92                  // Render slightly longer line on hour positions.
93                  renderDegreeMarkerTo(ctx: ctx, x: radius - 15, y: 0,
    radius: radius, color: .black)
94              } else {
95                  renderDegreeMarkerTo(ctx: ctx, x: radius - 10, y: 0,
    radius: radius, color: .black)
96              }
97              // Restore state before next translation.
98              ctx.restoreGState()
99          }
```

```
100     }
```

Metoda addDegreeMarkers() se volá hned v metodě draw(), která je určena právě pro vykreslování uživatelského rozhraní. Na řádku 84 se získá reference na grafický kontext, do kterého se vykresluje. Ve FOR cyklu na řádku 87 se postupně natáčí grafický kontext po jedné šedesátině a pokaždé se vykreslí čárka po obvodu kompasu a u každé páté iterace vykreslí čárka delší.

```
101     func renderDegreeMarkerTo(ctx: CGContext, x: CGFloat, y:
    CGFloat, radius: CGFloat, color: UIColor) {
102         let path = CGMutablePath()
103         path.move(to: CGPoint(x: radius, y: 0))
104         path.addLine(to: CGPoint(x: x, y: y))
105         path.closeSubpath()
106         ctx.addPath(path)
107         ctx.setLineWidth(1.5)
108         ctx.setStrokeColor(color.cgColor)
109         ctx.strokePath()
110     }
```

Metoda renderDegreeMarkerTo() má na starosti právě ono vykreslování čárek. Kód je velmi samo popisný a jenom dokazuje programátorskou přívětivost API Core Graphics. Nejdříve se na řádku 112 inicializuje proměnná path, která představuje "cestu", jež je grafickou reprezentací čárky. Tato cesta se posune na správnou pozici, vloží se do ní čára končící na souřadnicích x a y. Pak se cesta "uzavře", přidá se do grafického kontextu, nastaví se tloušťka čáry, její barva, a nakonec se vykreslí.

4.1.4 Obrazovka s orientací zařízení v prostoru

Většinu plochy této obrazovky zabírá obrázek telefonu, který graficky znázorňuje polohu v prostoru. Podle údajů z databáze se obrázek transformuje v trojrozměrném prostoru. Na rozdíl od předchozí obrazovky s kompasem se zde synchronizují s databází hned tři údaje najednou. Zahájení získávání dat ze senzorů je zde stejné jako kompasu, liší se ale následné zpracování.

Obrázek 14 Obrazovka s prostorovou orientací zařízení

```
111     func updateDevicePositionTo(_ data: CMDeviceMotion) {
112         let newYaw = Int(radiansToDegrees(radians:
    data.attitude.yaw))
113         let newRoll = Int(radiansToDegrees(radians:
    data.attitude.roll))
114         let newPitch = Int(radiansToDegrees(radians:
    data.attitude.pitch))
115
116         guard let yaw = position?.yaw else { return }
```

```
117         guard let roll = position?.roll else { return }
118         guard let pitch = position?.pitch else { return }
119
120         if newYaw != yaw || newRoll != roll || newPitch != pitch {
121             let positionDoc = Constants.positionDocument
122             positionDoc.setData([Constants.rollName: newRoll,
123                                  Constants.pitchName: newPitch,
124                                  Constants.yawName: newYaw,
125                                  Constants.compassHeadingName: posi-
    tion?.compassHeading ?? 0])
126         }
127     }
```

Nejprve se na řádcích 112 až 114 získají údaje ze senzorů a převedou se z radiánů do stupňů. Poté se načtou na řádcích 116 až 118 aktuálně načtená data z databáze. Podmínka IF začínající na řádku 120 zkontroluje, zdali je některý z nových údajů odlišný od těch, které jsou již v databázi a pokud ano, dojde k jejich aktualizaci na řádku 125.

Aplikace pro databázi Realm funguje velmi podobně. Pouze s tím rozdílem, že je aktualizována instance v paměti telefonu, a ta se následně sama postará o nahrání dat do databáze.

4.1.5 Datové modely

Každá z aplikací používá trochu jinou datovou vrstvu, hlavně z důvodu demonstrace možných řešení.

4.1.5.1 Datová vrstva pro aplikaci s Google Firestore

Zde jsou data uložena do struktury s názvem DevicePosition se čtyřmi proměnnými. Ta je inicializována z datového typu Dictionary získaného z databáze. Pokud mají všechny hodnoty správný datový typ, je vytvořena instance a předána zpět do view controlleru. Princip činnosti je vidět v ukázce zdrojového kódu níže.

```
128     init?(dictionary: [String : Any]) {
129         guard let roll = dictionary["roll"] as? Int,
130             let pitch = dictionary["pitch"] as? Int,
131             let yaw = dictionary["yaw"] as? Int,
132             let compassHeading = dictionary["compassHeading"] as?
    Int else { return nil }
133
134         self.roll = roll
```

```
135             self.pitch = pitch
136             self.yaw = yaw
137             self.compassHeading = compassHeading
138         }
```

Samotný konstruktor je typu failable, to znamená že nemusí vrátit žádnou instanci, pokud nedostane správná data. To se kontroluje na řádcích 129 až 132. Pokud je dostane, na řádcích 134 až 137 inicializuje všechny instanční proměnné a tím je instance vytvořena.

Verze aplikace pro Realm databázi naopak používá klasickou třídu, i když se stejným názvem. Je datového typu Object, který je součástí knihovny Realm. Ta se sama stará o inicializaci, pouze instanční proměnné musí nést příznak @objc dynamic, jelikož je knihovna Realm napsána v jazyce Objective-C a díky tomuto lze tuto knihovnu použít i v projektu napsaném ve Swiftu.

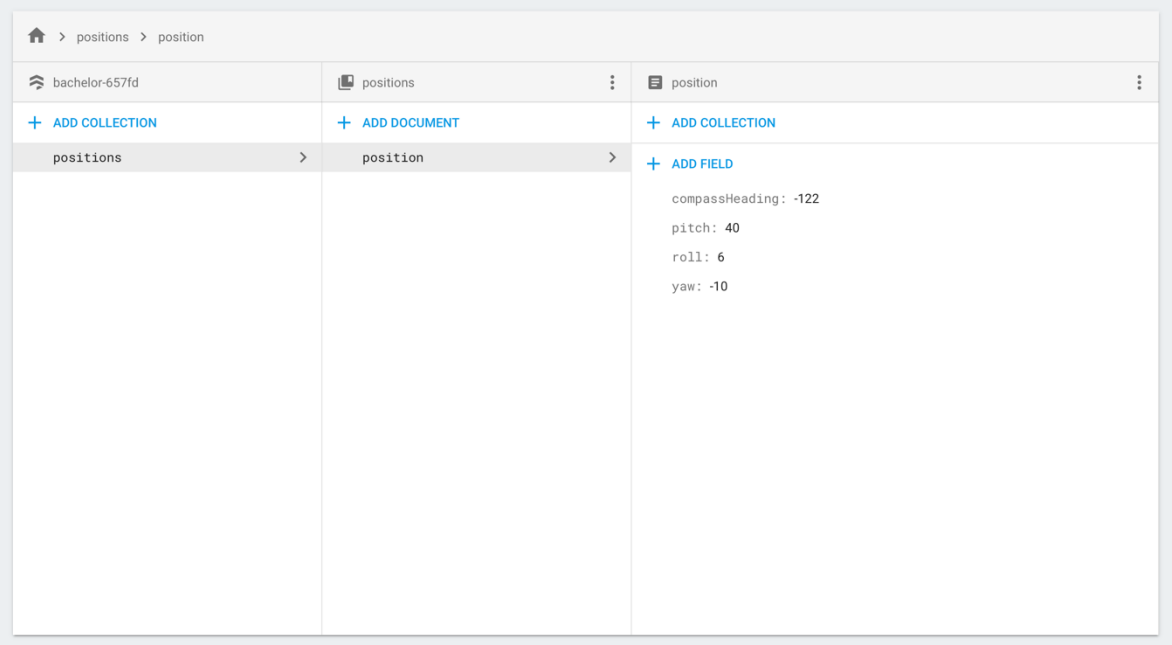
```
139 import RealmSwift
140 class DevicePosition: Object {
141 
142     @objc dynamic var itemId: String = UUID().uuidString
143     @objc dynamic var roll: Int = 0
144     @objc dynamic var pitch: Int = 0
145     @objc dynamic var yaw: Int = 0
146     @objc dynamic var compassHeading: Int = 0
147     @objc dynamic var timestamp: Date = Date()
148 
149     override static func primaryKey() -> String? {
150         return "itemId"
151     }
152 }
```

4.1.6 Správa dat v databázích

Jak Google Firestore, tak Realm Database umožňují spravovat databáze pomocí grafických rozhraní. Díky tomu je i pro relativní začátečníky velmi snadné mít přehled o svých datech a případně dělat jakékoli i velmi pokročilé operaci s nimi poměrně snadně.

4.1.6.1 Konzole Firebase pro Google Firestore

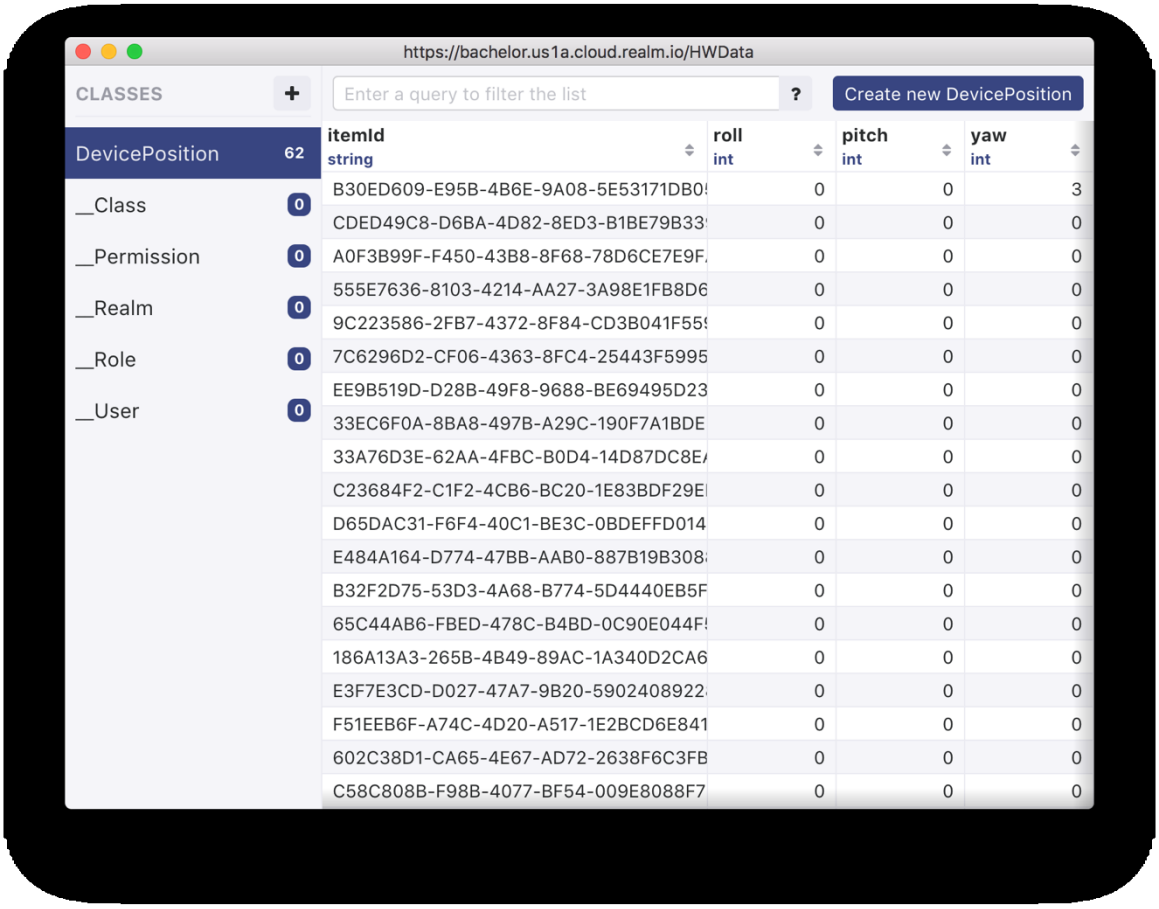
Společnost Google má tradičně velmi zdařilé administrační prostředí a konzole Firebase není výjimkou. Je v podstatě totožná s realtime databází Firebase, kdo zná tuto technologii tak nebude mít problém se zorientovat. Výřez grafického rozhraní pro administraci databáze je vidět na obrázku 14. Nejenom že jsou zde zobrazená data automaticky aktualizována tak jak je zasílají uživatelská zařízení, lze je také manuálně přepisovat a změny se okamžitě projeví na všech přihlášených účtech. Takto můžeme snadno testovat správné propojení s databází a řešit případné problémy.



Obrázek 15 Výřez grafického rozhraní administrace databáze Google Firestore

4.1.6.2 Realm Studio

Společnost Realm nabízí pro správu jejich databáze nativní aplikaci s názvem Realm Studio. Je k dispozici pro všechny hlavní platformy a díky ní je správa databáze opět velmi snadná. Po zadání přihlašovacích údajů a připojení k serveru můžeme vybírat jednotlivé Realmy, kde jsou uložena naše data. Ta jsou opět aktualizována v reálném čase a také je lze ručně měnit. Rozsah administrace není tak velký jako u komerčního řešení Google Firestore, pro většinu uživatelů ale bude plně dostačující.

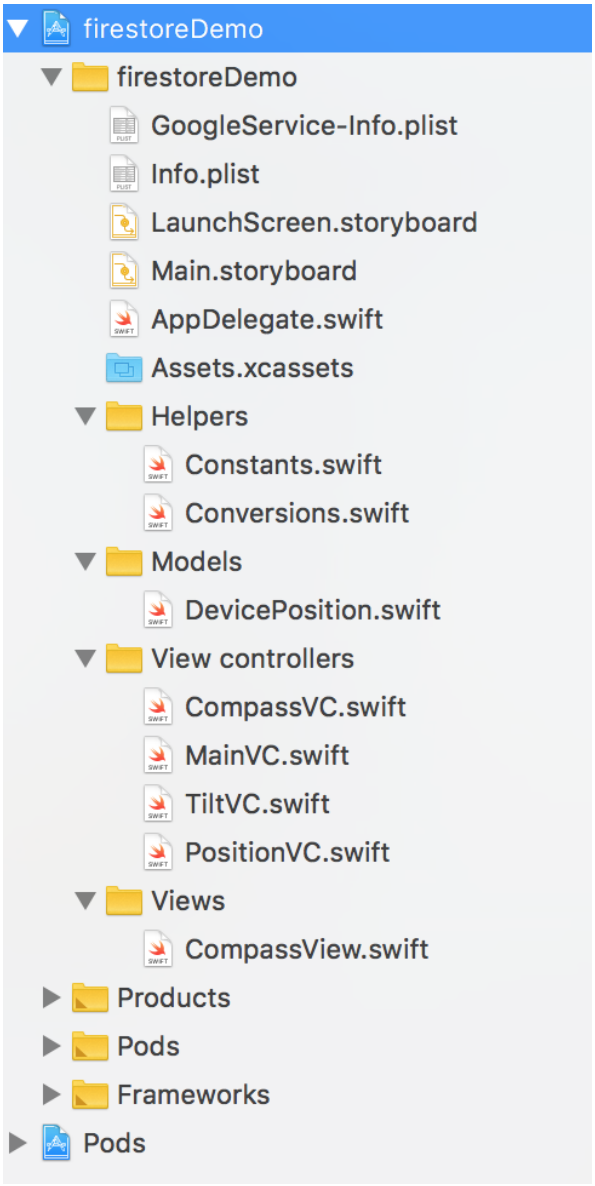


Obrázek 16 Náhled na administraci dat v programu Realm Studio

Z programu Realm Studio lze také přímo zakládat nové demo projekty pro platformy iOS, Android nebo webový Javascript framework Node.js.

4.2 Struktura projektů

Obě aplikace mají stejnou strukturu projektu. Ta odpovídá architektuře MVC, jednotlivé datové modely, view a view controllery mají své samostatné složky. Ve složce Helpers jsou dva soubory obsahující pomocné funkce pro konverzi jednotek a konstanty pro celý program. Obrázky pro aplikaci jsou uloženy ve speciálním pomocném souboru Assets v nejvyšší úrovni projektu. Na tom samém místě jsou pak i zbylé soubory nutné pro běh aplikace – jednotlivé Storyboardy s grafickým rozhraním a plist soubory s konfigurací.



Obrázek 17 Obrázek struktury projektu

ZÁVĚR

V teoretické části této bakalářské práce byl rozebrán vývoj mobilních aplikací pro operační systém iOS od firmy Apple. Byla probrána historie, současnost a budoucnost stejně jako vývojové nástroje a používané technologie. Čtenář tak získal ucelený přehled o tom, co vše byl měl znát a umět, pokud se chce vydat směrem vývoje pro tuto mobilní platformu.

V praktické části byly popsány dvě samostatné aplikace využívající pro synchronizaci dat mezi více zařízeními realtime databází. Obě byly postaveny na architektuře typu MVC, jež je také popsána v této práci. Ta byla zvolena s ohledem na její popularitu mezi vývojáři a případnou snadnou rozšiřitelnost. S tou také byly aplikace vyvíjeny od začátku, jsou totiž zamýšleny jako demonstrace a případný "odrazový bod" pro další možný vývoj.

Google Firestore jako horká novinka od společnosti Google zastupovala komerční sféru, a i když je stále v době psaní této práce v beta verzi, práce s ní byla poměrně jednoduchá na pochopení a v rámci možností intuitivní. Open source svět zastupovalo řešení Realm Database od společnosti Realm, které dokázalo, že otevřený zdrojový kód není překážkou pro kvalitní produkt.

Cílem této bakalářské práce tedy bylo především zjistit, jak obtížné je proniknout do rychle se rozvíjejícího světa realtime databází a vytvořit aplikace, které by srozumitelným způsobem demonstrovaly jejich integraci do praktických řešení. Samotný autor práce byl mile překvapen tím, jak dobře jsou dnes dostupné nástroje použitelné ve světě mobilních aplikací. Instalace knihoven je snadná díky použití široce používaných správců závislostí. Server s databází lze spustit v cloudu díky dobře zdokumentovaným postupům a v obou případech databázových technologií použitých v této práci toto lze učinit dokonce pro nenáročné použití naprosto zdarma. I díky tomu je jejich využití na raketovém vzestupu nejen mezi nezávislými vývojáři malých produktů. Už dnes jejich služeb využívají nadnárodní korporace a projekty s miliony uživatelů. Programátoři je mají v oblibě díky snadnému vývoji a obrovským možnostem škálování, kdy jejich databáze mohou bezproblémově růst zároveň s růstem jejich aplikace. A uživatelé takto postavených aplikací jsou spokojeni s ohledem na jejich spolehlivost a rychlost.

SEZNAM POUŽITÉ LITERATURY

[1] NEUBURG, Matt. iOS 11 programming fundamentals with Swift: Swift, Xcode, and Cocoa Basics. Sebastopol, CA : OReilly Media, Incorporated, 2017. ISBN 9781491999318.

[2] MANNING, Jon, BUTTFIELD-ADDISON, Paris and NUGENT, Tim. Learning Swift: building apps for macOS, iOS, and beyond. Sebastopol, CA : OReilly, 2017. ISBN 9781491967065.

[3] NEUBURG, Matt. Programming iOS 10 Dive Deep into Views, View Controllers, and Frameworks. s.l. : OReilly UK Ltd, 2016. ISBN 9781491970164.

[4] CHENG, Fu. Build mobile apps with Ionic 2 and Firebase: hybrid mobile app development. United States : Apress, 2017. ISBN 9781484227374.

[5] TIEPOLO, Gianluca. Getting started with rethinkdb. Packt Publishing Limited, 2016. ISBN 9781785884467.

[6] Cloud Firestore | Firebase. Google [online]. [cit. 24. 11. 2017]. Dostupné z: https://firebase.google.com/docs/firestore/

[7] LINDSTRÖM, Jan. *Real Time Database Systems* [online]. [cit. 2018-02-24]. Dostupné z: https://www.cs.helsinki.fi/u/jplindst/papers/rtds.pdf

[8] KERPELMAN, Todd. Cloud Firestore for Realtime Database Developers. *The Firebase Blog* [online]. 2017 [cit. 2018-02-24]. Dostupné z: https://firebase.googleblog.com/2017/10/cloud-firestore-for-rtdb-developers.html

[9] *Firebase Realtime Database* [online]. 2018 [cit. 2018-02-24]. Dostupné z: https://firebase.google.com/docs/database/

[10] Frequently asked questions. *RethinkDB* [online]. [cit. 2018-02-24]. Dostupné z: https://www.rethinkdb.com/faq/

[11] The Realm Platform. *Realm Docs* [online]. 2018 [cit. 2018-02-24]. Dostupné z: https://docs.realm.io/platform/getting-started/key-concepts

[12] *DB Engines* [online]. 2018 [cit. 2018-02-24]. Dostupné z: www.db-engines.com

[13] Configuring the Viewport. *Safari Web Content Guide* [online]. 2016 [cit. 2018-02-24]. Dostupné z: https://developer.apple.com/library/content/documentation/AppleApplications/Reference/SafariWebContent/UsingtheViewport/UsingtheViewport.html

[14] Model-View-Controller. *Cocoa Core Competencies* [online]. 2015 [cit. 2018-02-24]. Dostupné z: https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

[15] IOS Architecture Patterns. *IOS App Development* [online]. 2015 [cit. 2018-02-24]. Dostupné z: https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52

[16] *Xamarin* [online]. [cit. 2018-02-24]. Dostupné z: https://www.xamarin.com

[17] *React Native* [online]. [cit. 2018-02-24]. Dostupné z: http://www.reactnative.com

[18] *ARKit* [online]. 2018 [cit. 2018-02-24]. Dostupné z: https://developer.apple.com/documentation/arkit

[19] *Core ML* [online]. 2018 [cit. 2018-02-24]. Dostupné z: https://developer.apple.com/documentation/coreml

[20] Specialized Debugging Workflows. *Debugging with Xcode* [online]. 2017 [cit. 2018-02-24]. Dostupné z: https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/debugging_with_xcode/chapters/special_debugging_workflows.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

RTDBS	Real-Time Data Base System
REST API	Representational State Transfer Application programming interface
JSON	JavaScript Object Notation
BaaS	Backend as a Service
SDK	Software Development Kit
WYSIWYG	What You See Is What You Get
MVC	Model View Controller
MVVM	Model View ModelView
VIPER	View Interactor Presenter Entity Router
WWDC	Worldwide Developers Conference

SEZNAM OBRÁZKŮ

Obrázek 1 Schéma kolekce dokumentů uložených v databázi Firestore [8] 13
Obrázek 2 Stromová struktura dat v databázi Firestore [8] 14
Obrázek 3 Popularita jednotlivých databází [12] 18
Obrázek 4 Obrázek původního prostředí iPhone OS [13] 19
Obrázek 5 Rozhraní pro ladění uživatelského prostředí [20] 21
Obrázek 6 Schéma architektury MVC [14] 24
Obrázek 7 Schéma architektury MVVM [15] 25
Obrázek 8 Schéma architektury VIPER [15] 26
Obrázek 9 Use case diagram aplikace 32
Obrázek 10 Schéma aplikace z interface buildru 33
Obrázek 11 Přihlašovací obrazovka do databáze Firestore 34
Obrázek 12 Webová konfigurace uživatelů 36
Obrázek 13 Obrazovka s kompasem 37
Obrázek 14 Obrazovka s prostorovou orientací zařízení 42
Obrázek 15 Výřez grafického rozhraní administrace databáze Google Firestore 45
Obrázek 16 Náhled na administraci dat v programu Realm Studio 46
Obrázek 17 Obrázek struktury projektu 47

SEZNAM PŘÍLOH

PŘÍLOHA P I: NÁZEV PŘÍLOHY