Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Specifika vývoje aplikací pro systém iOS pomocí jazyka Swift a technologie React Native

Bakalářská práce

Autor: Jan Kodeš

Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Jan Dvořák

Hradec Králové Březen 2017

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

.....

V Hradci Králové dne 28. dubna 2017 Jméno a příjmení

Rád bych poděkoval Ing. Janu Dvořákovi za odborné vedení bakalářské práce a za vstřícnost, čas a poskytnutí cenných rad.

Anotace

Tématem této bakalářské práce je vývoj aplikací pro operační systém iOS, který je používán na mobilních zařízeních značky Apple a následně srovnání dvou zvolených platforem pro vývoj, a to konkrétně programovacího jazyka Swift a javascriptového frameworku React Native. V práci popisuji obě zmíněné možnosti vývoje a zkoumám jejich výhody či nevýhody. Praktická část se soustředí na zjištění výkonu obou platforem a analýze jednoduché aplikace vytvořené pouze za účelem benchmarku. Tato práce má za cíl poskytnutí obecných, ale i odborných informací o jazyku Swift a frameworku React Native. Dále podává informace o tom, jak se aplikace vytváří a jaké možnosti obě platformy nabízí. Poslední část práce je věnována srovnání obou zmíněných možností z hlediska zkušeností získaných během vývoje a dále výkonu aplikace.

Klíčová slova

Swift, React Native, iOS, Apple, JavaScript, mobilní aplikace

Annotation

The specifics of iOS application development using Swift and React Native technology

The topic of this bachelor thesis is the development of applications for operating system iOS, which is used on Apple mobile devices and then comparison of the two chosen development platforms, specifically the programming language Swift and the javascript framework React Native. In this thesis I describe both of these development methods and explore their advantages or disadvantages. The practical part focuses on performance evaluation of both platforms and analysis of a simple application created only for benchmark purposes. This thesis aims to provide general and expert information about the Swift language and the React Native framework. It also provides information on how the application is created and what options both platforms offer. The last part of my thesis focuses on the comparison of the two mentioned subjects from the point of view of experience during development of the application and it also includes informations about the performance of the created application.

Keywords:

Swift, React Native, iOS, Apple, JavaScript, mobile applications

OBSAH

1 ÚVOD

Vývoj mobilních aplikací je poslední roky velice žádanou záležitostí a tento trh se stále vyvíjí a rozrůstá. Podíl chytrých telefonů každým rokem značně stoupá a ze statistických dat jasně vyplývá, že tento trend bude pokračovat [1]. Na konci roku 2016 již došlo k tomu, že návštěvníci přistupují na webové stránky častěji z mobilních zařízení[1] než ze stolních počítačů či notebooků [2]. V roce 2016 byl překonán rekord[2] v počtu prodaných zařízení a bylo celkem prodáno 1,55 miliardy chytrých mobilů. Apple prodal celkem 215,4 milionů iPhonů (což byl pokles oproti roku 2015), to se rovná 14,5% všech zařízení (Samsung měl 20,8% podíl) [3]. Předešlé informace jasně implikují, že poptávka po programátorech mobilních aplikací bude aktuálně vysoká, a z toho také plyne obrovský počet mobilních aplikací (2 mil. aplikací v App Store[3]). Rozhodnutí začít vývoj jazyka Swift bylo pro Apple rozumnou volbou i z hlediska souboje s konkurencí. Jedinou možností vývoje aplikací pro systémy společnosti Apple byl jazyk Objective-C, který jak později zmiňuji, již nebyl dostačující v porovnání s modernějšími jazyky a představení jazyka Swift pomohlo nalákat spoustu nových vývojářů, kteří se nechtěli učit Objective-C. Nový a moderní jazyk Swift se pro ně stal impulsem k učení vývoje aplikací.

React Native, což je druhý „jazyk“, na který se v této práci zaměřuji, byl uveden také poměrně v nedávné době (2015). Jedná se o zcela rozdílnou architekturu oproti dvěma již zmiňovaným jazykům. Tato práce se nezaměřuje na vývoj pro více platforem (tzn. jedna verze aplikace pro iOS, Android), ale je vhodné to v souvislosti s React Native zmínit. React Native není programovací jazyk, ale jedná se o javascriptový framework, který je zpracováván pomocí nativní knihovny JavaScriptCore. Na rozdíl od jiných podobných technologií, které se zaměřují na alternativní vývoj aplikací (Cordova, PhoneGap), využívá React Native několik různých vláken, což ve výsledku zajišťuje uživateli plynulý chod grafického rozhraní a animací [4].

1.1 Cíl práce

Tato práce má za úkol přinést výsledek z hlediska porovnání výkonu dvou nových jazyků z oblasti vývoje pro mobilní zařízení. Zároveň práce přináší čtenáři úvod do historie obou

[1] Chytré telefony, tablety

[2] V roce 2015 bylo prodáno 1,44 miliardy chytrých mobilních telefonů.

[3] App Store je obchod s aplikacemi pro operační systém iOS a macOS firmy Apple.

jazyků, ukázku jejich syntaxe, popis vývojového prostředí, příklad grafických prvků a rozdíly v obou jazycích.

2 REŠERŠE

Jak programovací jazyk Swift, tak i framework React Native jsou relativně nové platformy, takže je to s materiály trochu složitější. V českém jazyce není možné najít literaturu vůbec, takže jsem využil pouze anglicky psané materiály.

2.1 Swift

Při uvedení jazyka Swift vydal Apple zároveň dvě knihy, které tento jazyk popisují. The Swift Programming Language, která detailně popisuje, jak celý jazyk funguje a z čeho se skládá [5]. Druhou knihou je Using Swift with Cocoa and Objective-C, která vysvětluje, jak používat jazyk Swift v kombinaci s Objective-C a frameworky Cocoa a Cocoa Touch [6]. Tyto knihy Apple pravidelně aktualizuje s každou novou verzí jazyka (aktuálně 3.1).

Během poměrně krátké doby, se začaly objevovat knihy, popisující možnosti jazyka Swift, ze kterých jsem si vybral hned několik a čerpal z nich. Jednou z nejznámějších webových stránek, zaměřujících se na programování pro systémy Apple, je raywenderlich.com. Kromě zveřejňování návodů a informací na webu, také vydali e-knihu zvanou Swift Apprentice – Beginning programming with Swift 3. Tato kniha nabízí více srozumitelnější přehled o konstrukci jazyka Swift oproti knize od firmy Apple [7].

Další knihou je Beginning iPhone Development with Swift 3. Jedná se o třetí verzi knihy od nakladatelství Apres, která vysvětluje Swift na konkrétních příkladech, počínaje jednoduššími až po komplexnější aplikace [8].

Z knihy Programming in Objective-C jsem čerpal informace o principech v objektovém programování, které jsou u Swiftu obdobné, jako pro jazyk Objective-C [9].

Big Nerd Ranch Guides píší knihy již od roku 2001, a to na různá témata. Kromě knihy zaměřené na všeobecný iOS vývoj, vydali i knihu věnovanou pouze jazyku Swift. Swift Programming: The Big Nerd Ranch Guide vysvětluje syntaxi a principy jazyku Swift. Poskytuje informace o tom, jak využívat Xcode a v závěru knihy jsou informace z knihy ukázané na tvorbě aplikace [10].

Od Apress jsem využil ještě další knihu, která detailně popisuje, jak testovat aplikace a jakým způsobem lze např. využívat nástroj Instruments [11].

2.2 React Native

React Native byl vydán o rok později než Swift a z hlediska změn a neustálého vývoje ho lze považovat za méně stabilní platformu. Vzhledem k tomu, že se každý měsíc vydává nová verze, která obsahuje i poměrně velké změny, je nejdůvěryhodnějším zdrojem oficiální webová dokumentace [12].

Pro úvod do knihovny React.js, ze které React Native vychází, jsem využil informace z knihy Developing React.js Edge, která vysvětluje, jak se tvoří webové aplikace pomocí React.js [13].

Další knihou věnující se React.js je kniha od nakladatelství Packt s názvem React.js Essentials. Kniha podává informace o tvoření uživatelských rozhraních, rozdíly mezi komponenty či např. význam Virtual DOM [14].

Od Packt jsem zvolil ještě jednu knihu. Konkrétně Getting Started with React Native, která nás učí, jak využívat React Native společně s Xcode a základy, které potřebujeme k tvoření jednoduchých aplikací [15].

Apress vytvořilo i knihu zabývající se React Native. React Native for iOS Development vysvětluje základy, jak samotný framework funguje. Následně je v knize ukázáno, jak lze vytvořit jednoduchou aplikaci pomocí různých komponent a poté i složitější koncepty typu komunikace se servery a databázemi [4].

3 PROGRAMOVACÍ JAZYK SWIFT

Tato kapitola poskytuje čtenáři krátký úvod do historie a vzniku jazyka Swift. Dále uvádí důležité základní datové typy a ukazuje, jak vypadá syntaxe tohoto jazyka. Představuje vývojové prostředí Xcode včetně jeho nástrojů a naznačuje, jakým způsobem jsou tvořeny aplikace a jaké elementy mohou obsahovat.

3.1 Historie jazyka

Apple představil jazyk *Swift* v roce 2014 na konferenci WWDC[4]. Jednalo se o zcela nový programovací jazyk, který má postupem času zcela nahradit starý a doposud používaný jazyk Objective-C. Počátky tohoto jazyka se datují k roku 1984 a firmě Stepstone, od které byl následně odkoupen společností NeXT[5]. Apple začal používat Objective-C, kdy koupil právě NeXT a Steve Jobs se vrátil zpět do vedení Apple. V roce 2006 bylo vydáno Objective-C 2.0, které přineslo např. lepší garbage collector[6] či vylepšení syntaxe tečkovou konvencí.

Objective-C se však ani tímto nestalo příliš atraktivním jazykem, protože různé jiné jazyky byly jednodušší na porozumění, což pro začínající vývojáře mohlo být odrazující a raději zvolili „uživatelsky“ přívětivější jazyk (např. JavaScript, Python, PHP, Ruby a další) [10]. V roce 2010 začal tajně vyvíjet nový jazyk pod vedením Chrise Lattnera, který v minulosti spolupracoval na LLVM kompilátoru. Swift byl poprvé zveřejněn pouze jako beta verze, ale již v září 2014 dosáhl označení 1.0 a nyní poslední verze nese označení 3.1 [7]. Hlavní výhodou Objective-C byla jeho rychlost, což je předností C-jazyků [8], ale jelikož je Swift zkompilován do stejného strojového kódu, tak je jeho výkon identický a ke své rychlosti ještě přidává srozumitelnější syntaxi a chytřejší kompilátor, který omezuje možnost potencionálních chyb a snižuje bezpečnostní rizika aplikací. Poslední důležitou poznámkou je, že v prosinci roku 2015 Apple zveřejnil Swift jako open-source [16] a tím umožnuje rychlý růst a vylepšování jazyka širokou komunitou. Je možné, že právě díky tomu, se Swift umístil tak vysoko v anketě (viz obrázek 1)

[4] Worldwide Developers Conferece je každoroční týdenní vývojářská konference pořádaná společností Apple.

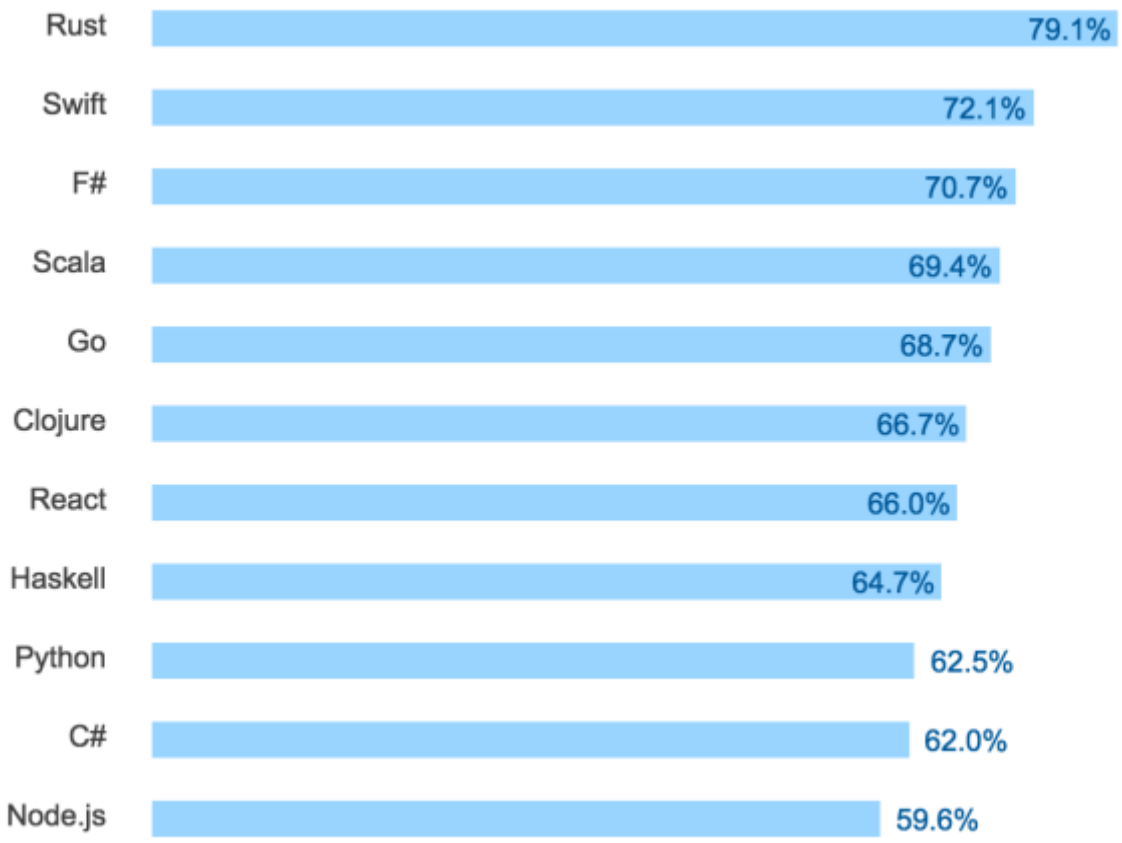
[5] NeXT je společnost založena (1985) Stevem Jobsem po odchodu ze společnosti Apple.

[6] Způsob automatické správy paměti.

[7] https://swift.org/download/ - releases

[8] Mezi tyto jazyky se řadí C, C++ a Objective-C.

populárního fóra stackoverflow.com. Anketa zobrazuje procento vývojářů, kteří aktivně využívají daný jazyk a jsou s ním spokojeni.



Obrázek 1: *Anketa zobrazující oblíbenost jazyků. Zdroj: StackOverflow, 2016*

3.2 Základní informace o jazyku Swift

Swift je vysokoúrovňový objektově orientovaný programovací jazyk vyvinut speciálně pro vývoj na platformách macOS a iOS. Je založen na struktuře jazyků C a Objective-C a snaží se adoptovat to nejlepší z těchto jazyků, ale bez omezení kompatibility, které s sebou nesl jazyk C. Designéři jazyka Swift (viz tabulka 1) se nechali inspirovat i u dalších jazyků jako např. Rust, Haskell, Ruby, Python, C#, CLU, ze kterých převzali moderní koncepty [17].

Funkce Swiftu	**Podobný programovací jazyk**
Closures	JavaScript
Type Inference	Haskell, C#
Tuples	Python

Funkce jako objekty	JavaScript, Python
Přetěžování operátorů	C++
Pattern Matching	Haskell, Scala
Automatic Reference Counting	Objective-C
Optional Types	Rust, Dart, Java, Hakell

Tabulka 1: *Vlastnosti jazyka Swift* [18]

Některé funkce však jazyku Swift chybí. Mezi tyto funkce patří např. absence nativní podpory pro zachytávání výjimek pomocí metod `try / catch`. Další funkcí, kterou lze považovat za chybějící, je nativní řešení více vláknových aplikací, které ale může být částečně vyřešeno pomocí frameworků Cocoa a Cocoa Touch.

Jelikož Swift využívá stejné běhové prostředí (angl. runtime), tak zůstala zachována i kompatibilita se starými verzemi systémů iOS a macOS (verze staré až 4 roky). Další výhodou je bezproblémová možnost využívání existujících sad frameworků Cocoa a Cocoa Touch, které jsou napsány v Objective-C. Důležitým faktorem této kompatibility je také použití stejného kompilátoru LLVM[9]. Ve výsledku je tedy možné využít v programu různé jazyky (C, Objective-C, C++ a Swift). Pro správu paměti používá Swift (obdobně jako Objective-C) tzv. automatické počítání referencí (Automatic Reference Counting)[10], což značně omezuje výskyt chyb způsobených programátorem, které souvisí s managementem paměti. ARC automaticky uvolňuje objekty v paměti, jakmile již nejsou používané. Na konci roku 2015 byl Swift zveřejněn jako open source[11] a společně s tímto byla uvedena i adaptace jazyka pro platformu Linux [19].

[9] LLVM kompilátor byl vytvořen Chrisem Lattnerem, který designoval Swift.

[10] Není však využíváno např. pro kód napsaný v jazyku C, či pro grafickou knihovnu Core Graphics.

[11]Zdrojový kód je veřejně přístupný.

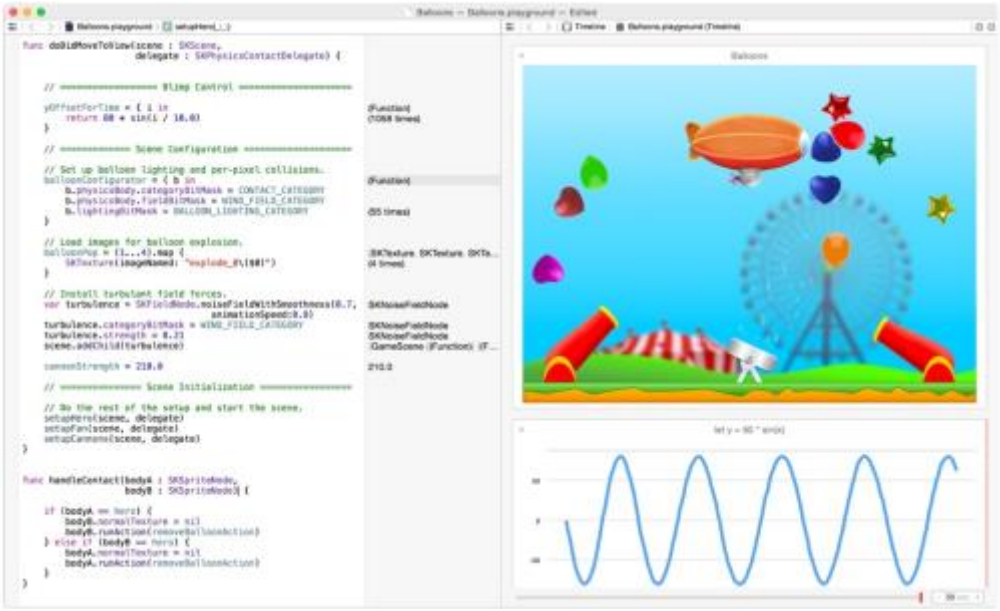
3.3 Vývojové prostředí

3.3.1 Xcode

Xcode je oficiální IDE (Integrated Development Environment) pro systém macOS[12] od společnosti Apple a umožňuje vývoj aplikací pro systémy macOS, iOS, watchOS a tvOS. Je možné si ho zdarma stáhnout z App Store. První verze se objevila v roce 2003 a aktuální verze nese označení 8.3.1 (duben 2017). Swift se poprvé objevil v beta verzi Xcode 6. Pro vývoj je možné využívat i jiné programy (např. JetBrains Appcode, CodeRunner 2), ale jsou spíše minoritní. Xcode používá pro kompilaci LLVM kompilátor a k němu speciální debugger LLDB, který je vyvíjen společně s LLVM a přístupný pomocí UI (user interface – uživatelské prostředí).

3.3.2 Playground

Zajímavou novinkou představenou v Xcode 6 jsou tzv. hřiště (angl. playground), která umožňují vývojářům experimentovat s jazykem Swift v reálném čase. Hřiště se dělí do dvou částí. Na levé straně se nachází editor kódu a na pravé straně se zobrazuje výsledek, jak můžeme vidět na obrázku 2. Velkou výhodou je, že není nutné procházet celým cyklem kompilace / spuštění / ladění aplikace, ale úpravy se zobrazí okamžitě [10].



Obrázek 2: *Ukázka hřiště v jazyku Swift a programu Xcode*

[12] Předchozí verze označovány jako Mac OS X, následně pouze OS X.

3.3.3 iOS Simulator

Od minulého roku (2016) Xcode umožňuje bezplatné testování aplikací na fyzickém zařízení, pokud vlastníte iPhone či iPad. Dříve bylo nutné mít zaplacený Apple Developer Program[13]. V případě absence fyzického zařízení je možné využít simulátor obsažený v Xcode, který nám dovoluje vybrat si typ zařízení (i konkrétní verzi systému) a testovat na něm. Po zkompilování kódu je nová verze aplikace vždy automaticky nahrána a nainstalována do běžícího simulátoru a následně spuštěna.

3.3.4 Interface Builder

Jedná se o nástroj přímo integrovaný v Xcode, který slouží k tvorbě grafického rozhraní. Toto rozhraní lze tvořit i pomocí ručně psaného kódu, interface builder (IB) nám tuto práci však značně usnadňuje. Formát, ve kterých IB ukládá svá rozložení, se již několikrát změnil. Posledním formátem je nyní tzv. Storyboard (.storyboard). Aplikace obsahují několik různých oken, která spolu spolupracují. Vztahy mezi těmito okny jsou definované právě ve Storyboards a přehledně nám naznačují, jak celá aplikace funguje.

3.3.5 AutoLayout

Jelikož mají zařízení různé velikosti displejů, tak je nutné přizpůsobovat jednotlivé elementy (tlačítka, textboxy atp.) tak, aby se chovaly identicky. Slouží k tomu tzv. omezení (angl. constraints), pomocí kterých se u elementů definuje to, jak se mají chovat při jeho změně, tedy např. při rotaci zařízení. Příklad omezení může být např. horizontální / vertikální centrování daného elementu či jeho vzdálenost od okraje, které se následně přizpůsobí (matematicky přepočítá) podle velikosti displeje [20–22].

3.3.6 Instruments

Důležitou součástí vývoje aplikací je i analýza a test jejich výkonu. Instruments, které jsou součástí Xcode, nám umožňují sledovat běžící procesy a následně analyzovat a vyhodnotit tato nashromážděná data. Lze zkoumat např. zatížení procesoru, využití paměti, či jak moc aplikace

[13] Vývojáři zde mají např. přístup k beta verzím systémů. Bez tohoto zakoupeného programu ($99/rok) není možné distribuovat vytvořené aplikace.

vybíjí baterii. Tento nástroj bude zdrojem dat pro porovnání obou testovaných způsobů vývoje v této bakalářské práci [11, 23].

3.4 Charakteristika jazyka Swift

3.4.1 Objektově orientované programování

Swift řadíme mezi jazyky objektově orientované. Objektově orientované programování (OOP) je programovací paradigma[14], jehož základním pilířem je vytváření malých a relativně samostatných objektů, které poté opakovaně využíváme. Mezi další základní koncepty OOP patří:

- **Třída** – množina objektů s určitými vlastnostmi. Definuje, jaké vlastnosti bude mít objekt této třídy, který nazýváme instancí třídy,
- **Dědičnost** – možnost zdědit atributy a chování od existujících tříd tzv. rodičovských tříd. Potomek může následně toto zděděné chování rozšířit o své vlastní atributy, metody,
- **Viditelnost** – pomocí modifikátorů vlastností je možné omezit, k jakým atributům v dané třídě bude možné přistupovat z jiné části aplikace. Existují tři modifikátory:
    - private – přístupné pouze z metod této třídy
    - protected – přístupné z metod této třídy a případných potomků
    - public – přístupné odkudkoliv
- **Zapouzdření** – jedná se o mechanismus, který zabezpečuje objekty a jejich data tak, že nejsou přístupné jiným objektům a ty je tedy nemohou upravovat. Toto zabezpečení je provedeno právě pomocí výše zmíněné viditelnosti [9].

3.4.2 Syntaxe jazyka Swift

Swift vychází hlavně z jazyků C a Objective-C, z čehož vyplývá jejich velká vzájemná podobnost. Jako konkrétní příklady lze zmínit např. bloky uzavřené složenými závorkami { }, dvojité lomeno jako jednořádkový komentář, importování knihoven atd. Není zde nutnost funkce `main()` (vyžadována v jazyku Objective-C) a také není nutné psát středník po každém příkazu (je možné je zanechat pro lepší přehlednost). Swift rozděluje proměnné a konstanty a

[14] Paradigma je styl, jakým v daném jazyku programujeme (pokud to daný jazyk podporuje).

je vhodné je správně využívat, protože konstanty šetří místo v paměti. Pro deklaraci proměnné se používá klíčové slovo var a pro konstantu použijeme slovo let. Je doporučeno používat způsob camelCase[15] pro pojmenovávání proměnných, tříd či metod.

Swift obsahuje své vlastní verze datových typů (uvedeno v tabulce 2), které lze najít v jazycích C a Objective-C a dále obsahuje ještě speciální typy jako např. Tuples, který umožňuje ukládání více hodnot různých datových typů do jedné proměnné. Datové typy není vždy nutno explicitně určovat, protože jsou doplněny automaticky podle obsahu proměnné [5].

Datový typ	**Rozsah**	**Popis**
Int, UInt	Podle platformy – 32bit nebo 64bit (+-2,147,483,648)	Celočíselné číslo s rozdělením buď signed(Int) nebo unsigned (UInt)
Float, Double	32bit, 64bit	Číslo s desetinnou čátkou
Bool	True, nebo false	Nelze používat číselné hodnoty 0 a 1
Character	Unicode	Jedná se o jeden znak (character)
String	Unicode	Řetězec znaků. Možnost používání emotikonů.

Tabulka 2: *Základní datové typy jazyka Swift*

3.4.2.1 Výčtový typ, struktura, třída

Výčtový typ (angl. enum) je poměrně jednoduchý datový typ, který je tvořen omezenou množinou konstantních hodnot. Enum se obvykle používá pro výčet měsíců, dnů, světových stran, atp. Povolené datové typy jsou Integer, String, Float a Boolean.

Struktury (Struct) a třídy (Class) jsou velice podobné. Obě umožňují ukládání dat pomocí atributů, definici vlastních metod, mohou být rozšířené pomocí extensions či protokolů a např. definovat inicializaci. Třídy dále nabízí dědičnost, přetypování a možnost definování de-

[15] Způsob psaní víceslovných frází, v nichž jednotlivá slova začínají velkým písmenem a nejsou oddělená mezerou.

inicializace. Struktury se obvykle používají většinou na jednoduchá data. Příkladem může být souřadnicový systém [5].

3.4.2.2 Rozšíření (angl. extensions)

Pomocí rozšíření můžeme upravovat či přidávat novou funkcionalitu k existujícím třídám, strukturám a protokolům. Obvyklým použitím je rozšíření třídy UIColor tak, aby podporovala vstupní data v hexadecimálním formátu, jak je ukázáno ve zdrojovém kódu 1.

```
extension UIColor
{
    class func uicolorFromHex(_ rgbValue:UInt32, alpha : CGFloat)->UIColor

    {
        let red = CGFloat((rgbValue & 0xFF0000) >> 16) / 255.0
        let green = CGFloat((rgbValue & 0xFF00) >> 8) / 255.0
        let blue = CGFloat(rgbValue & 0xFF) / 255.0
        return UIColor(red:red, green:green, blue:blue, alpha: alpha)
    }
}
```

Zdrojový kód 1: *Ukázka rozšíření třídy UIColor*

3.4.2.3 Protokoly

Další možností, kterou lze definovat chování třídy či struktury, jsou tzv. protokoly (angl. protocols). Protokoly předepisují, jaké funkce či atributy má daný prvek (třída, struktura, enum) obsahovat a využívat. Lze si pod nimi představit jakousi šablonu [7]. Jako příklad mohu uvést protokol FBSDKLoginButtonDelegate z knihovny Facebook SDK, který třídě předepisuje tři funkce, které je povinna implementovat.

```
class FBLoginViewController: UIViewController, FBSDKLoginButtonDelegate
```

Ukázka třídy, která je potomkem třídy UIViewController a využívá protokol z knihovny Facebook SDK.

3.5 Struktura aplikace

Pomocí programu Xcode je možné vytvořit novou aplikaci v programovacím jazyku Swift. Následně máme možnost začít tvořit naši aplikaci, která může obsahovat spoustu různých elementů, ze kterých se bude skládat. V této kapitole uvádím několik nejdůležitějších nástrojů a objektů, které se při tvorbě aplikací využívají.

3.5.1 CocoaPods

Aplikace mohou také využívat různé knihovny třetích stran. Pro správu těchto knihoven existuje nástroj zvaný CocoaPods, který usnadňuje práci s těmito rozšířeními. CocoaPods nabízí přes 30 000 knihoven[16], které je možné použít. V roce 2011 byl tento nástroj zveřejněn pro jazyk Objective-C a je možné ho používat i s novějším jazykem Swift. CocoaPods nabízí stažení námi zvolených balíčků a jejich následnou instalaci do projektu. Ovládá se primárně pomocí terminálu (příkazový řádek) a k běhu využívá programovací jazyk Ruby. Apple aktuálně vyvíjí svůj vlastní nástroj pro správu knihoven zvaný Swift Package Manager, ale zatím nepodporuje iOS[17].

Samotná specifikace požadovaných balíčků je uložena v textovém souboru `Podfile` (viz zdrojový kód 2) ve složce projektu. Soubor lze vytvořit pomocí příkazu `pod init` a poté do něho můžeme zapsat požadované knihovny. Dalším příkazem `pod install` se knihovny stáhnou (z úložiště Github), nainstalují se do „workspace“ (pracovní prostředí daného projektu) a vytvoří nový soubor, kterým otevíráme náš projekt (`<JmenoProjektu>.xcworkspace`). U jednotlivých knihoven je možné specifikovat i jakou verzi žádáme, abychom se případně mohli vyhnout nekompatibilitě s novou verzí. Pojmenování verzí funguje na principu sémantického verzování (viz obrázek 8). Pro aktualizaci knihoven se využívá příkaz `pod update`.

```
target 'SwiftThesisApp' do
  use_frameworks!

  # Pods for SwiftThesisApp
  pod 'FBSDKCoreKit'
  pod 'FBSDKShareKit'
  pod 'FBSDKLoginKit'
  pod 'GoogleMaps'

end
```

Zdrojový kód 2: *Ukázka souboru Podfile*

3.5.2 Frameworks

Knihovny zmiňované výše můžeme nazývat i frameworky. Pokud je autorem třetí strana, jsou spravovány pomocí CocoaPods. Apple však sám nabízí spoustu frameworků, které je možné

[16] https://cocoapods.org/

[17] https://github.com/apple/swift-package-manager/

použít nativně bez nutnosti dodatečné instalace. U iOS rozlišujeme dva různé typy: veřejné a interní frameworky. Interní frameworky využívají pouze aplikace vytvořené společností Apple. Příkladem veřejného frameworku, který muže naopak využít ve své aplikaci kdokoli, je např. UIKit, který má na starosti v podstatě celé uživatelské rozhraní. Dalším příkladem jsou:

- Foundation – práce s kolekcemi dat (pole, listy,…), Stringy, URL, vlákna, atp.,
- HealthKit – přístup k informacím o zdraví (např. počet kroků za den) uživatele telefonu,
- MapKit – zobrazení a práce s mapami,
- MetalKit – přístup k GPU. Využíván hlavně pro vývoj her (načítání textur, renderování).

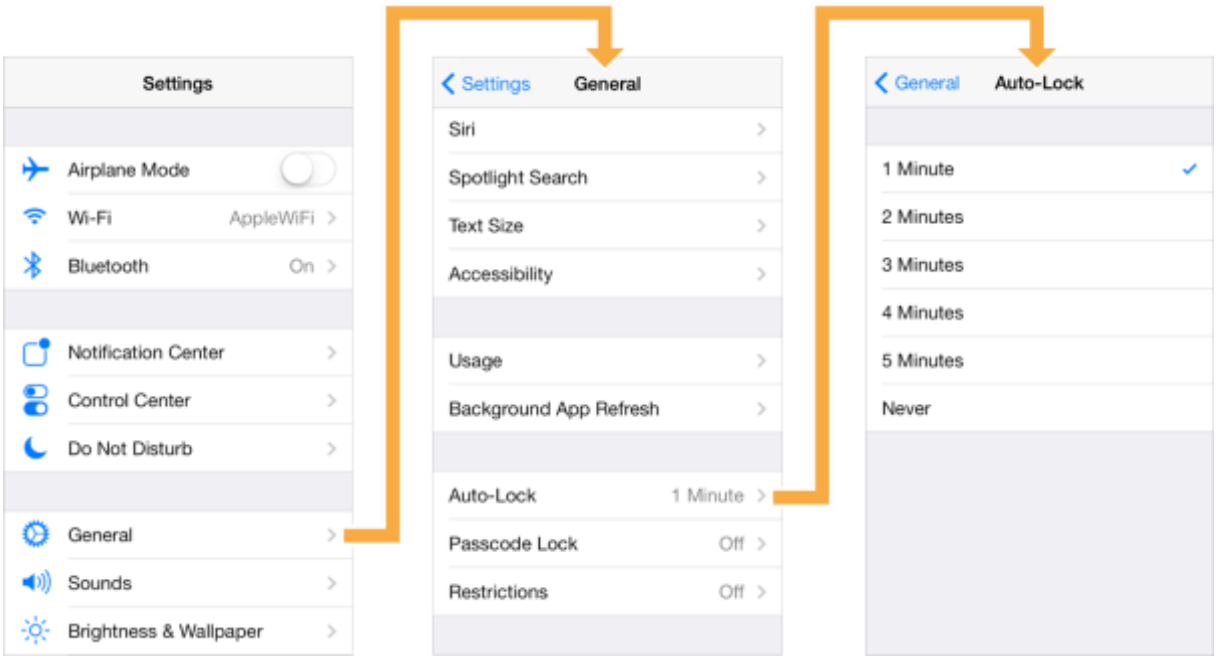
3.5.3 Uživatelské rozhraní

Jak již bylo zmíněno v 3.3.4 uživatelské rozhraní je možné tvořit pomocí nástroje Interface Builder a tzv. Storyboards. Tyto Storyboards nám umožňují propojit uživatelské rozhraní s kódem aplikace. Samotné rozhraní je tvořeno pomocí dalších objektů, které se vkládají do jednotlivých Storyboard a tím vytváří vzhled naší aplikaci a propojují model s částmi controller a view v architektuře MVC[18].

3.5.3.1 ViewController

ViewController je základní částí iOS aplikace. Každá aplikace obsahuje alespoň jeden či více těchto controllerů. Zmíněný controller poskytuje vrstvu mezi uživatelským rozhraním (View) a datovým modelem (Model) a řeší jejich společnou komunikaci pomocí tzv. vývodů (angl. outlet) [8]. Je součástí frameworku UIKit a jedná se o potomka třídy UIViewController. Dále existují tzv. kontejnery (angl. container) těchto controllerů. Mezi tyto kontejnery se řadí UINavigationController, UITabBarController či UISplitViewController. Kontejnery obsluhují navigaci mezi dalšími ViewControllery, což je zřetelné z obrázku 3. Poskytují také možnost animací při přechodech.

[18] Architektura, která dělí aplikaci na datový model (Model), uživatelské rozhraní (View) a řídící logiku (Controller).



Obrázek 3: *Příklad navigace s použitím UINavigationController. Zdroj: Apple Developer*

Propojení mezi grafickými prvky ve Storyboard a jejich definicí uvnitř třídy (ViewController) zajišťuje výše zmíněný outlet. Vytváří se v režimu editoru Assistant, který umožňuje zobrazit dvě okna vedle sebe (v tomto případě Storyboard a třídu, ve které chceme prvek používat). Pomocí stisknutí a držení tlačítka ctrl, kliknutí na daný prvek myší a přetažením do třídy, se vytvoří propojení. Je možné vybrat, zda se jedná o Outlet či Action [8]. Příklad definice vypadá takto:

```
@IBOutlet var mapView: GMSMapView!
```

Další možností komunikace s uživatelským rozhraním jsou akce (angl. Actions). Jedná se v podstatě o metody, které reagují na interakci s uživatelským rozhraním. Při vytváření propojení (viz výše), je možné určit, na jakou konkrétní událost má vytvořená metoda akce reagovat. Příkladem může být stisknutí tlačítka. Název metody je na nás.

```
@IBAction func buttonTapped(sender: UIButton) {}
```

3.5.3.2 Knihovna prvků uživatelského rozhraní

Knihovna objektů v Xcode obsahuje spoustu grafických prvků, které můžeme použít. Stačí je pouze přetáhnout myší do Storyboard. Příkladem jednoduchých grafických elementů mohou být tlačítka (Button, Switch), textová pole (Text Field, Label), či např. posuvníky (Slider). Dále v knihovně nalezneme např. rozpoznávače různých dotykových gest (Gesture Recognizer), která lze přidat do ViewControlleru a důležité jsou i prvky zajišťující navigaci (Navigation Bar,

Navigation Item). Mezi často používané prvky se řadí objekty, které mají na starost zobrazování dat. Pro práci s daty knihovna obsahuje:

- TableView – zobrazuje data v listu pod sebou a jedná se o nejpoužívanější možnost zobrazení dat. Nalezneme ho v téměř každé aplikaci od Apple (hudba, hodiny, nastavení, atp.). Skládá se z buněk třídy UITableViewCell, pro které můžeme definovat, jak budou vypadat a jak se budou chovat např. při uživatelské interakci,
- CollectionView – jedná se o kolekce buněk, které jsou většinou zobrazené jako čtverce horizontálně vedle sebe a následně se mohou opakovat v řádcích. Příkladem je galerie obrázků [8].

Dalším příkladem grafických prvků je Map Kit View, který umožňuje zobrazení mapy, Image View poskytující možnost zobrazení obrázku, Web View představující vrstvu webového prohlížeče a jako poslední příklad uvedu prvek Scroll View, který umožňuje práci s objekty, které by se jinak nevešly na displej, a to pomocí „scrollování“.

4 JAVASCRIPTOVÝ FRAMEWORK REACT NATIVE

Před tím, než mohu začít psát o React Native, je vhodné zmínit krátký úvod do programovacího jazyka JavaScript a následně přiblížit knihovnu React.js, ze které teprve samotný framework React Native vznikl.

4.1 JavaScript

Jedná se o objektově orientovaný skriptovací jazyk, který vznikl v roce 1995. Je jednou ze tří[19] nejdůležitějších webových technologií. JavaScript je na rozdíl od Swiftu jazyk interpretovaný, tzn. že není pro každé spuštění aplikace nutná zdlouhavá kompilace, ale program je přímo interpretován do strojového kódu. JavaScript je v současné době nejoblíbenější programovací jazyk[20] a existuje pro něj obrovské množství různých knihoven a frameworků (jQuery, Angular, Vue.js, React).

4.1.1 JSX

Jedná se o rozšíření JavaScriptu o možnost zápisu XML (HTML) značek bez užití apostrofů. JSX provádí optimalizaci při kompilaci do JavaScriptu a ve výsledku běží rychleji. Oproti JavaScriptu se jedná o jazyk staticky psaný jazyk, z čehož vyplývá, že většina chyb bude zachycena už během kompilace. Při tvorbě React Native aplikací je doporučeným způsobem zápisu šablon, byť je možné využívat i čistý JavaScript [13, 24].

4.2 React.js

React.js je open-source knihovna pro jazyk JavaScript a je využívána hlavně pro tvoření komponent na webových stránkách, u kterých se dynamicky mění data bez nutnosti obnovení stránky. Pokud bychom měli zařadit React do architektury MVC, tak React představuje „V“ neboli View. V roce 2013 byl představen Facebookem a od té doby stále roste díky komunitě vývojářů, kteří aktivně přispívají k vývoji. Framework je použit na stránkách jako je Netflix, Facebook, Airbnb, Feedly a další [4]. Výhodou používání jednotlivých komponent je možnost úprav a aktualizací pouze dané části. Není nutné měnit celou aplikaci. React.js využívá

[19] HTML, CSS a JavaScript jsou v dnešní době jádrem každé internetové stránky

[20] https://stackoverflow.com/insights/survey/2017

ke svému běhu a renderování většinou prohlížeč klienta, ale není to vždy pravidlem. Renderování HTML na straně serveru s sebou nese různé výhody (např. SEO optimalizace či rychlost) a proto je možné v Reactu narazit na kombinaci obou možností. Výjimkou není ani použití Reactu v kombinaci s jiným frameworkem jako je např. AngularJS. React se využívá i s dalšími různými knihovnami (ReduxJS, Flux), které zprostředkovávají data (Model, Controller), která jsou následně zpracována a zobrazena Reactem (View).

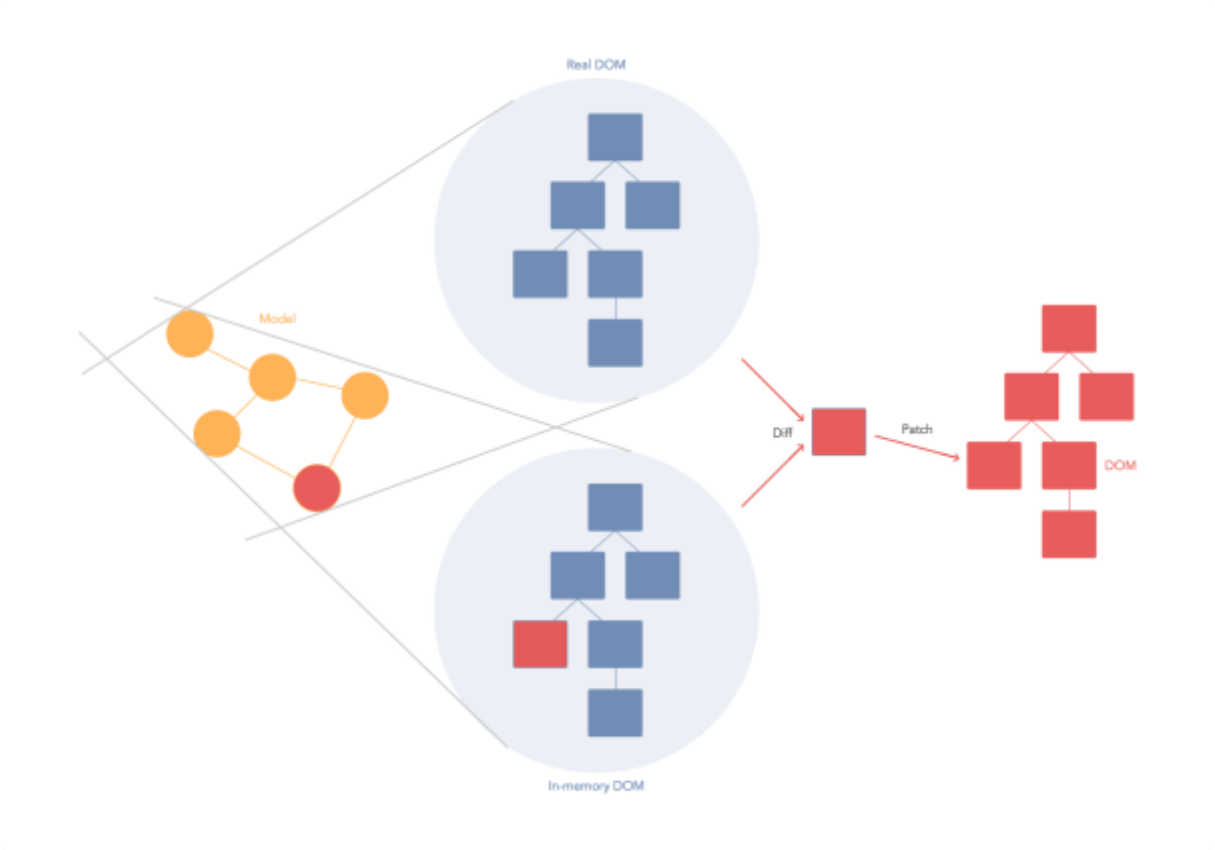
React byl vyvíjen s důrazem na User Experience (uživatelský prožitek) a snaží se o to, aby tato zkušenost byla podobná jako s desktopovými aplikacemi (např. překreslování pouze jednotlivých komponent pomocí DOM[21] a Virtual DOM). React však přináší i nový koncept z hlediska pohledu na vývoj aplikace. Před začátkem vývoje je vhodné zanalyzovat strukturu a části tvořené aplikace a následně vhodně určit jednotlivé komponenty, které bude nutné vytvořit či použít a tím si usnadnit vývoj [13].

4.2.1 Virtual DOM

Úprava webového DOM je jednou z nejnáročnějších operací internetových stránek. Pro vyřešení tohoto problému využívá React koncept zvaný Virtual DOM. Princip je takový, že si React zkopíruje stávající DOM jako virtuální (VDOM) a uloží si ho do paměti. Pokud se data v aplikaci změní (zajištěno pomocí state u každé komponenty), je vytvořen další virtuální dokument a je porovnán s předešlým. Samotné porovnání probíhá pomocí algoritmu React Diff a výsledkem je tzv. patch, který říká původnímu dokumentu, jaké komponenty má aktualizovat (naznačeno na obrázku 4) [14].

Tato funkce je nezbytná součást Reactu a umožňuje mu být velice rychlý. A rychlost je právě to, co React proslavilo a dává mu možnost použití pro velké a náročné aplikace (např. New York Times, Dropbox, Uber…). Rychlost Reactu se příliš neprojeví na menších aplikacích, které pracují s malým objemem dat a komponent, ale u komplexních aplikací je rychlost a odezva velice důležitá [4].

[21] Document Object Model je stromová struktura, kterou si prohlížeč vytvoří, když načte webovou stránku



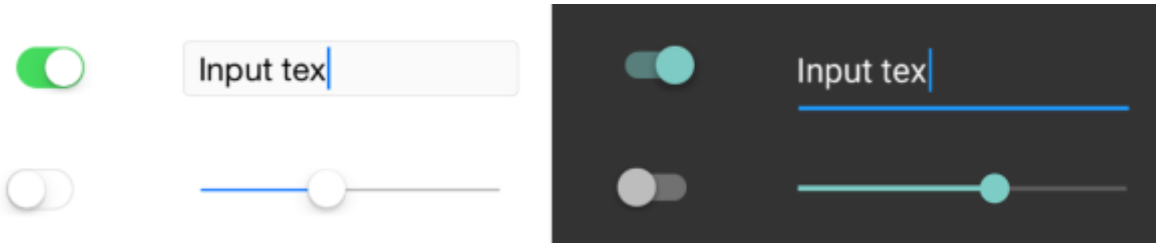
Obrázek 4: *Vizualizace Virtual DOM. Zdroj: StackOverflow.com, 2014*

4.3 React Native

Facebook představil framework React Native v roce 2015 a měl být revolucí ve vývoji aplikací pro mobilní zařízení. Zpočátku byl podporován pouze systém iOS [25], ale ten stejný rok byla přidána i podpora pro operační systém Android.

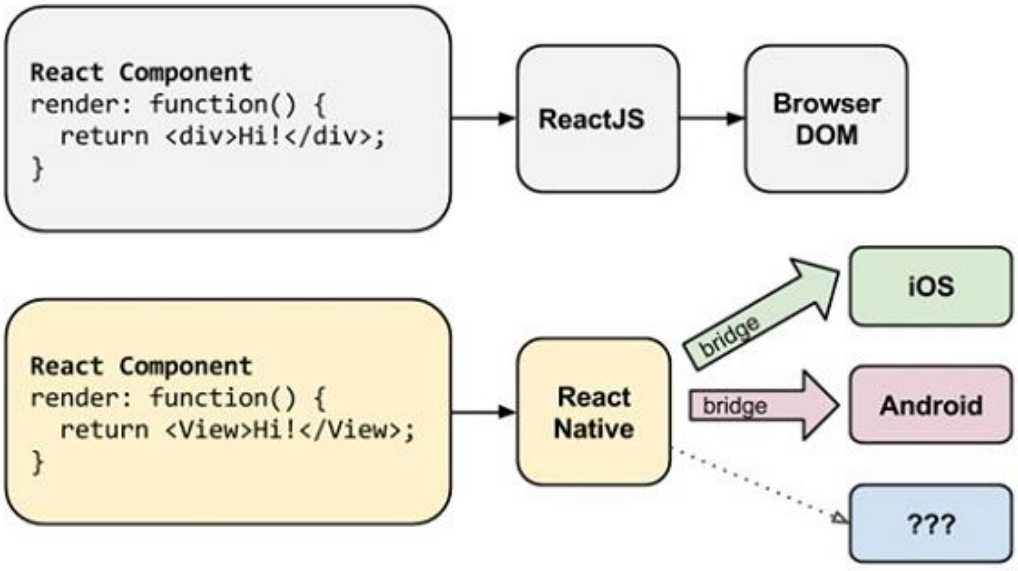
Hlavním účelem tohoto frameworku je možnost použití jedné verze aplikace pro dvě platformy, což je finančně i časově méně náročné. Pokud byla do té doby vyžadována aplikace pro oba zmíněné mobilní systémy, bylo nutné vytvářet dvě samostatné aplikace, případně existovala možnost použití tzv. hybridních frameworků (PhoneGap, Cordova, Ionic), u kterých je však nevýhodou, že komponenty nejsou tvořeny pomocí nativního kódu. Frameworky sice využívají přístup do nativního API, ve výsledku jsou však elementy tvořeny pomocí HTML a JavaScriptu, což je následně vloženo do nativní komponenty WebView, která se využívá pro zobrazování webových stránek. Tento přístup sebou nese nevýhody pro uživatele, které se mohou projevit při používání aplikace např. neplynulé pohyby v aplikaci, špatné chování klávesnice či navigace [12, 15]. React Native tyto problémy řeší tak, že ke svému běhu využívá

několik vláken. Ze čtyř vláken zmíním dvě, která jsou nejdůležitější. Jedná se o javascriptové vlákno a hlavní nativní vlákno, které má na starost běh grafického rozhraní. Výhodou použití více vláken je, že hlavní vlákno není vytížené, když aplikace zpracovává nějaký požadavek a tím je zajištěno zachování plynulosti grafického rozhraní. Aplikační logika je zpracována pomocí javascriptového vlákna, a to následně asynchronně předává data hlavnímu nativnímu vláknu, které na změny reaguje například překreslením grafických prvků.



Obrázek 5: *Příklad rozdílného vzhledu v systému iOS a Android* [26]

Rozdíly v aplikacích mezi systémy iOS a Android jsou z hlediska uživatelského prostředí pouze ve výsledném vzhledu, jak můžeme vidět na obrázku 5. Každý systém používá jiný vzhled pro podobné typy komponent. A toto je hlavní výhoda frameworku React Native. Ze stejného kódu vygeneruje dle dané platformy její nativní komponentu, která má identický vzhled a chová se tak, jakoby byla aplikace napsána např. v jazyku Swift či Java (Android) [15]. Obrázek 6 ukazuje, jakým procesem prochází komponenta před tím, než je vykreslena.



Obrázek 6: *Proces vykreslení komponenty v React a React Native* [27]

React Native je postaven na knihovně React a využívá výhody, které knihovna nabízí. Jako příklad lze uvést okamžité zpracování kódu a jeho případných úprav bez nutnosti zdlouhavé kompilace, tzv. hot-reloading [25].

4.4 Vývojové prostředí

4.4.1 NPM

Aplikace ve frameworku React Native využívá spoustu různých javascriptových modulů a knihoven. K jejich správě slouží NPM neboli Node Package Manager, což je správce javascriptových modulů, který nám umožňuje provázaní všech těchto rozšíření dohromady a zajišťuje jejich společné fungování. Nabízí nám stažení požadovaných balíčků, jejich případnou aktualizaci a samotné spuštění těchto balíčků pomocí jejich spouštěcího skriptu. NPM také zjednodušuje případnou spolupráci v týmu, protože si všichni jeho členové pomocí jednoho příkazu (npm install) mohou stáhnout všechny projektem vyžadované balíčky, které jsou definované v souboru package.json.

```
├─┬ react@16.0.0-alpha.6
│ ├─┬ fbjs@0.8.12
│ │ ├── core-js@1.2.7
│ │ ├── isomorphic-fetch@2.2.1
│ │ ├── setimmediate@1.0.5
│ │ └── ua-parser-js@0.7.12
│ ├─┬ loose-envify@1.3.1
│ │ └── js-tokens@3.0.1
│ └── object-assign@4.1.1
├─┬ react-native@0.43.3
│ ├── absolute-path@0.0.0
│ ├── art@0.10.1
│ ├── async@2.3.0
│ ├─┬ babel-generator@6.24.1
│ │ ├─┬ detect-indent@4.0.0
│ │ │ └─┬ repeating@2.0.1
│ │ │   └── is-finite@1.0.2
│ │ ├── jsesc@1.3.0
│ │ └── trim-right@1.0.1
```

```
├─┬ react-native-facebook-login@1.4.0
│ └── itypeof@1.1.2
├── react-native-maps@0.14.0
├─┬ react-native-navigation@2.0.0-legacy.12
│ ├─┬ react-redux@5.0.4
│ │ ├── create-react-class@15.5.2
│ │ ├── hoist-non-react-statics@1.2.0
│ │ ├── lodash-es@4.17.4
│ │ └── prop-types@15.5.6
│ └── redux@^2.0.0 || ^3.0.0
├─┬ react-native-video@1.0.0
│ └── keymirror@0.1.1
└── react-test-renderer@16.0.0-alpha.6
```

Zdrojový kód 3: *Ukázka stromu dependencies*

Package.json obsahuje informace jako jsou např. název nebo verze aplikace. Důležitou položkou je cesta ke skriptu, který se spustí jako první. Zmiňované balíčky jsou definované jako závislosti (angl. dependencies) a zápis těchto jednotlivých balíčků vypadá následovně: "react-native": "0.43.3" (viz zdrojový kód 3). Verze je velice důležitou informací, protože balíčky se často aktualizují a ne vždy je vhodné použít nejnovější verzi, protože může způsobit chyby v naší aplikaci. K tomu se využívá tzv. semantic versioning zobrazené na obrázku 7, které popisuje,

co znamenají jednotlivé číslice v označení verze. Je tedy možné např. omezit aktualizace balíčků pouze na méně důležité verze (Minor), které jsou zpětně kompatibilní.

Obrázek 7: *Příklad označení verzí pomocí Semantic Versioning. Zdroj: JDriven, 2016*

4.4.2 Editor (IDE)

Jelikož je React Native psán v JavaScriptu, je možné využít v podstatě jakýkoliv textový editor ze kterých mohu zmínit např. Sublime Text, Atom či Visual Studio Code. Je však nutné mít nainstalované prostředí Xcode včetně command-line tools, pomocí kterého se následně kompiluje výsledná aplikace a je spuštěn iOS Simulátor popsaný v sekci 3.3.3.

Já jsem pro vývoj projektu zvolil editor Atom společně s balíčkem Nuclide, což je rozšíření vytvořené společností Facebook právě pro vývoj pomocí frameworku React Native. Přidává do editoru Atom spoustu vylepšení a funkcí[22], které usnadňují vývoj. Příkladem je podpora našeptávání kódu, přehlednější stavová lišta a důležité nástroje pro debugging, které se automaticky připojí k simulátoru či reálnému zařízení.

4.5 Syntaxe a struktura aplikace

Jak již bylo zmíněno, aplikace využívá nástroje NPM a pomocí něho se i nová aplikace vytváří. Pro obsluhu NPM využíváme terminál (příkazový řádek). Samotný manažer NPM se může nainstalovat např. pomocí nástroje Homebrew, který má na starosti balíčky v systému macOS. Instalace a vytvoření nového projektu je ukázáno ve zdrojovém kódu 4.

[22] https://nuclide.io/docs/editor/basics/

```
$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
$ brew install node
$ brew install watchman
$ npm install -g react-native-cli
$ react-native init ReactThesisApp
$ cd ReactThesisApp
$ react-native run-ios
```

Zdrojový kód 4: *Příkazy na instalaci potřebných modulů a následné vytvoření a spuštění projektu*

Nově vytvořený projekt ReactThesisApp obsahuje:

- package.json – Soubor obsahující informace (název, verze) o balíčcích, na kterých je projekt závislý (angl. dependencies),
- node_modules/ – Složka obsahující zmíněné balíčky, které se pomocí NPM stáhnou z dostupných repozitářů. Dále je zde uložen skript (react-native/local-cli/cli.js), který ovládá celý projekt a spouští další své pomocné skripty,
- index.ios.js – Hlavní soubor celého projektu,
- iOS/ – Složka obsahující nativní kód aplikace. Je zde možné otevřít projekt pomocí .xcodeproj souboru, který se spustí v Xcode,
- index.android.js a android/ – Verze pro Android.

Jak již bylo zmíněno, aplikace začíná u souboru index.ios.js. První řádky souboru patří importu objektů, které jsou následně v tomto souboru využity (toto platí i pro další soubory, které mají na starosti např. jednotlivé stránky aplikace). Zdrojový kód 5 ukazuje, že import objektů je možný i z různých dalších balíčků. Zde konkrétně komponenta MapView, která je importována z balíčku react-native-maps, který byl nainstalován do projektu pomocí NPM.

```
import React { Component } from 'react';
import { Image,StatusBar,MapView, StyleSheet,Text,TouchableHighlight,View } from
'react-native';
import { MapView } from 'react-native-maps';
```

Zdrojový kód 5: *Ukázka importu objektů v React Native*

První řádek importuje tzv. komponentu, pod kterou si můžeme představit v podstatě jakýkoliv objekt, který se v aplikaci zobrazí. Pokud chceme např. vytvořit nové okno (angl. screen) v naší aplikaci, tak je nutné dědit právě třídu Component, která nové třídě předepisuje metodu render, pomocí které je náš nový objekt vykreslen. Zdrojový kód 6 ukazuje definici nové komponenty MapScreen, která má ve své vykreslovací metodě definováno vykreslení dalších komponent View a MapView.

```
export class MapScreen extends Component {
  render() {
    return(
      <View style={styles.container}>
        <MapView />
      </View>
    )
  }
}
```

Zdrojový kód 6: *Vytvoření komponenty MapScreen*

4.6 Základní komponenty React Native

React Native poskytuje mnoho komponent, které lze použít pro různé účely. Facebook nám nabízí přehlednou dokumentaci všech komponent, které React Native nativně obsahuje[12]. Kromě nabízených komponent, které jsou obsažené přímo v React Native, je možné využít spoustu dalších komponent, vytvořených komunitou vývojářů. Tyto komponenty lze opět instalovat jako balíčky pomocí NPM (Node Package Manager).

4.6.1 View

Nejdůležitejší komponenta z hlediska uživatelského rozhraní. Další komponenty se vkládájí do View, které poskytuje možnost pozicování pomocí technologie Flexbox (viz dále), která byla představena v CSS3[23]. Jednotlivé View je možné vkládat do sebe (angl. nesting) a tím např. rozdělovat grafické rozhraní na bloky. Jednoduchý příklad (zdrojový kód 7) naznačuje, jak lze vytvořit View a v něm dva stejně široké bloky pomocí Flexboxu, které budou mít

[23] CSS (kaskádové styly) –jazyk, pomocí kterého popisujeme zobrazení elementů použitých v jazycích HTML, XHTML, XML.

výšku 100 pixelů a šířka se rovná polovině šířky zařízení. Vložené View mají ještě definované i různé barvy pomocí vlastnosti backgroundColor.

```
<View style={{flexDirection: 'row', height: 100, padding: 20}}>
    <View style={{backgroundColor: 'blue', flex: 0.5}} />
    <View style={{backgroundColor: 'red', flex: 0.5}} />
</View>
```

Zdrojový kód 7: *Použití komponenty View v React Native*

4.6.1.1 Flexbox

Flexbox je základem grafického rozložení (angl. layout) aplikací v React Native. Jedná se o model, který nám umožnuje kontrolovat layout v aplikaci (viz zdrojový kód 7, který ukazuje, jak lze aplikaci rozdělit na dvě poloviny a následně by do těchto částí bylo možné přidat další obsah). Flexbox je velice podobný modelu Flexbox představeném v CSS3[24]. Vývojáři frameworku React Native ho však přepsali a vytvořili znovu speciálně pro iOS / Android[18].

Ve Flexboxu rozlišujeme dva typy elementů. Flex container a v něm obsahlé flex items. První zmíněný označuje kontejner, který popisuje chování položek (flex items) definovaných v něm. Příkladem chování může být směr, jakým se položky mají vykreslovat. Tzn. zda se mají uspořádat pod sebe jako řádky (row) či vedle sebe (column). K tomuto slouží vlastnost flexDirection. Dále je možné definovat zarovnání položek a pro jednotlivé položky je nejdůležitější vlastností poměrná šířka, kterou nastavujeme vlastností flex [15].

4.6.2 State a Props

Dále existují dvě speciální „property“, pomocí kterých lze předávat data v komponentách.

- State – Jedním z těchto způsobů je právě state. Jedná se o data, která se v průběhu aplikace mohou měnit (např. data získaná od uživatele),

[24] https://www.w3.org/TR/css-flexbox-1/

- Props – Druhý způsob práce s daty jsou tzv. props. Komponenty jsou inicializovány s různými parametry např. cesta ke zdroji obrázku či styly komponenty (viz zdrojový kód 7 –style). Tyto parametry nazýváme props a během života komponenty se nemění.

4.6.3 StyleSheet

Vzhled aplikace patří k tomu nejdůležitějšímu z hlediska vývoje. Všechny komponenty obsahují vlastnost (prop) style, pomocí které lze definovat jednotlivé styly. Názvy a hodnoty jsou téměř identické s CSS, které se používá na webových stránkách. Pro definici se využívá komponenta StyleSheet viz zdrojový kód 8. Definujeme ji většinou na konci souboru či ve speciálním souboru obsahující pouze styly (což ale porušuje komponentový přístup React Native).

```
const styles = StyleSheet.create({
  button: {
    textAlign: 'center',
    fontSize: 18,
    marginBottom: 10,
    marginTop: 10,
    color: 'blue'
  },
 separator: {
    height: 1,
    backgroundColor: '#CCCCCC',
  }
});

//Pouziti stylu
<Text style={styles.button}></Text>
```

Zdrojový kód 8: *Ukázka definice stylů a použití v React Native*

4.6.4 Text, TextInput

Pro práci s textem uvedu dvě základní komponenty, které je možné použít. Komponenta Text se využívá pro zobrazení obyčejného textu. TextInput nabízí možnost zadání textu uživatelem. Obsahuje props onChangeText a onSubmitEditing, kterým můžeme předat funkce, které se mají provést při změně textu či odeslání formuláře.

4.6.5 ListView

Komponenta umožňující zobrazení listu položek s rozdílnými daty, které však mají identickou strukturu. Jako příklad si můžeme představit telefonní seznam kontaktů. Zdrojem

(DataSource) ListView je obvykle kolekce či pole objektů. Pomocí funkce renderRow definujeme, jak by měl vypadat jeden řádek daného ListView.

4.6.6 MapView

MapView využívá nativní objekt MKMapView, pomocí kterého nám nabízí možnost použití map v naší aplikaci. MapView bylo původně součástí frameworku React Native. Od verze 0.42 je však považováno za zastaralé a je doporučeno používat komponentu vytvořenou společností Airbnb, kterou Facebook označil za lepší než jejich původní[25]. Toto je velice zajímavé a pozitivní, z hlediska budoucnosti vývoje celého frameworku, kdy je vidět, že komunita vývojářů React Native neustále zlepšuje a Facebook se nebojí soplupráce s komunitou. Ve zdrojovém kódu 9 můžeme vidět, jakým způsobem se komponenta používá. Je vložena ve View a obsahuje několik různých atributů, které ovládají chování této komponenty.

```
<View style={styles.container}>
  <MapView
    style={this.props.mapStyle}
    mapType = {this.props.mapType}
    showsUserLocation={this.props.showsUserLocation}
    followUserLocation={this.props.followUserLocation}
    onRegionChangeComplete={(region) => this._onRegionChangeComplete(region)}
    region={this.state.mapRegion}
  />
</View>
```

Zdrojový kód 9: *Příklad komponenty MapView a různými props*

4.6.7 Navigator

Navigace mezi několika obrazovkami je velice důležitou součástí většiny aplikací. Umožňuje uživateli přesouvat se z jednotlivých částí aplikace do dalších za pomoci definovaných cest (angl. routes). Komponentu je možné použít jak pro iOS, tak i pro Android a na obou platformách je možné upravovat její vzhled [15]. Existuje však i komponenta NavigatorIOS, která se využívá pouze pro systém iOS. Je lépe optimalizovaná, protože funguje identicky jako nativní UINavigationController (součást frameworku UIKit). Pomocí dostupné (vytvořena komunitou) komponenty react-native-navigation je možné zajistit využití

[25] https://facebook.github.io/react-native/docs/mapview.html

nativních navigací pro obě platformy (iOS, Android) a je dokonce doporučena v dokumentaci React Native.

4.6.8 TouchableHighlight

Jedná se o základní komponentu, která reaguje na interakci uživatele. Chová se jako „obálka“, do které lze vložit (povinná alespoň jedna) jakoukoliv další komponentu (např. obrázek). Při stisku je spuštěna funkce, kterou přiřadíme parametru onPress.

4.7 Přístup k nativnímu API

Součástí frameworku jsou i objekty, které přímo využívají nativní API systému iOS (či Android). Díky těmto objektům je možné využívat funkce typu geolokace, animace, vibrace telefonu atp.

- GeoLocation – Umožňuje nám přístup k aktuální GPS pozici a sledování změn pozice,
- Animated – Knihovna, která nabízí možnost tvoření animací. Ty lze využít pro komponenty Image, ScrollView, Text a View,
- Alert – Vytváří dialog, který obvykle uživateli zobrazuje nějaké důležité informace (např. povolení o přístup k informacím o poloze). Dialogu je možné přidat obsah a tlačítka, kterým lze na stisk přiřadit námi definované funkce, které vykonají nějakou akci v aplikaci,
- CameraRoll – Některé aplikace využívají fotografie uložené v telefonu. Přístup k těmto fotografiím nám poskytuje objekt CameraRoll. Toto API také poskytuje možnost uložení obrázku do paměti telefonu.

Vývojáři mají dále možnost vytvořit si své vlastní moduly, které budou přistupovat k nativnímu API. Tyto moduly využívají speciální protokol RCTBridgeModule, který umožňuje propojení mezi Objective-C (či Swift) a frameworkem React Native. Pomocí tohoto „mostu“ (a s ním souvisejících speciálních metod) můžeme napsat třídy a funkce v nativním kódu a následně k nim přistupovat v React Native [28].

5 METODIKA POROVNÁNÍ

V této kapitole představuji mé metody, které jsem zvolil pro zjištění dat a dosažení cíle této bakalářské práce jejich analýzou a porovnáním.

5.1 Aplikace

Pro zjištění výsledků mé práce jsem se rozhodl vytvořit dvě jednoduché identické aplikace, které budou sloužit pouze pro testovací účely a které tedy nemají žádné využití pro uživatele. Aplikace je rozdělena na čtyři části a každá z nich testuje jiné možnosti obou platforem.

5.2 Testování

Pro porovnání jazyka Swift a frameworku React Native jsem zvolil několik parametrů, podle kterých se pokusím určit výsledky. Výkon je důležitou součástí softwarových aplikací. Performance neboli výkon je popsán v normě ISO/IEC/IEEEE 24765 jako „*the degree to which a system or component accomplishes its designated functions within given constraints, such as speed, accuracy, or memory usage*“ [29]. V překladu to znamená, že výkonem je označována schopnost provedení požadovaných funkcí systému z pohledu parametrů jako rychlost, přesnost nebo využití paměti.

Porovnávání výkonu mezi dvěma zdroji nazýváme benchmark. Existuje spousta metod, jakými lze programovací jazyky porovnávat mezi sebou. Příkladem může být provedení algoritmů pro řešení binárních stromů, násobení matic či vyřešení příkladu sudoku[26].

Tyto metody jsou pro mé využití nevhodné, protože by se jednalo o srovnání JavaScriptu a Swift, na což se táto práce nezaměřuje. Rozhodl jsem se tedy pro vytvoření již zmiňované aplikace, kterou budu analyzovat pomocí nástroje Instruments (3.3.6) na reálném zařízení iPhone 5, a to z hlediska těchto parametrů:

- CPU (Time Profiler Tool) – využití procesoru a jeho chování,
- GPU (Core Animation) – výkon grafického procesoru z pohledu zobrazení snímků za sekundu,
- RAM (Allocations Tool) – sledování použité paměti,

[26] https://attractivechaos.github.io/plb/

- Velikost aplikace – velikost nainstalované aplikace.

Tyto čtyři parametry by měly přinést dostatek dat pro určení výsledku této práce z hlediska výkonu porovnávaných platforem. Obě aplikace byly napsány tak, aby se co nejvíce podobaly a výsledky testovaní tak mohly být autentické. Testy byly provedeny opakovaně a ve výsledcích byl využit průměr těchto hodnot.

Dále jsem ještě uznal za vhodné porovnání dvou dalších hodnot:

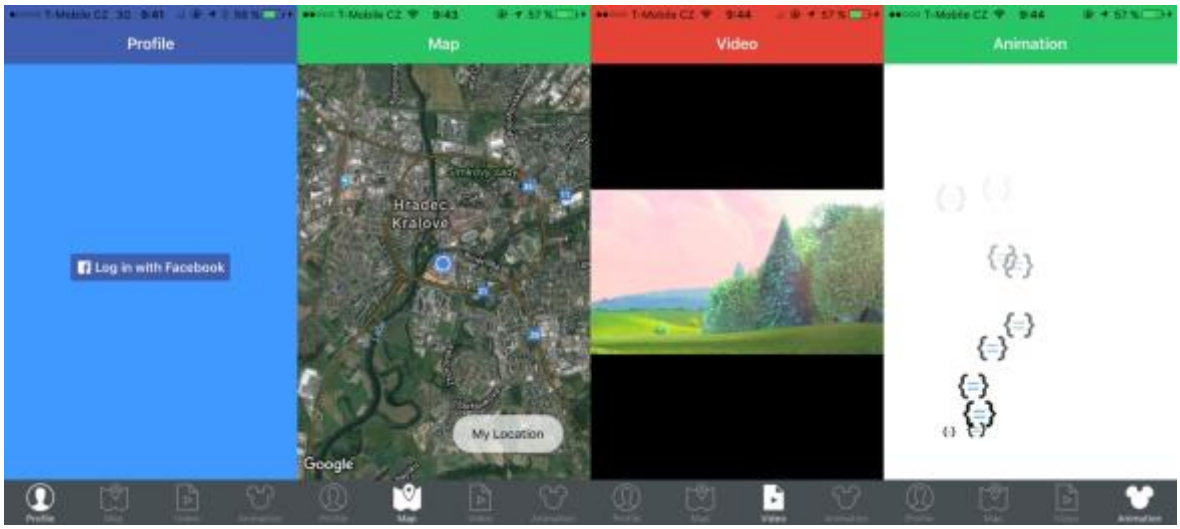
- Počet řádků napsaného kódu,
- Čas strávený vývojem a učením se obou platforem.

Tyto informace nám poskytují další pohled k porovnání vývoje obou platforem.

Posledním faktorem, na kterém bude z hlediska výsledku záviset, je můj osobní celkový názor na oba testované subjekty. Tento názor bere v potaz jasnost a porozumění kódu a dokumentaci, problémy se kterými jsem se setkal během vývoje a případné existující chyby (bugy), které mohly zpomalit tvorbu aplikace. V neposlední řadě bylo nutné vzít v potaz ergonomii vývojového prostředí.

6 IMPLEMENTACE

Tato část popisuje aplikaci, kterou jsem vytvořil pro účely porovnání jazyka Swift a frameworku React Native. Je rozdělena na čtyři části a každá tato část využívá jinou funkcionalitu. Bylo nutné vytvořit dvě identické aplikace, a i z těchto důvodů není aplikace příliš komplexní. Dalším důvodem byla časová náročnost naučení se a porozumění oběma platformám. Jelikož je aplikace z hlediska obsahu určena pouze pro testování, zvolil jsem její název jednoduše SwiftThesisApp[27], resp. ReactThesisApp[28] (viz obrázek 8).



Obrázek 8: *Ukázka oken aplikace vytvořené v React Native*

Aplikace je rozdělena pomocí navigačního prvku zvaného TabBar (ukázka na obrázku 8), který umožňuje vytvořit ve spodní části okna lištu s tlačítky, která navigují na jednotlivá okna aplikace.

Obrázek 9: *Ukázka grafického prvku TabBar*

V jazyku Swift se jedná o UITabBarController a jeho nastavení je provedeno v souboru AppDelegate.m, kde je nastaveno, aby se jako první zobrazil controller přihlášení.

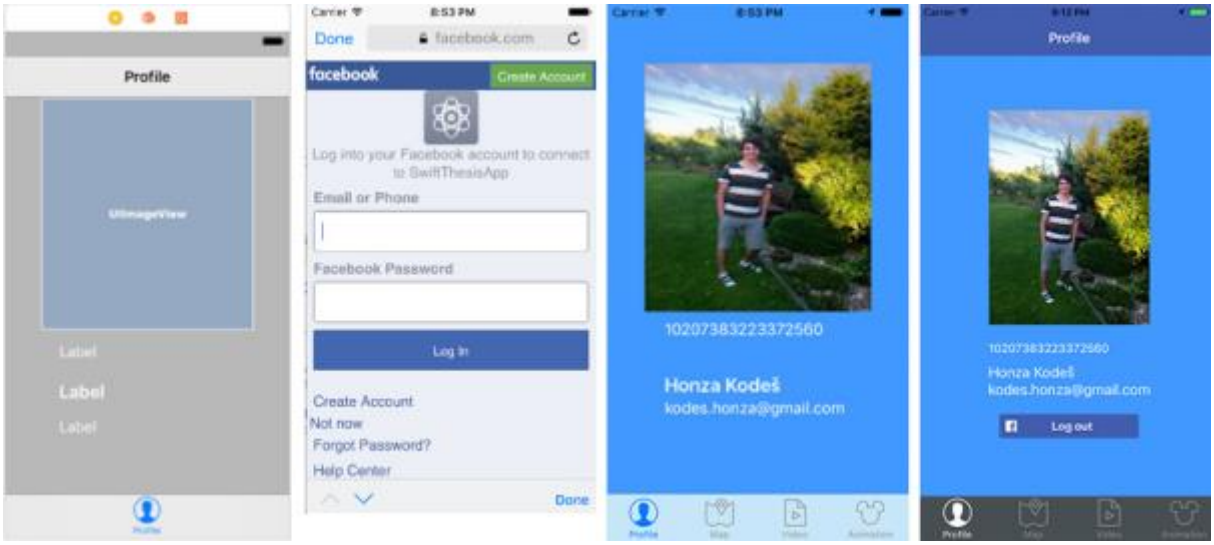
[27] Dostupné z: https://github.com/johnkodes/SwiftThesisApp

[28] Dostupné z: https://github.com/johnkodes/ReactThesisApp

Pro druhou verzi aplikace jsem zvolil volně dostupnou komponentu React Native Navigation, která nabízí možnost navigace pomocí TabBaru, ale i zobrazení horní navigační lišty (Navigator) a případné přidávání tlačítek do této lišty. Tato funkcionalita se obecně využívá hlavně u výpisu dat a navigací mezi jednotlivými položkami. V mé aplikaci byla horní lišta využita pouze pro informační účely.

6.1 Profile

První částí aplikace je okno (viz obrázek 9), obsahující přihlášení na sociální síť Facebook pomocí knihovny FBSDK. Tato knihovna obsahuje funkce k vytvoření tlačítka a dotázání na autorizaci Facebook aplikace, kterou jsem vytvořil ve webovém rozhraní Facebook for Developers. Pokud je aplikace autorizována, jsou získána data z použitého účtu k přihlášení a zobrazena v grafických prvcích, konkrétně: profilový obrázek, ID uživatele, jméno a email.



Obrázek 10: *Zleva Storyboard, autorizace Facebook aplikace, ukázka Swift, ukázka React Native*

Swift

Čas: 2h 30min[29]

[29] Hodnota vychází z času, který byl nutný k vyhledání vhodných objektů či komponent pro daný úkol, studování dokumentace a následné implementace a otestování. Toto platí i pro níže uvedené časové hodnoty.

Pro instalaci FacebookSDK (Software Development Kit) jsem využil CocoaPods, o kterých se zmiňuji v kapitole 3.5.1. Jelikož knihovna využívá Objective-C, bylo nutné vytvořit tzv. Bridging Header, který umožňuje propojení s jazykem Swift. Xcode při přidání Objective-C třídy nabídne automatické vytvoření tohoto souboru. Pojmenovává se např. SwiftThesisApp-Bridging-Header.h. Zdrojový kód 10 ukazuje, jakým způsobem je následně možné importovat knihovny, které jsou psané v jazyku Objective-C [6].

```
#import <FBSDKCoreKit/FBSDKCoreKit.h>
#import <FBSDKLoginKit/FBSDKLoginKit.h>
#import <GoogleMaps/GoogleMaps.h>
```

Zdrojový kód 10: *Ukázka obsahu souboru Bridging Header*

Třída FBLoginViewController zajišťuje přihlášení pomocí objektu FBSDKLoginManager. Využívá protokol, který definuje použití tří metod. Funkčně využívám pouze jednu, a to metodu loginButton, která se vykoná po dokončení přihlášení. Pokud proběhlo vše v pořádku, je

spuštěna funkce getProfile (viz zdrojový kód 11), která získá požadovaná data z Graph API Facebooku.

```
/*!
 *  Ziskani informaci o profilu z FBGraph Api
 */
func getProfile() {
    let parameters = ["fields": "id, name, email, picture.type(large)"]

    FBSDKGraphRequest(graphPath: "me", parameters: parameters ).start(completionHandler:
{ (connection, result, error) -> Void in
        if (error == nil){

            /** Objekt obsahující data z Graph API */
            let fbDetails = result as! NSDictionary

            /** Přiřazení jednotlivých hodnot grafickým prvkům */
            userName = (fbDetails.value(forKey: "name") as? String)!
            userEmail = (fbDetails.value(forKey: "email") as? String)!
            userID = (fbDetails.value(forKey: "id") as? String)!

            /** Získání URL obrázku a jeho stažení */
            if let imageURL = ((fbDetails.value(forKey: "picture") as
AnyObject).value(forKey: "data") as AnyObject).value(forKey: "url") as? String {
                let url = NSURL(string: imageURL)

                URLSession.shared.dataTask(with: url! as URL, completionHandler: {
(data, response, error) -> Void in
                    if error != nil { print(error!); return }
                    let image = UIImage(data: data!)
                    DispatchQueue.main.async(execute: { () -> Void in
                        userImage = image!
                    })
                }).resume()
            }
        }
    })
}
```

Zdrojový kód 11: *Ukázka funkce getProfile*

React Native

Čas: 2h 15min

Pro implementaci v React Native jsem využil komponenty FBLogin a FBLoginManager[30] a následně dvě další komponenty, které zobrazují získaná data. Komponenty využívají funkce componentWillMount, která je spuštěna před renderováním komponenty. V této funkci se provede získání dat z URL pomocí metody fetch [31], která tato data zpracuje a převede je

[30] https://github.com/magus/react-native-facebook-login

[31] https://facebook.github.io/react-native/docs/network.html

do formátu JSON. Následně jsou data přidána do objektu state dané komponenty pomocí metody setState. Příklad definice komponenty je ukázán na zdrojovém kódu 12.

```
class Photo extends Component {
  constructor(props) {
    super(props);
    this.state = {
      photo: null
    };
  }
  componentWillMount() {
    var user = this.props.user;
    var api =
`https://graph.facebook.com/v2.3/${user.userId}/picture?width=200&redirect=false&ac
cess_token=${user.token}`;

    fetch(api)
      .then((response) => response.json())
      .then((responseData) => {
        this.setState({
          photo : {
            url : responseData.data.url,
            height: responseData.data.height,
            width: responseData.data.width,
          },
        });
      })
      .done();
  }
}
```

Zdrojový kód 12: *Definice komponenty* Photo

6.2 Map

Druhou částí aplikace je práce s mapou. Úkolem tohoto okna je při kliku na tlačítko určit polohu uživatele a s animací se přemístit na tuto lokaci. Rozhodl jsem se použít Google Maps, protože se v reálných aplikacích častěji setkávám s mapami od Google. Jako tlačítko jsem využil položku v navigační liště (TabBar). Obě verze využívají Google Maps SDK[32]. Pro správné fungování je nutné upravit soubor AppDelegate.m v obou verzích aplikace, do kterého se přidává instance třídy GMSServices, která požaduje jako parametr klíč API. Tento klíč lze bezplatně získat na webové stránce Google Developers.

[32] https://developers.google.com/maps/documentation/ios-sdk/

Swift

Čas: 1h 45min

Opět jsem k instalaci Google Maps použil CocoaPods. Následně jsem vytvořil MapView, které využívá třídu obsaženou v Google Maps SDK, a to GMSMapView. Ke zjištění lokace se využívá objekt CLLocationManager, jenž obsahuje i protokol CLLocationManagerDelegate, který rozšiřuje ViewController o metodu locationManager. Jedná se o přetěžovanou metodu, která se liší pomocí předaných parametrů. Jednou z činností této metody je i sledování změn GPS pozice a následné reakce na tuto událost. Důležitým předpokladem pro zjištění lokace je povolení přístupu uživatelem k polohovým informacím. CLLocationManager vytvoří dialog se žádostí o povolení lokace, který je zobrazen uživateli.

React Native

Čas: 2.55 h (vysoký čas z důvodu bugu)

Ve frameworku React Native jsem využil komponentu React Native Maps od společnosti Airbnb, která nabízí i možnost zobrazení pomocí Google Maps. Při integraci komponenty jsem narazil na velké potíže z hlediska instalace, kdy se mi nedařilo (podle dokumentace) přidat potřebné soubory do aplikace a zprovoznit tuto komponentu. Jednalo se o chybu, kterou způsobila nová verze React Native a komponenta bohužel ještě nebyla správně aktualizována. K přidání komponenty do aplikace jsem využil NPM a pomocí CocoaPods jsem instaloval knihovnu Google Maps. CocoaPods je možné využívat i s frameworkem React Native.

6.3 Video

V další části aplikace testuji, jak jsou na tom obě platformy z hlediska zpracování médií, konkrétně přehrávání videa. Video je přehráváno z internetu a jedná se o volně dostupné video Big Buck Bunny[33]. Funkce pozastavení a spuštění je zajištěna pomocí kliknutí na libovolnou plochu na displeji.

Swift

[33] https://peach.blender.org/

Čas: 1h 5min

Pro zpracování audio-vizuálních souborů obsahuje Swift rozsáhlý framework AVFoundation. Tento framework obsahuje objekt AVPlayer (zdrojový kód 13 ukazuje použití této třídy), pomocí kterého lze spouštět lokální či internetové soubory. Obsahuje spoustu užitečných funkcí, pomocí kterých lze přehrávaná média obsluhovat. Pro zjištění kliknutí na obrazovku jsem využil metodu touchesBegan, kterou obsahuje třída UIViewController.

```
class VideoViewController: UIViewController {

    /** Přehrávač medií z frameworku AVFoundation */
    var player: AVPlayer?

    override func viewDidLoad() {
        super.viewDidLoad()

        /** URL k videu */
        let path =
"https://ia600201.us.archive.org/12/items/BigBuckBunny_328/BigBuckBunny_512kb.mp4";
        let videoURL = NSURL(string: path)

        /** Inicializace přehrávače AVPlayer */
        player = AVPlayer(url: videoURL! as URL)

        player!.actionAtItemEnd = AVPlayerActionAtItemEnd.none;

        /** Vrstva, pomocí které AVPlayer zobrazuje obraz */
        let playerLayer = AVPlayerLayer(player: player)

        playerLayer.frame = self.view.bounds
        playerLayer.videoGravity = AVLayerVideoGravityResizeAspect
        self.view.layer.masksToBounds = true

        self.view.layer.insertSublayer(playerLayer, at: 0)
        NotificationCenter.default.addObserver(self, selector:
#selector(VideoViewController.playerItemDidReachEnd), name:
NSNotification.Name.AVPlayerItemDidPlayToEndTime, object: player!.currentItem)

        /** Nastavení pozice přehrávače na počátek */
        player!.seek(to: kCMTimeZero)
        player!.pause()
    }
```

Zdrojový kód 13: *Část třídy VideoViewController zajišťující přehrávání videa*

React Native

Čas: 40min

V základu framework neobsahuje komponentu, která by poskytovala možnost přehrávání videa. Je tedy opět nutné zvolit komponentu od jiného vývojáře. Pomocí jsem nainstaloval React Native Video [34]. Pro zajištění zvuku je nutné opět upravit AppDelegate.m, do kterého je přidána instance objektu AVAudioSession. Implementace komponenty je poměrně jednoduchá. Stačí přidat tag <Video /> a nastavit požadované parametry. Video jsem umístil do elementu TouchableOpacity (viz zdrojový kód 14), pro možnost reakce na kliknutí uživatelem.

```
render() {
  return(
    <View style={styles.container}>
      <TouchableOpacity style={styles.backgroundVideo} onPress={() =>
{this.setState({paused: !this.state.paused})}}>
        <Video
          repeat
          resizeMode='contain'
          source={{uri:'https://ia600201.us.archive.org/12/items/BigBuckBunny_328/B
igBuckBunny_512kb.mp4'}}
          style={styles.backgroundVideo}
          paused={this.state.paused}
        />
      </TouchableOpacity>
    </View>
  );
}
```

Zdrojový kód 14: *Ukázka metody render ze souboru VideoScreen.js zobrazující element videa*

6.4 Animation

Poslední část aplikace se soustředí na využití možností animací. V této funkci aplikace by měl být Swift favoritem, protože výkon v animacích byl od počátku vývoje React Native trochu problémový, pokud se jedná o náročnější animace. Činností této části aplikace je vytvoření obrázku (logo UHK FIM), který je následně animován po vertikální cestě a postupně mizí. Obrázek je vytvořen při kliknutí na obrazovku a testování má smysl až při generaci většího množství těchto jednotlivých obrázků. Inspirací pro tyto létající elementy je aplikace Periscope.

[34] https://github.com/react-native-community/react-native-video

Swift

Čas: 2h 45min

Core Animation je infrastruktura, kterou iOS využívá pro animaci různých elementů. Další užitečnou knihovnou je Quartz 2D API, které je součástí frameworku Core Graphics. Tento framework poskytuje možnost např. vykreslování cest či transformací, které v této části aplikace využívám. Důležitou třídou je `UIBezierPath`, pomocí které lze definovat křivku, po které se obrázek bude pohybovat. Samotná animace je zajištěna pomocí objektu `CAKeyframeAnimation`. Tomuto objektu předáme křivku a dobu trvání. Plynulost animace se řídí pomocí časovacích funkcí (např. lineární, kubické, bézierovy křivky).

React Native

Čas: 2h 10min

Implementace v React Native se mi nepodařila zcela identickým způsobem. Cesta není definována jako křivka, ale je vypočítávána pomocí interpolace. Interpolace je využita i na postupné mizení létajících log. Jedná se konkrétně o funkci `interpolate`.

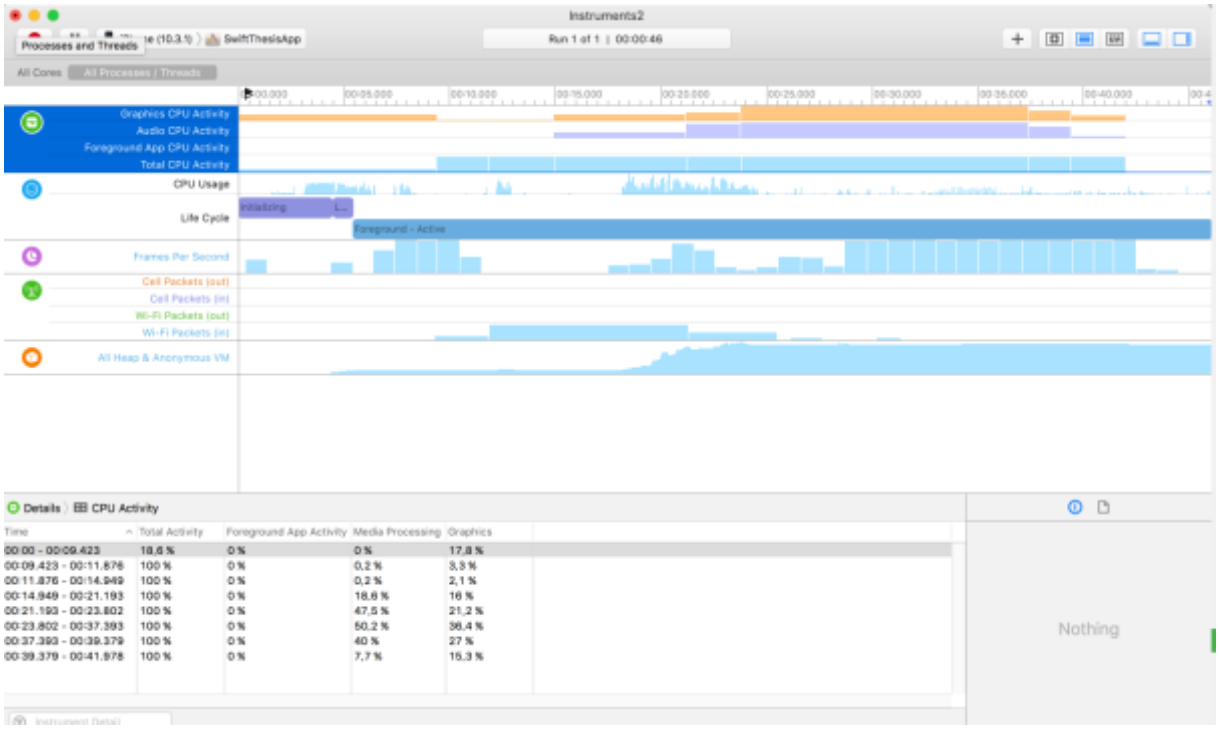
```
this._animation.opacity = this._animation.y.interpolate({
  inputRange: [0, 240],
  outputRange: [1, 0]
});

this._animation.x = this._animation.y.interpolate({
  inputRange: [0, 240/2, 240],
  outputRange: [0, 15, 0]
});
```

Zdrojový kód 15: *Ukázka interpolace opacity a pozice X v závislosti na pozici na ose Y*

Na zdrojovém kódu 15 můžeme vidět, jakým způsobem lze mapovat hodnotu v závislosti na jiné proměnné. Tato funkce má za úkol při zvyšování pozice Y zvyšovat průhlednost našeho obrázku. Tzn. např. v hodnotě `y = 120` by hodnota `.opacity` byla 0,5, tedy poloviční průhlednost. Obdobně je vypočítána i pozice X, která také závisí na pozici na ose Y. Pomocí této interpolace tak vytvoříme jednoduchou cestu pro obrázek loga.

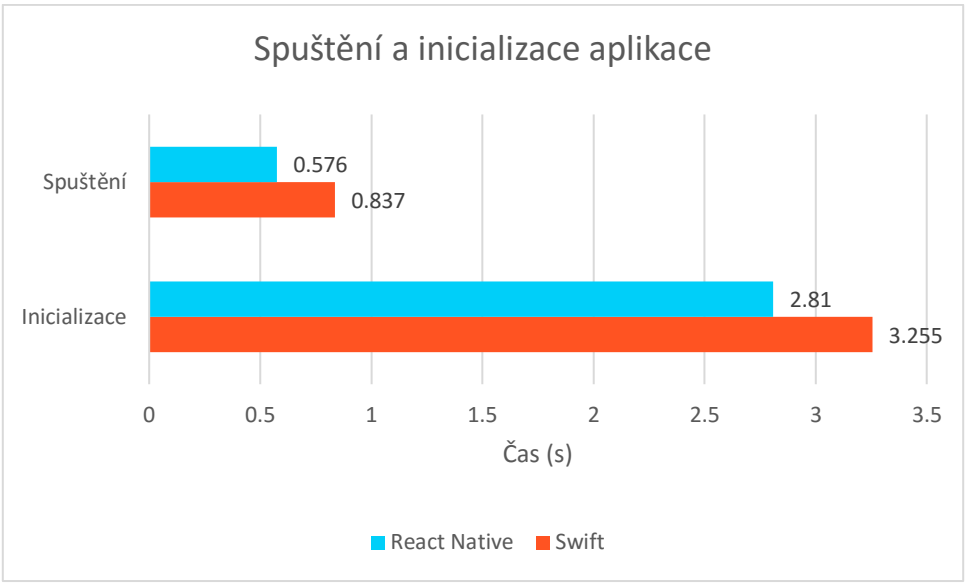
7 VÝSLEDKY

Tato část práce prezentuje data, která byla shromážděna během vývoje a testování obou aplikací. Tato data obsahují výsledky z nástroje Instruments (obrázek 10 ukazuje grafické rozhraní) a dále také časovou náročnost vývoje a s tím související počet řádků zdrojového kódu aplikace. Kapitoly jsou rozdělené podle jednotlivých částí aplikace.



Obrázek 11: *Ukázka nástroje Instruments využitého k získání dat z testování aplikace*

Před testováním jednotlivých částí aplikace, jsem navíc porovnával rychlost inicializace a spuštění aplikace. Měření zobrazené na grafu 1 jsem provedl pětkrát pro každou aplikaci a tyto hodnoty zprůměroval.

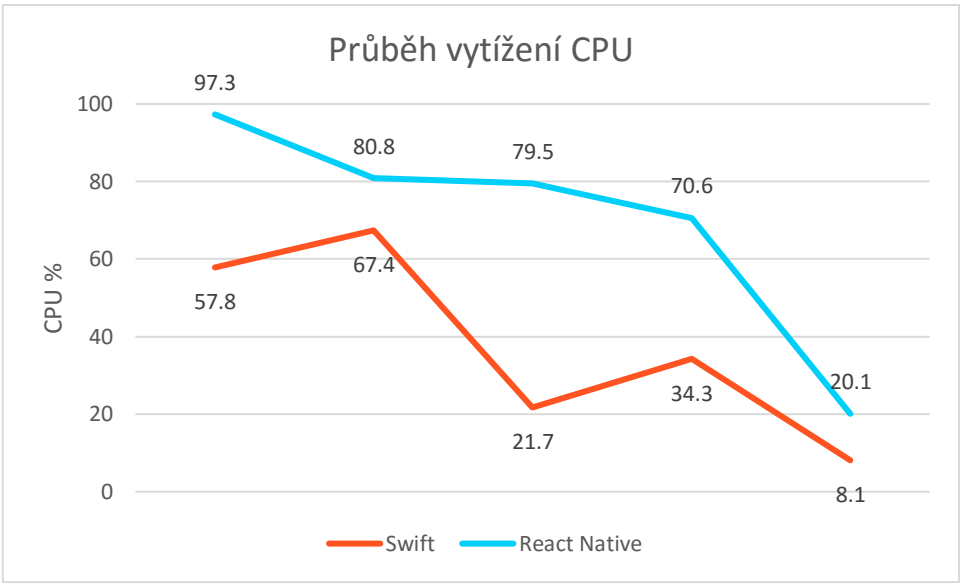


Graf 1: *Srovnání rychlosti spuštění aplikace*

Zajímavé je, že při opětovném spouštění aplikací, se časy postupně snižovaly. Konkrétně u Swift se hodnota druhého a třetího spuštění lišila o 1,7sekund.

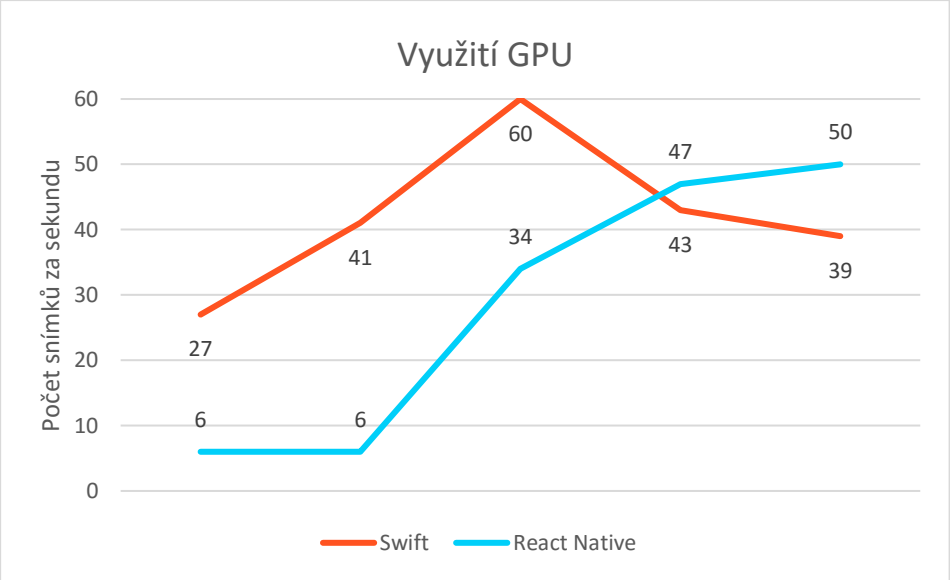
7.1 Profile

V následujících grafech je možné sledovat rozpoznatelné skoky v hodnotách (viz graf 2). První hodnota je po stisknutí tlačítka pro přihlášení na sociální síť Facebook. Následuje zobrazení okna prohlížeče, zpracování přihlášení a zobrazení dat v grafických prvcích.



Graf 2: *Vytížení procesoru první částí aplikace*

Obdobné chování vykazuje i graf 3 znázorňující GPU.



Graf 3: *Graf zobrazuje výkon GPU v okně Profile*

Graf značí, že pro React Native bylo náročné vytvoření okna webového prohlížeče, který se dotazuje na přihlášení.

Využití RAM	**Persistent (MB)**	**Total (MB)**
Swift	2,19	60,64
React Native	1,1	10,22

Tabulka 3: *Využití paměti RAM částí Profile*

Tabulka 3 zobrazuje dvě hodnoty paměti RAM. Persistent je paměť, kterou bylo nutné alokovat během sledovaného časového intervalu. Total označuje celkovou paměť, kterou aplikace využila v průběhu vykonávání dané úlohy (proces celého přihlášení zabral přibližně osm sekund). Při bližším zkoumání jsem zjistil, že obrovský rozdíl v paměti byl způsoben objekty frameworku Core UI, ale nepodařilo se mi zjistit příčinu.

7.2 Map

Sbírání dat pro tuto úlohu jsem rozdělil na dvě části. První při kliknutí na tlačítko Map v navigační liště. Druhou částí bylo použití tlačítka na zaměření aktuální lokace mobilního telefonu.

Využití CPU (%)	Přechod na okno mapy	Animace na aktuální lokaci
Swift	69,3	44
React Native	94	59

Tabulka 4: *Hodnoty využití procesoru při testování okna Map*

GPU (počet snímků / s)	Přechod na okno mapy	Animace na aktuální lokaci
Swift	24	34,5
React Native	19,5	28,5

Tabulka 5: *Tabulka hodnot grafického procesoru*

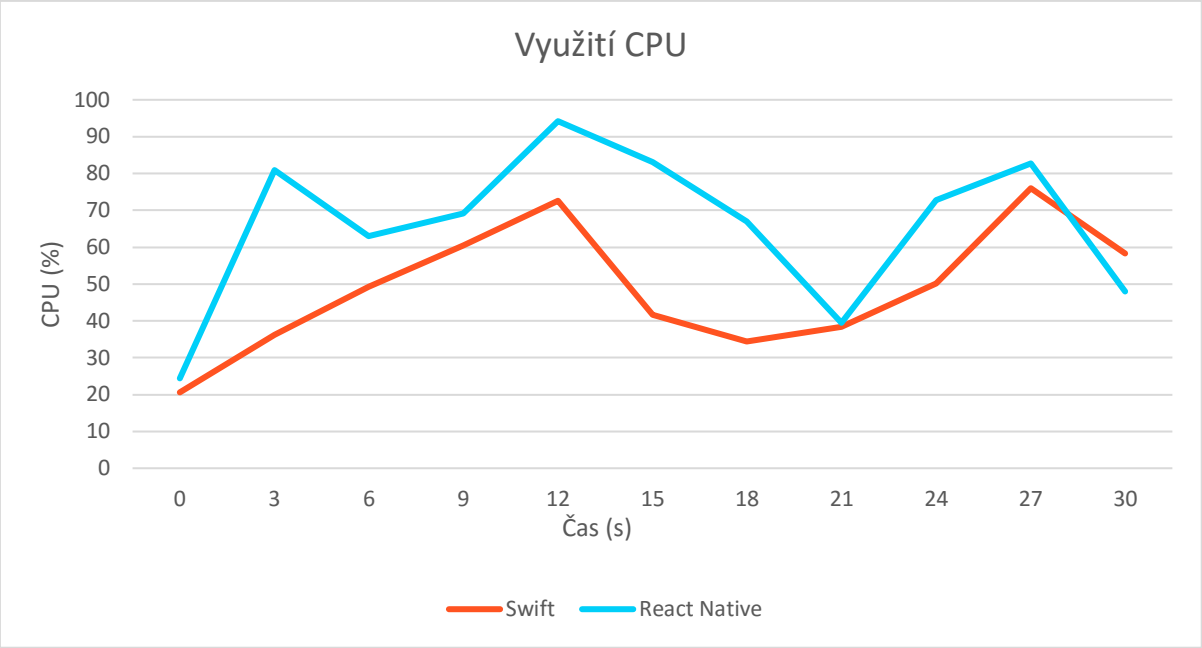
Využití RAM (MB)	**Persistent**	**Total**
Swift	29,55	252,43
React Native	10,3	101,29

Tabulka 6: *Tabulka zobrazující hodnoty využívání paměti RAM v okně Map*

Ze tabulek 4, 5, 6 výše lze zjistit, že z pohledu náročnosti na procesorovou jednotku je na tom React Native o něco hůře. Animaci v mapách však framework zvládá plynule a ve využití paměti je framework React Native jasným vítězem. Při přechodu na okno s mapou si Swift řekl o přibližně 150 megabajtů paměti a při zaměření polohy dalších cca 70 megabajtů. Mapa byla na paměť RAM nejnáročnější ze čtyř testovaných částí.

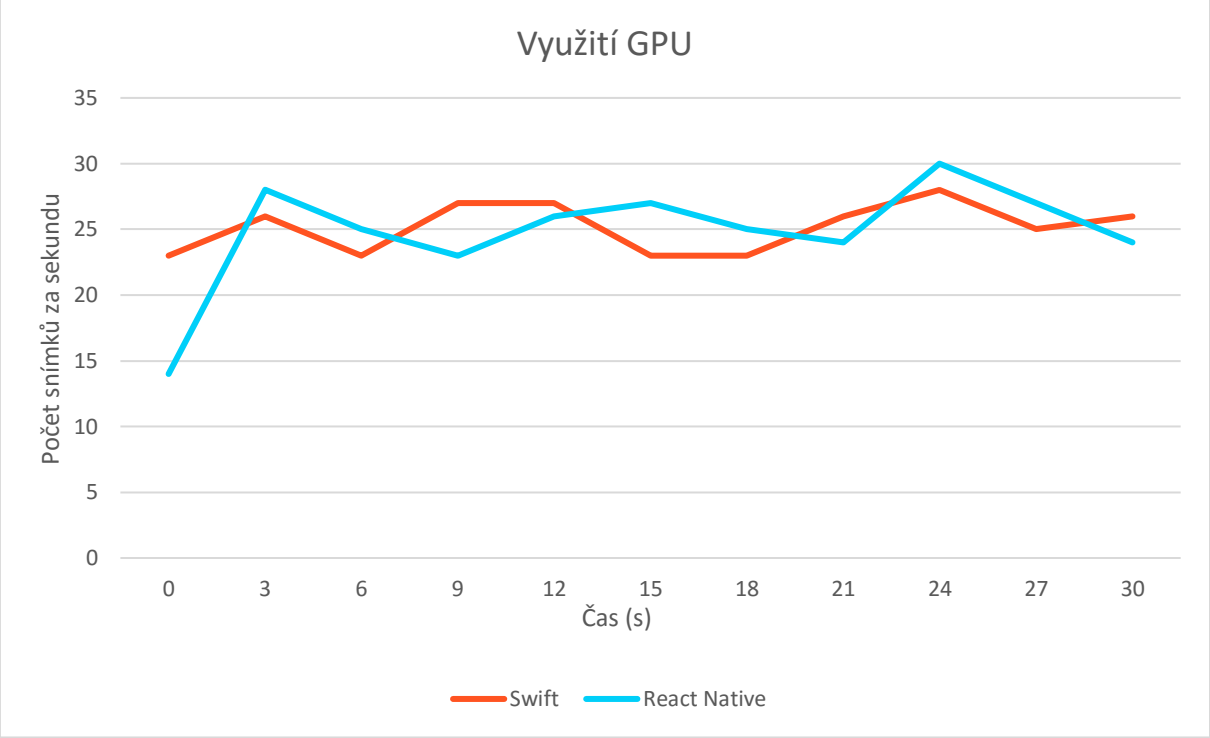
7.3 Video

Jako zdroj jsem zvolil video s použitým kodekem h.264 a rozlišením 1920x1080 pixelů. Obě platformy video plynule přehrát na zařízení iPhone 5, na rozdíl od simulátoru ve kterém jsem testoval při vývoji. Po spuštění přehrávání videa se využití paměti RAM drželo na stejných hodnotách během celého přehrávání. Běh videa jsem testoval po dobu třiceti sekund jak lze vyčíst z grafů 3 a 4.



Graf 4: *Graf zobrazující využití procesoru oknem Map*

Na grafu 4 vidíme, že křivky jsou si relativně podobné. Hodnoty stoupají a klesají, protože náročnost videa se neustále mění v závislosti na obsahu. Z výsledků vidíme, že Swift pracuje s procesorem o něco lépe.



Graf 5*: Graf zobrazující plynulost spuštěného videa pomocí hodnoty FPS*

Z hlediska snímků za sekundu (angl. frames per second – FPS) dosáhl framework React Native dokonce o trochu lepších hodnot. V jedné části videa dosáhl i na hodnotu 30 FPS, což se jazyku Swift nepodařilo (viz graf 5).

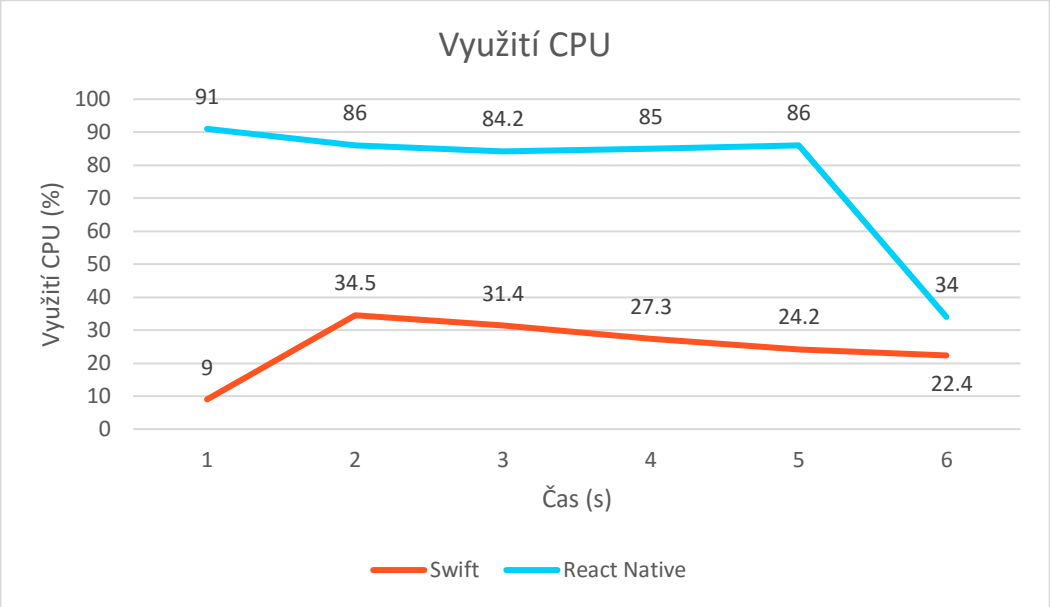
Využití RAM (MB)	Persistent	Total
Swift	3,84	55,12
React Native	0,569	65,32

Tabulka 7: *Vyčíslení využité paměti RAM v okně Video*

Hodnoty pro tabulku 7 byly získány z intervalu po kliknutí na položku v navigaci až po uplynutí stanovené doby testování videa (30 sekund).

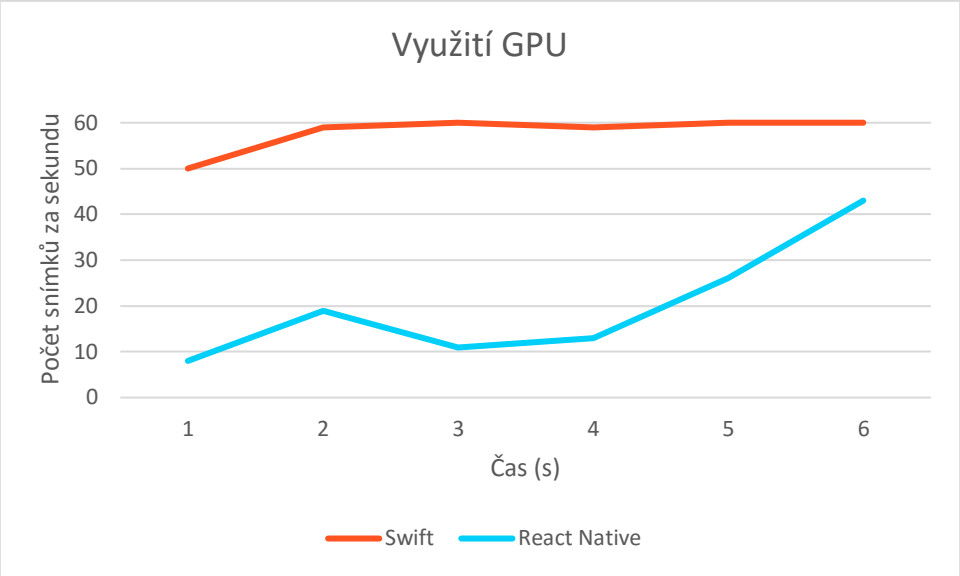
7.4 Animation

Poslední část aplikace jsem testoval po dobu pěti sekund, během kterých se každých 100ms generoval jeden obrázek. V této části zcela jasně předčil jazyk Swift svým výkonem framework React Native. API Animated, které využívá React Native, je spíše použitelné pro jednoduché animace či přechody a s větším množstvím objektů již měl framework potíže.



Graf 6: *Data zobrazující využití procesoru při vykreslování animací*

Tento test naplnil očekávání a Swift ho provedl znatelně lépe. Jak můžeme na grafu 6 vidět, procesor nebyl zdaleka tak namáhán jako v případě verze vytvořené v React Native.



Graf 7: *Graf zobrazující křivky plynulosti animací pomocí snímků za sekundu*

Z hlediska využití grafického procesoru je výsledek obdobný jako u CPU (viz graf 7). Swift si udržel vysoký počet snímků během celé doby trvání mého testu. React Native viditelně nezvládal vykreslení mnoha grafických objektů najednou. V případě testování s nižší frekvencí generování obrázků již dosahoval React Native dostatečného výkonu a animace se již jevila plynulá.

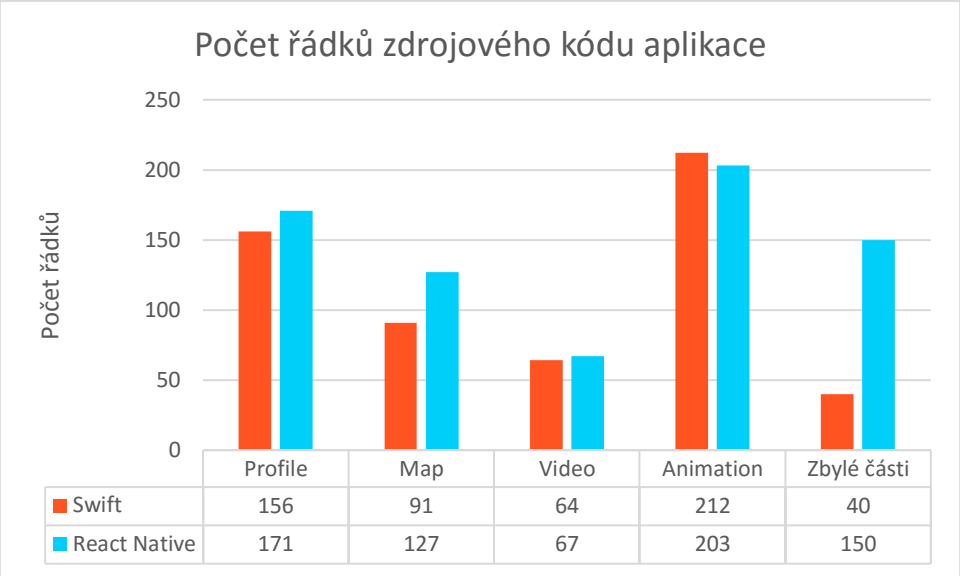
Využití RAM (MB)	Persisted	Total
Swift	0,264	6,36
React Native	0,75	13,44

Tabulka 8:*Tabulka zobrazující využití paměti v okně Animation*

Swift v tomto testu překonal druhou platformu i z hlediska paměti RAM, jak vidíme v tabulce 8. React Native alokoval dvakrát větší množství paměti nežli Swift.

7.5 Další výsledky

Dále jsem na aplikacích zkoumal tři další hodnoty. První z nich je počet řádků zdrojového kódu (viz graf 8), který byl nutný pro vytvoření aplikací. Jelikož se jedná o aplikace, které dělají identické činnosti, přišlo mi toto porovnání relativní.

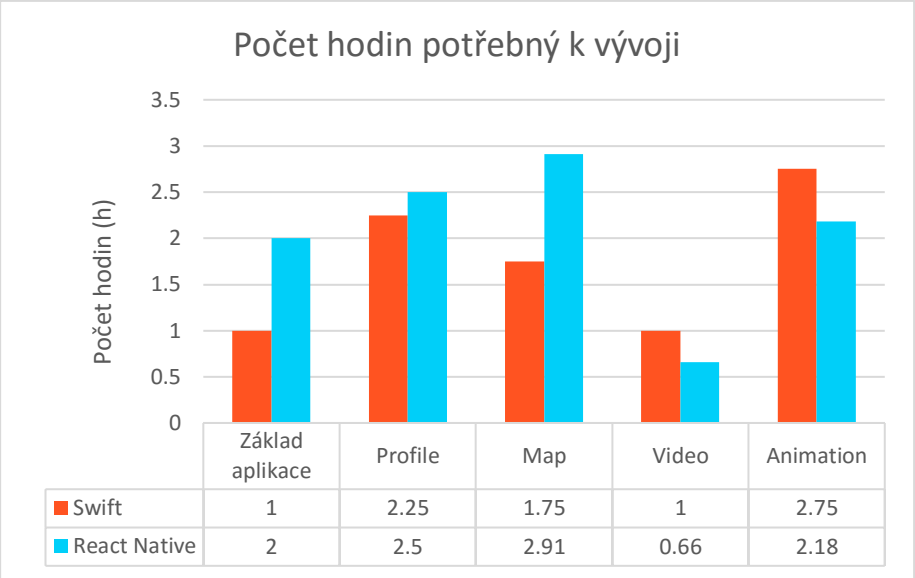


Graf 8: *Graf vykreslující počet řádků kódu v jednotlivých souborech pro konkrétní části aplikací*

Další hodnotou, která mi přišla zajímavá, byla celková velikost nainstalované aplikace. Velikost jsem zjistil pomocí instalace na reálném zařízení a položky Úložiště v nastavení iOS. Hodnota byla zjištěna ihned po instalaci aplikace do telefonu. Velikosti byly následující:

- SwiftThesisApp – 48,2 MB,
- ReactThesisApp – 26,4 MB.

Poslední proměnnou, kterou jsem se rozhodl zkoumat byl počet hodin využitý k vývoji (včetně výuky a hledání informací) obou aplikací.



Graf 9: *Graf zobrazující náročnost vývoje z hlediska časových jednotek*

Položkou základ aplikace v grafu 9 je označen čas, který byl potřeba k vytvoření prázdného projektu a následně přidání navigačních prvků a tvorba čtyř oken, ke kterým byla později doprogramována funkcionalita.

7.6 Vyhodnocení

7.6.1 Výkon

Ještě jednou se vrátím k datům, která jsem získal z testování a vývoje a pokusím se učinit konečné vyhodnocení obou testovaných subjektů. Celkově jsou z hlediska výkonu obě možnosti vývoje relativně vyrovnané.

V metodice testování jsem jako první zmínil porovnání z hlediska výkonu procesoru neboli jednotky CPU. Z grafů je vidět, že Swift umí využívat procesor efektivněji nežli React Native, což se projevilo téměř na všech oknech aplikace a nejznatelnější rozdíl byl v testu animací.

Z pohledu důležitosti je vhodné na druhém místě zmínit grafický procesor. V tomto ohledu nebyly výsledky tak rozdílné jako u testování CPU. React Native měl dokonce lepší výsledky v přehrávání videa, což jsem neočekával.

Třetím faktorem testování výkonu byla paměť RAM. Zde s nepatrným rozdílem zvítězil framework React Native, který dokázal s pamětí pracovat úsporněji, což lze nejlépe vidět na testu s mapou (viz tabulka 4).

Dále jsem zjistil, že aplikace v jazyku Swift je téměř dvakrát větší nežli ve druhé variantě. Po bližším zkoumání jsem zjistil, že důvodem je ukládání frameworku Swift přímo do aplikace. Konkrétně se jedná o 26,4 MB, bez kterých by byla následná velikost dokonce menší, nežli vygenerovaná aplikace v React Native, který žádné nativní frameworky neobsahuje.

7.6.2 Vývojářský dojem

Při procesu učení jsem čerpal z dokumentace, knih, či návodů na internetových stránkách. Z hlediska dokumentace je Swift detailněji zpracován. React Native však poskytuje přímo v dokumentaci ukázky kódu, které lze zkopírovat, použít a případně upravovat. Šikovným webovým nástrojem pro React Native je Expo Snack[35], který umožňuje psát a spouštět kód přímo v prohlížeči. Z pohledu syntaxe na mě lépe působí React Native, protože mi každý řádek kódu dával smysl, což je u komplexnějších objektů ve Swiftu někdy obtížnější. Dále např. vytváření grafického rozhraní je někdy pomocí kódu (React Native. Je však možné i v jazyku Swift) jednodušší nežli využívání nástroje Interface Builder (kapitola 3.3.4). Nevýhodou frameworku React Native je jeho neustálý vývoj a občas následná nekompatibilita mezi verzemi či komponentami třetích stran. Vývoj však lze považovat i za pozitivum, protože se framework neustále posouvá směrem dopředu a Facebook se ho snaží stále zlepšovat. Z tohoto důvodu je velice důležitá i komunita kolem React Native, která velice aktivně přispívá na řešení chyb a přidávání nových funkcí do frameworku. Během vývoje jsem několikrát hledal řešení a pomoc na webové službě Github. Podobné stanovisko se však dá říci i o jazyku Swift. K příspěvkům komunity vývojářů je však Apple mnohem více striktní a možnost zásahu do vývoje je méně pravděpodobná oproti React Native. Jelikož má o rok náskok, tak existuje k použití ještě více různých knihoven a bylo vyřešeno více problémů (Github, StackOverflow), ze kterých můžeme čerpat řešení.

[35] https://sketch.expo.io/

Největším rozdílem mezi oběma způsoby vývoje je syntaxe. Pokud již někdo ovládá JavaScript (ideálně i standard ES6), tak pro něho nebude obtížné vyvíjet v React Native a je určitě výhodnější zvolit tuto platformu. Opačným příkladem je vývojář, který má již zkušenosti s principy objektového programování nebo jazykem Objective-C. Pro tuto osobu bude Swift srozumitelnější.

Pokud bych v budoucnu vyvíjel aplikaci pro platformu Android i iOS, tak bych určitě zvolil React Native oproti vývoji dvěma různým aplikacím. Pokud by mnou vyvíjená aplikace byla cílena pouze na iOS, tak bych se přikláněl k využití jazyka Swift, který se postupem času stane (resp. Apple z něho udělá) plnohodnotným nástupcem Objective-C. Nativní přístup s sebou stále nese různé výhody (např. testovaný výkon). V některých případech (jednodušší aplikace) by však mohlo být výhodnější a rychlejší využít framework React Native.

8 ZÁVĚR

V této bakalářské práci jsem se věnoval dvěma možným způsobům vývoje mobilních aplikací pro operační systém iOS. Oba zmiňované způsoby jsou stále poměrně nové, ale v aktuální době je o ně velký zájem, protože využití mobilních zařízení stále roste, a z pohledu konzumace obsahu již předčily notebooky a stolní počítače.

Mým cílem této práce bylo zjistit, zdali lze frameworkem React Native nahradit nativní vývoj pomocí programovacího jazyka Swift. Tento cíl jsem ověřoval pomocí několika faktorů, ze které mi pomohli určit výsledek mé práce.

Obě porovnávané platformy jsou velice rozdílné z hlediska způsobu vývoje aplikací. Tyto rozdíly jsou viditelné již na první pohled z hlediska syntaxe. Swift se řadí mezi objektově orientované jazyky a od toho se odvíjí i způsob celého vývoje. Dále v porovnání s javascriptovým frameworkem nabízí větší komplexnost z pohledu programovacích možností. Mnoho z těchto možností však v práci není zmíněno, protože by bylo možné o nich napsat samostatnou práci. JavaScript se však stále vyvýjí a jeho poslední verze ECMAScript 6 již obsahuje velice podobné možnosti (kolekce, iterátory, pole určitého datového typu), jako moderní objektové jazyky Objective-C, Java či C++.

Funkcionalitu obou testovaných způsobů vývoje lze rozšiřovat pomocí rozsáhlého množství knihoven, které lze jednoduše využívat pomocí nástrojů, které platformy nabízí. Během vývoje jsem se seznámil s vývojovým prostředím Xcode a mohu říci, že se jedná o velice kvalitní nástroj. S Xcode se setkáme i při vývoji v React Native, ale spíše okrajově. Z hlediska ergonomie vývoje je Swift společně s Xcode jednodušší. Pro spuštění projektu stačí jedno tlačítko, ale nevýhodou je, že při změně jednoho řádku kódu se narozdíl od React Native kompiluje celý projekt znovu.

Pro dosažení cíle mé práce jsem vytvořil dvě aplikace, které jsem následně porovnával pomocí předem navržených testů. Během vývoje jsem pocítil, že se jedná o stále ještě poměrně mladé platformy. Projevilo se to zejména menším množstvím dostupných informací a ukázek. Případně jsem se setkal se staršími informacemi, které již nebyly aktuální, protože se např. změnila syntaxe. V tomto ohledu má Swift ve spojení s Xcode velkou výhodu, protože dokázal rozpoznat kód napsaný např. ve Swift verze 2 a navrhl automatickou korekci syntaxe na aktuální verzi. Pokud se zaměřím na výsledek mého výzkumu z hlediska výkonu, tak dopadl tak, jak jsem od počátku očekával. Rozdíly však

nebyly příliš velké a framework byl schopný konkurovat jazyku Swift. Konkrétně mě React Native překvapil u testu videa vysokého rozlišení, kde jsem se domníval, že bude zaostávat za nativním přístupem.

Pokud se zamyslím na dalším rozvojem tohoto výzkumu, tak mě napadá hned několik možností. Tato práce se věnovala pouze systému iOS a aplikace byla vyvíjena a testována pouze na mobilním telefonu. React Native podporuje i tablety a možná by mohlo být zajímavé porovnávat výkon těchto dvou skupin zařízení. Další možností pokračování výzkumu by mohl být systém tvOS, což je systém společnosti Apple pro produkt Apple TV. Na letošní konferenci F8 určené pro vývojáře, kterou pořádá Facebook, bylo hlavním tématem odvětví augmentované (rozšířená) reality. Pohled na možnosti vývoje pro tuto rozvíjející se technologii z hlediska Swift a React Native by mohly být dalším rozšířením této práce. A jako poslední možnost dalšího rozvoje uvedu téma multiplatformního vývoje mobilních aplikací. Tato práce se soustředila pouze na systém iOS. Obdobnou práci by však bylo možné vytvořit i pro systém Android a porovnat výkonnost programovacího jazyka Java s frameworkem React Native a následně s výsledky této práce.

SEZNAM POUŽITÉ LITERATURY

[1] POUSHTER, JACOB a PEWRESEARCHCENTER. *Smartphone Ownership and Internet Usage Continues to Climb in Emerging Economies* [online]. 2016 [vid. 2017-03-25]. Dostupné z: http://www.pewglobal.org/2016/02/22/smartphone-ownership-and-internet-usage-continues-to-climb-in-emerging-economies/

[2] STATCOUNTER. *Mobile and tablet internet usage exceeds desktop for first time worldwide* [online]. 2016 [vid. 2017-03-23]. Dostupné z: http://gs.statcounter.com/press/mobile-and-tablet-internet-usage-exceeds-desktop-for-first-time-worldwide

[3] SUI, Linda. *Strategy Analytics: Global Smartphone Shipments Hit a Record 1.5 Billion Units in 2016* [online]. 2017 [vid. 2017-03-23]. Dostupné z: https://www.strategyanalytics.com/strategy-analytics/news/strategy-analytics-press-releases/strategy-analytics-press-release/2017/01/31/strategy-analytics-global-smartphone-shipments-hit-a-record-1.5-billion-units-in-2016#.WMqpcxLysy5

[4] PAUL, A a A NALWAYA. *React Native for iOS Development* [online]. 1. vyd. B.m.: Apress, 2016. ISBN 9781484213964. Dostupné z: doi:10.1007/978-1-4842-1395-7

[5] APPLE INC. *The Swift Programming Language (Swift 3.1)* [online]. 2017 [vid. 2017-04-02]. Dostupné z: https://itunes.apple.com/cz/book/the-swift-programming-language-swift-3-1/

[6] INC., Apple. *Using Swift with Cocoa and Objective-C (Swift 3.1)* [online]. 2017 [vid. 2017-04-02]. Dostupné z: https://itunes.apple.com/cz/book/using-swift-cocoa-objective/id888894773?l=cs&mt=11

[7] CLAYTON, Janie, Alexis GALLAGHER, Matt GALLOWAY, Eli GANEM, Erik KERBER a Ben MORROW. *Swift Apprentice - Begining programming with Swift 2* [online]. B.m.: Razeware LLC, 2015. ISBN 978-1942878131. Dostupné z: https://store.raywenderlich.com/products/swift-apprentice

[8] MASKREY, Molly, Kim TOPLEY, David MARK, Fredrik OLSSON a JEFF LAMARCHE. *Beginning iPhone Development with Swift 3* [online]. Berkeley, CA: Apress, 2016. ISBN 978-1-4842-2223-2. Dostupné z: doi:10.1007/978-1-4842-2223-2

[9] KOCHAN, Stephen G. *Programming in Objective-C*. Sixth. B.m.: Pearson Education, 2014. ISBN 978-0-321-96760-2.

[10] MATHIAS, Matthew a John GALLAGHER. *Swift Programming (the Big Nerd Ranch Guide)* [online]. B.m.: Big Nerd Ranch Guides, 2015. ISBN 9780134398051. Dostupné z: https://www.bignerdranch.com/books/swift-programming/

[11] ALEXANDER, Brandon, J. Bradford DILLON a Kevin Y. KIM. *Pro iOS5 tools Xcode instruments and build tools* [online]. B.m.: Apress; 1 edition (December 8, 2011), 2011. ISBN 9781430236085. Dostupné z: http://dx.doi.org/10.1007/978-1-4302-3609-2

[12] FACEBOOK. *React Native Documentation* [online]. [vid. 2017-03-20]. Dostupné z: https://facebook.github.io/react-native/

[13] BAGNARDI, Frankie, Jonathan BEEBE, Richard FELDMAN, Tom HALLETT, Simon HØJBERG a KARL MIKKELSEN. *Developing a React Edge* [online]. B.m.: Bleeding Edge Press, 2014. ISBN 9781939902122. Dostupné z: https://bleedingedgepress.com/developing-react-js-edge/

[14] FEDOSEJEV, Artemij. *React.js Essentials* [online]. B.m.: Packt Publishing, 2015. ISBN 9781783551620. Dostupné z: https://www.packtpub.com/web-development/reactjs-essentials

[15] HOLMES, Ethan a Tom BRAY. *Getting Started with React Native* [online]. B.m.: Packt Publishing, 2015. ISBN 9781785885181. Dostupné z: https://www.packtpub.com/application-development/getting-started-react-native

[16] APPLE. *Swift.org* [online]. 2015 [vid. 2016-03-10]. Dostupné z: https://swift.org/about/#swiftorg-and-open-source

[17] LATTNER, Chris. *Chris Latner's Website* [online]. 2014 [vid. 2017-03-25]. Dostupné z: http://nondot.org/sabre/

[18] APPLE INC. *Apple WWDC 2014 - Swift Introduction* [online]. 2014 [vid. 2017-04-05]. Dostupné z: https://www.youtube.com/watch?v=MO7Ta0DvEWA

[19] APPLE. *The Swift Linux Port* [online]. 2015 [vid. 2017-03-27]. Dostupné z: https://swift.org/blog/swift-linux-port/

[20] APPLE. *Understanding Auto Layout* [online]. 2011 [vid. 2017-03-29]. Dostupné z: https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/AutolayoutPG/

[21] CODEPATH. *Auto Layout Basics* [online]. 2016 [vid. 2017-03-28]. Dostupné z: https://github.com/codepath/ios_guides/wiki/Auto-Layout-Basics

[22] APPLE. *Introduction to Auto Layout for iOS and OS X* [online]. 2012 [vid. 2017-04-07]. Dostupné z: https://developer.apple.com/videos/play/wwdc2012/202/

[23] APPLE. *Instruments User Guide* [online]. 2007 [vid. 2017-03-28]. Dostupné z: https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/

[24] DENA CO. *JSX Homepage* [online]. 2013 [vid. 2017-03-25]. Dostupné z: http://jsx.github.io/

[25] OCCHINO, Tom a FACEBOOK. *React Native: Bringing modern web techniques to mobile* [online]. 2015 [vid. 2017-03-29]. Dostupné z: https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/

[26] OCCHINO, Tom. *React Native: Bringing modern web techniques to mobile* [online]. 2015 [vid. 2017-03-21]. Dostupné z: https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/

[27] EISENMAN, Bonnie. *Writing Cross-Platform Apps with React Native* [online]. 2016 [vid. 2017-03-21]. Dostupné z: https://www.infoq.com/articles/react-native-introduction

[28] VILLA, Crysfel a Stan BERSHADSKIY. *React Native Cookbook* [online]. B.m.: Packt Publishing, 2016. ISBN 9781786462558. Dostupné z: https://www.packtpub.com/application-development/react-native-cookbook

[29] INTERNATIONAL ORGANIZATION OF STANDARIZATION. *ISO/IEC/IEEE 24765: Systems and software engineering* [online]. 2010 [vid. 2017-04-04]. Dostupné z: http://www.iso.org/iso/catalogue_detail. htm?csnumber=50518%0D

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM GRAFŮ

SEZNAM PŘÍLOH

PŘÍLOHA I: POPIS PŘILOŽENÉHO KOMPAKTNÍHO DISKU

Přiložený optický disk obsahuje dvě složky:

- /práce – obsahuje bakalářskou práci ve formátu doc a pdf,
- /aplikace – obsahuje dvě podsložky /react a /swift, které obsahují zdrojové kódy aplikací, které byly použity pro výzkum této bakalářské práce.

ZADÁNÍ PRÁCE

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Kodeš Jan	Barákova 585, Jičín - Nové Město	I1301339

TÉMA ČESKY:

Specifika vývoje aplikací pro systém iOS pomocí jazyka Swift a technologie React Native

TÉMA ANGLICKY:

The specifics of iOS application development using Swift and React Native technology

VEDOUCÍ PRÁCE:

Ing. Jan Dvořák - CZAV

ZÁSADY PRO VYPRACOVÁNÍ:

1. Úvod
2. Programovací jazyk Swift
3. Technologie React Native
4. Porovnání obou způsobů vývoje
5. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

-

Podpis studenta: Kodeš **Datum:** 26. 3. 2017

Podpis vedoucího práce: **Datum:** 26. 3. 2017

(c) IS/STAG , Portál - Podklad kvalifikační práce , I1301339 , 26.03.2017 16:42