VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

MOBILNÍ APLIKACE PRO STUDENTY VUT

MOBILE APPLICATION FOR BUT STUDENTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDREJ KLINOVSKÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Prof. Ing. TOMÁŠ HRUŠKA, CSc.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Klinovský Ondrej**
Program: Informační technologie
Název: **Mobilní aplikace pro studenty VUT**
Mobile Application for BUT Students
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte možné technologie vývoje mobilních aplikací pro platformy iOS a Android.
2. Navrhněte základní uživatelské rozhraní mobilní aplikace, které by mělo obsahovat především osobní rozvrh, elektronický index studenta a modul menzy.
3. Vyberte vhodné technologie pro jednotnou implementaci studentské aplikace pro platformy Android a iOS.
4. Implementujte aplikaci pro studenty využívající celoškolské přihlašování a data z Centrální databáze VUT obsahující minimálně moduly pro elektronický index a rozvrh.
5. Proveďte uživatelské testování aplikace na omezené skupině studentů VUT a na základě výsledků doporučte další rozvoj aplikace.
6. Vytvořte plakát nebo video prezentující práci s aplikací.

Literatura:

- Bonnie Eisenman, Learning React Native: Building Native Mobile Apps with JavaScript, ISBN-13: 978-1491929001
- Tony Buzan, Mobile App Development with Ionic, revised edition (anglicky), ISBN: 9781491998120
- Dále dle pokynu vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Hruška Tomáš, prof. Ing., CSc.**
Konzultant: Marušinec Jaromír, Ing., Ph.D., CVIS VUT
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 16. října 2019

Abstrakt

S rastúcou popularitou mobilných aplikácií rastie aj ich dopyt vo viacerých oblastiach a školstvo nie je výnimkou. Úlohou tejto práce je návrhnúť a implementovať mobilnú aplikáciu, ktorá má poskytovať študentom VUT informácie o ich štúdiu jednoducho a prehľadne. Na základe porovnávania dostupných riešení pre vývoj mobilných aplikácií bola zvolená technológia React Native pre implementáciu. React Native ponúka jednotný vývoj pre platformy Android a iOS, pre ktoré bude aplikácia dostupná.

Abstract

With the growth of their popularity, demand for mobile application continues to grow in various areas, including education institutions. The aim of this thesis is to design and implement a mobile application providing students of BUT easy access to their information. Based on analysis of available solutions for developing mobile applications, React Native was chosen an an implementaion technology. React Native offers development for both Anroid and iOS with a single codebase.

Kľúčové slová

Android, iOS, React Native, mobilná aplikácia

Keywords

Android, iOS, React Native, mobile application

Citácia

KLINOVSKÝ, Ondrej. *Mobilní aplikace pro studenty VUT*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Prof. Ing. Tomáš Hruška, CSc.

Mobilní aplikace pro studenty VUT

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením profesora Ing. Tomáša Hrušku, CSc. Ďalšie informácie mi poskytli pán Ing. Jaromír Marušinec a pán Bc. Jaroslav List. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Ondrej Klinovský
26. mája 2020

Poďakovanie

Rád by som poďakoval profesorovi Ing. Tomášovi Hruškovi, CSc. za vedenie tejto bakalárskej práce. Ďalej by som sa chcel poďakovať riaditeľovi CVIS Ing. Jaromírovi Marušincovi a najmä kolegovi Bc. Jaroslavovi Listovi za odbornú pomoc a cenné rady.

Obsah

Kapitola 1

Úvod

Málokto si vie v dnešnej dobe predstaviť život bez mobilných zariadení, ktoré ponúkajú nespočetné množstvo možností. Ľudia si s ich pomocou zjednodušujú a spríjemňujú svoj súkromný, ale aj pracovný život. O mobilné aplikácie majú čoraz väčší záujem aj spoločnosti, ktoré sa snažia posunúť svoj biznis na vyššiu úroveň. Mobilná aplikácia spoločnosť nielen zatraktívni, ale umožní jej efektívnejší spôsob komuníkácie so svojimi zákazníkmi. To sa týka tiež vysokých škôl, ktorých zákazníkmi sú najmä ľudia z vekovej kategórie 18-26, ktorá používa mobilné zariadenia asi najviac spomedzi ostatných. Mobilná aplikácia pre študentov vysokej školy môže nielen uľahčiť organizáciu a celkový priebeh ich štúdia, môže pomôcť aj vysokej škole so skvalitnením výučby a prilákaním viac študentov v budúcnosti.

Táto práca sa zaoberá návrhom, výberom technológie pre implementáciu a samotnou implementáciou mobilnej aplikácie pre platformy Android a iOS. Aplikácia bude slúžiť študentom fakúlt využívajúcich centrálny informačný systém, ktorý je spravovaný súčasťou VUT, CVIS (Centrum Výpočetných a Informačných Služeb VUT), pre ktorú je táto práca robená. Aplikácia má slúžiť ako odľahčená verzia existujúcej webovej aplikácie. Zmyslom tejto práce je teda vytvoriť mobilnú aplikáciu schopnú komunikovať so školským informačným systémom a poskytovať užívateľom aktuálne informácie. Hlavnými modulmi aplikácie budú elektronický index, osobný rozvrh a menzy. V rámci tejto práce budú pridané moduly aktuality z predmetov a vyhľadávanie osôb, ktoré budú v budúcnosti rozšírené o ďalšie moduly ako mapa VUT alebo prihlasovanie sa na skúšky.

V prvej kapitole budú predstavené súčasné technológie, ktoré sa používajú pre vývoj mobilných aplikácií. Pri každej technológii budú uvedené výhody a nevýhody vývoja ako aj príklady konkrétnych riešení. Kapitola 3 bude pozostávať z výberu vhodnej technológie pre implementáciu aplikácie a z predstavenia použitých knižníc, ktoré budú tvoriť hlavné časti aplikácie. Taktiež v nej budú popísané jednotlivé obrazovky aplikácie spolu s ich základným náčrtom. Na základe týchto náčrtov bol grafikom navrhnutý dizajn. Kapitola 4 popisuje implementáciu tohto dizajnu spolu s implementáciou funkcionalít ako je komunikácia so serverom vrátane prihlasovania sa, správa dát a ukladanie dát v internom úložisku. Posledná kapitola je venovaná užívateľskému testovaniu, kde na základe spätnej väzby od študentov VUT budú navrhnuté vylepšenia aplikácie.

Kapitola 2

Vývoj mobilných aplikácií

Častou otázkou pri vývoji mobilných aplikácií je, či je výhodnejšie vytvoriť aplikáciu napísanú priamo pre konkrétnu platformu a využiť jej plný potenciál, alebo použiť webové technológie. Ďalšou možnosťou vývoja je použitie technológie ako Cordova, ktorá ponúka hybridný spôsob vývoja, a teda využitie oboch natívnych aj webových prvkov. Existujú tiež platformy schopné generovať natívne aplikácie pre viac platforiem z jedného projektu bez využitia webových technológií.[20]

Pri výbere vhodnej technológie je potrebné vziať v úvahu viacero faktorov ovplyvňujúcich vývoj, ale aj následnú údržbu a rozširovanie projektu. Sú to najmä:

- Cena
- Testovanie
- Schopnosť portovania kódu na iné platformy
- Prístup k natívnym rozhraniam zariadení
- Konzistencia užívateľského rozhrania
- Distribúcia aplikácie a jej aktualizácií
- Výkon

Táto kapitola predstaví tri základné typy technológií používaných pre vývoj mobilných aplikácií:

- natívny
- webový
- hybridný

Ku každému typu bude uvedený príklad konkrétnej technológie a uvedené jej základné princípy a spôsob vývoja.

2.1 Natívne technológie

Najpoužívanejší spôsob vývoja mobilných aplikácií je natívny. Natívne aplikácie sú vyvíjané špeciálne pre danú platformu, pomocou programovacích jazykov a nástrojov, ktoré platforma podporuje. Ponúkajú najlepší výkon a optimalizáciu a reagujú svižnejšie na užívateľské vstupy v porovnaní s ostatnými technológiami. Vývojári majú prístup ku všetkým natívnym rozhraniam, ktoré operačný systém poskytuje. Na druhej strane vývoj aj údržba natívnej mobilnej aplikácie možu byť nákladnejšie, najmä ak má byť aplikácia dostupná na viacero platformách. Oprava chyby v aplikačnej logike alebo pridanie novej funkcionality bude vyžadovať písať viac krát to isté, len v inom jazyku. Preto tento prístup nemusí byť vhodný pre jednoduché aplikácie, ktoré nevyžadujú optimalizáciu a veľkú výpočetnú silu.[2]

2.1.1 Android

Android je open source operačný systém od spoločnosti Google postavený na jadre Linuxu. Ide o otvorený systém, čo umožňuje výrobcom ho použiť voľne vo svojich zariadeniach. Aplikácie pre tento systém sú dostupné v internetovom obchode Google Play.

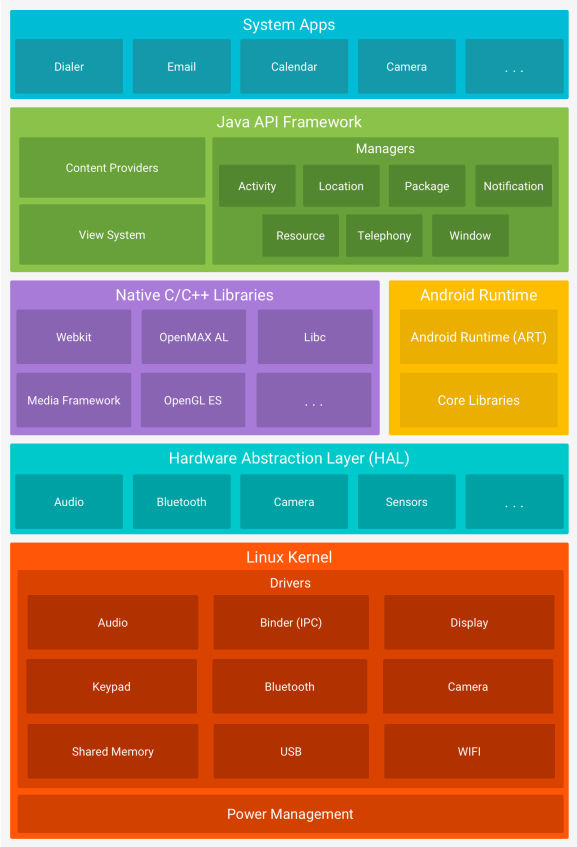
Systém

Systém Androidu je rozdelený na niekoľko vrstiev zobrazených na obrázku 2.1. Kernel, alebo tiež jadro, poskytuje funkcionality ako správu pamäte alebo procesov. Nad kernelom sa nachádza Hardware Abstraction Layer (HAL). Ten sprístupňuje hardware zariadenia vyšším urovniam (Java API framework). HAL pozostáva z modulov, kde každý modul implementuje rozhranie pre špecifický typ hardware (kamera, akcelerometer, atď.). Každá spustená aplikácia beží zároveň spolu s instanciou Android Runtime (ART). ART vykonáva bajtový kód, do ktorého sa kompilujú aplikácie písané pre Android, a spravuje pamäť aplikácie (Garbage Collection). Android poskytuje tiež natívne C/C++ knižnice, ktoré sprístupňujú funkcionalitu niektorých natívnych rozhraní aplikáciám. Je ňou napríklad knižnica OpenGL ES, s ktorou sa programuje 2D a 3D grafika. Všetka funkcionalita, ktorú Android ponúka je dostupná cez rozhranie napísané v programovacom jazyku Java (Java API Framework). Toto API ponúka následovnú funkcionalitu:

- View System - má na starosti tvorbu prvkov užívateľského rozhrania
- Resource manager - poskytuje prístup k statickým zdrojom aplikácie, napríklad k obrázkom alebo textu
- Notification Manager - zabezpečuje zobrazovanie notifikácií v notifikačnej lište
- Activity Manager - spravuje životnosť aktivít aplikácie
- Content Providers - spravuje prístup aplikácie k dátam ostatných aplikácií alebo zdieľanie dát iným aplikáciám

Vývoj

Pre vývoj Android aplikácií sa využívajú programovacie jazyky Java a Kotlin. Oba jazyky sú kompilované do Java bajtkódu, ktorý pre svoj beh potrebuje JVM (Java Virtual Machine). ART spĺňa úlohu JVM v Android zariadeniach. Pre vývoj aplikácií sa zvyčajne



Obr. 2.1: Úrovne abstrakcií systému Android. Prevziete z [14]

používa vývojové prostredie Android Studio poskytujúce všetky potrebné nástroje pre vývoj a testovanie. Tieto nástroje fungujú samostatne a Android Studio nie je nutnosť.

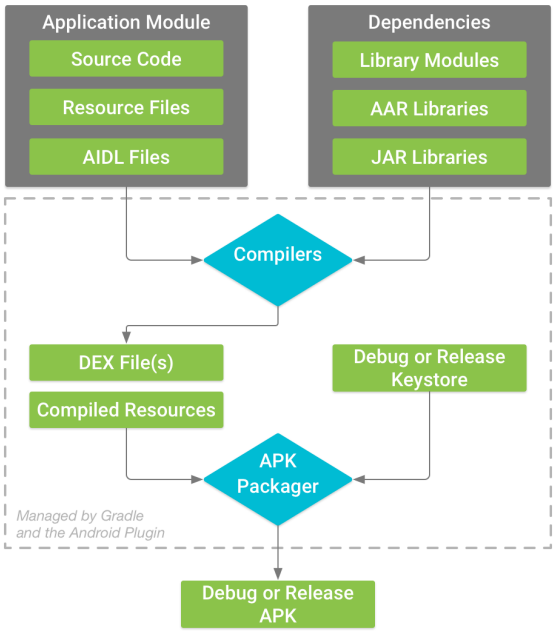
Zdrojové kódy a statické súbory sa zabaľujú do výsledného APK súboru. Android pre tento proces používa nástroj Gradle a pozostáva z následujúcich krokov:

1. Kompilátor zkompiluje zdrojový kód do DEX (Dalvik Executable) súborov obsahujúcich bajtkód.
2. APK Packager zabalí DEX a ostatné súbory do APK súboru. Pred naištalovaním na zariadenie musí byť však tento súbor podpísaný (signed).
3. Podpisovanie APK súborov má takisto na starosti APK Packager. Pre podpis potrebuje súbor zvaný keystore, ktorý slúži ako certifikát pre overenie vydvateľa aplikácie.[13]

Architektúra aplikácie

Každá aplikácia je tvorená komponentmi, ktoré možu byť následujúcich typov:

- Aktivita
- Service
- Broadcast receiver



Obr. 2.2: **Gradle.** Proces zostavovania APK súboru typickej Android aplikácie. Prevziate z [13]

- Content provider

Aktivita je vstupným bodom pre interakciu s užívateľom a poskytuje okno, v ktorom sa vykresľuje užívateľské rozhranie. Každá aktivita predstavuje zvyčajne jednu obrazovku aplikácie. Počas svojej existencie prechádza niekoľkými stavmi. Pre každý stav má definovanú metódu, ktorá je volaná systémom pri prechode do tohoto stavu:

- *onCreate()* - volaná po vytvorení aktivity. V tejto metóde by mala byť implementovaná logika, ktorú je potrebné vykonať len raz - na začiatku existencie aktivity. Je to jediná metóda, ktorú je potrebné vždy implementovať. Tiež je povinné v tejto metóde volať *setContentView()* pre definovanie rozloženia (layout) UI aktivity. Po skončení tejto metódy sa vždy zavolá metóda *onStart()*.
- *onStart()* - v tejto metóde by sa mala aktivita pripraviť prejsť do popredia a stať sa interaktívnou (inicializácia kódu aktualizujúceho UI, spustenie animácii, alebo pripravenie reakcií na užívateľské vstupy). Je volaná vždy po metóde *onCreate()* alebo *onRestart()*
- *onResume()* - potom, ako aktivita prejde do popredia, je volaná táto metóda. V tejto fáze je aktivita pripravená reagovať na užívateľské vstupy. Nachádza sa na vrchole zásobníku aktivít a zostane v tomto stave, dokým nedôjde k prerušeniu, čo sa môže stať napríklad pri prichádzajúcom hovore, navigovaní do inej aktivity alebo stlačením tlačítka späť. V tom prípade sa zavolá metóda *onPause()*.
- *onPause()* - indikuje, že užívateľ opušťa túto aktivitu a že aktivita nie je viac v popredí, ale stále môže byť viditeľná, preto nemusí ešte zastaviť aktualizáciu užívateľského rozhrania.

- *onStop()* - Aktivita nie je viac viditeľná pre užívateľa. Stáva sa to, keď je aktivita zrušená alebo ju prekryje iná. V tejto metóde by sa mali zastaviť všetky operácie potrebné pre aktualizáciu obrazovky ako napríklad zastavenie animácií. Keď sa aktivita vracia do popredia, volá sa metóda *onRestart()*, inak sa volá *onDestroy()*.
- *onRestart()* - aktivita sa opäť dostáva do popredia a je viditeľná.
- *onDestroy()* - volaná pred tým, ako je aktivita zničená. Väčšinou je implementovaná pre uvoľnenie používaných zdrojov.

Aktivitu je možné rozdeliť do viacero UI častí pomocou Fragmentov. Fragment sa dá predstaviť ako podaktivita s vlastným životným cyklom a metódami, ktoré sú však závislé od životnosti aktivity.

Service je komponent aplikácie umožňujúci vykonávať dlhé operácie na pozadí. Neposkytuje teda žiadne UI. Service aplikácie môže bežať aj keď sa užívateľ presunie do inej aplikácie. Príkladom dlhých operácií môže byť prehrávanie hudby alebo sťahovanie súborov. Existujú tri typy Service:

- Foreground - vykonáva operáciu viditeľnú pre užívateľa (prehrávanie hudby). Takáto Service musí zobrazovať notifikácie.
- Background - Operácie nie je priamo viditeľná pre užívateľa (ukladanie do interného úložiska)
- Bound - tento druh Service ponúka komunikáciu klient-server medzi Service a ostatnými komponentmi aplikácie.

Broadcast receiver je komponent umožňujúci aplikácii zachytávať a reagovať na udalosti vyvolané systémom. Napríklad, keď si užívateľ nastaví budík na následujúci deň, aplikácia nemusí byť počas tejto doby neustále spustnená. Vďaka Broadcast reciever sa spustí až keď dôjde k uplynutiu času.

Content provider spravuje zdieľané dáta aplikácie. Umožňuje iným aplikáciam čítať alebo modifikovať tieto dáta. Napríklad aplikácia Kontakty môže iným aplikáciám pomocou Content prividera dovoliť čítať zoznam kontaktov.

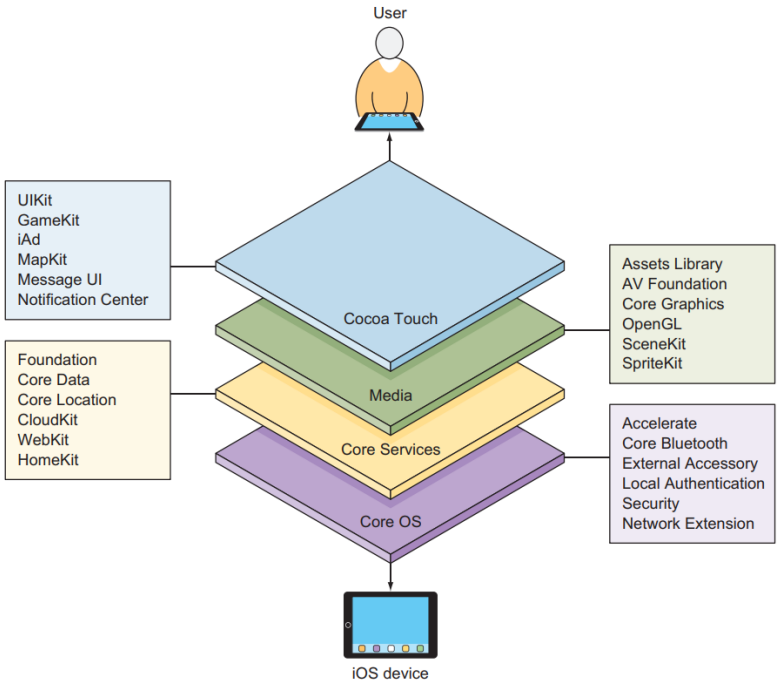
Pred spustením aplikácie, systém musí poznať zoznam všetkých komponentov danej aplikácie. K tomu slúži súbor AndroidManifest.xml, v ktorom sú deklarované všetky komponenty aplikácie. Okrem deklarovania komponentov slúži tento súbor k deklarovaniu minimálnej podporovanej verzie Androidu alebo systémových povolení potrebných pre beh aplikácie. [12]

2.1.2 iOS

iOS je mobilný operačný systém od firmy Apple postavený na ich desktopovom operačnom systéme OSX. Ide o uzavretý systém schopný fungovať len na zariadeniach Apple - iPhone alebo iPad. Aplikácie pre túto platformu sú dostupné prostredníctvom internetového obchodu App Store.

Systém

Architektúra systému iOS sa skladá zo štyroch vrstiev. Každá vrstva poskytuje niekoľko balíkov. Balíček pozostáva z dynamicky linkovanej knižnice spolu s hlavičkovými súbormi definujúcimi jej rohranie. Na najvyššej úrovni sa nachádza Cocoa Touch vrstva, ktorá poskytuje knižnice pre multitasking, spracovanie dotykov alebo notifikácie. Jej súčasťou je aj knižnica UIKit, ktorá sa používa pre tvorbu užívateľského rozhrania a obsahuje základný stavebný prvok iOS aplikácie - ViewController. Ďalšou vrstvou je Media vrstva. Nachádzajú sa v nej knižnice pre prácu s grafikou, zvukom a videom. Vrstva Core Services poskytuje aplikáciám kľúčovú systémovú funkcionalitu. Jedným z najdôležitejším balíčkom tejto vrstvy je Core Foundation obsahujúci kód v jazyku C pre prácu s textovými reťazcami, vláknami alebo kolekciami. Najspodnejšou vrstvou je vrstva Core OS. Pracuje s nízkoúrovňovou funkcionalitou ako konfigurácia siete, systém súborov (file system), alokácia pamäte, virtuálna pamäť, procesy a medziprocesová komunikácia. Obrázok 2.3 ukazuje jednotlivé vrstvy a balíčky, ktoré poskytujú.[15][17]



Obr. 2.3: Vrstvy systému iOS a balíčky, ktoré poskytujú. Prevziate z [15]

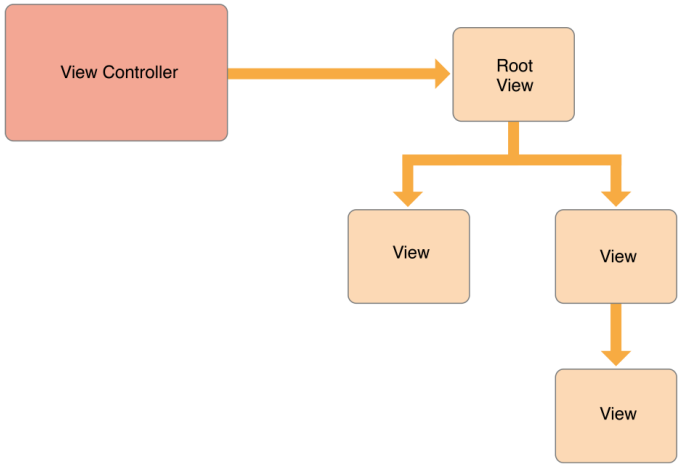
Vývoj

Pre vývoj mobilných aplikácií pre platformu iOS je nutné použiť operačný systém OSX a užívateľské prostredie XCode. Xcode pozostáva zo sady nástrojov využívaných pre vývoj, testovanie, optimalizáciu a nahratie aplikácie na App Store. Jedným z hlavným nástrojom, ktorý XCode ponúka, je Interface Builder. Jeho pomocou vývojár v grafickom prostredí vytvorí grafické prvky obrazovky aplikácie (Storyboard), ktoré následne napojí na ViewController.

Architektúra aplikácie

Základným stavebným prvkom iOS aplikácie je ViewController a každá aplikácia obsahuje minimálne jeden. ViewController predstavuje jednu obrazovku aplikácie a stará sa o všetko, čo sa na obrazovke deje - vykresľovanie užívateľského rozhrania, animácie, interakcia užívateľa alebo navigovanie. Každý ViewController obsahuje atribút *view*, ktorý obsahuje prvky užívateľského rozhrania zobrazené na obrazovke, napríklad UIButton (tlačítko) alebo UILabel (text). Každý element obsiahnutý vo ViewController je súčasťou stromovej hierarchie predstavujúcej UI (view hierarchy), kde atribút *view* predstavuje koreň. Viď obrázok 2.4.

Podobne ako Aktivity v Android aplikácii, aj ViewController má svoj životný cyklus a metódy, ktoré sa vykonávajú pri prechode z jedného stavu do druhého. Po načítaní view hierarchie do pamäte sa zavolá funkcia *viewDidLoad()*. Metódy *viewWillAppear()* a *viewDidAppear()* sú volané pred, resp. po zobrazení obrazovky, ktorú ViewController spravuje. Podobne fungujú metódy *viewWillDissapear()* a *viewDidDissapear()*.[23]



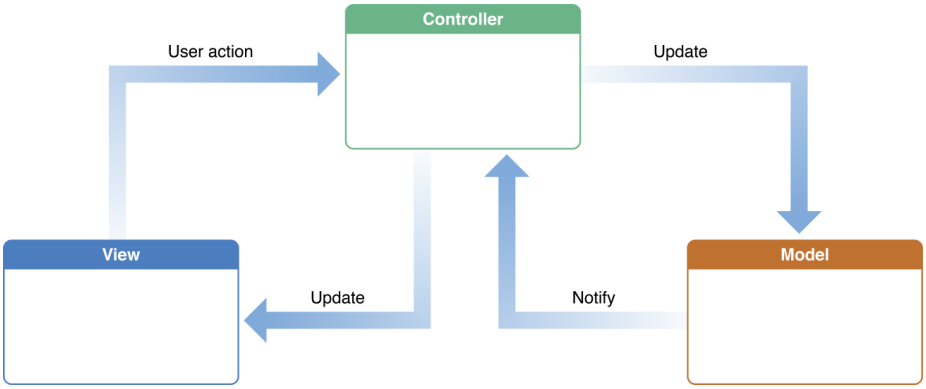
Obr. 2.4: **iOS MVC.** Model View Controller. Prevziate z [3]

Model-View-Controller (MVC) je návrhový vzor často používaný v iOS aplikáciách, ktoré rozdeľuje do troch kategórií:

1. Model - obaľuje dáta, s ktorými aplikácia pracuje a definuje logiku spracovávania týchto dát (parsovanie odpovedí od serveru, serializovanie pre interné uloženie). Model môže komunikovať s jedným alebo viac Controllermi, ale nemal by byť schopný priamo komunikovať so žiadnym View.

2. View - reprezentácia užívateľského rozhrania. Možu existovať samostatne alebo patriť do nejakej hierarchie. Príkladom View môže byť UIButton alebo koreňový View.

3. Controller - je sprostredkovávateľom medzi View a Modelom. Predáva dáta z Modelu do View a naopak. Controlleri tiež obsahujú logiku reagovania na užívateľské vstupy.[4]



Obr. 2.5: Model-View-Controller. Prevziate z [4]

2.2 Webové technológie

Mobilné webové aplikácie sú webové stránky optimalizované pre mobilné zariadenia. Sú dostupné na zariadeniach, ktoré majú webový prehliadač a internetové pripojenie. Nemusia byť stiahnuté z internetového obchodu ako natívne aplikácie. Sú vyvíjané pomocou webových technológií (HTML, CSS, JavaScript). Ich najväčšou výhodou je nenákladný vývoj a údržba, keďže vďaka webovým prehliadačom netreba vyvíjať pre každú podporovanú platformu zvlášť projekt. Aktualizácie webovej aplikácie nemusia prechádzať cez interetový obchod, najnovšia verzia sa načíta, keď si užívateľ otvorí webovú aplikáciu. Webová aplikácia je dobrou voľbou pre jednoduché aplikácie, kde nie sú potrebné natívne rozhrania zariadení. Keďže pre svoj beh potrebujú webový prehliadač, neponúkajú tak dobrú optimalizáciu a výkon ako natívne aplikácie.[2]

2.2.1 React

React je open-source JavaScriptová knižnica vyvíjaná firmou Facebook a používaná pre vývoj interaktívnych webových aplikácií. React umožňuje vytvárať užívateľské rozhrania skladaním menších, znovupoužitelných celkov nazývaných komponenty. React používa výhradne JavaScript, tradičné HTML šablóny sú nahradené technológiou JSX (JavaScript eXtension), ktorá umožňuje písať HTML kód priamo v JavaScripte, vďaka čomu programátor získava väčšiu kontrolu nad výsledným vzhľadom komponentu. [10]

Virtuálny DOM

Jednou z najdôležitejších vlastností Reactu je virtuálny DOM (Domain Object Model). DOM je stromová štruktúra reprezentujúca grafické prvky webovej stránky. Modifikácia tohto stromu je výpočetne náročná operácia a je problémom veľkého množstva aplikácií, kde prebieha príliš často a veľa krát zbytočne. U menších aplikácií toto nie je až taký problém, avšak u väčších projektoch môže dochádzať k zhoršeniu užívateľského zážitku. Na riešenie tohto problému prichádza React s virtuálnym DOMom. Virtuálny DOM je odľahčená kópia skutočného DOMu, ktorú si React aplikácia uchováva počas svojej existencie. Keď dôjde k zmene užívateľského rozhrania (užívateľ klikne na tlačítko), React vytvorí novú kópiu DOMu reflektujúcu túto zmenu. Následne sú tieto dve kópie porovnané. Uzly, ktoré sú zmenené, sú prekreslené v reálnom DOMe. Týmto so výrazne zmenší počet aktualizácií DOMu a výpočetná náročnosť aplikácie klesá.

JSX

JSX (JavaScript eXtension) je syntaktické rozšírenie JavaScriptu a je najviac používaný práve v knižnici React pre popis vzhľadu užívateľského rozhrania. JSX predstavuje zjednodušenie kódu (syntactic sugar) umožnením písania JavaScript a HTML kódu v jednom súbore. React je zástancom toho, že vykresľovacia logika je úzko spätá s logikou reagovania na vstupy užívateľov alebo zmenami stavu aplikácie.

Následujúci JSX kód:

```
const element = (
  <MyButton color="blue" shadowSize={2}>
       Click Me
  </MyButton>
)
```

bude preložený do:

```
const element = React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Komponenty

React komponenty umožňujú rozdeliť užívateľské rozhranie do menších, samostatných celkov, ktoré možno využívať na viacero miestach aplikácie. Komponenty sú podobné JavaScriptovským funkciám. Ako vstup prijímajú objekt zvaný *props*, ktorý obsahuje popis vlastností komponentu. Návratovou hodnotou je React element popisujúci čo sa má na obrazovke zobraziť. Komponent sa dá definovať ako funkcia (funkcionálny komponent):

```
const Welcome = (props) => {
  return <h1>Hello, {props.name}</h1>;
}
```

alebo ako trieda (triedny komponent):

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Výhodou triednych komponentov je možnosť uchovávať si stav. Okrem toho ponúkajú tzv. metódy životného cyklu (lifecycle methods), ktoré sú volané v určitých okamžikoch počas existencie komponentu, podobne ako u Aktivity alebo ViewControlleru. Sú nimi napríklad metódy *componentDidMount* alebo *componentWillUnmount*, ktoré sú volané po prvom vykreslení komponentu, resp. po odstránení komponentu z DOMu. Metóda *componentDidUpdate* sa volá vždy po akutializácii komponentu. Tieto metódy sú veľmi často používané pri vytváraní vedľajších efektov (napríklad načítanie dát zo serveru alebo zmena URL v navigačnej lište prehliadača.

Hooks

Až do verzie Reactu 16.8., funkcionálne komponenty neumožňovali uchovávať si stav alebo reagovať na udalosti životného cyklu, a preto bolo nutné používať pre tieto účely triedne komponenty. S príchodom React hooks sú funkcionálne komponenty schopné triedne komponenty plne nahradiť. Asi dva najpoužívanejšie hooky sú *useState* a *useEffect*. *useState* sa používa pre vytvorenie lokálneho stavu komponentu, jeho použitie:

```
const [value, setValue] = React.useState("initial value")
```

Premenná *value* predstavuje hodnotu súčasného stavu, *setValue* je funkcia, pomocou ktorej sa mení hodnota premennej *value*. Funkcia *React.useState* berie ako parameter počiatočnú hodnotu, teda hodnotu použitú pri prvom vykreslení komponentu. *useEffect* sa používa pre spravovanie vedľajších efektov miesto metód *componentDidMount*, *componentDidUpdate* alebo *componentWillUnmount*.

```
React.useEffect(() => {
  doSideEffect()
}, [dep1, dep2])
```

Hook *useEffect* prijíma dva vstupné argumenty: bezparametrovú funkciu s návratovým typom void, v ktorej dochádza k vedľajšiemu efektu, a pole závislostí obsahujúce premenné, na ktorých je tento efekt závislý. Počas vykresľovecej fázy komponentu sa porovnajú hodnoty závislostí súčasného a predošlého stavu komponentu a ak sa aspoň jedna zmenila, funkcia sa zavolá hneď po vykreslení komponentu.

Jednosmerný tok dát

Tok dát v React aplikácii je jednosmerný. React vytvára hierarchiu komponentov, kde je každý komponent zavislý od svojho rodičovského komponentu a vlastného stavu. Dáta sú predávané od rodiča svojim potomkom. V prípade, že aplikačná logika vyžaduje závislosť rodiča od jeho potomka, je potrebné, aby rodič predal potomkovi funkciu (tzv. callback), ktorou potomok pošle dáta svojmu rodičovi. Komponenty sú schopné predávať dáta svojim potomkom pomocou objektu *props*, ktorý predstavuje vstupný argument komponentu, na základe ktorého môžme meniť jeho vzhľad alebo správanie. To nám umožňuje jednoduchú delegáciu stavu aplikácie komponentom, na druhú stranu predávanie *props* od koreňového komponentu až po koncový môže byť pri väčších aplikáciách nepraktické, preto vznikli knižnice ako Redux alebo MobX, ktoré riešia tento problém.

Vývoj

Pri vývoji webových JavaScriptových aplikácií sa väčšinou používa NPM (Node Package Manager), správca balíčkov pre platformu Node.js, ktorý pomáha JavaScript vývojárom so zdieľaním JavaScriptových knižníc. Každý projekt využívajúci NPM obsahuje v koreňovom adresári súbor package.json, ktorý predstavuje manifest projektu. Nachádzajú sa v ňom informácia o názve a verzii projektu alebo zoznam JavaScriptových balíčkov spolu s ich verziami, na ktorých je projekt závislý. Pre ladenie a odchytávanie chýb aplikácií sa používajú vývojárske nástroje obsiahnuté vo webovom prehliadačí Google Chrome alebo Mozzila Firefox.

2.3 Hybridné technológie

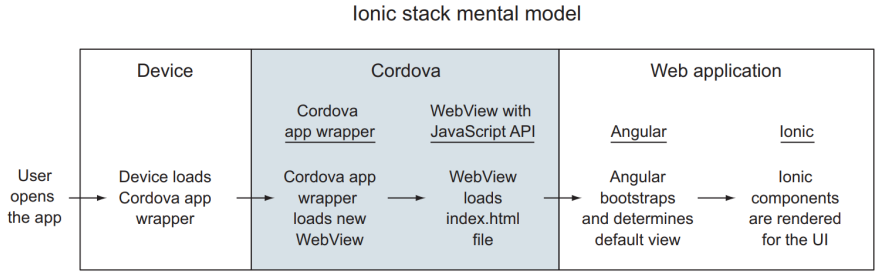
Hybridné aplikácie sú kombináciou webovových a natívnych. Ponúkajú mechanizmus zabalenia webovej aplikácie do natívnej, ktorého výsledkom je aplikácia navonok vyzerajúca ako natívna - dá sa nainštalovať rovnakým spôsobom ako natívne a majú prístup k natívným rozhraniam zariadenia. Natívna aplikácia, v ktorej je webová aplikácia zabalená, sa nazýva WebView - odľahčený webový prehliadač bez klasického užívateľského rozhrania ako url lišta alebo záložky. WebView poskytuje prístup k natívnym rozhraniam zariadenia, vďaka čomu môže webová aplikácia pracovať s kamerou alebo akcelerometrom. Tento prístup je značne obmedzený v dôsledku zachovania kompatibility naprieč rôznymi zariadeniami. U hybridných aplikácií je rovnaký problém s výkonom ako u webových, keďže stále ide o webovú stránku bežiacu v prehliadači.[2]

2.3.1 Apache Cordova

Cordova je populárny open source framework, poskytuje dva dôležité komponety pre vývoj hybridných aplikácií. Prvým je natívna aplikácia - WebView, v ktorom beží webová aplikácia. Druhým komponentom je sada pluginov, cez ktoré JavaScriptový kód dokáže komunikovať s natívnymi rozhraniami. Každý plugin má na starosti komunikáciu práve s jedným natívnym rozhraním. Cordova poskytuje sadu základných pluginov zvanú Core Plugins, ktoré poskytujú prístup k základným funkcionalitám zariadenie ako kamera, batéria alebo kontakty. Cordova však neposkytuje žiadne komponenty pre tvorbu užívateľských rozhraní. Vývojári môžu použit svoje vlastné komponenty alebo použiť existujúci framework. Jedným z najpopulánejších je Ionic.[1]

2.3.2 Ionic

Ionic kombinuje niekoľko technológii a nástrojov za účelom vytvárania hybridných aplikácií pre rôzne platformy. Je postavený na technológiách Cordova a Angular. Cordova poskytuje prostredie, v ktorom Ionic aplikácia beží. Angular je JavaScriptový framework používaný pre vývoj komplexných webových aplikácií. Jeho primárnou úlohou je riadiť logiku aplikácie. Ionic poskytuje komponenty užívateľského rozhrania, ktoré ponúkajú takmer natívny vzhľad. V novších verziách sa Ionic neviaže len na Angular, ale je možné ho použit s inými knižnicami, napríklad s knižnicou React. [24]



Obr. 2.6: Ionic využíva framework Cordova pre komunikáciu s natívnymi rozhraniami. Angular má na starosti aplikačnú logiku. Pevziate z [24]

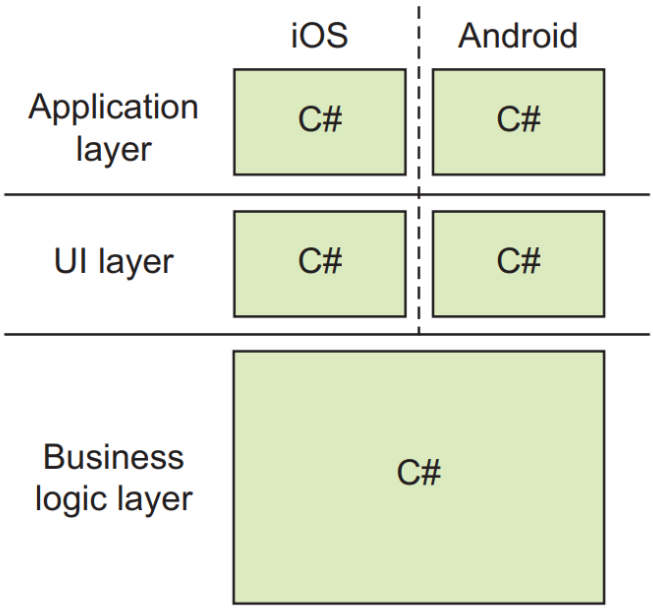
2.3.3 Vývoj

Pre vytvorenie Ionic projektu je potrebné použiť Ionic CLI (Command Line Interface), sadu nástrojov používaných v príkazovom riadku. Ako alternatívu ku Cordove je možné použiť Capacitor, ktorý funguje na podobnom princípe a je spätne kompatibilný s väčšinou Cordova pluginov. Keďže ide o webovú aplikáciu, Ionic CLI používa NPM pre správu dependecií projektu. Pre vytvorenie Android je potrebné nainštalovať všetky potrebné nástroje, ktoré vyžaduje natívny projekt. V prípade iOS je nutné mať k disopzícii zariadenie s operačným systémom OSX a nainštalovaným XCode.

2.4 Xamarin

Xamarin je framework od spoločnosti Microsoft umožňujúci vyvíjať mobilné aplikácie pre Android a iOS so zdieľaným zdrojovým kódom a s využitím platformy .NET. Platforma .NET poskytuje sadu knižníc a nástrojov pre vývoj aplikácií pre web alebo pre operačný systém Windows. Používa programovací jazyk C#, v ktorom sa píšu aj Xamarin aplikácie.

Xamarin poskytuje sadu tried, ktoré zabaľujú natívne rozhrania iOS a Androidu, vďaka čomu umožňuje pracovať s tymíto rozhraniami v jazyku C#. V Android aplikácii má správu nad obrazovkou na starosti trieda Activity. Vďaka zabaleniu tejto triedy do C# triedy je možné spravovať obrazovku priamo zo C# kódu. Podobne funguje aj UIViewController v iOS aplikácii. Aplikačnú logiku Xamarin aplikácie je možné zdieľať medzi platformami. Logiku pre interakciu s UI je však potrebné písať pre každú platformu zvlášť, ale dá sa to docieliť za použitia jediného programovacieho jazyka.



Obr. 2.7: Xamarin aplikácie sú písané v C#, vďaka čomu môže byť aplikačná logika zdieľaná medzi platformami. Prevziate z [6]

Ďalšou výhodou Xamarinu je kompilácia. Pri iOS platforme sa využíva AOT (Ahead of Time) kompilátor, ktorý kompiluje zdrojové kódy priamo do strojového kódu. Pri Androide sa využíva JIT (Just in Time) kompilátor, kedy sa najskôr zkompiluje zdrojový kód do medzikódu (Imtermediary code) a do strojového kódu sa kompiluje až za behu aplikácie.

Vývoj

Pre vývoj Xamarin aplikácií je najideálnejšie použiť operačný systém Windows s nainštalovaným vývojárskym prostredím Visual Studio, ktoré obsahuje všetky potrebné nástroje, vrátane Xamarinu. Vďaka otvorenosti Androidu je možné zostaviť aplikácie pre túto platformu priamo vo Visual Studiu, ktoré poskytuje aj niekoľko emulátorov Android zariadení. U platformy iOS je to komplikovanejšie a pre vytvorenie aplikácie treba mať k dispozícii zariadenie s OSX, ktoré je možné prepojiť s Visual Studiom a diaľkovo zostavovať aplikácie. Emulátory iOS je možné použiť cez zdieľanú obrazovku. [6]

2.5 React Native

React Native (RN) je framework umožňujúci vývojárom vytvárať natívne aplikácie s použitím JavaScriptu a knižnice React, popísanej v kapitole 2.2.1. Používa teda všetku funkcionalitu Reactu ako JSX, virtuálny DOM a komponenty. RN používajú vo svojich aplikáciách spoločnosti ako Facebook, Bloomberg, Tesla alebo Walmart. Od Ostatných JavaScriptových riešení ako napríklad Cordova sa líši tým, že miesto vykresľovania UI prvkov vo WebView používa natívne rozhrania a komponenty poskytované operačným systémom. Preto RN aplikácie pôsobia na užívateľa ako natívne aplikácie.

RN pozostáva z React komponentov, kde každý komponent reprezentuje korespondenčné natívne komponenty pre Android i iOS. Napríklad RN komponent *Text* sa pre Android zariadenie vykreslí ako *TextView* a pre iOS ako *UITextView*. Vďaka tomu je vývojárovi umožnené písať kód rovnako ako v hociakej inej React aplikácii, kde výstupom bude natívna aplikácia. [11]

RN aplikácia pozostáva z troch častí:

1. Natívny kód a moduly napísané v jazykoch Objective-C alebo Swift pre iOS alebo Java pre Android. Pokiaľ sa nepoužíva externá natívna RN knižnica, táto časť kódu je zväčša nezmenená.

2. JavaScript VM (Virtual Machine) vykonáva JavaScriptový kód. RN používa v oboch platformách JavaScriptový engine JavaScriptCore, ktorý je používaný vo webovom prehliadači Safari od firmy Apple. Pri iOS zariadeniach RN využíva JavaScriptCore poskytovaný systémom, pri Android zariadeniach je potrebné JavaScriptCore zabaliť spolu s aplikáciou, preto môže mať Android verzia väčšiu veľkosť o niekoľko megabajtov.

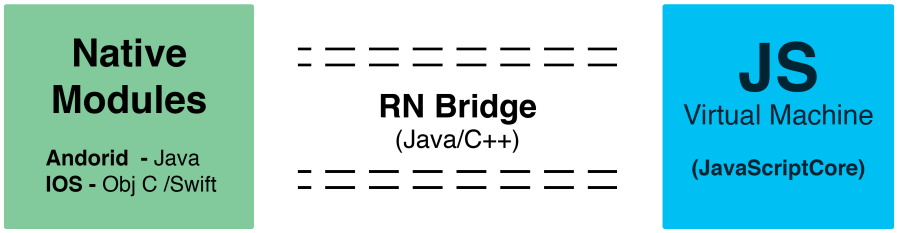
3. RN Bridge je takzvaný most napísaný v jazykoch C++ a Java slúžiaci pre komunikáciu medzi natívnym a JavaScriptovým vláknom.

Pri spustení RN aplikácie sa vytvorí niekoľko vlákien:

1. Hlavné vlákno sa spustí ako prvé a následne spúšťa JavaScriptové. Zachytáva všetky udalosti užívateľského rozhrania (dotyky), ktoré posiela JavaScriptovému vláknu cez RN Bridge.

2. JavaScriptové vlákno vykonáva JavaScriptový kód a teda v ňom beží celá logika aplikácie.

3. Vlastné natívne moduly majú každý svoje vlastné vlákno



Obr. 2.8: Schéma architektúry React Native aplikácie. JS Virtual Machine, v ktorom beží JavaScriptový kód, komunikuje s natívnymi rozhraniami cez tzv. Bridge. Previaze z [21]

Vývoj

React Native aplikácie sa píšu v jazyku JavaScript, pre správu JavaScriptových balíčkov sa využíva nástroj NPM. React Native ponúka react-native-cli balíček, ktorý pomáha pri vývoji so spúšťaním aplikácie v emulátore alebo na fyzikom zariadení. React aplikáciu je možné spustiť v debugger móde, kedy miesto bežania na mobilnom zariadení beží vo webovom prehliadači, vďaka čomu je možné použiť vývojárske nástroje, ktoré prehliadač poskytuje. Pre vývoj aplikácie pre Android je potrebné naištalovať všetky potrebné nástroje, ktoré táto platforma vyžaduje. iOS vývoj vyžaduje zariadenie s OSX a XCode.

Kapitola 3

Návrh aplikácie

Táto kapitola sa zaoberá v prvej časti výberom vhodnej technológie pre implementáciu aplikácie. Kapitola 3.2 predstaví knižnice, ktoré boli použité pri implementácii. Návrh užívateľského rozhrania aplikácie spolu s výberom navigácie bude popísaný v kapitole 3.3.

3.1 Výber technológie

Pri výbere vhodnej technológie bola zohľadňovaná najmä jednotná implementácia pre obe platformy. To vylúčilo možnosť písať aplikáciu s natívnymi technológiami. Aplikácia nebude vykonávať náročné operácie, preto optimalizácia a výkon nie sú až tak dôležité. Pre možnosť fungovania v offline režime musí mať aplikácia prístup k internému úložisku zariadenia. Je preto potrebné zvoliť technológiu schopnú komunikovať s natívnymi rozhraniami, čo vylučuje webové aplikácie. Pre svižnejšie reakcie na užívateľské vstupy som sa rozhodol použiť riešenie umožňujúce vytvárať natívne aplikácie, preto som vylúčil aj framework Ionic. Aplikácie napísané v Xamarine majú o niečo lepší výkon než aplikácie v React Native vďaka kompilovanému kódu. Avšak React Native je na rozdiel od Xamarinu kompletne open source a má niekoľkonásobne väčšiu komunitu, vďaka čomu existuje veľa dostupných a pravidelne udržovaných knižníc. Po konzultáciách s profesionálnymi vývojármi som sa rozhodol použiť pre implementáciu tejto aplikácie React Native.

3.2 Použité knižnice

Výhodou JavaScriptu je jeho vyspelý ekosystém punúkajúci veľké množstvo knižníc a nástrojov pre budovanie rýchlych, stabilných a udržiavateľných aplikácií. Ako implementačný jazyk bol zvolený TypeScript, ktorého výhody sú popísané v následujúcej kapitole. Pre tvorbu užívateľského rozhrania bola použitá knižnica React predstavená v kapitole 2.2.1. Práca s touto knižnicou bude popísaná v kapitole 4.4. Pre prácu s dátami, ktoré bude aplikácia prezentovať, bola použitá knižnica MobX, ktorej je venovaná kapitola 3.2.2.

3.2.1 TypeScript

TypeScript (TS) je programovací jazyk vyvíjaný firmou Microsoft, ktorý obohacuje JavaScript o statickú typovú kontrolu. Je nadmnožinou JavaScriptu, preto každý validný kód napísaný v JavaScripte je validný TS kód, avšak naopak to neplatí, preto je TS kompilovaný do JavaScriptu. Deklarovanie dátových typov v zdrojovom kóde umožňuje kompilátoru detekovať chyby ešte pred spustením aplikácie. Napríklad chyba predania argumentu nesprávneho

typu funkcii alebo nekontrolovanie *null* hodnoty premennej (nullcheck) sa detekuje už pri kompilácii a nie až po spustení aplikácie. TS môže vďaka tomu ušetriť vývojárom mnoho času pri vývoji. Dátové typy ponúkajú tiež istú formu dokumentácie kódu. Pre konfiguráciu TS kompilátoru slúži súbor tsconfig.json, ktorý môže vyzerať následovne:

```
"compilerOptions": {
  "target": "es2015",
  "lib": ["es2018", "es2019"],
  "noImplicitAny": true,
  "noUnusedLocals": true,
}
```

Položka *target* udáva, do ktorej verzie JavaScriptu sa má TS kód kompilovať. *lib* pridáva funkcionalitu budúcich verzií JavaScriptu. Ak sa chce zamedziť používaniu premenných bez deklarovaného dátového typu, *noImplicitAny* tomu zabráni. *noUnusedLocals* zobrazí varovanie ak sa v kóde nachádzajú nepoužité premenné.[18]

3.2.2 MobX

MobX je JavaScriptová knižnica inšpirovaná objektovo orientovaným a reaktívnym programovaním, uľahčujúca prácu so stavmi v JavaScriptových aplikáciách. Stav je srdcom aplikácie a často dochádza k jeho nekonzistencii, kedy lokálne premenné obsahujú iné, neaktualizované hodnoty, ako tie v globálnom stave aplikácie. Veľa riešení spočíva v obmedzení práce so stavmi, napríklad knižnica Redux neumožňuje priamu modifikáciu stavu, čím sa stáva pre programátora stav nemenným (immutable). Tým ale predstavuje nový problém - dáta musia byť normalizované a referenčná integrita nie je zaručená.

MobX rieši tento problém tým, že nedovolí, aby mohol vzniknúť nekonzistentný stav. Jeho stratégia je: všetko, čo sa dá odvodiť od stavu, bude odvodené, automaticky. Mobx má 4 základné koncepty:

1. Pozorovateľný stav - Mobx pridáva dátovým štruktúram vlastnosti pozorovateľných objektov (observables). Objekty sa stávajú pozorovateľnými označením JavaScriptovým dekorátorom *@observable*.

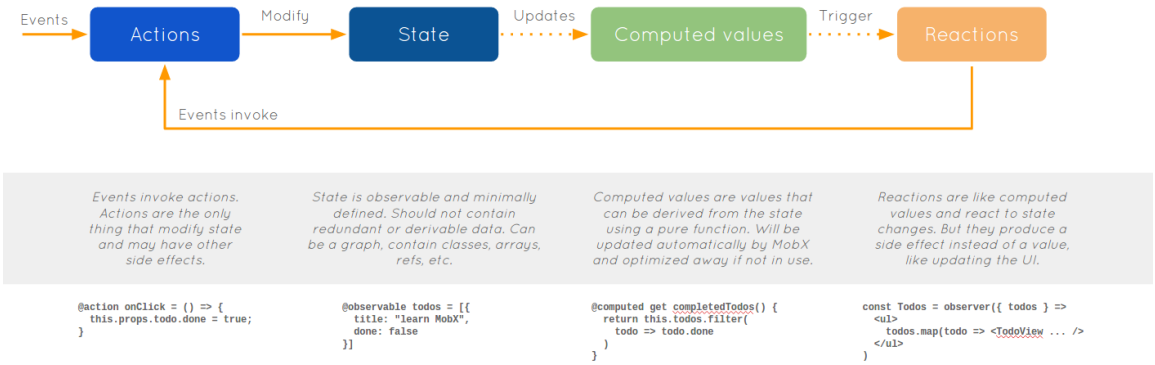
2. Odvodené hodnoty - Mobx umožňuje definovať hodnoty, ktoré budú automaticky odvodené, keď sú relevantné dáta modifikované. Tieto hodnoty sú označené dekorátorom *@computed*. Sú to väčšinou metódy využívajúce atribúty triedy pre výpočet odvodenej hodnoty. Tieto metódy by nemali nikdy modifikovať pozorovaneľné hodnoty - mali by byť čistými funkciami (pure functions), teda bez vedľajších efektov.

3. Akcie - funkcie modifikujúce pozorovateľný stav aplikácie. Sú označené dekorátorom *@action*.

4. Reakcie - podobné ako odvodené hodnoty reagujú na zmeny pozorovateľných hodnôt, ale miesto odvodzovania novej hodnoty vytvárajú vedľajšie efekty ako sú napríklad poslanie dát cez sieť, modifikovanie UI alebo pozorovateľných hodnôt. Reakcie sa vytvárajú pomocou funkcií *autorun*, *reaction* alebo *when*. Môžu byť vyvolávateľom akcií.

Mobx podporuje jednosmerný tok dát, kde akcie modifikujú stav aplikácie, na základe ktorého sa následne aktualizujú pozorovatelia. Odvodené hodnoty sú vypočítané automaticky, atomicky a synchrónne, vďaka čomu nedochádza k pozorovaniu medzivýpočtových

hodnôt a hodnoty sú ihneď k dispozícií. Mali by to byť čisté (pure) funkcie - neprodukujúce žiadne vedľajšie efekty (nemali by napríklad modifikovať stav aplikácie).



Obr. 3.1: Knižnica MobX podporuje jednosmerný tok dát. Prevziate z [19]

MobX Úložiská (stores)

Úložisko je obdobou kontroléru v architektúre MVC, ktorého hlavnou zodpovednosťou je presun logiky z UI komponentov do samostatných testovateľných jednotiek, ktoré sú nezávislé od zobrazovacej technológie. Väčšina aplikácií obsahuje aspoň dve úložiská: jedno pre stav UI a druhé - doménové - pre samotné dáta aplikácie. Úložisko UI neobsahuje veľa logiky, ale skôr informácie o užívateľskom rozhraní - informácie, ktoré sa neukladajú na backende:

- Jazyk aplikácie
- Téma aplikácie
- Aktuálna obrazovka
- Viditeľnosť modálneho okna

Doménové úložiská majú na starosti dáta, s ktorými aplikácia pracuje. V rámci tejto aplikácie bude existovať niekoľko doménových úložísk spravujúcich následovné domény:

- Štúdium
- Rozvrh
- Menzy
- Profil a osoby
- Aktuality z predmetov

Tieto úložiská majú následovné úlohy:

- Inicializácia dát
- Komunikácia s backendom

- Poskytovanie dát React komponentom
- Ukladanie dát do interného úložiska

MobX-React

MobX je možné použiť s hociakou JavaScriptovou knižnicou alebo len s obyčajným JavaScriptom. Pre integráciu s Reactom slúži knižnica MobX-React, ktorá poskytuje nástroje pre tvorbu reaktívnych React komponentov. K tomu slúži funkcia *observer*, ktorou sa zabalí komponent reagujúci na zmenu pozorovateľných dát - React komponenty spĺňajú úlohu reakcií.[19]

3.2.3 React Native Navigation

Pre implementáciu navigácie aplikácie bola použitá knižnica React Native Navigation (RNN) od spoločnosti Wix. RNN je implementovaná pre Android aj iOS natívne, čo má výhodu presunutia viac práce z JS vlákna do natívneho. Keďže natívne vlákno je podstatne rýchlejšie než JS, docieli sa tým značná optimalizácia aplikácie pri prechodoch medzi jednotlivými obrazovkami. Nevýhodou navigácií implementovaných v JS vlákne je aktualizovanie UI stromu pri každom prechode medzi obrazovkou, čo môže zaťažiť toto vlákno a narušiť užívateľský zážitok. V RNN predstavuje každá obrazovka samostatný UI strom, ktorý ma menšiu veľkosť a nedochádza k jeho zmene pri navigovaní.

RNN umožňuje nastaviť pomocou funkcie *setRoot* následovné rozloženia navigácie:

- Component - jediný komponent bez možnosti zobrazenia iných.
- Stack - zásobník obrazoviek s možnosťou vkladania nových obrazoviek na vrchol zásobníku pomocou funkcie *push* alebo ich odoberania pomocou tlačítka späť alebo funkcie *pop*.
- BottomTabs - tvorený niekoľkými potomkami (tabmi), ktoré môžu mať rozloženie Component alebo Stack
- SideMenu - tvorený potomkom center, ktorý predstavuje centrálnu obrazovku v navigácie bočného menu. Center môže byť jediný Component alebo Stack. Samotné bočné menu je reprezentované potomkom left, resp. right, ktoré može mať rozloženie Component.

Pre použitie React komponentu ako RNN obrazovky je potrebné tento komponent registrovať funkciou *registerComponent* pod určitým názvom. Následne sa tento názov použije pri konfigurácií rozloženia Component. Následujúca ukážka kódu demonštruje použitie funkcie *registerComponent* pre registráciu React komponentu ScreenComponent ako RNN obrazovky s názvom screen. Následne sa nástaví rozloženie navigácie na Stack, ktorý bude obsahovať na vrchole zásobníku práve túto obrazovku.[25]

```
Navigation.registerComponent("screen", () => ScreenComponent)

Navigation.setRoot({
  root: {
    stack: {
      children: [
        component: {
          name: "screen"
        }
      ]
    }
  }
})
```

3.3 Užívateľské rozhranie

Rozdiel medzi dobrým a zlým užívateľským rozhraním sa posudzuje podľa kvality užívateľského zážitku, alebo tiež UX (User Experience). Užívatelia majú veľké očakávania od mobilných aplikácií: rýchle načítanie dát, jednoduché a príjemné používanie. Pre úspešnú aplikáciu je potreba brať dizajn ako najdôležitejšiu časť aplikácie.

Táto kapitola je venovaná návrhu užívateľského rozhrania aplikácie. V prvej časti bude venovaná výberu vhodnej navigácie. V kapitole 3.3.2 sú popísané hlavné obrazovky aplikácie a základný náčrt ich vzhľadu.

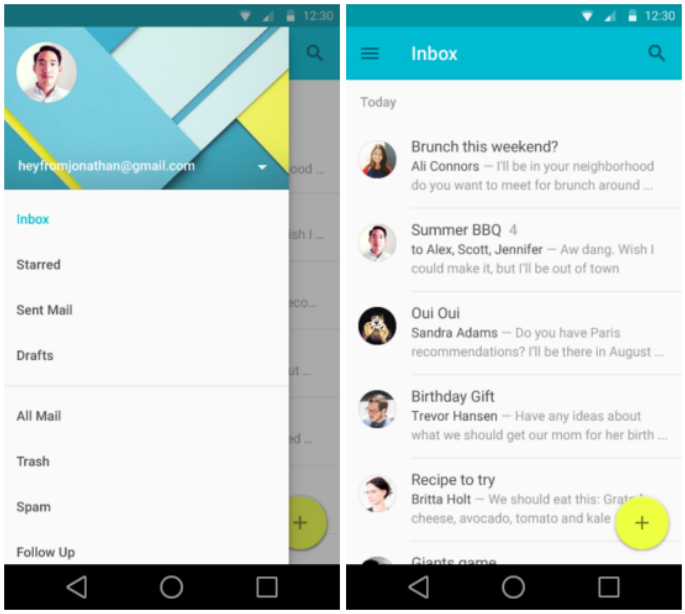
3.3.1 Navigácia

Ak aplikácia ponúka viacero obrazoviek, medzi ktorými sa môže užívateľ pohybovať, je potrebný mechanizmus, korý mu to jednoducho umožní. Navigácia v aplikácii by mala byť intuitívna a prediktívna. Noví užívatelia by mali byť schopní sa v aplikácii jednoducho pohybovať. Je zložité vytvoriť ľahko objavitelnú a dostupnú navigáciu pre zariadenia s malou obrazovkou, kde treba vyvážiť obsah aplikácie s navigačnými prvkami. V následujúcich riadkoch budú predstavené dva druhy navigácií, ich výhody a nevýhody a výber tej vhodnejšej pre túto aplikáciu.

Bočné menu

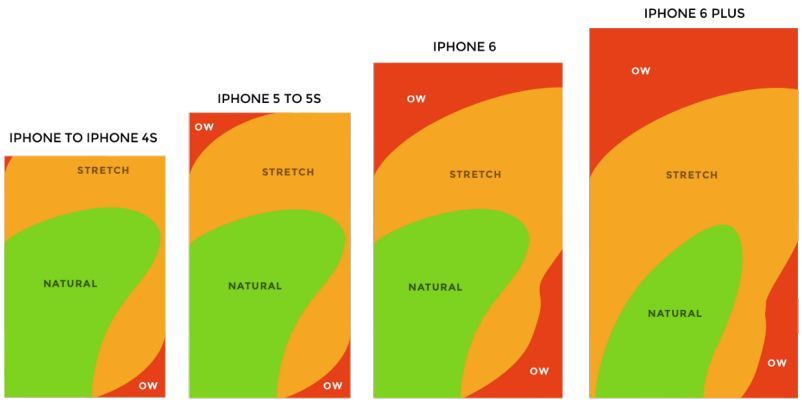
Bočné menu (sidebar menu, prezývané tiež hamburger menu) je jedným z najpopulárnejších navigácií používaných v mobilných aplikáciách. Bočné menu umožňuje skryť navigáciu za ľavý okraj obrazovky a v prípade potreby ho vysunúť späť, ako je ukázané na obrázku 3.2. V niektorých aplikáciách sa môže nachádzať na pravej strane obrazovky. Táto navigačná varianta je populárna v Android aplikáciách, kde sa stala štandardom.

Najväčšou výhodou tejto navigácie je veľké množstvo navigačných možností, ktoré je možné do vysúvacieho panelu umiestniť. Vďaka tomuto panelu sa ušetrí viac miesta pre obsah aplikácie. Ukrytá navigácia môže znamenať horšiu dostupnosť navigácie, čo dokazujú užívateľské testovania, ktoré hovoria, že užívatelia použijú skrytú navigáciu menej ako tú viditeľnú.[16] Nevýhodou tejto navigácie môže byť kolízia s navigačnými prvkami platformy, čo sa často stáva u aplikácií iOS, kedy sa vedľa menu ikony zobrazí ikona späť. Tým, že menu ikona sa nachádza v ľavom hornom rohu obrazovky môže byť problém u zariadení



Obr. 3.2: Ukážka aplikácie používajúcej bočné menu ako hlavnú navigáciu. Prevziate z [5]

s veľkým displejom, ktorých v dnešnej dobe nie je málo. Obrázok 3.3 ukazuje, že poloha menu ikony sa nachádza v ťažko dosiahnuteľnej oblasti obrazovky pre užívateľa držiaceho mobil v pravej ruke.

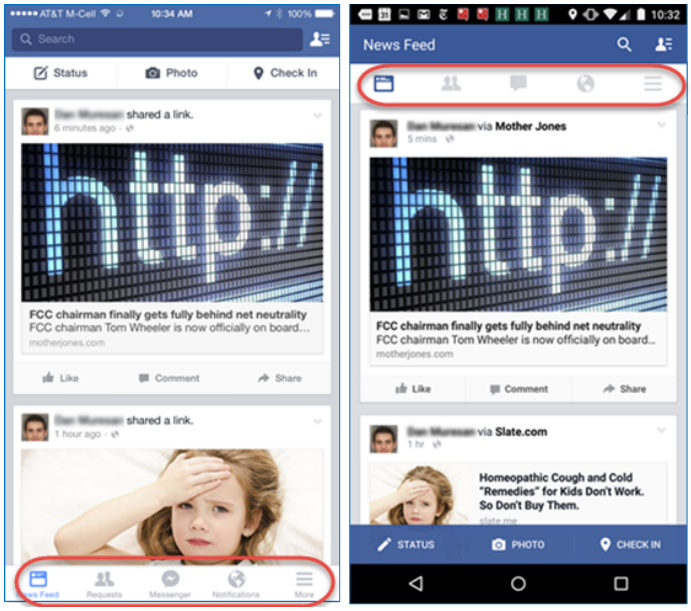


Obr. 3.3: Porovnanie dostupnosti oblastí displeja roznej veľkosti pri držaní zariadenia v pravej ruke. Prevziate z [22]

Tab bar

Tab bar je varianta navigácie pochádzajúca z desktopového dizajnu. Je viac rozšírená u iOS zariadení, kde sa väčšinou nachádza v spodnej časti obrazovky. U Android aplikácií je častejšie vidieť tab bary nachádzajúce sa v hornej časti obrazovky. Rozdiel medzi tymíto dvomi plaformami je možné vidieť na obrázku 3.4, ktorý porovnáva aplikácie Facebook pre iOS a Android. Tab bar obsahuje relatívne málo navigačných možností - tabov, ktoré predstavujú hlavné časti aplikácie a zo spodného panelu sú užívateľovi vždy dostupné. Pri

správnom prevedení dokáže byť tabbar veľmi intuitívna a jednoduchá navigácia. Vďaka dobre navrhnutým tabom sa užívateľ dokáže ľahko v aplikácií zorientovať. Taby užívateľovi napovedajú, ktoré sú hlavné časti aplikácie. Nevýhodou tabbaru môže byť obmedzený počet tabov, väčšinou 3-5. Viac tabov by mohlo spôsobiť problémy s výberom. Tabbar taktiež zaberá viac miesta ako bočné menu, čo môže byť problém u zariadení s menšími obrazovkami.[5][7]



Obr. 3.4: Porovnanie polohy tab baru aplikácie Facebook na zariadení iOS a Android. Prevziate z [7]

Výber navigácie

Pre potreby tejto aplikácie som sa rozhodol použiť tab bar ako hlavnú navigáciu. Vďaka tabom bude možné vybrať tri hlavné moduly aplikácie, ktoré bude mať užívateľ vždy viditelné a dostupné. Tab bar bude síce zaberať viac miesta ako bočné menu, avšak v súčasnosti prevládajú mobily s veľkými obrazovkami, preto by s nedostatkom miesta nemal byť problém. Práve veľké obrazovky by spôsobovali užívateľom ťažkosti pri výbere bočného menu, ktoré je ťažko dosiahnuteľné.

3.3.2 Návrh obrazoviek aplikácie

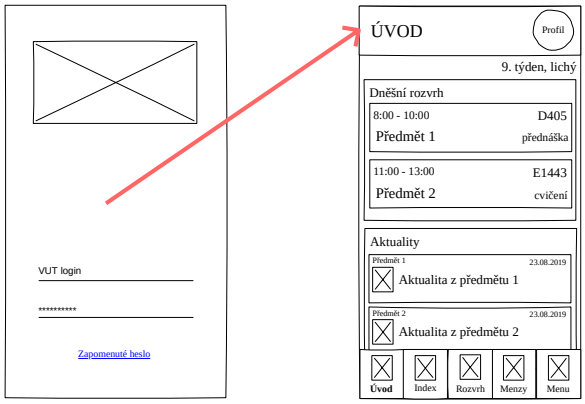
Aplikácia je určená pre študentov VUT, ktorí študujú na fakultách využívajúcich centrálny informačný systém. Cieľom je vytvoriť aplikáciu, ktorá bude študentovi poskytovať rýchlo a prehľadne aktuálne informácie o jeho štúdiu. Na základe štatistík nazbieraných z používania webovej aplikácie školského systému boli vybrané hlavné moduly aplikácie elektronický index, osobný rozvrh a menzy. Tieto tri moduly budú súčasťou spodného panelu, aby ich mal užívateľ vždy dostupné. Okrem toho sa bude v spodnej lište nachádzať úvodná obrazovka a menu, kde budú umiestnené nastavenia aplikácie a vedľajšie moduly.

Prihlasovacia obrazovka

Obrazovka zobrazená po prvom spustení aplikácie bude ponúkať jednoduché rozhranie s dvomi textovými poliami pre užívateľske meno a heslo. Pod nimi sa bude nachádzať odkaz na webovú stránku, ktorá užívateľovi poskytne pokyny v prípade problémov s prihlásením. Ako experiment minimalistického dizajnu som sa rozhodol neumiestniť sem tlačítko pre prihlásenie. Užívateľ potvrdí prihlásenie po napísaní hesla a kliknutím na potvrdzovacie tlačítko, ktoré je súčasťou klávesnice. Užívateľské testovanie ukáže, ako budú na to užívatelia reagovať.

Úvodná obrazovka

Obsah tejto obrazovky bude prvá vec, ktorú užívateľ uvidí po prihlásení sa do aplikácie. Mali by sa tu nachádzať informácie, ktoré bude užívateľ potrebovať vidieť v čo najkratšom možnom čase. Do tejto kategórie nepochybne patrí rozvrh pre aktuány deň. Užívateľ by mal hneď po otvorení aplikácie zistiť, kedy a kde má najbližšie vyučovanie. K tomu bude slúžiť agenda, krátky zoznam (2-3 položky) vyučovacích blokov, ktoré študenta v najbližšej dobe čakajú. Pri každom bloku nesmie chýbať miesto vyučovania a čas, za ktorý sa vyučovanie začína. Po kliknutí na daný blok sa užívateľ presunie do detailu vyučovania, kde bude mať k dispozícií všetky relevantné informácie o výučbe. Pre lepšiu orientáciu študenta v aktuálnom semestri bude zobrazený aj aktuálny týždeň s poradovým číslom a paritou. Užívateľ by mal mať aj prehľad o aktuánom dianí vo svojich zapísaných predmetoch. Vďaka zoznamu aktualít z predmetov nachádzajúci sa pod agendou by mu nemala újsť žiadna dôležitá informácia o výučbe.

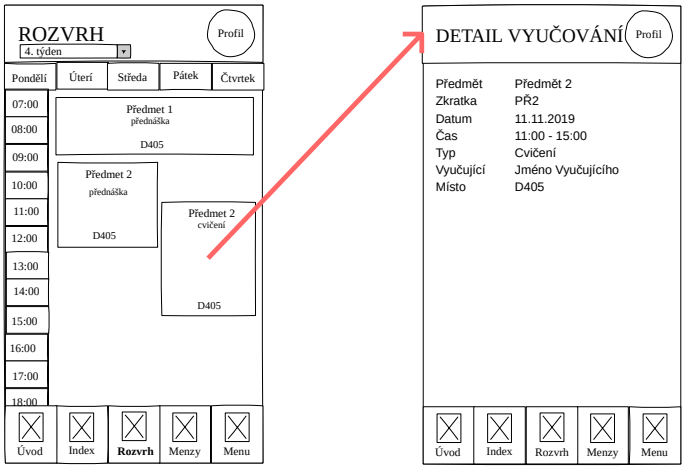


Obr. 3.5: Náčrt prihlasovacej a úvodnej obrazovky

Osobný rozvrh

Spolu s elektronickým indexom by toto mala byť najpoužívanejšia obrazovka v aplikácii, ktorá bude užívateľovi zobrazovať rozvrh s vertikálnou osou reprezentujúcou hodiny v rámci jedného dňa a horizontálnou reprezentujúcou dni v týždni. Jednotlivé vyučovacie bloky budú mať rôznu veľkosť v závislosti na dobe ich trvania a kolízií s ostatnými blokmi v rovnakom čase. V prípade kolízie dvoch blokov bude ich šírka zredukovaná na polovicu. Užívateľ bude mať k dispozícii dva režimy zobrazenia rozvrhu. V týždennom zobrazení bude môcť vidieť rozvrh pre celý týždeň. Pre šetrenie miesta sa víkendové dni budú zobrazovať len v prípade,

že má užívateľ vtedy výuku. Pri kolíziách v rozvrhu nemusí byť týždenné zobrazenie ideálne, preto bude mať užívateľ možnosť prepnúť na denné zobrazenie. Bloky budú obsahovať opäť len dôležité informácie ako tomu je v úvodnej obrazovke, teda názov predmetu, miesto a čas. Takisto bude možné prejsť do detailu vyučovania po kliknutí na daný blok.



Obr. 3.6: Náčrt obrazoviek osobného rozvrhu a detailu vyučovania.

Nie vždy užívateľ dokáže zo svojho rozvrhu vyčítať všetky potrebné informácie. Najmä ak jeho rozvrh obsahuje množstvo kolízií a jednotlivé karty vyučovaní sú príliš malé na zobrazenie všetkých relevantných informácií ako napríklad miesto vyučovania. Obrazovka detailu vyučévania ponúka všetky informácie o danej výuke. Odtiaľto sa bude schopný dostať do detailu predmetu alebo detailu vyučujúceho.

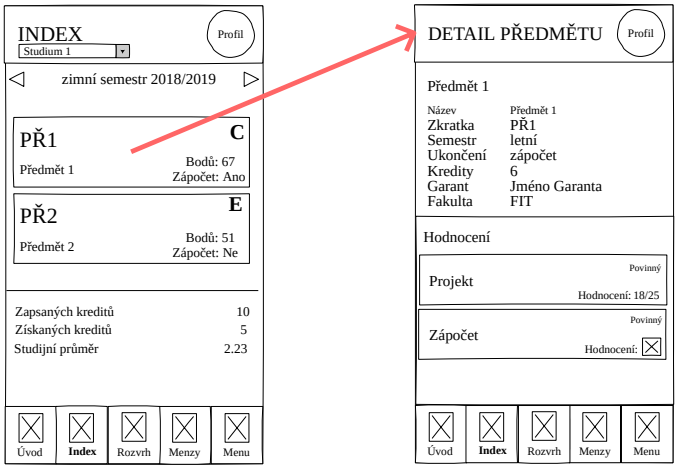
Elektronický index

Užívateľ v tejto obrazovke nájde zoznam predmetov zapísaných v aktuálnom semestri. O každom predmete budú zobrazené základné informácie ako názov a zkratka predmetu, počet kreditov a bodov, stav zápočtu a skúšky. Užívateľ si bude môcť takisto pozrieť predošlé semestre posúvaním sa do strán. Pod každým semestrom sa budú zobrazovať jeho štatistiky ako počet zapísaných a získaných kreditov. Po kliknutí na predmet bude užívateľ presunutý do detailu daného predmetu. Pre užívateľov, ktorí majú viacero štúdií (minulých aj aktívnych) bude umožnené prepínanie jenotlivých štúdií.

Detail predmetu ponúka užívateľovi pozrieť si okrem základných informácií o predmete aj zoznam hodnotení, ktoré užívateľ v tomto predmete získal. Môžu to byť hodnotené projekty, zápočty alebo semestrálne škúšky.

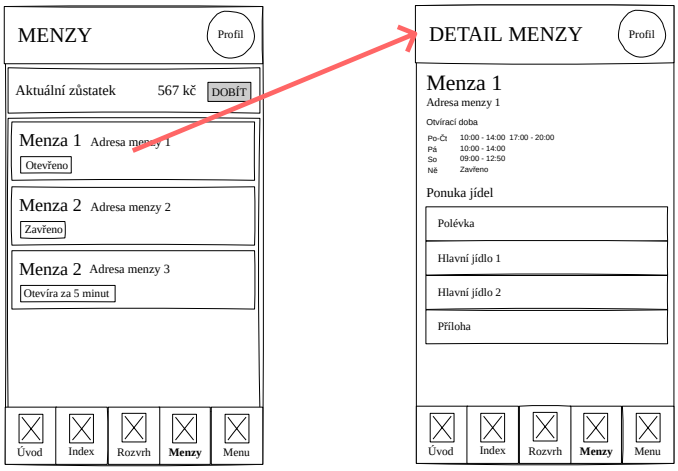
Menzy

Menzy sa síce netýkajú štúdia, ale nepochybne patria do každodenného života mnoho študentov. Obrazovka bude v podobnom duchu ako obrazovka elektronického indexu. Hlavnou časťou bude vertikálny zoznam obsahujúci karty menz. Každá karta bude zobrazovať názov menzy, jej addresu a súčasný stav. Karta bude môcť informovať za ako dlhu dobu menza otvára alebo zatvára. Kliknutím na kartu sa užívateľ presunie do detailu menzy, kde budú okrem informácii na karte aj otváracia doba a hlavne aktuálna ponuka jedál menzy.



Obr. 3.7: Náčrt obrazoviek elektronického indexu a detailu predmetu.

Nad zoznamom menz bude umiestnený aktuálny zostatok na účte menz spolu s nabíjacím tlačítkom odkazujúcim na platobnú bránu.



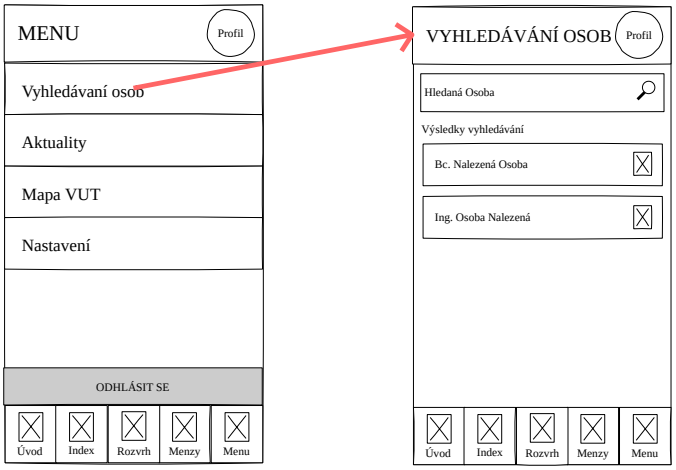
Obr. 3.8: Náčrt obrazoviek menz a detailu menzy.

Menu

Posledná obrazovka v spodnej lište bude slúžiť ako menu, obsahujúce sekundárne obrazovky spolu s možnosťou odhásenia sa. Užívateľ sa cez túto obrazovku dostane do vyhľadávania osôb, aktualít alebo nastavení. Táto obrazovka ponúka dosť miesta pre budúce rozšírenia aplikácie ako napríklad mapa VUT alebo prihlasovanie sa na skúšky.

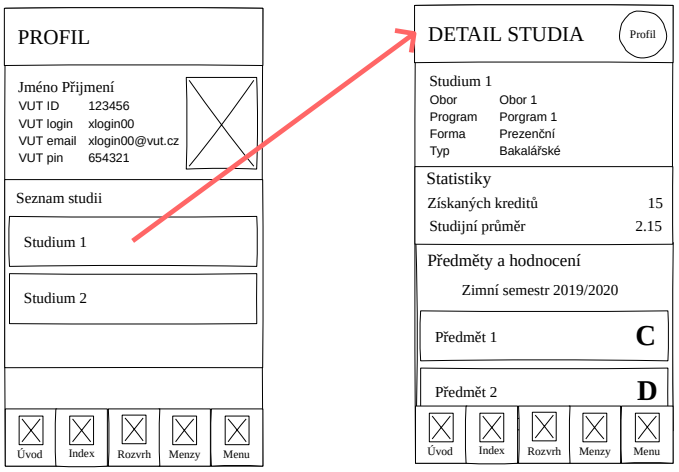
Profil

Do tejto obrazovky sa užívateľ dostane po kliknutí na svoju fotku, ktorá sa nachádza v pravej časti hornej navigačnej lišty. Užívateľ si v nej nájde svoje osobné informácie ako VUT ID, email alebo pin. Pod tymíto informáciami sa bude nachádzať zoznam užívateľových všetkých



Obr. 3.9: Menu obsahuje všetky sekundárne moduly aplikácie. Budúce rozšírenia aplikácie budú umiestnené na tejto obrazovke.

štúdií na VUT s možnosťou zobrazenia detailu každého štúdia. V detaily štúdia nájde užívateľ tri sekcie. Zákadné informácie budú zobrazovať obor, stupeň, titul alebo program štúdia. Druhá sekcia bude zobrazovať štatistiky štúdia ako vážený študijný priemer alebo percentil v rámci ročníku. Pod nimi bude výpis všetkých predmetov štúdia rozdelených do semestrov.



Obr. 3.10: Náčrt obrazovky profilu a detailu štúdia.

Kapitola 4

Implementácia

Táto kapitola sa zaoberá implementáciou mobilnej aplikácie. V prvej kapitole bude popísaná adresárová štruktúra projektu. V kapitole 4.2 bude predstavená práca s knižnicou MobX, ktorá má na starosti správu dát aplikácie. Kapitola 4.3 popisuje komunikáciu so serverom. Napokon bude v kapitole 4.4 popísaná štruktúra vybraných obrazoviek aplikácie.

4.1 Štruktúra projektu

Tento projekt obsahuje typickú štruktúru používanú v RN projektoch. V koreňovom adresári sa nachádzajú štyri hlavné zložky:

1. **android** - konfiguračné súbory Gradle pre zostavenie APK súborov alebo zdrojové súbory napísané v jazyku Java. S týmto adresárom sa pracuje pri konfigurácií výsledného APK súboru alebo pri pridávaní natívnych modulov tretích strán (napríklad RNN) do projektu.

2. **ios** - zdrojové súbory jazyku Objective-C alebo Swift a konfiguračné súbory pre XCode. Podobne ako u Androidu sa v ňom konfiguruje zostavovanie aplikácie a integrácie natívnych knižníc tretích strán.

3. **app** - RN projekt, popísaný nižšie.

V koreňovom adresári sa okrem nich nachádza aj vstupný bod JS aplikácie - index.js, package.json pre konfiguráciu NPM alebo tsconfig.ts pre konfiguráciu jazyka TypeScript.

Adresár app obsahuje následujúce zložky:

- **assets** - ikony a fonty, s ktorými aplikácia pracuje, sa nachádzajú v tomto adresári.

- **components** - React komponenty. Generické komponenty ako VutButton alebo SectionHeader sú umiestnené v koreni adresára. Špecializované komponenty sú rozdelené do jednotlivých domén: napríklad ScheduleView v podadresári schedule alebo CanteenCard v canteens.

- **hooks** - vlastné hooky, napríklad useStores hook umožňje React komponentu, ktorý ho využije, prístup k MobX úložiskám

- **i18n** - logika pre detekovanie jazyka zariadenia a české a anglické preklady.

- **models** - definície dátových štruktúr
- **modules** - moduly pre prácu s interným úložiskom alebo ikonami.
- **screens** - React komponenty predstavujúce jednotlivé obrazovky aplikácie.
- **services** - triedy zabezpečujúce komunikáciu s backendom, ktoré sú popísané v kapitole 4.3.
- **stores** - Mobx úložiská
- **utils** - pomocné funkcie pre prácu s časom a pre transformácie dát zo serveru na dátové štruktúry definované v models.

4.2 Práca s dátami

Jadro aplikácie tvoria MobX úložiská, ktorých úlohou je poskytovať dáta získané zo serveru React komponentom. Sú to:

- **UIStore** - správa užívateľského rozhrania, jazyka alebo aktuálneho času.
- **LoginStore** - prihlasovanie
- **StudyStore** - štúdiá, index a osobný orzvrh
- **ScheduleStore** - rozvrhy osôb a predmetov
- **CanteenStore** - menzy
- **ProfileStore** - profil študenta
- **PersonStore** - hľadané osoby
- **NewsStore** - aktuality z predmetov

Pre získanie dát zo serveru využívajú úložiská instanciu triedy VutService, ktorá bude popísaná v kapitole 4.3.3. Každé úložisko spravuje dáta svojej domény. Dáta získané zo serveru sú zabalené do dátovej štruktúry zvanej DataWraper. Tato štruktra obsahuje tri položky:

1. **data** - samotné dáta.
2. **error** - chyba pri načítavaní dát.
3. **state** - stav dát. Pri spustení aplikácie obsahuje hodnotu INITIAL. Pred počatím načitávania sa zmení na LOADING a po úspešnom alebo neúspešnom načítaní je nastavená na SUCCESS alebo ERROR.

Dáta zo serveru uložené v úložiskách predstavujú pozorovateľné objekty. Tie sú použité pre výpočet odvodených hodnôt, na ktorých zmeny následne reagujú React komponenty vykresľujúce užívateľské rozhranie. Príkladom môže byť zoznam štúdií obsahujúci aktívne aj neaktívne štúdia. Odvodenou hodnotou bude zoznam len aktívnych študií.

Ukladanie dát do interného úložiska

Pre ukladanie dát do interného úložiska je použitá knižnica AsyncStorage[1], ktorá bola súčasťou React Native, ale pre rýchlejší vývoj sa ju vývojári rozhodli presunúť do samostatnej knižnice. AsyncStorage poskytuje rozhranie s rovnakým názvom obsahujúce funkcie *getItem(key: string): Promise<string | null>* a *setItem(key: string, value: string): Promise<void>* slúžiace pre načítanie a ukladanie dát do interného úložiska. Signatúry funkcií ukazujú, že dáta sa budú ukladať vo forme textových reťazcov, preto je nutné pred uložením dáta serializovať funkciou *JSON.stringify* a po načítaní deserializovať funkciou *JSON.parse*.

4.3 Komunikácia so serverom

Aplikácia komunikuje so serverovou časťou pomocou protokolu WebSocket za účelom získavania všetkých potrebných dát. O prácu s WebSocketom sa stará trieda WebSocketHandler, s ktorým MobX úložiská komunikujú cez triedu VutService.

4.3.1 WebSocket

WebSocket je komunikačný protokol poskytujúci spojenie medzi klientom a serverom pomocou protokolu TCP (Transmission Control Protocol). Narozdiel do HTTP (Hyber Text Transfer Protocol), ktoré je bezstavým protokolom, WebSocket si udržiava spojenie medzi klientom a serverom po celú dobu komunikácie. Hlavným benefitom je zníženie réžie na strane klieta aj serveru, keďže spojenie je potrebné vybudovať len raz. HTTP protokol je použitý ako inicializačný protokol, následne je klientovi aj serveru umožnené posielanie správ asynchrónne bez nutnosti cyklu požiadavka/odpoveď.

V JavaScripte je protokol WebSocket sprístupnený cez triedu WebSocket. Táto trieda obsahuje 4 typy poslucháčov udalostí (event listeners):

- **onopen** - otvorenie spojenia
- **onmessage** - prijatie správy
- **onerror** - chyba spojenia
- **onclose** - uzatvorenie spojenia

Poslucháč udalostí je funkcia, ktorá sa vyvolá zakaždým, keď vznikne udalosť, na ktorú načúva. Funkcia *onopen* sa vyvolá pri každom otvorení spojenia rovnako tak sa funkcia *onmessage* vyvolá pri každej prichádzajúcej správe. Tieto funkcie si môže programátor definovať podľa seba a tým režírovať priebeh pripojenia.

Ako klient bude aplikácia inicializovať vytvorenie spojenia so serverom. Spojenie sa vytvorí zavolaním koštruktoru triedy WebSocket pomocou kľúčoveho slova *new*. Ako parametre berie URL adresu serveru, ku ktorému sa pripája a názov použitého protokolu, na ktorom sa musia obe strany predom dohodnúť. Pre poslanie správy slúži metóda *send*, ktorá posiela dáta v textovej forme, preto je nutné ich serializovať pomocou *JSON.stringify*. Spojenie sa uzavrie zavolaním metódy *close*.[9]

[1] https://github.com/react-native-community/async-storage

4.3.2 WebSocketHandler

Úlohou triedy WebSocketHandler (WSH) je spravovať pripojenie so serverom pomocou protokolu WebSocket. Pri mobilných aplikáciách, ktoré nemusia mať vždy stabilné sieťové pripojenie, môže byť spojenie často prerušované. WebSocket neposkytuje mechanizmus automatického znovupripojenia, preto bolo nutné túto logiku implementovať. Pomocou knižnice React Native NetInfo[2] je WSH schopný zistiť stav internetového pripojenia a nebude sa pokúšať pripojiť k serveru, ak je internet nedostupný. WSH obsahuje dve premenné reprezentujúce skutočný a žiadaný stav pripojenia, ktoré môžu nadobúdať následujúce hodnoty:

- CLOSING
- OPEN
- CONNECTING
- CLOSED
- LOGGING_IN
- LOGGED_IN

Úlohou WSH je zabezpečiť, aby skutočný stav bol vždy zhodný so žiadaným. Žiadaný stav sa mení volaním niektorej s následujúcich metód:

- connect
- disconnect
- logIn
- logOut
- sendRequest

Tieto metódy sú volané na základe interakcie užívateľa s aplikáciou. Po prvom spustení sa užívateľ dostane do prihlasovacej obrazovky, kedy sa zavolá metóda *connect* a žiadaný stav nadobudne hodnotu *OPEN*. Užívateľ sa nachádza v prihlasovacej obrazovke a očakáva sa, že sa bude chcieť prihlásiť. Ak je internet dostupný, WSH naviaže spojenie so serverom a po úspešnom pripojení sa skutočný stav zmení z *CLOSED* na *OPEN* - zhoduje sa so žiadaným. Po zadaní mena a hesla a potvrdení sa zavolá funkcia *logIn* a žiadaný stav sa zmení na *LOGGED_IN*. Na server sa odošle požiadavka prihlásenia a v prípade úspechu sa skutočný stav zmení na *LOGGED_IN* a užívateľ je prihlásený, inak žiadaný stav opäť nadobúda hodnotu OPEN a užívateľ je vyzvaný opäť zadať meno a heslo. Pri výpadku internetového pripojenia skutočný stav nadobudne hodnotu *CLOSED*. WSH sa pokúsi o prihlásenie (žiadaný stav má stále hodnotu *LOGGED_IN*) až keď mu NetInfo predá informáciu o dostupnosti k internetu, kedy WSH automaticky naviaže spojenie so serverom a prihlási sa.

Požiadavky posielané serveru môžu byž následujúcich typov:

- **query** - väčsina požiadavkov, ktoré aplikácia posiela na server sú tohoto typu. Odpovede na tieto požiadavku sú získané priamo z databáze.

[2] https://github.com/react-native-community/react-native-netinfo

- **http** - nie všetky dáta pochádzajú z jedného zdroja. Menzy a rozvrh pochádzajú z webových rozhraní, preto typ požiadavky pre tieto dáta je http.
- **file** - V prípade súborov ako sú napríklad profilové fotky študentov a zamestnancov je použitý tento typ požiadavku.

Prihlasovanie

Aplikácia sa na server prihlasuje pomocou mena a hesla zadaného od užívateľa a zostane prihlásená až do ukončenia spojenia. Aby nebolo nutné sa opätovne prihlasovať pomocou mena a hesla, aplikácia si do interného úložiska uloží tzv. prihlasovací token, ktorý jej pošle server. Tento token bude následne využívať pre prihlasovanie sa na server. V súčasnom stave server tento token neposiela, preto ako dočasné riešenie sa ako prihlasovací token používa práve meno s heslom. Z hľadiska bezpečnosti je to nevhodné riešenie, preto ho bude treba nahradiť skutočným tokenom. Pre prístup do aplikácie v režime offline je potrebné, aby bol tento token uložený v zariadení.

4.3.3 VutService

Táto trieda slúži ako rozhranie triedy WebSocketHandler, s ktorou komunikujú všetky MobX úložiská. VutService poskytuje metódy ako *fetchCanteens()* alebo *fetchStudies()*, ktoré poskytujú abstrakciu nad vytváraním požiadaviek, ktoré sa posielajú na server. Prijaté dáta sú transformované na dátové štruktúry, s ktorými dokážu úložiská jednoducho manipulovať. Ide väčšinou o transformáciu z plochých záznamov na štruktúrované dáta. Príkladom je transformácia zoznamu predmetov celého štúdia na zoznam semestrov, kde každý semester obsahuje zoznam predmetov.

Keďže sa jedná o sieťovú komunikáciu, treba riešiť problém asynchrónnych udalostí, kedy tieto metódy nevrátia hodnotu okamžite, ale až po prijatí odpovedi zo serveru. JavaScript ponúka triedu zvanú Promise používanú práve na prácu s asynchronnými udalosťami.

Promise je obálka nad dátami, ktoré nie sú známe v dobe jeho vytvárania. Môže sa nachádzať v stavoch:

1. pending - čakajúci
2. fulfilled - splnený
3. rejected - došlo k chybe[8]

Úložisku vyžadujúcemu dáta sa po odoslaní požiadavky na server vráti instancia Promise, ktorá je v stave *pending*. Úložisko pre tento Promise definuje funkcie *onSuccess* a *onError* reagujúce na úspech (uložením a zobrazením získaných dát), resp. na neúspech (informovaním užívateľa o chybe) tejto požiadavky. Ak dorazí odpoveď zo serveru v poriadku, Promise prejde do stavu fulfilled a je vyvolaná funkcia *onSuccess*, v opačnom prípade prejde do stavu rejected a vyvolaná bude funcia *onError*. Následujúca ukážka kódu znázorňuje volanie metódy *fetchStudies()* v úložisku.

```
vutService.fetchStudies()
        .then(onSuccess)
        .catch(onError)
```

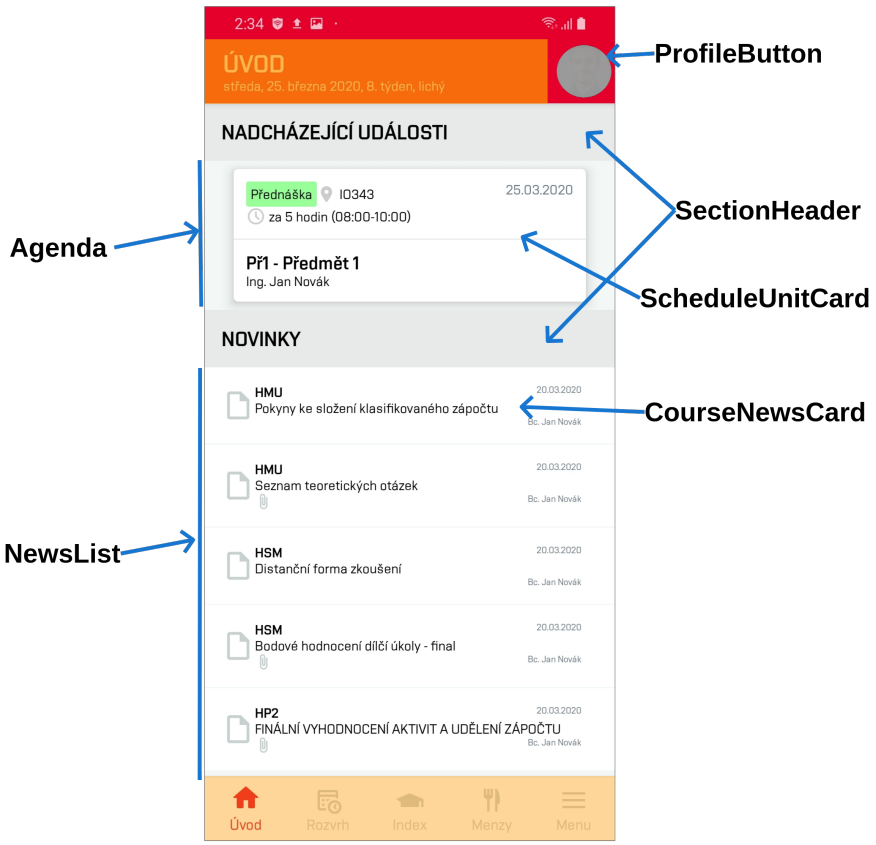
4.4 Užívateľské rozhranie

Užívateľské rozhranie je implementované pomocou knižnice React. React komponenty v aplikácii využívajúcej knižnicu MobX spĺňajú úlohu pozorovateľov, ktorí reagujú na zmeny pozorovaných dát prekreslením užívateľského rozhrania. Aby mohol byť komponent pozorovateľom, musí byť zabalený do funkcie *observer*, ktorú poskytuje knižnica MobX. Táto kapitola popisuje implementáciu troch obrazoviek: úvod, index a rozvrh. Popis budú dopĺňať obrázky z vyznačenými komponentmi, z ktorých sa daná obrazovka skladá. Časti obrazovky zvýraznené oranžovou farbou predstavujú navigačné prvky vykresľované knižnicou RNN. Tieto navigačné prvky možu byť natívne (spodný tabbar alebo nadpis a podnadpis v úvodnej obrazovke) alebo React komponenty s vlastnou logikou (nadpis a podnadpis predstavujúci rozbaľovacie menu v obrazovke indexu alebo rozvrhu).

4.4.1 Úvodná obrazovka

Zloženie úvodnej obrazovky je možné vidieť na obrázku 4.1. Pozostáva z komponentov *SectionHeader*, *Agenda* a *CourseNews*.

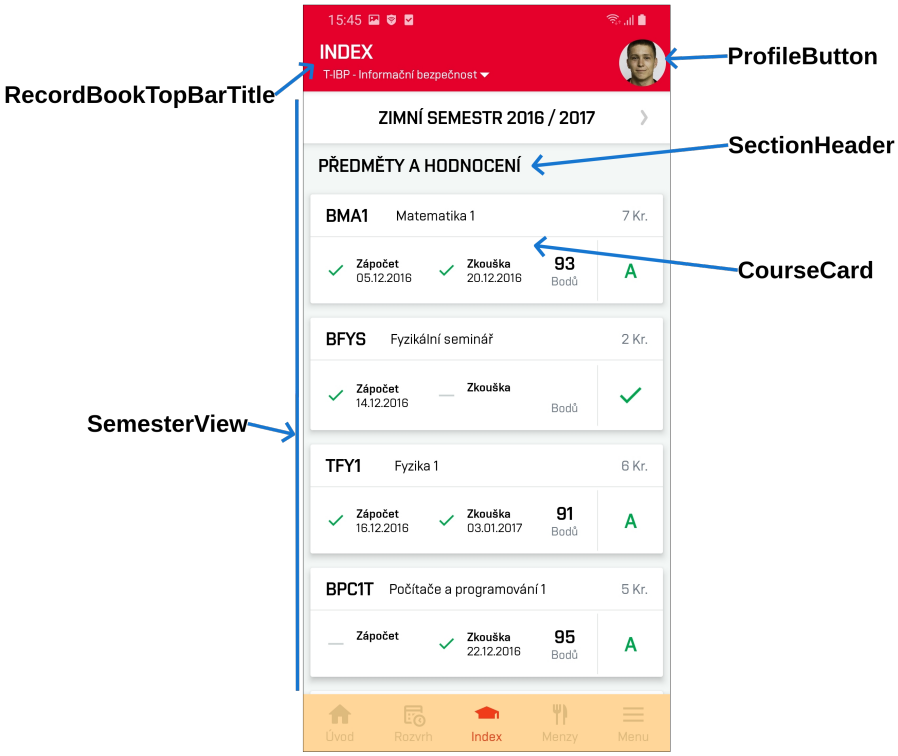
Komponent *Agenda* pozoruje z úložiska *StudyStore* zoznam najbližších vyučovacích blokov. Pri zmene tohto zoznamu (vyučovací blok sa skončí a odobere sa zo zoznamu) dochádza k prekresleniu len tohto komponentu, nie celej obrazovky, čím sa docieli optimalizácia vykresľovania UI. Podobne sa pri zmene zoznamu najnovších aktualít prekreslí len komponent *CourseNews*.



Obr. 4.1: Zloženie domovskej obrazovky.

4.4.2 Index

Obrazovka indexu pozostáva z horizontálneho zoznamu komponentov *SemesterView*, vďaka čomu je možné prechádzať jednotlivými semestrami posúvaním sa do strán. V komponente *SemesterView* sa nachádza vertikálny zoznam komponentov *CourseCard* predstavujúci predmety v danom semestri. Táto obrazovka reaguje na zmenu premennej *selectedStudy* uloženej v úložisku *StudyStore*, ktorej hodnotu modifikuje komponent *ScheduleTopBarTitle* obsahujúci rozbaľovacie menu so zoznamom štúdií. Zvolením štúdia bude hodnota premennej *selectedStudy* modifikovaná a index vykreslí zoznam semestrov vybraného štúdia.

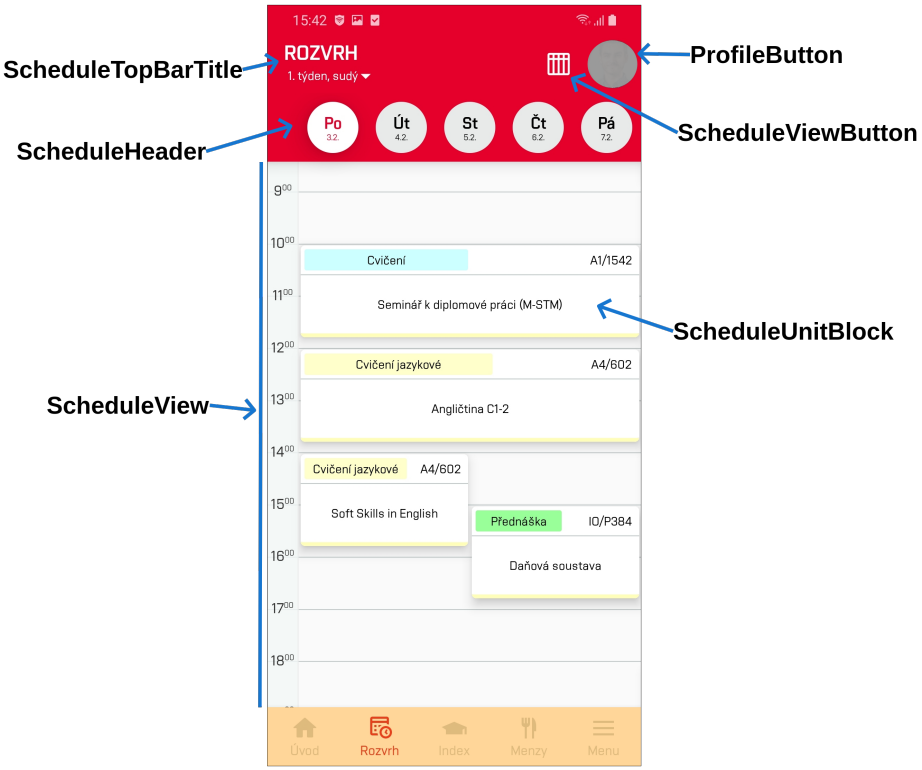


Obr. 4.2: Zloženie obrazovky elektronického indexu.

4.4.3 Rozvrh

Obrazovka rozvrh predstavuje najkomplexnejšiu obrazovku v aplikácii. Užívateľovi umožňuje meniť zobrazený týždeň, režim zobrazenia a presúvať sa medzi jednotlivými dňami pri dennom zobrazení. Rozvrh je tvorený komponentom *ScheduleView*, ktorý pomocou *ScheduleStore* úložiska komunikuje s navigačnými komponentami *ScheduleViewButton*, *ScheduleTopBarTitle* a *ScheduleHeader*. Na obrázku 4.3 je obrazovka v dennom režime. Po kliknutí na *ScheduleViewButton* je premennej *isWeeklyView* (boolean) priradená jej opačná hodnota. Na túto zmenu zareaguje komponent *ScheduleView* vykreslením týždenného zobrazenia a *ScheduleHeader* zrušením vyznačenia vybraného dňa. Reaguje aj samotný *ScheduleViewButton*, na základe ktorej vykreľuje ikonku predstavujúcu denné alebo týždenné zobrazenie. *ScheduleTopBarTitle* modifikuje premennú *selectedWeek* rovnakým spôsobom ako *RecordBookTopBarTitle* modifikuje *selectedStudy*. Pri posúvaní sa medzi dňami je volaná funkcia *onViewableItemsChanged*, ktorá indikuje zmenu viditelnej položky zo-

znamu a vďaka nej je možné zistiť index zobrazeného dňa. Ten sa ukladá do premennej *selectedScheduleDayIndex*, na ktorú v dennom režime reaguje komponent *ScheduleHeader* vyznačením zobrazeného dňa.



Obr. 4.3: Zloženie obrazovky rozvrhu.

Kapitola 5

Užívateľské testovanie

Neoddeliteľnou súčasťou vývoja mobilnej aplikácie je aj testovanie na skutočných užívateľoch. Jeho cieľom je odhlaliť chyby v návrhu užívateľského rozhrania a jeho implementácie. Tieto chyby predstavujú neintuitívnosť alebo neefektívnosť užívateľského rozhrania, ktoré môžu byť vývojárovi skryté pre jeho dobré znalosti daného systému. Práve neskúsený a neznalý užívateľ by mal tieto chyby odhaliť. Táto kapitola je venovaná návrhu užívateľského testovania, na základe ktorého výsledkov bude navrhnutý ďalší vývoj aplikácie.

5.1 Úlohy

Princípom užívateľského testovania je definovanie úloh, ktoré majú užívatelia s pomocou aplikácie splniť. Tieto úlohy predstavujú základnú prácu s aplikáciou a sú navrhnuté tak, aby užívateľ navštívil každú obrazovku aplikácie.

1. Otvorte aplikáciu a prihláste sa.
2. Zistite, koľkí je aktuálny týždeň semestra. Je párny alebo nepárny?
3. Aký máte dnes rozvrh? O koľkej a na akom mieste máte najbližšiu výuku?
4. Z akého predmetu máte poslednú aktualitu? Zistite email osoby, ktorá ju poslala.
5. Zobrazte si v rozvrhu aktuálny týždeň a potom si zobrazte aktuálny deň.
6. Pozrite sa na rozvrh v poslednom týždni aktuálneho semestra.
7. Koľko máte zapísaných kreditov v akuálnom semestri?
8. Aká bola vaša najlepšia známka v prvom semestri na VUT?
9. Zistite svoj pin. Dokedy je platný?
10. Aký vážený priemer vášho aktuálneho štúdia?
11. Aký je váš zostatok na konte KaM?
12. Pozrite sa na ponuku jedál menzy Pizzerie Mozzarella. Ktoré jedlo je najlacnejšie?
13. Zistite, v ktorej miestnosti má Bc. Jaroslav List kanceláriu.

14. Zobrazte si rozvrh prvého predmetu zobrazeného v indexe v aktuálnom semestri.
15. Zobrazte si rozvrh garanta tohto predmetu.
16. Zobrazte si aktuality len tohto predmetu.

Užívateľ je pri plnení týchto úloh sledovaný a na základe jeho správania je hodnotená funcionalita a intuitívnosť danej časti aplikácie. Čím menej krokov užívateľ vykoná a čím kratšiu dobu mu zabere splnenie úlohy, tým je aplikácia intuitívnejšia. Napríklad, ak užívateľ pre vyhľadanie VUT pinu prejde celú aplikáciu a nepodarí sa mu ho nájsť, tak umiestnenie pinu v aplikácii nie je intuitívne a je potrebné ho zmeniť.

5.2 Vyhodnotenie výsledkov testovanie

Aplikácia bola otestovaná na siedmych študentoch rôzných fakúlt VUT s Android zariadeniami. Fotky verzie aplikácie, ktorá bola naištalovaná na tieto zariadenia, je možné vidieť v prílohe B. S výnimkou prihlasovania im žiadna úloha nerobila výrazné problémy. Všetkých užívateľov zmiatlo chýbajúce prihlasovacie tlačítko a niektorým trvalo až pol minúty, než si všimli potvrdzujúce tlačítko na klávesnici. Bude preto treba zvážiť, či ponechať prihlasovaciu obrazovku minimalistickú, alebo toto tlačítko pridať a vyhnúť sa zmäteniu užívateľov hneď v prvej obrazovke, ktorú im aplikácie ponúkne. Riešením by tiež mohlo byť zobrazenie tohto tlačítka až po tom, ako užívateľ zadá heslo a zasunie klávesnicu, kedy je jasné, že potrvdzujúce tlačítko si nevšimol.

V úlohách č. 2 a 3, kde mali užívatelia zistiť aktuálny týždeň a najbližší rozvrh, a tým otestovať použíteľnosť a funkcionalitu úvodnej obrazovky, obsah tejto obrazovky preskočili a hneď si zobrazili rozvrh pre zistenie potrebných informácií. Dá sa očakávať, že úvodná obrazovka bude viac využívaná skúsenejším užívateľom, ktorý bude vedieť, kde sa čo v aplikácii nachádza.

Prácu s rozvrhom si užívatelia pochvaľovali a nerobilo im problém prepínať sa medzi denným a týždenným režimom a ani zobraziť si iné týždne semestra. Niektorých zmiatol týždenný režim, kde očakávali zobrazenie osí dní a hodín naopak, ako tomu je vo webovej aplikácii informačného systému. Čas ukáže, ako si na toto zobrazenie rozvrhu užívatelia zvyknú.

Následovali úlohy pre prácu s elektronickým indexom, kde užívatelia vytkli dva nedostatky. Pri úlohe č. 8 mnohí očakávali po kliknutí na nadpis semestru rozbaľovacie menu, kde by si vybrali semester rovnakým spôsobom, ako tomu je pri výbere štúdia alebo týždňa v rozvrhu. Rozbaľovacie menu by bolo užívateľsky prívetivejšie najmä v prípadoch, kedy si chce študent tretieho alebo štvrtého ročníka pozrieť svoj prvý semester. Namiesto prechádzania cez všetkých šesť alebo až osem semestrov by mu stačili dva kliky. Druhým nedostatkom, ktorého si niekorí všimli, bolo nezobrazovanie aktuálneho semestru po návrate do indexu, kde sa predtým prezerali predchádzajúce semestre.

Užívatelia strávili pomerne dosť času plnením úloh č. 9 a 10. Pri hľadaní VUT pinu im väčšinou ako prvé napadlo hľadať v nastaveniach aplikácie. Tým, že sa odkaz na profil nenachádza v menu, ale v hornej navigačnej lište v podobe profilovej fotky sa stáva menej objaviteľným a bude treba zvážiť jeho pridanie do menu aplikácie. Podobný problém bol s hľadaním váženého priemeru štúdia, ktorý užívatelia očakávali v indexe, a nie v profile. Užívatelia by očakávali odkaz na detail štúdia v indexe alebo zoznam štúdií v menu.

Úlohy týkajúce sa menz nerobili užívateľom žiadne problémy a vedeli hneď nájsť svoj zostatok na účte ako aj pozrieť si ponuku jedál konkrétnej menzy. Niektorí navrhli zobraziť

detail o jedle v samostatnom modálnom okne, kde by mohol byť text zloženia a alergénov jedla zobrazený väčším písmom.

Výhľadávanie rozvrhu vyučujúcich a predmetov prebehlo bez problémov. Užívatelia by však uvítali zvýraznenie ich vyučovacích blokov v týchto rozvrhoch pre lepšiu prehľadnosť.

Testovanie na študentoch z rôznych fakúlt odhalilo problém s aktualitami z predmetov. Kým študenti z Fakulty podnikateľskej alebo z Fakulty elektrotechnickej a komunikačných technológií mali desiatky položiek v zozname aktualít, u študentov z Fakulty strojného inžinierstva bol tento zoznam prázdny, kvôli čomu pôsobila úvodná obrazovka neúplne. Bolo by preto vhodné nájsť alternatívu, ktorá bude zobrazená v úvodnej obrazovke študentom týchto fakúlt.

Okrem odhalenia nedostatkov užívateľského rozhrania, užívatelia odhalili aj niekoľko logických chýb v aplikácii. Jednému užívateľovi sa zobrazovali nesprávne filtry aktualít predmetov v obrazovke aktualít. Ďalší užívateľ odhalil chýbajúci údaj o type vyučovania v detaile vyučovania, kedy sa mu pre príliš malý blok v rozvrhu (krátke trvanie - menej ako hodina) tento údaj nezobrazoval v tomto bloku a nemohol ho násť ani v detaile. Ďalším problémom bolo chýbajúci údaj pinu v profile, kde nebolo jasné, či je chyba na strane klienta alebo serveru.

Kapitola 6

Záver

Cieľom tejto práce bolo vytvoriť aplikáciu pre študentov VUT, ktorým má ponúkať aktuálny prehľad ich štúdia. Aplikácia bola vyvíjaná s použitím multiplatformového frameworku React Native pre platformy Android a iOS

Boli predstavené a porovnané moderné technológie používané pre vývoj mobilných aplikácií. Každý prístup ponúkal svoje výhody i nevýhody. Pri výbere bolo potreba zohľadniť nielen technické parametre výslednej aplikácie ako výkon alebo optimalizáciu, ale aj náklady pre vývoj, údržbu a rozširovanie aplikácie. Technológia React Native ponúka kompromis medzi odladenou aplikáciou pre cielenú platformu a zjednodušeným jednotným vývojom.

V práci bol vytvorený základný náčrt užívateľského rozhrania, kde bola definovaná hlavná navigácia a obsah jednotlivých obrazoviek. Na to naviazala práca grafika, ktorý navrhol grafický dizajn aplikácie. Výsledná mobilná aplikácia je schopná komunikovať zo školským serverom za účelom získania dát, ktoré dokáže užívateľovi poskytnúť i v offline režime. Aplikácia okrem hlavných modulov osobného rozvrhu, elektronického indexu a menz poskytuje užívateľom aj aktuality z predmetov, vyhľadávanie osôb a zobrazovanie rozvrhu predmetov a vyučujúcich. React Native umožňuje implementovať aplikačnú logiku pre obe platformy za použitia jedného jazyka, čo sa ukázalo ako veľká výhoda pri ladení aplikácie a opravovaní chýb. Avšak treba si dávať pozor pri implementácií logiky užívateľského rozhrania, kde nie vždy sa Android a iOS aplikácie správajú jednotne, a zmena opravujúca chybu u Android aplikácie môže spôsobiť chybu u iOS.

Aplikácia bola otestovaná na malej skupine študentov VUT so zariadeniami Android. Toto testovanie neodhalilo žiadnu zásadnú chybu v návrhu a implementácii aplikácie. Ďalší vývoj sa bude zameriavať na implementáciu mapy VUT, notifikácií a verzie iOS, ktorý je potrebný doladiť z hľadiska UI.

Literatúra

[1] *Cordova Introduction* [online]. The Apache Software Foundation, 2019 [cit. 2020-05-10]. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[2] *Native vs Hybrid vs Web App? Which One to Choose?* [online]. Krify Innovations, 2019 [cit. 2020-05-05]. Dostupné z: https://krify.co/difference-between-native-vs-hybrid-vs-web-apps-which-one-to-pick/.

[3] Apple. *Model-View-Controller* [online]. Apple, 2018 [cit. 2020-05-06]. Dostupné z: https://developer.apple.com/library/archive/featuredarticles/ViewControllerPGforiPhoneOS/.

[4] Apple. *Model-View-Controller* [online]. Apple, 2018 [cit. 2020-05-06]. Dostupné z: https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

[5] Babich, N. *Basic Patterns for Mobile Navigation* [online]. Nick Babich, 2016 [cit. 2020-05-09]. Dostupné z: http://babich.biz/basic-patterns-for-mobile-navigation/.

[6] Bennet, J. *Xamarin in Action*. Manning Publications Co., 2018. ISBN 9781617294389.

[7] Budiu, R. *Basic Patterns for Mobile Navigation: A Primer* [online]. Nielsen Norman Group, 2015 [cit. 2020-05-09]. Dostupné z: https://www.nngroup.com/articles/mobile-navigation-patterns/.

[8] Contributors, M. *Promise* [online]. Mozzila, 2020 [cit. 2020-05-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise.

[9] Contributors, M. *Writing WebSocket client applications* [online]. Mozzila, 2020 [cit. 2020-05-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_client_applications.

[10] Facebook. *React* [online]. [cit. 2020-05-12]. Dostupné z: https://reactjs.org/docs/getting-started.html.

[11] Facebook. *Core Components and Native Components* [online]. Facebook, 2020 [cit. 2020-05-09]. Dostupné z: http://reactnative.dev/docs/intro-react-native-components.

[12] Google. *Platform Architecture* [online]. 2019 [cit. 2020-05-10]. Dostupné z: https://developer.android.com/guide/components/fundamentals.

[13] Google. *Configure your build* [online]. 2020 [cit. 2020-05-10]. Dostupné z: https://developer.android.com/studio/build?hl=ar.

[14] Google. *Platform Architecture* [online]. 2020 [cit. 2020-05-10]. Dostupné z: https://developer.android.com/guide/platform.

[15] Grummitt, C. *IOS Development with Swift*. Manning Publications Co., 2017. ISBN 9781617294075.

[16] Kara Pernice, R. B. *Hamburger Menus and Hidden Navigation Hurt UX Metrics* [online]. 2016 [cit. 2020-05-09]. Dostupné z: https://www.nngroup.com/articles/hamburger-menus/.

[17] Kumar, T. *IOS Technology Overview - Layers of iOS* [online]. Mindstick, 2015 [cit. 2020-05-09]. Dostupné z: https://www.mindstick.com/Articles/1803/ios-technology-overview-layers-of-ios.

[18] Microsoft. *TypeScript* [online]. [cit. 2020-05-03]. Dostupné z: https://www.typescriptlang.org/.

[19] Mobx. *Mobx* [online]. [cit. 2020-05-10]. Dostupné z: https://mobx.js.org/.

[20] Pronschinske, M. *The State of Native vs. Web vs. Hybrid* [online]. DZone, 2014 [cit. 2020-05-05]. Dostupné z: https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

[21] Rahul Gaba, A. R. *React Native Internals* [online]. GitBook, 2015 [cit. 2020-05-09]. Dostupné z: https://www.reactnative.guide/3-react-native-internals/3.1-react-native-internals.html.

[22] Ridha, R. R. *3 Good Reasons Why You Might Want to Remove that Hamburger Menu from Your Product* [online]. 2018 [cit. 2020-05-09]. Dostupné z: https://medium.muz.li/3-good-reason-why-you-might-want-to-remove-that-hamburger-menu-from-your-product-69b9499ba7e2.

[23] Vries, R. de. *View Controllers Explained: Ultimate Guide For iOS & Swift* [online]. LearnAppMaking, 2019 [cit. 2020-05-06]. Dostupné z: https://learnappmaking.com/view-controller-uiviewcontroller-ios-swift/.

[24] Wilken, J. *Ionic in Action*. Manning Publications Co., 2016. ISBN 9781633430082.

[25] Wix. *React Native Navigation* [online]. [cit. 2020-05-10]. Dostupné z: https://wix.github.io/react-native-navigation/docs/before-you-start/.

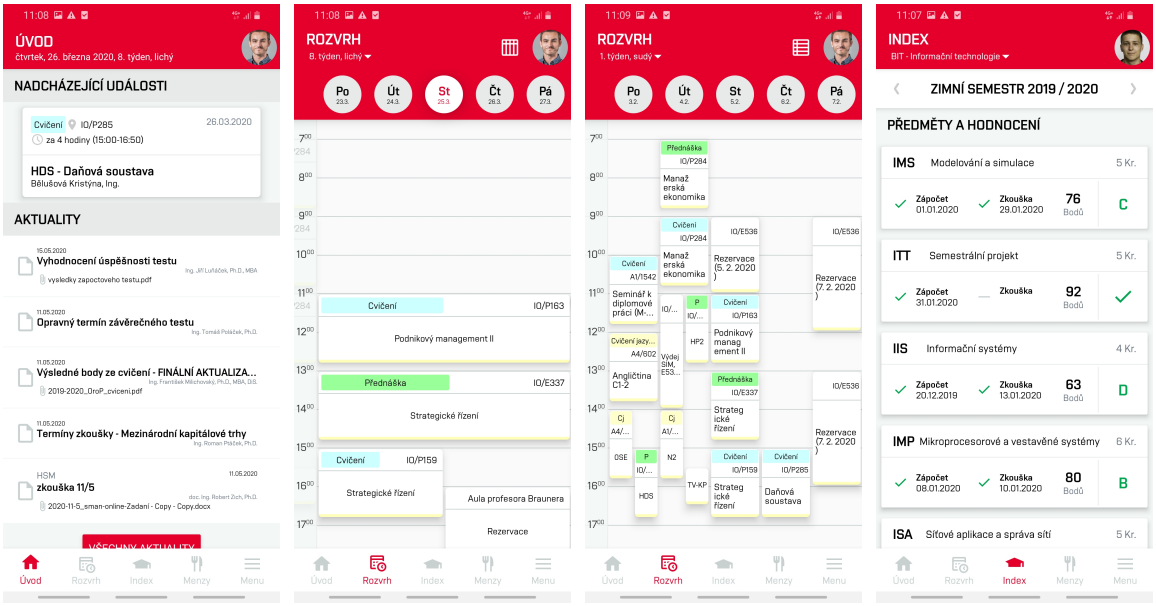
Príloha A

Obsah CD

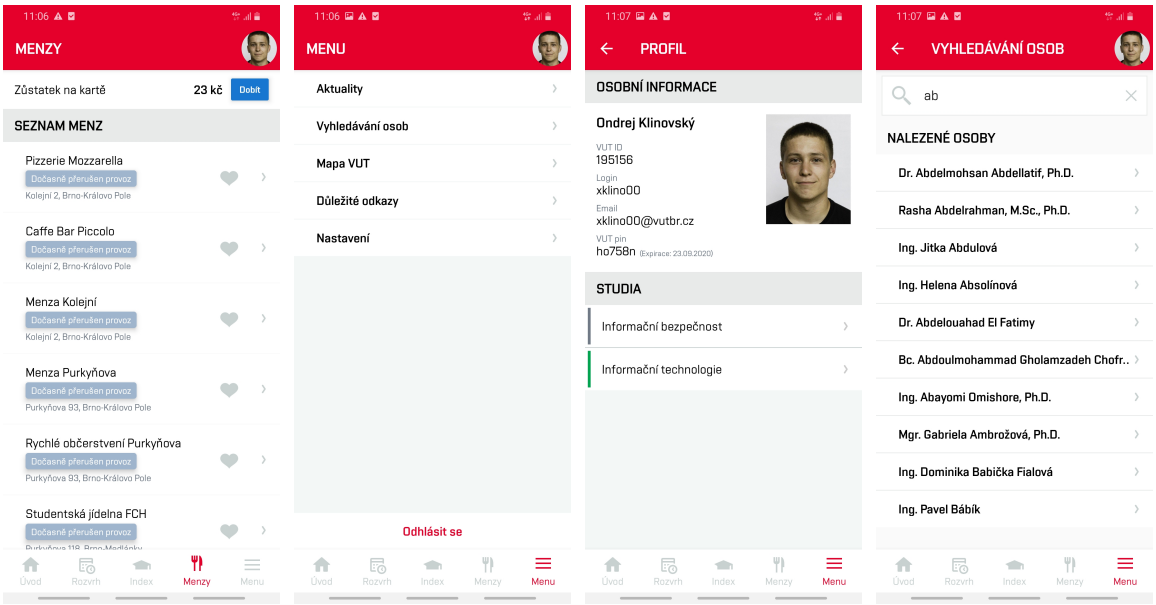
- **moje-vut-src** - zdrojový kód aplikácie
- **moje-vut.apk** - zostavená aplikácie pre zariadenia Android
- **poster.pdf** - plagát prezentujúci túto prácu vo formáte pdf
- **thesis.pdf** - táto práca vo formáte pdf
- **thesis-src** - zdrojový kód tejto práce

Príloha B

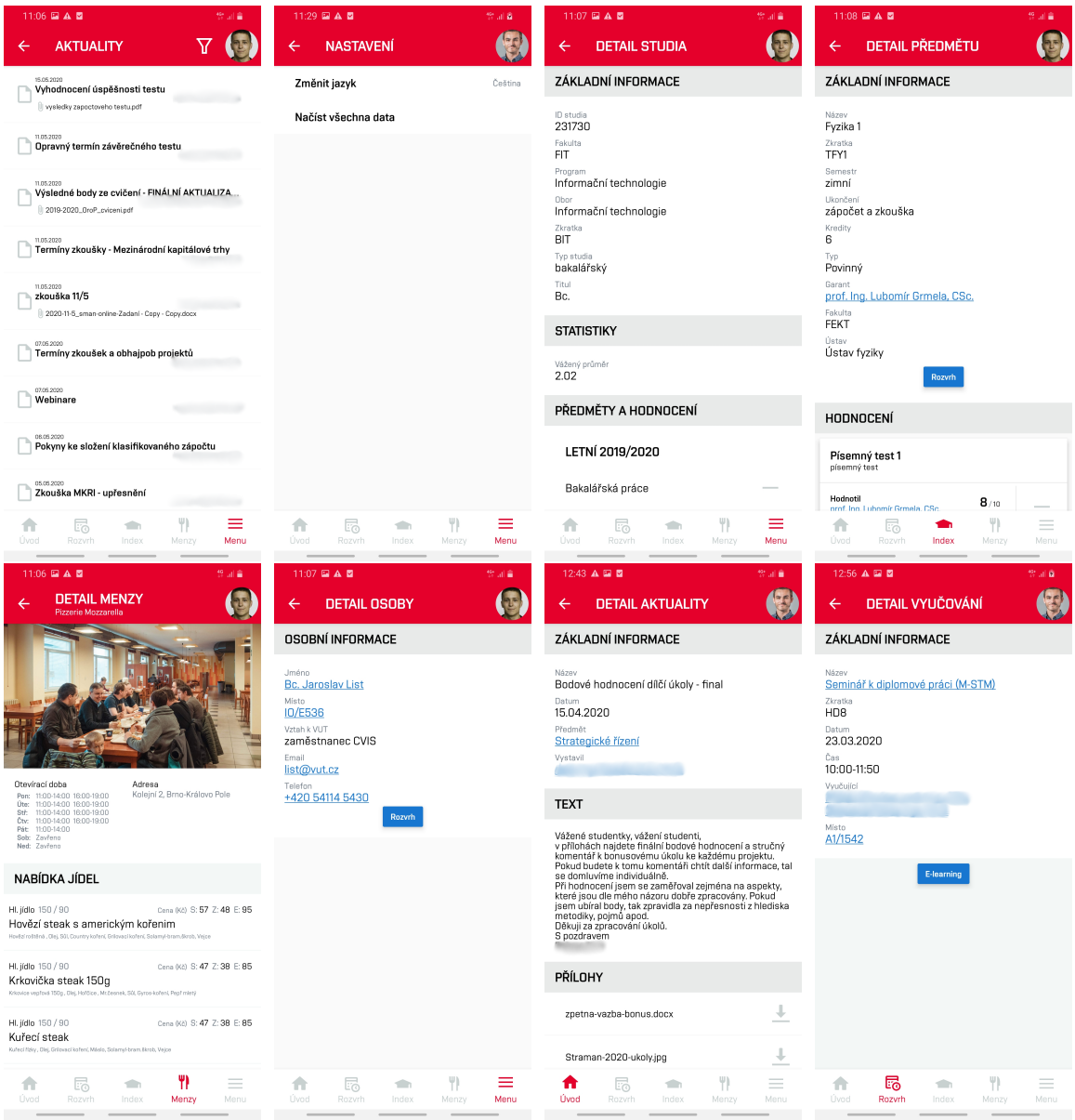
Fotky obrazoviek aplikácie



Obr. B.1: Fotky hlavných obrazoviek. Zľava: úvod, rozvrh - denné zobrazenie, rozvrh - týždenné zobrazenie, index a menzy.



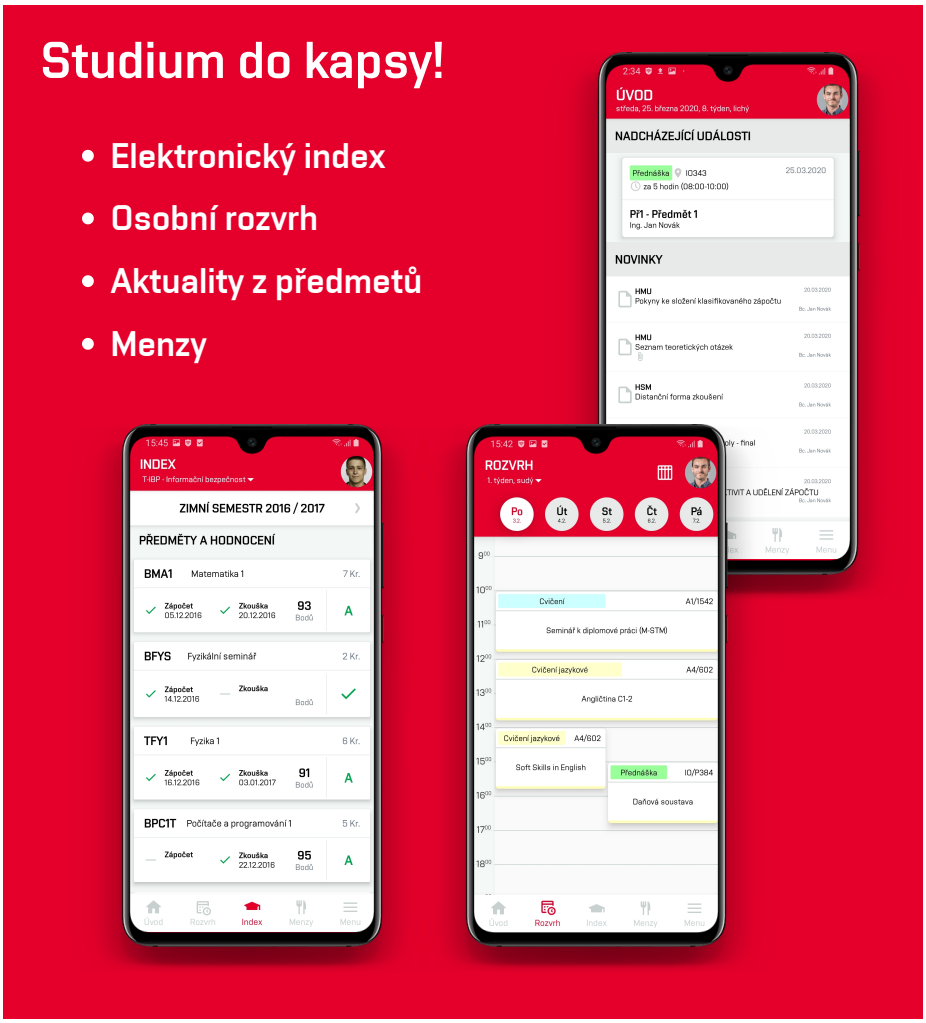
Obr. B.2: Fotky obrazoviek menz, menu, profilu a vyhľadávania osôb.



Obr. B.3: Fotky obrazoviek aktualít, nastavení aplikácie a detailov štúdia, predmetu, menzy, osoby, aktuality a vyučovania.

Príloha C

Plagát



Obr. C.1: Plagát Moje VUT.