Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Modul Uchazeč systému pro podporu výuky

Bakalářská práce

Autor: Monika Štorková
Studijní obor: Informační management

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Hradec Králové duben 2019

Prohlášení:

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

V Hradci Králové dne 8.4.2019 Monika Štorková

Poděkování:

Ráda bych poděkovala vedoucímu bakalářské práce doc. Mgr. Tomáši Kozlovi, Ph.D. za metodické vedení práce, cenné rady, připomínky a vstřícnost během zpracovávání práce. Dále děkuji své rodině za veškerou podporu.

Anotace

Cílem bakalářské práce bylo analyzovat, navrhnout a implementovat modul mobilní aplikace UHK Helper sloužící zájemcům o studium. Bakalářská práce se nejprve věnuje seznámení s obecnou problematikou mobilního vývoje, jako jsou mobilní platformy nebo typy mobilních aplikací. Dále jsou v textu popsány jednotlivé technologie, které byly při tvorbě modulu aplikace využity. Tato práce dále obsahuje popis návrhu modulu Uchazeč a podrobný popis implementace s ukázkami kódů. Modul Uchazeč uživateli nabídne možnost zjistit informace o studiu na vysoké škole, konkrétně pak na Univerzitě Hradec Králové. Dále má uživatel možnost shlédnout v aplikaci videa publikovaná na facebookových stránkách Univerzity Hradec Králové nebo jednotlivých fakult. Modul také uživateli umožňuje zaslat dotaz na konkrétní studijní oddělení příslušné fakulty.

Annotation

Title: Applicant Module for System of Education Support

The aim of the bachelor thesis was to analyze, design and implement the mobile application module UHK Helper serving students interested in studying. The bachelor thesis first deals with the general issues of mobile development, such as mobile platforms or types of mobile applications. Furthermore, the text describes the various technologies that were used for creating the application module. This bachelor thesis also contains a description of the Applicant module design and a detailed description of implementation with code samples. The Applicant module will offer to the user the opportunity to find out information about the study at a university, specifically at the University of Hradec Králové. Moreover, the user has the opportunity to watch the videos published on the Facebook pages of the University of Hradec Králové or individual faculties in the application. The module also allows the user to send questions to a particular study department of the relevant faculty.

Obsah

1 Úvod ... 1
2 Obecná problematika mobilního vývoje ... 2
2.1 Platformy ... 2
2.1.1 Android ... 3
2.1.1.1 Aktivity ... 4
2.1.1.2 Fragmenty ... 5
2.1.2 iOS ... 7
2.2 Typy aplikací ... 7
2.2.1 Nativní aplikace ... 7
2.2.2 Nativní multiplatformní aplikace ... 8
2.2.2.1 Xamarin ... 8
2.2.3 Hybridní aplikace ... 9
3 Výběr a popis zvolených technologií ... 11
3.1 Android Studio ... 11
3.1.1 Gradle ... 13
3.2 Java ... 13
3.3 Jsoup ... 14
3.4 JavaScript ... 14
4 Analýza a návrh modulu Uchazeč ... 16
4.1 Analýza ... 16
4.2 Návrh modulu Uchazeč ... 16
4.2.1 Informace ... 17
4.2.2 Videa ... 18
4.2.3 Kontakt ... 19
5 Popis vybraných aspektů implementace ... 21

5.1 Fragment Uchazeč 21
5.1.1 Drop down menu komponenta 24
5.1.2 Section button komponenta 28
5.2 Získávání informací 28
5.3 Zobrazování informací 32
5.4 Fragment Videa 33
5.5 Fragment Kontakt 39
6 Výsledky a závěr 41
6.1 Výsledky 41
6.2 Závěr 45
7 Seznam použité literatury 46
8 Přílohy 48

Seznam obrázků

Obrázek 1 Graf podílu mobilních operačních systémů na trhu v letech 2018–2019. Zdroj [1] 3
Obrázek 2 Vývojový diagram znázorňující životní cyklus aktivity. Zdroj [4] 5
Obrázek 3 Životní cyklus fragmentu (v případě běžící aktivity). Zdroj [5] 6
Obrázek 4 Sdílení kódu pomocí nástroje Xamarin. Zdroj [8] 9
Obrázek 5 Rozložení vývojového prostředí Android Studio. Zdroj [10] 12
Obrázek 6 Návrh fragmentu Uchazeč. Zdroj vlastní tvorba 17
Obrázek 7 Návrh fragmentu Uchazeč s rozbaleným menu. Zdroj vlastní tvorba 17
Obrázek 8 Návrh fragmentu Informace. Zdroj vlastní tvorba 18
Obrázek 9 Návrh fragmentu Videa. Zdroj vlastní tvorba 19
Obrázek 10 Návrh fragmentu Kontakt. Zdroj vlastní tvorba 20
Obrázek 11 Položka Uchazeč v hlavním menu. Zdroj vlastní tvorba 42
Obrázek 12 Fragment Uchazeč tvořící rozcestník modulu. Zdroj vlastní tvorba 42
Obrázek 13 Fragment Uchazeč s rozbaleným menu obsahujícím informace. Zdroj vlastní tvorba 42
Obrázek 14 Fragment Informace zobrazující informace o stipendiích. Zdroj vlastní tvorba 43
Obrázek 15 Fragment Informace zobrazující informace o Noci vědců. Zdroj vlastní tvorba 43
Obrázek 16 Fragment Videa zobrazující videa z Fakulty informatiky a managementu. Zdroj vlastní tvorba 44
Obrázek 17 Fragment Kontakt. Zdroj vlastní tvorba 44

Seznam tabulek

Tabulka 1 Celosvětový podíl operačních systémů pro mobilní telefony na trhu k lednu 2019 2
Tabulka 2 Přehled typů aplikací 10

1 Úvod

V dnešní době moderních technologií využívá velké množství lidí chytrý mobilní telefon, tzv. smartphone. Tyto smartphony slouží lidem už nejen k telefonování a zasílání SMS, ale je možné na nich pracovat podobně jako na počítačích. Chytré mobilní telefony pomáhají lidem v rozličných oblastech, ať už se jedná třeba o mapy, procházení webu, správu zdraví apod., jelikož existuje nepřeberné množství různých mobilních aplikací.

Bakalářská práce se bude zabývat vývojem určité části již existující mobilní aplikace UHK Helper určené pro platformu Android. Tato aplikace slouží studentům Univerzity Hradec Králové. Aplikace usnadňuje studentům orientaci jak přímo ve škole, tak také v časovém harmonogramu, umožňuje zapisování na zkoušky a mnoho dalšího. Cílem práce bude analyzovat, navrhnout a implementovat modul mobilní aplikace UHK Helper, který bude sloužit zájemcům o studium. Tento modul bude obsahovat informace o univerzitě, o různých akcích pořádaných jednotlivými fakultami a mnoho dalších informací, které by mohly pomoci studentům rozhodnout se při výběru vysoké školy.

2 Obecná problematika mobilního vývoje

Kapitola se zabývá základním rozdělením mobilních platforem, přičemž z důvodu vývoje mobilní aplikace pro Android je v textu detailněji představena právě platforma Android. Poté je zde stručně popsáno, jak aplikace na systému Android funguje a dále se práce o něco podrobněji věnuje důležitým komponentám Android aplikací, a to aktivitám. Je zde popsáno několik callback funkcí, pomocí kterých aktivity přecházejí do jednotlivých stavů. Jistá pozornost je věnována také fragmentům.

Dále jsou v kapitole popsány typy aplikací, které jsou rozděleny na aplikace nativní, nativní multiplatformní a v neposlední řadě aplikace hybridní. Vzhledem k možnosti multiplatformního vývoje se tato kapitola věnuje i nástroji Xamarin, který takovýto vývoj umožňuje.

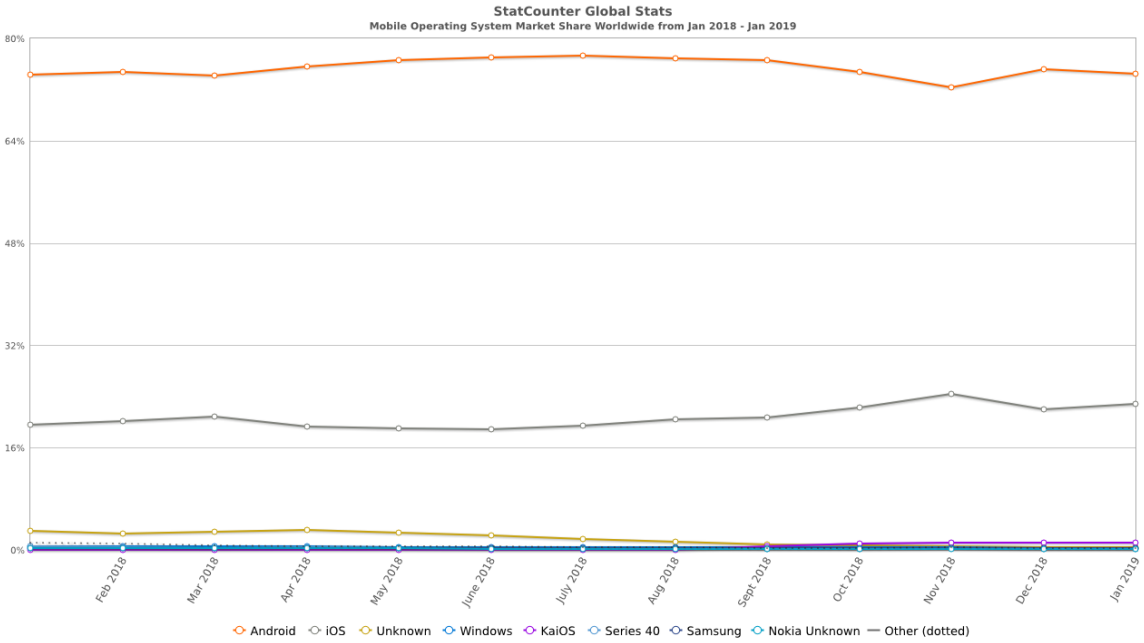
2.1 Platformy

V současné době jsou na trhu dvě nejrozšířenější platformy, a to Android od společnosti Google a iOS patřící společnosti Apple, přičemž jsou dále zastoupeny také například Windows Phone či Blackberry OS. Podíl mobilních operačních systému na trhu k lednu 2019 ukazuje tabulka č.1.

Tabulka 1 Celosvětový podíl operačních systémů pro mobilní telefony na trhu k lednu 2019

Operační systém	Podíl na trhu
Android	74,45 %
iOS	22,85 %
KaiOS	1,1 %
Jiný	0,41 %
Windows	0,3 %
Samsung	0,28 %

Zdroj [1]



Obrázek 1 Graf podílu mobilních operačních systémů na trhu v letech 2018–2019. Zdroj [1]

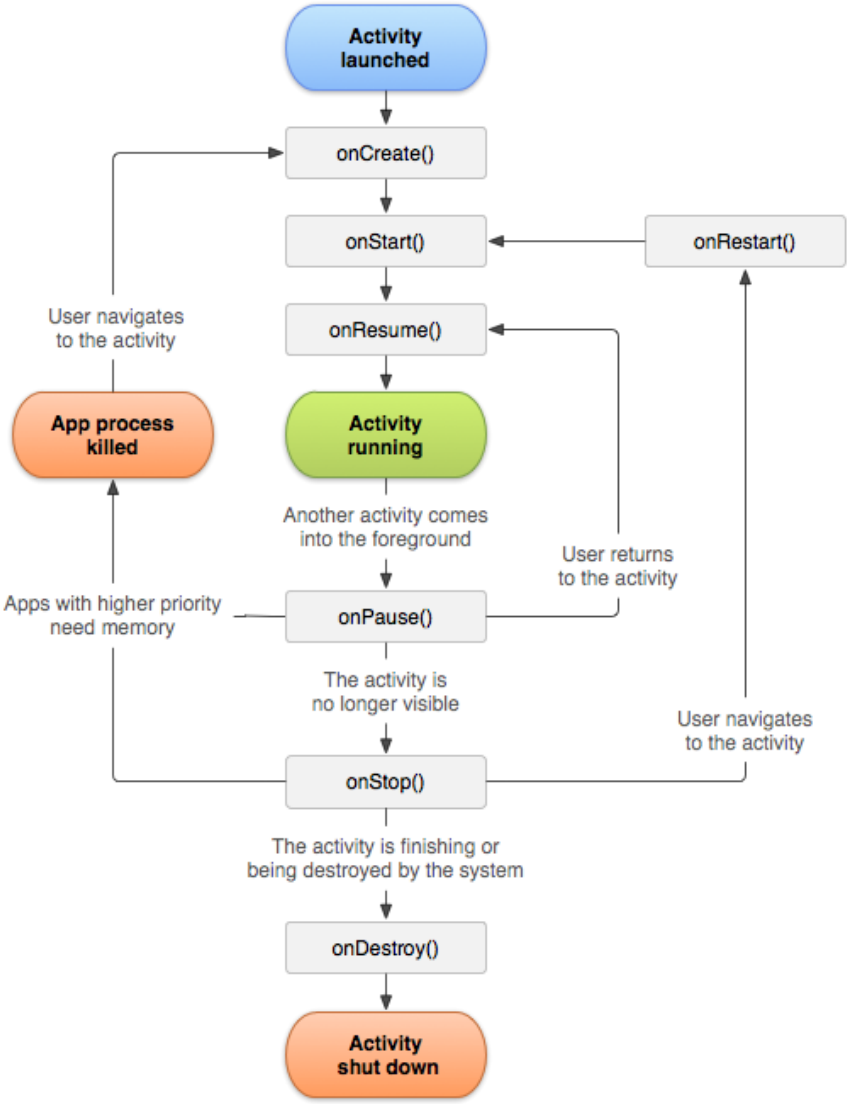
2.1.1 Android

Android je velmi rozšířený operační systém jak pro chytré mobilní telefony, tak také tablety, televize, chytré hodinky nebo automobily. Aplikace pro Android mohou být napsány v programovacích jazycích Kotlin, Java a C++. Pro vývoj aplikací je možné využít například IDE (Integrated Development Environment) Android Studio. Android SDK nástroje zkompilují napsaný zdrojový kód včetně všech dat a různých resource souborů do jednoho souboru s příponou .apk. Jedná se o archivační soubor Android package, který obsahuje veškerý obsah android aplikace. Tento soubor se využívá jako instalační soubor pro android zařízení. Každá aplikace běží ve virtuálním prostoru (sandboxu) izolovaně od ostatních, takže jedna aplikace nemůže zasahovat do adresního prostoru jiné aplikace. Operační systém Android je multi-user Linux systém, kde každá aplikace je zde uživatelem a má přidělené unikátní uživatelské ID. Na základě daných ID přiděluje systém určitá oprávnění všem souborům konkrétní aplikace, takže pouze aplikace s daným ID má k těmto souborům přístup. V určitém případě je možné, aby dvě aplikace sdílely stejné uživatelské ID a tím měly navzájem přístup ke svým souborům [2].

2.1.1.1 Aktivity

Jedná se o stěžejní komponentu android aplikací, která se zobrazí na displeji smartphonu a umožňuje přímou interakci s uživatelem. V této komponentě jsou vykreslovány veškeré UI elementy dané aktivity. Každá aplikace pro Android musí obsahovat vždy alespoň jednu aktivitu označovanou jako main activity. Tato aktivita se spustí v okamžiku, kdy uživatel klikne na ikonu konkrétní aplikace. Obvykle se však aplikace pro Android skládají z více aktivit. Například aplikace e-mailového klienta se bude skládat z minimálně třech aktivit. První aktivita je main aktivita, ve které jsou uživateli vykresleny jednotlivé e-maily. Další aktivitou může být aktivita, která bude sloužit k napsání nového e-mailu a třetí z nich může sloužit k zobrazování jednotlivých e-mailů. Tyto aktivity mezi sebou mohou komunikovat a předávat si data. Jedna aktivita také může spouštět jinou aktivitu. Při porovnání programování v programovacím jazyce Java pro desktop a pro Android je určitý rozdíl. Pro desktop je samotný kód spuštěn v metodě *main()*, zatímco na Androidu je kód spuštěn v main aktivitě v příslušné callback funkci, dále jen callback. Veškeré aktivity vycházejí ze třídy *Activity* [3].

Aktivita se vždy nachází v jednom z mnoha možných stavů. Aktivita umožňuje reagovat na jednotlivé přechody mezi stavy pomocí callback funkcí. Stavy se mohou měnit v závislosti na uživatelových akcích, případně akcích systémových. Při vytvoření aktivity je zavolán callback *onCreate()*. Tento callback musí implementovat každá aktivita. Když tato callback funkce skončí, následuje vždy callback *onStart()*. V tomto okamžiku se aktivita pro uživatele stává viditelnou. Dále následuje *onResume()* callback. Tento callback je zavolán těsně předtím, než může uživatel s aktivitou interagovat. Po této callback funkci vždy následuje callback *onPause()*. Daný callback je systémem zavolán ve chvíli, kdy se do popředí dostane jiná aktivita, například přijde SMS zpráva nebo hovor. Jakmile se uživatel rozhodne novou SMS zprávu otevřít, zavolá se callback *onStop()*. V tuto chvíli již uživatel danou aktivitu nevidí. Pokud se uživatel k aktivitě vrátí, zavolá se callback *onRestart()*, který zavolá callback *onStart()* a aktivita se dostává na začátek životního cyklu. Pokud je aktivita zcela ukončena, zavolá se callback funkce *onDestroy()*. Jedná se o poslední callback, který aktivita obdrží. Celý životní cyklus aktivity je znázorněn ve vývojovém diagramu na obrázku 2 [3].

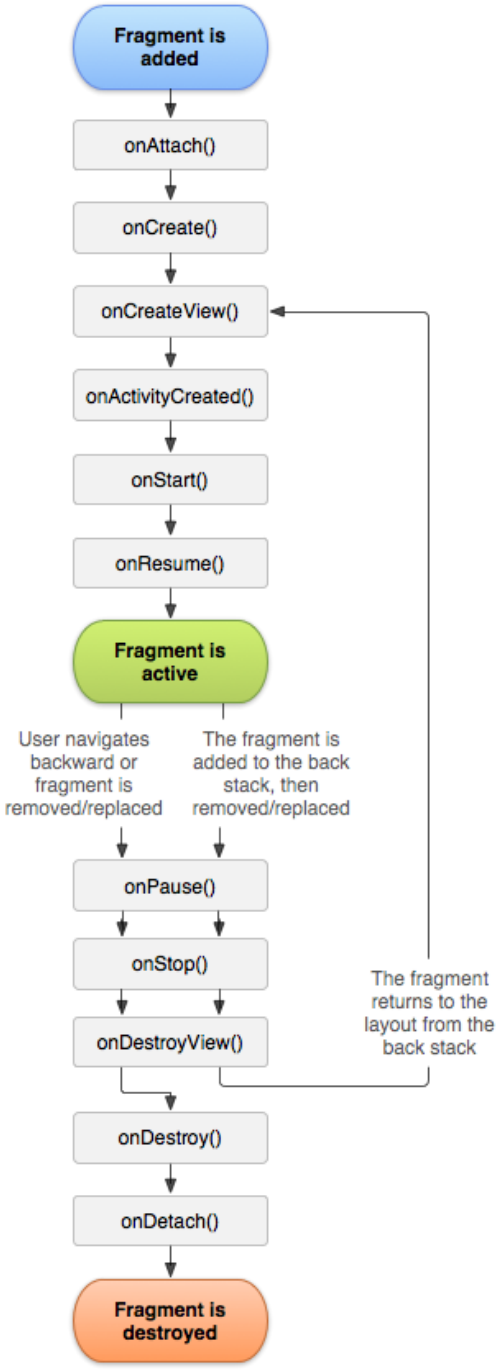


Obrázek 2 Vývojový diagram znázorňující životní cyklus aktivity. Zdroj [4]

2.1.1.2 Fragmenty

Jedná se o další významnou komponentu android aplikací. Fragment je možné si představit jako část aktivity, přičemž fragment má vlastní životní cyklus a přijímá vlastní vstupní události. Zatímco aktivita je v běžícím stavu, fragment je možné přidat nebo odebrat. Jeden fragment je možné využít ve více různých aktivitách. K vytvoření fragmentu je třeba vytvořit vlastní podtřídu třídy *Fragment*. Tato třída je z hlediska kódu velmi podobná třídě *Activity*. Třída *Fragment* totiž obsahuje callback metody podobné aktivitě, jako například *onCreate()*, *onStart()*, *onPause()* nebo *onStop()*. Na základě těchto podobností je v případě již existující

android aplikace jednoduše možné kód z callback metod v aktivitě přesunout do odpovídajících callback metod fragmentu. Každý fragment musí implementovat minimálně callback metody *onCreate()*, *onCreateView()* a *onPause()*, přičemž existuje několik dalších callback metod, které by měly být také využity. Životní cyklus fragmentu je znázorněn na obrázku 3.



Obrázek 3 Životní cyklus fragmentu (v případě běžící aktivity). Zdroj [5]

Důležitým faktem je, že fragment nemůže být umístěn samostatně, ale musí být vždy hostován v konkrétní aktivitě. Životní cyklus fragmentu je pak přímo ovlivněn životním cyklem hostující aktivity. Například pokud je určitá aktivita pozastavena, stejně tak jsou pozastaveny i veškeré fragmenty v ní. V případě, že má být uživateli umožněno vracet se zpět mezi jednotlivými fragmenty po stisknutí tlačítka zpět, je vhodné využít tzv. back stack. Back stack je spravován aktivitou a pokud byl určitý fragment přidán do back stacku, nedojde k jeho úplnému zničení, aby mohl být znovu využit. Jeho ukončování se zastaví v callback metodě *onDestroyView()*. Pokud dojde k navrácení se k fragmentu z back stacku, je zavolána metoda *onCreateView()* a fragment je oživen. [5] Pokud uživatel přechází z fragmentu A do fragmentu B, dále z B do C, přičemž fragmenty B i C byly přidány do back stacku, tak v případě, že uživatel dojde až do fragmentu C a stiskne tlačítko zpět, ocitne se ve fragmentu B. Analogicky při stisknutí tlačítka zpět ve fragmentu B doje k navrácení do fragmentu A.

2.1.2 iOS

iOS je operační systém UNIXového typu pro mobilní telefony či tablety vytvořený společností Apple Inc. Systém je rozdělen na základní čtyři vrstvy poskytující API a frameworky potřebné pro vývoj mobilních aplikací. Aplikace pro tento operační systém je možné psát například v programovacích jazycích Objective-C nebo také Swift. Oblíbeným a základním podporovaným vývojovým prostředím je XCode, který je však určený pouze pro macOS [6].

2.2 Typy aplikací

2.2.1 Nativní aplikace

Jedná se o aplikaci, která je napsána ve specifickém programovacím jazyce určeném pro konkrétní platformu. Například pro platformu Android se jedná o aplikace napsané v programovacích jazycích Java a Kotlin. Nativní aplikace jsou výhodné zejména pro aplikace vyžadující vysoký výkon a aplikace využívající specifické aspekty dané platformy (přístup ke specifickému hardwaru, senzorům apod.). Mezi další výhody nativních aplikací lze jednoznačně zařadit snadnou

dostupnost obrovského množství knihoven pro konkrétní platformy. Jednou z největších nevýhod nativních aplikací je nutnost programovat aplikaci pro každou platformu zvlášť, žádné sdílení vytvořeného kódu není možné. Z toho plyne nutnost vše udělat dvakrát i když se jedná o totožnou aplikaci. Je tedy nezbytně nutné mít samostatné týmy vývojářů se specifickými znalostmi a zkušenostmi pro danou platformu [7].

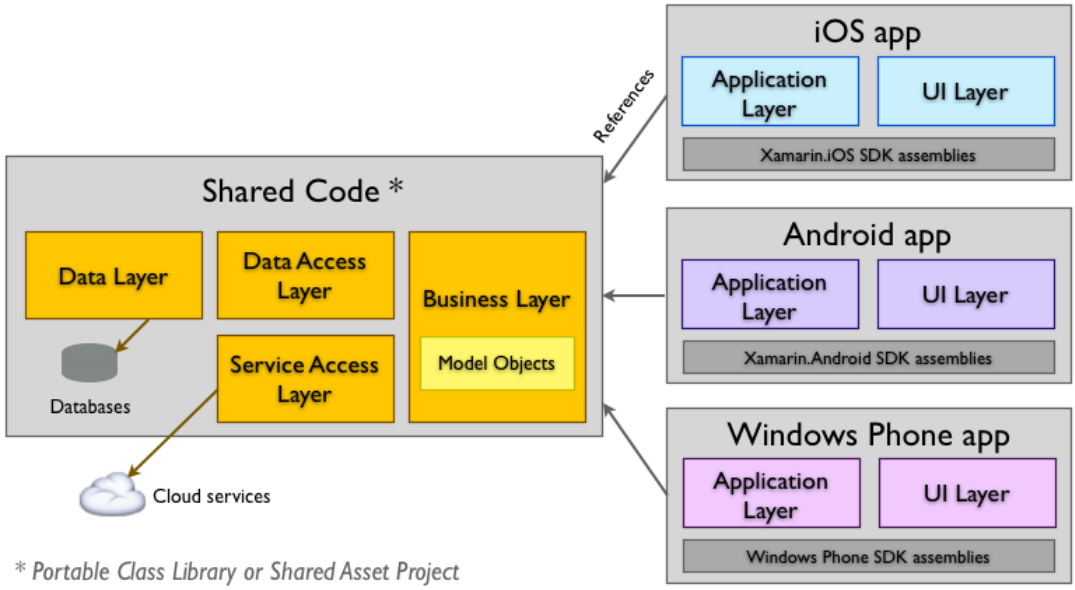
2.2.2 Nativní multiplatformní aplikace

Řešením zmíněných nedostatků u nativních aplikací je využití tzv. nativních multiplatformních aplikací. Princip těchto aplikací spočívá v použití jednoho programovacího jazyka a překladače. Výsledná aplikace je tedy napsána v jednom programovacím jazyce a poté je pomocí překladače přeložena pro konkrétní platformy. Takováto aplikace využívá prostředků dané platformy a její chování je zcela totožné s nativní aplikací. Nejpoužívanější nástroj je Xamarin, který umožňuje vyvíjet v jazyce C#. Nejpoužívanějším programovacím jazykem jde zde tedy C#, který tento kód umožňuje sdílet také mezi Windows či macOS na počítačích. Nespornou výhodou u tohoto principu je, že kód aplikace je napsán pouze jednou. Tento fakt velmi usnadňuje vývoj multiplatformních aplikací, přičemž zároveň šetří i finance. Výkon takovýchto aplikací je velmi blízký aplikacím nativním. I u tohoto typu aplikací je k dispozici dobrá dokumentace a podpora. Mezi další výhody tohoto přístupu patří možnost využití celých SDK prostředků jednotlivých platforem. Koncoví uživatelé nerozeznají výslednou aplikaci od aplikace nativní, jelikož UI (User Interface) využívá nativních ovládacích prvků. Drobnou nevýhodou je, že pro každou platformu se musí UI psát samostatně. Další nevýhodou těchto aplikací je, že oproti nativním aplikacím je k dispozici pouze omezené množství knihoven a dalších komponent [7].

2.2.2.1 Xamarin

Xamarin je volně dostupný nástroj od Microsoftu sloužící pro tvorbu mobilních aplikací. Největší výhodou daného nástroje je možnost současně vyvíjet aplikace pro hlavní platformy jako je Android, iOS či Windows Mobile. K vývoji je

možné využívat přímo Visual Studio a programovací jazyk C#. Značná část kódu je sdílena, kdy v projektu Shared Code je definována komunikace s webovými službami, práce s databází, business vrstva a datové modely. Dále jsou z této knihovny vystaveny API využívající jednotlivé projekty, které definují frontend aplikace na jednotlivých platformách. Sdílení kódu pomocí Xamarinu je znázorněno na obrázku 4 [8].



Obrázek 4 Sdílení kódu pomocí nástroje Xamarin. Zdroj [8]

2.2.3 Hybridní aplikace

Dalším typem aplikací jsou tzv. hybridní aplikace. Jedná se vlastně o webovou stránku zobrazující se uvnitř nativní aplikace, běžně pomocí komponenty WebView. Zde pak běží samotná webová aplikace napsaná v jazycích HTML, CSS (Cascading Style Sheets) a JavaScript. Tento typ aplikací využívá pro ovládaní vlastní HTML prvky. Nástroj využívající se v této kategorii nejčastěji je Apache Cordova. Největší výhodou hybridních aplikací je možnost sdílení veškerého napsaného kódu. Dále je za výhodu možné považovat i jednotný vzhled na různých platformách, což může být však v určitých ohledech i značnou nevýhodou. Velmi výraznou nevýhodou je malý výkon těchto aplikací, jelikož jsou vždy interpretovány. Další nevýhodou je poněkud horší přístup k prostředkům konkrétních platforem a nemožnost sdílet

kód na platformy jako je Windows nebo macOS z důvodu, že je daná webová stránka zobrazována uvnitř nativní aplikace [7][9].

Tabulka 2 Přehled typů aplikací

	Nativní aplikace	**Nativní multiplatformní aplikace**	**Hybridní aplikace**
Tech Stack	Závislý na platformě	C# .NET	HTML5, CSS, JS
Sdílení kódu	Žádné	Téměř 100 %	Úplné
UI/UX	Kompletně přizpůsobený pro konkrétní platformu	Možnost přizpůsobení dle jednotlivých platforem s využitím nativních ovládacích prvků	Zcela sdílené, menší možnosti přizpůsobení jednotlivým platformám
Výkon	Vysoký	Velmi dobrý, téměř jako nativní	Slabší (vzhledem k interpretaci HTML a JS)
Přístup k hardwaru	Bez omezení	Bez omezení	Omezený (pomocí prostředků třetích stran)

Zdroj [7]

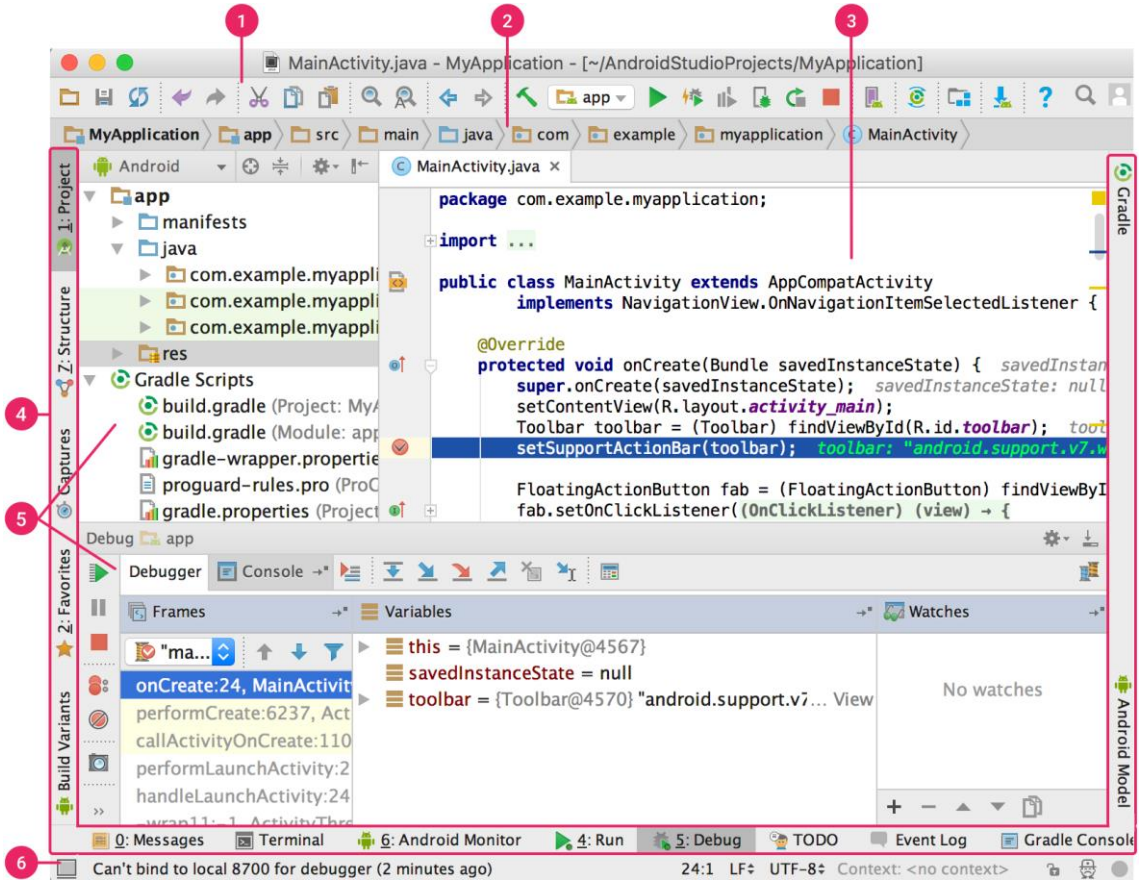
3 Výběr a popis zvolených technologií

Stávající mobilní aplikace UHK Helper je tvořena nativně v programovacím jazyce Java. Na základě toho byl zvolen programovací jazyk Java a bude využíváno Android Studio jako vývojové prostředí, které je v současné době velmi přehledné a umožňuje pohodlný vývoj aplikací. Jedná se o oficiální vývojové prostředí pro vývoj nativních aplikací pro Android. Android Studio nabízí nepřeberné množství funkcí, mezi které patří například emulátor mnoha androidových zařízení nebo pohodlný a přehledný drag and drop systém pro tvorbu GUI (Graphical User Interface).

3.1 Android Studio

Android Studio je oficiální vývojové prostředí určené pro vývoj mobilních aplikací pro systém Android. Toto IDE (Integrated Development Environment) je založeno na komerčním vývojovém prostředí IntelliJ IDEA od společnosti JetBrains. Android Studio nabízí velké množství funkcí, které přináší mnoho ulehčení při samotném vývoji mobilních aplikací. Mezi tyto funkce je možné zařadit například rychlý emulátor, který zároveň nabízí mnoho funkcí, integrovaný Git, velké množství nástrojů umožňujících testování nebo třeba flexibilní build projektu založený na Gradlu a mnoho dalšího [10].

Uživatelské rozhraní Android Studia je velmi přehledné. Hlavní okno je tvořeno hned z několika oblastí, jak je znázorněno na obrázku 5. Následuje popis grafického uživatelského rozhraní Android Studia, zpracováno podle [10]. Hlavní okno je také možné přizpůsobit si zcela dle vlastních preferencí skrytím nebo přesunutím jednotlivých lišt.



Obrázek 5 Rozložení vývojového prostředí Android Studio. Zdroj [10]

1) Panel nástrojů umožňuje spravovat velké množství akcí, jedná se například o spouštění vlastní aplikace a dalších nástrojů.
2) Navigační lišta usnadňuje navigaci v projektu a úpravu jednotlivých souborů. Dále poskytuje ucelený pohled na strukturu projektu viditelnou v okně Projekt.
3) Okno pro editaci slouží k samotné tvorbě a úpravě kódu. Zobrazení závisí na tom, jaký typ souboru je právě editován.
4) Lišta nástrojů se nachází podél okna vývojového prostředí. Tato lišta obsahuje jednotlivá tlačítka, která umožňují rozbalit nebo sbalit okna nástrojů.
5) Okno nástrojů umožňuje přistoupit ke konkrétním úlohám jako je třeba správa projektu, vyhledávání a další.
6) Stavový řádek zobrazuje aktuální stav projektu a vývojového prostředí samotného. Dále jsou zde zobrazeny veškeré zprávy a varování.

3.1.1 Gradle

Gradle je volně dostupný nástroj, umožňující automatizaci při sestavení (buildu) projektu. Jeho flexibilita umožňuje build téměř jakéhokoliv typu softwaru. Gradle běží na JVM (Java Virtual Machine), tím pádem vyžaduje nainstalované JDK (Java Development Kit). Díky tomu je také možné jednoduše využívat Gradle na různých platformách. Několik předních vývojových prostředí jako je Android Studio, IntelliJ IDEA, Eclipse nebo NetBeans umožňuje importovat Gradle buildy a interagovat s nimi [11].

Gradle je integrovanou součástí Android Studia. Android Studio využívá Gradle jako základ build systému více zaměřené na schopnosti androidu poskytnuté Android pluginem pro Gradle. Funkce Gradlu umožňují například vytvořit několik souborů APK (Android Package Kit). To je výhodné, pokud je například potřeba distribuovat jak volně dostupnou, tak placenou verzi aplikace. Gradle dále umožňuje automatizovaně spouštět testy a spravovat závislosti. Správa závislostí umožňuje jednoduchým konfiguračním souborem stanovit, která knihovna a v jaké verzi má být stažena ze vzdáleného repozitáře a nakonfigurována pro daný projekt [10].

3.2 Java

Java je objektově orientovaný dynamický programovací jazyk. Java byla poprvé vydána v roce 1995 společností Sun Microsystems, avšak první plná verze Javy se objevila 23. ledna 1996. Pro Javu jako takovou existuje široké využití [12][13].

Jedná se o oblíbený programovací jazyk pro smartphony se systémem Android, avšak Javu je možné využít rovněž na většině zařízení s operačním systémem Windows či Linux. Java byla navržena tak, aby se podobala jazyku C++, ale ve své podstatě je mnohem jednodušší a klade důraz na objektově orientované programování [13].

Zdrojový kód napsaný v jazyce Java, do textového souboru s příponou Java, je kompilovaný do bytekódu pomocí Java kompilátoru. JVM může interpretovat bytekód nebo využiít JIT (just-in-time) kompilátoru, který zkompiluje bytekód na spustitelný kód pro cílovou platformu. JVM je součástí JRE (Java Runtime Environment), který musí být na daném zařízení nainstalován. V případě využití JIT

kompilátoru je vykonávání kódu mnohem rychlejší než v případě interpretace JVM. Díky tomuto mechanismu je Java velmi multiplatformní a jednou napsaný kód je možné spustit na jakémkoliv zařízení [13].

3.3 Jsoup

Jsoup je volně dostupná knihovna z GitHubu[1] pro programovací jazyk Java zjednodušující parsování HTML dokumentů. Knihovna se snaží nabídnou programátorovi co největší pohodlí při manipulování s daty. Jsoup využívá nejlepší techniky z DOM (Document Object Model), CSS a jQuery. Knihovna Jsoup implementuje WHATWG (Web Hypertext Application Technology Working Group) HTML 5 specifikaci a po provedení parsování HTML vytvoří stejný DOM model, stejně tak jako moderní prohlížeče. Jsoup parsuje HTML buď z URL, přímo ze souboru, nebo z textového řetězce. Dále umí vyhledat a extrahovat data skrze DOM a CSS selektory. Jsoup také umožňuje manipulovat s HTML elementy, atributy a textem. Výhodné je i čištění uživatelem odeslaných dat pomocí tzv. whitelistu pro prevenci před XSS (Cross Site Scripting) útoky. Výsledkem parsování je čisté HTML. Příklad práce s knihovnou Jsoup je možné vidět v ukázce kódu 1. Nejprve dojde ke stažení stránky z wikipedie, vypíše se hlavní titulek a nadpisy obsahující nově přidaná témata s absolutní URL [14].

```
Document doc = Jsoup.connect("http://en.wikipedia.org/").get();
log(doc.title());
Elements newsHeadlines = doc.select("#mp-itn b a");
for (Element headline : newsHeadlines) {
    log("%s\n\t%s",
            headline.attr("title"), headline.absUrl("href"));
```

Ukázka kódu 1 Příklad práce s knihovnou Jsoup. Zdroj [14]

3.4 JavaScript

Určitá část práce vyžaduje načítání videí z facebookových stránek skrze Graph API. Jelikož Graph API již neumožňuje získávat surová binární data videa, bylo namísto toho využito získávání HTML elementu *iframe* obsahujícího video. Z tohoto

[1] https://github.com/jhy/jsoup/

důvodu je logika zobrazování videí naprogramována v programovacím jazyce JavaScript.

JavaScript je dobře známý skriptovací jazyk pro webové stránky, avšak je využíván také mnoha dalšími prostředími, jakými jsou například Node.js nebo Adobe Acrobat. JavaScript je multiplatformní dynamický skriptovací jazyk podporující objektově orientované programování. Umožňuje vytváření objektů skrze prototypy. Základní syntaxe jazyka je velmi podobná programovacímu jazyku Java a C++. JavaScript je standardizován standardem ECMAScript. Od roku 2015 pravidelně každý rok vychází ECMAScript standardy v nových verzích, přičemž současná verze je ECMAScript 2019.

V JavaScriptu je možné programovat jak procedurálně, tak objektově. Objekty jsou programově vytvářeny v JavaScriptu vkládáním metod a vlastností do jinak prázdných objektů za běhu kódu. To je rozdíl oproti kompilovaným jazykům jako jsou C++ a Java, kde je definice objektu uložena ve třídě [15].

4 Analýza a návrh modulu Uchazeč

4.1 Analýza

Cílem práce je vytvořit modul Uchazeč sloužící budoucím studentům zajímajícím se o studium na Univerzitě Hradec Králové. Modul bude rozšiřovat již existující aplikaci UHK Helper. Daný modul by měl potenciálním studentům nabídnout informace týkající se studia na jednotlivých fakultách Univerzity Hradec Králové. Dále by měl poskytnout informace o univerzitě samotné, o fakultách a různé další informace, které by mohly pomoci uchazečům o studium při výběru vysoké školy. Modul Uchazeč bude dále umožňovat procházet a přehrávat promo videa z facebookových stránek jak celé Univerzity Hradec Králové, tak také z jednotlivých fakult. V neposlední řadě bude tato část aplikace nabízet možnost kontaktovat jednotlivá studijní oddělení dle fakult.

Z výše uvedených důvodů vyplynul záměr rozdělit modul Uchazeč do třech logických sekcí. První sekce bude nazvána Informace a bude obsahovat veškeré zmíněné informace. Druhá sekce bude obsahovat videa o univerzitě a fakultách. Tato sekce bude pojmenována jako Videa. Poslední sekce nazvaná Kontakt se bude skládat z kontaktního formuláře umožňujícího napsat email na jednotlivá studijní oddělení fakult.

Stávající aplikace UHK Helper podporuje minimální API level 14, což odpovídá verzi Androidu 4.0.1 Ice Cream Sandwich. Z tohoto důvodu bude kladen důraz na optimalizaci i pro starší verze Androidu, které v současné době již nejsou běžně využívané.

4.2 Návrh modulu Uchazeč

Veškeré návrhy UI byly zrealizovány pomocí online nástroje pro tvorbu GUI FluidUI[2]. V tomto nástroji je možné po registraci pomocí bezplatné licence vytvořit jeden projekt obsahující až deset stránek (jednotlivých obrazovek), což bylo pro účely této práce zcela dostačující. Jedinou nevýhodou v bezplatné verzi je mírné

[2] https://www.fluidui.com/

omezení v nabízených nástrojích. V placené verzi nástroj umožňuje dokonce týmovou spolupráci několika členů týmu.

Jako první krok bude přidána vlastní položka do hlavního aplikačního menu. Skrze tuto položku v menu se bude přistupovat k hlavnímu fragmentu Uchazeč. Tento fragment bude sloužit jako rozcestí mezi dalšími fragmenty. V hlavním fragmentu Uchazeč se budou nacházet tři velká tlačítka, která umožní přechod na jednotlivé fragmenty Kontakt, Videa a Informace. V tlačítku Informace se bude nacházet ještě podmenu s konkrétními oblastmi informací. Návrh fragmentu Uchazeč je znázorněn na obrázku 6.

Pro informace je uvažováno rozbalovací menu, kde jednotlivé položky obsahují konkrétní informace. Návrh rozbaleného menu je znázorněn na obrázku 7.

Obrázek 6 Návrh fragmentu Uchazeč. Zdroj vlastní tvorba

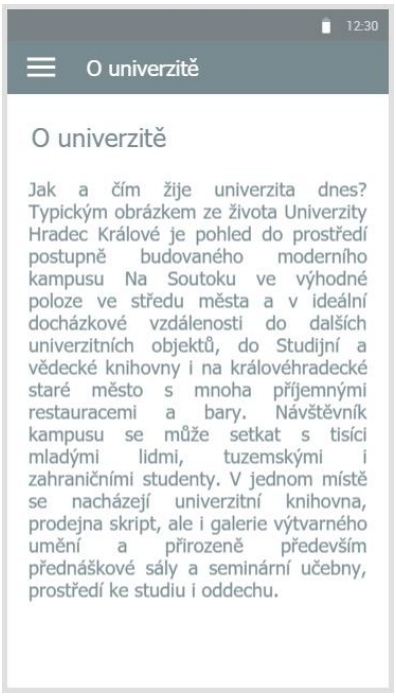
Obrázek 7 Návrh fragmentu Uchazeč s rozbaleným menu. Zdroj vlastní tvorba

4.2.1 Informace

Fragment Informace bude zobrazovat informace o univerzitě, o jednotlivých fakultách, o studijních oborech nabízených fakultami, dále informace o stipendiích, o Erasmu, o Noci vědců, o dnech otevřených dveří konkrétních fakult a také

informace o studentském životě, jako například ubytování, doprava, stravování, volný čas nebo důležitá místa vzhledem k Univerzitě Hradec Králové.

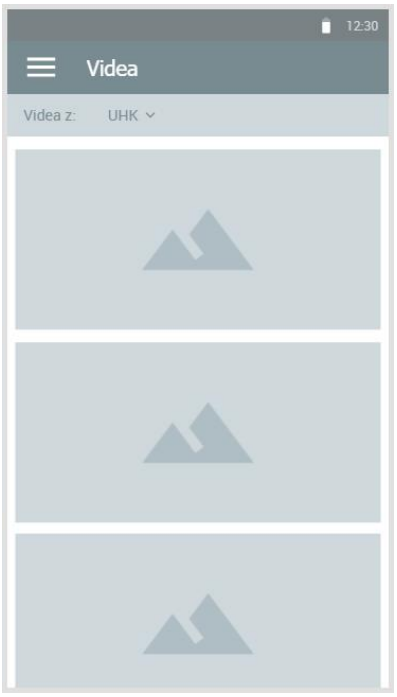
Data pro informace budou získávána z různých webů poskytujících informace o univerzitě, kdy ze stránky se vždy vyextrahují pouze důležité informace, které se mají uživateli zobrazit. Informace budou zobrazovány v android komponentě *WebView* jako naformátované HTML. Návrh fragmentu Informace je znázorněn na obrázku 8, kde se jedná již o konkrétní informace, a to o univerzitě.



Obrázek 8 Návrh fragmentu Informace. Zdroj vlastní tvorba

4.2.2 Videa

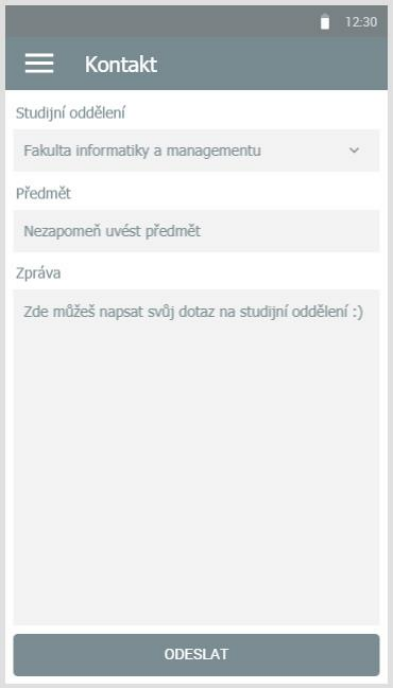
Fragment Videa bude sloužit k zobrazování jednotlivých videí. Videa budou získávána z facebookových stránek Univerzity Hradec Králové a z facebookových stránek všech fakult. Nejprve se zobrazí vždy pět nejnovějších videí, kdy uživatel bude moci načítat starší videa. V případě, že se uživatel dostane na konec daných pěti videí, načte se dalších pět videí. Jakmile se uživatel dostane na úplný konec, kdy již všechna videa byla načtena, bude uživateli tato skutečnost oznámena formou výpisu. Tato videa bude možné filtrovat podle jednotlivých fakult, případně přímo univerzity. Návrh fragmentu Videa je znázorněn na obrázku 9.



Obrázek 9 Návrh fragmentu Videa. Zdroj vlastní tvorba

4.2.3 Kontakt

Poslední fragment, do kterého bude možné přistupovat z hlavního fragmentu Uchazeč, se bude nazývat Kontakt. Tento fragment bude umožňovat uživateli odesílat dotazy na studijní oddělení příslušných fakult. Odesílání dotazů bude realizováno skrze nativní emailový klient. Fragment bude obsahovat komponentu *Spinner*, ze které bude možné zvolit studijní oddělení požadované fakulty. Dále se bude formulář skládat z pole pro zadání předmětu, a nakonec z pole pro zadání požadovaného dotazu na studijní oddělení. V poslední řadě bude formulář obsahovat tlačítko, kdy po kliknutí budou vyplněné informace předány nativnímu emailovému klientu. Takto bude moci uživatel poslat email ze své emailové adresy, na které může následně očekávat odpověď od studijního oddělení. Návrh vzhledu fragmentu Kontakt je znázorněn na obrázku 10.



Obrázek 10 Návrh fragmentu Kontakt. Zdroj vlastní tvorba

5 Popis vybraných aspektů implementace

Kapitola pojednává o klíčových implementačních detailech, které bylo nutné vyřešit. Převážná část implementace byla naprogramována v programovacím jazyce Java. Jistou roli zde sehrály i technologie jako JavaScript, HTML a CSS. Aplikace UHK Helper je cílená pro skupinu android zařízení v rozmezí API levelu 14 – 28, tj. pro verze Android 4.0.1 – 9 Pie. Z těchto důvodů bylo nutné provést určité optimalizace, aby byla zajištěna funkčnost na všech těchto zařízeních.

5.1 Fragment Uchazeč

Fragment Uchazeč je hlavní fragment tvořící rozcestí, který také zprostředkovává komunikaci mezi jednotlivými fragmenty. Fragment se skládá z jednotlivých komponent, které umožňují přechod na ostatní tři fragmenty Informace, Videa a Kontakt. Jedná se o komponentu drop down menu a section button. Tyto komponenty jsou popsány dále v textu.

Fragment Uchazeč vychází ze třídy *BaseFragment*, jejíž tvůrcem je autor aplikace UHK Helper. Ve třídě *BaseFragment* jsou již předdefinovány životní cykly fragmentu, různé komponenty specifické pro aplikaci a definovány abstraktní metody. To velmi zjednodušilo vývoj vlastních fragmentů, kdy všechny fragmenty vytvořené pro modul Uchazeč rozšiřují třídu *BaseFragment*. V podstatě bylo třeba pouze rozšířit třídu *BaseFragment* a přepsat metody *getTitleRes* a *getLayoutResId*. Metoda *getTitleRes* vrací ID zdroje textového řetězce obsahující nadpis pro daný fragment. Druhá metoda *getLayoutResId* vrací ID požadovaného grafického rozložení, dále jen layout.

Komponenta drop down menu obsahuje veškeré informace uvedené v kapitole **4.2.1 Informace**. V tomto fragmentu je přiřazena obsluha události po kliknutí na danou položku menu. Nejprve dochází k ověření, zda je zařízení připojeno k internetu. V případě, že zařízení připojeno není, je zobrazena vlastní komponenta *AlertDialog* informující uživatele o této skutečnosti. V opačném případě je zavolána metoda *handleDropDownMenuOnClickEvent*, kdy je v parametru předána reference na konkrétní *View*. V metodě *handleDropDownMenuOnClickEvent*

se nejprve získá ID *View,* na které bylo kliknuto a textový řetězec nadpisu. Tento nadpis se použije jako hlavní nadpis fragmentu *InformationViewer*. Za získávání informací je zodpovědná třída *InformationGetter*, která obsahuje jednotlivé metody pro získávání konkrétních informací. Každá z těchto metod vrací tzv. Callable task. Podle *View*, na které bylo kliknuto, je zavolána metoda nad instancí *InformationGetter*. Po získání Callable tasku je daný task předán *ExecutorService*, která je zodpovědná za spuštění tasku v samostatném vlákně a vrací instanci objektu *Future*. Systém Callable tasku, včetně tříd *ExecutorService* a *Future*, je součástí Java Concurrency API umožňující paralelní zpracování úloh v samostatných vláknech. Je to z důvodu, kdy získávání dat z webu a jejich následné parsování trvá poměrně dlouhou dobu. Není možné v hlavním UI vlákně androidu spouštět tak časově náročnou úlohu a zároveň by se aplikace mohla jevit uživateli jako „zamrzlá". Fragment Uchazeč využívá pro účely získávání informací jedno vlákno, které se vytvoří na začátku životního cyklu fragmentu v callback metodě *onViewCreated*. Toto vlákno existuje po celou dobu existence fragmentu Uchazeč. Jamile dojde k ukončování fragmentu Uchazeč, je v callback metodě *onDestroyView* dané vlákno terminováno. Po získání instance objektu *Future* je uživateli zobrazena načítací obrazovka. Hned poté je vytvořeno speciální vlákno, které slouží k tomu, aby nedocházelo k blokování hlavního UI vlákna vlivem čekání na výsledek objektu *Future*. Jakmile je totiž nad instancí *Future* zavolána metoda *get*, dojde k zablokování aktuálního vlákna, dokud není Callable task dokončen. Po získání dat z *Future* je třeba otestovat, zda se jedná o instanci typu *Url* nebo *Information*. Na základě toho je rozhodnuto, zda se otevře internetový prohlížeč, nebo zda se zobrazí informace ve fragmentu *InformationViewer*. Výše popsaná metoda *handleDropDownMenuOnClickEvent* je uvedena níže v ukázce kódu 2.

```
private void handleDropDownMenuOnClickEvent(View v) {
    int id = v.getId();
    String title = (String) v.getTag();

    final Future<Url> information;
    switch(id) {
        case R.id.menu_about_university:
            information = executor.submit(
                        informationGetter.aboutUniversity(title));
            break;
        case R.id.menu_faculty_fim:
            information = executor.submit(
                        informationGetter.aboutFacultyFim(title));
            break;
                .
                .   // další bloky case
                .
        default:
            return;
    }
    if(information == null)
        return;

    toggleLoadingVisibility();

    // vytvoreni cekaciho threadu, ktery ceka dokud neni ziskana
       informace, aby nedochazelo k blokovani main UI threadu
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                Url data = information.get();
                if(data instanceof Information) {
                     Information i = (Information) data;
                     openInformationViewerFragment(i);
                } else if(data instanceof Url) {
                     openBrowser(data.getUrl());
                     toggleLoadingVisibility();
                }
            } catch (ExecutionException e) {
                e.printStackTrace();
                toggleLoadingVisibility();
                showDialogProblemDuringGettingData();
            } catch (InterruptedException e) {
                e.printStackTrace();
                toggleLoadingVisibility();
            }
        }
    }).start();
}
```

Ukázka kódu 2 Obsluha události OnClick komponenty DropDownMenu. Zdroj: vlastní zpracování

5.1.1 Drop down menu komponenta

Jedná se o komponentu vytvořenou pro účely této práce, kdy bylo třeba zajistit vyhovující zobrazování informací. Jako vhodný způsob se ukázalo zobrazovat informace v rozbalovacím menu. Komponenta byla navržena obecně, aby bylo možné vytvořit komponentu jak s vlastním definovaným vzhledem zapsaným v souboru layout resource xml, tak s již předdefinovaným vzhledem. Mezi další významnou vlastnost komponenty patří schopnost zanořování položek až do sedmi úrovní. Jedná se o základní hodnotu, která může být v případě potřeby změněna. Komponenta se inicializuje skrze menu xml resource, kde je dané menu včetně podmenu nadefinováno. Část definovaného menu je znázorněna v ukázce kódu 3. Aby nedocházelo k nepřiměřenému zmenšování textu vlivem zanořování na základě výpočtu velikosti textu dle úrovně zanoření, je nastavena minimální hodnota velikosti textu. Základně je tato hodnota 8 sp[3] (scalable pixels), avšak i tuto hodnotu je možné přenastavit.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_about_university"
        android:title="@string/about_university" />

    <item
        android:id="@+id/menu_about_faculty"
        android:title="@string/about_faculty" >

        <menu>
            <item
                android:id="@+id/menu_faculty_fim"
                android:title="@string/faculty_fim"/>
            <item
                android:id="@+id/menu_faculty_prf"
                android:title="@string/faculty_prf"/>
        </menu>
    </item>
</menu>
```

Ukázka kódu 3 Část menu xml resource deklarující položky menu informací. Zdroj: vlastní zpracování

Třída *DropDownMenu* obsahuje dva základní konstruktory, přičemž první z nich přijímá následující parametry: *LinearLayout* jako kořenový element, do

[3] Jedná se o jednotku využívanou pro velikost textu nezávislou na hustotě pixelů. Jednotka také respektuje uživatelovo nastavení velikosti textu v systému [16].

kterého bude *DropDownMenu* komponenta vložena, ID menu resource xml, *Context*, *LayoutInflater* a *OnClickListener*. Druhý konstruktor místo *LinearLayoutu* přijímá *View* definující vzhled *DropDownMenu* komponenty, a navíc přijímá ještě další dva parametry: ID resource xml layoutu menu položky a ID resource xml layoutu menu s vnořenými položkami. První konstruktor umožnuje vytvořit komponentu s definovaným vzhledem, druhý se vzhledem vlastním.

Metodou zodpovědnou za vytváření celého menu je metoda *createMenu*. Tato metoda využívá rekurze, kdy projde vždy jednu úroveň v menu. V případě, kdy položka obsahuje další podmenu, je tato metoda rekurzivně zavolána. Postup vytváření položek menu, ať už se jedná o položku s podmenu či bez něho, je stejný, až na drobné výjimky. V prvé řadě dojde v metodě *createMenuItemView* k vytvoření objektu *View* pro konkrétní položku. Poté je nastaven text dané položky. Následuje vypočítání parametru *t*, který slouží pro výpočet velikosti textu a velikosti odsazení. Tyto vypočítané hodnoty jsou následně nastaveny konkrétnímu *View*. Vytváření položky menu je znázorněno v ukázce kódu 4.

```
private View createMenuItemView(int menuItemLayoutResId, MenuItem mi,
        int level, LayoutInflater inflater, LinearLayout rootElement,
        Context context) {
    // vytvoreni view pro polozku menu
    View v =inflater.inflate(menuItemLayoutResId, rootElement, false);
    v.setId(mi.getItemId());
    v.setTag(mi.getTitle());

    // nastaveni textu pro polozku menu
    TextView tv = v.findViewById(R.id.itemNameTextView);
    tv.setText(mi.getTitle());

    // vypocitani parametru t slouziciho pro vypocet velikosti textu
       a velikosti odsazeni
    float t = 1 - (level / 10f);

    // vypocet a nastaveni velikosti textu
    float calculatedTextSize = ViewUtils.pxToSp(
                                    tv.getTextSize(), context) * t;
    float textSize = (calculatedTextSize > minTextSize) ?
                            calculatedTextSize : minTextSize;
    tv.setTextSize(TypedValue.COMPLEX_UNIT_SP, textSize);

    // vypocet a nastaveni velikosti paddingu
    float paddingInDp = ViewUtils.pxToDp(
                                    tv.getPaddingLeft(), context);
    paddingInDp = paddingInDp * (1 - t + 0.1f) * 10;
    tv.setPadding((int) TypedValue.applyDimension(
                      TypedValue.COMPLEX_UNIT_DIP, paddingInDp,
                      context.getResources().getDisplayMetrics()),
                      tv.getPaddingTop(), 0, tv.getPaddingBottom());

    return v;
}
```

Ukázka kódu 4 Vytváření položky menu. Zdroj: vlastní zpracování

V metodě *createMenu* následuje nastavení *OnClickListeneru*, kdy pro položku s podmenu je přiřazen listener definující základní chování. Jedná se pouze o jednoduché zobrazování a skrývání obsahu podmenu. Pokud se jedná o položku bez podmenu, je přiřazen hlavní *OnClickListener*, který byl předán komponentě *DropDownMenu*. Poté jsou vytvořená *View* vložena do kořenového elementu. V případě položky s podmenu je rekurzivně zavolána metoda *createMenu*. Metoda *createMenu* je znázorněna v ukázce kódu 5.

```
private void createMenu(Menu menu, LinearLayout rootElement,
                int level, LayoutInflater inflater, Context context) {
    for(int i = 0; i < menu.size(); i++) {
        MenuItem mi = menu.getItem(i);
        if(mi.hasSubMenu()) {
            View v = createMenuItemView(menuItemWithSubitemsResId, mi,
                        level, inflater, rootElement, context);

            LinearLayout subItemsRootElement = v
                                .findViewById(R.id.itemsLinearLayout);

            // nastaveni vychoziho on click listeneru pro vychozi akci
               pri kliknuti na polozku menu s podmenickem
            v.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    LinearLayout itemsLinearLayout = v
                                .findViewById(R.id.itemsLinearLayout);
                    ImageView iconImageView = v
                                .findViewById(R.id.iconImageView);

                    if(itemsLinearLayout.getVisibility() ==
                       View.VISIBLE) {
                        itemsLinearLayout.setVisibility(View.GONE);
                        iconImageView.setImageResource(R.drawable
                        .ic_keyboard_arrow_right_color_primary_24dp);
                    } else {
                        itemsLinearLayout.setVisibility(View.VISIBLE);
                        iconImageView.setImageResource(R.drawable
                        .ic_keyboard_arrow_down_color_primary_24dp);
                    }
                }
            });
            // pridani polozky menu s podmenickem
               do korenoveho elementu
            rootElement.addView(v);

            // nejsme-li na konci ve vnorovani, pokracujeme
               rekurzivne v prohledavani podmenu
            if(level < maxNesting - 1)
                createMenu(mi.getSubMenu(), subItemsRootElement,
                                level + 1, inflater, context);
        } else {
            View v = createMenuItemView(menuItemResId, mi, level,
                        inflater, rootElement, context);

            // zaregistrovani on click listeneru pro polozku menu
            v.setOnClickListener(listener);

            // pridani polozky menu do korenoveho elementu
            rootElement.addView(v);
        }
    }
}
```

Ukázka kódu 5 Metoda vytvářející menu v drop down menu komponentě. Zdroj: vlastní zpracování

5.1.2 Section button komponenta

SectionButton je jednoduchá komponenta tvořící tlačítko s libovolným vzhledem definovaným v layout resource xml. Layout by měl obsahovat *TextView* a *ImageView*. Pro konstrukci komponenty je nutné v konstruktoru předat *View* definované vlastním layout resource xml. Dále ID zdroje textového řetězce, ID drawable resource a *OnClickListener*. Deklarace komponenty *SectionButton* je znázorněna v ukázce kódu 6.

```
public class SectionButton {

    private View view;

    @Nullable
    @BindView(R.id.titleTextView)
    TextView titleTextView;

    @Nullable
    @BindView(R.id.iconImageView)
    ImageView iconImageView;

    public SectionButton(View view, int titleResId, int iconResId,
                         View.OnClickListener listener) {
        this.view = view;
        ButterKnife.bind(this, view);
        titleTextView.setText(titleResId);
        iconImageView.setImageResource(iconResId);
        view.setOnClickListener(listener);
    }
}
```

Ukázka kódu 6 Deklarace komponenty SectionButton. Zdroj: vlastní zpracování

5.2 Získávání informací

Za získávání informací jsou zodpovědné třídy *InformationGetter* a *OldWebInformationGetter*. Informace jsou získávány z webů poskytujících informace o univerzitě. Hned poté jsou vyextrahovány pouze potřebné informace. K těm je následně přiřazen definovaný styl pro aplikaci. Nakonec je informace vrácena jako samostatný objekt obsahující HTML string se stylem.

Vzhledem k tomu, že se Univerzita Hradec Králové brzy chystá přejít na nové webové stránky, byl zvolen následující návrh pro získávání informací z webu. Aby bylo možné při změně webu co nejjednodušeji vyměnit implementaci, byla zavedena abstraktní třída *InformationGetter*. Tato třída definuje celkem devatenáct abstraktních metod pro získávání konkrétních informací. Dále třída obsahuje

metody pro práci s URL adresami, kdy se uvažují dvě URL adresy pro informace v českém a v anglickém jazyce. Třída *InformationGetter* má vnořenou třídu *Style* vyjadřující načtený styl, přičemž třída *InformationGetter* drží referenci na instanci třídy *Style*.

Třída *Style* načte klasický CSS styl, který může být obohacený o dvě specifické vlastnosti. První vlastnost umožňuje využívat třech zástupných symbolů pro barvy dle aktuálně zvoleného tématu android aplikace. Jedná se o tyto tři zástupné symboly: *?colorPrimary*, *?colorPrimaryDark*, *?colorAccent*. Druhou, o něco méně významnou vlastností, je možnost napsání jednořádkového komentáře, pokud řádek začíná symbolem „#“. V konstruktoru je načten CSS styl ze souboru dle názvu předaného v parametru. Během načítání CSS stylu jsou zástupné symboly barev nahrazeny konkrétní barvou nastavenou v tématu. Řádky začínající symbolem „#“ jsou vynechány. Třída *Style* také obsahuje metodu *applyStyle* přijímající v parametru objekt knihovny Jsoup obsahující HTML DOM elementy, na které je v této metodě aplikován načtený CSS styl. Třída *Style* je znázorněna v ukázce kódu 7.

```
protected class Style {
    private static final String COLOR_PRIMARY = "?colorPrimary";
    private static final String COLOR_PRIMARY_DARK =
                                                "?colorPrimaryDark";
    private static final String COLOR_ACCENT = "?colorAccent";
    private static final String COMMENT = "#";
    private String style = "";
    protected Style(Context context, String styleFileName) {
        try {
            BufferedReader reader = new BufferedReader(
                    new InputStreamReader(context.getResources()
                            .getAssets().open(styleFileName)));
            // ziskani hodnot barev nastavenych dle aktuálního
               themu v androidu
            TypedArray arr = context.getTheme()
                    .obtainStyledAttributes(
                            PrefManager.getStyleResouce(),
                            new int[]{R.attr.colorPrimary,
                                    R.attr.colorPrimaryDark,
                                    R.attr.colorAccent});
            int primaryColor = arr.getColor(0, 0);
            int primaryDarkColor = arr.getColor(1, 0);
            int accentColor = arr.getColor(2, 0);
            arr.recycle();
            String line;
            while((line = reader.readLine()) != null) {
                if(line.startsWith(COMMENT))
                    continue;
                if(line.contains(COLOR_PRIMARY_DARK)) {
                    line = line.replace(COLOR_PRIMARY_DARK,
                            intToHexColor(primaryDarkColor));
                }
                if(line.contains(COLOR_PRIMARY)) {
                    line = line.replace(COLOR_PRIMARY,
                            intToHexColor(primaryColor));
                }
                if(line.contains(COLOR_ACCENT)) {
                    line = line.replace(COLOR_ACCENT,
                            intToHexColor(accentColor));
                }
                style += line;
            }
            style = "<style>" + style + "</style>";
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Metoda aplikujici nacteny styl k danemu html.
    protected String applyStyle(Elements elements) {
        return style + elements;
    }
    // Pomocna metoda pro vyjadreni barvy v hash stringu.
    private String intToHexColor(int color) {
        return String.format("#%06X", (0xFFFFFF & color));
    }
}
```

Ukázka kódu 7 Deklarace třídy Style. Zdroj: vlastní zpracování

Třída *OldWebInformationGetter* je implementací pro starý web UHK. Jakmile přejde Univerzita Hradec Králové na nový web, postačí rozšířit třídu *OldWebInformationGetter* a přepsat metody získávající informace ze starého webu. Obecně lze říci, že kdykoliv bude třeba vyměnit data, odkud by se měla získávat, stačí vhodně rozšířit jednu z tříd a přepsat metody, které mají informace získávat.

Třída obsahuje nejprve deklaraci polí typu *String*, ve kterých jsou uloženy URL adresy pro jednotlivé webové stránky. Toto pole obsahuje buď jednu nebo dvě adresy. Na prvním místě pole se nachází adresa v českém jazyce, na druhém místě v jazyce anglickém. Při deklaraci polí obsahujících URL adresy je důležité, aby byl modifikátor přístupu nastavený na hodnotu *private*. Je to z důvodu, když bude třída rozšiřována, aby bylo možné v nové třídě deklarovat pole se stejným názvem a jinými adresami. Dále by nová implementace měla volat výchozí konstruktor předka (třídy *InformationGetter*) tak, jak je tomu u třídy *OldWebInformationGetter*. Třída dále obsahuje pomocné metody sloužící k vybrání pouze podstatných dat z webu. „Čištění" dat je prováděno na základě domény. Příklad metody provádějící „čištění" obsahu je uveden v ukázce kódu 8.

```
private Elements getClearedContentPknf(String[] urls)
                throws IOException {
    String url = getUrlAccordingToLocale(urls);
    Document doc = Jsoup.connect(url).get();
    Elements content = doc.getElementsByClass("item-page")
                          .select("article");
    content.first().addClass("pageContent");
    content.select("font").unwrap();

    return content;
}
```

Ukázka kódu 8 Příklad metody vybírající informace z webu. Zdroj: vlastní zpracování

Poté již třída obsahuje pro každou informaci z rozbalovacího menu implementaci metody, která vrátí tzv. Callable task. V tomto tasku se provede získání dat z webu a jejich parsování. Data se vrátí jako naformátovaný HTML string se stylem v objektu *Information*. Příklad metody vracející informaci jako Callable task je uveden v ukázce kódu 9. Jelikož ze současného webu univerzity nebylo možné vhodně získat informace o studijních oborech jednotlivých fakult, byl zvolen návrh dvou tříd umožňující vrátit jak naformátovanou informaci, tak samostatnou

URL adresu. Callable task je definovaný generickým rozhraním *Callable* obsahujícím metodu *call*. Výsledkem Callable tasku je získání požadovaného datového typu uvedeného v generickém parametru z metody *call*. Na základě toho byla vytvořena hierarchie tříd *Url* a *Information*, kdy veškeré metody z *InformationGetteru* vracejí Callable task parametrizovaný třídou *Url*. Třída *Url* obsahuje pouze string vyjadřující URL adresu. Třídu *Url* rozšiřuje třída *Information*, které sama potřebuje URL adresu. Díky tomu je z Callable tasku možné vrátit URL adresu, která se zobrazí v internetovém prohlížeči, nebo informaci, která se zobrazí naformátovaná ve fragmentu *InformationViewer*.

```
public Callable<Url> aboutUniversity(final String title) {
    return new Callable<Url>() {
        @Override
        public Url call() throws Exception {
            String baseUrl = getBaseUrl(URL_ABOUT_UNIVERSITY);
            Elements content =
                        getClearedContent(URL_ABOUT_UNIVERSITY);
            content.select("h2").remove();
            String htmlString = style.applyStyle(content);
            Information information = new Information(baseUrl,
                                              htmlString, title);

            return information;
        }
    };
}
```

Ukázka kódu 9 Příklad metody vracející informaci jako Callable task. Zdroj: vlastní zpracování

5.3 Zobrazování informací

Za zobrazování informací je zodpovědný fragment *InformationViewer*. Samotné informace jsou zobrazovány pomocí komponenty *WebView*. Fragment obsahuje statickou metodu *newInstance* přijímající informace jako objekt *Information*. Tato metoda slouží k vytváření instancí *InformationViewer* fragmentu, přičemž nové instanci fragmentu předává informace skrze objekt *Bundle*. Jakmile je fragment vytvořen, v callback metodě *onViewCreated* je provedena potřebná inicializace fragmentu. Nejprve jsou získány informace, které byly předány fragmentu při vytváření instance. Poté je nastaven hlavní nadpis fragmentu získaný z objektu *Information*. Dále je povolen JavaScript v komponentě *WebView* a nakonec komponenta načte data z HTML stringu získaného z objektu *Information*.

Komponentě *WebView* je také předána základní URL adresa. Základní adresa se předává komponentě *WebView* z důvodu, že HTML string často obsahuje odkazy na další stránky z daného webu. Aby bylo možné otevírat odkazy v externím prohlížeči, musí být možné sestavit celou URL adresu. Odkazy na webových stránkách se typicky píší v relativním tvaru, přičemž neobsahují doménu webu, na kterém se nachází. Výše popsaný fragment je znázorněn v ukázce kódu 10.

```
public class InformationViewerFragment extends BaseFragment {

    private static final String KEY_INFORMATION = "information";

    @BindView(R.id.webView)
    WebView webview;

    // Vytvori instanci InformationViewer fragmentu.
    public static InformationViewerFragment newInstance(
                        Information information) {
        InformationViewerFragment fragment =
                                        new InformationViewerFragment();
        Bundle bundle = new Bundle();
        bundle.putSerializable(KEY_INFORMATION, information);
        fragment.setArguments(bundle);

        return fragment;
    }

    @Override
    public void onViewCreated(@NonNull View view,
                          @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        final Information information =
        (Information) getArguments().getSerializable(KEY_INFORMATION);
        setTitle(information.getTitle());
        webview.getSettings().setJavaScriptEnabled(true);
        webview.loadDataWithBaseURL(information.getUrl(),
        information.getHtmlString(), "text/html", "utf-8", null);
    }
}
```

Ukázka kódu 10 Klíčové části třídy InformationViewerFragment. Zdroj: vlastní zpracování

5.4 Fragment Videa

Jedná se o fragment, jak je již z názvu patrné, sloužící ke zobrazování videí. Videa jsou načítána z facebookových stránek jednotlivých fakult a Univerzity Hradec Králové skrze Graph API. Jelikož Graph API již neumožňuje získávat surová binární data videa, je využíváno HTML elementu *iframe* obsahujícího video, sloužícího pro embedování videí na HTML stránku. Videa jsou tedy z Graph API získávána jako

HTML elementy *iframe*. Z tohoto důvodu je logika zobrazování videí ponechána na webových technologiích a naprogramována v programovacím jazyce JavaScript. Tato logika je uvedena v souboru *videos.html* nacházejícím se ve složce *assets*, kdy tento soubor je předán komponentě *WebView* ve fragmentu *Videos*.

Fragment obsahuje vlastní komponentu *VideoEnabledWebView*, protože ve vestavěné komponentě *WebView* není podporováno přehrávání HTML5 videí ve zobrazení přes celou obrazovku, tzv. fullscreen. Komponenta *VideoEnabledWebView* byla převzata z GitHubu[4]. Autorem komponenty je Cristian Perez, který ji zpřístupnil pod licencí MIT. Tato licence umožňuje jak soukromé, tak komerční využití, libovolné upravování a šíření.

V metodě *onViewCreated* je opět provedena inicializace samotného fragmentu. Nejprve je provedeno nastavení komponenty *VideoEnabledWebView*. Jelikož na zařízeních se starší verzí Androidu docházelo k zobrazování hlavního aplikačního menu pod videem, je využito zachytávání události *onToggledFullscreen*. Tato událost je zavolána ve chvíli, kdy se změní režim přehrávání videa. V případě, kdy je zavolána callback metoda *toggledFullscreen* s booleanovou hodnotou *true*, je aplikačnímu menu zakázáno se zobrazit. V opačném případě je toto aplikační menu opět povoleno. Pokud by nedošlo k opětovnému povolení zobrazení aplikačního menu v rámci callback metody *toggledFullscreen*, je zobrazení aplikačního menu opět povoleno při ukončování fragmentu *Videos* v metodě *onDestroyView*. Poté je nezbytné, aby byl povolen JavaScript. Nakonec dojde k načtení samotného souboru *videos.html*. Výše popsaná inicializace je uvedena v ukázce kódu 11.

[4] https://github.com/cprcrack/VideoEnabledWebView

```
public class VideosFragment extends BaseFragment {

    @BindView(R.id.webView)
    VideoEnabledWebView webView;

    private DrawerLayout drawer;

    @Override
    public void onViewCreated(@NonNull View view,
                        @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        drawer = getActivity().findViewById(R.id.drawer_layout);

        View nonVideoLayout = getActivity()
                              .findViewById(R.id.nonVideoLayout);
        ViewGroup videoLayout = getActivity()
                              .findViewById(R.id.videoLayout);
        View loadingView = getLayoutInflater()
                              .inflate(R.layout.video_loading, null);

        VideoEnabledWebChromeClient webChromeClient =
               new VideoEnabledWebChromeClient(nonVideoLayout,
                       videoLayout, loadingView, webView);
        webChromeClient.setOnToggledFullscreen(
               new VideoEnabledWebChromeClient
                       .ToggledFullscreenCallback() {
            @Override
            public void toggledFullscreen(boolean fullscreen) {
                // na starsim androidu se ve fullscreenu drawer
                   vykresloval pod videem
                if(fullscreen) {
                    drawer.setDrawerLockMode(
                              DrawerLayout.LOCK_MODE_LOCKED_CLOSED);
                } else {
                    drawer.setDrawerLockMode(
                              DrawerLayout.LOCK_MODE_UNLOCKED);
                }
            }
        });
        webView.setWebChromeClient(webChromeClient);
        webView.getSettings().setJavaScriptEnabled(true);
        webView.loadUrl("file:///android_asset/videos.html");
    }
}
```

Ukázka kódu 11 Inicializace třídy VideosFragment v metodě onViewCreated. Zdroj: vlastní zpracování

Soubor *videos.html* obsahuje základní HTML. Toto HTML se skládá z fixního menu, ve kterém se nachází komponenta select box sloužící pro výběr facebookové stránky, ze které se mají videa zobrazit. V položkách select boxu se nachází ID konkrétní facebookové stránky. Dále se v HTML nachází element *div* tvořící obal pro videa. Součástí souboru *videos.html* jsou také jednoduché CSS styly.

Pro komunikaci s Graph API je využíváno Facebook SDK pro JavaScript. Jakmile je SDK načteno, dojde k inicializaci aplikace. Během inicializace se načte prvních pět videí na základě zvolené výchozí možnosti v select boxu skrze funkci *loadVideos*. Funkce přijímá parametr ID facebookové stránky a ukazatel na stránku s dalšími pěti videi. Graph API vrátí počet záznamů dle nastaveného parametru limit, přičemž s vrácenými záznamy je obdržen i ukazatel na stranu s následujícími záznamy. Uvnitř funkce *loadVideos* je volána funkce *api* z Facebook SDK, které se předají tři parametry. Prvním parametrem je cesta k videím sestávající se z ID konkrétní facebookové stránky. Druhým parametrem je objekt sloužící k nastavení Facebook SDK. Posledním parametrem je callback funkce zpracovávající načtená videa.

V prvním kroku callback funkce je otestováno, zda aplikace získala bezchybnou odpověď. Poté je uložen následující ukazatel a testuje se, zda byla všechna videa načtena. Pokud tomu tak není, odpověď obsahuje videa, která jsou vložena do obalujícího elementu *div*. Nakonec dojde k přepočítání výšky videí podle poměru stran původního videa, protože videa se roztáhnou na maximální šířku obrazovky zařízení. Za přepočet je zodpovědná funkce *calculateHeight*, která také obsluhuje událost *onResize*. Funkce *loadVideos* je znázorněna v ukázce kódu 12.

```
function loadVideos(pageId, nextPage) {
    loading = true;
    if(allVideosLoaded)
        return;
    document.body.appendChild(loadingElement);

    FB.api(
        "/" + pageId + "/videos",
        {access_token: '104209816294440|xVZhfKIU2_TsCnsimmUoaM0YU68',
              fields: 'embed_html', limit: 5, after: nextPage},
        function (response) {
            if (response && !response.error) {
                nextPageCursor = response.paging.cursors.after;

                // pokud byla vsechna videa jiz nactena, vlozime nakonec
                   text o teto skutecnosti
                if(!response.paging.next) {
                    allVideosLoaded = true;
                    var node = buildNode(
             '<p class="message-videos-loaded">Žádná další videa!</p>');
                    document.body.appendChild(node);
                }

                removeNodes(loadingElement);

                response.data.forEach(function(element) {
                    var node = buildNode(element.embed_html);
                    document.getElementById(
                                'videos-wrapper').appendChild(node);
                });
                callculateHeight();
                loading = false;
            } else {
                removeNodes(loadingElement);
                document.getElementById(
                                'videos-wrapper').appendChild(errorMessage);
            }
        }
    );
}
```

Ukázka kódu 12 Funkce načítající videa z Graph API. Zdroj: vlastní zpracování

Jakmile uživatel provede výběr fakulty nebo univerzity, je nezbytné na tuto událost reagovat. K tomu slouží událost *onChange* na select boxu. Nejprve jsou

smazána všechna videa a zprávy, které byly vloženy do HTML dokumentu. Poté je zavolána funkce *init*, která uvede aplikaci do výchozího stavu a načte prvních pět videí dle zvolené facebookové stránky. Právě popsaný kód je uveden v ukázce kódu 13.

```
// obsluha eventu pri zmene fakulty/univerzity,
ze ktere se maji nacitat videa
var videoSource = document.getElementById("source");
videoSource.onchange = function(e) {
    var iframes = document.getElementsByTagName("iframe");
    var message = document.getElementsByClassName("message-videos-loaded");

    removeNodes(iframes);
    removeNodes(message);
    removeNodes(errorMessage);

    init();
};
```

Ukázka kódu 13 Obsluha události onChange. Zdroj: vlastní zpracování

Jelikož část funkcionality aplikace závisí na tzv. scrollování, je nutné obsluhovat *onScroll* událost. Videa jsou postupně po pěti videích načítána, jakmile se scroll okna přiblíží na konec stránky posunutý o určitý offset. Offset je vypočítán jako 5 % z aktuální výšky stránky. V případě, kdy by již horní panel nebyl vidět, dojde k nastavení tohoto panelu jako fixního menu. Obsluha *onScroll* eventu je uvedena v ukázce kódu 14.

```
// obsluha on scroll eventu
window.onscroll = function(e) {
    var doc = document.documentElement;
    var top = (window.pageYOffset || doc.scrollTop);
    var floatingPanel = document.getElementById("floating-panel");

    // postupne nacitani videi pri doscrolovani na konec - offset (cca 5%)
    if(!loading && (window.innerHeight + window.scrollY) >=
            (bodyHeight - offset)) {
        loadVideos(pageId, nextPageCursor);
    }

    // hlavni menu se nastavi jako fixni
    if(window.scrollY > floatingPanel.scrollTop) {
        floatingPanel.style.position = "fixed";
    } else {
        floatingPanel.style.position = "static";
    }
};
```

Ukázka kódu 14 Obsluha události onScroll. Zdroj: vlastní zpracování

5.5 Fragment Kontakt

Posledním fragmentem modulu Uchazeč je *ContactFragment* skládající se z kontaktního formuláře. Fragment umožňuje uživateli napsat email v aplikaci UHK Helper a poté jej odeslat příslušnému studijnímu oddělení pomocí nativního poštovního klienta.

Kontaktní formulář obsažený ve fragmentu *Contact* se skládá z několika komponent, mezi které patří *Spinner*, dvě komponenty *EditText* a *Button*. Komponenta *Spinner* obsahuje seznam položek, ze kterého může být zvolena vždy jedna položka. Položky komponenty *Spinner* jsou definovány v resource xml v elementu *string-array*. V daném formuláři komponenta *Spinner* obsahuje seznam fakult pro výběr emailové adresy studijního oddělení. První komponenta *EditText* slouží pro uvedení předmětu, druhá pro zápis samotného sdělení emailu. Po kliknutí na komponentu *Button* je iniciováno otevření externí emailové aplikace, které jsou předány pole obsahující emailovou adresu, předmět a text zprávy. Adresy pro studijní oddělení jsou získávány z výčtového typu *Faculty*, definujícího fakulty, vytvořeného autorem aplikace UHK Helper. Ve výčtovém typu je pro každou fakultu

přiřazena příslušná emailová adresa. Část třídy *ContactFragment* je znázorněna v ukázce kódu 15.

```
public class ContactFragment extends BaseFragment {

    @BindView(R.id.studyDepartmentSpinner)
    Spinner studyDepartmentSpinner;
    @BindView(R.id.subjectEditText)
    EditText subjectEditText;
    @BindView(R.id.messageEditText)
    EditText messageEditText;
    @BindView(R.id.sendButton)
    Button sendButton;

    @Override
    public void onViewCreated(@NonNull View view,
                              @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        // vytvoreni spinneru slouziciho pro vyber fakulty
        ArrayAdapter<CharSequence> adapter =
                ArrayAdapter.createFromResource(getContext(),
                R.array.faculty_array, R.layout.spinner_item);
        adapter.setDropDownViewResource(
                R.layout.spinner_dropdown_item);
        studyDepartmentSpinner.setAdapter(adapter);

        sendButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                openExternalMailApp();
            }
        });
    }

    private void openExternalMailApp() {
        String mail = getEmailAddress();
        String subject = subjectEditText.getText().toString();
        String message = messageEditText.getText().toString();

        Intent intent = new Intent(Intent.ACTION_SENDTO);
        intent.setData(Uri.parse("mailto:"));
        intent.putExtra(Intent.EXTRA_EMAIL, new String[]{mail});
        intent.putExtra(Intent.EXTRA_SUBJECT, subject);
        intent.putExtra(Intent.EXTRA_TEXT, message);

        if (intent.resolveActivity(getActivity()
                        .getPackageManager()) != null) {
            getActivity().startActivity(intent);
        }
    }
}
```

Ukázka kódu 15 Část třídy ContactFragment. Zdroj: vlastní zpracování

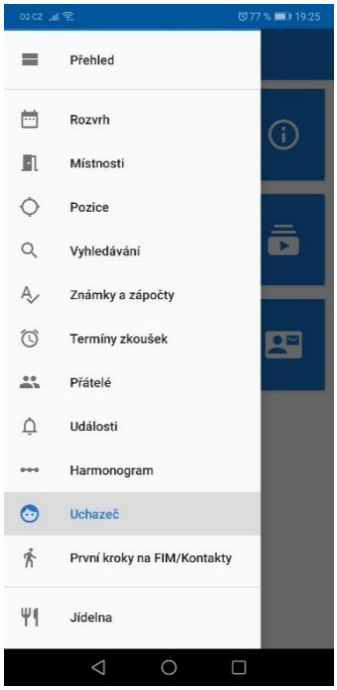
6 Výsledky a závěr

6.1 Výsledky

Výsledkem bakalářské práce je modul Uchazeč již existující mobilní aplikace UHK Helper. Tato část aplikace nabídne studentům přehledně jednotlivé informace o studiu na Univerzitě Hradec Králové, jako například informace o oborech, o studentském životě v Hradci Králové, nebo o podmínkách získání jednotlivých stipendií. Dále modul Uchazeč umožní prohlížet videa z facebookových stránek univerzity a jednotlivých fakult. Poslední zásadní funkcí pro uchazeče o studium je možnost kontaktovat studijní oddělení příslušné fakulty.

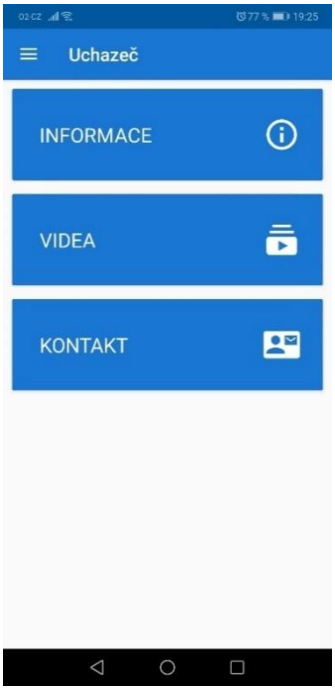
Aplikace byla testována především na dvou mobilních telefonech. Prvním zařízením byl mobilní telefon Huawei P20 Pro s verzí Androidu 8.1.0 Oreo a 9 Pie. Druhým zařízením byl Samsung Galaxy S3 Neo s verzí Androidu 4.4.2 KitKat. Modul Uchazeč by tedy měl být funkční jak na poměrně zastaralé, tak na nejnovější verzi Androidu.

Jako barva modulu je využívána již zavedená barva aplikace na základě zvolené fakulty. Celkový vzhled modulu byl zachován se stylem celé aplikace. Modul byl vytvořen s podporou jazyků čeština, slovenština a angličtina, stejně tak jako je tomu u celé aplikace. Modul Uchazeč byl tedy začleněn do stávající aplikace UHK Helper. Vlastní položka Uchazeč v hlavním menu aplikace je znázorněna na obrázku 11.

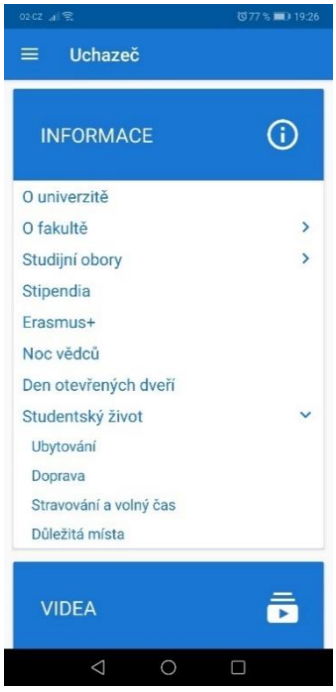


Obrázek 11 Položka Uchazeč v hlavním menu. Zdroj vlastní tvorba

Modul se skládá ze čtyř vlastních fragmentů. Prvním z fragmentů je hlavní fragment tvořící rozcestí modulu mezi ostatními fragmenty. Tento fragment je znázorněn na obrázku 12 a 13.

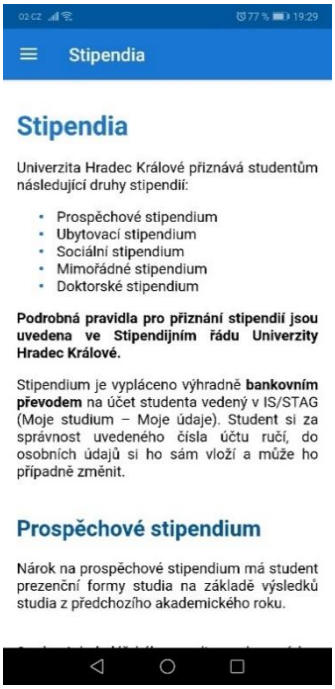


Obrázek 12 Fragment Uchazeč tvořící rozcestník modulu. Zdroj vlastní tvorba



Obrázek 13 Fragment Uchazeč s rozbaleným menu obsahujícím informace. Zdroj vlastní tvorba

Dalším fragmentem je fragment Informace. Tento fragment nabízí přehledně různé informace o studiu na Univerzitě Hradec Králové. Informace se zobrazují jako formátované HTML. Fragment Informace je znázorněn na obrázku 14 a 15.



Obrázek 14 Fragment Informace zobrazující informace o stipendiích. Zdroj vlastní tvorba

Obrázek 15 Fragment Informace zobrazující informace o Noci vědců. Zdroj vlastní tvorba

Dále modul obsahuje fragment Videa. Tento fragment obsahuje různá videa z facebookových stránek jednotlivých fakult a univerzity. Videa je možné třídit právě podle toho, z jaké fakulty mají pocházet, nebo zda mají být zobrazena videa o univerzitě. Fragment videa je znázorněn na obrázku 16.

Obrázek 16 Fragment Videa zobrazující videa z Fakulty informatiky a managementu. Zdroj vlastní tvorba

Posledním fragmentem modulu Uchazeč je fragment Kontakt, který je znázorněn na obrázku 17. Tento fragment obsahuje jednoduchý kontaktní formulář, kde stačí zvolit, které studijní oddělení chce uživatel kontaktovat, dále vyplnit předmět a dotaz, se kterým se chce uživatel na dané studijní oddělení obrátit.

Obrázek 17 Fragment Kontakt. Zdroj vlastní tvorba

6.2 Závěr

Cílem této bakalářské práce bylo analyzovat, navrhnout a implementovat modul mobilní aplikace UHK Helper sloužící zájemcům o studium na Univerzitě Hradec Králové. Cíle práce se podařilo úspěšně splnit, modul aplikace byl implementován dle stanovených požadavků.

Jelikož současní studenti univerzity aplikaci UHK Helper hojně využívají, modul sloužící uchazečům o studium na Univerzitě Hradec Králové bude jistě velkým přínosem. Tento modul nabídne budoucím studentům důležité informace, týkající se studia na vysoké škole nebo nabídne možnost kontaktovat studijní oddělení příslušné fakulty.

7 Seznam použité literatury

[1] *Mobile Operating System Market Share Worldwide | StatCounter Global Stats* [online]. c1999-2017 [cit. 2019-02-27]. Dostupné z: http://gs.statcounter.com/os-market-share/mobile/worldwide

[2] *Application Fundamentals* [online]. 2018 [cit. 2018-06-28]. Dostupné z: https://developer.android.com/guide/components/fundamentals

[3] *Introduction to Activities* [online]. 2018 [cit. 2018-06-28]. Dostupné z: https://developer.android.com/guide/components/activities/intro-activities#java

[4] *Understand the Activity Lifecycle* [online]. 2018 [cit. 2018-06-28]. Dostupné z: https://developer.android.com/guide/components/activities/activity-lifecycle

[5] *Fragments | Android Developers* [online]. [cit. 2019-03-19]. Dostupné z: https://developer.android.com/guide/components/fragments

[6] *Develop - Apple Developer* [online]. Apple, c2018 [cit. 2018-06-28]. Dostupné z: https://developer.apple.com/develop/

[7] *Multiplatformní vývoj mobilních aplikací – druhy a nástroje | Blog | Krosapp* [online]. Krosapp, 2018 [cit. 2018-06-28]. Dostupné z: http://www.krosapp.cz/Blog/multiplatformni-vyvoj-mobilnich-aplikaci-druhy-nastroje

[8] FISCHER, Roman. *Xamarin: Úvod do Xamarin Forms* [online]. 2017 [cit. 2018-06-28]. Dostupné z: https://www.skeleton.cz/uvod-do-xamarin-forms

[9] MICHÁLEK, Martin. *Weby versus aplikace* [online]. 2017 [cit. 2018-06-28]. Dostupné z: https://www.vzhurudolu.cz/prirucka/weby-vs-aplikace

[10] *Meet Android Studio | Android Developers* [online]. 2019 [cit. 2019-01-22]. Dostupné z: https://developer.android.com/studio/intro/

[11] *What is Gradle?* [online]. [cit. 2019-01-22]. Dostupné z: https://docs.gradle.org/current/userguide/what_is_gradle.html#what_is_gradle

[12] *What is Java technology and why do I need it*? [online]. [cit. 2019-01-25]. Dostupné z: https://www.java.com/en/download/faq/whatis_java.xml

[13] MCKENZIE, Cameron. *What is Java? - Definition from WhatIs.com* [online]. 2016 [cit. 2019-01-25]. Dostupné z: https://www.theserverside.com/definition/Java

[14] *Jsoup Java HTML Parser, with best of DOM, CSS, and jquery* [online]. Jonathan Hedley, c2009-2018 [cit. 2019-03-19]. Dostupné z: https://jsoup.org/

[15] *About JavaScript - JavaScript | MDN* [online]. 2018 [cit. 2019-03-19]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[16] *Support different pixel densities | Android Developers* [online]. [cit. 2019-03-19]. Dostupné z: https://developer.android.com/training/multiscreen/screendensities

8 Přílohy

1) Obsah na přiloženém CD
    a. Zdrojové kódy modulu Uchazeč v adresáři src
    b. Vygenerovaná aplikace ve formátu .apk

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Systémové inženýrství a informatika
Forma: Prezenční
Obor/komb.: Informační management (im3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Štorková Monika	Týnišťská 1197, Třebechovice pod Orebem	I1600799

TÉMA ČESKY:

Modul Uchazeč systému pro podporu výuky

TÉMA ANGLICKY:

Applicant Module for System of Education Support

VEDOUCÍ PRÁCE:

doc. Mgr. Tomáš Kozel, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Analyzovat, navrhnout a implementovat modul mobilní aplikace UHK Helper sloužící zájemcům o studium.

Osnova:
1. Úvod
2. Obecná problematika mobilního vývoje
3. Výběr a popis zvolených technologií
4. Analýza a návrh modulu Uchazeč
5. Popis vybraných aspektů implementace
6. Výsledky a závěr

SEZNAM DOPORUČENÉ LITERATURY:

Podpis studenta: Štorková **Datum:** 15.10.2018

Podpis vedoucího práce: **Datum:** 15.10.2018