Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

# Diplomová práce

# Mobilní aplikace IS/STAG na platformě Android

Plzeň 2019 Marek Zimmermann

**Místo této strany bude zadání práce.**

# Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 14. května 2019

Marek Zimmermann

# Poděkování

Rád bych poděkoval Ing. Lukáši Valentovi za trpělivost, vstřícný přístup, cenné rady a věcné připomínky, které mi během tvorby diplomové práce poskytl.

# Abstract

This thesis is focused on the development of a mobile application for accessing data in IS/STAG on Android. The theoretical part deals with the research of the modern ways of developing an Android application and exploring the existing applications that communicate with IS/STAG. The practical part describes choosing the right tools, libraries and application frameworks for developing an Android application. The next part describes the application design and analysis of what functions would fit the application. Lastly, the implementation of the application itself is described, the methods of its distribution and how it was tested.



# Abstrakt

Tato práce je zaměřena na vývoj mobilní aplikace napojené na IS/STAG. Teoretická část práce se zabývá seznámením se s problematikou moderního vývoje aplikací pro operační systém Android. Dále zkoumá již existující aplikace komunikující se systémem IS/STAG. V praktické části jsou popsány zvolené nástroje, knihovny a frameworky pro vývoj mobilní aplikace na operačním systému Android. Také je popsán návrh aplikace a analýza funkcí, které by byly vhodné pro mobilní aplikaci. Poté je popsána implementace aplikace, její architektura, funkce, možnosti její distribuce a použité metody jejího testování.

# Obsah

# 1 Úvod

Chytré mobilní telefony se v poslední době staly nedílnou součástí našich životů. Jsme zvyklí, že si s jejich pomocí můžeme nejen zavolat či napsat SMS, ale také poslechnout hudbu na cestě do práce či do školy, zkontrolovat emaily, přečíst si novinky ze zpravodajských serverů a dnes už přes ně nemalá část uživatelů provádí převody peněz v mobilní bankovní aplikaci. Mnozí už mobilní internet v telefonu berou jako samozřejmost a pravidelně na něm navštěvují různé webové stránky. Jenže ne všechny se stejně pohodlně ovládají v mobilu jako na pracovním počítači či notebooku. Proto pro spousty z nich vznikají i jejich ekvivalenty v podobě mobilních aplikací, které mají za úkol servírovat téměř identický obsah. Dělají to lépe, graficky povedeněji, většinou ve zvyku dané mobilní platformy tak, aby uživatel už při spuštění věděl, kde najde menu, jakým gestem se vrátit zpět atd.

Pro studenty Západočeské univerzity je jednou z těch důležitějších stránek web portálu IS/STAG. Ten, ačkoliv prochází v podstatě kontinuálním vývojem, jak přibývajícími funkcemi, tak mírnými i většími grafickými změnami, stále patří do právě zmiňované sekce webů, které se i po velkých úpravách na poli responzivity na mobilu neovládají vždy ideálně.

Cílem této práce je nastudovat možnosti vývoje mobilních aplikací na platformě Android, seznámit se s již existujícími aplikacemi pracujícími se systémem IS/STAG a se způsoby jejich komunikace s tímto systémem. Na základě výsledků tohoto zkoumání vybrat vhodné nástroje, knihovny a aplikační frameworky pro vývoj mobilní aplikace napojené na IS/STAG. Nakonec bude navržena aplikace pro mobilní přístup k IS/STAG pro různé uživatelské role, realizována, nasazena do provozu alespoň na ZČU a otestována její funkčnost. Součástí by pak měly být i případné návrhy na možnost budoucích úprav a vylepšení.

# 2 Android

V této kapitole je stručně popsán operační systém Android a pojmy, které jsou při jeho vývoji používány či s ním přímo souvisí.

## 2.1 Obecný popis

*Android* je mobilní operační systém založený na jádře Linuxu, který je dostupný jako otevřený software (open-source)[1]. [1]

Pro rozšíření funkcionality zařízení s OS Android lze instalovat dodatečné aplikace. Jejich hlavním zdrojem je oficiální obchod s aplikacemi – *Google Play* (dále GP). Existují i další obchody s aplikacemi – za zmínku stojí například *Galaxy Apps* od Samsungu či obchod zaměřený jen na aplikace s otevřeným zdrojovým kódem – *F-Droid*. Kromě možnosti instalovat aplikace z obchodů existuje možnost instalovat aplikace přímo přenesením *APK*, *Android Package* instalačního souboru s aplikací, na zařízení a jeho spuštěním. Protože jde o velmi rizikový krok (aplikace není před instalací nijak kontrolována), je instalace aplikací z neznámých zdrojů blokována a uživatel musí nejprve instalaci globálně povolit. [2]

Primárním IDE[2] pro vývoj Android aplikací je od prosince 2014 *Android Studio*, modifikovaná verze IDE *IntelliJ IDEA*. Aplikace lze vyvíjet v několika jazycích, mezi oficiálně podporované patří *Java*, *C++* a na akci Google I/O 2017 oznámený *Kotlin*. [31]

Google používá dva způsoby číslování verzí systému Android. První obsahuje číslo označující majoritní verzi následované jedním nebo několika čísly minoritních verzí a poté jménem. Verze jsou odděleny tečkami, např. *Android 6.0 Marshmallow*, *Android 4.0.4 Ice Cream Sandwich* atd. Toto značení je vhodné pro člověka, neboť lze vizuálně snadno verze porovnat a zjistit, jak moc se mezi sebou liší.

Druhý způsob zahrnuje číslování dle verze API spojené s danou verzí Android – čím vyšší číslo, tím novější verze a naopak, např. *API 16*, *API 22* atd. Toto značení se lépe zpracovává strojově.

[1] Některé části jsou uzavřenou technologií firmy Google, přesto je jím tento systém prezentován jako open-source.

[2] *Integrated Development Environment* – vývojové prostředí pro vytváření aplikací.

V tuto chvíli[3] je nejnovější verzí Android verze *9.0 Pie* (API 28) oficiálně představený 6. srpna 2018. Mezi novinky patří například podpora platformy pro zjišťování pozice uvnitř budov za pomocí Wi-Fi protokolu IEEE 802.11mc, nativní podpora výřezů displeje, vylepšené notifikace (podpora obrázků v notifikaci, možnost úpravy notifikačních kanálů v aplikaci atp.), podpora práce s více kamerami zároveň, vylepšený `JobScheduler`, který lépe pracuje s úlohami pracujícími se sítí, rozšíření funkčnosti `Neural Network API` a další. [3]

Android se v současné době stále těší velké popularitě. Dle průzkumu společnosti IDC ovládá trh s mobilními telefony v roce 2018 s podílem téměř 85%. [4]

Největším problémem při vývoji aplikací na platformě Android je jeho roztříštěnost. Android je možné provozovat na velkém množství různých zařízení a při vývoji je tedy nutné počítat zejména s různými displeji (rozlišení, velikost) a také se spousty různými verzemi samotného OS – v prosinci 2018 běží dle statistik od společnosti Google 99% Android zařízení na 10 různých verzích[4].

## 2.2 Služby na pozadí

Pokud pomineme komunikaci s mobilní síti pro korektní fungování základních funkcí telefonu (volání, posílání a příjem SMS atd.), mají chytré telefony spoustu služeb fungujících na pozadí. Jedním z projevů takových služeb jsou například velmi vítané vlastnosti chytrých telefonů: *notifikace* – oznámení, většinou formou krátké textové nebo multimediální zprávy, s účelem informovat o aktuálních událostech [5] – a *widgety* – elementy grafického rozhraní, v Androidu umísťované na domovské obrazovce (anglicky *Launcher*), které mohou zobrazovat různé informace a umožňují uživateli se systémem dále interagovat. [6]

Na rozdíl od klasického počítače, který mívá také systém notifikací, je výhodou chytrého telefonu fakt, že ho uživatel může mít stále u sebe aktivní. Aktuální oznámení se proto k uživateli mohou dostat rychleji. Oproti „hloupým“ telefonům pak zvětšuje spektrum dodávaných informací.

[3] K datu 5. prosince 2018

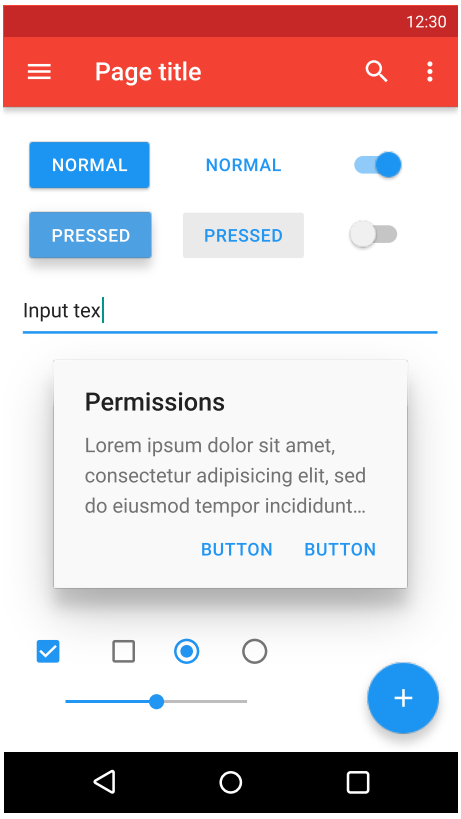
[4] Započítány jsou verze s podílem vyšším než jedno procento.

## 2.3 Material Design

Material design byl představen na konferenci Google I/O v červnu 2014.

Material design je v [7] definován jako *vizuální jazyk syntetizující klasické principy dobrého designu s inovací v oblastech technologie a vědy.* Ve světě Androidu je znám zejména svou sadou UI komponent (viz obrázek 2.1), kterými sjednocuje vzhled aplikací na platformě Android a umožňuje jejich intuitivnější a plynulejší ovládání.

Jiný zdroj [8] ho definuje jako Android-orientovaný designový jazyk, který podporuje dotykové ovládání pomocí vlastností a přirozených pohybů tak, že komponenty budí dojem objektů z reálného světa. Designéři tohoto cíle dosahují například 3D efekty, realistickým nasvícením (imitace stínů komponent) a animacemi konzistentně napříč celým GUI.




Obrázek 2.1: Ukázka UI komponent Material design.

Používat ho lze nativně ve verzích OS Android 5.0 nebo novějších. Pro podporu nižších verzí může vývojář použít tzv. *v7 appcombat* knihovnu (také referovanou jako *support library*), která dovoluje využívat komponenty Material design zpětně až do API verze 7 (Android 2.1).

Google jej rovněž používá ve většině svých službách na webu, například *Drive*, *Gmail*, *Maps* a dalších.

## 2.4 Android Jetpack

Na konferenci *Google I/O* v květnu 2018 představil Google knihovnu obsahující sadu komponent, nástrojů a průvodců pro snadnější a rychlejší tvorbu aplikací s názvem *Android Jetpack*. [9]

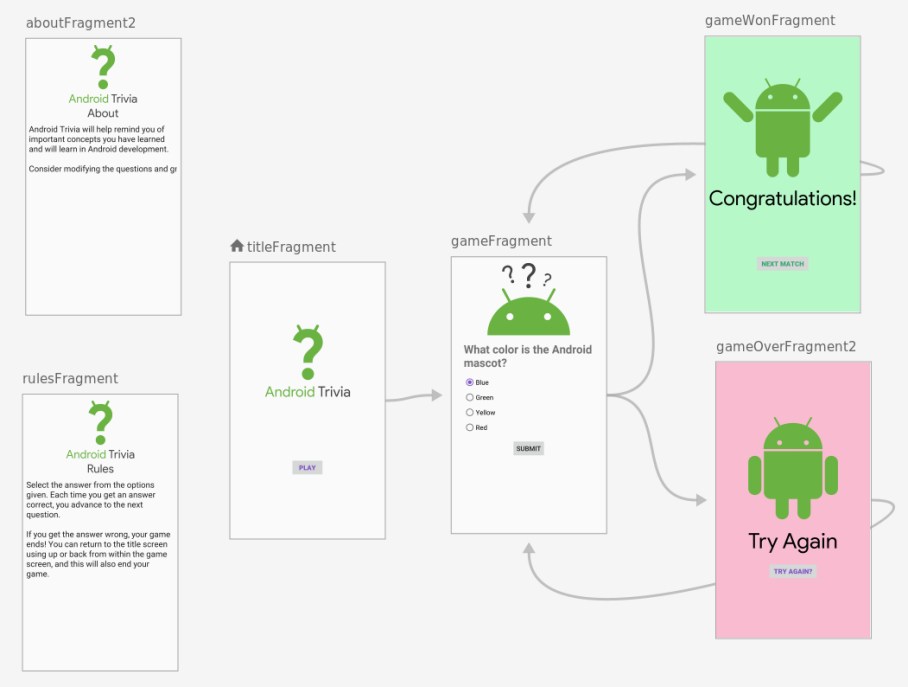
Knihovna používá jmenný prostor `androidx.*` a její obsah můžeme rozdělit do čtyř kategorií: *Foundation*, *Architecture*, *Behaviour* a *UI*.

*Foundation* komponenty pomáhají jednak s kompatibilitou pro používání nových funkcí i na starších zařízeních, ale také s testováním a lepší podporou jazyka Kotlin. Příkladem je například *AppCompat* – soubor tříd pro aktivity podporující Material design, *Action Bar*, standardní dialog pro sdílení dat atp. – *Android KTX* pro lepší práci s Android API v Kotlinu či *Test* přidávající podporu pro Unit a Android UI testy.

*Architecture* komponenty pomáhají s návrhem a implementací celkové architektury aplikace. Jako příklad lze uvést *DataBinding*, který umožňuje propojit datové struktury s UI komponentami, díky čemuž výrazně usnadňuje aktualizaci dat zobrazených pomocí UI. Jednou z nejnovějších přírůstků je pak *Navigation*, který se snaží řešit problémy s tvorbou navigace aplikace pomocí modelu jedné aktivity a fragmentů, jejichž interakce (přechody) a výměnu dat navíc umí zobrazit a konfigurovat pomocí grafu (pro ukázku zobrazení viz obrázek 2.2).

*Behavior* komponenty pomáhají se snadnější integrací Android služeb. Zmínit lze například *Notifications* pro integraci notifikací, *Permissions* pro správu práv pro aplikaci či *Download manager* pro stahování dat na pozadí.

Poslední částí jsou pak *UI* komponenty pomáhající s tvorbou UI, které ctí Material design a jsou pro uživatele snadno ovladatelné. Patří mezi ně například fragmenty (*Fragments*), *Animation & transitions* pro tvorbu animací a přechodu mezi obrazovkami či *Emoji* – zpětně kompatibilní font pro používání emoji.




Obrázek 2.2: Ukázka grafu komponenty *Navigation* knihovny Android Jetpack.

# 3 Webové služby

V této kapitole jsou v krátkosti popsány webové služby. Detailněji je pak popsán REST a podpora webových služeb na platformě Android.

## 3.1 Obecný popis

Definice webové služby se ve zdrojích různí. Jedna z obecnějších definic z [6] popisuje webovou službu jako aplikaci či zdroj dat, který je dostupný přes standardní webový protokol (*HTTP* či *HTTPS*) a narozdíl od webových aplikací jsou určeny pro komunikaci s jinými programy, spíše než pro přímou komunikaci s uživateli.

Ačkoliv může služba poskytovat data v různých formátech, nejčastěji používané bývají textové formáty *XML* a *JSON*.

Za nejpoužívanější protokol webových služeb je považován *SOAP*. Jeho zprávy jsou formátovány v XML a jsou posílány v HTTP[1]. Pro orientaci v tom, co webová služba nabízí, se používá *WSDL* soubor – ten popisuje jednotlivé koncové body, jejich vstupy a výstupy, čímž poskytuje formu „návodu“, jak danou webovou službu používat.

## 3.2 REST

REST je architektonický styl pro vývoj webových služeb. Jeho autorem je Roy Fielding.

REST obecně vzato pracuje se zdroji (anglicky *resources*). Pod těmi si lze představit nějaký objekt obsahující informace (obrázek, dokument, seznam strukturovaných článků atp.), který chceme zpřístupnit. Zdroje pak zpřístupníme pomocí koncových bodů (anglicky *endpoint*), což lze zjednodušeně popsat jako proceduru, jež daný zdroj zpřístupňuje a je dostupná pod nějakou dosažitelnou adresou – například URL `/api/document/42` je přístupem k dokumentu, který je identifikován číslem 42.

REST je založen na principu fungování protokolu HTTP. Ačkoliv není REST jakožto architektonický styl svázán s žádným protokolem, bavíme-li se o webo-

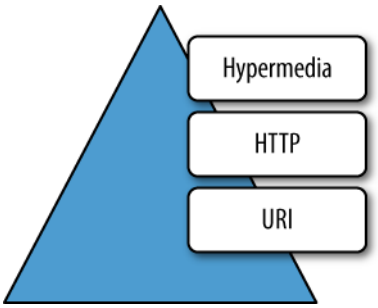
[1]Toto se týká standardních implementací – SOAP není vázán na HTTP protokol, je možné ho implementovat například nad FTP protokolem.

vých REST službách, komunikační protokol bývá téměř vždy HTTP. To s sebou nese podobné výhody – předávání stavových kódu při komunikaci, zabezpečení komunikace pomocí SSL či TLS, či nezávislost na jazyku, který REST implementuje – pokud jazyk umí pracovat s HTTP požadavky, umí pracovat i s RESTovou službou. Rovněž se však projevují i podobné nevýhody jako u HTTP protokolu, jako například bezestavovost – při komunikaci ve stylu požadavek – odpověď jsou při potřebě udržovat stav skrze několik požadavků kladeny vyšší nároky na klienta. Nebo také nemožnost serveru posílat klientovi jakoukoliv formou push notifikace ve chvíli změny stavu na serveru [32].

Jedním z problémů je také neurčitá definice toho, co znamená, že je služba „RESTová“, což je způsobeno faktem, že REST je „jen“ architektonický styl, bez referenční implementace či standardu, pomocí kterého by se dalo vyhodnotit, zda je design dané služby RESTový. Toto se snaží zlepšit například model Leonarda Richardsona.

### 3.2.1 Richardson Maturity Model

Richardson po analýze stovky webů implementujících webové služby rozdělil tyto weby do čtyř kategorií podle toho, jak moc jsou jejich služby „RESTové“ [10]. Jednoduchou ilustraci lze vidět na obrázku 3.1.




Obrázek 3.1: Ilustrace Richardsonova modelu, zdroj: restfulapi.net

- 0. úroveň – použití jednoho *URI* a jedné HTTP metody (typicky `POST`), například XML-RPC posílající tzv. POX – *Plain Old XML*,
- 1. úroveň – ke každému „typu“ zdroje je přistupováno přes vlastní URI,
- 2. úroveň – jako 1. úroveň, ale navíc jsou použity i další HTTP metody[2]

[2]Je jich celkem devět, standardně bývají používány: GET, POST, PUT, PATCH a DELETE.

(většinou pro aplikování principu CRUD[3]),

- 3. úroveň – jako 2. úroveň, ale navíc je ještě využito principu *HATEOAS* – Hypermedia As The Engine Of Application State – kdy klient dostane v odpovědi další URI pro související zdroje, čímž může prozkoumávat, co mu rozhraní nabízí

Cílem modelu je na první úrovni rozdělit jeden velký „monolit“ do menších celků podle hesla „Rozděl a panuj.“ Na druhé pak využít HTTP metody a tím sémanticky sloučit styl práce se zdroji. Na třetí úrovni se pak v rámci vráceného objektu vrací i další související možnosti práce se zdrojem[4], čímž vzniká prostor k tzv. objevitelnosti - lze pak například strojově generovat dokumentaci a popisovat vazby mezi různými URI a tím „sebedokumentovat“ danou webovou službu.

## 3.3 Podpora v OS Android

Android nemá standardně nativní podporu přímo pro jednotlivé webové služby. Pro používání webových služeb na OS Android máme dva způsoby.

První je využití nativních knihoven pro práci s HTTP a s pomocí nich si implementovat vlastní systém pro zpracování webových služeb. Android tyto knihovny nabízí dvě.

První je *Volley*. Mezi její výhody patří automatická správa fronty síťových požadavků, paralelní zpracování požadavků, podpora prioritizace požadavků, podpora API pro rušení požadavků, možnosti vlastních implementací jednotlivých částí zpracování požadavků (např. jakým způsobem při selhání opakovat poslání požadavku) a další. V základu umí zpracovávat některé druhy odpovědí, například JSON či obrázková data. Dle autorů [11] není vhodný pro zpracovávání dlouhých streamů (např. video streamů) či obecně stahování větších dat najednou.

Druhou možností je využít *Cronet* – síťový zásobník (network stack) webového jádra Chromium, kterého je možné využívat v Android aplikacích jakožto knihovnu – viz [12]. Již nyní se chlubí využitím v různých aplikacích od Google, jako například *YouTube*, *Google Photos*, *Maps* a dalších. Jeho zmiňovanými vlastnostmi jsou nativní podpora protokolů HTTP, HTTP/2

[3]CRUD – Create, Read, Update, Delete

[4]Například u zjištění informace o konkrétní knize bude URI pro její rezervaci v rámci systému.

a QUIC, prioritizace požadavků, kešování zdrojů, asynchronní požadavky, komprese dat a další.

Případně je také možné využít externí knihovny – jednou z nich může být například knihovna *OkHttp*. Ta se také chlubí podporou HTTP/2 a kešování zdrojů, dále pak podporou tzv. *connection pooling* (pokud nelze použít HTTP/2) či GZIP kompresí. Podporuje moderní TLS funkce (TLS 1.3, ALPN) s podporou *fallback*. Je možné ji použít v Android verze 5.0+ (API level 21+) či Java 8+. [13]

Druhý způsob je pak využít externí knihovnu pro zpracování požadavků dané WS. Pro zpracování RPC požadavků přes HTTP je možné využít framework *gRPC* od Google. Pro SOAP je možné využít například knihovnu *IceSoap*, využívající anotace s XPath cestami pro mapování XML obsahu na POJO. Pro REST je jednou z nejpoužívanějších knihoven *Retrofit*. Ten dle slov autorů „mění HTTP API na Java rozhraní“ – z definovaných rozhraní jsou pak přes builder vytvořeny třídy (tzv. *service*), jejichž metody je pak možné volat a obsahy jednotlivých odpovědí pak mapovat pomocí tzv. *converter* na POJO objekty (viz ukázka kódu 1).

```
public interface GitHubService {
@GET("users/{user}/repos")
Call<List<Repo>> listRepos(@Path("user") String user);
}
//------------------------------------------------------------
Retrofit retrofit = new Retrofit.Builder()
.baseUrl("https://api.github.com/")
.build();

GitHubService service = retrofit.create(GitHubService.class);
//------------------------------------------------------------
Call<List<Repo>> repos = service.listRepos("octocat");
```

Zdrojový kód 1: Ukázka práce s knihovnou Retrofit – vytvoření rozhraní, služby a její zavolání.

# 4 Kotlin

Kotlin je staticky typovaný multiplatformní programovací jazyk vytvořený pro běh nad JVM (*Java Virtual Machine*). V současné době je možné jazyk rovněž zkompilovat do JavaScriptu nebo nativního (strojového) kódu pomocí LLVM.

Jazyk vytvořila a v současné době dále vyvíjí společnost JetBrains. Společnost hledala alternativu jazyka Java, ve kterém měla napsané všechny své produkty. Protože nebylo proveditelné většinu jejich zdrojových kódů přepsat a jednou z jejich podmínek pro nový jazyk byla, aby byl jazyk staticky kompilovaný, splňoval jejich podmínky pouze jazyk Scala[1]. Ten jim však nevyhovoval zejména kvůli pomalé kompilaci a špatné podpoře v IDE. Nakonec tedy padlo rozhodnutí vytvořit vlastní jazyk, který splní požadavky, které nedokázala splnit Java, ale zůstane s ní plně interoperabilní – v jednom projektu bude možné používat jak kód Javy, tak i kód daného jazyka. [30]

První verze Kotlinu 1.0 byla vydána 15. února 2016. V březnu roku 2017 vyšla verze 1.1, hlavními novinkami byla podpora kompilace do JavaScriptu (do té doby byla pouze experimentální) a experimentální podpora korutin (anglicky coroutines), které lze použít pro vykonání neblokujícího asynchronního kódu. Na konci listopadu téhož roku vyšla verze 1.2, hlavní novinkou byla možnost multiplatformního projektu, kde by bylo možné v rámci projektu mít „modul“ pro společný (sdílený) kód a další moduly pro kód specifický pro danou platformu. Druhou významnou novinkou oproti předchozí verzi bylo urychlení doby kompilace. Posledním „velkým“ vydáním je verze 1.3, vydána na konci října 2018. Podpora korutin už v ní není experimentální, do beta verze se pak dostala podpora kompilace Kotlinu do nativního kódu pomocí LLVM.

Také je vhodné zmínit, že od února 2012 byl Kotlin uvolněn pod otevřenou licencí Apache 2. Rozšíření Kotlinu pomohlo také oznámení o oficiální podpoře pro vývoj aplikací na platformě Android společností Google na konferenci Google I/O v roce 2017.

Následující sekce popisují cílová prostředí pro běh jazyka, jeho vlastnosti a filozofii. Hlavním zdrojem je [29], napsaný dvěma vývojáři Kotlinu.

[1] Byl staticky typovaný a umožňoval využít stávající *codebase* v Javě.

## 4.1 Cílová prostředí

Hlavním cílem bylo vytvořit stručnější, produktivnější a bezpečnější alternativu Javy, která by byla použitelná v jakémkoliv kontextu, ve kterém je dnes používaná i Java.

Nejběžnější oblasti pro užití Kotlinu jsou:

- Serverový kód – například webový backend,
- Mobilní kód – aplikace běžící na OS Android

Zmíněny jsou však i jiné druhy použití. Například využití Intel Multi-OS Engine pro spuštění Kotlin kódu na zařízení s iOS či využití TornadoFX a JavaFX pro psaní aplikací pro desktop.

## 4.2 Hlavní vlastnosti jazyka

Kotlin je, stejně jako Java, staticky typovaný jazyk. Datový typ každého výrazu tedy musí být znám v čase kompilace a kompilátor má možnost ověřit správnost použití metod a atributů daného objektu. V čem se Kotlin liší od Javy[2] je možnost při deklaraci proměnné neuvádět její typ a nechat kompilátor typ detekovat podle toho, co vrací pravá strana výrazu, tzv. *type inference*.

Výhody staticky typovaného jazyka oproti dynamickému jsou:

- výkon – volání metod je rychlejší, neboť zjišťování, jaké metody je třeba zavolat, není prováděno až za běhu,
- spolehlivost – kompilátor může ověřit správnost programu, což snižuje možnost pádu programu za běhu,
- udržitelnost – lepší práce s „neznámým“ kódem, neboť je vidět, s jakými typy objektů se kde pracuje,
- podpora v IDE – staticky typované jazyky umožňují spolehlivější refactoring, lepší podporu psaní kódu (code completion) atd.

Jednou z nejdůležitějších částí je podpora tzv. *nullable types*. Kotlin rozlišuje mezi objekty, které mohou nabývat hodnoty `null` a těmi, které nemohou (viz zdrojový kód 2). Umožňuje detekovat místa, kde by mohla být vyvolána

[2] V době vzniku knihy ještě neexistovala verze Javy 10, kde bylo pro podobné účely zavedeno klíčové slovo `var`.

výjimka NPE – *Null pointer exception* již v době kompilace, což také pomáhá vytvářet spolehlivější programy.

```
var a: String = "abc"
a = null // chyba pri kompilaci

var b: String? = "abc"
b = null // ok
```

Zdrojový kód 2: Ukázka práce s nullable typy v Kotlinu.

## 4.3 Filozofie jazyka

Filozoficky byl Kotlin navržen jako jazyk pragmatický, stručný, bezpečný a interoperabilní.

Jeho pragmatičnost spočívá v tom, že byl vytvořen pro řešení existujících problémů. Jeho design byl navržen vývojáři s dlouhodobými zkušenostmi v oblasti programování IDE. Jazyk byl navíc před vydáním ostré verze testován a laděn vývojáři z JetBrains i komunitou. Nejde o akademický jazyk snažící se o vytvoření nového způsobu psaní a vývoje aplikací. Spoléhá se spíše na osvědčené metody, které se ukázaly být úspěšné v ostatních programovacích jazycích. Vývojáři navíc věří, že k vývoji kvalitních aplikací pomáhá i dobrá podpora v IDE, která umí poukázat na možná vylepšení či potenciální problémy v kódu.

Stručnost spočívá zejména v rozumné redukci tzv. *boilerplate* kódu – tedy kódu, který sám o sobě neplní žádnou funkční roli, ale jeho přítomnost je třeba pro správné fungování programu (například kvůli syntaxi jazyka). U Kotlinu je myšlenka taková, že čím méně je v kódu textu[3], tím jednodušší je pro cizího čtenáře[4] pochopit, jak kód funguje, a díky tomu se tak urychluje doba, kterou musí například jeden vývojář strávit opravou kódu, který napsal jiný vývojář. Samozřejmě, správné pochopení kódu není jen o jeho délce, je zde tedy také důraz na to, aby jména (klíčová slova, názvy tříd a metod atd.) co nejlépe vystihovala účel daného kusu kódu a bylo tak jasné, co vykonává.

Bezpečností se u jazyka myslí zejména návrh, který zabraňuje různým druhům chyb. Žádný jazyk neumožní programátorovi vyhnout se všem chybám,

[3]Autoři zmiňují, že i zde existuje rozumná mez, za jakou už není vhodné jít.
[4]Například programátora ze stejného oddělení.

zvláště když takovéto vyhýbání se chybám většinou něco stojí. Kompilátor musí získat více informací o záměru programu, aby mohl zkontrolovat, co program udělá. Jde tedy vždy o kompromis mezi bezpečností a mírou produktivity. V Kotlinu jde o snahu o zvýšení produktivity oproti Javě tak, aby bezpečnost zůstala na stejné úrovni nebo se mírně zvýšila. Dosahuje toho například již zmíněnou statickou typovou kontrolou a využíváním *type inference* – stále probíhá typová kontrola objektů, programátor se ale v mnoha případech nemusí „zdržovat“ psaním datového typu. Naopak bezpečnost zvyšuje dělením na null a non-null datové typy (viz ukázka zdrojového kódu 2 na straně 13).

Další pomoc spočívá ve snaze vyhýbat se výjimce `ClassCastException`, která bývá častá při přeskočení kontroly typu objektu. Na rozdíl od Javy se kontrola typu a přetypování dějí v Kotlinu v rámci jedné operace – viz ukázka zdrojového kódu 3.

```
if (value is String)                    // kontrola typu
    println(value.toUpperCase())        // value je automaticky
                                        // String
```

Zdrojový kód 3: Ukázka kontroly typu a přetypování v jedné operaci.

Posledním bodem je plná interoperabilita s Javou. Cílem bylo mít možnost v Kotlinu používat knihovny, API, anotace atp. napsané v Javě a naopak. To při prvním vydání Kotlinu znamenalo, že již na začátku byl k dispozici stejně velký knihovní „ekosystém“ jako v Javě. Správné fungování lze dokázat například tím, že Kotlin nemá vlastní knihovny pro práci s kolekcemi – místo toho plně spoléhá na standardní knihovny Javy, které vhodně doplňuje funkcemi pro jejich pohodlnější použití v Kotlinu.

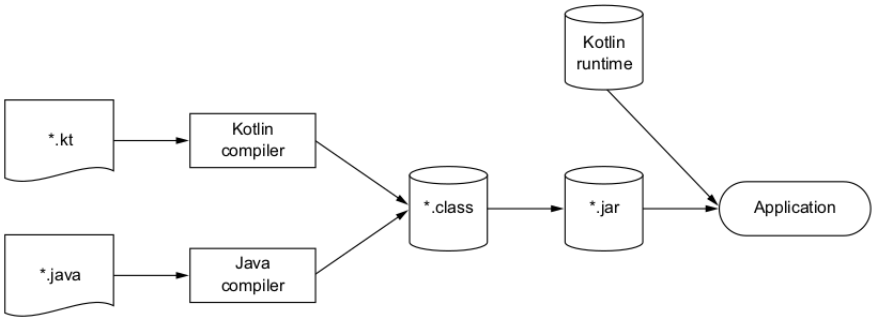
Dále mohou tímto vznikat i vícejazykové projekty – projekty, kde se nachází zdrojové soubory Javy i Kotlinu. Toto umožňuje zejména postupnou adopci Kotlinu – pokud se firma s produktem se zdrojovými kódy v Javě rozhodne používat Kotlin, nemusí do něj nejprve přepsat všechen svůj dosavadní kód. Může kód přepisovat postupně, popřípadě ho nemusí přepisovat vůbec, ale nové části mohou být psané v Kotlinu.

Jedna z věcí, která pomáhá s adaptací Kotlinu je přítomna v IDE *IntelliJ IDEA* – *Java-to-Kotlin* konvertor. Jde o automatický nástroj pro převod kódu v jazyce Java do kódu v jazyce Kotlin. Probíhat může dvěma způsoby – vložením Java kódu do `.kt` souboru, nebo kontextovou nabídkou u `.java` souboru – *Convert Java file to Kotlin file*. Tímto způsobem lze kód napsat

v Javě, nechat si ho přeložit do Kotlinu a tím se postupně učit jeho syntaxi a styl psaní kódu.

## 4.4 Sestavování

Proces sestavování je možné vidět na obrázku 4.1. Zdrojové soubory jsou ukládány v souborech s příponou `.kt`. Překladač (`kotlinc`) zdrojové soubory přeloží do souborů s příponou *.class*, které jsou pak zabaleny stejným procesem jako u Javy.




Obrázek 4.1: Zjednodušený pohled na sestavovací proces Kotlinu.

Kód přeložený v Kotlin kompilátoru je závislý na běhové knihovně Kotlinu (*Kotlin runtime library*), která obsahuje vlastní definice tříd a rozšíření dodané standardnímu Java API. Kotlin je kompatibilní se standardními sestavovacími nástroji – *Ant*, *Maven* i *Gradle*.

## 4.5 Frameworky

V Kotlinu je možné používat stejné knihovny a frameworky jako v Javě. Tato sekce však popisuje několik frameworků, které po vydání první verze Kotlinu začaly vznikat speciálně pro ulehčení programování právě v něm. Některé z nich jsou v této práci krátce popsány. Jeden z nejobsáhlejších seznamů popisujících knihovny, tutoriály a dokumentace a další zdroje lze nalézt v [14].

### 4.5.1 Koin

*Koin* je framework poskytující dependency injection. Je využitelný pro Android i pro JVM aplikace. Jeho výhodou je malá velikost a kompaktnost – snaží se co nejlépe poskytovat právě jednu funkci.

Na začátku se deklaruje modul v tzv. *Koin DSL* (*Domain specific language*), který deklaruje dostupné objekty (daných typů) pro vkládání. Poté ve startovacím místě (v Android ve třídě `Application`, v JVM aplikacích ve funkci `main` nebo jejích ekvivalentech) inicializuje Koin přes funkci `startKoin`. V tu chvíli je již možné ve třídách používat DI. Možné jsou oba dva způsoby – vkládání přes getter i přes konstruktor. [28]

### 4.5.2 Kodein

Dalším DI pro Kotlin je *Kodein* (*KOtlin DEpendency INjection*).

Jeho fungování se velmi podobá Koin, také pomocí svého DSL definuje dostupné objekty, v modulech vytváří jednotlivé instance, při startu aplikace je Kodein inicializován a poté se již v daných objektech může začít používat. [15]

### 4.5.3 Anko

*Anko* je knihovna od JetBrains určená pro snadnější vývoj Android aplikací. Dělí se na 4 části:

- *Anko Commons*
- *Anko Layouts*
- *Anko SQLite*
- *Anko Coroutines*

Anko Commons je sadou pomocných funkcí pro používání Android SDK. Pomáhá například s tvorbou intentů, dialogů, logováním, či se správou zdrojů.

Anko Layouts je DSL určený pro tvorbu dynamických layoutů pro aktivity či fragmenty.

Anko SQLite slouží k usnadnění práce s SQLite. Zejména zpřehledňuje práci s kurzory.

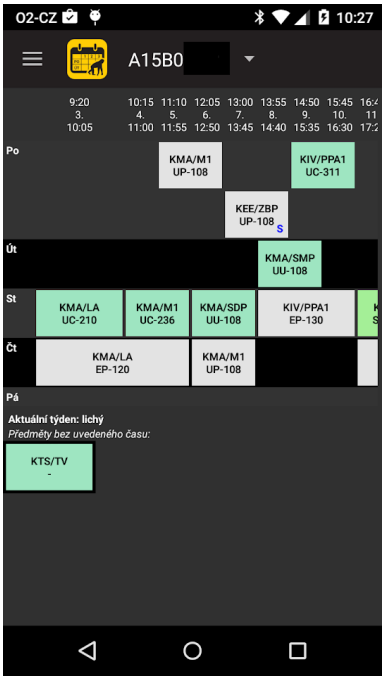
Anko Coroutines slouží pro usnadnění vykonávání práce na pozadí pomocí korutin v prostředí Android.

# 5 Existující aplikace

Tato kapitola popisuje některé z existujících aplikací pracujících s webovými službami IS/STAG.[1] Zdrojem pro tento seznam je především [16].

## 5.1 Primát.cz - rozvrh

Aplikace pro zobrazování rozvrhu a detailu jednotlivých předmětů pro studenty a učitele univerzit. Ukázku je možné vidět na obrázku 5.1. Vznikla v rámci diplomové práce Veroniky Dudové na Západočeské univerzitě v Plzni v roce 2012. [27]




Obrázek 5.1: Ukázka aplikace Primát.cz - rozvrh, zdroj: play.google.com

V současnosti kromě zmíněného zobrazování rozvrhu umí aplikace spravovat několik rozvrhů zároveň, podporuje gesta pro jejich přepínání či umožňuje zaměňovat rozvrhové akce. Podporováno je v tuto chvíli 9 univerzit[2].

Poslední aktualizace proběhla dle GP 19. ledna 2016.

[1] Všechny zmínky o aktuálnosti daných aplikací byly vyhodnocovány k datu 9. listopadu 2018.

[2] Některé nejsou podporovány v plném rozsahu – nabízí např. jen funkce pro učitele.

## 5.2 IS/STAG Evaluace

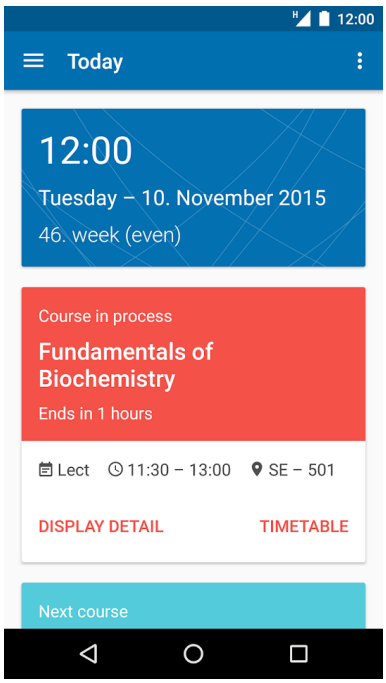
Aplikace pro zařízení s OS Android. Slouží pro hodnocení kvality výuky studenty prostřednictvím připravených anket předmětů. Byla vyvinuta ve frameworku pro mobilní aplikace `Apache Cordova`. [17]

Rovněž existuje varianta aplikace s názvem `IS/STAG Evaluace - demo`, kde si uživatel může hodnocení výuky vyzkoušet. Aplikace mu nabídne přihlášení pod fiktivním účtem, kde pak může vyplňovat připravenou anketu (rovněž fiktivní).

Poslední aktualizace proběhla 26. června 2017.

## 5.3 UPlikace

Aplikace vyvinutá Lukášem Novákem pro Univerzitu Palackého v Olomouci pro zařízení s OS Android. Je určená pro studenty a učitele zmíněné univerzity. Ukázku je možné vidět na obrázku 5.2.




Obrázek 5.2: Ukázka aplikace UPlikace, zdroj: play.google.com

Umí zobrazovat informace o předmětech, rozvrh, zkouškové termíny, probíhající akce, průběh studia, kvalifikační práce atd. Kromě zobrazování umožňuje zkouškové termíny zapisovat a odepisovat, umí uživatele notifikovat

ohledně nových událostí (nové zkouškové termíny, posudky kvalifikačních prací atp.) a rozvrhové akce umí navíc zobrazit i ve formě widgetu. [18]

Poslední aktualizace proběhla 24. září 2018.

## 5.4 UniApps

Multifunkční aplikace vyrobena stejnojmennou firmou UniApps s.r.o. pro Android, iOS a stále je ještě nabízena i na Windows Store.

Společnost na oficiálním webu [19] zmiňuje podporu 6 školských systémů (mezi nimi i IS/STAG). Nabízí velkou škálu funkcí závislých na oblasti, ve které je používána – od zobrazování rozvrhu a zkouškových termínů přes zobrazování jídelníčku menzy až po zobrazení autobusových spojení.

Poslední aktualizace verze pro Android[3] proběhla 8. srpna 2016. Oproti ostatním aplikacím má navíc poměrně nízké hodnocení – 3,0 na GP, 1,0 na App Store.

## 5.5 Studuj UTB

Aplikace vyvinutá studenty Univerzity Tomáše Bati ve Zlíně je určena pro uchazeče o bakalářské, magisterské a doktorské studium a je nabízena pro zařízení s OS Android, iOS a Windows Phone. [20]

Uživatelům umí zobrazit přehledy studijních programů, dnů otevřených dveří, seznamy kolejí a menz atd. Nabízí také náhledy univerzity pomocí fotografií a vizualizací.

Poslední aktualizace proběhla 2. září 2017.

Na GP se rovněž nachází aplikace *UTB* (s podobným logem v ikoně aplikace), kterou vytvořil Radek Vala. [21]

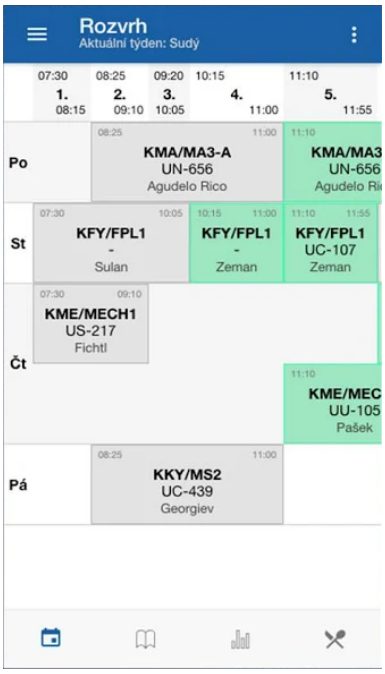
Ta nabízí možnost studentům zobrazit rozvrh, jídelníček menzy, novinky, kontaktní informace či informace o zkouškách.

Poslední aktualizace proběhla 2. října 2017

[3] Na GP pod názvem „UniApps v2 Beta“.

## 5.6 Student ZČU

Aplikace určená studentům ZČU. Ukázku je možné vidět na obrázku 5.3. V GP je jako autor uváděn *TommyLabs*, bohužel nikde v popisu aplikace nejsou uvedeny žádné další informace o jejím autorovi[4], ani žádné další zdroje informací (například webové stránky). [22]




Obrázek 5.3: Ukázka aplikace Student ZČU, zdroj: play.google.com

Aplikace nabízí studentům možnost nechat si zobrazit: rozvrh pro oba semestry, zkouškové termíny, studijní výsledky a jídelníček. Zkouškové termíny si lze navíc v aplikaci zapsat či odepsat.

Poslední aktualizace proběhla 25. ledna 2018.

## 5.7 Jídelníček ZČU

Aplikaci pro zařízení s OS Android vytvořil Martin Sloup. Aplikace nevyužívá WS IS/STAG, neboť informace o jídelníčku jednotlivých menz a bufetů zde nejsou uloženy. Nicméně protože některé ostatní aplikace mají ve výčtu funkcí zobrazování jídelníčků, byla aplikace do seznamu zařazena. [23]

[4]Autorem je Tomáš Balíček, působící na katedře KIV Fakulty aplikovaných věd, nicméně k této informaci jsem na stránce obchodu nikde nenašel žádný odkaz.

Kromě zmíněného zobrazení jídelníčků plzeňských menz a bufetů (které umí zobrazit i v jiný než aktuální den) umožňuje uživatelům jídla také hodnotit a jejich hvězdičkové hodnocení poté i zobrazovat. Na stránkách autora[5] je pak možné si kromě zmíněných jídelníčků zobrazovat i žebříček jídel seřazený podle hodnocení.

[5]https://menza.arcao.com/

# 6 Návrh aplikace

V této kapitole jsou probrány jednotlivé kroky návrhu aplikace a poté jsou popsány cíle aplikace – jaké má mít funkce.

## 6.1 Požadavky na aplikaci

Protože je cílem vytvoření oficiální IS/STAG aplikace, kterou mohou využívat uživatelé v několika uživatelských rolích a nasazení by v první fázi mělo proběhnout minimálně na ZČU, lze její požadované funkce kategorizovat podle zdroje a uživatelské role následujícím způsobem:

- WS IS/STAG
  - Studentské funkce
  - Učitelské funkce
- Funkce z jiných zdrojů

Role student a učitel jsou dvě nejpoužívanější role v systému, proto byly vybrány jako první pro implementaci rolí v aplikaci.

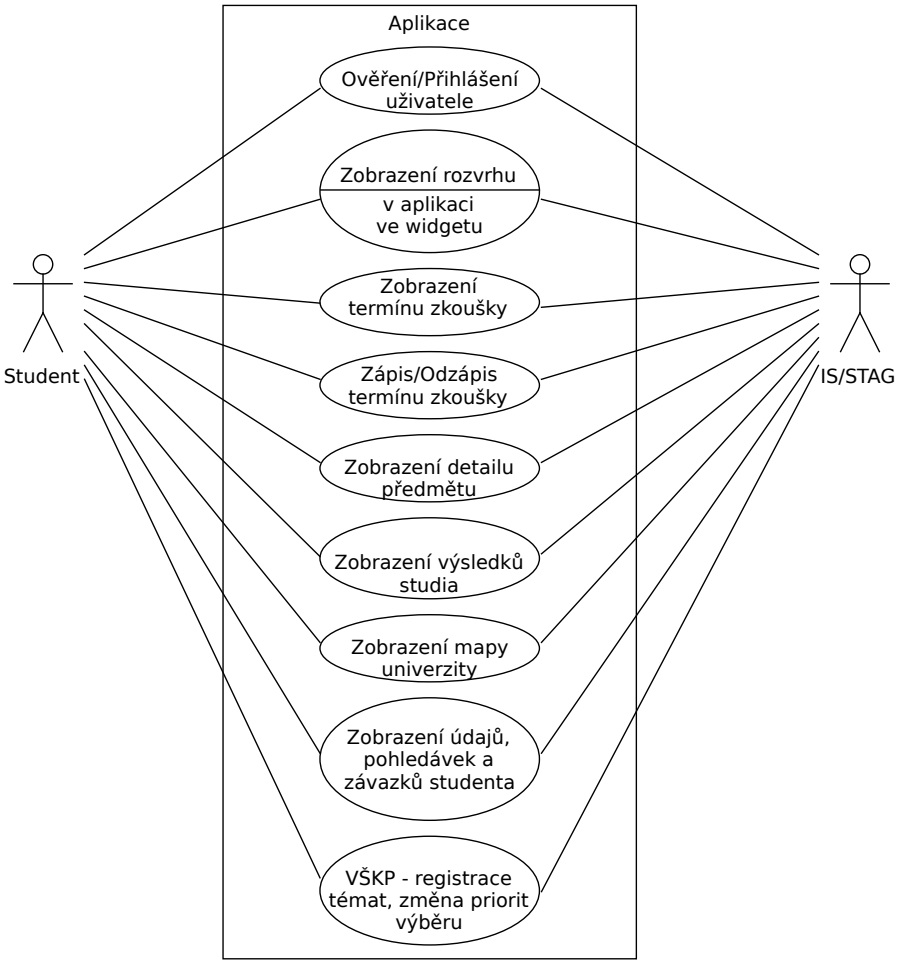
Dále budou blíže popsány funkce, u kterých bylo po počátečním průzkumu a konzultaci s analytikem systému IS/STAG uznáno za vhodné je implementovat v rámci mobilní aplikace.

### 6.1.1 Funkce z WS IS/STAG

Aby bylo jisté, že se uživatel dostane ke všem svým potřebným datům, bude aplikace vždy vyžadovat jeho přihlášení ve chvíli, kdy uživatel nebude autentifikován – jak funguje ověřování je popsáno v dokumentaci WS IS/STAG [24].

Funkce, které by aplikace měla studentovi poskytnout, můžeme vidět na obrázku 6.1 na straně 23. Základem by mělo být zobrazení rozvrhových akcí, dále zobrazení termínů zkoušek a možnosti jejich zápisu a odzápisu. Vhodné je také zobrazit výsledky studia, ať už ve formě aktuálně studovaných předmětů a jejich stavu (nesplněno, získaný zápočet, atd.), vizualizace stavu v rámci celého studia, či obojí. K tomu se váže možnost zobrazení detailů jednotlivých předmětů (popis předmětu, požadavky, vyučující atd.). Dále

by pak mohlo být užitečné zobrazení údajů studenta zadaných v IS/STAG (adresa, bankovní účet, stav žádosti o stipendia atd.) a zobrazení závazků a pohledávek pro potřebu kontroly. V orientaci by pak mohla studentům pomáhat také mapa se zobrazením univerzitních budov a jejich popis.




Obrázek 6.1: Diagram užití zobrazující interakci uživatele s rolí student s aplikací a IS/STAG.

Tato funkce by se v budoucnu mohla blíže integrovat s mapovou funkcí zmíněnou v aplikacích popsaných výše, díky čemuž bychom uživateli mohli poskytovat pokročilejší funkce, jako třeba s pomocí jeho GPS pozice naplánovat cestu na univerzitu tak, aby uživatel nezmeškal přednášku. Případně by mohlo být zajímavé k integraci podobné funkce využít jinou nainstalovanou aplikaci (*Mapy.cz*, *Google Maps* aj.).

Protože má rovněž velká skupina uživatelů k dispozici mobilní připojení

na jejich zařízeních, bylo by vhodné, aby aplikace umožňovala notifikovat studenta ohledně novinek (opravení semestrální práce, výpis nového termínu zkoušky atp.), čímž by získala velkou výhodu oproti použití webového portálu, který notifikace v tuto chvíli nemá, a tudíž musí student novinky aktivně kontrolovat.

Z výše uvedených funkcí bude pravděpodobně tou nejpoužívanější zobrazení rozvrhu (či nejbližších rozvrhových akcí). Ve chvíli, kdy je třeba znát jen název, místo a čas konání, bylo by vhodné mít možnost zobrazit rozvrh ve formě widgetu na hlavní obrazovku telefonu.

Protože se rozvrhy v průběhu roku příliš často nemění a ne každý má k dispozici připojení k internetu kontinuálně[1], bylo by vhodné data o rozvrhu uchovávat lokálně (offline). Pro správně fungující notifikace je třeba i datové podklady uchovávat lokálně.

Na obrázku 6.2 na straně 25 lze vidět, jaké funkce budou poskytovány uživatelům s rolí učitel. Stejně jako u role studenta může být učiteli nabídnuto zobrazení rozvrhu, termínu zkoušek, které daný učitel vypsal, a možnost vypsat nový termín zkoušky.

Dále bude učitel moci zaslat hromadnou zprávu skupině studentů (studující předmět učitele), což se může hodit třeba při uvíznutí v dopravní zácpě, kdy člověk většinou nemá po ruce počítač, ale mobil ano. Poslední funkcí, která se zdá být vhodná pro mobilní aplikaci, je zápis studentských známek do IS/STAG.

### 6.1.2 Funkce z jiných zdrojů

Tyto funkce bývají obecné (použitelné pro obě skupiny) a vázané na konkrétní oblasti (např. jen pro univerzitu ZČU), bude tedy pro ně třeba navrhnout dostatečně modulární architekturu.

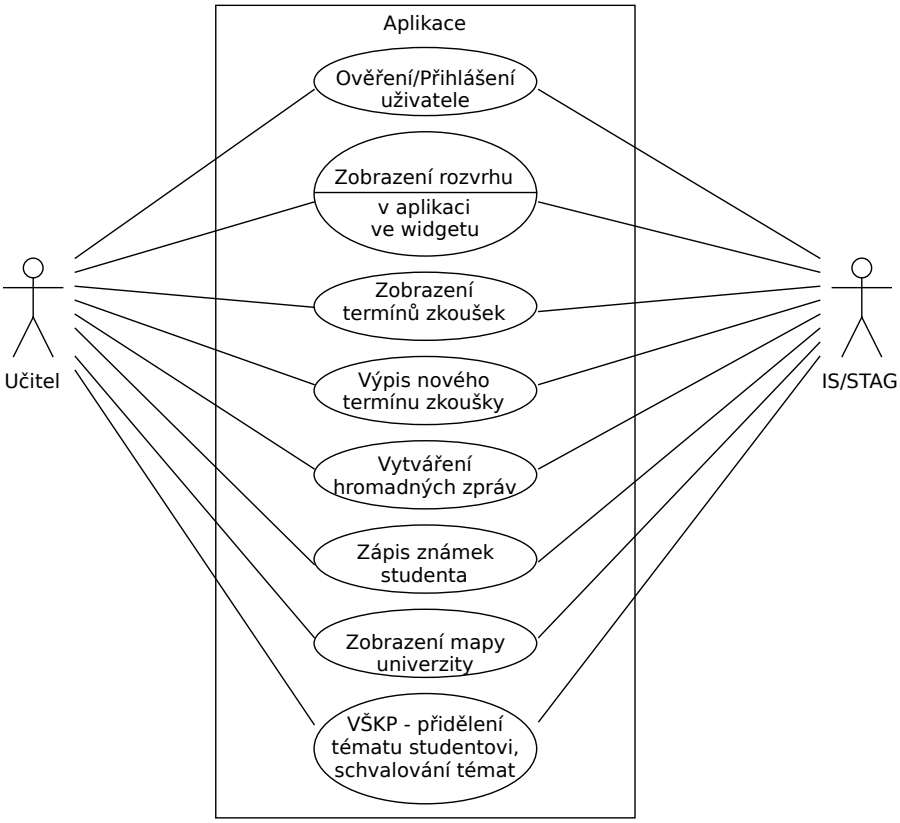
Protože je jedním z požadavků nasazení do provozu alespoň na ZČU, bude úsilí v této oblasti věnováno funkcím relevantním pro ZČU.

Navrhované funkce jsou zobrazené v obrázku 6.3 na straně 26. První funkcí bude zobrazování jídelníčku menz. Protože plzeňské menzy nenabízí možnost rezervace hlavních jídel, bude sloužit pouze k pasivnímu zobrazování.

Druhou funkcí pak bude zobrazení novinek popisující dění na univerzitě. Protože má ZČU „centrální systém“ pro novinky[2] na adrese `info.zcu.cz`,

[1] I s mobilními daty mohou nastat například krátkodobé výpadky signálu.

[2] A nově i pro pozvánky na různé akce.




Obrázek 6.2: Diagram užití zobrazující interakci uživatele s rolí učitel s aplikací a IS/STAG.

lze pro tento účel použít tento zdroj.

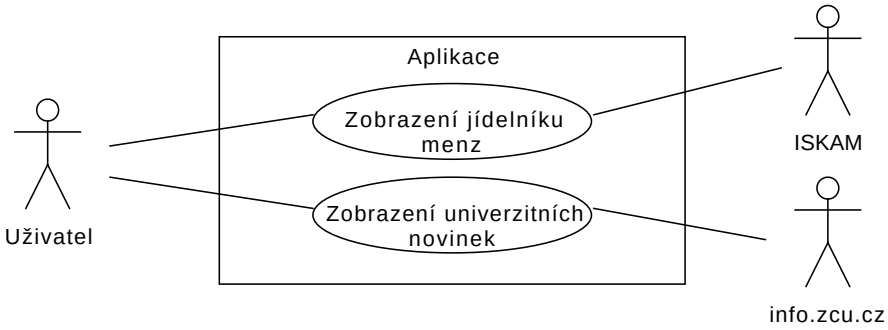
### 6.1.3 Určení priority implementace funkcí

V tabulce 6.1 jsou znázorněny hlavní (prioritní) funkce, které aplikace musí nabízet. Funkce, kterou daná role podporuje, je znázorněna znakem $x$.

V době vypracování požadavků na aplikaci chyběla podpora ze strany WS pro výpis údajů, pohledávek a závazků, zasílání hromadných zpráv a zadávání známek.

V tabulce 6.2 jsou pak vypsány funkce, které by bylo vhodné implementovat, nicméně v době návrhu funkcí není jisté, zda bude možné dané funkce implementovat.

Jako nejnáročnější se v odhadech jeví zasílání hromadných zpráv, neboť WS nemají zatím implementovaný žádný mechanismus pro práci s emaily.




Obrázek 6.3: Diagram užití zobrazující interakci uživatele s aplikací a okolními systémy – zdroji dat.

| **Funkce\Role** | **Student** | **Učitel** |
|---|---|---|
| Zobrazení rozvrhových akcí | x | x |
| Zobrazení zkoušek zapsaných/vypsaných | x | x |
| Zápis/Odzápis zkoušek | x | – |
| Zobrazení detailu předmětu | x | x |
| Zobrazení výsledků studia | x | – |
| Zobrazení údajů, pohledávek a závazků | x | – |
| Zadávání známek studentům | – | x |

Tabulka 6.1: Seznam hlavních funkcí a u kterých rolí budou nabízeny.

Ostatní se zdály být realizovatelné v krátkém časovém horizontu.

Navigace do budov bude předmětem dalšího zkoumání, závisí zejména na integraci s dalšími aplikacemi – navigacemi. U funkcí spojenými s VŠKP a vypisování termínů zkoušky šlo o chybějící WS a nejistotu ohledně odhadu časové složitosti jejich vytvoření.

Rovněž je vhodné zmínit, že při návrhu funkcí aplikace byl kladen důraz na jejich použitelnost na mobilním zařízení. Z toho důvodu je tedy například u funkcí VŠKP navrhováno implementovat registraci, prohlížení, přidělování témat a další, ale již ne vypisování nových témat – to už se jeví jako příliš komplikovaná záležitost[3] na to, aby byla realizována na mobilním zařízení. Dalším příkladem může být například zadávání známek. Zadat známku jednomu studentovi na chodbě na mobilu je mnohem lépe realizovatelné, než ji zadávat celé skupině na daném zkouškovém termínu – tam už se vyplatí využít klasický počítač, kde je navíc možné využít například předvyplnění

[3]Typicky je třeba napsat nemalé množství textu.

| **Funkce\Role** | **Student** | **Učitel** |
|---|---|---|
| Navigace do budov | x | x |
| Posílání hromadných zpráv | – | x |
| Výpis termínu zkoušky | – | x |

Tabulka 6.2: Seznam volitelných funkcí a u kterých rolí budou nabízeny.

některých polí, uložení všech známek v jednom kroku atp.

V sekci 9.1 na straně 52 jsou pak zmíněny možnosti přidání nových funkcí.

# 7 Popis implementace

V rámci práce byla mobilní aplikace vyvíjena pouze pro zařízení se systémem Android, neboť jde o nejrozšířenější mobilní OS, a tudíž je zde předpoklad nejširšího záběru, tj. že aplikaci bude moci využít největší počet studentů a učitelů.

Aplikace na platformě Android je možné oficiálně programovat ve třech programovacích jazycích: Java, C++ a Kotlin. Jazyk C++ je možné použít spolu s NDK – *Native Development Kit* – pro vývoj aplikací a knihoven, potřebujících využít všechen dostupný výkon zařízení, dosáhnout co nejmenších latencí či pro běh náročných aplikací, jako jsou hry či fyzikální simulace, nebo při používání již existujících C a C++ knihoven. V opačném případě doporučují vývoj v ostatních jazycích. Kotlin si na této platformě získává stále větší podporu[1] a slibuje lépe čitelný kód s dalšími výhodami slibujícími například bezpečnost (pro více informací viz sekce 4.2). Proto byl pro vývoj vybrán jazyk Kotlin a s ním použití Android SDK spolu se sadou knihoven Android Jetpack.

Před tvorbou je ještě důležité zvolit minimální podporované API (programátorsky uváděné jako *minSdkVersion*) – na jak starých zařízeních bude možné aplikaci provozovat. Volba je většinou závislá na podpoře použitých komponent, knihoven a funkcí a rozšíření dané verze OS Android. Zvolena byla podpora API 21 a vyšší (Android 5.0+), což na konci října 2018[2] znamenalo podporu cca 88,9 % zařízení.

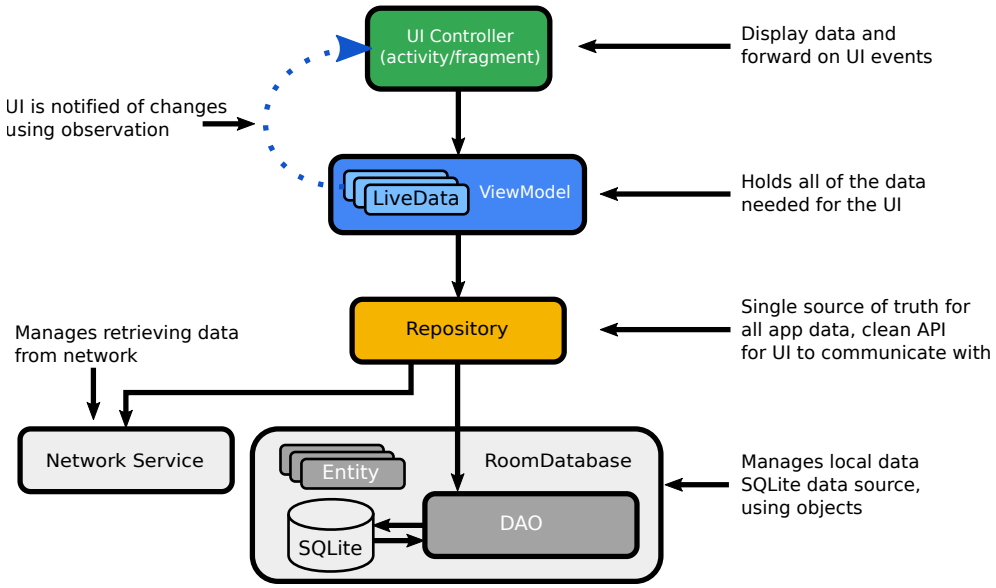
V kapitole bude dále popsána architektura realizované aplikace a podrobně rozebrány její součásti. Dále budou popsány finální stavy implementace jednotlivých funkcí a použité knihovny. Posledním bodem jsou pak možnosti vydání aplikace – její zpřístupnění uživatelům.

## 7.1 Architektura

Architekturu aplikace lze vidět na obrázku 7.1.

[1] Například většina návodů s příklady aplikací na codelabs od Google už bývá jen v Kotlinu.

[2] Google bohužel od té doby pravidelné měsíční statistiky neaktualizoval – procentuální podpora aplikace tak bude pravděpodobně ještě vyšší, než uváděná.




Obrázek 7.1: Upravený obrázek znázorňující komponenty architektury vytvořené aplikace, zdroj: codelabs.developers.google.com

Aplikace je „řízena“ přes uživatelské obrazovky (na obrázku 7.1 označeny jako *UI Controller*), skládající se z aktivit a fragmentů. Každý z nich může mít jeden či více *ViewModel* (dále VM) objektů, které mu poskytují data pro zobrazení či práci s nimi a které (až na výjimky) kopírují jejich životní cyklus.

Všechny VM objekty mívají jeden nebo více tzv. repozitářů, což jsou také objekty, které VM zprostředkovávají data pro předání aktivitám či fragmentům.

Repozitáře pak slouží k odstínění VM od zdroje dat. V aplikaci se využívají dva zdroje dat: lokální SQLite, sloužící jako cache, a online REST služby IS/STAG.

## 7.2 Uživatelské obrazovky

Aktivity a fragmenty umožňují aplikaci vhodně členit – většinou dle funkcí (například samostatná obrazovka pro zobrazování emailů a nastavení aplikace) či činností (jedna obrazovka pro seznam nepřečtených emailů, druhá pro vytvoření nového emailu atd.). Možné přístupy zahrnují použití pouze aktivit, pouze fragmentů (s jednou aktivitou hostující dané fragmenty[3])

[3]Jeden z hlavních rozdílů mezi aktivitou a fragmentem spočívá v tom, že fragment nemůže existovat sám o sobě.

a nebo kombinaci obou přístupů.

V aplikaci byla zvolena kombinace obou přístupů – jak samostatné aktivity, tak aktivita s fragmenty. Důvod bude blíže popsán v podsekci o navigaci.

### 7.2.1 Navigace

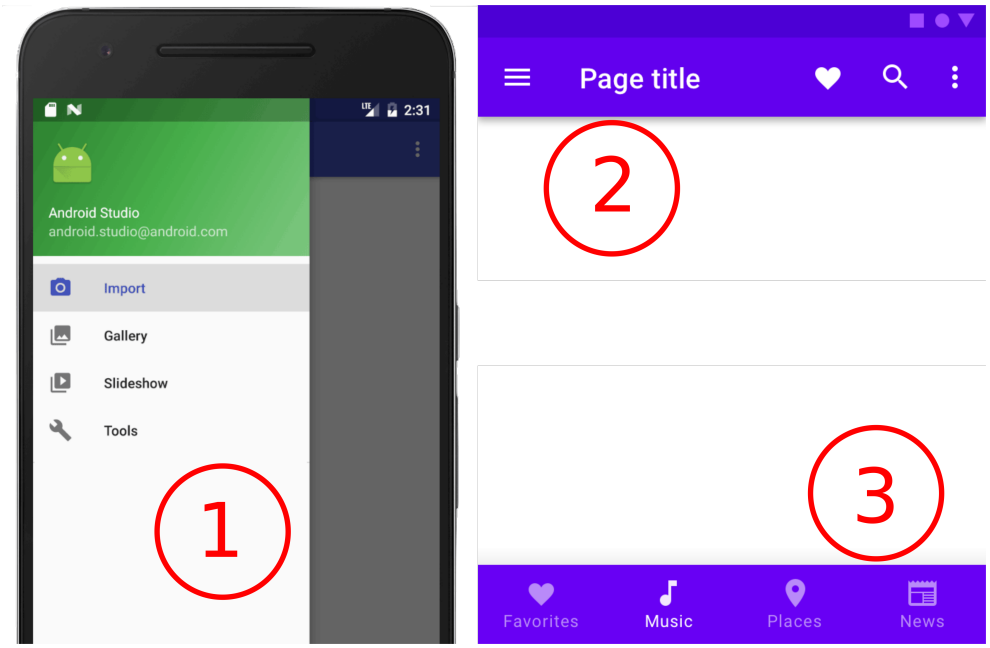
Navigace v aplikaci zahrnuje přesun mezi destinacemi – nezáleží na tom, jestli jde o aktivity, fragmenty či jiné komponenty. Pro navigaci v aplikaci byla použita Jetpack komponenta *Navigation*, jejíž výhodou je zejména jednotné API pro přechody mezi destinacemi. Navigation používá pro popis navigační struktury tzv. navigační graf, který je uložen v souboru formátu XML. V Android Studio lze graf tohoto souboru vizualizovat – jak lze vidět na obrázku 2.2 na straně 6, graf obsahuje jak *destinace* (fragmenty), tak přechody mezi fragmenty (znázorněné čarami zakončenými šipkami) nazývající se *akce*. Pro přechod mezi destinacemi není nutné definovat akce (proto jsou v grafu takové, ke kterým žádná šipka nevede), ty slouží spíše pro definici například přechodových animací či definování a kontrole parametrů, které si mezi sebou destinace vyměňují.

V rámci grafu je třeba u fragmentů definovat hostovskou aktivitu, ve které budou fragmenty zobrazeny[4]. Ta pak v rámci své struktury musí obsahovat grafickou komponentu `NavHostFragment`, ve které jsou pak zobrazovány jednotlivé fragmenty.

Protože Android nabízí několik způsobů, jak uživateli vizuálně umožnit navigaci mezi destinacemi, je součástí Navigation komponenty třída nazvaná `NavigationUI`, která se s pomocí inicializace v hostovské aktivitě umí postarat o aktualizaci obsahu těchto navigačních komponent (jejich ukázky je možné vidět na obrázku 7.2). Konkrétně se dokumentace zmiňuje o horní liště (*Top app bar*), u které je schopna měnit jak nadpis na levé straně (podle fragmentu či uživatelem definovaných pravidel), tak akce s tlačítky na pravé straně. Dále pak navigační panel (na telefonu skrytý vlevo s možností vysunutí na obrazovku, v originále se mu proto říká šuplík – *Navigation drawer*), u něhož kromě propojení navigace umí také automaticky spravovat zvýraznění zvolené destinace atp. Poslední podporovanou navigací je pak spodní navigace (*Bottom navigation*). [25]

V aplikaci byl pro navigaci využit navigační panel – spodní navigace nebyla

[4] V případě rozsáhlé aplikace s více hostovskými aktivitami se definuje více navigačních grafů.




Obrázek 7.2: Ukázka grafických navigačních komponent, 1 – Navigation drawer, 2 – Top app bar, 3 – Bottom navigation, zdroj: developers.google.com

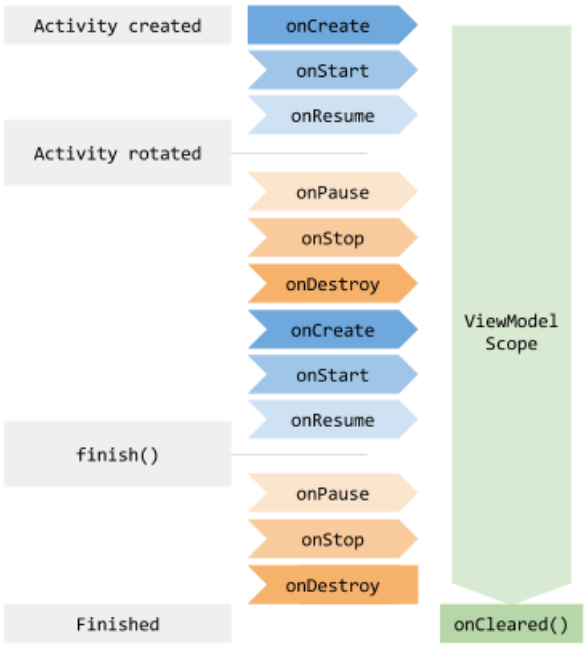
vhodná kvůli přílišné odlišnosti funkcí a jejich předpokládanému množství, navíc neumožňuje ani žádnou formu členění či kategorizace. Dále pak byla použita horní lišta – díky zobrazenému nadpisu fragmentu se uživatel může snadno informovat o tom, kde se zrovna nachází.

Dále je pak třeba určit domovskou obrazovku (*start destination*), která bude sloužit jako „hlavní vstup“ do aplikace, resp. do části aplikace spravované grafem. V navigaci aplikace je, jak již bylo zmíněno, využito v navigaci jak samostatných aktivit, tak fragmentů. Funkční obrazovky se nachází ve fragmentech, v samostatných aktivitách se pak nachází startovací aktivita – jejím účelem je při spuštění kontrolovat platnost autentizačního ticketu a při prvním spuštění nabídka výběru školy – a přihlašovací aktivita – jejím účelem je získání/znovuobnovení autentizačního ticketu. Fragmenty jsou v navigačním grafu zahrnuty, aktivity nikoliv. Zároveň je startovací aktivita nastavena jako vstupní bod celé aplikace. Vyjmutím aktivit ze zásobníku (*back stack*) aktivit po jejich opuštění a nezahrnutí do grafu způsobí, že při spuštění aplikace jimi uživatel projde (s platným ticketem uživatel tu přihlašovací přeskočí) a pak se dostane do hlavní aktivity s domovským fragmentem. Nicméně při stisknutí tlačítka pro návrat zpět se již uživatel nevrátí do zmíněných aktivit. Po návratu na domovský fragment (pokud se uživatel nacházel v jiných fragmentech) po dalším stisknutí tlačítka zpět uživatel aplikaci opustí.

Tím je tedy docíleno efektu, kdy jsou aktivity jen něčím „navíc“, čím uživatel v případě potřeby projde, ale jinak ho nijak dále neobtěžují a nezdržují. Celý navigační graf aplikace je požné vidět na obrázku D.1 v příloze D.

## 7.3 ViewModel

`ViewModel` třídy slouží k ukládání a zpracování dat pro zobrazení v UI s ohledem na životní cyklus UI komponent. VM třídy umožňují zachovat data při konfiguračních změnách, jako je například rotace displeje. Při těchto změnách totiž Android framework zpravidla danou UI komponentu zničí a znovu ji vytvoří. Tento proces lze vidět na obrázku 7.3. Hodí se tedy mít objekt, který těmito změnami zůstane nedotčen a může pak data předat nově vytvořené UI komponentě.




Obrázek 7.3: Ukázka životního cylku UI komponenty, aktivity, vlevo a ViewModelu vpravo, zdroj: developers.google.com

Druhý problém, který VM řeší, je potřeba častých asynchronních volání kvůli aktualizaci zobrazených údajů a uživatelem vyvolaných akcí. Ty je třeba nejen na pozadí vykonat, ale také po jejich vykonání uklidit pro zabránění potencionálních úniků paměti. To se mnohem lépe provádí u objektů, které nejsou ničeny a obnovovány procesem popsaným výše, a tudíž se nemusíme bát, že by se o proces nikdo nepostaral, ztratily se reference apod. Obecně

se tedy UI komponenty mají starat o zobrazování UI dat, reagování na uživatelem vyvolané akce a komunikaci s OS (například při žádosti o oprávnění schvalované uživatelem). Na VM je pak starost o všechno ostatní. [26]

V aplikaci se VM chovají přesně výše popsaným způsobem – starají se o vykonávání asynchronních volání pro získání dat a jejich aktualizaci. Třídy dědí od `AndroidViewModel`, díky čemuž jde o „lifecycle aware“ VM – jsou informovány o stavu životního cyklu UI komponenty.

Oproti popisu výše se v aplikaci VM starají ještě o kontrolu internetového připojení a případný výpis hlášky o nedostupnosti připojení uživateli na obrazovku. Mohlo by se zdát, že jde o mírné porušení principu výše z toho pohledu, že pro zjištění internetového připojení a výpis chybové hlášky je třeba získat aplikační kontext a referenci na kořenový grafický prvek UI komponenty – právě u něj by potenciálně vznikala potíž, pokud by v procesu byla UI komponenta zničena a znovu vytvořena. Není tomu nicméně tak, neboť VM si danou referenci drží jen po dobu vykonání funkce, a to jen té části vykonávané na hlavním UI vláknu. Druhý důvod, proč toto nevadí, je ten, že reference je potřebná jen na začátku funkce – pokud připojení není k dispozici, nevykoná se zbytek funkce (asynchronní kód se nezačne vykonávat vůbec), pokud je připojení k dispozici, začne se asynchronní kód vykonávat a reference již není používána. Výjimka slouží ke zpřehlednění kódu uvnitř UI komponenty (není zanesená kontrolou připojení internetu při každém spouštění asynchronní operace).

Pro zpracování asynchronních operací jsou použity korutiny dostupné přímo na úrovni jazyka Kotlin a vyvíjené společností JetBrains. Důvodem vzniku je řešení potřeby vykonávání asynchronních úloh na úrovni jazyka. Principem jde o abstraktní definování úloh, které jsou rozděleny na části, které jsou pak plánovačem (*dispatcher*) přidány do front vláken, na kterých jsou pak části vykonány, a výsledek je pak poskládán dohromady v garantovaném pořadí. Výhodou VM v tomto procesu je to, že dané úlohy lze při vytváření slučovat do „kategorií“ – tzv. *jobs*. V rámci VM si pak nadefinujeme instanci třídy `CoroutineScope` s parametrem našeho job a UI scope. Přes tuto instanci pak spouštíme všechny naše úlohy. To nám pak při implementaci metody `onCleared` rodičovské třídy zmíněné výše umožňuje zrušit všechny probíhající úlohy v případě, kdy je UI komponenta zničena a její VM bude zničen taktéž. Tím jsme tedy schopní po sobě „uklidit“ v případě, kdy výsledek asynchronních úloh již není potřeba, ale úlohy stále běží. Takto dojde k jejich korektnímu ukončení.

Protože se zobrazovaná data v UI komponentě mohou v průběhu času či

na základě uživatelských akcí měnit, je třeba definovat způsob, jakým budou vyměněna/aktualizována za novější. Ideální stav by byl takový, aby mohl existovat mechanismus, který po změně uložených dat automaticky vyvolá změnu v UI (notifikuje) a tím dojde k obměně zobrazených informací. Pro tento případ nabízí Jetpack třídu `LiveData`, která umožňuje obalit jakoukoliv instanci datové třídy tak, aby mohla být pozorována (*observable*) pozorovatelem, zda prochází nějakou změnou. Tímto mechanismem, který pak VM nabízí UI komponentě, si komponenta nastaví pozorování daného objektu (metodou `observe()`), v ní řekne, co se má v případě změny stát – například se aktualizují data v textových polích, objeví se notifikace atd. Z pohledu VM pak stačí při aktualizaci dat dát vědět UI, že se něco změnilo a metoda výše bude vykonána. Standardně si VM vytvoří instanci `MutableLiveData`, která dědí od `LiveData` a umožňuje modifikaci dat uvnitř, směrem k UI vystaví pouze možnost získání instance třídy `LiveData` (aby UI nemohlo s daty uvnitř manipulovat, jen je číst) a pak ve vhodné chvíli data změní a zavolá metodu `setValue()`, pokud chce data aktualizovat z hlavního vlákna, nebo `postValue()`, pokud chce data aktualizovat z jiného vlákna.

Tento mechanismus lze v některých případech ještě vylepšit, což je popsáno v sekci 7.5 na straně 35.

## 7.4 Repozitáře

Ve chvíli, kdy máme separovaná UI data od UI komponenty, potřebujeme vyřešit problém získávání těchto dat. Pro získávání dat ze STAGu bylo využito veřejně dostupných webových služeb, a to konkrétně jejich REST varianty. Pro získávání externích dat o jídlech nabízených v menze Bory na ZČU byla využita již předzpracovaná data ze systému ISKAM, dostupná také v podobě REST API na adrese hlavního webu ZČU. Více informací o těchto externích zdrojích bude poskytnuto v sekci 7.6 na straně 39.

Pro usnadnění získávání dat z REST WS byla použita knihovna Retrofit (ta byla zmíněna v sekci 3.3 na straně 9). Mohli bychom tedy při získávání dat pouštět jí vyrobené služby přímo z VM. To však naráží na dva problémy.

První z nich je problém oddělení zodpovědnosti (*separation of concerns*), kdy VM zastává příliš velkou část funkčnosti aplikace. Se vzrůstající složitostí při rozšiřování aplikace by se tak stala nepřehlednou a velmi těžko udržovatelnou částí. Navíc je VM vázán životním cyklem UI komponent, při jejich ukončení bychom tato data v procesu jejich získávání či po něm ztráceli.

To souvisí s druhým problémem: protože předpokládáme, že síťová dostupnost mobilního zařízení není vždy stoprocentní, je třeba myslet na to, abychom si data nějak lokálně ukládali. To nejenom ztěžuje situaci popsanou v bodu 1, kdy se VM přidává další zodpovědnost za další funkcionalitu, ale dokonce, jak je rovněž zmíněno v prvním problému, při ukončení životního cyklu UI komponenty můžeme data v procesu jejich získávání ztratit dříve, než je budeme schopni nějak uložit (o způsobu ukládání se zmiňuje sekce 7.5).

Proto je zde třeba další vrstva abstrakce – tzv. repozitáře (*repository*). Té VM deleguje proces se získáváním dat a jejich manipulací. Výsledkem jsou dvě věci. Za prvé je výsledkem čistější API z pohledu VM – pro získání dat či inicializaci jejich aktualizace prostě zavolá metodu a dostane případně výsledek. Za druhé se VM nestará o to, odkud data pocházejí – jestli z lokálního zdroje, nebo síťového zdroje.

V aplikaci se tedy repozitáře starají o inicializaci síťových služeb, získávání dat z lokálního úložiště, získávání dat ze síťových služeb a jejich ukládání do lokálního úložiště a nabízení dat VM. Jsou tedy takovými prostředníky mezi VM, lokálním úložištěm a síťovými službami.

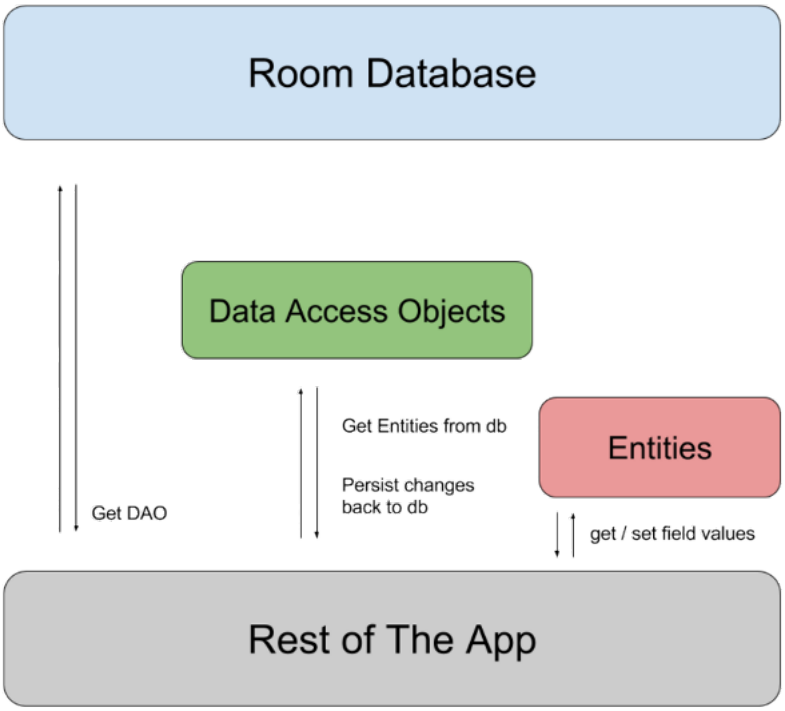
## 7.5 Úložiště dat

Kvůli výše zmiňované ne vždy dostačující síťové dostupnosti je třeba vymyslet systém pro lokální uchovávání našich dat. Pro ukládání lokálních dat dostupných jen naší aplikaci nabízí Android několik mechanismů. Bohužel, většina z nich se soustředí na ukládání primitivních dat na úrovni klíč – hodnota. To je pro naše potřeby nedostatečné, neboť ačkoliv data v našich objektech jsou většinou ve své podstatě primitivní datové typy, my potřebujeme ukládat celé kolekce objektů s těmito primitivními typy a potřebujeme alespoň omezenou schopnost práce s nimi – vyhledávat v nich, aktualizovat je, mazat všechny nebo vybrané záznamy atd. Z podporovaných způsobů se tedy jeví nejlépe použití *SQLite*, jediného oficiálně podporovaného databázového systému na platformě Android. Ten nám přesně tyto možnosti nabízí a jeho databázi lze skladovat jako soukromý soubor naší aplikace v úložišti telefonu. Neumožňuje sice pokročilejší funkce blízké jiným databázovým systémům (např. *MariaDB*), nicméně protože budeme úložiště využívat pouze jako cache, mělo by jít o řešení dostačující.

V aplikaci je nad SQLite použita abstrakční vrstva *Room*, která je rovněž součástí Jetpack a je oficiálně podporovaná vývojáři z Google. Jde ve své

podstatě o jednoduché ORM – Objektově Relační Mapování, které bylo využito ze dvou důvodů. Za prvé umožňuje výrazné zjednodušení práce s SQLite na úrovni kódu – stará se sám o inicializaci a správu databází a tabulek, korektně pracuje s kurzory při manipulaci či čtení dat z databáze a umožňuje funkčním způsobem namapovat jednotlivé instance tříd v kódu na databázové tabulky.

Pro své fungování potřebuje v podstatě tři funkční části: definici databáze, DAO (*Data Access Object*) a entit. Jednoduchý diagram architektury lze vidět na obrázku 7.4




Obrázek 7.4: Diagram Architektury Room, zdroj: developers.google.com

Databáze je třeba popsat anotací `Database`, je třeba, aby byla třída abstraktní, dědila od třídy `RoomDatabase` a obsahovala abstraktní metody s nulovým počtem argumentů, které vrací přidružené DAO instance. V rámci anotace je pak třeba uvést třídy (o těch bude více řečeno dále), které bude databáze spravovat (jejichž instance bude obsahovat). Za běhu je pak možné získat instanci databáze metodou `Room.databaseBuilder()` nebo pro tzv. in-memory databázi lze použít `Room.inMemoryDatabaseBuilder()`. Pak je databáze vytvořena pouze v paměti a není perzistentní – po ukončení procesu aplikace je databáze zničena. Hodí se tedy do testovacího prostředí. První volba je pak používaná v produkci.

Je také třeba zmínit, že vytváření vícero instancí objektů pro přístup k dané databázi je drahé a u drtivé většiny aplikací není potřeba. Je tedy lepší udržovat naživu jednu instanci a vhodně jí vracet všude, kde je potřeba.

DAO slouží k získávání entit z databáze a k ukládání změn do databáze. Standardně Room nepodporuje přístup k databázi v hlavním vlákně, neboť může jít o časově náročné úlohy, které mohou na nějakou dobu zablokovat UI, což se může projevit například reakcí od OS (například detekuje zaseknutí a začne nabízet ukončení aplikace). Pro definici DAO stačí rozhraní anotovat jako `Dao` a pak je do něj možné psát metody pro práci s daty. Metody jsou definovány taktéž anotacemi, pomocí kterých se určí, k čemu je daná funkce určena – `Query` potřebuje textový parametr pro vykonání obecného SQL příkazu, `Insert` slouží pro vkládání entit a má parametr pro strategii ohledně existujících záznamů, `Update` umí aktualizovat data objektu, `Delete` ho pak umí mazat. Dotazy u Query anotací jsou vyhodnocovány při překladu, chyba se tedy projeví už při něm, nikoliv až za běhu.

Za zmínku stojí také největší výhoda oproti jiným ORM řešením na platformě Android – Room umí pracovat s tzv. *observable queries*, tedy umí vracet objekty obalené třídou `LiveData` přímo z databáze. To má tu výhodu, že při změně dat, které Room umožňuje pozorovat, se změna automaticky projeví notifikací a není tedy třeba řešit jakoukoliv formu notifikací směrem k UI. Navíc při jejím vracení je kód automaticky provedený asynchronně, takže i prvotní volání po datech nepotřebuje žádná „ošetření“ týkající se volání metod z hlavního vlákna.

Poslední částí jsou pak entity – ve své podstatě POJO třídy anotované pomocí `Entity` obsahující parametry, které odpovídají jednotlivým sloupečkům v databázi. Třídy jako takové jsou pak tabulkami. Nutnou podmínkou je anotace jednoho z parametrů pomocí `PrimaryKey`, neboť v SQLite musí mít tabulka vždy alespoň jeden primární klíč. Standardně si ho skrytě vytvoří jako sloupec s názvem `rowid`, nicméně protože by ho pak ORM nemělo kam do objektu namapovat, Room alespoň jedno označení primárním klíčem vyžaduje. U entity je pak dále možné definovat vlastní název tabulky, cizí klíče či více primárních klíčů.

Room kvůli SQLite podporuje ukládání jen omezeného množství datových typů. Pokud tedy chceme uložit složitější datový typ, je třeba pro něj vytvořit tzv. typový konvertor (*type converter*), který se postará o převod ze složitého formátu do jednoduššího a zpět. Standardně SQLite podporuje datové typy označované jako `NULL`, `INTEGER`, `REAL`, `TEXT` a `BLOB`.

V aplikaci jsou použity tři databáze. První je „stagUserDb“, ve které jsou uloženy informace dvou objektů. Prvním jsou informace o přihlášeném uživateli, které WS IS/STAG pro přihlášení uživatele vrací po úspěšném přihlášení. Aplikace se z nich dozví, jakou roli uživatel má, jestli má přiřazený učitelský identifikátor, či zda je daná role uživatele aktivní. Druhou je přihlašovací ticket ve formě řetězce určeného jako hodnota parametru do HTTP hlavičky `Cookie` ve formátu `WSCOOKIE=token`. K tomu je v tabulce uložen čas a datum konce platnosti tokenu a zkratka školy s URL, na které jsou dostupné WS IS/STAG školy.

Druhou databází je „stagDb“, ve které jsou uloženy objekty získané z WS IS/STAG, používané pro zobrazování dat v UI. Díky roli databáze jakožto cache objektů je její „ERA“ model velmi jednoduchý – nemá žádné relace, jen tabulky objektů.

Poslední databází je „externalDb“, která podobným způsobem jako druhá skladuje data získané z externích zdrojů, tedy mimo IS/STAG. V současném stavu tedy uchovává data o aktuálně servírovaných jídlech v objektu menzy na Borech pro ZČU.

Strukturu všech tří databází je možné vidět v příloze E.

Při implementaci se ukázalo, že nastávají situace, kdy je třeba rozlišovat mezi dvěma druhy aktualizace dat – v aplikaci se pro ně používají výrazy *soft refresh* a *hard refresh*. Při vysokém počtu uživatelů používajících aplikaci najednou je totiž více než vhodné, aby si aplikace nestahovala aktuální data pokaždé, když uživatel navštíví obrazovku, která má daná data zobrazovat[5]. Zároveň je ale třeba data stále udržovat aktuální. Příkladem může být zobrazení rozvrhových akcí. Na začátku si je uživatel stáhne celé ze sítě, aktuální (tedy hard refresh). Rozvrhové akce ale bývají naplánovány většinou na začátku semestru a pak už se (až na výjimky) nemění. Není tedy třeba je neustále stahovat, nicméně je třeba je pravidelně udržovat – zejména odstraňováním akcí, které již proběhly (zde tedy soft refresh). Výsledkem implementovaným v aplikaci je tedy mechanismus, který si ukládá čas, kdy naposledy byl hard refresh proveden a je stanovený interval, po který by se další provádět neměl[6]. Při požadavku na automatickou aktualizaci dat – když si uživatel nechá zobrazit fragment s rozvrhovými akcemi – je nejprve zkontrolováno, jestli k aktualizaci v inkriminované době došlo. Pokud ano, je proveden pouze soft refresh. Pokud ne, nebo nebyl ještě nikdy proveden, jsou data stažena ze sítě a čas aktualizace zaznamenán pro

[5] I s ohledem na omezení množství přenášených dat.

[6] V aplikaci nastaven na 180 minut – tedy 3 hodiny.

budoucí kontrolu.

## 7.6 Síťové služby

Jak již bylo zmíněno, pro implementaci získávání dat z WS byla použita knihovna *Retrofit*. Ta umožňuje definovat rozhraní popisující parametry volání služby, jako je URL, nastavení hlavičkových parametrů (jako například `Cookie`) a další parametry posílaného dotazu (*query string*). Ty jsou většinou použity pro dodatečný popis parametrů hledaného zdroje (například při získávání seznamu knih by mohlo jít o filtr žánru, omezení pouze na knihy, které má obchod na skladě atp.).

V současné době existují dva možné způsoby vytváření těchto služeb: ruční a automatizované. U ručních si služby i doménové objekty vyrábí programátor sám. U automatizovaných se pro práci s REST službami používá framework se sadou nástrojů *Swagger*. Jeho editor umí generovat jak serverovou část kódu, tak klientskou za předpokladu, že je mu dodán vhodný deskriptivní soubor ve formátu vyhovujícímu *OpenAPI* specifikaci. Ten popisuje vše od dostupných URI a použitých HTTP metod až po parametry a jejich typ s možnými druhy odpovědí (JSON, XML atd.).

IS/STAG nabízí ke stažení soubory s WADL[7] a Open API specifikací. Protože však zatím není k dispozici žádný generátor pro Android v Kotlinu, ani generátor pro Retrofit, jsou služby a doménové objekty aplikace vytvářeny ručně. Výhodou tohoto přístupu je, že doménové objekty nemusí nutně obsahovat všechny atributy, které WS pošle. To ve výsledku zmenšuje databázi (kde se modelové objekty vytvořené z doménových ukládají) a při změně posílaných objektů ze strany IS/STAG pak potenciálně odpadá potřeba reagovat na změnu atributů, které si třída neukládá. Nevýhodou je potřeba ruční změny při změně atributů, které aplikace používá místo prostého vygenerování nových služeb a doménových objektů. Je však třeba zmínit, že v takovém případě je pravděpodobné, že bude ruční změna potřeba i tak (pokud by se změnou parametru změnila i funkčnost v aplikaci).

## 7.7 Funkce aplikace

V rámci úvodní aktivity byla implementována funkce pro výběr školy. V seznamu jsou všechny školy, které mají v rámci IS/STAG nasazený modul s webovými službami.

[7]REST ekvivalent popisného souboru pro SOAP – WSDL.

Po výběru školy přijde na řadu přihlašovací aktivita, která obsahuje UI komponentu *WebView* a jejím úkolem je umožnit uživateli přihlášení se jeho účtem v systému IS/STAG dané školy. Každá instance s nasazenými webovými službami má přihlašovací stránku na definovaném URL, kde se po jeho navštívení zobrazí možnosti přihlášení. Standardně je v nabídce přímé přihlašování do IS/STAG, některé školy mají však vlastní systém jednotného přihlašování (ZČU má například *WebAuth*), který je zde pak nabízen také.

Po přihlášení je uživatel přenesen do hlavní aktivity, jejíž obsah je zobrazován ve fragmentech. Prvním (startovacím) fragmentem je přehled (*dashboard*), na kterém jsou zobrazeny nejbližší události.

Při vysunutí levého menu má uživatel možnost vejít do obrazovky o aplikaci (*about*), ve které nalezne informaci o aktuální verzi aplikace, jméno autora včetně kontaktního emailu a poděkování různým autorům za možnost zdarma využívat rozličné ikony v aplikaci, díky kterým je navigace v ní příjemnější.

Dále má uživatel možnost se z aplikace odhlásit (*logout*), což vymaže všechny jeho záznamy a přesměruje ho na startovací obrazovku s výběrem školy.

Pro uživatele na ZČU je zde pak obrazovka s nabídkou podávaných jídel v univerzitní menze na Borech. Nabídka obsahuje výčet všech jídel, jejich ceny pro různé skupiny (studenti, zaměstnanci a externí) a popis jednotlivých alergenů. Jídla jsou zobrazována pro daný den, což je dáno zdrojem dat, který v současné chvíli nenabízí možnost volby data.

Tyto funkce by měly být viditelné každému, kdo má v rámci IS/STAG nějakou roli, se kterou se dokáže přihlásit do systému (s výjimkou jídelníčku, kde ještě záleží na konkrétní škole). Další funkce se potom objeví, pokud má uživatel roli studenta či vyučujícího.

První funkce určená pro studenty je v podstatě rozvinutá verze přehledu – rozvrh (*timetable*). Ten je zde vytvořen formou seznamu rozvrhových akcí, kde každá z nich má nějaký čas a datum konání, podle něhož jsou seřazeny (nejbližší akce na vrchu). Pokud jsou od sebe dvě akce vzdálené déle než 30 minut, je mezi akce vložen spoj, které o tomto informuje, čímž bloky předmětů vizuálně oddělí[8]. Druhým separátorem rozvrhových akcí je, pokud se dvě akce u sebe konají v jiný den. V tu chvíli je mezi ně vložen podobný separátor, který má na sobě text druhého dne, čímž od sebe předměty opět vizuálně odděluje, aby bylo viditelné, co se koná v jiný den.

[8]To se hodí, například když student hledá mezi předměty vhodnou pauzu na oběd.

Tento způsob zobrazení je na telefonu mnohem vhodnější než klasické tabulkové rozvrhové zobrazení, neboť tabulka vyžaduje hodně prostoru do šířky (většina uživatelů drží standardně telefon v portrétovém režimu, který moc široký není). Na vysokých školách navíc situaci komplikuje fakt, že si lze zapsat rozvrhové akce tak, že v jednu chvíli může probíhat více akcí zároveň. To tabulku může deformovat, zatímco v seznamu akcí jsou jen dané akce zobrazeny u sebe[9].

Druhou funkcí přístupnou studentům je pak seznam všech předmětů zapsaných v době studia a zobrazení jejich stavu. Pro vizuální odlišení používá zelenou barvu pro splnění a červenou pro nesplnění. Navíc zobrazuje splnění a nesplnění i v dílčích částech předmětů – zápočtu a (pokud jí předmět má) zkoušky.

Poslední funkcí je zobrazení zkouškových termínů. To ve své podstatě kopíruje zobrazení zkouškových termínů na portálu IS/STAG. Zobrazuje detaily o termínu – předmět, vyučující místnost, datum, obsazenost, maximální obsazenost atd. A potom se také umožňuje na zkoušku zapsat a odepsat se ze zkoušky. Ke klasické poznámce se navíc může přidat dodatečná popisná poznámka v případě, že by byl nějaký problém se zápisem nebo odzápisem. Typicky například pokud je již student zapsán na jiném termínu stejného předmětu či pokud už čas možný pro odzápis vypršel.

Učitelská funkce zahrnuje zobrazení seznamu rozvrhových akcí. Použit byl stejný adaptér jako v případě studentova rozvrhu, v seznamu jsou tedy rozvrhové akce s datem a časem konání, rozvrh pak umí zobrazit přednášky, cvičení a také zkouškové termíny, kterých se učitel účastní. Rozvrhové akce jsou pak vizuálně odlišeny barvou shodující se s barvami v IS/STAG (zelená pro cvičení, šedá pro přednášky atd.).

Ukázky obrazovek s popsanými funkcemi je možné vidět v příloze B.

## 7.8 Použité knihovny

Knihovna `Kotlin coroutines` od JetBrains ve verzi 1.3.21 byla použita pro vykonávání asynchronních úloh. Příkladem asynchronních úloh je například získání dat z databáze či z webové služby.

Pro implementaci ViewModelu, který se umí přizpůsobit životnímu cyklu UI komponent (*lifecycle aware*) byla použita knihovna `lifecycle-extensions`

[9] Je ovšem pravdou, že v seznamu je těžké jednoduše vizuálně naznačit, která z blízkých akcí již probíhá a která ještě ne, pokud by se jen částečně překrývaly.

pro Android ve verzi 2.0.0.

Pro implementaci lokálního úložiště byla použita databáze SQLite a nad ní ORM knihovna Android Room ve verzi 2.0.0.

Navigace v aplikaci funguje s pomocí knihovny `navigation-fragment` a pomocí dodatečné knihovny `navigation-ui` se pak navigace v aplikaci automaticky promítá (zvolené položky, animace atd.).

Všechny zobrazené seznamy v aplikaci používají `RecyclerView` – UI komponentu, která je nástupcem původního (a nyní zastaralého) `ListView`. Na rozdíl od něj umožňuje více způsobů zobrazení (při jiném než seznamovém zobrazení stačí dodat vhodný `LayoutManager`) a je šetrnější k paměti a procesoru znovupoužitím položek (*item reuse*), které uživatel již nevidí nebo byly ze seznamu odebrány. Netráví se tedy čas navíc uvolňováním paměti staré položky a alokací a inicializací nové. To umožňuje Android knihovna `recyclerview`, která byla použita ve verzi 1.0.0. Pro jednotlivé položky pak pro jejich grafické oddělení byla použita elevace pomocí obalení grafického rozložení (*layout*) položky pomocí `CardView`, které se nachází v knihovně `cardview`, taktéž použité ve verzi 1.0.0.

Většina grafických rozložení aplikace používá `ConstraintLayout` jako základ. Ten se od starších druhů layoutů, odlišuje zejména způsobem, jak definuje rozložení prvků na obrazovce. Tvůrci zmiňují jako důvod jeho vzniku snahu o zploštění (*flatten*) stromu komponent ve starších grafických rozloženích, kdy pro správné chování komponent, tj. jaké místo mají zabírat, co se má stát, když se jedna komponenta kvůli obsahu nebo jiné velkosti obrazovky zvětší či zmenší apod., musel strom se všemi komponentami výrazně růst (zejména do hloubky), neboť řešení spočívalo v umístění jednoho layout manageru do druhého layout manageru, který byl ve třetím atd. Při průchodu stromem kvůli hledání komponenty či při překreslování obrazovky pak mohlo toto procházení výrazně zpomalovat zařízení a zvyšovalo náročnost na HW (a tím snižovalo i výdrž na baterii). Nový layout se místo toho snaží definovat vztahy mezi potomky (kteří jsou ideálně všichni na stejné úrovni), pomocí kterých řídí svojí velikost a umístění, což by kromě popsaného problému mělo také lépe řešit například škálovatelnost na různých velikostech zařízení. Layout se nachází v knihovně `constraintlayout` ve verzi 1.1.3.

Pro získávání dat z webových služeb byla použita knihovna `Retrofit` ve verzi 2.5.0, pro automatickou deserializaci a serializaci z a do formátu JSON pak `converter-gson` a do textového formátu přímo `converter-scalars`,

což jsou konvertory zmíněné knihovny ve stejné verzi. Konvertor pro práci s JSON formátem používá ke svému fungování knihovnu GSON od Google.

Pro lepší práci s logy byla použita knihovna `Timber` ve verzi 4.7.1.

Protože Java 8 Time API je na Android dostupné až od API 26 (Android 8.0+), byla pro lepší práci s časy a daty použita knihovna *Joda-Time* ve verzi 2.10.

Pro vizuálně přitažlivější zobrazení rozvrhových akcí byla použita knihovna `MaterialTimelineView`, a to ve verzi 1.1.

Zjednodušit debugování aplikace umožňuje `Android Debug Database`, knihovna použitelná pro prohlížení, manipulaci a export databáze a jejího obsahu při spuštěné aplikaci v debug režimu. Použita byla verze 1.0.6.

Pro testování byl použit framework `kotlinTest`, který umožňuje testovat různé části aplikace podobně jako `JUnit`. Jeho výhodou je navržení syntaxe vhodné přímo pro Kotlin. Verze použitého framework byla 3.3.2.

Pro mockování byl použit testovací framework `Mockito` ve verzi 2.27.

Pro vytváření dokumentace „javadoc“ byla pro Kotlin použita knihovna *Dokka* ve verzi 0.9.18.

## 7.9 Distribuce aplikace

Pokud je aplikace připravena z hlediska funkcí a je otestována a stabilní, je dalším bodem její publikování – zpřístupnění aplikace uživatelům. Tento proces je na Android Developers rozdělen do dvou bodů, které pokrývají přípravu aplikace na vydání obecně (postup tedy není vázaný na Google Play, ačkoliv je GP v procesu několikrát zmíněn).

Prvním bodem je „příprava aplikace na vydání“. Ta zahrnuje 5 pod-bodů:

- Sběr materiálů a zdrojů
  - příprava kryptografických klíčů (Android vyžaduje, aby každá instalovatelná aplikace byla podepsána certifikátem vlastněným vývojářem),
  - vytvoření aplikační ikonky v souladu s pravidly pro tvorbu ikonek popsanými taktéž vývojáři Androidu,

  - v této fázi se také doporučuje vytvoření EULA,
  - vytvoření ostatních materiálů – například pro vydání na GP je třeba vymyslet popisek k aplikaci, pořídit ukázkové snímky obrazovky apod.
- Nastavení aplikace pro vydání
  - volba jména pro kořenový balíček kódu aplikace, který po vydání již nelze měnit[10] (např. `cz.zcu.fav.kiv.student`),
  - vypnutí debug symbolů a logů,
  - čištění složek projektu (nepoužité části zdrojových kódů, nepoužívané obrázky, knihovny atp.),
  - zkontrolovat nastavení v souboru manifestu – správné vyžádání práv (kontakty, zápis souborů atd.), nastavený název a ikonka aplikace, nastavené verze aplikace
  - pokud aplikace používá externí služby nebo servery, zkontrolovat jejich stav, nastavid do produkce
- Sestavení aplikace pro vydání
- Připravení externích zdrojů a serverů – důležité zejména v případech, kdy máme v aplikaci například možnost nákupu (*in-app purchase*)
- Finální test aplikace – test zda je aplikace skutečně připravená pro vydání

Druhým bodem je pak již samotné vydání. Aplikaci je možné k uživatelům dostat několika způsoby:

- Vydat v aplikačním obchodě – nejčastější varianta, dostane se k nejširšímu okruhu uživatelů.
- Vydat přes email – nejjednodušší varianta, pokud je email otevřen v aplikaci na Androidu, APK je rozpoznáno a je zobrazeno tlačítko pro instalaci[11].
- Vydat přes webovou stránku – klasické nabízení odkazu ke stažení instalačního APK souboru, který pak uživatel spustí a aplikaci nainstaluje (taktéž je třeba povolit instalaci z neznámých zdrojů).

[10] Přesněji – po jeho změně je aplikace systémem vnímána jako odlišná aplikace od té předchozí.

[11] To se však zobrazí pouze pokud má uživatel povolenou instalaci z neznámých zdrojů.

Protože je vydání aplikace na Google Play nejčastější, bude v krátkosti proces popsán v následující podsekci.

### 7.9.1 Vydání aplikace na Google Play

Pro vydávání aplikací na Google Play je třeba mít vývojářský účet pro přístup k *Google Play Console* – k vytvoření takového účtu je třeba zaplatit jednorázový poplatek ve výši 25 dolarů.

Poté už je možné přidat novou položku. Nejprve uvedeme název aplikace a výchozí jazyk. V dalším kroku jsme požádáni o poskutnutí krátkého popisu, který se zobrazí při rozkliknutí aplikace v obchodě (do 80 znaků). Dále pak úplný popis (do 4 000 znaků), hlavní ikonku ve vysokém rozlišení a ukázkové snímky obrazovky v aplikaci. Součástí obrázků je pak ještě obrázek pro „hlavní grafiku“, což je obrázek o velikosti 1024x500 px zobrazený ve spodní části stránky s popisem aplikace.

Dalším krokem je uvedení typu aplikace – nabízeny jsou pouze možnosti aplikace nebo hra – a kategorii (například vzdělávání, finance atd.).

Volitelnou možností je pak vyplnění kontaktních údajů specifických pro aplikaci – odkaz na webovou stránku, email a telefon. Pokud existuje pro danou aplikaci námi vytvořená stránka se zásadami ochrany soukromí, můžeme na ní rovněž zde uvést odkaz.

Protože je pro vydání aplikace třeba vytvořit podpisový klíč, kterým bude podepsán výsledný APK soubor, umožní nám Google Play jeho vytvoření v rámci tohoto formuláře – je ale možné klíč vytvořit i jinak a pak ho do formuláře (jeho veřejnou část) zadat.

Po nahrání APK souboru je pak ještě třeba projít dotazníkem ohledně povahy obsahu aplikace (tzv. *rating*) – typicky jestli je aplikace vhodná pro děti, pokud ano, od jakého věku atp. Rovněž se pak na jednotlivých trzích dle vyplněných hodnot aplikace blokuje, pokud to tamní právo vyžaduje. Příkladem je např. hodnocení PEGI pro Evropu a Střední východ či ESRB pro Ameriku.

Posledním bodem je volba typu vydání, možné jsou tři hodnoty:

- Alfa kanál – uzavřené testování, nainstalovat aplikaci mohou uživatelé pouze pokud jsou přes účet pozvaní,
- Beta kanál – otevřené testování, uživatel se může přihlásit sám a aplikaci si nainstalovat,

- Veřejný kanál – aplikaci si může nainstalovat kdokoliv, jehož zařízení splňuje minimální podmínky a jemuž v tom nebrání jiné okolnosti (např. právní)

# 8 Testování aplikace

Testování aplikací na této platformě lze rozdělit do dvou kategorií:

1. Jednotkové testování
2. Instrumentální testování
   (a) Lokální testování
   (b) Testování na virtuálním/fyzickém zařízení

První kategorií jsou klasické jednotkové testy. Ty běží na lokálním stroji, jsou tedy rychlé a testují části, které nepotřebují pro svou funkci nic z běhového prostředí Android. Pro potřeby těchto testů by měl být použitelný jakýkoliv testovací framework, který si rozumí s nástrojem Gradle. Pro testování aplikace tímto způsobem byl místo *JUnit* ve verzi 4.12, který je standardně přidán při vytváření nových Android projektů, použit framework *kotlinTest* ve verzi 3.32. Ten nabízí spousty možností testování Kotlin kódu. Protože však neřeší žádným způsobem mockování, byl ještě k němu použit již zmiňovaný framework *Mockito* ve verzi 2.27.

Druhou kategorií jsou tzv. instrumentální testy, což jsou testy, které v menší či větší míře závisí na běhovém prostředí OS Android. Protože se Android prostředí velmi špatně mockuje klasickými nástroji jako například zmíněné Mockito, musely dříve všechny tyto testy běžet na fyzickém či virtuálním zařízením s OS Android. Jde například o komponenty vyžadující data z HW senzorů, používající systémové služby (například pro zjišťování stavu připojení k síti) atd. I s použitím HW akcelerací byly výsledkem testy, které se dost často vykonávaly velmi pomalu.

Proto postupem času vznikly frameworky jako například *Robolectric*, které umí lépe simulovat podmínky Android prostředí, což zejména zpřehledňuje testy, které se nemusí zabývat správným mockováním prostředí.

Při vytváření aplikace nebyly lokální instrumentální testy použity. Naopak byla snaha o využití frameworku *Espresso* pro běh testů na virtuálním zařízení, nepodařilo se však správně mockovat všechny závislosti tak, aby test úspěšně proběhl.

## 8.1 Testovací zařízení

Většina testování správného fungování aplikace probíhala na dvou zařízeních. Krátký popis zařízení je možné vidět v tabulce 8.1.

| Funkce\Zařízení | Android | Rozlišení | Hustota displeje |
|---|---|---|---|
| Virtuální Nexus 5X | 8.1 (API 27) | 1080 x 1920 px | 423 PPI |
| Acer Liquid Z530 | 5.1 (API 22) | 720 x 1280 px | 294 PPI |
| Lenovo Vibe K5 | 5.1 (API 22) | 720 x 1280 px | 294 PPI |
| Sony Xperia XZ1 Compact | 9.0 (API 28) | 720 x 1280 px | 319 PPI |

Tabulka 8.1: Seznam testovacích zařízení.

Prvním bylo virtuální zařízení, nastavené tak, aby imitovalo reálné vlastnosti zařízení *LG Nexus 5X*. To mělo zastupovat moderní zařízení s poslední možnou verzí OS Android – verze 9.0 sice byla již na světě, v době vytváření virtuálního zařízení pro testy však ještě nebyl dostupný x86 obraz, který lze na vývojovém PC při běhu akcelerovat[1]. Navíc simulovaný telefon patří mezi mobilní telefony s vyšší hustotou displeje, lze na něm tedy testovat, jak se UI chová na zařízení s velkou hustotou pixelů.

Druhým zařízením, na kterém se hodně testovalo, byl reálný telefon *Acer Liquid Z530*. Ten zastupoval zařízení se starším podporovaným Android OS (5.1, což je druhý nejstarší aplikací podporovaný OS). A zároveň měl nižší hustotu displeje a hodil se tak jako reference pro test zobrazení UI na zařízeních s menší hustotou pixelů.

## 8.2 Jednotkové testy

Pro jednotkové testy byl použit již zmíněný framework *kotlinTest*. Otestovány byly prozatím jen modelové třídy, otestovala se správnost převodů mezi doménovými a modelovými objekty.

V rámci budoucích rozšíření by bylo vhodné testy rozšířit na další objekty, které je možné použít bez potřeby Android runtime.

[1] Větší plynulost při práci se zařízením výrazně urychluje testování implementovaných funkcí.

## 8.3 Uživatelské testy

Uživatelské testy byly provedeny formou testovacích scénářů, popsaných v následujících subsekcích.

### 8.3.1 Korektní přihlášení

Uživatel si vybere školu (testování probíhalo na serverech ZČU a ZČU Demo). Na přihlašovací stránce vyplní svoje přihlašovací údaje (na demo serveru přímo do formuláře, na produkčním do WebAuth formuláře).

**Očekávané chování**: Potvrzení úspěšného přihlášení a přesunutí do hlavní aktivity.

### 8.3.2 První přihlášení bez internetu

Uživatel spustí aplikaci bez přístupu k internetu.

**Očekávané chování**: Po zobrazení loga IS/STAG je uživateli zobrazeno modální okno s textem vysvětlujícím nutnost připojení k internetu při prvotní inicializaci. Po odkliknutí je aplikace ukončena.

### 8.3.3 Opětovné spuštění bez internetu

Uživatel spustí aplikaci bez přístupu k internetu. Aplikaci měl již alespoň jednou spuštěnou a aplikace má již uložená alespoň data o uživateli.

**Očekávané chování**: Zobrazí se modální okno s textem vysvětlujícím možnou neaktuálnost dat z důvodu chybějícího připojení k internetu. Po odkliknutí je uživatel přesunut do hlavní aktivity, aplikace je použitelná s lokálními daty, dokud není internet opět přístupný.

### 8.3.4 Opětovné přihlášení s obnoveným připojením

Po opětovném spuštění aplikace bez přístupu k internetu je obnoveno připojení k internetu a vyžádány aktuální data.

**Očekávané chování**: Aplikace před vysláním požadavku zkontroluje platnost tokenu, objeví prošlý token a přesměruje uživatele na přihlašovací obrazovku, aby token obnovil.

### 8.3.5 Výpadek sítě při přechodu mezi datovými obrazovkami bez uložených dat

Aplikace je spuštěna s internetovým připojením, před přechodem na jinou obrazovku, jejíž data nejsou uložena v lokálním úložišti, je připojení znepřístupněno.

**Očekávané chování**: Aplikace zobrazí informaci o tom, že nemá žádná data ke zobrazení a zároveň zprávu o tom, že aplikace nemá přístup k internetu.

### 8.3.6 Výpadek sítě při přechodu mezi datovými obrazovkami s uloženými daty

Aplikace je spuštěna s internetovým připojením, před přechodem na jinou obrazovku, jejíž data jsou uložena v lokálním úložišti, je připojení znepřístupněno.

**Očekávané chování**: Aplikace zobrazí uložená data a zároveň zprávu o tom, že nemá přístup k internetu.

### 8.3.7 Ověření správnosti zobrazených dat

Uživatel navštíví jednotlivé datové obrazovky a zároveň (buď na stejném zařízení, nebo jiném – například PC) navštíví web portálu ZČU se stránkou, která nabízí ekvivalentní informace k datové obrazovce.

**Očekávané chování**: Protože většina webových služeb nabízí datový ekvivalent portálových stránek, lze jejich porovnáním ověřit, že aplikace obdržela stejná data, jaká může uživatel najít v portálu. Data mohou být odlišná v detailech (jeden parametr je v aplikaci skrytý a na webu ne či obráceně), ale např. rozvrhové akce, studované předměty či zkouškové termíny jako takové by měly být identické.

### 8.3.8 Soft refresh při opakovaném zobrazení dat

Uživatel navštíví jednu z datových obrazovek (například rozvrhové akce), po stáhnutí dat se vrátí zpět do obrazovky přehledu a znovu navštíví stejnou datovou obrazovku.

**Očekávané chování**: Při druhém zobrazení nebudou stažena žádná data ze sítě, uložená data mohou být upravena (např. vymazání proběhlých rozvr-

hových akcí). Protože informace o druhu obnovy dat není nikde vypisována[2] (v běžném použití o tomto není vhodné uživatele informovat jinak), je toto testování prováděno se zařízením připojeným k počítači. V Android Studio je pak možné informaci zjistit z logovacích výpisů *logcat*.

## 8.4 Výsledky testů

Testovanými scénáři aplikace prošla uspokojivě a konzistentně. Samotné instance tříd pro síťové služby mají v sobě rovněž tzv. `Interceptor`, který kontroluje připojení k síti i při vykonávání požadavku. Protože však lze výpadek v této části velmi těžko nasimulovat, nebyl pro tento případ vytvořen žádný testovací scénář.

Testování probíhalo na mém účtu v produkčním serveru IS/STAG pro ZČU, dále pak na náhodně vytvořených účtech studentů a učitelů v demo instanci ZČU – Demo.

### 8.4.1 Profilování aplikace

Zároveň s testováním na virtuálním zařízení byl použit profiler integrovaný v IDE. Při testu byl použit můj studentský účet na produkčním serveru IS/STAG pro ZČU.

Dle naměřených výsledků zabírala aplikace průměrně 3,8 MB RAM zařízení, vytížení CPU bylo mizivé, v řádu jednotek procent zátěže. Android OS u aplikace hlásil po průchodu všemi dostupnými datovými obrazovkami cca 8 MB dat uložených aplikací na úložišti zařízení, profiler vypisoval u celkového využití sítě cca 500 KB stažených dat. Poslední údaj bude pravděpodobně vyšší při běžném provozu, neboť testování na vlastním účtu probíhalo ke konci semestru, kdy už rozvrhových akcí příliš mnoho nezbývá.

[2] Aktualizace je prováděna relativně často, takže zobrazování informace by mohlo být obtěžující. Lepší je informovat uživatele pouze rámcově o tom, že existují dva druhy aktualizace dat.

# 9 Popis možných rozšíření

Aplikace je v současné podobě stabilní a funkční. Díky jejímu cílení a zároveň i velikosti systému IS/STAG existuje velké množství cest, kterými se lze vydat v rámci možných zlepšení aplikace do budoucna.

## 9.1 Nové funkce

Architektura aplikace umožňuje do budoucna přidávat nové funkce. Učitelská část v tuto chvíli nabízí funkcionality málo – plán na vytvoření WS pro rozesílání emailů studentům předmětu daného učitele byl nakonec zamítnut s tím, že implementace mechanismu pro posílání emailů v modulu webových služeb IS/STAG je na takto specifickou záležitost přehnaný[1]. V plánu je nabídnutí uživateli možnost (pomoc), aby mohl email poslat studentům přímo ze zařízení za pomocí externí emailové aplikace (GMail či jiné), což je postup, na který jsou mobilní uživatelé zvyklí[2]. Dalším bodem, který se nakonec nerealizoval, byla funkce pro zadávání známek jednotlivcům učitelem a vypisování jednoduchých termínů zkoušek na jeden předmět. První bod má již oporu na straně IS/STAG (služby jsou připravené), těžší část úkolu spočívá ve vytvoření dostatečně kvalitního a přehledného návrhu obrazovek pro zadání známek, zejména s ohledem na ovladatelnost. Druhý bod byl pozdržen, neboť na straně IS/STAG nebylo dlouhou dobu možné věnovat přípravě služby dostatečnou pozornost z časových důvodů. Nicméně je stále v plánu v dalších verzích aplikace.

Funkce ohledně VŠKP byly poměrně brzy vypuštěny, neboť se po analýze zjistilo, že pro jejich možnou implementaci, jakožto webové služby, by bylo třeba netriviálně pozměnit portálový modul přímo v IS/STAG, čemuž v současné době nemůže být dána priorita. Tyto funkce tedy v dohledné době nebude možné implementovat.

Novou věcí, která by povýšila mobilní aplikaci nad webovým portálem, jsou notifikace. Ty byly původně v plánu, nicméně po zjištění, že je v plánu příprava systému notifikací přímo v IS/STAG, který by byl v podstatě takovým „notifikačním centrem“, byla implementace odložena, aby se jedna

[1] Myšleno náročný časově i prací ve vztahu k výsledku

[2] Delegace úkolů aplikacím k tomu určeným – emaily mailovým aplikacím, zobrazení adresy mapovým aplikacím apod.

funkcionalita nemusela implementovat dvakrát.

V blízké budoucnosti je také v plánu příprava widgetu na plochu zařízení zobrazujícího nejbližší rozvrhové akce. Fungování v aplikaci je na dobré úrovni, a protože jde o často kontrolovanou záležitost, bylo by vhodné umožnit uživatelům tuto kontrolu vykonávat i bez spuštění aplikace.

Jak se bude aplikace rozrůstat, vzroste pravděpodobně potřeba ukládat preference či jiná nastavení ve formě klíč – hodnota. Bude tedy třeba vytvořit další fragment pro uživatelská nastavení či preference (příklad: cenová kategorie v menze, tmavý či světlý vzhled atd.).

Aplikaci by také bylo možné rozšířit o další jazykové lokalizace k současné češtině a angličtině. Díky struktuře projektu jde o jednoduchý úkon – vytvořit další jazykovou verzi souboru `strings.xml`, ve kterém jsou uloženy všechny lokalizační řetězce. Služby IS/STAG jsou na toto rovněž připraveny, ačkoliv jim většina textů (kromě zmíněné češtiny s angličtinou) chybí a mají tedy často tendenci udělat fallback na anglické a nebo české texty.

## 9.2 Vylepšení stávajících funkcí

V rámci vylepšení stávajících funkcí by mohlo být užitečné přidat do fragmentu s rozvrhovými akcemi možnost přepínače, který by umožňoval zobrazovat i akce s nenaplánovaným datem či časem. V tuto chvíli jsou nenaplánované akce ignorovány a nejsou nikde zobrazeny.

Podobné vylepšení by mohlo proběhnout ve fragmentu se seznamem všech zapsaných studijních předmětů. Student by mohl mít možnost zobrazit nejen všechny předměty, ale třeba jenom předměty zapsané v jednom akademickém roce, či pouze v jednom semestru. Úprava by se dala provést dvěma formulářovými prvky a definováním reakce na jejich změnu.

Poslední navrhovanou úpravou je použití grafické knihovny s použitím sady ikonek. V současné podobě jsou ikonky sehnány z různých zdrojů (respektive od různých autorů) a z různých kolekcí, což znamená, že nemusí působit vizuálně jednotně. Proto by možná bylo vhodné použít nějakou grafickou sadu ikonek, která by jednotně působila, což by dále vylepšilo uživatelský komfort.

## 9.3 Technická vylepšení

Do budoucna bude určitě vhodné sledovat postup vývoje v rozšířenosti jednotlivých verzí Android na mobilních zařízeních. Android knihovny se vyvíjejí poměrně rychlým tempem a implementace nových technologií, resp. ukončování podpory starých verzí OS jsou klíčové pro to, aby aplikace zůstala moderní a uživatelsky přívětivá.

Druhou technickou poznámkou je možnost využít nějaký DI framework pro inicializaci jednotlivých komponent. Velmi dobře se jeví *Dagger 2*, framework vyvinutý Google, který je plně statický a generuje kód při překladu. Měl by tedy mít zanedbatelné nároky při provozu a zároveň by zpřehlednil kód.

# 10 Závěr

Cílem diplomové práce bylo navrhnout a vytvořit mobilní aplikaci pro OS Android napojenou na IS/STAG. V rámci práce byla zkoumána problematika moderního vývoje aplikací pro operační systém Android. Také byly prozkoumány některé dostupné knihovny, frameworky a nástroje pro usnadnění vývoje. Dále práce obsahuje průzkum již existujících aplikací, které rozebírá zejména z hlediska nabízených funkcí a podpory aplikace jejími autory.

Výsledkem praktické části je pak aplikace umožňující studentům a učitelům interakci se systémem IS/STAG. V textu práce je v návrhu podrobně popsán zejména proces výběru funkcí pro tvořenou aplikaci a jejich členění. V implementaci je pak velmi podrobně popsána architektura systému, její vrstvy, použité knihovny a frameworky a finální podoba implementovaných funkcí.

Aplikace byla testována jak automatickými, tak uživatelskými testy. Důraz byl dán zejména na otestování správného fungování aplikace bez internetového připojení a také s měnícími se podmínkami síťového připojení.

Protože je IS/STAG velmi rozsáhlým systémem, nabízí se mnoho cest, jak výslednou aplikaci dále vylepšovat. Proto jsou v samostatné kapitole popsány způsoby, jak lze aplikaci vylepšovat přidáváním nových funkcí i vylepšováním stávajících.

Aplikace bude v budoucnu vydána v aplikačním obchodu Google Play, v současné době je nabízena ke stažení v podobě instalačního APK souboru (viz příloha A).

# Seznam použitých zkratek

**ALPN** Application-Layer Protocol Negotiation

**API** Application Programming Interface

**APK** Android Package

**CC** Creative Commons

**DAO** Data Access Object

**DI** Dependency Injection

**DSL** Domain Specific Language

**GP** Google Play

**GZIP** GNU zip, kompresní formát

**HTTP** Hypertext Transfer Protocol

**HTTPS** Hypertext Transfer Protocol Secure

**HW** Hardware

**IDE** Integrated Development Environment

**JSON** JavaScript Object Notation

**JVM** Java Virtual Machine

**ORM** Objektově relační mapování

**OS** Operační systém

**POJO** Plain Old Java Object

**RPC** Remote Procedure Call

**SMS** Short Message Service

**SOAP** Simple Object Access Protocol

**SSL** Secure Sockets Layer

**TLS** Transport Layer Security

**URI** Uniform Resource Identifier

**WS** Web Services

**WADL** Web Application Description Language

**WSDL** Web Services Description Language

**XML** eXtensible Markup Language

**ZČU** Západočeská univerzita v Plzni

# Seznam obrázků

# Seznam tabulek

# Literatura


[1] *Android Definition* [online]. Sharpened Productions, 2016. [cit. 2018/12/05]. The Tech Terms Computer Dictionary. Dostupné z: https://techterms.com/definition/android.

[2] *Unknown sources in Android - Applivery Docs* [online]. Applivery, 2017. [cit. 2018/12/05]. Applivery – docs. Dostupné z: https://www.applivery.com/docs/troubleshooting/android-unknown-sources.

[3] *Android 9 features and APIs* [online]. Android Developers, 2018. [cit. 2018/12/05]. Android Developers - Docs. Dostupné z: https://developer.android.com/about/versions/pie/android-9.0.

[4] *IDC - Smartphone Market Share - OS* [online]. IDC, 2018. [cit. 2018/12/05]. IDC - Market Share. Dostupné z: https://www.idc.com/promo/smartphone-market-share/os.

[5] *Co je to Notifikace?* [online]. IT-Slovnik.cz team, 2018. [cit. 2018/12/05]. IT slovník. Dostupné z: https://it-slovnik.cz/pojem/notifikace.

[6] *What is widget?* [online]. TechTarget, 2015. [cit. 2018/12/05]. WhatIs.com. Dostupné z: https://whatis.techtarget.com/definition/widget.

[7] *Introduction - Material Design* [online]. Google, 2019. [cit. 2019/04/08]. Android Developers - Docs. Dostupné z: https://material.io/design/introduction/#principles.

[8] *What is Material Design?* [online]. Interaction Design Foundation, 2019. [cit. 2019/05/12]. Dostupné z: https://www.interaction-design.org/literature/topics/material-design.

[9] *Android Jetpack* [online]. Android Developers, 2015. [cit. 2019/01/23]. Android Developers - Docs. Dostupné z: https://developer.android.com/jetpack/.

[10] *Richardson Maturity Model – REST API Tutorial* [online]. RESTfulAPI.net, 2017. [cit. 2018/12/23]. Tech Target. Dostupné z: https://restfulapi.net/richardson-maturity-model/.

[11] *Volley overview* [online]. Android Developers, 2019. [cit. 2019/04/28]. Android Developers - Docs. Dostupné z: https://developer.android.com/training/volley.

[12] *Perform network operations using Cronet* [online]. Android Developers, 2019. [cit. 2019/04/28]. Android Developers - Docs. Dostupné z: https://developer.android.com/guide/topics/connectivity/cronet.

[13] *OkHttp* [online]. Square, Inc., 2016. [cit. 2019/04/28]. Android Developers - Docs. Dostupné z: https://square.github.io/okhttp/.

[14] *A curated list of awesome Kotlin frameworks, libraries, documents and other resources* [online]. Github, 2018. [cit. 2019/01/31]. Dostupné z: https://github.com/mcxiaoke/awesome-kotlin.

[15] *Kodein DI* [online]. Kodein Koders, 2018. [cit. 2019/01/28]. Dostupné z: https://kodein.org/di/.

[16] *IS/STAG - Další klienti* [online]. CIV-SIS ZČU, 2017. [cit. 2018/11/03]. Dokumentace IS/STAG - Mobilní aplikace. Dostupné z: https://is-stag.zcu.cz/zajemci/pristupy/dalsi_klienti.html.

[17] *IS/STAG - Mobilní evaluace* [online]. CIV-SIS ZČU, 2017. [cit. 2018/11/02]. Dokumentace IS/STAG. Dostupné z: https://is-stag.zcu.cz/zakaznici/mobilni_aplikace/mobilni-evaluace/.

[18] *Centrum výpočetní techniky: UPlikace: Centrum výpočetní techniky* [online]. Univerzita Palackého v Olomouci, 2018. [cit. 2018/11/02]. UPOL - CVT. Dostupné z: https://cvt.upol.cz/zpravodaj-cvt/uplikace/.

[19] *UniApps* [online]. UniApps s.r.o., 2014. [cit. 2018/11/03]. Dostupné z: https://uniapps.eu/index.html.

[20] *Mobilní aplikace Univerzity Tomáše Bati ve Zlíně* [online]. Univerzita Tomáše Bati ve Zlíně, 2013. [cit. 2018/11/04]. Dostupné z: http://aplikace.utb.cz/.

[21] *Mobilní aplikace UTB* [online]. Radek Vala, 2017. [cit. 2018/11/04]. Dostupné z: https://mobapp.utb.cz/.

[22] *Student ZČU – Apps on Google Play* [online]. TommyLabs, 2018. [cit. 2018/11/04]. Google Play. Dostupné z: https://play.google.com/store/apps/details?id=com.tommylabs.portalzcu.

[23] *Jidelníček ZČU - Přehledný jídelní lístek plzeňských menz a bufetů Západočeské univerzity.* [online]. Martin Sloup, 2015. [cit. 2018/11/04]. GitHub. Dostupné z: https://github.com/arcao/JidelnicekZCU/.

[24] *Zabezpečené operace, přihlašování* [online]. CIV-SIS ZČU, 2018. [cit. 2018/11/14]. Dokumentace IS/STAG. Dostupné z: https://is-stag.zcu.cz/napoveda/web-services/ws_prihlasovani.html.

[25] *Update UI components with NavigationUI* [online]. Android Developers, 2019. [cit. 2019/04/28]. Android Developers - Docs. Dostupné z: https://developer.android.com/guide/navigation/navigation-ui.

[26] *ViewModel Overview* [online]. Android Developers, 2019. [cit. 2019/04/29]. Android Developers - Docs. Dostupné z: https://developer.android.com/topic/libraries/architecture/viewmodel.

[27] Dudová, V. Rozvrh hodin pro mobilní zařízení, 2012.

[28] Giuliani, A. *insert-koin.io · a smart Kotlin dependency injection framework* [online]. Giuliani, Arnaud and Koin contributors, 2018. [cit. 2019/01/28]. Dostupné z: https://insert-koin.io/.

[29] Jemerov, S. D. . I. *Kotlin in Action*. Manning Publications Company, 2016. ISBN 978-1617293290.

[30] Krill, P. *JetBrains readies JVM-based language* [online]. InfoWorld, 2011. [cit. 2019/01/23]. Dostupné z: https://www.infoworld.com/article/2622405/java/jetbrains-readies-jvm-based-language.html.

[31] Miller, P. *Google is adding Kotlin as an official programming language for Android development* [online]. The Verge, 2017. [cit. 2018/12/05]. The Verge - Article. Dostupné z: https://www.theverge.com/2017/5/17/15654988/google-jet-brains-kotlin-programming-language-android-development-io-2017.

[32] Rouse, M. *What is REST (REpresentational State Transfer)?* [online]. Tech Target, 2017. [cit. 2018/12/23]. Tech Target. Dostupné z: https://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer.

# A Instalace

Pro spuštění aplikace je třeba zařízení s OS Android 5.0 nebo vyšší.

Aplikace pro svůj provoz potřebuje dvě oprávnění:

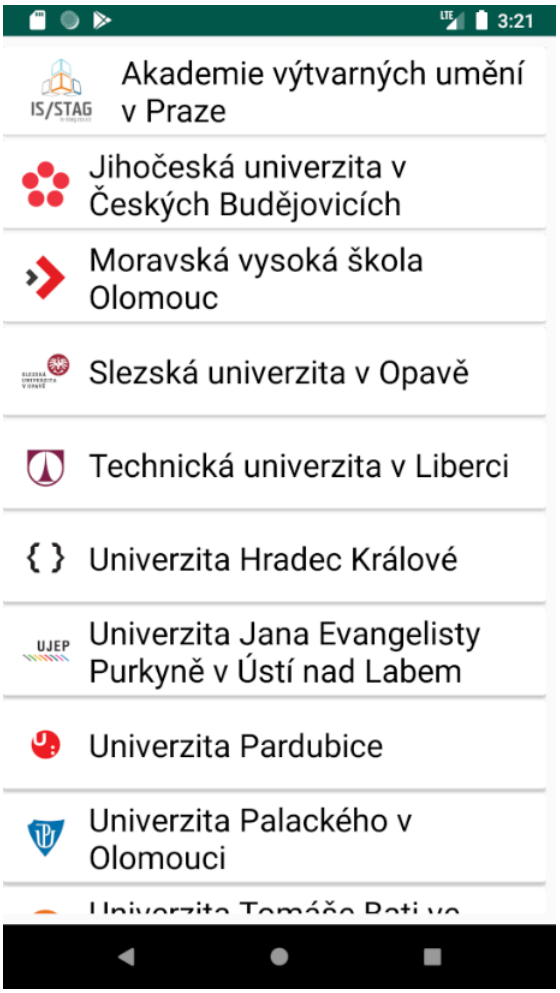
1. `ACCESS_NETWORK_STATE` – Přístup k informacím o síti pro detekování připojení k internetu,
2. `INTERNET` – Přístup k internetu pro přístup k WS IS/STAG a práci s jejich daty

Aplikaci je v současné době možné nainstalovat z přiloženého APK. Na webové stránce https://home.zcu.cz/ zimmma/issm-release/ je pak možné nalézt nejnovější verzi APK. Pro úspěšné dokončení je třeba mít v telefonu povolenou instalaci aplikací z neznámých zdrojů (pro více informací viz [2]).

V blízké době bude možné aplikaci nalézt na Google Play.

# B Uživatelská příručka

Při prvotním spuštění je nutné připojení k internetu (o čemž je uživatel v případě jeho absence informován). Po úspěšné instalaci vidí uživatel obrazovku s volbou školy. Příklad je možné vidět na obrázku B.1.

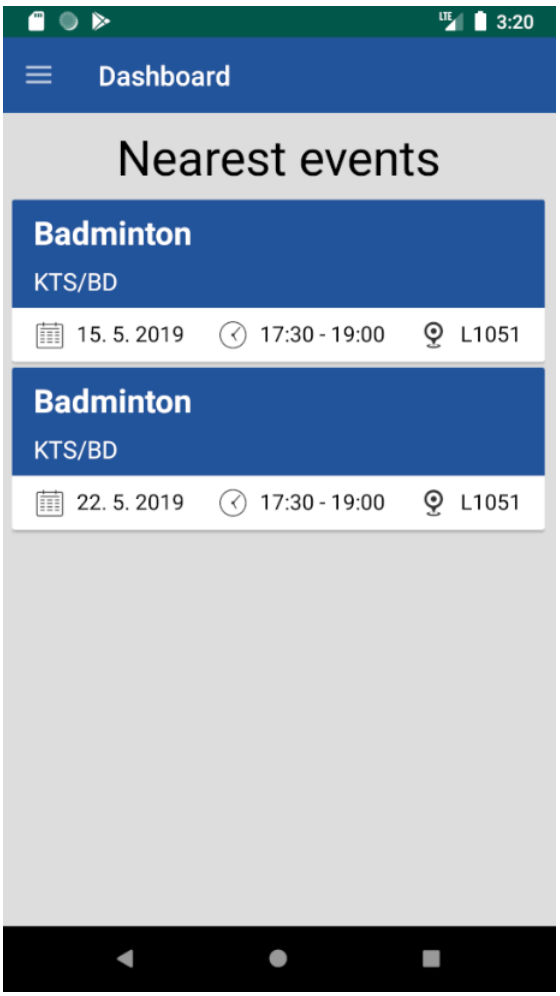



Obrázek B.1: Úvodní obrazovka s výběrem škol.

Po zvolení je uživateli zobrazena webová stránka, kde má možnost se přihlásit zadáním svých přihlašovacích údajů. Systém přihlášení a barevný styl stránky se mohou různit dle zvolené školy – například na ZČU je standardní přihlášení do IS/STAG pro studenty a učitele zaměněno autentizací přes WebAuth.

Po úspěšném přihlášení se zobrazí hlavní obrazovka s přehledem, na které

jsou zobrazeny další rozvrhové akce – viz obrázek B.2.

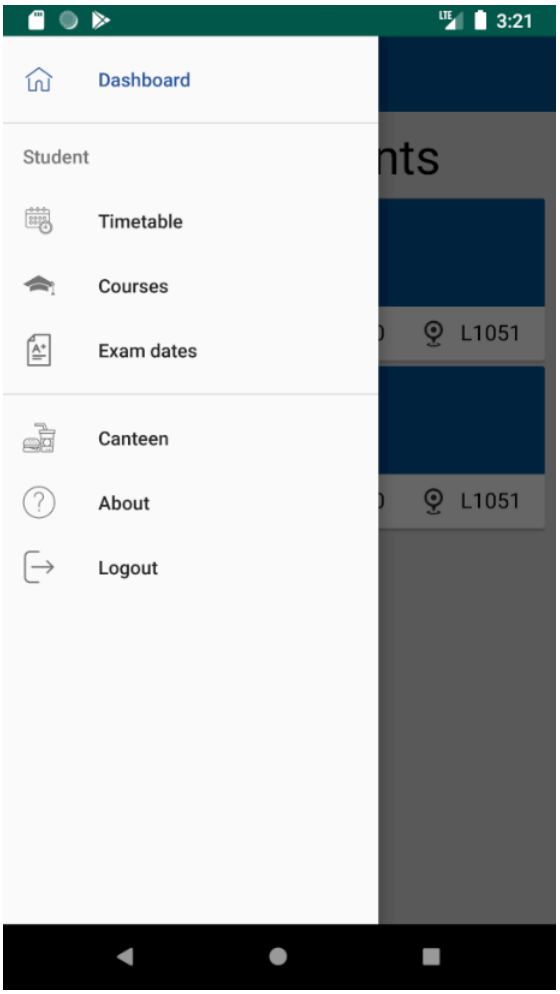



Obrázek B.2: Obrazovka přehledu.

Navigace po aplikaci je dostupná v panelu, který je možné vysunout z levé části. Ukázku je možné vidět v obrázku B.3. Předměty jsou tříděné na hlavní část (v současnosti pouze obrazovka s přehledem), část dle role (v ukázce část studenta) a ostatní obrazovky – tam patří i specifické obrazovky z externích systémů (jako třeba jídelníček ZČU).

Obrazovku jídelníčku je možné vidět na obrázku B.4.

Na obrazovce s rozvrhem je možné vidět list rozvrhových akcí zobrazených na časové ose. Standardně zobrazuje jednu hlavní akci (tu nejbližší právě probíhající, či v budoucnu chystanou) a ostatní, následující po ní. Ty mohou být odděleny buď to časovým úsekem, pokud jsou dostatečně daleko od sebe ve stejném dni, nebo datem následujícího dne, pokud se akce nekonají ve

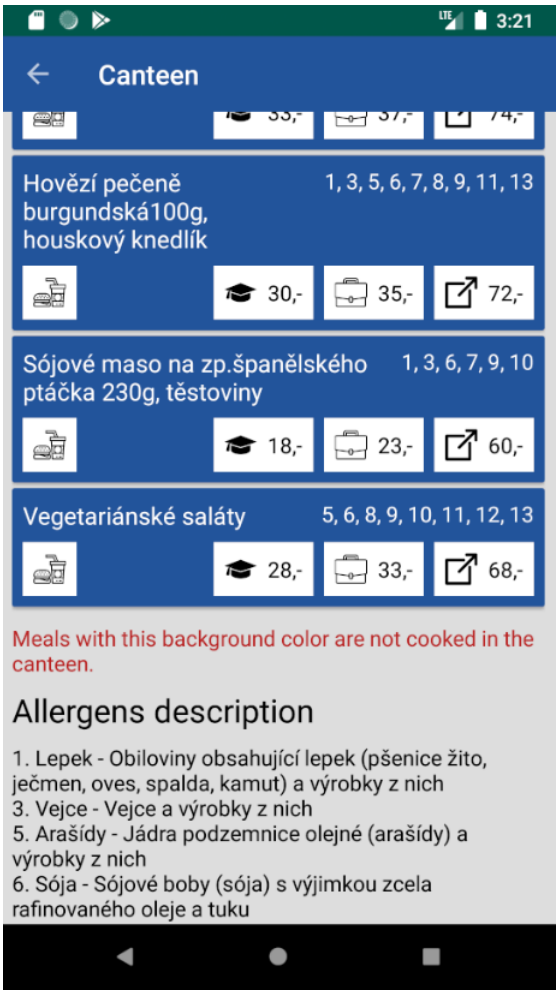



Obrázek B.3: Obrazovka navigace s výsuvným panelem.

stejném dni. Ukázku je možné vidět na obrázku B.5. Rozvrh je dostupný pro obě role – student i učitel.

Další obrazovkou je seznam studovaných předmětů pro studenta. Zobrazuje rovněž stav studia jednotlivých předmětů – zelenou barvou značí úspěšně dostudovaný předmět, červená značí nedostudovaný předmět. Stejným systémem značí i podsložky – zápočet i zkoušku. Ukázku je možné vidět na obrázku B.6.

Poslední obrazovkou pro studenta je zobrazení zkouškových termínů. To funkčně kopíruje portálovou stránku – zobrazuje všechny dostupné termíny a ukazuje důležité informace o jednotlivých termínech. Barevně odlišuje termíny, které si lze zapsat (modrým podbarvením), ty, které už student zapsané má (podbarvené zeleně) a nakonec termíny, na které se student zapsat nemůže (podbarvené červeně). K těm navíc zobrazuje případně i dodatečné

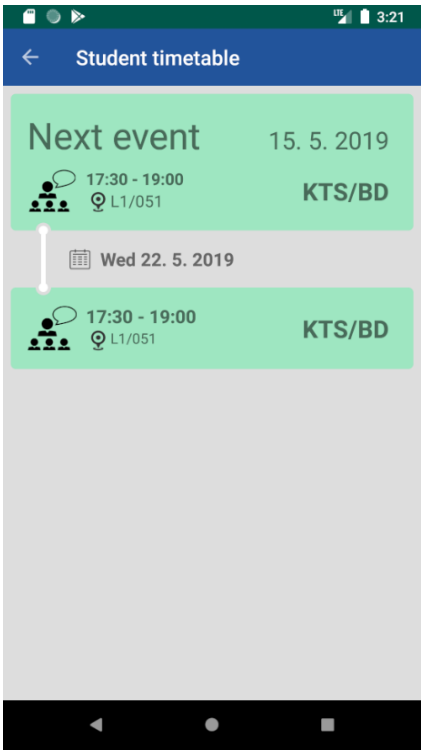



Obrázek B.4: Obrazovka jídelníčku.

zprávy – typicky proč si je nelze zapsat. Každý z termínů si lze rozkliknout pro zobrazení dodatečných detailů a tlačítka pro jejich zápis a odzápis. Ukázku je možné vidět na obrázku B.7.

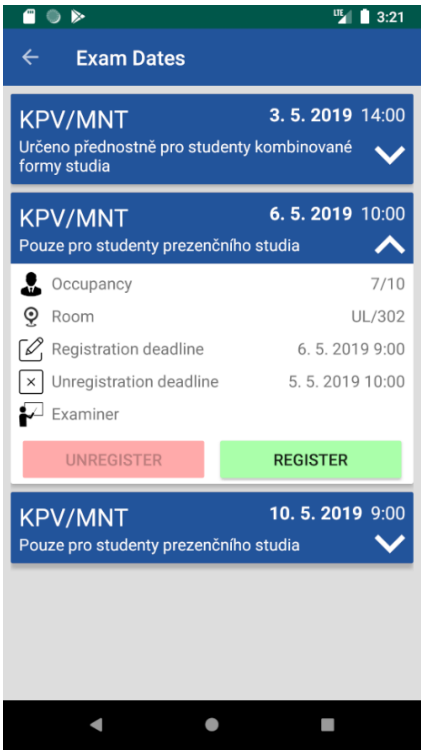
Další navigační možnosti zahrnují zobrazení obrazovky o aplikaci, která dává uživateli informaci o používané verzi aplikace, kontakt na autora aplikace a poděkování za užívání různých ikon zdarma v rámci licence CC. Ukázku je možné vidět na obrázku B.8. Poslední možností je odhlášení uživatele aplikace. To smaže veškerá uložená data a vrátí uživatele na startovní obrazovku s výběrem školy.

Všechny datové obrazovky mají systém pro automatickou aktualizaci zobrazených dat. Uživatel však může tento mechanismus obejít – přetažením prstu shora dolů vynutí manuální aktualizaci dat z internetu.

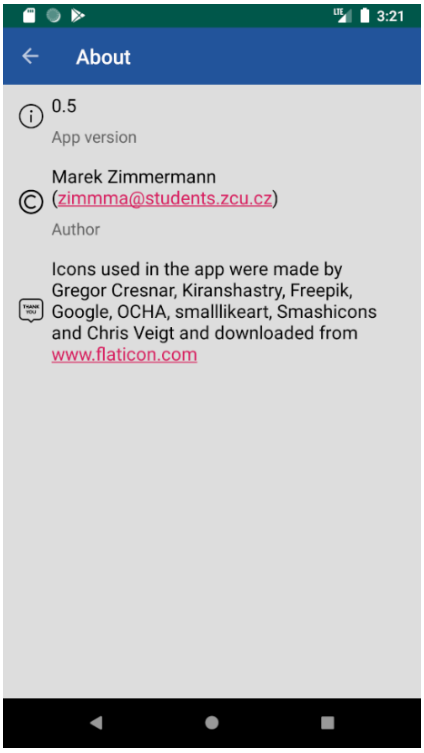



Obrázek B.5: Obrazovka rozvrhu.


Obrázek B.6: Obrazovka studovaných předmětů.




Obrázek B.7: Obrazovka zkouškových termínů.

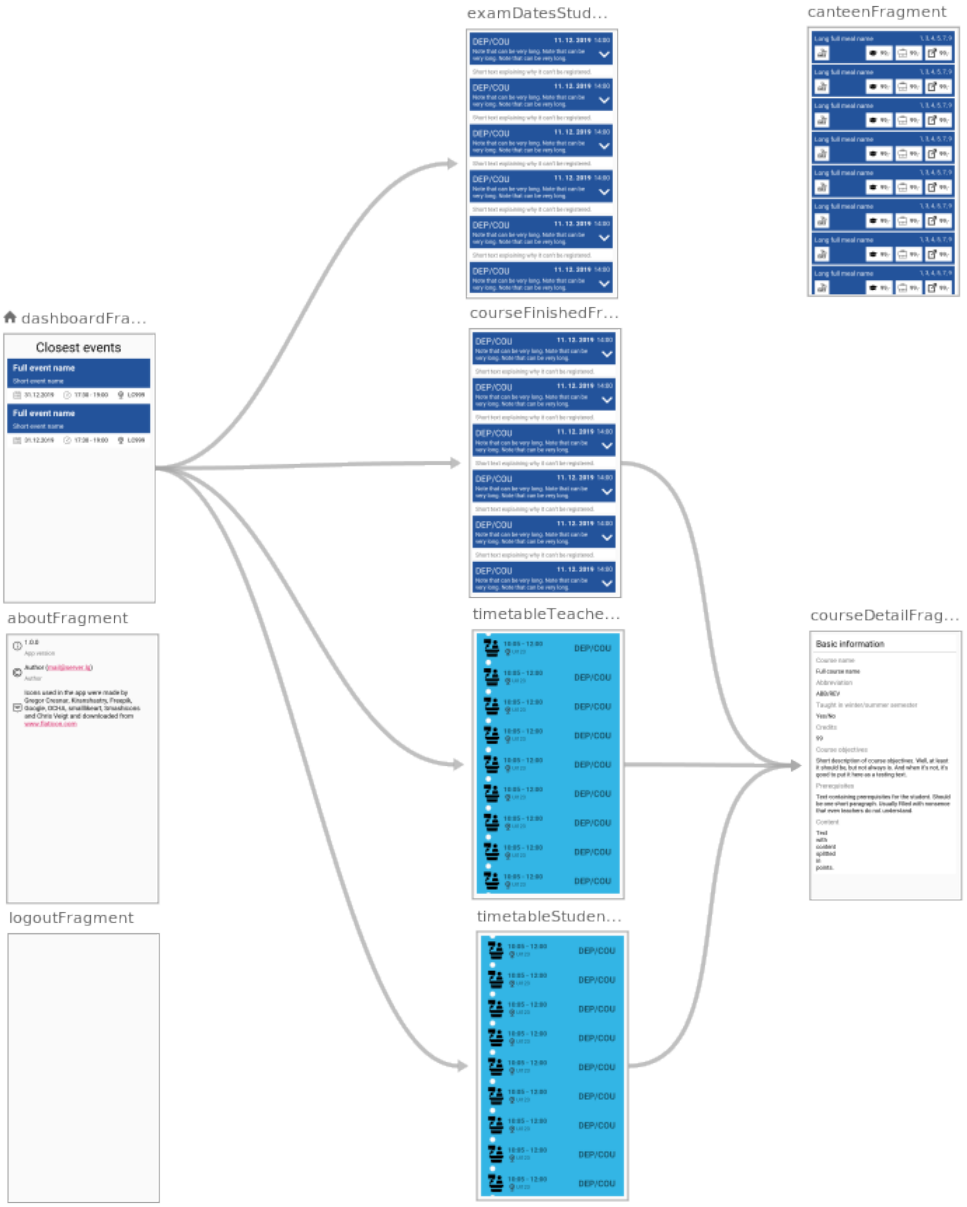



Obrázek B.8: Obrazovka o aplikaci.

# C Obsah přiloženého DVD

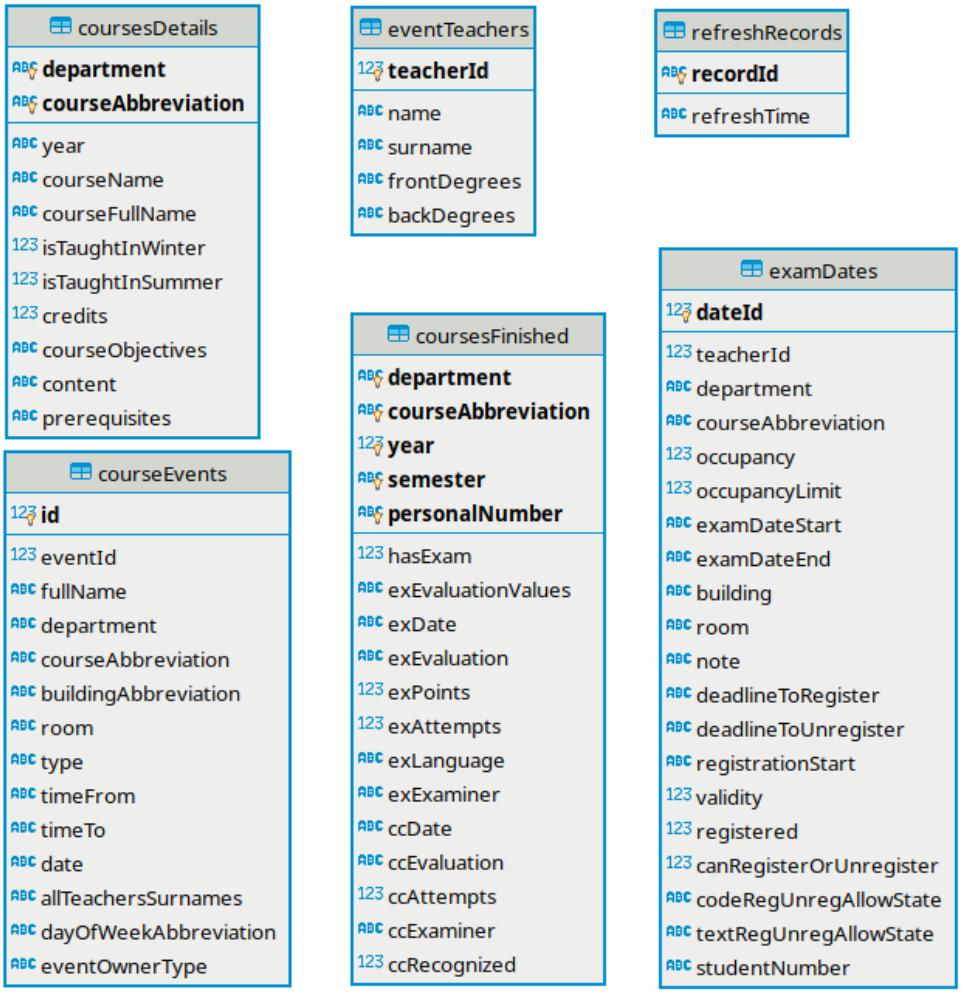
- složka `bin` – obsahuje instalační soubor aplikace `IS-STAG Mobile.apk`,
- složka `doc` – obsahuje `Zimmermann_DP.pdf` – dokument diplomové práce,
- složka `javadoc` – obsahuje dokumentaci aplikačního kódu vygenerovaného pomocí *Dokka*,
- složka `poster` – obsahuje `.pub` poster a jeho PDF verzi,
- složka `src` – obsahuje zdrojové kódy aplikace,
- složka `tex` – obsahuje zdrojové kódy dokumentu diplomové práce psané v LaTeXu,
- soubor `readme.txt` – zkopírovaný obsah této přílohy

# D Navigační graf aplikace

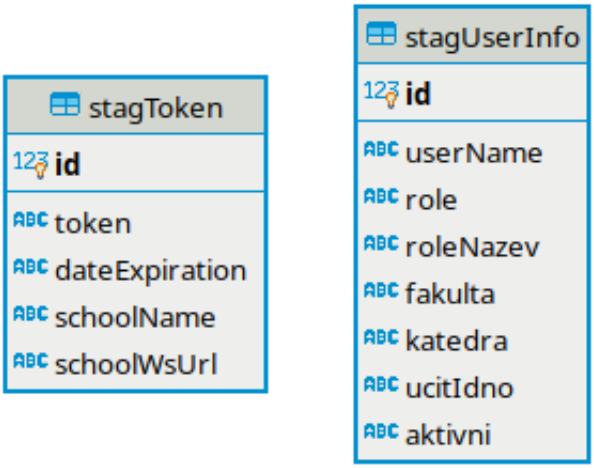



Obrázek D.1: Graf komponenty Navigation pro aplikaci.

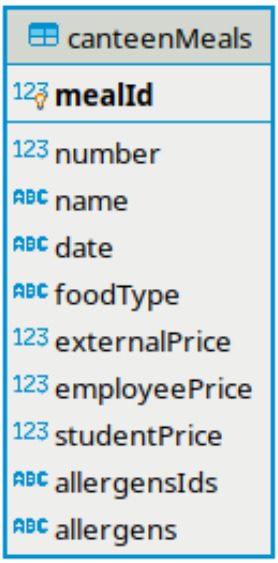
# E Struktura databází




Obrázek E.1: Struktura databáze uchovávající data z IS/STAG.




Obrázek E.2: Struktura databáze ukládající informace o uživateli.




Obrázek E.3: Struktura databáze ukládající data z externích zdrojů.