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

Mobilní a administrační webová aplikace pro podporu dne otevřených dveří na FIM

Diplomová práce

Autor: Bc. Dušan Salay
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2023

Prohlášení:

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

……

V Hradci Králové dne 10.5.2023 Dušan Salay

Anotace

Tato diplomová práce si klade za cíl vytvořit mobilní aplikaci, která by zlepšila zážitek při navštívení fakulty při dnu otevřených dveří. Společně s mobilní aplikací byla vytvořena i webová aplikace ze které si mobilní klient stahuje data přes REST rozhraní. Data je možná přes UI webové aplikace editovat. Teoretická část práce se věnuje popisu operačního systému Android a použití nového moderního přístupu tvorby UI pomocí Jetpack Compose. Dále se zabývá popisem platformy JAVA EE pro vývoj webových aplikací se zaměřením na SpringBoot Framework. Práce také popisuje výběr vhodných technologií se zaměřením na použití jednotného programovacího jazyka Kotlin od JetBrains. V další části je popsán framework KVision, který je postaven nad Kotlin/JS a používá se pro vývoj uživatelského rozhraní webových aplikací. Praktická část práce se už zabývá konkrétní implementací aplikací a popisuje jednotlivé části kódu, aby mohla posloužit i jako inspirace nebo návod pro vývoj moderních mobilních a webových aplikací v jazyce Kotlin.

Klíčová slova:

Android, Spring Boot, KVision, Kotlin, KotlinJS, Jetpack Compose, MVVM, REST API, Flyway, PostgreSQL, jednotný codebase

Annotation

This thesis aims to create a mobile application that would enhance the experience of visiting the faculty during an open day. Along with the mobile application, a web application was created from which the mobile client downloads data through a REST interface. The data can be edited through the web application UI. The theoretical part of the thesis is focused on the description of the Android operating system and the use of a new modern approach of UI creation using Jetpack Compose. It also discusses the description of the JAVA EE platform for web application development with a focus on the SpringBoot Framework. The thesis also describes the selection of appropriate technologies with a focus on the use of JetBrains' Kotlin unified programming language. The next section describes the KVision framework, which is built on top of Kotlin/JS and is used for developing the user interface of web applications. The practical part of the thesis deals already with the specific implementation of the applications and describes the different parts of the code, so that it can also serve as inspiration or guidance for the development of modern mobile and web applications in Kotlin.

Keywords:

Android, Spring Boot, KVision, Kotlin, KotlinJS, Jetpack Compose, MVVM, REST API, Flyway, PostgreSQL, unified codebase

Obsah

Seznam obrázků

Terminologický slovník

API ..... Application Program Interface - rozhraní k softwarové aplikaci

boilerplate kód ..... kód který se v projektu zbytečně opakuje a snižuje jeho čitelnost

CRUD ..... skupina čtyř základních operací: Create, Read, Update, Delete

DI ..... Dependenci Injection neboli vkládání závislostí

GUI ..... Graphic User Interface – grafické uživatelské rozhraní aplikace

ID ..... jednoznačný identifikátor (označení)

JSON ..... formát pro výměnu dat vycházející ze syntaxe JavaScriptu

layout ..... grafické rozložení stránky (plochy)

open-source ..... licence umožňující šířit celý projekt ke komerčním účelům

OS ..... operační systém

SQL ..... Structured Query Language - programovací jazyk pro práci s databází

UI ..... uživatelské rozhraní

UML ..... Unified Modeling Language - modelovací jazyk

Use Case Model ..... model služeb aplikace pro uživatele

XML ..... eXtensible Markup Language – rozšiřitelný značkovací jazyk

1 Úvod

Dnešní doba je neodmyslitelně spojená s rychlým rozvojem informačních a komunikačních technologií. Mobilní zařízení jsou dnes součástí našeho každodenního života a mohou nám výrazně usnadnit mnoho činností. Jednou z oblastí, kde mohou mobilní aplikace mít přínos, je organizace a řízení akcí, jako je den otevřených dveří.

Hlavním programovacím jazykem, který byl pro tuto práci vybrán, je Kotlin od společnosti JetBrains. Kotlin je jazyk, který se v posledních letech stal velmi oblíbeným pro vývoj aplikací pro operační systém Android. Avšak jeho možnosti nejsou omezeny jen na mobilní platformu, Kotlin lze úspěšně použít i pro vývoj webových aplikací.

1.1 Cíl práce

Cílem této diplomové práce je vytvořit mobilní a webovou administrační aplikaci pro podporu dne otevřených dveří na FIM. Mobilní aplikace by umožnila zlepšit návštěvníkům celkový dojem během takové události a pomohla by uchazeče informovat o všech událostech, které se v daný den na fakultě konají. Je dbáno na to, aby pro vývoj aplikací byly použité aktuální moderní technologie. V případě mobilní aplikace se jedná hlavně o použití nového Material Designu 3 a jeho implementaci přes Jetpack Compose. Záměrem je také aby aplikace používala vzory a barvy fakulty. Webová aplikace má sloužit jako poskytovatel dat pro mobilního klienta a také má jednoduše umožnit editovat data přes webové uživatelské prostředí.

Dále si práce dává za cíl vybrat a popsat použité technologie se zaměřením na použití jednotného programovacího jazyka Kotlin od JetBrains a zdůvodnit jejich způsob použití.

1.2 Metodika zpracování

Při zpracování teoretické části práce bylo čerpáno z odborných knižních publikací, ale také převážně z webových zdrojů, kvůli použití nových technologií, které ani svou odbornou tištěnou publikaci ještě nemají. Na základě získaných znalostí byly realizovány aplikace pro Andorid OS a webová aplikace. Obě napsané v jazyce

Kotlin. Autor v době psaní práce disponoval programovacími zkušenostmi z již šestileté praxe tudíž bylo čerpáno i z vlastních zkušeností v dané oblasti.

Vytvoření aplikací se realizovalo s pomocí programu Android Studio pro mobilní aplikaci a programu InteliJ Idea pro webovou aplikaci. Programátorské postupy byly nabývány zejména z oficiálních dokumentací daných frameworků a knihoven. Pro platformu Android je to stránka Android Developers. (1) Pro Spring Boot oficiální reference dokumentace. (2) A pro KVision dokumentace napsaná tvůrcem a vydaná na gitbooku. (3)

Tvorba obsahu pro mobilní aplikaci probíhala zejména na základě oficiálních stránek fakulty. Konkrétně šlo o sekci k dni otevřených dveří. (4)

2 OS Android

V současné době trh s mobilními zařízeními ovládli dva operační systémy, a to Android OS a iOS, které dohromady tvoří 99,9 % podílu na trhu. (5) Z toho 72 % mobilních zařízení využívá operační systém Android, díky čemuž se stal globálně nepoužívanějším operačním systémem. (6)

Android je mobilní operační systém založený na upravené verzi jádra Linuxu a dalším softwaru s otevřeným zdrojovým kódem, který je určen především pro mobilní zařízení s dotykovou obrazovkou, jako jsou chytré telefony, tablety navigace, fotoaparáty a chytré hodinky. Systém Android také disponuje rozsáhlým ekosystémem aplikací. V obchodě Google Play jsou k dispozici miliony aplikací pro různé potřeby a zájmy uživatelů. (7) Obchod Google Play slouží vývojářům jako centralizované tržiště pro distribuci jejich aplikací a umožňuje uživatelům snadno stahovat a aktualizovat aplikace.

2.1 Historie

Android byl původně vyvinut společností Android Inc., kterou v roce 2005 koupila společnost Google. První veřejná verze systému Android byla vydána v roce 2008 při uvedení zařízení HTC Dream na trh. (8)

Od té doby prošel systém Android několika iteracemi a aktualizacemi, přičemž každá verze byla pojmenována podle zákusku nebo sladké pochoutky v abecedním pořadí, například Donut (1.6) Ice Cream Sandwich (4.0), Jelly Bean (4.1-4.3), KitKat (4.4), Marshmallow (6.0), Oreo (8.0-8.1) nebo Pie (9.0) (Meier, 2016). Po Androidu 10 se společnost Google rozhodla ukončit pojmenovávání podle dezertů.

Obrázek 1: jednotlivé verze Android OS a jejich pojmenování podle sladkostí (9)

2.2 Základní Android komponenty

Aplikace pro Android se skládají ze čtyř základních součástí: Aktivity (Activities), Služby (Services), Přijímače (Broadcast Receivers) a Poskytovatelé obsahu (Content Providers). V dalších podkapitolách si je blíže popíšeme.

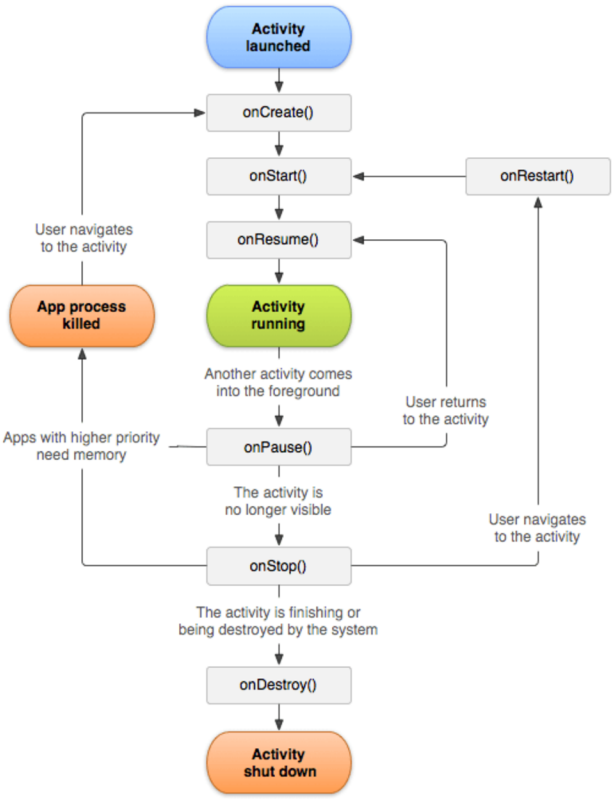
2.2.1 Aktivity

Aktivity slouží jako základní stavební prvky uživatelského rozhraní aplikace. Každá aktivita představuje jednu obrazovku v rámci aplikace a poskytuje uživatelům interaktivní prostředí pro konkrétní úkony v aplikaci. Aktivity hrají klíčovou roli v celkovém uživatelském zážitku, protože svou stavbou a vzájemným propojením definují strukturu aplikace. (8)

2.2.1.1 Životní cyklus aktivity

Aktivity mají přesně definovaný životní cyklus, který se skládá z různých stavů, jako je onCreate, onStart, onResume, onPause, onStop a onDestroy.

Správným používáním jednotlivých částí životního cyklu aktivity zajistíme, že systémové prostředky jsou alokovány a uvolňovány podle potřeby, což zabraňuje únikům paměti a problémům s výkonem. (8)



Obrázek 2: životní cyklus Android aktivity (10)

2.2.1.2 Intenty a sdílení dat (Intents)

Aktivity používají intenty jako systém zpráv pro komunikaci s ostatními součástmi aplikace nebo jinými aplikacemi v zařízení. Intenty lze použít ke spouštění nových aktivit, předávání dat mezi aktivitami, a dokonce i ke spouštění služeb na pozadí nebo práci s přijímači (broadcast receivers). Z toho plyne, že pomocí intentů může vývojář vytvářet i komplexnější workflows a propojit, tak svou aplikaci s aplikacemi třetích stran. (8)

2.2.2 Služby (Services)

Služby v systému Android jsou základní komponenty, které provádějí dlouhotrvající operace na pozadí bez uživatelského rozhraní a umožňují aplikacím pokračovat ve své činnosti, i když s nimi uživatel aktivně neinteraguje. (8) Služby lze používat k různým úkolům, jako je stahování dat z internetu, přehrávání hudby nebo provádění výpočtů na pozadí, a to vše bez nutnosti, aby uživatel nechal aplikaci otevřenou.
V systému Android existují dva hlavní typy služeb: spuštěné (started) služby a vázané (bounded) služby. (11)
Spuštění služby je iniciováno komponentou aplikace, například aktivitou nebo jinou službou, pomocí intentů. Po spuštění mohou tyto služby pokračovat v běhu na pozadí po neomezenou dobu, i když je komponenta, která je spustila, zničena. (8) Naproti tomu vázané služby jsou spojeny s jinými komponentami aplikace, což těmto komponentám umožňuje přímou interakci se službou. Vázané služby se obvykle používají, pokud aplikace vyžaduje trvalé spojení se službou za účelem výměny dat nebo provádění složitějších operací. (11)

2.2.3 Přijímače (Broadcast Receivers)

Přijímače umožňují aplikacím reagovat na události celého systému nebo specifické pro aplikaci, aniž by byly aktivní nebo viditelné pro uživatele. (8) Fungují jako komunikační kanál a umožňují aplikacím přijímat již zmíněné intenty, které mohou přijmou od jiných komponent v rámci téže aplikace nebo od samotného systému Android.
Běžným případem použití přijímačů je reakce na systémové události, jako jsou změny v připojení k síti, aktualizace stavu baterie nebo příchozí notifikace. (8) Aplikace může například pomocí přijímačů zjistit, kdy se zařízení připojí k síti Wi-Fi, a poté automaticky synchronizovat data se vzdáleným serverem. Aplikace tak zůstává aktualizovaná a funkční i v době, kdy není aktivně používána.

2.2.4 Content providers (Poskytovatelé obsahu)

Poskytovatelé obsahu jsou klíčovou součástí sloužící jako abstrakční vrstva pro správu a sdílení dat mezi různými aplikacemi. Umožňují aplikacím přistupovat k datům uloženým v databázích konzistentním a bezpečným způsobem. (8)

2.3 Nástroje a prostředí pro vývoj Android aplikací

V rámci ekosystém systému Android existuje řadu nástrojů, které usnadňují vývoj, testování a nasazení aplikací pro systém Android. Níže jsou popsány některé klíčové součásti vývojového prostředí systému Android.

2.3.1 Android Studio

Android Studio je oficiální vývojové prostředí (IDE) pro vývoj aplikací pro Android, postavené na platformě IntelliJ IDEA od společnosti JetBrains. Poskytuje komplexní sadu nástrojů pro vytváření, ladění a optimalizaci aplikací pro Android, včetně návrhu uživatelského rozhraní nebo modul pro práci s GITem (8). Android Studio obsahuje také výkonný emulátor pro testování aplikací na různých konfiguracích zařízení a verzích systému Android bez nutnosti použití fyzického hardwaru.

2.3.2 Gradle

Gradle je výchozí automatizační sestavovací systém pro aplikace Android, který je zodpovědný za správu závislostí, kompilaci kódu a generování souborů APK pro distribuci. Gradle používá doménově specifický jazyk (DSL) založený na programovacím jazyce Groovy nebo Kotlin, který vývojářům umožňuje psát sestavovací skripty. Gradle navíc podporuje paralelní spouštění jednotlivých částí buildu což pomáhá zvyšovat rychlost a efektivitu celého sestavování. (8)

2.3.3 Emulátory a virtuální zařízení

Emulátory a virtuální zařízení umožňují simulaci různých vlastností zařízení, jako je velikost obrazovky, rozlišení, hardwarové senzory nebo specifickou konfiguraci systému. Emulátor je také možné spustit s libovolnou verzí operačního systému Android, což umožňuje otestovat, zda je aplikace kompatibilní i se staršími verzemi operačního systému.

2.3.4 Android SDK

Android Software development Kit (SDK) je komplexní sada nástrojů potřebných pro vývoj Android aplikací. Obecně platí, že SDK nástroje jsou nezávislé na

platformě, na které vyvíjíme. Jedním z významných nástrojů je Android Debug Bridge. (11) Chcete-li používat ADB se zařízením připojeným přes USB, musíte povolit ladění USB v systémovém nastavení zařízení v sekci Možnosti pro vývojáře.

2.3.4.1 Android Debug Bridge (ADB)

Android Debug Bridge (ADB) je univerzální program pro příkazovou řádku, který nám umožňuje komunikovat s instancí OS na emulátoru nebo připojeným zařízením se systémem Android. Umožnuje například instalace a debug aplikací. Také poskytuje přístup k unixovému shellu zařízení a tím pádem umožňuje spouštět systémové příkazy. Je dobré mít na paměti, že ADB nemůže přímo měnit nastavení zařízení nebo provádět akce, které vyžadují přímou interakci uživatele, například klikání na prvky uživatelského rozhraní. (8) (11)

2.4 Zabezpečení a oprávnění

Android OS používá model oprávnění pro řízení přístupu jednotlivých aplikací k senzitivním informacím (údaje o poloze, kontakty, přístup k fotoaparátu a mikrofonu, odesílání SMS zpráv nebo přístup k internetu).
Vývojáři musí deklarovat jednotlivá oprávnění, která jejich aplikace vyžaduje, v souboru AndroidManifest.xml a uživatelé musí tato oprávnění udělit během instalace nebo v průběhu používání aplikace, v závislosti na verzi systému Android. (8) Runtime permissions byly zavedeny od verze systému 6.0 (Marshmallow).
Tento model zabezpečení pomáhá zajistit, aby aplikace měly přístup pouze k potřebným prostředkům, a poskytuje uživatelům plnou kontrolu nad jejich daty a možnostmi přístupu k hardwaru zařízení. (11)

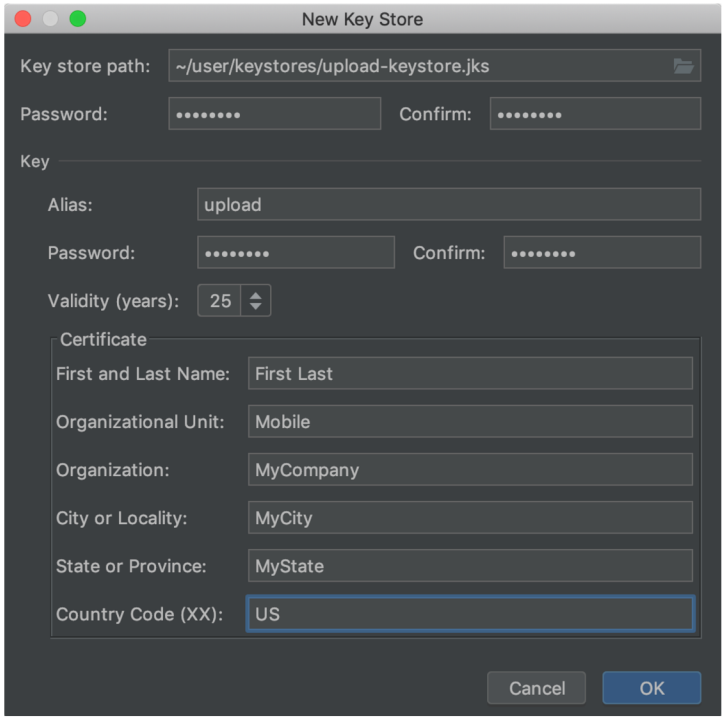
2.4.1 Application sandbox

Jedná se o takzvanou karanténu pro aplikace v Android OS, která izoluje aplikaci na úrovni Linux kernelu s využití user a group oprávnění. Tento přístup zajišťuje, že aplikace nemohou bez výslovného povolení přistupovat k datům nebo prostředkům jiných aplikací ani do nich zasahovat, čímž chrání soukromí uživatelů a napomáhá lepší stabilitě systému. (12)

2.4.2 Podepisování aplikací

Před publikováním aplikace přes online distribuční službu jako je třeba obchod s aplikacemi Google Play musí být aplikace digitálně podepsána soukromým klíčem. (11) Proces podepisování zahrnuje vytvoření digitálního podpisu aplikace (APK) pomocí kryptografického algoritmu, který lze později ověřit pomocí příslušného veřejného klíče. Digitální podpis slouží jako jedinečný identifikátor aplikace, který umožňuje systému Android ověřit její pravost a zajistit, že s ní nebylo po podepsání manipulováno nebo že nebyla upravena.

Vývojář může vygenerovat klíče například přes Java keytool nebo přes vestavěného průvodce přímo v Android Studiu. Soukromý klíč je spolu s přidruženým veřejným klíčem uložen v souboru Java KeyStore, který je chráněn heslem. (8) Je potřeba mít na paměti, že svůj soukromý klíč a soubor KeyStore je nutno uchovávat v bezpečí, protože je vyžadován následně i pro podepisování aktualizací aplikace.



Obrázek 3: snímek z Android Studia při podepisování APK (13)

3 Java EE (Jakarta EE)

Java Enterprise Edition (Java EE), nyní známá jako Jakarta EE, je široce používaná platforma navržená zejména pro vývoj rozsáhlých, robustních a bezpečných distribuovaných aplikací. Platforma nabízí sadu specifikací, APIs a služeb, které zjednodušují vývoj a správu vícevrstvých webových aplikací. V této kapitole se budeme zabývat stručnou historií, klíčovými vlastnostmi, základními technologiemi, vývojovými nástroji a bezpečnostními aspekty platformy Java EE.

3.1 Stručná historie

Java EE byla vytvořena společností Sun Microsystems s cílem poskytnout robustní platformu pro podnikové aplikace. V roce 2010 po akvizici Oraclem se stala Java EE součástí portfolia Oracle, který nabízel i placenou podporu pro enterprise klienty. V roce 2017 byla celá platforma předána pod křídla Eclipse Foundation, která jej přejmenovala na Jakarta EE. (14)

3.2 Hlavní Java EE komponenty

Jakarta EE zahrnuje řadu technologií a specifikací navržených pro vývoj podnikových aplikací. Některé z nich si krátce přiblížíme v kapitolách níže.

3.2.1 Servrlety a JSP

Servlety a JavaServer Pages (JSP) představují základní prvky pro vývoj webových aplikací na platformě Jakarta EE. Servrlety poskytují mechanismy pro zpracování HTTP požadavků. Konkrétně podtřídy třídy *javax.servlet.http.HttpServlet*, jsou určeny ke zpracování HTTP požadavků. Poskytují metody *doGet()*, *doPost()*, *doPut()*, *doDelete()* a další metody, které odpovídají protokolu HTTP. (15)

JSP se používá pro generování dynamického obsahu. Jedná se vlastně a textové soubory, které podporují kombinování statického HTML kódu a Java kódu společně s vlastním tagovacím systémem JSTL (JSP – Standard Tag Library). Díky JSTL je možné v JSP souborech používat například podmínky nebo iterační cykly. (15)

Jako alternativa ve Spring Frameworku je dostupný modul ThymeLeaf.

3.2.2 EJB

EJB je serverová komponenta, které zapouzdřuje business logiku aplikace. Byla implementována s cílem zjednodušit vývoj rozsáhlých distribuovaných aplikací. Umožňuje správu transakcí, zabezpečení, správu konkurenčnosti a také umožňuje dependency injection (DI). V porovnání se Spring Frameworkem se považováno spíše za těžkopádné řešení kvůli složitosti jeho API a nutnosti potřeby EJB kontejneru. (15)

3.2.3 JPA

JPA je specifikace standardizovaného rozhraní API pro objektově-relační mapování v jazyce Java. Nabízí způsob mapování mezi objekty Javy a relačními databázovými entitami. JPA umožňuje komunikovat s databází objektově orientovaným způsobem bez nutnosti psát SQL dotazy. (15)
Také Spring Framework poskytuje skvělou podporu pro JPA. Tato integrace je obecně známá jako Spring Data JPA.

3.2.4 REST

REST můžeme popsat jako architektonický způsob pro navrhování webových aplikací, které potřebují komunikovat po síti. Pro komunikaci se používá bez stavový klient-server komunikační protokol HTTP. REST je postaven na principu zdrojů, které jsou definovány pomocí adres URL a také čtyřmi základními metodami GET, POST, PUT a DELETE, které se překrývají s funkcemi CRUD. (15)

3.3 Nástroje a prostředí pro vývoj

3.3.1 InteliJ IDEA

IntelliJ IDEA je populární vývojové prostředí (IDE) vyvinuté společností JetBrains. Podporuje více programovacích jazyků, ale je známé především pro vývoj v jazyce Java a Kotlin. Tento software obsahuje integrované nástroje pro přístup k databázi, terminál, nástroje debugovaní aplikace, unit testování a také podporuje nástroje pro automatizaci sestavení aplikace jako je Maven nebo Gradle. Má dále podporu různých frameworků, jako jsou Jakarta EE, Spring Boot, Hibernate a další.

3.3.2 Aplikační servery

Aplikační servery jsou softwarové platformy, které umožňují nasadit a spravovat aplikace ve více vláknovém prostředí. Poskytují nástroje pro zabezpečení, zpracování transakcí, řízení konkurenčnosti a správy dostupných hardwarových zdrojů. (16)

Existuje několik typů aplikačních serverů (čerpáno z: (16)):
Java EE Application Servers – Jedná se o servery jako WildFly, GlassFish a IBM WebSphere, které podporují celý stack Java EE.
Servletové kontejnery – Jedná se o servery jako Apache Tomcat a Jetty. Jsou jednodušší a méně náročné, jelikož podporují především jen běh servletů a JSP.
Lightweight Application Servers – Tyto servery jsou vestavěné servery přímo ve frameworku Spring Boot. Může to být například Tomcat, Jetty nebo Undertow, jsou jednoduché, nenáročné na systémové zdroje a ideální pro použití v architektuře mikro-služeb.

4 Výběr vhodných technologií

Jedním z klíčových aspektů pro výběr použitých technologií byla možnost psát kód v jazyce Kotlin a tím mít sjednocený codebase v jednom programovacím jazyce pro celý projekt.

4.1 Spring Boot Webflux

Spring Boot WebFlux je součástí reaktivního stacku Spring 5, který umožňuje vývoj neblokujících, reaktivních webových aplikací. Framework je navržen tak, aby zvládal velké množství souběžných připojení s malým počtem vláken, a to díky implementaci paradigmatu reaktivního programování. (16)

Vychází z projektu Reactor a jeho typů *Flux* a *Mono*, které používá pro asynchronní zpracování datových toků. (17) Reactor je reaktivní knihovna již čtvrté generace, založená na specifikaci Java Reactive Streams. Slouží tedy pro vytváření neblokujících aplikací ve světě JVM. (18)

Hlavní výhodou při použití jazyka Kotlin společně s Webfluxem, je možnost použití Kotlin Coroutines. Coroutines pomáhají psát neblokující, asynchronní kód lineárnějším a čitelnějším způsobem.

4.1.1 Nasazení Spring Boot aplikací

Spring aplikace lze pro nasazení zabalit do souboru WAR na plnohodnotný aplikační server nebo jen jako kontejner servlet. Spring Boot však nabízí moderní přístup, jelikož umožňuje použít vestavěný server Netty, Tomcat nebo Jetty, který umožnuje celou aplikaci zabalit jako spustitelný soubor JAR a spustit tak aplikaci s minimální potřebou konfigurace. (16)

4.2 KVision

Jedním z dalších frameworků je KVision, který využívá Kotlin/JS a poskytuje komplexní a výkonnou platformu pro vytváření moderních webových aplikací. V nadcházejících podkapitolách bude popsán jeho účel, funkce a výhody. Dále budeme hlouběji zabývat architekturou tohoto frameworku a komponentami uživatelského rozhraní.

4.2.1 Kotlin/JS

Kotlin/JS je čím dál oblíbenější jazyk pro tvorbu webových aplikací, které mohou běžet jak na straně klienta (prohlížeče), tak na straně serveru. Jedná se o verzi programovacího jazyka Kotlin, která byla upravena pro jazyk JavaScript. Kotlin/JS nabízí několik výhod oproti jiným jazykům pro vývoj webových aplikací, jako jsou JavaScript, TypeScript a Dart.

Jednou z hlavních výhod jazyka Kotlin/JS je, že se jedná o silně typovaný jazyk. To znamená, že překladač kontroluje typ každé proměnné a funkce při kompilaci, což může pomoci zachytit chyby ještě před spuštěním kódu. Podle společnosti JetBrains, která stojí za Kotlinem, může silné typování pomoci snížit počet chyb při běhu až o 40 % ve srovnání s JavaScriptem. (19)

Další výhodou jazyka Kotlin/JS je jeho interoperabilita s knihovnami jazyka JavaScript. Kotlin/JS může používat libovolnou knihovnu JavaScriptu a kód Kotlinu lze volat z kódu JavaScriptu. To pomáhá využít rozsáhlý ekosystém knihoven JavaScriptu a zároveň používat silně typovaný jazyk.

Kotlin/JS také nabízí lepší podporu asynchronního programování než JavaScript. Podle studie společnosti Mozilla může být asynchronní programování založené na zpětných voláních (callbacks), které je v JavaScriptu časté, náchylné k chybám a obtížné na údržbu. V některých případech můžeme mluvit o takzvaném „callback hell“. Kotlin/JS naproti tomu nabízí coroutines, které poskytují efektivnější a čitelnější způsob zpracování asynchronních úloh. (20)

4.2.2 Definice a účel použití

KVision je open-source framework pro vývoj webových aplikací využívající programovací jazyk Kotlin/JS (21). Vytvořil jej Robert Jaroš a první verze byla vydána v roce 2018 s cílem zjednodušit a zefektivnit vývoj webových aplikací pro vývojáře v jazyce Kotlin (21). KVision poskytuje komplexní sadu komponent uživatelského rozhraní, reaktivní a typově bezpečný programovací model a bezproblémovou integraci s dalšími populárními webovými technologiemi (21). Díky využití výhod a flexibility jazyka Kotlin/JS umožňuje KVision vývojářům efektivněji vytvářet na funkce obsáhlé, dobře výkonné a snadno udržovatelné webové aplikace (21).

V rámci použití frameworku pro vývoj webové administrativní aplikace lze využít hlavně výhody jednotného programovacího jazyka Kotlin. Díky čemuž se doménové třídy můžou sdílet jak pro back-end a front-end aplikaci, tak i pro samotnou mobilní aplikaci pro Android.

4.2.2.1 Monorepo

Monorepo (monolitický repozitář) je strategie vývoje softwaru, kdy je kód pro více projektů uložen v jednom repozitáři. Tento přístup používá několik velkých softwarových společností, například Google, Facebook a Twitter. Tento přístup je v kontrastu s přístupem multi-repo, kdy má každý projekt svůj vlastní repozitář. (22) Když je všechen kód uložen na jednom místě, je snazší jej sdílet a opakovaně využívat v rámci více projektů. To může také pomoci zajistit lepší konzistenci domain modelu napříč všemi projekty. Další výhodou je, že změny napříč více projekty je v rámci monorepa mnohem jednoduší, jelikož máme k dispozici vždy celý codebase.

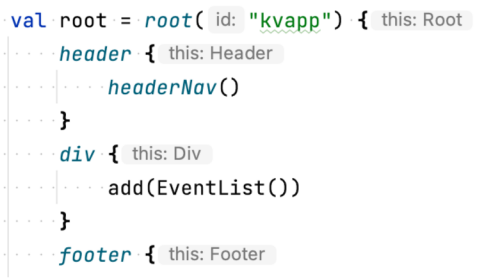
Použití monorepa dává velký smysl i v našem případě, jelikož všechny části projektu jsou napsány v Kotlinu a také mají sdílené části. Dobrý příklad může být struktura REST rozhraní, kterou musí implementovat jak back-end aplikace, tak mobilní klient.

4.2.3 Funkce a výhody

KVision nabízí několik zajímavých funkcí, které z něj činí atraktivní volbu pro vývoj full-stack webových aplikací.

4.2.3.1 Přístup na principu DSL

KVision využívá Kotlin DSL (Domain-Specific Language) k tvorbě uživatelských rozhraní deklarativním a typově bezpečným způsobem. Tento přístup umožňuje vývojářům definovat komponenty uživatelského rozhraní a jejich vlastnosti pomocí expresivní syntaxe jazyka Kotlin, což v konečném důsledku vede k čitelnějšímu, lépe udržovatelnému kódu. (23) Zároveň tento přístup zamezí zapsání syntaktické chyby nebo například použití atributu který není konkrétní HTML komponentou podporován.



Obrázek 4: sestavení skeletonu aplikace s využitím DSL builderů – kód webové aplikace [autor práce]

4.2.3.2 UI komponenty a data binding

KVision obsahuje celou řadu již sestavených komponent uživatelského rozhraní. Například layout kontejnery, ovládací prvky (tlačítka, selecty), formuláře, tabulky nebo grafy. Tyto komponenty lze snadno přizpůsobit a rozšířit tak, aby splňovaly specifické požadavky libovolné webové aplikace.

Jednou ze základních funkcí systému KVision je podpora takzvaného data bindingu, která vývojářům umožňuje automaticky synchronizovat UI komponenty s datovými modely. (23) Tento reaktivní přístup zajišťuje, že se změny vstupních dat automaticky promítnou do uživatelského rozhraní, což snižuje zejména potřebu ručních aktualizací.

```
formPanel<Form> {
    add(
        Form::text,
        Text(label = "Text field") {
            placeholder = "Enter text"
        })
    add(Form::password, Password(label = "Password field"))
    add(Form::textarea, TextArea(label = "Text area field"))
    add(Form::richtext, RichText(label = "Rich text field"))
    add(Form::date, DateTime(format = "YYYY-MM-DD", label = "Date field"))
    add(Form::time, DateTime(format = "HH:mm", label = "Time field"))
    add(Form::checkbox, CheckBox(label = "Required checkbox"))
    add(Form::radio, Radio(label = "Radio button"))
```

Obrázek 5: data binding pro formPanel v KVision (24)

4.2.4 KVision architektura

KVision podporuje oddělení front-end a back-end kódu uspořádáním aplikace do samostatných modulů. Tento modulární přístup umožňuje zachovat jednoznačné

oddělení jednotlivých částí aplikace čímž podporuje lepší organizaci kódu. Front-end moduly jsou zodpovědné za definici UI rozhraní aplikace a obecně za obsluhu veškerých interakcí s uživatelem, zatímco back-end moduly se starají o ukládání dat, jejich zpracování a business logiku na straně serveru. (23)

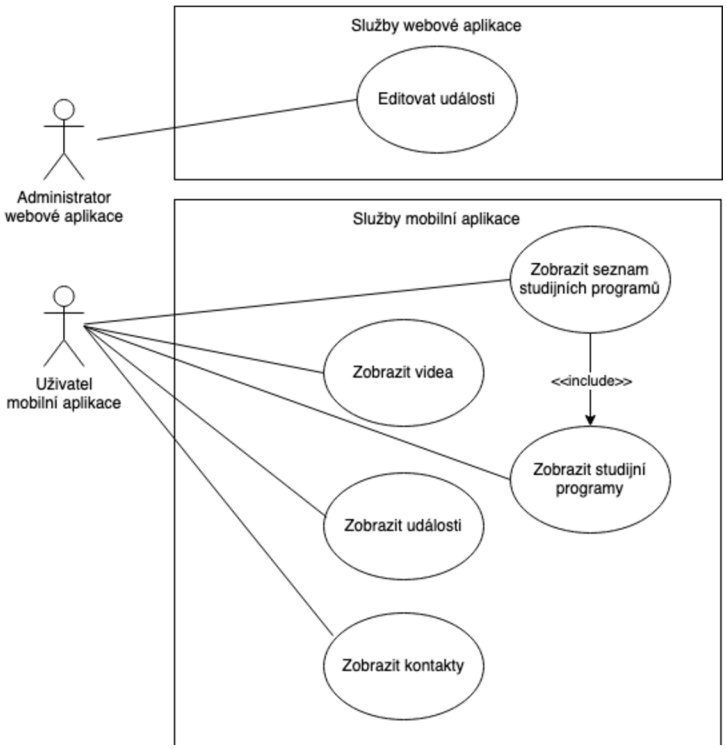
5 Návrh a vývoj mobilní a webové aplikace

V následujících kapitolách bude popsán vývoj webové a mobilní aplikace napsané v Kotlinu.

5.1 Funkční analýza

Pro znázornění funkčních požadavků aplikace lze použít modelovací jazyk UML. V modelu lze reprezentovat vzájemnou interakci mezi uživatelem a aplikací. Každý use case diagram se skládá z aktérů, činností a vztahů mezi nimi. Aktérovi je přiřazena jen taková typová úloha, kterou bude mít právo používat. Jestliže jedna úloha ke svému vhodnému vykonání činností vyžaduje úlohu druhou, je vazba mezi úlohami označena specifickým indikátorem <<*include*>>.

Následující obrázek byl vytvořen v online editoru pro tvorbu diagramů draw.io. Jedná se o individuální úlohy poskytované aplikacemi směrem k uživateli.



Obrázek 6: Use Case model aplikace

5.2 Webová aplikace

Webová aplikace se skládá ze dvou hlavních částí, a to backend napsaný pomocí frameworku Spring Boot 3 a frontend části, která je napsána za pomocí KVision frameworku. V navazujících kapitolách budou popsány jednotlivé části webové aplikace. Primárně se jedná o část s REST rozhraním pro poskytování dat mobilní aplikaci a druhou část společně s UI pro administraci jednotlivých událostí.

5.2.1 Ukládaní dat

V každé webové aplikaci je perzistence dat klíčovou součástí systému. Jde o mechanismus, který umožňuje ukládat a načítat data i po vypnutí nebo restartu aplikace. Ve webové aplikaci založené na platformě Spring to obvykle zahrnuje použití modulu Spring Data JPA (Java Persistence API) spolu s relačním databázovým systémem. V našem případě byla použita databáze PostgreSQL.
Pro použití Spring Data JPA, musíme do souboru `build.gradle.kts` zahrnout příslušné závislosti.

```
//data persistence
implementation( dependencyNotation: "org.springframework.boot:spring-boot-starter-data-r2dbc")
runtimeOnly( dependencyNotation: "org.postgresql:postgresql")
runtimeOnly( dependencyNotation: "org.postgresql:r2dbc-postgresql")

//db scheme migration
implementation( dependencyNotation: "org.flywaydb:flyway-core")
```

Obrázek 7: nutné závislosti pro ukládaní dat do PostreSQL přes Spring Data R2DBC [autor práce]

Dále je třeba vydefinovat správně nastavení pro připojení k databázi. Nastavení připojení k databázi se obvykle konfiguruje v souboru `application.properties` nebo `application.yml`.

```
spring:
  r2dbc:
    url: r2dbc:postgresql://localhost:5432/odof
    username: postgres
    password: toor
```

Obrázek 8: nastavení připojení k DB v application.yml [autor práce]

Dalším krokem je potřeba vytvořit třídy entit. Ve Spring Data JPA to jsou jednoduché data třídy jazyka Kotlin, které jsou mapovány na databázovou tabulku. Každá instance entity představuje jeden řádek dat v tabulce. Třídy entit jsou anotovány

pomocí @Table. Každý atribut, který má být persistován, musí být anotován pomocí @Column. Pro index sloupce se zase používá anotace @Id.

Zde je jednoduchý příklad mapování tabulky na entitu:

```
@Table("video_info")
data class VideoInfoEntity(
    @Id
    @Column val id: UUID? = null,
    @Column val name: String,
    @Column val url: String,
    @Column val thumbnail: String?,
    @Column val description: String?,
)
```

Obrázek 9: ukázka mapování entity na tabulku video_info - kód webové aplikace [autor práce]

Vytvořené entity se musejí zapojit do takzvaných repositories. Ve Spring Data JPA jsou to rozhraní (interfaces), které dědí z již naimplementovaných rozhraní knihovny Spring Data Repository, například *CrudRepository*. V našem případě byl použit *CoroutineCrudRepository*, který má v sobě již zaimplementovanou podporu coroutines. Tímto získáme mnoho CRUD funkcí, aniž bychom museli psát jakýkoliv implementační kód včetně SQL dotazů.

```
@Repository
interface DocumentInfoRepository : CoroutineCrudRepository<DocumentInfoEntity, UUID> {

    fun findAllByFieldOfStudyInfoId(fieldOfStudyInfoId: UUID): Flow<DocumentInfoEntity>
}
```

Obrázek 10: CRUD repository nad DocumentInfo entitou – kód webové aplikace [autor práce]

Pokud bychom ovšem chtěli napsat vlastní SQL dotaz nad databází můžeme do rozhraní přidat vlastní funkci s anotací @Query a do ní vložit databázový dotaz.
Jakmile nadefinuje všechny potřebné repositories, můžeme je libovolně v aplikaci používat pomocí Spring Bean autowiringu, například v komponentách @Service nebo @Controller.

5.2.1.1 PostgreSQL

PostgreSQL je open-source a objektově-relační databázový systém (ORDBMS), který klade důraz na rozšiřitelnost a kompatibilitu s SQL. Je plně kompatibilní s ACID (Atomicity, Consistency, Isolation, Durability), což díky transakčnímu systému

garantuje integritu dat i v případě chyb na straně DB systému. (25) PostgreSQL je vysoce rozšiřitelný. Umožňuje vytvářet vlastní funkce pomocí různých programovacích jazyků, jako je PL/pgSQL. Podporuje také přidávání nových datových typů, operátorů a indexových metod. (26)

5.2.2 Flyway – migrace databázového schématu

S vývojem aplikací může být správa změn databázového schématu náročná, zejména při práci v týmu. Flyway tento problém řeší tím, že databázi verzuje a automaticky aplikuje migrace, čímž zajišťuje, že každá instance spuštěné aplikace pracuje se správnou verzí databázového schématu. Pro integraci Flyway společně se Spring Bootem stačí přidat do build souboru závislost na knihovnu a v `application.yml` nastavit připojení do databáze na kterou se migrace budou aplikovat.

```yaml
spring:
  flyway:
    url: jdbc:postgresql://localhost:5432/odof
    user: postgres
    password: toor
    locations: classpath:db/migration
```

Obrázek 11: nastavení Flyway v application.yml [autor práce]

Atribut *locations* určuje cestu, kde jsou umístěny migrační skripty.

Migrační skripty jsou soubory s SQL skripty, které Flyway spouští v daném pořadí podle čísla verze. Jsou pojmenovány číslem verze, oddělovačem a popiskem, například `V1__createEventTable.sql`.

Flyway sleduje historii změn schématu pomocí vlastní tabulky v databázi, obvykle pojmenované *flyway_schema_history*. Každému migračnímu skriptu odpovídá řádek v této tabulce, což Flywayi umožňuje určit, které migrace již byly použity a které čekají na provedení. Do této tabulky je dobré se také koukat, pokud je potřeba řešit problémy s migracemi, které neskončili úspěšně nebo jen když potřebujeme zjistit, jak dlouho trvalo provedení migrace.

installed_rank	version	description	type	script	checksum	i...	installed_on	execution_time	success
1	1	createEventTable	SQL	V1__createEventTable.sql	1174126545	postgres	2023-04-03 00:44:58.686…	10	true
2	2	createFieldOfSt…	SQL	V2__createFieldOfStudyT…	2086958588	postgres	2023-04-08 22:15:09.745…	16	true
3	3	createVideoInfo…	SQL	V3__createVideoInfoTabl…	-1933985423	postgres	2023-04-08 22:15:09.776…	6	true

Obrázek 12: ukázka struktury tabulky flyway_schema_history [autor práce]

5.2.3 REST rozhraní

REST je architektonický styl pro komunikaci po síti. V kontextu vývoje webových aplikací umožňují REST rozhraní klientům (například mobilní aplikace) komunikovat s webovou službou pomocí protokolu HTTP.

Spring Boot poskytuje skvělou podporu pro vývoj REST APIs a má podobně jako pro persistenci dat již implementované komponenty. Založení takové služby vyžaduje vytvoření takzvaných controller tříd, které budou zpracovávat http požadavky a vytvářet odpovědi. Tyto třídy jsou anotovány pomocí @RestController a pro zpracování různých typů požadavků používají anotace na úrovni metod, jako jsou @GetMapping, @PostMapping, @PutMapping, @DeleteMapping.

```
@RestController
@RequestMapping(path = ["api/v1/"], consumes = ["application/json"], produces = ["application/json"])
class MobileApiController(
    private val universityEventRepository: UniversityEventRepository,
    private val fieldOfStudyInfoService: FieldOfStudyInfoService,
    private val videoInfoRepository: VideoInfoRepository,
) {

    @GetMapping("/event")
    suspend fun getEvents(): List<UniversityEventResponse> {
        return universityEventRepository.findAll().map { it.convertToResponse() }.toList()
    }

    @GetMapping("/video")
    suspend fun getVideos(): List<VideoInfoResponse> {
        return videoInfoRepository.findAll().map { it.toResponse() }.toList()
    }
```

Obrázek 13: část implementace REST rozhraní pro mobilní aplikaci – kód webové aplikace [autor práce]

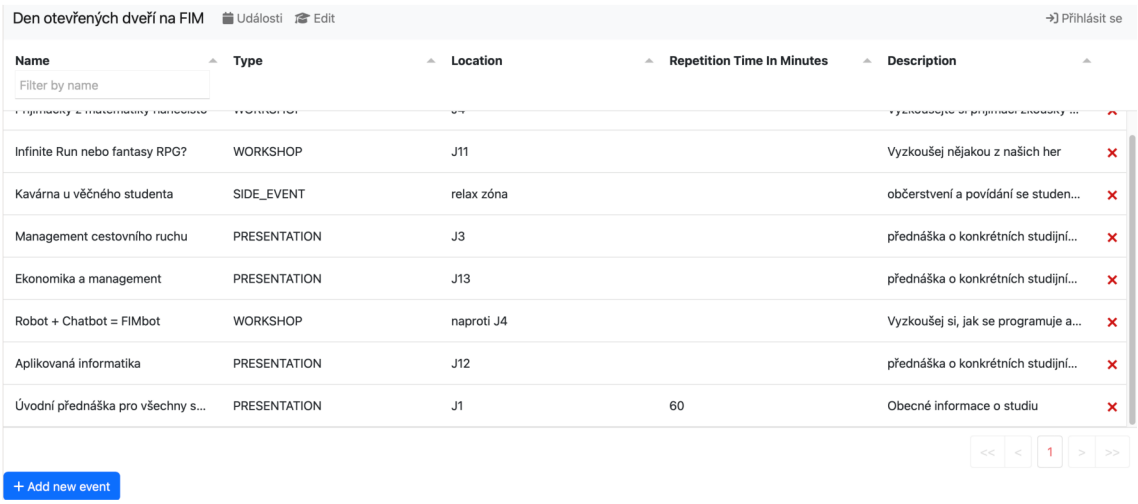
Spring poskytuje anotaci @ExceptionHandler pro definici metod, které zpracovávají výjimky. Tuto anotaci můžeme použít k vrácení příslušných stavových kódů HTTP a chybových zpráv, když se něco pokazí.

5.2.4 Uživatelské rozhraní

Uživatelské rozhraní webové aplikace bylo naimplementováno pomocí frameworku KVision popsaného v kapitole 4.2.

Pro výpis a editaci jednotlivých událostí byla použita předpřipravená komponenta tabulator vycházející z knihovny Tabulator.js. (27)

Tabulator je komponenta KVision pro vytváření interaktivních tabulek. Nabízí širokou škálu funkcí pro manipulaci s daty, jako je třídění, filtrování, seskupování a stránkování. seskupování a stránkování. Tabulátor také poskytuje různé interaktivní funkce, jako jsou editovatelné buňky, selectable řádky a podpora drag and drop přetahování. (28)



Obrázek 14: KVision – komponenta tabulator pro zobrazení jednotlivých událostí [autor práce]

5.3 Mobilní aplikace

V následujících kapitolách bude popsána architektura a jednotlivé části mobilní aplikace s ukázkami kódu a jejich popisem.

5.3.1 Nastavení manifestu

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="cz.uhk.dod">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:name="cz.uhk.dod.App"
        android:allowBackup="false"
        android:fullBackupContent="false"
        android:icon="@mipmap/ic_launcher"
        android:label="Den otevřených dveří na FIM"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AppCompat.DayNight.NoActionBar"
        android:usesCleartextTraffic="true">

        <activity
            android:name=".ui.MainActivity"
            android:exported="true"
            android:windowSoftInputMode="adjustResize"
            tools:ignore="Instantiatable">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>

        </activity>
    </application>
</manifest>
```

Obrázek 15: AndroidManifest – kód mobilní aplikace [autor práce]

Soubor AndroidManifest.xml je nezbytnou součástí každé Android aplikace a slouží jako hlavní konfigurační soubor, který poskytuje systému Android základní informace o aplikaci. Pro správný chod aplikace je třeba správně nadefinovat poskytnutí práv o které si aplikace může požádat uživatele při použití. V tomto případě musí systém udělit aplikaci jen jedno oprávnění pro přenos dat s externími službami nebo pro komunikaci s webovými službami: *android.permission.INTERNET*.

Toto oprávnění je klasifikováno jako běžné oprávnění, což znamená, že představuje relativně nízké riziko pro soukromí a bezpečnost uživatele. (8) V důsledku toho jej systém Android automaticky přiřazuje při instalaci a uživatelé jej nemusí explicitně udělovat během běhu.

5.3.2 AndroidX

AndroidX je sada knihoven, která nahradila původní Support Library pro zajištění zpětné i dopředné kompatibility. Což znamená, že aplikace navržené pro předchozí verze systému budou vždy fungovat na nejaktuálnějších verzích systému Android OS bez dalších úprav kódu aplikace. Knihovny AndroidX jsou dokonce udržovány a aktualizovány odděleně od platformy Android, což vývojářům umožňuje využívat nejnovější funkce a vylepšení bez čekání na novou verzi operačního systému kterou musí výrobce telefonu vydat nebo než uživatel takovou aktualizaci provede. (29)
Tato nová sada knihoven používá konzistentnější schéma pro pojmenování balíčků, které zahrnuje prefix *androidx*. To usnadňuje identifikaci a správu závislostí a také pochopení, které knihovny jsou součástí systému AndroidX.

Příklady knihoven (30):
AppCompat – Prvky uživatelského rozhraní, jako jsou aktivity, fragmenty a views s podporou zpětné kompatibility.
RecyclerView – View komponenta pro zobrazení velkých datových sad v rolovacím seznamu nebo gridu.
LiveData a ViewModel – Knihovny pro implementaci architektonického vzoru Model-View-ViewModel (MVVM)
Room – knihovna pro perzistenci da, která poskytuje abstrakční vrstvu nad SQLite a usnadňuje práci s databázemi.

5.3.3 Coroutines

Android aplikace mohou plnit téměř jakékoliv funkce. Například můžete chtít stahovat data, zadávat dotazy do databáze nebo provádět požadavky na webové rozhraní API. Provedení všech těchto operací může zabrat poměrně hodně času, zejména pokud jsme závislí na službě třetí strany, která se nechová stabilně. V tomto případě není žádoucí, aby uživatel čekal na dokončení těchto operací a nemohl pokračovat v používání aplikace.
A právě coroutines umožňují určit operace, které probíhají asynchronně na pozadí aplikace. Namísto toho, aby uživatel musel čekat, než se tato práce vykoná, umožňují

coroutines uživateli pokračovat v interakci s aplikací, zatímco se práce provádí na pozadí.

Spouštění coroutines se chová stejně jako spouštění samostatného vlákna (threadu). Práce s vlákny je již dlouho známa z jiných programovacích jazyků a stejně jako vlákna, tak coroutines mohou běžet paralelně a navzájem komunikovat. Klíčový rozdíl však spočívá v tom, že je výkonnější používat v kódu coroutines než vlákna. Jelikož procesor může obvykle spustit pouze omezený počet vláken najednou a z hlediska výkonu je efektivnější spustit co nejméně vláken. Oproti tomu coroutines běží ve výchozím nastavení na již dopředu vytvořeném a sdíleném zásobníku vláken (thread poolu). Zároveň na jednom vlákně může běžet více coroutines. (31)

```
class DODViewModel(
    private val repo: DODRepo,
) : ComposeViewModel() {

    fun fetchEvents() {
        launch(state = _eventState) { this: CoroutineScope
            repo.fetchAndSaveEvents()
        }
    }
```

Obrázek 16: použití launch coroutine builderu – kód mobilní aplikace [autor práce]

5.3.3.1 Coroutine scopes

Coroutine scopes definují životní cyklus coroutines a pomáhají řídit jejich spuštění. V systému Android lze pomocí předdefinovaných scopes, jako jsou *viewModelScope* a *lifecycleScope*, provázat coroutines přímo s životním cyklem jednotlivých UI komponent aplikace. To zajistí, aby se coroutines automaticky ukončily, když jsou jejich přidružené komponenty zastaveny. (8)

5.3.3.2 Suspend funkce

Suspend funkce jsou funkce označené klíčovým slovem *suspend*, které mohou pozastavit a znovu zahájit své vykonávání, aniž by došlo k zablokování aktuálního vlákna. Lze je zavolat pouze z jiných suspend funkcí nebo v rámci běžící coroutine.

5.3.3.3 Coroutine dispatchers

Coroutine dispatchers určují na jakém vlákně se coroutine spoustí. V systému Android můžeme použít *Dispatchers.Main* pro úlohy související s uživatelským rozhraním, *Dispatchers.IO* pro vstupní/výstupní operace, jako jsou síťové požadavky nebo přístup k databázi, a *Dispatchers.Default* pro úlohy náročné na výkon procesoru. (31)

5.3.3.4 Error handling v coroutines

V rámci coroutines se vyhozené exceptions (vyjímky) v hierarchii coroutines šíří směrem nahoru, dokud nenarazí na rodičovský coroutine nebo na *CoroutineExceptionHandler*. Pokud výjimka není zachycena a zpropaguje se do svého rodiče, tak v defaultním nastavení může ukončit všechny své coroutine job potomky. (31)

Pokud však použijeme *SupervisorJob* nebo *SupervisorScope*, mohou potomci takové coroutine skončit nezávisle chybou, aniž by to ovlivnilo jejich „sourozenecké“ coroutines (mají stejného rodiče). V takovém případě se výjimka propaguje dále do rodičovské coroutine nebo *CoroutineExceptionHandleru*, ale je důležité, že neukončí a nechá doběhnout své ostatní coroutines potomky. (31)

```
abstract class ComposeViewModel : ViewModel() {

    //Children of a supervisor job can fail independently of each other.
    private val job = SupervisorJob()
    private val scope = CoroutineScope( context: Dispatchers.Default + job)

    val launchState = MutableStateFlow<LaunchState>(None) //

    /**
     * @param block     Your block of application instructions.
     * @param state     State where are the lifecycle of launch is emitted. You can insert null
     *                  if you don't care about the lifecycle.
     * @param onError   Optional onError method that is called extra when error appears.
     */
    protected fun <Result> launch(
        onError: ((Exception) -> Unit)? = null,
        state: MutableStateFlow<LaunchState>? = this.launchState,
        block: (suspend CoroutineScope.() -> Result),
    ) = scope.launch(
        CoroutineExceptionHandler { _, error ->
            // Handle possible exception
            when (error) {
                is Exception -> {
                    onError?.invoke(error)
                    state?.tryEmit(Failure(error))
                }
                else -> {
                    // Optional
                }
            }
```

Obrázek 17: vytvoření scope se SupervisorJob zapojeným CoroutineExceptionHandlerem – kód mobilní aplikace [autor práce]

5.3.4 Jetpack Compose

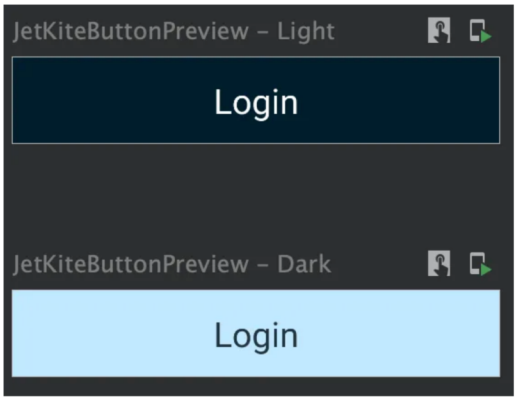
Jetpack Compose je moderní deklarativní framework pro tvorbu uživatelského rozhraní od společnosti Google. Společnost Google jej představila v roce 2019 s cílem je zjednodušit a zefektivnit vývoj uživatelského rozhraní systému Android. (8) Základem Jetpack Compose jsou takzvané composable funkce. Jedná se o funkce s anotací @Composable a slouží k popisu prvků uživatelského rozhraní. Composable funkce lze kombinovat a vnořovat do sebe a tím lze vytvářet složité hierarchie UI s menším množstvím kódu, než bylo možné při deklarování UI skrze XML.

V Android Studiu je dostupný nástroj pro zobrazení náhledu, jak bude composable funkce vypadat. Aktivuje se použitím anotace @Preview.

```
@Composable
@Preview(name = "Light")
@Preview(name = "Dark", uiMode = Configuration.UI_MODE_NIGHT_YES)
fun JetKiteButtonPreview() {
    JetKiteTheme {
        Surface {
            JetKiteButton(text = "Login") {
            }
        }
    }
}
```

JetKiteButtonPreview hosted with ❤ by GitHub view raw

Output:



Obrázek 18: Náhled na compose funkci při použití @Preview (27)

5.3.4.1 Stavy a rekompozice

Stavy a rekompozice jsou základem pro vytváření reaktivních uživatelských rozhraní v aplikaci Jetpack Compose.

Stav v Jetpack Compose označuje jakoukoli hodnotu, která se může v průběhu času změnit a způsobit aktualizaci uživatelského rozhraní. Stav v Compose se obvykle spravuje prostřednictvím objektu *androidx.compose.runtime.State*, který si uchovává hodnotu a může komponentu Compose notifikovat, když se její hodnota změní. (33)

Rekompozice je proces aktualizace uživatelského rozhraní při změně stavu. Když se změní objekt State, funkce Compose, která načítá stav, je rekomponována, což znamená, že je funkce znovu vyvolána, aby vygenerovala aktualizované uživatelské rozhraní. Překresluje se však pouze ta část uživatelského rozhraní, která závisí na změněném stavu, takže rekompozice se chová velmi efektivně s přihlédnutím na spotřebu prostředků zařízení. Důležitá je také funkce remember, která slouží k zapamatování stavu napříč rekompozicemi. (33)

5.3.5 Ktor Client (API)

Ktor je moderní, asynchronní a funkcemi nabitý framework vyvinutý společností JetBrains pro tvorbu webových aplikací. Je napsán modulárně, a tak není problém z celého frameworku využívat jen samotného http klienta.

Ktor Client je multiplatformní knihovna, která poskytuje rozšiřitelného http klienta pro různé platformy včetně Androidu. Pomocí tohoto klienta můžeme v aplikacích pro Android snadno provádět síťové požadavky a komunikovat s REST rozhraními a zpracovávat získané odpovědi. (34)

Jednou z hlavních výhod klienta Ktor je integrovaná podpora asynchronního programování pomocí Kotlin coroutines. To umožňuje provádět síťové požadavky bez blokování hlavního vlákna.

Chcete-li v aplikaci pro Android používat klienta Ktor, musíme do souboru build.gradle svého projektu přidat potřebné závislosti:

```
val ktorVersion = "2.0.3"
implementation("io.ktor:ktor-network:$ktorVersion")
implementation("io.ktor:ktor-client-core:$ktorVersion")
implementation("io.ktor:ktor-client-serialization:$ktorVersion")
implementation("io.ktor:ktor-client-android:$ktorVersion")
implementation("io.ktor:ktor-client-auth:$ktorVersion")
implementation("io.ktor:ktor-client-content-negotiation:$ktorVersion")
implementation("io.ktor:ktor-serialization-kotlinx-json:$ktorVersion")
```

Obrázek 19: závislosti pro KTor HTTP klienta – kód mobilní aplikace [autor práce]

Po přidání závislostí můžeme vytvořit instanci klienta Ktor s konkrétní konfigurací, například nastavit serializér JSON, přidat logovaní nebo nastavit defaultní hlavičky které se nastaví pro každý požadavek.

V příkladu níže lze vidět nastavení *content-type* a *accept* hlavičky na hodnotu application/json.

```
override val httpClient = HttpClient { this: HttpClientConfig<*>
    installLogging() //logging to console
    installJsonSerialization()
    installTimeout()
    defaultRequest { this: DefaultRequest.DefaultRequestBuilder
        url(BaseApiUrl)
        //headers
        contentType(ContentType.Application.Json)
        accept(ContentType.Application.Json)
    }
```

Obrázek 20: konfigurace http klienta – kód mobilní aplikace [autor práce]

S nakonfigurovanou instancí Ktor klienta můžete provádět různé síťové operace, například zadávat požadavky GET a zpracovávat jejich odpovědi pomocí metod *body*, která automaticky json odpověď deserealizuje do specifikovaného domain class typu. V ukázce níže lze vidět, že přijatou odpověď transformuje a uložíme do lokální databáze.

```
suspend fun fetchAndSaveEvents() =
    api.httpClient.get( urlString: "event") HttpResponse
        .body<List<UniversityEvent>>() List<UniversityEvent>
        .also { it: List<UniversityEvent>
            dao.replaceEvents(
                it.map(UniversityEvent::toEntity)
            )
        }
```

Obrázek 21: odeslání a zpracování GET požadavku přes Ktor klienta – kód mobilní aplikace [autor práce]

5.3.5.1 KotlinX Serialization

Kotlinx.serialization je multiplatformní knihovna jazyka Kotlin vyvinutá společností JetBrains, která poskytuje velmi výkonný nástroj na převodu domain objektů z a do různých datových formátů, jako jsou JSON, XML nebo ProtoBuf. (35) V počáteční konfiguraci http Ktor klienta byla nastavena přes metodu *installJsonSerialization()*. Použití anotace @Serializable na třídu instruuje serializační plugin, aby automaticky vygeneroval implementaci *KSerializeru* pro danou třídu, kterou lze použít k serializaci a deserializaci třídy.

```
@Serializable
data class VideoInfo(
    val name: String,
    val url: String,
    val thumbnail: String?,
    val description: String?,
) {
    fun toEntity() = VideoEntity(
        name = name,
        url = url,
        thumbnail = thumbnail,
        description = description,
    )
}
```

Obrázek 22: doménový objekt s anotací @Serialization – kód mobilní aplikace [autor práce]

5.3.6 Room (DB)

Room je knihovna pro perzistenci dat, která je součástí sady knihoven a nástrojů Android Jetpack a je určena ke zjednodušení přístupu k databázím a jejich správy.

Tato knihovna poskytuje abstrakční vrstvu nad SQLite a umožňuje vývojářům pracovat s databází pomocí objektů a anotací jazyka Java nebo Kotlin namísto psaní explicitních dotazů SQL. (36)

Room se skládá ze tří hlavních komponent: entit, DAOs (Data Access Objects) a samotné databáze.

5.3.6.1 Entity

Entity jsou třídy, které reprezentují tabulky v databázi. Jsou anotovány pomocí @Entity a každá třída odpovídá jedné tabulce v databázi SQLite. Atributy třídy definují sloupce tabulky a jejich datové typy, stejně jako případné indexy nebo vazby na další tabulky.

```
@Entity(
    primaryKeys = ["name", "fromTime", "toTime"]
)
data class EventEntity(

    val name: String,
    val type: UniversityEventType,
    val fromTime: String,
    val toTime: String,
    val location: String,
    val repetitionTimeInMinutes: Int?,
    val description: String,
) {
```

Obrázek 23: ukázka mapování entity – kód mobilní aplikace [autor práce]

5.3.6.2 DAO

DAO jsou v rámci kódu rozhraní (interfaces) nebo abstraktní třídy, které jednotlivými metodami definují přístup k datům uloženým v databázi a manipulují s nimi. Knihovna Room používá anotace, jako jsou @Insert, @Update, @Delete nebo @Query k namapování těchto metod na příslušné dotazy SQL. Zmíněná @Query umožnuje napsání vlastního SQL dotazu.

```
@Dao
interface DODDao {

    @Insert(onConflict = REPLACE)
    suspend fun insertFieldsOfStudy(fieldsOfStudy: List<FieldOfStudyEntity>)

    @Delete(entity = FieldOfStudyEntity::class)
    suspend fun deleteAllFieldsOfStudy()

    @Query("Select * From FieldOfStudyEntity")
    fun fieldsOfStudy(): Flow<List<FieldOfStudyEntity>>
```

Obrázek 24: ukázka DAO – kód mobilní aplikace [autor práce]

Dále knihovna podporuje nad metodou anotaci @Tranasction, která zabalí logiku v těle metody do jedné transakce. Taková transakce bude vyhodnocena jako úspěšná, pokud v těle metody nedojde k vyhození výjimky. V opačném případě dojde k databázovému rollbacku.

```
@Transaction
suspend fun replaceEvents(events: List<EventEntity>) {
    deleteAllEvents()
    insertEvents(events)
}
```

Obrázek 25: ukázka použití @Transaction – kód mobilní aplikace [autor práce]

5.3.6.3 Databáze

Třída *Database* je abstraktní třída, která rozšiřuje třídu RoomDatabase a funguje jako hlavní přístupový bod k databázi SQLite. Uvnitř třídy se specifikuje nastavení databáze (jaké obsahuje entity nebo verze schématu DB) a obsahuje metody které referují na jednotlivé DAOs.

```
@Database(
    version = DbVersion,
    entities = [
        FieldOfStudyEntity::class,
        VideoEntity::class,
        EventEntity::class,
    ]
)
@TypeConverters(Converters::class)
abstract class Database : RoomDatabase() {

    abstract fun dodDAO(): DODDao

    companion object {
        const val DbName = "DODDatabase.db"
    }
}

private const val DbVersion = 1
```

Obrázek 26: ukázka definice databáze – kód mobilní aplikace [autor práce]

Při přidávání nových nebo změnách stávajících funkcí v aplikaci je třeba přizpůsobovat databázové tabulky, aby reflektovali změny v jednotlivých třídách entit. Taková změna často vyžaduje změnu schématu databáze. Zároveň je důležité zachovat uživatelská data, která jsou již v databázi zařízení uložena. K řešení této problematiky knihovna Room podporuje automatické i manuální migrace schématu DB. (36)

SQLite podporuje pouze čtyři primitivní datové typy: INTEGER, REAL, TEXT a BLOB (37). Při ukládání složitějších datových sktruktur lze s pomocí anotace @TypeConverter vydefinovat metodu, která ve svém těle popisuje převod na primitivní datový typ. Každá metoda konvertoru by měla přijímat 1 parametr a mít návratový typ.

```
class Converters {

    @TypeConverter fun fromDegreeType(value: DegreeType) = value.name
    @TypeConverter fun toDegreeType(value: String) =
        enumValues<DegreeType>().find { it.name == value }!!
```

Obrázek 27: ukázka použití @TypeConverter – kód mobilní aplikace [autor práce]

Další velkou výhodou knihovny Room je podpora reaktivního programování a propojení s dalšími komponentami Android Jetpack, jako jsou *LiveData* nebo *ViewModel*. Díky tomu je možné automaticky propsat změny dat v databázi až do UI vrstvy aplikace.

```
@Dao
interface DODDao {

    @Query("Select * From EventEntity")
    fun events(): Flow<List<EventEntity>>

class DODViewModel(
    private val repo: DODRepo,
) : ComposeViewModel() {

    private val eventType = MutableStateFlow(UniversityEventType.PRESENTATION)
    val events = repo.events.combine(eventType) { events, type ->
        type to events.filter { item -> item.type == type }
    }

@Composable
fun EventScreen(
    viewModel: DODViewModel,
    openEvent: (EventEntity) -> Unit,
) {
    val events = viewModel.events.collectAsState(
        initial = UniversityEventType.PRESENTATION to emptyList(),
    )

content = { innerPadding ->
    LazyColumn(
        contentPadding = PaddingValues(
            top = innerPadding.calculateTopPadding(),
            bottom = 80.dp,
        ),
    ) { this: LazyListScope
        if (events.value.second.isNotEmpty()) {
            items(events.value.second) { this: LazyItemScope item ->
```

5.3.7 Koin (DI)

Koin je jednoduchý framework pro dependency injection (DI). Je navržen tak, aby zjednodušil správu závislostí a tím zlepšil čitelnost kódu. Framework je napsaný primárně pro Kotlin, přičemž využívá DSL buildery a extensions, díky čemuž poskytuje stručnou syntaxi při deklarování závislostí. Koin používá ke správě a řešení závislostí takzvaný registr služeb (service registry). Pokud je závislost vyžádána, Koin nahlídne do svého registru a vrátí instanci požadovaného typu závislosti. (38)

```
private fun Module.db() {
    single { this: Scope  it: ParametersHolder
        Room.databaseBuilder(
            androidApplication(),
            Database::class.java,
            DbName,
        ).build()
    }
    single { get<Database>().dodDAO() }
}
```

Obrázek 28: definice singleton závislosti pro DB – kód mobilní aplikace [autor práce]

V Koinu jsou závislosti uspořádány do modulů, což mohou být jednoduché třídy, které definují sadu souvisejících závislostí. Moduly se také vytvářejí pomocí DSL knihovny. Klíčové slovo *module* se používá k vytvoření modulu a klíčová slova *single* nebo *factory* se používají k vydefinování typu závislosti, takzvaného *scope*. Pokud je závislost typu *singleton*, Koin vrátí existující instanci nebo vytvoří novou, pokud již neexistuje. Jestliže je závislost typu *factory*, vytvoří Koin při každém požadavku novou instanci.a

Po vydefinování modulů je třeba Koin zinicializovat společně s vytvořenými moduly, aby bylo možné sestavit graf závislostí. To se obvykle u aplikací pro Android provádí ve hlavní spouštěcí třídě *Application*.

```
open class App : Application() {

    override fun onCreate() {
        super.onCreate()

        startKoin { this: KoinApplication
            androidContext(applicationContext)
            androidLogger(if (BuildConfig.DEBUG) Level.DEBUG else Level.NONE)
            modules(appModules)
        }
    }
}
```

Obrázek 29: App main class – kód mobilní aplikace [autor práce]

DSL funkce *startKoin* inicializuje Koin s poskytnutými moduly a umožní mu spravovat vydeklarované závislosti.

Poté je již vše nastaveno, aby bylo možné využívat vkládání závislostí napříč aplikací. Koin poskytuje hned několik extension funkcí pro injektování a řešení závislostí v kódu aplikace. V aplikacích pro systém Android můžeme použít funkce *inject* a *viewModel*. Zatímco *inject* je univerzální mechanismus pro vkládání závislostí libovolného typu, *viewModel* je speciálně navržen pro vkládání instancí *ViewModelu* s integrovanou podporou správy životního cyklu UI komponenty.

```
class DODRepo : KoinComponent {

    private val api by inject<DODApi>()
    private val dao by inject<DODDao>()
```

Obrázek 30: použití vkládání závislostí – kód mobilní aplikace [autor práce]

5.3.8 Coil (Načítání obrázků)

Coil je moderní knihovna pro načítání obrázků napsaná v jazyce Kotlin a určená pro Android aplikace. Zkratka je akronymem pro Coroutine Image Loader, což zdůrazňuje využití coroutines pro asynchronní načítání a zpracování obrázků. Tato knihovna bylo navržena primárně s ohledem na rychlost a paměťovou nenáročnost. Používá kombinaci ukládání do mezipaměti a na disk, aby zlepšila rychlost načítání obrázků a zároveň snížila spotřebu dat přes síť. Zároveň umožňuje transformaci obrázků nebo snadné zobrazení zástupných symbolů, pokud je požadovaný obrázek nedostupný. (39)

```
AsyncImage(
    model = ImageRequest.Builder(LocalContext.current)
        .data(video.thumbnail)
        .crossfade( enable: true)
        .error(R.drawable.ic_fim)
        .build(),
    contentScale = ContentScale.Crop,
    contentDescription = video.url,
    modifier = Modifier
        .width(100.dp)
        .aspectRatio( ratio: 1F),
)
```

Obrázek 31: použití image RQ builderu pro načtení obrázku přes Coil – kód mobilní aplikace [autor práce]

5.3.9 Model-View-ViewModel (MVVM)

Model-View-ViewModel (MVVM) je architektonický vzor, který byl poprvé představen společností Microsoft pro použití ve Windows Presentation Foundation (WPF) a Silverlightu, později byl díky své jednoduchosti a efektivitě adaptován komunitou vývojářů Androidu. V systému Android se tak stal jedním z nejvíce používaných architektonických vzorů a je oficiálně podporován společností Google. Používá se zejména při použití v kombinaci s komponentou LiveData a dalšími komponentami ze skupiny Android Architecture Components (AAC). ACC je ofciální kolekce knihoven, která obsahuje komponenty zohledňující jejich životní cyklus. Například dokáže řešit problémy se změnami v nastavení, podporuje perzistenci dat, redukuje boilerplate kód, pomáhá předcházet únikům paměti a zjednodušuje asynchronní načítání dat do uživatelského rozhraní. (40)
Návrhový vzor MVVM rozděluje aplikaci na tři vzájemně propojené části: Model, View a ViewModel.

5.3.9.1 Model

Model představuje data a business logiku aplikace. Zahrnuje persistence vrstvu pro přístup k datům a data model třídy. Může také obsahovat validaci a různé agregační výpočty na daty. Model je nezávislý na uživatelském rozhraní a neví nic o tom, jak budou data zobrazena uživateli. (8)

5.3.9.2 View

View je vizuální vrstva aplikace, se kterou uživatel komunikuje, včetně rozvržení, animací a ovládacích prvků uživatelského rozhraní. View je zodpovědné za vykreslování dat Modelu a předává akce uživatele (například kliknutí na tlačítko) ViewModelu. V systému Android může být View aktivitou, fragmentem nebo vlastním View. (8)

5.3.9.3 ViewModel

ViewModel se nachází mezi Modelem a View a poskytuje jejich propojení. Cílem je zachování toho, že View a Model vrstva jsou na sobě naprosto nezávislé. ViewModel zpřístupňuje relevantní data pro View a komunikuje s Modelem za účelem

aktualizace těchto dat. (8) Například ViewModel vrstva dokáže zařídit to, že při otočení orientace obrazovky a překreslení prvků se data neztratí. Při správném návrhu je důležité myslet na to, že přes ViewModel neinteragujeme s View, ale manipuluje s Modelem v návaznosti na akce, které se dějí na View vrstvě.

5.3.10 Implementace MVVM

Chceme-li implementovat MVVM ve své Android aplikaci, vytvoříme vlastní ViewModel třídu, která dědí z `androidx.lifecycle.ViewModel` a obsahuje LiveData obejkty pro všechna data potřebná pro View vrstu.

```
class DODViewModel(
    private val repo: DODRepo,
) : ComposeViewModel() {

    private val _eventState = MutableStateFlow<LaunchState>(None)
    val eventState = _eventState.asStateFlow()

    fun fetchEvents() {
        launch(state = _eventState) { this: CoroutineScope
            repo.fetchAndSaveEvents()
        }
    }
    private val eventType = MutableStateFlow(UniversityEventType.PRESENTATION)
    val events = repo.events.combine(eventType) { events, type ->
        type to events.filter { item -> item.type == type }
    }
```

Obrázek 32: část třídy s implementací ViewModelu – kód mobilní aplikace [autor práce]

5.3.11 Material Design 3

Material You, často označovaný jako Material 3 (M3), je evolucí jazyka Material Design společnosti Google, který byl poprvé představen na konferenci Google I/O 2021. Jeho cílem je vytvářet personalizovanější a dynamičtější rozhraní a umožnit uživatelům vyjádřit svůj osobní styl prostřednictvím velké přizpůsobitelnosti prvků uživatelského rozhraní. (41)

V době implementace mobilní aplikace byl Material 3 stále aktivně vyvíjen a v projektu je použita jeho ranná alfa verze, která ale již fungovala bez problémů.

```
implementation("androidx.compose.material3:material3:1.0.0-alpha16")
```

Obrázek 33: nutná závislost v projektu pro využití nového M3 [autor práce]

5.3.11.1 Scaffold

Scaffold je hlavní komponenta ve frameworku Material Design 3, která poskytuje strukturované rozvržení hlavních součástí uživatelského rozhraní. Zejména zjednodušuje implementaci standardních rozvržení tím, že za nás organizuje pozicování hlavních komponent na obrazovce, jako je TopAppBar, FAB (Floating Action Button), Content a spodní Bottom Navigation Bar. (42)

Scaffold je použit jako základní komponenta pro všechny obrazovky v aplikaci.

```
Scaffold(
    modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection),
    topBar = {
        LargeTopAppBar(
            title = {
                Text(
                    text = stringResource(Routes.Contacts.label)
                )
            },
            colors = TopAppBarDefaults.largeTopAppBarColors(
                containerColor = MaterialTheme.colors.background,
                titleContentColor = MaterialTheme.colors.onBackground,
            ),
            scrollBehavior = scrollBehavior
        )
    },
    containerColor = MaterialTheme.colors.background,
    content = { innerPadding ->
```

Obrázek 34: použití Scaffold layoutu pro ContantScreen – kód mobilní aplikace [autor práce]

topBar – Je to horní komponenta Scaffoldu a obvykle obsahuje název aktuální obrazovky, navigační ikony a ovládací tlačítka.

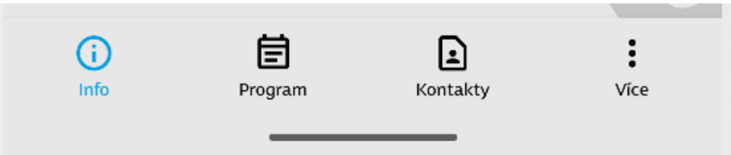
content – Jedná se o hlavní obsahovou oblast, která obvykle vyplňuje zbývající prostor v panelu Scaffold.

floatingActionButton (FAB) - Jedná se o defaultně kruhové tlačítko, které se „vznáší“ nad rozhraním a je určeno pro primární akci na dané obrazovce. Nicméně tento atribut přijímá jakoukoliv Composable komponentu, takže není problém kulaté tlačítko nahradit. Viz dále použití na obrazovce s přehledem programu.

bottomNavigationBar – Tento panel se nachází v dolní části panelu Scaffold a slouží k přepínání mezi jednotlivými obrazovkami. Používá se místo Draweru jako hlavní menu aplikace.

5.3.12 Menu Aplikace

Pro hlavní navigaci v aplikaci je použito menu ve spodní části aplikace. Při prvním spuštěním aplikace je defaultně otevřena položka info, která obsahuje obecné informace o fakultě a oborech. Pro implementaci menu byla použita komponenta *BottomNavigation*.



Obrázek 35: menu mobilní aplikace [autor práce]

Bottom Navigation je aktuálně velmi rozšíření navigační vzor, který umožňuje rychlý přístup k často používaným obrazovkám zobrazením sady ikon a popisků ve spodní části obrazovky. Balíček `androidx.compose.material` od Jetpack Compose obsahuje vestavěnou komponentu *BottomNavigation*, která zjednodušuje implementaci tohoto navigačního vzoru.

K implementaci této komponenty je třeba si vydefinovat seznam všech navigačních položek. Každá bude představovat jednu obrazovku v aplikaci a měla by mít jedinečný identifikátor, ikonu a popisek.

```
sealed class Routes(
    val icon: ImageVector,
    val route: String,
    val label: Int,
) {
    object Info : Routes(FimIcons.Info, DestinationInfo, R.string.info)
    object Programme : Routes(FimIcons.EventNote, DestinationProgramme, R.string.programme)
    object Contacts : Routes(FimIcons.ContactPage, DestinationContacts, R.string.contacts)
    object More : Routes(FimIcons.MoreVert, DestinationMore, R.string.more)

    companion object {
        fun values() = listOf(
            Info, Programme, Contacts, More
        )
    }
}
```

Obrázek 36: vydefinování položek menu pro *BottomNavigationItems* – kód mobilní aplikace [autor práce]

Následně je potřeba všechny položky pro iterovat a sestavit *BottomNavigationItem* komponenty. *BottomNavigationItem* představuje jednotlivé navigační položky v

rámci spodního navigačního panelu a přijímá několik modifikačních atributů, nejdůležitější jsou tyto:

icon – Lamba přijímající dva parametry, *image vektor* reprezentující ikonu a label, který reprezentuje popisek pod ikonou

selected – Jedná se o boolean hodnotu, která udává, zda je aktuální navigační položka vybrána, nebo ne

onClick – Jedná se o lambda výraz, který určuje akci, která se provede po kliknutí na navigační položku.

```
Routes.values().forEach { item ->
    val title = stringResource(item.label)
    BottomNavigationItem(
        icon = {
            Icon(
                imageVector = item.icon,
                contentDescription = title,
            )
        },
        label = {
            Text(
                text = title,
                fontSize = 10.sp,
                style = MaterialTheme.typography.body1,
            )
        },
        modifier = Modifier,
        selectedContentColor = MaterialTheme.colors.primary,
        unselectedContentColor = MaterialTheme.colors.primaryVariant,
        alwaysShowLabel = true,
        selected = currentRoute == item.route,
        onClick = {
            controller.navigate(item.route) { this: NavOptionsBuilder
                controller.graph.startDestinationRoute?.let { route ->
                    popUpTo(route) { this: PopUpToBuilder
                        saveState = true
                    }
                }
                launchSingleTop = true
                restoreState = true
            }
```

Obrázek 37: sestavení *BottomNavigationItem* komponenty – kód mobilní aplikace [autor práce]

To celé je možné zabalit do vlastní Composable komponenty. V našem případě BottomNav, kterou poté jen vložíme do skeletonu layoutu aplikace.

5.3.13 Informace o fakultě a přehled oborů

Obrazovka info je úvodní obrazovka zobrazená po otevření aplikace. Zobrazuje základní informace o fakultě a také seznam všech dostupný oborů na které se mohou uchazeči o studium zapsat. Tyto dvě sekce jsou reprezentované vlastní @Composable funkcí *CollapsibleSection*. Tato funkce vytvoří rozbalitelný oddíl s volitelnou ikonou a popiskem. Sekci lze kliknutím rozbalit nebo sbalit. Kliknutím na konkrétní položku se zobrazením obsahu s detailem, na který je aplikována animace s přechodem.

```
@Composable
fun CollapsibleSection(
    label: String,
    labelStyle: TextStyle = MaterialTheme.typography.h3,
    icon: ImageVector? = null,
    expanded: MutableState<Boolean>,
    content: @Composable () -> Unit,
) {

    //for animation of chevron icon
    val angle = animateFloatAsState(
        targetValue = if (expanded.value) -90F else 90F,
        animationSpec = tween(
            durationMillis = 250,
            easing = FastOutLinearInEasing,
        )
```

Obrázek 38: Vlasní compose funkce rozbalovacího listu – kód mobilní aplikace [autor práce]

label – Textový řetězec pro popisek sekce.

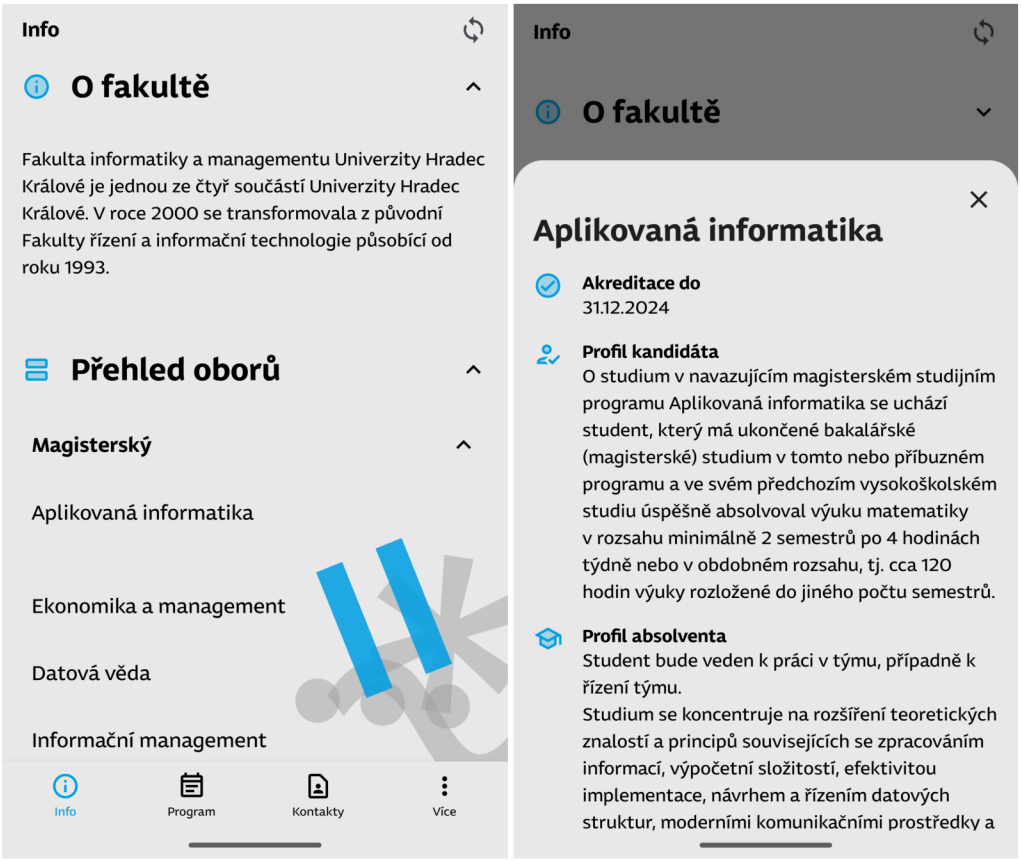
labelStyle – Objekt TextStyle pro stylování popisku. Ve výchozím nastavení je to typografie MaterialTheme h3.

icon – Volitelný objekt ImageVector pro ikonu, která se zobrazí vedle popisku.

expanded - Objekt MutableState<Boolean>, který vyjadřuje, zda je sekce rozbalená, nebo ne.

content – Composable funkce, která je obsahem dané sekce.

animateFloatAsState – Toto je animační funkce, která animuje úhel natočení ikony šipky ukazující směrem do prava. Úhel natočení je -90, když je sekce rozbalená, a 90, když je sbalená. Přechod mezi těmito dvěma stavy je animován po dobu 250 milisekund.

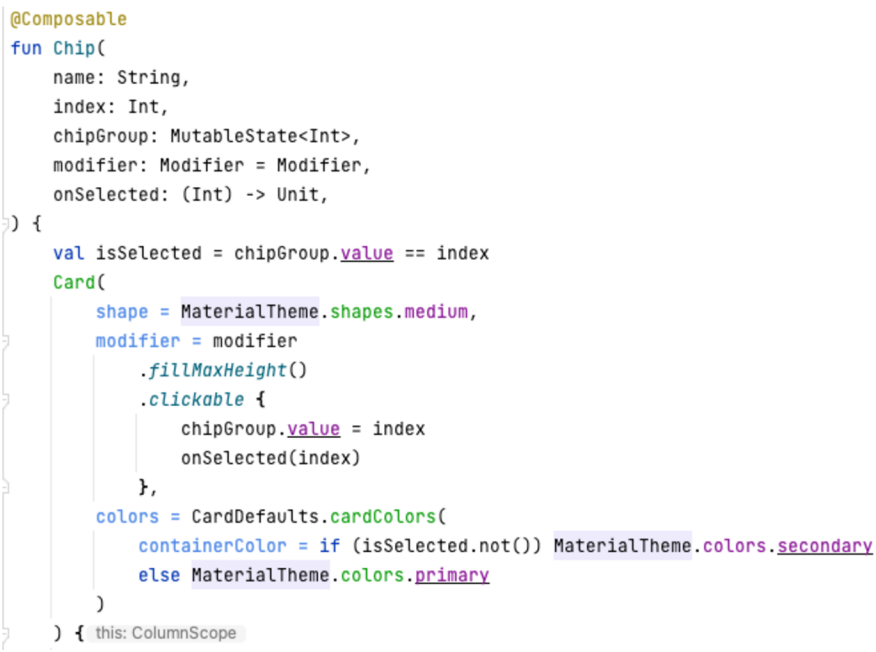


Obrázek 39: Obrazovka s přehledem jednotlivých oborů (nalevo) a obrazovka s detailem oboru (napravo) [autor práce]

5.3.14 Obrazovka s programem

Obrazovka program je vytvořena pro přehled všech událostí konaných během dne otevřených dveří. Události jsou roztřízeny do třech kategorií podle typu: přednášky, doprovodné akce a workshopy. Je možné si kliknout na jednotlivé události a zobrazit si tak jejich detail s popiskem, který se zobrazí zespodu obrazovky ve vyjíždícím *BottomSheetu*.

Zajímavým prvkem na této obrazovce je přepínač mezi typy událostí. Je sestaven jako vlastní composable Chips funkce. Takto sestavená komponenta se pak přiřadí do atributu *floatingActionButton* z komponenty Scaffold, která obaluje celou obrazovku.



```
@Composable
fun Chip(
    name: String,
    index: Int,
    chipGroup: MutableState<Int>,
    modifier: Modifier = Modifier,
    onSelected: (Int) -> Unit,
) {
    val isSelected = chipGroup.value == index
    Card(
        shape = MaterialTheme.shapes.medium,
        modifier = modifier
            .fillMaxHeight()
            .clickable {
                chipGroup.value = index
                onSelected(index)
            },
        colors = CardDefaults.cardColors(
            containerColor = if (isSelected.not()) MaterialTheme.colors.secondary
            else MaterialTheme.colors.primary
        )
    ) { this: ColumnScope
```

Obrázek 40: Implementace composable chipu – kód mobilní aplikace [autor práce]

Chip se skládá z několika částí:

name – Textový řetězec, který se má zobrazit na čipu.

index – Číslo, které představuje index čipu v rámci celé skupiny čipů.

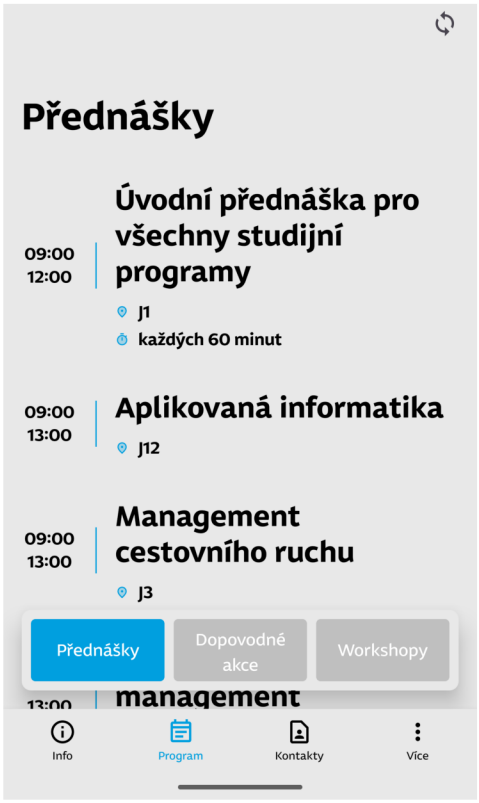
chipGroup - MutableState<Int>, který obsahuje index aktuálně vybraného čipu ve skupině.

modifier – Modifikátor, který lze použít k úpravě rozložení nebo vzhledu čipu.

onSelected – Funkce zpětného volání, která je spuštěna, když je čip vybrán. Jako parametr přijme index vybraného čipu.

isSelected – Boolean, který je true, pokud je tento čip aktuálně vybraným čipem ve skupině (tj. pokud se index tohoto čipu rovná hodnotě v chipGroup, která tento prvek obaluje).

card – Composable prvek z material design knihovny. Barva karty se mění podle toho zda je chip vybrán.



Obrázek 41: Obrazovka s přehledem jednotlivých událostí [autor práce]

5.3.15 Kontakty

Obrazovka kontakty zobrazuje přehledně informace o poloze sídla fakulty společně s adresou, otevírací dobu, adresu webových stránek a telefonní kontakt na studijní oddělení. Na jednotlivé informace jsou navázány systémové intenty, které se spustí po kliknutí na položku.

```
.clickable {
    val intent = Intent(Intent.ACTION_VIEW,
        Uri.parse( uriString: "http://maps.google.co.in/maps?q=" + address))
    try {
        activity.startActivity(intent)
```

Obrázek 42: Nastavení intentu pro zobrazení v mapách – kód mobilní aplikace [autor práce]

Po kliknutí na webovou stránku se otevře webový prohlížeč. Po kliku na adresu se zobrazí poloha fakulty v Google mapách. Pokud je v zařízení nainstalována Google Maps aplikace, tak se poloha otevře přímo v ní a je možno například spustit rovnou navigaci. Při interakci na telefonní číslo se otevře systémový dialog pro založení hovoru na dané číslo.

Obrázek 43: Obrazovka s kontakty [autor práce]

5.3.16 Seznam YouTube videí

Poslední obrazovka se používá pro výpis videí, které se týkají dne otevřených dveří a jsou nahrané na YouTube kanálu fakulty. Seznam obsahuje název videa, jeho popis a obrázek s náhledem videa. Po rozkliknutí se otevře přes intent url videa. Jestli je na zařízení instalována YouTube aplikace od Googlu, tak se video otevře přímo v ní. Pokud ne, otevře se odkaz ve webovém prohlížeči. Všechny tyto informace si mobilní aplikace stáhne ze serveru. Pro zobrazení miniatur se používá knihovna Coil popsána v kapitole 5.3.9. Pokud by nebyla miniatura s videem dostupná zobrazí se zástupný obrázek s FIM vzorem.

Obrázek 44: Obrazovka se seznamem videí [autor práce]

6 Výsledky, závěr

Během zpracování diplomové práce si autor zlepšil znalosti a získal spoustu zkušeností s programováním moderních mobilních aplikací s použitím JetPack Compose. Podařilo se použít komponenty z Material Designu 3, které v době implementace aplikace byli v rané fázi vývoje. Součástí práce se stala i implementace webové aplikace, která poskytuje data přes REST rozhraní mobilní aplikaci. Mobilní aplikace si umí stažená data lokálně uložit a může tak bez problému fungovat i bez neustálého připojení na internet. Webová aplikace umožnuje přes uživatelské rozhrání editovat jednotlivé události.

6.1 Shrnutí výsledků

V rámci této diplomové práce byly úspěšně vytvořeny dvě aplikace, mobilní aplikace pro návštěvníky dne otevřených dveří na FIM a administrační webová aplikace pro správu obsahu mobilní aplikace. Obě aplikace byly vytvořeny pomocí moderního programovacího jazyka Kotlin, což podtrhuje jeho univerzálnost a efektivitu v různých oblastech softwarového vývoje.

6.2 Doporučení

Pokud by se nápad s mobilní aplikací uchytil dala by se rozšířit o další sekce. Například by se mohl přidat modul s in-door navigací po budově fakulty, který by studenta dovedl na konkrétní místo konání eventu. Dále by se dala aplikace využít pro sběr zpětné vazby od uchazečů, kteří se dne otevřených dveří účastnili. Jednoduše po skončení události by uživatelům přišla notifikace se žádostí o vyplnění dotazníku, který by se po vyplnění odeslal na server, kde by se následné odpovědi mohli vyhodnotit. Pro nalákaní uchazečů do aplikace, by se dali rozvěsit po fakultě QR kódy pro stáhnutím aplikace. Také by bylo vhodné vytvoř aplikaci i pro platformu iOS, aby mohl aplikaci používat kdokoliv. Pokud by se webová aplikace pro administraci nasadila na veřejně dostupnou adresu bylo by webovou stránku schovat za přihlášení.

7 Seznam použité literatury

1. **Android Developer Doc.** ***Android Developer.*** **[Online] [Citace: 26. 4 2023.] https://developer.android.com/.**

2. Spring Boot Reference Documentation . ***Spring Boot .*** **[Online] [Citace: 26. 4 2023.] https://docs.spring.io/spring-boot/docs/3.0.6/reference/htmlsingle/.**

3. KVision Doc . ***KVision Doc .*** **[Online] [Citace: 26. 4 2023.] https://kvision.gitbook.io/kvision-guide/.**

4. Dny otevřených dveří 2023. ***Úvodní stránka Fakulta informatiky a managementu.*** **[Online] [Citace: 22. 4 2023.] https://www.uhk.cz/cs/fakulta-informatiky-a-managementu/prijimaci-zkousky/dny-otevrenych-dveri-2023.**

5. Android a iOS již mají 99,9 % podíl, ostatní systémy jsou mrtvé. ***mobilizujeme.*** **[Online] [Citace: 23. 7 2022.] https://mobilizujeme.cz/clanky/android-a-ios-jiz-maji-999-podil-ostatni-systemy-jsou-mrtve.**

6. Podíl na trhu Android a iOS: zveřejněny statistiky za rok 2022. ***Root Nation.*** **[Online] [Citace: 23. 7 2022.] https://root-nation.com/cs/ua/news-ua/it-news-ua/ua-android-ios-statistika-2022/.**

7. Global market share held by the leading smartphone operating systems. ***Statista.*** **[Online] [Citace: 23. 7 2022.] https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.**

8. Griffiths, Dawn and Griffiths, David. ***Head First Android Development: A Brain-Friendly Guide.*** **s.l. : O'Reilly Media, 2017. 9781491974056.**

9. The Evolution of Android. ***CellPhoneDeal.*** **[Online] [Citace: 23. 7 2022.] https://www.cellphonedeal.com/blog/the-evolution-of-android.**

10. Android life cycle of activity. ***Android life cycle of activity.*** **[Online] [Citace: 24. 4 2023.] https://www.javatpoint.com/android-life-cycle-of-activity.**

11. Meier, Reto a Lake, Ian. ***Professional Android .*** **místo neznámé : Wrox, 2018. 9781118949535.**

12. Application Sandbox: Android Open Source Project. ***Android Open Source Project.*** **[Online] [Citace: 15. Březen 2023.] https://source.android.com/docs/security/app-sandbox.**

13. App Signing. ***Android Developer.*** **[Online] [Citace: 23. 4 2023.] https://developer.android.com/studio/publish/app-signing#generate-key.**

14. JAKARTA EE WORKING GROUP: A VENDOR NEUTRAL COMMUNITY DEDICATED TO EVOLVING JAKARTA EE. ***jakarta EE.*** **[Online] https://jakarta.ee/about/working-group/.**

15. Späth, Peter. ***Beginning Jakarta EE: Enterprise Edition for Java: From Novice to Professional.*** **místo neznámé : Apress, 2019. 9781484250785.**

16. Walls, Craig. ***Spring in Action, Sixth Edition.*** **místo neznámé : Manning, 2022. 9781617297571.**

17. Dokuka, Oleh a Lozynskyi, Igor. ***Hands-On Reactive Programming in Spring 5.*** **místo neznámé : Packt, 2018. 9781787284951.**

18. Project Reactor reference documentation. ***Project Reactor.*** **[Online] [Citace: 28. 4 2023.] https://projectreactor.io/docs.**

19. Why KotlinJS is the future of web development. ***JetBrains Blog.*** **[Online] [Citace: 23. 7 2022.] https://blog.jetbrains.com/kotlin/2020/10/why-kotlin-js-is-the-future-of-web-development/.**

20. Asynchronous Programming in JavaScript. ***Mozilla Research.*** **[Online] [Citace: 23. 7 2022.] https://research.mozilla.org/files/2017/09/Asynchronous_Programming_in_JavaScript.pdf.**

21. Robert, Jaroš. KVision. ***KVision.*** **[Online] [Citace: 14. Březen 2023.] https://kvision.io/.**

22. Why Google Stores Billions of Lines of Code in a Single Repository. ***Communications ACM.*** **[Online] [Citace: 27. 4 2023.] https://cacm.acm.org/magazines/2016/7/204032-why-google-stores-billions-of-lines-of-code-in-a-single-repository/fulltext?mobile=t.**

23. Jaroš, Robert. KVision Guide . ***KVision Guide .*** **[Online] [Citace: 3. 14 2023.] https://kvision.gitbook.io/kvision-guide/.**

24. Data Binding. *KVision Guide.* [Online] [Citace: 24. 4 2023.] https://kvision.gitbook.io/kvision-guide/2.-frontend-development-guide/forms#data-binding.

25. PostgreSQL Documentation. *PostgreSQL.* [Online] [Citace: 23. 4 2023.] https://www.postgresql.org/docs/15/glossary.html.

26. PostgreSQL Documentation. *PostgreSQL extensions.* [Online] [Citace: 23. 4 2023.] https://www.postgresql.org/download/products/6-postgresql-extensions/.

27. Tabulator - Documentation. *Tabulator.* [Online] [Citace: 28. 4 2023.] https://tabulator.info/docs/5.4.

28. Tabulator Tables . *KVision Guide.* [Online] [Citace: 28. 4 2023.] https://kvision.gitbook.io/kvision-guide/3.-optional-ui-functionality-via-modules/tabulator-tables.

29. AndroidX Overview. *Android Developers.* [Online] [Citace: 26. 4 2023.] https://developer.android.com/jetpack/androidx.

30. AndroidX releases . *Android developers.* [Online] [Citace: 26. 4 2023.] https://developer.android.com/jetpack/androidx/versions.

31. Griffiths, Dawn a Griffiths, David. *Head first Kotlin.* místo neznámé : O'Reilly, 2021. 9781491996690.

32. Jetpack Compose Preview like a pro! *Medium.* [Online] [Citace: 27. 4 2023.] https://medium.com/mutualmobile/jetpack-compose-preview-like-a-pro-780a7cda52df.

33. State and Jetpack Compose. *Android Developers.* [Online] [Citace: 26. 4 2023.] https://developer.android.com/jetpack/compose/state.

34. Creating a client application . *Ktor Docs.* [Online] [Citace: 26. 4 2023.] https://ktor.io/docs/getting-started-ktor-client.html.

35. Serialization. *Kotlin Docs.* [Online] [Citace: 26. 4 2023.] https://kotlinlang.org/docs/serialization.html.

36. Save data in a local database using Room | Android Developers. *Android Developers.* [Online] [Citace: 25. 4 2023.] https://developer.android.com/training/data-storage/room.

37. https://www.sqlite.org/datatype3.html. *SQLite.* [Online] [Citace: 26. 4 2023.] https://www.sqlite.org/datatype3.html.

38. What is Koin? | Koin. *Koin.* [Online] [Citace: 26. 4 2023.] https://insert-koin.io/docs/reference/introduction.

39. Overview. *Coil.* [Online] [Citace: 26. 4 2023.] https://coil-kt.github.io/coil/.

40. Android Architecture Components . *Pro Android Dev Medium.* [Online] [Citace: 24. 4 2023.] https://proandroiddev.com/android-architecture-components-cb1ea88d3835.

41. Android Developer. *androidx.compose.material3.* [Online] [Citace: 26. 4 2023.] https://developer.android.com/reference/kotlin/androidx/compose/material3/package-summary.

42. Scaffold. *Jetpack Compose Playground.* [Online] [Citace: 26. 4 2023.] https://foso.github.io/Jetpack-Compose-Playground/material/scaffold/.

8 Přílohy

K práci je přiložen komprimovaný soubor, na kterém jsou uloženy všechny zdrojové kódy.

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

Jméno a příjmení:	Bc. Dušan Salay
Osobní číslo:	I1600893
Adresa:	Zahrádky 318, Dolní Bousov, 29404 Dolní Bousov, Česká republika
Téma práce:	Mobilní a administrační webová aplikace pro podporu dne otevřených dveří na FIM
Téma práce anglicky:	Mobile and admin web application for support open days on FIM
Jazyk práce:	Čeština
Vedoucí práce:	doc. Ing. Filip Malý, Ph.D. Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

1. Úvod
2. OS Android
3. Java EE (Jakarta EE)
4. Výběr vhodných technologií
5. Návrh a vývoj mobilní a webové aplikace
6. Výsledky, závěr
7. Literatura

Seznam doporučené literatury:

GRIFFITHS, Dawn a David GRIFFITHS. Head first Android development. 2nd edition. Beijing: O'Reilly, [2017]. ISBN 9781491974056.
Späth, Peter. Beginning Jakarta EE: Enterprise Edition for Java: From Novice to Professional.: Apress, 2019. ISBN 9781484250785.

Podpis studenta: Salay

Datum: 3. 5. 2023

Podpis vedoucího práce:

Datum: 3. 5. 2023