FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Multiplatformní mobilní aplikace pro rozvrh na FIT ČVUT v Praze
Student:	Oleksandr Petrov
Vedoucí:	Ing. Marek Suchánek, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Na FIT ČVUT v Praze existuje přívětivá webová aplikace Fittable pro procházení osobního rozvrhu, ale také rozvrhů místností a předmětů. Tato aplikace podporuje řadu filtrů a nastavení. Ačkoliv je webová aplikace responzivní, není zcela dobře použitelná na mobilních zařízeních a mobilní aplikace by tyto problémy mohla odstranit. Cílem této práce je tedy vytvořit takovou multiplatformní mobilní aplikaci pro Android a iOS s využitím technologie Kotlin Multiplatform Mobile.

- Analyzujte současné řešení Fittable ve formě webové aplikace, popište její možnosti a nedostatky především při použití na mobilních zařízeních.
- Popište specifika vývoje multiplatformních mobilních aplikací s využitím Kotlin Multiplatform Mobile (KMM). Dále popište důležité aspekty návrhu vhodného uživatelského rozhraní v mobilních aplikacích.
- Navrhněte vlastní mobilní aplikaci pro Fittable, která bude podporovat co nejbližší sadu funkcionalit jako webová aplikace. Současně dodržte konzistentní vzhled s webovou aplikací.
- Implementujte aplikaci dle návrhu s využitím KMM, další výběr technologií řádně zdůvodněte. Kód strukturujte a dokumentujte dle zvyklostí pro vybrané technologie.
- Výslednou aplikaci otestujte a zhodnoťte výhody oproti webové aplikace.
- Prozkoumejte možnosti distribuce aplikace skrze standardní kanály daných platforem a navrhněte další možnosti rozvoje.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 23. května 2023 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Multiplatformní mobilní aplikace pro rozvrh na FIT ČVUT v Praze

Oleksandr Petrov

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek, Ph.D.

10. ledna 2024

Poděkování

Rád bych poděkoval vedoucímu práce Ing. Marku Suchánkovi, Ph.D., za veškeré rady a odborné vedení během celého vývoje. Také bych chtěl poděkovat svým přátelům za velmi důležitou podporu, kterou mi poskytli během zpracování této bakalářské práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. ledna 2024

Odkaz na tuto práci

Petrov, Oleksandr. *Multiplatformní mobilní aplikace pro rozvrh na FIT ČVUT v Praze*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Abstrakt

Tato práce se zabývá vývojem multiplatformní mobilní aplikace pro správu rozvrhu na Fakultě informačních technologií ČVUT v Praze. Cílem práce byla analýza a implementace mobilní aplikace, která usnadní a zpříjemní studentům a zaměstnancům přístup k jejich rozvrhům. Pro dosažení těchto cílů byly použity moderní vývojové metody a frameworky, včetně multiplatformní technologie Kotlin Multiplatform Mobile. Výsledkem je uživatelsky přívětivá mobilní aplikace s ohledem na potřeby uživatelů identifikované během analýzy stávajícího systému. Aplikace nabízí velký prostor pro budoucí vývoj a snadné rozšíření.

Klíčová slova mobilní aplikace, Fittable, rozvrh, Android, iOS, Kotlin Multiplatform Mobile, Kotlin, Swift, Compose, SwiftUI

Abstract

This work focuses on the development of a multiplatform mobile application for schedule management at the Faculty of Information Technology at the Czech Technical University in Prague. The goal of the work was the analysis and implementation of a mobile application that will facilitate and enhance access to schedules for students and employees. To achieve these goals, modern development methods and frameworks were used, including the Kotlin Multiplatform Mobile technology. The result is a user-friendly mobile application, taking into account the needs of users identified during the analysis of the existing system. The application provides lot of space for future development and easy expansion.

Keywords mobile application, Fittable, timetable, Android, iOS, Kotlin Multiplatform Mobile, Kotlin, Swift, Compose, SwiftUI

Obsah

Seznam obrázků

Úvod

Náhled do rozvrhu je nezbytnou součástí života každého studenta a zaměstnance vysoké školy. Na Fakultě informačních technologií ČVUT v Praze se studenti potkávají s potřebou nahlížet do svých rozvrhů prostřednictvím existující webové aplikace Fittable. I přesto, že toto řešení nabízí širokou škálu funkcí, webová aplikace má své limity, zejména v optimalizaci pro mobilní zařízení.

V současné době skoro každý student nebo učitel na vysoké škole vlastní nějaké mobilní zařízení. Právě proto je mobilní aplikace důležitou součástí každého systému, který se zaměřuje na poskytování lepší uživatelské zkušenosti a tím pádem zvýšení celkové spokojenosti uživatele se systémem.

V tomto kontextu se nabízí vytvoření multiplatformní mobilní aplikace, která využije moderní vývojové technologie a poskytne uživatelům efektivní a intuitivní nástroj pro náhled do rozvrhu na hlavních mobilních platformách, jako jsou Android a iOS. Navíc díky využití Kotlin Multiplatform Mobile bude případně možné rozšířit podporu pro další platformy, jako jsou Desktop a Web, díky sdílenému kódu mezi platformami.

Cílem práce je navrhnout a vyvinout multiplatformní mobilní aplikaci pro Android a iOS s využitím Kotlin Multiplatform Mobile a prozkoumat užitečnost a efektivitu této nové multiplatformní technologie ve spojení s moderním způsobem vývoje na největších mobilních platformách. Výsledkem bude použitelná moderní aplikace, která bude určena pro studenty a zaměstnance FIT ČVUT.

Bakalářská práce je rozdělena do 6 kapitol, které popisují jednotlivé kroky analýzy, návrhu a implementace navržené práce. Analýza popisuje detailnější specifikace požadavků potřebných pro implementaci. Návrh a architektura podrobně popisují architekturu všech částí aplikace. Implementační část se zabývá realizací věcí navržených v předchozích kapitolách a podrobným popisem používaných knihoven. Testování se zaměřuje na popis testování kódu a požadavků na různých vrstvách aplikace, ať už jde o sdílenou logiku nebo UI testování. Distribuce probírá základní možnosti distribuce aplikace skrze standardní kanály, jako jsou Google Play pro Android a App Store pro iOS zařízení. Poslední kapitola zhodnotí výslednou aplikaci a nabídne další možnosti rozvoje.

Kapitola 1

Analýza

V této kapitole bude popsána současná verze webové aplikace, její nevýhody na mobilních zařízeních, popis potřebných API pro vývoj aplikace a budou probírány nejdůležitější technologie používané pro podporu sdílení kódu. Zároveň budou nadefinovány funkční a nefunkční požadavky a případy užití nezbytné pro implementaci výsledné aplikace.

1.1 Fittable webová aplikace

Fittable[1] představuje již existující webovou aplikaci, která je veřejně dostupná pro studenty a zaměstnance ČVUT. Aplikace poskytuje uživatelům širokou škálu možností, včetně nahlížení do vlastního rozvrhu, stejně jako přístup k rozvrhům jiných uživatelů, k nimž má daný uživatel oprávnění. Také umožňuje nahlížení do rozvrhu studoven a předmětů. [1]

Současná aplikace, s výjimkou autorizace, nabízí pouze jednu obrazovku, která obsahuje několik komplexních prvků rozhraní, což může být pro uživatele matoucí. Horní část obrazovky umožňuje výběr měsíce, roku nebo semestru podle preferencí uživatele. Neumožňuje ale přímý výběr konkrétního dne, a proto je nejprve nutné vybrat rok, měsíc a týden, a až poté je možné přejít na požadovaný den pomocí pohybu prstem (swipe gesto). Takový přístup je neobvyklý pro mobilní zařízení. [1]

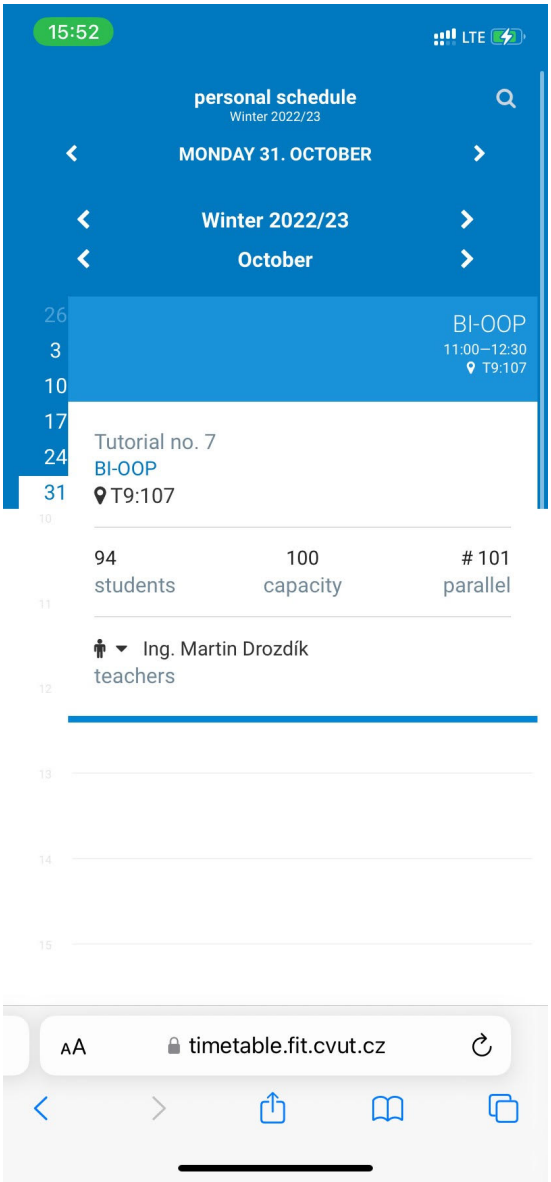
Webová aplikace také nemá výhody, jako je například lokální ukládání dat do zařízení uživatele pro nahlížení do rozvrhu bez přístupu k internetu. Nepodporuje také tmavý režim, což je velmi důležitou součástí moderní aplikace, jelikož kolem 82 procent všech uživatelů používá tmavý režim [11]. Během zkoušení aplikace se objevila situace, kdy po kliknutí na událost byl detail zobrazen mimo obrazovku, což může působit jako špatné UX. Občas po otevření události se grafické elementy překrývají na obrazovce a není vůbec jasné, co se děje 1.1. [1]

Aplikace rovněž umožňuje vyhledávání rozvrhů dalších uživatelů, místností nebo předmětů prostřednictvím vyhledávacího pole, kterým uživatel komunikuje se serverem přes API. Pokud uživatel má nezbytné oprávnění, zobrazí se

[1]Fittable je veřejně dostupná webová aplikace [10], jejíž repozitář se nachází na webové stránce GitLab [10].

požadovaný rozvrh. V opačném případě bude zobrazena informace, že uživatel nemá potřebný přístup.

Webová aplikace byla vyvíjena s cílem podporovat co největší škálu zařízení. Nicméně, při takovém postupu je těžké dosáhnout optimální uživatelské přívětivosti pro všechny typy zařízení. Z tohoto důvodu by bylo vhodné mít nativní aplikace pro každou platformu zvlášť, což umožní lépe optimalizovat uživatelské rozhraní.



Obrázek 1.1: Ukázka aplikace Fittable [1]

1.2 Sirius API

Sirius [12] je API pro správu rozvrhu na ČVUT FIT, které využívá KOS IS jako hlavní zdroj dat. Z perspektivy klientských aplikací je Sirius klasické REST API, které podporuje formát JSON. Přistupuje se k API pomocí autorizačního serveru Zuul, ke kterému by měl uživatel aplikace mít přístup, a to buď jako student nebo zaměstnanec ČVUT FIT. Autorizační server rovněž určuje oprávnění uživatele, a tím pádem stanovuje, k čemu má přístup. Například student nemůže nahlížet do rozvrhu jiných studentů, ale může do rozvrhu svého učitele [13].

Pro přístup k Sirius API je potřeba použít *OAuth 2.0* přístupové údaje z aplikace *Apps Manager*, jelikož všechny endpointy jsou chráněny autorizací. Příslušné aplikace musí obdržet oprávnění pro více tak zvanych *scopes*. Každý *scope* umožňuje přístup k určité omezené množině dat [14]. Principy OAuth 2.0 a podrobný popis aplikace *Apps Manager* budou dále popsány v následující sekci 1.3.

1.3 Apps Manager a OAuth 2.0

Apps Manager je aplikace, která poskytuje *Auth 2.0* oprávnění pro jiné aplikace, které využívají chráněné endpointy [15]. Auth 2.0 je moderní autorizační protokol, který je v současnosti standardem pro zabezpečení RESTových webových služeb. Hlavní výhoda této technologie spočívá v tom, že uživatel může poskytnout klientským aplikacím pouze přístup k jeho datům v nějaké konkrétní službě, aniž by poskytoval aplikaci své osobní údaje [14]. Tento způsob umožňuje mnohem bezpečnější přístup k datům uživatele, neboť klientská aplikace nemá přístup k soukromým údajům uživatele. Aplikace pak získá pouze tu podmnožinu dat, kterou jí poskytl sám uživatel.

OAuth 2.0 má několik rolí [14]:

- **Uživatel** – koncový uživatel nebo-li vlastník chráněného zdroje,
- **Služba** – služba, která obsluhuje požadavky,
- **Klient** – aplikace, která přistupuje ke chráněnému zdroji na resource serveru s oprávněními resource uživatele,
- **Autorizační server** – server, který klientovi vydává access token v případě jeho úspěšné autentizace od resource ownera (uživatele) a získání autorizace.

ČVUT FIT má vlastní *OAuth 2.0* server – ZUUL. Lze jej využít k získání tokenu od uživatele, což je nezbytné pro přístup k chráněným datům na Sirius API.

Během vytváření projektu v aplikaci *Apps Manager* a nastavování autorizace se vyskytly některé problémy:

1. I přesto, že *Apps Manager* má dokumentaci, chybí informace ohledně toho, co poskytují jednotlivé *scopes*. Během vývoje aplikace bylo nutné se obrátit na *helpdesk* kvůli nedostatku informací o tom, co dělají poskytované *scopes*, aby nakonec bylo možné přijít na množinu *scopů*, kterou aplikace potřebuje.

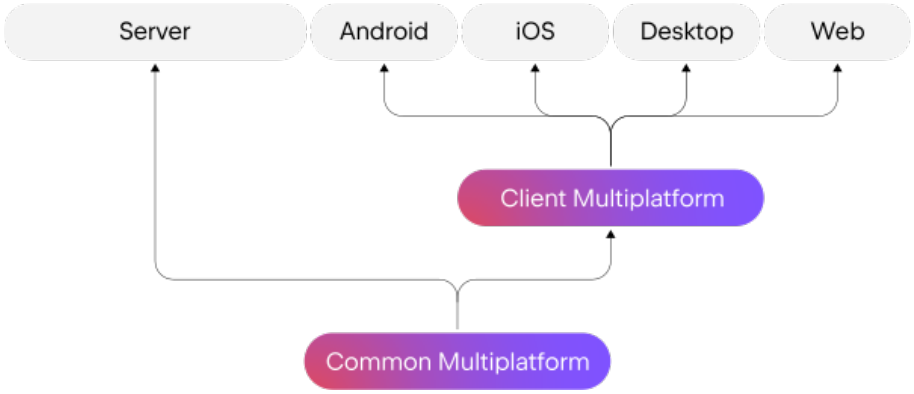
2. Chybí popis jednotlivých flow *OAuth 2.0. Apps Manager* má separátní flow pro mobilní zařízení, ale chybí jeho ukázka použití.

3. Autorizační server má několik autorizačních endpointů, konkrétně dva. První endpoint je starý a existuje pro něj dokumentace, na kterou ale není možné přistoupit z jejich webu. Pro druhý neexistuje žádná dokumentace. Z tohoto důvodu bylo pro implementaci zvoleno použít endpoint, ke kterému existuje dokumentace.

1.4 Multiplatformní aplikace s Kotlin Multiplatform

Kotlin Multiplatform (KM), jejíž zjednodušené znázornění je na obrázku 1.2, je zcela nová technologie, kterou vyvíjí společnost JetBrains. Hlavní cíl této technologie spočívá ve zjednodušení vývoje *cross-platform* projektů. Zaměřuje se na sdílení stejné části kódu mezi projekty, což obvykle zahrnuje byznysovou logiku, tak jak je stejná mezi platformy. Tímto způsobem zároveň udržuje flexibilitu a výhody nativního vývoje na různých platformách. V době psaní této práce pouze Kotlin Multiplatform Mobile (KMM) má stabilní verzi, což je verze tohoto frameworku určená pro mobilní aplikace. [2]

Další část sekce bude zaměřena na vysvětlení základních pojmů a principů fungování tohoto frameworku, které jsou nezbytné pro práci s ním a pro pochopení jeho důležitosti v současném světě mobilního vývoje.



Obrázek 1.2: Kotlin Multiplatform [2]

1.4.1 Nevýhody existujících multiplatformních frameworků

Největším problémem multiplatformních frameworků je sdílení části kódu které definujou uživatelské rozhraní. To je způsobeno tím, že právě tyto části se nejvíce liší mezi platformami. Je možné vytvořit aplikaci pomocí takových knihoven tak, aby vypadala podobně na různých platformách, ale taková aplikace neposkytne stejně dobrý *user-experience* jako nativní aplikace. Tyto frameworky také často mají mnohem horší výkon než nativní aplikace a podpora pro nové funkce bývá zpravidla zpožděná ve srovnání s nativními, protože je potřeba toto navíc naimplementovat v rámci příslušného frameworku. Samozřejmě existují

i výhody těchto frameworků, například rychlé vytvoření prototypů pro aplikaci, vývoj menších aplikací s omezeným rozpočtem a další [16].

1.4.2 Výhody oproti jiným frameworkům

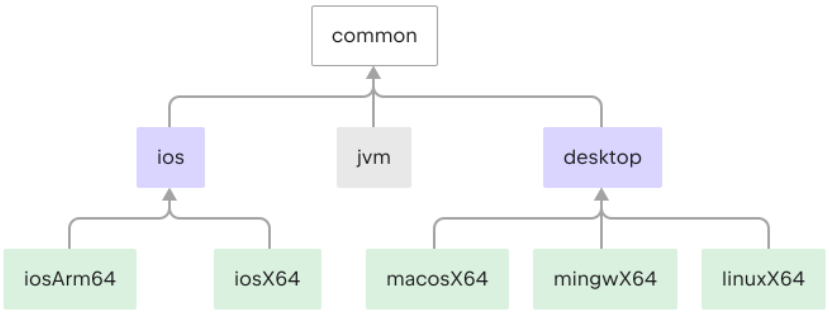
KM přichází s lepším přístupem v multiplatformním světě. Nabízí možnost sdílet pouze ty části aplikace, které jsou stejné na všech platformách pro stejný projekt, což je právě byznysová logika. Zbytek je možné nechat vyvíjet nativně. Takový přístup umožní ušetřit čas při psaní a údržbě stejného kódu mezi platformami a zároveň si ponechat výhody nativního vývoje. [17]

KM je flexibilní technologie a umožňuje zvolit, které části kódu bude vývojář sdílet. Buď jsou izolované malé části aplikační logiky, které jsou opakovaně používány mezi platformami, nebo je možné sdílet celou business logiku. Druhá varianta je ve většině případů nejlepší volba pro nové projekty, protože pro stejný projekt je tato logika stejná na všech platformách [2]. Je taky možné sdílet veškerý kód mezi různými platformami při použití další multiplatformní technologie, jako je *Compose Multiplatform*.

1.4.3 Sdílení kódu mezi platformami

Zjednodušená hlavní myšlenka sdílení kódu je znázorněna na obrázku 1.3. Kotlin Multiplatform umožňuje následující typy sdílení kódu mezi platformami:

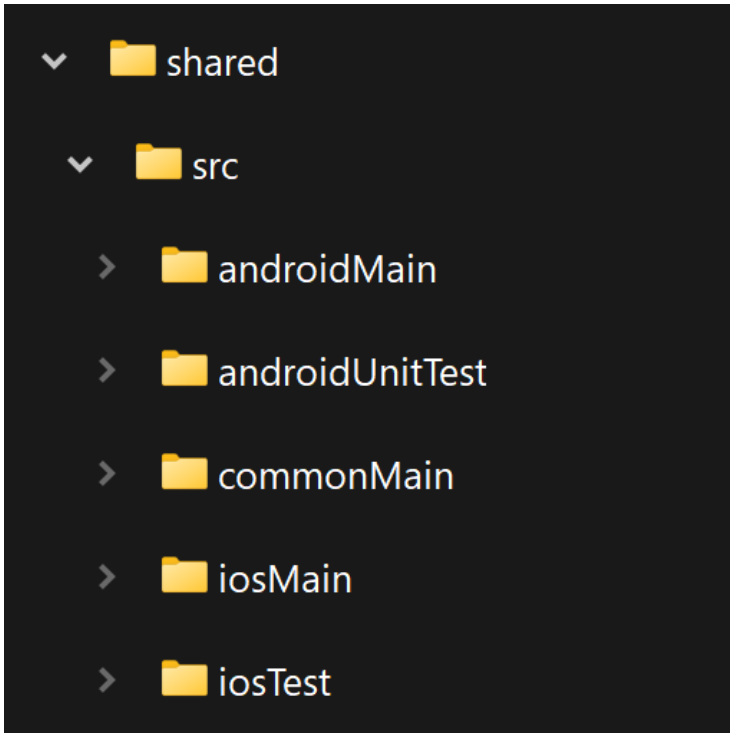
- Sdílení kódu mezi všechny platformy používané v rámci projektu.
- Sdílení kódu mezi některými platformami projektů, aby bylo možné znovu použít většinu kódu na podobných platformách.
- Přístup k API specifickým pro platformu ze sdíleného kódu pomocí očekávaných (expected) a skutečných (actual) deklarací.



Obrázek 1.3: Sdílení kódu [2]

1.4.4 Struktura projektu

V této podsekci bude ukázána základní struktura každého projektu v KM a budou vysvětleny její součásti. Základní kořenová struktura je znázorněna na obrázku 1.4.



Obrázek 1.4: Kořenový projekt [3]

1.4.4.1 Společný kód

Společný kód, neboli *Common code*, je kód, který je sdílen mezi různými platformami, typicky se nachází ve složce *commonMain*. Kompilátor získává zdrojový kód na vstupu, a výsledkem je množina binárních souborů specifických pro jednotlivé platformy [3]. V případě této práce budou výsledky pro Android platformu představovat .class soubory, a odpovídající soubory z iOS strany. Je důležité si uvědomit, že se zkompilují pouze ty části, které jsou multiplatformní. Pokud bychom například v *commonMain* zavolali nějakou funkci z JDK, dojde k chybě při kompilaci, protože na iOS platformě není tato třída dostupná.

1.4.4.2 Targets

Targets, neboli cílové platformy, definují množinu platforem, do kterých KM bude kompilovat veškerý kód umístěný ve společném kódu [18]. V případě této práce jsou to například Android a iOS. Kotlin Multiplatform také podporuje Desktop a Web, ale momentálně jsou pouze mobilní platformy považovány za *production-ready*.

1.4.4.3 Source Set

Source Set, nebo zdroj dat, je zdrojový kód, který má definovanou cílovou platformu a závislostí a je hlavním zdrojem KMM projektu [3]. V kontextu této práce budou vytvořené tyto zdroje: *iosMain*, *androidMain*, *commonMain*. Kde *commonMain* je multiplatformní zdrojový soubor s sdílenou logikou, zatímco *iosMain* a *androidMain* jsou specifické zdroje pro podporované platformy. *iosMain* a *androidMain* obsahují kód specifický pro dané platformy, například vytváření databáze nebo poskytování HTTP klienta.

1.4.5 Lint

Linting je proces spuštění statické analýzy na poskytovaném kódu. Pomáhá kontrolovat kvalitu kódu, umožňuje odhalit a opravit potenciální problémy a dodržovat určité standardy při psaní kódu [19]. Dodržování kvality kódu v projektu je jedním z nejdůležitějších úkolů při psaní a udržování kódu.

KMM, jelikož je napsaná v jazyce Kotlin, podporuje Kotlin lint nástroje, nejpopulárnější jsou:

- **Ktlint** – lehký nástroj pro analýzu, hlavní zaměření je dodržení konvencí kódu, umožňuje psát vlastní pravidla, ale nejsou moc flexibilní. Je snadno integrovatelný přes Gradle.
- **Detekt** – komplexnější nástroj pro statickou analýzu, má velkou sadu standardních pravidel, pokrývá širokou škálu problémů, umožňuje psát vlastní komplexnější pravidla. Je snadno integrovatelný přes Gradle.

V bakalářské práci budou využity výhody nástroje *Detekt*, jelikož integrace obou knihoven je zcela snadná, ale *Detekt* pokrývá větší sadu problémů, které jsou kritické pro zajištění dobré kvality výsledného kódu. Veškerý sdílený a Android kód bude analyzován pomocí výše zmíněného nástroje.

1.4.6 Testování

Pro testování multiplatformního kódu platí úplně stejná pravidla jako při testování Kotlin kódu. KM umožňuje testovat jak sdílené části kódu, tak i části specifické pro platformu [20]. Pro testování sdíleného kódu je nutné používat pouze multiplatformní knihovny. Pro testování určitého zdroje dat je možné bez další konfigurace vytvořit samostatný zdroj, například pro *commonMain* testovací zdroj je *commonTest*, pro *androidMain* – *androidTest*.

Několik důležitých testovacích frameworků, které podporuje KMM [21, 22]:

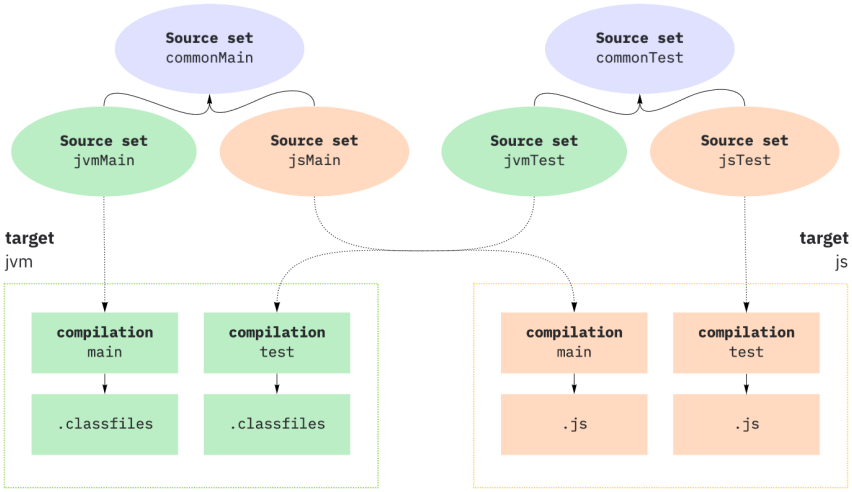
- **JUnit** – klasický testovací framework pro Java a Kotlin, je plně funkční a podporovaný v KMM projektech. Nabízí velkou sadu funkcí pro psaní a spouštění testů.
- **Kotest** – Moderní framework napsaný pro jazyk Kotlin, nabízí velkou sadu testovacích možností, umožňuje psát testy v různých stylech, který více vyhovuje vývojářům. Podporuje nejenom testování sdílených částí kódu, ale i specifického kódu pro platformu.
- **XCTest** – používá se na testování iOS, je nativním frameworkem pro psaní testů ve Swift, poskytuje velkou sadu testů pro iOS platformu.

Pro testování výsledné aplikace tato práce využije výhody, které nabízí testovací framework Kotest. Kotest poskytuje stejnou funkcionalitu jako JUnit, ale je modernější a byl specificky navržen tak, aby vyřešil problémy, které JUnit může mít při testování Kotlin projektů.

1.4.7 Kompilace

Kotlin Multiplatform projekty využívají kompilaci k vytváření artefaktů 1.5. Každý tak zvaný *target* může mít několik kompilací, například pro produkční nebo testovací účely [23]. Standardně nakonfigurované kompilace jsou:

- *Main*, *Test* pro JVM, JS a Native.
- Samostatná kompilace pro každou variantu sestavení pro Android projekt (*build variant*).



Obrázek 1.5: Kompilace

Pro jiné účely je možné nakonfigurovat vlastní kompilaci, buď pro všechny kompilace v projektu, nebo pro nějaký konkrétní *target*. V této práci je to využito například pro konfiguraci *SqlDelight* na iOS platformě, konkrétně pro přidání určitých parametrů do Linkeru.

1.5 Specifikace požadavků

Tato sekce se zabývá definováním funkčních a nefunkčních požadavků, jejich prioritou a také specifikací případů užití.

1.5.1 Funkční požadavky

F1: Přihlášení způsobem, který je popsán v kapitole 1.3 pomocí fakultního účtu.

F2: Zobrazení osobního rozvrhu v pomocí denního způsobu zobrazení událostí s příslušnými údaji, jako je název události, čas konání, učebna, datum události.

F3: Detail události, který zahrnuje přesnější popis zvolené události podle dat z Sirius API, například kapacitu a obsazenost události.

F4: Výběr data, podle kterého se zobrazují příslušné události.

F5: Vyhledávání rozvrhu podle možností, které poskytuje Sirius API, konkrétně zobrazení rozvrhu učeben, předmětů nebo uživatelů, ke kterým má přístup aktivní uživatel.

1.5.2 Nefunkční požadavky

NF1: Přístupnost k osobnímu rozvrhu uživatele bez nutnosti přístupu k internetu, konkrétně zobrazení poslední stažené z Sirius API verze dat.

NF2: Aplikace musí podporovat Android a iOS zařízení a to od určených verzí popsaných v kapitole 5.

NF3: Podpora dark/light módu, zvoleného v systému uživatele, a to platí pro obě platformy iOS a Android.

1.5.3 Priority

Prioritizace požadavků podle specifikace MoSCoW:

Must: F1, F2, F3, F4, NF2, jelikož jsou nejdůležitější požadavky, bez nich aplikace nedává smysl a nelze říci, že zadání práce je splněno.

Should: F5, NF1, jsou to důležité požadavky, díky kterým velká část uživatelů potenciálně bude používat aplikaci. Je to nutné mít ve výsledné aplikaci, pokud je to možné.

Could: NF3, i když to potenciálně zpříjemní uživatelskou zkušenost, ale celkově to nezabrání v používání aplikace, je to pouze možnost navíc.

Won't: další funkce, které nebudou implementovány v rámci této práce, ale vhodné pro účely dalšího rozšíření aplikace, budou probrány v sekci 6.2.

1.5.4 Případy užití

Diagram případů užití je znázorněn na obrázku 1.6.

UC1: Po kliknutí na tlačítko na obrazovce s přihlášením, uživatel se bude moci přihlásit na bránu ČVUT, podle F1.

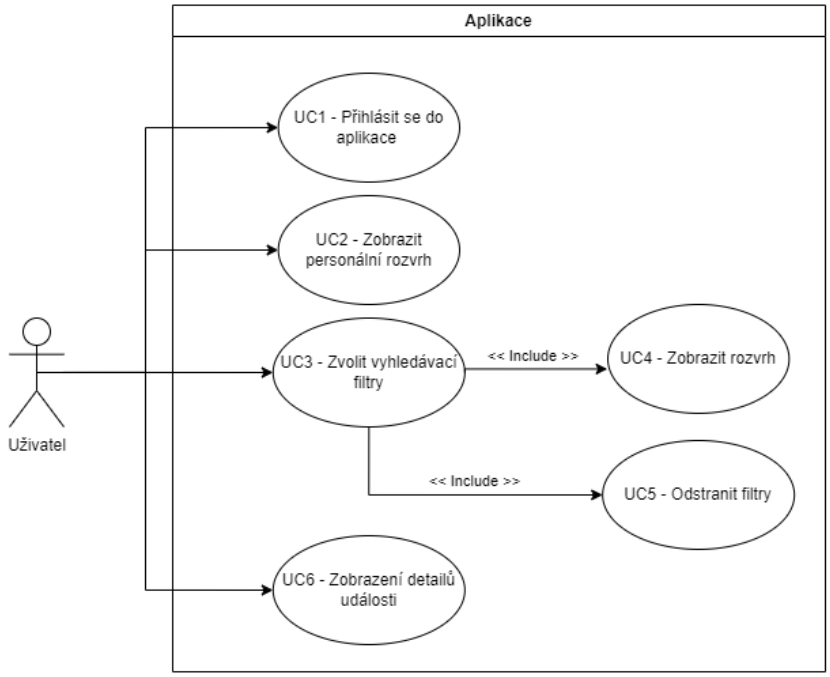
UC2: Zobrazení osobního rozvrhu podle F2.

UC3: Zvolení filtru rozvrhu na separátní obrazovce.

UC4: Zobrazení rozvrhu podle zvolených filtrů, požadavek F5.

UC5: Odstranění zvolených filtrů pomocí příslušného tlačítka na obrazovce s rozvrhem.

UC6: Zobrazení podrobných informací o události pomocí kliknutí na událost, podle požadavku F3.



Obrázek 1.6: Případy užití

Kapitola 2

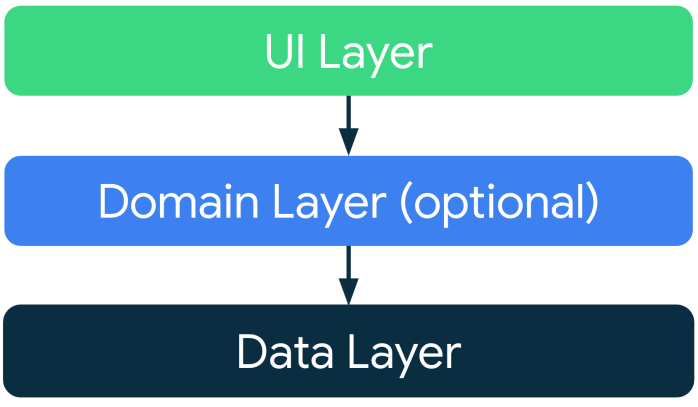
Návrh a architektura

Kapitola se zabývá popisem zvolené aplikační architektury do hloubky. Zde budou definovány vrstvy aplikace a jejich stručný popis. Také budou probírány části kódu, které se budou sdílet a důvody pro jejich sdílení.

2.1 Architektura

Pro zajištění lepší kvality výsledné aplikace v práci bude používána doporučená architektura od společnosti Google [4]. I když tato architektura není vždy vhodná pro všechny existující projekty, v tomto případě dokáže zajistit dobrou kvalitu kódu, oddělení zodpovědností, robustnost, škálovatelnost a poskytne možnosti testování výsledného kódu.

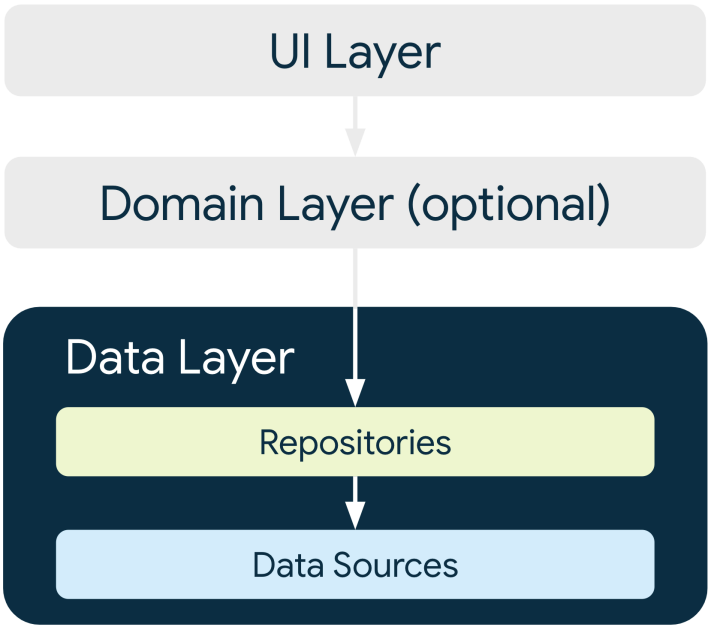
Je to zajištěno pomocí rozdělení kódu do několika vrstev – *doménovou*, *datovou* a vrstvu *uživatelského rozhraní* 2.1. *Datová* vrstva obsahuje byznysovou logiku a poskytuje uživatelskému rozhraní aplikační data. Vrstva rozhraní má za úkol dostat a zpracovat aplikační data, vytvořit z nich data vhodná pro uživatele a zobrazit je. *Doménová* vrstva je nepovinná a zapouzdřuje v sobě byznys logiku, která může být přepoužita ve vrstvě rozhraní.



Obrázek 2.1: Doporučená architektura od společnosti Google [4]

2.1.1 Datová vrstva

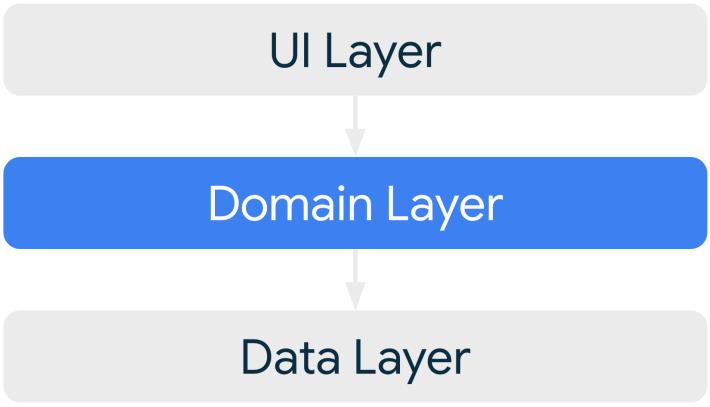
Datová vrstva většinou sestává z tzv. repozitáře (Repository) tříd, které v sobě drží a manipulují se zdroji dat 2.2. Jeden repozitář by měl odpovídat jednomu typu dat, který používá aplikace. V této práci bude potřeba vytvořit například *EventRepository*, *UserRepository*, *SearchRepository*. Tato vrstva má za úkol vystavovat data pro zbytek aplikace, řešit konflikty mezi několika zdroji dat [5], což jsou v této práci buď offline zdroj událostí nebo online zdroj událostí. Offline zdroj je napojen na lokální databázi na zařízení uživatele ve chvíli, kdy online zdroj je napojen na Sirius API 1.2, ale vyžaduje připojení k internetu.



Obrázek 2.2: Datová vrstva [5]

2.1.2 Doménová vrstva

Doménová vrstva má za úkol zapouzdření byznysové logiky, která bude přepoužita v UI vrstvě 2.3. Je nepovinná z toho důvodu, že ne každá aplikace má komplexní byznysovou logiku, nebo potřebuje přepoužívat ji na několika místech [6]. Doménová vrstva vyhovuje pro účely sdílení multiplatformního kódu, jelikož jediné, co bude potřeba vystavovat ze sdíleného kódu, budou doménové třídy, pak to lze jednoduše používat jak na Android platformě, tak i na iOS.



Obrázek 2.3: Doménová vrstva [6]

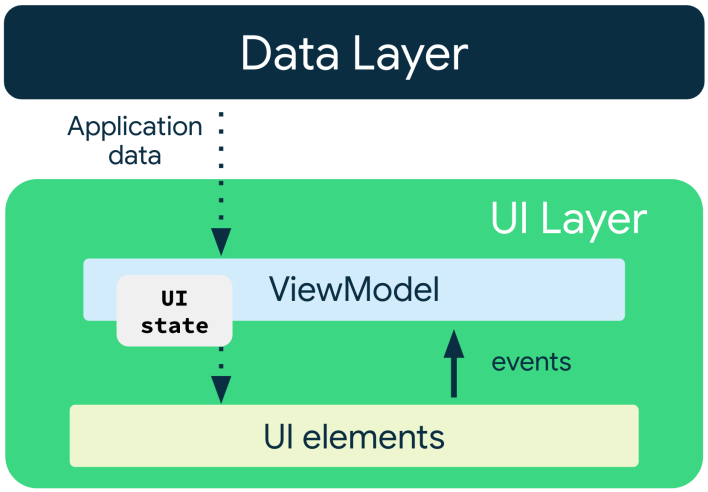
2.2 Sdílený kód

Pro sdílení kódu byla zvolena strategie sdílení veškeré byznysové logiky a vrstvy rozhraní nechat vyvíjet nativně. Takovým způsobem budou se sdílet části kódu, které jsou stejné, čímž lze ušetřit čas na vývoj a údržbu aplikace a zároveň zanechat výhody nativního vývoje. Z toho plyne, že sdílený kód bude rozdělen na dvě vrstvy – datovou a doménovou. Architektura uvnitř vrstev bude odpovídat již popsánému konceptu v kapitole 2.1. Pouze doménové třídy, neboli *UseCase* třídy, budou exponovane se sdíleného kódu. Zvolený přístup umožní zanechat čistý, testovatelný kód a vrstvy rozhraní dostanou veškerá nutná data pro další práci s nimi.

2.3 Vrstva rozhraní

UI vrstva má za úkol konverzi a zobrazení dat, které dostává z jiných vrstev, reagovat na změny dat v jiných vrstvách nebo na interakce od uživatele 2.4. Tato část většinou je implementována pomocí klasických UI architektonických vzorů, jako jsou MVP, MVC, MVVM, MVI a tak dále. Jelikož nejmodernější a doporučená architektura pro mobilní platformy je *MVVM* [7], z toho důvodu bude použita v práci pro definování vrstvy rozhraní.

Pro zobrazení grafických elementů budou používány deklarativní frameworky – *Jetpack Compose* pro Android a *SwiftUI* pro iOS. Tyto frameworky jsou podobné, takovým způsobem lze dodržet podobný styl kódu na obou platformách. Navíc tyto frameworky se aktuálně řadí mezi nejmodernější a v nových aplikacích se na ně postupně přechází.



Obrázek 2.4: Vrstva rozhraní [7]

2.4 Shrnutí architektury

Platformní implementace se bude skládat většinou pouze z odpovídajících *View Model* tříd, souborů, které definují grafiku na obrazovkách, a dalších pomocných tříd jako formátory a podobně. Zbytek logiky bude sdílen mezi platformami a přepoužit. Pro podporu další platformy, jako je například Desktop, bude potřeba ji nakonfigurovat pomocí Gradle v projektu a naimplementovat vrstvu rozhraní.

2.5 Minimální podporované verze

Tato kapitola se zabývá výběrem vhodných minimálních verzí podporovaných zařízení na Android a iOS platformách, nutných pro vývoj a distribuci.

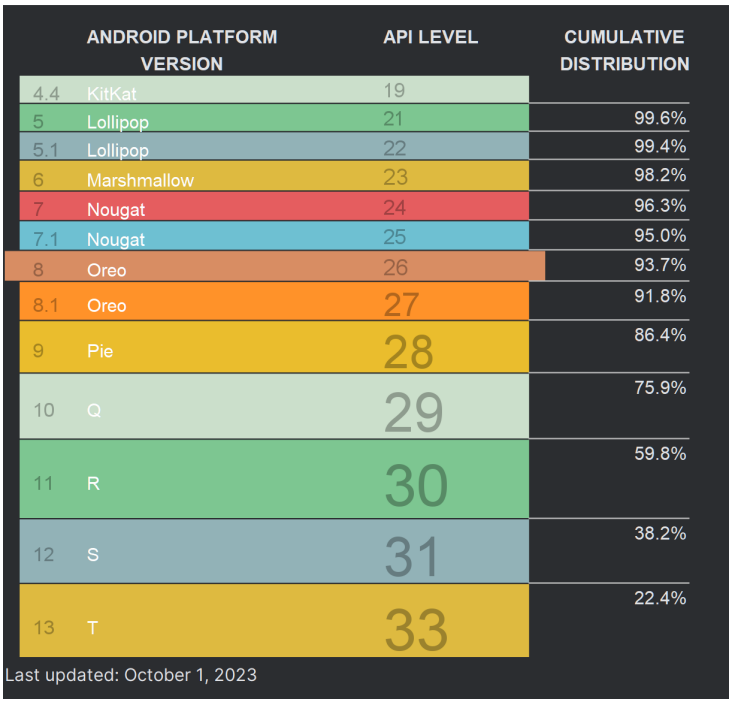
2.5.1 Android

Pro distribuci a vývoj aplikace je potřeba zvolit podporované verze Android zařízení. Na obrázku 2.5 je zobrazená kumulativní distribuce Android zařízení. Pro možnost použití moderních frameworků a technologií byla zvolena 8. verze Android. Jelikož se předpokládá, že typický uživatel je studentem nebo zaměstnancem technické vysoké školy na fakultě informatiky, je velká pravděpodobnost toho, že taková skupina uživatelů bude mít nadprůměrně vysokou verzi Android zařízení. Zároveň 8. verze umožní podporovat 93 procent všech existujících zařízení na trhu 2.5.

2.5.2 iOS

Stejná pravidla jako v kapitole 2.5.1 pro Android, platí i pro iOS platformu. Byla zvolena verze iOS 15 jako minimální, z toho důvodu, že podporuje 92,9

procent zařízení na trhu a zároveň nabízí moderní funkce, které umožňují lepší implementaci uživatelský přívětivé aplikace. Kumulativní podíl iOS distribuce je znázorněn na obrázku 2.6.



Obrázek 2.5: Android kumulativní distribuce [8]

Version	Released	Cumulative usage	Last iOS for
iOS 17	2023	4.9%	*Not applicable*
iOS 16	2022	82.4%	iPhone X, iPhone 8
iOS 15	2021	92.9%	iPhone 7, iPhone SE (gen 1), iPhone 6s
iOS 14	2020	95.4%	—
iOS 13	2019	97.0%	
iOS 12	2018	98.8%	iPhone 6, iPhone 5s
iOS 11	2017	99.1%	—
iOS 10	2016	99.4%	iPhone 5c, iPhone 5
iOS 9	2015	99.7%	iPhone 4s
iOS 8	2014	99.8%	—
iOS 7	2013	99.8%	iPhone 4
iOS 6	2012	99.9%	iPhone 3GS
iOS 5	2011	99.9%	—
iOS 4	2010	99.9%	iPhone 3G
iOS 3	2009	*No data*	iPhone (gen 1)
iOS 2	2008		—
iOS 1	2007		

Obrázek 2.6: iOS kumulativní distribuce [9]

2.5.3 Modularizace

Z toho důvodu, že výsledná aplikace není zásadně velká a neočekává se velké množství funkcí, bude vyvíjena v rámci jednoho modulu. Sdílená část bude samostatný modul, stejně jako moduly pro Android a iOS aplikace. Avšak v rámci těchto modulů se nepředpokládá žádná další modularizace, z výše zmíněných důvodů.

2.5.4 Sestavovací logika

Sestavení daného multiplatformního projektu vyžaduje velké množství procesů, které musí proběhnout před finálním sestavením a spuštěním aplikace. Proto byla využita výhoda automatického sestavovacího nástroje – Gradle. Přes tento nástroj byly nakonfigurovány všechny kroky a procesy, pomocí kterých se sestavuje výsledná aplikace. Veškerá sestavovací logika projektu bude shromážděna v předkonfigurovaném modulu *buildSrc*. Aby bylo možné takovou logiku používat a z důvodu dodržení čistého kódu v Gradle souborech, bylo využito takzvané *Gradle Convention Plugins*.

2.6 Obrazovky

Sekce se zabývá popisem obrazovek v aplikaci. Následuje krátký popis každé obrazovky a odkazy na příslušné obrázky prototypů.

Autorizace: Obrazovka, která po kliknutí tlačítka přesměruje uživatele na bránu ČVUT pro následnou autorizaci 2.7.

Zobrazení rozvrhu: Hlavní obrazovka, která má primární cíl zobrazit rozvrh uživateli 2.8.

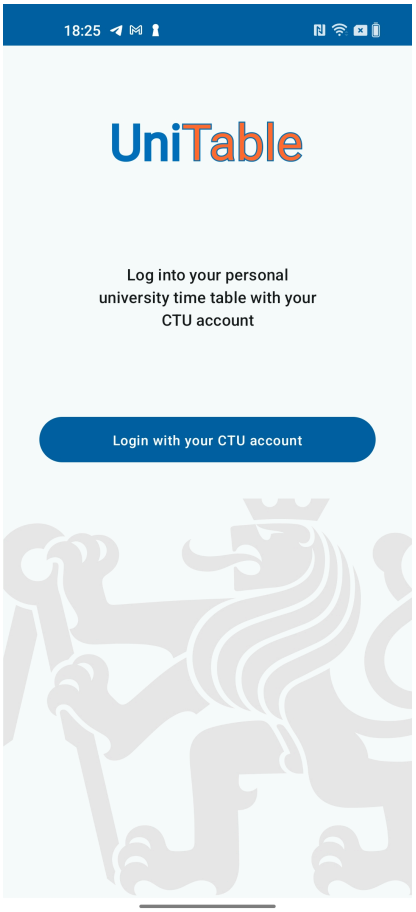
Kalendář: Část obrazovky v rámci zobrazení rozvrhu, umožňuje uživateli zvolit den a měsíc pomocí vhodných gest 2.9.

Vyhledávání rozvrhu: Obrazovka, která umožňuje uživateli zvolit filtr pro zobrazení rozvrhu 2.10 a 2.11.

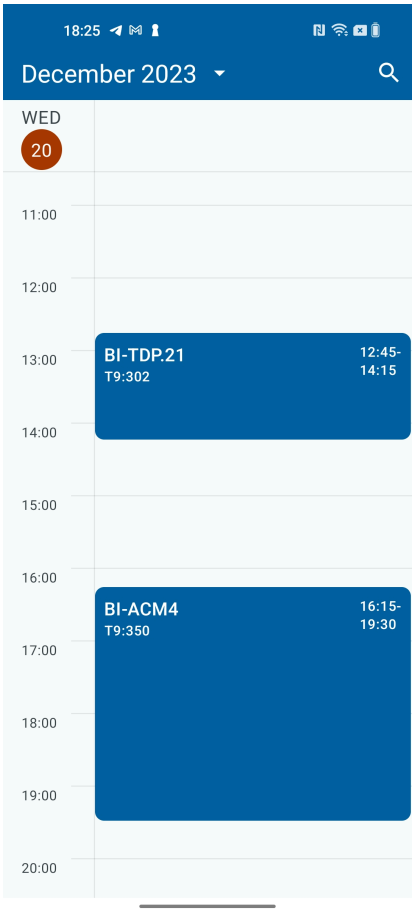
Detail události: Zobrazuje podrobnější popis události 2.12.

Tmavý režim: Ukázka, jak vypadá aplikace v tmavém režimu 2.13.

Filtrace: Zobrazení rozvrhu podle filtru 2.13.

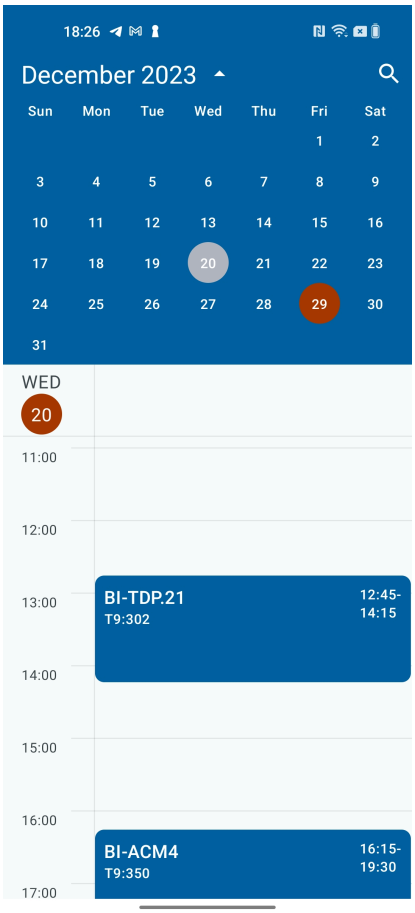


Obrázek 2.7: Autorizace

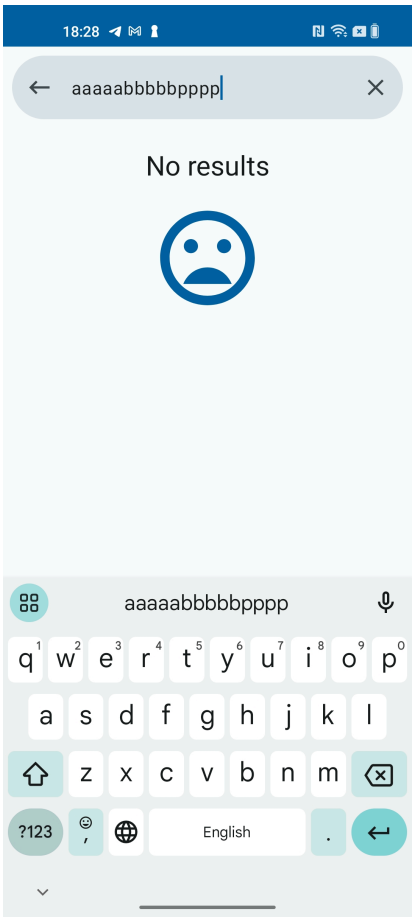


Obrázek 2.8: Zobrazení rozvrhu

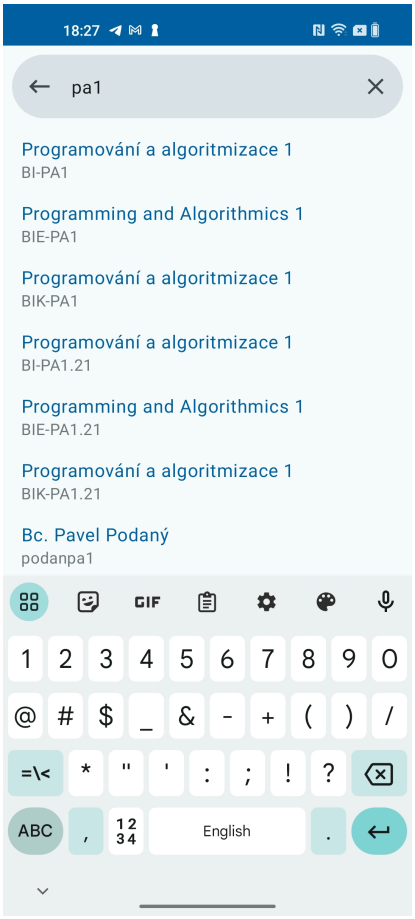
v



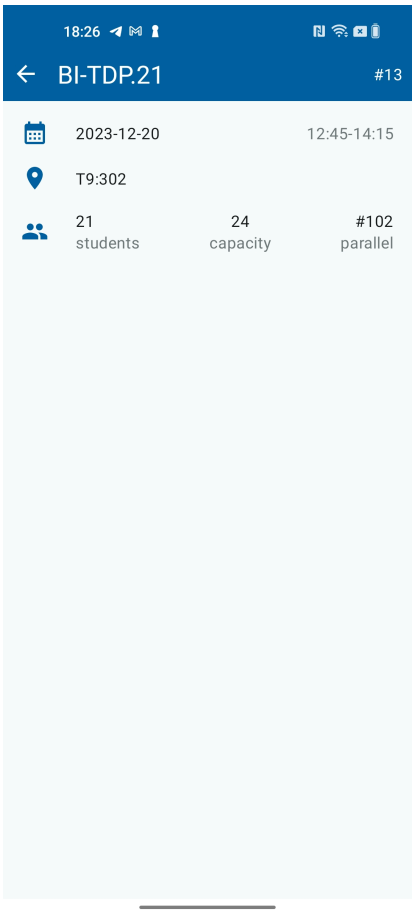
Obrázek 2.9: Kalendář



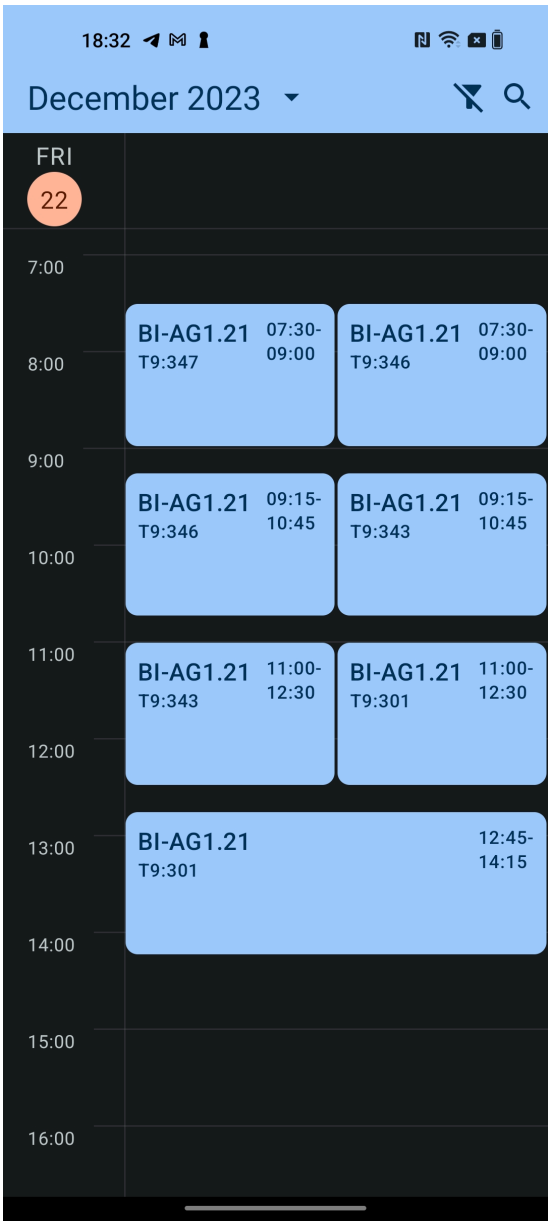
Obrázek 2.10: Vyhledávání, žádný výsledek



Obrázek 2.11: Vyhledávání rozvrhu



Obrázek 2.12: Detail události



Obrázek 2.13: Tmavý režim, rozvrh předmětu s aktivním filtrem

Kapitola 3

Implementace

Celá kapitola se zabývá popisem nejdůležitějších knihoven, které byly použity při vývoji aplikace, a také popisem rozhodnutí spojených s implementačním procesem.

3.1 Implementace sdílené logiky

V této sekci budou popsány nejdůležitější knihovny a do detailu rozebrány nejdůležitější části implementace sdíleného kódu.

3.1.1 Jazyk

Kotlin Multiplatform je technologie napsaná pro programovací jazyk Kotlin. Právě proto veškerý kód v sdílené části bude napsán pomocí jazyka Kotlin.

3.1.2 Databáze

Pro implementaci nefunkčního požadavku NF3, který je popsán v sekci 1.5, je potřeba použít databázi pro ukládání osobních událostí. Pro tyto účely bude využita databáze *SQLDelight*, která je multiplatformní a podporuje požadované platformy. *SQLDelight* generuje typově bezpečné Kotlin API z definovaných SQL příkazů. [24]. Pro vytvoření databáze na jednotlivých platformách je možné použít *expected/actual* deklarace:

```
// Android
actual class DriverFactory(
    private val context: Context,
) {
    actual fun createDriver(): SqlDriver {
        return AndroidSqliteDriver(
            TimetableDatabase.Schema.synchronous(),
            context,
            "TimetableDatabase.db",
        )
    }
}
```

```
// iOS
actual class DriverFactory {
    actual fun createDriver(): SqlDriver {
        return NativeSqliteDriver(
            TimetableDatabase.Schema.synchronous(),
            "TimetableDatabase.db"
        )
    }
}
```

3.1.3 DataStore

Pro ukládání malých objektů dat, jako je například token pro komunikaci s Sirius API, je lepší používat datové úložiště (DataStore) než databázi, jelikož ukládání obyčejného řetězce v samostatné tabulce je nonsens. Právě proto bude použita multiplatformní knihovna *DataStore* [25]. Toto řešení umožňuje ukládat objekty typu klíč-hodnota, používá na to formát *Protocol Buffers*. Tyto možnosti plně stačí pro uchování tokenu uživatele získaného z autorizačního serveru ČVUT. *DataStore* je potřeba vytvořit na každé platformě zvlášť:

```
// Android
fun dataStore(
    context: Context,
    dataStoreName: String,
): DataStore<Preferences> =
    createDataStore(
        producePath = {
            context.filesDir.resolve(dataStoreName).absolutePath
        },
    )
```

```
// iOS
@OptIn(ExperimentalForeignApi::class)
fun dataStore(
    dataStoreName: String,
): DataStore<Preferences> = createDataStore(
    producePath = {
        val documentDirectory: NSURL? =
            NSFileManager.defaultManager.URLForDirectory(
                directory = NSDocumentDirectory,
                inDomain = NSUserDomainMask,
                appropriateForURL = null,
                create = true,
                error = null,
            )
        requireNotNull(documentDirectory).path + "/$dataStoreName"
    },
)
```

3.1.4 Ktor

Ktor je framework pro správu HTTP připojení [26]. Je vhodný na použití v multiplatformních projektech. V této práci je použit pro dotazování na Sirius API. Je potřeba pro tento framework vytvořit HTTP klienta pro jednotlivé platformy. Konkrétně jsou vhodné OkHttp pro Android [27] a Darwin pro iOS [28] platformy. Vytvoreni HTTP klienta:

```
// Android
internal fun provideHttpClient() = HttpClient(OkHttp) {
    engine {
        config {
            retryOnConnectionFailure(true)
            connectTimeout(
                defaultConnectionTimeout.toJavaDuration()
            )
        }
    }
}
// iOS
internal fun provideHttpClient() = HttpClient(Darwin) {
    engine {
        configureRequest {
            setAllowsCellularAccess(true)
        }
    }
}
```

3.1.5 Service Locator

Pro správu závislostí ve sdílené části, a to platí i pro Android projekt, bude použita knihovna *Koin*. Jedná se o moderní a jednoduchou ve fungování knihovnu, která je napsaná v programovacím jazyce Kotlin a je plně multiplatformní [29]. Pro specifické platformní závislosti je možné použít *expected/actual* deklarace:

```
// Common Code
internal expect val platformModule: Module

// Android
actual val platformModule = module {
    // Zavislosti specificke pro platformu
}

// iOS
actual val platformModule = module {
    // Zavislosti specificke pro platformu
}
```

Pro poskytování závislostí ze sdíleného kódu ve Swiftu bude vytvořen jednoduchý kontejner, přes který budou vloženy potřebné závislosti:

```
object AuthorizationDependencyProvider : KoinComponent {
    val saveTokenUseCase: SaveTokenUseCase by inject()
}
```

3.1.6 Coroutines

Asynchronní programování je důležitou součástí každé mobilní aplikace, jinak UX nebude plynulý a příjemný. Tento problém řeší framework *Kotlin Coroutines* [30], který poskytuje podporu pro korutiny [31] na úrovni jazyka. Hlavní myšlenkou tohoto přístupu je, že existuje nějaká funkce, jejíž exekuce může být pozastavena a později pokračována. Konceptuálně je velmi podobná vláknu v tom smyslu, že vezme blok kódu ke spuštění, který běží souběžně se zbytkem kódu. Existuje však důležitý rozdíl: korutina není vázána na konkrétní vlákno, jelikož může pozastavit své provádění na jednom vlákně a obnovit ho na jiném [32]. Google doporučuje používat tento framework pro vývoj Android aplikací [30].

3.1.6.1 Flows

Asynchronous Flow je součástí frameworku *Kotlin Coroutines*, která je popsána v podsekci 3.1.6. *Suspend* funkce může vracet pouze jednu hodnotu, právě proto existuje takzvaný *Flow*, který umožňuje vracet několik asynchronně zpracovaných hodnot [33].

3.1.7 KotlinX Datetime

Pro manipulaci s časovými událostmi byla použita knihovna *KotlinX Datetime*. Je to jednoduchá v použití multiplatformní knihovna, plně napsaná v jazyce Kotlin a je implementována podle standardu ISO 8601. [34]

3.2 Android implementace

Tato sekce se zabývá knihovnami a implementací specifickou pro platformu Android.

3.2.1 Jazyk

Dva hlavní jazyky pro systém Android v současném světě jsou Kotlin a Java. Z toho důvodu, že celá sdílená část je napsána pomocí jazyka Kotlin, není důvod používat jazyk Java pro implementaci Android části. Navíc Kotlin je v současnosti doporučeným jazykem pro vývoj Android aplikací podle společnosti Google. [35]

3.2.2 AppAuth

AppAuth je SDK klient pro komunikaci s poskytovateli OAuth 2.0 a OpenID Connect [36]. Přímo mapuje požadavky a odpovědi těchto specifikací a zároveň dodržuje idiomatiky implementačního jazyka. Je vhodná pro komunikaci s autorizačním serverem ZUUL, který je popsán v sekci 1.3. Usnadňuje práci s protokolem OAuth 2.0 pro Android platformu.

3.2.3 Jetpack Compose

Pro implementaci uživatelského rozhraní byl zvolen moderní framework *Jetpack Compose*. *Compose* usnadňuje psaní a údržbu uživatelského rozhraní tím, že poskytuje deklarativní API, které umožňuje psát uživatelské rozhraní aplikace bez imperativního mutování hierarchických UI prvků [37]. *Compose* má samozřejmě své výhody a nevýhody oproti klasické hierarchické struktuře reprezentace UI. Například tento framework má srovnatelně horší *Runtime* výkon, navíc pokud napsaný kód je špatně navržen [38]. *Build time* je také srovnatelně horší [38], není to ale zásadně kritické, navíc s časem *Compose* se vyvíjí a postupně se ten rozdíl zmenšuje. Výhody zmíněného frameworku jsou mnohem větší, je možné psát celou aplikaci pouze v jazyce Kotlin bez použití XML, jak je to v situaci s klasickou *View* strukturou. Napsání UI komponentů je významně rychlejší a snadnější. Výsledná aplikace má méně kódu, což znamená, že její údržba je jednodušší a levnější. [39]

3.2.4 Navigace

Pro navigaci v rámci UI aplikace používá knihovnu *Navigation Compose*, kterou vyvíjí společnost Google. Tato komponenta se skládá ze tří nejdůležitějších komponent: *navigační graf*, *NavHost* a *NavController*. *Navigační graf* je centralizovaná informace o tom, jak se pohybovat v rámci aplikace. *NavHost* je kontejner, který zobrazuje určitou destinaci v rámci navigačního grafu. *NavController* je zodpovědný za správu navigace v rámci příslušného kontejneru. [40]

3.3 iOS implementace

Tato sekce se zabývá knihovnami a implementací specificky pro platformu iOS.

3.3.1 Jazyk

Pro vývoj produktu pro iOS prostředí je možné vybrat mezi jazykem Objective-C a Swift. Swift je mnohem modernější jazyk než Objective-C, podporuje různé moderní trendy a hlavně byl vyvinut pro zjednodušení vývoje aplikací napsaných v jazyce Objective-C. Právě z těchto důvodů bude použit pro implementaci iOS části. [41]

3.3.2 SwiftUI

SwiftUI je deklarativní UI framework pro vytváření grafických elementů pro iOS. Podporuje psaní v jazyku Swift. Je podobný jako *Compose* pro platformu Android, který je popsán v podsekci 3.2.3. Jiné možnosti jsou UIKit a AppKit. AppKit je nástroj pro vytváření aplikací pro sadu zařízení *Mac*, což tato aplikace nepotřebuje. UIKit je hierarchický framework podobný *Views* pro Android. Z toho důvodu, že *SwiftUI* je mnohem modernější a zcela podobný nástroji jako *Compose*, což umožní dodržovat skoro stejný kód na obou platformách, *SwiftUI* bude použit pro vytváření UI komponentů.

3.3.3 Combine

Combine je framework pro asynchronní zpracování hodnot. Používá tzv. *Publishers*, které produkují hodnoty, a *Subscribers*, které konzumují hodnoty. Protokol *Publisher* deklaruje speciální typ, který může s časem doručovat posloupnost hodnot. *Subscriber* okamžitě reaguje, když dostane nějaké změny od *Publisher*. V použití je docela podobný *Flow*, který je popsán v podsekci 3.1.6.1. [42]

3.3.4 OAuthSwift

OAuthSwift [43] je knihovna podobná AppAuth pro Android, která byla popsána v podsekci 3.2.2. Je jednoduchá na používání, napsaná v jazyce Swift, a podporuje OAuth 2.0, který je potřebný pro autorizaci na autorizačním serveru ZUUL.

3.3.5 Stinsen

Stinsen je knihovna pro navigaci uživatele mezi obrazovkami v iOS aplikaci. Implementuje doporučený vzor *Koordinátor* v SwiftUi, který byl představen Soroushem Khanlou na konferenci NSSpain v roce 2015 [44]. Knihovna deleguje zodpovědnost zpracování navigační logiky tak zvané *Coordinator* třídě. [45]

3.3.6 SKIE

Swift vyvolává Kotlin kód přes Objective-C rozhraní, z toho důvodu všechny důležité funkce jazyka Kotlin jsou vyloučeny. *SKIE* je knihovna, která má za cíl vyřešit podobný problém. Umožňuje používat nejdůležitější Kotlin funkce jako jsou *Sealed* třídy, *Suspend* funkce, *Coroutines*, *Flows*. *SKIE* vytváří pomocné *Wrapper* třídy pro Objective-C kód pro usnadnění komunikace mezi Kotlin a Swift kódem. [46]

SKIE podporuje sadu velmi užitečných funkcí [47]. Například pomocí *SKIE* lze jednoduše zavolat *Suspend* funkci z Kotlinu:

```
// Kotlin
class GetUserEventsUseCase {
    @Throws(CancellationException::class, ApiException::class)
    suspend operator fun invoke(...): List<EventDomain> {
        return ...
    }
}

// Swift
private func fetchEvents(...){
     Task{
        do{
            events = try await getEventsUseCase.invoke(...)
        } catch {
              ...
        }
    }
}
```

V aplikaci je používána pro volání sdíleného kódu, protože všechny funkce doménových tříd, které jsou vytvořeny pro použití ve vrstvách uživatelského rozhraní, jsou buď *Suspend* funkce nebo vracejí *Flows*. Tato knihovna pomáhá zmenšit hodně takzvaného *Boilerplate* kódu a obecně usnadňuje práci se sdíleným kódem.

3.4 Denní zobrazení událostí

Pro vytvoření vhodného denního zobrazení událostí, které je znázorněno na obrázku 2.8, nebyla nalezena žádná vhodná knihovna, která by dokázala mít vhodnou škálovatelnost a zároveň moderní UI. Z tohoto důvodu bude tato komponenta implementována ručně na každé platformě zvlášť. Rozhraní na každé platformě bude vytvořeno na základě doménových objektů, které vrstvy rozhraní obdrží ze sdíleného kódu.

Na obou platformách řešení bude vypadat podobně, struktura dané UI komponenty bude rozdělena na dva dynamické seznamy, které se překrývají, a při skrolování jednoho z nich bude pozice druhého automaticky synchronizována. Druhé řešení by mohlo být pomocí implementace tzv. vlastního rozložení (Custom Layout), takové řešení může potenciálně mít větší škálovatelnost při budoucím vývoji a lepší podporu animací při změně obsahu, ale čas potřebný na vývoj podobného rozhraní bude mnohem větší. Z ohledu na tuto informaci byla pro implementaci zvolena první nabízená varianta.

3.5 Gradle

Podle sekce 2.5.4 bude použit nástroj Gradle pro sestavení výsledného projektu. Pro dodržení čisté sestavovací logiky bylo použito *Gradle Convention Plugins*, které umožňují extrahovat sestavovací logiku do speciálních *Convention Plugins* a znovu používat mezi projekty. Nejprve je potřeba vytvořit takový *Plugin*, poté ho zaregistrovat v Gradle, a následně ho lze používat jako běžný *Gradle Plugin* [48]. Dále následuje ukázka vytvoření, registrace a použití jednoduchého *Gradle Convention Plugin*:

```
// Vytvoření
class ExamplePlugin : Plugin<Project> {
    override fun apply(project: Project) {
        // targetSdk, jiné pluginy
    }
}

// Registrace
gradlePlugin {
    plugins {
        register("example") {
            id = "fittable.android.example"
            implementationClass = "AndroidExampleConventionPlugin"
        }
    }
}
```

```
// Použití
plugins {
    id("fittable.android.example")
}
```

Pro správu verzí v rámci sestavovacího nástroje Gradle byla použita další funkce, kterou poskytuje Gradle. Tato funkce se jmenuje *Version Catalogues* [49]. *Version Catalogues* je způsob správy verzí, které jsou reprezentovány pomocí tzv. *Dependency Coordinates* [49]. Ukázka použití verzí pomocí tohoto nástroje:

```
dependencies {
    // Version Catalogues
    implementation(libs.kotlin.datetime)

    // Klasický způsob
    implementation("example:gradle-test:2.8.5")
}
```

Tento přístup má několik důležitých výhod oproti klasickému způsobu deklarace verzí:

- Pro každý katalog Gradle vytváří *Type-Safe Accessors*, aby bylo možné používat automatické doplňování v IDE.
- Lze vytvářet tzv. *Bundles*, které obsahují několik verzí najednou.
- *Version Catalogues* ukazuje na novější existující verze přímo v IDE.
- Všechny verze jsou na jednom jediném místě.
- Je možné *Version Catalogues* používat mezi dalšími projekty.

Pro používání *Version Catalogues* je potřeba přidat do souboru nastavení Gradle následující kód:

```
dependencyResolutionManagement {
    versionCatalogs {
        create("libs") {
            from(files("../gradle/libs.versions.toml"))
        }
    }
}
```

Kapitola 4

Testování

Tato kapitola se zabývá procesem testování výsledné aplikace.

4.1 Unit testování

Unit testy jsou jedním způsobem automatického testování funkcionality kódu. Pro implementaci *Unit* testů byl použit framework *Kotest*, způsob zvolení frameworku je popsán v sekci 1.4.6. *Kotest* je jednoduchý na použití a nabízí velké množství funkcionalit. Pro zjednodušení psaní *Unit* testů byla použita *mockovací* knihovna Mockk [50]. *Mokování* je vytvoření objektu, který imituje reálný objekt, ale jeho chování kontroluje vývojář [51].

Všechny nejdůležitější části kódu, konkrétně třídy obsahující byznysovou logiku, otestované pomocí *Unit* testů. Pro testování věcí, které vyžadují určité platformní závislosti, jako je například databáze, kde je potřeba mít vhodný pro platformu SqlDriver, bude testování provedeno v rámci platformního *Unit* testování.

Aplikace neobsahuje hodně obrazovek, proto budou UI vrstvy otestovány ručně na každé platformě zvlášť, jak během vývoje, tak i během uživatelského testování.

4.2 Uživatelské testování

Pro testování interakce uživatele s aplikací a pro UI testování bylo provedeno uživatelské testování. Poslední funkční verze aplikace pro platformu Android byla poskytnuta pěti studentům Fakulty Informačních Technologií, kteří aplikaci používali na denní bázi. Výsledky testování a zpětná vazba od uživatelů jsou důležitou součástí pro vývoj uživatelsky přívětivé aplikace. Obdržená zpětná vazba po uživatelském testování bude probrána v kapitole 6.

Pro testování aplikace na platformě iOS nebylo možné poskytnout aplikaci jiným studentům, protože by to vyžadovalo placený vývojářský účet. Z toho důvodu byla aplikace otestována v průběhu vývoje pomocí emulátoru, který poskytuje vývojové prostředí XCode.

4.3 Statická analýza

Pro účely zajištění kvality kódu byl v projektu použit nástroj pro statickou analýzu kódu *Detekt*, který je popsán v sekci 1.4.5. *Detekt* umožňuje provést statickou analýzu veškerého Kotlin kódu v projektu. Pravidla analýzy, kterou provádí *Detekt*, lze nastavit pomocí *.yml* souboru, který obsahuje veškerá pravidla, které aplikuje knihovna. Projekt většinou využívá standardní nastavení poskytované knihovnou, bylo však nutné provést drobné úpravy nastavení pro správné fungování s frameworkem *Jetpack Compose*, protože on obsahuje několik neobvyklých konstrukcí pro jazyk Kotlin.

Pro Swift kód nebyl použit žádný podobný nástroj z toho důvodu, že Swift kódu je v projektu málo, a nevyplatí se přidávat podobné nástroje. Je to potřeba zvážit v případě dalšího rozvoje aplikace.

Kapitola 5

Distribuce

V této kapitole budou probrány základní možnosti distribuce výsledné aplikace skrz standardní kanály pro podporované platformy.

5.1 Google Play

Google Play je v současné době nejpopulárnější platforma pro distribuci Android aplikací. *Google Play* má více než 2 miliony aktivních zařízení [52]. Skoro každé Android zařízení má tuto aplikaci standardně nainstalovanou v systému. Pomocí tohoto nástroje běžný uživatel může stahovat jakoukoli aplikaci, která je vydána na dané platformě.

Android aplikaci je možné distribuovat ve dvou formátech: buď *APK* (Android Package), nebo *AAB* (Android App Bundle). *APK* je spustitelný formát, který umožní nainstalovat určitou aplikaci na uživatelské zařízení. *AAB* je speciální distribuční formát, který má určité dynamické možnosti vytváření množiny *APK* pomocí speciálního nástroje Bundletool [53]. Pro podepsání *AAB* je potřeba použít speciální službu od společnosti Google, která se jmenuje Play App Signing [54], jelikož *AAB* spravuje *APK* sám podle potřeby.

Pro distribuci aplikace v této práci stačí vytvořit příslušné APK a manuálně ji vydat na platformu *Google Play*, není potřeba navíc využívat výhody, které poskytuje formát *AAB*.

Pro vydání Android aplikace na dané platformě je potřeba aplikaci předem připravit a udělat několik důležitých kroků [8]:

- Je potřeba sestavit tzv. *Release* verzi aplikace a podepsat ji.
- *Release* verze by měla mít vypnuté veškeré logování.
- Nad výslednou verzí aplikace, která bude vydána, je potřeba provést uživatelské testování.

Pro publikaci jakékoliv aplikace skrze *Google Play* je potřeba založit speciální vývojářský účet a projít verifikací. Stejně tak verifikaci musí projít i aplikace, kterou vývojář potřebuje vydat. [55]

V době psaní této práce společnost Google uplatnila nová pravidla pro distribuci aplikací skrze osobní účty vývojářů [56]. Pro nově vytvořené účty je potřeba najít 20 testerů, kteří budou nepřetržitě testovat aplikaci po dobu 14 dnů, což velmi komplikuje publikaci podobných aplikací.

5.2 App Store

App Store je jediná možnost publikace iOS aplikací pro veřejnost. Publikovat aplikace je možné pomocí vývojového prostředí Xcode od společnosti Apple [57]. Konfigurace výsledného sestavení aplikace je plně nastavitelná přes XCode. Podpis aplikace také provádí zmíněné vývojové prostředí. Pro publikaci je potřeba založit vývojářský účet a projít procesem verifikace, stejně tak je potřeba, aby aplikace prošla procesem verifikace po odevzdání žádosti o publikaci.

Aplikace musí odpovídat určitým náležitostem před odevzdáním ke kontrole na App Store:

- Aplikace musí odpovídat určitým směrnicím, které určuje společnost Apple [58].
- Před odevzdáním aplikace musí být funkční a otestovaná.
- Aplikace musí odpovídat technickým požadavkům, které určuje Apple.
- Nesmí to být kopie jiné existující aplikace, kterou se vývojář snaží prohlásit za svou.
- Aplikace nesmí obsahovat žádný nevhodný obsah.

5.3 Aktuální stav distribuce aplikací

I když publikace aplikací není součástí požadavků této práce, výsledné aplikace na platformách iOS a Android byly připraveny pro distribuci. Byly vytvořeny a připravené k publikování *Release* verze obou aplikací.

Android aplikace, v době odevzdání bakalářské práce, se nachází ve stavu interního testování na Google Play. Kvůli jejich novým podmínkám, popsaným v sekci 5.1, aplikace nemůže projít fází testování, jelikož se nepodařilo najít 20 testerů. Momentálně má aplikace 5 testerů, kteří jsou připraveni aplikaci používat a testovat, což však nevyhovuje požadavkům platformy.

iOS aplikace byla pouze připravena k distribuci a otestována, protože vyžaduje placený vývojářský účet, který je nutné nejprve zaplatit.

Kapitola 6

Zhodnocení a další rozvoj

Tato kapitola zhodnotí výslednou aplikaci, porovná funkcionality s existující aplikací Fittable a popíše další možnosti rozvoje na základě poskytnutého od testovatelů hodnocení.

6.1 Porovnání výsledné aplikace s Fittable

Vytvořená aplikace obsahuje nejdůležitější funkcionality, které poskytuje aplikace Fittable, s aplikováním vhodnějšího uživatelského rozhraní pro jednotlivé platformy. Výsledná aplikace zahrnuje všechny uvedené funkční a nefunkční požadavky, které byly popsány v sekci 1.5.1. Webová aplikace navíc má několik možností, které momentálně nemá její mobilní verze. Uživatel může například vybrat, zda chce zobrazovat ve svém rozvrhu přednášky, laboratoře nebo cvičení. Další funkcionalita, kterou mobilní aplikace nemá, je podpora českého jazyka. Momentálně je aplikace dostupná pouze v angličtině. Na druhou stranu, mechanismus podpory několika jazyků je přítomen v aplikaci, což znamená, že existující text v angličtině je potřeba pouze přeložit do češtiny a přidat do aplikace. Zbytek funkcí výsledná aplikace obsahuje, navíc má vylepšení navrhnute v sekci 1.5.1.

6.2 Další rozvoj

I přesto, že aplikace obsahuje veškeré důležité funkce pro pohodlné užívání, existujou další věcí, které by bylo vhodné přidat, aby se zvýšil počet potenciálních uživatelů.

Obecně byla aplikace považována všemi testery za dostatečně uživatelsky přívětivou a použitelnou, a žádné zásadní problémy při používání nebyly nalezeny. Někteří testeři se vyjádřili k tomu, že by bylo možné přidat několik užitečných vylepšení. Na základě jejich zpětné vazby byly vybrány nejdůležitější poznámky, které mohou být využity pro budoucí vývoj:

1. Přepínání dnů pomocí *Swipe* gesta. V současné době je pro přepínání dnů nutné otevřít kalendář a vybrat den. Pro zrychlení používání aplikace by bylo možné implementovat přepínání dnů pomocí *Swipe* gesta.

2. Přidání zkratky pro výběr rozvrhu přímo z detailu události. Například po otevření detailu události by bylo možné přejít do odpovídajícího rozvrhu pomocí stisknutí na příslušnou místnost nebo název předmětu. V současné době lze dosáhnout stejného výsledku pomocí vyhledávací obrazovky 2.11.

3. Přidání zkratky pro výběr semestru.

Další možnosti rozvoje aplikace:

1. Přidání oznámení, která budou upozorňovat uživatele na důležité události. Pro každého studenta je noční můra zapomenout na zkoušku nebo zápočtový test, a právě proto oznámení, která si uživatel může nastavit podle svých potřeb, mohou být užitečnou další funkcí.

2. Importování událostí z osobního kalendáře, které budou zobrazeny spolu s osobními událostmi uživatele v systému, aby aplikaci bylo možné používat jako jediný kalendář pro manipulaci s veškerými událostmi, které má uživatel.

3. Ukládání v lokální databázi nejenom osobních událostí, ale i událostí, které naposledy vyhledával uživatel.

4. Podpora češtiny, případně dalších jazyků.

5. Přidání filtrovacího UI prvku, který umožní zvolit, co uživatel v danou chvíli nepotřebuje zobrazovat.

Aplikace je implementována podle moderní doporučené architektury od společnosti Google, detailnější popis je v kapitole 2. Tato architektura zaručuje dobrou škálovatelnost kódu, a proto při správném sledování již fungující architektury nebude přidávání dalších funkcionalit představovat žádný problém. Navíc pro zajištění dobré kvality kódu byly použity nástroje pro statickou analýzu, což zajistí dobrou strukturu a čitelnost kódu při dalším rozvoji. Spolu s automatickými *Unit* testy, které umožní rozšiřovat aplikace bez obavy rozbit již fungující části kódu.

Závěr

Cílem této práci bylo navrhnout a implementovat mobilní multiplatformní aplikaci pro rozvrh hodin na FIT ČVUT a současně prozkoumat novou technologii KMM v multiplatformním světě.

Kapitola 1 popisuje současné řešení Fittable, jeho možnosti a nevýhody při používání na mobilních zařízeních. Tato kapitola také probírá použitou technologii KMM, jejíž základy, potřebné pro implementaci, byly detailně popsány, a byly rozebrány výhody a nevýhody zvoleného frameworku.

Následující kapitola 2 navrhuje architekturu aplikace podle nejmodernějších a doporučených standardů. Definuje také funkční a nefunkční požadavky podle priorit, které odpovídají nejdůležitějším možnostem již existující webové aplikace, a současně přidává sadu dalších rozšíření pro zlepšení uživatelské spokojenosti s novou aplikací.

Implementační část této práce, která je probrána v kapitole 3, popisuje do hloubky nejdůležitější knihovny, které byly použité pro vývoj této aplikace na různých vrstvách, jak pro sdílené části aplikace, tak i pro platformní. Následně byla aplikace naimplementována pro iOS a Android platformy, zdokumentována a otestována jak pomocí automatických nástrojů testování, tak i pomocí klasického uživatelského testování.

Byly prozkoumány nejdůležitější způsoby publikace aplikace skrz standardní distribuční kanály příslušných platforem, popsána distribuce je v kapitole 5.

Na závěr byly navrženy další možnosti vývoje aplikace a vyhodnocena zpětná vazba od uživatelů po testování, toto je zdokumentováno v kapitole 6.

Aplikace má všechny náležitosti uživatelský přívětivých moderních aplikací, má intuitivní rozhraní, obsahuje navíc animace, například při změně obsahu obrazovky, také při navigování uživatele mezi obrazovkami. Architektura aplikace zajišťuje dobrou a škálovatelnou strukturu aplikace, která bude snadno rozšiřitelná. Automatické testování a nástroje pro statickou analýzu kódu zajišťují dobrou údržbu aplikace. A použití moderních přístupů při sestavování umožní bez jakýchkoli problémů aplikaci sestavit a spustit.

Z toho plyne, že práce splňuje všechny body, které byly navrženy v zadání této práce a tím pádem ho splňuje.

Na závěr, výsledný produkt je použitelná aplikace, která nabízí studentům a zaměstnancům FIT ČVUT novou možnost příjemného a spolehlivého nahlížení do studijního rozvrhu.

Literatura

[1] Púčala, M.: Fittable Web. [online], [cit. 2023-12-30]. Dostupné z: https://timetable.fit.cvut.cz/new/

[2] JetBrains: Kotlin Multiplatform Official. [online], [cit. 2023-12-30]. Dostupné z: https://kotlinlang.org/docs/multiplatform.html

[3] JetBrains: The basics of Kotlin Multiplatform project structure. [online], [cit. 2023-12-30]. Dostupné z: https://kotlinlang.org/docs/multiplatform-discover-project.html

[4] Google: Guide to app architecture. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/topic/architecture

[5] Google: Data layer. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/topic/architecture/data-layer

[6] Google: Domain layer. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/topic/architecture/domain-layer

[7] Google: UI layer. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/topic/architecture/ui-layer

[8] Google: Publish your app. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/studio/publish

[9] Belinski, E.: iOS Distribution. [online], [cit. 2023-12-30]. Dostupné z: https://iosref.com/ios-usage

[10] Púčala, M.: Fittable Gitlab. [online], [cit. 2023-12-30]. Dostupné z: https://gitlab.fit.cvut.cz/pucalmar/fittable/-/tree/master/

[11] Zipdo: Dark Mode Usage Statistics. [online], [cit. 2023-12-30]. Dostupné z: https://zipdo.co/statistics/dark-mode-usage/

[12] CTU: Sirius github. [online], [cit. 2023-12-30]. Dostupné z: https://github.com/cvut/sirius

[13] CTU: Sirius Documentation. [online], [cit. 2023-12-30]. Dostupné z: https://cvut.github.io/sirius/docs/api-v1.html

[14] FIT, C.: OAuth 2.0. [online], [cit. 2023-12-30]. Dostupné z: https://help.fit.cvut.cz/dev/oauth2.html

[15] CTU: APPS MANAGER. [online], [cit. 2023-12-30]. Dostupné z: https://auth.fit.cvut.cz/manager/app-types.xhtml

[16] Luis Corral, T. R., Andrea Janes: Potential advantages and disadvantages of multiplatform development frameworks – A vision on mobile environments. [online], [cit. 2023-12-30]. Dostupné z: https://shorturl.at/diszY

[17] JetBrains: Kotlin Multiplatform Web. [online], [cit. 2023-12-30]. Dostupné z: https://www.jetbrains.com/kotlin-multiplatform/

[18] JetBrains: Set up targets for Kotlin Multiplatform. [online], [cit. 2023-12-30]. Dostupné z: https://kotlinlang.org/docs/multiplatform-set-up-targets.html

[19] IBM: Linting tools to identify and fix code mistakes. [online], [cit. 2023-12-30]. Dostupné z: https://www.ibm.com/garage/method/practices/code/tool_lint/

[20] JetBrains: Test your multiplatform app. [online], [cit. 2023-12-30]. Dostupné z: https://www.jetbrains.com/help/kotlin-multiplatform-dev/multiplatform-run-tests.html

[21] Reda, K.: Unit Testing in Kotlin Multiplatform Mobile (KMM). [online], [cit. 2023-12-30]. Dostupné z: https://medium.com/arconsis/unit-testing-in-kotlin-multiplatform-mobile-kmm-d687c6445ee7

[22] Piekielny, M.: How to test your KMM code. [online], [cit. 2023-12-30]. Dostupné z: https://medium.com/@maruchin/kmm-architecture-7-testing-93efd01f3952

[23] JetBrains: Configure compilations. [online], [cit. 2023-12-30]. Dostupné z: https://kotlinlang.org/docs/multiplatform-configure-compilations.html

[24] Square, I.: SqlDelight. [online], [cit. 2023-12-30]. Dostupné z: https://cashapp.github.io/sqldelight

[25] Google: Datastore. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/topic/libraries/architecture/datastore

[26] JetBrains: Ktor. [online], [cit. 2023-12-30]. Dostupné z: https://ktor.io/docs/welcome

[27] JetBrains: OkHttp. [online], [cit. 2023-12-30]. Dostupné z: https://ktor.io/docs/http-client-engines.html#okhttp

[28] JetBrains: Darwin. [online], [cit. 2023-12-30]. Dostupné z: https://ktor.io/docs/http-client-engines.html#darwin

[29] Giuliani, A.: Koin. [online], [cit. 2023-12-30]. Dostupné z: https://insert-koin.io/

[30] Google: Kotlin coroutines on Android. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/kotlin/coroutines

[31] Knuth, D. E.: *Fundamental Algorvithms*. Addison-Wesley, 1997, ISBN 978-0-201-89683-1.

[32] JetBrains: Coroutines Basics. [online], [cit. 2023-12-30]. Dostupné z: https://kotlinlang.org/docs/coroutines-basics.html

[33] Jetbrains: Asynchronous Flow. [online], [cit. 2023-12-30]. Dostupné z: https://kotlinlang.org/docs/flow.html

[34] JetBrains: KotlinX Datetime. [online], [cit. 2023-12-30]. Dostupné z: https://github.com/Kotlin/kotlinx-datetime

[35] Google: Android's Kotlin-first approach. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/kotlin/first

[36] Foundation, O.: AppAuth. [online], [cit. 2023-12-30]. Dostupné z: https://github.com/openid/AppAuth-Android

[37] Google: Thinking in Compose. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/jetpack/compose/mental-model

[38] Google: Compose Metrics. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/jetpack/compose/migrate/compare-metrics

[39] Google: Jetpack Compose. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/jetpack/compose

[40] Google: Navigating with Compose. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/jetpack/compose/navigation

[41] Apple: Swift. [online], [cit. 2023-12-30]. Dostupné z: https://developer.apple.com/swift/

[42] Apple: Combine. [online], [cit. 2023-12-30]. Dostupné z: https://developer.apple.com/documentation/combine

[43] Jin, D.: OAuthSwift. [online], [cit. 2023-12-30]. Dostupné z: https://github.com/OAuthSwift/OAuthSwift

[44] NSSpain: NSSpain2015. [online], [cit. 2023-12-30]. Dostupné z: https://2016.nsspain.com/

[45] Heitmann, J.: Stinsen. [online], [cit. 2023-12-30]. Dostupné z: https://github.com/rundfunk47/stinsen

[46] TouchLab: Skie. [online], [cit. 2023-12-30]. Dostupné z: https://skie.touchlab.co/

[47] TouchLab: Skie Features. [online], [cit. 2023-12-30]. Dostupné z: https://skie.touchlab.co/features/

[48] Gradle: Gradle Plugins. [online], [cit. 2023-12-30]. Dostupné z: https://docs.gradle.org/current/userguide/plugins.html

[49] Gradle: Sharing dependency versions between projects. [online], [cit. 2023-12-30]. Dostupné z: https://docs.gradle.org/current/userguide/platforms.html

[50] Pylypenko, O.: Mockk. [online], [cit. 2023-12-30]. Dostupné z: https://mockk.io

[51] Osherove, R.: *The art of unit testing*. Manning, 2009, ISBN 978-1-933988-27-6.

[52] Google: Google Play. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/distribute

[53] Google: Bundletool. [online], [cit. 2023-12-30]. Dostupné z: https://github.com/google/bundletool

[54] Google: Android Sign Application. [online], [cit. 2023-12-30]. Dostupné z: https://developer.android.com/studio/publish/app-signing

[55] Google: Developer Account Verification. [online], [cit. 2023-12-30]. Dostupné z: https://support.google.com/googleplay/android-developer/answer/10841920?hl=en

[56] Google: App testing requirements for new personal developer accounts. [online], [cit. 2023-12-30]. Dostupné z: https://support.google.com/googleplay/android-developer/answer/14151465?hl=en

[57] Apple: Xcode. [online], [cit. 2023-12-30]. Dostupné z: https://developer.apple.com/xcode/

[58] Apple: AppStore Review Guidelines. [online], [cit. 2023-12-30]. Dostupné z: https://developer.apple.com/app-store/review/guidelines/

Příloha A

Seznam použitých zkratek

AAB Android App Bundle

API Application Interface

APK Android Package

IDE Integrated Development Environment

JDK Java Development Kit

JS JavaScript

JSON JavaScript Object Notation

JVM Java Virtual Machine

KM Kotlin Multiplatform

KMM Kotlin Multiplatform Mobile

MVC Model View Controller

MVI Model View Intent

MVP Model View Provider

MVVM Model View ViewModel

REST Representational State Transfer

SDK Software Development Kit

UI User Interface

UX User Experience

XML Extensible Markup Language

Příloha **B**

Obsah elektronické přílohy

- `readme.txt` ..... stručný popis obsahu CD
- `code` ..... zdrojový kód
- `exe` ..... adresář se spustitelnou formou implementace
  - `app-release` ..... APK pro instalaci Android aplikace
- `src`
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
  - `figures` ..... obrázky nutné pro generování práce.
  - `extras` ..... další soubory nutné pro generovaní práce
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF