FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Porovnání frameworku Compose Multiplatform s ostatními multiplatformními frameworky
Student:	Bc. Jan Vepřek
Vedoucí:	Ing. Tomáš Nováček
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Frameworků pro vývoj mobilních aplikací pro více platforem zároveň je stále více. Jeden z nejnovějších přírůstků je Compose Multiplatform. Cílem této práce je porovnat jej s frameworky Flutter, React Native a PWA, co se týče výkonnosti a jednoduchosti použití a zjištění výhod a nevýhod jednotlivých přístupů.

1) Seznamte se s frameworky Compose Multiplatform a Kotlin Multiplatform pro tvorbu mulitplatformních aplikací.
2) Analyzujte existující řešení pro tvorbu multiplatformních aplikací (Flutter, React Native, PWA).
3) Implementujte funkční prototyp aplikace pro platformy Android, iOS a Web v daných technologiích dle bodu 2.
4) Vhodně otestujte Vaše řešení.
5) Porovnejte a vyhodnoťte výhody a nevýhody daných frameworků.
6) Diskutujte vhodnost frameworku Compose Mulitplatform a její možný rozvoj do budoucna.

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 16. listopadu 2023 v Praze.

Diplomová práce

POROVNÁNÍ FRAMEWORKU COMPOSE MULTIPLATFORM S OSTATNÍMI MULTIPLATFORMNÍMI FRAMEWORKY

Bc. Jan Vepřek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Tomáš Nováček
8. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat především vedoucímu této práce Ing. Tomáši Nováčkovi za veškeré rady. Dále bych chtěl poděkovat své rodině za podporu.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 8. května 2024

Abstrakt

Tato diplomová práce se zabývá frameworkem Compose Multiplatform určeným pro tvorbu multiplatformních aplikací. Seznamuji čtenáře s tímto frameworkem, popisuji rozdíly oproti frameworku Jetpack Compose. Analyzuji rovněž již existující frameworky zaměřené na tvorbu multiplatformních aplikací, jmenovitě frameworky Flutter, React Native a PWA. Ve vzájemném srovnání se věnuji podpoře základních funkcionalit, kvalitě dokumentace a dalším faktorům. V další části navrhuji uživatelské rozhraní prototypu aplikace v nástroji Figma a tento prototyp implementuji ve výše zmíněných frameworcích. Tyto prototypy následně řádně testuji. Navazuji srovnáním těchto platforem dle kvantitativních a kvalitativních měřítek, které zakládám mim jiné na ukázkovém průchodu aplikace a na řešení častých funkcionalit. V závěrečné kapitole porovnávám vhodnost frameworku Compose Multiplatform ve srovnání s ostatními platformami. Srovnávám zde kvalitu dokumentace, vybavenost frameworku, podporu knihoven a komunity.

Klíčová slova multiplatformní aplikace, Android, iOS, web, Compose multiplatform, Kotlin Multiplatform

Abstract

This thesis focuses on the Compose Multiplatform framework, designed for creating cross-platform applications. I introduce readers to this framework and describe its differences from the Jetpack Compose framework. I also analyze existing frameworks aimed at developing cross-platform applications, specifically Flutter, React Native, and PWA. In a comparative analysis, I consider the support of basic functionalities, the quality of documentation, and other factors. In the next section, I design the user interface of an application prototype in Figma and implement this prototype in the aforementioned frameworks. These prototypes are then thoroughly tested. I continue by comparing these platforms using quantitative and qualitative metrics, based on, among other things, a sample walkthrough of the application and the handling of common functionalities. In the final chapter, I compare the suitability of the Compose Multiplatform framework with other platforms, assessing the quality of documentation, framework features, library support, and community engagement.

Keywords multiplatform app, Android, iOS, web, Compose multiplatform, Kotlin Multiplatform

Seznam zkratek

API	Application Programming Interface
DSL	Domain-Specific Language
IDE	Integrated Development Environment
JVM	Java Virtual Machine
PWA	Progressive Web App
SDK	Software Development Kit
UI	User Interface
URL	Uniform Resource Locator
XML	Extensible Markup Language

Úvod

V dnešní době existuje mnoho platforem pro tvorbu multiplatformních aplikací a jejich počet narůstá. S ohledem na rostoucí náklady a čas potřebný na vývoj nativních aplikací pak více firem a vývojářů dává přednost právě multiplatformním řešením oproti nativním aplikacím pro každou platformu odděleně.

Jedním z nejnovějších frameworků je pak Compose Multiplatform, který vychází z UI frameworku Jetpack Compose, který je již zaběhnutý na platformě Android. Framework Compose Multiplatform je ještě stále v raném vývoji, s podporou pro webovou platformou v experimentální fázi. Z tohoto důvodu jej v této diplomové práci zkoumám, abych mohl na základě analýzy a práce s ním poskytnout doporučení a názor na možný rozvoj do budoucna.

V této diplomové práci tento framework zkoumám a přibližuji čtenáři. Ve druhé kapitole se zabývám jeho rozdíly oproti frameworku Jetpack Compose a použitelností pro vývoj, dále zde rozebírám nejčastější funkcionality v aplikacích a jejich možná řešení, která framework Compose Multiplatform poskytuje. Mimo to se zde věnuji i podpoře pro design systém Material 3 a kvalitu dokumentace.

V další kapitole analyzuji existující řešení, jmenovitě to jsou frameworky Flutter, React Native a PWA. Každý z nich analyzuji z několika hledisek, která jsou důležitá pro tvorbu multiplatformních aplikací. Také zde přibližuji čtenáři způsoby, jakými implementovat základní funkcionality v daném frameworku.

Dále pak navrhuji prototyp multiplatformní aplikace vycházející z Material Design 3 systému v aplikaci Figma, který je dále v této práci implementován. Návrh využívá různé UI komponenty (BottomBar, TopBar, BottomSheet, ...).

V další kapitole pak implementuji prototyp multiplatformní aplikace pro platformy Android, iOS a web ve všech výše zmíněných frameworcích. V tomto prototypu jsou zastoupeny nejčastější potřeby (načítání dat z API, ukládání dat do databáze, zobrazení fotky z URL, několik obrazovek, včetně přechodu mezi nimi a udržení stavu). Popisuji zde architekturu, která se pro tvorbu mobilních aplikací může využít a jakým způsobem lze aplikaci sestavit.

Na základě implementace těchto prototypů pak vyvozuji závěry a srovnávám tyto platformy dle kvantitativních a kvalitativních měřítek. Implementovaná řešení porovnávám dle výkonnosti, plynulosti UI pod zátěží, velikosti aplikačního souboru, rychlosti spuštění aplikace a dalšího.

V následující kapitole porovnávám vhodnost frameworku Compose Multiplatform oproti ostatním platformám. Srovnávám zde především kvalitu dokumentace, vybavenost frameworku, podporu knihoven a komunity.

Kapitola 1
Cíle

Hlavním cílem této práce je seznámit čtenáře s frameworkem Compose Multiplatform a popsat jeho klíčové vlastnosti a rozdíly oproti frameworku Jetpack Compose. Dále je cílem analyzovat existující frameworky pro tvorbu multiplatformních aplikací, jako jsou Flutter, React Native a PWA, a provést jejich srovnání z hlediska podpory základních funkcionalit, kvality dokumentace a dalších faktorů.

Dalším cílem práce je navrhnout uživatelské rozhraní prototypu aplikace v nástroji Figma a následně implementovat tento prototyp ve frameworku Compose Multiplatform, Flutteru, React Native a PWA. Součástí práce je také testování a porovnávání prototypů vytvořených v jednotlivých platformách z hlediska výkonnosti, použitelnosti a dalších kritérií.

Navazujícím cílem je také diskutovat o vhodnosti frameworku Compose Multiplatform v porovnání s ostatními platformami pro vývoj multiplatformních aplikací, zohledňující kvalitu dokumentace, vybavenost frameworku, podporu knihoven a komunity.

Posledním cílem této práce je poskytnutí doporučení a názorů na možný rozvoj frameworku Compose Multiplatform a jeho praktické využití pro tvorbu multiplatformních aplikací.

Kapitola 2

Frameworky Compose Multiplatform, Kotlin Multiplatform

V této kapitole představím framework Compose Multiplatform, který se ve spojení s frameworkem Kotlin Multiplatform dá použít k tvorbě multiplatformních aplikací.

2.1 Jetpack Compose

Společnost Google v roce 2019 vydala (v omezené preview verzi) UI toolkit s názvem Jetpack Compose. Ten představuje alternativu k tvoření uživatelského rozhraní Android aplikace přes nativní API. Tento framework představuje deklarativní způsob psaní UI. Jedná se o způsob, který je využívaný dalšími frameworky, například React Native a Vue.js. Ve frameworku Jetpack Compose se programuje za pomocí programovacího jazyka Kotlin. Mimo to podporuje interpolaci s nativním API pro UI, tudíž vývojář může spojit obě technologie v jednom UI. [2]

2.1.1 Kotlin

Kotlin je programovací jazyk vyvinutý společností JetBrains, oficiální verze 1.0 vyšla již v únoru 2016. Mezi jeho výhody patří úplná interoperabilita s jazykem Java, má syntaxi jež z tohoto jazyka vychází a zároveň šetří řádky kódu (odhadem kolem 40 % řádků kódu [3]). Mimo to podporuje `null-safety` (funkcionalita, která zajišťuje předcházení NullPointerException pomocí nullable a non-nullable referencí) a `smart casting` (funkce, díky které dochází k automatickému přetypování, není potřeba psát `cast`). [4]

Od roku 2019 jej společnost Google doporučuje jako preferovaný jazyk ke psaní nových aplikací na platformě Android. Pouhé dva roky předtím Google oznámil podporu pro Kotlin v Android Studio IDE. Jakožto preferovaný jazyk od té doby vychází nová Jetpack API přednostně pro Kotlin a je doporučovaný pro použití v novém projektu. Mezi výhody oproti jazyku Java společnost uvádí mimo jiné méně potřebného kódu k napsání, otestování a údržbě. [5]

V dnešní době ve spojení s UI kitem Jetpack Compose vývojáři stačí pouze znalost Kotlinu k tvorbě aplikací pro platformu Android (oproti jazykům Java a XML, jak tomu bylo v minulosti).

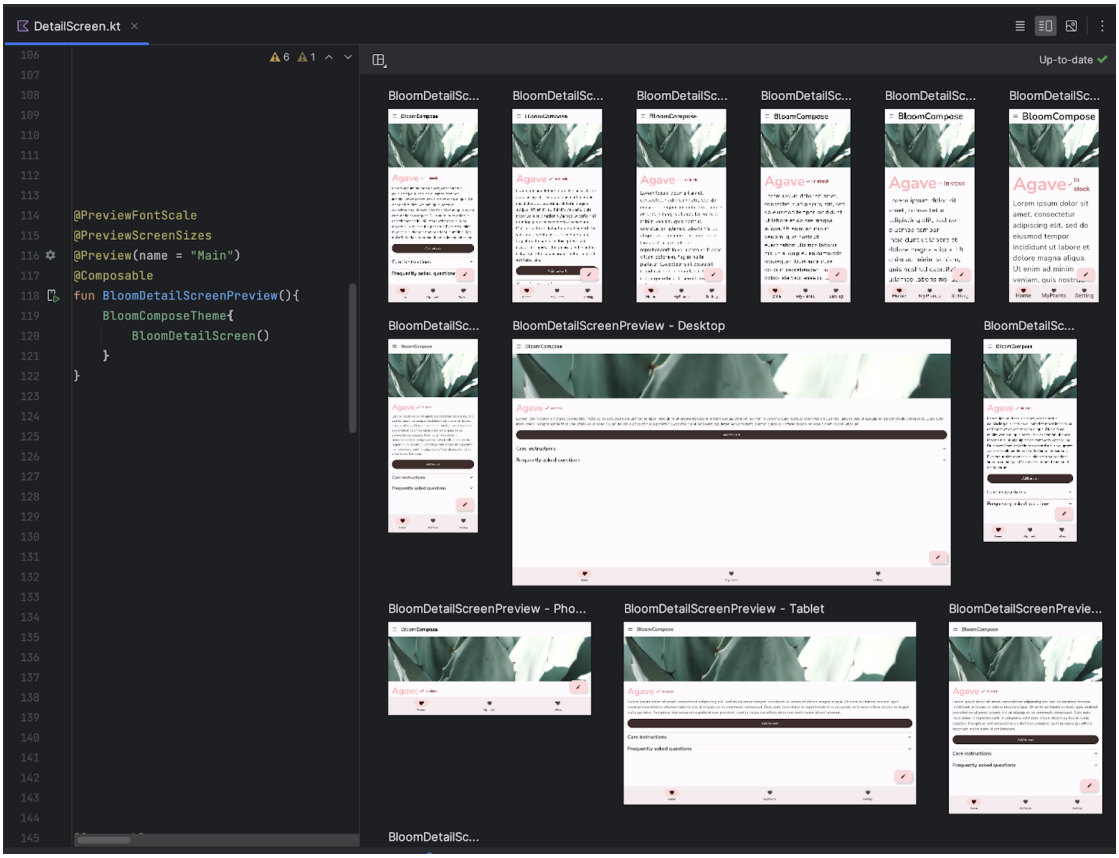
2.1.2 Composable funkce

Jedná se speciální funkce, pomocí níž se tvoří UI v tomto frameworku. Composable funkce se vytváří pomocí `@Composable` anotace nad názvem funkce. Celé uživatelské rozhraní díky tomu tvoří pomocí vnořených composable funkcí, které převážně nemají návratovou hodnotu (vrací `Unit`).

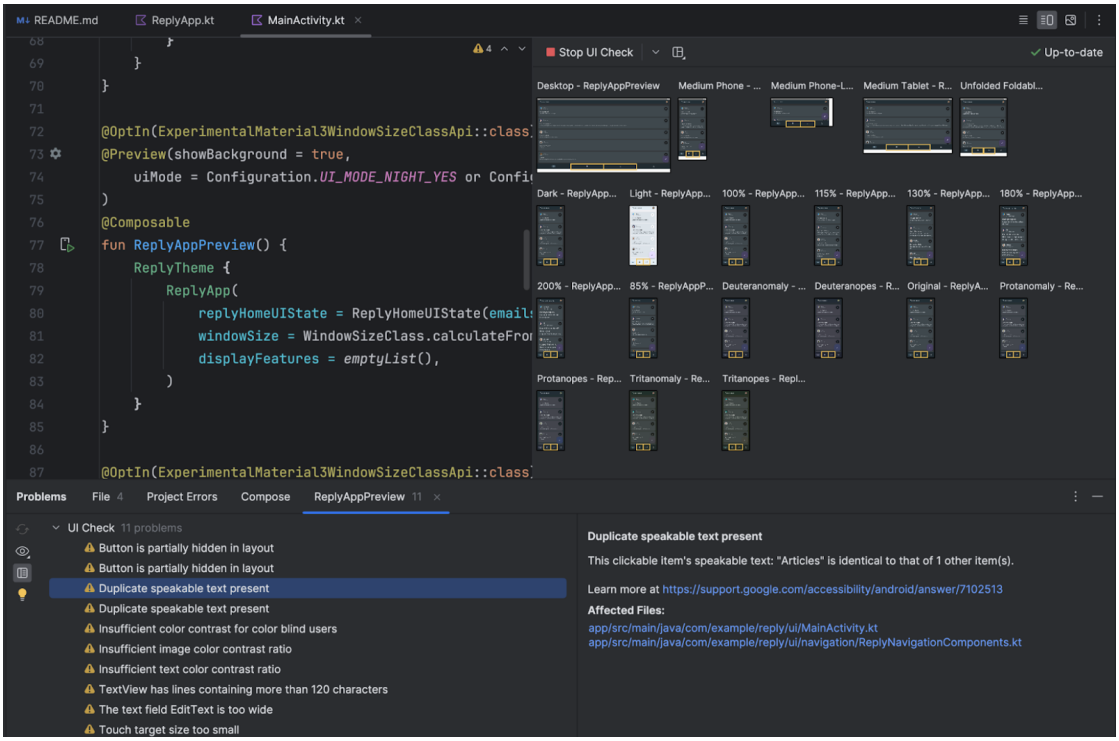
2.1.3 Preview

Díky `@Preview` anotaci si může vývojář v náhledu zobrazit composable funkci přímo v Android Studiu (oficiální IDE pro tvorbu aplikací pro platformu Android) bez nutnosti spouštět aplikaci na mobilním zařízení. V případě změny v kódu musí znovu dojít ke kompilaci, lze ale nastavit automatickou kompilaci v případě změn, v tomto případě se změny promítnou do náhledu okamžitě.

O tom, že je tato funkcionalita využívána vývojáři, dokládá fakt, že je neustále vylepšována v nových verzích Android Studia. V Android Studia verzi Hedgehog (2023.1.1) došlo k vylepšení náhledů, přibyla například **Multipreview** šablona, která poskytuje možnost zobrazit si náhled ve více různých velikostech zařízení či ve světlém i tmavém módu najednou, ukázka náhledu různých velikostí je vidět na obrázku 2.1. Mimo to přibyl režim galerie, kdy se vývojář může zaměřit pouze na jednu variantu. [6]



Obrázek 2.1 Multipreview šablony [6]



Obrázek 2.2 Výpis z UI Check Mode [7]

2.2 Compose Multiplatform

Compose Multiplatform je UI knihovna od společnosti JetBrains. V současné době plně podporuje platformu Android (skrze Jetpack Compose) a desktopové aplikace (Windows, MacOS, Linux). [10]

Podpora pro operační systém iOS v alfa verzi byla vydána teprve v květnu roku 2023. [16]

Podpora pro web je zatím v experimentální verzi, tudíž jí společnost JetBrains doporučuje vývojářům pouze na ozkoušení a k poskytnutí zpětné vazby. Tato knihovna je postavená na základě frameworku Jetpack Compose. Díky tomu je její použití velice jednoduché pro vývojáře, kteří jej již znají a využívají. [10]

2.2.1 Rozdíly oproti Jetpack Compose

Vzhledem k tomu, že mezi použitím frameworku Compose Multiplatform a Jetpack Compose není z vývojářského hlediska rozdíl, tak se dokumentace Compose Multiplatform odkazuje na dokumentaci pro Jetpack Compose. Mimo to obsahuje část věnující se komponentám frameworku Jetpack Compose, které jsou k dispozici pouze pro platformu Android. Dochází k tomu proto, že buď je daná komponenta specifická pro Android, nebo proto, že nebyla ještě přenesena na ostatní platformy. Mezi Android specifické komponenty se řadí například třídy `android.context.Context` (třída poskytující přístup k prostředí systému Android) a `ComposeView` (využívaná k interpolaci nativními Views) či knihovna `Hilt` poskytující Dependency Inejction (DI). [11]

Navigace v mobilních aplikacích představuje důležitou funkcionalitu – takřka každá aplikace se skládá z více než jedné obrazovky, a mezi nimi je zapotřebí se navigovat.

Momentálně není žádná navigační knihovna přímo součástí frameworku Compose Multiplatform, nicméně, jak uvádí dokumentace, v budoucnu se plánuje podpora out-of-the-box. Z tohoto

důvodu jsou v dokumentaci zatím pouze doporučovány knihovny třetích stran, jmenovitě se jedná o knihovny `Voyager`, `Decompose`, `Appyx` a `PreCompose`. U každé z nich je pak krátký popisek, aby vývojář měl představu, která knihovna mu bude pro jeho potřeby nejvíc vyhovovat. [12]

Další věc, pro kterou není v tomto frameworku přímá podpora, je zobrazování fotek a obrázků z URL. Vývojář si ale může vybrat z několika knihoven, které zde jsou k dispozici. Mimo použití knihoven je pak možnost dopsat si tuto funkcionalitu pomocí síťového klienta a využít k zobrazení Composable funkci Image. Tato možnost je popsána na GitHub repozitáři Compose Multiplatform[1]. Na stránkách dokumentace Compose Multiplatform jsou pak doporučovány knihovny **Compose ImageLoader**, **Kamel** a již dříve zmiňovaný **Ktor client**.

Z hlediska poskytovaných funkcí je nejlepší knihovna Kamel. Ta poskytuje jednoduchý a efektivní způsob, jak načítat, zobrazit a cachovat obrázky. Využívá klienta `Ktor` pro načítání zdrojů. Mimo načítání zdrojů z URL podporuje i načítání lokálních zdrojů ze souboru. Kromě toho poskytuje možnost definování lambda funkcí pro případ načítání zdroje a pro případ nenadálé chyby. Mezi další pokročilé funkce patří i možnost definování animací. [76]

Ve frameworku Jetpack Compose je velmi oblíbená knihovna Coil, která má momentálně v alfa verzi podporu pro Compose Multiplatform (ve verzích pro Android, iOS a desktop). Plná podpora pro Compose Multiplatform má přijít v plnohodnotném vydání verze 3.0, jak se píše na jejich blogu. [65]

Během vývoje se vývojář neobejde bez nějaké formy logování, ve frameworku Jetpack Compose se může používat Logcat, který je poskytován platformou Android. Na platformě Compose Multiplatform pak neexistuje přímá podpora pro multiplatformní logování, vývojář si musí buď implementovat logování sám či využít nějakou z nabízených knihoven. Nabízených knihoven je více, mezi nejpoužívanější pak patří knihovna Napier. [59]

2.2.2 Preferované IDE

Pro tvorbu Compose Multiplatform aplikací se může využít více různých IDE. Konkrétně se dá využít Android Studio, IntelliJ IDEA či JetBrains Fleet. Všechna tři IDE jsou od společnosti JetBrains. Jelikož se počítá s tím, že vývojář má zkušenosti s Jetpack Compose, je přímo v dokumentaci doporučováno k vývoji, navíc oproti ostatním poskytuje ještě platformně specifické funkcionality, které se dají využít k rychlejšímu ladění a vývoji. Dokonce jdou z Android Studia spouštět emulátory pro platformu iOS, jinak spouštět emulátory lze také z Xcode (IDE od společnosti Apple pro vývoj aplikací a programů pro systémy v jejím ekosystému).

Jak jsem se zmiňoval v minulé podsekci, `Preview` anotace se dá ve frameworku Jetpack Compose použít k zobrazování náhledu UI. Plnou podporu má tato funkcionalita u platformy Android (a Desktop, pokud se použije plugin Compose Multiplatform IDE Support[2]).

Nicméně v případě Compose Multiplatform aplikace tato anotace funguje pouze v IDE JetBrains Fleet, kde funguje pro kód ve společné části, s experimentální podporou pro parametry. [8]

JetBrains Fleet je nový textový editor a IDE s podporou všech programovacích jazyků od společnosti JetBrains. Poskytuje jak rychlost textového editoru, tak funkcionality IDE. Je připravený pro různé konfigurace – může se spustit jak na lokálním zařízení, tak můžou některé procesy běžet v cloudu. Jinak podporuje funkcionality, na které jsou uživatelé produktů od JetBrains již zvyklí – obsahuje terminál, podpora pro Git, ladění, témata. Fleet je v době psaní této práce zdarma k dispozici v preview verzi, prozatím například nepodporuje pluginy. [9]

2.2.3 Kotlin/Wasm

WebAssembly (Wasm) je binární formát pro virtuální stroje. Je navržený, aby se dal využít jako cíl kompilace pro programovací jazyky, díky němu se dá tento kód použít pro webový klient. Má

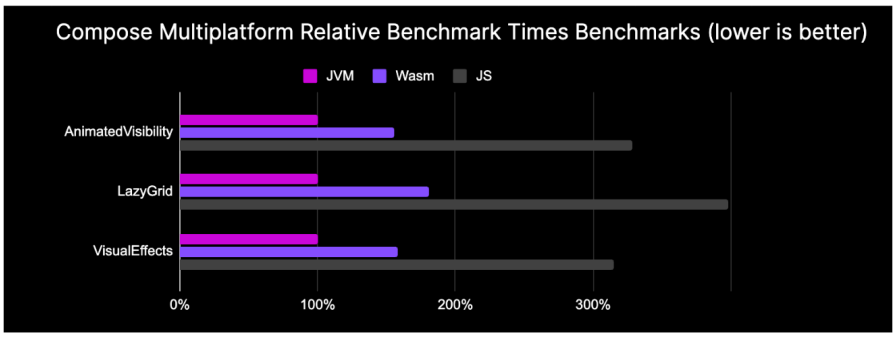
[1] `https://github.com/JetBrains/compose-multiplatform/blob/master/tutorials/Image_And_Icons_Manipulations/README.md#loading-images-from-device-storage-or-network-asynchronously`

[2] `https://plugins.jetbrains.com/plugin/16541-compose-multiplatform-ide-support`

podporu hlavních prohlížečů (**Chrome**, **Firefox**, **Safari** a **Edge**). [17]

Kotlin/Wasm představuje možnost jak kompilovat kód v jazyce Kotlin do formátu Wasm. Pro webové platformy je pak ještě zajímavý Kotlin for JavaScript (Kotlin/JS), který poskytuje možnost kompilovat kód psaný v jazyce Kotlin do jazyku JavaScript. Tento framework je vydaný ve stabilní verzi.

Compose Multiplatform pak používá cíl Kotlin/Wasm pro kompilaci pro webové klienty. Samotný Kotlin/Wasm je nyní v alfě. [20] I přesto má již nyní Compose Multiplatform na webové platformě poměrně nadějné rychlostní výsledky, kde se přibližuje rychlosti Java Virtual Machine (JVM). Je navíc mnohem rychlejší než target Kotlin/JS, jak je vidět na obrázku 2.3.



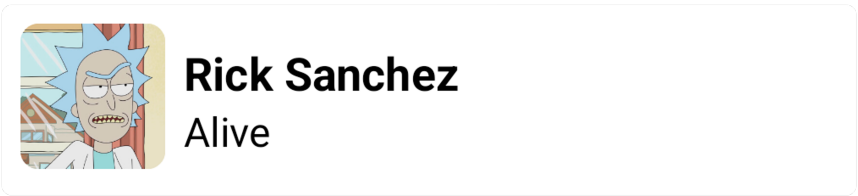
Obrázek 2.3 Porovnání rychlostí Compose Multiplatform [21]

Tím, že je Kotlin/Wasm v alfa verzi a Compose Mutliplatform pro web teprve v experimentální verzi, je potřeba zmínit nynější nedostatky těchto technologií. Důležitá věc, která je zapotřebí zmínit, je podpora prohlížečů.

Compose Multiplatform pro webovou platformu se opírá o technologii WebAssembly Garbage Collection (WasmGC). Tato technologie je momentálně dostupná pro většinu nejpoužívanějších prohlížečů. Prohlížeče Chrome, Edge jí podporují od verze 119, Firefox pak od verze 120, akorát prohlížeč Safari jej stále zatím nepodporuje. Mimo to se dá očekávat, že se Compose Multiplatform může v budoucnosti kdykoliv měnit. Je také důležité zmínit, že Kotlin/Wasm s Compose Multiplatform se v prohlížeči vykresluje přímo na canvas, nemá žádný DOM hierarchický strom, z tohoto důvodu tedy nejsou webové stránky přístupné lidem se zrakovým postižením a není zde podpora pro důležité funkce jako možnost zvětšit či vybírat text, překládat stránku a další. [75]

2.2.4 Ukázka

Na následujícím obrázku 2.4 je vidět jednoduchý `layout`, vytvořený pomocí Compose Multiplatform:



Obrázek 2.4 Jednoduchý layout obsahující obrázek a dva texty

Výpis kódu 1 se skládá z jednoduchého layoutu, kde se nachází dvě textová pole a obrázek načtený z `URL`. Jsou zde použity komponenty (composable funkce) `Row`, `Column`, `Text` a `KamelImage`. Texty jsou zde stylizované dle definovaného tématu.

```
Row(modifier = Modifier
                .fillMaxWidth()
                .padding(all = 8.dp),
    ) {
        KamelImage(
            resource = asyncPainterResource(character.iconUrl),
            contentDescription = "avatar",
            modifier = Modifier
                    .clip(8.dp)
                    .size(64.dp),
        )
        Column(modifier = Modifier.padding(all = 8.dp)) {
            Text(
                text = character.name,
                style = RickAndMortyTheme.typography.titleMedium,
            )
            Text(
                text = character.status,
                style = RickAndMortyTheme.typography.bodyMedium,
            )
        }
    }
```

Výpis kódu 1 Ukázka layoutu v Compose Multiplatform

2.2.5 Komponenty

V ukázce je použito hned několik komponent. Komponenty `Row` a `Column` určují kompozici prvků na obrazovce. Ve výchozím chování se totiž všechny vykreslené prvky překreslují přes sebe, tudíž je zapotřebí určit chtěné rozložení. `Row` se používá pro prvky umístěné horizontálně, `Column` pro ty umístěné vertikálně, mimo ně existuje ještě composable funkce `Box`, který umožňuje skládat prvky na sebe.

Komponenta `KamelImage` je součástí knihovny `Kamel`, o které jsem se v této kapitole již zmiňoval. Povinné parametry, které musí být předány této composable funkci, jsou `resource` a `contentDescription`. Parametr `resource` určuje zdroj, který se má načíst. Tento zdroj může být prezentován přes `asyncPainterResource`, což je composable funkce, která bere za argument více různých zdrojů dat, například webovou URL reprezentovanou řetězcem či lokální soubor. Parametr `contentDescription` je pak řetězec představující popis daného obrázku, tento parametr je zde kvůli přístupnosti aplikace, zejména pro uživatele se zrakovým postižením, kterým poskytuje popis obrázku.

Další komponentou je composable funkce `Text`, která se stará o vykreslení textu. Povinný argument je text, tedy řetězec, který se má vykreslit. Mimo to lze nastavovat vzhled textu, včetně různé stylizace, minimální/maximální počet řádků či chování v případě přetečení textu.

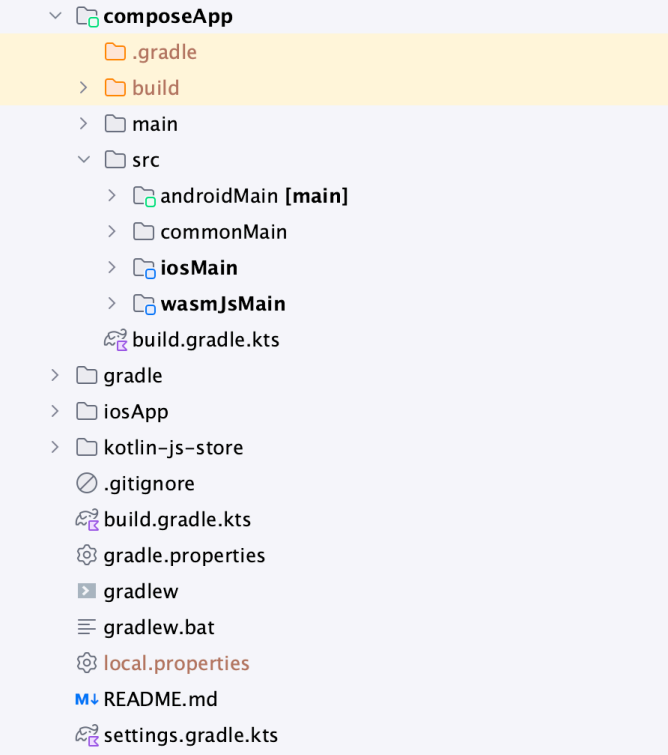
Důležité je ještě zmínit `Modifier`. Jedná se o composable funkci, která se používá k modifikaci dané composable funkce. Jejich využití je všestranné, od změny velikosti, přidání okrajů či změny barvy až po přidání fokusovatelnosti a klikatelnosti. Díky tomu pak může docházet k vyšší znovupoužitelnosti composable funkcí. Každá poskytovaná composable funkce jej pak bere jako nepovinný argument. Je dobrá praxe, že když vývojář píše vlastní composable funkci, přidá ji jako nepovinný parametr `Modifier`: `modifier: Modifier = Modifier`.

2.2.6 Struktura projektu

Projekt psaný ve frameworku Compose Multiplatform má strukturu vycházející z aplikace psané pro platformu Android ve frameworku Jetpack Compose. Tuto souborovou strukturu můžete vidět na obrázku 2.5. K sestavování projektu se tedy používá nástroj **Gradle**, v kořenovém adresáři jsou soubory `build.gradle.kts` a `settings.gradle.kts`, které slouží k definování pluginů.

Převážná většina kódu aplikace se pak nachází v modulu `ComposeApp`, zde se ve složce `src` nachází složky pro zdrojové kódy v jazyku Kotlin. Jsou rozděleny dle platformy, společný kód a zdroje jsou ve složce `commonMain`. Ostatní složky (`androidMain`, `iosMain` a `wasmJsMain` v tomto případě) obsahují composable funkce představující vstupní bod aplikace na dané platformě, dále pak platformě specifický kód implementovaný přes `expected/actual` funkce, čemuž se více věnuji v kapitole 2.3.1. Také se zde nachází soubor `build.gradle.kts`, ve kterém jsou definovány závislosti, které se v projektu využívají.

Dále se zde nachází moduly `gradle`, který obsahuje verzovací katalog `libs.versions.toml`. Ten se využívá ke správě závislostí a pluginů.

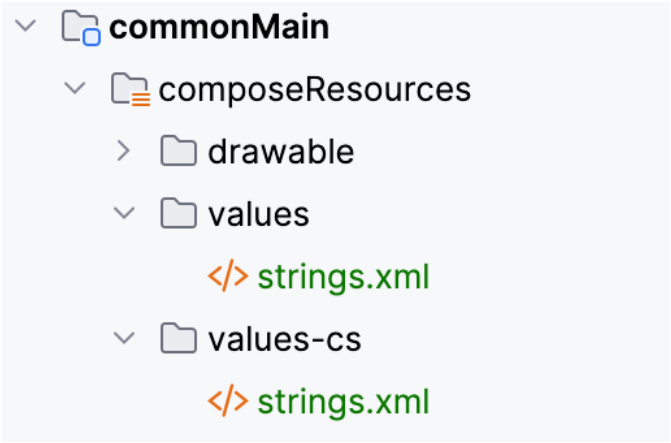


Obrázek 2.5 Struktura projektu ve frameworku Compose Multiplatform

2.2.7 Práce se zdroji

Statické texty a vlastní ikony tvoří nepostradatelnou součást každé mobilní aplikace. Texty bývají, v závislosti na platformě, definované v daných souborech (`strings.xml` pro Android, `Localizable.strings` pro iOS). Ikony mají různé formáty (platforma Android používá ikony v XML, iOS pak poskytuje třídu SymbolSF, jež obaluje vektorové ikony, například ve formátu svg). Na webu je možností mnoho, záleží na tom, kterou technologii vývojář použije, ale pro uložení textů se může použít například soubor ve formátu json. Ukládání ikon je pak úplně na

vývojáři, není zde omezován žádným formátem, nejčastěji se kvůli škálování a jednoduchosti používá formát svg. V průběhu psaní této diplomové práce vyšla aktualizace knihovny Compose Multiplatform (jmenovitě verze 1.6.0), díky které došlo ke zjednodušení práce se sdílenými zdroji. Díky ní se na zdroje odkazuje přes vygenerované soubory Res.kt. Do té doby se musely zdroje definovat pomocí řetězce, jež představoval umístění daného zdroje. Z tohoto důvodu knihovna neposkytovala možnost zkontrolovat správnost během kompilace a chybně zadaný zdroj se projevil až pádem aplikace. Nyní se se zdroji v podobě ikon, textu a fontů pracuje prakticky obdobně jako na platformě Android. Vkládají se do složky `composeResources`, hierarchie v ní je pak stejná jako v případě Androidu. Ikony jsou tedy ve složce drawable, řetězce ve složce values a fonty ve složce fonts. Stejně jako na platformě Android je zde podpora například pro překlady, tedy definování textu v různých jazycích a uložení v dané variantě složky values dle zkratky jazyka. Mimo to je zde podpora pro různé velikosti ikon, takže se v závislosti na rozlišení displeje vybere ta správná, aby se vykreslila v co nejlepší kvalitě, ale zároveň moc nezatěžovala systém. V ukázce 2.6 je vidět hierarchie společně s řetězci definovanými i pro český jazyk.



Obrázek 2.6 Struktura zdrojů v projektu pro Compose Multiplatform

Výhoda tohoto řešení pro práci s řetězci spočívá v tom, že tento způsob je zaběhlý na platformě Android. Tudíž služby, které se starají o překlady textů, mají možnost exportovat do `xml` souboru se stejnou strukturou a vývojář tedy nemusí řešit export do jiného formátu. Například služba Crowdin má zdarma k dispozici plugin Android XML String Exporter[3], který poskytuje možnost generovat `xml` soubory s přeloženými texty.

2.2.8 Material Design

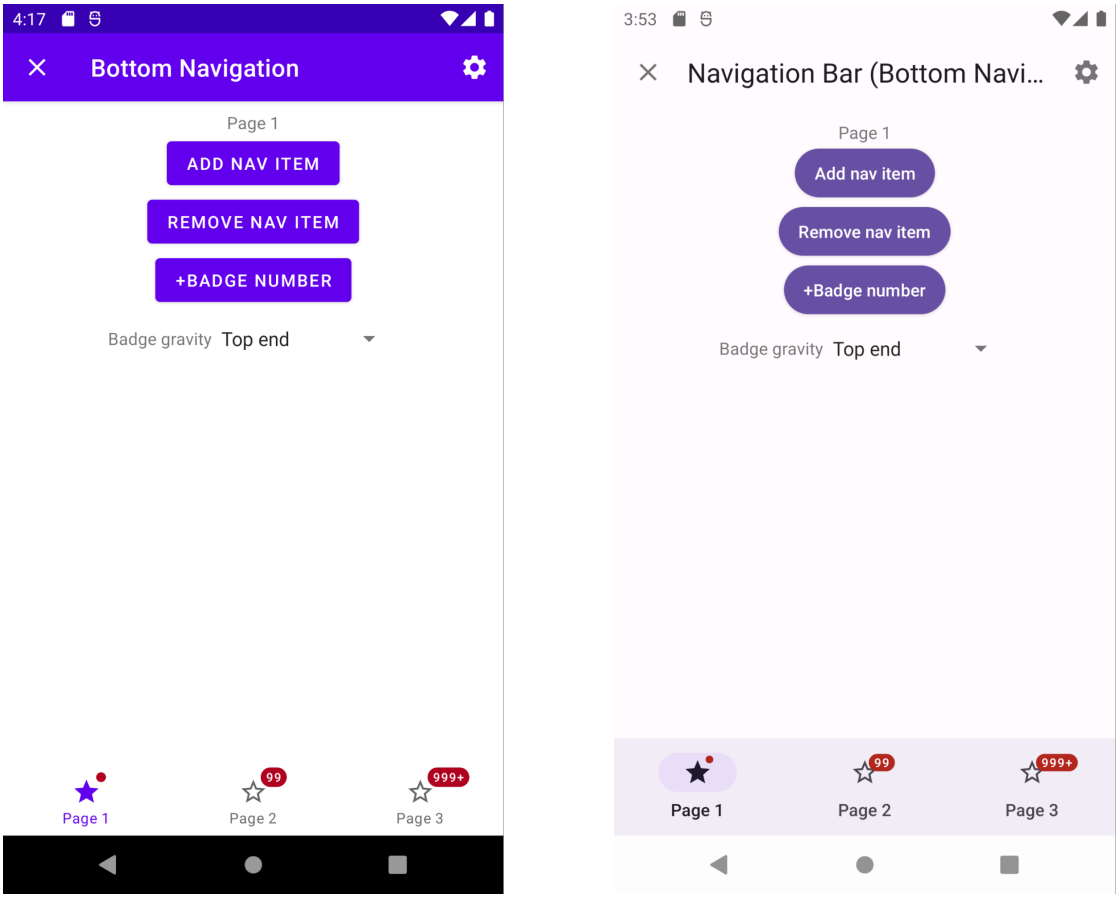
Každá aplikace v Compose Multiplatform má vydefinované vlastní téma, které sestává z barev, typografie a tvarů, které aplikace používá. Komponenty v knihovně Jetpack Compose se řídí design systémem Material Design. První verzi vyvinula společnost Google již v roce 2014 k příležitosti vydání nové verze Androidu 5.0 (Lollipop). Tento systém vznikl za účelem sjednocení vizuálního designu a interakcí napříč aplikacemi a webovými stránkami. [63] V roce 2018 vyšla nová verze – Material Design 2. Vizuální rozdíly oproti verzi 2 byly minimální. [64]

Nejnovější verze, Material Design 3, byla oznámena společností Google v květnu 2021. Přezdívá se jí také Material You. Mezi největší změny v této verzi patří personalizace stylů aplikací, aby byly design přístupný pro každou potřebu a adaptovaný pro každou obrazovku. Tohoto je docíleno pomocí unikátních Material palet, které se zvládnou přizpůsobit uživateli a vychází například z tapety na mobilu. Mimo to v této verzi dochází ke zlepšování přístupnosti díky větší

[3] `https://store.crowdin.com/android-string-exporter`

kontrole nad kontrastem a velikostí komponent. [22]

Na obrázku 2.7 jsou vedle sebe k porovnání rozdíly v komponentách těchto dvou verzí. Jak je vidět, změnami prošly téměř všechny komponenty. Například top bar v Material 3 verzi již nevystupuje z obrazovky, je součástí celku. Pozadí je pak lehce zabarvené, není čistě bílé, ale vychází z primární barvy. Tlačítka pak jsou více zakulacená a oproti druhé generaci již není text ve výchozím stavu psán verzálkami. BottomBar také prošel změnami – zbavil se stínu, přibylo lehké barevné podbarvení vybrané položky a zvýšila se jeho výška, aby byl více přístupný.



(a) Layout s komponenty Material Design 2

(b) Layout s komponenty Material Design 3

Obrázek 2.7 Rozdíly mezi verzemi Material Design 2 a Material Design 3

Compose Multiplatform (stejně jako Jetpack Compose) nabízí podporu jak pro Material 2, tak pro Material 3. Vývojář si tedy může vybrat, kterou z těchto knihoven využije pro svou aplikaci, ale Material Design 3 je silně doporučován pro novější aplikace. Nicméně, stále existují komponenty, které v něm implementovány nejsou, oproti jeho předchozí verzi. Na oficiálních stránkách Material Designu je pak k dispozici návod pro přechod z verze 2 na verzi 3. [25]

Framework Compose Multiplatform od verze 1.5.10 umožňuje využívat verzi Material Design 3 (před touto verzí nebyly k dispozici některé komponenty). Mezi nejnověji přidané komponenty patří `ModalBottomSheet` a `SearchBar`. [70]

Mimo Material Design může vývojář využít i možnosti rozšíření Material Design o vlastní barvy, typografii či tvary. Taková možnost se doporučuje v případě několika málo rozšíření. V případě, že se design systém aplikace vůbec neřídí Material Design systémem, může si vývojář nadefinovat vlastní systém a nahradit Material Design, buď po částech (například jen vlastní barevné schéma či vlastní typografie) nebo úplně. V takovém případě se ale doporučuje obalit

komponenty z Material Design knihovny a vystavit je tak vlastnímu systému. Výchozí chování těchto komponentů totiž využívá přednastavené hodnoty, například barevné schéma, které automaticky nastavuje barvu pozadí, ale v případě vlastního schématu se tato informace musí explicitně napsat. [23]

2.3 Kotlin Multiplatform

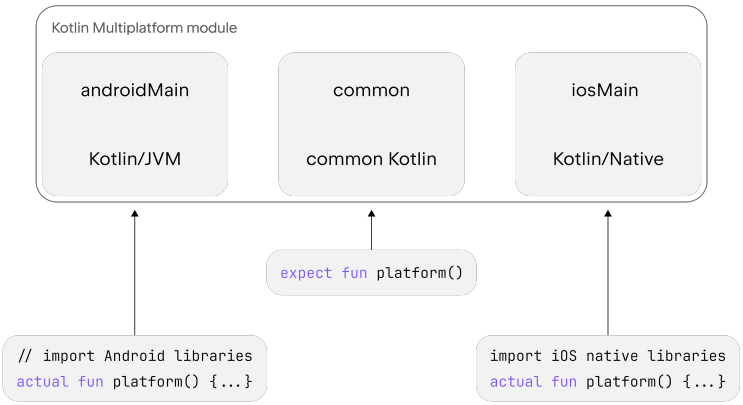
Kotlin Multiplatform (KMP) je open-source framework vytvořený společností JetBrains. Podporuje psaní multiplatformních aplikací pro platformy `Android`, `iOS`, `Web`, `Desktop` a pro `Server`. Nejedná se o UI framework, používá se ke sdílení aplikační logiky napříč platformami. [69]

Z tohoto důvodu se v Compose Multiplatform aplikacích využívají oba tyto frameworky dohromady.

2.3.1 Platformně specifické API

Pro platformu Kotlin Multiplatform existuje spoustu knihoven zaměřujících se na nejčastější funkcionality.

V případě, že nějaká funkcionalita uživateli chybí, může si ji dodefinovat pomocí takzvaných `expected` a `actual` funkcí. Fungují na následujícím principu: ve složce se sdíleným kódem se nachází `expected` deklarace, která je pak deklarovaná v každém platformním modulu pomocí **actual** deklarace. Toto je zajištěno kompilátorem – kód nepůjde zkompilovat, pokud na některé platformě chybí **actual** definice nějaké z `expected` funkcí. Tento mechanismus se dá použít na skoro všechny deklarace jazyku Kotlin (např. `fun`, `interface`, `enum` a na anotace). Definice tříd jsou ale zatím v beta verzi, tudíž při kompilaci vyskočí na vývojáře upozornění. [60] Obrázek 2.8 ilustruje tento mechanismus v případě platforem Android a iOS.



Obrázek 2.8 Ukázka `expected, actual` funkcí pro platformy Android a iOS [60]

2.3.2 Databáze

Skoro každá aplikace potřebuje nějakým způsobem ukládat data pro perzistentní užití. V případě menších aplikací (či menších dat, například nastavení aplikace) se toto dá řešit přes `key/value`

úložiště, které je k dispozici na daném systému. Na platformě Android se k tomuto účelu dá využít API SharedPreferences, na platformě iOS pak API NSUserDefaults. Poměrně oblíbená knihovna dostupná pro Kotlin Multiplatform je knihovna Multiplatform Settings[4]. Tato knihovna poskytuje implementace pro platformy Android, iOS, desktop a další. Pro platformy Android a iOS využívá výše zmíněné druhy úložišť.

U větších aplikací a složitějších dat je toto ale neúnosné, kvůli náročné serializaci a náročnější údržbě v případě migrací. Vývojář se pak neobejde bez databáze. Pro implementaci databáze ve frameworku Kotlin Multiplatform existují dvě hlavní knihovny: **Realm** a **SQLDelight**.

Na platformě Android převládá knihovna Room, jež je součástí frameworku Android Jetpack a je doporučovaná společností Google (je součástí kolekce knihoven Android Jetpack, jež přestavují doporučované knihovny pro tvorbu aplikací pro platformu Android). Tato knihovna poskytuje abstraktní vrstvu nad knihovnou SQLite. Pomocí této knihovny vývojář píše kód v jazyce Kotlin a knihovna pak zařídí konverzi do dotazů v jazyce SQL. [66]

Momentálně se na jejím zpřístupnění pro platformu Kotlin Multiplatform pracuje, pokrok se dá sledovat přes Google issue tracker. [68]

Knihovna SQLDelight je pak nejvíce používaná SQL databáze na platformě Kotlin Multiplatform. Podporuje platformy Android, iOS, desktop, JVM a JavaScript. Podpora pro Kotlin/Wasm je ale nyní rozpracovaná a počítá se s tím, že do budoucna přibude. [56]

SQLDelight funguje na do jisté míry opačném základě než knihovna Room. Základem je automatické generování souborů v jazyku Kotlin na základě SQL příkazů, které se nachází v souborech s příponou `.sq`. V těchto souborech vývojář napíše název dané funkce, jež se má vygenerovat a pod ní příkaz v jazyce SQL. Tato knihovna rovněž poskytuje pluginy do IDE od společnosti JetBrains, který poskytuje funkce jako zvýrazňování syntaxe, doplňování textu a další pro .sq soubory. Mimo to poskytuje možnost sledovat v reálném čase stav databáze na zařízení. [13]

Výpis kódu 2 představuje vytvoření tabulky `Character` a vydefinování funkcí insertCharacter, removeAllCharacters, selectAllCharacters, kde pod každou z nich je napsán dotaz v jazyce SQL, který se má provést. Knihovna SQLDelight po sestavení projektu vygeneruje třídu v Kotlinu představující databázi, která obsahuje třídu Queries, jenž obsahuje funkce v jazyce Kotlin pojmenované stejně jako naše vydefinované. Přes tyto funkce můžeme pracovat s entitami z databáze v kódu v Kotlinu.

2.3.3 Networking

Asi každá aplikace v dnešní době dostává nějaká data z webu. Na platformě Android je velmi populární klient Retrofit. Vzhledem k tomu, že je tato knihovna psaná v programovacím jazyku Java, nemůže být použita v KMP projektu.

Na této platformě je rozhodně nejpoužívanější klient Ktor. Tento framework je vyvíjen společnosti JetBrains, první stabilní verze vyšla v roce 2018. Nabízí jak klient, tak server, pro mnoho platforem. Multiplatformní HTTP klient je asynchronní, implementován s využitím coroutines. Mezi jeho funkcionality patří mechanismus pro tvorbu URL pomocí typů, nastavování metrik, možnost persistence dat mezi requesty díky sessions a podpora logování. [54]

Tato knihovna je jednoduchá na použití, na výpisu kódu 3 je vidět, že díky pár řádkům kódu se dá lehce vydefinovat HTTP klient.

Podpora pro Kotlin/Wasm je v době psaní této práce ještě rozpracovaná, těsně před vydáním. Do plného vydání je k dispozici rozpracovaná verze 3.0.0-wasm2. Hlavní důvod, proč tato podpora ještě není, je závislost na `kotlinx.coroutines` verze 1.8.0, která ale nyní již Kotlin/Wasm podporuje. [55]

[4] `https://github.com/russhwolf/multiplatform-settings`

```
 CREATE TABLE Character (
    id INTEGER NOT NULL,
    name TEXT NOT NULL,
    status TEXT,
    imageUrl TEXT NOT NULL
);

insertCharacter:
INSERT INTO Character(id, name, status, imageUrl)
VALUES(?, ?, ?, ?);

removeAllCharacters:
DELETE FROM Character;

removeCharacterById:
DELETE FROM Character
WHERE id = ?;

selectAllCharacters:
SELECT Character.*
FROM Character;
```

Výpis kódu 2 Ukázka .sq souboru

```
val httpClient: HttpClient =
HttpClient {
    install(ContentNegotiation) {
        json(Json { ignoreUnknownKeys = true })
    }
}
```

Výpis kódu 3 Ukázka definice HTTP klientu pomocí knihovny Ktor

2.3.4 Dependency Injection

U větších aplikací pro platformu Android se hojně užívá koncept Dependency Injection. Přímo na této platformě se používají frameworky Dagger (starší, dnes již zřídkakdy používané), Hilt (nadstavba nad frameworkem Dagger, jednodušší na použití) a Koin (postavený na jazyku Kotlin bez závislostí na systému Android). Pro Android vývojáře je převážně doporučovaný framework Hilt.

Jak jsem se zmiňoval dříve, Hilt není k dispozici pro Compose Multiplatform, stejně tak ani Dagger kvůli závislostem na platformně specifické funkcionality platformy Android. Nicméně, framework Koin pro něj podporu má.

Koin poskytuje jednoduchou možnost, jak implementovat dependency injection v jakékoliv aplikaci v jazyku Kotlin. Dosahuje toho díky jednoduchému DSL (Domain Specific Language) v jazyce Kotlin. Pro platformu Android navíc poskytuje specifická klíčová slova, například `viewmodel`, díky kterému je použití ještě jednodušší. Stačí pak na začátku chodu aplikace zavolat funkci `startKoin`, a framework se o vše postará. [18]

Oproti výše zmiňovaným má Koin nevýhodu v tom, že závislosti kontroluje až za běhu programu, tudíž zapomenutá deklarace se může projevit až pádem aplikace. Na druhou stranu se ale těmto situacím dá předejít díky ověření konfigurace – framework poskytuje funkce `verify` a `checkModules`, pomocí kterých se dá ověřit, že jsou vydefinovány všechny závislosti. [19]

Kapitola 3
Analýza existujících řešení

V této kapitole se budu věnovat analýze již existujících frameworků určených k tvorbě multiplatformních aplikací. Konkrétně se zaměřím na frameworky **Flutter**, **React Native** a **PWA**.

3.1 Flutter

Flutter je multiplatformní UI framework od společnosti Google. Vyšel v beta verzi 27. 2. 2018 při příležitosti **Mobile World Congress**. V době vydání tento framework cílil na platformy Android a iOS. Tento framework by měl spojovat výkonnost a integraci jako v případě nativních mobilních aplikací, ale zároveň multiplatformní dosah jako u UI toolkitů. Jakožto největší jeho výhody společnost zdůrazňuje:

- rychlost vývoje díky funkcionalitám jako `Hot reload` (viz podsekce 3.1.3),
- flexibilní design díky obsáhlým designovým knihovnám s widgety a animacemi,
- dostatečný výkon díky přenosnému graficky akcelerovanému rendereru a nativnímu ARM runtime.

Již v této době měl framework podporu v podobě pluginů ve vývojářských prostředích Visual Studio Code a Android Studio. Také zde byla podpora více než 1000 balíčků pro Flutter. [79]

3.1.1 Preferované IDE

Dokumentace se zmiňuje, že pro aplikace psané v tomto frameworku stačí jakékoliv IDE nebo textový editor společně s nástroji pro příkazovou řádku. Nicméně tým spravující framework Flutter doporučuje editor, který podporuje pluginy pro Flutter, jako Visual Studio Code nebo Android Studio. Tyto pluginy poskytují možnost kompletace kódu, podporu při ladění, pomoc s editováním widgetů a další funkcionality. Dále dokumentace poskytuje návod, jak zprovoznit IDE společně s pluginem pro Flutter. [39]

3.1.2 Dart

Dart je programovací jazyk, který se využívá při vývoji aplikací ve frameworku Flutter. Tento programovací jazyk podporuje asynchronní zpracování kódu díky `aync/await`, má plnou podporu pro objektově orientované programování. Od verze 2.12 podporuje `null safety`. [42]

3.1.3 Hot reload

Funkce Hot reload ve frameworku Flutter pomáhá k rychlejší implementaci, experimentování a ladění při tvorbě UI. Funguje díky injektování aktualizovaných zdrojových kódů do běžící Dart Virtual Machine. Po aktualizaci framework znovu sestaví strom widgetů, díky čemuž vývojář rychle uvidí své změny. K použití stačí spustit aplikaci v debug módu a využití jednoho z podporavaných IDE (viz 2.2.2).

3.1.4 Podpora platforem

K vydané verzi 3.19.3 framework Flutter podporuje platformy Android, iOS, macOS, Windows, Debian (Linux), Ubuntu (Linux) a webové platformy Chrome, Firefox, Safari a Edge. Flutter rozděluje podporu pro tyto platformy do tří úrovní: Podporované, Maximální úsilí a Nepodporované. Podporovaná úroveň znamená, že tým spravující framework Flutter testuje tyto platformy při každém commitu. Maximální úsilí znamená, že se tým Flutter snaží tyto platformy podporovat a testuje na nich ad-hoc. Nepodporované platformy nejsou podporované a tým na nich netestuje. V dokumentaci se nachází tabulka, ve které jsou u každé platformy vypsané podporované HW architektury a informace o tom, které verze konkrétní platformy jsou zahrnuty v jednotlivých podporovaných úrovních. [27]

3.1.5 Podpora balíčků

Platforma Flutter má obrovské množství balíčků, které můžou vývojáři ve svých aplikacích používat. Přímo společnost Google spravuje web **pub.dev** [80], kde se všechny balíčky nachází. V době psaní této práce se zde nachází více než 42 tisíc balíčků. Vývojář má možnost mezi balíčky hledat dle názvu, přímo na stránce se nachází i žebříčky s nejpopulárnějšími balíčky, top balíčky, dále pak s takzvanými „Flutter Favorites“ – balíčky, které dosahují té nejvyšší kvality, jsou vybírány Flutter Ecosystem Committee. Mimo to také na platformě YouTube vychází oficiální videa **Package of the Week**[1]) (nutno však podotknout, že nevychází každý týden, jak by se dle mohlo nabízet, videa vychází v průměru jednou za měsíc). Každý balíček má u sebe spoustu informací. Mezi základní z nich patří to, pro které platformy je určen, dále pak zda je kompatibilní s 3. verzí programovacího jazyku Dart a odkaz na vývojářský účet, který je vydal (ten může být ověřený, v takovém případě má u sebe zvýrazněnou ikonku). [80]

Aby se mohl vývojář lépe zorientoval a mohl si vybírat z těch nejkvalitnějších a nejoblíbenějších balíčků, je zde zavedený skóring. Každý balíček má informaci o počtu **liků**, **popularitě** a o počtu **pub pointů**. Like může udělit balíčku libovolný vývojář, balíček se tak uloží do oblíbených balíčků a vývojář jej může najít na svém seznamu. Popularita je měřená podle počtu aplikací, které na daném balíčku závisí během posledních 60 dnů. 100 procent znamená, že se jedná o jeden z top 1 procenta nejpoužívanějších balíčků, naopak 0 procent značí nejméně používané balíčky. Pub points se používají k měření kvality, jsou posuzovány v šesti kategoriích, hodnotí se například dodržování konvencí Dart balíčků (například jsou nutné soubory `pubsec.yaml`, `README.MD`, `CHANGELOG.MD` a licence), dokumentace k balíčků, podpora platforem (čím více, tím lépe) či kvalita kódu díky statické analýze. [81]

3.1.6 Widget

Ve frameworku Flutter se uživatelské rozhraní vytváří deklarativně. V tomto případě se jedná o strom komponent, zvaných widgety, s jedním kořenem. Kořenový widget ve výchozím chování zabírá celou plochu, která je k dispozici. Widgety ve frameworku Flutter často dědí ze tříd `StatelessWidget` nebo `StatefulWidget`, podle toho, jestli má daný widget nějaký stav. Základní

[1] `https://www.youtube.com/playlist?list=PLjxrf2q8roU1quF6ny8oFHJ2gBdrYN_AK`

funkcionalita widgetu spočívá v implementaci `build` funkce, která definuje widget pomocí nižších widgetů. Mezi základní widgety patří `Text`, `Row`, `Column`, `Stack` či `Container`. [28]

Rozdělení na `StatelessWidget` a `StatefulWidget` je zde z důvodu optimalizace – ne každý widget musí uchovávat stav, tedy jen ty, které tak činí se pak musí překreslit [28].

3.1.7 Ukázka

Obrázek 3.1 představuje jednoduchý layout, při jeho návrhu jsem se snažil co nejvíce se přiblížit ukázce v Compose Multiplatform (viz sekce 2.2.4).

Výpis kódu 4 se skládá z jednoduchého layoutu, kde se nachází dvě textová pole a obrázek načtený z `URL`. Jsou zde použity komponenty (widgety) `Expanded`, `Padding`, `Row`, `ClipRRect`, `Semantics`, `SizedBox`, `Image`, `Column` a `Text`. Texty jsou zde stylizované dle definovaného tématu.

Obrázek 3.1 Jednoduchý layout obsahující obrázek a dva texty

3.1.8 Komponenty

V ukázce je použito hned několik widgetů. Komponenty `Row` a `Column` mají stejnou funkci jako ve frameworku Compose Multiplatform, určují tedy kompozici prvků na obrazovce. Ve stromě vykreslených widgetů je jeden widget v kořeni. Widgety Row a Column pak mají jako parametry `children`, díky nim se tedy mohou přidávat další widgety do stromu. `Row` se používá pro prvky umístěné horizontálně, `Column` pro ty umístěné vertikálně. Pokud chce uživatel umisťovat na sebe, může k tomu využít widget `Stack`.

Widget `Padding` patří mezi ty nejpoužívanější, stará se totiž o vykreslování mezer mezi prvky. K vydefinování mezer a jejich velikostí se pak používá třída `EdgeInsets`. Widget `SizedBox` určuje velikost obsahu. `Expanded` widget umožňuje natažení widgetů na maximální možnou velikost. Dále je zde použitý widget `ClipRRect`, který slouží k oříznutí obsahu k obdélníku se zakulacenými rohy (rounded-rectangular clip, z toho jeho poněkud kryptický název).

Widget `Image` je součástí standardní knihovny, podporuje vykreslování obrázků a fotek z URL. Sám o sobě neposkytuje možnost vyplnit popisek, který je důležitý pro uživatele se zrakovým postižením. K vyplnění popisku může vývojář použit widget `Semantics`, kterému jako parametr `child` předá daný Image.

Dalším použitým widgetem je widget `Text`, jenž se stará o vykreslení textu. Povinný argument je text, tedy řetězec, který se má vykreslit. Mimo to lze upravovat vzhled textu, použití je velice podobné jako v případě composable funkce Text (viz ukázka kódu 2.2.5).

3.1.9 Material Design

Všechny aplikace psané ve frameworku Flutter mají definované téma. Každé téma definuje barvy, styly typografie a další parametry pro Material komponenty. Vývojář může mimo jiné rozšířit téma aplikace či stylizovat téma specifického widgetu. [37]

Tento framework využívá pro téma systém Material Design. Od verze 3.16 se ve Flutteru využívá jako výchozí Material Design verze 3. Vývojáři sice zatím můžou využívat Material Design 2,

```
return Expanded(
    child: Padding(
        padding: const EdgeInsets.all(8),
        child: Row(
          children: [
            ClipRRect(
              borderRadius: BorderRadius.circular(8),
              child: SizedBox.fromSize(
                  size: const Size.fromRadius(32),
                  child: Semantics(
                    label: 'avatar',
                    child: Image.network(iconUrl, fit: BoxFit.cover),
                  )),
            ),
            Padding(
              padding: const EdgeInsets.all(8),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(character.name,
                      style: Theme.of(context).textTheme.titleMedium),
                  Text(character.status,
                      style: Theme.of(context).textTheme.bodyMedium),
                ],
              ),
            ),
        ],
)));
```

Výpis kódu 4 Ukázka layoutu ve frameworku Flutter

pomocí nastavení příznaku `useMaterial3` na false. Nicméně, jak se upozorňuje v dokumentaci, v budoucnu tato verze již nebude podporovaná. V dokumentaci se nachází sekce pro vývojáře, které čeká přechod na Material Design 3. [37]

3.1.10 Dokumentace

Framework Flutter má opravdu pokročilou dokumentaci, která poskytuje spoustu důležitých informací pro tvorbu multiplatformních aplikací.

Nachází se zde například sekce „From another platform?“, ve které jsou články určené pro vývojáře přicházející z jiných platforem a frameworků. V nich jsou mimo jiné popsány rozdíly mezi frameworky a ukázky, jak napsat dané konstrukty ve Flutteru. Jmenovitě jsou zde sekce pro vývojáře zvyklé programovat pro platformy Android, iOS (jak varianta pro ty, kteří využívali UIKit, tak SwiftUI) či web. Dále pak pro ty znalé frameworku React Native a Xamarin.

Dále se zde nachází sekce ohledně platformní adaptace, která vyzdvihuje rozdíly mezi platformami Android a iOS. V této sekci je popsáno chování, které je pro tyto platformy rozdílné a které se tento framework snaží dodržet. Mezi nejzásadnější rozdíly patří například přechody při navigaci mezi obrazovkami. Na platformě Android a iOS jsou použity jiné přechody (na Androidu nová obrazovka vzejde ze spodku obrazovky, na iOSu pak ze strany). Mimo to jsou mezi platformami rozdíly například mezi ikonami, texty a chováním scrollování. [30]

3.1.11 Práce se zdroji

Framework Flutter podporuje zdroje v aplikaci. V souboru `pubsec.yaml` jsou vývojářem vydefinovaná umístění zdrojů. Konkrétní příklad je vidět na ukázce kódu 5. V kódu se pak přistupuje ke zdrojům pomocí objektu `AssetBundle`. Flutter poskytuje podporu pro obrázky různých velikosti, aby se vykreslila v závislosti na rozlišení displeje ta nejvhodnější. [36]

Bohužel tento framework nepodporuje ikony v svg či jiném vektorovém formátu. Nicméně, jsou k dispozici knihovny, které tuto funkcionalitu poskytují. Jedna z nejoblíbenějších je knihovna flutter_svg[2].

```
assets:
    - assets/icons/
```

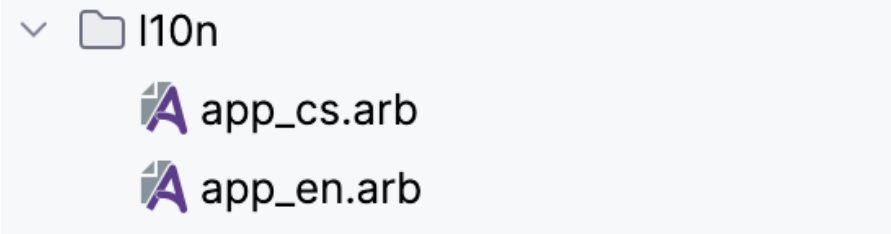
Výpis kódu 5 Vydefinované umístění zdrojů v souboru `pubsec.yaml`

O lokalizaci v aplikaci se stará balíček `Flutter_localizations`. K použití tohoto balíčku stačí přidat do kořenového adresáře soubor `l10n.yaml`, ve kterém jsou vydefinovány informace o umístění App Resource Bundle (.arb) souborů, které obsahují překlady řetězců. Dále je zde informace o názvu souboru, do kterého se mají lokalizované texty vygenerovat. V daných souborech pak jsou tyto překlady definovány pomocí formátu JSON (klíč, hodnota). Jednotlivé jazykové varianty se liší dle koncovky využívající mezinárodní jazykovou zkratku. Na obrázku 3.2 je vidět struktura složky, jež obsahuje český a anglický překlad řetězců aplikace. [35]

3.1.12 Podpora základních funkcionalit

Jak jsem se již zmiňoval, pro framework Flutter existuje spousta knihoven, které můžou ušetřit vývojáři práci. Samotný framework ale nabízí podporu pro nejdůležitější funkcionality multiplatformních aplikací. Na rozdíl od frameworku Compose Multiplatform je zde podpora pro zobrazování fotek a obrázků z URL nabízena ve standardní knihovně. Stará se o ní widget `Image`.

[2] `https://pub.dev/packages/flutter_svg`



Obrázek 3.2 Příklad složky obsahující český a anglický překlad

Tým starající se o framework Flutter (přezdívaný Dart Team) navíc doplňuje standardní knihovnu pomocí několika dalších balíčků na platformě pub.dev[3]. Mezi nejdůležitější z nich patří balíčky pro podporu navigace v aplikaci, logování a networking.

Framework Flutter dokonce nabízí dva různé systémy pro navigaci v aplikaci. V případě menších aplikací, které nejsou propojené přes deep linky (možnost otevírat specifickou část aplikace přes URL) se doporučuje třída `Navigator`. Větší aplikace s komplexnější navigací by pak měly využívat třídu `Router`. Router také nabízí podporu pro historii prohlížeče. [32]

Logování je obstaráváno `log` funkcí z balíčku `dart:developer`. Pokročilejší volby (které mimo jiné podporují různé úrovně logování) jsou rovněž k dispozici – jednoznačně nejoblíbenější je knihovna `logger`[4]. Má jednoduché použití a je inspirována logovacím systémem platformy Android.

Networking zajišťuje knihovna `http`[5]. Její základní využití je velmi jednoduché, pomocí třídy `http.Client`, která poskytuje dané metody. V závislosti na platformě si vývojář může vybrat implementaci klienta, poskytující různé funkcionality. Jednoduché použití je vidět na ukázce kódu 6.

```
var httpClient = http.Client();
final response = await httpClient.get(
  Uri.parse('https://rickandmortyapi.com/api/character/?page=$page'),
);
```

Výpis kódu 6 Ukázka použití HTTP klienta

Pro implementaci Dependency Injection v tomto frameworku jde využít třídu InheritedWidget[6]. Bohužel nemá úplně nejjednodušší použití, vývojář musí psát spoustu kódu okolo. Naštěstí ale existují populární balíčky, které tuto funkcionalitu poskytují, například `provider`[7] nebo `get_it`[8].

3.1.13 Lokální úložiště

Pro ukládání menších dat, jako jsou uživatelské preference, nabízí framework Flutter plugin `shared_preferences`[9]. Podporuje základní datové typy (`int`, `double`, `bool`, `String`, …). V implementaci používá platformní řešení. Je dostupná na všechny platformy.

V případě složitějších dat ukládaných do databáze je k dispozici několik možností, většina z nich je postavena na základech knihovny SQLite. Zdaleka nejpoužívanější a oficiálně doporu-

[3] `https://pub.dev/publishers/dart.dev/packages`
[4] `https://pub.dev/packages/logger`
[5] `https://pub.dev/packages/http`
[6] `https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html`
[7] `https://pub.dev/packages/provider`
[8] `https://pub.dev/packages/get_it`
[9] `https://pub.dev/packages/shared_preferences`

čovaná je knihovna `sqflite`. Podporuje platformy Android, iOS, MacOS. Podporu webu nabízí indexovaná databáze `idb_sqflite`[10] či v experimentální podobě `sqflite_common_ffi_web`[11], všechny tři od stejného vývojáře.

Pomocí knihovny `sqflite` se dá napřímo pracovat s databází přes SQL příkazy. Pro konverzi modelů v aplikaci do databáze si musí vývojář implementovat mapovací funkci. Jednoduchý příklad takovéto třídy a vkládání do databáze je v ukázce kódu 7. [34]

```
class Dog {
  final int id;
  final String name;
  final int age;

  Dog({
    required this.id,
    required this.name,
    required this.age,
  });

  // mapovací funkce
  Map<String, Object?> toMap() {
    return {
      'id': id,
      'name': name,
      'age': age,
    };
  }
}

Future<void> insertDog(Dog dog) async {
  // reference na databázi
  final db = await database;

  // vkládání, použití conflictAlgorithm, aby se nevložila
  // stejná položka dvakrát
  await db.insert(
    'dogs',
    dog.toMap(),
    conflictAlgorithm: ConflictAlgorithm.replace,
  );
}
```

Výpis kódu 7 Ukázka modelové třídy a vkládání do databáze [34]

3.2 React Native

Framework React Native vydala roce 2015 společnost Facebook. První verze podporovala platformu iOS, podpora pro Android vyšla později. Je založen na frameworku React, se změnami nutnými pro vývoj pro mobilní zařízení. Mezi důvody, které Facebook vedly k vývoji multiplat-

[10] `https://pub.dev/packages/idb_sqflite`
[11] `https://pub.dev/packages/sqflite_common_ffi_web`

formního frameworku, zmiňuje, mimo jiné, pomalý vývoj nativních mobilních aplikací oproti webu či použití reaktivního UI pro vývoj. [45]

3.2.1 Programovací jazyk

Framework je založen na programovacím jazyku JavaScript. React Native podporuje jak JavaScript, tak i TypeScript (programovací jazyk, který rozšiřuje JavaScript o podporu typů). U nových projektů je TypeScript brán jako výchozí, vývojář ale může využívat JavaScript, pokud chce. Navíc lze tyto jazyky kombinovat v rámci projektu. [48]

3.2.2 Podpora platforem

Oficiálně či s podporou komunity nyní React Native podporuje poměrně dost platforem. Přímá podpora je pro Android a iOS, od partnerů pak pro macOS a Windows. Komunita podporuje React Native pro tvOS (Apple TV a Android TV), Web (vykreslování přes React DOM) a React Native Skia (používá Skia pro renderování, podpora pro Linux a macOS). [49]

3.2.3 Podpora knihoven

Pro framework React Native existuje velké množství knihoven. Komunita kolem tohoto frameworku vytvořila stránku reactnative.directory[12]. Slouží k tomu, aby vývojáři věděli o dostupných knihovnách na jednom místě. Mimo to poskytuje možnost filtrování (podle platforem, statusu či typu). Dále je možnost řazení podle různých kritérií. Momentálně se zde nachází skoro 1400 knihoven. Odkazy na tuto stránku se nachází i přímo v oficiální dokumentaci. [44]

3.2.4 Ukázka

Ukázka kódu 8 představuje jednoduchý layout, který vyobrazuje kartu s postavou, stejnou jako v ukázce v podsekci s frameworkem Flutter 3.1.7). Jedná se tedy o layout obsahující dvě textová pole a obrázek načtený z URL.

V ukázce jsou použity komponenty `View`, `Text`, `Image` a `Card`.

3.2.5 Komponenty

Komponenta `View` je jedna z nejzákladnějších, které se ve frameworku nachází. Stará se o kompozici prvků na obrazovce (není zde rozdělení na vertikální a horizontální komponenty jako v případě Flutteru či Compose Multiplatform). O vykreslení textu se stará komponenta `Text`. Komponenta `Image` je použita k vykreslování obrázků z URL, nabízí v základu poměrně široké možnosti jako nastavení obrázku v případě problémů s načtením či `accessibilityLabel` pro popis obrázku pro osoby se zrakovým postižením.

Výše zmiňované komponenty jsou součástí standardní knihovny. Komponenta `Card` je součástí knihovny React Native Paper, o které se zmiňuji v následující podsekci. `Card` je použita pro implementaci karty, která vychází z Material Design systému.

Ke stylizování komponent se využívá `StyleSheet`, v němž se definují parametry podobným způsobem jako v případě CSS.

[12] `https://reactnative.directory`

```
return (
    <Card
        onPress={handleTap}
        onLongPress={handleLongPress}
        style={styles.card}>
        <View style={styles.row}>
            <Image
                source={{uri: character.iconUrl}}
                style={styles.image}
            />
            <View style={styles.column}>
                <Text style={{ color: theme.colors.onSurface }}>
                    {character.name}
                </Text>
                <Text style={{ color: theme.colors.onSurface }}>
                    {character.status}
                </Text>
            </View>
        </View>
    </Card>
);
```

Výpis kódu 8 Ukázka layoutu ve frameworku React Native

3.2.6 Material design

Framework React Native sám o sobě poskytuje pouze pár základních komponent. Ke stylování se používá parametr `style`, který obsahují všechny komponenty. Jména a hodnoty s stylů odpovídají těm vydefinovaným v CSS. Jediný rozdíl je v tom, že se používá camelCase. Pro pokročilejší styly se doporučuje použít `StyleSheet.create` pro definování více stylů na jednom místě. Doporučuje se, aby vývojářem vytvořené komponenty přijímaly parametr `style`. [51]

Framework přímo nepodporuje žádný design systém, stylování je ponecháno na vývojáři. Material Design tedy nemá přímou podporu, musí se použít knihovna. Pro Material Design 2 existuje několik knihoven, mezi nejpoužívanější patří knihovna MUI[13].

V případě Material Design 3 ale většina knihoven podporu nemá. Nejoblíbenější knihovna, která má podporu, je knihovna React Native Paper[14]. Poskytuje většinu potřebných komponent ze systému Material Design 3, má podporu pro témata a ikony. Nabízí mimo jiné i komponenty `AppBar` a `BottomBar`, které lze kombinovat s navigačními komponenty z ostatních knihoven. K použití této knihovny stačí obalit aplikaci v komponentě `PaperProvider`.

Některé komponenty a funkcionality zde ale chybí, dle dokumentace proto, že již jsou dobře implementovány někým jiným. Mezi chybějící funkce patří podpora pro téma vytvořené ze systémových barev zařízení. Chybějící komponenty jsou například BottomSheet či taby. [52]

3.2.7 Dokumentace

Dokumentace je poměrně rozsáhlá, ačkoliv framework sám o sobě neposkytuje tolik funkcionalit jako například framework Flutter. Nachází se v ní návody, jak poskytnout základní funkce jako zobrazování dat z URL, dostávání dat z URL či jak poskytnout navigaci v aplikaci, i když

[13] `https://mui.com/`
[14] `https://reactnativepaper.com/`

navigace není součástí základu, ale externí knihovny. Následně se zde nachází informace o tom, jak testovat aplikaci, jak měřit a zlepšovat výkon či jak stylizovat komponenty.

V případě mobilních platforem jsou zde návody, jak vydávat aplikaci pro danou platformu, jak využít interoperabilitu s nativním kódem či jak spustit aplikaci na simulátoru. [50]

3.3 Expo

Velmi využívaný a doporučovaný v oficiální dokumentaci je framework Expo. Jedná se o open-source framework pro aplikace psané pro Android, iOS a web. Poskytuje funkcionality vhodné pro urychlení vývoje jako možnost aktualizovat aplikaci na zařízení (podobně jako Hot reload ve frameworku Flutter viz sekce 3.1.3) či instantní sdílení aplikace. Může se přidat do jakékoliv aplikace psané ve frameworku React Native. Součástí jsou knihovny v rámci Expo SDK a CLI (command line interface). Má poměrně rozsáhlou dokumentaci, která svým rozsahem odpovídá té oficiální pro React Native.

Mimo to obsahuje i nepovinnou součást skládající se z cloudových služeb, pod názvem EAS (Expo Application Services). EAS usnadňuje sestavování aplikace, vydávání v obchodech s aplikacemi. Základní použití je zdarma, pokročilé funkce a vyšší kvóty jsou k dispozici v placeném předplatném. [46]

3.3.1 Práce se zdroji

Framework React Native podporuje obrázky ve formátu png, s možností vydefinovat přípony určující velikost (například `@2x`, což znamená že ikona je ve dvojnásobné kvalitě). V případě vektorových formátů se musí využít knihovna.

Například pro formát svg je doporučována knihovna react-native-svg[15], která se v nachází v základu frameworku Expo. Díky této knihovně může vývojář psát kód pro vykreslení svg ikony rovnou do komponenty, stačí ji obalit v `Svg` elementu.

Pro lokalizaci se opět musí použít knihovna, mezi nejpoužívanější patří knihovny react-intl[16] a react-native-i18n[17]. Druhá jmenovaná implementuje standardizovaný i18n. Díky tomu lze přidávat lokalizované texty do souborů ve formátu JSON, podobně jako v případě frameworku Flutter viz sekce 3.1.11.

3.3.2 Podpora základních funkcionalit

Základní funkcionality frameworku React Native jsou velice omezené bez použití knihoven či dalších frameworků, jako například již zmiňovaného frameworku Expo.

Navigaci v aplikaci framework nepodporuje, neobsahuje žádné typy menu mezi svými komponentami. Nicméně, nejvíce používaná je knihovna React Navigation[18]. Tato knihovna poskytuje platformně specifické chování, jednoduché použití a je open-source. V oficiální dokumentaci je pak jednoduchý návod, jak ji použít.

Networking, stejně jako načítání obrázků a fotek z URL, je pak součástí standardní knihovny. O networking se stará Fetch API[19]. K zobrazení obrázků z internetu lze využít komponentu `Image`[20], které se předá jako parametr URL.

K logování se dá použít na webové platformě dobře známé `console.log` či použít knihovnu s pokročilými funkcemi, například react-native-logs[21].

[15] `https://github.com/software-mansion/react-native-svg`
[16] `https://www.npmjs.com/package/react-intl`
[17] `https://www.npmjs.com/package/react-native-i18n`
[18] `https://reactnavigation.org/`
[19] `https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API`
[20] `https://reactnative.dev/docs/images#network-images`
[21] `https://www.npmjs.com/package/react-native-logs`

3.3.3 Lokální úložiště

V případě ukládání menších dat byl k dispozici balíček AsyncStorage[22] již v základu. Tento balíček již ale není podporovaný, v dokumentaci se doporučuje použít některou z komunitních knihoven. Jedna z možností je knihovnu se stejnojmenným názvem[23], která spadá pod framework Expo. Podporuje všechny platformy včetně webu.

Pro ukládání dat do databáze se dají použít různé knihovny. Mezi nejpoužívanější se řadí knihovna Expo SQLite[24], která má podporu pro mobilní platformy, nepodporuje web. Tato knihovna také pracuje napřímo s SQL příkazy, jak je vidět v jednoduché ukázce 9.

```
public addCharacterToFavourites = async (character: CharacterModel):
Promise<void> => {
    return new Promise((resolve, reject) => {
        db.transaction(
            tx => {
                tx.executeSql(
                    'INSERT INTO Character (id, name, status, imageUrl,
                    isFavourite)
                    VALUES (?, ?, ?, ?, ?);',
                    [character.id, character.name, character.status,
                    character.iconUrl, 1],
                    (_, { rowsAffected }) => {
                        if (rowsAffected > 0) {
                            resolve();
                        } else {
                            reject(new Error('Failed to add character'));
                        }
                    },
                );
            },
            error => reject(error)
        );
    });
};
```

Výpis kódu 9 Ukázka práce s databází z knihovny Expo SQLite

3.4 PWA

Progressive Web Apps (PWA) jsou aplikace, vytvořené pomocí webových technologií, které mají uživateli poskytnout zážitek jako při použití nativních aplikací. PWA můžou běžet na několika platformách pomocí jednotného kódu. Můžou být nainstalovány na zařízení a můžou interagovat s ostatními aplikacemi v zařízení. PWA aplikace mohou být nainstalovány jak z obchodu s aplikacemi na dané platformě, tak přímo z webu. [71]

Stále se jedná o webové stránky, tudíž ke svému běhu potřebují webový prohlížeč. Mají také stejné funkce jako webové stránky, tedy HTML stránku, která načítá CSS a JavaScript. Mimo to má každá taková aplikace **Web app manifest** a **service worker**, které jsou nezbytné k offline funkcionalitě. [72]

[22] https://reactnative.dev/docs/asyncstorage
[23] https://docs.expo.dev/versions/latest/sdk/async-storage/
[24] https://docs.expo.dev/versions/latest/sdk/sqlite/

Z tohoto důvodu nebudu v této části rozebírat dané jazyky či využívané knihovny. Vzhledem k tomu, že lze udělat PWA takřka z každé webové stránky, rozeberu zde pouze části specifické pro PWA, které jsou stejné nezávisle na zvolený framework pro implementaci webové aplikace.

3.4.1 Web app manifest

Jedná se o povinnou část každé PWA, podléhá **Web Application Manifest** specifikaci. Používá se k tomu, aby poskytoval informace nutné k instalaci PWA do zařízení, tedy obsahuje jméno a ikonu aplikace. Jedná se o soubor ve formátu JSON, který poskytuje informace o dané webové aplikaci. Tento JSON objekt obsahuje klíče, kterým se říká **members**, povinné klíče jsou: `name`, `icons`, `start_url` a `display` nebo `display_override`. Mimo tyto povinné klíče manifest obsahovat například link na aplikaci v obchodě na dané platformě, pomocí klíče `related_applications` či barvu pozadí přes klíč klíče `background_color`. Ne všechny klíče jsou podporovány všemi prohlížeči, na dokumentaci Mozilly[25] je vidět tabulka se všemi klíči a s úrovní jejich podpory na nejpoužívanějších prohlížečích. [73]

Na ukázce kódu 10 je vidět jednoduchý manifest. Obsahuje název aplikace, zkrácený název, parametr display určující jak moc UI má být viděno, v tomto případě má aplikace vypadat jako samostatná aplikace s vlastním oknem, ikonou v launcheru, se status barem. Dále jsou zde vydefinovány ikony a odkaz na spjatou aplikaci v Google Play.

```
{
  "name": "Example app",
  "short_name": "Example",
  "start_url": ".",
  "display": "standalone",
  "background_color": "#fff",
  "description": "An example app",
  "icons": [
    {
      "src": "images/icon96.png",
      "sizes": "96x96",
      "type": "image/png"
    },
    {
      "src": "images/icon192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ],
  "related_applications": [
    {
      "platform": "play",
      "url": "https://play.google.com/store/apps/details?id=com.veprek.honza.example"
    }
  ]
}
```

Výpis kódu 10 Ukázka PWA manifestu

[25] `https://developer.mozilla.org/en-US/docs/Web/Manifest`

3.4.2 Service worker

Service Worker API slouží k propojení webových aplikací, prohlížeče a sítě. Primárně slouží k zajištění offline funkcionalit a zařízení správné funkcionality i když není připojení k internetu. Navíc mají přístup k push notifikacím a API pro synchronizaci na pozadí. Jedná se o JavaScriptový soubor, který může ovládat stránku, se kterou je spojený a pomocí úpravy navigačních a zdrojových požadavků. Nemá přístup k DOM a běží na jiném vlákně než JavaScript, který pohání aplikaci, je tedy neblokující. Z bezpečnostních důvodů běží pouze přes HTTPS. [74]

3.4.3 Workbox

Workbox představuje sadu open-source knihoven stvořených za účelem zjednodušení tvorby PWA od společnosti Google. Mezi populární moduly patří workbox-strategies, workbox-routing a workbox-precaching. Týkají se strategií cachování a přijímání požadavků. Workbox se dá použít několika způsoby, například přes CLI či npm modul. [62]

3.4.4 Dokumentace

Tím, že PWA je univerzální pojem pro webové aplikace poskytující uživateli podobný zážitek jako nativní aplikace, má vývojář na výběr více dokumentací. Nejrozsáhlejší jsou ty od společnosti Mozilla a Google. První jmenovaná poskytuje informace o těch nejdůležitějších pojmech, nabízí návody a tutoriály. Návody se týkají toho, jak poskytnout obsah offline, jak funguje cachování, jaké jsou best practices a jak se dají PWA instalovat. Nachází se zde tutoriál, který provede vývojáře kompletní tvorbou PWA a tutoriál studující zdrojový kód existující stránky. [61]

V případě společnost Google je dokumentace[26] ještě obsáhlejší – mimo výše zmiňovaného obsahuje i case studies a pokročilá témata. Mimo to obsahuje kurz[27], skládající se z 25 článků, který poskytuje všechny důležité informace pro tvorbu PWA.

[26] `https://web.dev/explore/progressive-web-apps`
[27] `https://web.dev/learn/pwa`

Kapitola 4
Návrh

V této části představím návrh aplikace, kterou budu implementovat ve dříve zmiňovaných frameworcích. Jedná se o aplikaci **Rick and Morty**, vycházející z domácího úkolu, který zadáváme na FIT na předmětu BI-AND.21[1]. Tato aplikace dostává data z API, jehož dokumentace je zde[2]. Konkrétně se jedná o `/character` endpointy.

Návrhy obrazovek vytvořené společností Ackee jsou veřejně k dispozici na platformě Figma zde[3]. Tyto návrhy ale dodržují systém Material design 2, což je starší verze systému Material Design, viz sekci 2.2.8. Z tohoto důvodu jsem vypracoval vlastní návrh na platformě Figma, který se řídí systémem Material Design 3. U výběru barevného tématu tohoto návrhu jsem se inspiroval i přímo seriálem *Rick and Morty*. Využil jsem nástroj Material Theme Builder[4], který poskytuje možnost vytvořit barevné schéma z fotky. Je zde možnost vyexportovat barvy v různých formátech, mimo jiné právě i ve formátech pro platformy Flutter a Jetpack Compose. V druhém jmenovaném případě se vyexportují barvy v souboru `Color.kt` a téma v souboru `Theme.kt`. Na obrázku 4.1 je vidět Theme Builder dialog. Plugin jsem napřímo využil ve svém projektu na Figmě, díky tomu se aplikovaly všechny barvy do použitých komponent.

Mnou implementovaný návrh v platformě Figma je k dispozici zde[5].

Jedná se o jednoduchou aplikaci sestávající ze seznamu postav ze seriálu *Rick and Morty*.

4.1 Obrazovky

Tato aplikace se skládá ze 4 základních obrazovek – `Seznam`, `Oblíbené`, `Detail` a `Vyhledávání`. Aplikace by měla podporovat všechny typy mobilních zařízení, měla by mít responzivní design, tedy vypadat rozumně jak na mobilech, tak na tabletech či tzv. ohebných telefonech. V případě webové platformy se primárně počítá s mobilními obrazovkami, ale aplikace by měla být schopna se přiměřeně škálovat i pro větší obrazovky monitorů.

4.1.1 Seznam

Tato obrazovka zobrazuje v seznamu postavy, každá položka obsahuje fotku postavy, dále její jméno a status (živá, mrtvá, nevíme). Po kliknutí na postavu se přejde na její detail. Při dlouhém

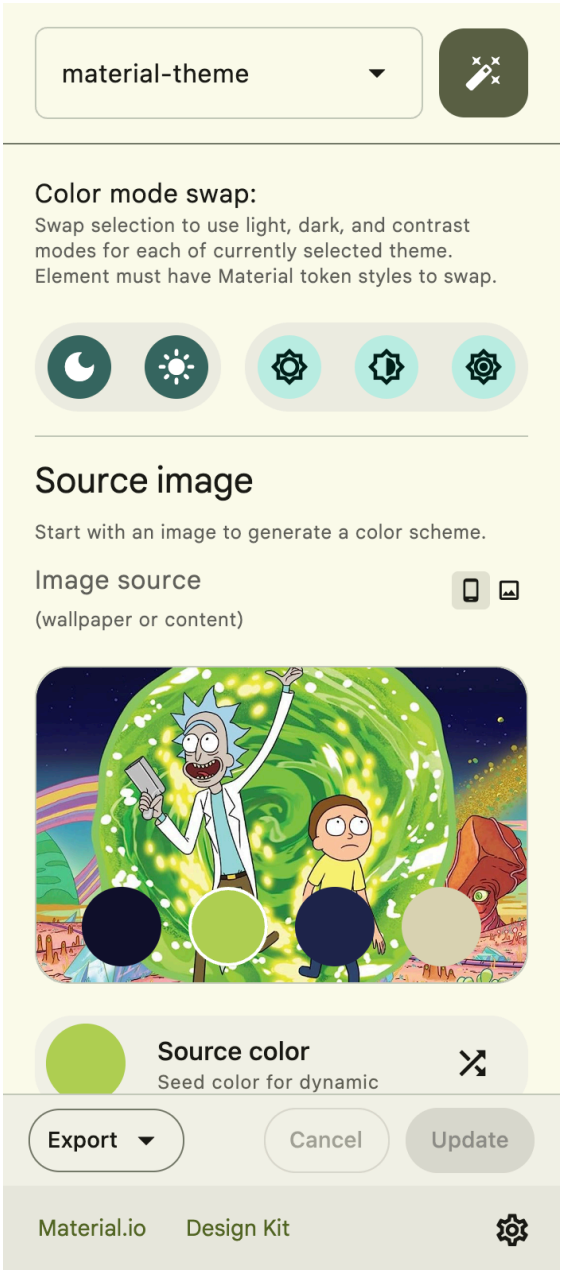
[1] `https://courses.fit.cvut.cz/BI-AND/`

[2] `https://rickandmortyapi.com/documentation`

[3] `https://www.figma.com/file/0w76BZ8TviO2TQF84574Et/Rick-and-Morty-Test-Task-Design?type=design&node-id=0-360&mode=design`

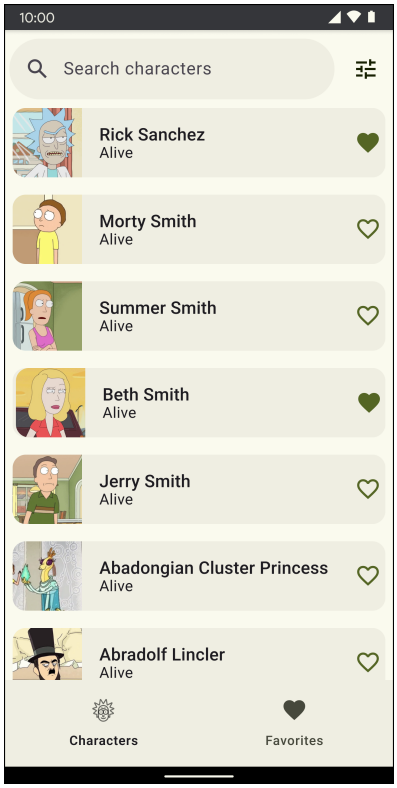
[4] `https://www.figma.com/community/plugin/1034969338659738588/material-theme-builder`

[5] `https://www.figma.com/file/TRpNre9ykkxxLMDKQcUwLP/Rick-and-Morty-Design?type=design&node-id=0%3A1&mode=design&t=GReR264KcQjqelHo-1`



Obrázek 4.1 Material Theme Builder s barvami dle vybraného obrázku

stisku či kliknutí na ikonu srdce se pak označí jako oblíbená. Oblíbené postavy jsou zvýrazněny vybarveným srdcem, neoblíbené obrysem srdce.



Obrázek 4.2 Obrazovka se seznamem všech postav

4.1.2 Oblíbené

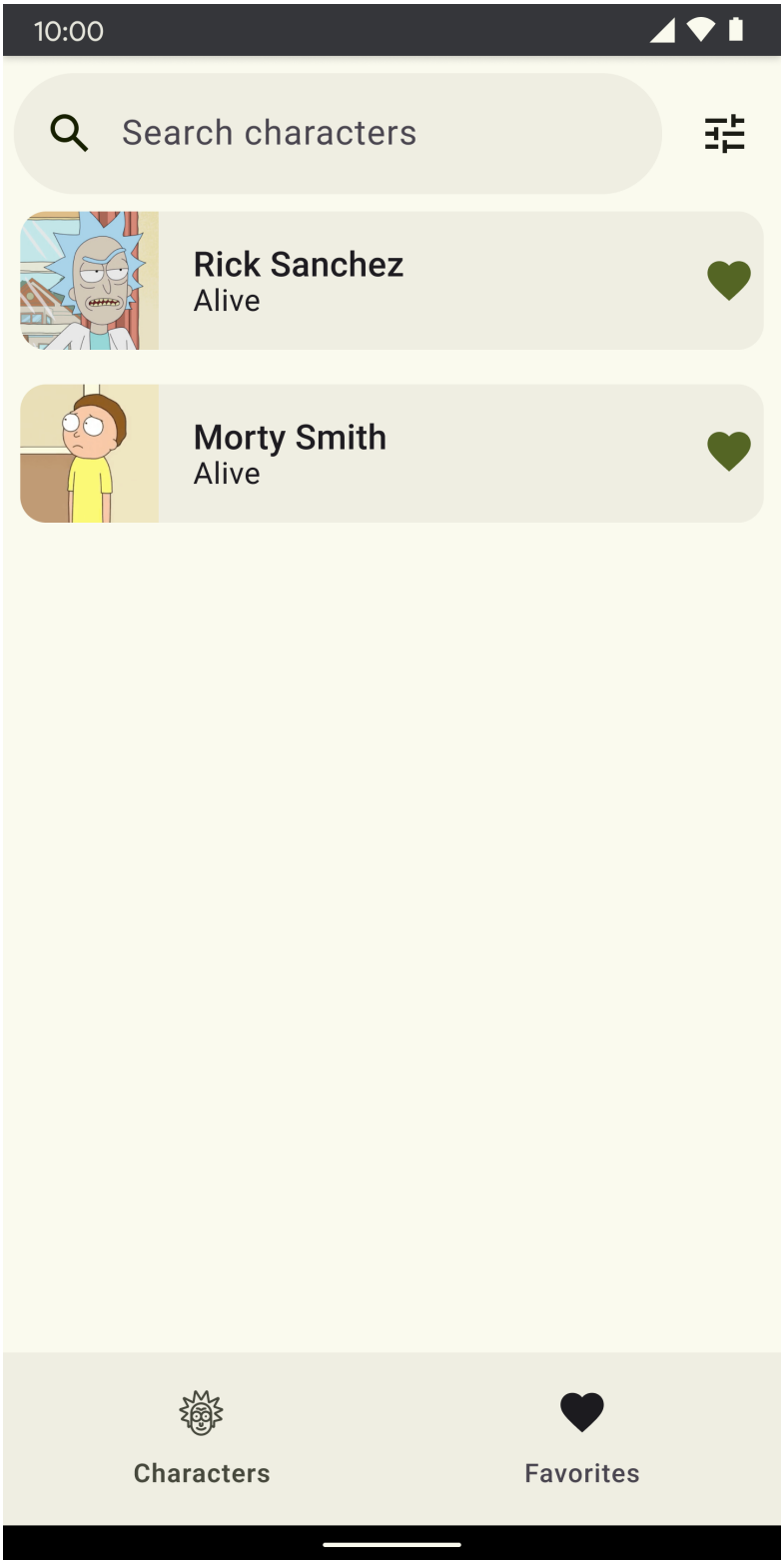
Jedná se o takřka totožnou obrazovku jako Seznam. Rozdíl je pouze v tom, že se zde nachází jen postavy, které byly přidány mezi oblíbené. Oblíbené postavy by si měla aplikace pamatovat – měly by se ukládat do perzistentního úložiště.

4.1.3 Detail

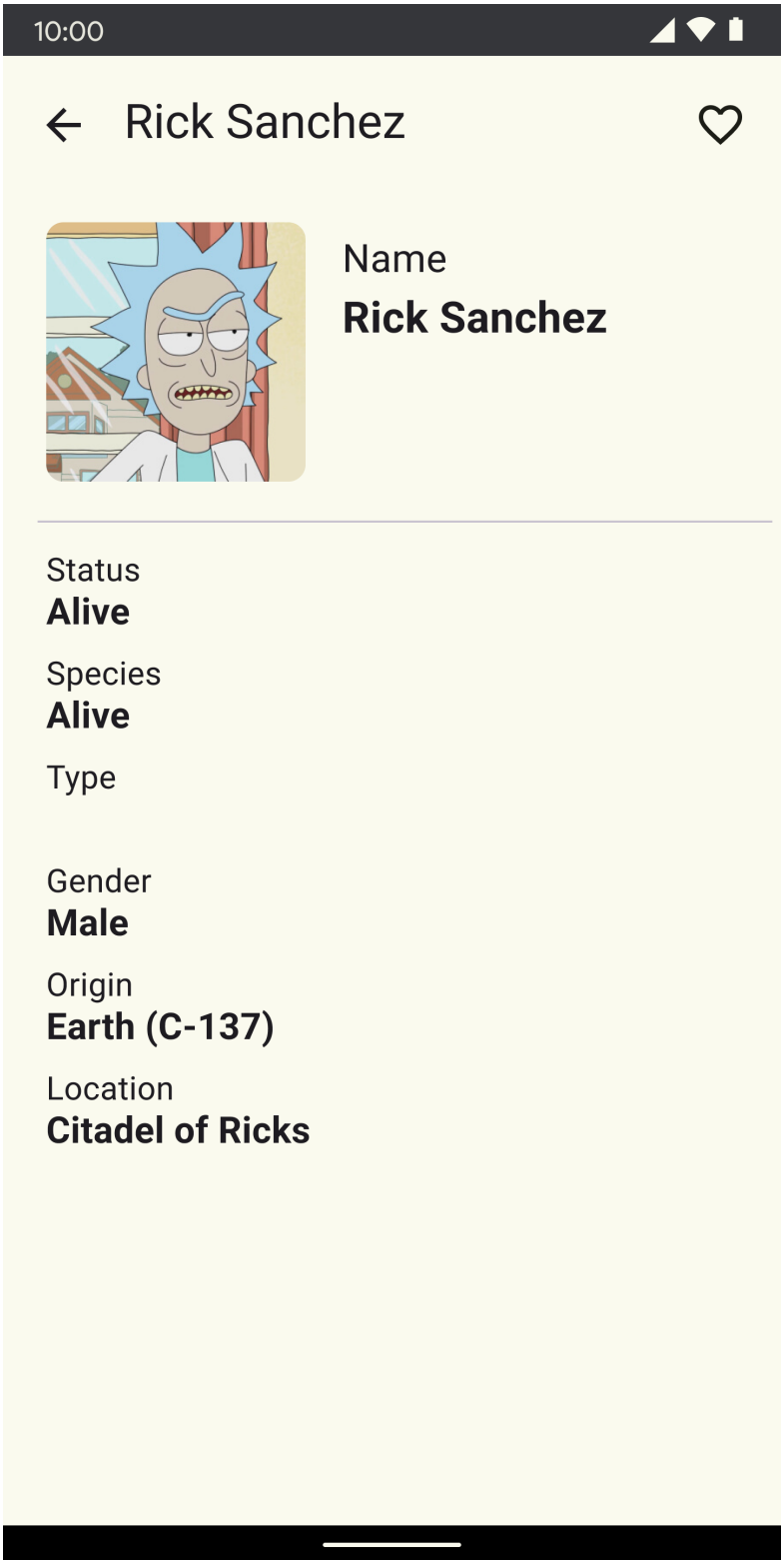
Tato obrazovka zobrazuje podrobné informace o dané postavě. Je zde opět možné přidat či odebrat postavu z oblíbených, tentokrát díky kliknutí na ikonku nacházející se na top baru. Provolává se jiný endpoint než v případě seznamu.

4.1.4 Vyhledávání

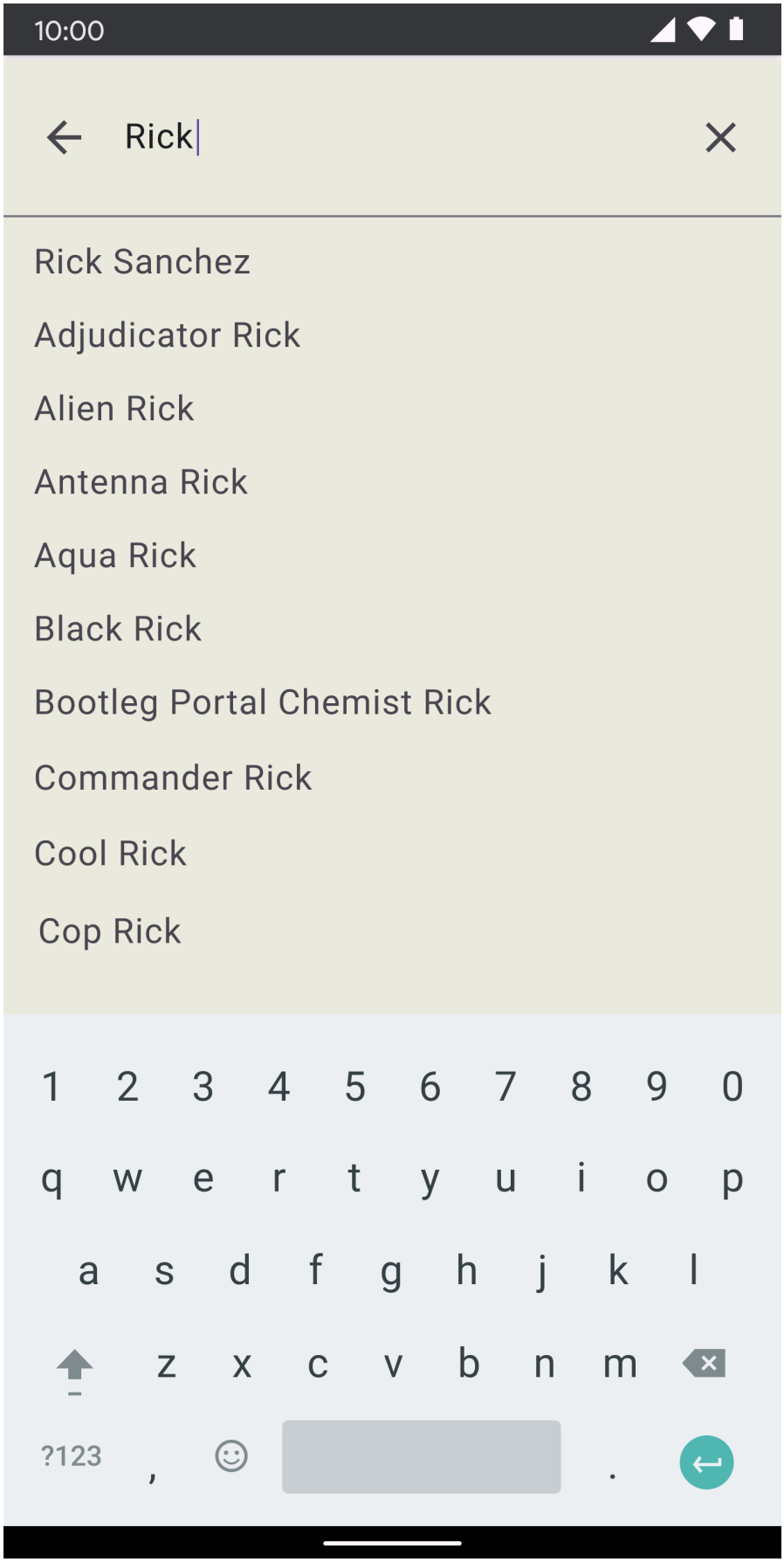
Tato obrazovka poskytuje uživateli možnost hledat postavu dle názvu. Nalezené postavy jsou zobrazeny v seznamu pod vyhledávacím boxem, po kliknutí na ně dojde k přechodu na obrazovku detailu dané postavy. V případě, že uživatel potvrdí vyhledávání, vrátí se na seznam všech postav, filtrovaných dle jména a použitého filtru.



Obrázek 4.3 Obrazovka se seznamem oblíbených postav



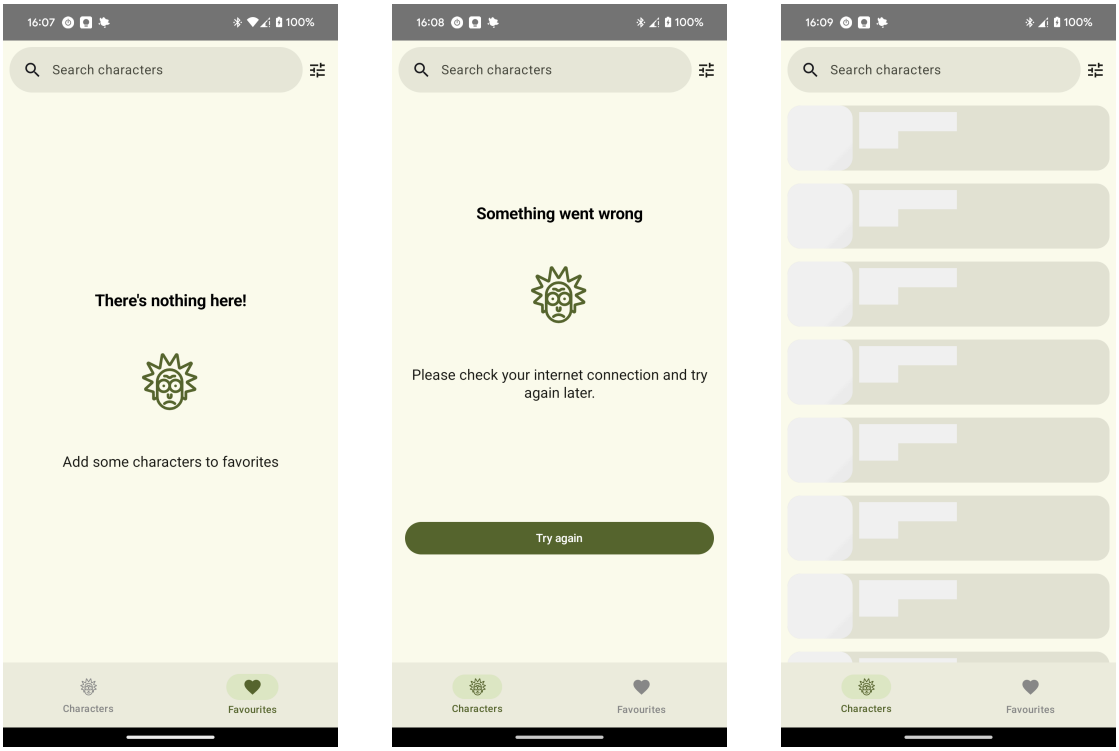
Obrázek 4.4 Obrazovka s detailem dané postavy



Obrázek 4.5 Obrazovka s vyhledáváním

4.1.5 Stavy obrazovek

Aby měl uživatel dobrý zážitek z aplikace, jsou zde vydefinované stavy obrazovek, do kterých může aplikace přejít. Mimo ideální stav se tak jedná o chybový, prázdný a načítací stav. Chybový stav nastává v případě problému se získáváním dat z API (například kvůli chybějícímu připojení k internetu) či z databáze. Načítací stav je určen pro stavy, kdy aplikace inicializuje obrazovku a dostává nová data (z API i z databáze). Prázdný stav se pak používá například v případě prázdného seznamu oblíbených postav. Na obrázku 4.6 jsou tyto stavy vedle sebe.



(a) Prázdný stav obrazovky **(b)** Chybový stav obrazovky **(c)** Načítací stav obrazovky

Obrázek 4.6 Stavy obrazovek

Kapitola 5
Implementace

V následující kapitole rozeberu implementovanou multiplatformní aplikaci ve výše zmíněných frameworcích (**Compose Multiplatform**, **Flutter**, **React Native** a **PWA**). U každé z nich se budu věnovat především použité architektuře, výběru a správě knihoven, práci se zdroji aplikace, podpoře pro Material Design a databázi.

5.1 Compose Multiplatform

Při vývoji na této platformě jsem se snažil co nejvíce přibližovat vývoji pro platformu Android, přeci jen Compose Multiplatform vychází z frameworku Jetpack Compose. Je zamýšlený k použití především pro Android vývojáře, kteří pak nemají s vývojem problémy a přechod je pro ně plynulý. Společnost JetBrains připravila stránku[1] pro tvorbu projektů ve frameworcích Compose Multiplatform a Kotlin Multiplatform. Pomocí wizardu si může vývojář vybrat, kterou platformu bude podporovat, dále pak jestli využije nativní UI. V případě mobilní aplikace jsou k dispozici šablony, ve kterých jsou základní obrazovky a práce s API.

5.1.1 Architektura

V dokumentaci pro Android se doporučuje moderní architektura pro tvorbu aplikací na této platformě. Doporučovaná architektura se řídí principy jako vrstvená architektura (oddělení vrstev, například modelů v aplikaci od UI), **Single source of truth** (SSOT) – centralizované změny dat na jednom místě, v neposlední řadě pak principem **Unidirectional Data Flow**, tedy že stav vede pouze jedním směrem mezi vrstvami. Společnost Google zde zmiňuje architekturu **Modern App Architecture**, která tyto techniky podporuje. Mimo to podporuje stav zjednodušující komplexitu UI, coroutines a flow (funkcionality jazyku Kotlin pro vícevláknové programování) či nejlepší praktiky pro Dependency Injection. [24]

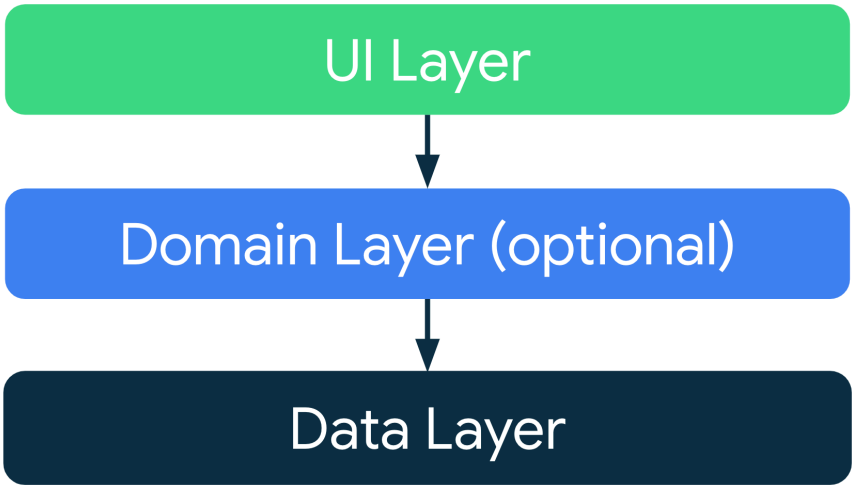
Pro menší aplikace tato architektura přináší komplexitu. U větších aplikací je ale díky ní implementace nových funkcionalit jednodušší, z vlastní zkušenosti to je preferovaná volba.

Na obrázku 5.1 jsou vidět vrstvy, ze kterých se skládá typická architektura mobilní aplikace. Jedná se o vrstvy:

- UI
- Doménová
- Datová

[1] `https://kmp.jetbrains.com/`

UI vrstva obsahuje elementy, které vykreslují data na obrazovce a `state holders`, komponenty, které spravují data, poskytují data pro UI a starají se o logiku. Na platformě Android se implementují pomocí třídy `ViewModel`. Doménová vrstva obstarává komplexní byznys logiku, či logiku znovu používanou napříč ViewModely. Třídy v této vrstvě se jmenují „use case“ či „interactor“. Datová vrstva se stará o byznys logiku, skládá se z tříd „repository“ (repozitář). Repozitář obsahuje libovolné množství zdrojů dat. Každý typ dat by měl mít svůj vlastní repozitář.



Obrázek 5.1 Diagram vrstev v architektuře aplikace [24]

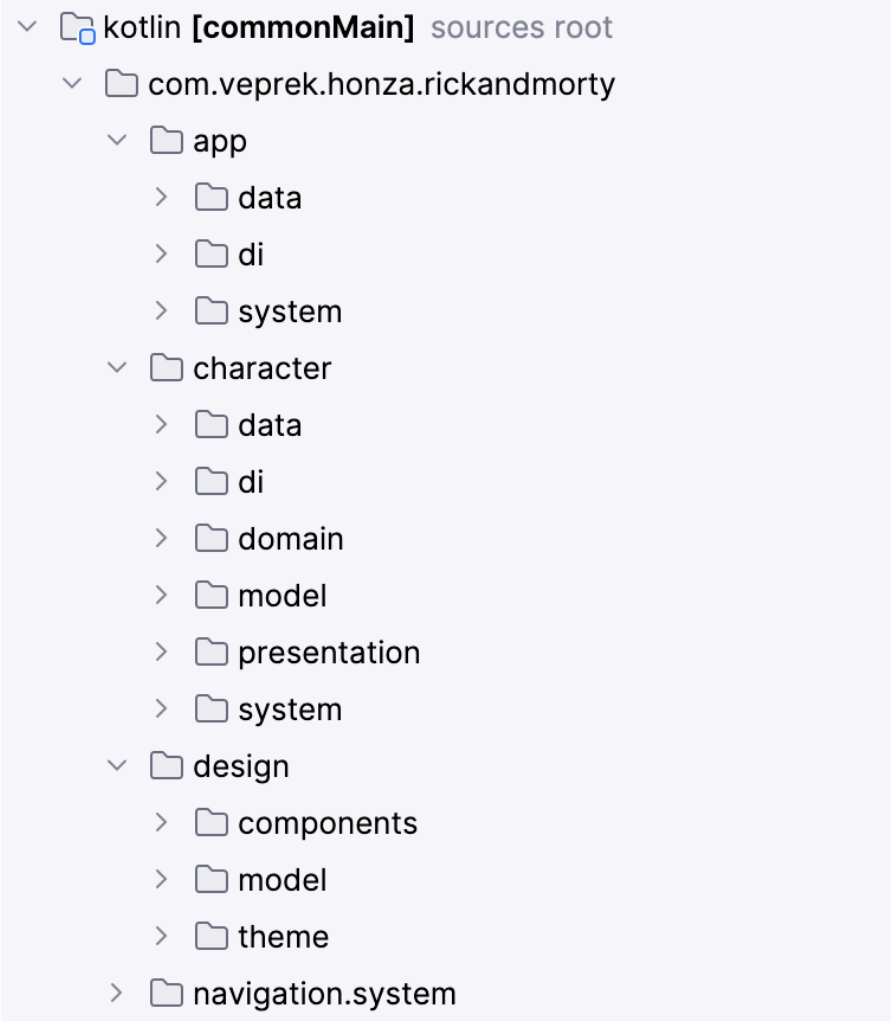
V implementaci jsem se řídil principy **Modern App Architecture** a **Clean Architecture**. Podpora pro tyto architektury je vesměs dostačující, nicméně Compose Multiplatform v době psaní této práce nepovoluje vícemodulovou aplikaci, tudíž modularizace těchto aplikací není zatím možná. Rozdělil jsem tedy aplikaci do vrstev, ve ViewModelech volám pouze usecasy, ty vystavuji pouze svůj interface. Tam, kde to jde, využívám interface a pro implementaci DI jsem si vybral Koin.

Struktura souborů také vycházela z Clean Architecture, jak je vidět na obrázku 5.2.

5.1.2 Výběr knihoven

Jak jsem zmiňoval v kapitole 2, ve frameworku Compose Multiplatform není podpora pro navigaci, tudíž jsem se rozhodoval mezi vhodnými knihovnami. Vybíral jsem z těch, jež jsou doporučovány společností JetBrains v dokumentaci ke Compose Multiplatform. **Precompose** [47] – tato knihovna se mi zamlouvala, protože poskytuje navigaci a ViewModely silně inspirovány frameworkem Jetpack Compose. Nemusel jsem se tedy v tomto případě učit zacházet s novou knihovnou, protože koncepty zde použité jsem již znal z frameworku Jetpack Compose. Oproti Jetpack Compose navigaci pak má omezenou funkcionalitu, chybí zde například možnost přidat listener na změnu obrazovky. Mimo to má pouze velmi jednoduchou dokumentaci s těmi nejzákladnějšími příklady použití. Během implementace jsem pak objevil chybu při implementaci spodní navigační lišty. Jakmile dojde v aplikaci k překliku na oblíbené postavy, přestanou kompletně fungovat ViewModely. Je pravděpoobné, že se jedná o chybu v knihovně Precompose či Koin, vzhledem k tomu, že jsou nyní obě v alfa verzích.

Dále jsem vybíral knihovnu pro zobrazení fotek z URL, která by podporovala i placeholdery a základní cachování. Jak jsem zmiňoval, je zde několik možností, mezi ty nejpoužívanější patří knihovny Kamel a Compose ImageLoader. Další možností by bylo použít alfa verzi knihovny Coil, která má rozpracovanou podporu Compose Multiplatform. Knihovna Kamel je svým návrhem



Obrázek 5.2 Struktura projektu, vycházející z Clean Architecture

API podobná knihovně Coil, což je výhoda pro vývojáře znalé vývoje pro Android. Stejně jako ona poskytuje cachování, možnost vlastního placeholderu a další velice žádané funkce. Další věc, o kterou se tato knihovna stará, je přidání žádosti o povolení k připojení k internetu na platformě Android. Oprávnění se vkládají do souboru AndroidManifest.xml. Knihovna Kamel jej přidá do svého manifestu a díky tomu se o tuto věc nemusí vývojář starat. Knihovna Compose ImageLoader oproti tomu poskytuje „pouze“ možnost načítání fotek z URL, vývojář pak načtenou fotku vloží do Image composable funkce.

Neexistuje zde podpora pro logování, tudíž jsem se rozhodl vybrat si logovací knihovnu. Vybral jsem knihovnu Napier, kde je inicializace z pohledu Compose Multiplatform velmi jednoduchá. Po přidání knihovny do projektu je pro inicializaci zapotřebí pouze přidat jeden řádek kódu do vstupní Composable funkce. V mém případě do funkce App: **Napier.base(DebugAntilog())**. Logování pak funguje bez problémů na každé implementované platformě, kde využívá k výstupu mechanismy dané platformy. Na platformě Android tedy používá **Logcat**, na platformě iOS **print** a na webu **console.log**. Jak by se dalo očekávat, podporuje různé úrovně logování, které vycházejí z těch pro platformu Android, jsou to následující: **VERBOSE (Napier.v()), DEBUG (Napier.d()), INFO (Napier.i()), WARNING (Napier.w()), ERROR (Napier.e()) a ASSERT (Napier.wtf())**. Mimo to tato knihovna poskytuje možnost napojení na Firebase Crashlytics, což je hojně využívaná služba pro logování chyb a pádů mobilních aplikací (mimo jiné díky neomezenému logování úplně zdarma).

5.1.3 Správa knihoven a závislostí

Ke správě knihoven a závislostí vývojář může využít soubory Gradle build (což je primární způsob, jakým se zpravují knihovny a závislosti v nástroji Gradle). Závislosti se definují pro každý modul zvlášť, v souborech `build.gradle`. Na platformě Android se jedná o nejčastější způsob. Dokonce, od verze 4.0 se můžou tyto soubory psát v jazyce Kotlin oproti jazyku Groovy, tudíž je celý codebase ještě blíže k tomu, aby byl psán v jednom jazyce. [77]

Nicméně, za tímto účelem se ve frameworku Compose Multiplatform primárně používá katalog verzí ve formátu `toml`. Systém Gradle využívá pro katalog verzí soubor `libs.versions.toml`. Popularita využívání katalogů verzí na platformě Android vzrostla s využíváním frameworku Kotlin Multiplatform. Aplikace psané pomocí něj se totiž rozdělují do modulů. Starání se o verze knihoven napříč moduly je bez využití katalogů obtížné, v případě aktualizace knihovny dochází k aktualizaci pevně zvolených konstant. Katalog verzí poskytuje možnost, jak centrálně vydefinovat závislosti k možnému použití v daných modulech. Pro Android vývojáře, kteří by chtěli přejít na využívání katalogu verzí, je v dokumentaci připravená sekce, která je provází migrací. [78]

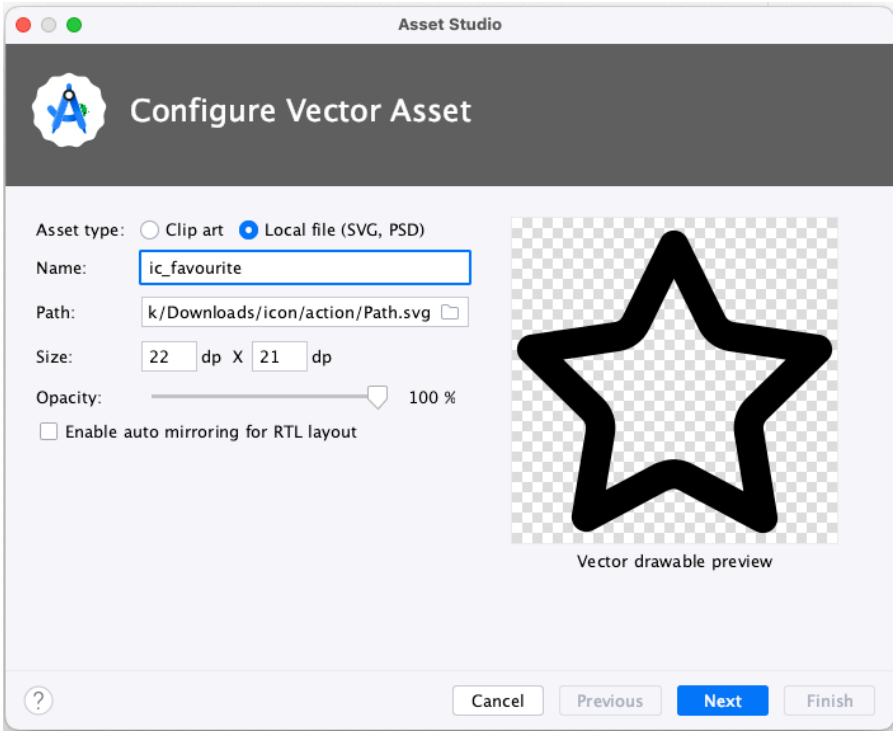
Katalog verzí se využívá i v projektu, který vznikne z KMP wizardu[2]. Na ukázce kódu 5.3 je vidět rozdíl mezi definováním závislosti.

```
dependencies {
    implementation("androidx.core:core-ktx:1.9.0")
}
[versions]
ktx = "1.9.0"

[libraries]
androidx-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "ktx" }
```

Obrázek 5.3 Rozdíly mezi definováním závislosti v souboru `build.gradle.kts` a pomocí katalogu verzí `libs.versions.toml`

[2] `https://kmp.jetbrains.com/`



Obrázek 5.4 Vector Asset Studio, součást Android Studia

5.1.4 Práce se zdroji

Se zdroji se pracuje obdobně jako na platformě Android, viz sekce 2.2.7. Oproti ní má ale tento přístup ještě jisté nevýhody. Dokud nedojde k překompilování projektu, je název přidaného zdroje zvýrazněn jako nenalezený (i když je již vydefinovaný v souboru strings.xml). Dále, v případě platformy Android, není možné předefinovat zdroj dle daného klíče v souboru strings.xml, v Android Studiu dojde ke zvýraznění zdrojů se stejným klíčem a projekt nepůjde zkompilovat. Ve frameworku Compose Multiplatform toto omezení není, může tedy dojít k tomu, že si vývojář omylem předefinuje řetězce se stejným klíčem a přepíše si tak předchozí hodnotu.

Dále, pro projekt na platformě Android poskytuje Android Studio možnost vytvořit ikonu přes dialog, viz obrázek 5.4. V případě tvorby ikon pro Compose Multiplatform Android Studio tuto možnost neposkytuje, nicméně tím, že je práce se zdroji ještě v alfa verzi, je možné že do plného vydání bude podpora i v Android Studiu.

5.1.5 Stavy obrazovek

Každá obrazovka by měla mít vydefinované stavy, ve kterých se může nacházet. Například v případě, že se ještě načítají data z API nebo z databáze, by měl uživatel vidět, že se něco na pozadí děje a že aplikace nezamrzla. Běžně se pro tyto účely používá takzvaný shimmer efekt. Jedná se o zašedlý útvar, přes který se mihotá lesklá linie. To signalizuje probíhající procesy na pozadí.

Shimmer efekt jsem si tedy musel vytvořit sám, inspiroval jsem se videem od Philippa Lacknera. [15]

5.1.6 Material Design

Projekt v Compose Multiplatform vytvořený přes wizard využívá starší verzi, Material Design 2, nicméně novější verze Material Design 3 je již plně podporovaná frameworkem Compose Multiplatform. Jak jsem se zmiňoval v kapitole 2.2.8, je doporučována pro nové aplikace. Protože jsem vycházel z návrhu, ve kterém jsou použity Material Design 3 komponenty, rozhodl jsem se, že jako design systém aplikace využiji tuto novější verzi Material Design.

5.1.7 Podpora Kotlin/Wasm

V této sekci bych rád zmínil, že spousta knihoven ještě nepodporuje target Kotlin/Wasm. Měl jsem tím pádem omezené možnosti výběru. Většina knihoven, se kterými jsem pracoval, jej buď má v nějaké práci rozpracovaný a počítá s ním, či dokonce má k dispozici předběžnou verzi. Z tohoto důvodu jsem nakonec implementoval dvě aplikace – jednu pouze pro platformu Android a iOS, ve které jsou použity knihovny v co nejstabilnějších verzích, a druhou, která podporuje i webovou platformu, ve které jsem používal jiné knihovny či ještě nestabilní verze kvůli omezené podpoře. K načítání obrázků z webové URL jsem na mobilním projektu použil knihovnu Kamel. Ta webovou podporu připravuje, je blokována závislostí. Za tímto účelem je vytvořené issue, které je zatím otevřené. [53] V projektu, kde cílím i na target Kotlin/Wasm používám knihovnu Compose ImageLoader. Ta momentálně poskytuje podporu pouze ve snapshot verzi. [58] Obě knihovny ale posloužily dobře, během vývoje se mi nestalo, že bych narazil na nějaký problém, či že by snad zapříčinily pád aplikace.

Tři z navigačních knihoven (`Voyager`, `Decompose` a `PreCompose`) zatím poskytují podporu pro webovou část pouze ve svých předběžných alfa/beta verzích. Některé funkcionality, které by uživatel čekal od chování webových stránek, v nich tedy zatím nejsou implementované. Například knihovna Precompose nepodporuje historii prohlížeče, nefunguje zde například navigovaní zpět a vpřed pomocí UI prohlížeče.

5.1.8 Databáze

Pro implementaci databáze jsem zvolil knihovnu SQLDelight. Její použití v multiplatformním projektu je poměrně jednoduché – po přidání potřebných závislostí, včetně driverů pro danou platformu, si stačí přes `expected` funkci vydefinovat driver, jehož implementace se pro danou platformu implementuje přes **actual** funkci. Pokud ale vývojář cílí na platformu iOS, musí přidat do linkeru vlajku `-lsqlite3`, a to buď přímo v iOS projektu v XCode (přes `Other Linker Flags` pole v `Build Settings`) či přidáním do gradle.build souboru. [14]

Zde se vyskytl problém. Knihovna SQLDelight nemá podporu pro platformu Kotlin/Wasm, neposkytuje tedy driver, který by se dal využít. V současné době neexistuje databázová knihovna/framework, která by poskytovala podporu pro tuto platformu.

Existuje sice repozitář, kde je proof-of-concept řešení využívající SQLDelight, nicméně toto řešení rozhodně není ideální. Opírá se o target Kotlin/JS, nevyužívá Kotlin/Wasm. Navíc, příklad, který je součástí tohoto repozitáře, není funkční, o čemž sám autor ví. [57]

Vzhledem k tomu, že jsem si projekt v Compose Multiplatform rozdělil na dva projekty, implementoval jsem databázi ve verzi mobile-only. Ve webové verzi aplikace jsem se řídil proof-of-concept řešením, ale bez úspěchu, tudíž tato verze poskytuje pouze ukládání in-memory.

5.1.9 IDE

Protože nejčastěji programuji pro platformu Android, jsem dobře obeznámen s vývojovým prostředím Android Studio. Z tohoto důvodu jsem začal aplikaci implementovat právě v tomto IDE. Oproti programování ve frameworku Jetpack Compose pro platformu Android zde nebyly skoro

žádné rozdíly. Největší rozdíly byly již zmiňované nefungující Preview anotace, viz podsekce 2.1.3, a práce se zdroji (viz 5.1.4). Dále pak našeptávání v Android Studiu není tak propracované jako v případě Jetpack Compose aplikace.

Dále jsem pro vývoj aplikace využíval IDE Fleet. Použil jsem jej, jelikož se jedná o jediné IDE, které momentálně podporuje Preview anotaci v Compose Multiplatform. Při otevření projektu dochází (stejně jako v Android Studiu) k aktualizaci indexů a dalším inicializačním procesům, například k zapínání tzv. „smart“ režimu. Rád bych se zde zmínil o časové náročnosti těchto procesů. Ačkoliv se jedná o menší aplikaci, tyto procesy, které se spouští při otevření projektu v IDE, trvaly na mém vývojářském stroji (MacBook Pro (2021), 3,2 GHz procesor Apple M1 Pro) skoro dvě minuty. To je poměrně dlouho, když tyto časy porovnám s časem pro otevření projektu psaném v Jetpack Compose, který využívá Kotlin Multiplatform pro implementaci multiplatformní logiky. K porovnání jsem použil otevření projektu se zdrojovými kódy aplikace, která má oproti mému prototypu řádově víc závislostí i obrazovek. Ale i přesto, že je tento projekt mnohem větší, celkový čas nutný pro všechny procesy, k otevření v Android Studiu trval pouze něco málo přes minutu.

5.1.10 Build aplikace

V případě generování artefaktů pro web byl postup přímočarý – řídil jsem se dokumentací[3], stačilo spustit příkaz `./gradlew wasmJsBrowserDistribution` a poté jsem ve složce `composeApp/build/dist/wasmJs/productionExecutable` měl vygenerováno vše potřebné k nasazení na webu. V dokumentaci byl dokonce i návod, jak nasadit web na GitHub pages. To jsem také udělal, výsledek je dostupný zde[4].

Pro generování aplikačních souborů pro platformu Android jsem využil menu v Android Studiu. Na platformě Android se rozlišuje mezi aplikací dvěma módy, debug (vývojářský) a release (pro vydání v Google Play). Generují se .apk soubory, pro vydání na Google Play pak existuje i proprietární formát aab. Generování obou variant apk souborů proběhlo bez problémů.

5.2 Flutter

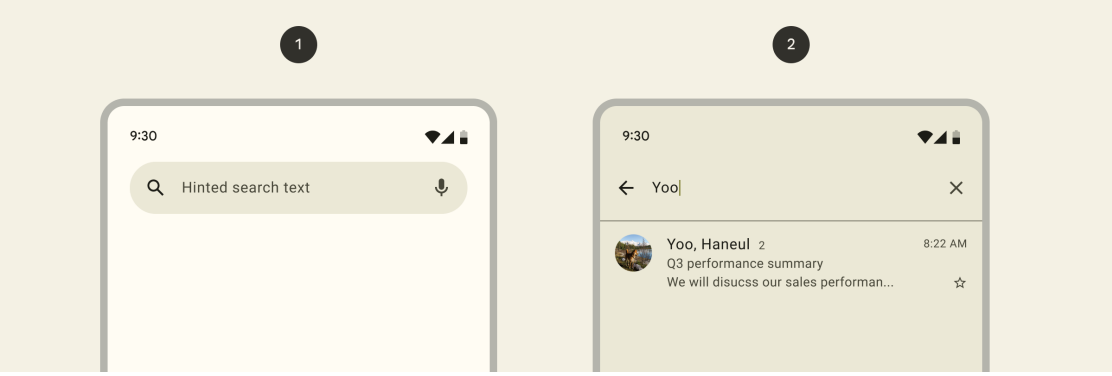
Pro platformu Flutter jsem vycházel z návrhu a z již implementovaného prototypu ve frameworku Compose Multiplatform. Od toho se odvíjel výběr architektury, použitých knihoven, widgetů a vývojářského prostředí.

5.2.1 Architektura

Stejně jako ve frameworku Compose Multiplatform jsem se řídil architekturou **Clean Architecture**. Vybral jsem ji proto, abych mohl porovnávat co nejpodobnější přístupy. Přímou podporu pro tuto architekturu framework Flutter neposkytuje, nicméně zde existuje spoustu knihoven, díky kterým ji lze implementovat. Jednou z nich je knihovna provider.

Knihovna provider, společně s třídou ChangeNotifier, se dá využít k implementaci ViewModelu. V UI vrstvě se vydefinuje Consumer, tedy komponenta, která naslouchá změně dat v ChangeNotifier. Dochází pak k přestavění komponent v případě změn. K informování o změnách se ve třídě ChangeNotifier používá funkce `notifyListeners`. [31]

[3] `https://kotlinlang.org/docs/wasm-get-started.html#publish-on-github-pages`
[4] `https://janveprek.github.io/`



Obrázek 5.5 Material 3 komponenty Search bar a Search view [26]

5.2.2 IDE

Jak jsem se zmiňoval v podsekci 3.1.1, mezi vývojovými prostředími doporučovanými pro psaní patří VS Code, IntelliJ Idea či Android Studio. Vybral jsem si Android Studio, aby byl zážitek co nejpodobnější jako při psaní prototypu v Compose Multiplatform. Díky dostupným pluginům jsem mohl využívat funkce jako `Hot Reload`, Flutter Inspector poskytující větší vhled do vykreslených widgetů a Flutter Performance, díky kterému jsem měl možnost pozorovat výkon aplikace a optimalizovat jej.

5.2.3 Výchozí UI komponenty

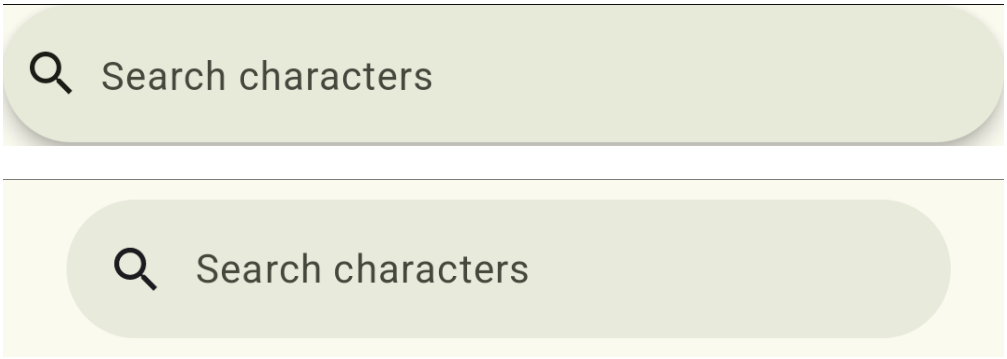
Během implementace prototypu jsem zjistil, že implementace UI komponent v systému Material Design 3 je rozdílná ve frameworcích Flutter a Compose Multiplatform. Nejvýraznější rozdíl jsem našel u `Search` komponenty.

5.2.4 Search

Dle dokumentace systému Material 3 by se k vyhledávání v aplikaci měly používat komponenty Search bar a Search view. Pomocí komponenty Search bar uživatel vyhledává, výsledky hledání by se pak měly vždy zobrazit v Search view. Search view představuje stav, kdy je Search bar vybraný. Ve výchozím chování tento stav zabírá celou obrazovku [26]. Na obrázku 5.5 jsou obě tyto komponenty vidět v jejich výchozí implementaci dle doporučených guidelines.

Výchozí implementace ve frameworcích Flutter a Compose Multiplatform jsou velmi rozdílné, jak je vidět na obrázku 5.6. Za použití výchozích parametrů jsou vidět patrné rozdíly. Komponenta v Compose Multiplatform (dole) je velice podobná výchozí komponentě z Material 3 dokumentace 5.5. Má nastavené horizontální a vertikální mezery a nemá žádný stín. Na druhou stranu, komponenta ve frameworku Flutter zabírá celou šířku obrazovky, nemá nastavené žádné mezery a má nastavený stín. Aby vývojář přiblížil vzhled výchozímu vzhledu, musí v Bottombaru nastavit `shadowColor: MaterialStateProperty.all(Colors.transparent)`. Nastavování barvy stínu na transparentní barvu mi nepřijde jako nejhezčí řešení, zvláště když stíny pro tuto komponentu se v Material 3 nepoužívají.

Další výrazný rozdíl je v použití Search view. Komponenta BottomBar ve frameworku Compose Multiplatform obsahuje v parametru položku `content` s Composable funkcí jako parametrem. V případě, že se předá prázdná funkce, zobrazí se celá obrazovka s prázdným obsahem. Opět, toto je chování, které odpovídá vydefinovaným guidelines. Ve frameworku Flutter ovšem Search view komponenta vůbec nemusí být vydefinována pro danou BottomBar komponentu. To ovšem vůbec neodpovídá guidelines, kde se výslovně píše, že výsledky vyhledávání mají vždy



Obrázek 5.6 SearchBar komponenty ve frameworcích Flutter (nahoře) a Compose Multiplatform (dole)

být zobrazeny právě v této komponentě [26]. Search view komponenta je v tomto frameworku implementována pomocí widgetu `SearchAnchor`, kterým se obalí `SearchBar` a seznam výsledků se pak vytvoří díky `suggesstionBuilder` parametru.

5.2.5 Barvy

K vygenerování barevného tématu jsem použil Theme Builder, stejně jako v kapitole 4. Všechny barvy v obou výše zmiňovaných frameworcích jsou nastaveny na stejné hodnoty. Každá aplikace má ale trochu jinak nabarvené komponenty – jmenovitě komponenty SearchBar, Card a BottomBar používají rozdílné barvy v každém frameworku.

5.2.6 Build aplikace

Jak jsem se zmiňoval, Flutter má obsáhlou dokumentaci, nachází se v ní i návod, jak sestavit aplikaci pro každou podporovanou platformu. V případě platformy Android stačilo použít příkaz `flutter build apk` s přepínači `--release` či `--debug`. Webová aplikace se dá sestavit obdobně – pomocí příkazu `flutter build web`. Dokumentace dokonce nabízí návod, jak aplikaci nasadit na různé služby, například na Firebase Hosting či GitHub Pages. [33]

5.3 React Native

Pro platformu React Native jsem vycházel z návrhu a z již implementovaných prototypů ve frameworcích Compose Multiplatform a Flutter. Od toho se odvíjel výběr architektury, použitých knihoven a UI komponent.

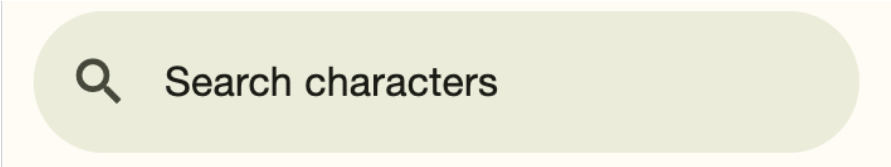
5.3.1 Architektura

Implementací jsem se opět snažil co nejvíce přiblížit předchozím dvěma prototypům. K implementaci ViewModelu jsem využil State komponenty, které jsou dostupné v Reactu. Jedná se o `useState`, `setState` a `useEffect` funkce, jedná se o preferovaný způsob popisovaný v dokumentaci React Native[5].

[5] `https://reactnative.dev/docs/state`

5.3.2 Výchozí UI komponenty

Stejně jako v případě frameworku Flutter neodpovídaly UI komponenty Material Design 3 těm, jak jsou definovány ve specifikaci. Nicméně, u většiny komponentů tento rozdíl nebyl tak velký, například `Search` komponenta vypadala velice podobně jako ve frameworku Compose Multiplatform, viz obrázek 5.7.



Obrázek 5.7 SearchBar komponenta ve frameworku React Native

5.3.3 Build aplikace

Expo framework také nabízí ve své dokumentaci informace o sestavení aplikace přímo v dokumentaci[6]. Využívá se soubor `eas.json`, ve kterém si vývojář vydefinuje různé konfigurace. K sestavení jsem pak použil příkaz `eas build`, kde jsem pomocí přepínačů určil platformu a vybranou konfiguraci. Tato metoda aplikaci sestaví na serveru, přístup k ní je pak možný pouze přes vytvořený Expo účet. Jako další možnost samozřejmě zůstává lokální sestavení, pomocí `npx expo run:android`. Při této možnosti ale musí vývojář připravit adresáře pro danou platformu, pomocí příkazu `prebuild`.

5.4 PWA

Při implementaci PWA v prostředí Expo jsem narazil na určité obtíže. Řídil jsem se přímo dokumentací[7], nicméně nepodařilo se mi propojit Service Worker s vygenerovaným webem. Jak jsem zjistil, je to tím, že tato podpora byla v dřívějších verzích, nyní ale framework Expo využívá Metro, které nemá podporu pro PWA, viz diskuze na GitHubu[8]. Ohledně jiných webových technologií, které bych využil, jsem nenašel žádnou s plnou podporou systému Material Design 3. Pravděpodobně je to kvůli faktu, že Material 3 komponenty pro web jsou stále ještě připravované, viz roadmap[9]. Rozpracovaná je například podpora Search, BottomBar či dalších navigačních komponent. Z tohoto důvodu jsem prototyp v technologii PWA neimplementoval.

[6] `https://docs.expo.dev/deploy/build-project/`
[7] `https://docs.expo.dev/guides/progressive-web-apps/`
[8] `https://github.com/expo/router/discussions/408`
[9] `https://github.com/material-components/material-web/blob/main/docs/roadmap.md`

Kapitola 6

Testy

V této kapitole se zaměřím na testování aplikace vyvinuté ve frameworku Compose Multiplatform. Testování je důležitou součástí vývoje softwaru, díky které můžeme zajistit kvalitu a spolehlivost naší aplikace. Pomocí dobře napsaných testů snadno odhalíme chyby a problémy ještě předtím, než se dostanou do produkčního prostředí.

Budu se zabývat různými druhy testů, jmenovitě unit testy a UI testy. Dále ukážu, jak je psát v daných frameworcích. Díky testování budu mít možnost ověřit správnost chování komponent, aplikační logiky a interakce mezi jednotlivými částmi aplikace.

Dále v této kapitole popíšu, s jakými nedostatky jsem se setkal, a ukážu využití nástroje pro zjištění code coverage (metrika určující, kolik procent kódu je pokryto testy).

6.1 Typy testů

V této diplomové práci se budu zabývat unit testy a UI testy.

Unit testy se zaměřují na malou část systému. Každý test se zabývá nějakou oddělenou jednotkou, kterou testuje. Většinou jsou psány programátory. Jsou malé a rychlé – spouštějí se často během vývoje. [82]

UI testy ověřují aspekty softwaru, se kterými přichází uživatel do styku. Zaměřují se na vizuální část programu a ne na interní logiku. Mezi věcmi, které zkoumají, patří responzivita, výkonnost a přístupnost. Provádí se jak manuálně, tak automaticky pomocí různých frameworků. [83]

Code coverage je metrika, která určuje, jak velká část zdrojového kódu je pokryta testy. Díky této metrice se dá zjistit, jaká je kvalita testů na daném projektu a které části kódu byly spuštěny během testování. [84]

Nástroje pro code coverage pak poskytují možnost generování reportu, ve kterém jsou rozdělená pokrytí testy pro funkce, výrazy, větve kódu a další.

6.2 Compose Multiplatform

Pro psaní Unit testů je na platformě Android několik možností, mezi nejpoužívanější patří knihovna JUnit[1]. Tato knihovna je dostupná i pro platformu Kotlin Multiplatform. Tuto knihovnu jsem použil společně s knihovnou Kotest, jmenovitě její část Assertions[2]. Tato knihovna poskytuje DSL pro vytváření testů, díky kterému jsou testy přehlednější. Dále jsem použil fra-

[1] `https://junit.org/`

[2] `https://kotest.io/docs/assertions/assertions.html`

mework MockK[3], který poskytuje mockování testovacích objektů. V neposlední řadě jsem využil knihovnu Turbine[4], kterou jsem využil k testování `Flow` hodnot vracených z coroutines.

Pro code coverage je na platformě Kotlin často využívaný framework Kover[5]. Tento framework je přímo vyvíjen týmem, který stojí za jazykem Kotlin. Generuje výpisy ve formátu XML a HTML. Poskytuje podporu pro platformy JVM, Kotlin Multiplatform a Android. V rámci testování aplikace psané ve frameworku Compose Multiplatform tedy můžeme tuto knihovnu využít, použití je pak stejné jako v případě platformy Android.

6.2.1 Použití

Pro Knihovny Junit, MockK a Kotest stačilo přidat závislost do souboru `build.gradle.kts` a psát testy v adresáři s příponou `UnitTest`. Mimo tyto knihovny je zapotřebí pro práci s coroutines (mechanismus v jazyce Kotlin pro asynchronní programování) přidat závislost na knihovně `coroutines-test`. V kódu se daný test obalí do bloku `runTest`. Díky němu nedojde k zániku kontextu a použije se speciální scope, ve kterém daná coroutine poběží.

K použití knihovny Kover stačí přidat plugin do souboru `build.gradle`, jak je vidět na ukázce kódu 11. Po přidání pluginu jsou k dispozici Gradle tasky, díky kterým se dají generovat reporty a verifikace. Zajímavé jesou zvláště tasky s reportem. Pro vygenerování reportu v modulu composeApp jsem použil task `./gradlew composeApp:koverHtmlReportDebug`. Tento task projede všechny testy a vygeneruje report, který uloží do souboru `build/reports/kover/htmlDebug/index.html`.

```
plugins {
    id("org.jetbrains.kotlinx.kover") version "0.7.6"
}
[versions]
kover = "0.7.6"

[libraries]
kover = {id = "org.jetbrains.kotlinx.kover", version.ref = "kover"}
```

Výpis kódu 11 Přidání pluginu Kover pomocí souboru `build.gradle.kts` a pomocí katalogu verzí `libs.versions.toml`

6.2.2 Ukázka

Na ukázce kódu 12 je vidět ukázka unit testu, který využívá `Test` anotaci z knihovny JUnit, knihovnu mockK pro vytvoření „mockovaného“ objektu, dále metody `test` a `awaitItem` z knihovny Turbine a funkci `ShouldBe` z knihovny Kotest. Tento test ověřuje, že při vytvoření ViewModelu dojde ke změnu stavu v proměnné `charactersState`.

V bloku metody `test` se očekává emitace položek z flow, k jejich odchycení slouží funkce `awaitItem`. Knihovna Turbine je poněkud striktní – všechny emitované položky musí být odchyceny, jinak test neprojde.

Při používání knihovny Kover jsem využil její rozšíření, díky kterým lze, mimo jiné, nastavit, které třídy se mají při tvorbě reportu vynechat. Díky tomu pak nemám v reportu třídy představující vnitřní logiku aplikace, stejně tak ani UI funkce opatřené anotací `@Composable`. Výpis z reportu je vidět na obrázku 6.1. Procentuálně jsem otestoval téměř každou třídu, v případě metod pak skoro 94 procent. Celý report je k dispozici v příloze.

[3] `https://mockk.io/`
[4] `https://github.com/cashapp/turbine`
[5] `https://github.com/Kotlin/kotlinx-kover`

```
@Test
fun `should get characters during init`() = runTest {
    val characterList = listOf(
        CharacterModel(
            id = 1,
            name = "Rick",
            status = "alive",
            iconUrl = "",
            isFavourite = false
        )
    )
    coEvery { getAllCharacters() } returns
            ResultWrapper.Success(characterList)
    val viewModel = createViewModel()

    viewModel.charactersState.test {
        val initialState = awaitItem()
        initialState.state shouldBe ScreenState.Loading
        initialState.characters shouldBe emptyList()
        val screenState = awaitItem()
        screenState.state shouldBe ScreenState.Success
        screenState.characters shouldBe characterList
    }
}
```

Výpis kódu 12 Ukázka jednoduchého Unit testu pro ViewModel

6.3 Flutter

V dokumentaci k frameworku Flutter se nachází několik částí, jež se věnují testování aplikace. Testy jsou zde rozděleny do třech kategorií: unit testy, widget testy a integrační testy. V tabulce, která se v dokumentaci nachází, jsou tyto testy srovnány dle důvěry, jež poskytuji, nákladů na podporu, nutných závislostí a rychlosti provedení.[40]

Budu se zde věnovat, stejně jako v případě Compose Multiplatform, pouze unit testům. Ostatní testy se v praxi moc často nepíší. Ve frameworku Flutter se pro psaní unit testů dá využít balíček test. Ten poskytuje základní podporu pro psaní testů v jazyku Dart, je součástí základní knihovny. Testy by měly být umístěné v adresáři `test` a jednotlivé testové soubory by měly

Current scope: composeApp | all classes

composeApp: Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %	Instruction, %
all classes	98.1% (52/53)	93.5% (100/107)	67.5% (54/80)	97% (325/335)	94% (2043/2173)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %	Instruction, %
com.veprek.honza.rickandmorty.character.data	100% (2/2)	90.9% (10/11)	90% (18/20)	98% (48/49)	95.5% (386/404)
com.veprek.honza.rickandmorty.character.data.entity	100% (10/10)	100% (23/23)	0% (0/10)	100% (35/35)	79.4% (127/160)
com.veprek.honza.rickandmorty.character.data.mapper	100% (1/1)	100% (4/4)	100% (2/2)	100% (26/26)	100% (70/70)
com.veprek.honza.rickandmorty.character.domain	100% (10/10)	89.5% (17/19)		92.3% (24/26)	79.2% (137/173)
com.veprek.honza.rickandmorty.character.model	100% (9/9)	100% (9/9)		100% (24/24)	100% (129/129)
com.veprek.honza.rickandmorty.character.presentation.detail	100% (4/4)	100% (6/6)	75% (6/8)	100% (34/34)	99.1% (221/223)
com.veprek.honza.rickandmorty.character.presentation.detail.state	100% (1/1)	100% (1/1)		100% (2/2)	100% (15/15)
com.veprek.honza.rickandmorty.character.presentation.favorite	100% (6/6)	100% (10/10)	72.2% (13/18)	100% (45/45)	98.7% (367/372)
com.veprek.honza.rickandmorty.character.presentation.favorite.state	100% (1/1)	100% (1/1)		100% (5/5)	100% (37/37)
com.veprek.honza.rickandmorty.character.presentation.list	100% (6/6)	100% (12/12)	68.2% (15/22)	100% (70/70)	99% (481/486)
com.veprek.honza.rickandmorty.character.presentation.list.state	100% (1/1)	100% (1/1)		100% (5/5)	100% (37/37)
com.veprek.honza.rickandmorty.character.system	50% (1/2)	60% (6/10)		50% (7/14)	53.7% (36/67)

generated on 2024-04-25 17:47

Obrázek 6.1 Report generovaný pomocí frameworku Kover.

končit `_test.dart`. Můžou se sdružovat do pojmenovaných skupin pro větší přehlednost. [41]

Pro mockování testovacích objektů se dá využít knihovna Mockito, která je doporučována v dokumentaci[6] frameworku Flutter.

Code coverage se dá vytvořit pomocí balíčku test, generovaný výstup je ve standardním formátu `lcov.info`. V případě souborů, u kterých vývojář nechce zjišťovat pokrytí (automaticky generované soubory, databáze) lze jednoduše přidat na začátek souboru komentář `// coverage:ignore-file`. Díky němu budou tyto soubory ignorovány. Jak jsem ale zjisti, tento výstup ukazuje pouze počet řádek, které jsou pokryty testy. To není ve většině případů ten nejdůležitější ukazatel. Mnohem zajímavější je například pokrytí funkcí a větví.

Z tohoto důvodu jsem zkoušel použít balíček coverage[7]. Přestože jsem se řídil dokumentací, při spouštění příkazu `dart pub global run coverage:test_with_coverage` jsem dostával hlášky „`Some tests failed.`“ (přestože s použitím příkazu coverage všechny testy prošly) a dále „`Unhandled exception: ProcessException: ...`“. Ani prohledání repozitáře na GitHubu mi nepřineslo kýžený výsledek, tudíž jsem se spokojil s výstupem v souboru `lcov.info`, který zobrazuje pouze pokrytí počtu řádků kódu.

6.3.1 Použití

K psaní unit testů není zapotřebí přidávat žádné závislosti do projektu, protože balíček test je v novém projektu již automaticky přidaný do souboru `pubsec.yaml`. Pro použití balíčku mockito je zapotřebí přidat jej a zároveň i závislost na `build_runner`, aby mohlo docházet k automatickému generování kódu: `flutter pub add http dev:mockito dev:build_runner`. Pro generování mocků je zapotřebí přidat anotaci `GenerateMocks` s názvy tříd, které se mají vygenerovat. Mocky se pak generují díky příkazu `dart run build_runner build`. Po tomto příkazu se mocky generují do souboru s koncovkou `.mocks.dart`, ze kterého se importují do testů.

6.3.2 Ukázka

Na ukázce kódu 13 je jednoduchý test provolání UseCasu. Tento test využívá mockované objekty díky anotaci `@GenerateMocks`. Test je definován díky funkcím `group` a `test` z balíčku test. Pomocí funkce `when` z knihovny Mockito je definovaná odpověď. Je zde použit konstrukt `await`, aby došlo k aktualizaci hodnoty před jejím dalším otestování.

Funkce `expect` porovnává hodnota s očekávaným výsledkem. Zde je dobré si uvědomit na základě čeho tato funkce objekty porovnává. V případě, že daná třída má definovaný operátor `=`, tak se použije k porovnání. V opačném případě se objekty porovnávají na rovnost, jestli jde o stejnou instanci objektu. To ne vždy chceme, tudíž u námi vytvořených tříd, které chceme testovat, musíme předefinovat operátor `=` a `hashCode`. Dále je zde rozdíl v porovnávání objektů, které jsou a nejsou `const` (z podstaty věci, jedná se o dva naprosto rozdílné objekty, tudíž porovnání selže).

Code coverage jsem měl vygenerovanou v souboru `lcov.info`, tento formát lze jednoduše konvertovat do HTML či zobrazit v nějakém online prohlížeči. Výpis reportu je vidět na obrázku 6.2. U automaticky generovaných souborů jsem zajistil, aby se ve výčtu nevyskytovaly. Dle reportu došlo k otestování skoro 91 procent řádků, pokrytí funkcí a větví bohužel podporováno není. Na GitHubu je za tímto účelem otevřené issue[8].

[6] `https://docs.flutter.dev/cookbook/testing/unit/mocking`
[7] `https://pub.dev/documentation/coverage/latest/`
[8] `https://github.com/flutter/flutter/issues/108313`

```
test('should get characters during init', () async {
  final characterList = [
    CharacterModel(
        id: 1,
        name: "Rick",
        status: "alive",
        iconUrl: "icon",
        isFavourite: true)
  ];
  when(getAllCharacters()).thenAnswer((_) async => Success(characterList));
  final viewModel = CharactersListViewModel(
    getAllCharacters: getAllCharacters,
    getCharactersByName: getCharactersByName,
    addCharacterToFavorites: addCharacterToFavourites,
    removeCharacterFromFavorites: removeCharacterFromFavourites,
  );
  expect(viewModel.charactersState.state, const LoadingState());
  await Future.delayed(Duration.zero);
  expect(viewModel.charactersState.state, const SuccessState());
  expect(viewModel.charactersState.characters, characterList);
});
```

Výpis kódu 13 Ukázka jednoduchého Unit testu pro ViewModel

6.4 React Native

Ve frameworku React Native se k testování dá použít knihovna Jest. Je to knihovna, kterou využívají přímo ve společnosti Facebook. Dají se s ní dělat Unit testy, snapshot testy, mockování modulů a funkcí. [67]

Pro testování React Hooks se dá použít knihovna react-hooks-testing-library[9]. Bohužel tato knihovna nepodporuje React 18 (ten je již dávno používán ve frameworku React Native) a je nahrazena knihovnou react-testing-library. Tato knihovna ale není kompatibilní s frameworkem React Native. Dále jsem zkoušel knihovnu React Native Testing Library[10], zde jsem ale narazil na nekompatibilitu mezi verzemi. Z tohoto důvodu jsem neimplementoval testy ViewModelů.

6.4.1 Použití

K psaní unit testů jsem přidal knihovnu `jest`. Dále jsem vydefinoval krok `test` v části skriptů v souboru `package.json`, abych si mohl jednoduše spouštět všechny testy najednou. Soubory s testy se označují koncovkou `.test`, je dobrá praxe je mít ve složce `__tests__`.

6.4.2 Ukázka

Vzhledem k tomu, že se mi nepodařilo implementovat testy ViewModelu, přikládám níže ukázku testu komponenty CharacterRepository. V tomto testu dochází k mockování komponent, které jsou definovány pomocí funkcí z knihovny `jest`. Je zde použita funkce `mockResolvedValue`, díky které je vydefinované chování funkce. Funkce `expect` a `toBe` se zde používají k porovnáním výsledku s očekávaným.

[9] `https://github.com/testing-library/react-hooks-testing-library`
[10] `https://callstack.github.io/react-native-testing-library/docs/getting-started`

```
test('getAllCharacters returns favorite characters', async () => {
        const db = new MockSQLiteCharacterDatabase();
        const api = new MockCharacterApi();
        api.getAllCharacters.mockResolvedValue({
                toModel: () => ({
                    data: [
                        new CharacterModel({
                            id: 1,
                            name: "Rick",
                            status: "alive",
                            imageUrl: "icon",
                            isFavourite: false
                        }),
                        new CharacterModel({
                            id: 2,
                            name: "Morty",
                            status: "alive",
                            imageUrl: "icon",
                            isFavourite: false
                        })]
                })
            }
        );
        db.getFavouriteCharacters.mockResolvedValue([
            new CharacterModel({id: 1, name: "Rick", status: "alive",
            imageUrl: "icon", isFavourite: true})
        ]);
        const repository = new CharacterRepositoryImpl(api, db);

        const result = await repository.getAllCharacters();
        expect(result instanceof SuccessResult).toBe(true);

        if (result instanceof SuccessResult) {
            expect(result.value.length).toBe(2);
            expect(result.value).toEqual([
                new CharacterModel({id: 1, name: "Rick", status: "alive",
                imageUrl: "icon", isFavourite: true}),
                new CharacterModel({id: 2, name: "Morty", status: "alive",
                imageUrl: "icon", isFavourite: false})
            ]);
        }
    });
});
```

Výpis kódu 14 Ukázka jednoduchého Unit testu pro repozitář

6.5 UI testy

Na UI testy jsem využil průchod webovou aplikací, viz sekce 7.3, a dále průchod mobilní aplikací. Během vývoje jsem používal různá zařízení, abych si mohl ověřit, že UI vypadá dobře na větších i menších obrazovkách.

Mimo to jsem využil Firebase Test Lab[11]. Jedná se o nástroj, který spouští daný APK soubor na různých zařízeních. Pro obdobný test na zařízeních pod systémem iOS je zapotřebí nahrát soubor ve formátu IPA, pro vygenerování tohoto souboru je ale zapotřebí distribuovat aplikaci pod placeným Apple Developer účtem, který nemám.

Testy jsem pustil na třech různých zařízeních (Medium phone, Small phone a Medium tablet). Tato zařízení měla velikost úhlopříčky mezi 4,7 a 10 palci. Nechal jsem náhodný průchod aplikací (na výběr je i specifický průchod aplikací). Chtěl jsem vydefinovat scénář průchodu, ale nástroj, který je k dispozici v Android Studiu, bohužel nepodporuje tvorbu u UI tvořeného v Jetpack Compose. Na obrázku 6.3 je vidět jeden z průchodů aplikací. Díky těmto spuštěním jsem si mohl ověřit, že UI vypadá na různých zařízeních přijatelně.

[11] `https://firebase.google.com/docs/test-lab`

LV **LCOV Viewer**

Back | All Files

90.4% lines 265/293
100.0% functions 0/0
100.0% branches 0/0

Collapse All Expand All

File	Lines	Line Coverage	Functions	Function Coverage	Branches	Branch Coverage
– All Files	265/293	90.4%	0/0	100.0%	0/0	100.0%
– lib	265/293	90.4%	0/0	100.0%	0/0	100.0%
– character	251/279	90.0%	0/0	100.0%	0/0	100.0%
– data	62/75	82.7%	0/0	100.0%	0/0	100.0%
character_repository_impl.dart	37/40	92.5%	0/0	100.0%	0/0	100.0%
– entity	25/35	71.4%	0/0	100.0%	0/0	100.0%
character_detail_dto.dart	14/20	70.0%	0/0	100.0%	0/0	100.0%
character_dto.dart	7/9	77.8%	0/0	100.0%	0/0	100.0%
paged_result_dto.dart	4/6	66.7%	0/0	100.0%	0/0	100.0%
– model	55/66	83.3%	0/0	100.0%	0/0	100.0%
character_detail.dart	24/27	88.9%	0/0	100.0%	0/0	100.0%
character_model.dart	29/32	90.6%	0/0	100.0%	0/0	100.0%
result_wrapper.dart	2/2	100.0%	0/0	100.0%	0/0	100.0%
filter.dart	0/5	0.0%	0/0	100.0%	0/0	100.0%
– domain	18/18	100.0%	0/0	100.0%	0/0	100.0%
get_favorite_characters_use_case.dart	3/3	100.0%	0/0	100.0%	0/0	100.0%
add_character_to_favorites.dart	3/3	100.0%	0/0	100.0%	0/0	100.0%
get_all_characters_use_case.dart	3/3	100.0%	0/0	100.0%	0/0	100.0%
get_characters_by_name_use_case.dart	3/3	100.0%	0/0	100.0%	0/0	100.0%
get_character_by_id_use_case.dart	3/3	100.0%	0/0	100.0%	0/0	100.0%
remove_character_from_favorites_use_case.dart	3/3	100.0%	0/0	100.0%	0/0	100.0%
– presentation	116/120	96.7%	0/0	100.0%	0/0	100.0%
– favorite	35/37	94.6%	0/0	100.0%	0/0	100.0%
favorite_characters_view_model.dart	30/32	93.8%	0/0	100.0%	0/0	100.0%
– state	5/5	100.0%	0/0	100.0%	0/0	100.0%
favorite_characters_state.dart	5/5	100.0%	0/0	100.0%	0/0	100.0%
– detail	24/25	96.0%	0/0	100.0%	0/0	100.0%
character_detail_view_model.dart	19/20	95.0%	0/0	100.0%	0/0	100.0%
– state	5/5	100.0%	0/0	100.0%	0/0	100.0%
character_detail_state.dart	5/5	100.0%	0/0	100.0%	0/0	100.0%
– list	57/58	98.3%	0/0	100.0%	0/0	100.0%
character_list_view_model.dart	49/50	98.0%	0/0	100.0%	0/0	100.0%
– state	8/8	100.0%	0/0	100.0%	0/0	100.0%
character_list_state.dart	8/8	100.0%	0/0	100.0%	0/0	100.0%
– design	14/14	100.0%	0/0	100.0%	0/0	100.0%
– components	9/9	100.0%	0/0	100.0%	0/0	100.0%
favorite_icon.dart	9/9	100.0%	0/0	100.0%	0/0	100.0%
– model	5/5	100.0%	0/0	100.0%	0/0	100.0%
screen_state.dart	5/5	100.0%	0/0	100.0%	0/0	100.0%

LCOV Viewer | Report generated at Tue Apr 30 2024 18:34:31 GMT+0200 (Central European Summer Time)

Obrázek 6.2 Report generovaný pomocí coverage, zobrazený v LCOV Viewer.

Obrázek 6.3 Průchod aplikací

Kapitola 7
Vyhodnocení

V této kapitole provedu vyhodnocení a srovnání implementovaných prototypů.

7.1 Velikost aplikace

Jednou z metrik, kterou jsem vybral, je velikost sestavené aplikace. Mobilní telefony v dnešní době mívají čím dál větší velikost úložiště, nicméně stále je důležité pamatovat na velikosti aplikace. Na platformě iOS jsem si nemohl vygenerovat IPA soubor, tudíž zde porovnání není. Jak jsem se zmiňoval, v systému Android jsou aplikace rozděleny do dvou variant – debug a release. Varianty se liší ve velikosti, v release verzi je aplikace menší díky různým optimalizacím, dochází zde i k obfuskaci kódu. Debug verze se podepisuje výchozím klíčem, dá se ladit. V tabulce 7.1 je vidět rozdělení velikostí release a debug verze aplikací z daného frameworku. Debug verze aplikace z frameworku Flutter je opravdu velká. Nicméně, release verze aplikace není o moc větší než Compose Multiplatform verze. Zajímavé je, že aplikace ve frameworku React Native měla stejnou velikost v debug i release verzi. Je to z toho důvodu, že ve výchozím chování zde není zapnutý Proguard.

V tabulce je tedy velikost se zapnutým nástrojem Proguard pro tuto variantu, pro ostatní jsem nic neměnil a nechal výchozí chování. I tak má ale aplikace psaná ve frameworku React Native zdaleka největší velikost APK souboru v release verzi.

Platforma	debug verze (MB)	release verze (MB)
Compose Multiplatform	21,6	15,5
Flutter	165,7	22
React Native	77,9	74,1

Tabulka 7.1 velikost APK souboru

7.2 Výkon mobilní aplikace

K měření výkonu mobilní aplikace jsem využil Firebase Test Lab, o které jsem se zmiňoval v kapitole 6.5. Pro UI testy mi stačily virtuální zařízení, u kterých jsem nemusel tak dlouho čekat na doběhnutí a u kterých je větší denní kvóta. Pro měření výkonu se ale nehodí, protože výkonnostní statistiky u nich nejsou k dispozici. Z toho důvodu jsem vybral reálné zařízení – telefon Pixel 5. Nicméně, moc zajímavých informací jsem z tohoto nástroje nedostal. Mezi nejzajímavější informace patří čas potřebný k prvnímu vykreslení, jež byl u frameworku Flutter

přes tři sekundy. Aplikace z frameworku Compose Multiplatform a React Native měly čas kolem půl vteřiny.

7.3 Výkon webové aplikace

Pro porovnání výkonnosti aplikace jsem využil vývojářské možnosti nacházející se v prohlížeči Google Chrome. Ve všech variantách jsem porovnával debug verze, aby bylo porovnání co nejpřesnější.

7.3.1 Recorder

Prvním z využitých nástrojů je nástroj Recorder[1], jenž se používá k zaznamenávaní uživatelských flow (ukázkových průchodů aplikací).

Použiti je jednoduché – vývojář si vybere název, zadefinuje viewport (šířku, výšku zařízení, zda se jedná o mobil, landscape, ...). Dále se vydefinují, které se mají projít, a nakonec se tato nahrávka přehraje.

Na obrázku 7.1 je vidět část průchodu, který jsem pro testování vytvořil. V horní části je nastavený viewport, aby odpovídal mobilnímu zařízení (stejné rozměry má Pixel 7). V tomto průchodu dochází k přechodu na hlavní stranu, kliknutí na první položku seznamu, přechod na její detail. Z detailu se přejde zpět na hlavní obrazovku, zde dojde k napsání písmen „ko“ do vyhledávacího pole, dále přechod na první položku vyfiltrovaného seznamu, zpět na hlavní obrazovku a vymazání textu z vyhledávacího pole. Posledními kroky jsou kliknutí na ikonku pro přidání do oblíbených u položky „Adjudicator Rick“, přechod na oblíbené položky a přechod na první položku tohoto seznamu.

Průchod jsem musel definovat pro každou implementaci zvlášť. Důvod je ten, že aplikace v Compose Multiplatform jen vykresluje na canvas, není zde žádný popis u kroků, které se děly (vše jsou jen kliknutí). Oproti tomu v React Native se jedná o kliknutí na elementy — jak je vidět na obrázku, kde je dodána informace Element „Search characters“. Dále ve frameworku Flutter se také elementy, pomocí níž během průchodu dochází ke kliknutí, jmenují jinak, tudíž jsem nemohl použít jeden průchod pro všechny.

7.3.2 Performance profile

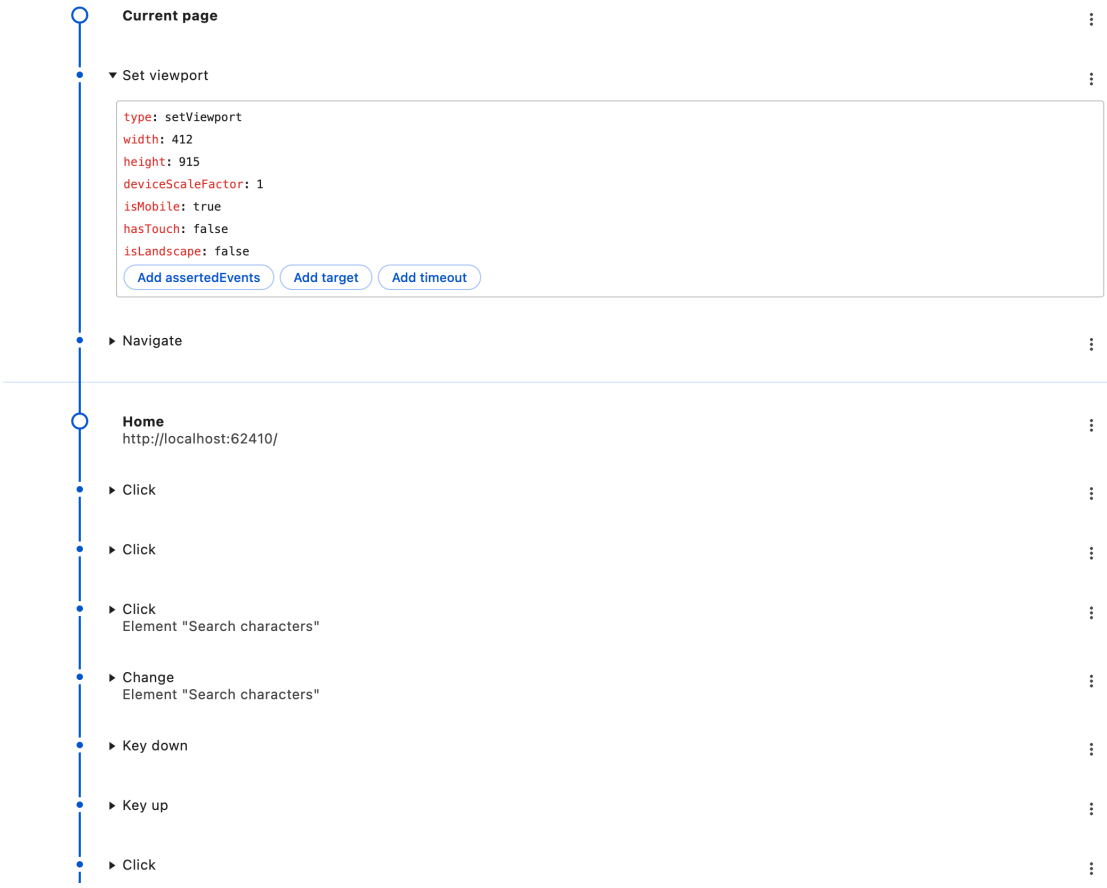
Průchod aplikací se dá využít k vyšetření výkonu. Stačí vybrat možnost Performance panel. Na obrázku 7.2 je vidět takový výpis. Zajímavé jsou Frames, ve kterých je vidět, jak dlouho trvalo vykreslování daných frames. Všechny programy na tom byly z tohoto hlediska podobně.

Ve výpisu je vidět i paměťová náročnost běhu programu. Aplikace v Compose Multiplatform využívala paměti na haldě (JS Heap). Společně s grafem je udávané rozmezí. Pro aplikaci v Compose Multiplatform je toto rozmezí (14,3–49,6 MB). Aplikace v React Native pak 13,1–44,9 MB). Aplikace ve frameworku Flutter využívala zdaleka nejvíc paměti (39,7–129 MB).

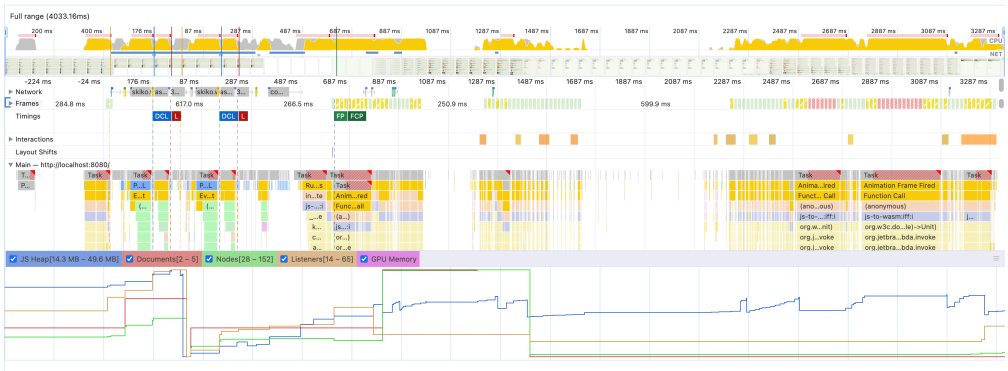
Dále je k dispozici přehled, ve kterém jsou rozepsány časy potřebné k načtení stránky společně s rozepsaným rozdělením, jak je vidět na obrázku 7.3. Stejně jako u výkonu zde vychází nejlépe framework React Native, nicméně framework Compose Multiplatform je na druhém místě před frameworkem Flutter.

Nicméně, mimo React Native se poměrně často stávalo, že scénář neproběhl celý a že kvůli načítání různých komponent nedošlo například k přechodu na detail. Tyto problémy se stávaly jak ve frameworku Compose Multiplatform, tak ve frameworku Flutter. Ovšem při zpomaleném průchodu (průchod nabízí možnosti normální, pomalý, velmi pomalý a extrémně pomalý) proběhly průchody na obou platformách přesně podle plánu.

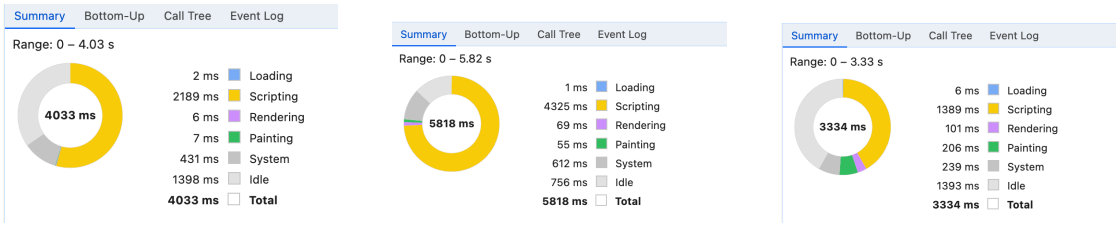
[1] `https://developer.chrome.com/docs/devtools/recorder`



Obrázek 7.1 Ukázkový průchod aplikací



Obrázek 7.2 Výpis z výkonnostního profilu aplikace v Compose Multiplatform



(a) Compose Multiplatform **(b)** Flutter **(c)** React Native

Obrázek 7.3 Výkonnostní reporty

7.3.3 Page load

Další nástroj z Chrome Dev Tools, jež jsem použil, je Performance insights. Tato funkce je zatím v preview fázi, stejně jako předchozí recorder. Na obrázku 7.4 je screenshot z použití. Ve spodní části je timeline, která se dá v různých rychlostech přehrát, je tak možné si projet, jak se vykreslovalo UI. Například, mezi prvky, které se vykreslily jako poslední, patří spodní ikona ve formátu svg a načtené obrázky z URL. Mezi nejzajímavější údaje patří FCP (čas vykreslení prvního obsahu) a TTI (čas kdy začne být stránka interaktivní), oba měřené od začátku načítání, udávány ve vteřinách. Další zajímavý ukazatel je Long task – jedná se o task běžící na hlavním vlákně, který trval déle než 50 ms. Pro implementované platformy vychází počet těchto tasků následovně: Flutter (8), Compose Multiplatform (5) a React Native (2).

Abych se výkonnostně přiblížil mobilním zařízením, spustil jsem Page load znovu, tentokrát se čtyřnásobně omezeným CPU (stejný postup je v dokumentaci použit pro simulaci mobilního zařízení). V tabulce níže 7.2 jsou vidět časy, po kterých došlo k vykreslení prvního smysluplného obsahu. Jak je vidět, v případě zpomalení dochází ke zřetelnějším rozdílům mezi frameworkem Flutter a dalšími dvěma. V případě Compose Multiplatform a React Native totiž není rozdíl až tak velký.

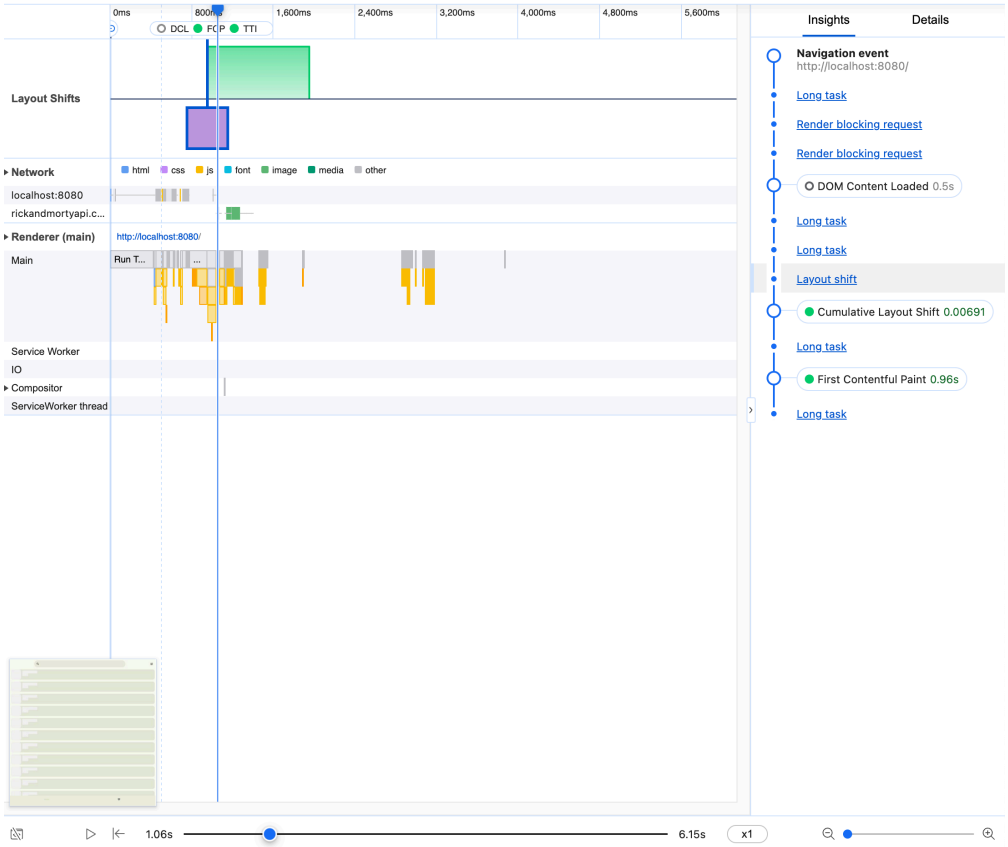
Platforma	čas (s) Normální CPU	čas (s) 4x zpomalené CPU
Flutter	2,49	10,29
Compose Multiplatform	0,96	4,8
React Native	0,54	3,88

Tabulka 7.2 Čas prvního vykreslení obsahu (FCP), udávaný v sekundách

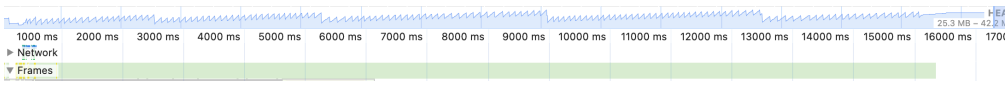
7.3.4 Načtení seznamu

Mimo testování procházení aplikace jsem zkoumal výkon při načítání a projíždění dlouhým listem (s 1000 položek). Na obrázku 7.5 je vidět průchod v implementaci v Compose Multiplatform. Ta se ukázala jako nejvýkonnější v tomto testu, jako na jediné zde nedocházelo k vynechání snímků či viditelným zásekům. Vycházela nejlépe i v porovnání využité paměti. Pro aplikaci v Compose Multiplatform bylo rozmezí (25,3–42,2 MB). Aplikace v React Native pak 29,9–139 MB). Aplikace ve frameworku Flutter opět využívala zdaleka nejvíc paměti (181–226 MB).

Celkově z těchto výsledků vyšel framework Compose Multiplatform docela dobře. Z výkonnostního hlediska při průchodu aplikací byl na druhém místě, s velkým předstihem před frameworkem Flutter. Při testování dlouhého seznamu na tom byl výkonnostně dokonce nejlépe. Co se velikosti aplikace týče, byl na tom podobně jako Flutter, s velikostí aplikace ve frameworku React daleko vzadu.



Obrázek 7.4 Načtení stránky (v implementaci v Compose Multiplatform)



Obrázek 7.5 Využití paměti (nahoře) a čas potřebný k vykreslení snímku (dole) při projíždění dlouhým seznamem ve frameworku Compose Multiplatform

Kapitola 8

Vhodnost frameworku Compose Multiplatform a její možný rozvoj do budoucna

V této kapitole bych se rád věnoval celkové vhodnosti frameworku Compose Multiplatform ve stavu, v jakém je v době psaní této diplomové práce, a o možných změnách a rozvoji do budoucna. Následující informace jsou platné k datu 24. 4. 2024.

8.1 Vybavenost frameworku

Mezi jednu z největších nevýhod frameworku Compose Multiplatform bych zařadil podporu základních funkcionalit napřímo ve frameworku bez nutnosti použití externích knihoven. Mezi základní funkcionality, které zde chybí, beze sporu patří navigace v aplikaci a načítání fotek z URL. Z konkurenčních frameworků má navigaci v aplikaci v základu pouze framework Flutter, jak jsem se zmiňoval v podsekci 3.1.12. Ten ale nabízí vývojáři hned dvě knihovny pro navigaci. Navigaci nicméně plánují ve frameworku Compose Multiplatform implementovat, tudíž v budoucnu si myslím, že tento problém zmizí. Již nyní je k dispozici v beta verze 1.6.10 tohoto frameworku, která nabízí podporu pro navigaci a ViewModely.

Načítání fotek pak umí konkurenční frameworky bez použití knihoven. Zde je ale již nyní slušný výběr knihoven, které tuto funkcionalitu nabízí. Navíc se plánuje verze 3.0 knihovny Coil (oblíbená knihovna na platformě Jetpack Compose), která bude podporovat framework Compose Multiplatform. Momentálně je dostupná v alfa verzi. Myslím si tedy, že načítání fotek nebude v budoucnu problém, protože vývojáři z platformy Android se nebudou muset učit pracovat s jinou knihovnou. Navíc dostupné knihovny jsou jednoduché k využití a jsou udržované (např. knihovna Kamel).

Celkově bych tedy řekl, že vybavenost frameworku Compose Multiplatform bude v budoucnu díky přidání navigační komponenty do základu a díky ViewModelům na vysoké úrovni. Již nyní je v některých ohledech lepší než u frameworku React Native (podpora design systému, různá menu, gesta, ...).

8.2 Compose Multiplatform pro web

Compose Multiplatform pro web by se ve stavu, v jakém je, neměl používat v produkci, jak jsem se již zmiňoval v sekci 2.2.

Dle mého názoru má v nynějším stavu řadu nevýhod, které by se měly vyřešit před jeho použitím pro tuto platformu. Dále pak podpora knihoven pro target Kotlin/Wasm je nyní velmi omezená. Z části to je z důvodu, že framework Ktor, který je zdaleka nejvíce využívaný HTTP klient na platformě Kotlin, nyní nemá pro tento target podporu. Knihovny, které na něm závisí proto s podporou vyčkávají nebo ji mají teprve rozpracovanou. Podpora ze stran knihoven se ale určitě v budoucnu zlepší. Momentálně ale spoustu knihoven použít nejde, právě kvůli chybějící podpoře. Na rozdíl od frameworku Flutter a React Native, kde některé knihovny nepodporuji všechny platformy, je lze přidat do projektu a pouze si ošetřit místa využití v kódu. U platformy Compose Multiplatform bohužel toto udělat nejde a projekt nepůjde zkompilovat kvůli neexistující knihovně pro target Kotlin/Wasm. Z tohoto důvodu jsem například musel vytvořit dva projekty pro aplikaci v Compose Multiplatform – jednu, která cílila jen na mobilní platformy, jež obsahovala knihovnu SQDelight, a druhou jež cílila i na web, a která tím pádem nemohla obsahuvat závislost na této knihovně.

Dalším velkým nedostatkem je fakt, že se nyní Compose Multiplatform vykresluje na `canvas`, z tohoto důvodu nefunguje podpora pro základní funkce jako vyhledávání v textu, vybírání textu, překlad textu či zoom. Toto jsou úplně základní věci, které by měly podporovat všechny webové stránky. Tento nedostatek má za následek, že stránka v něm napsaná není nijak vhodná pro osoby s postižením, nepodporuje ani základní přístupnost ve smyslu zvětšení textu pro lidi s horším zrakem či popis obrázků a fotek pro využití ve čtečkách obrazovky.

8.3 Compose multiplatform pro mobilní platformy

Vzhledem k tomu, že jsem implementoval nezávisle na sobě aplikaci pouze pro mobilní platformy (iOS a Android) a aplikaci, která také cílila na web, mohl jsem v každém projektu využívat jiné knihovny. Compose Multiplatform pro mobilní platformy nemá takové problémy s dostupností knihoven. Na rozdíl od webové části jsem nemusel žádnou z využívaných knihoven používat v jiné než stabilní verzi.

V době implementace ještě nebyla k dispozici verze 1.6.0 frameworku Compose Multiplatform, tudíž jsem ji využíval beta verzi, protože v této verzi přibyla podpora zdrojů (což je důležitá funkce potřebná pro psaní aplikací) a omezená podpora přístupnosti na platformě iOS. Mimo to v této verzi přibyly změny chování, díky kterým má aplikace s cílem na platformu iOS chování podobné tomu, na jaké jsou její uživatelé zvyklí.

Mezi výhody frameworku Compose Multiplatform patří interoperabilita s platformními UI frameworky – na platformě iOS není problém volat kód ve SwiftUI přímo v Compose Multiplatform a naopak. Stejně tak funguje podpora pro UIKit. Na platformě Android lze zase přímo z Compose volat views, psané v XML. V případě některých chybějících funkcionalit je lze dopsat v kódu pro danou platformu.

8.4 Podpora platforem

Jak jsem se zmiňoval v minulé sekci, pro mobilní platformy je Compose Multiplatform na celkem vysoké úrovni. Podporuje jak Android, tak iOS (ten zatím v beta verzi, ale s každou novu verzí tohoto frameworku přichází nové funkcionality pro tuto platformu). Stejně tak podpora pro desktop je dobrá (desktop byla druhá platforma, kterou Compose Multiplatform podporoval hned po platformě Android). Nicméně, kvůli podpoře pro web, jež je v experimentální verzi, je na tom z hlediska podpory různých platforem hůře než Flutter i než React Native. V případě frameworku React Native je podpora nejvyšší, je to ale způsobeno komunitou, která poskytuje podporu například i pro visionOS[1].

[1] `https://github.com/callstack/react-native-visionos`

8.5 Podpora IDE

Jak jsem se zmiňoval v podsekci 2.2.2, podpora pro Compose Multiplatform ze strany JetBrains je poněkud roztříštěná. Tím, že se tento framework zakládá na frameworku Jetpack Compose, využívaném při vývoji aplikací pro platformu Android, očekával bych, že se JetBrains více zaměří na jeho podporu v Android Studio IDE. JetBrains ale v oficiální dokumentaci používá IDE IntelliJ IDEA.

Pro obě jmenovaná IDE existuje plugin `Compose Multiplatform IDE Support`, ten ale poskytuje pouze podporu pro `@preview` anotaci v aplikacích určených pro desktopovou platformu. A to je tento plugin k dispozici již od roku 2021, avšak beze větších změn ve funkcionalitě. V novějších verzích jak Android Studia, tak IntelliJ IDEA nedochází k lepší podpoře Compose Multiplatform. Funkcionalita `@preview` anotace se sice s každou verzí Android Studia zlepšuje, stále ale funguje pouze v sekci pro Android, což je nepoužitelné pro projekt v Compose Multiplatform. Dále v Android Studiu není podpora vytváření zdrojů, na rozdíl od platformy Android. Když tedy vývojář chce přidat zdroj do aplikace v Compose Multiplatform, nemůže si vygenerovat xml soubory přímo v Android Studiu, jak je zvyklý z platformy Android. Tato funkce se sice může objevit v nějaké z novějších verzí tohoto IDE, ale vzhledem k tomu, že se podpora pro Compose Multiplatform moc nezlepšila, myslím si, že to je nepravděpodobné. Mimo to, během implementace testů jsem narazil na další problém – po přidaní testů mi Android Studio začalo hlásit, že se v projektu nachází moduly, které se nebudou kompilovat a nabízí možnost je odebrat. Nicméně, i přes toto omezení se testy normálně kompilovaly a šly spouštět.

Na druhou stranu společnost JetBrains propaguje své nové IDE JetBrains Fleet, které v době psaní této práce je ve fázi veřejného preview, tudíž zatím zdarma. Momentálně tedy není vhodné pro psaní produkčních aplikací a nepodporuje některé základní funkcionality, například pluginy. Jako jediné IDE podporuje `@preview` anotace ve společné části projektu, což jej činí nejvhodnějším kandidátem pro tvorbu aplikací v Compose Multiplatform. Na druhou stranu, není zatím známý jeho cenový model, je ale jisté, že po skončení preview fáze nebude zdarma. To je další nevýhoda oproti Android Studiu, o které se stará společnost Google a nechává jej k dispozici zdarma v neomezené verzi, oproti prostředím od společnosti JetBrains. Všechny nové funkce ohledně podpory frameworku Compose Multiplatform vyšly právě v tomto IDE.

V rámci osobního rozhovoru s vývojářem Sebastianem Aignerem, který pracuje pro JetBrains, jsem se dozvěděl, že technologie Compose Multiplatform i IDE Fleet jsou zatím v tzv. „teenage years“, tudíž jsou funkcionality, které chybí a na kterých se pracuje až na základě zpětné vazby od komunity. Dále jsem se dozvěděl, že firma JetBrains počítá s tím, že Fleet bude v budoucnu poskytovat všechny funkce potřebné k vytvoření multiplatformní aplikace. Dále si Sebastian zapsal mou poznámku o chybějící podpoře pro importování ikon pomocí konverze z svg do xml, jako je možné v Android Studiu.

8.6 Podpora knihoven

Za dobu jeho existence vzniklo mnoho knihoven pro Jetpack Compose. Takřka každá z populárních a využívaných knihoven má rozpracovanou či naplánovanou podporu pro Compose Multiplatform. Jedná se převážné o zbavení se závislostí na platformně specifické API platformy Android. Dále se zvyšuje počet knihoven pro Kotlin Multiplatform. Také knihovny, které jsou k dispozici jen pro Android se připravují na přechod na Kotlin Multiplatform, například jak jsem zmiňoval v podsekci 2.3.2, knihovna Room. Z tohoto pohledu mi přijde, že je framework na dobré cestě, co do počtu se sice nemůže rovnat frameworku Flutter (momentálně přes 42 tisíc balíčků), ani frameworku React Native (přes 1400 knihoven na platformě React Native Directory [44]), ale každým dnem přibývají nové.

Navíc, některé části jako navigace či implementace ViewModelu se již nyní dostávají do beta verze tohoto frameworku.

Rád bych zde vyzdvihl práci se zdroji, která ve frameworku Compose Multiplatform funguje obdobně jako na platformě Android. Díky ní je jednoduché přidávat do aplikace statické texty (včetně překladů) a obrázky. Texty a překlady jsou věc, která nemá přímou podporu ani ve frameworku Flutter, ani v React Native. Dále, podpora pro vektorové obrázky se musí na těchto platformách řešit přes knihovny. Přidávání vektorových obrázků do aplikace bylo v případě frameworku Compose Multiplatform značně jednodušší než u konkurenčních frameworků.

8.7 Dokumentace

Oproti konkurenčním řešením, zvláště pak proti frameworku Flutter, je dokumentace velice omezená. Ve většině případů se odvolává na dokumentaci k frameworku Jetpack Compose, s upozorněním, že některé třídy a metody nejsou dostupné pro Compose Multiplatform. Co se ukázkového použití týče, obsahuje dokumentace stránku[2] s odkazy do repozitářů s ukázkovými aplikacemi. Nicméně, chybí mi zde nějaký návod krok za krokem, jak docílit základních funkcionalit, pro základní použití by mi přišel jednodušší a přívětivější než procházet kód celé aplikace.

Jak dokumentace pro framework Flutter, tak pro React Native obě obsahují návody, jak dosáhnout základních funkcionalit. Mimo to obsahují sekce věnující se vydávání aplikace. Celkově jsou mnohem obsáhlejší.

V případě dokumentace pro framework Flutter se v ní nachází i návody pro vývojáře přicházející z jiných platforem, dále pak i části věnující se rozdílům v chování komponent pro různé platformy (iOS vs. Android) viz sekce 3.1.10.

8.8 Vývoj frameworku

Tím, že je tento framework ještě v počátcích, tak do něj v každé aktualizaci vychází zajímavé a potřebné funkcionality. Příjemně mě překvapila rychlost, se kterou se nové verze vydávají. Oznámení verze 1.6.0 vyšlo 29. února, beta verze 1.6.10 již 18. dubna. Doufám, že tato rychlost vydávání nových verzí v nejbližší době nepoleví. Určitě svědčí o tom, že na něm společnost JetBrains usilovně pracuje, což je dobré znamení.

8.9 Komunita

O tom, že je o framework Compose Multiplatform zájem, svědčí mimo jiné oblíbenost jeho repozitáře[3], kde má skoro patnáct tisíc hvězdiček.

Tento rok jsem se účastnil konference mDevCamp 2024[4], na které se probíraly novinky a zajímavosti ze světa mobilního vývoje. Mezi nejnavštěvovanější přednášky, kterých jsem se účastnil, patřila přednáška o frameworku Compose Multiplatform[5]. Celá přednášková hala byla zaplněná, po skončení přednášky se nastřádalo více než patnáct dotazů, na které bohužel nebyl čas. Pro srovnání, účastnil jsem se i přednášky ohledně frameworku Flutter[6], která se konala ve více než poloprázdném sálu, a na jejíž konci bylo dotazů jen pár.

Po této přednášce jsem právě hovořil se Sebastianem Aignerem. Bylo vidět, že ho práce na frameworku Compose Multiplatform baví a že je rád za každou zpětnou vazbu od vývojářů. Díky tomuto rozhovoru si myslím, že většina problémů, které Compose Multiplatform má, se v budoucnu vyřeší a že bude schopný plně konkurovat ostatním multiplatformním řešením.

[2] https://www.jetbrains.com/help/kotlin-multiplatform-dev/multiplatform-samples.html
[3] https://github.com/JetBrains/compose-multiplatform
[4] https://mdevcamp.eu/
[5] https://mdevcamp.eu/schedule.html#sebastian-aigner-modal
[6] https://mdevcamp.eu/schedule.html#lukas-klingsbo-modal

Kapitola 9
Závěr

Cílem této diplomové práce byla analýza frameworku pro tvorbu multiplatformních aplikací Compose Multiplatform. V rámci analýzy jsem jej srovnával s frameworkem Jetpack Compose, ze kterého vychází, zkoumal jsem kvalitu dokumentace, připravenost pro základní použití atd. Dále jsem v této práci představil konkurenční frameworky pro tvorbu multiplatformních aplikací – Flutter, React Native a PWA. Také u nich jsem provedl analýzu základních funkcionalit a dokumentace. Dále jsem navrhl uživatelské rozhraní prototypu aplikace v nástroji Figma, dodržoval jsem design systém Material Design 3.

V rámci implementace jsem kvůli rozdílné podpoře knihoven implementovat prototyp ve frameworku Compose Multiplatform rozděleně pro mobilní platformy a pro mobilní platformy i s webem. Vzhledem k experimentálním verzím použitých knihoven jsou v těchto prototypech chyby, které by se měly opravit následným vývojem. Dále jsem implementoval prototypy ve frameworcích Flutter a React Native. Prototyp v PWA jsem nakonec neimplementoval kvůli nedostatečné podpoře systému Material Design 3 a dále kvůli nekompatibilitě s frameworkem React Native.

Prototyp aplikace jsem otestoval – napsal jsem unit testy pro všechny implementované platformy a otestoval jsem i UI díky spouštěním aplikace na různých rozměrech zařízení.

Následovalo srovnání frameworků – kvantitativně jsem srovnal webové a mobilní aplikace vytvořené frameworky Compose Multiplatform, Flutter a React Native na základě několika kritérií. V případě webové aplikace jsem zkoumal výkon, rychlost načítání a ukázkový průchod aplikací. V případě mobilní aplikace jsem srovnával velikost aplikačních souborů a výkon na různých zařízeních. Frameworky jsem také srovnával kvalitativně z hlediska kvality dokumentace, podpory komunity a podporovaných funkcionalit.

Nakonec jsem provedl diskuzi ohledně vhodnosti frameworku Compose Multiplatform v nynější podobě, připravenosti webové části a výhledu tohoto frameworku do budoucnosti.

Literatura

[1] Google. *Icons*. web.dev [online]. 2024 [cit. 3. 2. 2024]. Dostupné z: https://web.dev/learn/design/icons.

[2] LARDINOIS, Frederic. *Google launches Jetpack Compose, an open-source, Kotlin-based UI development toolkit*. TechCrunch [online]. 7. 5. 2019 [cit. 3. 2. 2024]. Dostupné z: https://techcrunch.com/2019/05/07/google-launches-jetpack-compose-an-open-source-kotlin-based-ui-development-toolkit/.

[3] Jetbrains. *What advantages does Kotlin give me over the Java programming language?*. kotlinlang.org [online]. 27. 12. 2023 [cit. 10. 3. 2024]. Dostupné z: https://kotlinlang.org/docs/faq.html#what-advantages-does-kotlin-give-me-over-the-java-programming-language.

[4] Jetbrains. *Kotlin FAQ*. kotlinlang.org [online]. 27. 12. 2023 [cit. 12. 2. 2024]. Dostupné z: https://kotlinlang.org/docs/faq.html.

[5] LARDINOIS, Frederic. *Kotlin is now Google's preferred language for Android app development*. TechCrunch [online]. 7. 5. 2019 [cit. 7. 2. 2024]. Dostupné z: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/.

[6] MOHAN, Sandhya. *Android Studio Hedgehog is stable*. Google [online]. 30. 11. 2023 [cit. 24. 2. 2024]. Dostupné z: https://android-developers.googleblog.com/2023/11/android-studio-hedgehog-is-stable.html.

[7] SICARD-GREGORY, Neville. *Android Studio Iguana is stable*. Google [online]. 29. 2. 2024 [cit. 1. 3. 2024]. Dostupné z: https://android-developers.googleblog.com/2024/02/android-studio-iguana-is-stable.html.

[8] AIGNER, Sebastian. *Compose Multiplatform 1.6.0 – Resources, UI Testing, iOS Accessibility, and Preview Annotation*. Jetbrains [online]. 29. 2. 2024 [cit. 2. 3. 2024]. Dostupné z: https://blog.jetbrains.com/kotlin/2024/02/compose-multiplatform-1-6-0-release/.

[9] Jetbrains. *JetBrains Fleet: The Next-Generation IDE by JetBrains* [online]. 29. 2. 2024 [cit. 2. 3. 2024]. Dostupné z: https://www.jetbrains.com/fleet/. https://www.jetbrains.com/fleet/

[10] Jetbrains. *Compose Multiplatform UI Framework*. jetbrains.com [online]. 2024 [cit. 5. 2. 2024]. Dostupné z: https://www.jetbrains.com/lp/compose-multiplatform/.

[11] Jetbrains. *Android-only components*. jetbrains.com [online]. 28. 11. 2023 [cit. 5. 2. 2024]. Dostupné z: https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-android-only-components.html.

[12] Jetbrains. *Navigation and routing – Kotlin Multiplatform Development*. jetbrains.com [online]. 2024 [cit. 8. 2. 2024]. Dostupné z: https://www.jetbrains.com/help/kotlin-multiplatform-dev/compose-navigation-routing.html.

[13] CashApp. *SQLDelight - Generates typesafe Kotlin APIs from SQL* [online]. 2. 3. 2024 [cit. 4. 3. 2024]. Dostupné z: https://github.com/cashapp/sqldelight.

[14] LiewJunTung. *Kotlin Multiplatform (iOS): not working with Cocoapods plugin* [online]. 21. 8. 2019 [cit. 4. 3. 2024]. Dostupné z: https://github.com/cashapp/sqldelight/issues/1442.

[15] LACKNER, Philipp. *How to Create a Shimmer Loading Effect in Jetpack Compose (WITHOUT Library!)*. Youtube.com [online]. 15. 1. 2023 [cit. 12. 2. 2024]. Dostupné z: https://www.youtube.com/watch?v=NyO99OJPPec.

[16] DE SIMONE, Sergio. *JetBrains Compose Multiplatform for iOS Reaches Alpha*. InfoQ [online]. 28. 5. 2023 [cit. 4. 2. 2024]. Dostupné z: https://www.infoq.com/news/2023/05/compose-multiplatform-ios-alpha/.

[17] WebAssembly. *WebAssembly* [online]. 2024 [cit. 5. 2. 2024]. Dostupné z: https://webassembly.org/.

[18] InsertKoinIO. *Koin* [online]. 28. 11. 2023 [cit. 5. 3. 2024]. Dostupné z: https://github.com/InsertKoinIO/koin.

[19] pedrofsn. *Verifying your Koin configuration*. Koin [online]. 25. 2. 2024 [cit. 5. 3. 2024]. Dostupné z: https://insert-koin.io/docs/reference/koin-test/checkmodules/.

[20] JetBrains. *Kotlin Wasm* [online]. 2024 [cit. 5. 2. 2024]. Dostupné z: https://kotlinlang.org/docs/wasm-overview.html.

[21] JetBrains. *Kotlin/Wasm performance* [online]. 7. 12. 2023 [cit. 5. 2. 2024]. Dostupné z: https://kotlinlang.org/docs/wasm-overview.html#kotlin-wasm-performance.

[22] Material Design. *Unveiling Material You* [online]. 18. 5. 2021 [cit. 3. 3. 2024]. Dostupné z: hhttps://material.io/blog/announcing-material-you.

[23] Android Developers. *Custom design systems in Compose* [online]. 4. 3. 2024 [cit. 4. 3. 2024]. Dostupné z: https://developer.android.com/jetpack/compose/designsystems/custom.

[24] Android Developers. *Guide to app architecture* [online]. 12. 12. 2023 [cit. 30. 3. 2024]. Dostupné z: https://developer.android.com/topic/architecture.

[25] WILLIAMS, James. *Migrating to Material Design 3* [online]. 27. 10. 2021 [cit. 5. 2. 2024]. Dostupné z: https://material.io/blog/migrating-material-3.

[26] Google. *Search – Material Design 3* [online]. 4. 8. 2020 [cit. 24. 3. 2024]. Dostupné z: https://m3.material.io/components/search.

[27] SANSONE, Anthony. *Supported deployment platforms*. flutter.dev [online]. 24. 2. 2024 [cit. 25. 3. 2024]. Dostupné z: https://docs.flutter.dev/reference/supported-platforms.

[28] LOUGHEED, Parker. *Building user interfaces with Flutter*. flutter.dev [online]. 1. 3. 2024 [cit. 25. 3. 2024]. Dostupné z: https://docs.flutter.dev/ui.

[29] LOUGHEED, Parker. *Use themes to share colors and font styles* [online]. 1. 3. 2024 [cit. 25. 3. 2024]. Dostupné z: https://docs.flutter.dev/cookbook/design/themes.

[30] LOUGHEED, Parker. *Automatic platform adaptations*. flutter.dev [online]. 5. 12. 2023 [cit. 25. 3. 2024]. Dostupné z: https://docs.flutter.dev/platform-integration/platform-adaptations.

[31] LOUGHEED, Parker. *Simple app state management*. flutter.dev [online]. 25. 3. 2024 [cit. 30. 3. 2024]. Dostupné z: https://docs.flutter.dev/data-and-backend/state-mgmt/simple.

[32] LOUGHEED, Parker. *Navigation and routing*. flutter.dev [online]. 1. 3. 2024 [cit. 2. 4. 2024]. Dostupné z: https://docs.flutter.dev/ui/navigation.

[33] LOUGHEED, Parker. *Navigation and routing*. flutter.dev [online]. 1. 3. 2024 [cit. 2. 4. 2024]. Dostupné z: https://docs.flutter.dev/ui/navigation.

[34] LOUGHEED, Parker. *Persist data with SQLite*. flutter.dev [online]. 25. 3. 2024 [cit. 3. 4. 2024]. Dostupné z: https://docs.flutter.dev/cookbook/persistence/sqlite.

[35] LOUGHEED, Parker. *Deployment*. flutter.dev [online]. 5. 4. 2024 [cit. 24. 4. 2024]. Dostupné z: https://docs.flutter.dev/deployment.

[36] LOUGHEED, Parker. *Adding assets and images*. flutter.dev [online]. 1. 3. 2024 [cit. 30. 3. 2024]. Dostupné z: https://docs.flutter.dev/ui/assets/assets-and-images.

[37] LOUGHEED, Parker. *Material Design for Flutter*. flutter.dev [online]. 1. 3. 2024 [cit. 25. 3. 2024]. Dostupné z: https://docs.flutter.dev/ui/design/material.

[38] LOUGHEED, Parker. *Hot reload*. flutter.dev [online]. 25. 3. 2024 [cit. 29. 3. 2024]. Dostupné z: https://docs.flutter.dev/tools/hot-reload.

[39] LOUGHEED, Parker. *Set up an editor* [online]. 1. 3. 2024 [cit. 27. 3. 2024]. Dostupné z: https://docs.flutter.dev/get-started/editor.

[40] LOUGHEED, Parker. *Testing Flutter apps* [online]. 5. 4. 2024 [cit. 29. 4. 2024]. Dostupné z: https://docs.flutter.dev/testing/overview.

[41] LOUGHEED, Parker. *An introduction to unit testing* [online]. 5. 4. 2024 [cit. 29. 4. 2024]. Dostupné z: Anintroductiontounittesting.

[42] LOUGHEED, Parker. *Introduction to Dart* [online]. 27. 2. 2024 [cit. 25. 3. 2024]. Dostupné z: https://dart.dev/language.

[43] LOUGHEED, Parker. *Dart overview* [online]. 20. 3. 2024 [cit. 25. 3. 2024]. Dostupné z: https://dart.dev/overview.

[44] TENODI, Tomislav. *React Native Monthly #2*. Reactnative.dev [online]. 28. 7. 2017 [cit. 29. 3. 2024]. Dostupné z: https://reactnative.dev/blog/2017/07/28/react-native-monthly-2#expo.

[45] OCCHINO, Tom. *React Native: Bringing modern web techniques to mobile*. Engineering.fb [online]. 26. 3. 2015 [cit. 29. 3. 2024]. Dostupné z: https://engineering.fb.com/2015/03/26/android/react-native-bringing-modern-web-techniques-to-mobile/.

[46] VATNE, Brent. *FAQ – Expo Documentation*. expo.dev [online]. 25. 4. 2024 [cit. 29. 4. 2024]. Dostupné z: https://docs.expo.dev/faq/.

[47] Tlaster. *Tlaster/Precompose* [online]. 1. 2. 2024 [cit. 12. 2. 2024]. Dostupné z: https://github.com/Tlaster/PreComposel.

[48] reactnative.dev. *Using TypeScript* [online]. 22. 4. 2024 [cit. 24. 4. 2024]. Dostupné z: https://reactnative.dev/docs/typescript.

[49] reactnative.dev. *Out-of-Tree Platforms* [online]. 22. 4. 2024 [cit. 24. 4. 2024]. Dostupné z: https://reactnative.dev/docs/out-of-tree-platforms.

[50] reactnative.dev. *Introduction · React Native* [online]. 21. 4. 2024 [cit. 28. 4. 2024]. Dostupné z: https://reactnative.dev/docs/getting-started.

[51] reactnative.dev. *Style · React Native* [online]. 22. 4. 2024 [cit. 29. 4. 2024]. Dostupné z: https://reactnative.dev/docs/style.

[52] pchmn. *Recommended libraries* [online]. 23. 3. 2024 [cit. 29. 4. 2024]. Dostupné z: https://callstack.github.io/react-native-paper/docs/guides/recommended-libraries.

[53] luca992. *Add wasmJS target* [online]. 2. 1. 2024 [cit. 26. 2. 2024]. Dostupné z: https://github.com/Kamel-Media/Kamel/issues/85.

[54] JOSHI, Amrata. ***Custom design systems in Compose*** [online]. 21. 11. 2018 [cit. 5. 3. 2024]. Dostupné z: https://hub.packtpub.com/kotlin-based-framework-ktor-1-0-released-with-features-like-sessions-metrics-call-logging-and-more/.

[55] Daniele Baroncelli . *Ktor client for Kotlin/Wasm* [online]. 16. 2. 2024 [cit. 5. 3. 2024]. Dostupné z: https://youtrack.jetbrains.com/issue/KTOR-5587/Ktor-client-for-Kotlin-Wasm.

[56] IlyaGulya. *[WIP] Support wasmJs target* [online]. 16. 1. 2024 [cit. 26. 2. 2024]. Dostupné z: https://github.com/cashapp/sqldelight/pull/4965.

[57] Burtan. *Example broken* [online]. 28. 2. 2023 [cit. 26. 2. 2024]. Dostupné z: https://github.com/dellisd/sqldelight-sqlite-wasm/issues/1.

[58] DRSchlaubi. *Support for WASM* [online]. 18. 5. 2023 [cit. 26. 2. 2024]. Dostupné z: https://github.com/qdsfdhvh/compose-imageloader/issues/151.

[59] AAkira. *Aakira/Napier* [online]. 4. 1. 2024 [cit. 24. 2. 2024]. Dostupné z: https://github.com/AAkira/Napier.

[60] JetBrains. *Use platform-specific APIs* [online]. 18. 12. 2023 [cit. 5. 2. 2024]. Dostupné z: https://www.jetbrains.com/help/kotlin-multiplatform-dev/multiplatform-connect-to-apis.html.

[61] Mozilla. *Progressive web apps* [online]. 2019 [cit. 1. 5. 2024]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_appsl.

[62] Google. *Workbox*. web.dev [online]. 10. 1. 2022 [cit. 1. 5. 2024]. Dostupné z: https://web.dev/learn/pwa/workbox.

[63] BANES, Chris. *Implementing Material Design in Your Android app*. Android Developers [online]. 24. 10. 2014 [cit. 15. 2. 2024]. Dostupné z: https://android-developers.googleblog.com/2014/10/implementing-material-design-in-your.html.

[64] HALL, Stephen. *What exactly is this so-called 'Material Design 2,' and what will it look like?*. 9To5Google.com [online]. 26. 4. 2014 [cit. 26. 2. 2024]. Dostupné z: https://9to5google.com/2018/04/26/what-is-material-design-2-examples-launch-io/.

[65] WHITE, Colin. *Multiplatform image loading: Coil 3.0*. CashApp [online]. 12. 7. 2023 [cit. 26. 2. 2024]. Dostupné z: https://code.cash.app/multiplatform-image-loading.

[66] Google. *Save data in a local database using Room*. Android Developers [online]. 3. 1. 2024 [cit. 11. 2. 2024]. Dostupné z: https://developer.android.com/training/data-storage/room.

[67] BEKKHUS, Simen. *Testing React Native Apps*. jestjs.io [online]. 12. 9. 2023 [cit. 5. 5. 2024]. Dostupné z: hhttps://jestjs.io/docs/tutorial-react-native.

[68] el...@google.com. *Make Room KMP 20*. Google Issue Tracker [online]. 5. 9. 2023 [cit. 11. 2. 2024]. Dostupné z: https://issuetracker.google.com/issues/299168035.

[69] JetBrains. *Kotlin Multiplatform* [online]. 10. 11. 2023 [cit. 6. 2. 2024]. Dostupné z: https://kotlinlang.org/docs/multiplatform.html.

[70] Jetbrains. *New Material 3 components in common code* [online]. 1. 1. 2023 [cit. 21. 2. 2024]. Dostupné z: https://blog.jetbrains.com/kotlin/2023/11/compose-multiplatform-1-5-10-release/#new-material-3-components-in-common-code.

[71] Mozilla. *Progressive web apps* [online]. 25. 10. 2023 [cit. 25. 2. 2024]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.

[72] Mozilla. *What is a progressive web app?* [online]. 4. 7. 2023 [cit. 7. 3. 2024]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Guides/What_is_a_progressive_web_app.

[73] Mozilla. *Web app manifests* [online]. 28. 6. 2023 [cit. 7. 3. 2024]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Manifest.

[74] Mozilla. *Service Worker API* [online]. 8. 2. 2024 [cit. 7. 3. 2024]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API.

[75] STEINER, Thomas. *Progressive web apps* [online]. 31. 10. 2023 [cit. 26. 2. 2024]. Dostupné z: https://developer.chrome.com/blog/wasmgc.

[76] Kamel. *Kamel-Media/Kamel* [online]. 24. 1. 2024 [cit. 6. 2. 2024]. Dostupné z: https://github.com/Kamel-Media/Kamel.

[77] Android Developers. *Migrate your build configuration from Groovy to Kotlin* [online]. 12. 3. 2024 [cit. 20. 3. 2024]. Dostupné z: https://developer.android.com/build/migrate-to-kotlin-dsl.

[78] Android Developers. *Migrate your build to version catalogs* [online]. 19. 2. 2024 [cit. 20. 3. 2024]. Dostupné z: https://developer.android.com/build/migrate-to-catalogs.

[79] LADD, Seth. *Announcing Flutter beta 1: Build beautiful native apps*. Google Developers [online]. 27. 2. 2018 [cit. 11. 3. 2024]. Dostupné z: https://arstechnica.com/gadgets/2018/02/google-starts-a-push-for-cross-platform-app-development-with-flutter-sdk/.

[80] pub.dev. *The official repository for Dart and Flutter packages.* [online]. 11. 3. 2024 [cit. 11. 3. 2024]. Dostupné z: https://pub.dev/.

[81] pub.dev. *Package scoring and pub points* [online]. 11. 3. 2024 [cit. 11. 3. 2024]. Dostupné z: https://pub.dev/help/scoring.

[82] FOWLER, Martin. *Unit Tets*. martinfowler.com [online]. 5. 5. 2014 [cit. 20. 4. 2024]. Dostupné z: https://martinfowler.com/bliki/UnitTest.html.

[83] BOSE, Shreya. *UI Testing: A Detailed Guide*. browserstack.com [online]. 18. 6. 2023 [cit. 20. 4. 2024]. Dostupné z: https://www.browserstack.com/guide/ui-testing-guide.

[84] PITTET, Sten. *What is code coverage?*. Atlassian [online]. 2024 [cit. 17. 4. 2024]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/code-coverage.

Obsah příloh

- `readme.txt` ..... stručný popis obsahu média
- `apks` ..... adresář s vygenerovanými APK soubory
- `web` ..... adresář s webovými aplikacemi
- `tests` ..... adresář s vygenerovanými daty z testování
- `performance` ..... adresář s vygenerovanými daty z měření výkonu
- `src`
  - `links.txt` ..... linky na repozitáře se zdrojovými kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF