Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Vývoj cross-platformové mobilní aplikace s využitím nástrojů Flutter

Jakub Smolík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jakub Smolík

Systémové inženýrství a informatika
Informatika

Název práce

Vývoj cross-platformové mobilní aplikace s využitím nástrojů Flutter

Název anglicky

Cross-platform mobile application development using Flutter tools

---

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku cross-platformového vývoje aplikací pro mobilní operační systémy. Hlavním cílem práce je analyzovat cross-platformové SDK (Software Development Kit) Flutter na platformě iOS a Android.

Dílčí cíle práce jsou:
– charakterizovat problematiku vývoje aplikací pro iOS a Android,
– analyzovat vývoj pomocí cross-platformového SDK Flutter,
– zhodnotit kompatibilitu komponent uživatelského rozhraní aplikace vyvinuté v SDK Flutter.

Metodika

Teoretická část bakalářské práce se bude zakládat na analýze a rešerši odborných zdrojů.

V praktické části práce bude na základě poznatků zjištěných v analytické části zhodnocen cross-platformový vývoj mobilní aplikace s využitím SDK Flutter. Budou hodnoceny rozdíly v komponentách uživatelského rozhraní ve vztahu k vývoji nativnímu.

Na základě syntézy teoretických a praktických poznatků budou zpracovány závěry bakalářské práce.

Doporučený rozsah práce

30–40 stran

Klíčová slova

Flutter, cross-platform mobile development, Android, iOS, Dart, mobile development

---

Doporučené zdroje informací

BIESSEK, Alessandro. Flutter for Beginners: An Introductory Guide to Building Cross-Platform Mobile Applications with Flutter and Dart 2 [online]. N/A [cit. 2020-11-23]. ISBN 9781788990523.

Flutter documentation [online]. [cit. 2020-11-23]. Dostupné z: https://flutter.dev/docs

LACKO, Ľuboslav. Vývoj aplikací pro Android. Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

LACKO, Ľuboslav. Vývoj aplikací pro iOS. Přeložil Martin HERODEK. Brno: Computer Press, 2018. ISBN 978-80-251-4942-3.

SHAH, Kewal, Harsh SINHA a Payal MISHRA. Analysis of Cross-Platform Mobile App Development Tools. In: 2019 IEEE 5th International Conference for Convergence in Technology (I2CT) [online]. IEEE, 2019, 2019, s. 1-7 [cit. 2020-11-23]. ISBN 978-1-5386-8075-9. Dostupné z: doi:10.1109/I2CT45611.2019.9033872

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Michal Stočes, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 24. 11. 2020

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 25. 11. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 03. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci Vývoj cross-platformové mobilní aplikace s využitím nástrojů Flutter jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 12.3.2021 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Michalu Stočesovi, Ph.D.
za ochotnou výpomoc s prací a konzultace v neobvyklých částech dnech.

Vývoj cross-plartformové mobilní aplikace s využitím nástrojů Flutter

Abstrakt

Cílem této bakalářské práce je analyzovat cross-platformové SDK Flutter na platformě iOS a Android. Dílčími cíli práce jsou charakterizování problematiky vývoje aplikací pro iOS a Android, analyzování vývoje mobilní aplikace pomocí cross-platformového SDK Flutter a zhodnocení kompatibility komponent uživatelského rozhraní aplikace vyvinuté právě za použití SDK Flutter. Jako modelový příklad aplikace vyvinuté s SDK Flutter je mobilní aplikace využívající data zpracována z interních souborů pro zobrazení jich do náhodně zvolených komponent SDK Flutter. Uživatel má nejen možnost data procházet, ale i zabavit sebe a okolí použitím integrované gamifikace. Aplikace je určena pro platformy iOS a Android, komponenty jsou porovnány mezi platformami.

Klíčová slova: Flutter, cross-platformový vývoj mobilní aplikace, Android, iOS, Dart, vývoj mobilní aplikace

Cross-platform mobile application development using Flutter tools

Abstract

The aim of this bachelor's thesis is to analyse the cross-platform SDK Flutter on the iOS and Android platforms. Sub-objectives of the work are to characterize the problematics of iOS and Android app development, analyze the development of a mobile app using the cross-platform SDK Flutter and assess the compatibility of the app user interface components developed just using SDK Flutter. As a model example of an application developed with SDK Flutter is a mobile application using data processed from internal files to display them into randomly selected SDK Flutter components. The user not only has the ability to browse the data, but also to entertain himself and others by using integrated gamification. The app is for iOS and Android platforms, with components compared between platforms.

Keywords: Flutter, cross-platform mobile development, Android, iOS, Dart, mobile development

Obsah

1 Úvod **7**

2 Cíl práce a metodika **8**

2.1 Cíl práce 8

2.2 Metodika 8

3 Literární rešerše **9**

3.1 Výhody cross-platformového vývoje 9

3.2 Nevýhody cross-platformového vývoje 9

3.3 Výhody nativních aplikací 10

3.4 Vývoj nativní aplikace pro iOS 10

3.4.1 Nástroje a náležitosti pro vývoj 10

3.4.2 Programové prostředí pro vývoj Xcode 11

3.4.3 Programovací jazyk Swift 13

3.4.4 Softwarová architektura aplikace 14

3.4.5 Architektura aplikace pro iOS MVC 15

3.4.6 Architektura aplikace pro iOS MVVM 16

3.4.7 Architektura aplikace pro iOS MVP 16

3.4.8 Architektura aplikace pro iOS VIPER 18

3.5 Vývoj nativní aplikace pro Andriod 18

3.5.1 Nástroje a náležitosti pro vývoj 18

3.5.2 Programové prostředí pro vývoj Android Studio 20

3.5.3 Programovací jazyk Kotlin 23

3.5.4 Architektura aplikace pro Android MVC 24

3.5.5 Architektura aplikace pro Android MVP 25

3.5.6 Architektura aplikace pro Android MVVM 26

3.6 Vývoj cross-platformové aplikace s SDK Flutter 26

3.6.1 Nástroje a náležitosti pro vývoj 27

3.6.2 Programovací jazyk Dart 27

3.6.3 Flutter jako framework založený na widgetech 29

3.6.4 Architektura Frameworku Flutter 30

3.6.5 Architektura aplikace ve Flutter BLoC 31

3.6.6 Flutter pluginy 32

4 Praktická část **34**

4.1 Použité technologie pro vývoj 35

4.2 Návrh navigace aplikace 37

4.3 Vytvoření projektu aplikace 38

4.4 Nastavení aplikace pro widgety knihovny material 38
4.4.1 Flutter localizations 39
4.5 Widget domovské obrazovky 40
4.5.1 Zpracování dat ze souboru .json 42
4.6 Uživatelská navigace v aplikaci 42
4.7 Obrazovky aplikace 46
4.7.1 Použití widgetu *StreamBuilder* 46
4.7.2 Třída vrstvy architektury BLoC 48
4.7.3 Tvorba obrazovek z widgetů 49
4.7.4 Tvorba seznamu 50
4.7.5 Zakomponování pluginu třetí strany 51
4.8 Konfigurace cross-platformové aplikace 51

5 Výsledky a diskuse 52

6 Závěr 57

7 Seznam použitých zdrojů 58

8 Přílohy 60

Seznam obrázků

Obrázek 1 - MVC iOS model (Idryshev).....15
Obrázek 2 - iOS MVVM iOS model (Idryshev).....16
Obrázek 3 - MVP iOS model (Idryshev).....17
Obrázek 4 - VIPER iOS model (Idryshev).....18
Obrázek 5 - MVC Android model (MISHRA).....24
Obrázek 6 - MVP Android model (MISHRA).....25
Obrázek 7 - MVVM Android model (MISHRA).....26
Obrázek 8 - Struktura widgetů ve Flutteru (Shah, 2019).....29
Obrázek 9 - Schéma architektury Flutteru (Shah, 2019).....30
Obrázek 10 - BLoC Flutter vrstvy (Kayfitz).....31
Obrázek 11 - Struktura vzoru BLoC (Kayfitz).....32
Obrázek 12 - Přehled architektury pluginů ve Flutteru (Flutter.dev, 2020).....33
Obrázek 13 - Wireframe ukázkové aplikace.....35
Obrázek 14 - Návrh navigace ukázkové aplikace.....37
Obrázek 15 - Rozvržení widgetu *Scaffold* pro ukázkovou aplikaci.....43
Obrázek 16 - Rozvržení každé obrazovky ukázkové aplikace.....46
Obrázek 17 - Zhodnocení widgetu *ListView*.....53
Obrázek 18 - Zhodnocení widgetu *Scaffold*.....54
Obrázek 19 - Náhled widgetů obrazovky *MovieScreen*; Android vlevo, iOS vpravo.....55
Obrázek 20 - Náhled obrazovky po využití pluginu *share*; Android vlevo, iOS vpravo.....56

Seznam ukázek zdrojového kódu

Zdrojový kód 1 - Hlavní metoda spuštění aplikace 39
Zdrojový kód 2 - Flutter localizations dependence 39
Zdrojový kód 3 - Widget domovské obrazovky 40
Zdrojový kód 4 - Metody *Future* domovské obrazovky 41
Zdrojový kód 5 - Zpracování dat dle modelů 42
Zdrojový kód 6 - Uživatelská navigace *bottomNavigationBar* 44
Zdrojový kód 7 - Uživatelská navigace *appBar* 45
Zdrojový kód 8 - Konfigurace widgetu *StreamBuilder* 46
Zdrojový kód 9 - Komunikace s BLoC vrstvou z uživatelského rozhraní 47
Zdrojový kód 10 - Třída vrstvy BLoC 48
Zdrojový kód 11 - Konfigurace widgetu *SingleChildScrollView* 49
Zdrojový kód 12 - Konfigurace widgetu *ListView* 50
Zdrojový kód 13 - *share* dependence 51
Zdrojový kód 14 - Vložení knihovny *share.dart* do souboru 51

1 Úvod

Nelze popřít rychlý růst mobilního průmyslu i služeb, které jsou s ním spojeny. S rostoucím počtem uživatelů mobilních zařízení přichází příležitost pro taktní marketing a nové obchodní strategie. Mezi prvními osvojiteli jsou IT průmysl, maloobchodníci, podniky, a dokonce i **e-commerce** (elektronický obchod) startupy. Díky hektické a rychlé povaze velkých korporací a konglomerátů se stalo klíčovým řízení efektivity času a pracovní zátěže. Hlavní úlohu při uspokojování potřeb uživatelů hrají *počítačové* programy určené pro provoz na mobilním zařízení, běžně označované jako mobilní aplikace nebo pouze *aplikace*. Než vývojář začne s tvorbou aplikace, je důležité se rozhodnout, na jakou platformu se chce zaměřit. V této práci používám pojmy mobilní operační systém a platforma zaměnitelně.

Podle celosvětových statistik podílu operační mobilních operačních systémů na trhu (StatCounter) tvoří ze **72% Android** vyvinut společností Google, z **27% iOS** vyvinut společností Apple Inc. a **1% zbylé** – Samsung, KaiOS, Windows (prosinec, 2020). Vývoj pro určitou platformu zvlášť, nativní vývoj aplikace pro danou platformu, vyžaduje zcela odlišné dovednosti. Pro softwarové vývojáře chtějící sdílet aplikaci pro obě platformy je nativní vývoj nákladnější z hlediska času, financí a úsilí. Vývoj nativních aplikací má své výhody, ale chybí zde možnost přepoužít kód mezi platformami. Kódy jsou si navzájem nekompatibilní, proto přichází na scénu cross-platformový vývoj. Nástroje pro cross-platformový vývoj umí sestavit z jednoho zdrojového kódu aplikace pro více platforem. Volba technologie vývoje mobilní aplikace záleží na funkcionalitách výsledného produktu. Neexistuje přímé tvrzení, jaká technologie je nejlepší, volba záleží na daném projektu. (Shah, 2019)

Tato bakalářská práce se zabývá porovnáním nativního a cross-platformového vývoje a zhodnocením cross-platformového SDK Flutter, které umožňuje tvorbu aplikace pro platformy Android a iOS napsanou jedním kódem v programovacím jazyce Dart.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této bakalářské práce je analyzovat cross-platformové SDK (Software Development Kit) Flutter pro mobilní operační systémy Android a iOS.

Dílčími cíli jsou:

- charakterizování problematiky vývoje aplikací pro iOS a Android jednotlivě,
- analýza vývoje aplikace pomocí cross-platformového SDK Flutter,
- zhodnocení kompatibility komponent uživatelského rozhraní aplikace vyvinuté v SDK Flutter.

2.2 Metodika

Teoretická část se bude zakládat na analýze a rešerši odborných zdrojů. V praktické části práce bude na základě poznatků zjištěných v analytické části zhodnocen cross-platformový vývoj mobilní aplikace s využitím SDK Flutter. Budou hodnoceny rozdíly v komponentách uživatelského rozhraní ve vztahu k vývoji nativnímu. Na základě syntézy teoretických a praktických poznatků budou zpracovány závěry bakalářské práce.

3 Literární rešerše

Cross-platformová aplikace je ta, která funguje mezi více platformami (operačními systémy) a zařízeními různých typů (smartphone, chytrá TV, chytré hodinky). Ve světě mobilních aplikací se za cross-platformovou, nebo někdo multi-platformní aplikaci považuje aplikace fungující na operačních systémech iOS a Android. (Klubnikin, 2017)

Nativní aplikace je ta, která splňuje požadavky konkrétního operačního systému tím, že podporuje vlastní **SDK** (sada nástrojů pro vývoj softwaru) a množinu technologií jako na příklad jsou hardwarová paměť, kamera, senzory a různé programy v zařízení nainstalované. (Klubnikin, 2017)

3.1 Výhody cross-platformového vývoje

- Nástroje pro cross-platformový vývoj aplikace jsou též označovány jako **WORA** (Write Once, Run Anywhere), což je hlavní výhoda a fakt za rozhodnutím se pro tento typ vývoje. Aplikace je implementována podle jednoho zdrojového kódu a může být nasazena na více platformách.
- Nástroje pro cross-platformový vývoj obvykle používají známé programovací jazyky a syntaxe, práce s nimi je tedy rychlá a efektivní. Je zde výhoda znalosti jednoho jazyku a technologie pro více platforem.
- V komunitě rozšířené technologie a jejich nástroje s **open source** licencí mají nemalou podporu a nové moduly jsou tak průběžně doplňovány, aktualizovány a revidovány dle potřeby. Není zde přímá vazba na určitou skupinu vývojářů.
- Díky jednoduššímu a časově méně náročnějšímu vývoji jsou aplikace dostupnější z hlediska ceny.
- Opravy a aktualizace aplikace mohou být vzdány pro platformy v jeden čas. (Shah, 2019)

3.2 Nevýhody cross-platformového vývoje

- Nativní aplikace jsou navrženy tak, aby pro každou platformu fungovaly bezchybně. Jejich prodleva při používání je minimální díky rychlejšímu vykreslování procesoru.
- Nativní aplikace mají hlubší integraci se zařízením, zajišťují lepší spolupráci například s akcelerometry, lokalizačními službami, kamerami. Různé senzory zařízení fungují plynuleji s nativním kódem aplikace.

- Nativní aplikace poskytují lepší **UX** (User Experience = uživatelský zážitek) z důvodu kvalitnějšího rychlejšího vykreslování, které je možno dosáhnout pouze v nativní aplikaci.
- V obchodech pro distribuci a prodej aplikací jsou cross-platformové aplikace méně podporovány. To může snížit dosah nabídky aplikace potencionálnímu uživateli.
- Cross-platformové aplikace bývají většinou složitější než nativní. To zapříčiňuje menší stabilitu – čím jsou aplikace složitější, tím je pravděpodobnější jejich zamrzávání, nebo pád. (Shah, 2019)

3.3 Výhody nativních aplikací

Nativní aplikace určené pro specifickou platformu disponují oproti cross-platformním nebo webovým lepším zabezpečením, plynulejší a rozsáhlejší interakcí s uživatelem, plným přístupem ke všem funkcím zařízení, menší chybovostí a lepším celkovým dojmem z používání aplikace uživatele. (Technocrats, 2019)

3.4 Vývoj nativní aplikace pro iOS

3.4.1 Nástroje a náležitosti pro vývoj

Nativní vývoj mobilních aplikací pro operační systém (déle **platformu**) iOS není možný ze zařízení s jiným operačním systémem nežli MacOS. Pomineme-li možnost běhu MacOS na neoficiálních zařízeních, vzniká nám první náležitost, a to zařízení s MacOS značky Apple Inc. (Lacko, 2018)

„Abyste mohli své aplikace spouštět na více reálných zařízeních a po otestování je šířit přes App Store, je potřeba se zaregistrovat do placeného programu na vývojářském portálu developer.apple.com.“ (Lacko, 2018) Uživatel telefonu na platformě iOS nemůže z bezpečnostních důvodů instalovat aplikace, které nejsou schváleny v obchodě App Store. (App Store Review Guidelines, 2020)

Kromě hardwaru předpokládá vývoj základní znalosti:

- „Znalost základů programování: výhodou je znalost některého z moderních programovacích jazyků, jako jsou Java, C++, C#. Jazyk **Swift**, používaný na vývoj aplikací pro iOS, je těmto jazykům velmi blízký.

- Výhodou je znalost základních principů **objektově orientovaného programování**.
- Znalost základů SQL (nejlépe SQLite, ale postačuje základní všeobecný přehled).
- Znalost základů formátů XML a JSON.“ (Lacko, 2018)

IDE (Integrated Development Enviroment), do češtiny přeloženo jako *Programové prostředí pro vývojáře*, je striktně dané pro vývoj pro platformu iOS. Je jím vývojové Xcode, též společnosti Apple. Xcode je v aplikačním obchodě pro MacOS k dispozici zdarma. (Lacko, 2018)

3.4.2 Programové prostředí pro vývoj Xcode

Xcode je univerzální aplikace, která běží na procesorech Intel a Apple Silicon. Zahrnuje sjednocenou sadu **MacOS SDK**, která zahrnuje všechny frameworky, kompilátory, debuggery a další nástroje pro vývoj aplikace nativně běžící na Apple Silicon a procesoru Intel x86_64. (Introducing Xcode 12, 2020)

Xcode disponuje následující funkcionalitou:

- **Editor zdrojového kódu:** navrhuje dokončení rozepsaného kódu, edituje psaný kód, zvýrazňuje syntaxe, ukazuje zprávy v bublinách obsahující varování, chyby a další kontextové informace přímo spjaté se psaným kódem.
- **Pomocný editor:** Tlačítko *Assistant* rozdělí editor na dvě části. Vytvoří sekundární panel, který automaticky zobrazí nejužitečnější soubory na základě editovaného kódu.
- **Editor verzí:** Editor verzí Xcode zobrazuje běžící časovou osu *commitů*, pomáhá určit problémy a graficky porovnává soubory zdrojových kódů s jejich historickými verzemi při plné podpoře systémů Subversion a Git source control (**SCM**).
- **Integrované rozhraní *Builder*:** jedná se o rozhraní, které umožňuje vytvoření obrazovek aplikace pomocí grafického skládání komponent uživatelského rozhraní bez nutnosti psaní kódu. Jednotlivé komponenty se poté v editoru Xcode v kódu nadefinují.

- **Simulátor:** s využitím **iOS SDK** (sada nástrojů pro vývoj softwaru) může Xcode vytvářet, instalovat, spouštět a ladit aplikace **Cocoa Touch** (vysvětleno níže) v simulátoru přímo v MacOS. Vývoj aplikace, tedy pracovní postup je tímto usnadněn a urychlen.

- **Integrovaný *Build* systém:** zpracovává nejkomplexnější a pro Mac nejnáročnější sestavení aplikace. Automaticky podepíše, udělí provizi a nainstaluje aplikaci na zařízení s iOS.

- **Překladače:** výkonný open-source překladač **LLVM** (Low Level Virtual Machine) pro jazyky C, C++ a Objective C je zabudován v Xcode. K dispozici je z Terminálu MacOS. Díky tomu je kompilace kódu rychlá a efektivní především pro procesory v zařízeních s iOS.

- **Grafický debugger:** umožňuje ladit aplikaci přímo v Xcode. Najetím kurzor myši na proměnnou zobrazí její hodnotu. Funkcí *Quick Look* (Rychlý pohled) nám zobrazí podrobnější informace. Stiskem pravého tlačítka na myši přidá zvolenou proměnnou do seznamu sledování.

- **Kontinuální integrace:** Xcode Server ovládá boty na straně serveru, kteří kontinuálně vytvářejí, analyzují testují a archivují projekty v Xcode. **Xcode IDE** konfiguruje tyto boty, analyzuje výsledky testů a může vysledovat chybu při sestavení aplikace.

- **Katalog assetů:** editor katalogu assetů, který spravuje obrázky v aplikaci. Seskupuje různá rozlišení jednoho obrázku. Při sestavování aplikace zkompiluje katalog obrázků do nejefektivnějšího balíčku pro finální distribuci.

- ***Open Quickly*:** Stiskem nastavené klávesové zkratky okamžitě otevře jakýkoliv soubor v pracovním prostoru za pomoci primárního editoru. *Open Quickly* je nezbytný nástroj pro efektivní vývoj.

- **Zachycení obsahu OpenGL:** stiskem jediného tlačítka pořídíme aktuální prezenci obsahu OpenGL. Xcode zobrazuje informace o shaderu a může vizuálně konstruovat zachycený obsah v debuggeru.

- **Kompletní dokumentace:** možnost snadného vyhledání informací v nápovědě, nebo prohlížeči API dokumentace.

- **Zobrazení chyb aktuálně:** upozornění na chyby a nepřesnosti v kódu dopuštěných při editování ještě před pokusem o kompilaci.

- **Opravit:** funkcionalita umožňující opravu syntaktické chyby v kódu, na kterou Xcode upozorní díky předchozí zmíněné funkcionalitě.

- **Rychlá nápověda:** Během psaní a editování kódu se zobrazuje zkrácená API dokumentace, včetně komentářů programátora. Během dokončování kódu je uveden stručný přehled. V oblasti *Utility* je k dispozici více odkazů.

- **XCTest framework:** Rozhraní XCTest API usnadňuje vytváření unit testů, které prověřují funkčnost aplikace na počítačích Mac, iPad, iPhone nebo v simulátoru.

- **Statická analýza:** najde chyby v kódu ještě před spuštěním kompilátoru tím, že nechá vestavěný analyzátor vyzkoušet tisíce možných cest běhu kódu v během několika vteřin. Programátor obdrží hlášení o potencionálních chybách, které by bez použití analýzy mohly zůstat skryté, nebo nemožné replikovat. (Xcode IDE, 2020)

3.4.3 Programovací jazyk Swift

Nativní aplikace pro iOS se primárně píše v programovacím jazyce **Swift**. „Programovací jazyk Swift je poměrně mladý, poprvé byl prezentován v červnu roku 2014 na konferenci **Apple WWDC** (Worldwide Developers Conference). V roce 2014 byla k dispozici i první verze. Byla integrovaná do vývojového prostředí Xcode ve verzi 6. O rok později Apple zveřejnil zdrojový kód Swiftu, od té doby je tedy tento programovací jazyk typu open-source.

Swift je bezpečný, rychlý a interaktivní programovací jazyk, který kombinuje nejlepší moderní programátorské myšlení s osvědčenými postupy, ať už softwarových inženýrů Apple či open-source komunity. Kompilátor je optimalizovaný na výkon a jazyk je optimalizovaný na efektivní vývoj. Apple jazyk Swift v době jeho uvedení porovnával na svých stránkách s jiný-mi jazyky z hlediska výkonu a tvrdil, že Swift je 2,6krát rychlejší než Objective-C a 8,4krát rychlejší než Python.

Velmi důležité je, že Swift se dá velmi rychle naučit, neobsahuje žádné hůře pochopitelné speciality, jako jsou ukazatele v programovacím jazyce C. Navíc Apple umožnil experimentovat s kódem ve speciálním prostředí **Playground**.

Kód ve Swiftu je kompilovaný a optimalizovaný tak, aby co nejvíce využil moderní hardware mobilních zařízení Apple. Syntaxe a standardní knihovna byly navrženy na základě osvědčené-ho principu, aby způsob zápisu kódu byl jasný a přehledný. Kombinace bezpečnosti a rychlosti dělá ze Swiftu ideální volbu pro všechny typy aplikací od nejjednodušších až po ty náročné, jako jsou například práce s databázemi či emulátor jiného operačního systému.

Tvůrci Swiftu se kromě jednoduchosti soustředili i na bezpečnost, tento programovací jazyk
je tedy z hlediska bezpečnosti poměrně striktní. Vyžaduje například inicializaci proměnných před jejich prvním použitím, definování hodnot konstant či neumožňuje, aby objekty bez to-ho, aby to vývojář při inicializaci vysloveně připustil, měly hodnotu nil. Pokud to pro některý objekt vědomě připustí, musí takovouto situaci v kódu náležitě ošetřit.

Jelikož se do roku 2015 aplikace pro iOS vyvíjely hlavně v jazyce Objective-C, ve vývojovém prostředí Xcode je zabudovaná kompatibilita s Objective-C. V praxi to znamená, že v nových projektech využívajících moderní Swift je možné využívat knihovny a frameworky vytvořené v Objective-C. Na jejich integraci je potřeba definovat hlavičkový soubor, takzvaný bridging header, který umožní použití kódu Objective-C v projektech psaných ve Swiftu. Příchod Swiftu neznamená žádné problémy s existujícími aplikacemi v Objective-C, protože v jedné aplikaci pro iOS může existovat kód Objective-C i Swift. Není problém se psaním nových modulů v programovacím jazyce Swift, které jsou kompatibilní s existujícím kódem v Objective-C.“ (Lacko, 2018)

3.4.4 Softwarová architektura aplikace

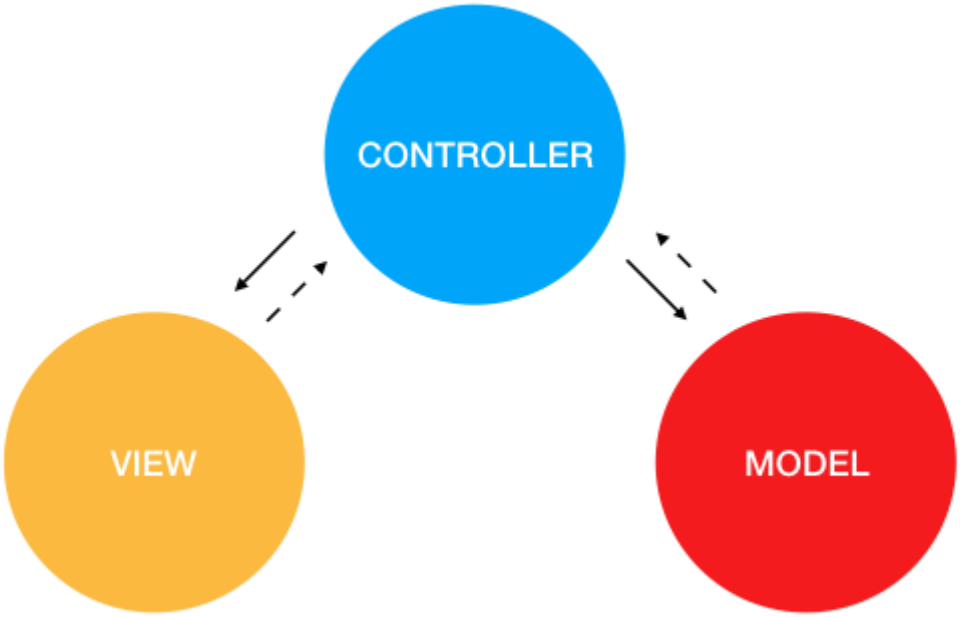
Softwarová architektura je důležitou disciplínou softwarového inženýrství, zasluhuje se o kvalitu výsledného systému, aplikace. Softwarová architektura získala popularitu v průběhu devadesátých let, avšak myšlenka zajištění softwaru na vysoké úrovni, z níž plyne, se objevila v sedmdesátých letech. (Agile Software Architecture, 2013)

Softwarová architektura systému je sada struktur potřebných k uvažování o systému (aplikace), která zahrnuje softwarové prvky, vztahy mezi nimi a vlastnosti obou. (Bass, 2003)

3.4.5 Architektura aplikace pro iOS MVC

MVC (Model-View-Controller) vzor architektury pro nativní vývoj doporučený přímo Apple.

MVC je výsledkem obrovské práce. Vynalezl ji Trygve Reenskaug jako výsledek práce na projektu Dynabook v Xerox PARC v roce 1979. Dynabook je osobní počítač pro děti všech věkových kategorií. A byl to opravdu revoluční projekt. Dynabook měl usnadnit používání počítačů a zároveň umožnit uživateli spravovat složité aplikace. Tehdy byly poprvé vytvořeny základy grafických rozhraní a koncepty *uživatelsky přívětivého rozhraní*. Nejedná se o schéma rozkladu modulů aplikace. Nikdo nemůže poskytnout řešení s určitým množstvím tříd, které bude fungovat pro jakýkoliv problém, určitou logiku a hlavní cíl určité aplikace. Doporučuje se vymyšlení architektury přímo na daný problém. (Idryshev)



Obrázek 1 - MVC iOS model (Idryshev)

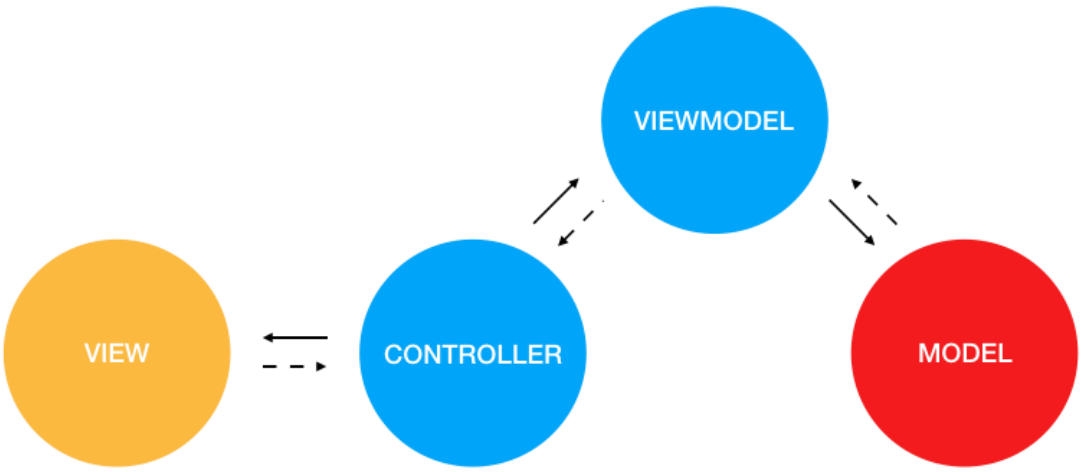
View je to, co uživatel vidí na obrazovce, **Model** jsou *data* – jejich struktura a **Controller** je prostředník viz Obrázek 1, kde šipky vyjadřují komunikaci, dotazy a odpovědi mezi vrstvami. Data z modelu zobrazuje ve View uživateli. Na základě interakce uživatele z View pracuje s daty v Modelu.

Byť se jedná o vzor architektury doporučený přímo Apple, je špatný a v praxi se nepoužívá. Vzor pracuje pouze s View a daty, přičemž mobilní aplikace mají spoustu další logiky, která by zde měla být vykonávána v Controlleru, což není pěkné řešení.

Controller je zodpovědný za správu hierarchie View, kterému je přiřazen. Reaguje na načtení, zobrazení, zmizení View a podobně. Drží logiku dál od View a Modelů pro znovupoužitelnost a menší zátěž aplikace. (Idryshev)

3.4.6 Architektura aplikace pro iOS MVVM

Jednou z komunitou nalezených alternativ k MVC.



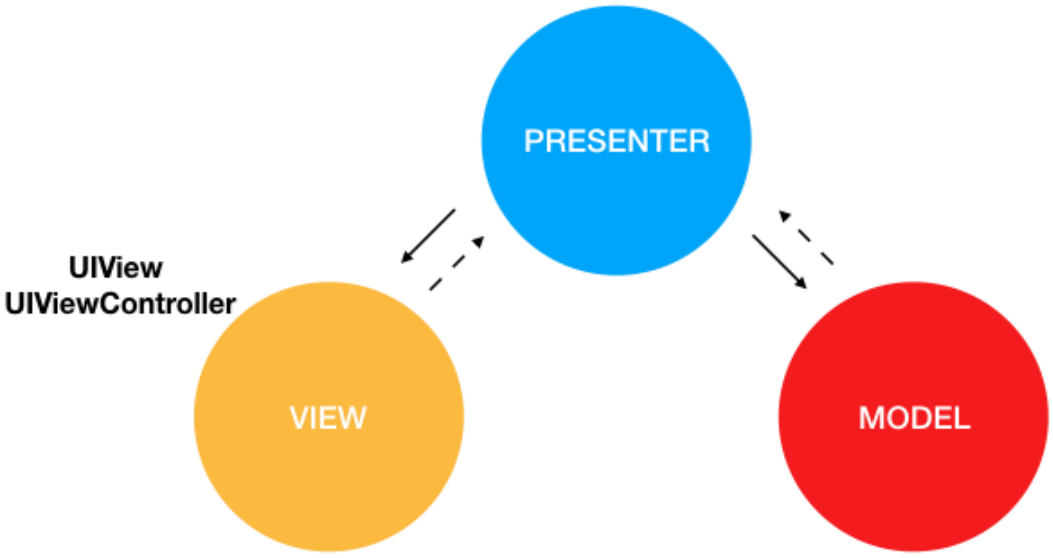
Obrázek 2 - iOS MVVM iOS model (Idryshev)

MVVM přidává novou vrstvu **ViewModel** k rozdělení kódu za pomoci **Controlleru** viz Obrázek 2. Ve skutečnosti to ale není řešením a nevyřeší to všechny problémy, které architektura MVC měla. Komunita je rozdělena na ty, kterým se MVVM zamlouvá a na její odpůrce. (Idryshev)

3.4.7 Architektura aplikace pro iOS MVP

Jedná se o další pokus komunity o vyřešení otázky volby architektury. Začíná se považováním **ViewControlleru** za View a veškerá logika přejde do nové třídy **Presenter**. Toto řešení se populárním nestalo, protože ve finále byly všechny problémy přesunuty na ViewController a Presenter.

MVP zavedl v roce 1996 Mike Potel jako modifikaci MVC. Ve své práci na MVP Potel naznačil, že není třeba dělit widgety na View a Controllery. Moderní uživatelská rozhraní operačního systému již poskytují většinu funkcí Conrollerů ve třídách View, a proto se zdá být Controller poněkud nadbytečný. Controller byl tedy odebrán a byla vytvořena nová třída Presenter jako prostředník mezi View a Modelem. (Idryshev)

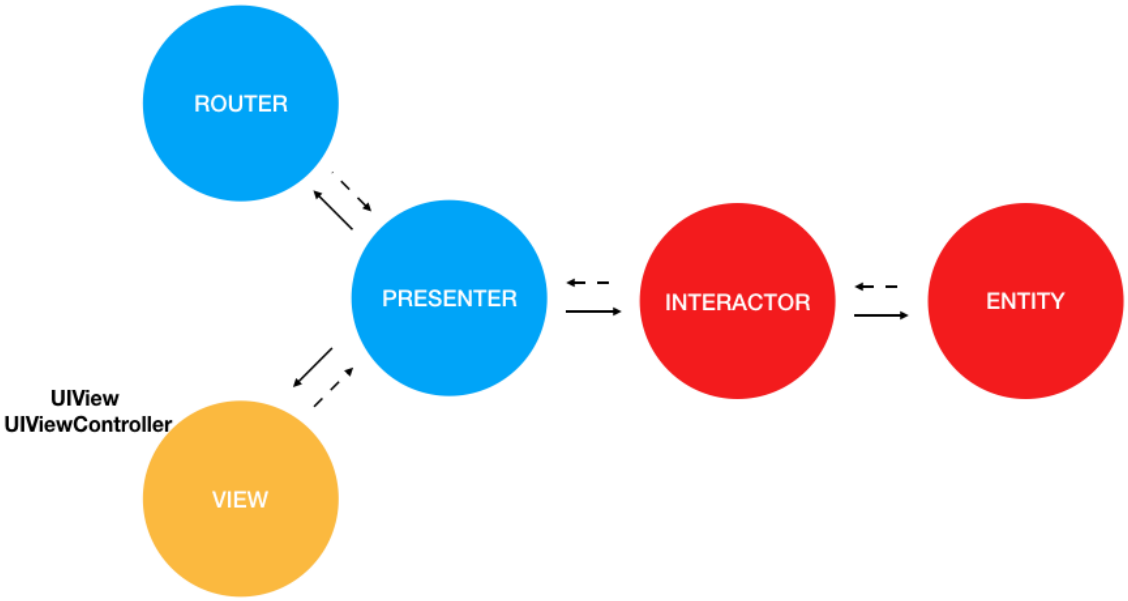


Obrázek 3 - MVP iOS model (Idryshev)

MVP používá k manipulaci s Modelem **Supervising Controller** viz Obrázek 3. Widgety předávají Supervising Controlleru uživatelská gesta. Widgety nejsou rozděleny na View a Controllery. Presentery jsou považovány podobným Controllerům bez počáteční manipulace interakcí uživatele. Důležité je poznamenat, že Presenéry jsou obvykle na úrovni formuláře, nikoli widgetu – to je nejspíš největší rozdíl. (Idryshev)

3.4.8 Architektura aplikace pro iOS VIPER

Komunita se domnívala, že je zapotřebí větší rozklad kódu do více tříd a vynikl VIPER. Kód je tedy rozdělen do **View**, **Presenteru**, nově **Routeru** a **Interactoru** viz Obrázek 4.



Obrázek 4 - VIPER iOS model (Idryshev)

Za krátkou dobu byl VIPER populární, než se přišlo i na jeho chyby. Tento vzor architektury vyžaduje mnoho protokolů, tříd a předávání dat mezi vrstvami. Ve výsledku ale kód není o tolik čitelnější a efektivnější než u architektur předchozích. (Idryshev)

3.5 Vývoj nativní aplikace pro Andriod

3.5.1 Nástroje a náležitosti pro vývoj

Základem vývoje, jako obvykle, je znalost programovacího jazyka pro danou technologii, platformu. Uvádí se, že hlavní jazyky, které by měl vývojář Android aplikací znát jsou Java a značkovací jazyk XML. Mezi základní znalostní prvky Java patří objekty a třídy, balíčky (**packages**), dědění a rozhraní, znakové řetězce, čísla a generika, sbírky (collections) a řízení procesů na více vláknech. (Verma, 2020)

Mezi nejpopulárnější jazyk mezi Android vývojáři se ale řadí **Kotlin**. Programovací jazyk řešící nedostatky jazyku Java. Jedná se o staticky psaný programovací jazyk, který lze právě kombinovat s předem zmíněným jazykem Java pro efektivnější a výkonnější aplikace.

Kotlin má velmi jednoduchou a čistou syntaxi, psaní kódu je tedy rychlejší a efektivnější. Kvůli své jednoduchosti je nejoblíbenějším jazykem pro psaní aplikací pro zařízení na platformě Android. (Agarwal, 2020)

V prostředích pro vývoj (IDE) máme na výběr mezi **Android Studiem** a **Eclipse**, přičemž Android Srtudio je v komunitě používanější. V těchto IDE máme možnost se seznámit a naučit se sadou nástrojů (SDK) **Apache Maven** - nástroj pro správu, řízení a automatizaci sestavení aplikací; **Apache Ant** – nástroj pro automatizaci kompilací, testování a na příklad tvorby výsledných balíčků pro distribuci; **Gradle** – nástroj pro automatizace postavený na zkušenostech z předchozích nástrojů Apache. (Verma, 2020)

Komponenty aplikací jsou základními stavebními kameny vývoje pro Android. Každá z komponent je interakčním vstupem/výstupem pro aplikaci. Ačkoli každá z nich existuje jako vlastní entita a hraje specifickou roli, některé na sobě navzájem záleží. Ne všechny komponenty jsou skutečnými vstupnímy body aplikace.

Komponenty jsou děleny do pěti různých kategorií, z nichž každá slouží odlišnému účelu s odlišným životním cyklem. Ten u každé komponenty definuje, kdy je vztvořena a kdy zaniká. Dělení je následovné:

- „**Activities:** Jedná se o komponentu, která představuje jednu obrazovku s uživatelským rozhraním. Aktivity spolupracují a vytvářejí v aplikaci jednotný *uživatelský zážitek* (**User Experience**). Nicméně každá z nich je nezávislá.
- **Services:** Jedná se o komponentu, která běží na pozadí a provádí činnosti pro vzdálené procesy nebo dlouhodobé operace. Neposkytuje uživatelské rozhraní.
- **Content providers:** Jedná se o komponentu spravující sdílenou bázi dat v aplikaci. Prostřednictvím této komponenty mohou být data ukládána buď v souborovém systému, na webu, v databázi SQLite, dotazována nebo dokonce měněna - pokud to umožňuje poskytovatel obsahu. Komponenta je užitečná pro zápis a čtení dat, která nejsou sdílena a jsou lokální pouze pro danou aplikaci.
- **Broadcast receivers:** Jedná se o komponentu reagující na celosystémová hlášení. Komponenta nezobrazuje uživatelské rozhraní ale může vytvořit upozornění, které je zobrazeno ve stavovém řádku. Uživatel je tak upozorněn na událost nesenou právě touto komponentou. Obecně je to brána k ostatním komponentám odvádějící jen minimum práce.

- **Activating components:** Synchronní záměrná zpráva, která aktivuje 3 ze 4 komponent (tj. services, activities a broadcast receivers). Za běhu aplikace kombinu a spojuje komunikaci jednotlivých komponent ptařícím do dané aplikace nebo ne.“ (Verma, 2020)

Android je rozsáhlý operační systém s velkou škálou používaných verzí a znatelným množstvím typů podporovaných zařízeních. To značí, že vývojář musí myslet a mít přehled o podpoře a údržbě aplikací. To zapříčiňuje další náklady. V aplikaci nemůže být použito libovolné písmo, uživatelské rozhraní musí brát v potaz různorodou velikost obrazovek, různá zařízení mají různá nastavení, možnosti a dostupné senzory.

Vývojář musí rozvrhnout, které služby mají běžet na pozadí a které ne. Rozhoduje se podle typu služby pro nejlepší uživatelský zážitek z aplikace. Na příklad uživatelké rozhraní a interakce z uživatelem musím běžet vždy na popředí a nesmí být blokována jiný procesem, aby uživatel věděl, co se odehrává a naopak na příklad komunikace se serverem, výpočty atd. běží na pozadí, aby zařízení uživatele nebylo zamrzlé po tuto dobu operace. (Verma, 2020)

Zařízení, na kterém jste schopni vzvýjet mobilní aplikace pro Android musí mít operační systém Windows, MacOS nebo libovolnou distribuci Linuxu. Dále open source vývojovou platformu Eclipse, **ADT** (Android Development tool) plugin a **Android SDK**. Dokumentace Google poskztuje podrobnější popis a návod pro tyto nástroje, které jsou k dispozici zdarma. (Verma, 2020)

Pro sestavení aplikace na reálném zařízení je potřeba doinstalování USB ovladačů a připojení zařízeního samotného k zařízení na němž probíhá vývoj. Mobilní zařízení musí mít povolené ladění přes USB. Po správném nastavení se mobilní zařízení zobrazí v nabídce v IDE Android Studio. (Lacko, 2015)

3.5.2 Programové prostředí pro vývoj Android Studio

Android Studio je oficiální IDE pro vývoj pro platformu Android. Pomáhá vývojáři s nejefektivnějším vývoje stabilní aplikace pro jakékoliv zařízení s operačním systémem Android.

IDE založené na **Intellij IDEA** pro nejvyšší možnou rychlost práce:

- Funkce aplikace změň umožňuje okamžité synchronizování sestavené aplikace s právě editovaným kódem bez nutnosti restartu – v některých případech i bez restartování dané aktivity. Tato flexibilita pomáhá a šetří čas při drobných změnách.
- Editor kódu napomáhá psát kvalitnější kód a efektivitu pokročilým, nabízením doplňování kódu, refaktorováním a analýzy kódu. Při psaní kódu Android Studio poskytuje návrhy pro dokončení.
- Emulátor Android instaluje a spouští aplikace rychleji než skutečné zařízení a umožňuje prototyp a testování na různých konfiguracích zařízení Android: telefonech, tabletech, Android Wear a zařízeních Android TV. Můžete také simulovat celou řadu hardwarových funkcí, jako je GPS lokace, síťová latence, pohybové senzory a multidotykový vstup. (Google Developers, 2020)

Při každém zásahu vývojáře do kódu pomáhá Android Studio vytvářet nejlepší možný kód:

- Android Studio obsahuje šablony projektů a kódů, které usnadňují přidávání již zavedených vzorů, jako je *navigation drawer* a *view pager*. Můžete začít čistou šablonou kódu nebo rozpracovaným ukázkovým kódem pro danou komponentu. Import funkčních obrazovek ze zdrojů třetích stran je povolen a podporován.
- Android Studio poskytuje robustní framework statické analýzy a zahrnuje přes 365 různých nástrojů pro kontrolu nad vlákny v celé aplikaci. Dodatečně poskytuje několik nástrojů pro rychlé opravy mimo logické chyby.
- Aplikace Android Studio poskytuje rozsáhlé nástroje, které pomáhají otestovat aplikace pro Android programem **JUnit 4** a funkčním frameworkem pro testování uživatelského rozhraní. Pomocí technologie **Espresso Test Recorder** může být generován testovací UI zaznamenáváním interakcí s aplikací na reálném zařízení nebo v emulátoru. Testy mohou být prováděny na zařízeních, v emulátoru, v průběžném integračním prostředí nebo ve **Firebase Test Lab**. (Google Developers, 2020)

Projektová struktura v Android Studiu se sestaveními aplikací založených na Gradle dodávají flexibilitu pro generování **APK** pro všechna zařízení Android:

- IDE nabízí automatizaci sestavení aplikací, správu závislostí a upravitelné konfigurace sestavení aplikací. Projekt je nakonfigurován tak, že zahrnuje místní a hostované knihovny, definuje varianty sestavení, které obsahují různé kódy a prostředky, a používá různé konfigurace pro zmenšování kódů a pro podepisování aplikací.
- V IDE jsou integrované nástroje pro správu a kontrolu verzí, jako je na příklad **GitHub** a **Subversion**. Tým tedy zůstává stále synchronizován s kódem a jeho verzemi. Open sourcový **Gradle build** umožňuje přizpůsobit sestavení aplikací danému prostředí a spustit jej na kontinuálním integračním serveru jako je **Jenkins**.
- IDE poskytuje jednotné prostředí, ve kterém je možné vytvářet aplikace pro telefony a tablety s Androidem, Android Wear zařízení, Android TV a Android Auto. Moduly strukturovaných kódů umožňují rozdělit projekt do jednotlivých funkcí, které můžete nezávisle sestavit, otestovat a ladit. (Google Developers, 2020)

Android Studio pracuje s informací, že ne všechen kód je čistá Java a že ne všechny služby jsou spuštěny na zařízení uživatele:

- IDE plně podporuje úpravu kódu projektu v **C/C++**, takže v aplikaci zvládne rychle sestavit komponenty **JNI**. IDE poskytuje zvýrazňování syntaxe a refaktoring pro C/C++, a ladicí program založený na **LLDB**. Ten umožňuje ladit kód Java a současně C/C++. Nástroje pro sestavení mohou bez úprav spouštět skripty **CMake** a **ndk-build** a poté přidávat sdílené objekty do aplikace.
- **Firebase Assistant** pomáhá propojit aplikaci s bází dat Firebase a propojit se službami jako jsou Analytics, Authentication, Notifications a další přímo v IDE. Vestavěné nástroje pro platformu **Google Cloud** pomohou integrovat koncové body Google Cloud a projektové moduly pro **Google App Engine**. (Google Developers, 2020)

Android Studio poskytuje **GUI** nástroje pro zjednodušení statisticky méně záživných částí vývoje:

- Při práci se soubory XML poskytuje IDE *drag-and-drop* vizuální editor, který usnadňuje vytvoření nového rozložení. Editor rozvržení byl vytvořen v unisonu s ConstraintLayout API rozhraním, takže vývojář je schopen rychle sestavit

rozvržení, které se přizpůsobuje různým velikostem obrazovky přetažením komponent a následným přidáním omezení rozvržení komponenty.

- Použitím **APK Analyzeru** je vývojář schopen nahlédnout, do kontentu aplikace a redukovat její velikost, zobrazit **DEX** soubory a porovnat rozdíly mezi dvěma aplikacemi (APK).
- IDE umožňuje vytvoření různé velikosti obrázků v assetech pro každý z nich. Pomocí **Vector Asset Studio** může vývojář vybírat z ikon a materiálů poskytovaných Googlem nebo importovat vlastní soubor SVG, PSD. Nástroj Vector Asset Studio může generovat bitmapové soubory pro každé rozlišení obrazovky a podporovat tak starší verze Androidu, jež nepodporují Android formát pro vykreslování vektorů.

Editor překladů poskytuje zobrazení všech vašich přeložených zdrojů na jednom místě, což usnadňuje změnu nebo přidání překladů a hledání těch chybějících bez nutnosti otevření jednotlivých verzí strings.xml. Poskytuje odkaz na jednotlivý překlad. (Google Developers, 2020)

3.5.3 Programovací jazyk Kotlin

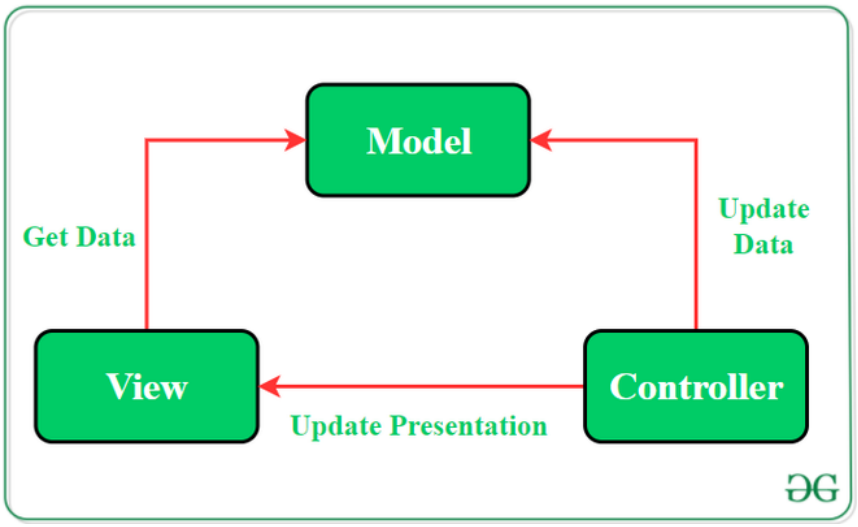
„Kotlin by se dal stručně představit jako modernější Java. V roce 2011 **JetBrains**, firma vyvíjející několik velmi populárních IDE, oznámila nový programovací jazyk pro **JVM**, běhové prostředí Javy. Aplikace v něm vytvořené tedy fungují všude, kde je nainstalována populární Java. Jeden z členů vývojářského týmu nového jazyka zdůvodnil svůj výtvor tím, že tehdejší náhrada za Javu, Scala, trpěla pomalou kompilací, tedy pomalým převodem Scaly do kódu pro Java prostředí, a mizernou podporou pro vývoj Android aplikací. Přestože Kotlin má jinou syntaxi, tak zkompilovaný kód je plně kompatibilní s Javou, proto se v jazyce Kotlin dají i používat Java knihovny. Od roku 2017 Kotlin je oficiální jazyk pro vývoj pro Android a vsadilo na něj hned několik velkých firem včetně Google. Protože se jedná o relativně nový jazyk, tak si s sebou z minulosti nenese nejrůznější problémy a nabízí řadu moderních přístupů.

Po stránce návrhu se jedná o objektový, kompilovaný jazyk s typovou kontrolou. Poměrně blízko tak má k jazykům jako jsou **Swift**, **Scala** a jim podobným. Kotlin je od úplného začátku, kdy firma JetBrains oznámila jeho vývoj, open source a proto aplikace v něm napsané jdou spustit téměř na všech operačních systémech. Kotlin je zpětně kompatibilní. V Kotlinu lze napsat cokoliv, co by bylo psáno v jazyce Java.

Jak již bývá zvykem u modernějších jazyků, jako např. u **Dartu**, Kotlin umožňuje své vlastní zdrojové kódy zkompilovat do jazyka **JavaScript**.“ (Kodytek, 2018)

3.5.4 Architektura aplikace pro Android MVC

Vzor architektury **MVC** (Model-View-Conroller). Model je vrstva pro ukládání dat. Odpovídá za zpracování logiky (real-world business rules), komunikaci s bázemi dat a síťovými vrstvami. **View** je vrstva **UI** (uživatrelského rozhraní). Poskytuje vizualizaci dat z **modelu**. **Controller** je nositelem základní logiky. Ze získaných informací interakce uživatele se zařízením model dle potřeby aktualizuje.

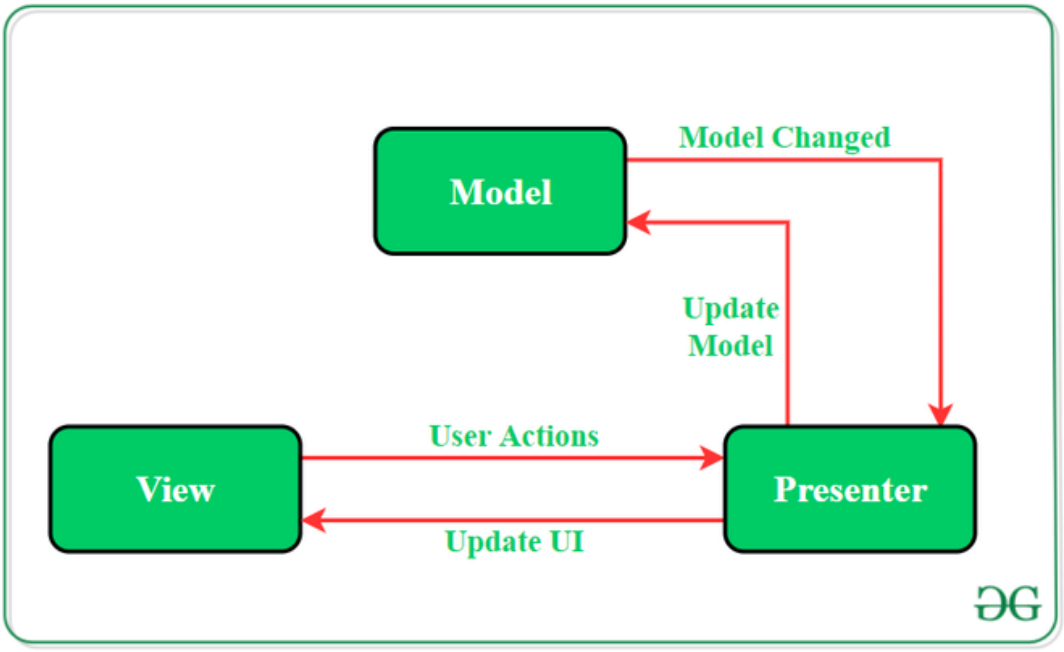


Obrázek 5 - MVC Android model (MISHRA)

Ve vzoru MVC závisí View i Controller na Modelu viz Obrázek 5. Data aplikací jsou aktualizována Controllerem a View odtud dostane data. V tomto vzoru může být Model testován nezávisle na uživatelském rozhraní, protože je oddělen. MVC lze použít mnoha způsooby. Buď mohou aktivity a fragmenty působit jako Controller, který je odpovědný za zpracování údajů a aktualizaci View. Další způsob je použít aktivity a fragmenty jako View a Controller, stejně jako Modely, které jsou samotnou třídou bez nadtřídy. Pokud View respektuje *zásadu jediné odpovědnosti*, pak je jeho úlohou pouze aktualizovat Controller pro každou interakci uživatele se zařízením a pouze zobrazovat data z Modelu. V tomto případě testy uživatelského rozhraní postačují k pokrytí funkcí View. (MISHRA)

3.5.5 Architektura aplikace pro Android MVP

Vzor architektury **MVP** (Model-View-Presenter) je všeobecně akceptovaný a nadcházejícím vývojářům komunitou doporučovaný. Nachází se v něm nová vrstva **Presenter**, která se základě získaných dat z modelu a logiky rozhoduje, co se zobrazí ve **View**. **Model** a View plní stejnou funkci jako ve vzoru architektury MVC. View komunikuje pouze ale s Presenterem.

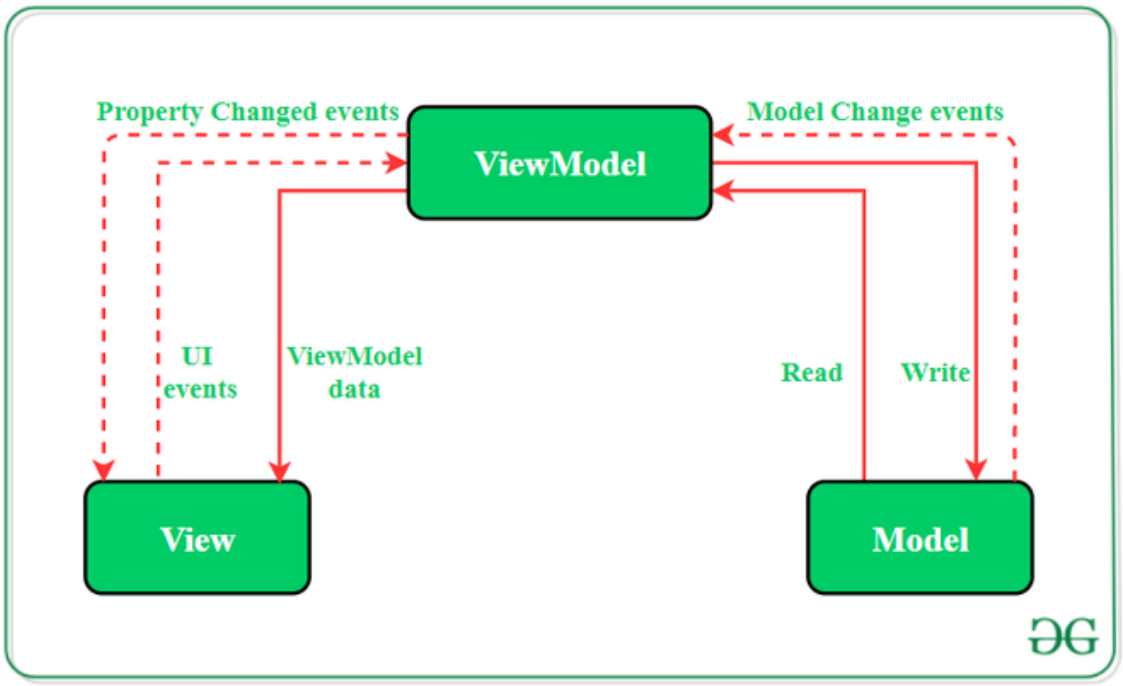


Obrázek 6 - MVP Android model (MISHRA)

Ve vzoru architetury MVP jsou View a Presenter blízce příbuzní s vazbou mezi sebou viz Obrázek 6. Pro čitelnost a srozumitelnost kódu se používá třída **Contract** pro definování vztahu Presenteru k View. View je abstraktní, jeho rozhraním umožňuje Presenteru **Unit Testování**. (MISHRA)

3.5.6 Architektura aplikace pro Android MVVM

Vzor architektury **MVVM** (Model-View-ViewModel) je Google Android týmem doporučovaná architektura. Vrstva Model je zodpovědná za abstrakci datových zdrojů. **Model** a **ViewModel** spolupracují pro práci s daty. Účelem vrstvy **View** je informovat ViewModel o interakci uživatele se zařízením. Vrstva ViewModel pracuje s daty z Modelu a zpřístupňuje je uživateli skrze View.



Obrázek 7 - MVVM Android model (MISHRA)

Ve vzoru MVVM View informuje ViewModel o aktivitách. View má odkaz na ViewModel, zatímco ViewModel nemá žádnou informaci o View. Vztah many-to-one existuje mezi View a ViewModelem. MVVM podporuje obousměrnou výměnu dat mezi objekty viz Obrázek 7. (MISHRA)

3.6 Vývoj cross-platformové aplikace s SDK Flutter

Flutter je bezplatný open-source mobilní **UI framework** vytvořený společností Google, vydaný v květnu 2017. Umožňuje vytvořit nativní mobilní aplikaci pouze s jedním kódem pro více platforem. To znamená, že pro vytvoření dvou různých aplikací (pro iOS a Android) může být použit jeden programovací jazyk a jeden kód.

Flutter je tvořen ze dvou důležitých částí:

- **SDK (Software Development Kit):** kolekce nástrojů, které pomáhají při vývoji aplikace. Zahrnuje nástroje pro kompilaci kódu do kódu nativního pro dané zařízení (iOS a Android).
- **Framework (knihovna UI založená na widgetech):** kolekce opakovaně použitelných prvků uživatelského rozhraní (tlačítka, textové vstupy, posuvníky a další komponenty). (Thomas)

3.6.1 Nástroje a náležitosti pro vývoj

Základním požadavkem pro vývoj aplikací za použití **SDK Flutter** je samotné zařízení, na kterém bude vývoj probíhat. Zařízení může mít operační systém Windows, MacOS, Chrome OS nebo libovolnou distribuci Linuxu. Testování a sestavení aplikace pro iOS lze pouze s operačním systémem MacOS.

Komunitou a samotnými Google vývojáři jsou doporučovány dvě IDE: Android Studio, VS Code. VS Code je IDE od společnosti Microsoft a díky jeho menším nárokům na výkon je vývoj v něm efektivnější a rychlejší. Pokud ale má být aplikace sestavena pro testování v emulátoru, vývojář se instalaci Android Studia nevyhne. V případě iOS, musí být nainstalován emulátor z Xcode.

Na vývojovém zařízení vývojáře musí být nainstalovaný kompilátor programovacího jazyka Dart. Znalost Dartu a **OOP** (Objected Oriented Programming) jsou nedílnou součástí dovedností vývojáře. (Flutter documentation)

3.6.2 Programovací jazyk Dart

Programovací jazyk Dart se využívá v jádru sady nástrojů Flutter. Moderní framework, jako je Flutter, vyžaduje moderní jazyk na vysoké úrovni, aby byl schopen poskytnout vývojáři nejlepší komfort a umožnit efektivní tvorbu mobilních aplikací. Porozumění programovacímu jazyku Dart je zásadní, vývojáři by měli znát původ jazyka, jak na něm komunita pracuje, jak se vyvíjí, jeho silné a slabé stránky a proč je zvolen jako programovací jazyk právě pro Flutter. Jazyk je vyvinutý společností Google a používá se pro vývoj aplikací **desktopových**, **webových**, **serverových** a **mobilních**. Cílem jazyka Dart je nést kladné vlastnosti většiny jazyků na vysoké úrovni. (Biessek, 2019)

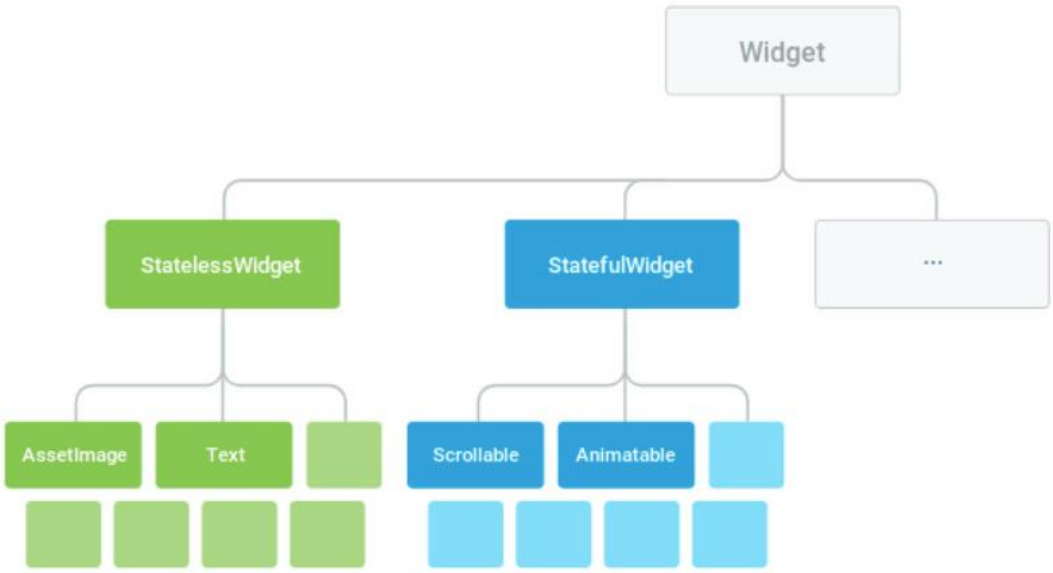
Dart byl vydán v roce 2011, ale jeho stabilní vydání přišlo až 2013, přičemž zásadních změn se dočkal ve verzi 2.0 koncem roku 2018. Koncepčně měl nahradit jazyk JavaScript pro tvorbu webových aplikací. Nyní je však Dart primárně zaměřen na tvorbu mobilních aplikací. Snažil se vyřešit problém s robustností JavaScriptu, byl tedy uveden jako jeho zralý nástupce. (Biessek, 2019)

Kód psaný v jazyce Dart může být spuštěn dvěma způsoby. Za pomoci **Dart VM** virtuálního stroje, nebo JavaScriptovým kompilátorem. Spuštění probíhá v Dart-capable environmentu, které poskytuje Runtime systémy, Dart core knihovny a Garbage sběrače. Spuštění má dva módy, **Just-In-Time (JIT)** a **Ahead-Of-Time (AOT)**. JIT kompilace je místo, kde je zdrojový kód načten a zkompilován do nativního kódu virtuálního stroje Dart VM. Používá se za ke spuštění kódu v rozhraní příkazové řádky. Nebo při vývoji mobilní aplikace, kde se využívá funkcí ladění a opětovného načítání. AOT kompilace je místo, kde Dart VM a kód jsou předkompilovány a funguje spíše jako runtime systém, poskytující metody z Dart SDK aplikaci. (Biessek, 2019)

3.6.3 Flutter jako framework založený na widgetech

Jedná se o framework, jehož veškeré prvky a komponenty jsou děděny ze dvou nadtříd widgetu, které určují, zda bude widget dynamický nebo statický. Komponentami jsou myšleny tlačítka, rozbalovací menu a další interakční prvky, ale také stylistické prvky, jako je na příklad barva písma a funkce rozvržení, na příklad výplň a okraje komponent.

Aplikace založená na widgetech zpracovává každou komponentu jako widget a tím sleduje každý model objektu. Spíše, než aby byl framework závislý na nativních komponentách, dynamicky vytváří vlastní pomocí vlastního vykreslovacího enginu převodem kódu aplikace (v jazyce Dart) do nativního kódu cílového zařízení. (Shah, 2019)



Obrázek 8 - Struktura widgetů ve Flutteru (Shah, 2019)

Framework Flutter se dá kategorizovat dle typu widgetů do dvou kategorií viz Obrázek 8:

- **StatelessWidget**: tento typ widgetu je neměnný, nereaguje tedy na interakci uživatele se zařízením a nemůže být po vykreslení přepsán. To brání ve znovunačtení.
- **StatefulWidget**: proměnlivý, dynamický widget vhodný pro interakci uživatele a zařízení na němž aplikace je spuštěna. (Shah, 2019)

3.6.4 Architektura Frameworku Flutter

Obrázek 9 - Schéma architektury Flutteru (Shah, 2019)

K tvorbě komponent používá Flutter programovací jazyk Dart a grafický engine **Skia 2D**. Kód Frameworku je spouštěn na nízkoúrovňovém virtuálním stroj Dartu. Byť kód ve frameworku lze napsat v jazyce Dart, grafický engine je implementován v **C++**. Dart je zkompilován do nativního kódu pomocí **AoT** (Ahead of Time) kompilace. Kód enginu C/C++ je zkompilován:

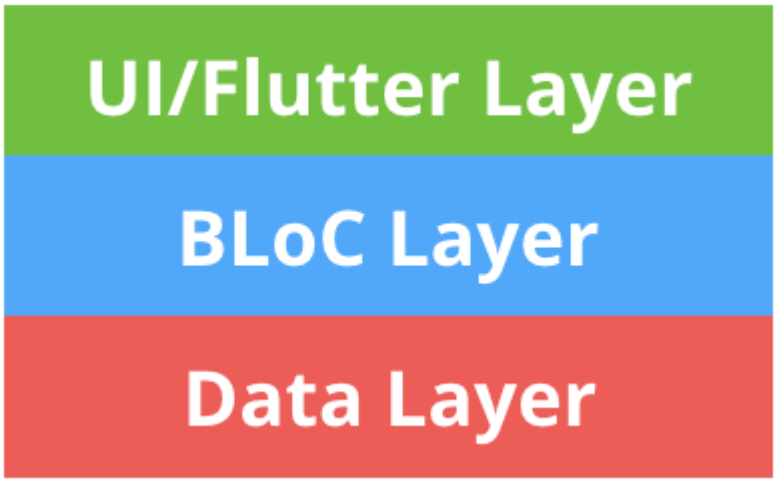
- Androidím **NDK** (Native Development Kit)
- Nízkoúrovňovým virtuálním počítačem (**LLVM**) pro platformu iOS

Framework se skládá z knihovny **Material design**, knihovny widgetů ve stylu iOS nazvané **Cupertino** a dalších knihoven pro animace, gesta atd. Knihovna **Foundation** obsahuje všechny třídy a funkce nejnižší úrovně používané všemi ostatními vrstvami Flutter frameworku. Část Engine se skládá z 2D grafických enginů, především **Skia** a **Dart Virtual Machine** viz Obrázek 9. (Shah, 2019)

3.6.5 Architektura aplikace ve Flutter BLoC

Podstatou vzoru architektury **BLoC** (Business Logic Components) je. Že by vše v aplikaci mělo být prezentováno jako **stream** aktivit, událostí: Widgety předpokládají události; ostatní widgety reagují. BLoC řídí konverzaci z prostřední pozice. Dart přichází se syntaxí pro práci se streamy, které jsou součástí jazyka.

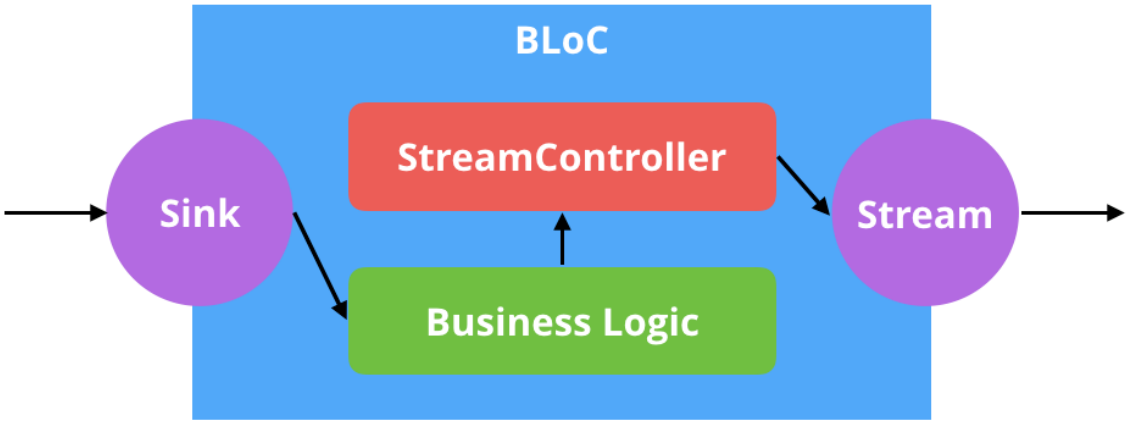
Pro tento vzor architektury není potřeba importu přídavných modulů, Flutter již má vše potřebné.



Obrázek 10 - BLoC Flutter vrstvy (Kayfitz)

Vzor se neliší příliš již od známého vzoru MVP. **UI/Flutter** vrstva může být řízena pouze vrstvou BLoC a jak už nejspíš napovídá její název, stará se v aplikaci o uživatelská rozhraní. BLoC vrstva odesílá události do vrstev UI/Flutter a Data. Zpracovává logickou část aplikace. Vrstva Data je zodpovědná za modely dat a připojení aplikace k serveru. O uživatelském rozhraní neví nic viz Obrázek 10. S rostoucí aplikací se může struktura razantně zvětšovat. (Kayfitz)

Vzor architektury BLoC je ve skutečnosti pouze rozhraní streamů Dartu.



Obrázek 11 - Struktura vzoru BLoC (Kayfitz)

Streamy, jako je na příklad Future (potencionální hodnota dostupná v budoucnu), jsou poskytovány balíčkem *dart:async*. Stream je podobný Future, ale místo navrácení jedné hodnoty asynchronně, stream jich může navrátit více.

Balíček *dart:async* poskytuje objekt zvaný **StreamController**. StreamControllery jsou správci objektů kteří vytvářejí instanci **streamu** a **sinku**. Sink je pravý opak streamu. Pokud stream přináší výstupní hodnoty, sink zase ty vstupní.

BLoC jsou objekty zpracovávající a ukládající *business* logiku. Za pomoci sinků přijímají vstup a za pomoci streamů výstup viz Obrázek 11. (Kayfitz)

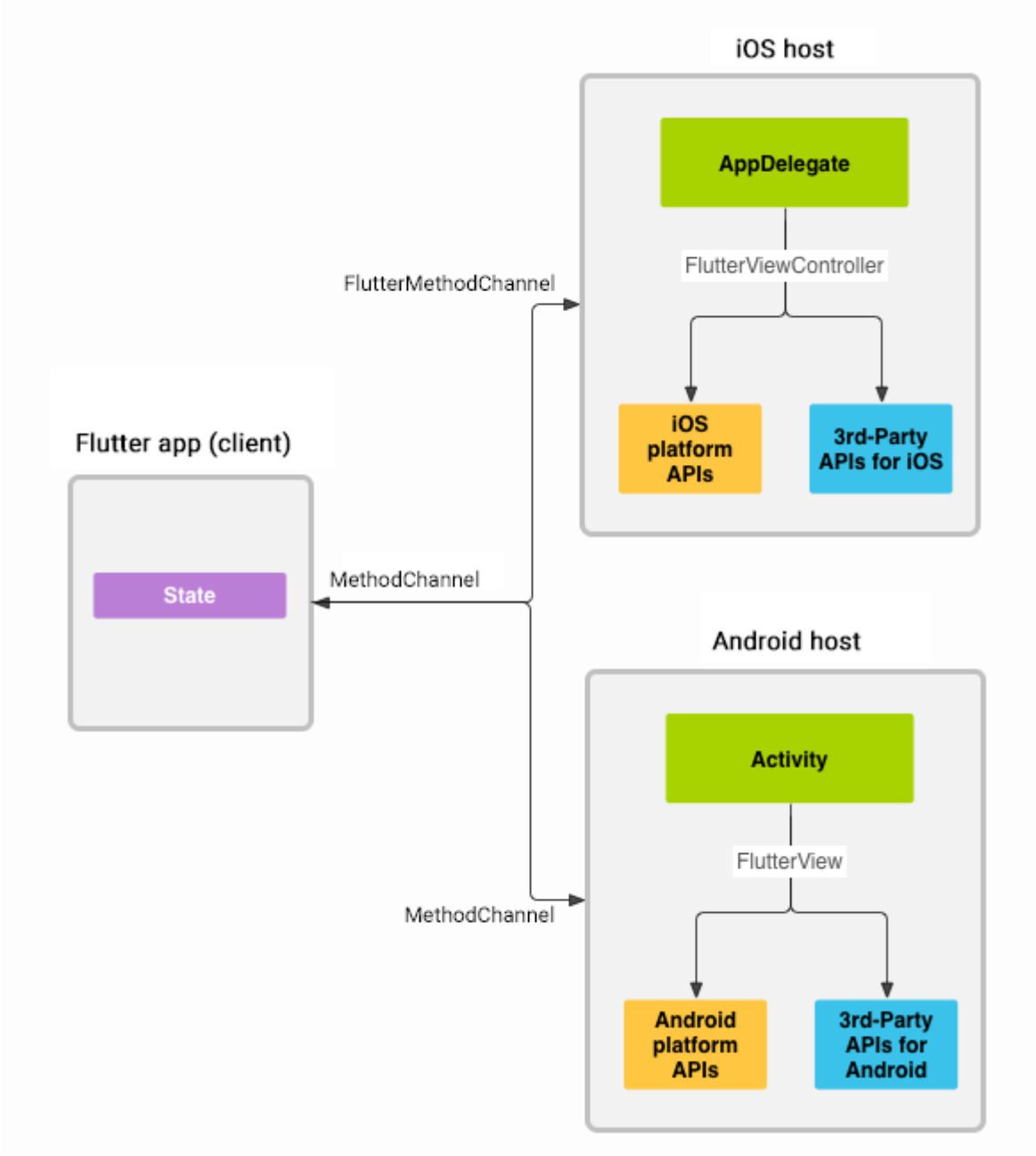
3.6.6 Flutter pluginy

Pluginy Flutter obalují nativní kód **Androidu** (Kotlin, Java) a **iOS** (Swift, Obj C). Je to doplněk pro získaní nativní funkcionality na zařízení.

Podpora API pro zabudovanou platformu Flutter nespoléhá na generování kódu, ale spíše na *pružný* styl předávání zpráv. Alternativně lze balíček **Pigeon** použít pro zasílání strukturovaných zpráv typu *typesafe* prostřednictvím generování kódu viz Obrázek 12:

- Část aplikace ve Flutteru posílá zprávy svému hostiteli, části aplikace v iOS nebo Android, přes platformový kanál.

- Hostitel poslouchá platformy na kanálu a zprávu obdrží. Poté volá do několika pro platformu specifických API, za pomoci nativního programovacího jazyka, a odešle odpověď zpět klientovi, části aplikace ve Flutteru.



Obrázek 12 - Přehled architektury pluginů ve Flutteru (Flutter.dev, 2020)

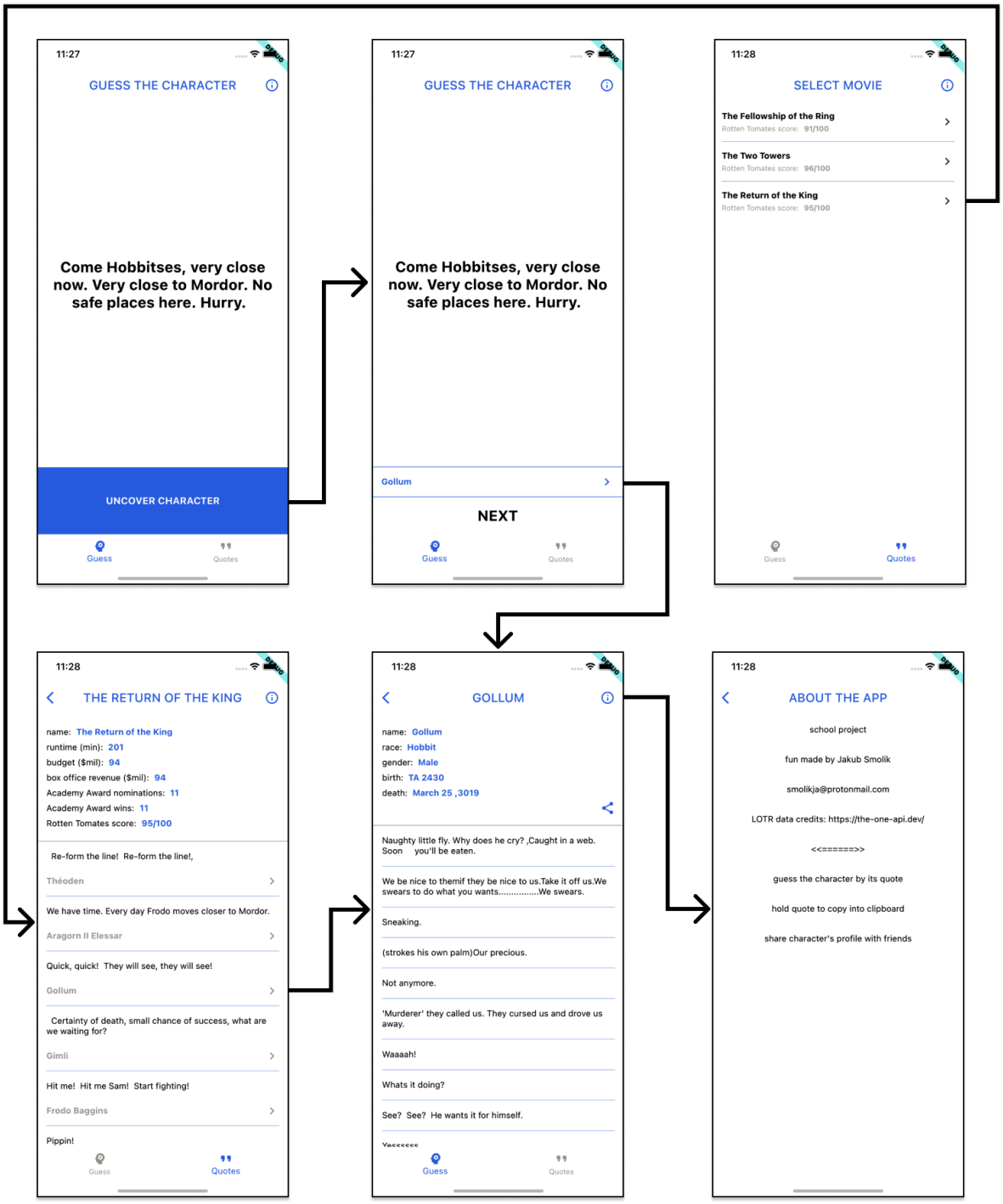
Zprávy jsou předávány mezi klientem (**UI**) a hostitelem (**platformou**) pomocí kanálů platformy asynchronně, aby bylo zajištěno, že uživatelské rozhraní bude reagovat. (Flutter.dev, 2020)

4 Praktická část

Zhodnocení cross-platformového vývoje mobilní aplikace za pomoci nástrojů Flutter představuji za pomoci demonstrativní aplikace mnou nazvané: ***LOTR: Guess the character!*** Tato aplikace ukazuje uživateli určitá data (citace a postavy z trilogie Pán prstenů) zpracována z .json souborů, které jsou obsaženy v aplikaci. Umožňuje uživateli náhled těchto dat dle předem určené selekce nebo gamifikace.

Uživateli je zobrazena citace z trilogie Pán prstenů a on se snaží hádat jméno postavy, jež citaci vyslovila. Uživatel následně stiskem tlačítka odkryje jméno postavy. Stiskem jména se dostává na informace postavy a její citace z trilogie. Uživatel je schopen procházet citace dle jednotlivých děl trilogie zvlášť, přičemž u každé citace je jméno postavy, jež ji vyslovila a uživatel se stiskem jména dostává opět na stránku o postavě a její citace.

Aplikace obsahuje čtyři stránky obrazovky, do kterých se data promítají, a jednu stránku obrazovky informativní viz Obrázek 13. Soubor s daty je z volně dostupného, *open source* (otevřený zdroj) ***API*** (Application Programming Interface) https://the-one-api.dev/.



Obrázek 13 - Wireframe ukázkové aplikace

4.1 Použité technologie pro vývoj

Aplikaci psanou v programovacím jazyce Dart využívající framework Flutter lze psát ve vícero vývojových prostředí. Pro svůj vývoj aplikace jsem zvolil známý editor zdrojového kódu, Visual Studio Code firmy Microsoft z důvodu náročnosti na systém. Pro plné využité

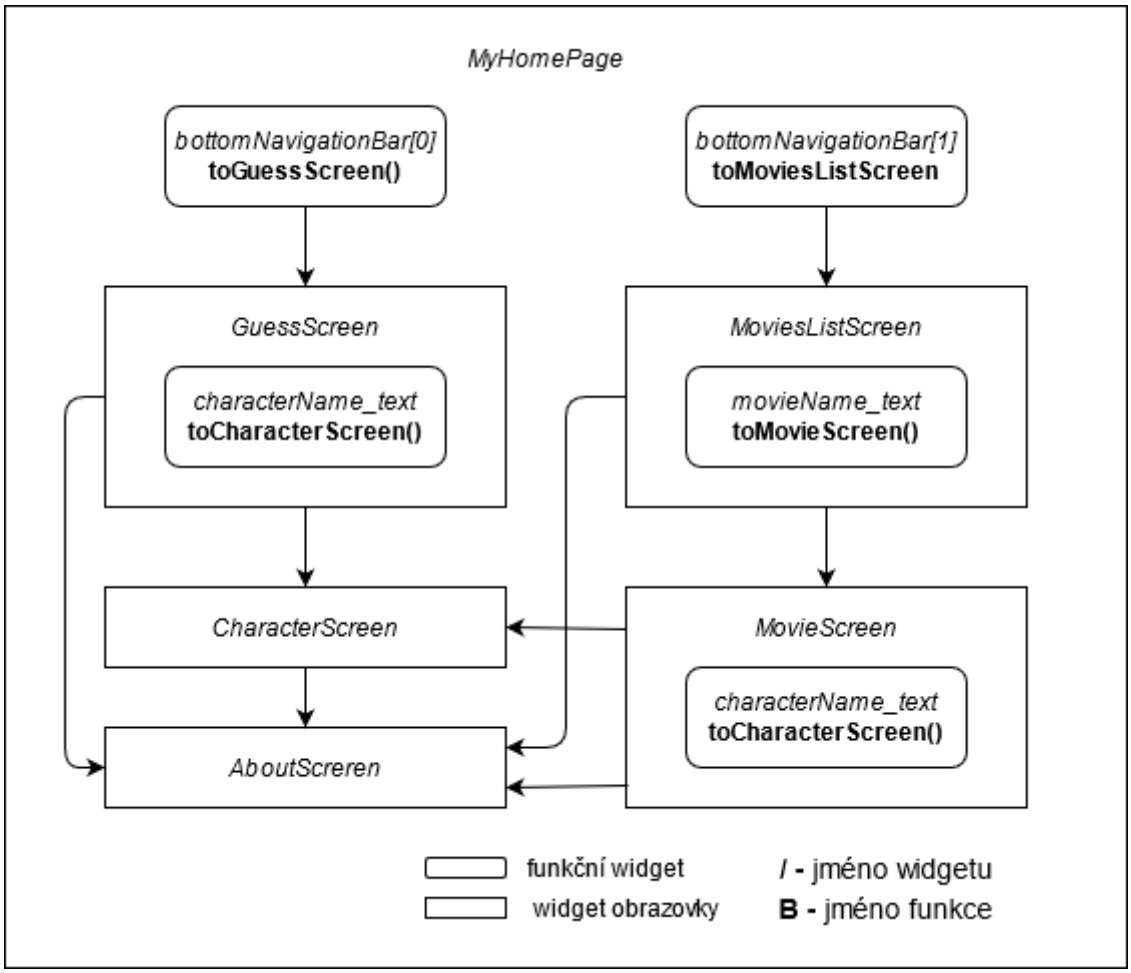
vývojového prostředí je zapotřebí doinstalovat potřebná rozšíření, a to alespoň programovací jazyk Dart (https://dartcode.org/) a framework Flutter.

Samozřejmou nutností je předem stažené Flutter SDK a upravení cesty v konzoli k němu pro možnost používat příkazů SDK Flutter z konzole v operačním systému.

Poslední náležitostí při vývoji mimo operační systém MacOS je nainstalované Android Studio a v mém případě i Android emulátoru (obsažen v Android studiu) pro ladění aplikací bez nutnosti připojení fyzického zařízení k vývojovému.

Pro možnost ladění a sestavování aplikací na platformě iOS je zapotřebí Xcode (volitelně iOS emulátor, který je v Xcode obsažen), tedy vývojové zařízení s operačním systémem MacOS. Při správně nastavené cestě v konzoli k SDK Flutter stačí napsat příkaz *flutter doctor* a nástroj zkontroluje veškeré náležitosti, popřípadě rozepíše problémy a tipy, jak je vyřešit (https://flutter.dev/docs/get-started/install).

4.2 Návrh navigace aplikace



Obrázek 14 - Návrh navigace ukázkové aplikace

Aplikace se celkem skládá z pěti obrazovek, mezi kterými uživatel operuje viz Obrázek 14. Z každé obrazovky lze dotekem tlačítka pravé strany *appBaru* navigovat na obrazovku *AboutScreen*, kromě jí samotné. Obrazovkou zobrazenou po spuštění aplikace je *GuessScreen*, obrazovka první cesty, kde je uživateli zobrazena náhodná citace z trilogie *Pán prstenů* a uživatel se snaží uhodnout postavu, od které citace zazněla. Stiskem tlačítka uživatel odhalí jméno postavy a skrze proklik se naviguje na obrazovku *ScharacterScreen*, kde jsou zobrazeny zajímavosti postavy a její citace z trilogie. Druhá cesta navigace vede skrze stisk tlačítka *bottomNavigationBaru*. Po stisku tlačítka *Quotes* se uživatel dostává na obrazovku *MoviesListScreen*, kde si ze seznamu vybere film z trilogie. Po výběru je přesměrován na obrazovku *MovieScreen*, kde jsou zobrazeny informace o filmu, citace v něm zazněné a odkazy na postavy, jež citaci vyslovili. Stisk jména postavy uživatele přesune na

obrazovku *CharacterScreen*. Skrze šipku zpět v *appBaru*, nebo navigační klávesy mobilního zařízení se uživatel vždy může vrátit na předešlou obrazovku.

4.3 Vytvoření projektu aplikace

Počin vytvoření nového projektu byl proveden přes konzoli spuštěním příkazu *flutter create <jméno_apliakce>* v lokalitě, kde se projekt nachází. Přístup k projektu a vývoji je umožněn přes vývojové prostředí otevřením složky vytvořené Flutter SDK.

Flutter SDK po provedení příkazu vygeneruje defaultní aplikaci používající framework Flutter.

4.4 Nastavení aplikace pro widgety knihovny material

V této kapitole je popsáno, jak nastavit aplikaci pro využívání widgetů z frameworku Flutter. Rootování hlavní obrazovky aplikace probíhá ze souboru *main.dart* v podsložce *lib*. Spuštěním je volána funkce *main() => runApp*. Parametrem je widget *MaterialApp* určující, že aplikace bude využívat widgety z Flutter knihovny *material.dart*.

```
import 'package:flutter/material.dart';

void main() {
  runApp(MaterialApp(
    supportedLocales: [Locale('en', 'US')],
    localizationsDelegates: [
      AppLocalizations.delegate,
      GlobalMaterialLocalizations.delegate,
      GlobalWidgetsLocalizations.delegate,
    ],
    title: 'LOTR: Guess the character!',
    theme: ThemeData(
        primarySwatch: createMaterialColor(kPrimaryColor),
        primaryTextTheme: TextTheme(headline6: TextStyle(
                      color: createMaterialColor(kPrimaryColor))),
        visualDensity: VisualDensity.adaptivePlatformDensity,
        scaffoldBackgroundColor: Colors.black,
        textTheme: TextTheme(
            bodyText1: TextStyle(),
            bodyText2: TextStyle()).apply(bodyColor: Colors.white)),
    home: MyHomePage(),
  ));
}
```

Zdrojový kód 1 - Hlavní metoda spuštění aplikace

Widget *MaterialApp* má vícero parametrů k poskytnutí chtěných funkcí. Jednou z nich je například internacionalizace *Locales*. Umožňuje přepínání jazyků textů v aplikaci dle jazykových sad. Parametrem *theme* nastavujeme globální téma aplikace. Jedná se o nejrůznější vizuální prvky. Parametr *home* nastavuje obrazovku nadcházející po nastavení předchozích náležitostí.

4.4.1 Flutter localizations

V základu projekt není připraven pro internacionalizaci textů aplikace. Vyžaduje se úprava *pubspec.yaml* souboru, přesněji přidání nové dependence:

```
dependencies:
  flutter_localizations:
    sdk: flutter
```

Zdrojový kód 2 - Flutter localizations dependence

Následně je vyžadováno spuštění příkazu *pub get packages* v místě umístění projektu skrze konzoli. Dojde k instalaci balíčku *flutter_localizations*. Vytvořením třídy a souborů typu .json (pro jazykové sady jednotlivě) docílíme možnosti volání jakéhokoliv textu z hodnoty právě podle klíče v souborech pro každou jazykovou sadu. Má aplikace využívá data pouze v anglickém jazyce je tedy zbytečné vytvářet Locales pro jiný jazyk než angličtinu.

4.5 Widget domovské obrazovky

Jedná se o widget *FutureBuilder*, který čeká na dokončení funkce *Future* a dle tohoto inputu vrací určitý widget.

```
import 'package:flutter/material.dart';
import 'package:bp_flutter_app/globals.dart';
import 'package:bp_flutter_app/helpers/json_parse_helper.dart';

FutureBuilder(
      future: _fetchDataFuture,
      builder: (context, snapshot) {
        if (snapshot.connectionState != ConnectionState.done) {
          return Center(
            child: CircularProgressIndicator(),
          );
        }

        globalCharacters = JsonParseHelper().getCharacters(snapshot.data[0]);
        globalMovies = JsonParseHelper().getMovies(snapshot.data[1]);
        globalQuotes = JsonParseHelper().getQuotes(snapshot.data[2]);

        return WillPopScopeWidget();

      },
    )
```

Zdrojový kód 3 - Widget domovské obrazovky

Zdali funkce Future nebude vykonán, uživatel bude obeznámen o čekáním prostřednictvím widgetu *CircularProgressIndicator* (rotující kolečko značící načítání). Po dokončení funkce se její výstupy uloží do globálních proměnných a je načten následující widget.

V případě třídy *HomePage* je funkcí *Future* načtení dat ze souborů .json uložených v *assetech* aplikace. Funkce je volána ihned při inicializaci třídy.

```
import 'package:flutter/material.dart';

@override
  void initState() {
    super.initState();
    _fetchDataFuture = Future.wait([
            _charactersFuture(),
            _moviesFuture(),
            _quotesFuture()
       ]);
  }

  Future<String> _charactersFuture() async {
    return DefaultAssetBundle.of(context).loadString('assets/characters.json');
  }

  Future<String> _moviesFuture() async {
    return DefaultAssetBundle.of(context).loadString('assets/movies.json');
  }

  Future<String> _quotesFuture() async {
    return DefaultAssetBundle.of(context).loadString('assets/quotes.json');
  }
```

Zdrojový kód 4 - Metody *Future* domovské obrazovky

4.5.1 Zpracování dat ze souboru .json

Pro úkony jako je na příklad zpracování dat ze souboru různých datových typů framework Flutter obsahuje nápomocné knihovny. Za pomoci knihovny *dart:convert* jsme schopni převést datový typ *String* .json souboru do instance určitého předem vytvořeného modelu (datového typu).

```
import 'dart:convert';
import 'package:bp_flutter_app/models/movies_model.dart';

MoviesModel getMovies(String response) {
    if (response == null) {
      return null;
    }

    Map<String, dynamic> map = jsonDecode(response);
    return MoviesModel.fromJson(map);
  }
```

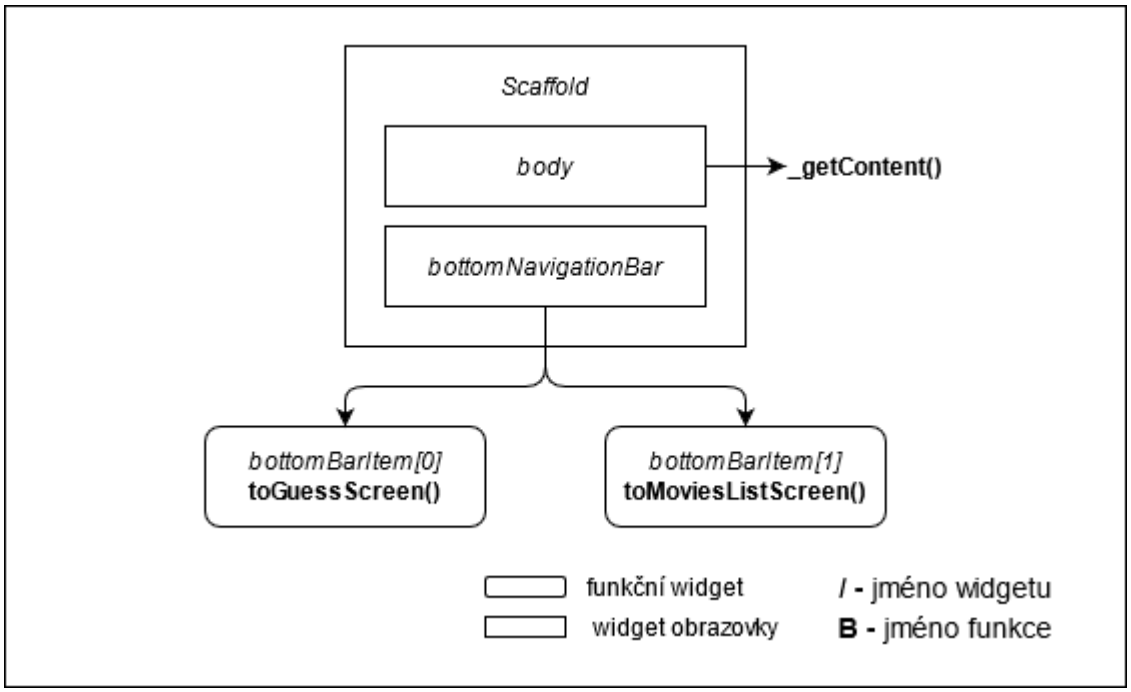
Zdrojový kód 5 - Zpracování dat dle modelů

4.6 Uživatelská navigace v aplikaci

Widget *Scaffold* primárně umožňuje rozvržení obrazovky do třech základních částí: *appBar* – bar na vrchu aplikace, *body* – tělo aplikace, tedy střední část a *bottomNavigationBar* – bar na spodku aplikace. Žádná z těchto komponent nemusí být nutně ve *Scaffoldu* použita.

V aplikaci je použito více *Scaffold*ů pod sebou, pro vytvoření uživatelské navigace takové, kde se uživatel pohybuje za pomoci *appBaru* nezávisle na *bottomNavigationBaru* a naopak viz Obrázek 15.

Nejdříve je inicializován *Scaffold* s *bottomBarNavigation*, který umožňuje přepínání mezi dvěma typy navigace pro appBar. Je zde použito souboru *constants.dart*, kde jsou uloženy globální konstanty pro projekt.



Obrázek 15 - Rozvržení widgetu *Scaffold* pro ukázkovou aplikaci

```
import 'package:flutter/material.dart';
import 'package:bp_flutter_app/helpers/constants.dart';

Scaffold(
     body: SafeArea(
        top: false,
        child: _getContent(),
       ),
     bottomNavigationBar: BottomNavigationBar(
        backgroundColor: Colors.black,
        unselectedItemColor: kGreyDarkColor,
        type: BottomNavigationBarType.fixed,
        currentIndex: _currentIndex,
        onTap: (int index) {
                _navigatorKeys[_currentIndex].currentState.popUntil((route) =>
                     route.isFirst);
                setState(() {
                  _currentIndex = index;
                });
              },
        items: _bottomBarItems.map((_BottomBarItem item) {
           return BottomNavigationBarItem(
              icon: item.icon,
              activeIcon: item.iconActive,
              label: AppLocalizations.of(context).translate(item.titleKey),
                );
              }).toList(),
          ),
        )
```

Zdrojový kód 6 - Uživatelská navigace *bottomNavigationBar*

Jakožto *body* představené instance *Scaffold*u je vždy widget typu *Scaffold* s *appBar* navigací patřící danému tlačítku na *bottomNavigationBar*u. Vše je indexováno.

AppBar je tedy součástí podobrazovek. V celé aplikaci je používán vlastní *appBar* umožňující měnit svoji velikost, titulek a schování tlačítka pro obrazovku *AboutScreen*.

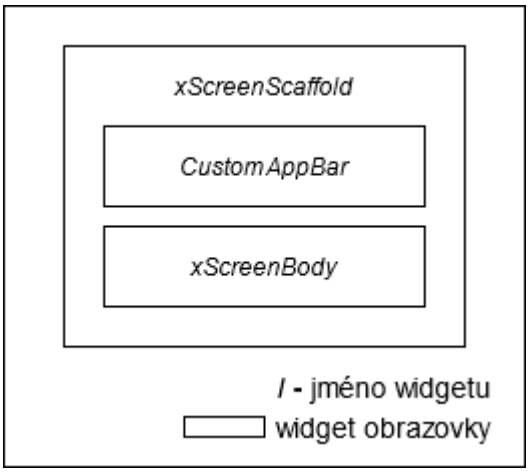
```dart
import 'package:flutter/material.dart';
import 'package:bp_flutter_app/screens/base_stateless_widget.dart';

AppBar(
    iconTheme: IconThemeData(
      color: Theme.of(context).primaryColor,
    ),
    title: Text(title.toUpperCase()),
     centerTitle: true
     backgroundColor: Colors.black
     brightness: Brightness.dark,
     actions: [
      if (!_hideAbout)
        InkWell(
          onTap: () {
            fullscreenPush(AboutScreen(fullscreenPush: fullscreenPush));
          },
          child: Padding(
            padding: EdgeInsets.symmetric(horizontal: 16.0),
            child: Icon(
              Icons.info_outline_rounded,
              color: Theme.of(context).primaryColor,
            ),
          ),
        ),
    ],
  )
```

Zdrojový kód 7 - Uživatelská navigace *appBar*

4.7 Obrazovky aplikace

Veškeré obrazovky aplikace jsou tvořeny *Scaffold*em viz Obrázek 16, kde parametru *appBar* dáváme vytvořený *CustomAppBar* a parametru *body* tělo dané obrazovky.



Obrázek 16 - Rozvržení každé obrazovky ukázkové aplikace

4.7.1 Použití widgetu *StreamBuilder*

Pro obrazovky, kde se na úkor interakce uživatele s aplikací mění uživatelské rozhraní se pro rodiče widgetů těla aplikace používá *StreamBuilder*. Odvádí logické opera mimo třídu s uživatelským rozhraním, chod aplikace je tedy plynulý a bezproblémový. Jedná se o použití architektury BLoC, která je velmi doporučovaná u rozsáhlejších projektů kvůli práci s pamětí a vláknům procesoru.

```
import 'package:flutter/material.dart';
import 'package:bp_flutter_app/bloc/guess_screen_bloc.dart';

StreamBuilder<Object>(
          stream: _bloc.guessScreen,
          initialData: true,
          builder: (context, blocSnapshot) {
            _covered = blocSnapshot.data;
            return ChildWidget();
              },
            );
          })
```

Zdrojový kód 8 - Konfigurace widgetu *StreamBuilder*

Proměnná *_bloc* obsahuje instanci pomocné třídy BLoC ovladače pro danou obrazovku. Parametru *stream* tak podáváme vstup do streamu ovladače.

Po interakci uživatele je zavolána funkce s určitým eventem pro vykonání. Po vykonání dojde k opětovnému renderu všech widgetů pod *StreamBuilderem* s novými parametry.

```
import 'package:flutter/material.dart';
import 'package:bp_flutter_app/bloc/guess_screen_bloc.dart';

GestureDetector(
           child: Widget(),
           onTap: () {
                     _bloc.guessScreenEventSink.add(UncoverEvent());
                     },
               )
```

Zdrojový kód 9 - Komunikace s BLoC vrstvou z uživatelského rozhraní

4.7.2 Třída vrstvy architektury BLoC

Metody pro správnou tvorbu architektury BLoC nám poskytuje Flutter knihovna *dart:async*. V případě následující ukázky se jedná o BLoC třídu patřící k obrazovce *GuessScreen*.

```
import 'dart:async';
import 'package:bp_flutter_app/events/guess_screen_event.dart';

class GuessSrceenBloc {
  bool _isCovered;

  final _guessSrceenStateController = StreamController<bool>();
  StreamSink<bool> get _inGuessScreen => _guessSrceenStateController.sink;

  Stream<bool> get guessScreen => _guessSrceenStateController.stream;

  final _guessSrceenEventController = StreamController<GuessScreenEvent>();

  Sink<GuessScreenEvent> get guessScreenEventSink =>
                                _guessSrceenEventController.sink;

  GuessSrceenBloc() {
    _guessSrceenEventController.stream.listen(_mapEventToState);
  }

  void _mapEventToState(GuessScreenEvent event) {
    _isCovered = (event is CoverEvent) ? true : false;
    _inGuessScreen.add(_isCovered);
  }

  void dispose() {
    _guessSrceenStateController.close();
    _guessSrceenEventController.close();
  }
}
```

Zdrojový kód 10 - Třída vrstvy BLoC

Public proměnná *guessScreen* datového typu *Stream<bool>* zprostředkovává stream. Proměnná *guessScreenSink* zprostředkovává sink pro signalizaci interakce. Privátní metoda *_mapEventToState* změní logickou hodnotu privátní proměnné *_isCovered*.

4.7.3 Tvorba obrazovek z widgetů

Widgety umožňují mít potomky. Většina obrazovek je skládána z widgetu *SingleChildScrollView()* který umožňuje nekonečně na výšku velký obsah s tím, že se v něm dá listovat. Tento widget má pak většinou potomka *Column*, tedy sloupec, jehož potomci budou řazeni vertikálně pod sebe. Jakýkoliv UI widget může být zabalen na příklad widgetem *Padding*, který obstará odsazení dle uvážení, nebo *Center*, který vycentruje své potomky vertikálně i horizontálně.

```
import 'package:flutter/material.dart';

SingleChildScrollView(
          child: Column(
          children: [_getTextWidgets(_rows)],
        ))
```

Zdrojový kód 11 - Konfigurace widgetu *SingleChildScrollView*

4.7.4 Tvorba seznamu

Pro tvorbu seznamu má knihovna *material.dart* widget jménem *ListView*. Ten má vícero metod pro jeho tvorbu, avšak v ukázkové aplikace se nejvíce používá metoda *separated*.

```
import 'package:flutter/material.dart';
import 'package:bp_flutter_app/widgets/character_list_tile.dart';
import 'package:bp_flutter_app/widgets/list_divider.dart';

ListView.separated(
         physics: NeverScrollableScrollPhysics(),
         shrinkWrap: true,
         separatorBuilder: (context, index) => ListDivider(indent: 16.0),
         itemCount: _quoteData.length,
         itemBuilder: (context, index) {
                if (_quoteData[index].dialog.isNotEmpty) {
                    return Column(
                             mainAxisAlignment: MainAxisAlignment.start,
                             children: [
                                 Container(
                                   width: MediaQuery.of(context).size.width,
                                   child: Padding(
                                   padding: const EdgeInsets.only(
                                      left: 16.0,
                                      right: 16.0,
                                      top: 16.0,
                                      bottom: 8.0),
                                   child: GestureDetector(
                                        child: Text(_quoteData[index].dialog),
                                       onLongPress: () {},
                                    ),
                                  ),
                                ),
                                 CharacterListTile(
                                      fullscreenPush: widget.fullscreenPush,
                                      characterId: _quoteData[index].character)
                               ],
                            );
                          } else {
                            return null;
                          }
                        },
                    )
```

Zdrojový kód 12 - Konfigurace widgetu *ListView*

4.7.5 Zakomponování pluginu třetí strany

Flutter jakožto cross-platformový framework nemá možnost využít všechny funkce platforem a jejich zařízeních. Jako řešení se využívají pluginy psány v nativním jazyce pro danou platformu. V případě ukázkové aplikace byl použit plugin *share* (https://pub.dev/packages/share).

Pro jeho instalaci musí být poupraven soubor *pubspec.yaml*, kde se připíše dependence a její verze.

```
dependencies:
  share: ^2.0.0
```

Zdrojový kód 13 - *share* dependence

Příkazem v konzoli s lokací složky projektu *flutter pub get* dojde k instalaci balíčku. Pro použití je potřeba balíček do souboru importovat.

```
import 'package:share/share.dart';
```

Zdrojový kód 14 - Vložení knihovny *share.dart* do souboru

4.8 Konfigurace cross-platformové aplikace

Konfiguraci aplikace, jako je název, ikona, verzování, je nutno vykonat pro každou platformu zvlášť.

Úkony pro konfiguraci pro Android se provedou v podsložkách *android* v souboru *AndroidManifest.xml*. Úkony pro konfiguraci pro iOS v podsložkách *ios* v souboru *info.plist*. Obrázky ikonek aplikace musí mít požadované parametry a být též nahrány do již zmíněných podsložek. Nejjednodušší cesta těchto úkonů je vykonání pro Android v Android Studiu a pro iOS v Xcode s pro to určenými funkcionalitami.

5 Výsledky a diskuse

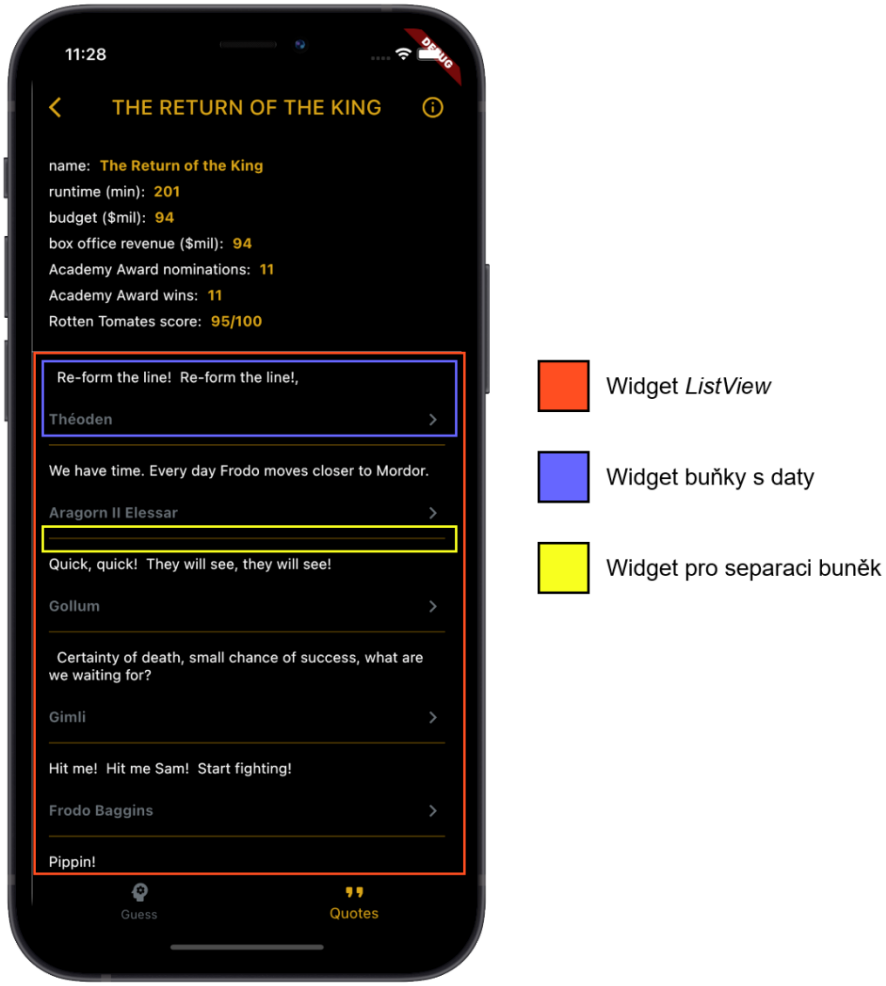
Analýza cross-platformového SDK Flutter, charakterizování problematiky vývoje mobilní aplikace pro platformy iOS a Android jednotlivě a analýza vývoje za využití SDK Flutter byla čerpána z odborné literatury v části teoretické.

Znalosti z části teoretické, byly uplatněny při vývoji ukázkové aplikace *LOTR: Guess the character!* na které byl demonstrován vývoj cross-platformové aplikace s využitím SDK Flutter.

Bylo zvoleno vývojové prostředí pro vývoj aplikace (Visual Studio Code), doinstalovány rozšíření prostředí pro možnost vývoje v programovacím jazyce Dart a využití SDK Flutter. Do zařízení, na němž vývoj aplikace probíhal bylo nainstalováno SDK Flutter a v konzoli byla natavena cesta pro dostupnost jeho nástrojů.

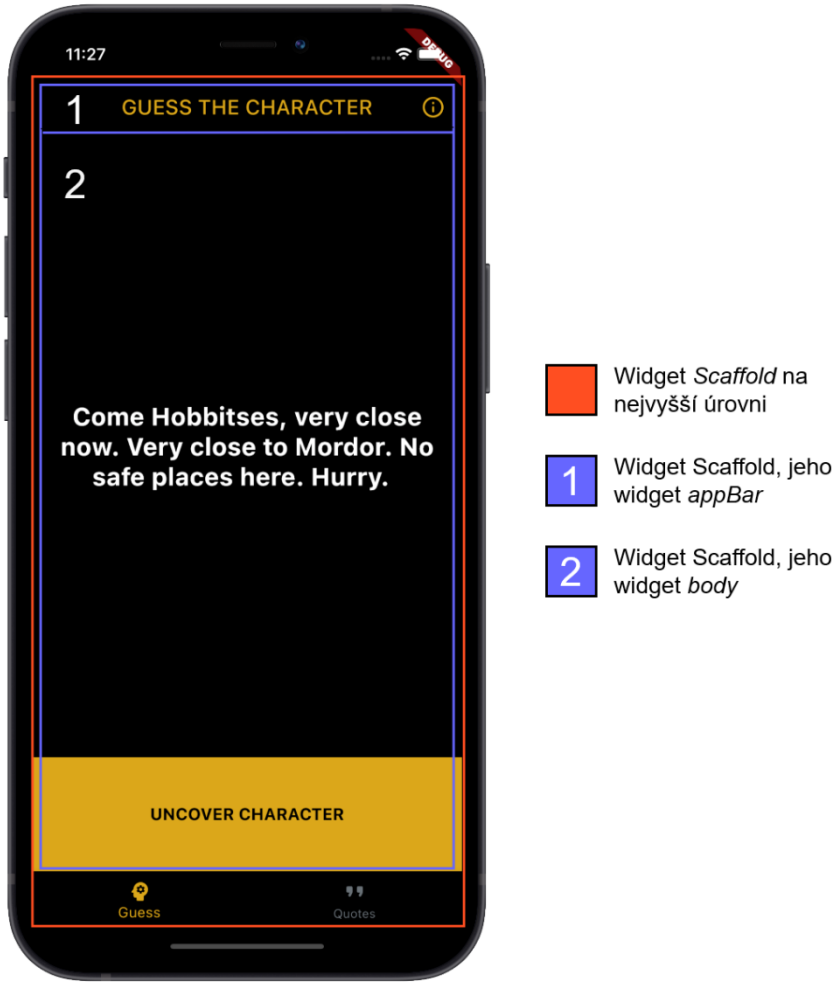
Kód aplikace byl psán pouze v jazyce Dart. Aplikace pro obě platformy mají společný kód. Bylo využito knihovny *material.dart*, která je poskytována SDK Flutter, pro komponenty, z nichž je aplikace sestavena.

Komponenty jsou tvořeny widgety, které mají potomky další widgety. Ku příkladu pro vytvoření listu obsahujícího data, je vytvořen widget *ListView*, jednotlivé buňky jsou widgety jakéhokoliv typu, ale pro doporučován je widget *ListTile*. Buňky jsou odděleny widgetem knihovnou pro *ListView* daným, nebo opět kterýmkoliv na míru vytvořeným widgetem viz Obrázek 17.



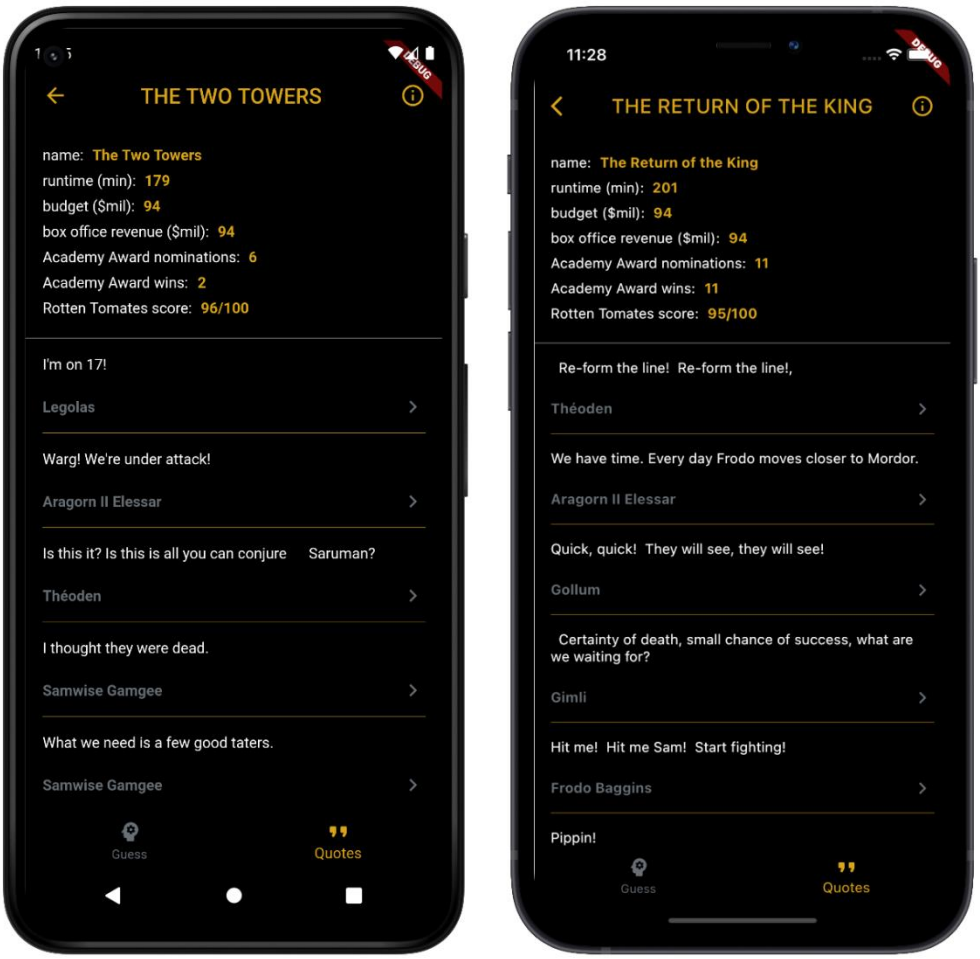
Obrázek 17 - Zhodnocení widgetu *ListView*

Ostatní komponenty a celé obrazovky jsou tvořeny podobným vnořováním widgetů navzájem. Widgetem na nejvyšší úrovní je *Scaffold* obsahující *bottomNavigationBar* (navigační tlačítka na spodku obrazovky). Ve zmíněném widgetu *Scaffold* na nejvyšší úrovni je potomek *Scaffold* obsahující navigační *appBar* umístěn na samotném vrchu obrazovky a tělo obrazovky nacházející se mezi widgety *appBar* a *bottomNavigationBar* viz Obrázek 18.



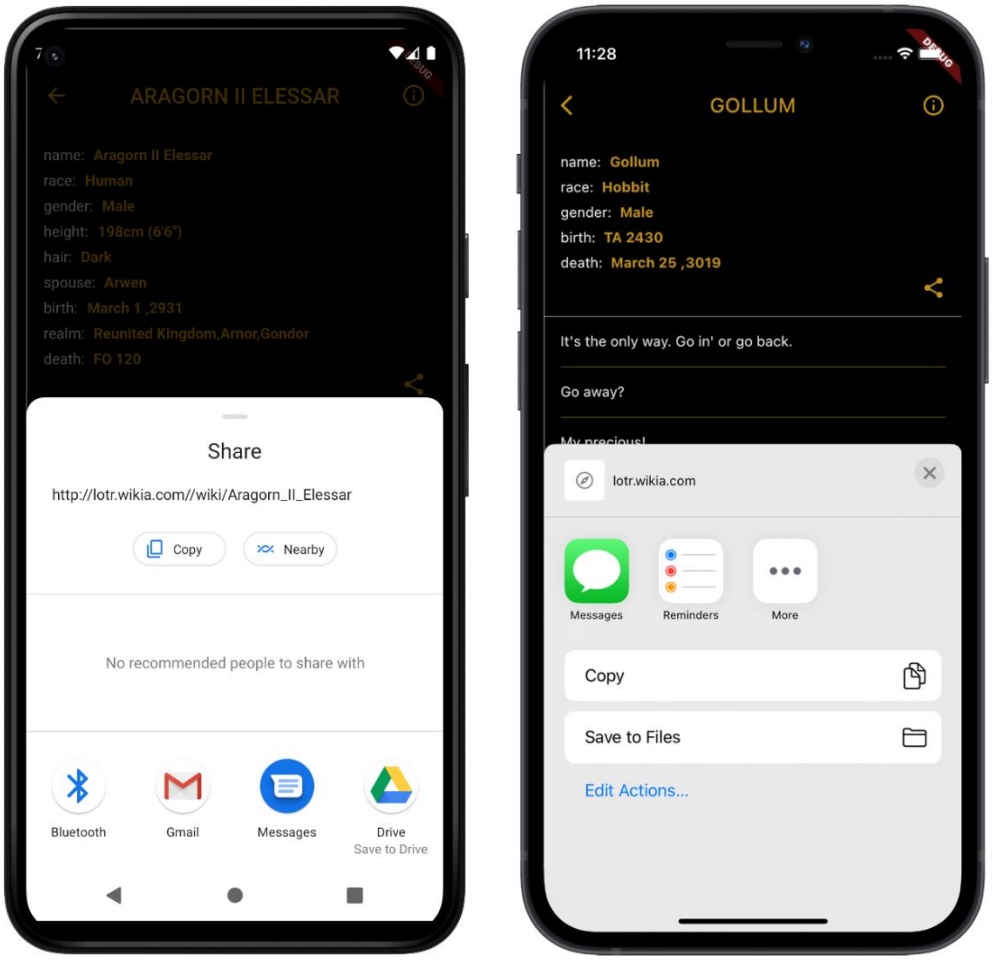
Obrázek 18 - Zhodnocení widgetu *Scaffold*

Komponenty knihovny *material.dart*, jež je SDK Flutter poskytována, z vizuálního hlediska vypadají, i se chovají na každé platformě identicky vůči sobě navzájem viz Obrázek 19**Error! Reference source not found.**. Jsou tedy kompatibilní a nebyl objevena žádná chyby při zobrazování, nebo používání komponent v aplikaci použitých. Byla odzkoušena funkce dlouhého stisku, kdy při dlouhým stisku citace se text citace zkopíruje do paměti zařízení a je možné jej vkládat v rámci zařízení. Komponenty nejsou v každém zařízení zvlášť překládány do nativních komponent, jedná se tedy o vlastní komponenty knihovny, které jsou renderování vlastním grafickým strojem SDK Flutter.



Obrázek 19 - Náhled widgetů obrazovky *MovieScreen*; Android vlevo, iOS vpravo

Při použití pluginů, zde se uživatelské rozhraní a funkcionalita liší dle platformy. Při použití pluginů je jejich UI a funkcionalita identická k aplikacím nativním, oboje je psáno jazykem pro platformu nativním – viz Obrázek 20. Nastavení konfigurace aplikace a sestavování verzí probíhá pro každou platformu samostatně.



Obrázek 20 - Náhled obrazovky po využití pluginu *share*; Android vlevo, iOS vpravo

6 Závěr

Hlavním cílem bakalářské práce bylo analyzovat SDK Flutter pro vývoj cross-platformové mobilní aplikace spustitelné na platformě iOS a Android. Dalšími cíli byly charakterizování problematiky vývoje aplikací pro iOS a Android zvlášť, analyzování vývoje mobilní aplikace za pomoci cross-platformového SDK Flutter a na základě poznatků zhodnotit kompatibilitu komponent uživatelského rozhraní vyvinutého s SDK Flutter na iOS a Android.

V teoretické části bakalářské práce byla charakterizována problematika vývoje mobilní aplikace pro platformy iOS a Android zvlášť. Bylo analyzováno SDK Flutter pro vývoj cross-platformové mobilní aplikace spustitelné na platformě iOS a Android. Část praktická vychází z poznatků části teoretické. Byla zde navržena a vyvinuta zkušební aplikace zpracovávající interní data pro zobrazení jich uživateli na bázi gamifikace a procházení jimi. Pro vývoj bylo vybráno vývojové prostředí a byly doinstalovány doplňky vývojového prostředí. SDK Flutter bylo nainstalováno do zařízení, na němž vývoj aplikace probíhal. Aplikace byla vyvinuta za použití SDK Flutter, byla psána v programovacím jazyce Dart. Pro obě platformy, tedy iOS a Android, má aplikace pouze jednu kódovou základnu a využívá pro obě platformy stejné komponenty, widgety. Data zpracována aplikací slouží právě jako obsah komponent, které se od sebe mezi platformami z vizuálního ani funkčního hlediska neliší. Widgety a funkce, které tuto vlastnost nemají se v SDK Flutter nenacházejí a jsou zabaleny jako pluginy. Pluginy jsou vyvíjeny třetí stranou a jsou psány nativním kódem pro každou platformu.

7 Seznam použitých zdrojů

AGARWAL, Devansh, 2020. *10 Best Mobile Development Programming Languages in 2020* [online]. **2020** [cit. 2020-12-12]. Dostupné z: https://medium.com/front-end-weekly/10-best-mobile-development-programming-languages-in-2020-77439f9b10c1

Agile Software Architecture: Aligning Agile Processes and Software Architectures [online], 2013. Elsevier Science & Technology [cit. 2021-03-15]. ISBN 9780124078857.

App Store Review Guidelines [online], 2020. Apple Inc. [cit. 2020-12-06]. Dostupné z: https://developer.apple.com/app-store/review/guidelines/

BASS, Len, Paul CLEMENTS a Rick KAZMAN, 2003. *Software architecture in practice*. 2nd ed. Boston: Addison-Wesley. ISBN 0-321-15495-9.

BIESSEK, Alessandro, 2019. *Flutter for Beginners: An Introductory Guide to Building Cross-Platform Mobile Applications with Flutter and Dart 2* [online]. [cit. 2020-11-23]. ISBN 9781788990523.

Flutter documentation [online]. [cit. 2020-11-23]. Dostupné z: https://flutter.dev/docs

Flutter.dev: Writing custom platform-specific code [online], 2020. [cit. 2020-12-15]. Dostupné z: https://flutter.dev/docs/development/platform-integration/platform-channels

Google Developers: Android Studio [online], 2020. [cit. 2020-12-15]. Dostupné z: https://developer.android.com/studio/

IDRYSHEV, Amirzhan. *The only viable iOS architecture* [online]. [cit. 2020-12-06]. Dostupné z: https://medium.com/flawless-app-stories/the-only-viable-ios-architecture-c42f7b4c845d

Introducing Xcode 12 [online], 2020. Apple Inc. [cit. 2020-12-06]. Dostupné z: https://developer.apple.com/xcode/

KAYFITZ, Brian. *Getting Started with the BLoC Pattern* [online]. **2019** [cit. 2020-12-15]. Dostupné z: https://www.raywenderlich.com/4074597-getting-started-with-the-bloc-pattern

KLUBNIKIN, Andrei, 2017. *Cross-platform vs Native Mobile App Development: Choosing the Right Development Tools for Your Project* [online]. [cit. 2021-03-12]. Dostupné z:

https://medium.com/all-technology-feeds/cross-platform-vs-native-mobile-app-development-choosing-the-right-dev-tools-for-your-app-project-47d0abafee81

KODYTEK, Samuel, 2018. *Lekce 1 - Úvod do jazyka Kotlin, platformy a IntelliJ* [online]. [cit. 2020-12-13]. Dostupné z: https://www.itnetwork.cz/kotlin/zaklady/uvod-do-jazyka-kotlin-platformy-a-intellij

LACKO, Ľuboslav, 2015. *Vývoj aplikací pro Android*. 1. vyd. Brno: Computer Press. ISBN 978-80-251-4347-6.

LACKO, Ľuboslav, 2018. *Vývoj aplikací pro iOS*. 1. vydání. Přeložil Martin HERODEK. Brno: Computer Press. ISBN 978-80-251-4942-3.

MISHRA, RISHU. Android Architecture Patterns. *GeeksforGeeks* [online]. **2020** [cit. 2020-12-15]. Dostupné z: https://www.geeksforgeeks.org/android-architecture-patterns/

SHAH, Kewal, Harsh SINHA a Payal MISHRA, 2019. Analysis of Cross-Platform Mobile App Development Tools. In: *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)* [online]. IEEE, s. 1-7 [cit. 2020-11-23]. ISBN 978-1-5386-8075-9. Dostupné z: doi:10.1109/I2CT45611.2019.9033872

StatCounter: Mobile Operating System Market Share Worldwide. *StatCounter* [online]. [cit. 2020-12-09]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide

TECHNOCRATS, App, 2019. What are the Key benefits of Native Mobile Apps?. *Elite mCommerce* [online]. [cit. 2021-01-05]. Dostupné z: https://www.elitemcommerce.com/blog/2019/10/22/benefits-of-native-mobile-apps/

THOMAS, Gaël. *What is Flutter and Why You Should Learn it in 2020* [online]. **2019** [cit. 2020-12-15]. Dostupné z: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/

VERMA, Eshna, 2020. *5 Android App Development Fundamentals for Beginners* [online]. **2020** [cit. 2020-12-12]. Dostupné z: https://www.simplilearn.com/android-app-development-fundamentals-article

Xcode IDE [online], 2020. Apple Inc. [cit. 2020-12-06]. Dostupné z: https://developer.apple.com/xcode/features/ Chybovat je lidské.

8 Přílohy

Zdrojový kód ukázkové aplikace *LOTR: Guess the character!* je dostupný v přiloženým *.zip* souboru a na přiloženém CD.