Mobilní aplikace demonstrující doporučené postupy a návrhové vzory frameworku Flutter

David Sedlář

Bakalářská práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav informatiky a umělé inteligence

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**David Sedlář**
Osobní číslo:	**A17147**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Mobilní aplikace demonstrující doporučené postupy a návrhové vzory frameworku Flutter**
Téma práce anglicky:	**Illustrating Flutter Design Patterns and Good Practices Using a Mobile Application**

Zásady pro vypracování

1. Vypracujte přehled aktuální stav problematiky tvorby nativních a multiplatformních aplikací.
2. Popište framework Flutter a související technologie.
3. Navrhněte ukázkovou aplikaci s využitím frameworku Flutter.
4. Vytvořte příklady a vzorová řešení demonstrující klíčové prvky typických mobilních aplikací.
5. Demonstrujte výsledky svého návrhu.

Rozsah bakalářské práce:
Rozsah příloh:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. SOMMERHOFF, Peter a Bernhard RUMPE. Kotlin for Android app development. Boston: Addison-Wesley, [2019]. ISBN 978-0134854199.
2. BENNETT, Jim. Xamarin in action: creating native cross-platform mobile apps. Shelter Island: Manning, [2018]. ISBN 978-1617294389.
3. NEUBURG, Matt. IOS 10 programming fundamentals with Swift: Swift, Xcode, and Cocoa Basics. Third edition. Sebastopol, California: O'Reilly Media, [2017]. ISBN 978-1491970072.
4. Flutter Documentation – Flutter . Flutter – Beautiful native apps in record time [online]. Dostupné z: https://flutter.dev/docs
5. Xamarin.Forms – Xamarin | Microsoft Docs. [online]. Dostupné z: https://docs.microsoft.com/cs-cz/xamarin/xamarin-forms/
6. API reference | Android Developers. Android Developers [online]. Dostupné z: https://developer.android.com/reference/

Vedoucí bakalářské práce: **Ing. Erik Král, Ph.D.**
Ústav počítačových a komunikačních systémů

Datum zadání bakalářské práce: **28. listopadu 2019**
Termín odevzdání bakalářské práce: **15. května 2020**

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Ve Zlíně dne 9. prosince 2019

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 10.08.2020 David Sedlář, v. r.

ABSTRAKT

Cílem této bakalářské práce je popsat multiplatformní framework pro vývoj mobilních aplikací Flutter a jeho návrhové vzory. Úvod práce je věnovaný srovnání na trhu dostupných multiplatformních frameworků na trhu a jejich základní činnosti. Teoretická část se zabývá popisem základních metodik frameworku Flutter, jeho jednotlivých návrhových vzorů a jejich užitím v praxi. V praktické části je na základě předchozích znalostí navržena a následně naprogramována aplikace demonstrující některé z návrhových vzorů a techniky vývoje v tomto frameworku.

Klíčová slova: Mobilní aplikace, Flutter, Android, iOS, Multiplatformní vývoj, Reaktivní programování

ABSTRACT

The aim of this bachelor thesis is to describe a multiplatform framework for the development of mobile applications, Flutter and its design patterns. The introduction is devoted to the comparison of existing multiplatform frameworks on the market and their basic activities. The theoretical part is focused on the description of the basic methodologies of the Flutter framework, as well as the description of individual design patterns and their use in practice. In the practical part, based on previous knowledge, an application demonstrating and subsequently programming an application demonstrating some of the design patterns and development techniques in this framework is designed and subsequently programmed.

Keywords: Mobile applications, Flutter, Android, iOS, Multiplatform development, Reactive programming

Rád bych poděkoval panu Ing. et Ing. Erikovi Královi, Ph.D., za vedení této práce, veškeré připomínky, doporučení, rady, bez kterých by bylo o mnoho těžší udržet směr a kvalitní zpracování této práce. A především děkuji za myšlenku vytvořit aplikaci, která má využití nejen v reálném světě, ale mohla by sloužit spolu s touto prací jako základ výukového materiálu budoucích studentů fakulty, budoucích mobilních programátorů.

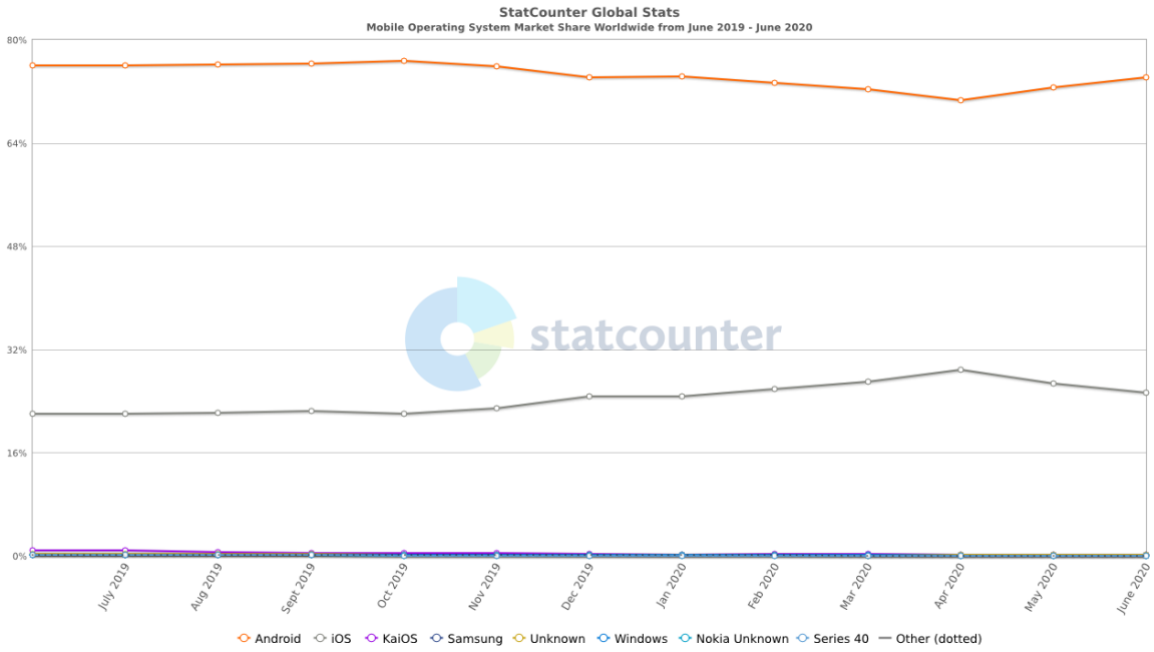
Mé poděkování patří taktéž mé mámě a bratrovi, kteří mi byli vždy pevnou oporou a podpořili mě í na složitých životních cestách.

OBSAH

ÚVOD

V současné době se s mobilními zařízeními setkáváme na denní bázi a většina z nás je schopna své mobilní zařízení využívat každý den nejen k zábavě a běžné osobní agendě, ale také v rovině pracovní.

V oblasti operačních systémů na globálním trhu dominuje operační systém Android americké společnosti Google, s tržním podílem 74,3 % v lednu 2020. Tento systém následuje operační systém iOS taktéž americké společnosti Apple, který v lednu 2020 dosáhl tržního podílu 24,76 %. S mizivými procenty pod 0,5 % se na trhu stále drží ostatní operační systémy jako například KaiOS, Samsung OS nebo Windows OS [1].



Obrázek 1 - Tržní podíl mobilních operačních systémů

Právě rozmanitost těchto operačních systémů má za následek velké množství technik vývoje a jednotlivých přístupů. Neexistuje zcela univerzální jazyk, který by byl použitelný pro vývoj všech platforem. Naopak je problematické vyvíjet jednu aplikaci na vícero systémů, protože každý operační systém vyžaduje vlastní nástroje, jazyky a metody pro vývoj a distribuci aplikace. Mluvíme o nativním vývoji mobilních aplikací.

Z tohoto důvodu již několik let existuje možnost multiplatformního vývoje aplikací. Tento vývoj má za cíl pomocí jednoho nástroje, frameworku, či jazyka poskytnout nástroje a

prostředí pro vývoj aplikace pro jeden či více systémů zároveň. Při vývoji pomocí multiplatformních frameworků či nástrojů hovoříme o hybridních mobilních aplikacích.

Pojem hybridní mobilní aplikace dříve pojednával zejména o formě vývoje, při kterém byly využívány webové technologie jako například HTML, CSS a Javascript spolu s prvky nativními. Tyto aplikace byly realizovány pomocí nástrojů pro webový vývoj, kód byl postavený podobně jako webová stránka a doplnění nativních prvků, např. o senzory GPS, Bluetooth modul či další, umožňovalo vytvářet velmi rychle mobilní aplikace zároveň pro obě platformy.

V dnešní době s rozmachem technologií nespoléhá multiplatformní vývoj pouze na kombinaci technologií webových, ale stále častěji jsou dostupné frameworky, které umožňují vývoj pomocí většího počtu známých a dostupných jazyků. Jedním z příkladů může být například framework Xamarin od společnosti Microsoft, díky němuž je možné vyvíjet aplikace v jazyce C# za pomocí .NET standartu [2].

Když se zaměříme na samotný framework Flutter, zjistíme, že funguje na jazyce Dart [3], taktéž vyvíjený společností Google. Samotný framework je společností prezentován jako UI toolkit pro vývoj aplikací pro mobilní zařízení, web a desktopové aplikace s jednou kódovou základnou [4].

I. TEORETICKÁ ČÁST

1 PROBLEMATIKA TVORBY NATIVNÍCH A MULTIPLATFORMNÍCH APLIKACÍ

Při vývoji mobilních aplikací je důležité zejména ujasnit si, pro jakou platformu chceme danou aplikaci cílit. Zde může nastat zlomový problém v případě, kdy chce společnost cílit na zařízení jak z rodiny Android [6], tak i na zákazníky společnosti Apple využívající iOS systém [7]. Každá platforma vyžaduje specifický přístup a vývojové techniky, a právě díky tomu, že je nutné aplikaci vyvíjet zvláště pro každý operační systém, vzniká finanční náročnost na celý projekt. Ne vždy však můžeme spoléhat na multiplatformní vývoj a využití dostupných frameworků. Některé aplikace je potřeba psát zvláště pro každý systém cíleně, a to z důvodu právě využívání dané systémové úrovně, ovládání senzorů zařízení či jen při zvýšených nárocích na performance aplikace. Obecně se dá říct, že hybridní či multiplatformní aplikace napsané v určitém frameworku jsou o něco málo pomalejší než aplikace napsané nativně.

1.1 Nativní vývoj aplikací

Nativní vývoj nám umožňuje dodávat aplikace specificky přímo pro danou platformu. Výsledkem jsou rychlé aplikace, které není fungují na úrovni samotného hostitelského systému, mají přístup ke všem specifikům zařízení a doprovází je jednotný vzhled základních grafických komponent, které se pro jednotlivé platformy odlišují.

Nativní aplikace jsou také jednodušeji publikovatelné na distribuční platformy jednotlivých systémů. Veškeré aplikace, které vývojář potřebuje a chce nahrát do těchto systémů a následně distribuovat, jsou kontrolované danou platformou. Když se zaměříme například na publikování iOS aplikací, zjistíme, že v roce 2017 došlo ke zpřísnění zásad pro publikování aplikací na AppStore a značně začaly být omezovány aplikace, které ve velké míře obsahovaly technologie WebView [8]. Tyto technologie využívalo velké množství právě hybridních aplikací založených na webových technologiích.

Samotný vývoj nativních aplikací se taktéž liší programovacím jazykem, kde například u systému Android ve vývoji mnoho let převládala Java, která je však nyní postupně nahrazována jazykem Kotlin od mezinárodní společnosti JetBrains. [5] Kotlin je postupně prosazován společností Google jako oficiální jazyk pro vyvíjení Android aplikací. Samotný jazyk vychází z jazyka Java a jeho kompatibilita a možnost konverze umožňuje stávajícím

Java programátorům plynule přejít na Kotlin či rovnou přeložit dané zdrojové soubory napsané v Javě právě do Kotlinu.

Vývoj aplikací pro systémy iOS od společnosti Apple je o něco více omezen a je podložen řadou nástrojů existujících pouze pro iOS prostředí nebo vývojářských licencí s ročními poplatky. Samotný vývoj iOS aplikací probíhá v jazyce Swift, který je výše zmiňovanému jazyku Kotlin velice stylisticky podobný, avšak existence Swiftu sahá mnohem dále. V nedávné době byl společností Apple vyvinut nový UI jazyk zvaný SwiftUI, který až nápadně připomíná framework Flutter a má být alternativou Flutteru jen pro iOS vývojáře [9]. Tento jazyk má za úkol umožnit iOS vývojářům rychlý a snadný vývoj aplikací s velice příjemným vzhledem, který je jednoduchý na sestavení. Tento jazyk je možné využít nejen pro mobilní zařízení ale i pro desktopové zařízení s iOS.

1.2 Android

Android je operační systém vyvinutý americkou společností Google. Základ tohoto systému je postavený na linuxovém jádře.

Existuje mnoho verzí tohoto operačního systému, z nich každá má různé úrovně vývojových API a podporuje rozdílnou funkcionalitu a ovládání hardwarových prvků. Při vývoji je tedy nutné zjistit, s jakými prvky potřebujeme pracovat a která verze systému nám toto bude umožňovat. Nejrozšířenější verzí tohoto systému je v současnosti Android 9.0 s kódovým označením Pie, běžící na více než 31,3 % veškerých zařízeních [10].

Jak již výše zmíněno, vývoj aplikací na tento operační systém byl mnoho let primárně v jazyce Java, avšak v dnešní době se však klade větší důraz na jazyk nový a to Kotlin. Společnost Google se snaží Kotlin referencovat jako oficiální jazyk pro vývoj Android platformy. Oba z těchto uvedených jazyků je možné používat ve vývojovém prostředí Android Studio [11], které nabízí veškeré potřebné nástroje pro vývoj této platformy, od nástroje pro aktualizaci jednotlivých vývojových komponent, nástroje pro debugging kódu, správy virtuálních zařízení, generátor podpisových klíčů aplikací až po možnost instalace pluginů k dalším vývojovým nástrojům jako například Gradle.

Při vývoji pro platformu Android nemusíme cílit jen na oblast mobilních technologií. Rozsah platformy se rozšiřuje i na chytré hodinky, aplikace pro chytré televizory nebo například systém do osobních automobilů a to Android Auto.

V současné době máme k dispozici řadu nástrojů napomáhajících kvalitnímu a rychlému vývoji v této platformě. Společnost Google v roce 2018 publikovala Android Jetpack, sadu nástrojů a komponent sloužící právě ke kvalitnímu vývoji aplikací pro tuto platformu.

Tato sada nástrojů obsahuje například komponenty pro správu velikostně velkých download akcí na pozadí, vylepšené komponenty pro správu uživatelských povolení nebo například komponenty pro přehrávání audia či videa [12].

Důležitou součástí vývoje je také použití správných vývojových technik, což nám při vývoji rozsáhlých aplikací může velice usnadnit práci a zvýšit znovu použitelnost kódu.

1.3 iOS

Operační systémy od společnosti Apple můžeme rozdělit na dvě hlavní skupiny. První je operační systém pro počítače Mac a notebooky MacBook označován jako macOS nebo též OSx. Do druhé skupiny můžeme zařadit systém pro přenosná zařízení typu iPhone, iPod nebo iPad nazývající se iOS. Právě vývoj pro iOS je druhou nejrozsáhlejší skupinou vývoje nativních aplikací, což je dáno především tržním podílem společnosti Apple [7].

Oproti vývoji hybridních mobilních aplikací, kde je velká řada frameworků a vývojových nástrojů dostupná zcela zdarma, nebo s licenčními poplatky, je zde vývoj aplikací značně ztížen.

Nástroje potřebné pro vývoj jako například IDE XCode [13], virtuální obrazy pro iOS simulátory nebo iOS SDK jsou dostupné pouze v systémech OSx nikoli například ve Windows, nebo Linuxu. Je tudíž nutné vlastnit zařízení z rodiny Apple, což pro mnohé vývojáře může být velice velkou investicí. Další možností je virtualizace tohoto systému, která však se sebou přináší mnohá omezení a taktéž vyžaduje vlastnictví daného systému.

Převážná většina aplikací pro iOS je napsána v jazyce Swift. Pro tento jazyk existuje výše zmíněné IDE Xcode, které sdružuje velké množství nástrojů usnadňujících vývoj. Jedná se například o editor konfiguračního souboru s příponou ".plist". Jde se o konfigurační soubor psaný v jazyce XML a obsahující většinu primárních nastavení aplikací. Dále můžeme využít například nástroj pro správu "workspace", ve kterém můžeme jednoduše zpracovávat obrázkové assety, vytvářet "splash screeny" a kategorizovat. IDE rovněž nabízí možnost debuggingu dané aplikace, její build, vytvoření archivu, následně spustitelného IPA souboru a přímou integraci a nahrání aplikace do AppStore po dokončení jejího vývoje.

Právě AppStore je koncový bod vývoje mobilních aplikací pro iOS. Tento obchod aplikací umožňuje distribuci aplikace pro celou rodinu iOS zařízení. Pro úspěšné zařazení aplikace do AppStore je třeba splnit podmínky pro úspěšné splnění AppReview. Kontrola aplikací před vydáním je v podání společnosti Apple velmi přísná a společnost klade důraz na zařazovat do koloběhu aplikací pouze ty, které jsou pro uživatele či jeho zařízení bezpečné, kvalitně zpracované a s adekvátním množstvím obsahu. Velice často se ve vývojářské komunitě stalo, že aplikace neprošla koncovým review a byla vrácena vývojářům. Důvody mohly být různé, od nedostatečného popisu využití uživatelských oprávnění až po zmiňovaný nedostatečný obsah aplikace, které sloužily například jako propagační firemní katalog. V reakci na tyto problémy Apple vydal "App Store Review Guidlines". Tato dokumentace vývojářům splnit veškerá kritéria pro úspěšné nahrání aplikace do AppStore [14].

Pro testovací účely a zlepšení koncového uživatelského zážitku nabízí Apple nástroj pro testování aplikací před samotným zveřejněním. Požadovaná aplikace se označí jako vhodná pro testování, vývojář doplní do nastavení webového rozhraní emaily testerů, následně si testeři stáhnou aplikaci TestFlight, která jim umožní instalaci testované aplikace. V rámci zpětné vazby jsou testeři schopni reportovat chyby v aplikaci a ty následně instantně zasílat vývojářům, kteří mají možnost dokončit či opravit aplikaci. Před každým zveřejněním, tedy i testovacím vydáním, musí být aplikace schválena a musí plnit kritéria výše zmiňovaného AppStore Review [15].

2 POPIS FRAMEWORKU FLUTTER A SOUVISEJÍCÍCH TECHNOLOGIÍ

V dnešní době existuje velká řada nástrojů pro multiplatformní tvorbu mobilních aplikací.

2.1 Xamarin

Xamarin je framework pro vývoj multiplatformních aplikací od společnosti Microsoft, fungující na jazyce C# a s ním spojené platformě .NET. Základním specifikem tohoto frameworku bylo rozšíření možnosti využít stávající kód pro desktopové či webové aplikace také na mobilní zařízení s operačním systémem Android, iOS, či Windows Phone. V praxi to znamená, že zdrojový kód hotové desktopové aplikace můžeme využít i pro aplikaci mobilní, s úpravami a sestavením UI pro požadovanou aplikaci.

Uživatelské rozhraní je možno psát přímo C# kódem, což může být v mnoha případech značně nepřehledné, nebo je možné využít rozšíření jazyka ve formě Xamarin.Forms, díky kterému jsou UI části napsané v jazyce XML. Psaní UI v XMl umožňuje lepší přehlednost tohoro kódu a přidává možnost myšlenkové vizualizace daného rozložení uživatelského rozhraní. Samotné části uživatelského rozhraní je možné propojit se zdrojovým C# kódem a využitím Bindingů a mnohých dalších metod je možné propojit, dynamicky měnit a upravovat data zobrazená pro uživatele, i naopak data, která zpracováváme interně v aplikaci.

V dnešní době se tento framework řadí stále mezi jedny z nejvyužívanějších multiplatformních frameworků, avšak s neustálým vývojem nových nástrojů upadá jeho popularita, především kvůli náročnosti vytváření komplexnějších UI layoutů nebo rozsáhlejších animací. Mnohé nově vzniklé frameworky totiž umožňují snadnější vývoj pěkného grafického rozhraní za menší vývojovou a časovou dotaci. Xamarin si však pravděpodobně udrží své místo, a to zejména díky možnosti psát aplikace zároveň pro desktopové nebo webové aplikace založené na .NET technologii a psané v jazyce C# [2].

2.2 Ionic

Technologie tohoto frameworku je založena na vývoji pomocí již výše zmiňovaných webových technologiích. Základními nástroji jsou HTML elementy, stylování pomocí CSS a převážná část funkcionality je řešena pomocí Javascriptu. Vývojářský tým stojící za tímto frameworkem jej prezentuje jako framework zaměřený převážně na „user experience“. Projevuje se zde snaha poskytnout nástroje pro snadnou tvorbu velmi vzhledného uživatelského rozhraní [16].

Výhodou tohoto frameworku je podpora několika dalších programovacích nástrojů. V praxi to znamená, že je možné vytvářet Ionic aplikace pomocí Reactu, Angularu nebo VueJS. Webový vývojář, který dostane do ruky tento nástroj, se s ním v relativně krátkém čase naučí pracovat a následně může jednoduše produkovat mobilní aplikace.

Framework také obsahuje řadu předem nastylovaných komponent usnadňujících vývoj. Jedná se například o tlačítka, pickery data či času, sady ikon, pohledy pro zpracování listů a mnohé další.

Ionic je jedním z frameworků založených právě na technologii „WebView“. To znamená, že výsledná aplikace není překompilována do nativního kódu a jedná se tedy spíše o webovou aplikaci běžící v zobrazovací komponentě konkrétní platformy. I tak se ale jedná o framework umožňující distribuci do jednotlivých obchodů operačních systémů jako PlayStore společnosti Google, nebo AppStore společnosti Apple. Další frameworkem využívajícím tuto technologii je například Adobe PhoneGap [17].

2.3 React Native

React Native je technologie vycházející z Reactu, který je primárně využíván pro frontendový vývoj webových aplikací. Na rozdíl od ReactJS, React Native nepracuje s webovými prvky, namísto toho je výsledná aplikace složena z prvků nativních. Při vývoji je využíváno Javascriptového standartu ES6, který přidává do klasického Javascriptu mnohá vylepšení.

Základním kamenem jsou tzv. komponenty. Komponenta je vizuální část aplikace složená z jednoho, či více prvků. Skladba komponentů probíhá pomocí syntaxe velice podobné HTML, kde namísto divu je využíván element „View“. Komponenty můžeme stejně jako v HTML skládat do sebe a vytvářet komplexnější objekty.

Každá komponenta má svůj vlastní stav „State“. Stav je proměnlivý a může se měnit v jekémkoli čase. Ve stavu definujeme, co se při jeho změně má stát, např. vepsání proměnné do komponenty, a dále vzhled složený z komponent. Při změně stavu dojde ke znovu vyrenderování související komponenty a provedení definovaného stavu, v našem případě by tedy šlo o vepsání proměnné do určité komponenty vždy, když by se daný stav změnil.

Tato tzv. reaktivita je výchozím bodem tvorby aplikací pomocí Reactu a jeho souvisejících technologií.

Dalším zásadním typem při vývoji aplikací v React Native jsou tzv. „props“, které jsou neměnné a slouží k předávání informací mezi jednotlivými komponentami. Základem jsou dvě komponenty, jedna kontejnerová, obsahující definovaný stav a vzhled komponenty, a druhá, prezentační komponenta, která přebírá data od kontejnerové komponenty při změně jejího stavu, a s těmito daty následně vytvoří svou vlastní definovanou vizuální část.

Stylování veškerých komponent probíhá pomocí tzv. „StyleSheets“ které jsou velmi obdobné jako například CSS při webovém vývoji. Díky své funkcionalitě je však vývojář schopný přidat i například rozšiřující mixiny, tj. třídy, které mohou dále rozšiřovat již definované styly. Toto chování je známé například ze SASSu [18].

2.4 Flutter

Obsah této práce je založen na popsání jednoho z nejnovějších frameworků pro vývoj multiplatformních aplikací. Flutter byl vydán v roce 2017 společností Google, přičemž první stable verze byla vypuštěna do světa v prosinci roku 2018.Možnosti Flutteru nejsou zdaleka omezeny pouze na vývoj aplikací mobilních, ale i na vývoj aplikací desktopových a to pro zařízení běžící pod systémy macOS,a také pro aplikace webové. Podpora pro vývoj desktopových aplikací na systémech Windows či Linux se momentálně nachází ve fázi "Developer preview" a dokončují se úpravy před zařazením do hlavní verze frameworku.

2.4.1 Způsoby kompilace kódu

Flutter narozdíl od ostatních hybridních frameworků fungujících na principu webových technologií a zobrazujících aplikace skrze WebView umožňuje překlad do nativního kódu a využívá nativních komponent uživatelského rozhraní.

Co je velmi značnou výhodou oproti jiným frameworkům je využití AOT kompilace, která při vydávání produkční verze aplikace zajišťuje lepší výkon, snižuje konečnou velikost a odstraňuje funkcionality, které byly důležité zejména ve vývojové verzi.

Výsledná aplikace je tedy kompilována do nativního kódu dané platformy, a to jak pro iOS, tak i pro Android systémy. V případě vývoje webových aplikací pomocí Flutteru je výsledná aplikace kompilována do čistého Javascriptu.

2.4.2 Vykreslování komponent

Způsob vývoje vychází z frameworku React Native, kde jsme se mohli setkat s takzvanou "reaktivitou". Pokud nadefinujeme naše uživatelské rozhraní, a budeme na určité komponentě chtít provést změnu, například změnit text určitého labelu, nadefinujeme změnu proměnné právě v určitém stavu. Při aktualizaci stavu si Flutter, nebo další reaktivní frameworky vytvoří kopii daného uživatelského rozhraní, změní naši nadefinovanou proměnnou, nebo provedou další definované metody, a poté porovnají změněné rozhraní s předešlým.

Výsledkem je překreslení pouze těch elementů, které se reálně změnily v průběhu volání aktualizace stavu, a nepřekreslí se nám celá stránka. Díky tomuto řešení se můžeme setkat s velmi dobrým výkonem vykreslování jak jednoduchých, tak komplexních aplikací i na starších zařízeních. Flutter si klade za cíl vykreslovat s výkonem až 60 snímků za vteřinu s možností vykreslování 120 snímků za vteřinu na schopných zařízeních.

2.4.3 Stavba aplikace

Základním stavebním kamenem aplikace ve Flutteru jsou takzvané "widgety". Pomocí nich definujeme stavbu a vzhled uživatelského rozhraní. Každá část uživatelského rozhraní ve Flutteru je widget. A tím opravdu můžeme myslet vše, s čím pracujeme. Jako widget si můžeme představit například tlačítko, ale uvnitř tlačítka se ukrývá widget vykreslující text, tlačítko leží v námi nastylovaném formuláři, který je taktéž widget, a konečně, tento formulář se nachází na stránce, a i ta je samostatný widget. Tato technika vývoje nám tudíž zaručuje rozsáhlou možnost manipulace s objekty, díky tomuto můžeme například vložit celou stránku do stránky navigační a tu dále zabalit do námi vytvořeného widgetu. Pro manipulaci s obsahem a jednotlivými elementy máme k dispozici dva základní widgety lišící se způsobem zpracování vlastního stavu.

Jedním z nich je StatelessWidget. Jedná se o widget, popisující vzhled uživatelského rozhraní složeného z jednoho a více widgetů. Tento popis je opakován, dokud widget nebude dokončen a stromová struktura nedojde až k nejposlednějšímu widgetu. Jedná se o widgety, které nemění svůj stav a jsou vykresleny převážně jednou. Jako příklad můžeme vzít vykreslený text v barevném obdélníku, v tomto příkladě by se mohlo jednat o dva separátní widgety, a to daný obdélník a text uvnitř. Oba tyto widgety můžeme sloučit do jednoho Stateless Widgetu a vytvořit například celou textovou komponentu, kterou budeme využívat napříč aplikací. Díky tomuto přístupu se vyhneme opakovanému psaní rozsáhlých komponent a namísto toho je budeme pouze recyklovat a využívat na jiných místech. Problém může nastat, pokud budeme chtít widget překreslovat a měnit jej. Pro tento scénář budeme nuceni využít Statefull Widget. Jedná se o widget, složený ze dvou základních částí, a to samotný widget a jeho State, tedy stav. Tyto widgety jsou immutabilní, a své měněné hodnoty uchovávají ve svém stavu, který je typově přiřazený widgetu samotnému. Pro vytvoření takového widgetu je tedy zapotřebí vytvoření dvou separátních tříd, nejprve třídu widgetu uchovávající immutabilní parametry, jako například určitý vstupní text. Dále je třeba vytvořit třídu dědící ze třídy State, tedy samotnou třídu definující proměnlivý stav widgetu, v této třídě je zapotřebí nadefinovat „build()“ metodu a v ní samotný vzhled widgetu. Pro použití tohoto stavu a správné fungování aplikace musíme ve widgetové třídě přepsat bázovou metodu „createState()“ a flutteru říci, jaký stav pro tento widget má použít. Nesmíme zapomínat na typovou spojitost stavu a widgetu. V samotném stavu dále můžeme definovat property, které budeme chtít měnit, za příklad si uveďme počet stisknutí určitého tlačítka, tedy proměnnou typu „int“, kterou na začátku nastavíme na 0. Po stisku tlačítka budeme chtít proměnnou a její hodnotu změnit a tuto změnu vepsat dále do UI. To provedeme zavoláním metody „setState ()“, ve které definujeme, co se při překreslení stavu má provést, v našem případě inkrementace proměnné. Tato metoda má za následek provedení veškerých vnitřních příkazů a následně překreslení samotného widgetu. Takto můžeme tvořit určité prvky uživatelského rozhraní a dynamicky měnit jejich property, parametry, případně jejich vzhled a chování. Avšak i při tomto využití může nastat problém zejména při složitější struktuře widgetů, kde budeme chtít překreslovat pouze jednu jedinou komponentu zvnějšku jí samotné, a ne celý strom widgetů, ve kterém je komponenta obsažena. Pro tyto případy je ve flutteru využito reaktivní programování a je možné implementovat velké množství knihoven, které jsou popsány níže v této práci.

Posledním a nejdůležitějším widgetem, se kterým se v rámci vývoje v tomto frameworku můžeme setkat je Inherited Widget. Z názvu si můžeme všimnout, že tento widget nám bude pomáhat vytvářet aplikace pomocí určitého dědění. Tento widget budeme využívat především v případě komplexní a vysoce zapouzdřené stromové struktury našich widgetů, například na jedné stránce s komplexním grafickým rozhraním, kde budeme potřebovat předat hodnoty, či provést funkce z kořenového widgetu i v nejspodnějších zapouzdřených widgetech. Pokud bychom chtěli předat hodnotu níže do stromové struktury, můžeme ji předat v parametru do dalšího widgetu, problém však nastane, když bude logiku předat ještě dále a hlouběji. Namísto předávání parametrů stále níže a zvyšování nepřehlednosti kódu existuje možnost využití právě tohoto widgetu. Jedná se o komponentu, opalující kořenový widget a obsahující veškeré požadované property, či metody, které bychom chtěli použít i v níže zapouzdřených widgetech. Samozřejmě namísto vytváření nových instancí tohoto widgetu, či předávání jeho samotného si můžeme z kontextu dané struktury vytáhnout právě jednu instanci tohoto widgetu a využívat uvnitř něho vytvořené metody či property. Vytažení právě této jedné instance je zapříčiněno definováním statického jmenného konstruktoru ve třídě widgetu. Ten se postará o uložení instance tohoto objektu widgetu do kontextu aplikace a ochranu vázanou typem widgetu. Díky tomuto přístupu není nutné vytvářet specifické komponenty, které by byly závislé na přebírání rodičovských parametrů a můžeme využívat více generické komponenty napříč aplikací.

Všechny tyto třídy tvoří základní strukturu stavby jednotlivých widgetů v celé aplikaci. Samotný framework flutter však obsahuje velké množství komponent, tedy widgetů, které jsou již touto metodikou vytvořené a jsou připravené pro užití vývojářem. Může se jednat o widgety jednoduchých, například widgety udávající strukturu aplikace a držící své potomkové widgety, nebo widgety komplexnější například uživatelské vstupy, u kterých je třeba měnit daný stav, ovládat interakci mezi uživatelem a aplikací, nebo widgety určené k usnadnění vytváření animací jako například AnimatedContainer, což je widget měnící svůj vzhled na základě vložených a při stavu změněných hodnot [4].

3 NÁVRH UKÁZKOVÉ APLIKACE S VYUŽITÍM FRAMEWORKU FLUTTER

Pro vývoj aplikací ve Flutteru a Dartu můžeme využít mnoha různých metodik. Pomocí různých knihoven můžeme být schopni tvořit aplikace jiným způsobem než za pomoci čistého Dartu a nástrojů frameworku.

Mnoho těchto balíčků přidává silná rozšíření nad stávajícími možnostmi jazyka a poskytují tak nástroj pro kvalitnější vývoj například rozsáhlejších aplikací. Velká část balíčků se zaměřuje na rozšíření nástrojů pro asynchronní programování a dynamické aktualizace jednotlivých widgetů pomocí různých toků dat, zvaných Streamy.

Streamy jsou jednou z metod asynchronního vývoje v tomto jazyce. Jsou sekvencí asynchronních událostí, které můžeme brát podobně jako kolekce. Na rozdíl od kolekcí ale nedostaneme žádanou událost ihned, ale dostaneme ji v tu chvíli, jakmile bude připravena k použití.

V asynchronním programování v Dartu existují též Future metody, které jsou obdobou například Tasků v jazyce C#. Jedná se o metody, jejichž výpočet není proveden ihned a provedení této metody vyžaduje vykonání různých dalších operací či výpočtů. V takovém případě nám Future metoda řekne, jakmile budeme mít výsledek připravený ke zpracování.

Příklad Future metody můžeme najít při přihlášení uživatele, kdy po zadání přihlašovacích informací je nutné provést vícero dotazů na server. Při použití normálních metod by se prováděly okamžitě a výsledkem by mohlo být zaseknutí grafického uživatelského rozhraní. V případě využití Future metody se metoda bude provádět asynchronně, můžeme tedy uživateli zobrazit načítací prvek, který nebude zaseklý a bude v pohybu. Jakmile bude přihlašování dokončeno, můžeme na základě výsledku z Future metody pokračovat v chodu programu.

3.1 ReactiveX

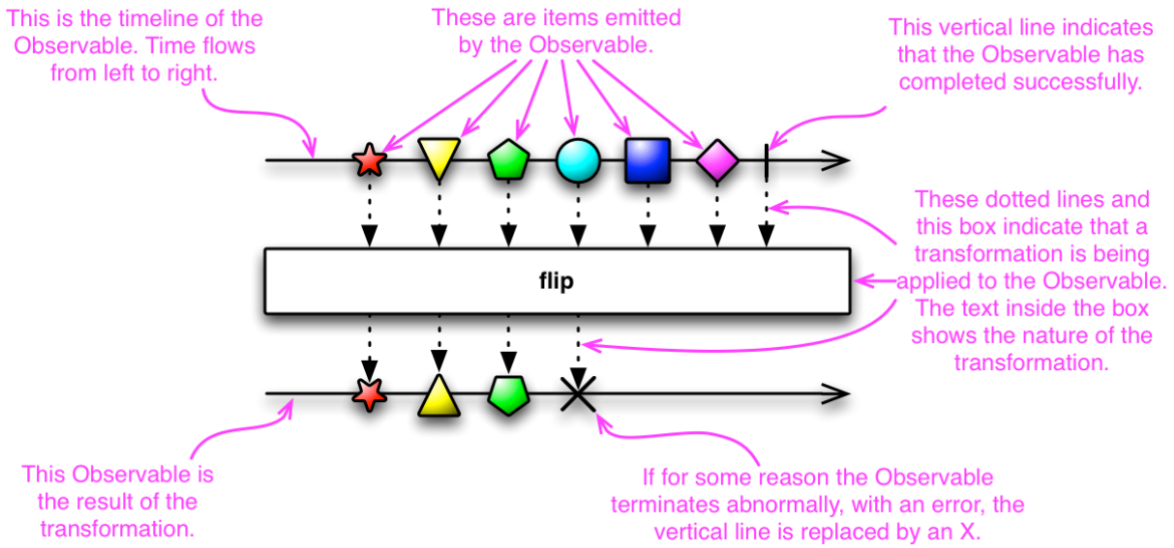
Většina níže popisovaných knihoven vychází svou funkcionalitou z knihovny ReactiveX, která je základem pro novodobé asynchronní programování založené na využití eventů a observables. Tato knihovna si klade za cíl vytvořit rozhraní velice podobné práci například s poli či seznamy, ale určené pro práci s asynchronními toky dat. Tato metodika rozšiřuje samotný 'Observer pattern', který je hojně využíván nejen v mobilním programování. Hovoříme tedy o zpracování určité sekvence dat, kterou je možné kompozitovat do sebe a jejíž abstrakce využita v tomto modelu nám umožňuje nezaměřovat se na věci, jako je například zpracování vláken, synchronizacemi konkurenční datové struktury nebo neblokující vstupy a výstupy, tato knihovna veškeré tyto záležitosti dokáže řešit za nás.

Pojem Observable udává jednoduchou kolekci dat čekající na vyvolání. Při asynchronním programování však různé toky dat mohou probíhat paralelně a mohou své hodnoty navracet až později. Je proto nutné k naší Observable připojit observery. Observer je ve své podstatě funkce, která se provede při různé akci v dané Observable. Pro jednoduchost si můžeme představit základní funkce, například „onNext(), onError(), onComplete()“. Tyto funkce jsou observery, které je nutné připojit k naší Observable. Provedení těchto observerů je podníceno stavem Observable a v těchto metodách jsme schopni zpracovat získaná data z asynchronního proudu a využít je v jednotlivém stádiu chodu Observable.

Metodu „onNext()“ vyvolává Observable při každém okamžiku, kdy emituje hodnotu. Další z metod, metoda „onError()“, je vyvolána, pokud observable selže při získávání požadovaných dat. Zde jsme schopni zachytit samotnou chybu, která metodu vyvolala. Vyvolání této metody zamezuje dalšímu volání metod „onNext()“ nebo „onComplete()“.

Poslední nejdůležitější metodou je metoda „onComplete()“, která je Observable vyvolána při finálním volání pouze tehdy, pokud nebyla zavolána žádná chybová metoda.

Ve frameworku Flutter je toto chování řešeno pomocí asynchronních streamů. Ve své podstatě fungují stejně velice podobně jako Observables a i na streamy je možné připojit funkcionální callbacky, tedy Observery. Pro jednodušší práci se streamy či zjednodušení přechodu z jiných jazyků a frameworků vzniklo velké množství knihoven fungujících právě na bázi reaktivního programování, které umožňuje lépe ovládat proudy dat v aplikaci či vykreslování koncového UI aplikace [19].



Obrázek 2 – Zjednodušené schéma knihovny ReactiveX

3.2 RxDart

Pojednání rxDart hovoří o využití ReactiveX návrhového vzoru a jeho implementaci a využití v jazyce Dart. Samotný rxDart se nepatrně liší od jeho rodiče ReactiveX a namísto přidání vlastních observables do jazyka Dart přináší velké množství tříd na bázi dartových Streamů, které značně rozšiřují kapacity a možnosti streamového asynchronního programování. Tyto metody nám umožňují vytvářet například opakující se streamy, streamy časové, rozdělovat streamy, spojovat streamy nebo využívat podobné metodiky, jakou známe například z knihoven LINQ a provádět operace jako například Zip, nad celými streamy. Využívání těchto metodik nám dává prostor vysoké flexibility a zjednodušené práce s velkým množstvím streamu v komplexních aplikacích [20].

3.3 Redux

Návrhový vzor Redux vyhledává problematiku moderních aplikací s komplexním uživatelským rozhraním a nutností jednoduchého a rychlého vykreslování daných stránek či jednotlivých elementů a snaží se přijít se způsobem, který díky určitým pravidlům a omezením určuje, jak a co se má v danou chvíli vykreslovat. Knihovna Redux byla původně vytvořena pro řešení logiky v single-page javascriptových aplikacích, ale velice rychle vznikly mnohé knihovny umožňující využití Reduxu i v již existujících frameworcích, a to například v Reactu za pomoci knihovny react-redux nebo v námi používaném Flutteru.

Redux funguje za pomoci třech základních principů, kdy prvním z nich je jednotný zdroj stavu aplikace, tím říkáme, že celá aplikace je uložena ve stromovém objektu s jedním stavovým storem. Druhým principem je vlastnost stavu pouze ke čtení, změnu stavu je možné provést pouze za pomoci emitování akce, to znamená objektu, který popisuje, co se stane. Posledním principem této knihovny je změna stavů za pomoci jednoduchých reducerů. což jsou metody, beroucí jako argument starý stav, akci a na základě těchto parametrů vyhodnotí a vrátí změněný nový stav.

3.3.1 Dílčí části pro vývoj pomocí návrhového vzoru Redux

V principu funguje Redux velice podobně jako Bloc návrhový vzor s rozdílem, že nemusíme mít celou jednu komponentu obstarávající nám změnu daného stavu, ale stačí nám pouze metoda, tedy reducer, který nám bude přetvářet stav a danou akci na nový stav. Dále je potřeba využít komponentu, která bude tvořit základ stromové struktury a bude přebírat Store. Store je prvek držící počáteční stav a ovládající reducer, díky tomuto prvku a StoreProvideru dosáhneme několika principů, a to, že bude stav dostupný po celé stromové struktuře, a také, že bude stav měnitelný pouze za pomoci reduceru. Pro vyvolání stavu využijeme widget StoreConnector, poskytující nám přístup ke storu samotnému a k zavolání callbacku, v našem případě metody reduceru. Můžeme také narazit na situaci, kdy je potřeba daný stav převést na jiný hodnotový typ, například převedení Int stavu na stav řetězce String. Pro tento případ je možné využít parametru 'converter' ve StoreConnectoru, který je schopen poskytnout nám prostor pro konverzi stavu [21].

3.4 MobX

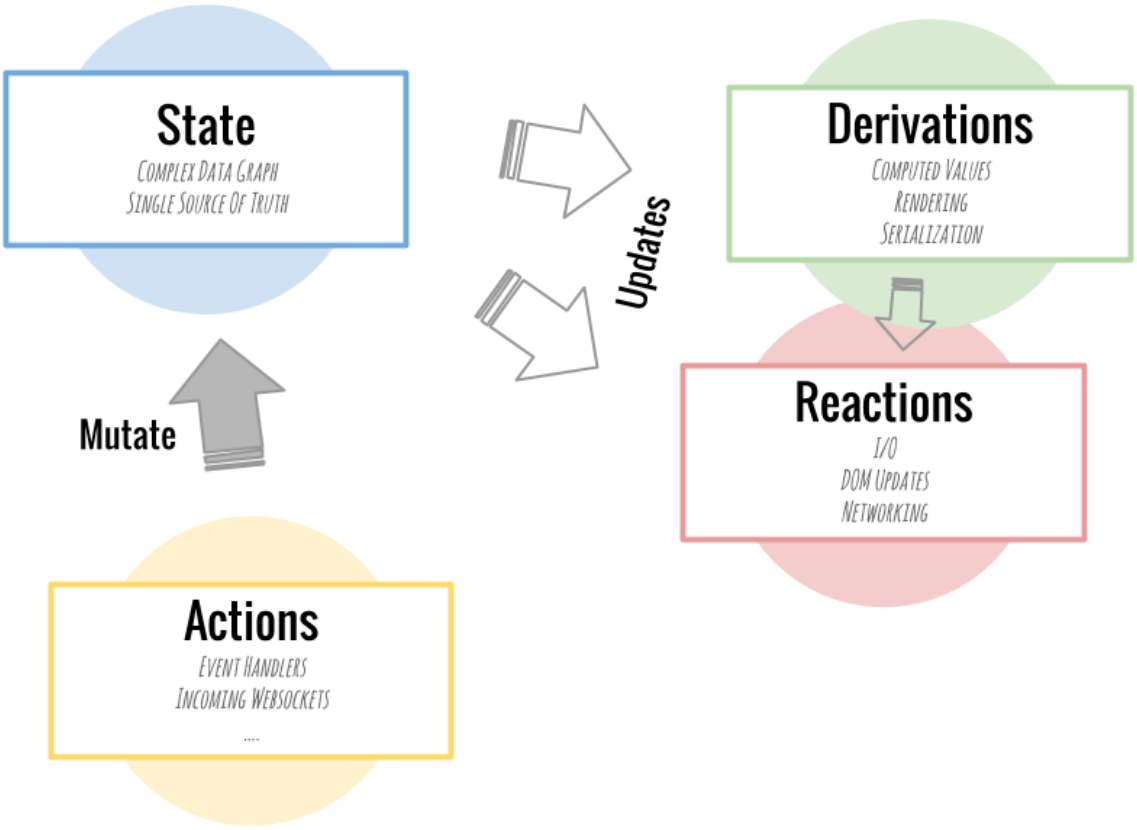
MobX je knihovna zaměřující se na správu stavů v aplikaci umožňující jednoduše propojit reaktivní data s uživatelským rozhraním. Základem této knihovny jsou tři hlavní komponenty. Veškeré deklarace komponent z tohoto balíčku předbíhá jejich anotace, která po spuštění generování dodatečných scriptů vytvoří rozšiřující soubory, které bude moci knihovna zpracovávat a potřebovat pro chod v programu.

První komponentou jsou Observables neboli pozorovatelné proměnné. Jsou to proměnné, na které jsou schopné reagovat další složky této knihovny a případně ihned reagovat a zpracovávat aktualizovaná data.

Druhou částí této knihovny jsou akce neboli Actions. Akce popisují, jakým způsobem zpracováváme naše proměnné a přidávají v logice naší aplikace sémantický význam. V rozsáhlém kódu rozhodně nechceme naše proměnné zpracovávat například pouze pomocí předponových operátorů. Bude daleko přehlednější, když vytvoříme Akci, ve které definujeme metodu měnící stav naší proměnné. Akce již ponese více přehledný a popisující název jako například "zvyš proměnnou".

Poslední komponentou, kterou můžeme v této knihovně použít, jsou reakce, ty jsou velmi podobné definování chování streamu ve výše zmíněném rxDartu. Definujeme tedy reakci obsahující metodu, která uvádí, co se má stát při změně naší proměnné pomocí její akce nebo změny samotné. Oproti předcházejícímu balíčku tato knihovna nepracuje pouze se streamy, či jiným způsobem zpracování dat ale rozšiřuje možnosti reaktivnosti v aplikaci právě díky těmto třem komponentám.

Na rozdíl od předchozích balíčků disponuje MobX speciálním prvkem, který je možné při vývoji v Dartu využít, a to je automatické generování kódu podobně jako u serializace a deserializace dat. Pro tuto funkcionalitu se využívá mobx_codegen, ten nám pomocí anotací umožňuje nadefinovat Třídu, kterou budeme využívat v rámci Mobx vzoru. Před generováním kódu nadefinujeme pomocí anotace jednotlivé observery v abstraktní třídě, kde tato třída bude předlohou mobx modelu. Při anotacích můžeme vytvořit i jednotlivé akce, tedy metody starající se o změnu daného stavu. Při vykreslování a užívání do teď definovaných prvků se využívá stejné konvence jako při tvorbě základních Flutter komponent, a to využití Statefull Widgetu a jeho stavu [22].



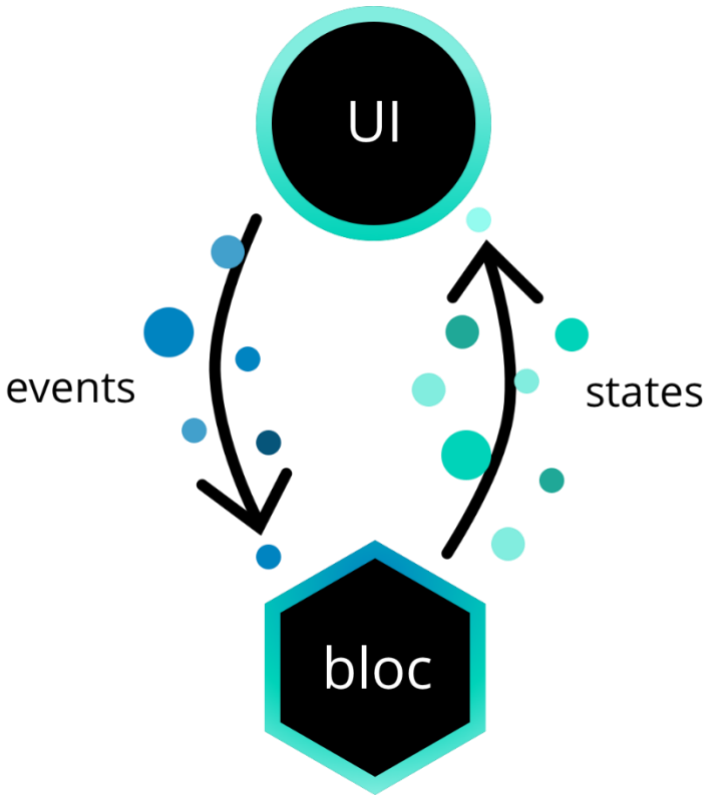
Obrázek 3 – Životní cyklus MobX komponent

3.5 Knihovna BloC

Návrhový vzor Bloc je vzor, který vznikl díky vývojářům americké společnosti Google v roce 2018, kdy byl poprvé představen na konferenci DartConf 2018 [23]. Jedná se o návrhový vzor tvořící rozhraní nad klasickými Streamy dostupnými v jazyce Dart, a který nám umožňuje oddělit logickou a designovou část aplikace. Přestože byl BloC převážně koncipován a vyvinut pro framework Flutter, existují i jeho odnože, a to například angular_bloc, umožňující jednoduše implementovat BloC logiku v Angular projektech nebo hydrated_bloc, který tvoří pomocnou knihovnu pro implementaci Cubit návrhového vzoru ve Flutter aplikacích a vývoji. Samotný Cubit je odlehčenou verzí návrhového vzoru BloC, který nevyžaduje eventy pro rozhodování a tvorbu stavů, ale naopak využívá metod, které nový stav emitují.

3.5.1 Dílčí části pro vývoj

Při rozdělování aplikace pomocí vzoru Bloc hovoříme především o datovém podkladu, který má za úkol tvořit nejnižší vrstvu aplikace, obstarávat spojení s databází a zařizovat výběr čistých dat. Datový podklad dále rozdělujeme na samotného poskytovatele dat a repositář, jehož úkolem je poskytování požadovaných dat z jednoho či více datových poskytovatelů. Nad těmito datovými podklady nalezneme samotný Bloc podklad, z anglického překladu Business Logic, který má za úkol zpracovávat eventy z prezentačního podkladu a tyto eventy převést na nové stavy, které prezentační podklad dále využívá při tvorbě UI či animací. Při vytváření BloC komponenty je důležité dědit ze třídy BloC. Hlavním prvkem této třídy bude určitý Stream, převádějící vstupní eventy na výstupní stavy, které přebírá prezentační podklad. Obecně ve streamech, které Dart poskytuje můžeme využít klíčového slova 'yield', které funguje obdobně jako návratové slovo 'return' a umožňuje nám navracet hodnoty přímo za chodu Streamu. Toto použití je v BloC velice běžné a umožňuje zpracovávat data streamu a ihned navracet požadované hodnoty. Posledním prvkem je prezentační podklad, jehož úkolem je vykreslovat Ui dané aplikace na základě získaných stavů z BloC komponent [24].



Obrázek 4 – Průběh zpracování stavů v knihovně BloC

3.6 flutter_control

Poslední popisovanou knihovnou bude flutter_control, který je naprosto komplexním řešením v oblasti state managementu a samotného ovládání celé aplikace. Knihovna pro svou logiku sdružuje velké množství existujících knihoven a výsledkem je řešení, umožňující oddělení aplikační logiky a uživatelského rozhraní, poskytnutí dependency injection návrhového vzoru napříč aplikací, řešení routování aplikace, jazykových lokalizací a implementaci eventového systému, díky kterému můžeme vyvolávat globální hodnotové nebo metodové prvky aplikace. Využití této knihovny je značně jiné na rozdíl od předchozích popisovaných, protože se nejedná jen o knihovnu obstarávající změny stavů v aplikaci a překreslování widgetů, ale i o knihovnu, pomáhající ovládat celou aplikaci. Pro svou komplexnost a jednoduchost zároveň je většina prezentační aplikace napsána právě za pomoci této knihovny a její jednotlivé, nejdůležitější prvky zde budou dále vysvětleny a popsány.

3.6.1 Dílčí části pro vývoj pomocí knihovny flutter_control

Základním prvkem této knihovny je ControlRoot, prvek, tvořící základ stromové struktury aplikace a obalující dokonce celou hlavní MaterialApp v hlavním spouštěcím souboru. V tomto kořenovém prvku jsme schopni knihovně předat mnohé základní parametry a určit do značné míry její chování. Mezi definovatelné prvky patří například definice rout jednotlivých stránek aplikace, které se dále interně využívají pro navigaci v aplikaci, dále definice lokalizačních souborů, volba načítání aplikace, vytvoření instancí přístupných kdekoli v aplikaci za pomoci kontrolního provideru nebo například definice defaultní stránky, která bude první vykreslenou právě v rámci MaterialApp. Knihovna taktéž poskytuje zjednodušené využívání tématu aplikace, kde po předchozím definování tématové třídy můžeme jednoduše a globálně v rámci celé aplikace přistupovat k hodnotám například barev, velikostem jednotlivých paddingů či marginů nebo stylům písma.

Při dělení aplikační struktury dále pracujeme s kontrolery a widgetovými modely. Pro vytváření uživatelského rozhraní knihovna obsahuje velké množství widgetových modelů, jedná se převážně o wrappery nad klasickými stateless či statefull widgety, poskytující funkcionalitu navíc, a to například ovládání metod jazykové lokalizace či právě přístup k tématu aplikace. Příkladem může být například BaseControlWidget, což je widget, který je obdobou StatelessWidgetu, jeho stav je immutabilní a nemá žádné kontrolery, které by umožňovaly změny jeho chování, neboť veškeré chování musí být měněno ze zevnějšku.

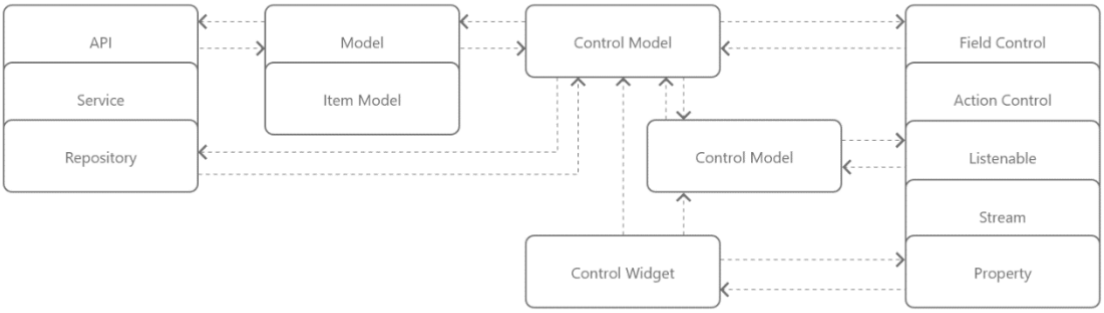
Naopak můžeme použít například SingleControlWidget, jedná se o widgetový model obsahující jeden kontroler přímo k němu vázaný. V daném kontroleru můžeme řešit business logiku této komponenty a oddělit veškeré hodnoty a metody. Pokud bychom potřebovali model s více kontrolery, můžeme použít ControlWidget. Jedná se o model, umožňující pracovat s více než jedním kontrolerem, díky čemuž jsme schopni danou logiku ještě více zpřehlednit, a to například rozdělením na grafické kontrolery starající se o ovládání animací a UI daného modelu, a dále o business kontrolery, které řeší logiku v modelu. Pro využití kontrolerů v modelech je vždy nutné je inicializovat pomocí bázových metod modelů.

Knihovna dále obsahuje velkou řadu mixinů. Pod pojmem mixin se v programátorské rovině označuje třída, obsahující určité metody, které po implementaci do jiné třídy rozšíří její funkcionalitu právě o tyto metody. Není zde třeba řešit dědičnost či jiné vztahy, čímž je zaručena velká znuvupoužitelnost kódu. Tyto třídy je možné využít ve většině tříd, nejen ve třídách widgetových. Díky nim jsme schopni využívat například metody pro lokalizaci textů, předávání argumentů, routování a navigaci v aplikaci a mnoho dalšího. Příkladem může být například třída RouteControlProvider, díky které můžeme volat jinak v kontroleru nepřístupné metody pro vytvoření dialogu či obsahového modálního okna. Tyto metody jsou jinde přístupné pouze ve widgetových třídách, protože jsou závislé na kontextu daného widgetu či stránky. Druhou velmi užitečnou třídou může být LocalizationProvider, který nám taktéž umožňuje téměř v každé třídě volat metody pro lokalizaci textů. Díky tomuto chování jsme schopni i v obyčejných třídách, držících a zpracovávajících data, překládat požadované texty.

I v tvorbě mobilních aplikací je především důležité využívat právě výše zmiňované reaktivní programování pro dynamické změny na stránce bez nutnosti překreslení celé stránky. Pro tyto případy knihovna obsahuje celou řadu vlastních komponent podobně jako MobX nebo BloC. Primární části je zpracování logiky dané proměnné či prvku. Pro tuto situaci můžeme využít FieldControl, který je schopný držet jakýkoli datový typ. Existují i třídy odvozené speciálně pro primitivní typy, a to například IntegerControl, StringControl nebo BoolControl. Veškeré tyto třídy jsou typově vázané a drží požadovanou hodnotu. Důležitou vlastností, kterou však disponují, je vnitřní logika založená na streamech, která se stará o překreslení odpovídajícího builder widgetu v grafické části.

Oproti kontrolním třídám existují zmíněné builder widgety. Při vytváření widgetu je potřeba definovat jeho logický kontroler, který se bude díky vnitřnímu streamu starat o překreslení

potomka uvnitř builderu. Pomocí frameworkového builderu jsme schopni vytvořit požadovaný widget a předat si hodnotu z kontroleru. Při každé změně hodnoty v kontroleru dojde k notifikaci a překreslení potomka ve widgetu spolu s předáním aktualizované proměnné. Příklad tohoto chování je v naší aplikaci na hlavní stránce v podobě hodin, aktualizujících se každou vteřinu, kde je pomocí časového streamu vždy ukládán aktuální čas a vždy po přidání tohoto času dojde k překreslení hodin, což zaručí vizuální efekt vteřinové proměnlivosti [25].



Obrázek 5 - Zjednodušené schéma knihovny flutter_control

II. PRAKTICKÁ ČÁST

4 TVORBA PŘÍKLADŮ A VZOROVÝCH ŘEŠENÍ DEMONSTRUJÍCÍCH KLÍČOVÉ PRVKY TYPICKÝCH MOBILNÍCH APLIKACÍ

Aplikace je zaměřena na zlepšení komunikace mezi studenty a jejich přiřazeným vyučujícím na seminář či přednášku. Po přihlášení do aplikace budou mít uživatelé nastavená práva a možnosti manipulace s aplikací podle rozdělení na studenta, nebo vyučujícího. Pro studenty bude aplikace zobrazovat jejich základní informace jako, jejich osobní číslo, denní rozvrh a časové rozdělení, obor a fakultu. Pro vyučujícího bude aplikace zobrazovat základní informace podobně jako pro studenta, včetně rozvrhu vyučujícího, a dále bude nabízet rozšířené možnosti, a to zejména snadnou komunikaci směrem ke studentům. Vezměme si například situaci, kdy se v zimních měsících zastaví doprava ve městě a vyučující ví, že bude mít zpoždění. Studenti již ale čekají na přednášce a nemají o situaci žádné informace. Vyučující tedy otevře aplikaci a pomocí nastavení času a přidání krátkého popisu odešle zprávu všem studentům z daného semináře, že bude mít zpoždění právě tolik minut. Pomocí těchto kanálů pro daný předmět bude moci jednoduše se studenty sdílet i informační zprávy a komunikovat efektivněji než přes email. Vzhledem k hlavní prezentační činnosti aplikace, a to upomínkování, nese aplikace pracovní název "UTBeReminded"-

Aplikace by případně mohla sloužit jako výukový podklad do předmětů objasňujících zákonitosti vývoje mobilních aplikací, kde by studenti měli možnost seznámit se s tímto frameworkem a v případě zájmu či aktivity by mohli pracovat na různých částech aplikace a následně pomocí merge requestů by proběhla kontrola jejich práce a případné zařazení do celkového projektu. Studenti by tímto dostali šanci pracovat na reálném projektu s možností zviditelnění jejich práce.

4.1 Verzovací systém

Pro vývoj aplikace jsme se rozhodli využít systému pro správu verzí Git. Volba tohoto systému pro nás byla jednoznačná. Většina vývojářských společností využívá ke správě verzí právě GIT, především kvůli k jeho velkému množství funkcí a možností využití. GIT také velmi dobře funguje při sestavování kontinuální integrace a kontinuálního nasazování (CI/CD), které bude pro ukázkové účely v základní verzi nachystané i pro tuto aplikaci. Pro tyto potřeby bude také nutné mít repositář uložený i na online serveru, v našem případě bude využita platforma GitHub momentálně vlastněná společností Microsoft [26]. Uložení

repositáře na server nám též přidá větší bezpečnost týkající se zejména ztráty dat na lokálním počítači, která nebudou nenávratně ztracena právě díky uložení na serveru.

4.2 Struktura projektu

Projekt bude svou strukturou logicky vyčleňovat třídy určené pro vytváření uživatelského rozhraní, pages neboli stránky, třídy obsahující logiku jednotlivých stránek, tzv. kontrolery, objektové třídy neboli entity, třídy starající se o připojení a sběr dat z API, dále třídy obsahující komplexnější widgety uživatelského rozhraní i s vlastní logikou, například tlačítka, komponenty pro uživatelský vstup, které budeme nadále využívat napříč aplikací. Poslední část bude sdružovat pomocné nástroje, jako například funkce pro parsování dat, možné extension funkce nad hlavními funkčními typy anebo například enumy.

Poslední sekcí je složka assetů, zde budeme sdružovat obrázky, ikony, lokalizační soubory v JSON typu, fonty nebo další externí soubory, které budeme potřebovat.

4.3 Použité technologie

Pro uložení uživatelů, předmětů rozvrhu a dalších informací bude využita databáze reálného času FirebaseDatabase spadající pod vývojářskou platformu Google. Tato databáze poskytuje bezplatné řešení uložení a správy dat při vývoji multiplatformních aplikací, a to nejen mobilních. Pomocí platformy Firebase a její kategorie Firebase Authentication bude též řešeno přihlašování a registrování uživatelů. Tento nástroj poskytuje implementačně jednoduché řešení, které při napojení do aplikace zcela obstarává registraci a uložení nových uživatelů nebo naopak jejich přihlášení a případně v kombinaci s databází zisk jejich dat. [27]

4.3.1 Firebase Authentication

Pro přihlašování uživatelů, jejich registraci, ošetření zapomenutého hesla či jiných záležitostí je použit nástroj z kolekce Firebase od společnosti Google. Jedná se o nástroj Firebase Authentication.

Tento nástroj umožňuje jednoduchou registraci, při které je uložen uživatelský email a jeho heslo, a taktéž je vygenerován kód uživatele. Data jsou chráněná a hesla nejsou přístupná ani z administračního prostředí. Při ztrátě hesla je možné poslat na daný email odkaz umožňující resetování hesla. Samotné resetování hesla, generování odkazu a vypracování

nového hesla pomocí formuláře si vytváří přímo samotná Firebase a není tedy nutné nic dalšího programovat.

Pro ukládání uživatelských dat se pomocí následujícího balíčku vytvoří dokument uživatele, který jako identifikační číslo bude obsahovat řetězec vygenerovaný při registraci. Následně můžeme ukládat další uživatelská data, která budeme využívat při chodu aplikace.

Při provedení přihlášení a správném zadání vstupních údajů nám Authentication vrátí právě identifikační řetězec uživatele, pomocí kterého právě přistoupíme k vlastním datům.

4.3.2 Firebase Cloud Firestore

Jako úložiště dat je využita online realtime databáze z kolekce nástrojů Google Firebase a to CloudFirestore. Jedná se o dokumentovou databázi, která poskytuje jednoduchou a rychlou implementaci napříč zařízeními.

Hlavní strukturu databáze tvoří tři kolekce. Do kolekcí postupně vkládáme dokumenty, kde je každý dokument jednotlivou entitou. Každý dokument obsahuje unikátní vygenerované číslo sloužící k jeho identifikaci. První z nich je kolekce všech uživatelů. Uživatel vzniká registrací, při níž se uloží jeho přihlašovací údaje. Dále je po uživateli vyžadováno zadat další údaje, jako celé jméno, studentské číslo a rok. Dokumenty roku obsahují namísto identifikačního řetězce číslo, označující, o jaký rok se jedná. Po zvolení roku má uživatel možnost si vybrat předměty, kterých se bude účastnit. Volení předmětů v daném roce je řízeno datem registrace, v návaznosti na toto datum si může uživatel vybrat předměty pouze pro aktuální semestr. Potvrzením registrace vzniká dokument uživatele obsahující identifikační čísla předmětů, kterých se zúčastňuje, dále je přidáváno identifikační číslo uživatele do příslušného předmětu.

Entita předmětů obsahuje základní informace o předmětu a identifikace všech uživatelů, kteří se předmětu účastní. Entita roku obsahuje pouze identifikace jednotlivých předmětů v daném roce. Slouží zejména pro registraci a výběr předmětů ke studiu.

V případě potřeby je možné dokumentům přidávat další vnitřní kolekce a libovolně tak upravovat strukturu dat.

4.3.2.1 Asynchronní metody

Pro získání dat z API, naší Firebase, vytvoříme jednotlivé třídy sdružující jednotlivé požadavky a volání na server.

Pro volání na server budeme využívat asynchronní metody pomocí Future metod, které Dart obsahuje. Future funkce může být určitého typu, v případě typu void neočekáváme žádný výsledek, pouze provedení funkce. Pokud Future funkci nastavíme typ, například primitiv boolean, očekáváme po dokončení asynchronní operace vrácení jedné či druhé hodnoty tohoto typu. Můžeme tedy například kontrolovat správnost provedení funkce. Pokud je uživatel řádněpřihlášen a získáme v pořádku jeho data, vrátí nám Future funkce hodnotu "true", pokud nastane určitá chyba, vrátí nám hodnotu opačnou tudíž "false", na kterou jsme schopni reagovat a případně uživatele v uživatelském rozhraní upozornit na selhání požadavku. V případě komplexnějších api s využitím služeb jako je například REST, můžeme pro představu požadovat vrácení response kódu. Pokud obdržíme pozitivní kód například o hodnotě "HTTP 200" nebo "HTTP 204", víme, že požadované volání proběhlo v pořádku, pokud se vrátí kód s hodnotou jinou, jsme schopni jednoduše určit, co přesně za chybu nastalo a přesně o ní uživatele informovat.

Samotné vyvolání Future provedeme pomocí klasického funkcionálního volání "future()", pokud však chceme počkat na dokončení této funkce, je nutné tuto funkci provést v asynchronní metodě a tuto metodu počkat pomocí příslušného operátoru jako "await future()". V případě, že chceme vyčkat na dokončení funkce a pracovat se získaným výsledkem, můžeme použít následující volání "future().then((result) {})". Toto volání vyčká na dokončení funkce, vrátí nám výsledek operace, který můžeme v interní funkci zpracovávat. Zápis je možný též pomocí "arrow function".

4.3.3 Json_Serializable

Přímo v aplikaci pomocí jednoduchých requestů dostaneme potřebná data, která bude nutné serializovat z dokumentů a vytvořit z nich objektové třídy. Pro samotnou serializaci dat z databáze využijeme balíček "json_serializable". Tento balíček je schopný na základě notace v třídě a jejich properties vytvořit nový soubor, který bude obsahovat další třídové komponenty, takzvané jmenné konstruktory, sloužící pro sestavení třídového objektu z objektu JSON a opačně. Nejprve je však nutné definovat factory objekt, což je konstruktor umožňující obsahovat logiku, ve které jsme schopni získat JSON data z dokumentového obrazu, a teprve tato data následně serializovat pomocí jmenných konstruktorů naší třídy.

Vývojářům tedy odpadá nutnost tyto metody psát ručně. Balíček doprovází pouze dvě nevýhody. Pro vygenerování partového souboru je nutné vyvolat sestavovací akci pomocí daného příkazu. Než je balíček vygenerovaný, kompilátor hlásí chyby v entitě, protože

partový soubor ani serializační metody prozatím neexistují. Druhý problém nastává při vytváření složitějších entit, které místo primitivních typů obsahují také například další entity.

S tímto případem si balíček bohužel není schopný poradit a vývojář je tedy nucen psát serializační metody ručně. [28]

4.3.4 Offline zpracování dat

Pro využití offline budou hlavní data ukládaná pomocí balíčku "hive". [29] Jedná se o key-value databázi, ukládající data do interní paměti zařízení. Databáze funguje na podobném principu jako například SharedPreferencies, dosahuje však lepšího výkonu ve čtení a zápisu než samotné preference a razantně většího výkonu než SQLite databáze pro mobilní zařízení.

Pomocí již definovaných jmenných konstruktorů můžeme data uložit přímo v objektu JSON pouze se zvoleným klíčem, a díky tomuto klíči je kdykoli opět získat a serializovat. Díky této funkcionalitě můžeme například držet uživatelská data a informace o daných předmětech bez nutnosti volat dotazy na databázi.

5 POPIS KLÍČOVÝCH ČÁSTÍ NAVRŽENÉHO ŘEŠENÍ

Příkladová aplikace využívá mnohé knihovny usnadňující vývoj ve frameworku Flutter. Jejich výběr závisí na vývojářských preferencích.

5.1 BloC

Knihovna BloC je jednou z nejrozšířenějších knihoven pro reaktivní vývoj ve frameworku Flutter. Tato knihovna byla představena softwarovými designery ze společnosti Google v roce 2018 [30].

5.1.1 Popis příkladu

V naší aplikaci budeme využívat BloC pro demonstraci zpracování jednoduchých eventů a jejich přetvoření na Stavy vykreslené v UI aplikace. Hlavní komponentou, kde bude tento vzor využit, bude komponenta MessageDialog, jejímž úkolem je poskytnout jednoduché rozhraní pro vytvoření zpráv pro studenty a její následné odeslání. Požadavkem je vytvoření tříkrokového rozhraní, které umožní zadat předmět zprávy, její obsah, výběr předmětů, tzn. i studentů, kterým bude zpráva doručena, a její následné odeslání.

Pro toto použití máme tedy vytvořenou komponentu MessageDialog, která dědí z flutterového Stateless, tedy bezstavového widgetu. V této komponentě je dále využit mixin LocalizationProvider umožňující nám použít metody obsažen v balíku flutter_control pro jednoduchou jazykovou lokalizaci v aplikaci. Tento nástroj je blíže popsán v kapitole '5.5. Jazykové lokalizace' této práce. Druhou komponentou je MessageDialogBloc rozšiřující samotný Bloc. V této komponentě bude nezbytnou součástí také enum StateEvent definující jednotlivé stavy, které budeme zpracovávat, a na základě kterých budeme následně vykreslovat UI aplikace.

5.1.2 Implementace řešení v aplikaci

Základní komponentou vytvořenou pro ukázky práce BloC návrhového vzoru je MessageDialog. Pro usnadnění případné práce s globálními proměnnými či lokalizací tento dialog dědí BaseControlWidget z knihovny flutter_control. Tato dědičnost však pouze přidává vlastnosti StatelessWidgetu a nemá žádný vliv na reaktivní zpracování UI stránky. Pro práci s BloCem je třeba vytvořit instanci BloCu samotného, zde v příkladu vytvoříme přímo getter s novou instancí MessageDialogBloc. Pro základ našeho UI je třeba využít

widget knihovny s názvem BlocProvider, typovaný BloCem samotným. Díky tomu můžeme BloC ovládat ve všech vnořených komponentách.

```
class MessageDialog extends BaseControlWidget with RouteControl {
  MessageDialogBloc get control => MessageDialogBloc();

  @override
  Widget build(BuildContext context) {
    return BlocProvider<MessageDialogBloc>(
      create: (context) => MessageDialogBloc(),
      child: Padding(
        padding: const EdgeInsets.symmetric(
            horizontal: AppTheme.padding_mid,
            vertical: AppTheme.padding_head * 2.5),
        child: Container(
          decoration: BoxDecoration(
              color: AppTheme.background_color,
              borderRadius: BorderRadius.circular(30)),
          width: Device.of(context).width,
          height: 300,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            mainAxisSize: MainAxisSize.min,
            mainAxisAlignment: MainAxisAlignment.start,
            children: <Widget>[
```

Zbylou část tvoří převážně dekorativní a kompozitní widgety udávající stavbu UI.
V nejnižší části je vložen vlastní widget, starající se o změny obsahu dialogu. Tento widget bude obsluhovat BloC samotný.

```
                GestureDetector(
                  onTap: () => Navigator.pop(context),
                  child: Icon(
                    Icons.close,
                    color: AppTheme.primary_color,
                    size: 25,
                  ),
                )
              ],
            ),
          ),
          Expanded(child: MessageDialogBlocBuilder()),
        ],
      ),
    ),
  );
}
```

V Záhlaví widgetu samotného inicializujeme ještě další kontrolery pro obsluhu uživatelských vstupů v obsahové části. Zde v tomto widgetu využijeme výše definovaného BlocProvideru a možnosti získání BloCu ve vnořených widgetech. Pomocí jmenného konstruktoru a typového označení si za pomocí kontextu dané stránky vytáhneme instanci BloCu. Ten budeme dále využívat v metodách back() a next(), které budou obsluhovat samotné výměny obsahu.

```
class MessageDialogBlocBuilder extends StatelessWidget
    with LocalizationProvider {
  final TextEditingController subjectControl = TextEditingController();
  final StringControl subjectIdControl = StringControl('');
  final InputControl titleControl = InputControl();
  final InputControl contentControl = InputControl();

  @override
  Widget build(BuildContext context) {
    MessageDialogBloc msgDialogBloc =
    BlocProvider.of<MessageDialogBloc>(context);

    return Center(
      child: BlocBuilder<MessageDialogBloc, String>(
        builder: (context, value) =>
            Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                _buildWidget(value),
                Row(
                  mainAxisSize: MainAxisSize.min,
                  children: <Widget>[
                    ButtonBox(
                        Text(
                          localize('back').toUpperCase(),
                          style: AppTheme.textButtonStyle(
                              color: AppTheme.background_color),
                        ),
                            () => back(value, msgDialogBloc, context)),
```

Metody samotné pouze na základě jednoduché identifikace stránky obsahu pomocí programového switche obsluhují vnitřní BloCové metody přidáním stavu do BloCu.

```
void next(
    String value, MessageDialogBloc msgDialogBloc, BuildContext context) {
  switch (value) {
    case 'subject':
      if (subjectIdControl.value.isNotEmpty) {
        msgDialogBloc.add(StateEvent.title);
      }
      break;
    case 'title':
      if (titleControl.value.isNotEmpty) {
        msgDialogBloc.add(StateEvent.content);
      }
      break;
    case 'content':
      if (contentControl.value.isNotEmpty) {
        msgDialogBloc.add(StateEvent.end);
      }
      break;
    case 'end':
      Navigator.pop(context, [
        MessageData(
            subjectIdControl.value, titleControl.value, contentControl.value)
      ]);
      break;
  }
}
```

BloC samotný dědí třídu BloC, kde typem určíme enum držící stavy pro změnu chování BloCu a návratovou hodnotu, která se bude y daných stavů po změně navracet, v našem případě String tedy textový řetězec. V samotné třídě je zapotřebí přepsat dvě metody třídy rodičovské. První z nich je metoda geteru definující počáteční stav BloCu. Druhou z nich je Stream převádějící jednotlivé stavy na výstupní proměnné. Reálně bychom mohli pomocí switche definovat jednotlivá chování pro určitý typ akce, pro náš případ však pouze potřebujeme navrátit vstupní hodnotu pouze převedenou na řetězec. Tuto úpravu provedeme pomocí námi naprogramovaného parseru, protože z výkonnostních důvodů byly z dartu pro flutter odstraněny metody pro práci s enumy.

```dart
enum StateEvent { title, content, subject, end }

/// [MessageDialogBloc] is Bloc component responsible for handling BloC events
in [MessageDialog]
///
/// For better explanation will be all logic (animation handling included)
written here.
/// In real production apps we could also separate UI logic and our Business
Logic
class MessageDialogBloc extends Bloc<StateEvent, String> {
  @override
  String get initialState => Parsers.enumValueToString(StateEvent.subject);

  @override
  Stream<String> mapEventToState(StateEvent event) async* {
    yield Parsers.enumValueToString(event);
  }
}
```

Při každém volání metody se změnou stavu pomocí metod next() a back() se tyto stavy vyhodnotí a navrátí hodnota, která je dále využívána pro vytvoření obsahové rozhraní dialogu. O překreslení těchto komponent se stará BloC samotný a není tedy nutné z naší strany provádět jakékoli metody uvnitř metody setState(). Můžeme tedy pozorovat reaktivní chování i uvnitř StatelessWidgetu.

```dart
Widget _buildWidget(String value) {
  switch (value) {
    case 'subject':
      return PartSubject(subjectControl, subjectIdControl);
      break;
    case 'title':
      return PartTitle(titleControl);
      break;
    case 'content':
      return PartContent(contentControl);
      break;
    case 'end':
      return PartOverview(
          subjectControl.text, titleControl.value, contentControl.value);
      break;
    default:
      return Container();
  }
}
```

5.2 Redux

Vývoj pomocí knihovny Redux je velmi podobný vývoji v knihovně Bloc, s řadou změn, zejména využitím kořenového Storu, který usnadňuje využití knihovny v potomkových komponentách.

5.2.1 Popis příkladu

V aplikaci budeme Redux taktéž využívat, obdobně jako návrhový vzor Bloc, a to pro změnu UI na stránce s výpisem předmětů studenta a to v SubjectPage. Požadavkem pro změnu UI bude skrývání požadovaných předmětů, označení jich jako oblíbené nebo otevírání jejich detailu. Tyto akce nám bude ovládat daný reducer.

5.2.2 Implementace řešení v aplikaci

Základní potřebou při tvorbě této stránky bylo vytvoření individuálního předmětového modelu, který bude sám nezávisle regovat na změny v jeho stavu a překreslovat svoje Ui podle potřeby. Pro využití veškerých metodik reduxu bylo nutné jako kořenovou komponentu vytvořit StoreProvider, který jako parametr bere inicializovaný store, ten se nachází v oddělené kontrolní třídě pro model, kde je při inicializaci třídy sám inicializován.

```
enum SubjectAction { Favourite, Detail, Hide, Unhide }

class SubjectModel extends SingleControlWidget<SubjectModelControl>
    with RouteControl {
  final Subject subject;
  final SubjectController subjController;

  SubjectModel(this.subject, this.subjController);

  @override
  SubjectModelControl initControl() => SubjectModelControl(this);

  @override
  Widget build(BuildContext context) {
    return StoreProvider<String>(
      store: control.store,
      child: Padding(
        padding:
            const EdgeInsets.symmetric(horizontal: AppTheme.padding_extended),
        child: Stack(
          alignment: Alignment.center,
          children: <Widget>[
            Material(
              borderRadius: BorderRadius.circular(15),
              clipBehavior: Clip.antiAlias,
              shadowColor: Colors.transparent,
              color: Colors.transparent,
              child: Container(
                  width: 250,
                  height: 80,
```

Pro možnost ovládání změn stavů a možnost ovládat samotný reducer z určitého widgetu je nutné tento widget obalit pomocí StoreConnectoru, díky jeho builderu jsme schopni dostat přímý typ vycházející z reduceru, ale také mnohé další funkcionality. V našem případě se jedná o volání na reducer pomocí kliku na ikonu viditelnosti, to je provedeno pomocí předání callback funkce ze StoreConnectoru právě do volaného callbacku nad ikonou ve widgetu gestureDetector, který je schopný odchytávat uživatelské klikové interakce.

```
child: Center(
  child: StoreConnector<String, VoidCallback>(
      builder: (context, callback) =>
          Padding(
            padding: const EdgeInsets.symmetric(
                horizontal: AppTheme.padding),
            child: GestureDetector(
              child: Icon(
                Icons.remove_red_eye,
                color: control.visible
                    ? AppTheme.primary_color
                    : AppTheme.tint_color,
              ),
              onTap: callback,
            ),
          ),
      converter: (store) {
        return () {
          store.dispatch(SubjectAction.Unhide);
          control.visible =
          control.visible ? true : false;
        };
      }),
),
```

Samotné chování reduceru je definováno ve výše zmiňovaném kontroleru, ten slouží pouze ke zpřehlednění kódu a oddělení aplikační logiky od UI. Při inicializaci kontroleru je nutné inicializovat Store samotný, dále je nutné vytvořit reducer a v našem případě pomocí jednoduchého switche definovat akce, či výstupy, které se mají provést při zavolání jednotlivé akce nad storem. Enum jednotlivých akcí je definovaný nad naším modelem. Právě volání pouze jednoduché akce nad storem za pomocí enumu podobně jako v BloC knihovně má za následek jednoduché, s omezením chybovosti, chování potřebné pro zpracovávání požadovaných dat a případné překreslování obsahu.

```
class SubjectModelControl extends BaseControl
    with LocalizationProvider, RouteControlProvider {
  Api get api => Control.get<Api>();
  Store<String> store;
  final SubjectModel model;
  bool visible = true;
  bool isFav = false;

  SubjectModelControl(this.model);

  @override
  void onInit(Map args) {
    super.onInit(args);
    store = Store<String>(actionReducer, initialState: '');
  }
  String actionReducer(String state, dynamic action) {
    switch (action) {
      case SubjectAction.Favourite:
        if (!api.subject.favourites.contains(model.subject)) {
          api.subject.saveFavSubject(model.subject);
          isFav = true;
        } else {
          api.subject.removeFromFav(model.subject);
          isFav = false;
        }
        break;
      case SubjectAction.Detail:
        routeOf<SubjectDetailDialog>()
            .openDialog(args: {'subject': model.subject});
        break;
      case SubjectAction.Hide:
        return 'hidden';
        break;
      case SubjectAction.Unhide:
        return 'visible';
        break;
    }
  }
}
```

Samotný redux je dále typově založený a při nedodržení určitého datového typu například při vytváření StoreConnectoru nebude aplikace, tedy její stránka vůbec spustitelná a bude obsahovat chybovou hlášku vyžadující přesné typové definice, čímž redux zabraňuje jakékoli nesourodosti.

5.3 flutter_control

Tato knihovna sdružuje většinu výše popsaných metodik a postupů v reaktivním vývoji a svými nástroji vytváří abstraktnější a jednoduché řešení pro vývoj mobilních aplikacích a to nejen díky předem definovaným komponentám.

5.3.1 Popis příkladu

Pomocí této knihovny je programována majoritní část aplikace. Důvodem ke zvolení této knihovny bylo převážně ještě více abstraktnější oddělení UI a business logiky, kde uživatel nemusí obalovat jednotlivé komponenty či stránky widgety, které zajistí potřebnou komunikaci se svým rozhraním, a to jak BloCem či reducery. Při inicializaci aplikace knihovna sama registruje jednotlivé routy pro námi definované stránky, tím se nám usnadní vnitřní řízení navigace v aplikaci, knihovna načte jednotlivé překladové json soubory a pomocí jednoduchých metod nám umožní lokalizaci veškerých textů v aplikaci a pomocí poskytovaných widgetů budeme moci dynamicky ovládat jednotlivé prvky aplikace. Toto se děje v komponentě ControlRoot, která má jako potomka právě výchozí MaterialApp, obalující celou UI část aplikace.

```
class MainApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    SystemChrome.setPreferredOrientations([
      DeviceOrientation.portraitUp,
      DeviceOrientation.portraitDown,
    ]);
    return ControlRoot(
      debug: true,
      loadLocalization: true,
      routes: [
        ControlRoute.build<DashboardPage>(builder: (_) => DashboardPage()),
        ControlRoute.build<InitPage>(builder: (_) => InitPage()),
        ControlRoute.build<MainPage>(builder: (_) => MainPage()),
        ControlRoute.build<SignUpPage>(builder: (_) => SignUpPage()),
        ControlRoute.build<SubjectPage>(builder: (_) => SubjectPage()),
        ControlRoute.build<MessageTeacherPage>(
            builder: (_) => MessageTeacherPage()),
        ControlRoute.build<MessagePage>(builder: (_) => MessagePage()),
        ControlRoute.build<MessageDialog>(builder: (_) => MessageDialog()),
        ControlRoute.build<SubjectModel>(
            builder: (_) => SubjectModel(null, null)),
        ControlRoute.build<SubjectDetailDialog>(
            builder: (_) => SubjectDetailDialog()),
      ],
      root: (_, __) => InitPage(),
      locales: {'cs': 'assets/localization/cs.json'},
      entries: {
        'api': Api(),
      },
      app: (BuildContext context, Key key, Widget home) => MaterialApp(
        key: key,
        home: home,
        title: APP_NAME,
        theme: AppTheme.style(),
      ),
    );
  }
}
```

Samotné kontroly a vykreslovací widgety jsme si mohli prohlédnout jak na příkladech z BloC knihovny, nebo z knihovny redux. V příkladech knihovny redux také můžeme sledovat využití routovacích metod, přesněji metod „routeOf<type>()“ a nad ní metody „openPage()“, nebo „openDialog()“. Metody pro překlady a lokalizace aplikace byly také využity napříč aplikací, zejména ve stránkách s kompletním UI.

5.4 Zbylé části řešení v aplikaci

V příkladové aplikaci byly využity i metodiky usnadňující jazykovou lokalizaci aplikace, nebo přechody mezi jednotlivými stránkami aplikace.

5.4.1 Jazyková lokalizace

Jazyková lokalizace je v celé aplikaci řešena pomocí metod v knihovně flutter_control. Při vývoji ve frameworku samotném je nutné v inicializaci aplikace a vytvoření základní komponenty MaterialApp definovat jednotlivé podporované jazyky a jazykové delegáty používané při vývoji. Jazykové lokalizace je dále možné využívat především pomocí knihovny „intl“, která je na základě zvoleného jazykového kódu země schopna přeložit určité texty nebo formátovat data a čas.

Při použití metod knihovny flutter_control stačí pouze vytvořit jednoduchý key-value json soubor s klíči a jejich požadovanými překlady. Tento soubor je dále nutné referencovat při inicializaci aplikace, aby kontrolní třída ControlRoot věděla, odkud lokalizace brát. Pro lokalizaci daného textu následně stačí pouze zavolat metodu „localize(„key“)“, kde key nahrazuje klíč z překladového souboru.

```
{
  "username": "Přihlašovací jmnéno",
  "password": "Heslo",
  "login": "Přihlásit",
  "sign_up": "Registruj se",
  "email": "Email",
  "invalid_mail": "Neplatný email",
  "invalid_pass": "Neplatné heslo",
  "name": "Jméno",
  "surname": "Příjmení",
```

```
Text(
  localize('login'),
  style: TextStyle(
      color: AppTheme.background_color),
),
```

5.4.2 Routování

Framework Flutter v základu nabízí velkou řadu metod pro navigaci v aplikaci, označení jednotlivých rout a dokonce dekoraci jednotlivých rout pomocí přechodů či vytvoření vlastních rout a jejich přechodů. Pro udržení čistější aplikační struktury je i zde využita knihovna flutter_control a její metody pro navigaci. Nástroje v této knihovně poskytují svým způsobem wrapper nad klasickými flutter navigačními metodami s rozšířenými možnosti ale zjednodušeným ovládáním. Veškeré routy jsou při inicializaci aplikace definovány a uloženy v RouteStore. Samotný model rout, ControlRoute se skládá z routy, přechodu, následného widget builderu a možných nastavení routování. Metody pro volání přechodu můžeme dále rozšířit o přímé vyvolání dialogu nad požadovanou routou, vyvolání modálního okna, specifikaci přechodu, případně přepsat aktuální routu novější nebo požadovanou routu vložit na vrchol navigačního stacku. Samotná metodika je rozšířena i o výše v práci zmiňovaný mixin RouteControlProvider, umožňující přistupovat k navigačním metodám z jakékoliv třídy.

```
routes: [
  ControlRoute.build<DashboardPage>(builder: (_) => DashboardPage()),
  ControlRoute.build<InitPage>(builder: (_) => InitPage()),
  ControlRoute.build<MainPage>(builder: (_) => MainPage()),
```

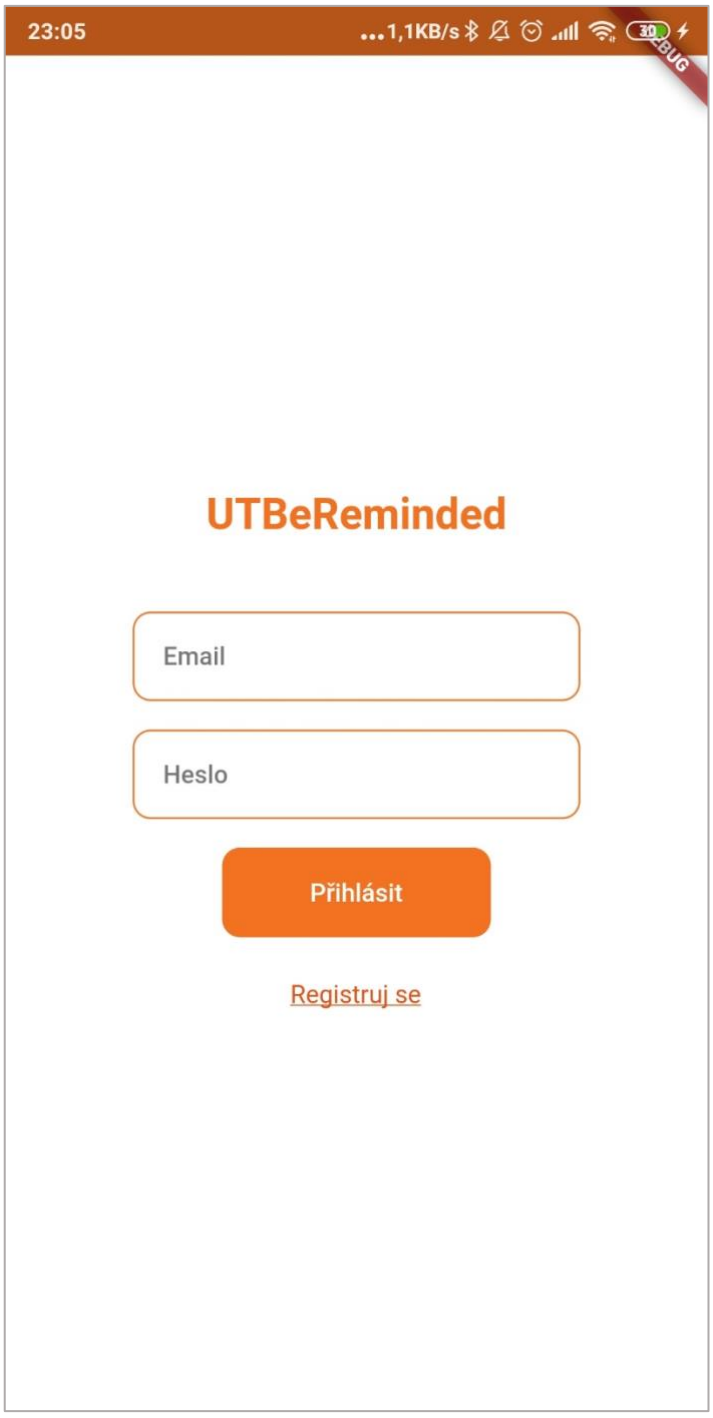
```
void signUp() => routeOf<SignUpPage>().openRoute();
```

5.4.3 Výsledná aplikace

Design výsledné aplikace je inspirován grafickým tématem Univerzity Tomáše Bati a veškeré barvy jsou v souladu s jejím logo manuálem.

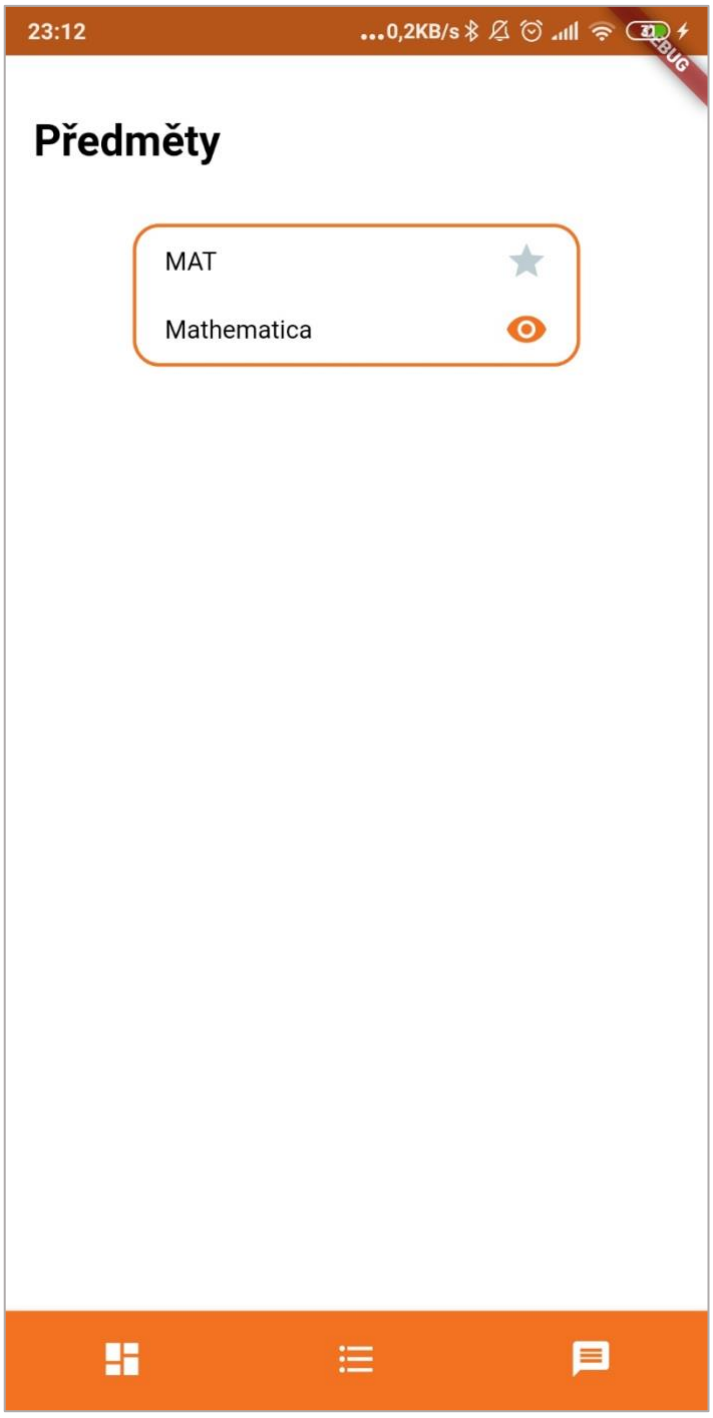
Pro přihlášení aplikace poskytuje jednoduché uživatelské rozhraní, které po úspěšném provedení přihlášení či registrace vpustí uživatele do zbytku aplikace. Ta je rozdělena do tří primárních obrazovek, kdy každá z nich představuje rozdílné funkcionality a zobrazení dat.

Vzhled stránek je řízený právě přihlášeným uživatelem, kdy učitel dostává přístup k rozšířeným funkcionalitám, jako je například vytvoření zprávy pro studenty, kdežto studenti mohou zprávy pouze zobrazovat.



Obrázek 6 - Přihlašovací obrazovka

Obrázek 7 - Hlavní stránka



Obrázek 8 - Stránka s předměty

ZÁVĚR

Cílem této bakalářské práce bylo obeznámit čtenáře s vývojem mobilních aplikací, přiblížit jim jednotlivé možnosti vývoje, jak nativní, tak multiplatformní a poskytnout jim náhled na krátce nový framework s vysokým potenciálem. Samotný popis se skládal z objasnění několika metodik využívaných nejen v mobilním programování, ale také v programování webovém, a to zejména reaktivní programování. Pohled na framework samotný, jeho fungování a základní stavební metodiky dají čtenářům přípravu pro práci s dále definovanými knihovnami usnadňujícími vývoj a zejména kvalitu kódu a jeho znovu použitelnost u komplexnějších aplikací. Jednotlivé knihovny byly popsány a byl podán náhled na jejich základní funckionalitu obohacenou několika názornými ukázkami přímo v praktické aplikaci. Výběr knihovny k vývoji dále zůstává na vývojáři samotném a na jeho preferencích, avšak všechny tyto knihovny vycházejí již z mnoha let vyvíjených knihoven využívaných právě ve webovém programování.

Větší část příkladů byla přikládána knihovně flutter_control, která je dle mého názoru momentálně naprosto bezkonkurenční právě díky její povýšené abstrakci, zjednodušené práci s jejími komponenty. Knihovna také obsahuje vlastní rozhraní pro jazykové lokalizace, navigaci a routování v aplikaci nebo implementaci dependency injection patternu. Díky těmto a mnohým dalším vlastnostem se stává oproti své konkurenci velmi silným a jednoduchým nástrojem, který posouvá programování v tomto frameworku na zcela novou úroveň.

Zásadní výhodou tohoto frameworku a jeho knihoven je také velká obliba a stále rostoucí dosah v programátorských kruzích, což zajišťuje neustálý vývoj a aktualizace jak frameworku samotného, tak jednotlivých knihoven. Tato obliba též podporuje tvorbu mnoha fór zaměřených na tento framework, tvorby tutoriálů, vývojářské soutěže, a především velkou řadu programátorů, kteří jsou schopni velmi rychle a kvalitně podat vysvětlení a řešení těm, kteří si s prací v tomto frameworku nebudou vědět rady a narazí na jakýkoli problém.

Momentálně v tomto frameworku pracuji již rok a půl a využívám ho nejen na malé projekty, ale i na projekty velmi rozsáhlé a zatím jsem byl schopen objevit pouze minoritní problematiku, která by svou složitostí či problematikou znemožňovala využití tohoto frameworku a nedosáhla k vytvoření daného řešení. Veškeré tyto problémy byly však

vyřešeny s aktualizacemi využívaných knihoven či s aktualizací jádra frameworku nebo pouze smysluplnějším vývojem dané problematiky.

SEZNAM OBRÁZKŮ

SEZNAM POUŽITÉ LITERATURY

[1] Mobile Operating System Market Share Worldwide | StatCounter Global Stats. StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share [online]. Copyright © StatCounter 1999 [cit. 03.07.2020]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] Xamarin | Open-source mobile app platform for .NET. .NET | Free. Cross-platform. Open Source. [online]. Dostupné z: https://dotnet.microsoft.com/apps/xamarin

[3] Dart documentation | Dart. Dart programming language | Dart [online]. Dostupné z: https://dart.dev/guides

[4] Flutter documentation - Flutter . Flutter - Beautiful native apps in record time [online]. Dostupné z: https://flutter.dev/docs

[5] Kotlin Programming Language. Kotlin Programming Language [online]. Dostupné z: https://kotlinlang.org/

[6] Android | The platform pushing what's possible. Android | The platform pushing what's possible [online]. Dostupné z: https://www.android.com/

[7] iOS 13 – Apple (CZ). Apple [online]. Copyright © [cit. 27.07.2020]. Dostupné z: https://www.apple.com/cz/ios/ios-13/

[8] New rules following WWDC 2017 - App Store Review Guidelines History. App Store Review Guidelines History [online]. Dostupné z: http://www.appstorereviewguidelineshistory.com/articles/2017-06-08-new-rules-following-wwdc-2017/

[9] SwiftUI: Getting Started | raywenderlich.com. raywenderlich.com | High quality programming tutorials: iOS, Android, Swift, Kotlin, Flutter, Server Side Swift, Unity, and more! [online]. Copyright © 2020 Razeware LLC [cit. 01.08.2020]. Dostupné z: https://www.raywenderlich.com/3715234-swiftui-getting-started

[10] Android versions market share 2019 | Statista. • Statista - The Statistics Portal for Market Data, Market Research and Market Studies [online]. Copyright © Statista 2020 [cit. 03.07.2020]. Dostupné z: https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/

[11] Download Android Studio and SDK tools | Android Developers. Android Developers [online]. Dostupné z: https://developer.android.com/studio

[12] Android Jetpack | Android Developers. Android Developers [online]. Dostupné z: https://developer.android.com/jetpack

[13] Xcode Tutorial for Beginners (Just updated for Xcode 11). CodeWithChris - The Leader in iOS Foundations Training [online]. Copyright © 2020 ByDesign Development Inc. [cit. 26.06.2020]. Dostupné z: https://codewithchris.com/xcode-tutorial/

[14] App Store Review Guidelines. App Store Review Guidelines [online]. www.apple.com: Apple, 2020 [cit. 2020-15-06]. Dostupné z: https://developer.apple.com/app-store/review/guidelines/

[15] TestFlight - Apple. TestFlight - Apple [online]. Copyright © 2020 Apple Inc. All rights reserved. [cit. 12.07.2020]. Dostupné z: https://testflight.apple.com/

[16] Ionic Framework - Ionic Documentation. Ionic - Cross-Platform Mobile App Development [online]. Dostupné z: https://ionicframework.com/docs

[17] About. PhoneGap [online]. Copyright © 2016 Adobe Systems Inc. All rights reserved. [cit. 12.07.2020]. Dostupné z: https://phonegap.com/about/

[18] Introduction · React Native. React Native · A framework for building native apps using React [online]. Copyright © 2020 Facebook Inc. [cit. 03.08.2020]. Dostupné z: https://reactnative.dev/docs/getting-started

[19] ReactiveX - Intro. ReactiveX [online]. Dostupné z: http://reactivex.io/intro.html

[20] rxdart | Dart Package. Dart packages [online]. Dostupné z: https://pub.dev/packages/rxdart

[21] Getting Started with Redux | Redux. Redux - A predictable state container for JavaScript apps. | Redux [online]. Copyright © 2015 [cit. 12.07.2020]. Dostupné z: https://redux.js.org/introduction/getting-started

[22] MobX: Ten minute introduction to MobX and React. MobX [online]. Dostupné z: https://mobx.js.org/getting-started

[23] Flutter - Reactive Programming - Streams - BLoC. Flutter - Didier Boelens [online]. Copyright © 2020 [cit. 22.07.2020]. Dostupné z: https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/

[24] Bloc. Bloc [online]. Dostupné z: https://bloclibrary.dev/#/gettingstarted

[25] flutter_control | Flutter Package. Dart packages [online]. Dostupné z: https://pub.dev/packages/flutter_control

[26] Microsoft acquires GitHub - Stories. Stories | Microsoft news, features, events, and press materials [online]. Dostupné z: https://news.microsoft.com/announcement/microsoft-acquires-github/

[27] Documentation | Firebase. Firebase [online]. Dostupné z: https://firebase.google.com/docs

[28] json_serializable | Dart Package. Dart packages [online]. Dostupné z: https://pub.dev/packages/json_serializable

[29] Hive Docs. Hive Docs [online]. Dostupné z: https://docs.hivedb.dev/

[30] Flutter - Reactive Programming - Streams - BLoC. Flutter - Didier Boelens [online]. Copyright © 2020 [cit. 26.07.2020]. Dostupné z: https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/

SEZNAM PŘÍLOH

Příloha č. 1: CD se zdrojovými kódy a bakalářskou prací