FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Kooperativní mobilní multiplatformní hra
Student:	Jan Bittner
Vedoucí:	Ing. Marek Suchánek
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

Cílem práce je navrhnout a vyvinout multiplatformní mobilní hru, která bude klást důraz na kooperaci více hráčů připojených z různých zařízeních přes síť. Ve hře bude využito specifik moderních mobilních zařízení (například různé typy senzorů a možnosti ovládání). Při práci budou uplatněny tradiční postupy softwarového inženýrství.

- Proveďte průzkum existujících podobných aplikací a trendů v oblasti mobilních her.
- Analyzujte a porovnejte vybrané technologie pro vývoj multiplatformních mobilních aplikací.
- Navrhněte samotnou hru a herní logiku, sestavte požadavky na aplikaci a případy užití.
- Navrhněte architekturu a uživatelské rozhraní aplikace s důrazem na snadnou rozšiřitelnost a použitelnost.
- Implementujte a otestujte aplikaci dle návrhu.
- Vytvořte uživatelskou a vývojářskou dokumentaci.
- Zhodnoťte výslednou aplikaci a popište námět na rozšíření.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 17. října 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Kooperativní mobilní multiplatformní hra

Jan Bittner

Katedra softwarového inženýrství
Vedoucí práce: Ing. Marek Suchánek

30. května 2020

Poděkování

Rád bych na tomto místě poděkoval vedoucímu práce Ing. Markovi Suchánkovi za cenné rady a přátelský přístup. Také bych rád poděkoval své rodině za podporu během studia.

Prohlášení

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

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

V Praze dne 30. května 2020 . . . . . .

Odkaz na tuto práci

Bittner, Jan. *Kooperativní mobilní multiplatformní hra.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020. Dostupný také z WWW: ⟨`https://github.com/tenhobi/bachelors-thesis`⟩.

Abstrakt

Bakalářská práce se zabývá návrhem a implementací kooperativní mobilní hry pro mobilní operační systémy Android a iOS za pomoci multiplatformního frameworku Flutter. Důraz je kladen na rešerši a výběr frameworku, knihovny pro správu stavů, databáze, senzorů, architektury a testování. Dále je v práci na základě poznatků z rešerše provedena analýza, návrh a implementace herních principů, uživatelského rozhraní a architektury aplikace. Výsledkem práce je open-source mobilní hra, ve které hráči vzájemnou komunikací plní cíle misí hry.

Klíčová slova mobilní hra, kooperace, state management, multiplatformní framework, Flutter, BLoC, Clean Architecture

Abstract

The bachelor's thesis describes the design and implementation of the cooperative mobile game for Android and iOS mobile operating systems using the multi-platform Flutter framework. Emphasis is placed on the research and selection of the framework, state management library, database, sensors, architecture, and testing. Furthermore, analysis, designing, and implementation of the game principles, user interface, and application architecture is performed based on the research. The result of this thesis is an open-source mobile game, in which players fulfill the objectives of the game's missions by communicating with each other.

Keywords mobile game, cooperation, state management, multi-platform framework, Flutter, BLoC, Clean Architecture

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Úvod

Poslední desetiletí jsou mobilní telefony nedílnou součástí všedního dne. Telefony používáme pro práci, komunikaci na sociálních sítích i zábavu. Obchody s aplikacemi obsahují obrovské množství aplikací a her, které si může každý stáhnout během několika chvil. Velkou část aplikací lze stáhnout i bezplatně, případně lze bezplatně stáhnout alespoň určitou část hry.

Podle statistik a trendů z herního odvětví ze zdroje [8] lze vidět, že na světě je přes 2,5 miliardy hráčů. 77 % uživatelů mobilních zařízeních jsou hráči, přičemž v roce 2013 to bylo pouhých 52,3 % uživatelů. 76 % lidí preferuje hraní her na mobilních zařízeních, oproti 62% lidí, kteří preferují hraní na PC. Průměrný věk hráčů je 33 let, průměrný věk hráček je 37 let. 92 % her dostupných na obchodě Google Play bylo zdarma ke stažení. [8]

Z těchto trendů vyplývá motivace pro vývoj právě mobilní hry, která je zaměřena především na mladší část publika, je dostupná zdarma a je snadná na pochopení i hraní, což umožní poskytnout hru většímu množství potenciálních hráčů.

Tato práce se zaměřuje na analýzu, návrh, implementaci a testování multiplatformní mobilní herní aplikace s prvky kooperace. Aplikace je určena pro převážně mladší hráče a je navžena dostatečně jednoduše, aby byla nenáročná a tím atraktivnější.

Kapitola 1

Cíle práce

Cílem práce je analýza, návrh a implementace kooperativní multiplatformní mobilní herní aplikace, ve které se hráči připojí do místnosti a společně se snaží splnit mise hry.

Nejprve jsou v rámci teoretické části práce zhodnoceny klady a zápory konkurenčních aplikací, aby bylo zjištěno, kde je vhodné se inspirovat, a čemu se naopak vyvarovat. Dalším úkolem je získání přehledu o moderních technologiích a provedení rešerše možností, jednotlivě je prozkoumat a popsat, a na závěr každé technologie zhodnotit nejvhodnější výběr. Neméně důležitý je také úkol samotného navrhnutí principů a logiky hry, jejích funkcí, požadavků na aplikaci a případů užití.

Praktická část má za úkol vhodně navrhnout architekturu aplikace a uživatelské rozhraní a následně provést implementaci řešení tak, aby byla aplikace plynulá, udržitelná a snadno rozšiřitelná. Dalším úkolem je také aplikaci řádně otestovat automatizovanými testy a provést uživatelské testování.

V neposlední řadě je úkolem také vytvoření uživatelské a vývojářské dokumentace.

Kapitola 2

Konkurenční aplikace

Po prozkoumání podobného žánru her (kooperace, komunikace a rychlost reakcí) dostupných v obchodě Google Play, byly vybrány konkurenční herní aplikace. Tyto hry byly vybrány zejména podle popularity, počtu stažení aplikace v obchodě a podle subjektivně vnímaného potenciálu, který má daná hra k inspiraci pro tvorbu herní aplikace.

V následujících podkapitolách jsou popsány vybrané čtyři konkurenční aplikace. U každé aplikace jsou také popsány klady a zápory.

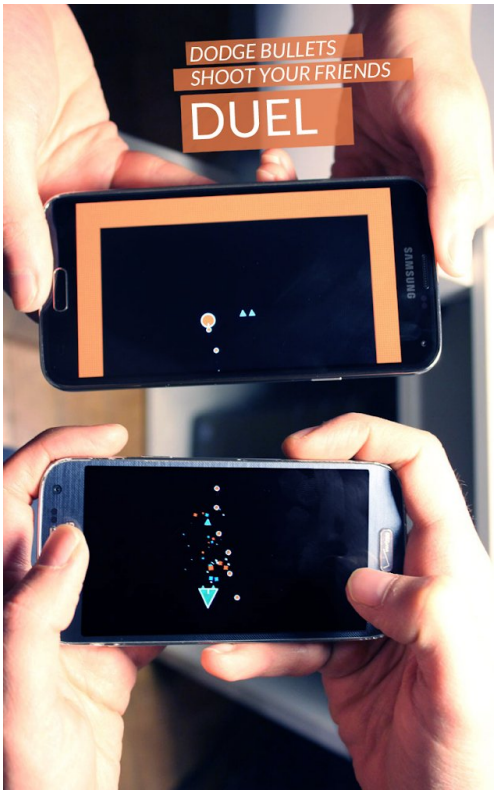
2.1 DUAL!

První vybranou hrou je *DUAL!*. Tato hra využívá zejména prvky kooperace a rychlosti reakcí a je velmi akčně laděná. Jak říká první věta stránky hry v obchodě Google Play: *„Between people, across screens.“* [9] Společná hra se odehrává na zařízeních více hráčů.

Hra má několik módů. Jedním z módů je mód *DUEL*, ve kterém dva hráči, každý na svém zařízení, soupeří proti sobě. Hra obsahuje další dva módy, které však nejsou přístupné zdarma, proto zde nebudou zohledněny a popsány.

Herní plocha je rozdělena do dvou 2D částí. Každý hráč se může pohybovat pouze ve své části, na svém zařízení. Herní postava hráče se ovládá natáčením jeho displeje. Postava má zásobník střel a po klepnutí na plochu postava vystřelí. Při podržení se postupně ze zásobníku nabíjí střely a vystřelí se větší, ale trochu pomalejší, střela. Rychlým použitím gesta švihnutí se postava otočí. Pokud hráč vystřelí na své ploše, střela za chvíli doletí do protihráčovy části. Střele protihráče se dá takticky vyhnout, nebo zneškodnit vystřelením vlastní střely.

Cílem hry je opakovaně zasáhnout protihráče a tím ho zneškodnit. Po ukončení hry je možné v dané relaci, s protihráčem, začít další kolo hry.



Obrázek 2.1: Screenshot hry DUAL! [9]

Klady

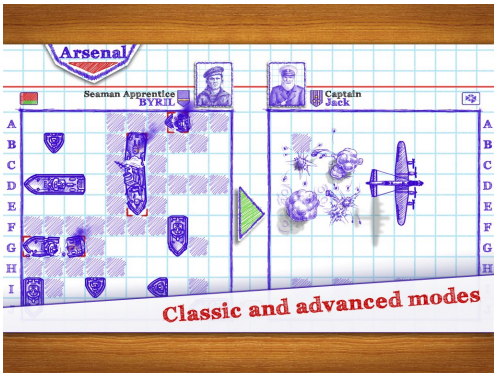
- UI je velmi jednoduché, avšak přívětivé svými barvami a srozumitelností.
- Hra je velmi krátká a dá se mnohokrát opakovat.

Zápory

- K dispozici zdarma je pouze jeden herní mód.
- Nejsou dostupné žádné speciální funkce, které by hru obohatily.

2.2 Sea Battle 2

Hra *Sea Battle 2* sice není postavena na kooperaci a komunikaci mezi hráči, ale představuje hezkou reprezentaci klasické stolní hry v moderním pojetí. Hráči z celého světa mezi sebou na svých zařízeních vedou souboje a za každé vítězství postupně postupují v námořních hodnostech. [10]



Obrázek 2.2: Screenshot hry Sea Battle 2 [10]

Herní plocha obsahuje 10×10 políček, kam si hráč rozloží své lodě. Vedle toho jsou postupně vyobrazována políčka protihráče. UI je laděno do podoby čtverečkovaného papíru s ručně kreslenými prvky.

Cílem hry, stejně jako v klasické stolní hře, je zneškodnit všechny protivníkovy lodě. Toho lze docílit klasicky pomocí bombardování, nebo v pokročilém módu pomocí nejrůznějších letadel a pokročilých děl. Pokročilý mód rozšiřuje hru o nové strategie, čímž činí hru zajímavější.

Klady

- Známý koncept hry.
- Obohacení hry s pokročilým módem o nové prvky.

Zápory

- UI nepůsobí přívětivě.
- UI nelze přibližovat, což způsobuje problémy s dotyky na jednotlivá políčka.

2.3 Keep Talking and Nobody Explodes

Ve hře *Keep Talking and Nobody Explodes* závisí všechno na správné kooperaci a komunikaci. Tato velmi oceňovaná hra existuje ve verzi pro desktop, konzole, virtuální realitu i mobily. [11]

Hra má předpřipravená herní kola s daným typem bomby. Bomba je koncipována jako kufřík s několika moduly a časomírou, která se nebezpečně rychle odpočítává. Jeden hráč vidí bombu. Další hráč, případně skupina hráčů, mají k dispozici manuál s instrukcemi. Manuál mohou prohlížet jak v elektronické, tak i v papírové podobě. Manuál je totiž pro všechny hry shodný.

Obrázek 2.3: Screenshot hry Keep Talking and Nobody Explodes [11]

Veškerá komunikace probíhá pouze slovně. Hráč s bombou tak nahlásí hráči s manuálem například to, že vidí červené tlačítko s textem „defuse“. Druhý hráč pak v manuálu vyhledá daný modul tlačítka, řídí se instrukcemi a podstatné informace k zneškodnění předá prvnímu hráči.

Cíl každého kola je zneškodnění bomby. Důraz je také kladen na čas zneškodnění a počet udělaných chyb. Po skončení kola může hráč porovnat svůj výsledek s ostatními hráči v žebříčku.

Klady

- UI je laděné jako pohled na stůl, na kterém je bomba a časomíra, což dodává hře atmostféru a vtáhne hráče do děje.
- Světla v místnosti blikají, případně se náhodně vypínají a zapínají. To přidává na efektu zejména v situacích, kdy zbývá už jen pár sekund a bomba není dobře vidět.
- Přestože je grafika jednodušší, samotná bomba je velmi přehledná.
- Manuál je stejný pro všechna herní kola. Lze jej tedy i vytisknout.

Zápory

- U složitějších modulů je manuál občas hůře pochopitelný.

Obrázek 2.4: Screenshot hry Spaceteam [12]

2.4 Spaceteam

Poslední hra, *Spaceteam*, si zakládá na rychlosti reakcí a velmi rychlé komunikaci [12]. To shrnuje první věta na stránce hry v obchodě Google Play: *„Mačkáš rád talčítka a řveš na své kamarády? Jo? A i kdyby ne, zkus se na chvíli stát členem vesmírné posádky a zažij, co to znamená být Spaceteam.“* [12]

Herní plocha je plná náhodných tlačítek, přepínačů, posouvátek a dalších věcí, které dohromady tvoří ovládací prvky hry. Každému hráči přichází postupně do zařízení série příkazů, které cílí na určitou interakci s ovládacími prvky.

Hra je rozdělena na herní kola. Každé herní kolo má vlastní množinu ovládacích prvků a příkazů.

Složitost komunikace spočívá v tom, že příkazy se netýkají pouze prvků na daném zařízení, ale i prvků ze zařízení ostatních hráčů. Některé příkazy také cílí na všechny hráče. Příkladem takového příkazu je například zatřesení všemi zařízeními najednou.

Každý chybný krok je penalizován. Penalizace se postupně projevuje například zhoršením přístupnosti k ovládacím prvkům. Cílem hry je dostat se přes co nejvíce herních kol. Po skončení hry následuje shrnující statistika a ocenění jednotlivých hráčů.

Klady

- UI úvodní obrazovky a přihlašování do hry vypadá velmi efektně.
- Tlačítka i příkazy obsahují řadu technologických pojmů, což dodává hře na správné atmosféře.
- Finální ocenění hráčů působí vřelým dojmem.

Zápory

- Herní obrazovka má poněkud zastaralé, až nevkusné prvky.
- Hra probíhá až příliš rychle. Nutno přiznat, že tento fakt není nutně nevýhoda, avšak ubírá to na prvku kvalitnější komunikace.
- Hra nemá možnost přihlášení, a tedy shromažďování a uchovávání statistik.

2.5 Zhodnocení

Aplikace vyvíjená v rámci praktické práce je koncepčně podobná ke hrám *Spaceteam* a *Keep Talking and Nobody Explodes*. Obě tyto hry pracují s variací modulů, které musí být nějakým způsobem splněny. Zbylé dvě hry naopak pracují s jiným stylem hry, avšak také zachovávají prvky kooperace a komunikace. Ze všech her je vhodné se inspirovat jednoduchým UI a krátkými hrami, například v podobě misí, jelikož jde vidět, že kratší hry mají velký úspěch, zřejmě proto, že je lze hrát kdykoli a kdekoli. Zároveň je velkým kladem všech zmíněných her příběh a atmosféra, zatímco se zdá, že hezké UI nehraje příliš velkou roli a je tedy vhodnější zaměřit se na hratelnost samotnou. Nevýhodou hry *Spaceteam* je nemožnost přihlášení a udržování statistik napříč zařízeními. Cílem je využít zmíněné klady a zápory a vhodně je zakomponovat do vyvíjené aplikace.

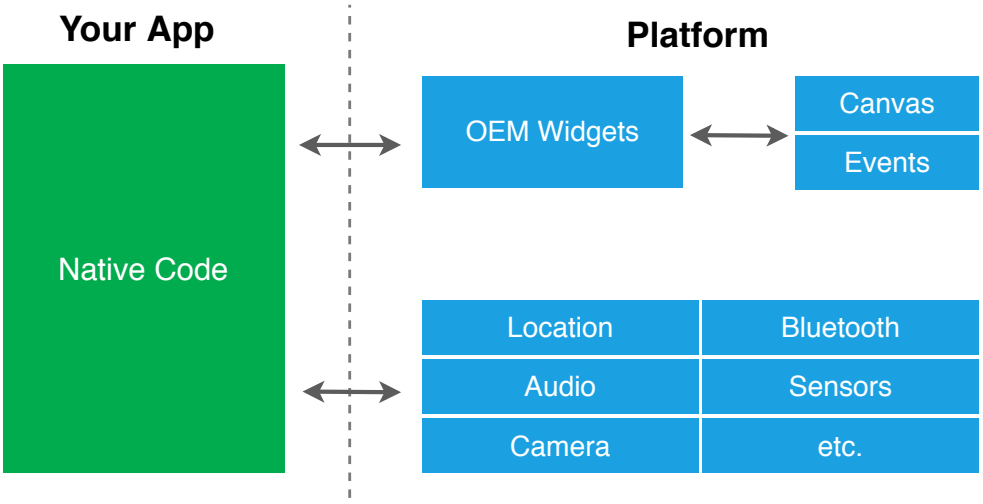
Kapitola 3

Rešerše technologií

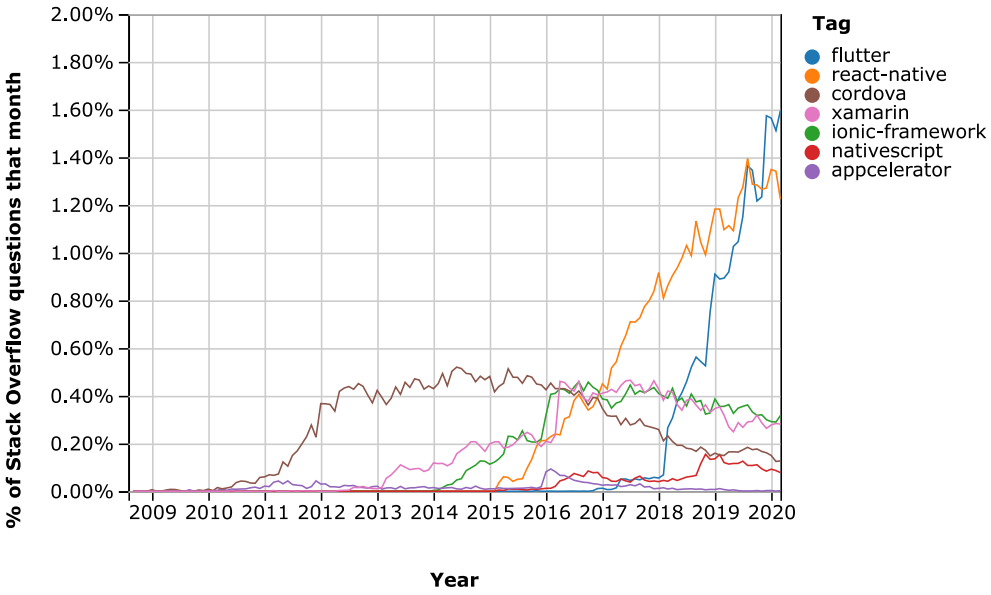
Tato kapitola je věnována rešerši jednotlivých skupin technologií, které budou použity v rámci praktické části. V každé podkapitole jsou shrnuty možnosti, klady a zápory jednotlivých technologií a rovněž je provedeno celkové zhodnocení, v němž je odůvodněný výběr dané technologie pro praktickou část práce.

3.1 Multiplatformní framework

Vývoj mobilních aplikací je dle [8] na vzestupu. To přináší nároky na vydavatele a vývojáře k rychlejší tvorbě mobilních aplikací a her. Aplikace pro mobilní zařízení se běžně vytvářejí pomocí programovacích jazyků,



Obrázek 3.1: Schéma frameworku specifického pro platformu [13]

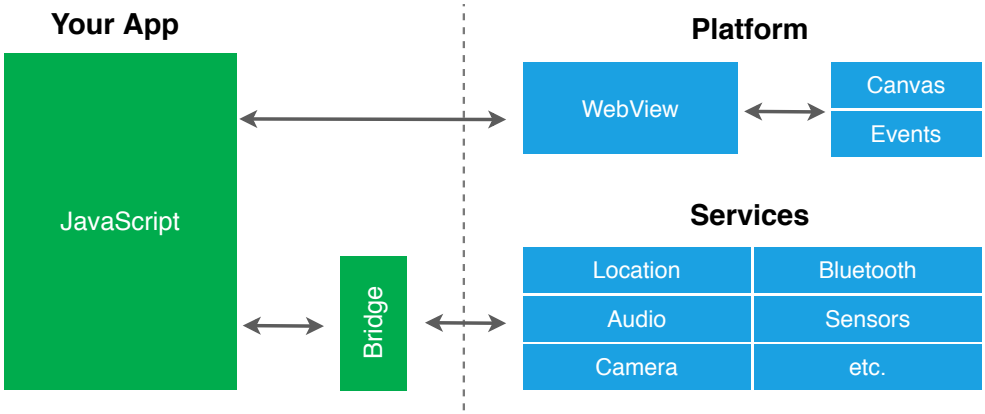


Obrázek 3.2: Vývoj trendu multiplatformních frameworků [15]

které jsou úzce spjaty s mobilními zařízeními. Příkladem takového programovacího jazyka je jazyk Java, který se používá pro vývoj nativních aplikací v platformě Android. Tyto jazyky mají úzkou podporu pro vývoj na daných zařízeních, včetně přístupu k hardwarovému rozhraní (kamera, senzory, ...) [14].

Snahu urychlit vývoj mobilních aplikací řeší také mobilní frameworky. Tyto frameworky jsou tvořeny většinou stejnou společností, která také vytváří danou platformu, a jsou publikovány jako *Software Development Kit* (dále jen SDK). Pro platformu Android je vyvíjeno Android SDK, spolu s podpůrnými programy jako integrované vývojové prostředí (dále jen IDE) a emulátory. Tyto frameworky jsou díky jejich úzkému propojení a plné kompatabilitě považovány za rychlé, avšak mnohdy vyžadují speciální znalosti pro konkrétní platformu. Právě cílení výhradně na danou platformu je největší nevýhoda těchto typů frameworků. Na obrázku 3.1 lze vidět architekturu těchto frameworků. [14]

Další snahou pro ještě rychlejší a snadnější vývoj aplikací jsou frameworky s podporou multiplatformnosti, a tedy frameworky cílící na více platforem. To může být pouze cílení na mobilní platformy, ale dokonce také i cílení na web, osobní počítače a další. Tyto frameworky podporují populární jazyky, umožňují sdílet kód mezi jednotlivými platformami a v závislosti na architektuře se blíží rychlostí výsledných aplikací k aplikacím psaných nativními jazyky [13]. Pomocí těchto technologií lze tedy vyvíjet aplikace pohodlně, rychle a benefitem „zadarmo“ je podpora multiplatformnosti vyvíjené aplikace [14].



Obrázek 3.3: Schéma architektury s WebView [13]

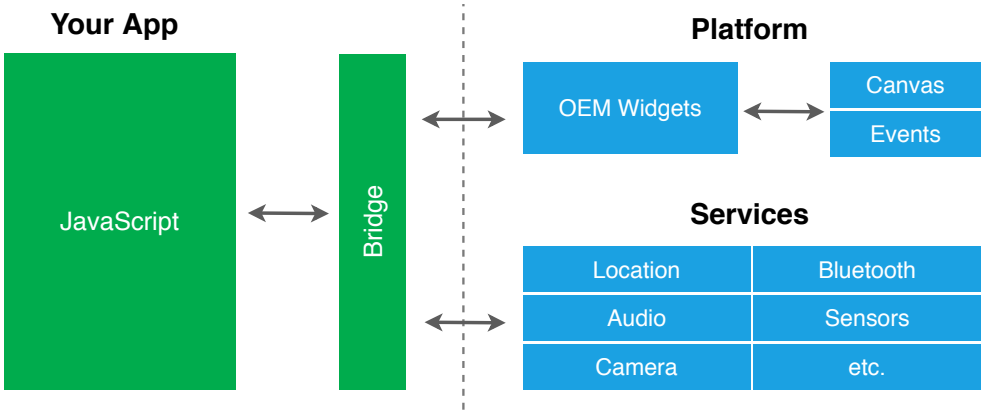
V následujících sekcích je vybráno a popsáno několik multiplatformních frameworků. Tyto frameworky byly vybrány na základě trendů [15] na síti Stack Overflow. Na obrázku 3.2 lze vidět reprezentace těchto trendů, ve kterých lze vyčíst, jak se trend pro daný jazyk vyvíjí v čase. Nejvíce populární na této síti je tedy framework Flutter, který překonal popularitu frameworku React Native, avšak oba frameworky si udržují dlouhodobý růst. Ostatní frameworky si trend udržují, nebo jim dokonce klesá.

3.1.1 Ionic

První multiplatformní frameworky začaly pro multiplatformnost využívat WebView. Tyto frameworky aplikaci vykreslují pomocí webových technologií a komunikaci s hardwarovým rozhraním řeší pomocí speciálního komunikačního mostu [13]. Taková architektura je vidět na obrázku 3.3. Z tohoto obrázku jde vidět, jak moc je zdrojový kód oddělený od dané platformy.

Jedním z představitelů těchto frameworků je i framework Ionic, který je stále značně využívaný i v dnešní době, dle trendů z obrázku 3.2. Framework Ionic využívá jazyky jako HTML, CSS a JavaScript. V tomto frameworku je možné vytvořit aplikaci pro mobilní platformy Android a iOS. Významnou výhodou je, že díky využití mobilních technologií, které se používají naprosto shodně jako při vývoji webových aplikací, se nemusí vývojáři učit nové technologie a přístupy k vývoji, čímž se také celý proces zrychlí. Důsledkem používání webových technologií jsou ale i jejich nevýhody, zejména nižší výkon. [14]

S frameworkem Ionic lze využít také frontend knihovny a frameworky. Příkladem jsou knihovny a frameworky React, Angular či Vue. Součástí je také knihovna komponent UI, které se automaticky přizpůsobí dané platformě. Součástí jsou také podpůrné konzolové nástroje pro vytváření, produkování a testování aplikací. [16]



Obrázek 3.4: Schéma architektury frameworku React Native [13]

3.1.2 React Native

Další přístup k vývoji multiplatformních aplikací představuje framework React Native. Tento framework využívá pomocí komunikačního mostu nativní komponenty platformy díky vzorům z reaktivního programování zjednodušuje celkový vývoj [13]. Framework je založen na populární webové knihovně ReactJs. Jelikož framework nepoužívá nativní komponenty platformy, a tedy ani WebView, jsou aplikace produkované tímto frameworkem rychlejší. Pro popis UI je používán vlastní jazyk založený na XML, nazývaný JSX [14].

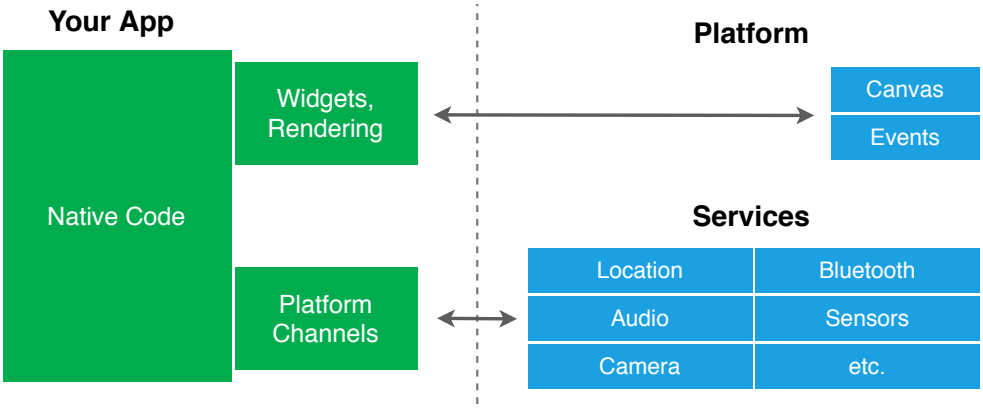
Framework React Native obsahuje *hot reloading*, což je vlastnost, která umožní vidět všechny změny ihned po jejich změně, bez zbytečně zdlouhavého procesu sestavování celé aplikace. K dispozici je také velká škála komponent, včetně velkého množství komponent od vývojářů třetí strany, které mohou vývojáři volně používat. [14]

Framework byl vytvořen v roce 2015 společností Facebook [13], která jej od té doby spravuje. Za zmínku stojí populární aplikace, které framework využívají, mezi nimi jsou například mobilní aplikace Facebook, Instagram, Skype, Discord a mnoho dalších [17].

Na obrázku 3.4 lze vidět schéma architektury frameworku React Native. Jak lze z obrázku vyčíst, tato architektura, oproti architektuře s WebView, komunikuje s platformou kompletně přes komunikační most pomocí jazyku JavaScript.

3.1.3 Flutter

Posledním frameworkem je Flutter, který je vyvíjený společností Google. Tento framework je nejmladším z představených frameworků, avšak dle trendů z obrázku 3.2, kde je vidět razantní nárůst, se jim vyrovná.



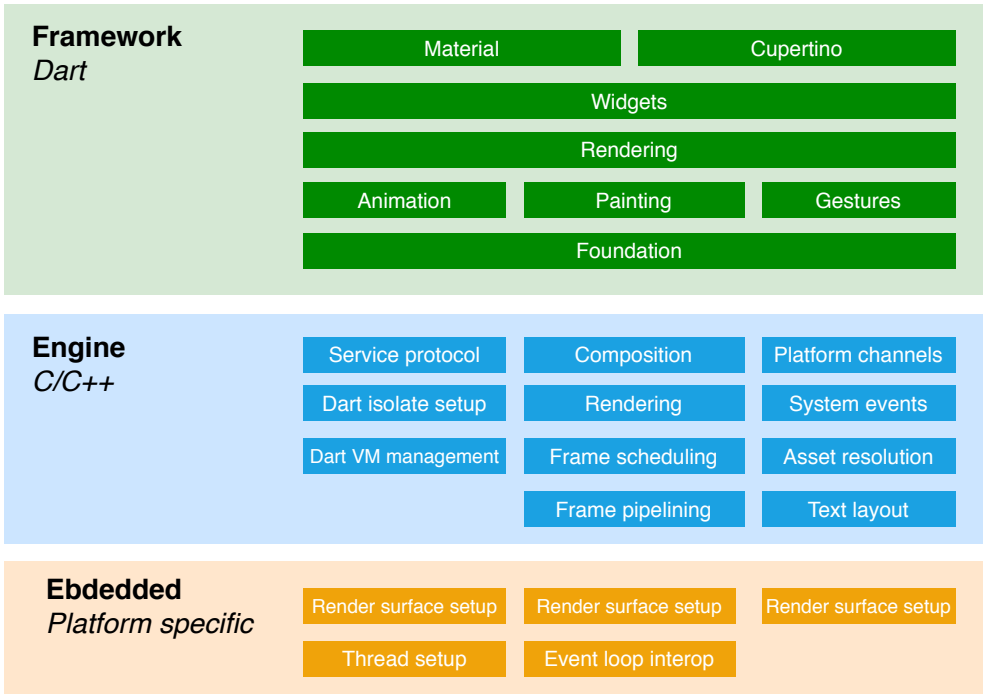
Obrázek 3.5: Schéma architektury frameworku Flutter [13]

Framework Flutter přistupuje k multiplatformnosti jinak, než ostatní mobilní frameworky. Nejenže tento framework nevyužívá programovací jazyk JavaScript, což samo o sobě přináší řadu výhod, ale také přesouvá starost s vykreslováním z platformy na framework [13], jak lze vidět na obrázku s reprezentací architektury 3.5, a také se plně kompiluje do nativního kódu [14]. Podporované nejsou pouze mobilní platformy Android a iOS, ale také platformy web a desktop [18]. Platforma web je momentálně dostupná ve verzi beta, platforma desktop je dostupná ve verzi alfa [19, 20].

Framework Flutter využívá programovací jazyk Dart, který umožňuje jak kompilaci *ahead of time* (dále jen AOT), tak i kompilaci *just in time* (dále jen JIT) [13]. Těmito schopnostmi kompilace do nativního ARM kódu je framework schopen nemít žádný komunikační most mezi platformou a procesorem, což umožňuje tvořit rychlejší a výkonější aplikace. Aplikace mohou dosahovat až 120 snímků za vteřinu [14]. Díky možnostem kompilace JIT je také framework schopen poskytnout *hot reload*, který umožní velmi rychle provádět změny stavů aplikace zatímco je spuštěna. Vydání aplikace jsou vždy kompilovány naopak AOT kompilací, která produkuje velmi rychlou a výkonnou aplikaci [13].

Přesto, že je framework Flutter relativně nový, existuje řada aplikací, které tento framework úspěšně využívají. Za zmínku stojí například aplikace Google Ads, Baidu, Hamilton a Reflecty [18].

Tento framework využívá sice widgety (komponenty UI), které vypadají jako nativní komponenty pro danou platformu, avšak všechny tyto widgety implementuje a udržuje pro vlastní vykreslovací engine Skia, taktéž od společnosti Google [18]. Engine Skia je známý 2D grafický engine, který je využíván například v projektech jako Google Chrome [21].

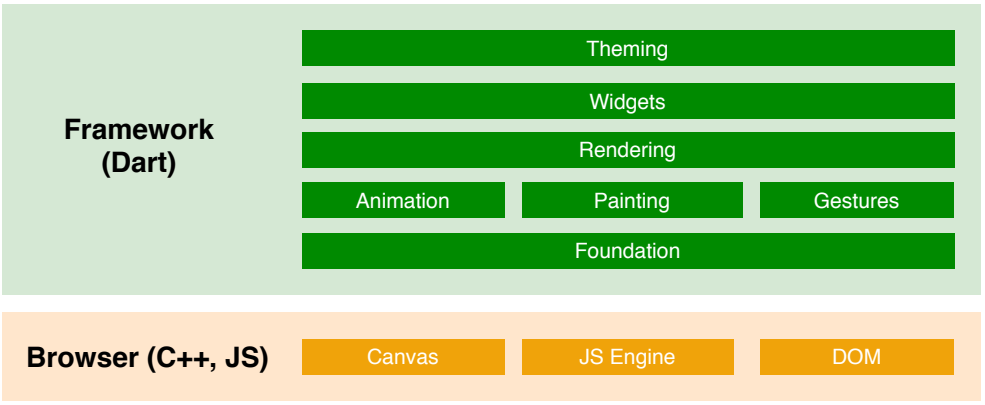


Obrázek 3.6: Vrstvy frameworku Flutter [22]

Princip frameworku

Framework Flutter je ve svém principu velmi jednoduchý. Jak je uvedeno v dokumentaci frameworku Flutter: *„Everything's a widget.“* [22] Oproti ostatním frameworkům, které dělí aplikaci do několika vrstev, obsahuje framework Flutter widgety jako základní stavební bloky pro všechno. Widgety jsou kompozicí spojovány a společně utváří celek, aplikaci. Každý widget zdědí vlastnosti od jeho rodiče v podobě kontextu, se kterým může dále samotný widget pracovat. [22]

Pro skladbu widgetů je upřednostňována kompozice před rozšířením dědičností. Vhodnou skladbou widgetů a jejich kombinací jsou prvky aplikace uspořádány, ostylovány, transformovány či jiným způsobem upraveny. Například widget `Padding` přidá okolo widgetu odsazení, widget `Row` uspořádá widgety–potomky do řádku, zatímco widget `Column` uspořádá widgety–potomky do sloupce a widget `Stack` uspořádá widgety–potomky přes sebe. [22]



Obrázek 3.7: Vrstvy frameworku Flutter pro platformu web [19]

Na obrázku 3.6 lze vidět uspořádání frameworku do vrstev architektury. Vrstva frameworku samotného je na úrovni widgetů, které píší vývojáři aplikací v jazyce Dart. Vrstva enginu ovládá framework na nižší urovni, kompozici widgetů, jejich vykreslování a podobně. Engine je vyvíjen v programovacích jazycích C a C++. A poslední vrstva je zaměřena na požadavky specifické platformy. [22]

Pokud se widget potřebuje měnit na základě dynamických faktorů, jako je například interakce uživatele nebo tok dat, je tento widget označován jako *stateful* (widget se stavem) a je reprezentovaný třídou `StatefulWidget`. V opačném případě je widget označován jako *stateless* a je reprezentovaný třídou `StatelessWidget`, která má stav neměnný. Každý stateful widget má proměnlivý stav `State` a kdykoli je stav změněn, je potřeba zavolat metodu `setState()`, která signalizuje frameworku, že je widget nutno překreslit s novým stavem. [22]

Další platformy

Framework Flutter momentálně vyvíjí také podporu pro další platformy jako jsou platformy web a desktop. Tyto platformy jsou zatím podporovány pro produkci, avšak dle obrázku 3.7 lze vidět, že framework Flutter je navržen vhodně k tomu, aby tato a další úsilí podpořil. Spodní specifické vrstvy platformy jsou vyměněny za vykreslování pomocí webových technologií, což v budoucnu umožní vytvářet aplikace nejen pro platformy Android a iOS, ale i již zmíněný web a desktop. [19, 20]

3.1.4 Zhodnocení

Jak je již vidět z obrázku 3.2, nejpopulárnější multiplatformní frameworky jsou frameworky Flutter a React Native. Přičemž framework Flutter je v porovnání s frameworkem React Native relativně stále nový. I přes to má ale dlouhodobě strmější růst v trendu.

Trend frameworku Ionic dlouhodobě stagnuje, až klesá, a navíc kvůli již zmiňovaným nevýhodám, zejména díky architektuře, není až tolik vhodný pro vývoj nových aplikací, s předpokladem pro dlouhodobější udržitelnost.

Framework React Native je velmi populární a nabízí řadu výhod. Tento framework je však moc spjat s webovými technologiemi a je nutné využívat komunikační most, což přináší menší rychlostní i výkonostní nevýhody. Framework Flutter oproti tomu přináší spoustu výhod, a jak lze vidět z obrázku 3.5, tak také zaručuje svou architekturou stejné zobrazení aplikace na všech verzích operačních systémů dané platformy, díky vlastnímu vykreslování. Oba frameworky jsou zajisté vhodné pro vývoj nové aplikace s důrazem na udržitelnost a rozšiřitelnost, avšak framework Flutter převyšuje svými výhodami, a proto bude v praktické části práce využit právě framework Flutter.

3.2 Správa stavů

Používání proměnných či funkcí pro řízení vykreslování aplikace může být snadné a logické řešení pro menší aplikace, avšak pro větší aplikace — a zejména aplikace, které dbají na udržitelnost a rozšiřitelnost — je vhodné využít některé z řešení pro správu stavů dílčích částí aplikace, známé především jako *state management*.

Stavy nám umožňují přemýšlet o UI ve Flutteru deklarativně. Ve Flutteru platí, že UI je výsledkem `build` metody, která pracuje s aktuálním stavem aplikace. Pro změnu barvy widgetu se tak nevolá funkce jako `widget.setColor`, ale změní se stav a podle něj se příslušně překreslí UI. [23]

Pojem *stav* můžeme chápat více způsoby. V této práci je stav chápán jako všechna data, která jsou potřeba k překreslení UI v daný moment. Dle [24] lze stavy dělit do dvou typů:

Ephemeral state Někdy také nazývaný *UI state* nebo *local state*, je typ stavu využívaný většinou pro dílčí část UI a ostatní widgety jen zřídka potřebují přístup k těmto stavům. Příkladem je aktivní stránka ve widgetu `PageView`, index zvolené záložky `BottomNavigationBar`, aktuální stav animace a podobně. Tento typ je většinou řešen jako třída `State` pro daný `StatefulWidget` pomocí vlastností třídy a metody `setState()` pro aktualizaci stavu.

App state Někdy také nazývaný *shared state* je typ stavu sdílený mezi několika částmi aplikace. Příkladem je uživatelské nastavení, notifikace, nákupní košík, informace o přihlášení a podobně.

```
@override
Widget build(BuildContext context) {
    return SomeWidget(
        MyListItem(myData, myTapCallback),
    );
}

void myTapCallback(Item item) {
    print('user tapped on $item');
}
```

Výpis kódu 3.1: Manipulace se stavem pomocí předávání dat a callbacku [1]

Neexistuje přesné pravidlo, jak určit typ stavu a spolu s tím jak implementovat daný stav v aplikaci. Všechny stavy mohou být implementovány pomocí `State` a `setState()`, a pro mnoho malých aplikací to může být nejjednodušší řešení. Stejně tak lze všechny stavy implementovat jako App state. [24]

Následující sekce obsahují popis možností pro práci s App state. Každá sekce uvede danou možnost a popíše klíčové vlastnosti. Na závěr je provedeno porovnání a zhodnocení.

3.2.1 Předávání dat a callbacků

Ve Flutteru je samozřejmě možné mezi widgety předávat data konstruktory a zpracovávat reakce pomocí callbacků. Pomocí callbacků lze interagovat se stavem, který se přesune do nejvýše postaveného widgetu ve stromu widgetů. Důvod je ten, že ve Flutteru, který je deklarativní, je nutné pro překreslení UI spustit `build` [1]. To je zajisté funkční a snadné řešení, avšak mnohokrát je nutné data a callbacky předávat přes více widgetů, čímž se aplikace velmi rychle stane nelehce rozšiřitelnou.

Jak lze vidět v ukázce 3.1, vnořenému widgetu jsou předány data a pro změnu stavu z vnořeného widgetu je tomuto widgetu předán callback. Ten se využije pro interakce se stavem.

Flutter má způsoby, jak mohou widgety sdílet stavy bez toho, aniž by se snižovala rozšiřitelnost aplikace pomocí nepřehledného předávání dat a callbacků. Stejně jako vše ve Flutteru, i toto je řešeno pomocí widgetů. Za zmínku stojí `InheritedWidget`, na který lze jednoduše odkázat odkudkoli z jeho podstromu pomocí `MyInheritedWidget.of(context)`. Tento widget a další funkcionality zaobaluje knihovna Provider, která zjednodušuje použití a snižuje množství potřebného kódu. [1]

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

part 'counter.g.dart';

class Counter = CounterBase with _$Counter;

abstract class CounterBase with Store {
    @observable
    int value = 0;

    @action
    void increment() {
        value++;
    }
}
```

Výpis kódu 3.2: Ukázka kódu počítadla v knihovně MobX [2]

3.2.2 MobX

MobX je knihovna pro správu stavů, která implementuje vzor observer. Díky jednoduchému rozhraní je vhodná pro začátečníky a je používána i ve velkých projektech. Dle [2] pracuje knihovna se třemi základními koncepty:

Observables Reprezentují stavy v podobě libovolného objektu s daty. Stavy mohou být i odvozené na základě jiných odvozených a základních stavů. Každá změna stavu vyvolá upozornění pozorovatelů, a tedy jejich reakce.

Actions Popisují změny stavů. Stavy se nemění přímo, ale pomocí akcí, které změnám přidávají sémantický význam. Namísto přímého volání `value++` se volá akce `increment()`. Akce vyvolává změnu stavu.

Reactions Automaticky sledují změnu stavů. Neprodleně po každé změně se provede reakce. Reakce vyvolávají akce.

Z výpisu kódu 3.2 lze vypozorovat, že knihovna pracuje s generovanými kódy z pomocných generátorů. Tím se dle [2] snaží redukovat množství kódu, které by bylo jinak vyžadováno napsat k docílení stejné funkčnosti. Knihovna také využívá anotace, kterými je kód zpřehledněn. Generování kódu však není nejrychlejší, což je jedna z nevýhod tohoto přístupu.

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

enum CounterActions {
    increment
}

class AppState {
    int value;

    AppState({
        this.value = 0,
    });
}

AppState counterReducer(AppState state, action) {
    switch (action) {
        case CounterActions.increment:
            return AppState(
                value: state.value++,
            );
        default:
            return state;
    }
}
```

Výpis kódu 3.3: Ukázka kódu počítadla v knihovně Redux [3]

3.2.3 Redux

Jedna z nejznámějších knihoven řešící správu stavů je knihovna Redux. Dle [3] pracuje Redux se třemi základními koncepty:

Actions Obsahují data, která se posílají z aplikace do *Store* a jsou jediným zdrojem informací.

Reducers Popisují, jak se změní stav aplikace v závislosti na akci.

Store Pojí předešlé dva koncepty dohromady. Drží stav aplikace, určuje počáteční stav a poskytuje aktuální stav.

Dle [3] knihovna Redux obsahuje právě jeden *Store* a členění se provádí pomocí kompozice *Reducers*. Z výpisu kódu 3.3 je vidět, že ačkoli je zápis přehledný, ve větší aplikaci nemusí být omezení pouze jednoho *Store* vhodné.

3.2.4 BLoC

Oproti předchozím příkladům není Business Logic Component (dále jen BLoC) knihovna, ale návrhový vzor. Tento vzor je tvořen několika jednoduchými pravidly, kterými se každá implementace musí řídit, přičemž konkrétní prvky implementace jsou přenechány na vývojáře. Pojmem BLoC také označujeme třídu, která uchovává stav a dodržuje daný vzor [25]. Dle [25] těmito pravidly pro návrh jsou:

1. Inputs and outputs are simple `Stream`/`Sink` only.
2. Dependencies must be injectable and platform agnostic.
3. No platform branching allowed.
4. Implementation can be whatever you want if you follow the previous rules.

Spolu s těmito pravidly pro vzor BLoC byla představena pravidla i pro návrh UI. Ta se zabývají zejména vztahem mezi komponentou UI, komponentou BLoC a podobou dat, které se posílají do a z každé třídy BLoC [25]. Dle [25] těmito pravidly jsou:

1. Each “complex enough” component has a corresponding BLoC.
2. Components should send inputs “as is”.
3. Components should show outputs as close as possible to “as is”.
4. All branching should be based on simple BLoC boolean outputs.

Tento návrhový vzor vznikl za účelem oddělení kódu s logikou od závislostí na konkrétní platformě [25]. Oddělením vrstvy UI a logiky má vzor za cíl tvořit kód více udržitelný a testovatelný [26]. Oproti knihovně Redux si lze z uvedených pravidel všimnout, že už samotný návrhový vzor BLoC pracuje s více zdroji pro uchování stavu.

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

enum CounterEvent { increment }

class CounterBloc extends Bloc<CounterEvent, int> {
    @override
    int get initialState => 0;

    @override
    Stream<int> mapEventToState(CounterEvent event) async* {
        switch (event) {
            case CounterEvent.increment:
                yield state + 1;
                break;
        }
    }
}
```

Výpis kódu 3.4: Ukázka kódu počítadla v knihovně Bloc [4]

3.2.4.1 Knihovna Bloc

Jednou z implementací tohoto vzoru je knihovna Bloc od Felixe Angelova. Tato implementace se zaměřuje na potřeby vývojářů, například zpřístupnění informací o stavech aplikace v daný moment, snadné testování, znovupoužitelnost, rychlý vývoj a mnoho dalšího [27]. Dle zdroje [4] pracuje knihovna Bloc se čtyřmi základními koncepty:

Events Reprezentují události v podobě libovolného objektu a jsou jediným vstupem do *Bloc*. Tyto objekty jsou většinou vytvářeny dle potřeb UI.

States Představují stavy, jednotlivých částí aplikace, taktéž v podobě libovolného objektu. S těmito objekty následně pracuje UI.

Transitions Popisují změnu z jednoho stavu do jiného. Transakce se skládají z aktuálního stavu, události a následujícího stavu. Sekvence těchto záznamů tedy tvoří historii změn stavů.

Blocs Reprezentují třídy, které pojí předešlé koncepty dohromady. Tyto třídy přijímají události, produkují stavy a jednotlivé změny zaznamenávají v podobě transakcí. Jednotlivé změny stavů jsou prováděny v metodě `mapEventToState(event)`, která produkuje tok stavů.

Z výpisu kódu 3.4 lze vidět, že knihona Bloc pracuje asynchronně pomocí toků dat za využití tříd `Stream` a `Sink`. To umožňuje pro jednu událost snadno vyvolat několik změn a zároveň reagovat na změny stavu asynchronně i v UI.

3.2.5 Zhodnocení

Vyzdvihování stavu, jak je popsáno v sekci 3.2.1, není ideální řešení pro udržitelnost a rozšiřitelnost. Knihovna Provider sice řeší vkládání závislostí, avšak neřeší samotnou správu stavů.

Knihovna MobX silně závisí na generování kódu, což může mít vliv na její rychlost. Oproti tomu jsou knihovny Bloc a Redux sice náročnější na porozumění, avšak nabízí asynchronní přístup, který je vhodný pro práci s daty ve velkých aplikacích.

Při porovnání knihoven Redux a Bloc lze vidět, že se obě knihovny zdánlivě velmi podobají. Konkrétní implementace i principy se však rozcházejí. Jeden z největších viditelných rozdílů je počet zdrojů, které udržují stav. Pro knihovnu Redux existuje pouze jeden zdroj, *Store*. V knihovně Bloc se používá libovolný počet těchto zdrojů, reprezentovaných třídami `Bloc`. Zároveň s tím se v knihovně Redux mění pouze části stavu, kdežto v knihovně Bloc se při změně nahrazuje celý stav stavem novým. Knihovna Bloc také svou implementací umožňuje velmi snadné asynchronní změny stavu, čímž umožňuje již zmiňované vícenásobné vyvolání změn pro jednu událost.

V této práci bude tedy z výše uvedených důvodů využívána knihovna Bloc. Tato knihovna umožňuje nejsnadnější asynchronní práci, sledování změn stavů pomocí transakcí a velmi dobře podporuje oddělení od externích závislostí.

3.3 Databáze

Aby aplikace mohla uchovávat data napříč aplikacemi, musí využívat některé databázové řešení. Databáze je souhrn informací, které jsou vhodně uspořádány tak, že jsou snadno dostupné a udržovatelné. Oproti jednoduchému ukládání dat do souboru na disku mají databáze hodně výhod. Data lze vyhledávat pomocí speciálních dotazů, lze je slučovat pomocí spojovacích funkcí, jsou tolerantní k chybám a může je používat více uživatelů najednou. A to všechno při zachování optimální rychlosti. [28]

Dle [29] existuje několik typů databází a přístupů k datům, přičemž největší rozdíly jsou v tom, jak ukládají který typ dat, a jak je k těmto datům umožněno přistupovat. *„Relational databases are structured, while non-relational databases are document-oriented and distributed.“* [29]

Zatímco Structured Query Language (dále jen SQL) databáze byly donedávna preferovaným způsobem databází, trendem posledních let jsou Non SQL (dále jen NoSQL) databáze [29]. Popis těchto typů databází, včetně výhod a nevýhod jednotlivého řešení, je uveden v následujících sekcích.

3.3.1 SQL

Data v SQL databázích jsou ukládány strukturovaně, čímž tyto databáze dosahují největších výhod. Výhodou je také jejich široká podpora a snadné řešení chyb. Principem těchto databází jsou vlastnosti, které shrnuje ACID (atomičnost, koexistence, izolace, trvanlivost). Těmito vlastnostmi databáze redukuje anomálie a chyby způsobené transakcemi. Díky dlouhé době, za kterou se SQL databáze již používaly, existuje spousta nástrojů a doplňků, které usnadňují vývoj a řešení případných chyb. Nevýhodou je však náročnější škálování databáze při vývoji aplikace. Příkladem je například databáze MySQL. [29]

3.3.2 NoSQL

Data v NoSQL databázích jsou většinou nestrukturovány. Tím získávají tyto databáze velký náskok co se flexibility dat a rychlosti přístupu týče. Tyto databáze je vhodné vnímat především jako kolekce, které obsahují dokumenty. NoSQL databáze jsou vhodné pro ukládání velkého množství nestrukturovaných dat, u kterých není nutné předem definovat jejich podobu. Nevýhodou je však relativní novost tohoto přístupu, což znamená i menší počet nástrojů, jako jsou nástroje k analýze a řešení problémů. Tyto databáze většinou používají vlastní dotazovací jazyk či jiné standardizace, což může vést k problémům při přechodu na jinou databázi [29]. Příkladem je například databáze Cloud Firestore [30].

3.3.3 Firebase: Cloud Firestore

Firebase je projekt, který pomáhá vývojářům s mobilními a webovými aplikacemi. Díky Firebase mohou být aplikace tvořeny rychleji, bez času stráveného budováním funkcionalit jako analytika a databáze. Tyto produkty jsou tvořeny na infrastruktuře Google a jsou využívány společnostmi jako jsou The New York Times, Duolingo, Alibaba a další. [31]

Cloud Firestore je flexibilní NoSQL databáze dostupná v projektu Firebase. Tato databáze lehce spolupracuje s ostatními produkty Firebase, jako jsou autentizace, správa práv nebo automatické aktualizace dat. Spolu s databází je poskytována řada balíčků pro vývoj mobilních a webových aplikací, které poskytují přátelské rozhraní k vývoji s touto databází. Dostupná je také funkce offline podpory, která automaticky ukládá často používaná data, která je možno používat i bez internetového spojení. Data je možné filtrovat pomocí speciálních dotazů na kolekce i dokumenty a tyto dotazy je také možné kombinovat a řadit. [30]

3.3.4 Zhodnocení

Vývoj herní aplikace je ideální pro využití NoSQL databází. V takovéto aplikaci se najde plno příležitostí, kde lze využít rysů těchto databází, kde by naopak rysy SQL databází působily zbytečné obtíže. I když nedisponují vlastnostmi ACID, je využití tohodo typu databáze, konkrétně databáze Cloud Firestore, nejvhodnější pro vývoj praktické části práce, proto bude použita právě databáze Cloud Firestore.

3.4 Senzory

Použití senzorů v herních aplikacích je velmi časté. Senzory totiž mohou dodat aplikaci data, která jí pomohou obohatit o detaily, případně na nich může aplikace postavit funkčnost dílčích částí. Počet ušlých kroků, navigace pomocí kompasu, ovládání závodního auta natáčením telefonu, nebo počítání provedených kliků — všechny tyto příklady využívají senzory zařízení. [32]

Mobilní zařízení má několik vestavěných senzorů, které měří pohyb, orientaci v prostoru a plno dalších věcí. Každé zařízení může mít rozdílný počet senzorů a senzory mohou být buď hardwarové, nebo softwarové. Softwarové senzory zpracovávají data z jednoho nebo více hardwarových senzorů a imitují očekávaný výstup [32]. Dle zdroje [32] se senzory dělí na tři základní typy:

Senzory pohybu Snímají otřesy a posuny, tedy zrychlení a rotaci, pomocí tří os. Příkladem je akcelerometr, senzor gravitace, gyroskop a rotační vektorový senzor.

Senzory polohy Snímají fyzickou pozici zařízení. Případem je magnetometr a senzor přiblížení.

Senzory prostředí Snímají hodnoty prostředí jako je teplota, tlak či vlhkost vzduchu. Příkladem je termometr a barometr.

Pro herní aplikace jsou nejdůležitější senzory pohybu a senzory polohy. Senzory prostředí závisí na datech prostředí [33], která se nemění tak často, a tedy nemají pro hru větší využití. Mnoho využití naopak umožňují senzory pohybu a polohy. V násedujících sekcích proto budou popsány senzory akcelerometr, gyroskop a senzor přiblížení.

3.4.1 Akcelerometr

Akcelerometr určuje akceleraci v rámci os daného mobilního zařízení spolu s gravitací, která na toto mobilní zařízení v jednotlivých jeho osách působí. Z tohoto důvodu akcelerometr naměří výsledky i v případech, kdy mobilní zařízení například leží na stole, konkrétně s hodnotou udanou gravitační silou $g = -9,81 m/s^2$ [34]. Tento senzor je tedy vhodný pro detekování pohybu zařízení a lze říci, že jej obsahují všechna zařízení se systémy Android a iOS.

3.4.2 Gyroskop

Gyroskop měří pohyb rotace jednotlivých os zařízení. Oproti akcelerometru tento senzor při nehybném zařízení nevydává žádná data, senzor totiž měří pouze změnu. Změna rotace je udávaná jako radiány za sekundu [34]. Tento senzor lze využít například pro rotaci ve 3D při 360° pohledu, kdy uživatel sedí na otočné židli a otáčením židle prohlíží okolí v aplikaci.

3.4.3 Senzor přiblížení

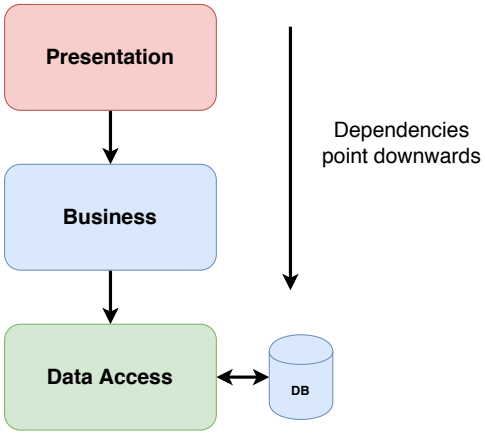
Senzor přiblížení určuje vzdálenost objektu od zařízení. Senzor je běžně umístěn na přední straně mobilního zařízení a určuje absolutní vzdálenost, nicméně některé druhy tohoto senzoru určují pouze stavy *blízko* a *daleko* [35]. Například systém Android vypíná obrazovku, zatímco uživatel telefonuje se zařízením přiloženým u ucha. Tento senzor lze využít například pro snímání vzdálenosti ruky od zařízení, nebo v kombinaci s gyroskopem k určení stavu, kdy je mobil položený na stole a otočený směrem k němu.

3.4.4 Zhodnocení

Jak již bylo uvedeno, nejvyšší uplatnění v herních aplikacích mají senzory pohybu a polohy. V předešlých kapitolách byly popsány senzory, které lze běžně v herních aplikacích využít. V praktické části práce proto budou moci být použity právě tyto senzory, které nabízí vhodné rozhraní, z volně dostupných balíčků.

3.5 Architektura

Stroje, které sestavují či spouštejí aplikace, nezajímá vzhled či uspořádání zdrojového kódu. Naopak vývojáři, jakožto lidé, potřebují udržovat kód snadno čitelný a srozumitelně rozdělený do jednotlivých metod, tříd, struktur či vrstev aplikace. Potřebují také vysokoúrovňové jazyky či pokročilé knihovny a frameworky, které ulehčí psaní kódu a nenutí vývojáře psát vše znovu. I přes to ale nejsou aplikace psány dle nejlepších doporučení a vývojáři velmi často píší či naráží na tak zvaný „spaghetti code“ [5]. To je mnohdy zapříčiněno nárokem na rychlou implementaci nových funkcí či úprav, kde vývojáři na úkor času zanedbají čitelnost a strukturu kódu a čím dál tím více postupem času prohlubují dluh kvality daného kódu. *„Software has two types of value: the value of its behavior and the value of its structure. The second of these is the greater of the two because it is this value that makes software soft.“* [36, str. 140] Ze zmíněného zdroje také plyne, že se dlouhodobě vyplatí vývoj směřovat k dobré struktuře a architektuře, než k samotné implementaci, jelikož implementace se snadno opraví, kdežto opravit architekturu bez velkého zásahu je značně obtížné [36, str. 135–146].



Obrázek 3.8: Schéma architektury zaměřené na databázi [5]

Cílem každé aplikace je mít udržitelný a snadno rozšiřitelný kód. Je proto třeba stanovit řadu pravidel, které pomohou s vývojovými otázkami při návrhu metod, tříd, rozhraní či celých modulů. Architektura a její správný a promyšlený návrh umožňuje vyvarovat se chyb a dluhu na kvalitě a modularitě, který znemožní snadný proces udržitelnosti aplikace. *„Good architecture makes the system easy to understand, easy to develop, easy to maintain, and easy to deploy. The ultimate goal is to minimize the lifetime cost of the system and to maximize programmer productivity.“* [36, str. 137] Dobře navržená a udržovaná architektura tak umožní z dlouhodobého hlediska produkovat kvalitnější kód [36, str. 135–146]. Kvalitnější architektura tedy logicky minimalizuje nutnost oprav a přizpůsobení struktury aplikace při vývoji každé nové funkcionality.

Dle zdroje [5] lze rozdělit architektury do dvou skupin: architektury zaměřené na databázi a architektury zaměřené na doménu. V následujících sekcích budou popsány rozdíly mezi oběma skupinami a vybranými konkrétními případy architektur.

3.5.1 Architektury zaměřené na databázi

Architektury zaměřené na databázi, také nazývané *database-centric architectures*, byly dle [5] prvním typem softwarových architektur. Tyto architektury staví databáze do středu dění, jak jde vidět z obrázku 3.8.

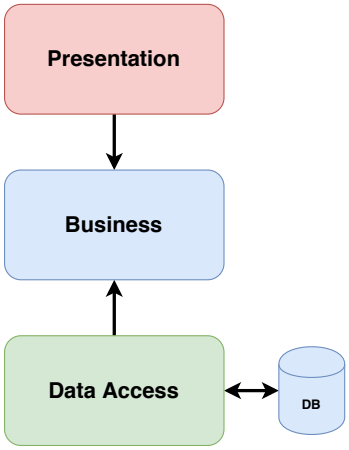
Příkladem tohoto typu architektury je tradiční 3-vrstvá architektura. Ta je robustní a škálovatelná. Skládá se ze tří vrstev. První vrstva, prezentační vrstva, obsahuje tvorbu UI, bez samotného kódu s logikou. Druhá vrstva, aplikační vrstva, obsahuje samotný kód s logikou aplikace. Tato vrstva by měla být nezávislá na prezentační vrstvě, avšak má explicitní závislost na datové vrstvě. Poslední vrstva, datová vrstva, je nejnižší vrstvou, která se stará o ma-

```
package ua.com.crosp.testapp.domain;
import ua.com.crosp.testapp.datalayer.PostsRepository;

public class GetRecentPostsUseCase
implements GetRecentPostsUseCaseContract {
    private PostsRepository mPostsRepository;

    public Single<Post.List> execute(Params params) {
        // Execute
    }
}
```

Výpis kódu 3.5: Ukázka přístupu zaměřeného na databázi v jazyce Java [5]



Obrázek 3.9: Schéma architektury zaměřené na doménu [5]

nipulaci a tok dat z databáze do nižších vrstev. V této architektuře je největší role přikládána databázi a často je tak datová a aplikační vrstva úzce propojena. V nejhorších případech na datové vrstvě závisí i vrstva prezentační [5]. Ukázka přístupu pomocí této architektury je znázorněna ve výpisu kódu 3.5.

3.5.2 Architektury zaměřené na doménu

Architektury zaměřené na doménu, také nazývané *domain-centric architectures*, jsou produktem vývoje z předchozího typu architektur. Jak jde vidět z obrázku 3.9, střed dění je přesunut na aplikační vrstvu, protože právě tato vrstva je v aplikaci nejdůležitější, jelikož v ní probíhá implementace nových funkcionalit, změn či oprav [5]. *„The way you keep software soft is to leave as many options open as possible, for as long as possible. What are the options that we need to leave open? They are the details that don't matter.“* [36, str. 140]

```
package ua.com.crosp.testapp.domain;
import ua.com.crosp.testapp.domain.PostsRepositoryContract;

public class GetRecentPostsUseCase
implements GetRecentPostsUseCaseContract {
    private PostsRepositoryContract mPostsRepository;

    @Inject
    public GetRecentPostsUseCase(PostsRepositoryContract r) {
        mPostsRepository = r;
    }
    public Single<Post.List> execute(Params params) {
        // Execute
    }
}
```

Výpis kódu 3.6: Ukázka přístupu zaměřeného na doménu v jazyce Java [5]

Výhodou architektur zaměřených na doménu také je, že díky návrhu samotnému abstrahují implementační detaily do podoby kontraktů. Těmto kontraktům může nadále vyhovět několik konkrétních implementací, což umožní snadnou výměnu služeb či jiných závislostí [36, str. 135–146]. Lze tedy jednoduše nahradit například konkrétní databázi — z databáze MySQL na databázi Cloud Firestore —, bez toho, aniž by se muselo zasahovat do samotné aplikační vrstvy aplikace. Ukázka přístupu pomocí tohoto typu architektury je znázorněna ve výpisu kódu 3.6.

3.5.3 Clean Architecture

Clean Architecture je architektura popisovaná ve stejnojmenné knize od Roberta C. Martina. *„Good software systems begin with clean code. On the one hand, if the bricks aren't well made, the architecture of the building doesn't matter much. On the other hand, you can make a substantial mess with well-made bricks. This is where the SOLID principles come in.“* [36, str. 57] Architektura hojně využívá principů SOLID, které byly představeny stejným autorem jako kniha. Principy SOLID jsou následující:

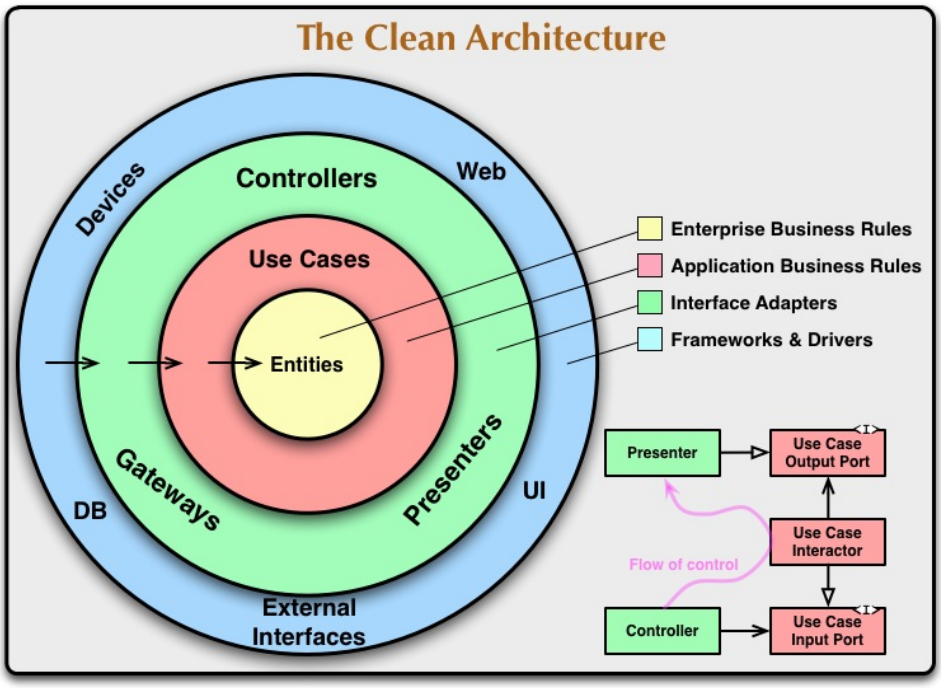
Single-responsibility principle *„An active corollary to Conway's law: The best structure for a software system is heavily influenced by the social structure of the organization that uses it so that each software module has one, and only one, reason to change.“* [36, str. 57–59]

Open–closed principle *„Bertrand Meyer made this principle famous in the 1980s. The gist is that for software systems to be easy to change, they must be designed to allow the behavior of those systems to be changed by adding new code, rather than changing existing code.“* [36, str. 57–59]

Liskov substitution principle *„Barbara Liskov's famous definition of subtypes, from 1988. In short, this principle says that to build software systems from interchangeable parts, those parts must adhere to a contract that allows those parts to be substituted one for another.“* [36, str. 57–59]

Interface segregation principle *„This principle advises software designers to avoid depending on things that they don't use.“* [36, str. 57–59]

Dependency inversion principle *„The code that implements high-level policy should not depend on the code that implements low-level details. Rather, details should depend on policies.“* [36, str. 57–59]



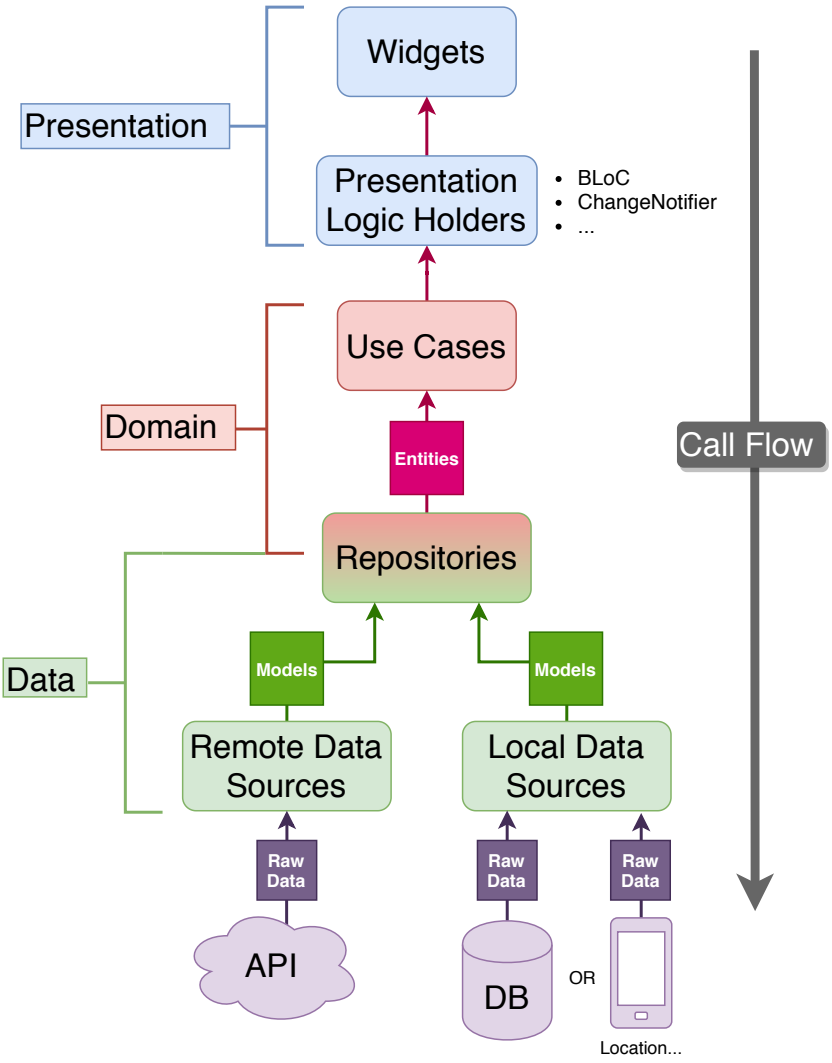
Obrázek 3.10: Schéma architektury Clean Architecture [37]

Princip inverze závislostí je jedním z nejdůležitějších principů SOLID, který Clean Architecture využívá. Tento princip říká, že by kód neměl záviset na detailech přímo, ale raději na obecném kontraktu (například pomocí dědičnosti nebo rozhraní), který daný detail implementuje. Další principy popisují, že každá třída by měla mít zodpovědnost právě za jednu věc, rozšíření by mělo být možné pouze za pomoci přidání nového kódu, pokud kód využívá třídu, musí fungovat i pokud místo ní použijeme její podtřídu a třídy by neměly záviset na více věcech, než je potřeba [36, str. 61–91]. Obdobné principy jsou aplikovány i na úrovni komponent aplikace a dohromady s principy SOLID způsobí, že takový systém je testovatelný, nezávislý na konkrétním UI (web, mobilní zařízení, desktop), použitém frameworku, použité databázi či dalších službách [37].

Cílem architektury Clean Architecture je hlavně vhodné oddělení zodpovědnosti. Jak jde vidět z obrázku 3.10, závislost jednotlivých vrstev aplikace může mířit pouze dovnitř. Zároveň jsou věci uvnitř více abstraktní, zatímco věci na kraji více konkrétní. Středem všeho jsou entity, které mohou obsahovat samotná data, ale i jejich funkce. S entitami pracují *use cases*, které implementují za pomocí entit všechny případy užití. Poslední vrstva většinou neobsahuje implentaci a pouze volá vnitřní vrstvu. Do této vrstvy může patřit například mobilní framework (framework Flutter), který volá kontroler (state management BLoC), který zařídí pouze samotné spuštění správného use case. Tato pravidla nejsou striktní a už pouhým rozdělením do vhodných vrstev či pravidlem pro směr závislostí lze vytvořit velmi udržitelný software. [37]

Otázka architektury je také dělení kódu do nezávislých sekcí či balíčků. Přístupy se mohou lišit, nejběžnějším je však rozdělení dle vrstev, tedy nejčastěji dle prezentační, aplikační a datové vrstvy. Takovýto přístup je pro začátek dostačující a často je doporučován tutoriály. Dalším přístupem je rozdělení dle stejných vlastností (*features*). To znamená, že se kód nedělí horizontálně, ale vertikálně. Kód je sice možná lehčí nalézt, než u přístupu rozdělení dle vrstev, avšak pro změnu neřeší oddělení dat, domény a UI. Dalším z přístupů je rozdělení kódu jak dle vlastností, tak následně dle vrstev, což přináší výhody z obou předešlých způsobů. [36, str. 303–321]

Tato architektura je samozřejmě značně abstraktní, proto je nutné konkrétní věci organizovat dle potřeb aplikace, například dle návrhu *Reso Coder's Flutter Clean Architecture Proposal*, který je vidět na obrázku 3.11. Tento návrh rozděluje kód do vrstev tak, že framework Flutter a správa stavů je ve vrstvě prezenční, use cases, entity a kontrakty repozitářů jsou ve vrstvě aplikační a modely (které dědí z entit) a implementace repozitářů jsou ve vrstvě datové. Třídy modelů oproti entitám obsahují transformační metody jako převod z a do formátu JSON, jelikož tyto metody jsou pro aplikační vrstvu nepodstatným detailem. [38]



Obrázek 3.11: Schéma Reso Coder's Flutter Clean Architecture Proposal [38]

3.5.4 Zhodnocení

Architektury zaměřené na databázi až příliš lpí na detailech, na kterých ve výsledku nezáleží. Architektury zaměřené na doménu tento problém naopak řeší, využitím kontraktů namísto specifických detailů. Architektura Clean Architecture je vhodným příkladem takovéto architektury. Díky principům SOLID a dalších vhodných pravidlech a jejich využíváním je možné lehce tvořit udržitelný a rozšiřitelný kód, ve kterém se lze snadno pohybovat a vše má své dané místo. V praktické části bude proto využíváno právě architektury Clean Architecture.

3.6 Testování

Testování se v průběhu let stalo nedílnou součástí vývoje aplikací. Je vhodné pro všechny velikostí aplikací, především proto, že testování vede k vyšší kvalitě kódu a tím k snadnější udržitelnosti aplikace. Cílem testů je ověřit a dále kontrolovat, že specifické části aplikace fungují a že zachovávají tuto funkčnost při jejich úpravě [39]. *„Dijkstra once said, »Testing shows the presence, not the absence, of bugs.« In other words, a program can be proven incorrect by a test, but it cannot be proven correct. All that tests can do, after sufficient testing effort, is allow us to deem a program to be correct enough for our purposes.“* [36, str. 31]

V této podkapitole jsou popsány typy testů, které pomáhají k snažšímu testování. Také jsou zmíněny metody a přístupy, které umožní tvoření kvalitnějších testů a využití testů při připojování kódu ve verzovacích systémech.

3.6.1 Typy testů

Testovat je možné i ručně. To je však z globálního hlediska neefektivní a zdlouhavé. V praxi se proto využívají automatizované testy, které lze spustit samostatně kdykoli během cyklu vývoje aplikace. Dle [40] se automatizované testy dělí do těchto kategorií:

Unit testy Testují jednu funkci, metodu nebo třídu. Tento typ testů je snadné vytvořit, rychle se spouští a jejich smyslem je ověřit správnou funkčnost dané části. Externí závislosti jsou nahrazeny jejich testovací kopií, která neprovádí žádnou funkčnost.

Widget testy Někdy také nazývané jako *component testy*, testují jeden widget jako celek. Jejich smyslem je ověřit, jestli se daný widget chová a vypadá dle očekávání. Tento typ testů je složitější vytvořit a udržovat.

Integrační testy Testují celou aplikaci nebo její části. Tento typ testů je složité vytvořit, mají velké množství závislostí a jsou velmi pomalé na spuštění. Na druhou stranu testují korektnost aplikace, či její části, jako celku.

Ve výpisu kódu 3.7 lze vidět příklad unit testu. Unit testování vyžaduje jako závislost pro vývoj balíček `test`. Příklad obsahuje čítač reprezentovaný třídou `Counter`, která umí svou hodnotu `value` inkrementovat pomocí metody `increment()`. Jak lze vidět, použitý balíček umožní vývojáři využívat například metody `test()` a `expect()`, díky kterým lze popsat očekávaný stav každého testu. [6]

```dart
import 'package:test/test.dart';
import 'package:counter_app/counter.dart';

void main() {
    test('Counter value should be incremented', () {
    final counter = Counter();

    counter.increment();

    expect(counter.value, 1);
    });
}
```

Výpis kódu 3.7: Ukázka unit testu [6]

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

void main() {
    testWidgets(
        'MyWidget has a~title and message',
        (WidgetTester tester) async {
        await tester.pumpWidget(
            MyWidget(title: 'T', message: 'M'),
        );

        final titleFinder = find.text('T');
        final messageFinder = find.text('M');

        expect(titleFinder, findsOneWidget);
        expect(messageFinder, findsOneWidget);
        },
    );
}
```

Výpis kódu 3.8: Ukázka widget testu [7]

Ve výpisu kódu 3.8 lze naopak vidět příklad widget testu. Testování widgetů vyžaduje jako závislost pro vývoj balíček `flutter_test`. Příklad ukazuje ukázku toho, jak lze u testovaného widgetu otestovat očekávané položky *title* a *message*. Princip funkčnosti je stejný, jako u balíčku `test`, jen se namísto metody `test()` využívá metoda `testWidgets()`. [7]

3.6.2 Statická analýza

Kromě samotného testování kódu je také možné testovat kód pomocí statické analýzy. Statická analýza provádí kontrolu programu ještě před tím, než je spuštěn, což přináší výhody právě v tom, že je možné vidět chyby přímo při vývoji. Kontrolovat se tak mohou například styl, týmové konvence a standardy, struktura a další. Možnosti jednotlivých jazyků a technologií se liší a většinou jsou nástroje na testování poskytovány pomocí služeb či nástrojů třetích stran. Jazyk Dart má však nástroj *Analyzer for Dart*, který poskytuje statickou analýzu a vývojáři mohou využít řadu pravidel pro nastavení a přizpůsobení projektu. [41]

3.6.3 Code coverage

Code coverage znamená pokrytí kódu. V testování to je pojem, který značí, do jaké míry jsou třídy a jejich vlastnosti a metody otestovány. Určitě ale neplatí, že 100 % code coverage značí, že je aplikace správně otestována. Tato metoda identifikuje kód, který již není používán, nebo ani nikdy nebyl, nebo dokonce netestovatelný kód, způsobený špatnou implementací. [42]

Code coverage také monituruje vývoj v čase, případně popisuje změny, které by se do aplikace zanesli při připojení kódu ve verzovacím systému. V rámci systémů rozšiřující verzovací systém lze využívat několik nástrojů, které code coverage — v rámci kontinuální integrace, která je popsána v kapitole 3.6.5 — moniturují a podávají vývojářům zprávy. Příkladem těchto služeb může být CodeCov či Coveralls. Každý soubor či složka jsou ohodnoceny procenty, které značí poměr otestovaného a neotestovaného kódu. Služba pracující s code coverage při každé obdržené dávce kódu spustí průzkum kódu aplikace a porovná jej se stávající verzí. [42]

3.6.4 Test driven development

Test driven development (dále jen TDD) je metoda pro vývoj aplikací tím způsobem, že se začíná psaním testů, které je následováno psaním produkčního kódu, které testy splňuje [43]. Dle [43] má TDD tři jednoduchá pravidla:

1. Není dovoleno psát žádný produkční kód kromě toho, který opravuje selhávající test.

2. Není dovoleno psát více unit testů než je dostačující k selhání; a selhání kvůli kompilaci jsou také selhání.

3. Není dovoleno psát více produkčního kódu, než je dostatečné pro úspěšné provedení unit testu.

Nejdříve je tedy napsán unit test pro funkcionalitu, kterou chce vývojář přidat. Následně vývojář napíše jen tolik kódu, kolik je třeba k opravě daného testu a celý proces se znovu opakuje. Celá myšlenka stojí na tom, že vývojář může psát pouze kód, který opravuje selhávající test [43]. *„Now most programmers, when they first hear about this technique, think: »This is stupid.« »It's going to slow me down, it's a waste of time and effort, it will keep me from thinking, it will keep me from designing, it will just break my flow.« However, think about what would happen if you walked in a room full of people working this way. Pick any random person at any random time. A minute ago, all their code worked.“* [43]

Jak již bylo zmíněno, dle výzkumu [39] je navíc tento přístup velmi efektivní, jelikož vede k vyšší kvalitě kódu. Dodržováním těchto tří pravidel přesně víme, jak volat určité rozhraní, jelikož pro toto rozhraní existuje test. Kdokoli potřebuje znát cokoli o aplikaci, existuje test, který tuto činnost popisuje [43].

3.6.5 Kontinuální integrace

Jakákoli aplikace se běžně skládá z mnoha částí a obsahuje mnoho řádků kódu, zatímco do sebe musí vše dokonale zapadat a i nejmenší chyba může způsobit selhání sestavení aplikace. Při přidávání nových funkcionalit nastává velké riziko, že aplikace obsahuje chybu. Při snaze připojení změn pomocí nástrojů rozšiřující verzovací systémy tak proto nastává příležistost pro CI, která provede sestavení aplikace a spustí automatizované testy. V tomto okamžiku dostane vývojář skrze rozšiřující aplikace indikaci, zda změny kód rozbijí, nebo zda aplikace zůstane v funkční. [44]

Použití CI je výhodné především při práci v týmu. Právě díky zmíněným výhodám lze připojovat změny rychle a bez obav o zanesení jinak skrytých chyb. Tyto nástroje jsou buď zabudované do nástrojů rozšiřujících verzovací systémy jako jsou GitHub a jejich GitHub Actions či GitLab a jejich GitLab CI, nebo nezávislé nástroje, jako jsou Travis CI či CircleCI. [44]

3.6.6 Zhodnocení

Psát testy je velmi důležité pro každý softwarový produkt. Zatímco psaní widget testů a integračních testů může být zdlouhavé a mohou se často měnit požadavky, psaní unit testů je velmi důležité a dalo by se říci, že i nutností. V praktické práci budou tedy využity zejména unit testy, které pomáhají testovat samotnou funkcionalitu daných částí a zaručují, že stanovené rozhraní se nemění. Tyto testy nadále budou spouštěny v rámci code coverage, což je sice nejistý indikátor, ale i tak je dobré jej udržovat co nejvýše. Testy a code coverage je vhodné spouštět automaticky pomocí kontinuální integrace. TDD je skvělá metoda, která má mnoho benefitů, jak bylo popsáno v kapitole 3.6.4. Aplikace této metody ovšem vyžaduje spoustu zkušeností v rámci několika měsíců, proto bude tento přístup při vývoji pouze upřednostňován.

Kapitola 4

Analýza

Tato kapitola se věnuje analýze praktické části práce, mobilní hře. V nadcházejících podkapitolách je popsán princip hry, její funkční a nefunkční požadavky i jednotlivé případy užití.

4.1 Popis hry

Mobilní hra je laděna satiricky do prostředí jaderné elektrárny, kterou musí uživatelé, hráči hry v rozdílných rolích, zachránit od zničení. Hra se nazývá *Totally Not Chernobyl* a její děj se odehrává v neznámé jaderné elektrárně. V této elektrárně má většina pracovníků dovolenou, a proto jsou přítomni jen dva lidé, technik a manažer. Náhlý problém zapříčiní, že mezi těmito pracovníky byla zablokována cesta, a tak se tito dva zaměstnanci musí domluvit pouze hlasem a zařídit opravu elektrárny. Technik vidí všechny řídící prvky, manažer vidí plány a manuály. Jejich cílem je tak společnou spoluprací zabránit neštěstí v podobě exploze.

Hra je členěna do několika kapitol a každá kapitola obsahuje několik statických misí. Mise jsou statické v tom významu, že mají jasně přednastaveny prvky, které moduly obsahuje, avšak mění se jejich konkrétní nastavení. To zajišťuje jak možnost znovuopakování, tak poměřování s ostatními hráči.

Každá mise, a tedy konkrétní hra samotná, obsahuje několik modulů, které reprezentují určité řídící prvky elektrárny. To mohou být nejrůznější tlačítka či přepínače, ale i speciální moduly, reprezentující prvky elektrárny, které nelze vypnout či opravit a musí být periodicky kontrolovány, jako pumpa dodávající chladící vodu do reaktoru. Kromě modulů je potřeba ve hře okamžitě provádět akce, které se zobrazí. Akce jsou většinou jednorázové, kdy stačí provést danou akci.

Konkrétní mise začíná po tom, co jsou oba pracovníci připraveni a je stisknuto tlačítko *AZ-5*, které by mělo zajistit rychlé odstavení reaktoru, avšak z neznámého důvodu nefunguje. Po skončení každé mise, případně souhrnně

za všechny odehrané hry, jsou zobrazeny statistiky hry. Statistiky po ukončení hry zobrazí uplynulý čas, potřebný k opravě. Souhrnné statistiky zobrazují celkový počet odehraných her a poměr úspěšně splněných her.

Akce jsou přerušením, které musí hráči okamžitě vykonat. Příkladem akce může být *zatřeste zařízením*. Význam jednotlivých akcí je zjevný z jeho názvu. Akce jsou využívány zejména v pokročilejších kapitolách, jelikož mohou značně komplikovat hru.

4.2 Funkční požadavky

Funkční požadavky popisují jednotlivé požadavky na funkcionalitu aplikace. Z pohledu uživatelů aplikace popisují akce, které může uživatel provést. Jednotlivé funkční požadavky, či jejich uskupení, mohou tvořit jednotlivé nezávislé moduly aplikace. [45]

F1 Přihlášení – Uživatel se bude moci do aplikace přihlásit, resp. registrovat, pomocí účtu Google.

F2 Informace o aplikaci – Aplikace obsahuje obrazovku s informacemi o aplikaci, včetně jména autora, verze, popisu a odkazu na projekt.

F3 Vytvoření hry – Přihlášený uživatel může vytvořit hru. Současně hráč nastaví misi ke hraní.

F4 Připojení do hry – Přihlášený uživatel se může připojit do hry pomocí kódu.

F5 Spuštění hry – Po potvrzení, že jsou hráči připraveni na hru, mohou spustit hru s danou misí.

F6 Ukončení hry – Hráč může během přípravy hry nebo hraní hry danou hru ukončit.

F7 Statistika po hře – Hráčům je po skončení hry zobrazena herní statistika.

F8 Zobrazení svého profilu – Přihlášený uživatel si může zobrazit svůj profil.

F9 Zobrazení svých statistik – Přihlášený uživatel si může zobrazit své souhrnné statistiky ze všech hraných misí.

F10 Změna nastavení – Přihlášený uživatel si může změnit nastavení.

F11 Odhlášení – Přihlášený uživatel se bude moci z aplikace odhlásit.

4.3 Nefunkční požadavky

Nefunkční požadavky nepopisují konkrétní chování systému, ale spíše jeho vlastnosti a omezení. Tyto vlastnosti a omezení také přímo souvisí s očekáváními, které mají uživatelé na aplikaci, ať už se jedná o snadné použití funkcí, rychlost a plynulost jednotlivých kroků, nebo například bezpečnost. [45]

N1 Multiplatformnost – Aplikaci lze po nasazení spustit v současných verzích Android a iOS.

N2 Udržitelnost a rozšiřitelnost – Kód aplikace je psán přehledně, s vhodnou architekturou a vhodnými konvencemi tak, že je lehce udržitelný a rozšiřitelný.

N3 Plynulost – Aplikace funguje na všech obrazovkých plynule i na výkonnostně slabších zařízeních.

N4 Vícejazyčnost – Aplikace podporuje český jazyk a je připravena na rozšíření o další jazyky.

4.4 Případy užití

Funkční a nefunkční požadavky sice definují funkcionalitu a vlastnosti aplikace, avšak pro správné pochopení fungování a souhru těchto požadavků, je nutné definovat případy užití, které popisují jednotlivé problémy. Případy užití jsou vytvořeny na základě stanovených funkčních požadavků, s ohledem na stanovené nefunkční požadavky.

UC1 Přihlášení

Tento případ užití popisuje proces přihlášení do aplikace. Přihlášení probíhá pomocí účtu Google. Proces zahrnuje i registraci, pokud je uživatel v aplikaci nový.

1. Uživatel klikne na tlačítko *Přihlásit se* a následně proběhne přihlášení pomocí účtu Google.
2. V závislosti na tom, jestli je uživatel již registrovaný, mohou nastat dvě situace:
    a) pokud uživatel již byl registrovaný, je uživatel přihlášen a přesměrován na hlavní obrazovku;
    b) pokud uživatel ještě nebyl registrovaný, zobrazí se obrazovka, ve které vyplní své osobní údaje a následně je přihlášen a přesměrován na hlavní obrazovku.

UC2 Zobrazení informací o aplikaci

Tento případ užití popisuje proces zobrazení informací o aplikaci.

1. Uživatel se nachází na hlavní obrazovce a klikne na tlačítko *O aplikaci*.
2. Uživatel je přesměrován na obrazovku s informacemi o aplikaci, kde jsou zobrazeny informace jako jsou jméno vývojáře, verze aplikace, odkaz na repozitář se zdrojovým kódem nebo nejčastěji kladené otázky.

UC3 Změna nastavení

Tento případ užití popisuje změnu nastavení, kde lze zejména nastavit jazyk aplikace.

1. Uživatel se nachází na hlavní obrazovce a klikne na tlačítko *Nastavení*.
2. Uživatel je přesměrován na obrazovku s nastavením. Zde může uživatel například přepínat jazyk aplikace.

UC4 Hra

Tento případ užití popisuje proces vytvoření, spuštění a průběhu hry. Případ užití je rozdělen do dvou scénářů. První scénář popisuje uživatele, který hru zakládá. Druhý scénář popisuje uživatele, který se do hry připojuje.

Scénář A – zakládající uživatel

Tento scénář popisuje případ užití pro uživatele, který hru založí a hostuje. Tento uživatel má také práva pro nastavení hry, jako je kapitola a mise nebo role.

1. Uživatel klikne na tlačítko *Nová hra* a aplikace jej přesune na obrazovku místnosti.
2. Zde uživatel vidí speciální kód, který předá spoluhráči.
3. Uživatel vybere kapitolu, ve které vybere misi, a určí svou roli ve hře — technik či navigátor.
4. Uživatel klikne na tlačítko *Připraven*.
5. Až budou oba hráči připraveni ke hře, uživatel klikne na tlačítko *AZ-5* a zahájí tak hru.
6. Po skončení hry se uživateli zobrazí statistika ze hry.

Scénář B – připojující se uživatel

Tento scénář popisuje případ užití pro uživatele, který se připojuje do již vytvořené hry. Tento uživatel nemá žádná práva pro nastavení.

1. Uživatel klikne na tlačítko *Připojit se* a aplikace jej přesuna na obrazovku místnosti, kde zadá kód hry.
2. Uživatel klikne na tlačítko *Připraven*.
3. Po skončení hry se uživateli zobrazí statistika ze hry.

UC5 Ukončení hry

Tento případ užití popisuje případy, kdy a jak lze ukončit probíhající hru.

Scénář A – dohrání

Tento scénář popisuje standardní případ, kdy hráč skončí hru dle předpokladů, ať už úspěšně či neúspěšně.

1. Uživatel se připojí do hry.
2. Uživatel hraje hru.
3. Uživatel dohraje hru, ať už úspěšně nebo neúspěšně.
4. Aplikace zobrazí herní statistiky.
5. Uživatel následně opustí hru kliknutím na tlačítko *zpět*.

Scénář B – předčasné ukončení hry

Tento scénář popisuje situaci, kdy se jeden z hráčů rozhodne předčasně, kvůli jakémukoli důvodu, opustit hru a tím jí ukončit.

1. Uživatel se připojí do hry.
2. Uživatel hraje hru.
3. Uživatel klikne na tlačítko *Ukončit hru* a potvrdí dialog.
4. Aplikace zobrazí hlášku o předčasném ukončení hry.

UC6 Zobrazení profilu a statistik

Tento případ užití popisuje, jak uživatel zobrazí svůj profil a své souhrnné statistiky.

1. Uživatel klikne na tlačítko *Profil*.
2. Aplikace zobrazí obrazovku profilu, která zobrazuje informace o profilu a souhrnné statistiky profilu.

4.4.7 Přehled realizace požadavků

Přehled realizace případů užití funkčními požadavky lze vidět v tabulce 4.1.

	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11
UC1	X										X
UC2		X									
UC3										X	
UC4			X	X	X	X	X				
UC5						X					
UC6								X	X		

Tabulka 4.1: Realizace případů užití a splnění požadavků

Kapitola 5

Návrh

Po rešerši a analýze, kde byly popsány možnosti technologií a požadavky, je v návrhu cílem tyto poznatky využít a navrhnout jednotlivé aspekty aplikace. V následujících podkapitolách jsou postupně popsány návrhy modulů, uživatelského rozhraní, architektury a lokalizace. Ve všech částech je brána v potaz možnost rozšiřitelnosti a snadného použití.

5.1 Herní moduly

V této podkapitole je popsáno několik základních herních modulů, které jsou v aplikaci využívány. Tyto moduly jsou krátce popsány v následujících sekcích, ve kterých je vždy popsán jejich základní princip.

5.1.1 Modul Tlačítko

Modul Tlačítko je jednoduchý modul, který je reprezentován tlačítkem určité barvy a určitého textu. Podle barvy a textu musí hráč zmáčknout tlačítko v určitý moment, nejčastěji podle zbývajícího času hry.

5.1.2 Modul Posuvník

Modul Posuvník je nutné dle instrukcí manuálu přesunout na určenou polohu a následným stiskem tlačítka akceptovat.

5.1.3 Modul Sekvence

Modul Sekvence má za cíl ve správném pořadí stisknout určitá tlačítka. V manuálu tak může být posloupnost 4 tlačítek, zatímco na obrazovce je tlačítek 8. Tlačítka mohou obsahovat různé znaky.

5.1.4 Modul Bity

Modul Bity je modul, kde je za cíl zapsat číslo dle manuálu v bitové formě aktivací diod a následně potvrdit stiskem tlačítka. Dané číslo je v manuálu určeno podle klíčového slova.

5.2 Architektura

Návrh architektury je zásadní krok při vývoji aplikace. Nevhodně zvolená architektura může znemožnit snadné rozšíření aplikace a její údržbu. Při návrhu architektury je proto dbán zřetel na jednoduchost a univerzálnost. Návrh se netýká jen aplikace na nejvyšší úrovni, ale i dílčích částí. V následujících sekcích jsou popsány navrhované řešení herní smyčky, dělení aplikace na oddělené moduly a využití knihovny Bloc v aplikaci.

5.2.1 Herní smyčka

Pro herní aplikaci je důležitá herní smyčka, která popisuje samotné fungování hry a věci, které je nutné mezi zařízeními sdílet. Ve hře je potřeba udržovat informace o splněných herních modulech, splněných akcích a o dalších informacích o hře. Některé informace, jako jsou informace o herních modulech, mohou zůstat na zařízení hráče. Hráč s manuálem nepotřebuje, a neměl by, vědět všechny informace. Tyto informace mohou být v rámci týmové komunikace předány ústně. Díky tomu je možné návrh zjednodušit a tyto informace nesdílet a udržovat je pouze v zařízení hráče s herními moduly.

Důležité informace pro oba hráče jsou informace o splnění všech modulů, což znamená ukončení hry s výhrou. Dále informace o překročení časového limitu, což znamená ukončení hry s prohrou. A také informace o splnění akce na daném zařízení, resp. splnění akce na obou zařízeních, což vede k odblokování obrazovek a možnosti pokračovat ve hře. Mezi zařízeními budou tedy sdíleny pouze zmíněné informace – dokončení hry, překročení času a společné splnění akce. Jelikož není potřeba sdílet většinu informací, nejsou na hráče kladeny nároky na rychlost spojení.

Samotné nastavení mise obsahuje seznam modulů, které jsou v misi využity. Pro každý modul je zaznamenáno konkrétní nastavení, které se aplikuje do hry jak do manuálu, tak i do podmínek pro splnění modulu. Dále obsahuje informaci o čase k dokončení hry.

5.2.2 Moduly

Aplikace bude dle architektury Clean Architecture rozdělena na úrovni modulů na tři části. Tyto moduly obsahují zvlášť prezentační, aplikační a datovou vrstvu. První modul se zaměřuje na obrazovky mimo hru a její nastavení a řeší

autentizaci a přihlášení. Druhý modul se zaměřuje na nastavení hry a její spuštění a řeší propojení hráčů. Třetí modul se zaměřuje na hru samotnou a řeší její průběh. Kromě těchto specifických modulů je v obecném kontextu řešeno nastavení aplikace, které zmíněné moduly využívají.

5.2.3 Bloc

Prezenční vrstvu s vrstvou aplikační spojují kontrolery knihovny Bloc. Ty jsou reprezentovány třídami `Bloc` a udržují a mění stav dané části. Sdílená třída `Bloc` je například pro nastavení, které je nutné sdílet mezi všemi moduly.

Modul home obsahuje třídu `Bloc` pro autentizaci a přihlášení. Tyto třídy jsou rozděleny, protože zatímco autentizace řeší přepínání obrazovek podle stavu uživatele, přihlášení řeší přihlášení samotné na dané obrazovce. Protože každý stav dané třídy odpovídal pouze tomu, co by měl obsahovat, tyto třídy nelze spojit. Příkladem je obrazovka přihlašování, která se zobrazí pokud stav autentizace je `unauthenticated`, avšak může také zobrazit chybu, která je reprezentována stavem přihlašování `failure`. Pokud by tento stav byl v rámci autentizace, byl by ztracen kontext.

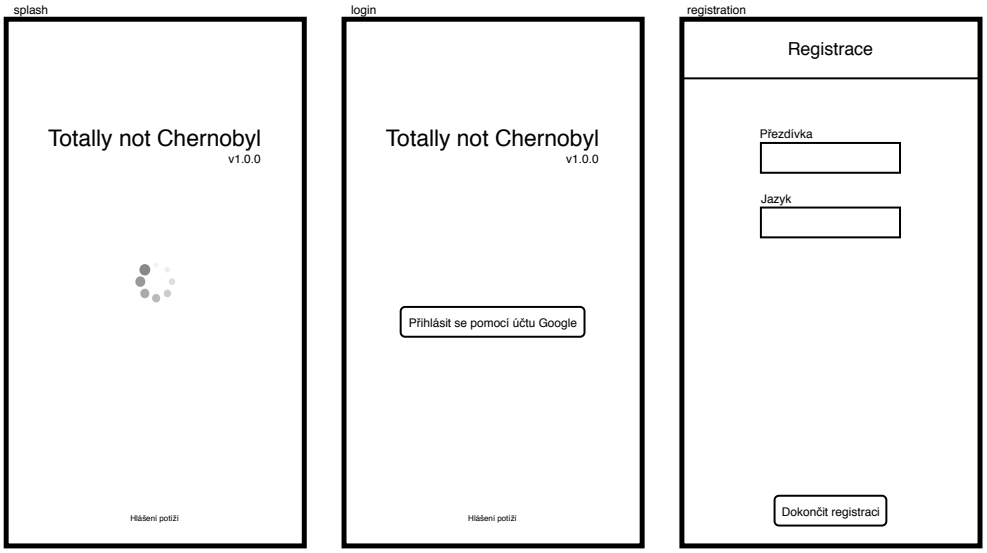
Modul lobby obsahuje třídu `Bloc` reprezentující stav vytváření hry. Jelikož se v tomto modulu nemění více stavů, je přítomen pouze jeden. Tomuto stavu se však průběžně mění data jako zvolená mise, přítomní hráči a zda jsou uživatelé připraveni.

Obdobně funguje třída `Bloc` modulu game, která však obsahuje stav i pro obrazovku ukončené hry se statistikami. Navíc má i stav pro hráče s manuálem, díky kterému se zobrazí informace namísto herních modulů. Třída pro hráče s herními moduly shromažďuje shromažďuje data o jednotlivých modulech a jejich řešení. Tento modul obsahuje ještě speciální třídu `Bloc` pro akce, které mohou překrývat obrazovku samotné hry.

5.3 Uživatelské rozhraní

Dle rešerše konkurenčních aplikací je cílem návrhu uživatelského rozhraní vytvořit jednoduchou grafiku, ve které se bude snadno navigovat a kterou uživatel snadno pochopí. Pro návrh uživatelského rozhraní je využíváno wireframů, což je metoda pro návrh rozhraní, kdy je obrazovka naskreslena pomocí jednoduchých tvarů a je tedy zobrazována struktura dané obrazovky, místo konkrétního zpracování. To slouží pro vyjádření základní reprezentace rozložení, které ovšem nemusí být finální a výsledná podoba obrazovek se může lišit. Veškeré wireframy byly vytvořeny pomocí programu draw.io.

V následujících sekcích jsou popsány obrazovky jednotlivých modulů s obrázkami daných wireframů.



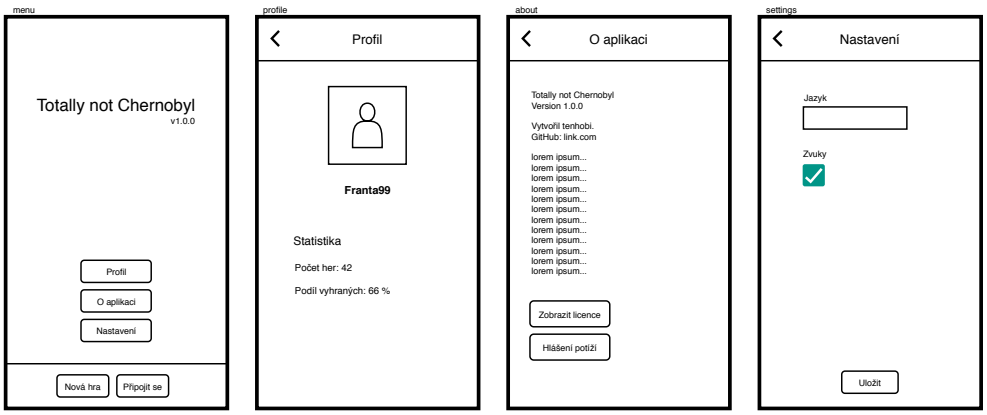
Obrázek 5.1: Wireframy obrazovek splash, login a registration

5.3.1 Modul home

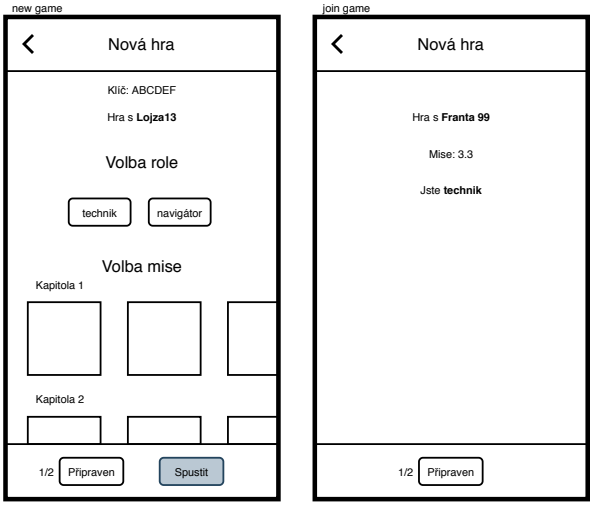
Modul home slouží pro navigaci uživatele mimo hru samotnou. Návrh tohoto modulu je velmi důležitý, jelikož uživatele do samotné aplikace uvede a umožní další postup. Cílem návrhu tohoto rozhraní je snadno provést uživatele přihlášením, registrací, menu, profilem, informacemi o aplikaci a nastavením. Modul obsahuje celkem sedm obrazovek.

Na obrázku 5.1 lze vidět obrazovky splash, která slouží pro načítání aplikace, login, která slouží pro přihlášení uživatele, a registration, která slouží pro vyplnění informací při registraci nového uživatele. Tyto obrazovky jsou první, které uživatel uvidí. Proto je nutné, aby byly obrazovky snadno pochopitelné a jednoduché. Uživatel na obrazovkách vidí pouze logo a příslušné ovládací prvky, což splňuje podmínky na jednoduchost.

Po přihlášení do aplikace se uživatel přesune na obrazovku menu. Z této obrazovky se může uživatel přesunout kamkoli v aplikaci, a proto jsou uživateli přehledně poskytnuta tlačítka, která ho srozumitelně provedou. Jedním z nejdůležitějších tlačítek je vytvoření hry, které je přehledně umístěno do spodní lišty. Z této obrazovky se uživatel může dostat na obrazovku profil, kde jsou zobrazeny informace o uživateli a jeho statistiky. Z obrazovky menu se lze také prokliknout na obrazovky about a settings. Obrazovka about obsahuje informace o aplikaci, včetně názvu aplikace, verze, vývojáře a tak podobně. Na obrazovce settings může uživatel nastavit novou přezdívku nebo změnit jazyk aplikace. Tyto obrazovky lze vidět na obrázku 5.2.



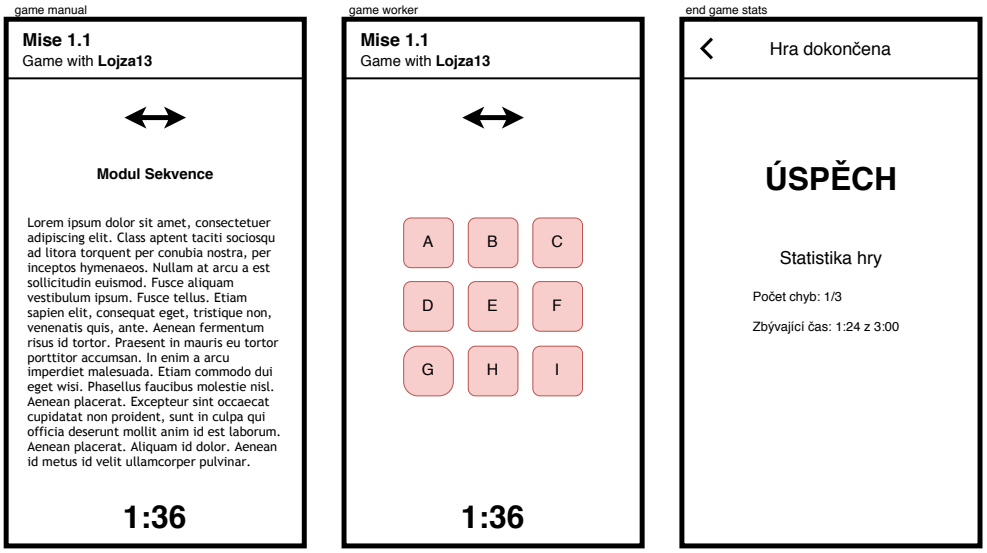
Obrázek 5.2: Wireframy obrazovek menu, profile, about a settings



Obrázek 5.3: Wireframy obrazovek new game a join game

5.3.2 Modul lobby

Pro vytvoření hry se musí uživatel přesunout na obrazovky modulu lobby, kde probíhá samotné nastavení hry před jejím vytvořením. Na tyto obrazovky se dostane hráč z modulu home po kliknutí na tlačítko vytvoření hry nebo připojení se do hry.



Obrázek 5.4: Wireframy obrazovek game manual, game worker a end game stats

Na obrazovce new game může zakládající uživatel nastavit hru, zejména si zvolit roli a vybrat misi. Na obrazovce uživatel vidí klíč pro připojení a hráče, který se do hry připojil. Na dolní liště uživatel přehledně vidí tlačíko na signalizaci ke startu hry a k odstartovánímu samotnému. Na obrazovce join game vidí uživatel, který se připojuje do hry, pouze statické informace o zvoleném nastavení. Na dolní liště uživatel přehledně vidí tlačítko na signalizaci ke startu. Tyto obrazovky jsou zobrazeny na obrázku 5.3.

5.3.3 Modul game

Samotný modul hry je složen ze tří obrazovek. Dvě obrazovky se věnují samotné hře a zobrazují zbývající čas. První obrazovka game manual slouží pro zobrazení informací ke vyřešení hry. Druhá obrazovka je zajímavější. Obsahuje samotné herní prvky, mezi kterými musí hráč přepínat a plnit je. Třetí obrazovka je obrazovka po skončení hry, která zobrazuje úspěch či neúspěch a statistiky hry. Tyto obrazovky jsou vidět na obrázku 5.4.

Kapitola 6

Implementace

Tato kapitola se zabývá popisem implementace aplikace. V následujících podkapitolách jsou popsány použité nástroje, použité knihovny a vybrané ukázky. Aplikace byla implementována dle návrhu z kapitoly 5.

6.1 Použité nástroje

Během vývoje byly použity nástroje, které usnadnily vývoj aplikace či její ladění. Pro samotné ladění aplikace byl využit nástroj *scrcpy*, který umožní promítat obrazovku mobilního zařízení na obrazovku počítače i s možností bezdrátového spojení.

6.1.1 Verzování

Během vývoje bylo využito verzovacích nástrojů. To je běžná součást každého projektu a slouží k postupnému zaznamenávání změn. V této práci byl využit verzovací systém Git.

Kromě nástroje Git byla využita i webová služba GitHub. Tato webová služba, obdobně jako konkurenční řešení (například GitLab), nabízí kromě bezplatného umístění repozitáře také doplňkové služby jako jsou code coverage, ticket systém, posuzování změn a podobně.

```
adb tcpip 5555
adb connect DEVICE_IP:5555
scrcpy --window-title "Totally Not Chernobyl" --always-on-top
--bit-rate 2M --max-size 800
```

Výpis kódu 6.1: Spuštění nástroje scrcpy pro bezdrátové použití

6.1.2 Editor

Během vývoje aplikace byly využity dva editory. Editor IntelliJ IDEA od společnosti JetBrains a editor VS Code od společnosti Microsoft. Editor IntelliJ IDEA je robustní editor, který obsahuje nápovědu a analýzu kódu. Díky výhodám jazyka Dart a frameworku Flutter byla však i práce v editoru VS Code snadná. Pro oba editory jsou dostupné v editorech chytré doplňky, které využívají integrovaných nástrojů dodávaných s jazykem Dart, ale i věci navíc jako poskytování útržků kódu pro usnadnění práce.

6.2 Použité knihovny

Při implementaci bylo využito několik knihoven, které elegantně řeší dané specifické problémy. Mezi nejzajímavější použité knihovny patří knihovny `bloc` a `flutter_bloc`, které poskytují rozhraní knihovny a widgety pro práci s nimi. Knihovna `easy_localization` řeší překlad uživatelského rozhraní aplikace do několika jazyků. Uživatel mezi jazyky může přepínat a tím měnit jazyk prvků. Lokalizace může být řešena více způsoby, nejjednoduší a jeden z nejvíce efektivních je pomocí vytvoření několika souborů, například ve formátu `JSON` nebo `YAML`, které jednoduše a přehledně umožní definovat slovník překladu. Samotná definice ovšem ne vždy stačí. Knihovna umožňuje vkládat parametry, které se do daného překladu vloží. Knihovna `google_sign_in` poskytuje jednoduché rozhraní pro přihlašování pomocí účtu Google. Jelikož jazyk Dart nepodporuje třídy union, knihovna `freezed` poskytuje rozhraní a generátor pro snadné generování těchto tříd. Knihovna `url_launcher` umožní spouštění webových odkazů nehledě na platformu. Knihovna `cloud_firestore` poskytuje rozhraní k práci se stejnojmennou databází Cloud Firestore. Dále je využito knihovny `effective_dart`, která poskytuje souhrn pravidel pro statickou analýzu dle konvencí jazyka Dart.

6.3 Ukázky vybraných částí

V následujících sekcích je uvedeno několik ukázek z implementace. Každá ukázka reprezentuje určitou část vývoje a popisuje její zajímavosti.

6.3.1 Datová třída

S využitím knihovny `freezed` je možné tvořit nejen třídy union, ale také datové třídy. Tyto třídy po vygenerování obsahují kopírující konstruktor, metody pro porovnání rovnosti, metody pro převod do formátu JSON a podobně. Práce s takovouto třídou je tedy velmi snadná, ukázka jde vidět v kódu 6.2.

6.3.2 Implementace widgetu

V ukázce kódu 6.3 je uvedena reprezentace implementace widgetu ve frameworku Flutter. Implementován je widget pro zobrazení loga aplikace. Jedná se o stateless widget, což znamená, že takovýto widget je zobrazen staticky a nic se v něm nemění. Widget může mít několik vstupních proměnných, které využívá, a dokonce může využívat i další widgety, které mohou být stateful.

Jak lze z ukázky vidět, je využito standardních widgetů jako je widget `Column`, který slouží pro uspořádání prvků do sloupce pod sebe a widgetu `Text`, který slouží pro výpis textu. Oba widgety mají řadu parametrů, které náležitě modifikují jejich styl. V ukázce je také vidět, že je využita metoda `tr()`, která slouží pro vyžádání překladu pomocí knihovny `easy_localization`.

Zajímavostí jazyka Dart také je, že umožňuje vkládat do seznamu podmíněné položky, jak lze vidět ve spodní části kódu. To umožňuje tvořit více čtivý kód a redukuje množství kódu.

6.4 Dokumentace

Spolu s implementací byla vytvořena i vývojářská a uživatelská dokumentace. Obě dokumentace jsou dostupné v příloze C. Vývojářská dokumentace je vytvořena automaticky ze zdrojového kódu aplikace, pomocí nástroje `dartdoc`. Tento nástroj vytváří dokumentaci ve formě webové aplikace, která vývojáře provede implementací podle komentářů z kódu. Ikdyž je samotné uživatelské rozhraní velmi přívětivé, na základě výsledné aplikace byla vytvořena i uživatelská dokumentace, která má za cíl poskytnout uživatelům jednoduchý a jasný návod, jak aplikaci ovládat.

```
@freezed
abstract class User with _$User {
    const factory User({
        @required String username,
        @required String language,
        @Default(0) int gamesCount,
        @Default(0) int winGamesCount,
    }) = _User;

    factory User.fromJson(Map<String, dynamic> json) =>
      _$UserFromJson(json);
}
```

Výpis kódu 6.2: Implementace datové třídy

```
class AppLogo extends StatelessWidget {
    final bool withVersion;
    final Brightness brightness;

    AppLogo({
    Key key,
    this.withVersion = true,
    this.brightness = Brightness.light,
    }) : super(key: key);

    @override
    Widget build(BuildContext context) {
        return Column(
            crossAxisAlignment: CrossAxisAlignment.center,
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
            Column(
                crossAxisAlignment: CrossAxisAlignment.end,
                children: <Widget>[
                Text(
                    tr('title'),
                    textAlign: TextAlign.end,
                    style: GoogleFonts.ubuntu(
                    fontSize: 32,
                    color: brightness == Brightness.light
                        ? Colors.white
                        : Colors.black,
                    ),
                ),
                if (withVersion == true)
                  Version(brightness: brightness),
                ],
            ),
            ],
        );
    }
}
```

Výpis kódu 6.3: Implementace widgetu ve frameworku Flutter

6.5 Zhodnocení

Výsledná aplikace Totally Not Chernobyl byla úspěšně vytvořena dle analýzy a návrhu. Aplikace obsahuje jednoduché uživatelské rozhraní a srozumitelné ovládací prvky. Byl implementován univerzální systém, který je možné snadno rozšiřovat a používat, což bylo jedním z cílů práce. Hra obsahuje několik ukázkových předpřipravených misí, které je však možné snadno obohatit či přidat další. Aplikace je testována několika způsoby, díky čemuž je zejména pro uživatele snadno použitelná a bez chyb.

6.6 Možnosti rozšíření

Aplikaci lze v budoucnu rozšířit mnoha způsoby. Jako první možnost k rozšíření se nabízí vylepšení obsahu hry, tedy přidání a rozšíření herních misí. Další možnosti k rozšíření lze najít v přidání herních modulů, které mohou celkový prožitek ze hry velmi obohatit. Samozřejmě se nabízí i možnost zcela upravit či rozšířit samotné funkce či možnosti hry, jako jsou přidání podpory pro další platformy (platformy web a desktop), využití více typů senzorů a práce s nimi nebo rozšíření o další herní módy (mód s více hráči).

Kapitola 7

Testování

Tato kapitola se zabývá popisem testování aplikace. Aplikace byla testována jak při vývoji vývojářem, tak i automatizovanými testy. Bylo provedeno také uživatelské testování.

Díky vlastnostem frameworku Flutter bylo možné ladit a testovat aplikaci při vývoji díky vlastnosti *hot reload*. Tato vlastnost při spuštění aplikace ve vývojářském režimu aplikuje změny do běžící aplikace, avšak uchovává současný stav jednotlivých widgetů. Díky tomu je možné okamžitě reagovat na změny a chyby v kódu a nemusí se tak čekat na zdlouhavé sestavení aplikace.

Cílem automatizovaných testů je otestovat funkčnost jednotlivých částí aplikace a otestovat příslušné rozhraní. Automatizované testy lze spouštět ručně i automaticky, například v rámci kontinuální integrace, což je velmi praktické a bylo při vývoji hojně využíváno. Během vývoje aplikace byla vytvořena řada unit testů, které testují patřičné části aplikace. Widget testy vytvořeny nebyly, protože nemají takovou vypovídající hodnotu a jsou velmi náchylné na změny v UI.

Velký zřetel byl dán na testování použitelnosti. Toto testování má za cíl odhalit jak chyby aplikace, tak zejména samotnou srozumitelnost a jednoduchost průchodu aplikace. Toto testování je dále popsáno v podkapitole 7.2.

7.1 Ukázky vybraných částí

V následujících sekcích je uvedeno několik ukázek z testování.

7.1.1 Získání uživatele

V ukázce kódu 7.1 lze vidět ukázka testování získání uživatele dle jeho ID. Tento test testuje příslušný usecase pomocí metody mockování, což je proces který imituje závislost kódu. Pro daný test tak není podstatná implementace závislosti, tedy jak je získán daný uživatel, ale skutečnost, že je využito správné metody repozitáře.

```
GetUserByUid usecase;
MockUserRepository mockUserRepository;

setUp(() {
    mockUserRepository = MockUserRepository();
    usecase = GetUserByUid(mockUserRepository);
});

final tUid = 123;
final tUser = User(username: 'test', language: 'test');

test(
    'should get user for the uid from the repository',
    () async {
        // arrange
        when(mockUserRepository.getUserByUid(any))
            .thenAnswer((_) async => Right(tUser));
        // act
        final result = await usecase(Params(uid: tUid));
        // assert
        expect(result, Right(tUser));
        verify(mockUserRepository.getUserByUid(tUid));
        verifyNoMoreInteractions(mockUserRepository);
    },
);
```

Výpis kódu 7.1: Test získání uživatele dle id

7.1.2 Serializace

V ukázce kódu 7.2 je testována správná serializace entity uživatele. V tomto testu je využito metody využívající takzvané fixtures, což jsou předpřipravená data v souboru, se kterými může test dále pracovat. Díky tomu může test načíst soubor ve formátu JSON a převést jej do mapy v jazyce Dart. Tento výstup je následně porovnán s očekávaným výsledkem. Tento test tedy testuje, zda je ze souboru JSON správně vytvořen objekt entity.

7.2 Testování použitelnosti

Testování použitelnosti testuje aplikaci z pohledu uživatelů. Uživatel dostane několik úkolů popsaných scenáři, které musí splnit a následně je porovnáván postup či samotný úspěch řešení. Jednotlivé testovací scénáře jsou umístěny v příloze B.

```
final tUser = User(
    username: 'test username',
    language: 'test language',
    gamesCount: 3,
    winGamesCount: 1,
);

group('fromJson', () {
    test(
        'should return a~valid model when the JSON number'
        'is an integer',
        () async {
            // arrange
            final Map<String, dynamic> jsonMap =
                json.decode(fixture('user.json'));
            // act
            final result = User.fromJson(jsonMap);
            // assert
            expect(result, tUser);
        },
    );
});
```

Výpis kódu 7.2: Test serializace

Testování použitelnosti se zúčastnilo celkem 11 osob. Skupina respondentů byla složena z osob různého pohlaví a věku. Všichni respondenti měli mobilní zařízení s operačním systémem Android.

Testování proběhlo pomocí videohovorů za přítomnosti vývojáře a respondenta. Každý respondent dostal sadu úkolů dle scénářů z přílohy B a byl seznámen s podstatou herní aplikace. Tyto úkoly popsané scénáři mají za cíl najít či odhalit chyby nebo nedostatky v aplikaci z pohledu uživatele.

7.2.1 Průběh testování

Scénář B.1 se zabývá přihlášením do aplikace. Respondenti neměli s vyhledáním tlačítka přihlášení a s vyplněním registrace žádný problém. Obrazovka přihlášení a registrace byla dle respondentů přehledná a srozumitelná.

Ve scénáři B.2 měli respondenti za úkol zobrazit informace o aplikaci. Tento úkol splnili všichni bez potíží a po stisku tlačítka *O aplikaci* se přesunuli bez komplikací na obrazovku s potřebnými informacemi.

Ve scénáři B.3 nejprve vývojář vytvoří hru a respondent má za úkol se do této hry připojit pomocí obdrženého speciálního kódu. Většina respondentů bez problémů našla tlačítko směřující na obrazovku připojení do hry, avšak tři respondenti měli menší obtíže dané tlačítko vyhledat. Pro dané tlačítko byla proto zvolena vhodnější ikona. Na zobrazené obrazovce neměl již žádný z respondentů problém vyplnit kód a připojit se do místnosti.

Ve scénáři B.4 je obrácený úkol oproti předešlému scénáři. Respondent má nyní za úkol vytvořit danou hru. Najít příslušné tlačítko nebyl pro žádného respondenta problém a respondenti hodnotili velmi kladně výraznost prvku. Volba mise kapitoly a následné spuštění hry proběhlo bez komplikací.

Ve scénáři B.5 měli respondenti za úkol zobrazit profil a statistiky uživatele. Oba úkoly proběhly bez komplikací. Tlačítko pro přesun na příslušnou obrazovku našli všichni respondenti.

Ve scénáři B.6 bylo úkolem odhlášení se z aplikace. Většina respondentů tento úkol správně splnila přechodem na obrazovku menu a kliknutím na příslušnou ikonu tlačítka odhlášení. Dva respondenti tlačítko neviděli, avšak po menší pomoci i oni tlačítko úspěšně našli.

Výstup tohoto testování je hodnocen velmi kladně. Respondenti aplikaci a její přehlednost hodnotili dobře a ve většině případů neměli problém s navigací a vyhledáním potřebných informací.

Kapitola 8

Sestavení a distribuce aplikace

Pro testování výsledků aplikace je nutné aplikaci spustit. Pro spuštění je však nutné aplikaci sestavit. Sestavení aplikace za vývojáře obstará framework, zatímco vývojář se musí ujistit, že správně nainstaloval a nakonfiguroval příslušné nástroje. Před samotným sestavením aplikace je nutné provést kroky k nastavení prostředí a příslušných nástrojů:

1. Nainstalovat programovací jazyk Dart. (Tento krok nemusí být vyžadován, avšak některé externí nástroje s ním pracují.)
2. Nainstalovat framework Flutter.
3. Nainstalovat balíčky Dart a Flutter pro dané IDE.
4. Nainstalovat Android Studio pro platformu Android a Xcode pro platformu iOS.
5. Je-li potřeba, nainstalovat emulátor Android či iOS zařízení.

Správné nastavení lze ověřit spuštěním nástroje `flutter doctor`. Po nastavení prostředí a příslušných nástrojů přichází na řadu nastavení služeb, které aplikace využívá. Konkrétně je nutné přidat konfigurační soubory pro zprovoznění služby Firebase. Konfigurační soubor lze stáhnout z webové aplikace Firebase Console, kde se také vytváří a konfiguruje příslušná databáze Cloud Firestore.

Po tomto nastavení je možno aplikaci sestavit, například pro účely ladění. Další kroky nastavení pro distribuce aplikace pro danou platformu jsou zmíněny v dalších sekcích. Aplikaci sestavíme pomocí příkazu `flutter build`. Je také možné aplikaci přímo spustit buď na daném emulátoru, nebo na připojeném fyzickém zařízení. V případě fyzického zařízení je vhodné se ujistit, že je zařízení řádně připojeno a detekováno, pomocí příkazu `flutter devices`. Po úspěšném detekování zařízení lze aplikaci sestavit a spustit na vybraném zařízení pomocí příkazu `flutter run`.

V následujících sekcích jsou popsány postupy k distribuci aplikace na platformách Android a iOS. Následně je také popsáno, jak bude probíhat sestavení a distribuce na dalších potencionálních platformách, které budou přidány v průběhu dalšího vývoje frameworku Flutter.

8.1 Android

Pro sestavení aplikace na platformě Android je nutné stáhnout a přidat soubor `android/app/google-services.json`, který obsahuje nastavení služby databáze Cloud Firestore. Aplikaci je vhodné přidat vlastní ikonu, která se uloží do složky `android/app/src/main/res/`. Pro samotnou distribuci je nutné sestavit aplikaci s podepsaným certifikačním klíčem, jinak aplikace nepůjde publikovat v obchodě Google Play. Tento klíč se vygeneruje pomocí nástroje `keytool` a je nutné na něj odkázat ze souboru `android/key.properties`. V souboru `android/app/src/main/AndroidManifest.xml` lze také nastavit položku `application android:label` pro nastavení názvu aplikace. V souboru `android/app/build.gradle` lze naopak nastavit údaje pro distribuci, konkrétně položka `applicationId` obsahuje unikátní identifikátor aplikace a položky `versionCode` a `versionName` obsahují název a číslo konkrétního vydání aplikace. [46]

Příkaz `flutter build appbundle` sestaví výslednou aplikaci pro distribuci, kde *app bundle* je preferovaný formát pro distribuci, oproti klasickému formátu *apk*. Následně je možné distribuovat aplikaci přes rozhraní Google Play Console do obchodu Google Play.

8.2 iOS

Pro sestavení aplikace na platformě iOS je nutné stáhnout a přidat soubor `ios/Runner/GoogleService-info.plist`, který obsahuje nastavení služby databáze Cloud Firestore. V souboru `ios/Runner/Info.plist` je možné nastavit položku `CFBundleName` pro změnu názvu aplikace. Ve stejném souboru lze také nastavit položky `CFBundleShortVersionString` a `CFBundleVersion`, reprezentující název a číslo konkrétního vydání. Nastavení názvů a dalších údajů, včetně certifikátů a ID aplikace, lze nastavit v programu Xcode. Program Xcode také za vývojáře automaticky spravuje certifikát. [47]

Příkaz `flutter build ios` sestaví výslednou aplikaci pro distribuci. Následně je možné distribuovat aplikaci do obchodu Apple Store.

8.3 Další platformy

Framework Flutter momentálně podporuje pouze platformy Android a iOS. Platformy jako je web a desktop jsou dostupné pro testování. Aplikace je díky univerzálnosti frameworku připravena k rozšíření i na další platformy [48].

Stejně jako pro mobilní platformy, i tyto platformy budou muset nakonfigorovat databázi Cloud Firestore. Potřeba konfigurace ostatních služeb závisí na potřebách jednotlivých balíčků. Předpokladem je, že části aplikace využívající například senzory, budou muset být pro tyto platformy upraveny, či zcela vynechány, jelikož platformy web ani desktop běžně používanými senzory nedisponují. Platforma web se sestaví pomocí příkazu `flutter build web`. Sestavení je dostupné ve složce `build/web/` a soubory jsou automaticky minifikovány.

Závěr

Cílem této bakalářské práce bylo analyzovat, navrhnout a implementovat kooperativní multiplatformní mobilní herní aplikaci.

Na základě cílů je v teoretické části práce provedena rešerše konkurenčních aplikací a možností technologií pro multiplatformní framework, správu stavů, databáze, senzory, architekturu a testování. Po rešerši technologií, která čítá popis několika možností pro výběr technologie, byly vybrány jako nejlepší možnosti multiplatformní framework Flutter, knihovna pro správu stavů Bloc, databáze Cloud Firestore a architektura Clean Architecture.

V rámci analýzy byly popsány principy hry. Principy byly podrobně analyzovány a na základě toho byly vytvořeny požadavky a následně případy užití, které popisují několik scénářů průchodů aplikace.

Dle rešerše a analýzy bylo navrženo uživatelské rozhraní, které dbá na jednoduchý a srozumitelný vzhled, a architektura aplikace. Další kapitoly se zabývají popisem implementace a testování, kde byly uvedeny zajímavé ukázky kódu a konkrétní implementace částí aplikace. Aplikace byla implementována a otestována testy, které jsou automatizované a jsou spouštěné například v rámci průběžné integrace.

Na základě implementace byla vytvořena uživatelská a vývojářská dokumentace. Uživatelská dokumentace vhodně provádí uživatele všemi scénáři průchodu aplikací a možnostmi nastavení.

Všechny cíle této práce byly splněny. Aplikaci lze navíc v budoucnu rozšířit mnoha způsoby, ať už vylepšením samotného obsahu hry, herních misí, dostupnými moduly, tak i vylepšením o zcela nové funkce. Příkladem možných funkcí mohou být například přidání podpory pro webové prohlížeče, využití nových typů senzorů a práce s nimi, nebo rozšíření hry o mód s více hráči.

Bibliografie

1. GOOGLE, Inc. Simple app state management. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: `https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple`.
2. MOBX.DART. *Core Concepts* [online]. 2020 [cit. 2020-03-30]. Dostupné z: `https://mobx.netlify.com/concepts`.
3. RYAN, John. *Redux.dart Basics* [online]. 2018 [cit. 2020-03-30]. Dostupné z: `https://github.com/johnpryan/redux.dart/blob/master/doc/basics.md`.
4. ANGELOV, Felix. Core Concepts. In: *Bloc Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: `https://bloclibrary.dev/#/coreconcepts`.
5. MOLOCHKO, Alexander. *Clean Architecture : Part 1 – Database vs Domain* [online]. 2017 [cit. 2020-04-07]. Dostupné z: `https://crosp.net/blog/software-architecture/clean-architecture-part-1-databse-vs-domain`.
6. GOOGLE, Inc. An introduction to unit testing. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: `https://flutter.dev/docs/cookbook/testing/unit/introduction`.
7. GOOGLE, Inc. An introduction to widget testing. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: `https://flutter.dev/docs/cookbook/testing/widget/introduction`.
8. WEPC. *2020 Video Game Industry Statistics, Trends & Data* [online]. 2020 [cit. 2020-03-21]. Dostupné z: `https://www.wepc.com/news/video-game-statistics`.
9. SEABAA. *DUAL!* [software]. 2019. Verze 1.4.02 [cit. 2020-03-28]. Dostupné z: `https://play.google.com/store/apps/details?id=com.Seabaa.Dual`.

10. BYRIL. *Sea Battle 2* [software]. 2020. Verze 2.2.4 [cit. 2020-03-28]. Dostupné z: https://play.google.com/store/apps/details?id=com.byril.seabattle2.

11. STEEL CRATE GAMES, Inc. *Keep Talking and Nobody Explodes* [software]. 2019. Verze 1.9.3 [cit. 2020-03-28]. Dostupné z: https://play.google.com/store/apps/details?id=com.steelcrategames.keeptalkingandnobodyexplodes.

12. HENRY SMITH, Inc. *Spaceteam* [software]. 2019. Verze 2.8 [cit. 2020-03-28]. Dostupné z: https://play.google.com/store/apps/details?id=com.sleepingbeastgames.spaceteam.

13. LELER, William. What's Revolutionary about Flutter. In: *Hacker Noon* [online]. 2017 [cit. 2020-03-17]. Dostupné z: https://hackernoon.com/whats-revolutionary-about-flutter-946915b09514.

14. KUKUSHKINA, Natalia. Mobile App Development Frameworks in 2019. In: *DashMagazine* [online]. 2019 [cit. 2019-11-26]. Dostupné z: https://codeburst.io/mobile-app-development-frameworks-in-2019-f8fb2ece20a8.

15. Stack Overflow Trends. In: *Stack Overflow* [online]. 2020 [cit. 2020-03-29]. Dostupné z: https://insights.stackoverflow.com/trends?tags=flutter%2Creact-native%2Cionic-framework.

16. *Ionic - Cross-Platform Mobile App Development* [online]. 2019 [cit. 2020-03-29]. Dostupné z: https://ionicframework.com.

17. *React Native · A framework for building native apps using React* [online]. 2015 [cit. 2020-03-29]. Dostupné z: https://reactnative.dev.

18. *Flutter - Beautiful native apps in record time* [online]. 2020 [cit. 2020-03-29]. Dostupné z: https://flutter.dev.

19. GOOGLE, Inc. Web support for Flutter. In: *Flutter Documentation* [online]. 2020 [cit. 2020-04-03]. Dostupné z: https://flutter.dev/web.

20. GOOGLE, Inc. Desktop support for Flutter. In: *Flutter Documentation* [online]. 2020 [cit. 2020-04-03]. Dostupné z: https://flutter.dev/desktop.

21. *Skia Graphics Library* [online]. 2020 [cit. 2020-03-29]. Dostupné z: https://skia.org.

22. GOOGLE, Inc. Technical overview. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-29]. Dostupné z: https://flutter.dev/docs/resources/technical-overview.

23. GOOGLE, Inc. Start thinking declaratively. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: https://flutter.dev/docs/development/data-and-backend/state-mgmt/declarative.

24. GOOGLE, Inc. Differentiate between ephemeral state and app state. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: https://flutter.dev/docs/development/data-and-backend/state-mgmt/ephemeral-vs-app.

25. SOARES, Paolo. Flutter / AngularDart – Code sharing, better together (DartConf 2018). In: *Google Developers* [online]. 2020 [cit. 2020-03-30]. Dostupné z: https://www.youtube.com/watch?v=PLHln7wHgPE.

26. GALVÃO, Kelven. Analyzing BLoC and MobX. In: *Flutterando* [online]. 2020 [cit. 2020-03-30]. Dostupné z: https://medium.com/flutterando/analyzing-bloc-and-mobx-7118053e913a.

27. ANGELOV, Felix. Why Bloc? In: *Bloc Documentation* [online]. 2019 [cit. 2020-03-30]. Dostupné z: https://bloclibrary.dev/#/whybloc.

28. HARVEY, Robert. Why use a database instead of just saving your data to disk? In: *Stack Overflow* [online]. 2013 [cit. 2020-03-28]. Dostupné z: https://softwareengineering.stackexchange.com/a/190483/265114.

29. ASHWINI, Amit. *Should You Use NoSQL Or SQL Db Or Both?* [online]. 2017 [cit. 2020-04-02]. Dostupné z: https://www.cognitiveclouds.com/insights/should-you-use-a-nosql-db-sql-database-or-both/.

30. GOOGLE, Inc. *Cloud Firestore* [online] [cit. 2020-04-01]. Dostupné z: https://firebase.google.com/docs/firestore.

31. GOOGLE, Inc. *Firebase* [online] [cit. 2020-04-01]. Dostupné z: https://firebase.google.com.

32. GAUR, Lavanya. Sensors in Android. In: *Heartbeat* [online]. 2020 [cit. 2020-04-03]. Dostupné z: https://heartbeat.fritz.ai/sensors-in-android-215df2c618de.

33. GOOGLE, Inc. Sensors Overview. In: *Android Developers* [online] [cit. 2020-04-03]. Dostupné z: https://developer.android.com/guide/topics/sensors/sensors_overview.

34. GOOGLE, Inc. Motion sensors. In: *Android Developers* [online] [cit. 2020-04-03]. Dostupné z: https://developer.android.com/guide/topics/sensors/sensors_motion.

35. GOOGLE, Inc. Position sensors. In: *Android Developers* [online] [cit. 2020-04-03]. Dostupné z: https://developer.android.com/guide/topics/sensors/sensors_position.

36. MARTIN, Robert C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. In: Prentice Hall, 2018, s. 31, 57–91, 135–146, 303–321. ISBN 0134494164. Dostupné také z: https://www.amazon.com/Clean-Architecture-Craftsmans-Software-Structure-ebook/dp/B075LRM681.

37. MARTIN, Robert C. *The Clean Architecture* [online]. 2012 [cit. 2020-04-07]. Dostupné z: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.

38. REŠETÁR, Matej. Flutter TDD Clean Architecture Course [1] – Explanation & Project Structure. In: *Reso Coder* [online]. 2019 [cit. 2020-04-11]. Dostupné z: https://resocoder.com/2019/08/27/flutter-tdd-clean-architecture-course-1-explanation-project-structure.

39. GEORGE, Boby; WILLIAMS, Laurie. An Initial Investigation of Test Driven Development in Industry. In: *Proceedings of the 2003 ACM Symposium on Applied Computing*. Melbourne, Florida: Association for Computing Machinery, 2003, s. 1135–1139. SAC '03. ISBN 1581136242. Dostupné z DOI: 10.1145/952532.952753.

40. GOOGLE, Inc. Testing Flutter apps. In: *Flutter Documentation* [online]. 2019 [cit. 2020-03-30]. Dostupné z: https://flutter.dev/docs/testing.

41. GOOGLE, Inc. *analyzer | Dart Package* [online]. 2020 [cit. 2020-04-13]. Dostupné z: https://pub.dev/packages/analyzer.

42. MCCABE, Maurice. *Combined code coverage for Flutter and Dart* [online]. 2018 [cit. 2020-04-06]. Dostupné z: https://medium.com/@nocnoc/combined-code-coverage-for-flutter-and-dart-237b9563ecf8.

43. MARTIN, Robert C. *The Three Laws of TDD* [online]. 2005 [cit. 2020-02-11]. Dostupné z: http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd.

44. WRIGHT, Graham. *Continuous Integration (CI)* [online]. 2018 [cit. 2020-03-27]. Dostupné z: https://medium.com/@gwright_60924/continuous-integration-ci-e81032bb8502.

45. LABUNSKIY, Evgeniy. What comes first: Functional or non-Functional Requirements? In: *Scrum Ukraine* [online]. 2017 [cit. 2020-04-13]. Dostupné z: https://medium.com/agiletransformation/what-comes-first-functional-or-non-functional-requirements-b3ee96424742.

46. GOOGLE, Inc. Build and release an Android app. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: https://flutter.dev/docs/deployment/android.

47. GOOGLE, Inc. Build and release an iOS app. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: `https://flutter.dev/docs/deployment/ios`.

48. GOOGLE, Inc. Build and release a web app. In: *Flutter Documentation* [online]. 2020 [cit. 2020-03-30]. Dostupné z: `https://flutter.dev/docs/deployment/web`.

Příloha A

Seznam použitých zkratek

UI User interface

BLoC Business Logic Component

SDK Software Development Kit

IDE Integrated Development Environment

AOT Ahead of Time

JIT Just in Time

CI Continuous Integration

SQL Structured Query Language

NoSQL Non SQL

Příloha B

Testovací scénáře

B.1 Přihlášení do aplikace

Modelová situace

Stáhli jste si aplikaci z obchodu s aplikacemi na vaší mobilní platformě. Aplikaci jste spustili, avšak pro spuštění hry je nutné přihlášení. Přihlašte se, popřípadě se zaregistrujte.

Postup

1. Na úvodní obrazovce nalezněte tlačítko pro přihlášení a klikněte na něj.

2. Na obrazovce pro registraci vyplňte údaje a klikněte na tlačítko pro přihlášení.

B.2 Zobrazení informace o aplikaci

Modelová situace

Chcete si zobrazit informaci o aplikaci. Zkuste tyto informace v aplikaci vyhledat a najít informace o jménu tvůrce, verzi aplikaci a odkazu na repozitář.

Postup

1. Přesuňte se na hlavní obrazovku.

2. Nalezněte tlačítko směřující na informace o aplikaci.

3. Na zobrazené obrazovce najděte potřebné informace.

B.3 Připojení do hry

Modelová situace

Dostali jste speciální kód pro připojení do hry, do které se chcete připojit. Zkuste se do již vytvořené hry připojit.

Postup

1. Přesuňte se na hlavní obrazovku.
2. Klikněte na tlačítko pro připojení se do již probíhající hry.
3. Na zobrazené obrazovce vyplňte obdržený speciální kód.
4. Potvrďte připojení.

B.4 Vytvoření hry

Modelová situace

Chcete vytvořit vlastní hru, do které se bude moci připojit další uživatel. Zkuste najít tlačítko, které vás přesune do vytváření místnosti. Následně najděte speciální kód, který předáte druhému uživateli. Zkuste nastavit hru na 2. misi 1. kapitoly. Najděte způsob, jak zapnout hru a hrát.

Postup

1. Přesuňte se na hlavní obrazovku.
2. Klikněte na tlačítko pro vytvoření nové hry.
3. Na zobrazené obrazovce najděte speciální kód, který předejte druhému uživateli.
4. Najděte nastavení pro volbu mise a zvolte 2. misi 1. kapitoly.
5. Signalizujte, že jste připraven ke spuštění hry.
6. Až bude připraven i druhý uživatel, klikněte na pro spuštění hry.

B.5 Zobrazení profilu a statistik

Modelová situace

Hráli jste již několik her. Chcete zobrazit svůj profil a vaše statistiky, abyste se mohli podívat na vaše výsledky. Najděte obrazovku profilu. Najděte statistiky vašeho profilu.

Postup

1. Přesuňte se na hlavní obrazovku.
2. Klikněte na tlačítko pro zobrazení profilu uživatele.
3. Na obrazovce najděte informace o profilu a vašich statistikách.

B.6 Odhlášení z aplikace

Modelová situace

Chcete se odhlásit z aplikace. Přesuňte se na hlavní obrazovku a odhlaště se.

Postup

1. Přesuňte se na hlavní obrazovku.
2. Klikněte na tlačítko pro odhlášení se z aplikace.

Příloha C

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `doc` ..... adresář s dokumentací
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Bittner_Jan.pdf` ..... text práce ve formátu PDF