ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Diplomová práce

Návrh aplikace pro snížení digitálních závislostí

Bc. Serhii Sinelnikov

Květen 2022
Vedoucí práce: doc. Ing. Daniel Novák, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Sinelnikov** Jméno: **Serhii** Osobní číslo: **496373**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Návrh aplikace pro snížení digitálních závislostí

Název diplomové práce anglicky:

Design of Therapeutical Application for Digital Addiction Reduction

Pokyny pro vypracování:

1) Make a literature review about digital addictions and current solutions on the market.
2) Design a mobile app to help reduce addiction on new media in Flutter framework.
3) Test the application on 5 users at least. The application must include registration/authorization, several sessions for setting restrictions, rules and screen free zones, ability to change them later and daily control. The app must work online and offline.

Seznam doporučené literatury:

K. Lukavska,The effects of parental control on problematic internet use in adolescents: A prospective cohort study, Journal of Behavioral Addictions, 2020
K. Lukavska, The immediate and long-term effects of time perspective on Internet gaming disorder, Journal of Behavioral Addictions, 2020
Vondrackova,P., Gabrhelık, R .(2016).Prevention of internet addiction:A systematicr eview.J ournal of Behavioral Addicitons,5(4), 568–579

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Daniel Novák, Ph.D. Analýza a interpretace biomedicínských dat FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **07.02.2022** Termín odevzdání diplomové práce: **20.05.2022**

Platnost zadání diplomové práce: **30.09.2023**

doc. Ing. Daniel Novák, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Poděkování / Prohlášení

Chtěl bych poděkovat vedoucímu doc. Ing. Danielu Novákovi, Ph.D. za odborné vedení této práce, podporu, a také za možnost častých konzultací. Dále bych chtěl poděkovat Bc. Eriku Gadireddi a Bc. Jakubu Lhotákovi za spoluprací na projektu a cenné rady během vývoje. Nakonec bych rád poděkoval své rodině za podporu během studia.

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.

Abstrakt / Abstract

Tato diplomová práce se zabývá analýzou, návrhem a implementací mobilní aplikace Digital Parenting, která řeší problematiku digitální závislosti u děti. Hlavním cílem aplikace je seznámení rodičů s daným problémem a poskytnutí doprovodu pro komunikaci s dítětem ohledné tohoto problému.

Analýza nezbytně obsahuje popis problematiky digitální závislosti a existující řešení na trhu. Dále analýza obsahuje popis různých nástrojů pro tvorbu multiplatformních mobilních aplikaci. Detailněji byl popsán vývojový nastroj Flutter, princip vytváření aplikace v daném frameworku a jeho zásadní částí.

Na začátku návrh zahrnuje přehled funkčních a nefunkčních požadavků kladených na aplikaci. Potom, jsou popsané prvky aplikace, datový model, použité knihovny a technologie.

V implementační částí jsou uvedené struktura a zajímavé částí implementace. Práce obsahuje vhodné uživatelské testovaní a jeho výsledky.

Klíčová slova: Flutter, Digitální závislost, Mobilní aplikace, BLoC

The thesis focuses on the analysis, design and implementation of the mobile application Digital Parenting, which resolves the issue of digital addiction of the children. The main goal of the application is to acquaint parents with the problem and provide support for communication with the child regarding this problem.

The analysis necessarily contains a description of the issue of digital addiction and existing solutions on the market. Furthermore, the analysis contains a description of various tools for creating multiplatform mobile applications. The Flutter development framework was described in more detail, the principle of creating an application in a given framework and its essential parts.

The first part of design includes an overview of functional and non-functional requirements for applications. Then, the functions of the application, the data model, the libraries used and the technology are described.

The developing process and interesting parts of the implementation are presented in the implementation part. The work contains appropriate user testing and its results.

Keywords: Flutter, Digital addiction, Mobile application, BLoC

Title translation: Design of Therapeutical Application for Digital Addiction Reduction

Obsah /

/ Obrázky

Úvod

Každý rok se technologie extrémně rozvíjí a více dětí začíná používat internet a elektronické zařízení již od raného věku. Většina rodičů je přesvědčena o tom, že je to normální a nevidí v tom problém. Aplikace `Digital Parenting` se snaží poučit rodiče o problému digitální závislosti u dětí a představuje, jak tomu předejít.

V rámci projektu spolupracuje více lidí: psychologové, backend a iOS vývojáři. Tato aplikace navazuje na privátní datové úložiště, které bylo vytvořeno speciálně pro tento projekt.

Vybral jsem si tuto práci, protože rád vytvářím mobilní aplikace a zaujal mě tento problém. Rozhodl jsem tuto aplikaci napsat ve frameworku Flutter, aby byla přístupná na všech platformách a aby oslovila co nejvíce lidí.

Kapitola 1
Cíl

Hlavním cílem této diplomové práce je navrhnout a vytvořit aplikaci vedoucí ke snížení digitální závislosti u dětí. Teoretická část práce se bude věnovat problematice digitálních závislostí a analýze podobných aplikací, které se danému tématu věnují. Jelikož je aplikace vyvíjena ve frameworku Flutter, analýza nezbytně obsahuje zavedení pojmu multiplatformního vývoje a porovnání nejpoužívanějších nástrojů. Nakonec sleduje popis samotného Flutter frameworku a přístupu jeho vývoje.

Návrh obsahuje funkční a nefunkční požadavky, které byly sestaveny po konzultaci s vedoucím práce. Dále bude popsána funkcionalita aplikace a použité technologie a knihovny.

V implementaci budou popsána zajímavá řešení při vývoje aplikace. Na závěr bude provedeno uživatelské testování podle předem navrhnutých scénářů.

Kapitola 2
Analýza

V této kapitole se zaměřím na analýzu problematiky digitálních závislostí a na existující řešení na trhu. Dále budu analyzovat technickou část práce. To znamená, že popíšu problematiku multiplatformního vývoje, několik přístupu k tvorbě multiplatformních aplikaci, a detailněji framework Flutter.

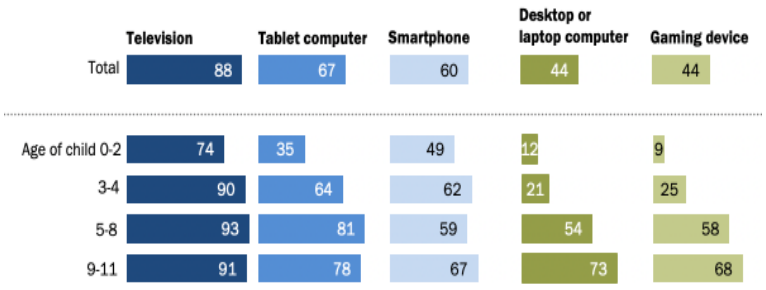
2.1 Digitální závislost

`Digitální závislost` je škodlivá závislost na digitálních médiích a zařízeních, jako jsou chytré telefony, videohry a počítače. Někteří psychologové se domnívají, že závislost na elektronických zařízeních a médiích by měla být klasifikována podobně jako poruchy spojené s užíváním návykových látek. Studie zjistily silnou korelaci mezi používáním vysokofrekvenčních digitálních médií a poruchami duševního zdraví, jako je deprese a úzkost [1].

Digitální závislosti můžeme rozdělit na následující druhy:

- nadměrné používání internetu,
- závislost na internetu (IA),
- problematické používání internetu [2],
- hraní online her [3] a další.

Mnoho studií poukazuje na negativní účinky hraní, jako jsou fyzické problémy, problémy v osobním životě (tj. konflikty s přáteli nebo s rodinou), akademické problémy (tj. absence ve škole a zhoršený výkon) atd [3].



Obrázek 2.1. Procenta amerických rodičů dítěte, kteří vědí, že jejich dítě někdy používá nebo interaguje s elektronickými zařízeními [4]

Tato statistika je založena na průzkumu provedeném mezi 2. až 15. březnem roku 2020 mezi 3 640 rodiči v USA, kteří mají alespoň jedno dítě nebo děti do 17 let.

Na obrázku 2.1 vidíme, že většina dětí do 2 let rovněž interaguje se zařízeními.

To se neshoduje s tím, že velké množství průzkumů nedoporučuje používat žádné zařízení dětem mladším 2 let, ale doporučují trávit tzv. 'sedavý čas' (čas, který netráví pohybem) poslechem četby či jinou interakcí s pečující osobou [5].

2.1.1 Detekce digitální závislosti

Hodně rodičů neví, jak digitální závislost u dítěte odhalit, protože neví, jaké jsou příznaky, jelikož jsou často velmi individuální. Dále jsou uvedeny nejběžnější příznaky, podle kterých je závislost možné identifikovat:

- Odstoupení od dříve příjemných činností ve prospěch používání digitálních médií.
- Chování, které překáží každodennímu fungování.
- Poruchy spánku.
- Chování, které způsobuje konflikty ve vztazích.
- Chování, které narušuje školní výkon.
- Lhaní nebo skrývání používání digitálních médií [1].

Pokud dítě má alespoň jeden z těchto příznaků, je potřeba vzít v úvahu jeho chovaní a zjistit jeho příčinu.

2.1.2 Doporučeni

Pro děti ve věku od 2 do 4 let se doporučuje trávit u obrazovky maximálně 60 minut denně. Také se doporučuje stanovení pevných pravidel, jako je například vyhýbání se obrazovkám během jídla či 1 hodinu před spaním. Pro starší děti, ve věku od 6 do 10 let, se doporučení mění v popis zásad a otázek k diskuzi s rodiči a dětmi samotnými. V takovém věku děti mají tendence porušovat a nerespektovat rodičem ustanovená pravidla. Je doporučeno vytvořit mediální plán domácnosti, do kterého se zapíšou časové limity, zásady a pravidla o používání obrazovek, které dítě musí respektovat. Více než 50 % dětí starších 11 let trpí nedostatkem fyzické aktivity a mají výrazné patologické symptomy (závislost, depresivita, snižující se spokojenost se životem). V tomto věku se rodičům doporučuje proces užívání obrazovek moderovat, nebo alespoň monitorovat [5].

Mgr. Kateřina Lukavská, Ph.D. ve svém článku uvádí, že v České Republice chybí kvalitní primárně preventivní programy proti poruchám spojeným s užíváním moderních digitálních technologií [5].

2.1.3 Negativní následky

Digitální závislost u dítě provokuje radu sociálních a fyzických negativních následku:

- Problémy se snem – v noci dítě tráví čas na internetu místo toho aby odpočívalo.
- Problémy s vahou – mohou vzniknout problémy s nadbytečnou vahou, kvůli tomu že preferuje obrazovku místo fyzické aktivity. Navíc, při prohledávaní videi, se často neobejde bez rychlých kalorii(brambůrky, sycené napoje, atd.) [7]
- Zvýšená možnost rozvoje deprese, problémů s pozorností a dalších [6].

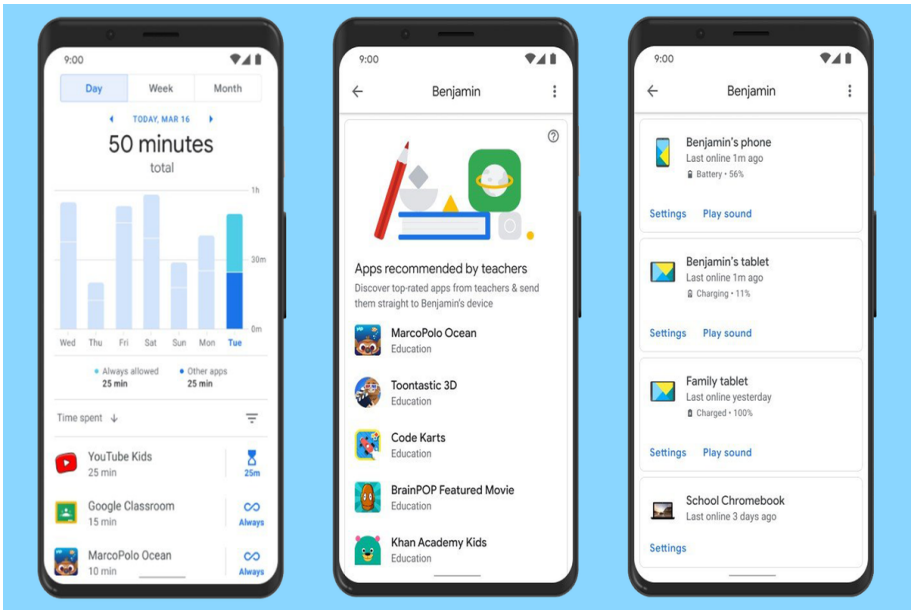
2.2 Analýza existujících řešení na trhu

V této sekci budou probrány a analyzovány podobné aplikace, které již existují v Play Market pro platformu Android, nebo v App Store pro platformu iOS. Testovaní aplikací, pokud to bude možné, bude prováděno na zařízení Xiaomi 9, verze Android 12.2 a na zařízení iPhone 12, verze iOS 15.3. Seznam analyzovaných aplikací byl sestaven na základě popularity aplikace a kladných recenzí uživatelů.

- Norton 360 Deluxe – je program od společnosti, která většinou vyrábí ochranný a antivirový software pro operační systémy Windows a MacOs. Jedním z jejich

produktů je balíček produktu Norton 360 Deluxe, součástí kterého je rodičovská kontrola a který je dostupný pro systémy Windows, Android a iOS. Hlavní funkce této aplikace nabízí rodičům nástroje, pomocí kterých uvidí, jaká videa děti sledují, jaké navštěvují weby, jaké vyhledávají termíny, aplikace, které stahují, k tomu navíc je připojeno sledování polohy GPS pro systémy Android a iOS či filtrování obsahu pro počítače PC. Také nabízí možnost spravovat více dětí (více zařízení) na jednom účtu. Rodičovská kontrola je součástí celí antivirové aplikace. Nejde však zakoupit pouze tato funkce. Aplikace je jednoduchá k použití a na základě hodnocení a recenzí vypadá, že uživatelé jsou spokojeni. Na trhu existují i další podobné aplikace od antivirových společností, například Kaspersky Safe Kids [8], Eset Parent Control, McAfee Total Protection apod. Všechny programy mají podobnou funkcionalitu jako Norton 360 Deluxe, většinou podporují platformy Windows, Android a iOS. Hodně takových aplikací má roční předplatné.

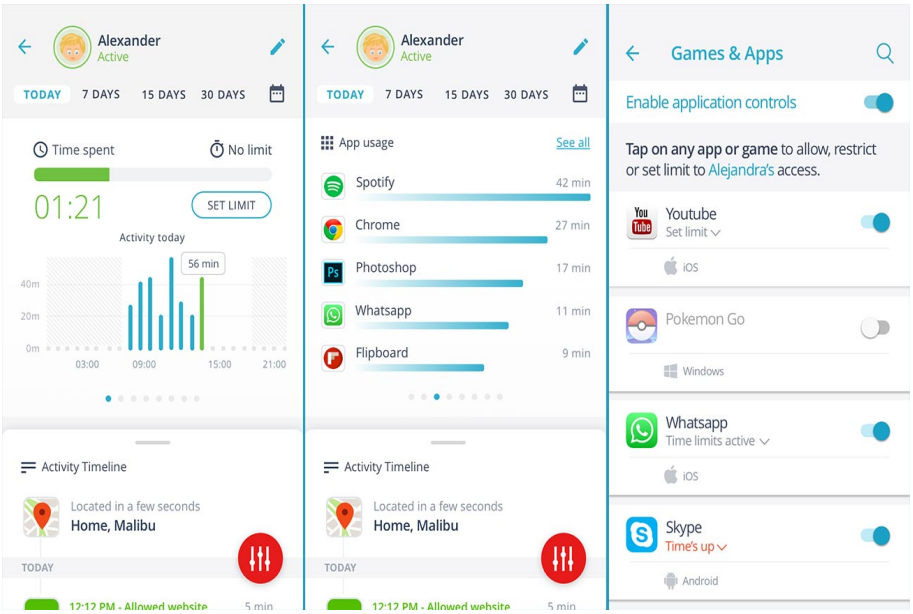
- Google Family Link – aplikace, která je přímo integrovaná do systému Android. Funkcionalita je podobná předchozí aplikaci, ale zde je všechno zadarmo. Pro použití aplikace je třeba mít Google účty rodiče a dítěte. Z analýzy recenzí vyplývá, že mnoho uživatelů má problém se spojením účtu dítěte a svého účtu, lokace navždy je správná a aplikace se můžou odblokovat, když mají být zablokované. Také si uživatelé stěžují na špatný UX přístup. Tato aplikace má stejné výhody a nevýhody jako předchozí, jenom je zadarmo a je přímo udělaná pro systém Android.



Obrázek 2.2. Google Family Link [9]

- Nastavení systému iOS – uživatelé systému iOS můžou nastavit rodičovskou kontrolu přímo v zařízení dítěte bez nainstalování externích aplikací. Na stránkách Apple je návod, jak a co jde změnit [10]. Takovým způsobem jde například nastavit čas strávený u telefonu, zákaz nákupu v iTunes a App Store, skrývání aplikací, zákaz některého obsahu apod.
- Qustodio Parental Control App – je multiplatformní aplikace pro Windows, Android a iOS zařízení, která má podobnou funkcionalitu jako Norton 360 Deluxe nebo Google Family Link, navíc k tomu má funkci sledování a natáčení obrazovky, čtení zpráv, prohlížení sociálních sítí a sledování toho, kdo dítěti volá. Uživatelské rozhraní je jednoduché a srozumitelné. V Play Marketu je mnoho kladných

hodnocení, většina lidí si stěžuje na nesrozumitelnost uživatelského rozhraní a na nějaké menší chyby. Je to dobrá alternativa Google Family Link, a to v případě, kdy rodiče a dítě mají na zařízení různé operační systémy.



Obrázek 2.3. Qustodio app [11]

Ve závěru můžeme říct, že nejlepší aplikace pro Android v porovnání ceny a kvality je Google Family Link, jelikož je zadarmo a má všechno potřebné pro sledování dítě. Pro iOS zařízení bude nejlepší volbou použít nastavení v mobilu, nebo použít Qustodio Parental Control.

Tyto aplikace sice nastavují limity pro děti, ale neuvádí rodiče do problematiky digitálních závislostí. Cílem naší aplikace je poučit rodiče o problematice, aby byli informováni o internetových hrozbách, ale také aby pochopili, proč je důležité nastavit limity. Dalším cílem je představení digitální závislosti jako celku.

2.3 Analýza přístupu

Existuje mnoho přístupů pro vytváření aplikace pro mobilní zařízení: nativní vývoj pro každou platformu zvlášť, multiplatformní řešení, mezi které patří Apache Cordova, Xamarin, React Native, Flutter.

Je nutné uvést, že aplikace na iOS již existuje a je popsána v bakalářské práci Erika Gadireddi [16]. Pro pokrytí zbývající části uživatelů je potřeba buď nativně vytvořit aplikace pro Android, nebo použít jedno z multiplatformních řešení.

Základní charakteristikou nativního vývoje je zaměření na určitou platformu. Pokud hovoříme o vývoji mobilních aplikací pro OS Android, budeme nazývat nativní aplikace programem, vyvinutým v programovacím jazyce *Java* nebo *Kotlin*, hlavně používající sadu nástrojů *Android SDK*. U vývoje pro iOS se používá *Swift* nebo *Objective-C*. Zkompilovaná aplikace může využívat všechny funkce, nabízené vývojáři operačního systému. U multiplatformního nebo hybridního přístupu můžeme získat přístup k těmto funkcionalitám pomocí knihoven třetích stran, což může vést k negativním důsledkům [17].

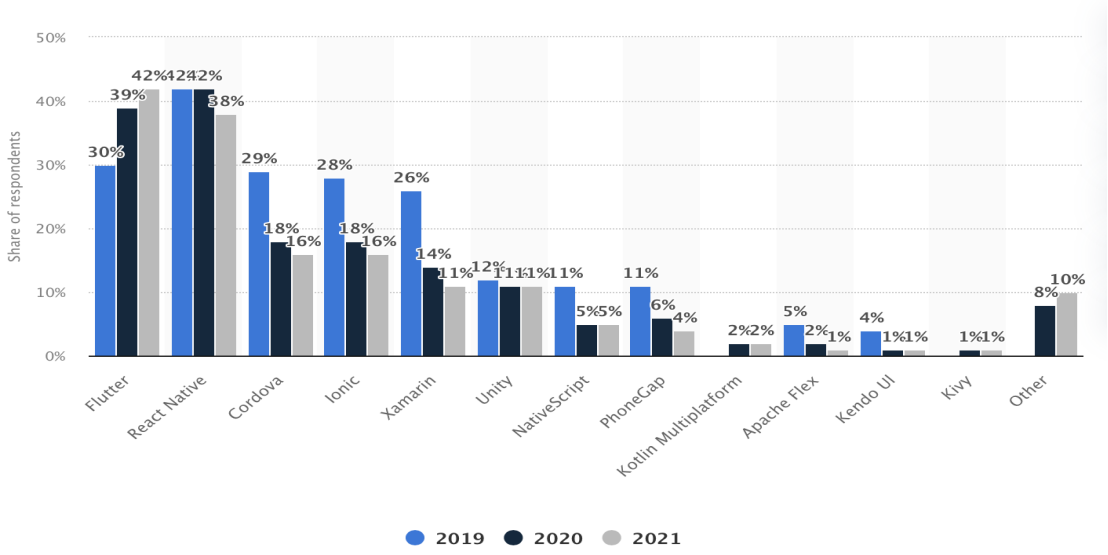
Základní myšlenka multiplatformního vývoje spočívá v optimalizaci procesů vývoje mobilní aplikace. Multiplatformní aplikace jsou tedy navrženy tak, aby bylo

možné s minimálním úsilím zkompilovat zdrojový kód pro provedení na několika mobilních platformách, ale výsledkem každé samostatné kompilace budou samostatně spustitelné soubory [17].

Výhody pro multiplatformní řešení jsou následující:

- `Efektivita nákladu` – Jeden tým, který se zabývá oběma platformami.
- `Snadná údržba` – Je jednodušší udržovat a zavádět změny, protože není nutné udržovat aplikace na každé platformě samostatně.
- `Identičnost` – Pixel perfekt na každém přístroji [13].

2.3.1 Porovnaní nástrojů



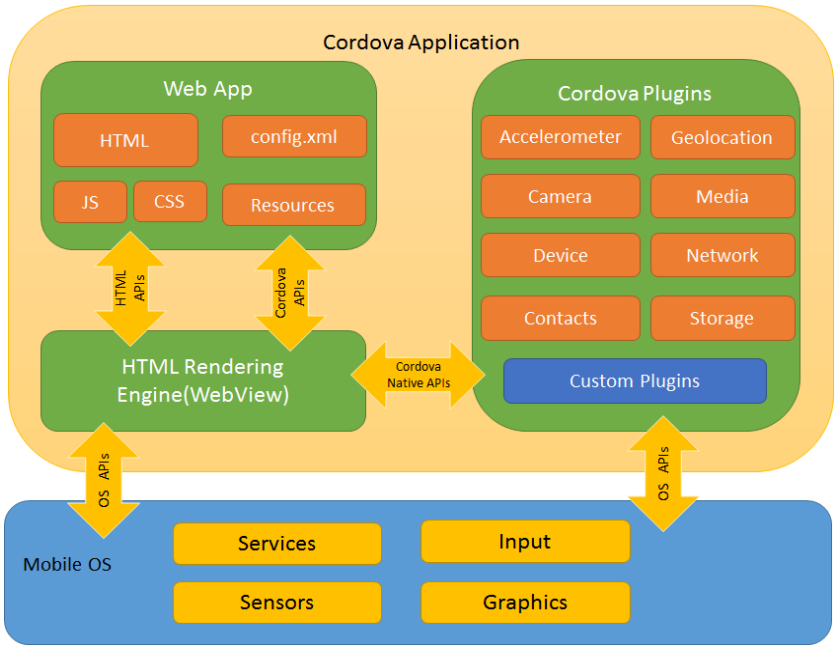
Obrázek 2.4. Multiplatformní frameworky používané po celém světě od roku 2019 do roku 2021 [12].

Obrázek 2.4 ukazuje graf, který porovnává oblíbenost různých multiplatformních řešení v letech od 2019 do 2021. Podle něho vidíme současnou situaci a také trend, které řešení se bude rozvíjet v bližší budoucnosti.

Vidíme trend, který ukazuje, že poslední 2 roky zájem o nástroje mimo Flutter klesá. V průměru pro všechny technologie mimo Flutter a React Native klesl zájem skoro o polovinu.

2.3.2 Apache Cordova

Apache Cordova (dříve PhoneGap) je framework pro vytvoření mobilních aplikací, který vytvořila společnost Nitobi v roce 2009. V roce 2011 byl PhoneGap rebrandován jako Apache Cordova a předán komunitě jako open-source. Umožňuje používat standardní webové technologie – HTML5, CSS3 a JavaScript pro multiplatformní vývoj. Aplikace jsou spuštěny v rámci balíčků zacílených na každou platformu uvnitř webového prohlížeče WebView. Spoléhají na vazby API kompatibilní se standardy, aby měly přístup ke schopnostem jednotlivých zařízení, jako jsou senzory, data, stav sítě atd [14].



Obrázek 2.5. Architektura Apache Cordova [14]

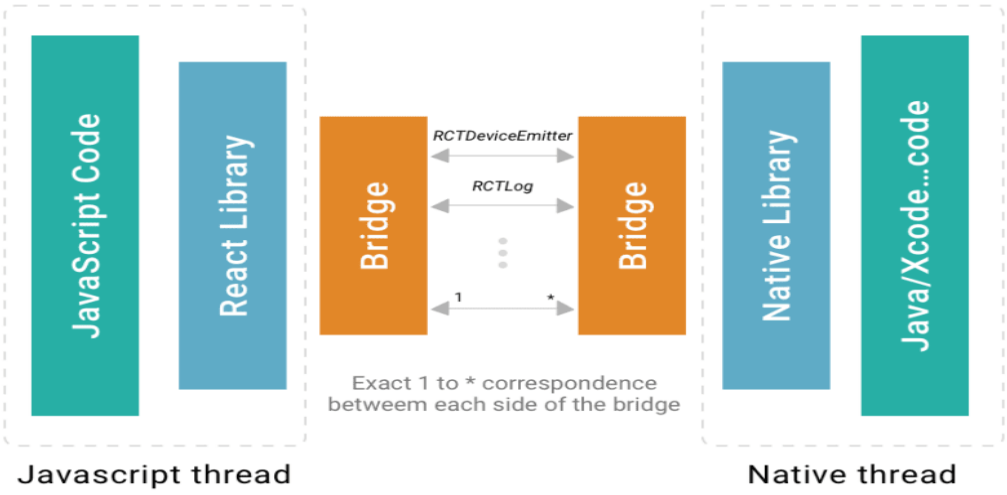
Hlavní nevýhodou přístupů založených na WebView je to, že neexistuje přístup k nativním prvkům uživatelského rozhraní a taková aplikace je fyzicky spuštěna uvnitř ovládacího prvku WebViev, což znamená: špatný výkon (zejména WebView na starších verzích Androidu) a obrovské problémy se zobrazováním.

2.3.3 React Native

React Native je javascriptový framework pro tvorbu multiplatformních mobilních aplikací. Je založen na Reactu, Javascriptové knihovně pro vytváření uživatelských rozhraní. Zatímco výpočetně těžké funkce mohou být implementovány s nativními moduly pro iOS a Android, zbytek kódu lze zapsat pomocí JavaScriptu a sdílet přes platformy. React Native používá takzvaný most, který zavolá nativní rozhraní pro vykreslování pro iOS a Android v Swiftu a Javě respektive. Výsledná aplikace je tedy vykreslena pomocí skutečných komponent mobilního uživatelského rozhraní a vypadá jako jakákoliv jiná nativní mobilní aplikace.

Nativní kontroly a nativní moduly v React Native zlepšují výkon. React Native vykresluje některé komponenty s nativními API, na rozdíl od jiných multiplatformových frameworků, jak je zmíněno výše. Zatímco přístup WebView výrazně snižuje výkon, React Native komunikuje s cílovými komponentami pro iOS nebo Android a vykresluje kód nativním API přímo a nezávisle. Tímto způsobem React používá samostatné vlákno z uživatelského rozhraní, což také zvyšuje výkon.

React Native se zabývá pouze uspořádáním uživatelského rozhraní. Nízkoúrovňové funkce OS, jako jsou práce se soubory, geopozice, kamera a.t.d., pracují prostřednictvím nativních modulů, který poskytují rozhraní do JavaScript kódu [17].



Obrázek 2.6. React Native most [15]

2.3.4 Flutter

Ve své bakalářské práci jsem se zabýval porovnáním vývojových nástrojů [17], zejména Flutter a React Native, které jsou stále nejpoužívanější a nejlépe hodnocené technologie.

Od roku 2019, kdy byla tato bakalářská práce napsána, se technologie posunuly o mnoho dále, například aplikace napsané ve Flutteru už fungují nejenom na Android a iOS, ale také pro Linux, Windows a především pro web. Aplikace Flutter jsou kompilovány přímo do strojového kódu, ať už jde o Intel x64 nebo ARM, nebo do JavaScriptu, pokud cílí na web.

2.3.4 Historie

V roce 2014, ve snaze o rychlost tým Google Chrome experimentoval s vykreslením obsahu stránek. Účelem experimentu bylo ověřit, zda bylo možné zrychlit vykreslování. Základem experimentu byl motor *Blink* a výsledkem bylo dvacetinásobné zvýšení výkonu díky následujícím změnám: jednoduchá sada omezení ovlivňujících polohu prvku na obrazovce (Box Model) je minimální a maximální šířka a výška. Namísto velké sady pravidel, které lze aplikovat na libovolný prvek, definuje každá komponenta svůj jednoduchý rozvrhový model (centrování, výplň, sloupec). Malý počet pravidel umožňuje výrazně optimalizovat layout na úrovni komponenty. Jednoduchý model omezení, který sahá od rodiče dolů po stromě, umožňuje lineární vztah mezi počtem prvků ve stromu a časem rozvržení. Všechno se děje tak rychle, že Flutter neodděluje „vytvoření" a „modifikaci" uzlů. Jestli se nadřazený widget změní, anuluje se celý podstrom. Výsledek tohoto experimentu byl vzat jako základ pro vývoj Flutteru [17].

Výjimečnost architektury Flutter je v tom, že Dart kód pracuje přímo s Canvasem, samostatně kreslí každý pixel, ovládá gesta i animaci. Nepoužívá widgety OEM jako React Native. Místo toho tým Flutter vytvořil dvě sady widgetů pro hlavní mobilní platformy: Materiál pro Android a Cupertino pro iOS. Tak překreslují všechny komponenty uživatelského rozhraní z obou mobilních platforem a zcela opakují jejich chování. S nízkoúrovňovým API (geolokace, zvuk, Bluetooth) komunikuje prostřednictvím Platform Channels [18].

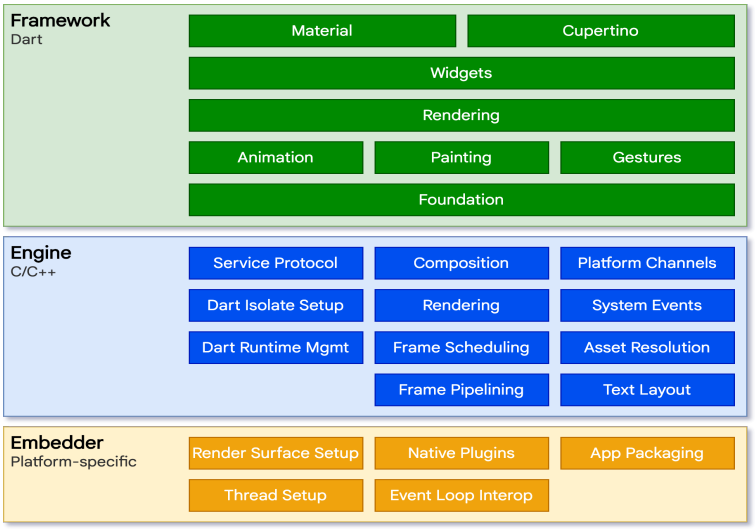
Příklady úspěšných aplikaci napsaných ve Flutteru:

- Alibaba
- Google Pay
- eBay

2.3.4 Flutter architektura

Jádrem Flutteru je Flutter engine, který je většinou napsán v C++ a podporuje primitiva nezbytná pro podporu všech Flutter aplikací. Engine je zodpovědný za vykreslení scén, kdykoli je to potřeba. Poskytuje nízkoúrovňovou implementaci základního API Flutter, včetně grafiky (prostřednictvím Skia), rozvržení textu, souborových a síťových vstupů/výstupů, architektury zásuvných modulů, prostředí Dart a kompilačního toolchainu.

Vývojáři obvykle komunikují s Flutter prostřednictvím Flutter frameworku, který poskytuje moderní, reaktivní framework napsaný v jazyce Dart. Zahrnuje bohatou sadu platforem, rozložení základních knihoven, složených z řady vrstev.



Obrázek 2.7. Architektura Flutter [20]

`Foundation` v sobě zahrnuje třídy a služby stavebních bloků, jako je animace, malba a gesta, které nabízejí běžně používané abstrakce. Vykreslovací vrstva poskytuje abstrakci pro řešení rozvržení. Pomocí této vrstvy můžeme vytvořit strom vykreslovatelných objektů. S těmito objekty můžeme manipulovat dynamicky, přičemž strom automaticky aktualizuje rozvržení, aby odráželo změny. Vrstva widgetů je abstrakce kompozice. Každý objekt vykreslení ve vrstvě vykreslování má odpovídající třídu ve vrstvě widgetů. Knihovny Material a Cupertino nabízejí komplexní sady ovládacích prvků, které využívají primitiva kompozice vrstvy widgetů k implementaci návrhových jazyků Material nebo iOS. Flutter vrstva je relativně malá; mnoho funkcí vyšší úrovně, které mohou vývojáři používat, je implementováno jako balíčky, včetně zásuvných modulů pro platformu, jako je kamera a webview [20].

2.3.4 Dart

Všechny aplikace Flutter jsou napsány v programovacím jazyce Dart. Obtížnost naučení Dartu není pro vývojáře nativních platforem náročná, protože je podobný jazykům jako TypeScript, Swift nebo Kotlín.

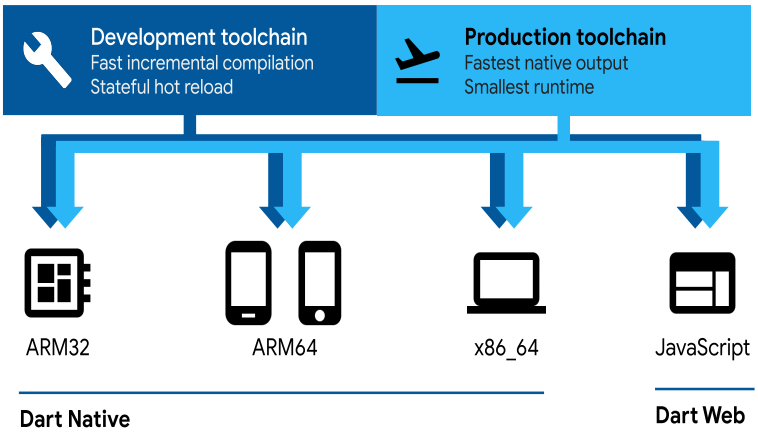
Jazyk Dart je typově bezpečný a používá kontrolu statického typu, aby se zajistilo, že hodnota proměnné vždy odpovídá statickému typu proměnné. Systém psaní Dart je také flexibilní a umožňuje použití dynamického typu kombinovaného s kontrolami za běhu, což může být užitečné při experimentování, nebo pro kód, který musí být obzvláště dynamický. Od verze 2.12 Dart nabízí nulovou bezpečnost, což znamená, že hodnoty nemohou být nulové, pokud explicitně nezadefinujeme, že mohou být. Díky nulové bezpečnosti Dart zachraňuje před nulovými výjimkami za běhu prostřednictvím statické analýzy kódu [19].

Technologie kompilátoru Dart umožňuje spouštět kód různými způsoby na základe cílené platformy:

- Nativni platforma - Pro aplikace zacílené na mobilní zařízení, počítače, servery, Dart zahrnuje jak virtuální stroj Dart s kompilací just-in-time (JIT), tak kompilátor ahead-of-time (AOT) pro vytváření strojového kódu.
- Webova platforma - Webová platforma: Pro aplikace zacílené na web obsahuje Dart jak kompilátor doby vývoje (dartdevc), tak kompilátor doby produkce (dart2js). Oba kompilátory překládají Dart do JavaScriptu [19].

JIT kompilace pomáhá rychle vyvíjet díky své `hot reload` funkcionalitě (inkrementální rekompilací).

Když jsou aplikace připraveny k nasazení do produkce, kompilátor Dart AOT umožňuje kompilaci předem do nativního strojového kódu ARM nebo x64. Aplikace zkompilovaná AOT se spouští s konzistentním krátkým spouštěcím časem [19].



Obrázek 2.8. Kompilace Dart kódu [19]

2.3.4 Asynchronní programovaní

Asynchronní operace umožňují programu provadět další operace když jedna z úloh se nachází v čekajícím stavu. Příklady úloh závislých na I/O jsou:

- Načítání dat ze sítě.
- Zápis do databáze.
- Čtení dat ze souboru [21].

Future

Základní entitou asynchronního programování v Dart je Future - třída která provádí asynchronní operace, chrání jejich stav, a poskytuje metody pro zpracování výsledků.

Future může být v jednom ze dvou stavu - dokončeným a nedokončeným. Když se vola funkce která vrátí Future, funkce plánuje a pouští úlohu a vrátí nedokončenou Future. Když spuštěná přes Future úloha se končí, Future obsahuje bud výslednou hodnotu bud chybu [21].

Příklad spuštění Future a zpracování výsledků a chyb:

```
Future<Data> future = fetchDataFromServer(url);
future.then((data) => handleData(data))
      .catchError((error) => handleError(error));
```

async/await

Výšeuvedený způsob pracování s Future je základní, ale nedoporučuje se k použiti z několika důvodu:

- Vede k tzv. callback-hell - situaci kdy několik Future se kombinuje mezi sebou, vynořují se a stává se složitě pochopit v jakém pořadí se vykonávají.
- Složitě se ladí
- Výrazně snižuje čitelnost kódu.

Příklad callback-hell:

```
Future.all([
 doA().then(aOut =>
   Future(resolve => {
     doC(aOut).then(cOut => {
       resolve([aOut, cOut]);
     });
   })),
 doB().then(bOut =>
   Future(resolve => {
     doD(bOut);
     resolve(bOut);
   }))
]).then(values => {
 doE(values[0][1], values[0][0], values[1]);
});
```

Tento problém poprvé vznikl v jazyce Javascript (tam místo Future se používá pojem Promise, ale podstata je stejná), a byl vyřešen zavedením nové jazykové konstrukce - async/await. Stejná konstrukce byla implementovaná i v Dartu. Klíčová slova async a await poskytují deklarativní způsob, jak definovat asynchronní funkce a používat jejich výsledky [22].

Použitím klíčového slova await lze dočkat se kdy vykonávání Future se dokončí, a přečíst výslednou hodnotu v synchronním stylu. Výjimky a chyby se zpracovávají za pomoci mechanismu try/catch, stejně jako v synchronním kódu. Klíčové slovo await může být použito jenom ve funkci která je označena klíčovým slovem async.

Klíčové slovo async přetvářejí asynchronní funkce na funkce která vrátí Future a dovoluje používat v takové funkce klíčové slovo await.

Stejná funkce z příkladu callback-hell, ale s async/await:

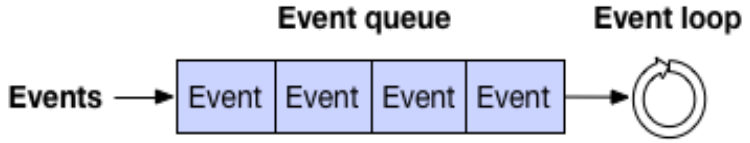
```
void someAsyncFunction() async {
 try {
 final aOut = await doA();
        final bOut = await doB();
```

```
            final cOut = await doC(aOut);
            doD(bOut);
            doE(cOut, aOut, bOut);
        } catch (e) {
        processError(e);
        }
      }
```

2.3.4 Event Loop

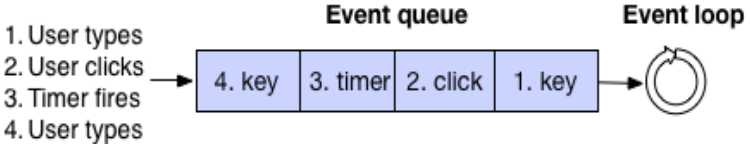
Event Loop (smyčka událostí) je mechanismus který zajišťuje asynchronní vykonávání v Dartu. Všechna rozhraní API a jazykové funkce na vysoké úrovni, které má Dart pro asynchronní programování – Future, Stream, async a await – jsou všechny postaveny na této jednoduché smyčce a kolem ní [23].

Účelem event loop je vzít prvek z fronty události a zpracovat ho, opakující tyto dva kroky až do momentu kdy fronta nemá žádné prvky.



Obrázek 2.9. Fronta události [24]

Prvek ve frontě může reprezentovat uživatelský vstup, notifikace o souborovém I/O, časovače atd.



Obrázek 2.10. Příklad fronty události [24].

Aplikace napsaná v jazyce Dart má jediný event loop který obsahuje dvě fronty: frontu události a frontu mikrouloh. Fronta události spravuje všechny vnější události: I/O, pohyb myší, události kreslení, časovače, zprávy mezi vlákny atd. Fronta mikroúloh je nutná protože kód zpracování úloh občas musí dokončit úlohu později, ale před vrácením kontroly event loop. Například, když pozorovaný objekt se mění, kombinuje několik mutaci dohromady a hlásí je asynchronně. Fronta mikroúloh povoluje pozorovanému objektu hlasit teto mutace před tím jak DOM ukáže nekonzistentní stav [23].

2.3.4 Isolates

Předchozí část vysvětluje jak asynchronní kód pracuje v rámci jednoho vlákna (vlákno má jediný event loop který řídí posloupnost vykonávání kódu ve vlákně). Pro popis vícevláknových aplikaci v Dart je nutně popsat pojem Isolate. Uvnitř aplikace, kód běží v Isolate. Každý Isolate má jediné vlákno (a event loop) a nesděluje žádné změnitelné objekty s jinými isolate. Komunikace mezí Isolate se dělá za pomoci zpráv [25].

Isolate má svojí vlastní paměť a jediné vykonávající vlákno na kterém běží event loop [23].

Chybějící technika sdílené paměti (kterou mají takové jazyky jako Java nebo C++ a která je nejrychlejším způsobem komunikace mezi vlákny) výrazně snižuje strop

efektivity multivláknových aplikace v jazyce Dart, ale na oplátku výrazně zjednodušuje psaní multivláknového kódu.

Co se tyče použití multivláknového kódu v mobilních aplikaci, absolutní většina z nich nepoužívá algoritmy, které by mohli efektivně využít několik jader najednou, a proto koncept izolatu je víc vyhovující než sdílená paměť.

Spuštění nového izolatu se děla za pomoci funkce `Isolate.spawn`, která vrátí Future:

```
Future<Isolate> spawn<T>(
    void entryPoint(T message),
    T message,
    {
        bool paused = false,
        bool errorsAreFatal = true,
        SendPort? onExit,
        SendPort? onError,
        @Since("2.3") String? debugName
    }
)
```

Má jenom dva vyžadovaných parametru - entry point a message. Entry point je funkce nejvyšší úrovní nebo statická metoda která má jeden vstupní parametr. Message se předává jako vstupní parametr entry point, takže musí být stejného typu [26].

2.3.4 Widgets

Celé uživatelské rozhraní ve Flutteru je složeno z widgetů. `Everething is a widget`, jak to popsal zakladatel frameworku. Widget je způsob, jak je deklarováno a konstruováno UI. Ve Flutteru jsou widgety (podobné komponentám v Reactu) reprezentovány neměnnými třídami, které se používají ke konfiguraci stromu objektů. Tyto widgety se používají ke správě samostatného stromu objektů pro rozvržení, který se pak používá ke správě samostatného stromu objektů pro skládání. Flutter je ve svém jádru řada mechanismů pro efektivní procházení upravených částí stromů, přeměnu stromů objektů na stromy objektů nižší úrovně a šíření změn napříč těmito stromy [20].

Widgety jsou neměnné a existují pouze do doby, než je třeba je změnit. Kdykoliv se widgety nebo jejich stav změní, Flutter framework vytvoří nový strom instancí widgetů. Widgety popisují, jak by měl jejich pohled vypadat vzhledem k jejich aktuální konfiguraci a stavu. Všechno, co vidíme na obrazovce, jsou Widgety.

Mnoho widgetů nemá žádný měnitelný stav: nemají žádné vlastnosti, které by se v průběhu času změnily (například ikona nebo štítek). Odsud pochází koncept Stateful a Stateless widgetů.

2.3.4 Stateless Widget

StatelessWidget je velmi jednoduchý widget, který nemá žádný proměnlivý stav, a pokud je potřeba změnit vlastnosti, widget se musí vytvořit znovu.

Velkou výhodou Stateless widgetu je jeho konstantní konstruktor. Díky němu Flutter muže efektivně překreslit pouze ty části, které skutečně potřebuje.

Stateless widget popisuje část uživatelského rozhraní složením dalších widgetů, které popisují uživatelské rozhraní konkrétně. Proces sestavování pokračuje rekurzivně, dokud není popis uživatelského rozhraní zcela konkrétní [27].

```
@override
  Widget build(BuildContext context) {
    return Align( check if has const constructor
      alignment: Alignment.topLeft,
      child: const SizedBox(
        height: 50,
        child: Center(
          child: Text(
              'Ahoj Světe!'
          ),
        ),
      )
    );
 }
```

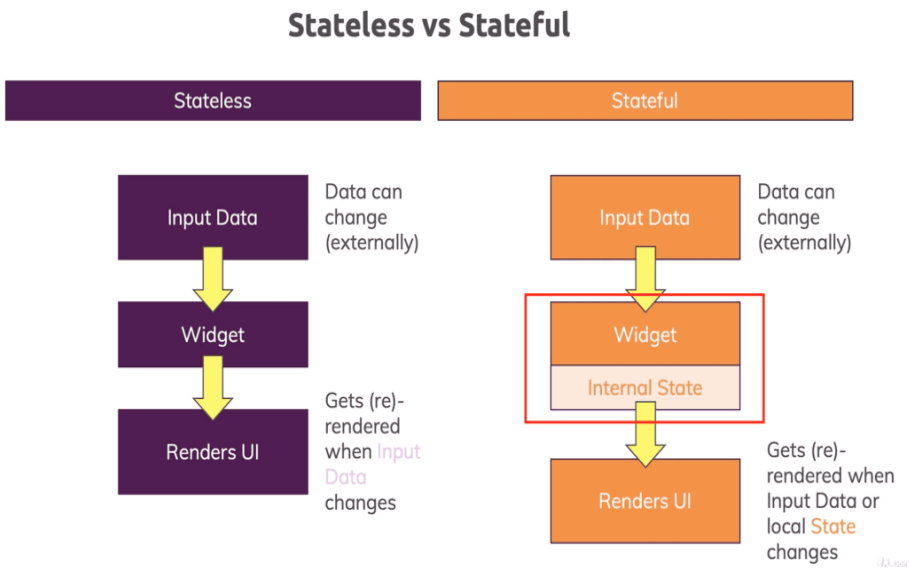
V ukázce kódu každé dítě v našem stromě widgetu přidává více a více detailů (kde bude widget zobrazen, jakou bude mít velikost, nakonec obsah, který bude listem stromu).

StatelessWidget widget je užitečný, když část uživatelského rozhraní, kterou popisujeme, nezávisí na ničem jiném než na konfiguračních informacích v samotném objektu a na BuildContext, ve kterém je widget nafouknutý. U skladeb, které se mohou dynamicky měnit, např. kvůli stavu řízenému vnitřními hodinami nebo v závislosti na stavu systému, je lepší používat StatefulWidget, aby se pokaždé nevolila metoda `build()`, což výrazně hraje na výkonu.

2.3.4 Statefull Widget

StatefulWidget je užitečný při vytváření uživatelského rozhraní založeného na nějakém proměnlivém stavu. V tomto případě se Widget znovu vytvoří pokaždé, když je stav změněn, což odráží změnu stavu v jeho stromu widgetů. StatefulWidget musí kromě vytvoření stromu widgetů, který mají také StatelessWidget, vytvořit objekty `State`, které budou mít proměnlivý stav.

Na obrázku 2.11 je znázorněn rozdíl mezi Stateless a Stateful widgety.



Obrázek 2.11. Stateful oproti Stateless [28]

Stav je definován dvěma věcmi:

- Data používaná widgetem se mohou změnit.
- Všechny stavy musí být stanoveny v době volání metody `build()`.

2.3.4 Návrhové vzory

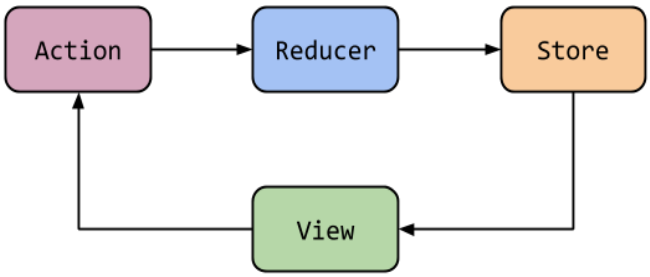
Flutter je deklarativní. To znamená, že Flutter vytváří své uživatelské rozhraní tak, aby odráželo aktuální stav aplikace. Namísto toho, aby při vykreslování uživatelského rozhraní bylo třeba modifikovat jeho jednotlivé widgety jako při imperativním přístupu, se tyto widgety vytvoří znovu. Widgety jsou tedy neměnlivé, a proto je třeba definovat stav, který widget potřebuje k tomu, aby jej bylo možné vytvořit.

Existuje mnoho přístupů a řešení, jak jej řešit. Nejjednodušší způsob, jak si ukládat stav, je lokální ve widgetu a měnit jej pomocí funkce `setState()`. Volání setState upozorní framework, že se vnitřní stav objektu změnil způsobem, který by mohl ovlivnit uživatelské rozhraní. Toto řešení je vhodné pro ukládání dat, která se týkají přímo designového rozhraní widgetů, ale nikoliv pro ukládání dat byznys logiky [29].

Nejčastěji vývojáři z Google uvádí ve svých příkladech a na konferencích BLoC (Business Logic Component). Avšak existuje mnoho dalších přístupů k řízení stavu: Redux, GetX, Riverpod a mnoho dalších.

2.3.4 Redux

Redux je architektura s jednosměrným tokem dat, která usnadňuje vývoj aplikací, které se snadno testují a udržují. Tato architektura je dobře známá webovým vývojářům. To je vlastně defaultní architektura pro React aplikace.



Obrázek 2.12. Redux [30]

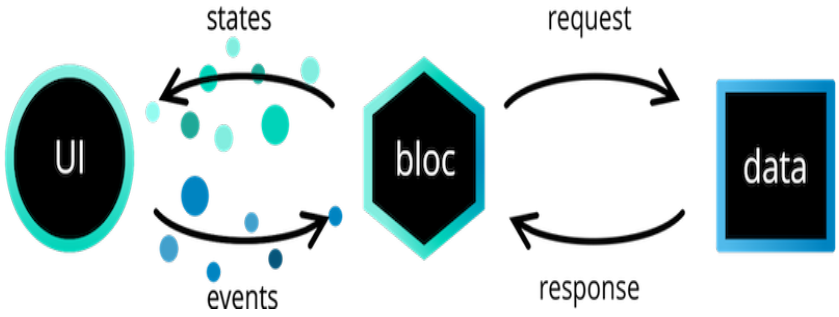
`Store` představuje stav celé aplikace. Je to jediný zdroj pravdy, který je globálně dostupný a který uchovává objekt State, který představuje stav celé aplikace. Každá událost aplikace (buď od uživatele nebo externí) je reprezentována jako `Action`, která je odeslána do funkce `Reducer`, který aktualizuje `Store` o nový stav v závislosti na tom, jakou akci obdrží. Kdykoli je prostřednictvím obchodu protlačen nový stav, zobrazení se znovu vytvoří, aby odráželo změny.

S Reduxem je většina komponent oddělena, takže změny uživatelského rozhraní jsou velmi snadné. Kromě toho jediná obchodní logika spočívá ve funkcích Reducer. Reducer je funkce, která provádí akci a vrací nový objekt State. Proto je snadné ji otestovat, protože můžeme napsat test jednotky, který nastaví počáteční stav a zkontroluje, že `Reducer` vrací nový a upravený stav [30].

2.3.4 Business Logic Component (BLoC)

BLoC je často přirovnáván ke vzoru MVVM (Model-View-ViewModel), se kterým se často setkávají Android a iOS vývojáři s tím rozdílem, že ViewModel z MVVM je nahrazen BLoC. BLoC je zkratka pro Business Logic Component.

Myšlenka je, že blok přijímá události, které spouštějí změny stavu, které jsou pak vysílány z bloku. Proto Bloc oddělil vstupy a výstupy. Vstupem poskytujete události a z výstupů přijímáte stav. Výstupy lze sledovat pomocí widgetů uživatelského rozhraní, které reagují na změny stavu [31].



Obrázek 2.13. BLoC komunikace [31]

BLoC má zároveň svou implementaci, která je jedna z nejpoužívanějších knihoven pro Flutter.

Funguje jako prostředník mezi uživatelským rozhraním a vrstvou pro přístup k datům, která je poměrně často reprezentována repozitářem. To znamená, že události spouštějí určitou byznys logiku uvnitř bloku, která může například požádat úložiště, aby provedlo požadavek API. Jakmile je odpověď přijata, může blok zpracovat data a vydat odpovídající stav. Stav může být libovolný objekt. Ve většině případů se jedná o samostatnou třídu, která má uložena všechna data pro konkrétní část aplikace.

2.3.5 REST API

API je mechanismus, který umožňuje dvěma softwarovým komponentům vzájemně komunikovat pomocí sady definic a protokolů.

REST je zkratka pro Representational State Transfer. REST definuje sadu funkcí jako GET, PUT, DELETE atd., které mohou klienti používat pro přístup k datům serveru. Klienti a servery si vyměňují data pomocí HTTP protokolu [32].

Hlavním rysem REST API je bezstavovost. Bezstavovost znamená, že servery neukládají klientská data mezi požadavky. Odezvou serveru jsou prostá data ve standardizovaném formátu (JSON, XML).

REST implementuje čtyři základní metody, známé pod označením CRUD, tedy Create, Retrieve, Update a Delete. Tyto metody jsou implementovány pomocí odpovídajících metod HTTP protokolu [33].

GET nejčastěji slouží pro získání údaje. Pro identifikace musí každý zdroj mít URI (jednotný identifikátor zdroje). Odesláním požadavku GET na URI dostaneme požadovanou informaci.

POST se používá k odesílání dat na server pro vytvoření/aktualizaci zdroje. V momentě odeslání požadavku není známa URI, následně ji dostaneme v odpovědi od serveru, ale pouze v tom případě, zda operace proběhne úspěšné. Data odeslaná na server pomocí POST jsou uložena v těle požadavku.

PUT se používá k odesílání dat na server k vytvoření/aktualizaci zdroje.

Rozdíl mezi POST a PUT je ten, že požadavky PUT jsou idempotentní. To znamená, že volání stejného požadavku PUT vícekrát bude vždy generovat stejný výsledek. Naproti tomu opakované volání požadavku POST má vedlejší efekty vytváření stejného zdroje vícekrát.

DELETE slouží pro vymazání konkrétního zdroje na zadané URI [34].

Kapitola 3
Návrh

V této kapitole budou popsány funkční a nefunkční požadavky a funkcionalita aplikace. Dále je popsán datový model, Digital Parenting API a použité knihovny.

3.1 Funkční požadavky

Ve spolupráci s vedoucím práce byly sepsány následující požadavky:

- Aplikace musí uživateli popsat, k čemu slouží.
- Aplikace musí komunikovat se serverem.
- Aplikace musí synchronizovat data na více zařízeních.
- Aplikace podporuje více jazyků.
- Uživatel může přidat více dětí.
- Uživatel může zadat a měnit omezení a pravidla pro každé dítě.
- Uživatel muže provádět kontrolu dodrženi.
- Uživatel může vymazat svůj profil a všechna data.

3.2 Nefunkční požadavky

- Aplikace musí být napsaná v nástroji Flutter, který podporuje multiplatformní tvorbu.
- Aplikace musí komunikovat se serverem prostřednictvím REST API.
- Aplikace musí fungovat bez připojení k internetu.

3.3 Funkcionalita aplikace

3.3.1 Sezení

Každé sezení je zaměřeno na nějaké téma. Sezení slouží především pro to, aby se rodič mohl dozvědět užitečné informace, navázat rozhovor s dítětem a domluvit se na pravidlech a limitech, které by mělo dítě dodržovat.

Sezení obsahují interakční prvky. Například druhé sezení obsahuje užitečné informace ohledně stráveného času u obrazovky, dále umožňuje nastavení časových limitů.

3.3.2 Pravidla

Aplikace musí povolit uživateli nastavit vlastní pravidla, nebo vybrat ze seznamu nabízených. Uživatel je schopen nastavit určitá pravidla po dokončení příslušného sezení. To zaručuje, že uživatel bude vědět, k čemu to bude sloužit a proběhne takzvaný 'onboarding'.

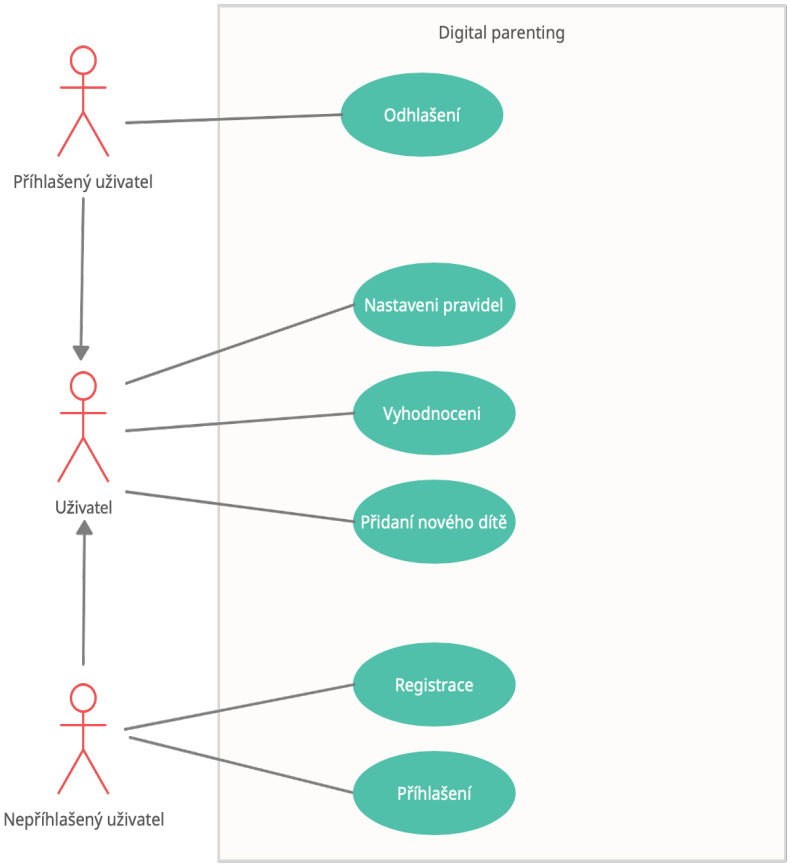
3.3.3 Vyhodnoceni

Po nastavení pravidel a omezení bude uživatel schopen vyhodnocovat, jestli dítě dodržuje pravidla. Tato data se musí odeslat na server, kde budou v budoucnu sloužit k její analýze.

3.4 Diagramy

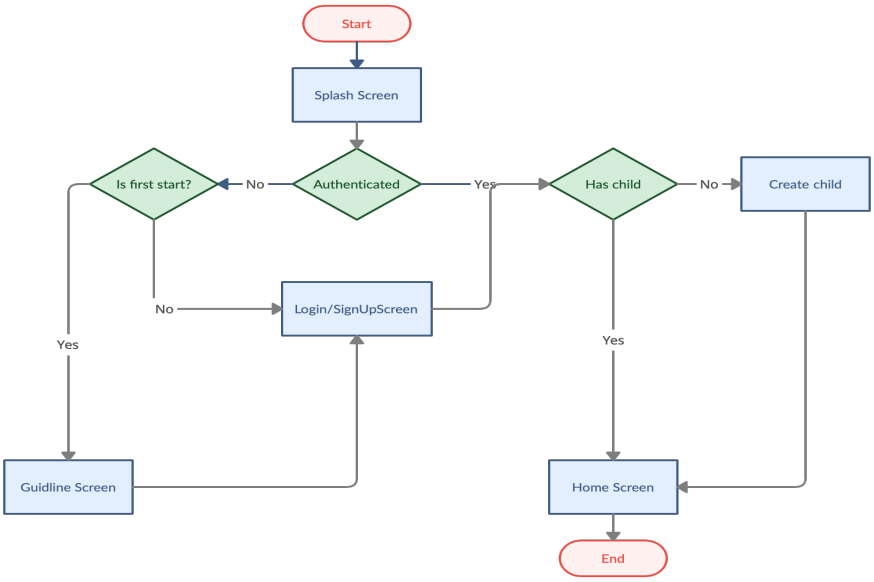
`Use Case diagram`(Diagram případů užití) je diagram, který zachycuje vnější pohled na modelovaný systém. Jde o posloupnost souvisejících transakcí mezi aktérem (uživatelem v určité roli) a systémem během vzájemného dialogu.

Aktér v našem případě může být ve 2 stavech: přihlášený a nepřihlášený.



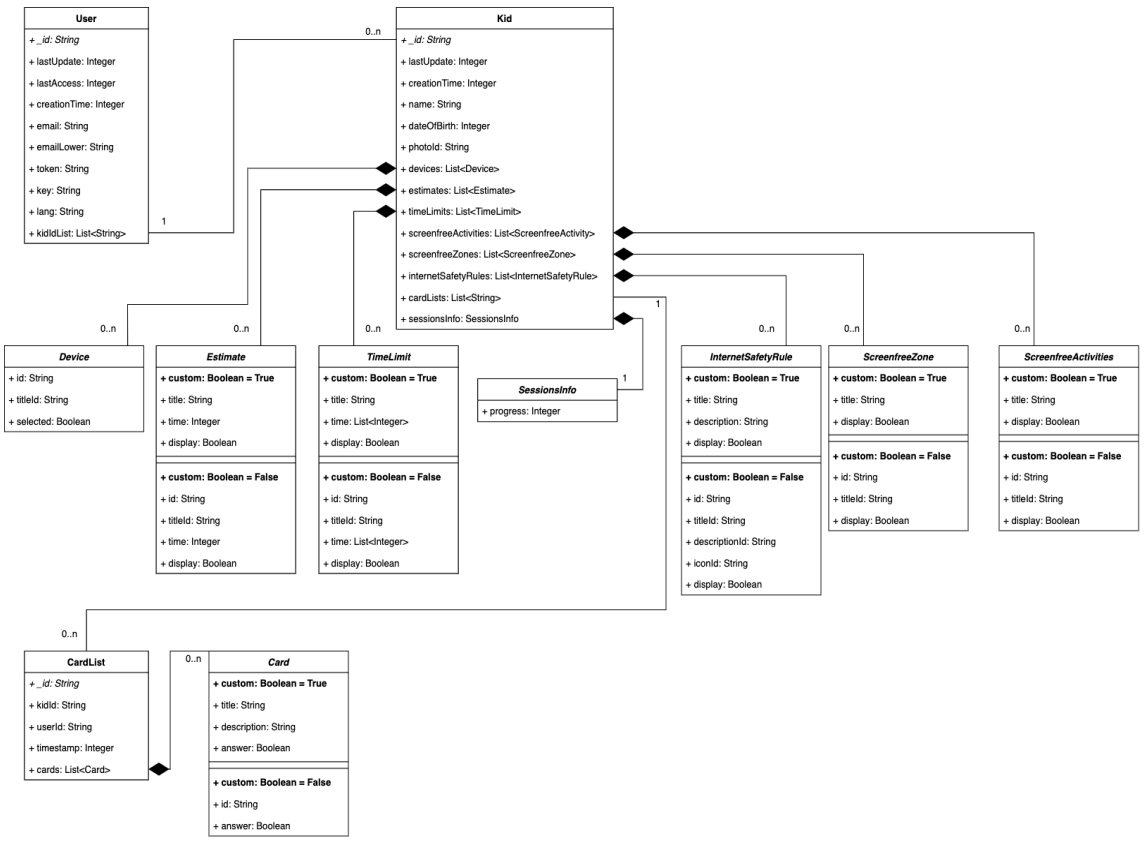
Obrázek 3.1. Diagram případů užití

Na obrázku 3.2 je zobrazen postup jak se uživatel dostane do hlavni obrazovky. V případe prvního spuštěni aplikaci uživateli se zobrazí obrazovky s popisem k čemu aplikace slouží. Potom se uživatel dostane do přihlášeni či registrace a po úspěšném procházeni musí přidat dítě. Kdyby z nějakého z těchto kroku aplikaci zavřel, po opětovném spuštěni se jemu zobrazí obrazovka na které skončil. Až uživatel vytvoří dítě, aplikace se přepne na hlavni obrazovku.



Obrázek 3.2. Diagram aktivit

3.5 Datový model



Obrázek 3.3. Datový model.

V datovém modelu aplikace 2 entity reprezentují lidi, a ostatní se vztahují k dítěte.

- id – unikátní identifikátor dítě.
- Estimation - kolik času bude dite travit u konkretni aktivity.
- ScreenFreeZone – zóna, kde dítě nesmí používat zařízeni.
- ScreenfreeActivity - Aktivita, při které dítě nesmí používat zadně elektronické zařízeni.
- TimeLimit – maximální čas, který by dítě mělo trávit u aktivity. Zadává se pro každý den týdne.
- SessionInfo - v jaké fázi sezení se uživatel nachází.
- InternetSafetyRule – pravidlo, které musí dítě dodržovat.
- lastUpdate – čas poslední změny jakékoliv informace o dítě. Ten údaj nám bude sloužit pro správnou synchronizace se serverem.

3.6 Digital Parenting API

Serverovou část pro aplikace Digital Parenting implementoval Bc. `Jakub Lhoták` jako diplomovou práci. Dále budou popsány zásadní části, které budou použity v této diplomové práci.

Server odpovídá na požadavky ve JSON formátu. Úspěšně zpracovaný požadavek vrací:

```
{
    status: 0,
    target: další vnořeny objekt
}
```

`target` je JSON objekt, který je definovány pro konkretní požadavek.
V případe chyby se zpracováním server vrací:

```
{
    status: int,
    message: string
}
```

`status` je vetší 0 v případe chyby. `message` je popis chyby
Za prvé se potřebujeme zaregistrovat do aplikace.
POST http://digital-parenting.herokuapp.com/login-service/register
Body:

```
{
    "email": "email@gmail.com",
    "password": "qwerty"
}
```

Server na tento požadavek vrátí:

```
{
    "status": 0,
    "target": {
        "token": "R2AFMOlxm2UBynZPziRsJTLgoOt4rTMj"
    }
}
```

Dále budeme potřebovat přidat dítě, kterému zadáme jeho jméno, a datum narozeni.

Hodnotu atributu `token` budeme dále používat aby server věděl s kým probíhá komunikace.

POST http://digital-parenting.herokuapp.com/api/kid/create

Body:

```
{
    "kid": {
        "name": "Test Kid",
        "dateOfBirth": "1425767851000"
    }
}
```

A vrátí následující datovou strukturu:

```
{
    "status": 0,
    "target": {
        "kid": {
            "_id": "626af5527360498cb26ec7d3",
            "cardLists": [],
            "creationTime": 1651176786089,
            "dateOfBirth": 1425767851000,
            "devices": [
                {
                  "id": "4D4B71FE-7234-424E-A996-66DA5B0F36C8",
                  "selected": false,
                  "titleId": "Devices.smartPhone"
                },
                ...
            ],
            "estimates": [
                 {
                  "custom": false,
                  "display": false,
                  "id": "179BD34E-042E-446F-BAE2-BC85F070BB63",
                  "time": -1,
                  "titleId": "Activities.Fun.watchingVideos"
                },
                ...
            ],
            "internetSafetyRules": [
              {
              "custom": false,
              "descriptionId": "InternetSafetyRules.content.desc",
              "display": false,
              "iconId": "lightbulb.fill",
              "id": "0EDC5CDA-FA80-4A74-9537-8776FEE692F1",
              "titleId": "InternetSafetyRules.content.title"
              },
              ...
            ],
            "lastUpdate": 1651176786089,
            "name": "Test Kid",
            "photoId": null,
            "screenfreeActivities": [
```

```
                {
                    "custom": false,
                    "display": false,
                    "id": "3CFAE54A-B519-41C5-A432-FC523CAF7269",
                    "titleId": "Activities.WithoutScreens.eating"
                },
                ...
            ],
            "screenfreeZones": [
                {
                    "custom": false,
                    "display": false,
                    "id": "5445A769-1F19-4CCE-B3D7-A3814B849A3F",
                    "titleId": "Zones.car"
                },
                ...
            ],
            "sessionsInfo": {
                "progress": 0
            },
            "timeLimits": [
                {
                    "custom": false,
                    "display": false,
                    "id": "179BD34E-042E-446F-BAE2-BC85F070BB63",
                    "time": [
                        -1,
                        -1,
                        -1,
                        -1,
                        -1,
                        -1,
                        -1
                    ],
                    "titleId": "Activities.Fun.watchingVideos"
                },
                ...
            ]
        }
    }
}
```


Atribut `display` po přidání dítě je pro všechny aktivity, zóny a bezpečnostní pravidla false. Poté se mění na základě vstupu od uživatele.

Další API metody používané v aplikaci:

- POST http://digital-parenting.herokuapp.com/login-service/login
- POST http://digital-parenting.herokuapp.com/login-service/update
- POST http://digital-parenting.herokuapp.com/login-service/new-token
- POST http://digital-parenting.herokuapp.com/login-service/delete
- GET http://digital-parenting.herokuapp.com/api/user/get
- GET http://digital-parenting.herokuapp.com/api/user/get-full
- GET http://digital-parenting.herokuapp.com/api/kid/get?kidId=id
- POST http://digital-parenting.herokuapp.com/api/kid/update?kidId=id

- DELETE http://digital-parenting.herokuapp.com//api/kid/delete?kidId=id

3.7 Použité knihovny

Dále, popíšu hlavni knihovny, které byli použité v aplikaci.

3.7.1 flutter bloc

flutter_bloc je knihovna která implementuje koncept BLoC pro Flutter.

Je to stavené na 3 entitách: Event, Bloc a State.

`Event` je nejaka akce vyvolana uzivatelem z uzivatelskeho rozhrani(stisknuti tlacitka, vstup dat, atd). Obsahuje informace o akci a je předávaná do bloku ke zpracovaní.

Bloc přijme Event a muze na nej reagovat emitaci nového State.

`BlocProvider` je widget, který poskytuje bloc nebo cubit svým dětem v rámci podstromu.

```
BlocProvider(
  create: (BuildContext context) => BlocA(),
  child: ChildA(),
);
```

Pokud je potřeba poskytnout vice bloků existuje další widget, který vezme list bloků, a poskytne je potomku. Je to `MultiBlocProvider`.

```
MultiBlocProvider(
    providers: [
      BlocProvider(create: (context) => Bloc1()),
      BlocProvider(create: (context) => Bloc2())
    ],
    child: MyApp()
  );
```

`BlocBuilder` je widget, který odpovídá za aktualizace UI na základě nového stavu. Pokaždé, když se stav změní, překreslí se widget, který je zabalen v BlocBuilder. Pokud chceme explicitně zadefinovat, že nechceme překreslovat widget, existuje vlastnost `buildWhen`, která poskytuje aktuální a předchozí stav, na základě kterých můžeme udělat porovnání a říct, jestli chceme aktualizovat.

```
BlocBuilder<BlocA, BlocAState>(
  buildWhen: (previousState, state) => previousState != state
  builder: (context, state) {
    // vrátit widget na základě BlocA stavu.
  }
)
```

`BlocListener` je podobný jako BlocBuilder s tím rozdílem, že neaktualizuje UI. Místo toho lze provést operace na základě změny stavu, například přejít na další obrazovku, ukázat chybu, nebo otevřít Dialog. Namísto buildWhen vlastnosti obsahuje `listenWhen` se stejným principem.

Vlastnost `listener` se volá jednou na změnu stavu.

```
BlocListener<BlocA, BlocAState>(
  listenWhen: (previousState, state) {
    //  vratit true/false jestli zpracovavat listener
```

```
  },
  listener: (context, state) {
    // provést akci na základě BlocA stavu
  },
  child: Container(),
)
```

Pokud je potřeba použit funkcionalitu obou výše uvedených widgetů existuje `BlocBuilder` [35].

```
BlocConsumer<BlocA, BlocAState>(
  listener: (context, state) {
    // provést akci na základě BlocA stavu
  },
  builder: (context, state) {
    // vrátit widget na základě BlocA stavu.
  }
)
```

3.7.2 get it

Další důležitá věc pro vytvoření čistého a lehce testovatelného kódu je použití nějakého druhu inverze kontroly (IOC). Get_it knihovna je lokátor služeb, který obsahuje centrální registr, kam se registrují třídy, a poté je možné získat instance této třídy. Nezatěžuje strom uživatelského rozhraní speciálními widgety pro přístup k datům, jako to dělá Redux [36].

3.7.3 json serializable

Většina mobilních aplikací komunikuje se serverem a přijímá data ve formátu JSON. Existují dva způsoby, jak JSON serializovat: manuální a s použitím generací kódu. Serializace je proces, který převádí datovou strukturu do stringu. Deserializace je opačný proces, který převádí string do datové struktury.

Manuální serializace je vhodná pro prototypy nebo malé projekty, kdy je potřeba se rychleji dostat k funkčnímu řešení. To je dosazeno vestavěným JSON dekodérem `dart:convert`. JSON ve string formátu se předává do jsonDecode() funkce, která vrací `Map<String, dynamic>`, ve kterém můžeme hledat hodnoty podle klíče. Uvedený přístup je nevhodný pro větší projekty, protože je velká možnost udělat překlep ve zpracování a aplikace kvůli tomu může padnout za běhu.

Druhá možnost používá určitou knihovnu pro generaci kódu. Jednou z těchto knihoven je json_serializable. Stačí označit datový model anotaci @JsonSerializable(), zadefinovat soubor, kam se model vygeneruje.

```
part 'example.g.dart';
```

A nakonec definovat fabriční konstruktor, který vytvoří instance třídy z `Map<String, dynamic>`.

```
  factory Example.fromJson(Map<String, dynamic> json) =>
    _$ExampleFromJson(json);
  Map<String, dynamic> toJson() => _$ExampleToJson(this);
```

Pouštěním následujícího přikážu se nám vytvoří soubor `example.g.dart` s implementaci `fromJson` a `toJson`.

```
    flutter pub run build_runner build
```

3.7.4 Hive

Aby aplikace mohla fungovat bez připojení k internetu, je potřeba někam ukládat data lokálně. Data se mohou ukládat mnoha různými způsoby: NoSQL, SQLite, SharedPreferences a další. Aplikace v rámci diplomové práce využívá lokální NoSQL databázi s názvem Hive jako primární úložiště.

Obrázek 3.4. Hive oproti jiným úložištím [38].

Hive je extrémně rychlá NoSQL offline databáze. Je napsána v Dartu, takže nevyžaduje žádné implementace specifické pro zařízení. Mezi nejlepší funkce Hive patří rychlé operace, žádné nativní závislosti, jednoduché API a automatická migrace. Ukládá data bezpečně pomocí šifrování AES-256 (robustní šifrovací standard). Hive umožňuje ukládat data nejenom v podobě primitivních typů, ale i v podobě objektů HiveObject, což také umožňuje určité vztahy mezi objekty.

Data se ukládá do takzvaných `krabic`. Je to podoba pojmu `bucket` ve většině NoSQL databázích. Tyto krabice jsou uložené v mezipaměti, a proto jsou dostatečně rychlé oproti jiným řešením. Z toho důvodu, že je NoSQL flexibilní, můžeme pro každou krabici definovat jakoukoli strukturu. V důsledku toho přítomnost krabice neznamená existenci konzistentního datového modelu. To je významnou výhodou, protože naše aplikace vyžaduje flexibilní datové sloupce pro stejný typ entity [38].

3.7.5 easy localization

Tato knihovna poskytuje rychlé a snadné řešení pro lokalizace aplikace. Dělá více než jen podporu jazykových překladů. Podporuje také pohlaví, směr textu a další [39].

Pro využití této knihovny je potřeba udělat několik kroků:

- Vytvořit adresář, do kterého přidáme soubory s textem v požadovaném jazyce, například `en.json` a `cz.json`.
- Deklarovat adresář textu v souboru pubspec.yaml.
- Vygenerovat klíč pro každé slovo spuštěním následujícího kódu:

```
flutter pub run easy_localization:generate -S
    assets/translations -f keys -O lib/generated
        -o locale_keys.g.dart
```

Vygeneruje se tím soubor s klíčem, pomocí kterých se provolá text.

- Obalit aplikace widgetem `EasyLocalization` a poskytnout mu cestu do souborů s překladem, vygenerovaný soubor klíčů, podporované jazyky, a defaultní jazyk.
- Přístup k textům máme pomocí contextu: `LocaleKeys.Test.tr()`. Třída LocaleKeys je převzata z vygenerovaného souboru klíčů. Tečkový zápis se používá pro přístup k jeho vlastnostem. Funkce tr() se používá k vyzvednuti textu z generovaného souboru.

  Existují dva přístupy k výběru jazyka aplikace:
  Ruční výběr jazyka v aplikaci:

```
context.locale = Locale('en', 'US');
```

Použití systémového jazyka:

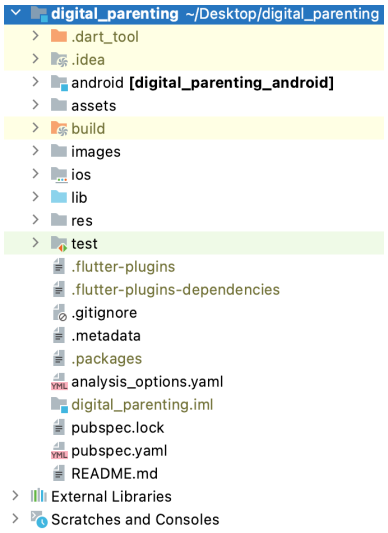
```
context.locale = context.deviceLocale
```

V aplikace ze začátku je zapnutý česky jazyk a jde ho změnit v profilu.

Kapitola 4
Implementace

V této kapitole jsou popsána struktura aplikace a zajímavé implementační části.

4.1 Základní prvky aplikací



Obrázek 4.1. Struktura Flutter projektu

Na obrázku 4.1 je zobrazena struktura každého Flutter projektu.

Každý projekt Flutter obsahuje soubor pubspec.yaml. Základní pubspec se vygeneruje při vytvoření nového projektu. Nachází se v kořenu projektu a obsahuje metadata o projektu, která nástroj Dart a Flutter potřebuje znát. Pubspec je napsán v YAML jazyce, který je čitelný pro lidi.

Je to hlavní soubor projektu, ve kterém se definují autor aplikace, verze, assety, fonty a také závislosti na externí Flutter moduly a Dart knihovny.

```
version: 1.0.2+2
```

Verze aplikace, která se propaguje do android a iOS.

```
environment:
sdk: ">=2.12.0 <3.0.0"
```

Flutter SDK má být v tomto rozmezí a všechny knihovny, které jsou definované v `pubspec.yaml`, musí používat verze Flutter SDK v rovněž definovaném intervalu. Při sestavení aplikace se Flutter snaží vybrat vyhovující verze knihoven, a pokud se to nepodaří, sestavení selže s výstupem, v jaké knihovně je problém.

```
dependencies:
  flutter:
```

```
    sdk: flutter
  flutter bloc: 7.0.0
  hive: 2.0.4
  ...
```

```
flutter:
  uses-material-design: true
  assets:
    - assets/translations/
    - images/
```

Tady definujeme kde máme uložené assety.

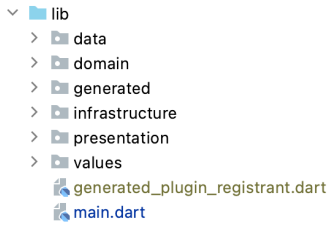
Po sestavení aplikace se vygeneruje soubor pubspec.lock, který uvádí konkrétní verze každé závislosti, které balíček používá.

Složky `ios` a `android` obsahují nativní projekty pro tyto platformy. Při sestavení aplikace pro platformu Android se flutter kód převede na nativní. Výsledný nativní kód se vloží do Android složky a nakonec se aplikace pro Android sestaví. Stejný princip se používá pro ios. Pokud by se chtělo aplikace spouštět pro platformy jako Linix, Windows nebo web, bylo by potřeba to zadefinovat v projektu, aby se vytvořily další složky. Tyto složky jsou využívané Flutter SDK a vývojář by v nich nic dopisovat neměl.

Do složky `build` se generují spustitelné soubory pro platformy.

4.2 Struktura Digital Parenting

Struktura projektu je rozdělena do 6 částí.



Obrázek 4.2. Architektura Digital Parenting

Složka `infrastructure` obsahuje abstraktní třídy pro přístup k databázi a k jejich implementaci. Aby bylo snadné vyměnit současnou databázi za jinou, je vhodná praktika používat rozhraní. V tom případě přibyde další úroveň abstrakce. Komunikace s API a databáze prochází přes rozhraní `IBaseApi`.

Ve složce `data` se nachází datové modely.

Složka `domain` obsahuje celou byznys logiku aplikace. Každá podsložka odpovídá za logiku buď jedné obrazovky, nebo za její části.

`Presentation` má na starosti věci týkající se uživatelského rozhraní, přičemž na této úrovni je zakázaná veškerá komunikace s API či s datovým úložištěm.

`Generated` obhajuje generované třídy sloužící pro lokalizaci aplikace.

`Values` obsahuje třídy konstant. Jsou zde barvy, fonty, styly, které se budou používat v aplikaci.

`main.dart` je výchozí bod aplikace.

V souboru `main.dart` se aplikace spouští. Nejprve se musí zaregistrovat adaptéry pro zpracování modelu naší lokální databáze:

```
Hive.registerAdapter<KidJson>(KidJsonAdapter());
Hive.registerAdapter<User>(UserAdapter());
```

Potom se pomocí injekce závislosti zaregistruje repozitář a přístup k databázi:

```
GetIt.I.registerSingleton<IUserRepository>(HiveUserRepository());
GetIt.I.registerSingleton<IBaseApi>(
    DioClient(GetIt.I<IUserRepository>())
);
```

Teď `HiveUserRepository` a `DioClient` jsou zaregistrované do GetIt a můžeme získat instance těchto tříd kdekoliv pomocí `GetIt.I<třída>()`.

```
class SplashCubit extends Cubit<SplashState> {
  final IUserRepository _userRepository;
  final IBaseApi _api;

  SplashCubit(this._userRepository, this._api)
```

Například, `SplashCubit` používá databáze a přistup k API. Pomoci BlocProvider, popsanému v návrhu vytvoříme Cubit, kterému zpřístupníme instance třídy `HiveUserRepository` a `DioClient` a potomka, který bude mít přístup k instanci.

```
return BlocListener<SplashCubit, SplashState>(
    listener: (context, state) {
      if(state.userStep == UserStep.createChild) {
        Navigator.pushNamed(context, '/create-child');
      } else if(state.userStep == UserStep.authenticated) {
        Navigator.pushNamed(context, '/home');
      } else if(state.userStep == UserStep.firstStart) {
        Navigator.pushNamed(context, '/guidance');
      } else if(state.userStep == UserStep.afterGuidance) {
        Navigator.pushNamed(context, '/auth',  arguments: AuthArgs(
            AuthStatus.login
        ));
      }
    },
);
```

Zavedeni lokalizace

```
EasyLocalization(
  supportedLocales: [Locale('en', 'US'), Locale('en', 'CZ')],
  startLocale: Locale('en', 'CZ'),
  path: 'assets/translations',
  assetLoader: CodegenLoader(),
  fallbackLocale: Locale('en', 'CZ'),
  child: MyApp()
)
```

```
@override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Digital Parenting',
      theme: ThemeData(
          primarySwatch: Colors.blue,
          scaffoldBackgroundColor: const Color(0xFFF7FAFD),
```

```
            brightness: Brightness.light
        ),
        localizationsDelegates: context.localizationDelegates,
        supportedLocales: context.supportedLocales,
        locale: context.locale,
        initialRoute: '/',
        routes: {
         '/': (context) {
         return BlocProvider(create: (context) =>
          SplashCubit(GetIt.I<IUserRepository>(), GetIt.I<IBaseApi>())
            child: SplashScreen(),);
          },
          ....
```

MaterialApp je rozšířením widgetu nejvyšší úrovně `WidgetsApp`. WidgetsApp je widget, který abstrahuje řadu funkcí požadovaných pro většinu mobilních aplikací, jako je například nastavení navigátoru a používání tématu pro celou aplikaci. MaterialApp přidává funkce a možnosti stylu specifické pro Material Design, který je nativní pro Android aplikace. Například animace přechodu mezi stránkami jsou navrženy tak, jak je na zařízení Android [40]. Poskytuje snadnou navigaci mezi stránky.

4.3 Navigace

Všechny přechody mezi stránky jsou zadefinované v MaterialApp.

```
'/home': (context) {
  return  MultiBlocProvider( providers: [
    BlocProvider(
      create: (context) => ProfileCubit(
      GetIt.I<IBaseApi>(), GetIt.I<IUserRepository>()
      ),
    ),
    BlocProvider(
      create: (context) => SessionsCubit(GetIt.I<IUserRepository>()),
    ),
    BlocProvider(
     create: (context) => CardCubit(
     GetIt.I<IUserRepository>(), GetIt.I<IBaseApi>()
     ),
    ),
  ], child: MainScreen(),);
}
```

Funguje to jako mapa, kde je jako klíč používán identifikátor stránky v textové podobě, a hodnotou je funkce, která vrací stránku. `MultiBlocProvider` jednoduše sloučí více BlocProvideru do jednoho stromu widgetů.

Navigátor je zásobník, který funguje na principu LIFO (Last in, first out). Jednoduše je možné přidat stránku do Navigátoru pomocí metody.

```
Navigator.pushNamed(context, '/home')
```

Pokud bychom potřebovali speciální logiku jako například přidat obrazovku na místo aktuální, lze použít.

```
Navigator.of(context).pushReplacementNamed('/home')
```

nebo

```
Navigator.popAndPushNamed(context, '/home')
```

Nejzajímavější metoda pro přidávaní nakonec je

```
Navigator.of(context).pushNamedAndRemoveUntil('/auth',
    (Route<dynamic> route) => false)
```

tato metoda vymaže ze zásobníku všechno a do něho přidá stránku. Je to užitečné při odhlašovaní, protože nechceme mít možnost se vrátit zpět.

V některých případech je taky potřeba předat informace z jedné obrazovky do druhé. V tom případe se musí zadefinovat datová třída, v jakem formátu je data předávaná.

```
class AuthArgs {
  final AuthStatus? status;

  AuthArgs(this.status,);
}
```

Tuto datovou třídu pak je potřeba předat v argumentech.

```
Navigator.of(context).pushNamedAndRemoveUntil('/auth',
    (route) => false, arguments: AuthArgs(AuthStatus.login)
)
```

Předávanou informace lze vyzvednout v miste kde je definovaná cesta pomoci `ModalRoute`.

```
final args = ModalRoute.of(context)!.settings.arguments as AuthArgs;
```

Musí také existovat možnost, jak stránku zavřít a jak se vrátit zpět na předchozí. Nejpoužívanější metoda je `Navigator.pop()`, která odstraní aktuální stránku ze zásobníku. Pokud zásobník nebude prázdný, aplikace se vrátí na předchozí stránku. V opačném případě by se aplikace zavřela. Pokud nechceme, aby se aplikace v tomto případě zavírala, lze použít metodu `maybePop()`, která se dovolí vrátit zpět pouze v případě, že je kam se vracet [41].

4.4 Synchronizace

Z funkčních požadavků vychází, že jeden účet mohou spravovat oba rodiče na více zařízeních. S tím přichází problém synchronizace dat. Například matka prošla několik sezení a vytvořila několik pravidel pro dítě. Tatínek, když se spustí aplikace, musí získat zaktualizovaná data. Na serveru je pro každé dítě uložen atribut `lastUpate`. To je čas poslední úpravy dítěte, který je představen v milisekundách od počátku epochy. Aplikace to řeší tak, že vezme čas poslední úpravy každého dítěte a ho porovná s časem uloženým v lokální databázi. Pokud je na serveru čas větší, do databáze se nahrají data ze serveru.

```
_api.getFullUser().then((userFromApi) {
  _userRepository.getUser().then((userFromDB) {
    for (KidJson kidFromDB in userFromDB.kidList) {
      var kidFromApi = userFromApi?.kidList.firstWhere((element) =>
      element.id == kidFromDB.id);
```

```
      if (kidFromApi!.lastUpdate > kidFromDB.lastUpdate) {
        _userRepository.updateKid(kidFromApi.id, kidFromApi);
      }
    }
  });
});
```

Další zajímavou částí jsou sezení. Fungují jako konstruktor a můžeme lehce vyměňovat stránky, přidávat nové, odebírat je či měnit obsah.

Každé sezení je seznam stránek, kde jednotlivou stránku lze rozdělit na:

- Informační - je to stránka, která obsahuje jenom informace pro rodiče.
- S možností odklikávání nějaké události (výběr pravidla, zóny nebo aktivity bez obrazovky).
- S možností nastavení času – je to stránka, která má obsah pro rodiče a nastavení času.

Všechny tyto stránky se nachází na prezentační úrovni. V Cubitu, který odpovídá za sezení, přidáme do seznamu stránky, které chceme na konkrétním sezení vidět.

```
List<Widget> sessionsWidgets = [];

sessionsWidgets.add(
    SessionInfoScreen(
      title: LocaleKeys.Sessions_Day1_title.tr(),
      description: LocaleKeys.Sessions_Day1.tr(),
      imageUrl: 'images/sessions/session_1/3.svg',
    )
);

sessionsWidgets.add(
    SelectScreen(
      title: LocaleKeys.Sessions_Day1_FunActivities5_title.tr(),
      key: Key('select_estimaton_fun'),
      isTimePicker: false,
      chosenType: ChosenType.Estimation,
      estimationType: EstimationType.Fun,
    )
);
```

4.5 Práce s databází

Interakce s databázi se prochází před `IUserRepository`. V Cubitu který už má instance IUserRepository zavoláme metodu:

```
    await _userRepository.addKid(kid);
```

V `HiveUserRepository`, která se dědi od `IUserRepository` se zavolá funkce

```
@override
  Future<void> addKid(KidJson kid) async {
    var userModelBox = await Hive.openBox<UserFull>('userModel');
    var user = userModelBox.getAt(0);
    user?.kidList.add(kid);
```

```
    user?.save();
  }
```

Nejprve, musíme otevřít krabice, se kterou budeme pracovat. Pokud chceme dostat určitý typ, musíme ho uvést při otevřeni krabice. To, že typ bude odpovídat požadovanému můžeme zaručit tím, ze používáme Hive adaptéry.

`UserFull` je třída děděna od `HiveObject`, což ji zpřístupňuje metodu `save()`, která ukládá upraveny objekt.

4.6 Vyhodnoceni

Vyhodnocení probíhá způsobem, že až pro dané dítě se projdou všechny sezení, na hlavni obrazovce se objeví kartičky, na které je možné zodpovědět bud dítě dodrželo pravidlo nebo limit v daný den, nebo ne. Nové kartičky se nahrávají každý den. Aplikace jednoduše umožňuje to nastavení v budoucnu změnit. Na základě odpovědi, získaných od rodičů v budoucnu bude vytvořeny graf, aby rodič měl přehled v čase, jak se pravidla a omezení dodržuji.

Při vstupu na hlavní obrazovku aplikace se podíváme do databáze, kdy jsme naposledy vyhodnocovali kartičky, a pokud je rozdíl vetší nuly pošleme kartičky z databáze na obrazovku, v opačném případě se na obrazovce zobrazí `Další den si zkontrolujte kartičky`.

```
_userRepository.getLastCardsUpdate().then((milisecondsFromEpoch) {
        if(DateTime.fromMillisecondsSinceEpoch(milisecondsFromEpoch)
              .difference(DateTime.now()).inDays != 0) {
          _userRepository.getCards().then((cards) {
            emit(state.copyWith(
                cardsToProcess: cards,
                status: RequestStatus.finish
            ));
          });
        } else {
          emit(state.copyWith(
              cardsToProcess: [],
              status: RequestStatus.finish
          ));
        }
      });
```

Vyhodnocení kartičky probíhá následujícím způsobem. Na prezentační úrovni změníme atribut kartičky `answer` na true/false, a pošleme, zavoláme metodu `finishCard` v Cubitu, aby se kartička odmazala.

```
    state.cardsToProcess[i].answer = true/false;
    context.read<CardCubit>().finishCard(state.cardsToProcess[i]);
```

Objekt stavu a všechny jeho atributy měli by být nezměnitelné, proto je potřeba nejprve vytvořit kopie atributu a tu upravenou kopie pak předávat do `emit`.

```
void finishCard(Card card) async {
    var cardsToProcess= List.of(state.cardsToProcess);
    var finishedCards= List.of(state.finishedCards);
    cardsToProcess.remove(card);
    finishedCards.add(card);
```

```
    emit(state.copyWith(
      finishedCards: finishedCards,
      cardsToProcess: cardsToProcess
    ));
}
```

Po vyhodnocení poslední kartičky se zavolá metoda `finish()`, která uloží odpovědi a aktuální čas.

```
void finish() async {
    await _userRepository.saveCardAnswers(state.finishedCards);
    await _userRepository.setLastSaveCards(DateTime.now());
    emit(state.copyWith(
        finishedCards: state.finishedCards,
        cardsToProcess: []
    ));
}
```

Kapitola 5
Testovaní

Jedním z cílů této práce je testování na reálných uživatelích. V rámci testovaní jsme se zaměřili na funkcionalitu aplikace, a jak přínosná je pro koncové uživatele. Testeři byli vybrané podle kritéria této aplikace, to znamená, že mají děti ve věku od 0 do 15 let. Kontakty zajistil vedoucí práce. K testovaní byla taky nabídnuta iOS aplikace pro uživatele iPhone.

5.1 Dotazník

Součásti testování byl dotazník, který se uživatele měli vyplnit po ukončeni testovaní. Otázky v dotazníku se dělí na uzavřené a otevřené.

5.1.1 Uzavřené otázky

Uzavřené otázky slouží pro testování ovládaní již existujícího prototypu a pro kontrolu užitečnosti aplikace. Zkratka Č.n označuje číslo otázky v grafu zobrazeném na 5.1.

- Č.1 - Instalace aplikace proběhla bez problémů.
- Č.2 - První přihlášení do aplikace a vytvoření nového dítěte bylo snadné.
- Č.3 - Používání aplikace bylo jednoduché a intuitivní.
- Č.4 - Aplikace fungovala bez problémů.
- Č.5 - Tipy a rady z jednotlivých sezení shledávám jako užitečnou pomůcku pro vytváření pravidel se svým dítětem.
- Č.6 - Pokud bych chtěl/a poučit své dítě o škodlivosti digitálních technologií, použil/a bych tuto aplikaci.
- Č.7 - Funkce přidání více dětí do aplikace je užitečná.
- Č.8 - Uživatelský účet s možnost přihlášení z jiného zařízení shledávám jako užitečné.
- C.9 - Aplikaci bych doporučil/a svým známým.

Uživatel mohl vybrat z teto nabídky odpovědi:

- Souhlasím
- Spíše souhlasím
- Ani souhlasím, ani nesouhlasím
- Spíše nesouhlasím
- Nesouhlasím

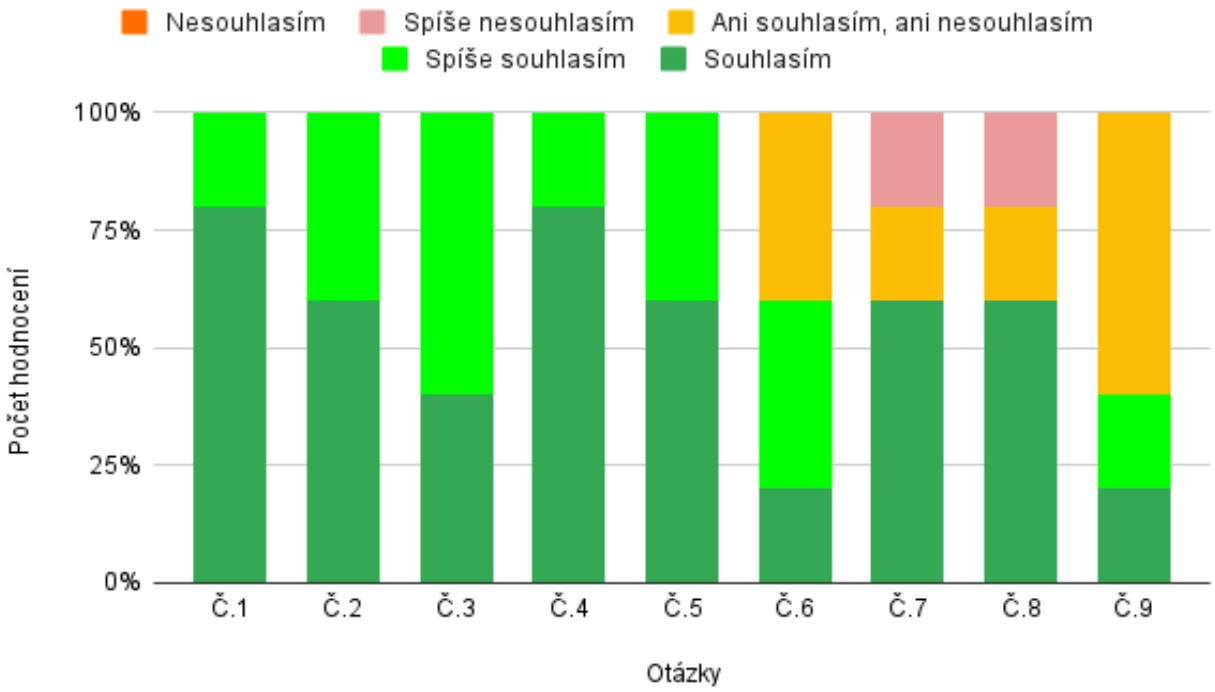
5.1.2 Otevřené otázky

Otevřené otázky jsou zaměřené na názor uživatele, a slouží pro zpětnou vazbu na další zlepšeni aplikace.

- Co se Vám na aplikaci líbilo nejvíce?

- Co se Vám naopak na aplikaci nelíbilo?
- Co byste na aplikaci do budoucna vylepšil/a?
- Co Vám v aplikaci chybělo? Co bylo naopak zbytečné?

5.1.3 Výsledky



Obrázek 5.1. Porovnáni počtu odpovědi dle otázky

Na grafu 5.1 jsou uvedené získané odpovědi na uzavřené otázky. Ve sloupci grafu je ukázán počet respondentů, a v řádku – číslo otázky.

Většina testerů neměla problém s nainstalováním a ovládáním aplikace. Majorita rodičů našla rady v aplikaci zajímavými a užitečnými. Jeden z testerů neocenil přidávání více děti. Předpokládám, že to bylo kvůli tomu, že má jedno dítě. Další uživatel neocenil přihlášení z více zařízení. Je to možné kvůli tomu, že aplikace je spíše edukační pro rodiče, a mohou to projít spolu s partnerem z jednoho zařízení. Tři z pěti testerů se váhají ohledné doporučení této aplikace svým známým, ostatní se hlásí na tuto otázku kladně.

Dále, následují odpovědi respondentů na otevřené otázky. Diakritika není brána v úvahu, odpovědi byli překopírované v poslaném tvaru.

Co se Vám na aplikaci líbilo nejvíce?

- Jednoduchost a informační obsah.
- Design
- Velmi povedené grafické zpracování aplikace a množství užitečných informací.
- Dcera nejdříve nechtěla na tématu úplně spolupracovat, ale s postupem v aplikaci se rozpovídala. Paráda. :-)
- Jednoduchý design

Co se Vám naopak na aplikaci nelíbilo?

- Nevím.
- Neni mi jasne, z jakeho duvodu pridavam dite, k cemu slouzi nastavovani pravidel?
- Některé možnosti byly pouze v angličtině. Např. zadávání data narození dítěte, kdy názvy měsíců nebyly přeložené do čestiny. Dále nepřeložena např. slova `password` a `delete`.
- Některé otázky nebyly přes postupové šipky či kvůli malému rámečku úplně vidět.

Co byste na aplikaci do budoucna vylepšil/a?

- Asi jsem uplne nepochopil, k cemu ma aplikace slouzit:-) Jen obecny pruvodce sezenimi, ktere si s ditetem projdu a spolecne si nastavime pravidla? Jak ale bude dite upozorneno, kdyz nejake pravidlo nedodrzi? Budou mu pravidla nejak automaticky pripominana?
- Narazila jsem na pár gramatických chyb, ale nebyly nijak zásadní. Lepší překlady do čestiny.
- Drobné překlepy, někdy zaclonění textu šipkami nebo malým rámečkem.
- Více obsahu

Co Vám v aplikaci chybělo? Co bylo naopak zbytečné?

- Uvítala bych podrobnější vysvětlení jak vlastně s dítětem s touto aplikací pracovat v běžném životě.
- Při nastavování času u obrazovek více zdůraznit:
  Nyní si stanovte se svým dítětem čas strávený u obrazovek, který chcete dodržovat.

Z technické strany uživatele upozornili na malé překlepy v gramatice a problém s lokalizaci. Tyto chyby byli vzaté v úvahu a budou upravené v další verzi aplikace.

Někteří respondenti neporozuměli principu použití této aplikace. Bude to vyřešeno detailnějším popsáním v `onboarding` sekci. Několik testerů znázornili, že jim chybí více obsahu a že by chtěli, aby aplikace měla větší interakce s dítětem. Tyto možnosti budou probrané s vedoucím práce a budou navržené v nejbližším čase.

V závěru můžeme říct, že většina testerů byla s aplikací spokojena a hodnotili ji pozitivně. Bylo zdůrazněno několik technických chyb, které budou co nejdříve opraveny.

Závěr

Na začátku byl analyzován a popsán problém digitálních závislosti, jejich negativní následky a doporučeni pro prevence. Dále nasledovala analýza existujících řešeni pro předcházeni digitální závislosti na trhu. Byli vybrané nejpopulárnější řešeni a popsané jejich klady a zápory. Potom byli popsané nejznámější multiplatformní řešeni a detailnější popis Flutter frameworku. Tento popis obsahuje princip vytvářeni aplikace v dánem frameworku a jeho zásadní častí.

Třetí kapitola se zabývala návrhem aplikace, její funkčními a nefunkčními požadavky, popisem funkcionality a použitými technologii a knihovnami.

V implementační častí byla popsaná struktura projektu a ukázky zásadních častí. Dále nasledovala testovací kapitola, ve které byli natrhnuté testovací otázky, a byla provedena analýza odpovědi testerů. Všechny názory byly vzaté v úvahu a budou implementované v nejbližší budoucnosti.

Ve závěru můžeme říct, že všechny cíle teto diplomové prací můžeme považovat za splněné, ale vývoj na aplikace se bude pokračovat i nadále.

Literatura

[1] *United brain association: Digital Addiction Fast Facts* [online]. [cit. 2022-04-10]. Dostupné z: https://unitedbrainassociation.org/brain-resources/digital-addiction

[2] Lukavská, K., Vacek, J., and Gabhelík, R. (2020). *The effects of parental control and warmth on problematic internet use in adolescents: A prospective cohort study. Journal of Behavioral Addictions 9, 3, 664-675, Dostupne z:* https://doi.org/10.1556/2006.2020.00068 *[cit. 2022-04-10]*

[3] Lukavska, Katerina. (2018). *The immediate and long-term effects of time perspective on Internet gaming disorder. Journal of Behavioral Addictions. 7. 1-8. 10.1556/2006.6.2017.089.*

[4] Pew Research Center: - Parenting Children in the Age of Screens [online]. [cit. 2022-04-12]. Dostupné z: https://www.pewresearch.org/internet/2020/07/28/ childrens-engagement-with-digital-devices-screen-time

[5] Slussareff, Michaela, and Kateřina Lukavská. Technologie a děti: Pediatrie pro praxi. [cit. 2022-04-12], Vol. 22, No. 2, pp. 117-120. ISSN 12130494. Dostupné z DOI 10.36290/ped.2021.021. Dostupné z: http://www.pediatriepropraxi.cz/doi/10.36290/ped.2021.021.html

[6] Internet Addiction. Addiction Center [online]. [cit. 2022-04-12]. Dostupné z: https://www.addictioncenter.com/drugs/internet-addiction

[7] Internet addiction. The center of parenting education [online]. [cit. 2022-04-12]. Dostupné z: https://centerforparentingeducation.org/library-of-articles/kids-and-technology/how-much-time-internet-kids

[8] Kaspersky [online].[cit. 2022-04-13]. Dostupné z: https://www.kaspersky.cz/safe-kids

[9] Google Family Link [online]. [cit. 2022-04-13]. Dostupné z: https://www.reviewgeek.com/74191/google-family-links-new-update-gives-parents-better-control-of-app-time-limits

[10] Apple Parental control [online]. [cit. 2022-04-13]. Dostupné z: https://support.apple.com/enus/HT201304

[11] Qustodio [online].[cit. 2022-04-14]. Dostupné z: https://reviews.vc/review/qustodio

[12] Statista [online]. [cit. 2022-04-14]. Dostupné z: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours

[13] Cross-Platform Mobile Apps Development – Pros and Cons [online]. [cit. 2022-04-15]. Dostupné z: https://www.netguru.com/blog/pros-and-cons-of-cross-platform-mobile-app-development

[14] Documentation. Apache Cordova [online]. [cit. 2022-04-14]. Dostupné z: https://cordova.apache.org/docs/en/11.x/guide/overview/index.html

[15] *React Native OTA update.* Dev [online]. [cit. 2022-04-15]. Dostupné z: https://dev.to/erdenezayaa/react-native-ota-update-how-to-deploy-new-version-of-in-1-minute-cfi

[16] Gadireddi, Erik. Design of Therapeutical Application for Digital Addiction Reduction. Bachelor Thesis. České vysoké učení technické v Praze, Fakulta elektrotechnická, 1 2022

[17] SINELNIKOV, Serhii. *Porovnání vývojových nástrojů pro tvorbu multiplatformních mobilních aplikací.* 2019. Bakalářské práce. Univerzita J.E. Purkyně v Ústí nad Labem, Katedra informatiky. Vedoucí práce Jiří Fišer.

[18] Inside flutter [online]. [cit. 2022-04-16]. Dostupné z: https://flutter.dev/docs/resources/inside-flutter

[19] Dart Overview. Dart [online]. [cit. 2022-04-16]. Dostupné z: https://dart.dev/overview

[20] Flutter architectural overview [online]. [cit. 2022-04-16]. Dostupné z: https://docs.flutter.dev/resources/architectural-overview

[21] Asynchronous programming: futures, async, await. Dart [online]. [cit. 2022-04-19]. Dostupné z: https://dart.dev/codelabs/async-await

[22] The Problem Of Async Programming. Medium [online]. [cit. 2022-04-19]. Dostupné z: https://medium.com/connect-platform/the-problem-of-async-programming-and-a-crazy-idea-for-solving-it-cf368a9ea949

[23] Isolates and Event Loop. Medium [online]. [cit. 2022-04-22]. Dostupné z: https://medium.com/dartlang/dart-asynchronous-programming-isolates-and-event-loops-bffc3e296a6a

[24] Event loop. Dart [online]. [cit. 2022-04-22]. Dostupné z: https://dart.cn/articles/archive/event-loop

[25] *Concurrency.* Dart [online]. [cit. 2022-04-22]. Dostupné z: https://dart.dev/guides/language/concurrency

[26] Dart - Spawn, Kill & Send Message to Isolate. Woolha [online]. [cit. 2022-04-25]. Dostupné z: https://www.woolha.com/tutorials/dart-spawn-kill-send-message-to-isolate

[27] StatelessWidget. Flutter [online]. [25]. Dostupné z: https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html

[28] Relation Between Stateful and Stateless Widgets. Flutter agency [online]. [cit. 2022-04-17]. Dostupné z: https://flutteragency.com/relation-between-stateful-and-stateless-widgets-in-flutter

[29] State Management. Flutter [online]. [cit. 2022-04-25]. Dostupné z: docs.flutter.dev/development/data-and-backend/state-mgmt/declarative

[30] Introduction to Redux. Novoda [online]. [cit. 2022-04-25]. Dostupné z: https://blog.novoda.com/introduction-to-redux-in-flutter

[31] Bloc. Bloc Library [online]. [cit. 2022-04-28]. Dostupné z: https://bloclibrary.dev

[32] What is API. AWS [online]. [cit. 2022-04-28]. Dostupné z: https://aws.amazon.com/what-is/api

[33] *Zdroják: REST - architektura pro webové API* [online]. [cit. 2022-04-28]. Dostupné z: https://zdrojak.cz/clanky/rest-architektura-pro-webove-api

[34] HTTP methods. REST API Tutorial [online]. [cit. 2022-05-02]. Dostupné z: https://restfulapi.net/http-methods

[35] flutter_bloc. Pub.dev [online]. [cit. 2022-05-02]. Dostupné z: https://pub.dev/packages/flutter_bloc

[36] Get it. Pub.dev [online]. [cit. 2022-05-02]. Dostupné z: https://pub.dev/packages/get_it

[37] Json_serializable. Pub.dev [online]. [cit. 2022-05-03]. Dostupné z: https://pub.dev/packages/json_serializable

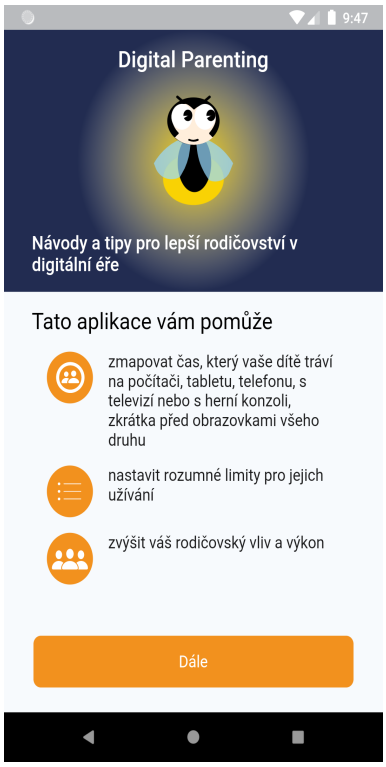
[38] Hive. Pub.dev [online]. [cit. 2022-05-03]. Dostupné z: https://pub.dev/packages/hive

[39] Easy Localization. Pub.dev [online]. [cit. 2022-05-05]. Dostupné z: https://pub.dev/packages/easy_localization

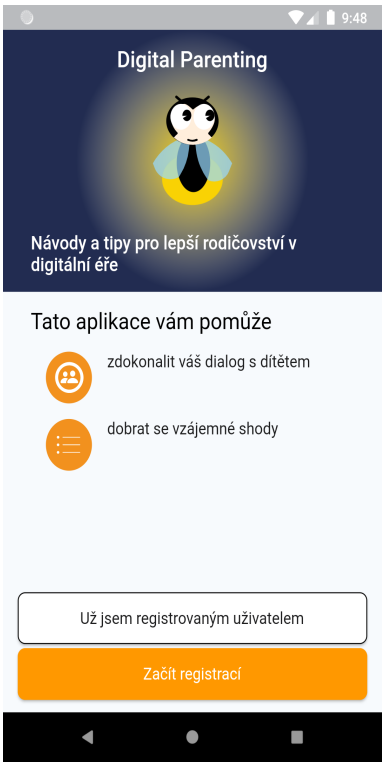
[40] Material App. Flutter [online]. [cit. 2022-05-05]. Dostupné z: https://api.flutter.dev/flutter/material/MaterialApp-class.html

[41] Navigator. Flutter [online]. [cit. 2022-05-05]. Dostupné z: https://api.flutter.dev/flutter/widgets/Navigator-class.html

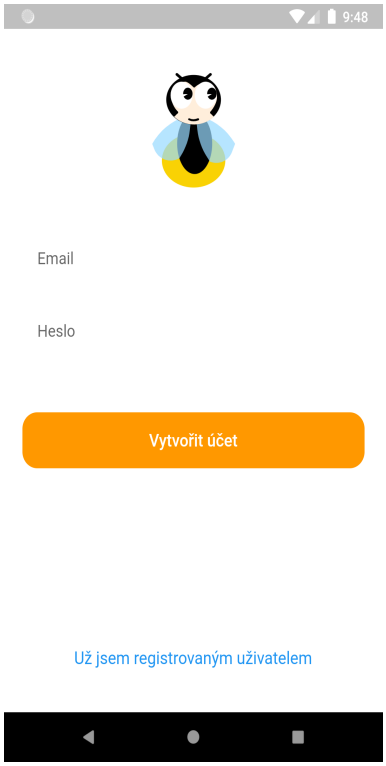
Příloha A
Obrazovky



Obrázek A.1. Onboarding 1

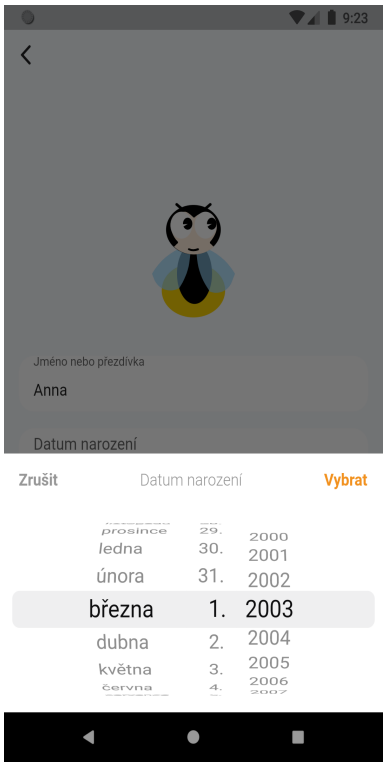


Obrázek A.2. Onboarding 2

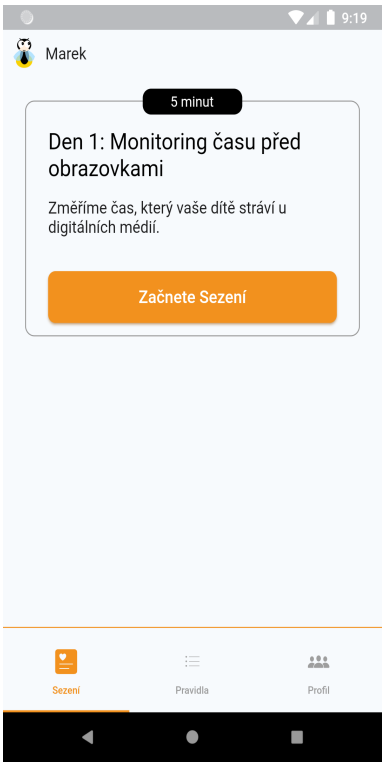


Obrázek A.3. Registrace.

Obrázek A.4. Přidávaní dítě

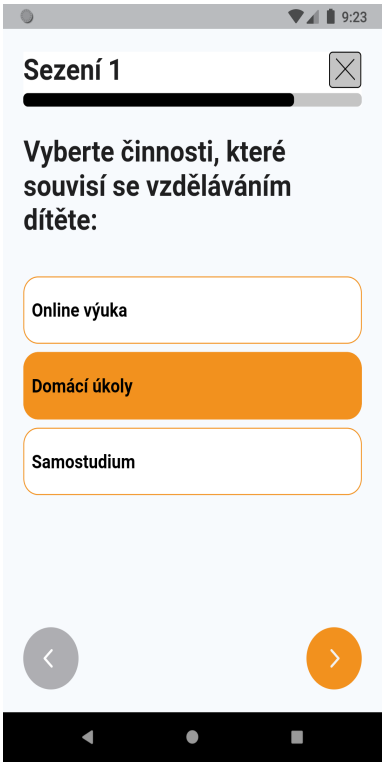


Obrázek A.5. Datum narozeni



Obrázek A.6. Hlavní obrazovka

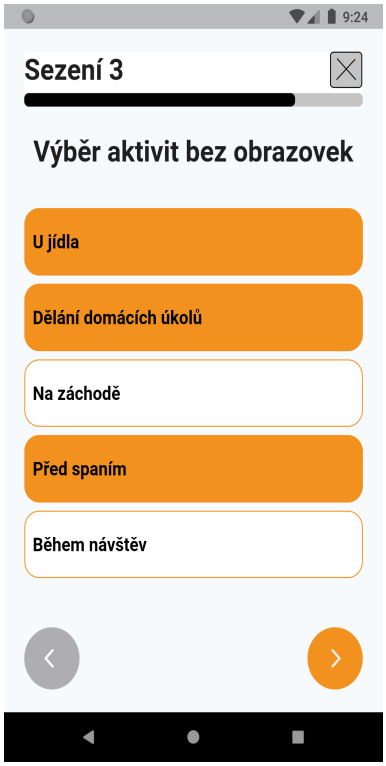
Obrázek A.7. První sezení



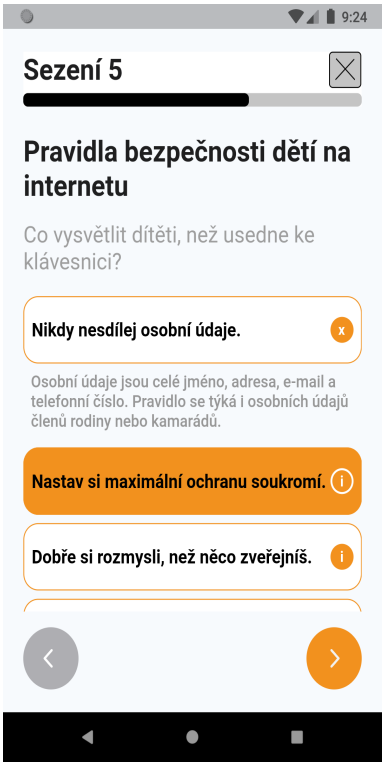
Obrázek A.8. Výběr činnosti

Obrázek A.9. Sezení 2

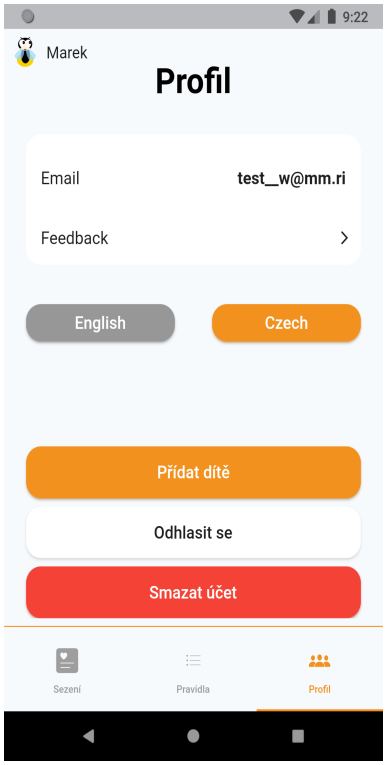
Obrázek A.10. Nastavení limit v sezení



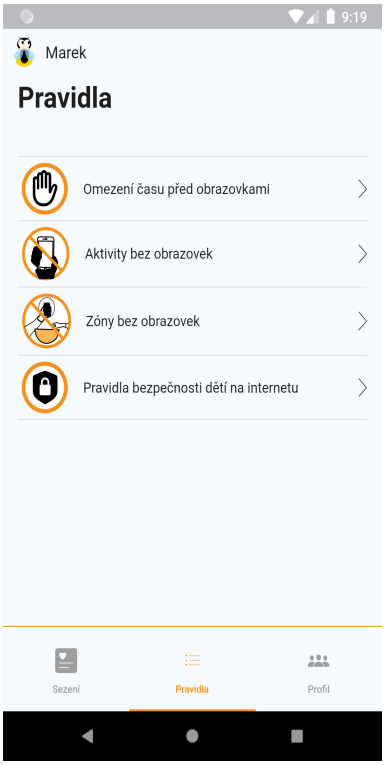
Obrázek A.11. Výběr aktivit v sezení



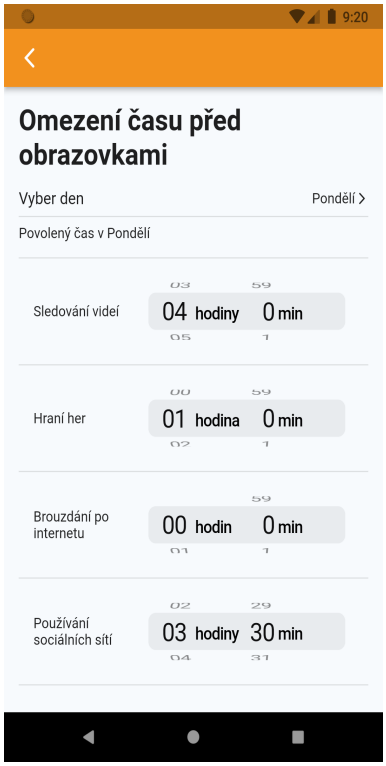
Obrázek A.12. Výběr pravidel v sezení



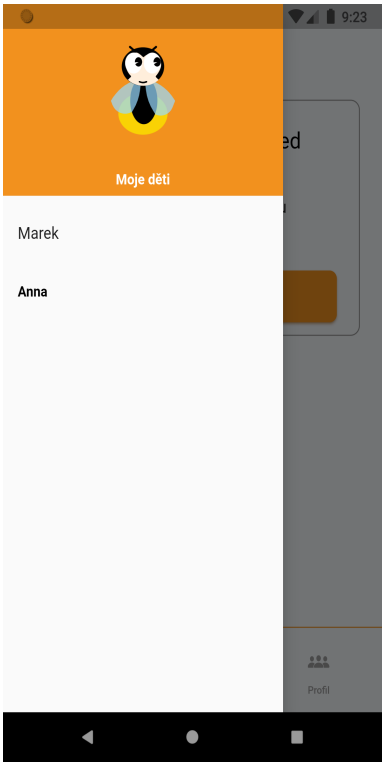
Obrázek A.13. Profil



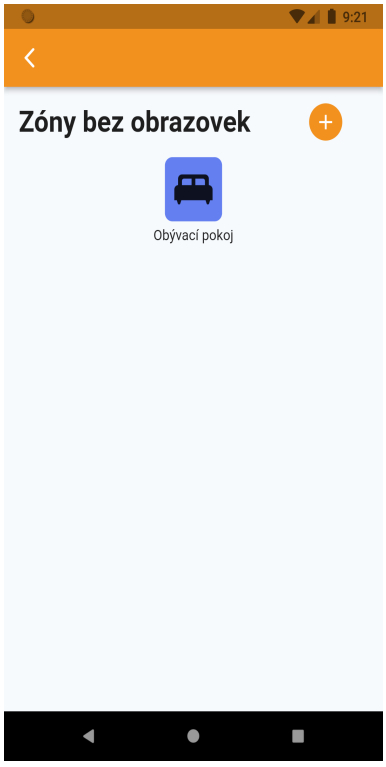
Obrázek A.14. Pravidla



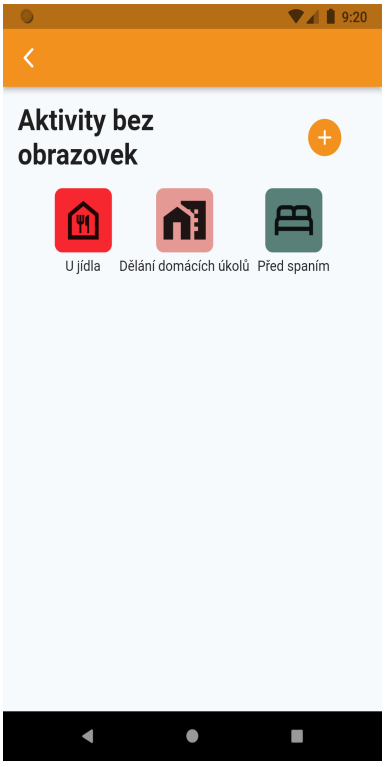
Obrázek A.15. Nastavení času v pravidlech



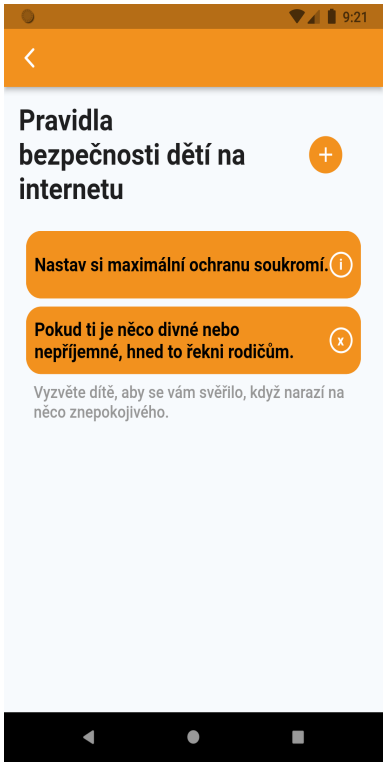
Obrázek A.16. Výběr aktuálního dítě



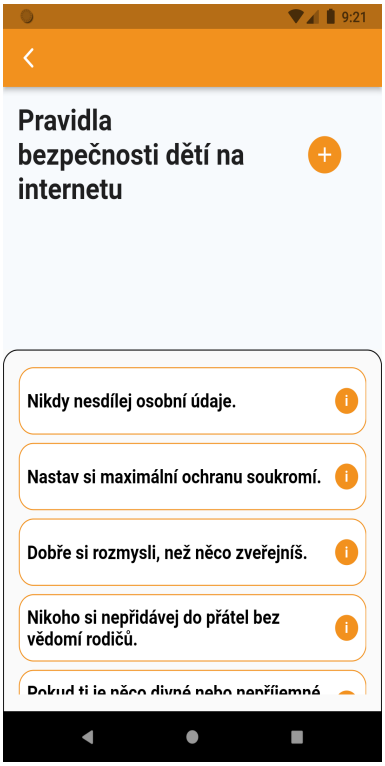
Obrázek A.17. Zóny v pravidlech



Obrázek A.18. Aktivity v pravidlech



Obrázek A.19. Pravidla bezpečnosti



Obrázek A.20. Přidávaní pravidel bezpečnosti

Příloha B
Obsah přiloženého CD

`readme.txt`	Návod na sestaveni a spuštěni aplikace
`digital_parenting_code.zip`	Kód projektu
`digital_parenting_app.apk`	Instalační soubor aplikace.
`diplomova-prace.pdf`	Text diplomové práce.