Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra systémového inženýrství

Diplomová práce

Vývoj mobilní aplikace pro Android a iOS, která je zaměřena na sportovní výcvik psů

Bc. Juraj Kožík

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Juraj Kožík

Informatika

Název práce

Vývoj mobilní aplikace pro Android a iOS, která je zaměřena na sportovní výcvik psů.

Název anglicky

Development of a mobile application for Android and iOS, which is focused on the sports training of dogs.

Cíle práce

Cílem diplomové práce je navrhnout a vyvinout mobilní aplikaci pro platformy Android a iOS, která bude sloužit jako nástroj pro majitele psů, poskytující užitečné informace, funkce a komunitní prvky v technologii React Native. Aplikace bude poskytovat uživatelům přehledné a interaktivní prostředí pro trénink a výcvik psů. Hlavním zaměřením bude umožnit uživatelům vytvářet a spravovat cvičební plány a sledovat pokrok svých psů. Hlavním cílem práce poté bude analýza, návrh a následná implementace funkční aplikace, která bude přístupná na mobilních zařízeních s operačním systémem iOS a Android.

Metodika

Práce sestává z teoretické a praktické části. V teoretické práci se bude pracovat s odbornými informačními zdroji a na základě takto získaných poznatků. Dále budou popsány vybrané metody a technologie zvolené pro praktickou část práce, a to programovací jazyky, jednotlivé knihovny a technologie pro vývoj multiplatformní aplikace pomocí frameworku React Native.

Praktická část se bude skládat z analýzy, návrhu a samotné implementace mobilní aplikace zaměřené na výcvik psů. Analýza bude zaměřena na rozvržení aplikace a definici jednotlivých funkcionalit aplikace, které budou implementovány. Analýza bude základem pro volbu potřebných technologií, knihoven pro vhodné vyhotovení aplikace. Následovat bude implementace, kde budou vytvořeny komponenty a funkce pro tvořenou aplikaci. Výsledná aplikace bude uživatelsky otestována a budou zhodnoceny dosažené výsledky a poznatky. Závěrem budou diskutovány další možnosti a rozšíření aplikace.

Doporučený rozsah práce

60-80

Klíčová slova

React Native, Expo, iOS, Android, multiplatformní aplikace, vývoj aplikace

---

Doporučené zdroje informací

Eloquent JavaScript, 3rd edition (2018) [online]. HAVERBEKE, Marijn. [cit. 2023-06-10]. Dostupné z: https://eloquentjavascript.net
Expo Documentation [online]. [cit. 2023-06-10]. Dostupné z: https://reactnative.dev/docs/getting-started
Introduction [online]. [cit. 2023-06-10]. Dostupné z: https://reactnative.dev/docs/getting-started

---

Předběžný termín obhajoby

2023/24 LS – PEF

Vedoucí práce

Ing. Martin Pelikán, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 10. 9. 2023

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 3. 11. 2023

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 28. 03. 2024

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Vývoj mobilní aplikace pro Android a iOS, která je zaměřena na sportovní výcvik psů" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 29.03.2024 _____

Poděkování

Rád bych poděkoval panu Ing. Martinu Pelikánovi, Ph.D. za jeho cenné rady a podporu během celého procesu psaní této práce. Také bych chtěl vyjádřit vděk své rodině za jejich podporu, pochopení a povzbuzení během mých studií. Nakonec bych chtěl poděkovat svým přátelům za jejich podporu a užitečné diskuse, které mi pomohly rozvinout mé myšlenky.

Vývoj mobilní aplikace pro Android a iOS, která je zaměřena na sportovní výcvik psů

Abstrakt

Teoretická část práce je zaměřena na výběr React Native frameworku a porovnání s alternativami pro vývoj multiplatformních mobilních aplikací. Dále jsou podrobně popsány jednotlivé použité technologie využité u vývoje aplikace, jako například Expo a technologie poskytované platformou Firebase.

Praktická část práce je zaměřena na popis vývoje aplikace. Prvním krokem je analýza požadavků na mobilní aplikaci, poté následuje analýza technologií pro samotný vývoj aplikace a analýza uživatelského prostředí a návrh řešení. Po navržení všech potřebných částí aplikace následuje popis konfigurace prostředí a dalších technologií a samotná implementace aplikace pomocí React Native. Poslední částí je validace funkčnosti aplikace, kde je vyobrazena funkčnost aplikace.

Klíčová slova: React Native, Expo, iOS, Android, multiplatformní aplikace, vývoj aplikace

Development of a mobile application for Android and iOS, which is focused on sports training of dogs

Abstract

The theoretical part of the thesis is focused on the selection of React Native framework and comparison with alternatives for the development of multiplatform mobile applications. Furthermore, the various technologies used in the development of the application, such as Expo and the technologies provided by the Firebase platform, are described in detail.

The practical part of the thesis focuses on the description of the application development. The first step is the analysis of the requirements for the mobile application, followed by the analysis of the technologies for the actual development of the application and the analysis of the user experience and the design of the solution. After designing all the necessary parts of the application, the next step is to describe the configuration of the environment and other technologies and the actual implementation of the application using React Native. The last part is the validation of the application functionality where the functionality of the application is demonstrated.

Key words: React Native, Expo, iOS, Android, multiplatform applications, application development

Obsah

1 Úvod

V dnešní době, kdy mobilní technologie ovlivňují téměř každý aspekt našeho života, je klíčové hledat inovativní způsoby, jak využívat tuto sílu ke zlepšení našich každodenních činností. Jednou z oblastí, kde může moderní mobilní aplikace přinést výrazný přínos, je výcvik domácích mazlíčků, konkrétně psů. Věrnost, oddanost a radost, kterou psi přinášejí do našich životů, jsou nepřeberné, avšak proces jejich výcviku může být náročný a vyžaduje systematický a pozitivní přístup. Vzhledem k dnešní velké nabídce a široké dostupnosti chytrých telefonů by aplikace byla dostupná pro široké spektrum uživatelů.

Diplomová práce je zaměřena na vývoj mobilní aplikace pro výcvik psů, která využívá moderní technologie jako React Native a Expo. Tyto nástroje nám umožňují vytvářet multiplatformní aplikace s vysokou úrovní flexibility a uživatelské přívětivosti. V kombinaci s pokročilými možnostmi mobilních zařízení může tato aplikace poskytnout prostředí, ve kterém majitelé psů mohou efektivně a zábavně pracovat na výcviku svých chlupatých společníků.

Výběr tématu diplomové práce byl motivován zájmem o problematiku vývoje mobilních aplikací a jejich využití pro běžné aspekty života a získávání dalších praktických zkušeností s vývojem.

2 Cíl práce a metodika

V následujícím textu bude představen cíl diplomové práce a metodika zpracování práce, která přiblíží členění práce a metody a technologie použité pro praktickou část práce.

2.1 Cíl práce

Cílem diplomové práce je vytvořit komplexní mobilní aplikaci pro výcvik psů, která kombinuje nejlepší prvky učení, interakce a motivace. Aplikace bude vyvinuta pomocí technologií React Native a Expo, umožňujících snadnou dostupnost na různých mobilních platformách. V rámci této práce budou definovány klíčové funkce aplikace, implementován systém uživatelského rozhraní a provedeno testování, aby se zajistila funkcionalita a uživatelská spokojenost.

2.2 Metodika

Práce sestává z teoretické a praktické části. V teoretické práci se bude pracovat s odbornými informačními zdroji a na základě takto získaných poznatků. Dále budou popsány vybrané metody a technologie zvolené pro praktickou část práce, a to programovací jazyky, jednotlivé knihovny a technologie pro vývoj multiplatformní aplikace pomocí frameworku React Native. Praktická část se bude skládat z analýzy, návrhu a samotné implementace mobilní aplikace zaměřené na výcvik psů. Analýza bude zaměřena na rozvržení aplikace a definici jednotlivých funkcionalit aplikace, které budou implementovány. Analýza bude základem pro volbu potřebných technologií, knihoven pro vhodné vyhotovení aplikace. Následovat bude implementace, kde budou vytvořeny komponenty a funkce pro tvořenou aplikaci. Výsledná aplikace bude uživatelsky otestována a budou zhodnoceny dosažené výsledky a poznatky. Závěrem budou diskutovány další možnosti a rozšíření aplikace.

3 Teoretická část práce

V této kapitole budou podrobně rozebrána teoretická východiska, která jsou klíčová pro pochopení a implementaci aplikace pro výcvik psů v prostředí React Native s využitím Expo a dalších nástrojů.

3.1 React

React představuje moderní JavaScriptový framework pro tvorbu uživatelských rozhraní (1). Jeho koncepty se výrazně liší od tradičních přístupů k vývoji webových stránek, což jej činí vhodným nástrojem pro vytváření dynamických a efektivních mobilních aplikací.

React Native je rozšířením Reactu, které umožňuje vývoj mobilních aplikací pro iOS a Android s využitím známé syntaxe Reactu (2). Na rozdíl od tradičního vývoje, React Native umožňuje sdílet kód mezi oběma platformami, což výrazně usnadňuje a zrychluje vývoj.

3.2 React Native

Použití React Native pro vývoj mobilní aplikace má více důvodů. Prvním z nich je určitě multiplatformní podpora. S React Native se kód sdílí mezi platformami iOS a Android, což by v praxi znamenalo snížení nákladu na vývoj. Funkce "Hot Reload" umožňuje okamžitě vidět změny kódu v reálném čase během vývoje v simulátoru jak pro iOS, tak i Android zařízení, což zrychluje proces vývoje. Pro iOS simulátor je zapotřebí přístup k zařízení s macOS a XCode. React Native používá jazyk JavaScript, který je jedním z nejrozšířenějších a nejrychleji se rozšiřujících jazyků (3). Tento jazyk patří do skupiny jazyků jako C++/C#/Java a je používán pro tvorbu webových stránek. Proto je tento jazyk lehce naučitelný a tím pádem lehce uchopitelný. Velká komunita vývojářů pracujících s React Native poskytuje spoustu dostupných knihoven, modulů a návodů, což usnadňuje řešení problémů a implementaci nových funkcí. React Native je aktivně vyvíjen a podporován firmou Meta, což znamená pravidelné aktualizace, opravy chyb a inovace. Samotná společnost Meta používá React Native pro vývoj aplikací jako je Facebook, Oculus nebo například Messenger Desktop. I další velké společnosti na této platformě staví svoje aplikace. Microsoft využívá React Native například pro Microsoft Office, Outlook a Microsoft Teams. Firem využívajících React Native je ale mnohem více. React Native umožňuje dosahovat vysokého výkonu, protože využívá nativní prvky. Aplikace se tak chovají podobně jako plně nativní aplikace. Pokud je preferován jednodušší začátek s méně složitými požadavky, lze využít Expo, což je nadstavba nad React Native, která eliminuje některé administrativní úkoly jako je nastavení a nasazení (4). React

Native má příznivou podporu pro asynchronní operace, což z něj činí ideální volbu pro integraci s backendovými službami.

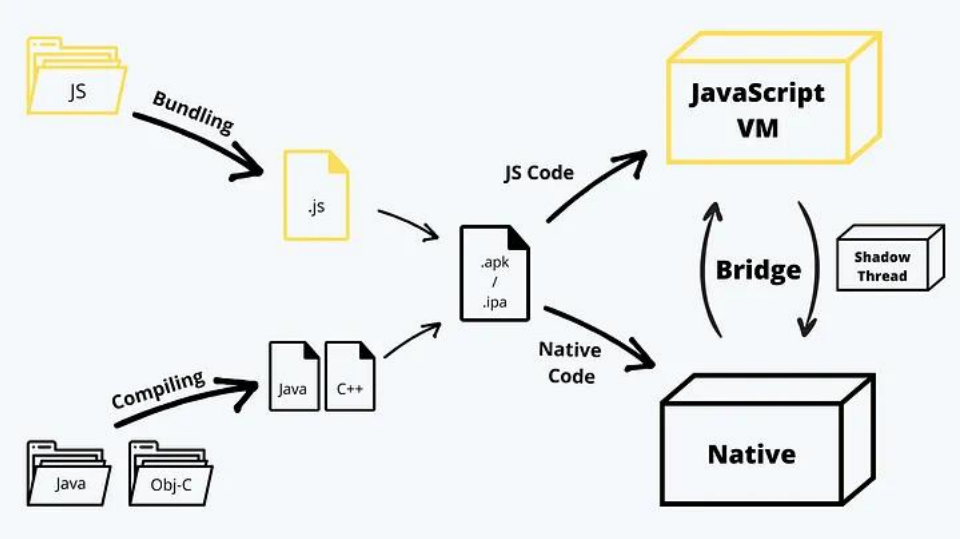
Celkově vzato, použití React Native při vývoji nové mobilní aplikace nabízí kombinaci efektivního vývoje, vysoké flexibility a široké podpory komunity, což může být pro mnohé projekty optimální volbou.

3.2.1 Platformy podporované React Native

Pro plné využití výhod React Native je důležité porozumět, které platformy jsou podporovány. Podporované platformy jsou zejména iOS, Android a Web (5). Možný je vývoj aplikací i pro platformy macOS a Windows. Pro iOS nabízí React Native plnou podporu pro vývoj na platformě iOS, což zahrnuje především iPhone, iPad a potažmo další zařízení od společnosti Apple. Vývoj nativní aplikace umožňuje přímý přístup k všem nativním prvkům a API (rozhraním pro programování aplikací) operačního systému iOS. To umožňuje vývojářům plně využít specifické funkce a schopnosti zařízení Apple. React Native poskytuje kompletní podporu pro vývoj na platformě Android. Výhodou je možnost vytváření nativní Android aplikace s vysokým výkonem a přímý přístup k nativním prvkům a API operačního systému Android.

3.2.2 Jak funguje React Native

Myšlenka za fungováním React Native je kombinace dvou separátních komponent, a to konkrétně JavaScript kódu a Nativního kódu – Kotlin/Java pro Androin a Objective-C/Swift pro iOS, a přiměje je mezi sebou spolupracovat (6). Na odlišných platformách se framework chová různě. Na iOS zařízeních je zabudovaný engine JavaScriptCore, který kompiluje a spouští JavaScript kód. Android zařízení nemají zabudovaný podobný engine, proto je spojen s React Native frameworkem. JavaScript se liší od nativních jazyků, Java/Objective-C, tudíž nemohou napřímo komunikovat a vyměňovat si data a signály. Obě strany, jak JavaScript tak Java/Objective-C, umí pracovat s datovým formátem JSON a proto spolu mohou komunikovat přes prostředníka. V React Native je prostředníkem skupina programů, která je pojmenována Bridge.



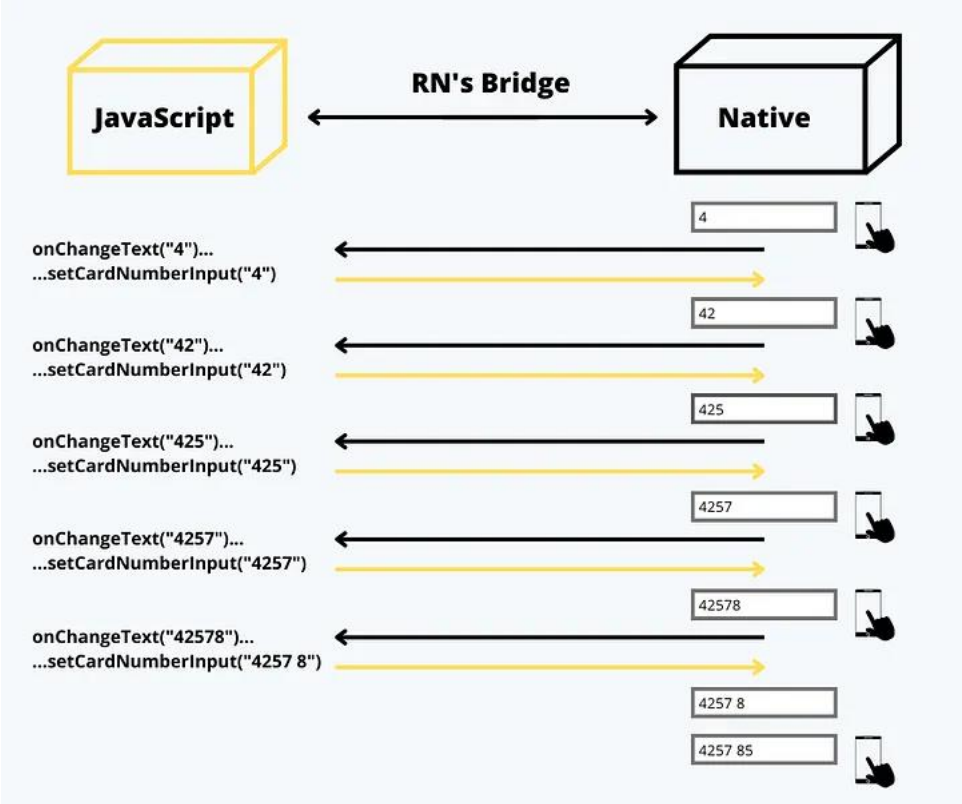
Obrázek 1 React Native Bridge technologie (Internetový zdroj dle (6))

Technologie Bridge je důležitým prvkem v React Native architektuře zejména kvůli již zmíněné funkcionalitě, která umožňuje komunikaci mezi nativní vrstvou a vrstvou JavaScriptovou pomocí dat ve formátu JSON. Tento přístup se sebou přináší i problémy zejména v krajních případech, a to hlavně proto, že Bridge je asynchronní. V některých případech však asynchronní přístup nestačí a bylo by lepší použít synchronní přístup. Například implementace jednoduchého vstupního pole, kam uživatel zadává číslo karty a vývojář by chtěl implementovat do tohoto pole, aby po každých čtyřech číslicích byla mezera – jako na fyzické kartě. Tuto funckionalitu lze implementovat například pomocí funkce, která bude zavolána pokaždé, když uživatel napíše číslo:

```
const handleCardNumberInputChange = (cardNumber) => {
  const formatted = cardNumber.toString().replace(/\d{4}(?=.)/g, '$& ');
  setCardNumberInput(formatted)
}
```

Obrázek 2 Funkce prezentující fungování Bridge (Internetový zdroj dle (6))

Jak je vidět na obrázku níže, ze strany nativního kódu je číslice již vykreslena na obrazovce, zatímco přichází na stranu JavaScriptové funkce. Až poté se tato funkce zabývá tím, jestli má být vstup ponechán beze změny, anebo jestli se má vložit mezera mezi již zadané poslední číslo. V druhém případě dochází k překreslení vstupního pole. Uživatel nejdřív vidí pět číslic vedle sebe a až potom se mezi čtvrté číslo a páté číslo vloží mezera (‘42578’ se změní na ‘4257_8’).



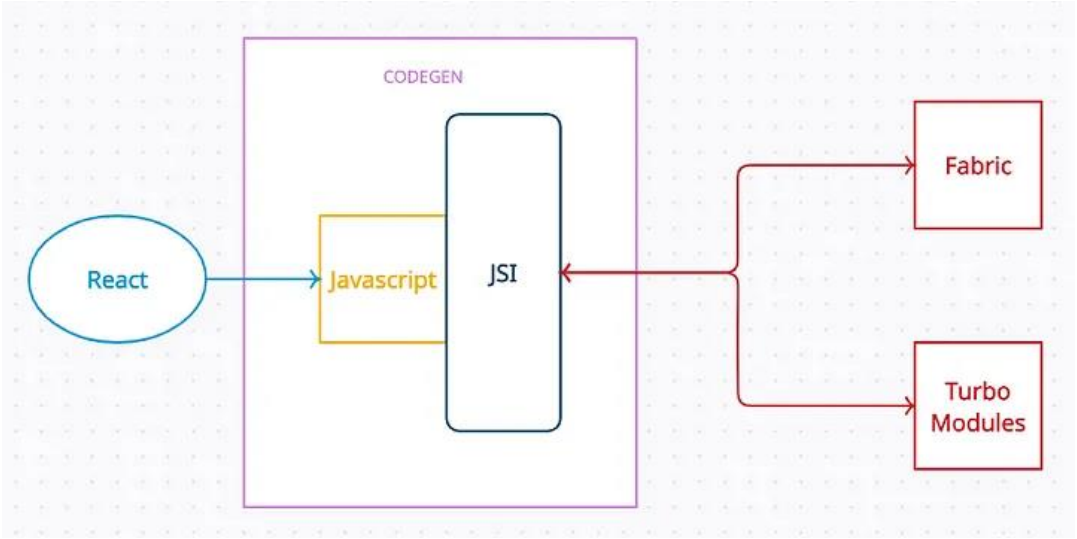
Obrázek 3 Fungování React Native Bridge mezi JavaScriptem a Native (Internetový zdroj dle (6))

V případě, že by Bridge zpracovával pouze tuto změnu, tak by operace předávaní informací byla tak rychlá, že by si to uživatel ani nevšimnul. Nicméně může probíhat více operací vyměňujících si informace a zároveň operace na pozadí. Dalším problémem je, že informace, které se vyměňují mezi JavaScript vrstvou a Nativní vrstvou, je potřeba serializovat a deserializovat. Tyto operace jsou časově náročné. Čím víc aplikace roste, tím víc roste počet těchto operací, které tím víc zatěžují Bridge.

Kvůli zmíněným problémům s touto technologií se od verze React Native 0.68 začalo používat JavaScript Interface (JSI) (7). Toto nové rozhraní je sjednocené. Jedná se o univerzální vrstvu, kterou možno použít v jakémkoliv JavaScript enginu a odstraňuje problémy, které Bridge způsoboval. Pomocí HostedObject jsou nativní metody a objekty vystaveny JavaScriptu, který bude obsahovat odkaz na tento objekt. To umožní přístup k metodám a objektům napřímo. Provést to lze synchronně ve stejném vlákně nebo asynchronně vytvořením nového vlákna, a to v podstatě posouvá React Native na zcela novou úroveň pomocí využití JavaScriptu a C++. Pomocí JSI může JavaScript držet reference na hostitelské objekty a volat jejich metody. Pomocí těchto objektů je možné používat nativní objekty v JavaScriptovém kontextu.

Výše zmíněný Bridge, který byl nevyhovující kvůli tomu, že byl největším hrdlem je nyní rozdělen do následujících částí:

- Fabric
- Turbo Modules
- Codegen



Obrázek 4 Fungování JavaScript Interface (Internetový zdroj dle (7))

Fabric je nový systém vykreslování od společnosti Facebook. Nový systém je přebudováním správce uživatelského rozhraní. Je implementován pomocí jazyka C++ a jeho jádro je sdílené mezi platformami. V předchozí implementaci tvorba rozložení (layout) zahrnovala několik kroků jako třeba serializaci JSON a přechody přes Bridge, což mělo velký dopad v podobě chyb v momentě, kdy byl Bridge zatížený. Výsledný dopad byl v podobě snížení plynulosti při procházení seznamu, který mohl být dlouhý – až nekonečně dlouhý. Nová implementace správce uživatelského rozhraní umožňuje vytvořit Shadow Tree přímo v C++, což výrazně zlepšuje uživatelskou interakci tím, že snižuje počet skoků mezi různými oblastmi. Operace jsou synchronní a bezpečné v rámci vlákna, převáděné z Reactu (JavaScript) do vykreslovače (C++), obvykle na vlákně JavaScriptu. Tato implementace rovněž snižuje serializaci dat, protože hodnoty JavaScriptu lze přímo získat z JSI. Tato přímá kontrola také pomáhá při prioritizaci akcí; vykreslovač nyní může určit pořadí určitých uživatelských interakcí, aby bylo zajištěno jejich promptní zpracování. To výrazně zlepšuje výkon při práci se seznamy, navigací a při zpracování gest.

Pro Turbo Modules v předchozí implementaci nebyl dostupný odkaz na nativní moduly, takže každý modul byl načten při spuštění, což zvyšovalo čas do interaktivity (TTI – Time To Interactive). S Turbo Moduly (Turbo Modules) je možné moduly načítat až tehdy, když jsou skutečně potřeba, což pomáhá zlepšit čas spuštění. Možnost psát moduly v jazyce C++ přináší výhodu v tom, že snižuje duplicitní implementaci napříč různými platformami.

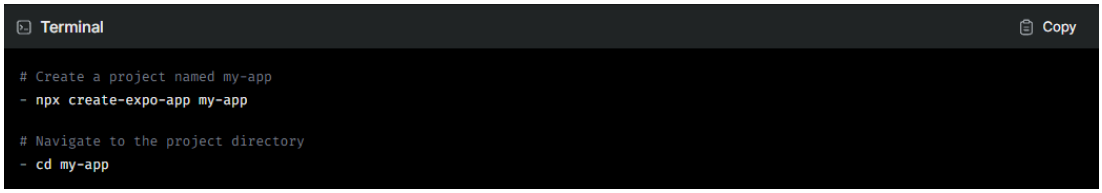
Codegen slouží k propojení obou světů – JS a nativní části v React Native. Zajišťuje kompatibilitu a automatizovanou typovou kontrolu a používá modulární přístup, umožňující podporu různých staticky typovaných jazyků JavaScriptu. Tím, že využívá typovaný JavaScript, generuje rozhraní pro Fabric a TurboModuly, což umožňuje bezpečný přenos zpráv mezi oblastmi. Codegen přináší bezpečnost typů v době kompilace, což znamená menší velikost kódu a rychlejší provedení. Tím, že vyžaduje typování i v JavaScript kódu, umožňuje vytvářet důvěryhodná data a odhalit potenciální problémy již během vývoje. Snižuje riziko fatálních chyb a zlepšuje uživatelskou zkušenost.

Mezi hlavní výhody JSI patří:

- Přímá manipulace s nativními UI prvky
- Rychlé a přímé vazby na nativní kód
- Kompatibilita s různými engine, ne pouze JavaScript
- Lazy loading nativních modulů
- Statická kontrola typů

3.3 Expo

Expo představuje nadstavbu nad React Native, která usnadňuje a urychluje vývoj mobilních aplikací. Je zaměřen na odstranění některých administrativních úkolů a umožnuje vývojářům zaměřit se na samotný obsah aplikace. Expo poskytuje jednoduchý přístup ke startu nového projektu. S využitím nástroje Expo CLI (Command Line Interface) lze vytvořit nový projekt jedním příkazem, což zjednodušuje inicializační proces.



Obrázek 5 Vytvoření Expo aplikace (Internetový zdroj dle (4))

Jedním z hlavních přínosů Expo frameworku je možnost vyvíjet aplikace bez nutnosti instalovat a konfigurovat nativní vývojové nástroje pro iOS a Android. Tím pádem lze začít s vývojem i bez Macbooku pro vývoj iOS aplikace. Expo umožňuje snadné nasazení aplikací. Expo Hosted Services mohou hostovat aplikaci, což eliminuje potřebu konfigurace serverů a infrastruktury pro nasazení. Expo podporuje Hot Reloading, a tím umožňuje vidět změny v reálném čase během vývoje. Live preview umožňuje okamžité testování na reálném zařízení pomocí Expo Client aplikace. Expo umožňuje vytvářet univerzální kód, který může běžet na iOS, Android i webových platformách, což zjednodušuje správu kódu a minimalizuje duplicity. Expo tedy slouží jako nástroj, který zjednodušuje mnoho aspektů vývoje v React Native. Představuje ideální volbu pro projekty s nižší složitostí a pro vývojáře, kteří chtějí rychle začít s vývojem bez potřeby konfigurace složitějších nástrojů.

Expo Go je mobilní aplikace, která funguje jako klient pro Expo projekty. Tato aplikace je klíčovým prvkem Expo frameworku a umožňuje vývojářům okamžitě vizualizovat svou práci na reálných mobilních zařízeních. Pro využití stačí aplikaci nainstalovat na mobilní zařízení Apple s iOS verze 13 nebo novější anebo Android zařízení s verzí Lollipop (5) nebo novější. Poté stačí naskenovat QR kód – pro iOS aplikací Fotoaparát, pro Android zařízení pomocí aplikace Expo Go – a vyvíjená aplikace se spustí.

3.4 JSX

JSX (JavaScript XML) je syntaxe, která umožňuje psát komponenty v React Native a Expo způsobem, který kombinuje JavaScript s XML – nebo HTML – podobným kódem. JSX je kombinace JavaScriptu a XML, která umožňuje vývojářům deklarovat uživatelské rozhraní (UI) pomocí syntaxe, která kombinuje sílu JavaScriptu s deklarativním zápisem podobným XML (8). To usnadňuje čitelnost a psaní kódu. JSX umožňuje vytvářet React komponenty jednodušeji. Při vývoji v React Native a Expo může JSX obsahovat speciální komponenty pro mobilní prvky, jako jsou <View>, <Text>, <Image> a další. To znamená, že vývojáři mohou pohodlně pracovat s logikou aplikace a UI v jednom souboru.

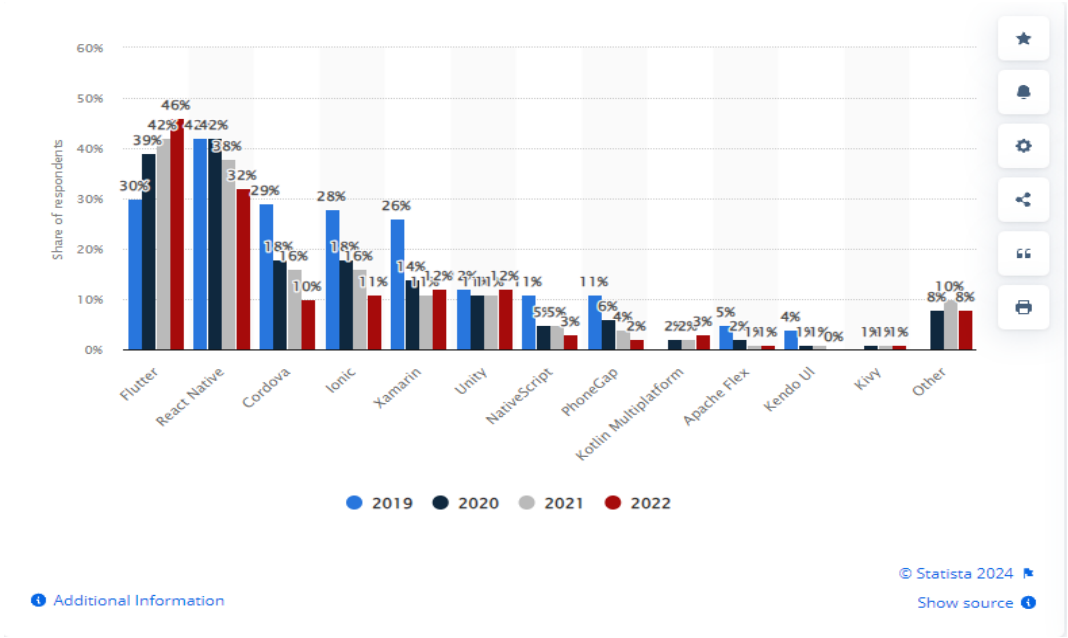
```jsx
export default function TodoList() {
  return (
    <>
      <h1>Hedy Lamarr's Todos</h1>
      <img
        src="https://i.imgur.com/yXOvdOSs.jpg"
        alt="Hedy Lamarr"
        className="photo"
      />
      <ul>
        <li>Invent new traffic lights</li>
        <li>Rehearse a movie scene</li>
        <li>Improve the spectrum technology</li>
      </ul>
    </>
  );
}

```

Obrázek 6 Ukázka použití JSX (Internetový zdroj dle (8))

3.5 Porovnání React Native s ostatními multiplatformními jazyky

Při výběru technologie pro vývoj mobilních aplikací je nezbytné porovnat různé multiplatformní jazyky a frameworky. Multiplatformní jazyky a frameworky mají výhodu, že lze najednou budovat aplikace jak pro iOS, tak pro Android, případně i pro web, a tím se snižují náklady na vývoj mobilních aplikací. V případě, že by se vyvíjela mobilní aplikace pro iOS a Android separátně, tedy pro iOS by se vyvíjela aplikace na zařízení s macOS s využitím XCode a programovacího jazyku Swift, a pro Android na zařízení podporující jazyk Kotlin a Android studio, bylo by potřeba buď dvakrát více času programátora anebo dva separátní programátoři a dvě zařízení. Multiplatformní vývoj zjednodušuje vývoj pro oba typy zařízení a snižuje náklady. Následující obrázek znázorňuje procentuální využití různých frameworku za roky 2019 až 2022 (9).



Obrázek 7 Multiplatformní vývoj dle frameworků (Internetový zdroj dle (9))

Z obrázku je zřejmé, že nejpopulárnější framework mezi respondenty je Flutter od společnosti Google, který každým rokem získává na popularitě. V roce 2019 měl nižší popularitu než React Native, 30% pro Flutter proti 42% pro React Native. Z počátku byl vývoj ve Flutter náročnější, nicméně vývoj se ulehčuje i díky platformě jakou je FlutterFlow. Následuje ho React Native, kterému se popularita snižuje, pravděpodobně i kvůli již zmíněné platformě FlutterFlow (více v kapitole o Flutter). Dále je důležité zmínit framework Xamarin od společnosti Microsoft. Poslední vybraný příklad je NativeScript.

3.5.1 Flutter

Flutter je open-source framework vyvinutý společností Google pro tvorbu mobilních aplikací (10). Co Flutter odlišuje je jeho schopnost vytvářet nádherně vypadající aplikace s vysokým výkonem na různých platformách pomocí jediného kódu. Nativní výkon Flutteru je dosažen díky vlastnímu enginu zvanému Skia, který je používán k vykreslování uživatelského rozhraní. Flutter nabízí funkci "hot reload," což znamená, že vývojáři mohou vidět změny v reálném čase bez nutnosti znovu spouštět celou aplikaci. Widgety jsou stavební bloky Flutter aplikací. Každé rozhraní je sestaveno z widgetů, které jsou hierarchicky strukturovány. Flutter používá jazyk Dart, který kombinuje prvky objektově orientovaných jazyků s některými vlastnostmi funkcionálních jazyků, což jej činí efektivním pro vývoj mobilních aplikací (11). Flutter umožňuje dosáhnout nativního vzhledu na různých platformách a tím umožňuje vytvářet aplikace pro iOS, Android a také pro web a desktop, což mu přidává na přidává flexibilitě a univerzálnosti.

FlutterFlow je platforma postavená na Flutteru, která usnadňuje vývoj mobilních aplikací a webových stránek (12). Dále umožňuje tvorbu aplikací pomocí vizuálního designu a logiky, která minimalizuje potřebu psaní kódu. Tím umožňuje jak nízko-kódovým tak bez-kódovým vývojářům snadno vytvářet aplikace. Podobně jako Flutter i FlutterFlow podporuje "hot reload" a online preview, což znamená rychlý vývoj a okamžité zobrazení změn. FlutterFlow poskytuje bohatou knihovnu připravených šablon a komponent, které usnadňují vývoj. Umožňuje snadnou integraci s externími službami a API pro rozšíření funkcionality aplikací. Nabízí také škálovatelnost od jednoduchých projektů až po složité aplikace. FlutterFlow tedy rozšiřuje možnosti Flutteru tím, že umožňuje tvorbu aplikací bez hlubší znalosti kódu, což je vhodné pro různé typy vývojářů s různými úrovněmi zkušeností. Poslední bod je důležitý z pohledů některých vývojářů, protože pro ně mohl být jazyk Dart komplikovanější na naučení.

3.5.2 Xamarin

Xamarin představuje otevřenou platformu pro vytváření moderních a výkonných aplikací pro iOS, Android a Windows, využívající .NET (13). Jedná se o vrstvu abstrakce, která usnadňuje komunikaci mezi sdíleným kódem a základním kódem pro danou platformu. Xamarin je spuštěn v prostředí s automatickou správou paměti, což zjednodušuje aspekty jako alokace a uvolňování paměti. S pomocí Xamarinu mohou vývojáři sdílet až 90 % svého kódu mezi různými platformami. Tato strategie umožňuje napsat veškerou obchodní logiku v jednom jazyce nebo znovupoužít existující kód aplikace, a přesto dosáhnout nativního výkonu, vzhledu a chování na každé platformě. Aplikace vytvořené pomocí Xamarinu lze vyvíjet na PC nebo Macbooku a kompilovat. Pro otestování vyvíjené aplikace je nutné mít fyzické zařízení, a to Androud Studio pro otestování aplikace na Android zařízené, nebo macOS s XCode pro otestování na zařízeních Apple. Xamarin umožňuje vývojářům používat jazyk C#, což je silný a moderní objektově orientovaný jazyk. Tento jazyk je mezi vývojáři oblíbený a poskytuje mnoho pokročilých funkcí. Aplikace v Xamarinu jsou postaveny pomocí architektury Xamarin.Forms nebo Xamarin.Native. Xamarin.Forms umožňuje sdílení většiny kódu pro UI mezi různými platformami, zatímco Xamarin.Native umožňuje plně nativní UI pro každou platformu. Kód může být sdílen mezi různými platformami. To zahrnuje logiku aplikace, komunikační vrstvy, a mnoho dalšího, co výrazně snižuje čas a úsilí potřebné k vývoji pro každou platformu zvlášť. Xamarin umožňuje dosáhnout nativního vzhledu a chování na každé platformě, což je klíčové pro poskytnutí konzistentního uživatelského zážitku. Je plně integrován s platformou Microsoft, což usnadňuje integraci s Azure, Visual Studio a dalšími nástroji a službami. Xamarin umožňuje snadné testování aplikací na různých zařízeních a

platformách. Funkce jako Xamarin Test Cloud umožňují testování na širokém spektru zařízení v cloudu.

3.5.3 NativeScript

NativeScript je open-source framework pro vývoj mobilních aplikací, který umožňuje vývojářům psát aplikace v JavaScriptu nebo TypeScriptu a následně je kompilovat do nativního kódu pro iOS a Android (14). Využívá mnoho existujících knihoven a nástrojů v ekosystému JavaScriptu. NativeScript umožňuje vytvářet nativní rozhraní jak na platformě iOS, tak i na platformě Android. To znamená, že výsledné aplikace mají nativní vzhled a chování, což přispívá ke konzistentnímu uživatelskému zážitku. S NativeScriptem je možné sdílet značnou část kódu mezi iOS a Android. Logiku aplikace, včetně většiny obchodní logiky a komunikační vrstvy, lze sdílet a tím urychlit vývoj pro obě platformy. NativeScript podporuje mnoho populárních frameworků a knihoven včetně Angular, Vue.js a React. Vývojářům je tím umožněno vybrat si framework, který jim nejvíce vyhovuje. JavaScriptovým frameworkům se často říká "flavor". NativeScript je rozšiřitelný pomocí pluginů, což znamená, že lze integrovat různé nástroje a funkce do aplikace podle potřeby. Existuje bohatý ekosystém pluginů, který je k dispozici pro vývojáře. Díky živému náhledu (live preview) a rychlému překladu může vývojář vidět změny v reálném čase, což urychluje vývojový cyklus. NativeScript poskytuje přístup k nativním API pro obě platformy, a to umožňuje vývojářům využívat specifické funkce každého zařízení.

3.5.4 Výhody a nevýhody React Native ve srovnání s ostatními možnostmi

Nyní následuje shrnutí klíčových výhody a nevýhody React Native ve srovnání s ostatními multiplatformními jazyky. Budou zohledněny faktory jako produktivita vývoje, rychlost a jednoduchost učení, což by mělo poskytnout užitečný přehled pro rozhodování o vhodné technologii pro naši aplikaci pro výcvik psů.

Díky srovnání bude možné lépe porozumět kontextu, ve kterém se React Native nachází v rámci nabídky multiplatformních možností pro mobilní vývoj (15).

React Native:

- UI: Je dostatečně flexibilní na pokrytí většiny webových případů. Neobsahuje všechny protějšky nativních komponent, nicméně tyto jsou většinou nahrazeny komunitními plugin[1]

[1] Neboli zásuvný modu

- Výkonnost: Využívá architektury takzvaných můstků pro všechny pohledy, u kterých vznikají problémy při implementaci animací, a to může snižovat výkonnost
- Přizpůsobení: Využití vlastních přizpůsobení je možné, nicméně je limitované a dokumentace je zanedbaná, skoro až neexistující
- Stabilita: Některé release mohou způsobit problémy a způsobit nefunkčnost aplikace, zejména při změnách souvisejících s nativními platformami. Někdy vývojáři nemohou řešit nastalé problémy okamžitě, a proto někdy pomáhá komunita s opravou chyb pomocí knihoven třetích stran
- Kompatibilita s webem: Lze sdílet logickou část aplikace s webovou aplikací a je možné vytvářet i desktopové aplikace

Flutter:

- UI: Využívá vlastní vykreslovací engine, který dovoluje vytváření UI různých složitostí, a v některých případech může být tento engine rychlejší než nativní
- Výkonnost: Je stejná jako nativní aplikace Android a iOS. Flutter aplikace mohou být i rychlejší než nativní Kotlin nebo Java aplikace a to díky architektuře, kterou je možné rozložit do tří komponent: embedder pro konkrétní platformu, engine pro vykreslování komponent a framework vrstva, se kterou interagují vývojáři
- Přizpůsobení: UI je extrémně flexibilní a dovoluje vytváření UI stejně jako nativní platformy
- Stabilita: Má stabilní release, které nezpůsobí nefunkčnost vyvíjené aplikace
- Kompatibilita s Webem: Obsahuje plnou verzi pro vytváření webových aplikací i desktopových aplikací s omezením

NativeScript:

- UI: Podobné jako pro React Native, staví na nativních pohledech. Má podobné problémy jako React Native a dovoluje využít Angular 2+, Vue.js a jejich framework pro vytváření UI (16)
- Výkonnost: Jelikož vše funguje na JavaScriptových vláknech, má NativeScript omezenou výkonnost
- Přizpůsobení: Není možné vytvářet nová rozložení a využívat vlastní přizpůsobení
- Stabilita: Má problémy se správou paměti (memory leaks), které mohou vést méně zkušené vývojáře k problémům, které nemusí umět samostatně vyřešit

- Kompatibilita s Webem: Stejně jako u React Native lze sdílet pouze logickou část aplikace

Xamarin:

- UI: Umožňuje používat kompletně nativní prvky (17)
- Výkonnost: Výkonnostně je na tom Xamarin podobně jako React Native, a tedy zaostává za Flutterem. Po kompilaci jsou aplikace stejně rychlé jako nativně vyvíjené aplikace
- Přizpůsobení: Má podobné možnosti přizpůsobení jako Flutter. Má předinstalované elementy rozložení, které značí volnost v crossplatform vývoji mobilních aplikací a jejich kustomizaci
- Stabilita: Využívá výhod nativních frameworků a kompilovaných binárních souborů, čím je poskytován až nativní výkon (18)
- Kompatibilita s Webem: Xamarin se zaměřuje na vývoj mobilních a desktopových crossplatformových aplikací a základní podpora pro web aplikace není

Z hodnocení se může zdát, že by bylo nejlepší použít Flutter místo React Native. Nicméně u Flutteru je potřeba zmínit i používaný jazyk – a to je jazyk Dart (19). Bez předchozí zkušenosti s tímto jazykem by vývoj aplikace byl složitější (20). U React Native jsou potřebnými prerekvizitami znalost HTML, CSS a JavaScriptu, které se využívají při tvorbě webových stránek.

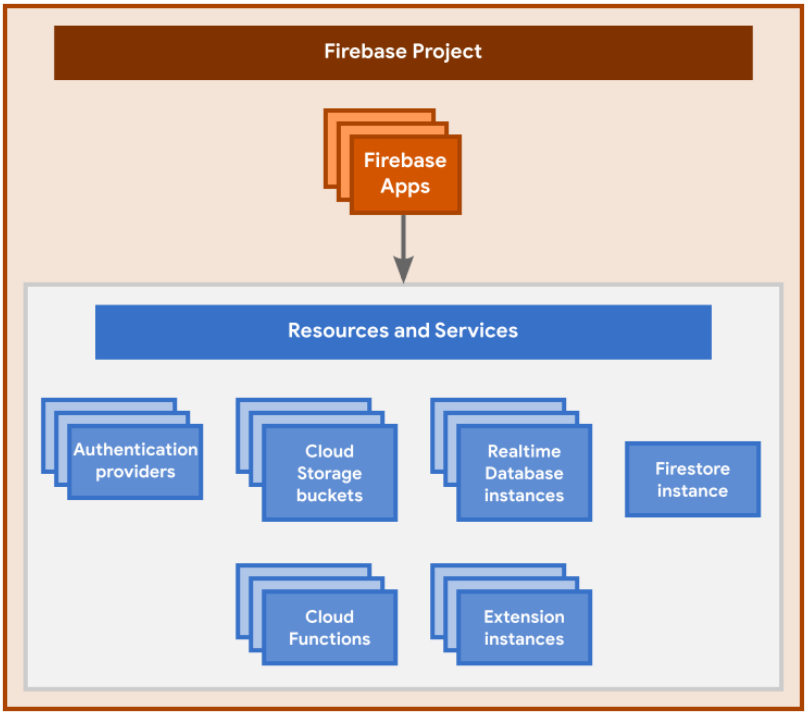
Zkušenost s Reactem by ještě více usnadnila začátek vývoje s React Native, ale předchozí zmíněné prerekvizity jsou dostačující (21).

3.6 Firebase

Firebase je komplexní platforma od společnosti Google, která poskytuje širokou škálu služeb pro vývoj mobilních a webových aplikací. Jako Backend as a Service (BaaS) nabízí kompletní backendové řešení pro vývojáře bez potřeby správy vlastního serveru (22). Realtime Database je NoSQL databáze v reálném čase, která umožňuje ukládat a synchronizovat data mezi různými klienty v reálném čase. To znamená odesílání aktualizací v reálném čase všem připojeným klientům. Firebase Authentication poskytuje nástroje pro ověřování identity uživatelů. Ověřování probíhá pomocí e-mailu, hesla, telefonních čísel, sociálních médií a dalších poskytovatelů, jako Facebook, Apple, Microsoft, Google a další. Firestore je NoSQL databáze, která nabízí robustní možnosti dotazování a indexace a podporuje dotazování v

reálném čase. Cloud Functions jsou serverless funkce, které mohou být spouštěny událostmi v rámci Firebase nebo v rámci jiných služeb Google Cloud. Firebase Storage je úložiště objektů pro ukládání a načítání souborů jako jsou obrázky nebo videa. Lze tak nahrávat a stahovat soubory přímo z aplikace.

Pro pochopení hierarchie projektu Firebase poslouží následující diagram, který znázorňuje klíčové vztahy.



Obrázek 8 Rozložení Firebase projektu (Internetový zdroj dle (22))

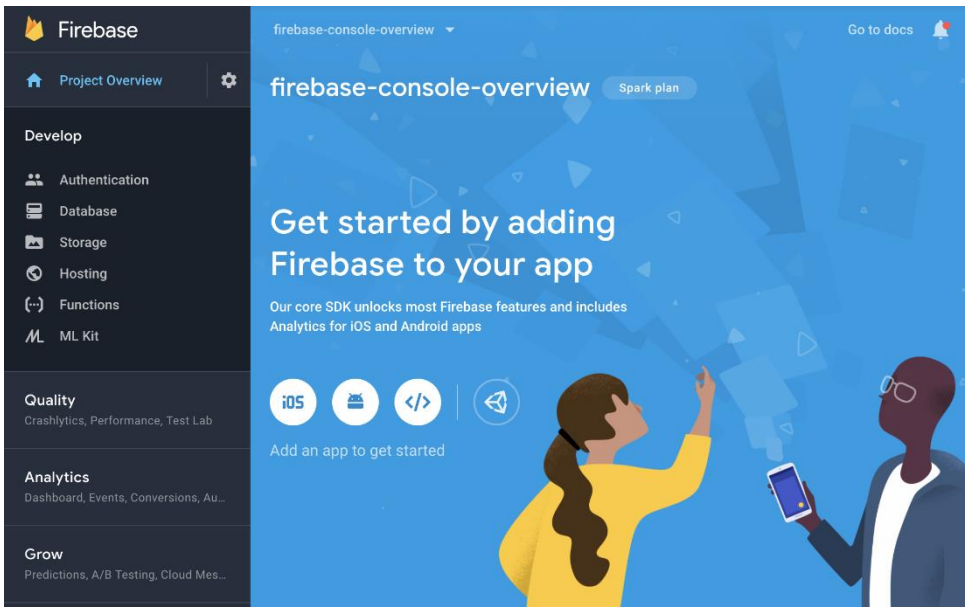
Mezi klíčové vztahy patří:

- Projekt Firebase slouží jako kontejner pro všechny aplikace a pro všechny použité zdroje a služby pro konkrétní projekt
- Každý projekt ve Firebase může mít zaregistrovanou jednu nebo více aplikací – verze pro různé platformy (iOS, Android, WEB) - nebo také různé verze pro jednu platformu, například pro iOS bezplatnou a placenou verzi aplikace
- Všechny aplikace zaregistrované ve stejném Firebase projektu sdílejí přístup ke všem zdrojům, např. k databázi nebo k storage a ke všem službám jako je autentizace, cloudové funkce apod.

- Je možné využít Google Analytics pro všechny aplikace Firebase registrované do stejného Firebase projektu

S projektem ve Firebase lze interagovat pomocí několika různých nástrojů a rozhraní. První možností je konzole Firebase.

Konzole nabízí nejbohatší prostředí pro správu produktů, aplikací a nastavení na úrovni projektu. Na levém panelu konzoly jsou umístěny produkty Firebase a jsou uspořádané dle kategorií nejvyšší úrovně. Uprostřed konzole jsou zobrazena tlačítka, která spouštějí různé pracovní postupy pro registraci a nastavení různých typů aplikací.



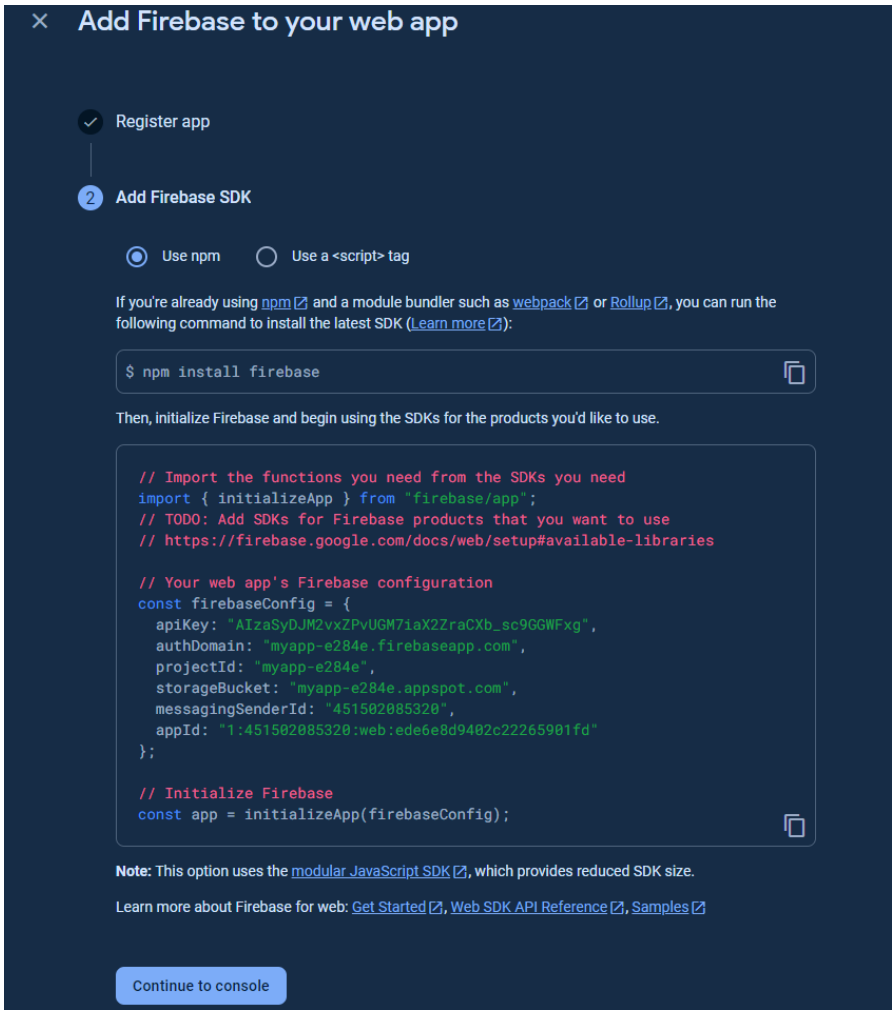
Obrázek 9 Přehled Firebase konzole (Internetový zdroj dle (22))

Dalším nástrojem je Firebase CLI. Jedná se o nástroj příkazového řádku pro konfiguraci a správu produktů nabízených Firebase, jako je například Firebase Hosting, Cloud Functions pro Firebase a Firebase Extensions. Po nainstalování nástroje příkazového řádku je k dispozici globální příkaz `firebase`. Pomocí CLI se propojí adresář aplikace s projektem ve Firebase.

Posledním nástrojem je Firebase Management REST API. Toto rozhraní umožňuje programově spravovat projekt Firebase. Umožňuje například programově zaregistrovat aplikaci do projektu nebo vypsat aplikace, které jsou již zaregistrované pro různé platformy.

Pro inicializaci Firebase ve vyvíjené aplikaci je nutné poskytnou konfiguraci projektu Firebase aplikace (23). Konfiguraci lze získat kdykoliv, nicméně v průběhu vytváření projektu v konzoli Firebase ji lze získat. Již v průběhu vytváření aplikace je konfigurace k dispozici, nebo

kdykoliv ji lze kdykoliv nalézt v nastavení projektu. Dále je nutné nainstalovat Firebase do projektu pomocí příkazu *npm install firebase*.



Obrázek 10 Získání konfigurační soubor Firebase projektu

Na obrázku uprostřed je vidět konfiguraci pro nově vytvořenou aplikaci. Dle dokumentace se nedoporučuje jednotlivé hodnoty přepisovat, protože by v průběhu přepisování mohla nastat chyba a aplikace s chybně inicializovanými hodnotami by mohla způsobovat vážné problémy pro uživatele aplikace. Konfiguraci následně stačí uložit do souboru v adresáři aplikace, například firebaseConfig.js, a naimportovat do souboru, ve kterém se využívají funkce poskytující Firebase.

3.7 Analýza a výběr povelů a cviků pro pejsky

K výchově psa s důrazem na adaptaci na společné soužití, například k výcviku psa pro venčení nebo zvládání klidné doby doma, je vhodné přistoupit téměř okamžitě po jeho příchodu do nového domova, tj. již od osmého týdne věku (24). Výcvik základních povelů může začít v této rané fázi, přičemž u náročnějších povelů a psích sportů hraje klíčovou roli mnoho faktorů. Například u sportů jako jsou bull sporty nebo canicross je nezbytné začít s výcvikem až poté, co pes dospěje a jeho fyzický vývoj to umožní. Výcvik psa představuje náročnou disciplínu, protože každý jedinec může na stejný povel a typ výcviku reagovat unikátně v závislosti na své povaze. Existuje mnoho různých přístupů k výcviku psa, avšak při zapojení do hry s pozitivním odměňováním (v podobě pamlsků, oblíbených hraček a slovních pochval) existuje větší pravděpodobnost, že se pes bude rychleji učit a ochotněji předvádět naučené povely (25). Majitel psa může přistoupit k výcviku sám s využitím různých publikací, video návodů apod., nebo může zvolit spolupráci se soukromým trenérem či s trenérem v rámci kynologických klubů a sportovišť. Mezi klíčové zásady úspěšného výcviku patří volba vhodného času tréninku s ohledem věk psa (26). Čím je pes mladší, tím kratší dobu udrží pozornost. Je také důležité najít rovnováhu, aby výcvik nezabral během dne příliš mnoho času. Integrace cvičení do krátkých sérií v rámci 5-10 minut během běžné procházky může usnadnit efektivní trénink. Další klíčovou zásadou je postupovat pomalu, začínat s jedním až dvěma novými cviky najednou a až po jejich úspěšném zvládnutí přidávat další. Opakování již naučených povelů je také nezbytné pro pevné zakotvení dovedností. Důraz je kladen na konzistenci, kdy pes dostává stejné povelové signály a je pochválen za konzistentní výkon, aby nedošlo k zmatení. Vybrané základní povely:

- Sedni
- Lehni
- K noze
- Ke mně
- Zůstaň

Do kategorie pokročilých cviků budou vybrány různé cviky z celé škály psích spotů, resp. cviků. Jelikož existuje mnoho různých psích sportů, pro aplikaci bude vybrána určitá skupina sportů, které by mohla většina uživatelů ocenit. Mezi sporty budou zařazeny například:

- Agility – parkúr pro psy
- Dogdancing – Divize C: freestyle
- Canicross
- Obedience
- Sportovní kynologie
- Coursing – terénní dostih
- Bull sport – High jump

4 Vlastní práce

Kapitola bude zaměřena na jednotlivé kroky ve vývoji softwaru. Vývoj softwaru zahrnuje analýzu požadavků na vyvíjený software, dále analýza použitých technologií, uživatelského rozhraní a dalších. V dalším kroku bude popsán návrh řešení, databáze, jednotlivé kroky v navigaci atd. Na konec budou představeny nastavení konfigurace pro jednotlivá prostředí a samotnou implementaci. Poslední část se zaměří na obtížné kroky v průběhu implementace a na vzniklé problémy.

4.1 Analýza požadavků

Analýza požadavků je proces identifikace, specifikace a správy požadavků na vyvíjený systém, v tomto případě aplikace. Cílem analýzy je zajištění jasných, konzistentních a přijatelných požadavků na aplikaci, které budou zároveň dostatečně přesné pro jejich implementaci.

Aplikace by měla fungovat na mobilních platformách iOS a Android a měla by sloužit zejména jako mobilní aplikace pro majitele psů, kteří by rádi zaznamenávali průběh a jednotlivé kroky při výcviku jejich čtyřnohého miláčka. Uživatel bude mít s pomocí aplikace přehled o všech svých psech ve výcviku a o jejich aktuální úrovni. U každého psa bude k dispozici základní sada cviků, např. povely typu “Sedni!”, “Lehni!”, “K noze!” apod. Dále bude k dispozici sada náročnějších cviků, např. psí agility cvičení - “Parkúr pro psy”, “Dog dancing”, psí poslušnost “Obedience” a další. V případě, že by uživatel ve výběru aplikace nenašel požadovaný psí sport, aplikace nabízí i možnost, aby si uživatel vytvořil vlastní cvik. Při vytváření vlastního cviku je potřeba zadat název cviku a jeho popis v potřebné délce. Volitelná je možnost nastavit si maximální počet bodů u vytvořeného cvičení. Pro každý cvik by mělo být možné přidávat a odebírat body pro konkrétní cvik v závislosti na zlepšení nebo zhoršení cvičeného psa. K úpravě bodů bude sloužit detail cviku, kde bude k dispozici popis cviku, a i možnost měnit počet bodů. Všechny cviky se pak promítají do celkového hodnocení psa. Pro lepší sledování průběhu výcviku bude k dispozici tréninkový diář. To dovolí uživateli si zaznamenávat jednotlivé tréninkové relace, kde si zaznamená jednotlivé cviky, které trénoval se svým pejskem, krátký popis celého tréninku a čas datum tréninku. V aplikaci bude k dispozici i kalendář pro zaznamenávaní různých událostí, které se týkají konkrétního psa. U události je možné nastavit název a popis události, místo konání události a také datum a čas. V kalendáři se také bude zobrazovat datum narození psa. V další části aplikace budou i také základní zdravotní tipy a rady, ohledně toho, co by se mohlo psovi stát, a to nejen při výcviku, ale i

mimo něj. Například při vytrvalostním nebo intenzivním cviku by mohlo dojít k přehřátí, zvracení a podobným potížím u psa a aplikace navede uživatele, jak poskytnout správnou první pomoc v takových případech.

Na základě těchto funkčních a jednoho nefunkčního požadavku, respektive dvou nefunkčních, kde druhý nefunkční požadavek je požadavek na bezpečnost, jsou na aplikaci v rámci minimálního životaschopného produktu požadované následující funckionality:

- Zaregistrování nového uživatele pomocí emailu a hesla
- Přihlášení již existujícího uživatele pomocí emailu a heslo
- Přihlášení uživatele pomocí služby Google Sign In
- Vytvoření nového psa v aplikaci (s možností úprav)
- Využití a bodování cvičení dostupných spolu s aplikací
- Vytvoření vlastního cviku
- Vytváření událostí v kalendáři
- Gamifikace ve formě získávaných ocenění
- Vytváření tréninkových relací
- Dostupnost na obou mobilních platformách – iOS a Android
- Bezpečnost dat a přenosu dat

4.2 Analýza technologií

Software lze vyvíjet pomocí různých technologií v různých vývojových prostředích a v různých programovacích jazycích, databázích a nástrojích. Zaměříme se na výběr jednotlivých technologií pro vývoj zadané aplikace. Výběr těchto technologií bude vycházet z předchozí analýzy a stejně tak i z teoretické části, kde jsme se jednotlivým možným frameworkům a technologiím již věnovali. Vybrané technologie, frameworky a další pak budou použity k samotné implementaci aplikace.

Jedním z požadavků na aplikaci je multiplatformní přístup. Aplikace tedy má být dostupná na zařízeních jak s iOS, tak i na zařízeních s Android platformou. K vývoji multiplatformní aplikace lze využít mnoho různých frameworků nebo neefektivně vytvářet dvě aplikace ve specifickém nativním přístupu. Pro iOS by to byly buď Swift, anebo Objective-C. Pro Android platformu by to byly Kotlin nebo Java. React Native umožňuje vývoj na jakémkoliv operačním systému, např. na Windows i na macOS, je lehký na naučení a použití, a také sdílí podobnost s často používanými jazyky. V React Native lze použít jazyky JavaScript a TypeScript.

TypeScript je JavaScript s datovými typy. Použit bude jazyk JavaScript. S použitím React Native jako platformy pro vývoj aplikace je spojena otázka ohledně použití React Native CLI nebo třetí strany, například platformy Expo. Při použití první možnosti, tedy CLI, by byl vývoj značně komplikovanější a zpravidla je použit pro komplexnější aplikace. Na druhou stranu použití platformy Expo usnadňuje vývoj aplikace, a i její brzké testování, díky mobilní aplikace Expo Go, ve které lze aplikaci testovat okamžitě po naskenování QR kódu, které Expo vygeneruje v terminálu.

V dalším kroku bude potřeba databáze, do které se budou ukládat data uživatelů, jednotlivé cviky, události z kalendáře a další data potřebná pro chod aplikace. Využita bude NoSQL databáze poskytována službou Firebase od firmy Google. Tato databáze poskytuje synchronizaci dat v reálném čase např. při registraci nebo při vytváření událostí. Autentizace, tedy přihlášení a registrace bude taky využita služba Firebase, která poskytuje i tyto služby.

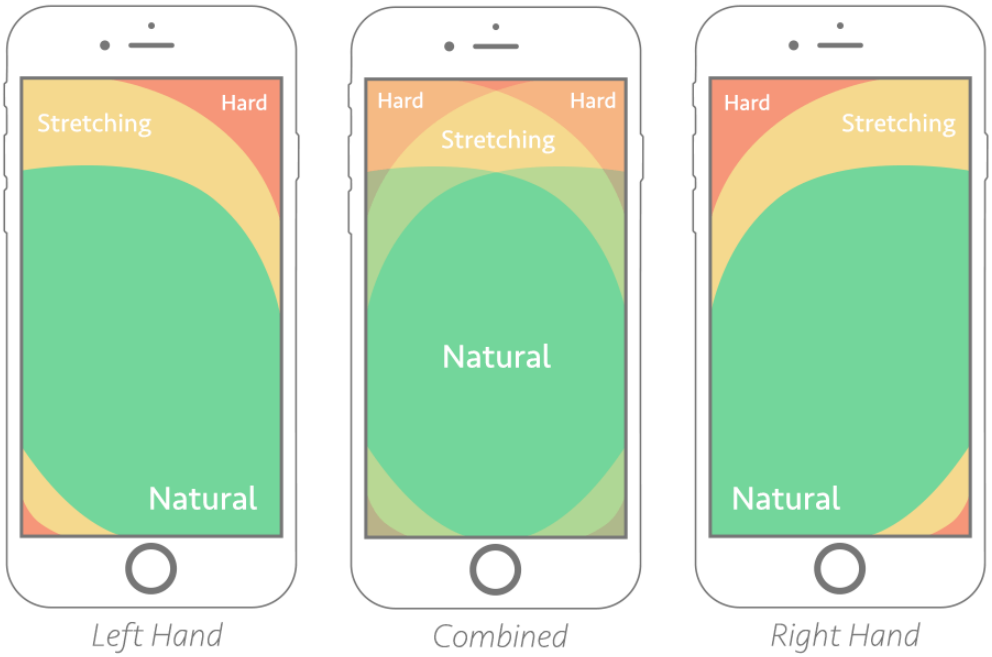
Na konec je potřeba zvolit vývojové prostředí, ve kterém bude aplikace vyvíjena. K dispozici je mnoho různých prostředí, například Visual Studio Code, Sublime Text, Atom, Nuclide a další (27). Z dostupných je nejlepší možností Visual Studio Code (VS Code) anebo Sublime Text. Konečnou volbou je VS Code zejména kvůli jeho přehlednosti a širokému spektru dostupných plug-inů jako například integrace Github pro verzování, zálohování a sdílení kódu. VS Code spolu s Githubem bude využit pro implementaci na Windows i na macOS, načež implementace bude probíhat na obou systémech, aby bylo možné otestovat funkčnost jak na iOS, tak na Android zařízeních.

Analýza přinesla zjištění, které technologie budou potřeba pro implementaci, sdílení a zálohování kódu a testování aplikace. Shrnutí potřebných technologií:

- React Native JS
- Expo
- Firebase – databáze a autentizace
- Visual Studio Code
- Github

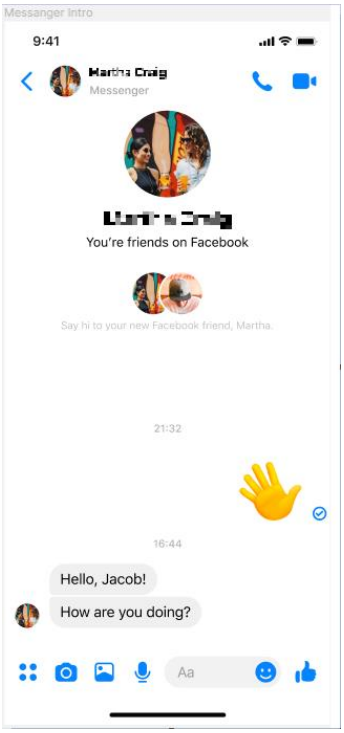
4.3 Analýza uživatelského prostředí

Analýza uživatelského rozhraní (UI) pro mobilní aplikaci je klíčovým prvkem vývoje. Uživatelské rozhraní hraje zásadní roli v tom, jak uživatelé vnímají a interagují s aplikací. Pro dobrou interakci je důležité mít ovládací prvky doslova na dosah prstu. V tomto pomůže definování zón dosahu prstu pro v případě, že uživatel bude držet mobil jednou rukou (28). Tyto zóny jsou znázorněny na následujícím obrázku.



Obrázek 11 Zóny dosahu prstů (Internetový zdroj dle (28))

Obrázek znázorňuje dosahy prstů pro jednotlivé ruce na mobilním telefonu, a i případ, kdy je mobil držen oběma rukama. Při pohledu na zóny dosahu pro pravou ruku z obrázku vyplývá, že by ovládací prvky měly být umístěny, pokud možno zejména v zelené části, případně ve žluté, pokud to rozložení obrazovky nedovoluje. Umisťování ovládacích prvků by se mělo vyvarovat červené části obrazovky, pro pravou ruku to je levý horní roh. V této části je však zpravidla tlačítko pro návrat na předchozí obrazovku. Viz ukázka z aplikace Facebook Messenger (29).



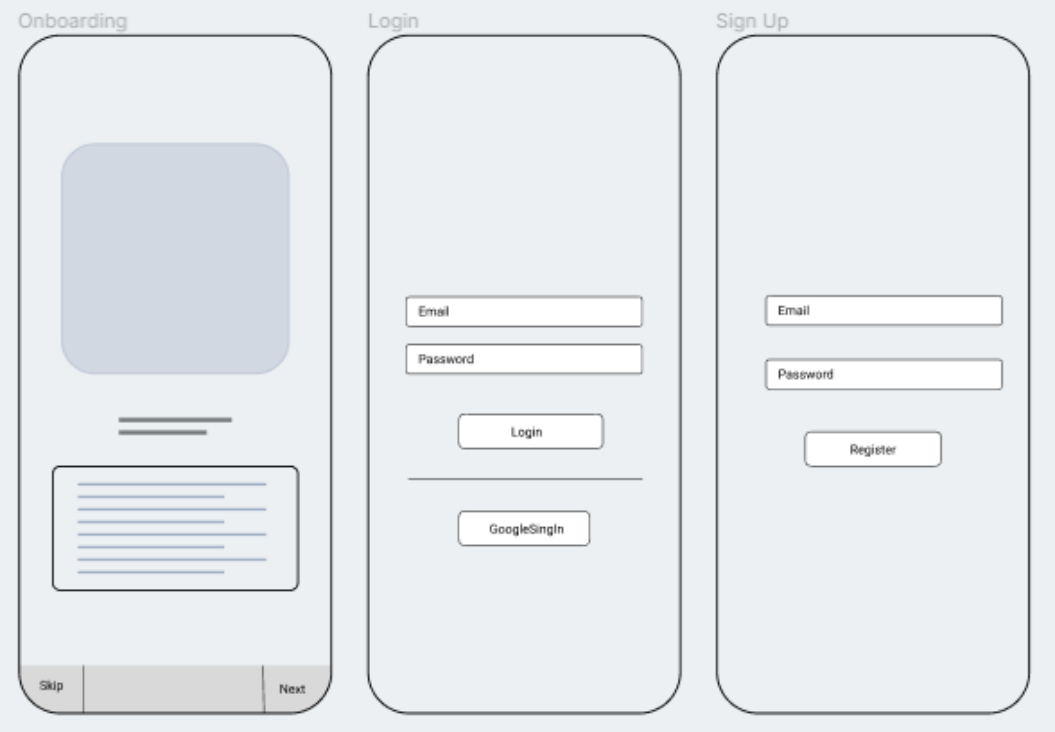
Obrázek 12 Ukázka z aplikace Messenger (Internetový zdroj dle (29))

Uvedený nedostatek má na různých platformách různé způsoby řešení. Na iOS zařízeních je vyřešen pomocí takzvaného swajpnutí (swipe) - přejetím prstu od levého okraje obrazovky k pravému. Popsaným pohybem se lze vrátit na předchozí obrazovku i v případě tlačítka umístěného mimo dosah prstu. U Android zařízení je k dispozici malé menu vespod obrazovky, kde je tlačítko pro akci zpět vždy přítomné.

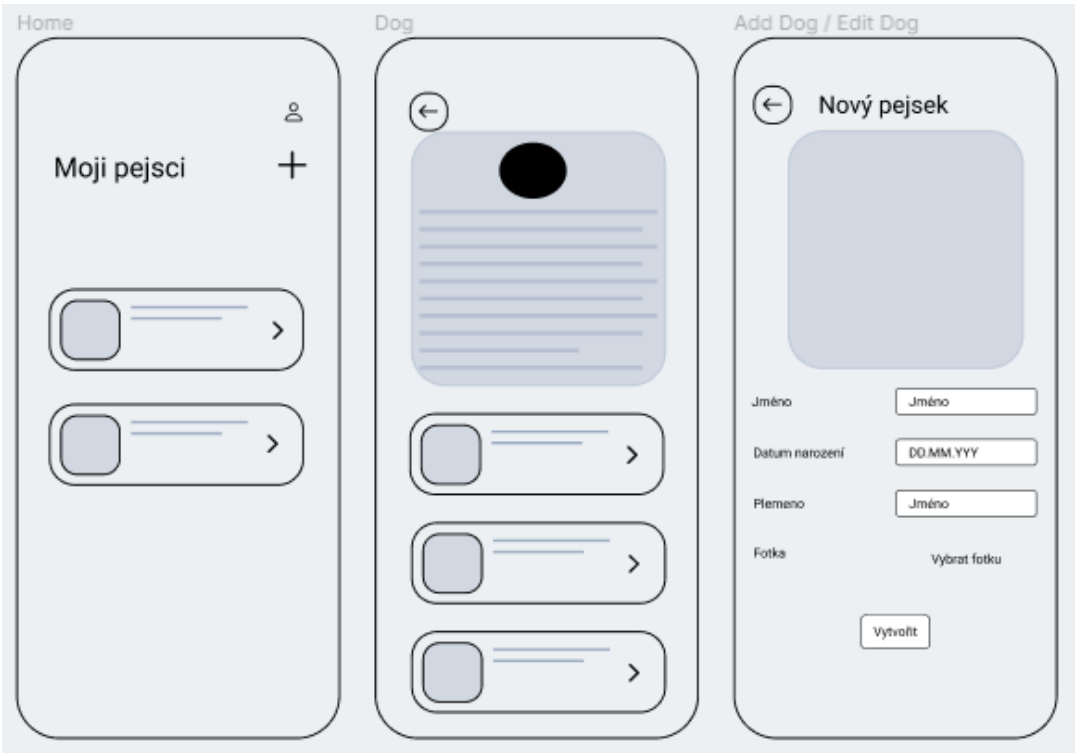
4.3.1 Wireframe

Návrh vizuální stránky pomocí wireframu je důležitý z hlediska představy o podobě výsledné aplikace a zároveň usnadňuje vývoj grafické části aplikace. Wireframy poskytují první pohled na aplikaci, rozložení jejich prvků a náhled pro zákazníka, testera a vývojáře.

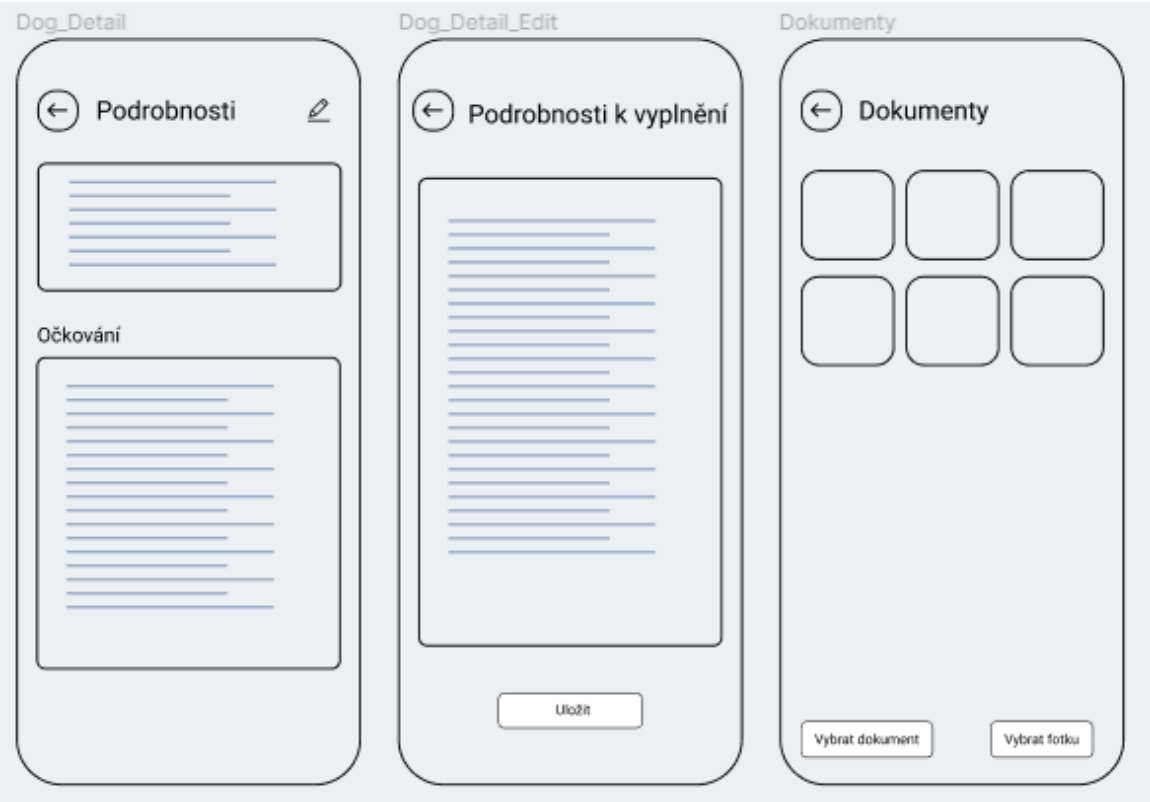
Tvorba wireframů je možná v různých online nástrojích. Nejčastěji se využívá nástroj Figma. Jedná se o intuitivní nástroj, který a umožňuje lehkou tvorbu jednotlivých obrazovek pomocí šablon a prvků pro návrh uživatelského rozhraní (30). Navrhnuty pro aplikaci budou všechny obrazovky – od přihlašovací a registrační obrazovky, domovské obrazovky a obrazovky psa s jednotlivými detaily, až po obrazovku přehledu cviků a detail cviku.



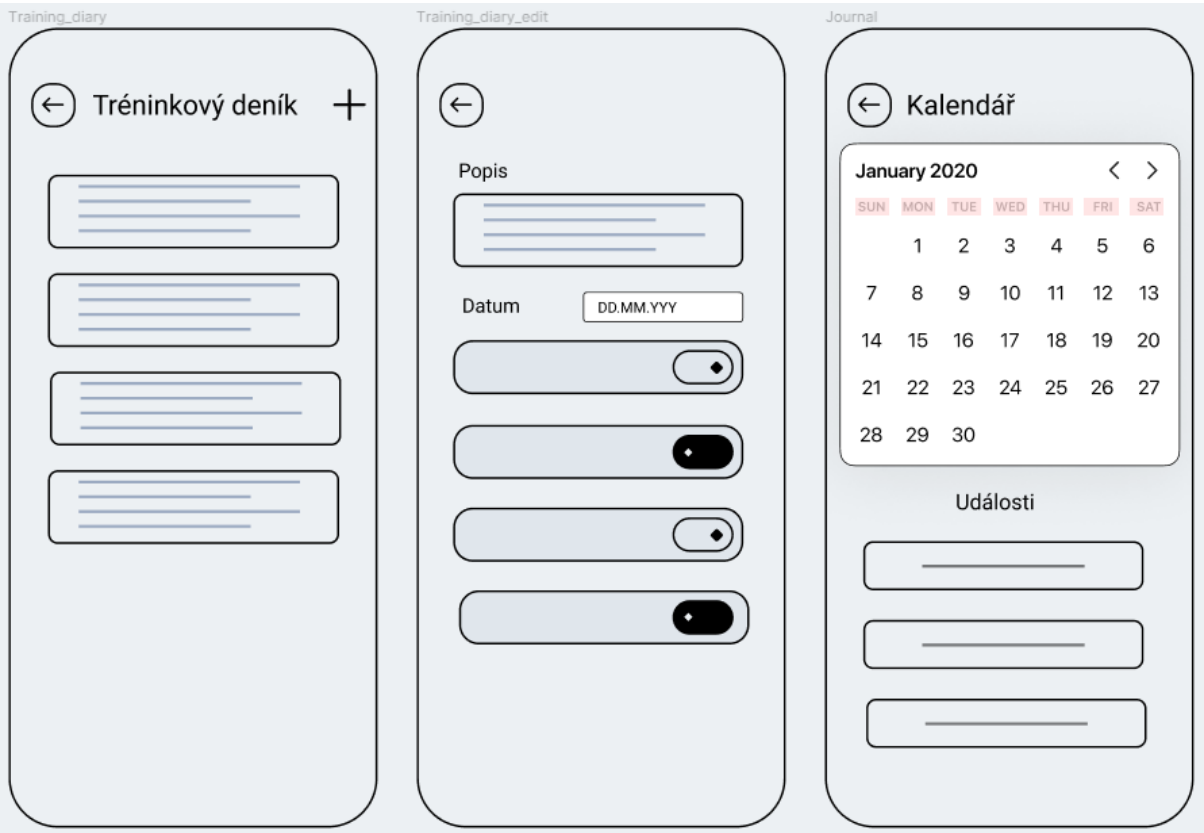
Obrázek 13 Návrh obrazovek z leva: Onboarding, Přihlašovací obrazovka, Registrační obrazovka



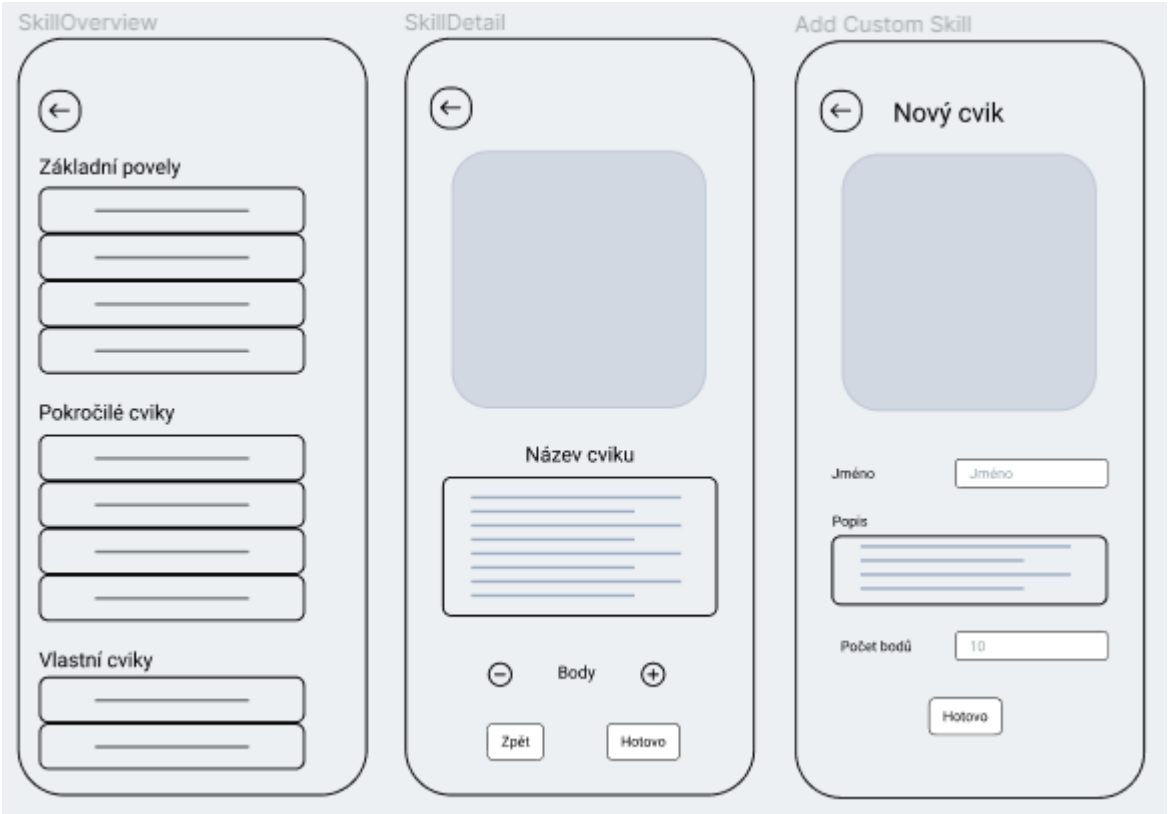
Obrázek 14 Návrh obrazovek, z leva: Domovská obrazovka, obrazovka Pejska, Vytvoření pejska



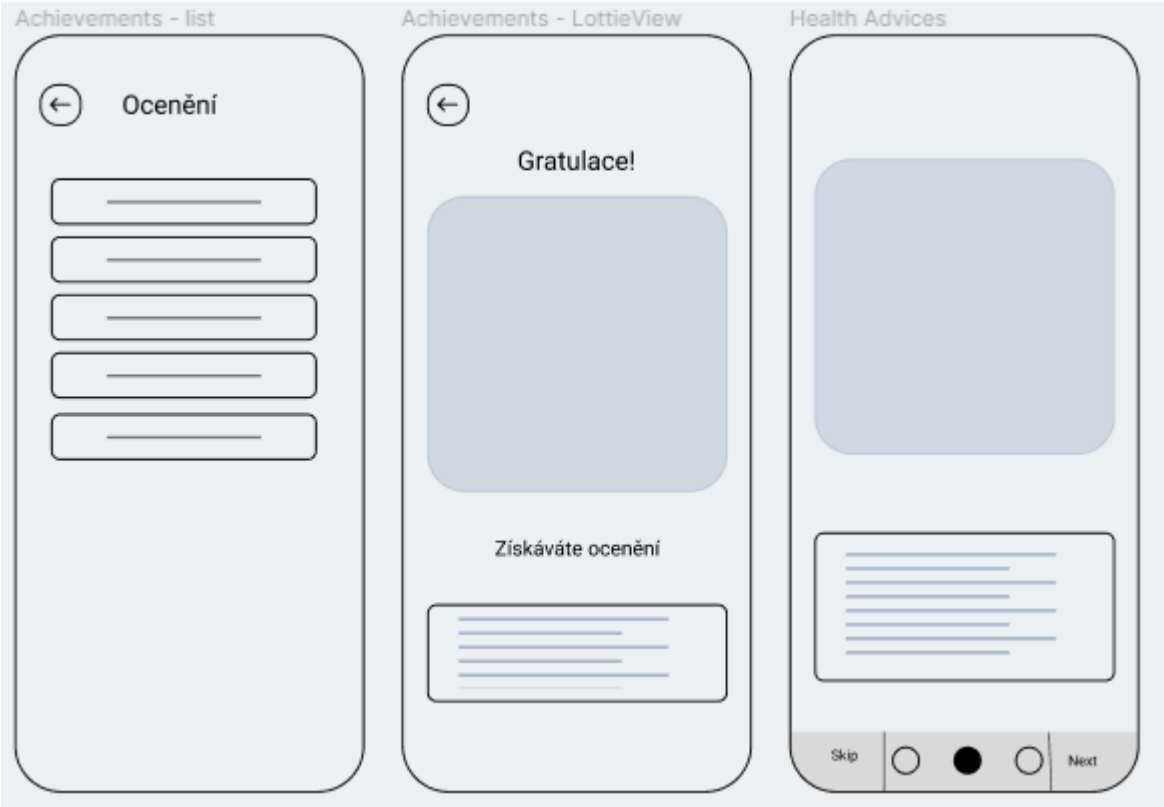
Obrázek 15 Návrh obrazovek, z leva: Podrobnosti, Úprava podrobností, Dokumenty



Obrázek 16 Návrh obrazovek, z leva: Tréninkový deník, Vytvoření tréninkového deníku, Kalendář



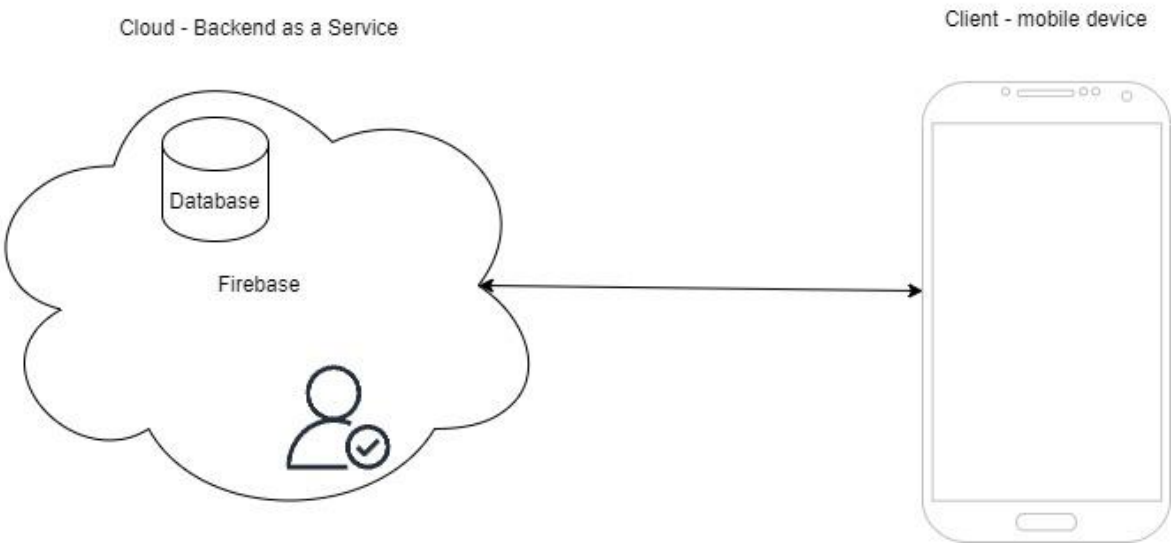
Obrázek 17 Návrh obrazovek, z leva: Přehled cviků, Detail cviku, Vytvoření nového cviku



Obrázek 18 Návrh obrazovek, z leva: Přehled ocenění, Detail ocenění, Zdravotní rady

4.4 Návrh řešení

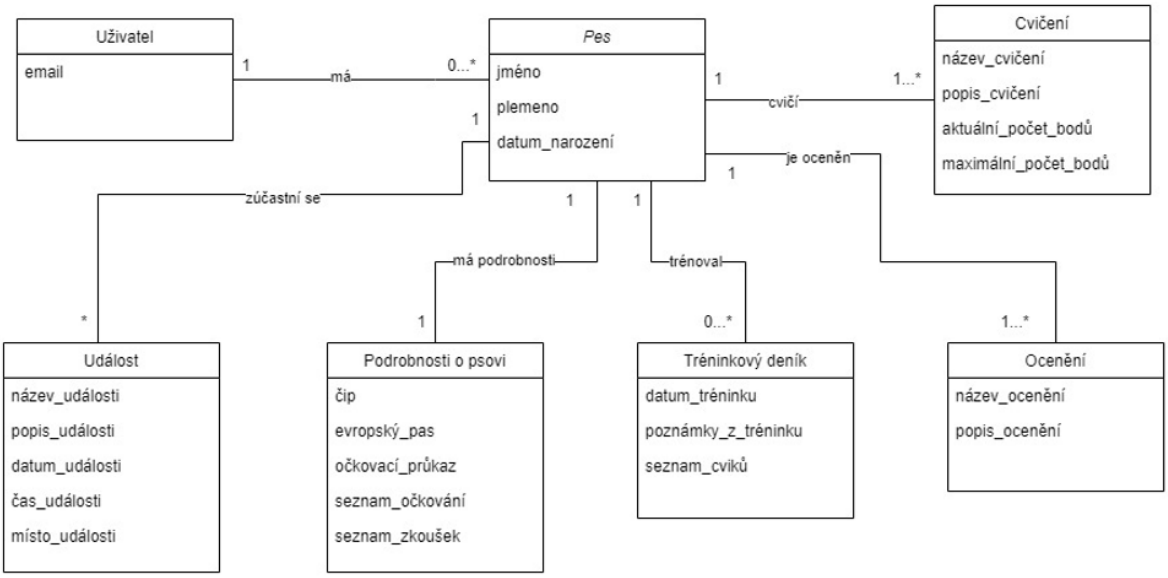
Následující část bude zaměřena na návrh jednotlivých částí aplikace, např. databáze, propojení jednotlivých obrazovek a high-level modelu. První částí bude návrh high-level modelu neboli abstraktní reprezentace systému nebo jeho části na vyšší úrovni abstrakce. Tato abstrakce může být vytvořena za účelem snazšího porozumění, návrhu a implementace software.



Obrázek 19 Návrh high level diagramu aplikace

Na obrázku je vidět návrh high-level diagramu řešení pro vyvíjenou aplikaci. Diagram je složen ze dvou částí. První je cloudová služba Firebase, která poskytne backend pro aplikaci ve formě databáze a autentizace uživatelů.

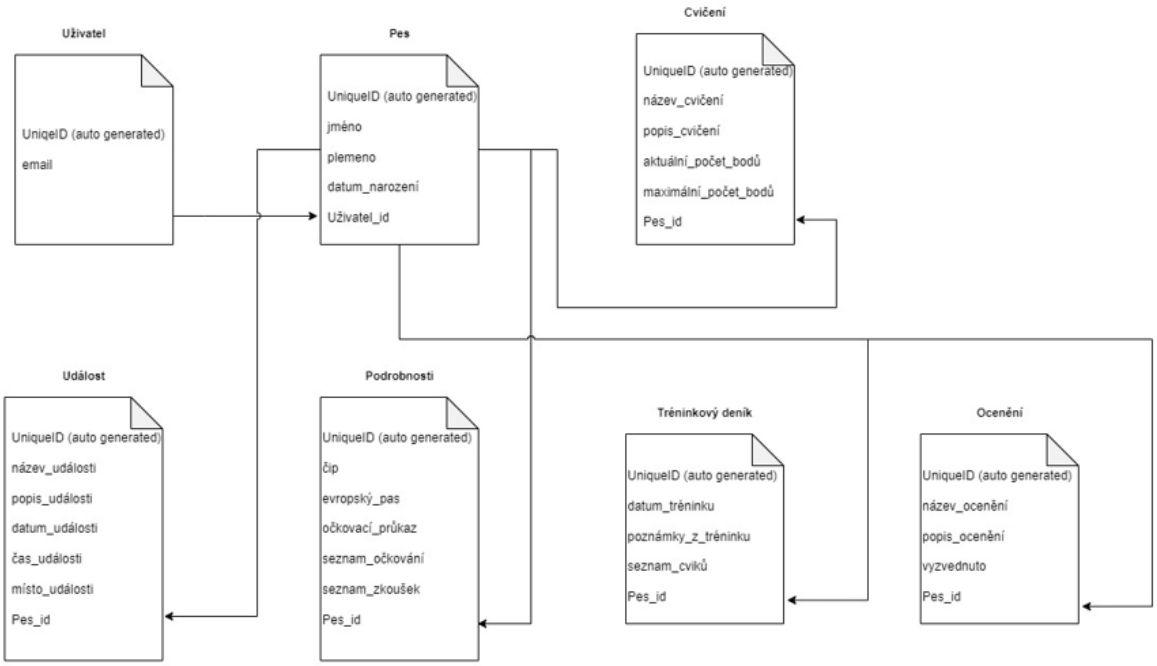
V předchozí části je zmíněna databáze, kterou lze získat z objektového návrhu, který pak bude přetvořen na návrh databáze. Jedním z hlavních objektů potřebných pro aplikaci bude objekt psa, který bude obsahovat jméno, datum narození, plemeno. Navíc bude i objekt pro detaily o pejskovi jako například veškerá možná očkování. Uživatel bude dalším důležitým objektem. Dále je potřeba definovat objekt pro události, tedy název události, místo jejího konání, popis a datum a čas konání události. Dalším nezbytným objektem je cvičení. Objekt „cvičení“ sestává z názvu cviku, jeho popisu, určení maximálního možného počtu bodů za cvik a také aktuálního počtu bodů za cvik. I tréninkový diář má vlastní tabulku, ve které se zaznamenává datum tréninku, krátký popis daného tréninku, např. ve formě souhrnu a seznam cviků a povelů, které uživatel cvičil s pejskem.



Obrázek 20 Objektový model – diagram tříd

Objektový model lze lehce převést na model databázový přidáním databázových prvků – cizí klíče, identifikátory objektů apod. V tomto případě je databáze typu NoSQL. Cizí klíče se nebudou používat, ale odkazy na jiné objekty budou provedeny pomocí proměnné, která v sobě bude mít hodnotu objektu, na který se má odkazovat. Firebase při založení nového dokumentu – obdoba řádku v relační databázi – dává na výběr vytvoření automaticky generovaného identifikátoru/klíče ve formě stringu. Vývojář může vytvářet i vlastní identifikátor, který musí být unikátní.

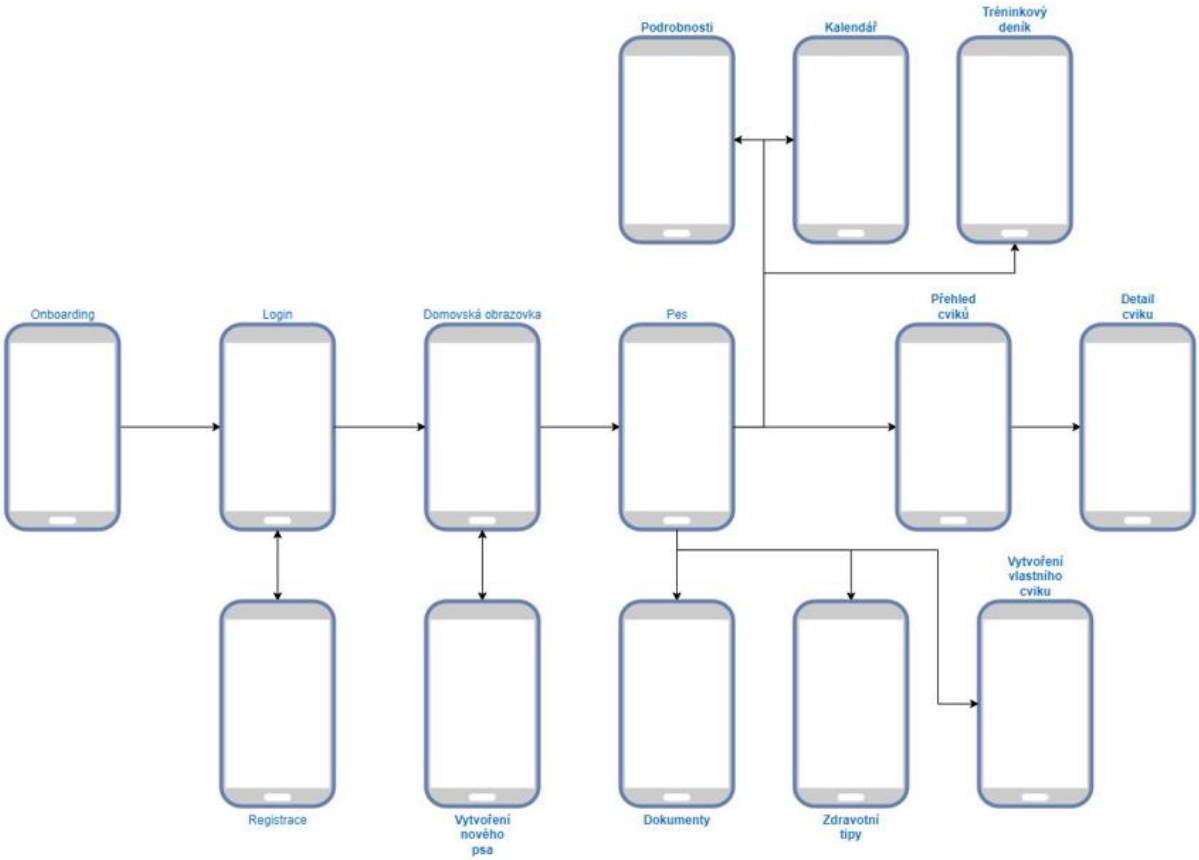
Návrh databáze, jak již bylo zmíněno, bude vycházet z návrhu objektového modelu. Do modelu budou přidány cizí klíče a typy atributů. Tento krok bude nápomocen k pochopení jednotlivých závislostí mezi jednotlivými tabulkami.



Obrázek 21 Databázový model NoSQL

Posledním návrhem je návrh návaznosti neboli navigace mezi jednotlivými obrazovkami. Návrh uvede, z jaké obrazovky se lze dostat na jakou další. Ujasnění jednotlivých kroků bude nápomocné při vytváření navigační logiky aplikace a její vnitřní struktury. Prvním krokem budou obrazovky onboardingu uživatele, který vysvětluje klíčové funkce a možnosti aplikace, díky čemuž uživatelé získají rychlý přehled o tom, co mohou od aplikace očekávat. Uživateli může pomoci začít pracovat s aplikací a snížit pocit zmatku. Následovat budou obrazovky pro registraci a přihlášení uživatele pomocí emailu a hesla anebo pro lepší dostupnost aplikace pomocí Google Sign In. Po zaregistrování a přihlášení se uživatel dostane na domovskou obrazovku, kde najde přehled jednotlivých pejsků ve výcviku. Po kliknutí na vybraného pejska se uživatel dostane na detail, kde jsou vidět podrobnosti ve formě jména, věku, data narození a míry vycvičení pejska. Po kliknutí na horní část obrazovky, detail, kde jsou informace o pejskovi, je uživatel přesměrován na detail o pejskovi, kde nalezne informace o očkování. Pod detailem budou jednotlivé tlačítka pro přechod na další obrazovky s přehledem jednotlivých cviků, kalendář, úprava dat o pejskovi, možností vymazání pejska a první pomoc ve formě zdravotních tipů (dokumenty). Na další obrazovku se pak lze dostat pouze z obrazovky s přehledem cviků. Odtud je možné se dostat na jednotlivé cviky či si vytvořit vlastní cvičení dle potřeb uživatele a cvičeného pejska. Na obrazovce „Kalendář“ se na další obrazovku uživatel nedostane, nicméně zde bude modální (vyskakovací) okno pro vytvoření nové události.

Z obrazovky pejska se lze dostat na tréninkový diář. Na první obrazovce lze vidět přehled již absolvovaných tréninkových relací. Na druhé je možné nový trénink přidat.



Obrázek 22 Návrh propojení obrazovek

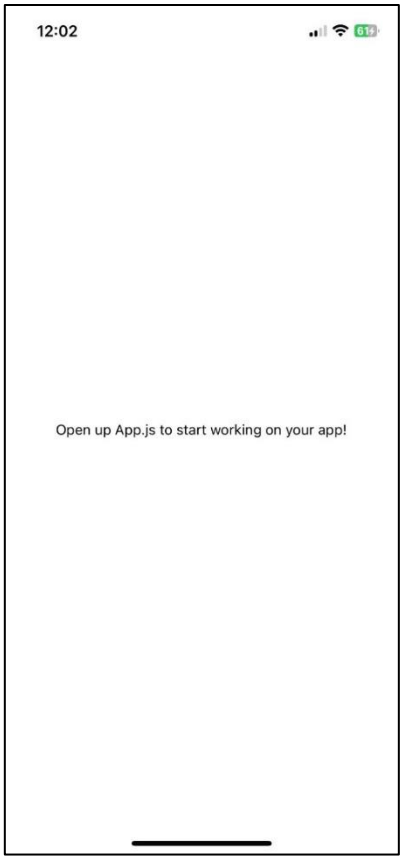
4.5 Vytvoření aplikace a použití Expo GO

Pro vytvoření aplikace ve Visual Studio Code v terminálu stačí napsat příkaz *npx create-expo-app [název_aplikace]* (31). Do složky vytvořeného projektu lze vstoupit pomocí příkazu *cd ./[název_aplikace]*. Aplikace je vytvořena a připravena pro vývoj a testování. Nově vytvořenou aplikaci lze hned spustit pomocí mobilní aplikace Expo GO nebo pomocí simulátorů. Pro použití aplikace Expo GO stačí naskenovat QR kód, který se vygeneruje v terminálu pomocí příkazu *npx expo start*.

Obrázek 23 Spuštění aplikace pomocí Expo

Z obrázku je patrné, že lze použít příkazy pro otevření jednotlivých simulátorů nebo aplikaci pro web (v tomto případě to není možné, jelikož je vyvíjena pouze mobilní aplikace). Aplikaci je také možné znovu načíst. Funkce opětovného načtení je důležitá, pokud je *hot reload,* nedostačující, např. při opětovném načítání dat z databáze.

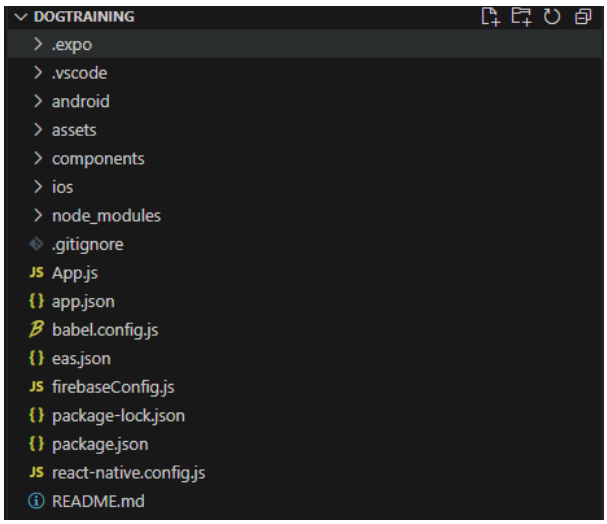
Nově vytvořená aplikace bez jakéhokoliv zásahu je zachycena na následujícím obrázku. Snímek obrazovky je zachycen na mobilním zařízení, kde je aplikace spuštěna pomocí Expo GO.



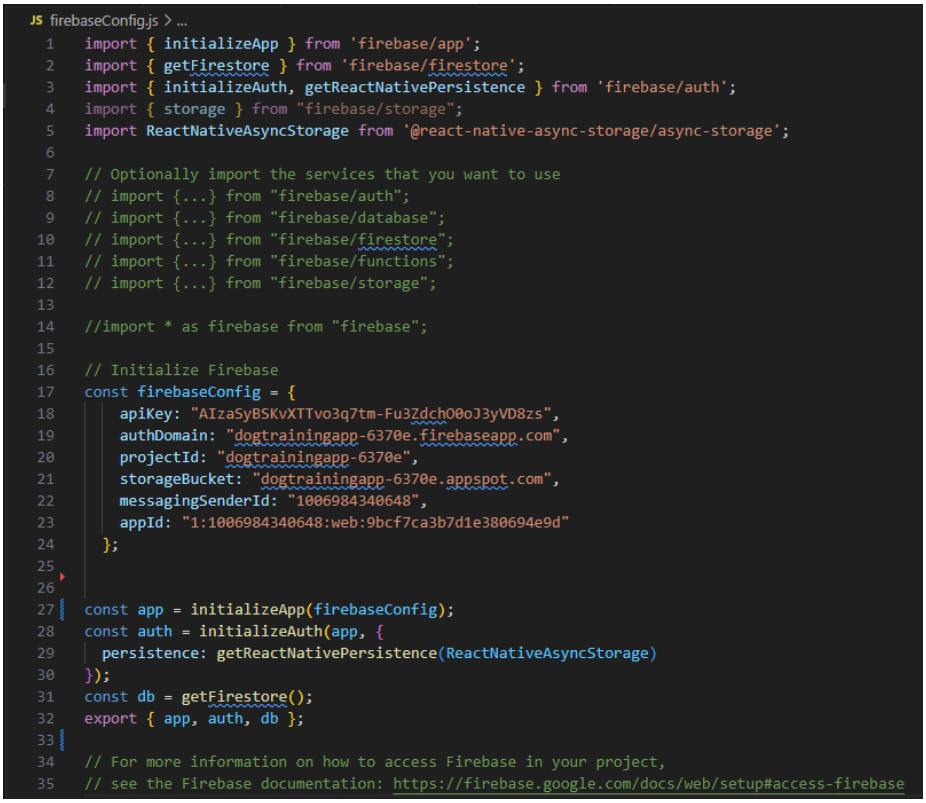
Obrázek 24 Mobilní obrazovka nové aplikace

4.6 Konfigurace Firestore a propojení s aplikací

Pro využívání Firebase technologií je potřeba propojit vyvíjenou aplikaci s Firebase prostředím pomocí konfiguračního souboru. Obsah souboru je dostupný ve Firebase konzoli. Získanou konfiguraci stačí nakopírovat do souboru, např. firebaseConfig.js. Obsah firebaseConfig.js vypadá následovně.



Obrázek 25 Struktura aplikace ve Visual Studio Code

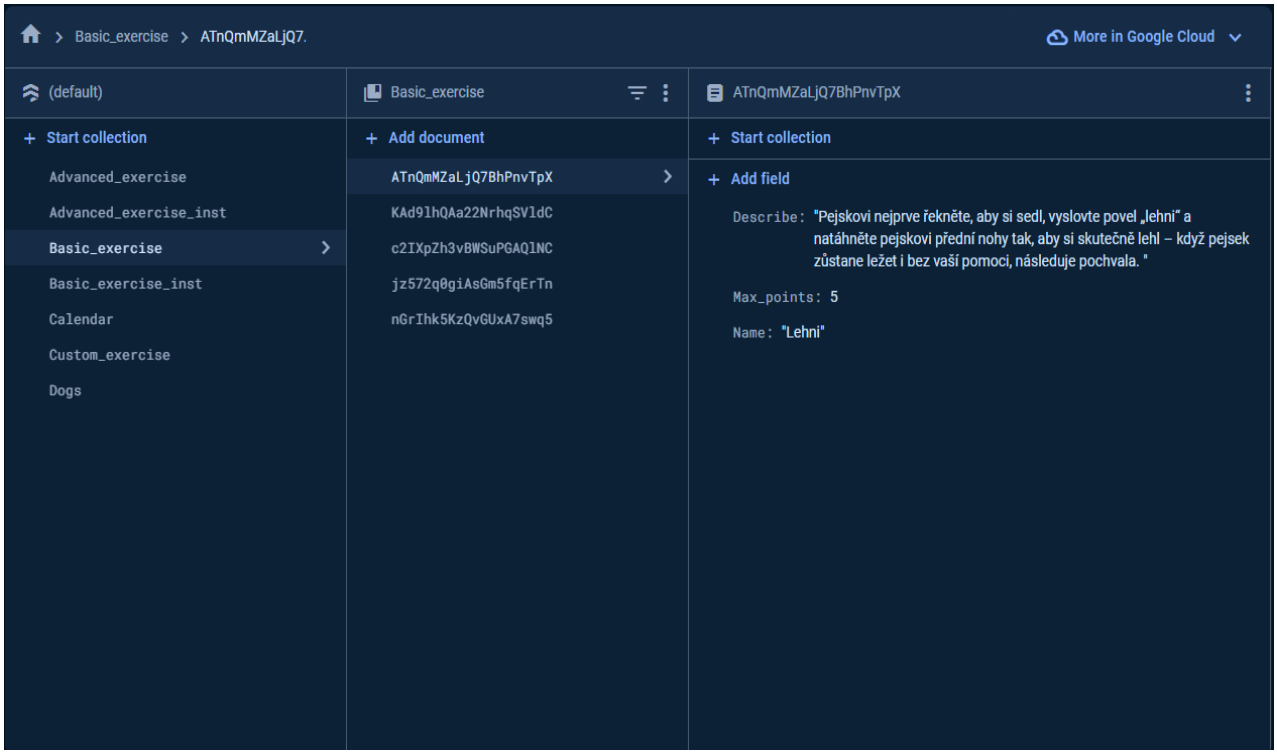


```
JS firebaseConfig.js > ...
1   import { initializeApp } from 'firebase/app';
2   import { getFirestore } from 'firebase/firestore';
3   import { initializeAuth, getReactNativePersistence } from 'firebase/auth';
4   import { storage } from "firebase/storage";
5   import ReactNativeAsyncStorage from '@react-native-async-storage/async-storage';
6
7   // Optionally import the services that you want to use
8   // import {...} from "firebase/auth";
9   // import {...} from "firebase/database";
10  // import {...} from "firebase/firestore";
11  // import {...} from "firebase/functions";
12  // import {...} from "firebase/storage";
13
14  //import * as firebase from "firebase";
15
16  // Initialize Firebase
17  const firebaseConfig = {
18      apiKey: "AIzaSyBSKvXTTvo3q7tm-Fu3ZdchO0oJ3yVD8zs",
19      authDomain: "dogtrainingapp-6370e.firebaseapp.com",
20      projectId: "dogtrainingapp-6370e",
21      storageBucket: "dogtrainingapp-6370e.appspot.com",
22      messagingSenderId: "1006984340648",
23      appId: "1:1006984340648:web:9bcf7ca3b7d1e380694e9d"
24    };
25
26
27  const app = initializeApp(firebaseConfig);
28  const auth = initializeAuth(app, {
29    persistence: getReactNativePersistence(ReactNativeAsyncStorage)
30  });
31  const db = getFirestore();
32  export { app, auth, db };
33
34  // For more information on how to access Firebase in your project,
35  // see the Firebase documentation: https://firebase.google.com/docs/web/setup#access-firebase
```

Obrázek 26 Konfigurační soubor Firebase

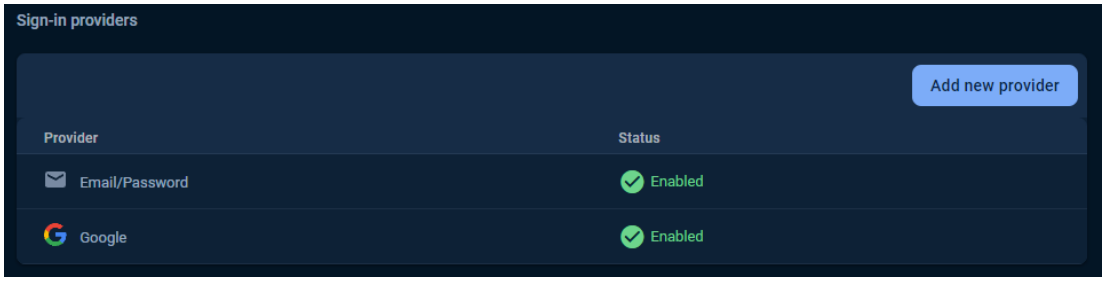
Jak je vidět na konci obrázku, exportuje se *app, auth* a *db* pro jednotlivé funkcionality, které budou využity V komentářích lze najít informaci o možnosti využití další funkcionality v případě potřeby. Při inicializaci funkce pro autentizaci je využita knihovna *ReactNativeAsyncStorage*, pomocí které si bude aplikace pamatovat přihlášeného uživatele i po vypnutí aplikace. Asynchronní uložiště dále poskytne funkcionalitu i pro zapamatování si, jestli už uživatel prošel takzvaným onboardingem aplikace.

Následuje připravení databáze pro aplikaci. Vytvořena bude tabulka *Dogs*, pro jednotlivé pejsky, *Calendar* pro kalendář jednotlivých pejsků, a dále tabulky pro cviky. Cviky jsou rozděleny do tří kategorií na základní cviky (*Basic_exercise_inst*), pokročilé cviky (*Advanced_exercise_inst*) a vlastní cviky (*Custom_exercise*). Vzhledem k výše popsanému řešení je potřeba v databázi o dvě tabulky navíc. Jedná se o přehlednější řešení, než aby byly jednotlivé cviky natvrdo napsány v kódu. Samotná aplikace je přehlednější a správa aplikace a jednotlivých cviků je výrazně lehčí. Při vytváření karty nového pejska jsou zmíněné dvě tabulky procházeny a jsou vytvářeny jednotlivé instance těchto cviků v tabulkách s koncovkou *_inst.*



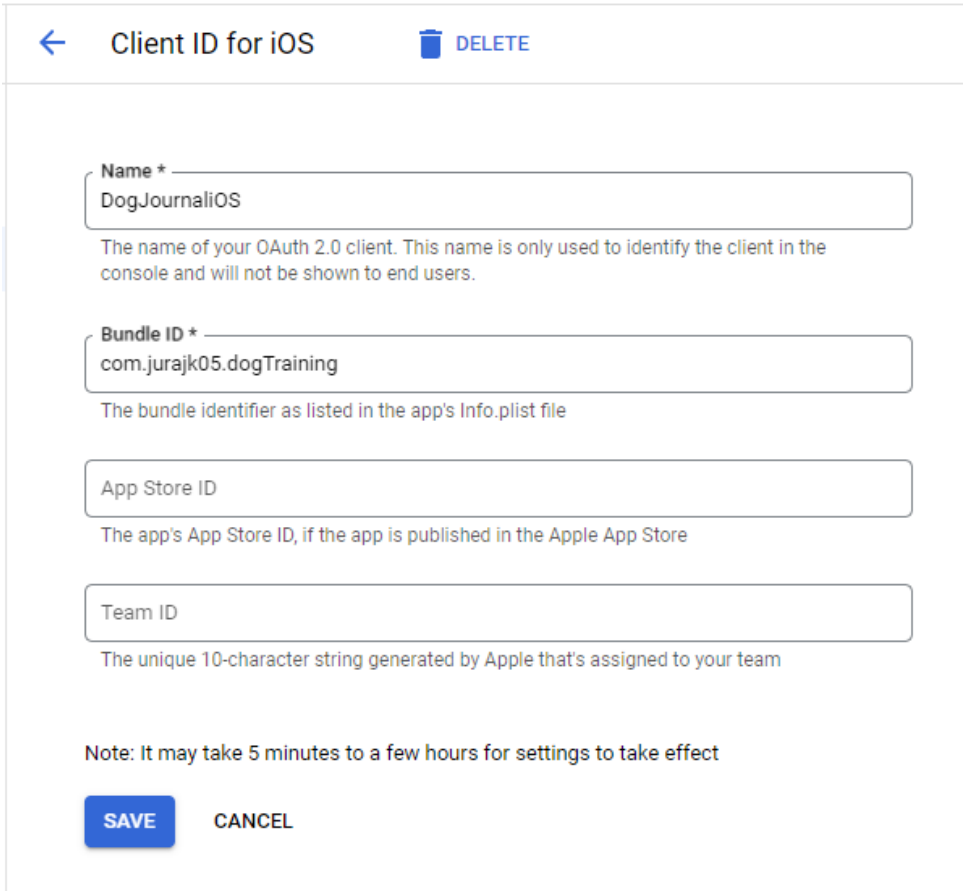
Obrázek 27 NoSQL databáze ve Firebase

Dalším krokem je nastavení možností přihlašování do aplikace. V sekci *Authentication* ve Firebase konzoli lze vybrat požadované zprostředkovatele. V případě vyvíjené aplikace to jsou možnosti *Email/Password* a přihlášení pomocí Google.



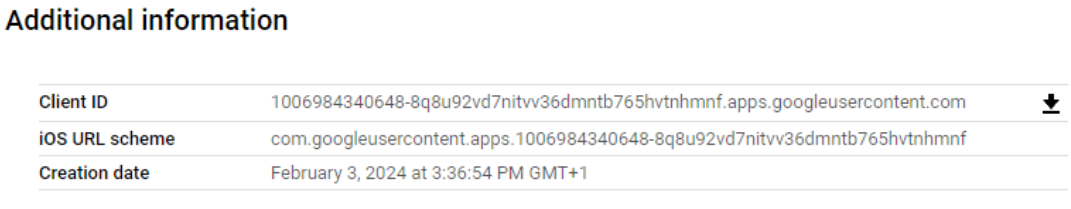
Obrázek 28 Firebase - sign-in providers

Poté je nutné vytvořit ve Firebase dvě aplikace pro náš projekt. Jedna aplikace je pro iOS a druhá aplikace je pro Android. Pro každou aplikaci je nutné vyplnit informace o aplikaci. Pro iOS to je *Bundle ID*, pro Android to je *Package Name* a *SHA-1* certifikát. Následně je nutné vytvoření *Credentials* v *Google Cloud* konzoli v sekci *APIs & Services*. Pro obě platformy je potřeba vytvořit separátní *Credentials*, kam se vyplní název přístupu a opět *Bundle ID*. Obdobné to je i pro Android.



Obrázek 29 Google Credentials - vytvoření údajů pro Google SignIn

Po vytvoření *Credentials* je vygenerován *Client ID*, které se nakopíruje do aplikace v sekci přihlašování: Jedná se o jeden z potřebných atributů funkce pro použití Google přihlášení. Ve stávajícím stavu je aplikace připravena pro použití *GoogleSignIn*.



Obrázek 30 Google Credentials – vytvořené přihlašovací údaje

```
68    GoogleSignin.configure({
69      webClientId:
70        "1006984340648-vs9huss8bo66cpmmbpdssmh9m658nmob.apps.googleusercontent.com",
71      iosClientId:
72        '1006984340648-8q8u92vd7nitvv36dmntb765hvtnhmnf.apps.googleusercontent.com',
73    });
```

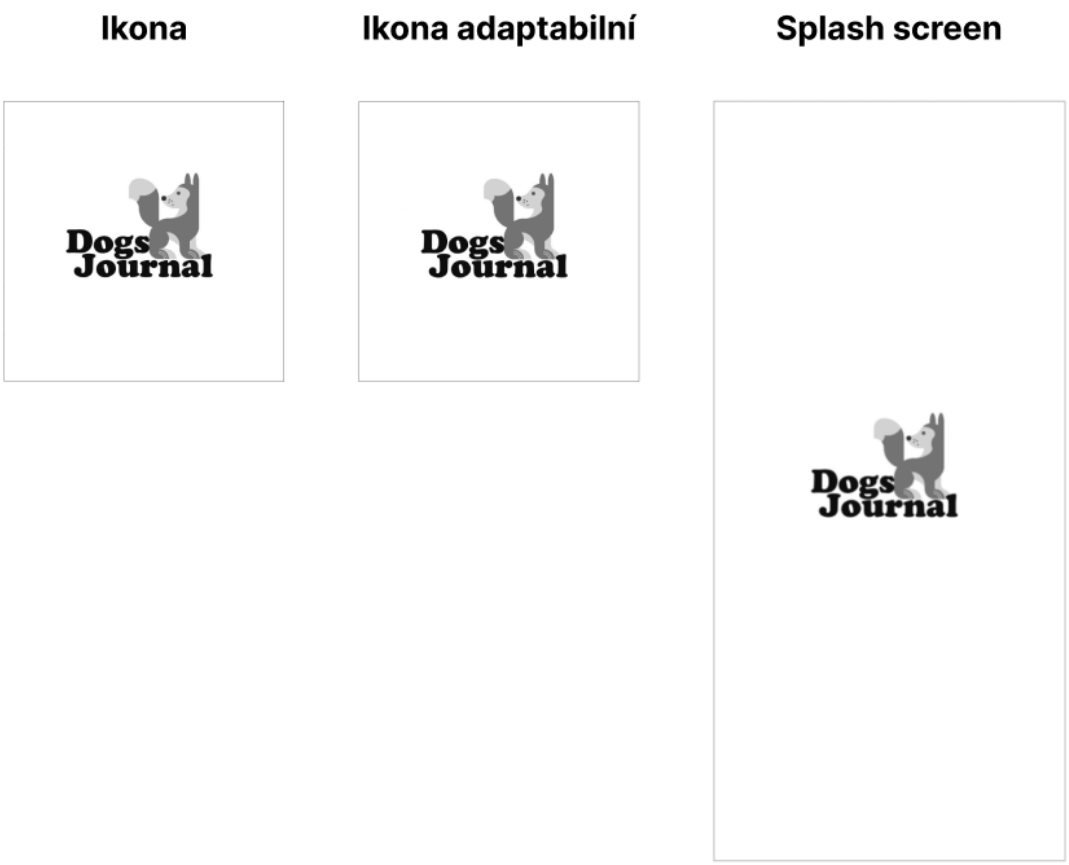
Obrázek 31 Nastavení Google SignIn

4.7 Implementace

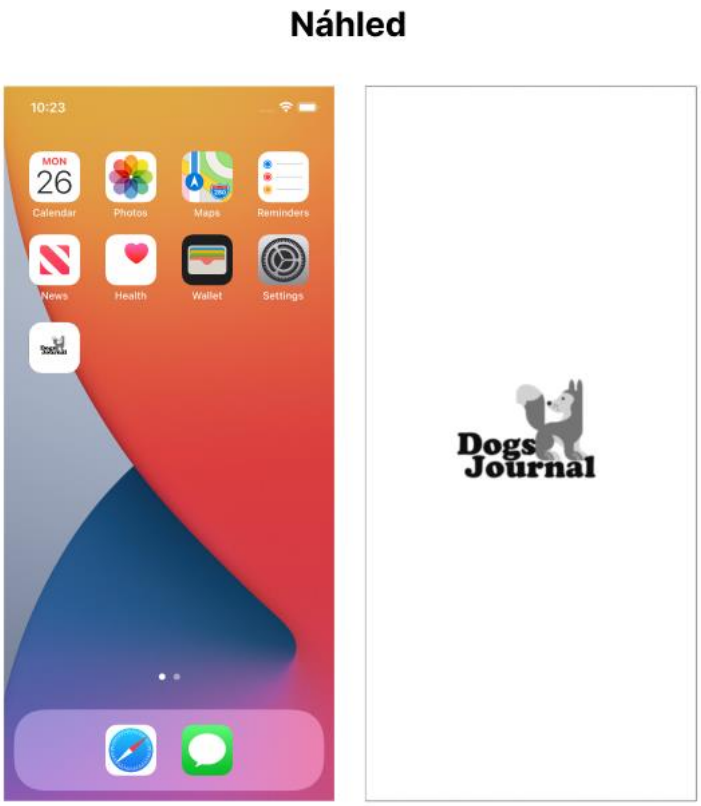
Popis implementace aplikace bude probíhat v postupných krocích, podle předpokládaného průchodu uživatele aplikací. Jak se dá předpokládat, první kroky budou onboarding, přihlášení nebo registrace.

4.7.1 Splash screen a ikony aplikace

Při vývoji mobilní aplikace je vizuální dojem klíčový. Splash screen a ikony poskytují první pohled na aplikaci a mají výrazný vliv na uživatelský zážitek. V React Native s Expo je proces vytvoření a nastavení splash screenu a ikon relativně jednoduchý. Splash screen je obrazovka, která se zobrazí okamžitě po spuštění aplikace před načtením hlavního obsahu. Účelem Splash screen je poskytnout vizuální zpětnou vazbu o spuštění aplikace. Ikonky aplikace jsou malé grafické prvky, které reprezentují aplikaci na domovské obrazovce zařízení. Jsou klíčové pro rozpoznání a přístupnost aplikace. Pro návrh ikon a splash screenu bude opět využit nástroj Figma. Figma disponuje šablonou kam stačí nahrát návrh pro ikonu a splash screen.



Obrázek 32 Vytvořené ikony a Splash screen



Obrázek 33 Náhled ikony a Splash screen

Vytvořené ikony a splash screen je nutné uložit do složky projektu, kam se ukládají použité obrázky, animace a další. Poté je potřebné nastavit proměnné pro ikony a splash screen v konfiguračním souboru app.json.

4.7.2 Vstupní část aplikace

Po spuštění se aplikace nachází v souboru *App.js*, ve kterém je řešen stav uživatele a případný onboarding. V případě prvního spuštění aplikace je uživatel proveden onboardingem, a následně je odeslán na přihlašovací obrazovku.

```
31  const [firstTimeOpened, setFirstTimeOpened] = useState(true);
32  useEffect(() => {
33    const checkForFirstTimeLoaded = async () => {
34      const result = await AsyncStorage.getItem("isFirstTimeOpen");
35      if (result === null) {
36        console.log('first time');
37        setFirstTimeOpened(true);
38        await AsyncStorage.setItem("isFirstTimeOpen", "false");
39      } else {
40        setFirstTimeOpened(false);
41        console.log('not first time');
42      }
43    };
44    checkForFirstTimeLoaded();
45  }, []);
```

Obrázek 34 Využití AsyncStorage k zobrazení onboardingu pouze při prvním zapnutí aplikace

Zde je ověřeno, zda uživatel už prošel onboardingem. Pokud ano, pak aplikace uživatele pošle na přihlašovací obrazovku anebo na domovskou obrazovku v závislosti na stavu přihlášení. Přihlášení uživatele obstarává *ReactNativeAsyncStorage* a přechod na další obrazovky obstarává *Stack.Navigator*, kam se jednotlivé obrazovky přidávají. Obrazovky je nejdříve nutné naimportovat, aby byly v tomto souboru dostupné.

```
import Home from "./components/Home/Home";
import Dog from "./components/Dog/Dog";
import Docs from "./components/Dog/Docs";
import NewCustomSkill from "./components/Dog/NewCustomSkill";
import SkillOverview from "./components/Dog/SkillOverview";
import SkillDetail from "./components/Dog/SkillDetail";
import AddDog from "./components/AddDog/AddDog";
import EditDog from "./components/AddDog/EditDog";
import Journal from "./components/Journal/Journal";
import Login from "./components/Login/Login";
import HealthAdvice from "./components/HealthAdvice/HealthAdvice";
import Register from "./components/Login/Register";
import Achievements from "./components/Achievements/Achievements";
import OnboardingScreens from "./components/Login/OnboardingScreens";
import DogDetail from "./components/DogDetail/DogDetail";
import DogDetailEdit from "./components/DogDetail/DogDetailEdit";
import TrainingDiary from "./components/TrainingDiary/TrainingDiary";
import AddTrainingDiary from "./components/TrainingDiary/AddTrainingDiary";
```

Obrázek 35 Import jednotlivých obrazovek aplikace

Pro uspořádání aplikace byla vytvořena složka *components*, kde jsou vytvářeny další složky pro logické uspořádání jednotlivých obrazovek. Propojení všech obrazovek pomocí *NavigationContainer* zobrazeno na následujícím obrázku.

```
<NavigationContainer independent={true}>
  <Stack.Navigator>
    <Stack.Screen component={Auth} name="Auth" />
    <Stack.Screen name="Home" component={Home} options={{ headerShown: false, }} />
    <Stack.Screen name="Dog" component={Dog} options={{ headerShown: false, }} />
    <Stack.Screen name="AddDog" component={AddDog} options={{ headerShown: false, }} />
    <Stack.Screen name="EditDog" component={EditDog} options={{ headerShown: false, }} />
    <Stack.Screen name="Journal" component={Journal} options={{ headerShown: false, }} />
    <Stack.Screen name="Docs" component={Docs} options={{ headerShown: false, }} />
    <Stack.Screen name="NewCustomSkill" component={NewCustomSkill} options={{ headerShown: false, }} />
    <Stack.Screen name="SkillOverview" component={SkillOverview} options={{ headerShown: false, }} />
    <Stack.Screen name="SkillDetail" component={SkillDetail} options={{ headerShown: false, }} />
  </Stack.Navigator>
</NavigationContainer>
```

Obrázek 36 Nastavení NavigationContaineru

U obrazovek onboardingu, není mnoho funkcionalit. Za zmínku stojí knihovna *react-native-onboarding-swiper*, která usnadňuje použití onboardingu. Celkový výsledek je zachycen na následujícím obrázku.

```
<Onboarding
  nextLabel="Další"
  skipLabel="Přeskočit"
  onDone={() => navigation.navigate("Login")}
  onSkip={() => navigation.navigate("Login")}
  pages={[
    {
      backgroundColor: "white",
      image: (
        <View>
          <LottieView
            source={require("../../assets/animations/animation_running_dog.json")}
            autoPlay={true}
            loop={true}
            style={{ width: width * 0.8, height: width }}
          />
        </View>
      ),
      title: "Cviky a povely",
      subtitle: "Vycvičte své pejsky a naučte je nové cviky a povely!",
    },
    {···
    },
    {···
    },
  ]}
/>
```

Obrázek 37 Kód onboardingu

OnboardingData je pole obsahující informace pro jednotlivé obrazovky onboardingu ve formě JSONu.

Dalším krokem je přihlášení uživatele. Pokud má uživatel vytvořený účet nebo využívá *GoogleSingIn,* stačí pokud se přihlásí na přihlašovací obrazovce. Přihlášení pomocí emailu je řešeno pomocí funkce *signInWithEmailAndPassword()*, které je poskytnut vstup od uživatele v podobě emailu a hesla. Funkce buď zařídí přihlášení nebo uživatel obdrží informaci o nemožnosti přihlásit se.

```
const handleLogin = () => {
  signInWithEmailAndPassword(auth, email, password)
    .then((userCredentials) => {
      const user = userCredentials.user;
    })
    .catch((error) => alert(error.message));
};
```

Obrázek 38 Kód přihlášení emailem a heslem

Pro *GoogleSingIn* je použita funkce *signInAndRetrieveDataWithCredential*, která provede ověření uživatele pomocí Google API, a poté samotné přihlášení uživatele.

```
45  const handleSingInGoogle = () => {
46    try {
47      const result = Expo.Google.logInAsync({
48        androidClientId: "Your Client ID",
49        //iosClientId: YOUR_CLIENT_ID_HERE,  <-- if you use iOS
50        scopes: ["profile", "email"]
51
52      })
53      if (result.type === "success") {
54        const credential = firebase.auth.GoogleAuthProvider.credential(result.idToken, result.accessToken);
55        firebase.auth().signInAndRetrieveDataWithCredential(credential).then(function (result) {
56          console.log(result);
57        });
58        this.props.navigation.navigate('Where you want to go');
59      } else {
60        console.log("cancelled")
61      }
62    } catch (e) {
63      console.log("error", e)
64    }
65  };
```

Obrázek 39 Přihlášení pomocí Google SignIn

Pro použití *GoogleSignIn* je nezbytný klientský identifikátor aplikace, který byl nastaven a vložen do aplikace v předchozí podkapitole o konfiguraci aplikace a přidružených prostředích. Jsou potřeba dvě ID, jak pro iOS aplikaci, tak pro Android aplikaci. V případě neúspěšného přihlášení pomocí poskytnutého rozhraní od Google je uživatel informován o příslušné chybě. Uživatel je po přihlášení přesměrován na domovskou obrazovku.

V případě neexistujícího účtu uživatel může využít možnost registrace pomocí emailu a hesla. Funkcionalita je poskytnuta knihovnami Firebase a slouží k tomu jednoduchá funkce, která zajistí vytvoření účtu, a v případě chyby uživatele informuje o chybě.

4.7.3 Domovská obrazovka

Po úspěšném onboardingu a přihlášení je uživatel přesunut na domovskou obrazovku, která slouží pro přehled jednotlivých pejsků ve výcviku. Ke každému pejskovi je zobrazeno jméno a fotka. Dále má uživatel možnost přidat dalšího pejska, stejně tak odhlásit se, nebo přejít na obrazovku s detailem zvoleného pejska.

```
34  useEffect(() => {
35    async function fetchData() {
36      const collectionRef = collection(db, "Dogs");
37      const q = query(collectionRef, where("UserId", "==", user.uid));
38      const snapshot = await getDocs(q);
39      setDogs(snapshot.docs.map((doc) => ({ ...doc.data(), id: doc.id })));
40    }
41    fetchData();
42  }, [isFocused]);
```

Obrázek 40 Načítání pejsků z databáze pro konkrétního uživatele

Funkce *fetchData* v *useEffect* hooku se stará o načtení všech pejsků z Firebase databáze. Jedná se o asynchronni funkci, která využívá funkcí Firebase. Na začátku máme již zmíněný import pro tyto funkce, společně s dalšími prvky jako *collection*, nebo *query* a dalšími.

Ve funkci *fetchData* je nejprve vytvořena databáze pomocí *getFirestore()*. V této databázi je potřeba pracovat s kolekcí dat *Dogs*. Následně je vytvořen dotaz neboli *query*, kde se specifikuje, jaké záznamy má dotaz vrátit. V tomto případě jsou vráceni všichni pejsci, jejichž identifikátor uživatele, *UserID*, je rovno právě přihlášenému uživateli. V případě, že uživatel nemá žádného pejska ještě vytvořeného, je vyzván k jeho vytvoření.

Pro odhlášení uživatele slouží jednoduchá funkce, která uživatele po odhlášení přesune na přihlašovací obrazovku.

```
51    const doLogOut = () => {
52        signOut(auth).then(() => { navigation.replace('Login') });
53    };
```

Obrázek 41 Kód odhlašování uživatele

V horní části obrazovky, kde je fotka pejska a informace o něm, je součástí počet bodů a informace o vycvičení pejska. Úroveň vycvičení udává slovní hodnocení a barevná ikonka pejska. Hodnocení vychází z jednoduchého rozdělení počtu získaných bodů a maximálního možného počtu bodů za všechny cviky. Rozdělení je následující:

- 100 % až 81 % - pejsek je vycvičen
- 51 % až 80 % - pejsek ještě není vycvičen, ale je na střední úrovni
- 0 % až 50 % - pejsek není vycvičen

Horní část obrazovky je určena i pro rozkliknutí detailu pejska. V tomto detailu je vidět na jednom místě všechny informace o pejskovi a jeho očkováních. V případě, že si uživatel nastaví, resp. aktualizuje informace o očkování, pak je vidět datum expirace tohoto očkování. Zahrnuty jsou hlavní očkování pro pejsky. Detaily o očkování jsou uloženy v databázové tabulce *Dog_details*.

4.7.4 Vytvoření nového pejska

Vytvoření nového pejska obsahuje složitější funkcionalitu. Je nezbytné vytvoření samotného profilu pejska a je nutné i vytvoření instancí jednotlivých cviků od základních až po pokročilé. V kalendáři se vytvoří událost pro narozeniny pejska. Nejdříve uživatel vyplní základní údaje o novém pejskovi, ke kterým patří jméno, plemeno a datum narození. V

posledním kroku si uživatel může vybrat fotografii pejska ze své galerie fotek. Pro výběr fotografie je použita funkce *pickImage*, která využívá *ImagePicker* knihovny a umožňuje lehký přístup k fotografiím. Funkce umožňuje i lehkou úpravu fotky pomocí atributu *allowsEditing*, který je nastaven tak, aby úpravy umožňoval. Pokud se nevyskytne chyba, fotka je zobrazena již na obrazovce při vytváření profilu pejska, a by si uživatel mohl zkontrolovat, že vybral správnou fotku.

```
94    const pickImage = async () => {
95        let result = await ImagePicker.launchImageLibraryAsync({
96            mediaTypes: ImagePicker.MediaTypeOptions.All,
97            allowsEditing: true,
98            aspect: [4, 3],
99            quality: 1,
100       });
101
102       if (!result.canceled) {
103           setImage(result.assets[0].uri);
104       }
105   };
```

Obrázek 42 Kód pro výběr obrázku pejska

Vybraná fotka pejska se nikam nekopíruje, ukládá se pouze odkaz na ní, aby se omezilo přenášené množství dat. Poslouží to také rychlejšímu načítání domovské obrazovky a obrazovky detailu pejska, jelikož se nebude muset fotka pokaždé stahovat z databáze a nebude potřeba ji uchovávat v lokálním uložišti aplikace. Nevýhoda je, že pokud uživatel vymaže tuto fotku, pak o ní definitivně přijde a pro pejska v aplikaci bude muset být použit výchozí obrázek.

Samotná funkce pro vytvoření nového pejska obsahuje několik úloh, které je potřeba vykonat, aby byly všechny potřebné tabulky v databázi naplněné a připravené pro další použití. Funkce *handleNew()* se stará o všechny potřebné procesy. Prvním je vytvoření samotného pejska.

```
49  const handleNew = async () => {
50    require("moment/locale/cs.js");
51    const normalFormatDate = moment(date).format("L");
52    console.log(normalFormatDate);
53    const collectionRefDog = collection(db, "Dogs");
54    const payloadDog = {Name: textName, Breed: textBreed,DOB: normalFormatDate,UserId: item.uid,Photo: photoImage,Received_points: 0};
55    let dogID;
56    await addDoc(collectionRefDog, payloadDog).then((collectionRefDog) => {
57      dogID = collectionRefDog.id;
58    });
```

Obrázek 43 Funkce vytvářející nového pejska v databázi

Zde se nejdřív volá metoda *require()*, která nastavuje český formát času, který je potřeba kvůli datu narození. Dále se pomocí *getFirestore()* připraví databáze, která se zaměří na kolekci *Dogs*, protože se bude vytvářet nový dokument právě v této kolekci. Naplníme si proměnnou *payloadDog* hodnotami, které jsme získali od uživatele pomocí vstupů, a navíc se nastaví

unikátní identifikátor uživatele podle právě přihlášeného uživatele. Počet získaných bodů neboli *Received_points*, je nastaven na počáteční hodnotu nula. Poté je vytvořena nová proměnná *dogID*, do které se po vytvoření pejska nastaví hodnota nově vytvořeného dokumentu ve Firebase databázi. Nastavená hodnota slouží jako unikátní identifikátor pejska, a je využita pro jednotlivé cviky a pro události v kalendáři. Úplně nakonec je voláno samotné vytvoření záznamu v databázi v podobě dokumentu. Pro vytvoření záznamu, respektive dokumentu v databázi se používá funkce *addDoc()*, která sama vytvoří samotný záznam i unikátní identifikátor pro tento záznam, tedy *UID*. Funkce *addDoc()* je použita ve všech případech vytváření záznamů v databázi. Po vytvoření nového záznamu lze získat identifikátor právě vytvořeného záznamu pomocí *.then(collectionRefDog => { dogID = collectionRefDog.id; })*, které je potřeba v následující části. V další části budou připraveny a vytvořeny všechny základní cviky. Cviky získáme z kolekce *Basic_exercise*.

```
62    const collectionRefBasic_inst = collection(db, "Basic_exercise_inst");
63    const snapshotBasic = await getDocs(collection(db, "Basic_exercise"));
64
65    snapshotBasic.forEach((doc) => {
66        const constNewBasicPayload = { Name: doc.data().Name, Describe: doc.data().Describe,
67                                       DogID: dogID, Max_points: doc.data().Max_points, Points: 0,
68                                       Type: 'Basic_exercise_inst' };
69        addDoc(collectionRefBasic_inst, constNewBasicPayload);
70    });
```

Obrázek 44 Kód vytvářející jednotlivé cviky a povely – základní varianta

Je vytvořen objekt pro kolekci *Basic_exercise_inst*, kam se budou nově vytvořené cviky, instance, pro konkrétního pejska zapisovat. V objektu *snapshotBasic* jsou uloženy všechny cviky, které byly nalezeny v kolekci cviků. Díky tomu lze lehce přes všechny tyto cviky iterovat a získávat informace o jednotlivých cvičeních pomocí zápisu *doc.data()*. Ku příkladu *doc.data().Name* vrátí jméno konkrétního cvičení, například Lehni. Na konci se přiřazuje každému cviku nula bodů jako výchozí hodnota. I když je tato hodnota natvrdo napsaná v kódu, není to problém, protože každý cvik musí začínat na nule. Popsaný přístup umožňuje přidávat cviky do tabulky cviků v databázi a zároveň není potřeba měnit kód, který tak je lehce udržovatelný a přehledný.

Pro pokročilejší cviky je funkcionalita obdobná, pouze se změnou jednotlivých kolekcí, které jsou použity v kódu. Tedy místo *Basic_exercise* je zde *Advanced_exercise* apod.

Poslední funkcionalitou je vytvoření událostí typu narozeniny do kolekce *Calendar*. Zde se využije vstupu od uživatele, který vložil datum narození. Vytvoří se patnáct událostí tohoto

typu. Průměrná délka dožití je podle (32) deset až třináct let, ale jelikož pejsci v aplikaci budou podstupovat různé cviky, tak by se jejich průměrný věk dožití mohl posunout o rok nebo dva.

V *payloadJournal*, tedy v datech pro vytvoření nového dokumentu v kolekci *Calendar*, je *DateOfEvent* (datum dané události), unikátní identifikátor pejska a název události a *SelectedColor* a *SelectedEvent*. *SelectedColor* slouží pro barevné vyznačení události v kalendáři a také z pohledu uživatele určovat důležitost události. V uvedeném případě se jedná o významnou událost, a tedy červená barva značí velkou důležitost. *SelectedEvent* udává, jestli má být událost vyznačena v kalendáři. Je nastavena na hodnotu *true*, tedy že událost má být vidět. Po vytvoření všech záznamů je uživatel automaticky vrácen na domovskou obrazovku pomocí funkce *navigation.goBack()*.

Uživatel má také možnost upravit informace, které na začátku o pejskovi zadal. Může upravit stejné informace jako ty, které zadával při jeho vytváření. upravit všechny informace, které zadával při vytváření profilu pejska. Jediným rozdílem je použití jiné funkce pro zápis dat do databáze. Jedná se o funkci *updateDoc()*, která obsahuje dva parametry. Jedním z nich je reference na záznam, který chceme upravovat a druhým parametrem jsou data, která mají být změněna.

```
40  const handleEdit = async () => {
41    await updateDoc(doc(db, "Dogs", item.id), {
42      Breed: textBreed,
43      Name: textName,
44      Photo: newImage,
45    });
46    navigation.goBack();
47  };
```

Obrázek 45 Funkce upravující základní data o pejskovi

Nakonec je opět volána funkce *navigation.goBack()*, pro návrat na obrazovku s detailem o pejskovi.

4.7.5 Vytvoření nového cviku

Uživatel si může vytvořit vlastní cvik v případě, že má specifické požadavky na trénink svého pejska nebo pokud potřebuje specifický cvik na závody. Z uvedených důvodů je možné si vlastní cvik vytvořit. Pro vytvoření vlastního cviku je potřeba jméno cviku, popis cviku, aby si uživatel nemusel pamatovat detaily související s cvičením a maximální počet bodů. Je přednastaveno dosažení deseti bodů, ale uživatel si může v případě potřeby navýšit maximum dle svých potřeb.

```
39  const handleNew = async () => {
40    const collectionRef = collection(db, "Custom_exercise");
41    const payload = {
42      Name: textName,
43      Describe: textDescription,
44      DogID: item.id,
45      Points: Number("0"),
46      Max_points: Number(maxPoints),
47      Type: "Custom_exercise",
48    };
49    await addDoc(collectionRef, payload);
50    navigation.goBack();
51  };
```

Obrázek 46 Funkce vytvářející nový vlastní cvik

Vlastní cvik je uložen do tabulky *Custom_exercise* s vyplněnými daty od uživatele. Po uložení nového cviku do databáze je uživatel přesunut na předešlou obrazovku s detailem pejska.

4.7.6 Přehled cviků a detail cviku

Obrazovka s cviky představuje přehled všech cviků od základních, přes pokročilé až po vlastní, pokud si je uživatel vytvořil. Při přechodu na tuto obrazovku se okamžitě provedou tři téměř identické funkce pro načtení dat jednotlivých cviků z databází. Níže jsou jednotlivé funkce pro načtení všech cviků pro konkrétního pejska.

```
39  useEffect(() => {
40    async function fetchData() {
41      const collectionRef = collection(db, "Basic_exercise_inst");
42      const q = query(collectionRef, where("DogID", "==", dog.id));
43      const snapshot = await getDocs(q);
44      setBasics(snapshot.docs.map((doc) => ({ ...doc.data(), id: doc.id })));
45    }
46    fetchData();
47  }, [isFocused]);
```

Obrázek 47 Funkce načítání cviků z databáze – základní varianta

Funkce načtou všechny cviky. Úplně na konci v hranatých závorkách je vidět proměnná *isFocused*, která je inicializovaná na začátku funkce pro tuto obrazovku.

```
25  const isFocused = useIsFocused();
```

Obrázek 48 Funkce pro zjištění, jestli je konkrétní obrazovka aktivní

Uvedená funkce je z knihovny *@react-navigation/native* a poskytuje informaci, jestli je tato obrazovka momentálně zobrazena na displeji telefonu, proměnná *isFocused* nabývá hodnoty *true*. Pokud se uživatel přesměruje tlačítkem na jinou obrazovku aplikace, zobrazení původní obrazovky již netrvá a proměnná *isFocused* je nastavena na *false*. Toto je důležité pro použití *useEffect* hooku, který má dva parametry, první je funkce, která může obsahovat například

funkci, v tomto případě načtení dat z databází, a druhým parametrem je pole závislostí. Pokud by pole bylo prázdné, hook by byl zavolán pouze při prvním vykreslení konkrétní obrazovky. To znamená, že v případě, kdy by uživatel byl přesměrován na obrazovku s detailem cviku a udělal by tam změnu – přidal by nebo ubral body pro konkrétní cvik – tato změna by nebyla reflektována po návratu na přehled všech cviků. S ohledem na uvedené potřeba volat metodu pro načtení informací o jednotlivých cvicích znova. K tomu je využita proměnná *isFocused* jako druhý parametr, aby *useEffect* pokaždé vykonal operace uvnitř. Vykreslení všech cviků zajištuje jediná komponenta *renderSkill*, jelikož všechny cviky mají podobné až stejné parametry.

```
70  const renderSkill = ({ item }) => {
71    return (
72      <TouchableOpacity
73        key={item.id}
74        onPress={() => navigation.navigate("SkillDetail", { item: item })}
75      >
76        <View style={styles.lineWrapper}>
77          <View style={styles.leftWrapper}>
78            <Entypo
79              name="baidu"
80              size={30}
81              color="black"
82              style={styles.leftIcon}
83            />
84            <Text style={styles.textStyle}>{item.Name}</Text>
85          </View>
86          <View style={styles.rightWrapper}>
87            <AntDesign
88              name="right"
89              size={28}
90              color="black"
91              style={styles.rightIcon}
92            />
93          </View>
94        </View>
95      </TouchableOpacity>
96    );
97  };
```

Obrázek 49 Univerzální funkce pro vykreslování všech typů cvičení a povelů

Všechny cviky tedy vypadají stejně. Obsahují název a aktuální počet bodů, které pejsek získal a také maximální počet bodů za cvik, který lze získat. Celý cvik je ohraničen v *TouchableOpacity*, čím je celá plocha vyhrazena cviku na obrazovce klikatelná, a přesune uživatele na další obrazovku spolu s informacemi o konkrétním cviku. To zajišťuje funkce v atributu *onPress navigation.navigate("SkillDetail", { item: item, })*, konkrétně proměnná *item*.

V případě, že uživatel nevytvořil žádné cviky, je potřeba zajistit, aby se tato část nevykreslovala na obrazovce, protože k vykreslení kromě názvu sekce nic není. Tohoto lze docílit pomocí

podmínky, která říká, že pokud proměnná, do které se ukládají cviky vytvořené uživatelem prázdná, nebude se nevykreslila.

```
{customsExists && (
  <View style={styles.exerciseWrapper}>
    <Text style={styles.typeOfSkill}>{customSkillText}</Text>
    <View>
      <FlatList
        style={styles.flatListItem}
        data={customs}
        renderItem={renderSkill}
        keyExtractor={(item) => item.id}
      />
    </View>
  </View>
)}
```

Obrázek 50 Podmíněné vykreslování vlastních cviků

Proměnná *customsExists* určuje, zda se zmíněná část vykreslí nebo nikoliv. Obrazovka detailu jednotlivých cviků je stejná pro všechny tři druhy cviků. Na tuto obrazovku je uživatel přesunut po kliknutí, respektive klepnutím, na jím vybraný cvik. Tato obrazovka slouží zejména pro přidávání nebo ubírání bodů pro konkrétní cvik. Je zobrazen název cviku, jeho popis v plném rozsahu a také aktuální počet bodů včetně maximálního možného počtu bodů. Obrazovka dále obsahuje jednotlivá tlačítka pro přidání bodu, odebrání a pro dokončení úpravy.

```
const handleAddPoint = () => {
    const addedPoint = Number(actualPoints + 1);
    if (addedPoint <= item.Max_points) {
        setActualPoints(Number(addedPoint));
    }
};

const handleDeductPoint = () => {
    const deductedPoint = Number(actualPoints - 1);
    if (deductedPoint >= 0) {
        setActualPoints(Number(deductedPoint));
    }
};
```

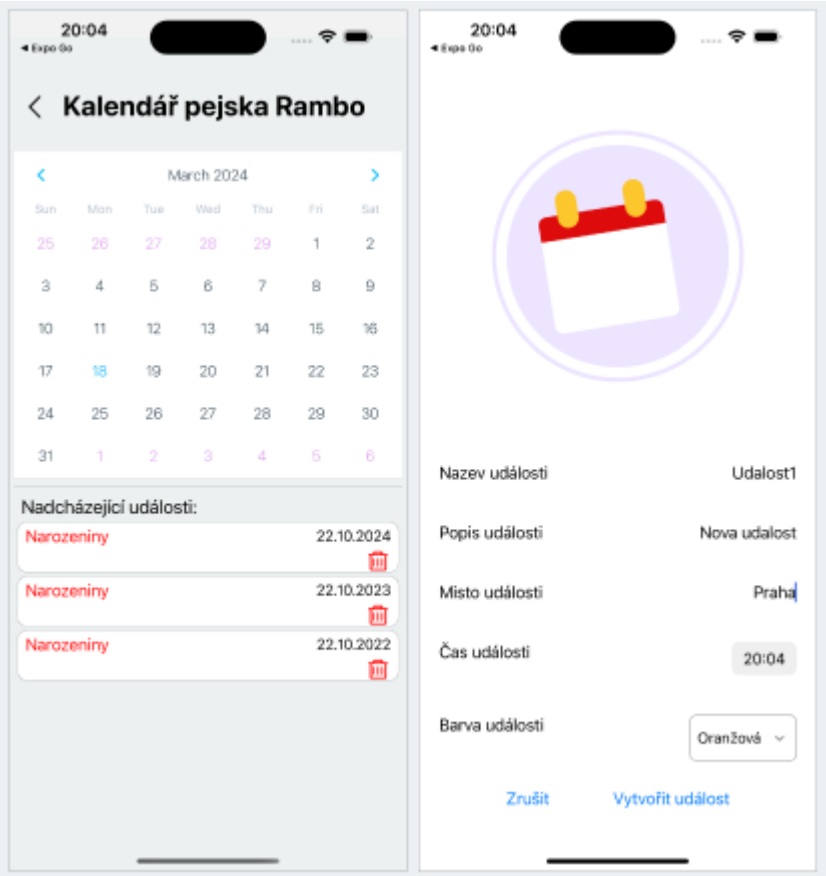
Obrázek 51 Přidání a odebrání bodů pro konkrétní cvik

Základem funkcionality jsou dvě jednoduché funkce *handleAddPoint* a *handleDeductPoint*. Při návrhu uvedených funkcí se nabízely dvě možnosti řešení. První možností bylo posílat po každém stisknutí tlačítka pro přidání bodu nebo ubrání bodu metodu *updateDoc()*, která by okamžitě odeslala informaci do databáze. Tento přístup je ale neefektivní z pohledu posílání dat přes síť a zapisování do databáze. Byla zvolena druhá možnost, kde se jednotlivé přidávání nebo ubírání bodů zaznamenává do proměnné. Následně, po kliknutí uživatele na tlačítko pro

dokončení akce, se informace odešle do databáze pomocí *updateDoc()* funkce. Následně je uživatel přesunut na předchozí obrazovku.

4.7.7 Kalendář

Obrazovka kalendáře se skládá ze dvou částí. V horní části se nachází samotný kalendář. V kalendáři je možné procházet jednotlivé měsíce a roky. Ve spodní části jsou zobrazeny nadcházející události seřazené od nejbližší události – úplně nahoře – a níže jsou události dále v čase. Události lze vytvořit při procházení kalendáře v horní části kliknutím na konkrétní den, ve který se má událost konat. Po kliknutí na konkrétní den se zobrazí modální okno. V okně lze vyplnit jednotlivé informace o události. Údaji, které lze vyplnit je název události, krátký popis události a místo jejího konání. Poslední informací o události je čas.



Obrázek 52 Obrazovka kalendáře a modální okno pro vytvoření nové události

Část kalendáře je součástí knihovny *react-native-calendars*, která je přizpůsobitelná pro potřeby uživatele pomocí atributu *theme* ve vykreslovací části kódu. V nastavení je možné přizpůsobit vše od barvy pozadí, barvy kalendáře, barvy čísel, až po různé možnosti ohraničení, jak tloušťka, barva či zaoblení ohraničení. Pro modální okno je využita základní knihovna *react-native*, která umožňuje příjemné používání a přizpůsobení modálního okna.

Při načtení okna se načítají informace o událostech z databáze *Journal*, kde se vybírají všechny záznamy, které mají unikátní identifikátor pejska dle pejska, se kterým je aktuálně manipulováno. Data jsou následně zobrazena ve spodní části o událostech a jsou také vyznačena v kalendáři barvou dle důležitosti události.

```
81   useEffect(() => {
82     async function fetchData() {
83       try {
84         const yearAway = new Date(
85           new Date().setFullYear(new Date().getFullYear() + 1)
86         );
87         const yearAwayString = moment(yearAway).format("DD.MM.YYYY");
88         const snapshot = await getDocs(
89           query(
90             collection(db, "Calendar"),
91             where("DogID", "==", dog.id),
92             orderBy("DateOfEvent", "desc")
93           )
94         );
95         let markedDateArray = [];
96         let markedDateColorArray = [];
97         let tmpArray = [];
98       > snapshot.forEach((doc) => {···
109        });
110        setSavedEvents(tmpArray);
111
112        let counterColor = 0;
113        markedDateArray.forEach((day) => {
114          newDaysObject[day] = {
115            selected: true,
116            marked: true,
117            selectedColor: markedDateColorArray[counterColor],
118          };
119          ++counterColor;
120        });
121        setMarkedEventsSaved(newDaysObject);
122      } catch (e) {
123        console.log(e);
124      }
125    }
126    fetchData();
127  }, [isFocused]);
```

Obrázek 53 Načítání dat z databáze pro obrazovku Kalendář

Po načtení všech událostí, je nutná úprava data do formátu, který je pak možno poslat do objektu kalendáře, aby jej mohl zobrazit. Formát událostí pro kalendář je zobrazen na následujícím obrázku.

```
markedDates={{
  '2012-03-01': {selected: true, marked: true, selectedColor: 'blue'},
  '2012-03-02': {marked: true},
  '2012-03-03': {selected: true, marked: true, selectedColor: 'blue'}
}}
```

Obrázek 54 Formát Marked Dates

Prostřední část *snapshot.forEach(doc)* prochází všechny záznamy získané z databáze a upravuje datum událostí do požadovaného formátu pro poslední část, kde se dává dohromady datum a další informace o události jako je barva, jestli je událost označena a jestli je vybrána. *MarkedEventsSaved* jsou data která byla upravena do požadovaného formátu v předešlé části, a *markedDates* je atribut kalendáře, který pak datumy označí tečkou barvy události.

Datum události je vybráno pomocí kliknutí na konkrétní den v kalendáři pomocí funkce *onDayPress*. Kliknutím je také vyvoláno modální okno pro vytvoření události. Vytváření nové události obsluhuje metoda *handleNew()*, která mimo prostého poslání dat do databáze ještě musí upravit datum události do českého formátu pomocí knihovny *moment*.

```
129 const handleNew = async () => {
130   selectedEvent = true;
131   selectedColor = valueColor;
132
133   setShowModal(!showModal);
134   require("moment/locale/cs.js");
135   const normalFormatDate = moment(dateOfEvent).format("L");
136   const normalFormatTime = moment(timeOfEvent).format("HH:mm");
137   onChangeText_nazev("");
138   onChangeText_misto("");
139   onChangeText_popis("");
140
141   const collectionRef = collection(db, "Calendar");
142   const payload = { DogID: dog.id, Name: text_nazev, Description: text_popis, Place: text_misto,
143                     DateOfEvent: normalFormatDate, TimeOfEvent: normalFormatTime,
144                     SelectedEvent: selectedEvent, SelectedColor: selectedColor,
145   };
146   await addDoc(collectionRef, payload);
147 };
```

Obrázek 55 Vytváření nové události

Funkce *handleDeleteEvent()* je volána v případě, že uživatel chce smazat určitou událost. Tato událost je identifikována pomocí *event.id*.

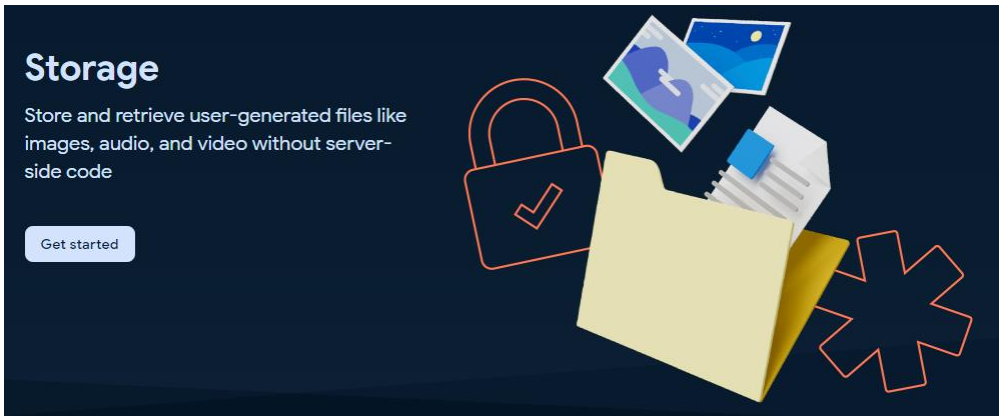
Jedním z míst, kde je potřeba rozlišovat mezi platformami iOS a Android, je uvnitř modálního okna pro vytvoření nové události. Když se zadává čas, je použit *DateTimePicker*, který má odlišné chování pro různé platformy.

Atribut display je část, která se chová různě v závislosti na platformě a z tohoto důvodu je zde rozcestník. V případě platformy iOS je výběr data zobrazen jako kolečko – jako je například v aplikaci Budík – a pro Android je zobrazen jako virtuální hodiny s ručičkou.

4.7.8 Dokumenty

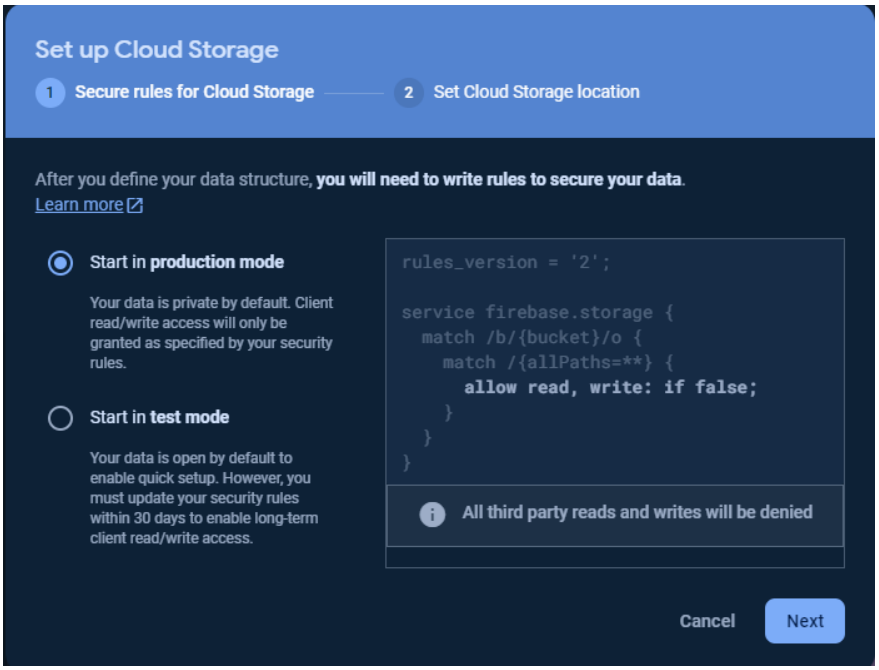
Obrazovka dokumentů slouží pro uživatele jako uložiště dokumentů a obrázku pro konkrétního pejska. Rozdělení podle jednotlivých pejsků je vhodné z důvodu přehlednosti, aby

uživatel nemusel hledat mezi vícero dokumenty různých pejsků. Uživatel zde může ukládat dokumenty a obrázky, které již má v mobilním zařízení. Ukládání zajišťuje dvě podobné funkce, jedna pro dokumenty a druhá pro fotky. Dokumenty i fotky se ukládají do Firebase Storage. Pro náš projekt je nezbytné nejdříve zpřístupnit Storage. Zprovoznění lze provést ve Firebase konzoli následovně.



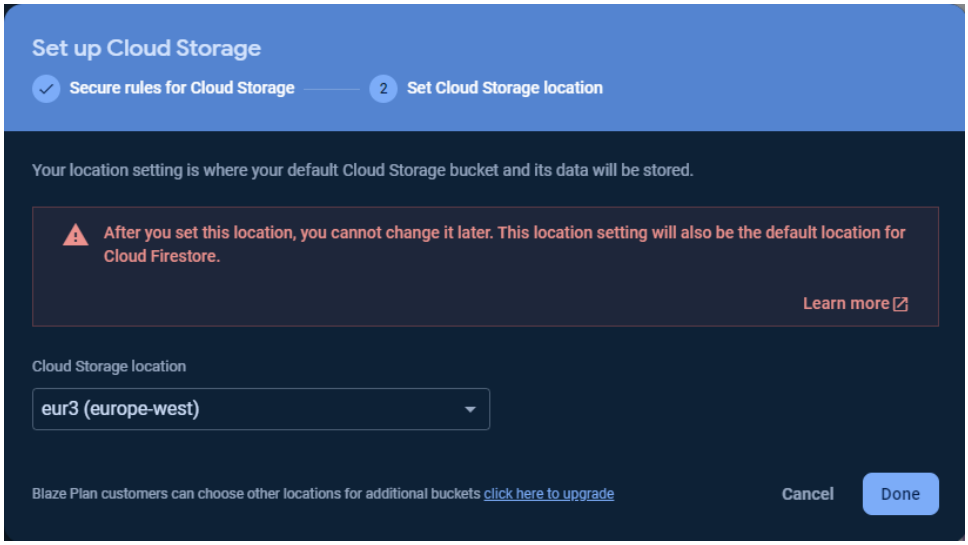
Obrázek 56 Firebase použití Storage

Lze nastavit jak testovací, tak produkční verzi uložiště. Jednotlivé verze se v zásadě ničím neliší, proto je vybraná produkční verze.



Obrázek 57 Firebase vytvoření Storage

Posledním krokem je výběr lokace uložiště. Na výběr je mnoho různých lokací, pro tento projekt je vybrána lokace *eur3(europe-west)*.



Obrázek 58 Firebase vytvoření Storage 2

Tímto je vytvořeno uložiště pro projekt, které lze používat pro dokumenty a další soubory, které je potřeba ukládat v naší aplikaci. Výhodou je možnost vytvářet různé složky pro různé dokumenty, anebo fotky dle potřeb aplikace a uživatelů.

```
const pickDocument = async () => {
  let result = await DocumentPicker.getDocumentAsync({});

  if (!result.canceled) {
    const blob = await new Promise((resolve, reject) => {
      const xhr = new XMLHttpRequest();
      xhr.onload = function () {
        resolve(xhr.response);
      };
      xhr.onerror = function (e) {
        reject(new TypeError("Network request failed"));
      };
      xhr.responseType = "blob";
      xhr.open("GET", result.uri, true);
      xhr.send(null);
    });
    const storage = getStorage();
    const refs = ref(storage, `userDocuments/${item.id}/${result.name+uuid.v4()}`);
    const metadata = { contentType: "document" };
    await uploadBytesResumable(refs, blob, metadata);
    blob.close();
  }
};
```

Obrázek 59 Vybrání dokumentu

Pro dokumenty je zavolán *DocumentPicker*, který zajišťuje přístup k dokumentům na telefonu. Pokud uživatel vybere dokument a výběr nebyl zrušen, je potřeba tento dokument převést na *Blob*, tedy binární data z dokumentu pomocí *XMLHttpRequestu*. Metoda *ref()* je použita k vytvoření cesty ke konkrétnímu umístnění uložení dokumentu, kde je využita funkce pro náhodné generování unikátního identifikátoru *uuid.v4()* knihovny *react-native-uuid*.

Do metadat se vloží typ ukládaných dat, v uvedeném případě typ dokument. Binární data jsou nahrána pomocí metody *uploadBytesResumable()* na požadované místo na server. Nakonec je nově nahraný dokument přidán na konec zobrazovaných dokumentu pomocí *setDocuments()*. Tímto uživatel okamžitě vidí nově nahraný dokument i na obrazovce dokumentů.

Obdobně funguje funkce pro nahrávaní fotek uživatelem. Rozdíl je pouze ve využití *ImagePicker* namísto *DocumentPicker*, a v nastavení metadat na formát *image/jpeg*.

Pokud uživatel již nahrál dokumenty nebo fotky do online uložiště, pak je nutné tyto dokumenty i fotky získat z uložiště a zobrazit na obrazovce dokumentů. Popsanou funkcionalitu zajišťuje hook *useEffect()*.

```
77  useEffect(() => {
78    setDocuments([]);
79    const storage = getStorage();
80    const documentsListRef = ref(storage, `userDocuments/${item.id}`);
81    listAll(documentsListRef).then((response) => {
82      response.items.forEach((document) => {
83        getDownloadURL(document).then((url) => {
84          setDocuments((prevDocuments) => [...prevDocuments, url]);
85        });
86      });
87    });
88  }, []);
```

Obrázek 60 Načítání dat ze Storage

Funkce nejprve nastaví proměnnou *documents* na prázdné pole pomocí funkce *setDocuments()*. Funkce *ref()* vytváří odkaz na adresář v online uložišti, který odpovídá místu uživatelských dokumentů a identifikátoru pro konkrétního pejska. Funkce *listAll()* vrací seznam všech objektů, respektive dokumentů v daném adresáři. Následně jsou procházeny všechny nalezené dokumenty v daném adresáři a pro každý dokument se získává URL pomocí funkce *getDownloadURL()*. Získaná URL adresa je poté přidávána do proměnné *documents* pomocí *setDocuments()*. Nakonec jsou všechny získané dokumenty zobrazeny pro uživatele.

Pro mazání jednotlivých dokumentů je využita funkce *onLongPress()* komponenty *TouchableOpacity*. Pomocí funkci *alert()* je uživatel varován o mazání dokumentu. Uživatel má na výběr, jestli chce dokument smazat, nebo ponechat.

4.7.9 Gamifikace – ocenění

Gamifikace je strategie, která aplikuje herní prvky, design a mechanismy mimo herní prostředí a hry. Cílem je zapojení, motivace a výkon uživatelů. Gamifikace využívá aspekty lidské povahy, které jsou běžně spojovány například s hraním her, aby přenesla tyto prvky do

neherního kontextu. Využívá jednu nebo více herních prvků jako jsou body, odměny, úrovně, výzvy, soutěže a vytváření cílů a jiné motivující mechanismy.

V případě vyvíjené aplikace je gamifikace řešena pomocí sbírání ocenění za plnění jednotlivých cviků a povelů. Uživateli dávají motivaci cvičit pejska a splnit všechny cviky. Přehled všech ocenění je na vlastní obrazovce. Jednotlivá ocenění, respektive jejich základ je uložen v databázové tabulce *Achievements*, kam je možné jednoduše přidávat další ocenění dle potřeby. Při vytvoření pejska se automaticky všechna ocenění vytvoří a jsou uložena do tabulky *Achievements_inst*. Jednotlivá ocenění se skládají z názvu a popisu.

Pokrok v oceněních lze sledovat pokaždé, když uživatel otevře obrazovku konkrétního pejska. Zde se zavolá funkce *AchievementController()*, kde se nachází logika kontrolující jednotlivé pokroky v oceněních. V kontroleru se nachází jedna hlavní funkce a jedna pomocná. Hlavní funkce získává data z databáze o cvičeních a povelech, přesněji o jejich plnění a dokončení. Poté pomocí jednoduchých pravidel kontroluje, zda je některé z ocenění splněno. Pokud ano, je přidáno do pomocného pole. Pokud toto pole po kontrole všech ocenění není prázdné, je volána pomocná funkce, ve které se kontroluje, zda ocenění již nebylo získáno. Pokud ne, je aktualizována hodnota, která vypovídá o konkrétním ocenění. Pomocná funkce by nemusela být implementována, načež se později kontroluje i to, jestli ocenění bylo již vyzvednuto uživatelem.

Níže na obrázcích je ukázka kódu, který obsluhuje základní cviky/povely pejska. První obrázek ukazuje získaní těchto cviků a informace o nich z databáze.

```
34  const collectionRefBasic = collection(db, "Basic_exercise_inst");
35  const queryBasic = query(collectionRefBasic, where("DogID", "==", dogID));
36  const snapshotBasic = await getDocs(queryBasic);
37  const basicSkills = snapshotBasic.docs.map((doc) => ({
38    ...doc.data(),
39    id: doc.id,
40  }));
```

Obrázek 61 Výběr dat z tabulky Basic_exercise_inst pro Achievement Controller

Poté se zkontroluje, kolik základních cviků/povelů již bylo splněno, a také celkové množství těchto cviků. Následně jsou pomocí jednoduchých pravidel kontrolována jednotlivá ocenění, jestli jsou splněna. Pokud jsou splněna tyto pravidla, jsou poslány do pomocné funkce.

```
87    let achievementsToUpdate = [];
88    if (completedBasics == 1) {
89      let achvID = setDataForAchievement(achievDoc, 1);
90      if (achvID != "") {
91        achievementsToUpdate.push(achvID);
92      }
93    }
94
95    if (completedBasics == 2) {
96      let achvID = setDataForAchievement(achievDoc, 2);
97      if (achvID != "") {
98        achievementsToUpdate.push(achvID);
99      }
100   }
101
102   if (completedBasics == maximumOfBasics) {
103     let achvID = setDataForAchievement(achievDoc, 3);
104     if (achvID != "") {
105       achievementsToUpdate.push(achvID);
106     }
107   }
```

Obrázek 62 Kontrola jednotlivých ocenění

V pomocné funkci *setDataForAchievement()* se kontroluje, zda konkrétní ocenění již nebylo splněno. Pokud ještě ocenění nebylo splněno, vrátí funkce identifikátor konkrétního ocenění.

```
12  function setDataForAchievement(
13    achievDocs,
14    numberOfAchiev
15  ) {
16    let returnString = "";
17    achievDocs.forEach((achievDoc) => {
18      if (
19        Number(achievDoc.NoAchievement) == Number(numberOfAchiev) &&
20        !achievDoc.isAvailable
21      ) {
22        returnString = achievDoc.id;
23      }
24    });
25    return returnString;
26  }
```

Obrázek 63 Kontrola, zda je konkrétní ocenění splněné a má být zobrazeno

Na obrazovce pro konkrétního pejska pracuje funkce, která kontroluje, zda není nějaké ocenění ve stavu pro vyzvednutí. Funkce je reprezentována pomocí hooku *useEffect()* pokaždé, kdy je obrazovka pejska zobrazována. Popsané je zajištěno pomocí hooku *useIsFocused()* a proměnnou *isFocused*. Za začátku je nastavena proměnná *isAchievementAvailable* na hodnotu *false* pomocí *setIsAchievementAvailable()*. Pokud je hodnota pro tuto proměnnou *true*, pak je zobrazen červený symbol vykřičníku u tlačítka pro obrazovku ocenění. Zároveň to znamená, že některá ocenění ještě nebyla zobrazena.

```
118   useEffect(() => {
119     setIsAchievementAvailable(false);
120     async function fetchAchievements() {
121       AchievementController(dogInfo.id);
122       const collectionRefAchievement = collection(db, "Achievements_inst");
123       const queryAchievement = query(
124         collectionRefAchievement,
125         where("DogID", "==", dogInfo.id)
126       );
127       const snapshotAchievement = await getDocs(queryAchievement);
128       snapshotAchievement.forEach((doc) => {
129         if (doc.data().isAvailable && !doc.data().alreadyShown)
130           setIsAchievementAvailable(true);
131       });
132     }
133     fetchAchievements();
134   }, [isFocused]);
```

Obrázek 64 Kontrola, zda je nějaké ocenění k vyzvednutí

Samotná obrazovka zobrazující ocenění dokáže zobrazit dva různé obsahy. První obsah je přehledem všech ocenění. Druhý je zobrazení doposud nevyzvednutých ocenění. To, který obsah bude vykreslen se řídí pomocí podmíněného vykreslení a proměnné *showAchievement*. Samotné vykreslení nevyzvednutého ocenění se skládá z názvu ocenění, popisu, za co je toto konkrétní ocenění získáno, z tlačítka k vyzvednutí a uzavření ocenění a z animací. Zabezpečení animace probíhá pomocí *LottieView*. Pro ocenění byla zvolena animace s vizualizací konfet.

```
44   useEffect(() => {
45     async function fetchData() {
46       const collectionRef = collection(db, "Achievements_inst");
47       const q = query(collectionRef, where("DogID", "==", dog.id));
48       const snapshot = await getDocs(q);
49       const data = snapshot.docs.map((doc) => doc.data());
50       setAchievements(data);
51       //console.log(data);
52       let counter = 1;
53       snapshot.forEach((doc) => {
54         if (
55           doc.data().isAvailable &&
56           !doc.data().alreadyShown &&
57           counter == 1
58         ) {
59           setCurrAchievName(doc.data().Name);
60           setCurrAchievIsAvailable(doc.data().isAvailable);
61           setCurrAchievAlreadyShown(doc.data().alreadyShown);
62           setCurrAchievDesc(doc.data().Description);
63           setCurrAchievId(doc.id);
64           setShowAchievement(true);
65           counter++;
66         } else if (doc.data().isAvailable && !doc.data().alreadyShown) {
67           setAchievementAvailable((prevState) => [...prevState, doc.data()]);
68         }
69       });
70     }
71     fetchData();
72   }, []);
```

Obrázek 65 Načítání a příprava dat pro obrazovku Achievements

Zobrazena funkce získá všechny ocenění z databáze a uloží do pole *achievements* pro zobrazení všech ocenění v seznamu. V dalším kroku se prochází jednotlivé záznamy z databáze. První záznam, pro který platí, že je dostupný a nebyl ještě zobrazen, se uloží do proměnných. Proměnné jsou použity u vyzvednutí ocenění, a proměnná *showAchievement* se nastaví na hodnotu *true* pro zobrazení vyzvednutí ocenění. Další ocenění, které jsou k vyzvednutí se uloží do pole *achievementAvailable,* pro postupné vyzvedávaní. Po stisku tlačítka pro ukončení vyzvednutí ocenění se zavolá funkce *handleCloseAchievement()*, která nastaví pro aktuálně vyzvedávané ocenění atribut *alreadyShown* na hodnotu *true*. Poté, pokud je další ocenění k vyzvednutí, vybere se z pole dostupných ocenění následující prvek a nastaví se jednotlivé proměnné. Do pomocného pole se nakopíruje zbytek ocenění k vyzvednutí. Takto se pokračuje, dokud jsou další ocenění k vyzvednutí. Pokud již není další ocenění k vyzvednutí, zobrazí se seznam všech ocenění.

```
107  const handleCloseAchievement = async () => {
108    setShowAchievement(false);
109    const db = getFirestore();
110    await updateDoc(doc(db, "Achievements_inst", currAchievId), {
111      alreadyShown: true,
112    });
113    if (achievementAvailable.length != 0) {
114      setShowAchievement(true);
115      let counter = 1;
116      achievementAvailable.forEach((item) => {
117        if (counter == 1) {
118          setCurrAchievName(item.Name);
119          setCurrAchievIsAvailable(item.isAvailable);
120          setCurrAchievAlreadyShown(item.alreadyShown);
121          setCurrAchievDesc(item.Description);
122          setCurrAchievId(item.id);
123          counter++;
124        } else {
125          setAchievementsAvailableTmp((prevState) => [...prevState, item]);
126        }
127      });
128    } else {
129      setCurrAchievName();
130      setCurrAchievIsAvailable();
131      setCurrAchievAlreadyShown();
132      setCurrAchievId();
133    }
134    setAchievementAvailable([]);
135    setAchievementAvailable(achievementAvailableTmp);
136    setAchievementsAvailableTmp([]);
137  };
```

Obrázek 66 Funkce pro získání ocenění a načtení dalšího

Přehled všech ocenění:

- První krok – jeden základní povel splněn
- Jen tak dále – dva základní povely splněny
- Zkušený začátečník – splněny všechny základní cviky
- První složitější krok – splněn jeden ze složitějších cviků
- Je to v kapse – splnění všech složitějších cviků
- Sběratel – splnění všech základních a složitějších cviků
- První vlastní cvik – vytvoření prvního vlastního cviku
- Vytvoření 5 vlastních cviků
- Zkušený pejsek – splnění všech dostupných cviků, jak přednastavených, tak vlastních

4.7.10 Tréninkový diář

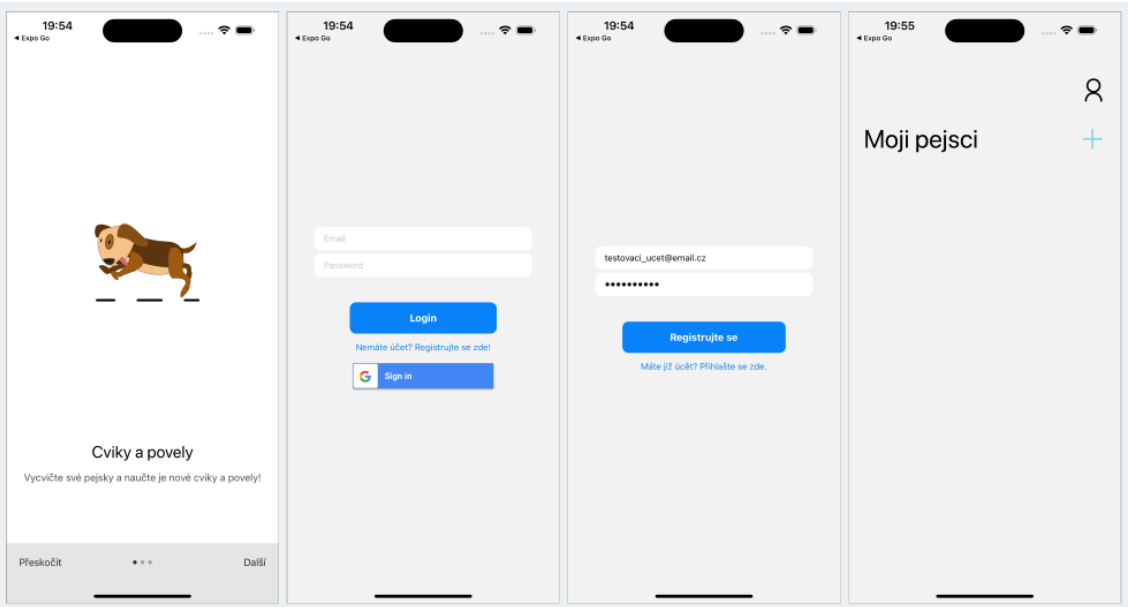
Tréninkový diář slouží uživateli pro přehled jednotlivých tréninkových relací. Diář je složen ze dvou obrazovek. První je přehled jednotlivých tréninkových relací, který je seřazen od nejnovějšího po nejstarší. Každá jedna tréninková relace je složena z popisu, který udává, co se cvičilo a jaký byl průběh, datumu cvičení a jednotlivých povelů a cviků, které byly cvičeny. Na druhou obrazovku se uživatel dostane pomocí tlačítka ve tvaru plus. Zde uživatel vytváří jednotlivé tréninkové relace. Uživatel vypíše podrobnosti ke konkrétní relaci, datum tréninku, a pak výběr cvičených cviků a povelů. Zde se uživateli vykreslí všechny cviky, které mu jsou k dispozici pro konkrétního pejska. Jednotlivý cvik může vybrat pomocí jednoduchého switch prvku. Pak kliknutím na tlačítko „Vytvořit“ je relace vytvořena a nahrána do databázové tabulky *Training_diary*.

4.7.11 Zdravotní rady

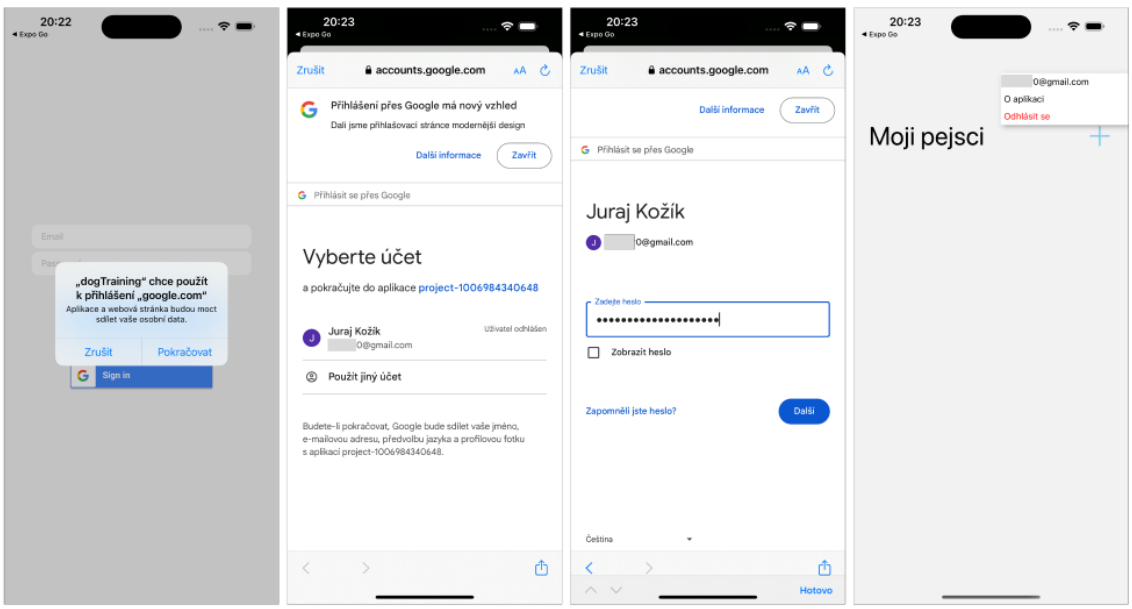
Obrazovka pro zdravotní tipy využívá jednoduchosti a flexibility obrazovky onboardingu a její technologii. Zdravotní rady obsahují výčet problémů, se kterými se může majitel pejska setkat v průběhu výcviku, např. přehřátí, kulhání apod. Zdravotní rady obsahují i návod, jestli a jak je lze různé situace řešit doma bez veterináře a kdy je v případě nezlepšení na místě návštěvu u lékaře neodkládat.

4.8 Validace a testování aplikace

Pro testování aplikace bude použit emulátor iOS zařízení. Pro validaci bude použit emulátor pro Android. V části testování bude předvedena funkcionalita aplikace na platformě iOS. Bude předvedeno například přihlášení pomocí emailu a hesla, přihlášení pomocí *Google SignIn*, vytvoření nového pejska a další funkcionality implementovány v předchozích částech. V další části bude validace aplikace v prostředí Android. Pro obě platformy byl vytvořen development build pomocí Expo frameworku. Takto vytvořen vývojářský build lze stáhnout a nainstalovat na emulátor iOS a Android. Poté ji lze pouštět jako každou jinou aplikaci.

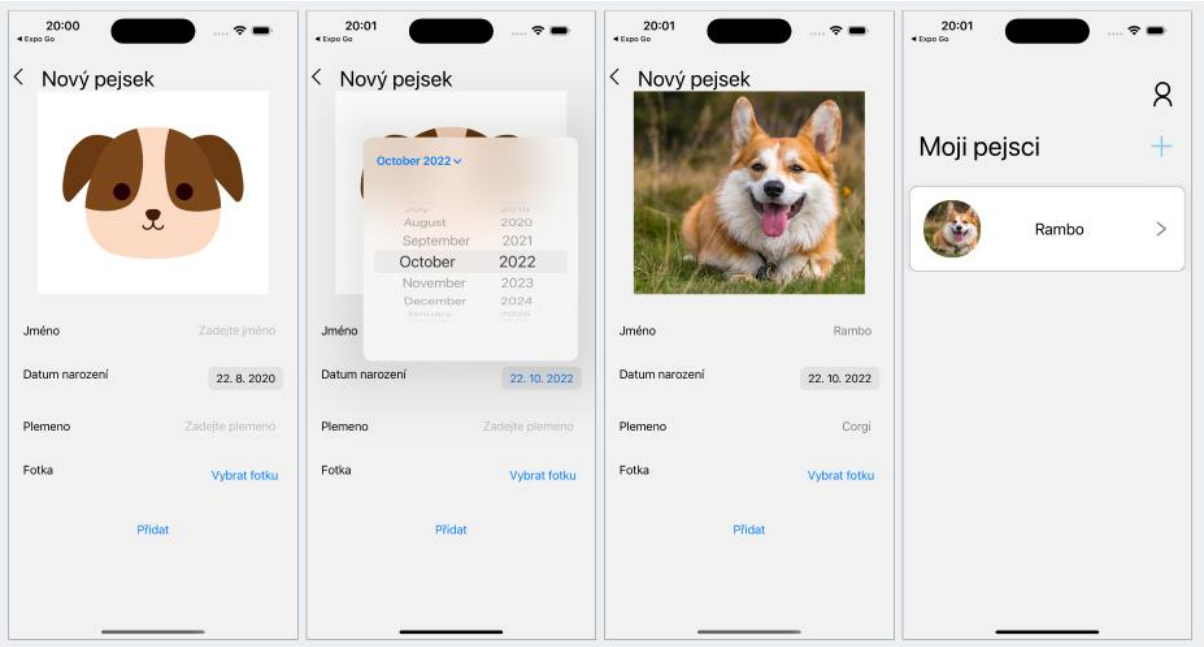


Obrázek 67 Testování – onboarding, přihlášení pomocí emailu

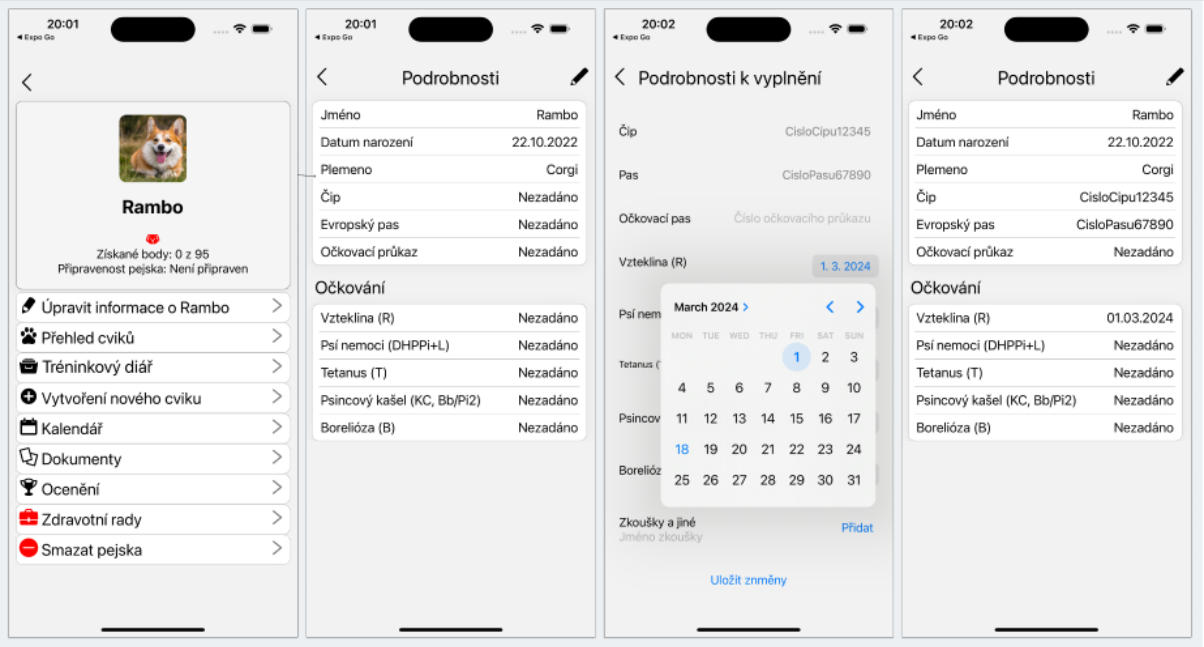


Obrázek 68 Testování – přihlášení pomocí Google SignIn

Výše uvedené obrázky zobrazují první obrazovku onboardingu. Následným krokem je přihlášení prostřednictvím e-mailu a hesla. Dále je zobrazena možnost přihlásit se pomocí funkce *Google SignIn*.

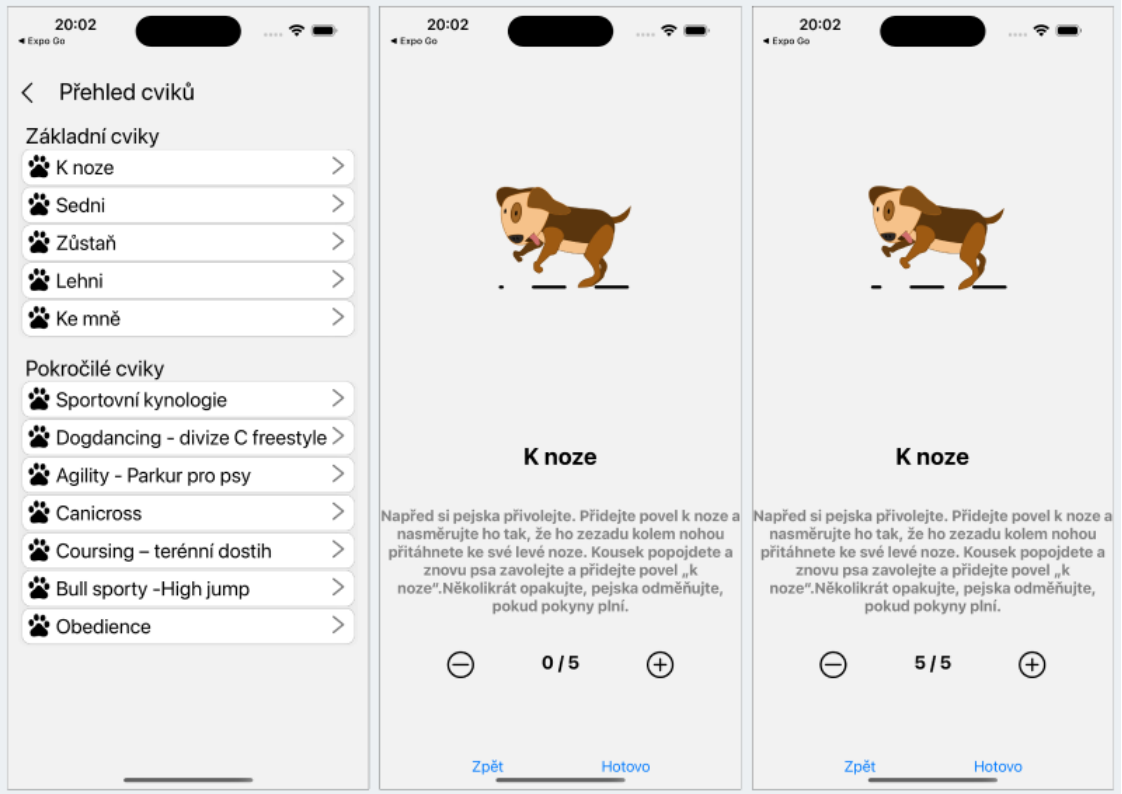


Obrázek 69 Testování – vytvoření nového pejska

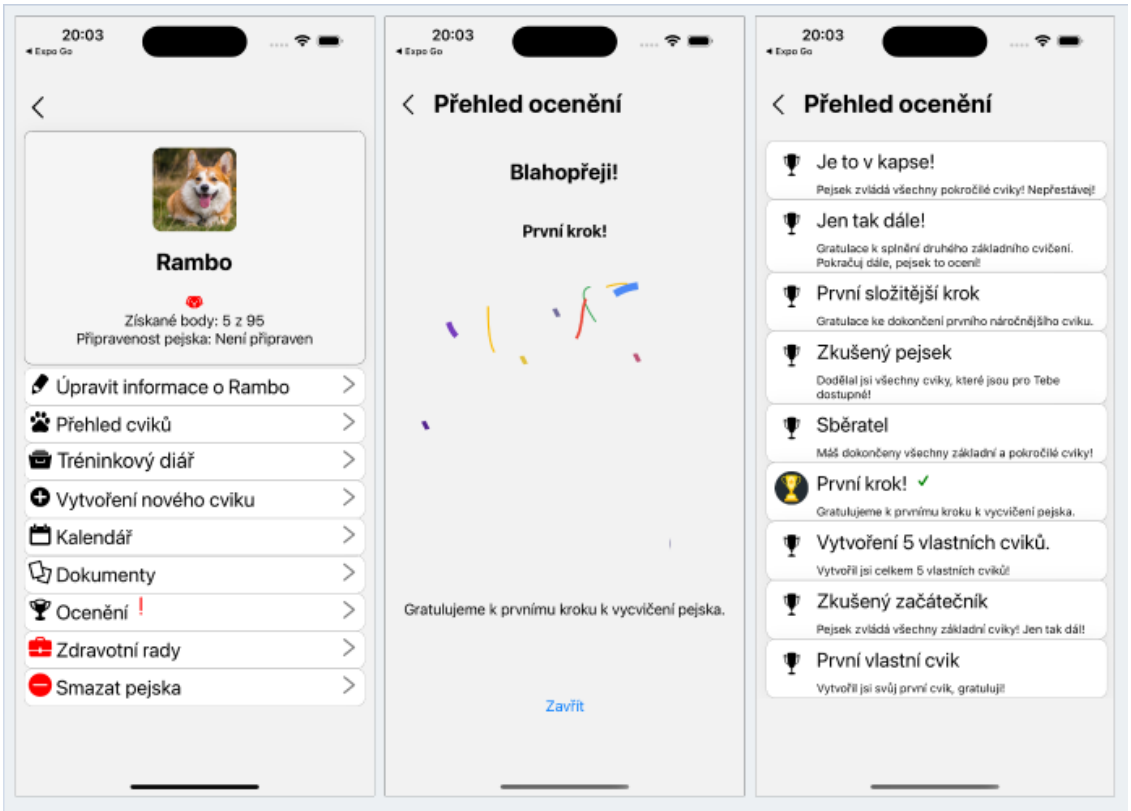


Obrázek 70 Testování – aktualizace podrobností

Na předchozích obrázcích je vidět aktivitu vytvoření pejska a následné vyplnění dalších podrobností ve formě čísla čipu, evropského pasu a očkování na vzteklinu. Obrazovka podrobnosti je dostupná po kliknutí na čtverec s informacemi o pejskovi.



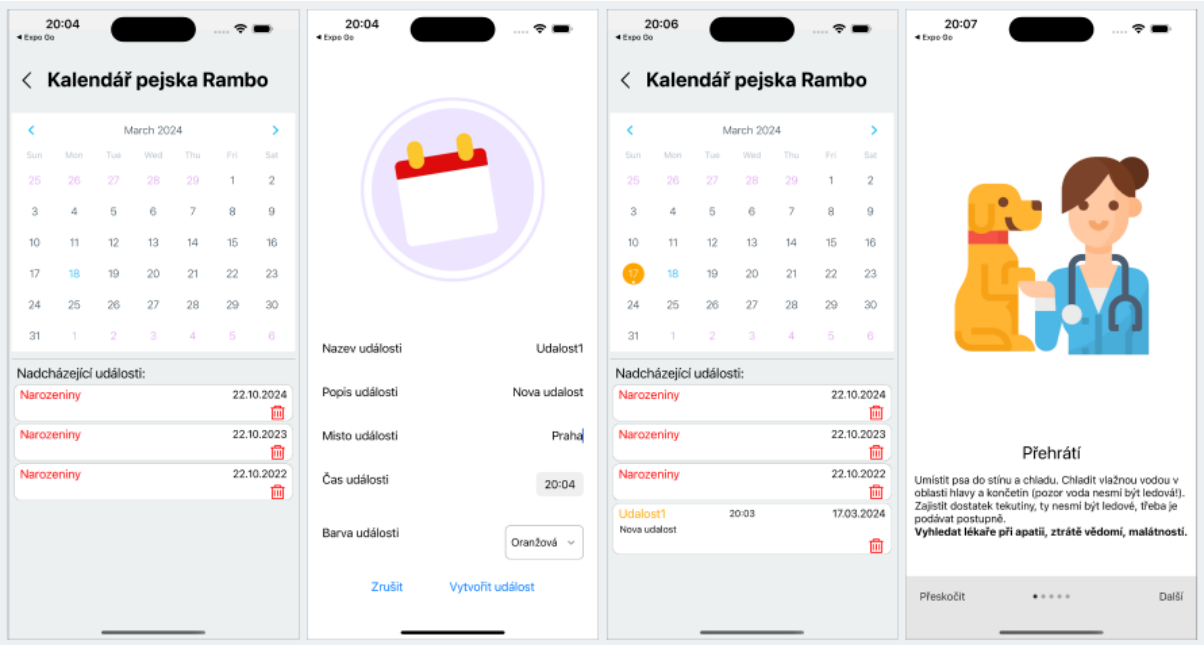
Obrázek 71 Testování – přehled cviků a přidání bodů



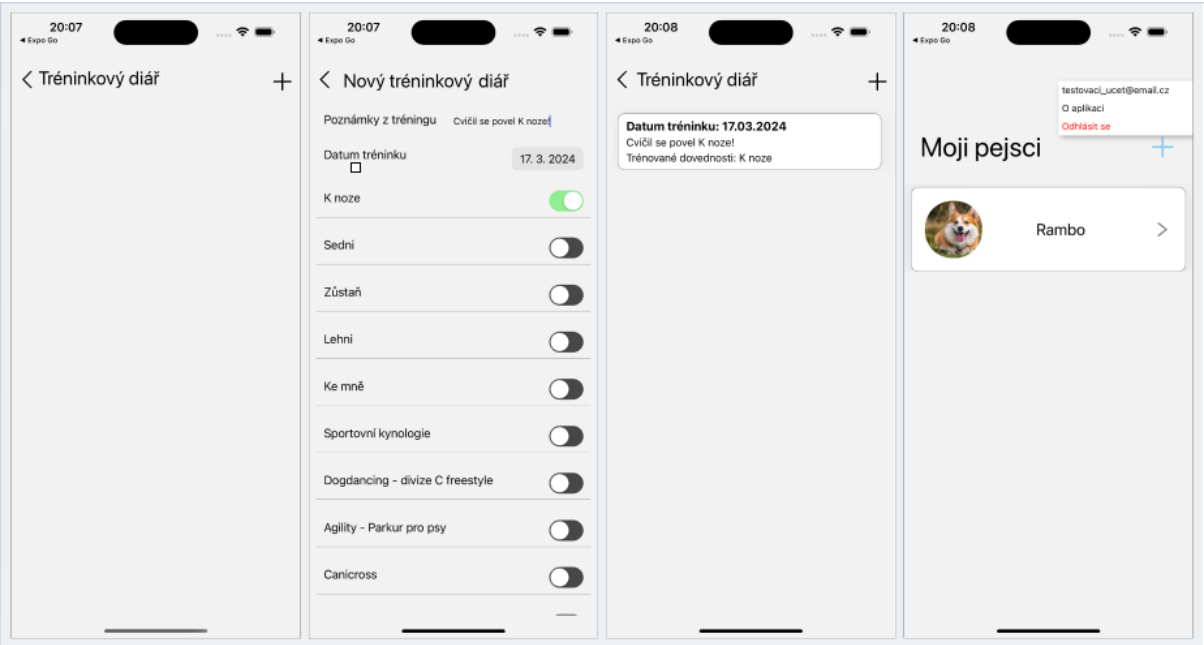
Obrázek 72 Testování – vyzvednutí ocenění

Výše je vidět, jak funguje prvek gamifikace. Příklad znázorňuje získání ocenění za splnění jednoho základního povelu za plný počet možných bodů. Pokud je ocenění dostupné, na obrazovce pro konkrétního pejska je vidět červený vykřičník u ocenění.

Obrazovky níže zobrazují kalendář pejska a ukázku zdravotních rad. Níže je zobrazen tréninkový diář, kam si uživatel má možnost zapisovat jednotlivé tréninkové relace.

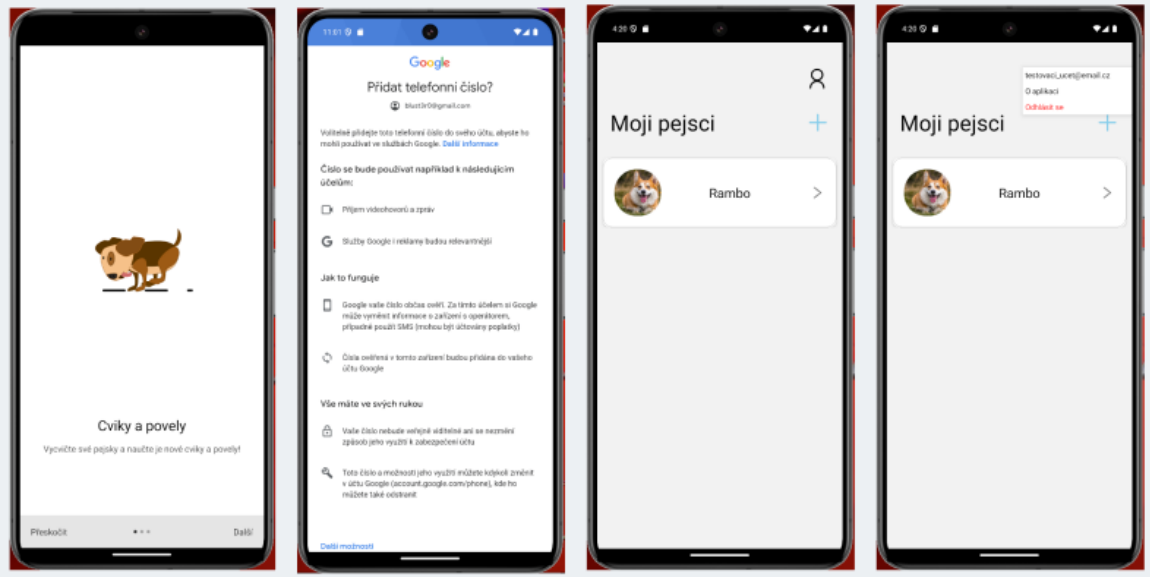


Obrázek 73 Testování – kalendář, vytvoření nové události a zdravotní rady

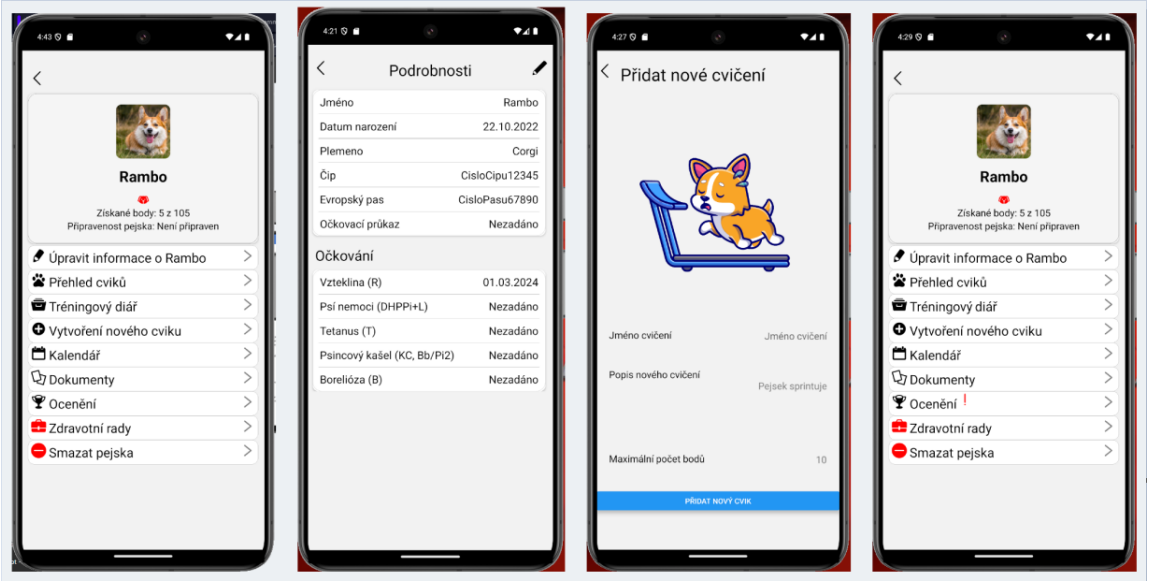


Obrázek 74 Testování – tréninkový diář a hlavní obrazovka

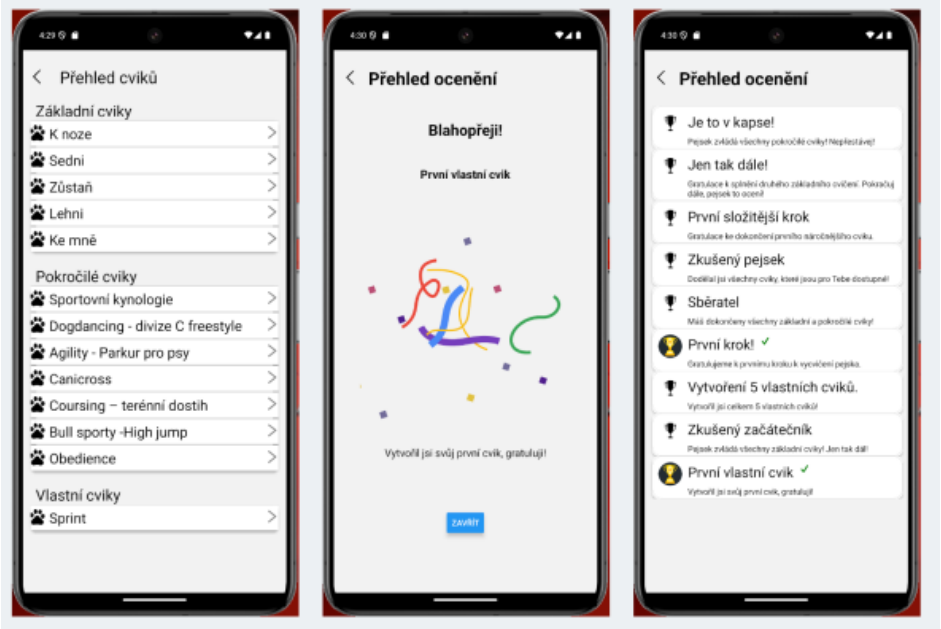
Validace pro Android spočívá v procházení aplikace a vyzkoušení alespoň jedné její funkcionality. Na obrázku níže je zobrazeno vytvoření nového cviku a vyzvednutí ocenění za vytvoření prvního cviku.



Obrázek 75 Validace – přihlášení



Obrázek 76 Validace – podrobnosti, vytvoření nového cviku



Obrázek 77 Validace – přehled cviků, vyzvednutí ocenění

Po testování a ověření mobilní aplikace na platformách iOS a Android byly vyvozeny následující klíčové závěry. Základní funkce aplikace, jako je registrace uživatele, přihlášení a načítání dat, fungovaly podle očekávání. Uživatelské rozhraní (UI) bylo intuitivní, s konzistentními designovými prvky na všech platformách. Doba načítání se pohybovala v přijatelných mezích, což zajišťovalo bezproblémový uživatelský zážitek. Pro konkrétní konfigurace zařízení bylo nutné provést drobné úpravy.

5 Zhodnocení výsledků

Výsledkem práce je funkční multiplatformní mobilní aplikace pro výcvik psů, která funguje jak na platformách Android, tak i iOS. Aplikace byla vyvinuta pomocí technologií React Natvie JS a Expo frameworku. React Native je z pohledu vývoje zamýšlené aplikace vhodně použitým nástrojem, který umožňuje poměrně uživatelsky nenáročnou práci s kódem. Platforma Firebase poskytla pro aplikaci databázi, uložiště pro dokumenty a službu autentizace uživatelů. Expo framework zajistil včasné testování. Testování probíhalo na mobilním zařízení s iOS, nicméně po použití nativních prvků bylo nutné vytvořit vývojářský build aplikace a testovat ji dále na simulátorech.

V průběhu vývoje aplikace byla vyžadována aktualizace React Native a Expo frameworku, co způsobilo celkovou nefunkčnost aplikace. Obecně při velkých aktualizacích React Native dochází ke znefunkčnění knihoven. Přizpůsobení knihoven v takovém případě může trvat až několik dní. Následná oprava ve vyvíjené aplikaci proběhla například za pomoci aktualizací některých balíčku použitých knihoven. V logu chyby bylo potřeba najít konkrétní knihovnu, která problém způsobila a zajistit její opravu pomocí příkazu.

Vyvinuta aplikace poskytuje uživateli možnost sledování průběhu výcviku psa pomocí předdefinovaných cviků. Zároveň si uživatel dle potřeby může vytvářet cviky vlastní. Pomocí tréninkových deníku uživatel sleduje průběh výcviku, zaznamenává, kdy byl výcvik a jaké cviky a povely byly cvičeny. Aplikace také disponuje gamifikací formou ocenění, které uživatel získává za plnění jednotlivých cviků a vytváření vlastních cviků.

I když pracuji jako vývojář s prací v React Native jsem zatím neměl zkušenost a vzhledem k nižší zkušeností s vývojem podobné mobilní aplikace oceňuji uživatelsky nenáročné praktické použití React Native. Práci s tímto frameworkem ulehčuje i dostupnost velkého množství podkladů pro vývoj. Jelikož jsem s vývojem mobilních aplikací neměl žádnou zkušenost, považuji React Native za vhodný a adekvátní nástroj, který mi poskytnul vhled do této problematiky.

Kód aplikace je dostupný ve veřejném repositáři Githubu na adrese: https://github.com/jurajkozik/DogsJournal.

6 Závěr

Cílem práce bylo navrhnout a vyvinout mobilní aplikaci pro platformy Android a iOS. Pro vývoj byl použit framework React Native. Jednotlivé kroky návrhu a vývoje mobilní aplikace od její analýzy po její implementaci byly podrobně popsány a vysvětleny v hlavní části práce. Teoretická část práce byla soustředěna na samotné fungování frameworku React Native, před a po změně vnitřního fungování frameworku, tedy po nahrazení technologie Bridge. Návrh, konfigurace a implementace aplikace byly popsány v dalším textu práce s důrazem na implementaci. Kapitola popisující implementaci aplikace může zároveň sloužit jako její dokumentace. Aplikace byla definována pomocí analýzy požadavků, které určily předpoklady funkčnosti aplikace. Tvorba aplikace probíhala na podkladě vlastní zkušenosti a dostupné dokumentace pro vývoj v React Native.

Výstupem práce je funkční aplikace pro výcvik psů. I přes to, že byla aplikace vyvinuta do funkční podoby, existují možnosti a výzvy pro další vývoj. V budoucnu lze zvážit rozšíření podpory na další platformy, například na tabletová zařízení. Navíc je možné zkoumat možnosti integrovat moderní technologie, jako je strojové učení nebo AI, pro lepší personalizaci a automatické vytváření tréninkových plánů.

Ve využití React Native a vývoje mobilních aplikací vidím do budoucna možný potenciál pro vlastní praxi, nakolik jsem už delší dobu přemýšlel o vývoji mobilních aplikací. I v budoucnu bych se chtěl věnovat vývoji mobilních aplikací pro vlastní potřeby nebo v rámci praxe.

7 Seznam použitých zdrojů

1. Quick Start [online]. [cit. 2024-03-13]. Dostupné z: https://react.dev/learn

2. Introduction [online]. [cit. 2024-03-13]. Dostupné z: https://reactnative.dev/docs/getting-started

3. HAVERBEKE, Marijn. Eloquent JavaScript [online]. 2024 [cit. 2024-03-14]. Dostupné z: https://eloquentjavascript.net

4. Tutorial: Introduction [online]. [cit. 2024-03-14]. Dostupné z: https://docs.expo.dev/overview/

5. Introduction [online]. [cit. 2024-03-14]. Dostupné z: https://reactnative.dev/docs/getting-started

6. KOSMAL, Jakub. How does React Native work? Understanding the architecture [online]. 2022-11-11 [cit. 2024-03-14]. Dostupné z: https://medium.com/front-end-weekly/how-does-react-native-work-understanding-the-architecture-d9d714e402e0

7. BALODI, Karan. Javascript Interface(JSI): Overview and the need for re-architecture of react native [online]. 2022-11-25 [cit. 2024-03-14]. Dostupné z: https://medium.com/@karanbalodi/javascript-interface-jsi-overview-and-need-for-re-architecture-of-react-native-6ab6a7b3f660

8. Writing Markup with JSX [online]. [cit. 2024-03-14]. Dostupné z: https://react.dev/learn/writing-markup-with-jsx

9. VAILSHERY, Lionel Sujay. Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2022 [online]. 2023-06-01 [cit. 2024-03-14]. Dostupné z: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

10. Flutter architectural overview [online]. [cit. 2024-03-14]. Dostupné z: https://docs.flutter.dev/resources/architectural-overview

11. Introduction to Dart [online]. [cit. 2024-03-14]. Dostupné z: https://docs.flutter.dev/resources/architectural-overview

12. Introduction [online]. [cit. 2024-03-14]. Dostupné z: https://docs.flutterflow.io

13. What is Xamarin? [online]. 2022-09-20 [cit. 2024-03-14]. Dostupné z: https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

14. Introduction [online]. 2024-02-23 [cit. 2024-03-14]. Dostupné z: https://docs.nativescript.org

15. KHOROSHULIA, Stanislav. React Native App Development Guide: Challenges and Best Practices [online]. 2024-01-18 [cit. 2024-03-14]. Dostupné z: https://mobidev.biz/blog/react-native-app-development-guide

16. KARANPURIYA, Harshal. NativeScript vs React Native: Which is the Best Platform To Use? [online]. 2023-07-13 [cit. 2024-03-14]. Dostupné z: https://www.linkedin.com/pulse/nativescript-vs-react-native-which-best-platform-use-karanpuriya

17. ALCANJA, Daniel. Xamarin vs. React Native: Which Software To Choose in 2024? [online]. 2022-02-28 [cit. 2024-03-14]. Dostupné z: https://www.trio.dev/react-native/resources/xamarin-vs-react-native

18. ARCITECH. Xamarin vs. React Native: A Comprehensive Comparison of Cross-Platform Frameworks [online]. 2023-07-17 [cit. 2024-03-14]. Dostupné z: https://www.linkedin.com/pulse/xamarin-vs-react-native-comprehensive-comparison-cross-platform

19. KHAN, Samra. Flutter vs. React Native vs. Native: A Comprehensive Comparison for Mobile App Development [online]. 2023 [cit. 2024-03-14]. Dostupné z: https://medium.com/@samra.sajjad0001/flutter-vs-react-native-vs-native-a-comprehensive-comparison-for-mobile-app-development-601b09e2fa56

20. IT CREATIVE LABS. Flutter vs. React Native in 2023: Which is Better for Mobile App Development? [online]. 2023-07-31 [cit. 2024-03-14]. Dostupné z: https://www.linkedin.com/pulse/flutter-vs-react-native-2023-which-better-mobile-app

21. AIL, Vijit. What to learn before learning React Native [online]. 2022-10-10 [cit. 2024-03-14]. Dostupné z: https://blog.logrocket.com/what-learn-before-learning-react-native/

22. Understand Firebase projects [online]. [cit. 2024-03-14]. Dostupné z: https://firebase.google.com/docs/projects/learn-more

23. Learn more about Web and Firebase [online]. [cit. 2024-03-14]. Dostupné z: https://firebase.google.com/docs/web/learn-more#config-object

24. Výcvik štěněte: jak a kdy s ním začít? [online]. 2022-05-12 [cit. 2024-03-14]. Dostupné z: https://www.czvz.cz/vycvik-psa-jak-a-kdy-s-nim-zacit?fbclid=IwAR1bXuDDrBmhLi8jUd57VLJ_NAFEOm5OcyN7UCv8BpUUpQer_hp6a-KVj9E

25. Základní pravidla pro trénink [online]. 2022, 12.05.2022 [cit. 2024-03-14]. Dostupné z: https://skolavycvikupsu.cz/zakladni-pravidla-pro-trenink/?fbclid=IwAR02DLXwMbHcvcMkfs1BqsUmP3XPj8S4v941WWczA4F7zyHkNylBt9f19cY

26. DUBOVÝ, Jan. VÝCVIK PSA – ÚVOD DO VÝCVIKU – PUBLIKACE [online]. [cit. 2024-03-14]. Dostupné z: https://kynologie-zbraslav.cz/2015/12/16/vycvik-psa-uvod-do-vycviku-publikace/?fbclid=IwAR1Tf-5RkWt24ifPjaW4K6kxef4Z_tuOcPf6AoQDTa6RS6zUmI0SUdQk5Zc

27. HALDIYA, Prankur. Top React Native IDEs and Editors in 2024 [online]. 2023-09-21 [cit. 2024-03-14]. Dostupné z: https://ripenapps.com/blog/top-react-native-ides-and-editors/

28. INGRAM, Samantha. The Thumb Zone: Designing For Mobile Users [online]. 2016-09-19 [cit. 2024-03-14]. Dostupné z: https://www.smashingmagazine.com/2016/09/the-thumb-zone-designing-for-mobile-users/

29. Facebook Messenger UI Screens [online]. [cit. 2024-03-14]. Dostupné z: https://www.figma.com/community/file/874577850804632750

30. Creative tools meet the internet. [online]. [cit. 2024-03-14]. Dostupné z: https://www.figma.com/about/

31. Create your first app [online]. [cit. 2024-03-14]. Dostupné z: https://docs.expo.dev/tutorial/create-your-first-app/

32. Jak vypočítat věk psa? [online]. 2021 [cit. 2024-03-14]. Dostupné z: https://www.peliskydog.cz/blog/jak-vypocitat-vek-psa/

8 Seznam obrázků