VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO SLEDOVÁNÍ STAVU KOMPONENT JÍZDNÍHO KOLA

MOBILE APP FOR MONITORING THE CONDITION OF BIKE PARTS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JOSEF KOTOUN**
AUTHOR

VEDOUCÍ PRÁCE **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Kotoun Josef**
Program: Informační technologie
Název: **Mobilní aplikace pro sledování stavu komponent jízdního kola**
Mobile App for Monitoring the Condition of Bike Parts
Kategorie: Uživatelská rozhraní

Zadání:

1. Seznamte se s problematikou návrhu a vývoje mobilních aplikací.
2. Prostudujte a analyzujte aplikace podobné navrhovanému řešení, identifikujte prostor pro zlepšení.
3. Navrhněte mobilní aplikaci pro monitorování stavu komponent jízdního kola.
4. Prototypujte aplikaci a testujte jednotlivé části uživatelského rozhraní s uživateli.
5. Iterativně vylepšujte vytvořenou aplikaci na základě uživatelského testování.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Tidwell et al.: Designing Interfaces: Patterns for Effective Interaction Design, O'Reilly, 2020
- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299
- Joel Marsh: UX for Beginners: A Crash Course in 100 Short Lessons, O'Reilly 2016
- Jonathan Lebensold: React Native Cookbook: Bringing the Web to Native Platforms, O'Reilly 2018

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Práce se zabývá tvorbou mobilní aplikace, která umožňuje sledovat stav komponent jízdního kola. Aplikace je implementována v Reactu Native, pro uchování dat a autentizaci je využito služeb Firebase. Výsledné řešení poskytuje uživateli možnost zaznamenávání počtu najetých kilometrů a času v provozu u jednotlivých komponent i kol a zaznamenávání stavu opotřebení a provedených servisních úkonů u komponent. Najeté kilometry a čas v provozu jsou získávány ze záznamů o jízdách na kolech, které je uživateli umožněno zadávat ručně pomocí formuláře, nebo je synchronizovat se svým Strava účtem. Výsledná aplikace je k dispozici na Google Play pod názvem Bike Components Manager.

Abstract

The thesis deals with the creation of a mobile application that allows to monitor the condition of bicycle components. The application is implemented in React Native, Firebase services are used for data storage and authentication. The resulting solution provides the user with the ability to record mileage and ride time for individual components and bikes, and to record the state of wear and services performed on components. Mileage and ride time are obtained from the ride records, which the user can enter manually using a form or sync with their Strava account. The resulting app is available on Google Play under the name Bike Components Manager.

Klíčová slova

mobilní aplikace, React Native, Firebase, uživatelské testování, Strava, správa komponent jízdního kola

Keywords

mobile app, React Native, Firebase, usability testing, Strava, bike components management

Citace

KOTOUN, Josef. *Mobilní aplikace pro sledování stavu komponent jízdního kola.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Mobilní aplikace pro sledování stavu komponent jízdního kola

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. a uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Josef Kotoun
9. května 2022

Poděkování

Rád bych poděkoval svému vedoucímu práce panu prof. Ing. Adamu Heroutovi, Ph.D. za odborné vedení a cenné rady při tvorbě této práce. Dále bych rád poděkoval přátelům, kteří se podíleli na testování aplikace.

Obsah

Kapitola 1

Úvod

Jedním z nejdůležitějších údajů o opotřebení komponent jízdního kola jsou najeté kilometry a hodiny v provozu. Pokud by ovšem chtěl cyklista tento údaj zjistit, musel by si všechny jízdy zapisovat, případně si je zaznamenávat pomocí nějaké aplikace, a vypočítat z nich tyto hodnoty. To ovšem nemusí být tak jednoduché, pokud například cyklista tyto komponenty mezi jízdami různě měnil, servisoval a podobně.

Cílem této práce je vytvořit mobilní aplikaci, která usnadňuje správu stavu komponent jízdního kola. V aplikaci bude umožněno spravovat kola a k nim přiřazené komponenty. Ke každé komponentě bude možno přidávat záznamy o provedených servisních úkonech, díky kterým může uživatel jednoduše zjistit, kolik ho stála údržba dané komponenty. Dále aplikace bude umožňovat přidání záznamu o stavu komponenty při aktuálním nájezdu kilometrů. Celkový stav najetých kilometrů a hodin v provozu komponent a kol bude počítán na základě záznamů o jízdách na kolech. Aplikace bude kromě možnosti manuálního zaznamenání jízdy prostřednictvím formuláře také poskytovat možnost synchronizace jízd s aplikací Strava, která je jednou z nejpoužívanějších aplikací pro zaznamenávání sportovních aktivit. Pokud bude mít uživatel v aplikaci Strava k aktivitě přiřazeno kolo, najeté kilometry a čas jízdy se započítají ke kolu importovanému ze Stravy a ke komponentám, které byly v čase jízdy k tomuto kolu přiřazeny.

V kapitole číslo 2 je popsána bližší specifikace cílové aplikace a srovnání s existujícím řešením. Kapitola číslo 3 se věnuje seznámení s použitými technologiemi pro návrh, vývoj a testování. Kapitola číslo 4 popisuje proces tvorby návrhu, testování návrhu s uživateli a jeho postupné vylepšování. Kapitola číslo 5 se věnuje implementaci aplikace. Kapitola číslo 6 popisuje testování finální aplikace a vydání na Google Play.

Kapitola 2

Sledování stavu komponent jízdního kola

Stav komponenty nelze jednoduše vyjádřit jednou hodnotou, jako je například procento opotřebení na základě najetých kilometrů. U různých komponent jsou totiž jako ukazatel opotřebení sledovány různé údaje. Výsledná aplikace tedy musí podporovat zaznamenávání různých ukazatelů opotřebení.

Nejběžnějším ukazatelem opotřebení je nájezd kilometrů, který sledujeme u většiny komponent. Zde se uživatel může řídit přímo doporučením výrobce, tedy pouze sledovat najeté kilometry na komponentě a po dosažení určitého nájezdu ji vyměnit, případně servisovat. Pokud je ovšem uživatel zkušenější, je vhodné mu poskytnout možnost si stav komponenty zkontrolovat sám a zapsat si stav při aktuálním nájezdu kilometrů do aplikace. Díky této funkcionalitě tak může uživatel sledovat vývoj opotřebení komponenty v průběhu používání. Příkladem může být zaznamenávání vytahání řetězu v milimetrech při různém nájezdu kilometrů. Tyto hodnoty lze také využít pro porovnání výdrže komponent od různých výrobců. Vhodnou možností je také k záznamu o stavu komponenty moci přiložit fotografii, která může sloužit pro porovnání opotřebení při různých nájezdech kilometrů u komponent, jejichž opotřebení lze sledovat vizuálně. To může být například úbytek vzorku u plášťů nebo opotřebení ozubených kol kazety a převodníku.

U některých komponent je nejdůležitějším údajem počet hodin v provozu. Tento údaj je sledován například u odpružených vidlic nebo tlumičů horských kol, které by se měly servisovat pravidelně po určitém počtu hodin v provozu. Tato hodnota bývá určená výrobcem.

Užitečnou funkcí je také možnost zaznamenání prováděných servisních úkonů, včetně případné ceny daného servisu. Díky tomuto údaji může uživatel sledovat, kolik ho stála údržba dané komponenty a zda se mu například nevyplatí danou komponentu po určité úrovni opotřebení vyměnit namísto nákladného servisování.

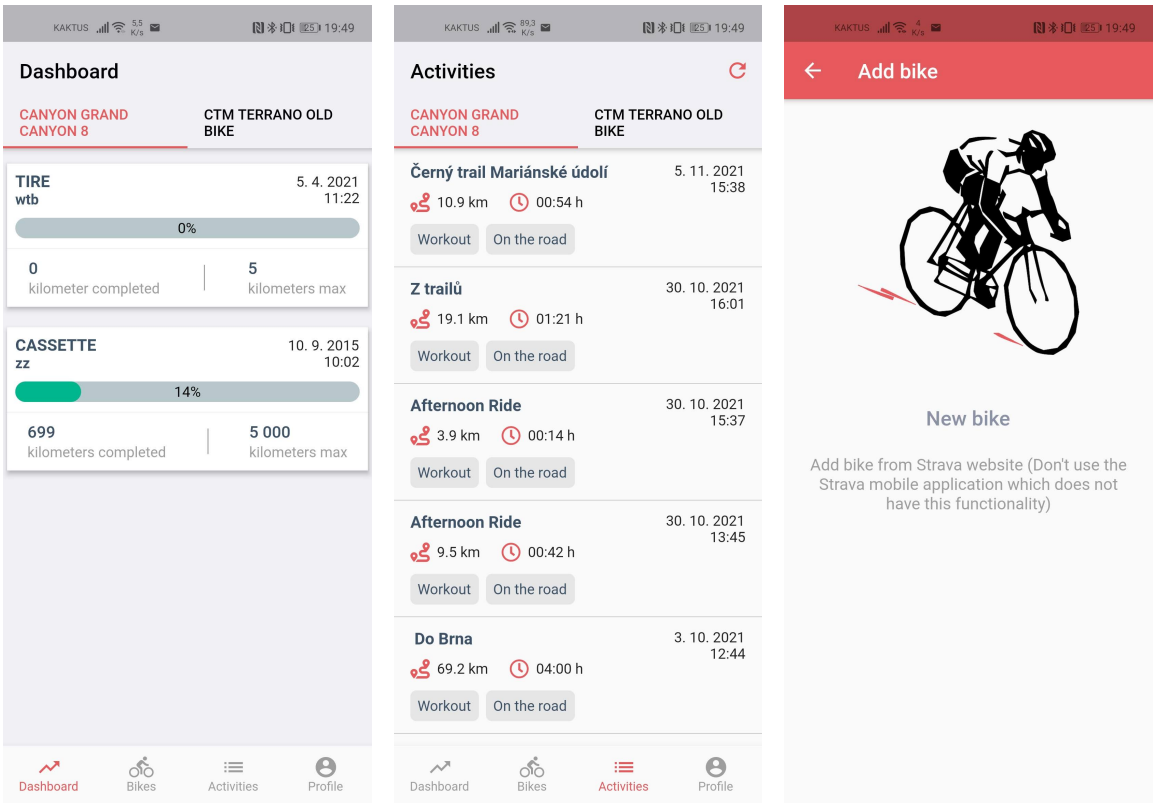
2.1 Existující řešení

2.1.1 BikeManager

Aplikace BikeManager poskytuje základní funkce potřebné pro sledování stavu komponent – počítání kilometrů a hodin v provozu. Při přidání komponenty je umožněno si zvolit vlastní sledovanou jednotku opotřebení – najeté kilometry, hodiny v provozu, případně pevný časový interval, po kterém je nutné komponentu vyměnit nebo servisovat. Opotřebení je poté zobrazováno v seznamu komponent ukazatelem procenta opotřebení. Aplikace ovšem

u komponent, které mají ukazatel na maximu, neumožňuje žádným způsobem tento ukazatel resetovat. Pokud by tedy komponenta byla po zvoleném nájezdu kilometrů servisována, ukazatel by nadále zobrazoval hodnotu nad 100 % a jediným řešením by bylo komponentu smazat a založit novou, případně editací zvýšit životnost komponenty. Aplikace neposkytuje žádné detailnější možnosti zaznamenání opotřebení ani servisů. Není zde také možnost komponenty přesouvat mezi jednotlivými koly.

Aplikace podporuje synchronizaci dat s aplikací Strava. Hlavní nevýhodou aplikace je nutnost využívání aplikace Strava. Jediný způsob, jak se do aplikace BikeManager přihlásit je pomocí Strava účtu, stejně tak je jedinou možností přidání svých kol a zaznamenávání aktivit prostřednictvím synchronizace se Strava účtem. Pokud tedy uživatel nevyužívá aplikaci Strava, je tato aplikace prakticky nepoužitelná. Na obrázku 2.1 lze vidět ukázku tří snímků obrazovky z aplikace BikeManager. První snímek zobrazuje seznam komponent na zvoleném kole, na druhém snímku lze vidět seznam jízd provedených na zvoleném kole a poslední snímek ukazuje obrazovku pro přidání kola, na které je pouze hláška říkající, že kolo lze přidat pouze prostřednictvím aplikace Strava.

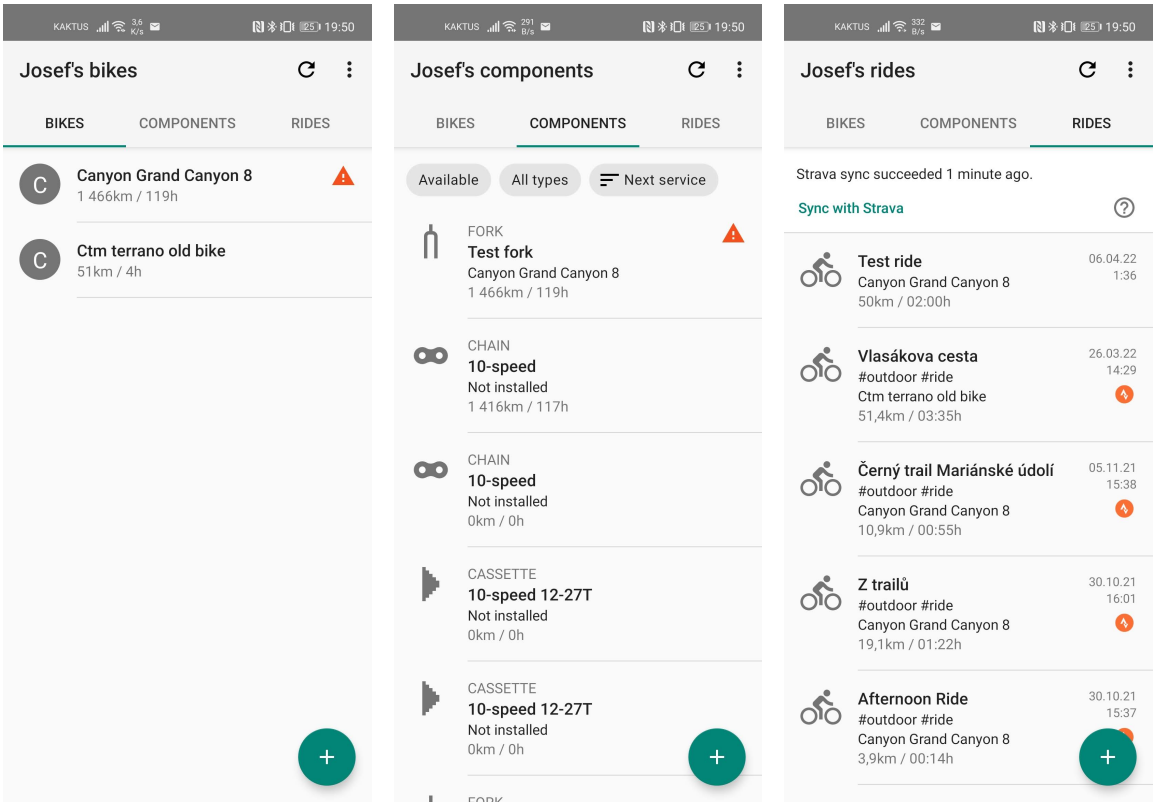


Obrázek 2.1: Část rozhraní aplikace BikeManager obsahující seznam komponent, seznam jízd a obrazovku pro přidání kola.

2.1.2 ProBikeGarage

Aplikace ProBikeGarage má již více funkcí než aplikace BikeManager. U komponent zaznamenává najeté kilometry a hodiny v provozu na základě zapsaných jízd. Umožňuje komponenty přesouvat mezi koly a zobrazit si historii těchto změn u každé komponenty. U těchto záznamů lze také zpětně editovat čas instalace a odinstalace nebo záznam smazat. Na zá-

kladě této změny jsou najeté kilometry a hodiny v provozu přepočítány. Záznamy o jízdách je možno přidávat jak ručně, tak pomocí synchronizace s aplikací Strava. Aplikace ProBikeGarage ovšem neumožňuje u komponent zaznamenávání žádných detailnějších statistik, jako mohou být například provedené servisní úkony nebo opotřebení při určitém nájezdu kilometrů nebo hodinách v provozu. Jediná statistika, kterou tak má uživatel k dispozici o stavu komponenty, je její nájezd kilometrů a čas v provozu. Velkou nevýhodou aplikace je také nemožnost přidat kolo do aplikace jiným způsobem, než synchronizací s aplikací Strava. To ji činí nepoužitelnou, pokud uživatel nepoužívá aplikaci Strava, stejně jako je tomu u aplikace BikeManager. Na obrázku 2.2 lze vidět tři snímky obrazovky z aplikace ProBikeGarage. První snímek ukazuje seznam všech kol uživatele, na druhém lze vidět obrazovku se seznamem všech komponent a jejich opotřebením a na posledním je zobrazen seznam všech jízd.



Obrázek 2.2: Část rozhraní aplikace ProBikeGarage obsahující seznam kol, seznam komponent a seznam jízd.

Kapitola 3

Použité technologie pro vývoj mobilní aplikace

Tato kapitola popisuje principy použitých technologií a standardů pro implementaci. Kapitola také obsahuje popis aplikace Strava a jejího aplikačního rozhraní, neboť součástí implementované aplikace je možnost propojení uživatelského účtu s účtem aplikace Strava.

3.1 TypeScript

TypeScript [14] je programovací jazyk, který poskytuje rozšíření jazyka JavaScript usnadňující psaní čitelného a udržitelného kódu. JavaScript totiž původně začal jako jednoduchý programovací jazyk, který byl využíván pro podporu jednoduché interaktivity na webových stránkách. Postupně se ale vyvinul v jazyk, který je využíván ve velkých projektech, a to nejenom na webu, ale například i na serverové části aplikací s využitím Node.js, a nebo, jako je tomu v případě této práce, i pro tvorbu mobilních aplikací. Pro psaní rozsáhlých projektů má ovšem JavaScript velkou nevýhodu – absenci typové kontroly.

Tato vlastnost JavaScriptu vede často k těžko odhalitelným chybám. Příkladem může být funkce, která očekává jako argument celočíselnou hodnotu, ale je jí předáno číslo ve formátu řetězce. Program je úspěšně spuštěn, ale chová se jinak, než bychom očekávali. S využitím typové kontroly lze ale definovat, že daný parametr funkce má být typu celé číslo a chyba by byla odhalena již před interpretací.

TypeScript tedy rozšiřuje JavaScript o statické typování. Proměnné, nebo i funkci, je možné přiřadit jeden nebo více datových typů, které daná proměnná přijímá. Speciálním datovým typem je typ *any*, který definuje, že do proměnné může být přiřazena hodnota jákehokoliv datového typu. Dalším rozšířením je podpora rozhraní. Využití těchto rozšíření je volitelné, není tedy vynuceno definování typů proměnných nebo návratových hodnot funkcí. Před kompilací je TypeScript transpilován do JavaScriptu.

3.2 React Native

React Native je JavaScriptový framework založený na frameworku React, který usnadňuje vývoj mobilních aplikací Android a iOS. Při běžném nativním vývoji je totiž nutné udržovat dva různé kódy pro verzi aplikace na Android a iOS, protože nativní aplikace pro Android mohou být napsány v jazyku Java nebo Kotlin, zatímco nativní aplikace pro iOS jsou psány v jazyku Objective-C nebo Swift. React Native umožňuje udržovat pouze jeden kód

pro všechny platformy. Nejedná se ovšem o využití webview, které pouze zobrazuje webový obsah jako aplikaci, jako je tomu například při využití nástrojů Ionic nebo Cordova. React Native prostřednictvím JavaScriptového vlákna komunikuje přímo s aplikačním rozhraním dané platformy, pomocí kterého jsou vykreslovány nativní komponenty [3].

Hlavní výhodou tohoto přístupu je rychlost vývoje a znovupoužitelnost kódu pro různé platformy se zachováním nativního vzhledu aplikace. Nevýhodou může být výkonnost. Přestože se velmi blíží výkonnosti nativních aplikací, v některých scénářích, kdy je nutné velké množství volání nativních komponent, nemusí být dostačující [16]. Výkonnost je přesto mnohem lepší, než u řešení založených na webview.

Ačkoliv je hlavním zaměřením Reactu Native vývoj aplikací pro Android a iOS, existují i komunitní rozšíření přidávající podporu dalších platforem, jako je například Windows nebo web.

3.2.1 Komponenty

Aplikace se skládají z komponent, což jsou nezávislé znovupoužitelné části kódu. Komponentám lze předávat atributy, které se nazývají props. Jednotlivé komponenty lze do sebe libovolně zanořovat.

Zápis komponent pomocí JSX

JSX (JavaScript Syntax Extension) je syntaktické rozšíření jazyka JavaScript pro usnadnění zápisu React komponent. Při kompilaci je zápis v JSX přeložen do ekvivalentního zápisu s využitím funkce `React.createElement(component, props, ...children)` [11]. Využití JSX tedy není povinné, ale použití základní syntaxe JavaScriptu by u složitějších komponent vedlo k velmi nepřehlednému kódu. Porovnání zápisu stejné komponenty pomocí funkce `createElement` a s využitím syntaktického rozšíření JSX lze vidět na obrázku 3.1.

```
export default function CardBase(props) {
    return React.createElement(
        View,
        {style: {backgroundColor: "#FDFDFD"}},
        props.children
    );
}
```

```
export default function CardBase(props) {
    return (
        <View style={styles.cardBackground}>
            {props.children}
        </View>
    )
}
```

Obrázek 3.1: Srovnání ekvivalentního zápisu komponenty s využitím funkce createElement a s využitím syntaktického rozšíření JSX.

Základní sada komponent Reactu Native

React Native poskytuje několik základních vestavěných komponent, které jsou mapovány na odpovídající nativní komponenty dané platformy. Nejdůležitějšími pro základní funkčnost jsou:

- **Text** – Komponenta, do které musí být zabalen jakýkoliv text, který chceme vypsat (výpis mimo komponentu vede k vyvolání výjimky).
- **View** – Kontejner podporující rozložení typu flexbox a další styly pro definici rozložení a vzhledu.

- **Scrollview nebo Flatlist** – Komponenty umožňující scrollování v obsahu, který se nevejde na jednu obrazovku.
- **TouchableOpacity nebo Pressable** – Komponenty umožňující definici akce po události kliknutí na komponenty, které jsou do těchto komponent zabaleny. Akce, která má být vykonána, je předána atributem `onPress`.
- **Image** – Komponenta pro zobrazení obrázku předaného pomocí atributu `source`.
- **TextInput** – Komponenta pro textový vstup od uživatele.

Tvorba vlastní komponenty

React Native nabízí dva přístupy k tvorbě vlastních komponent – s využitím tříd nebo s využitím funkcí.

Pokud vytvoříme komponentu jako třídu, musí tato třída dědit z třídy `Component`. Jedinou povinnou metodou je metoda `render`, jejíž návratový typ je `React element`, který má být vykreslen. Ten lze vytvořit pomocí zápisu ve formátu JSX nebo s využitím funkce `React.createElement`.

Pokud zvolíme způsob tvorby komponenty pomocí funkce, stačí pouze definovat funkci, která jako svůj jediný argument přijímá objekt s atributy (v Reactu Native nazývaný *props*) a vracející `React element`.

Stav komponenty

Komponenty v Reactu Native pracují s dvěma typy dat – vstupními daty, nazývanými *props* a s vnitřním stavem, reprezentovaným stavovými proměnnými. *Props* jsou data, která jsou komponentě předána od rodičovské komponenty a jsou neměnenná po celou dobu života komponenty. Stavové proměnné slouží k uložení hodnot, které mohou být po dobu života komponenty měněny. Na změnu jejich hodnoty je reagováno překreslením komponenty.

Při využití třídního přístupu je stavová proměnná definována v konstruktoru. V konstruktoru je třeba zavolat konstruktor rodičovské třídy, tedy třídy `React.Component`. Dále je zde nastavena počáteční hodnota objektu state, který je definován nad instancí komponenty. Hodnotu stavových proměnných lze aktualizovat použitím funkce `setState`, které je jako argument předán objekt obsahující definici nových hodnot stavových proměnných. Použití stavové proměnné v třídním přístupu lze vidět na obrázku 3.2.

```
class Example extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <View>
        <Text>You clicked {this.state.count} times</Text>
        <Button title='Click me' onPress={() => this.setState({ count: this.state.count + 1 })}/>
      </View>
    );
  }
}
```

Obrázek 3.2: Příklad použití stavové proměnné při využití třídního přístupu pro tvorbu komponent. Příklad převzat a upraven z React dokumentace [11].

Při využití funkčního přístupu pro tvorbu komponent můžeme pro práci se stavem komponenty využít hook `useState`. Hook `useState` má jeden argument, který určuje počáteční hodnotu stavové proměnné. Návratovými hodnotami hooku jsou stavová proměnná a funkce, jejíž zavoláním lze hodnotu stavové proměnné změnit na hodnotu předanou v argumentu. Příklad použití hooku `useState` lze vidět na obrázku 3.3.

```
function Example() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Obrázek 3.3: Příklad práce se stavovou proměnnou pomocí hooku `useState` při využití funkčního přístupu tvorby komponent. Příklad převzat a upraven z React dokumentace [11].

Životní cyklus komponenty

Struktura komponent je v Reactu Native, stejně jako v Reactu, reprezentována pomocí DOMu (Document Object Model). Jediným rozdílem je, že DOM v Reactu Native není mapován na DOM prohlížeče, ale jeho jednotlivé části jsou vykresleny pomocí nativního API dané platformy. Komponenty jsou při běhu aplikace do DOMu vkládány a z DOMu odebírány. Každá komponenta má tři hlavní fáze životního cyklu – vkládání do DOMu, aktualizace komponenty (například při změně stavu) a odstraňování z DOMu. React Native umožňuje reagovat na změny v životním cyklu komponenty.

Pokud využíváme způsob tvorby komponent pomocí tříd, jsou nám k dispozici metody životního cyklu třídy `React.Component`. Nejběžněji používanými z nich jsou:

- `componentDidMount()` – Vyvolána ihned po vložení komponenty do DOMu – tedy po vykreslení komponenty. Je vhodná pro zasílání síťových požadavků, například pro zasílání požadavků na aplikační rozhraní [11]. V kombinaci se stavovými proměnnými tak lze nejprve vykreslit komponentu zobrazující informaci o načítání dat, například formou animace načítání. Poté ihned po vykreslení zaslat požadavek na aplikační rozhraní, vyčkat na odpověď a aktualizovat hodnotu stavové proměnné určující, že jsou data načtena, čímž se komponenta znovu překreslí. Pokud by se před vykreslením čekalo na získání všech dat, aplikace by působila zasekaně, neboť by se po dobu čekání na odpověď od serveru nezobrazovalo nic.
- `componentWillUnmount()` – Vyvolána těsně před vymazáním komponenty z DOMu. Je tedy vhodná pro provedení uvolnění paměti, pokud je to nutné.
- `componentDidUpdate()` – Vyvolána při změně stavu komponenty. Není volána při počátečním vykreslení komponenty.

Pokud tvoříme komponenty pomocí funkcí, můžeme využít takzvané hooky. Ekvivalentem metody `componentDidMount()` je hook `useEffect`. Tento hook má dva parametry – prvním je akce, která má být vykonána při vyvolání hooku a druhým je pole závislostí. Pokud je jako pole závislosti předáno prázdné pole, tak je chování hooku `useEffect` stejné jako chování metody `componentDidMount()`, tedy akce, která je předána v prvním parametru, je volána po každém vykreslení komponenty. Pokud závislosti nejsou pouze prázdné pole, tak je akce zavolána při změně některé z hodnot předaných v poli závislostí. Hook `useEffect` také nahrazuje metodu `componentDidUpdate`. Pokud chceme reagovat na změnu hodnoty stavové proměnné, stačí ji předat v poli závislostí. Metodu `componentWillUnmount()` lze rovněž nahradit hookem `useEffect`. Stačí z hooku jako návratovou hodnotu vrátit funkci, která se má vykonat při odstranění komponenty z DOMu.

Stylování komponent

Všechny základní komponenty v Reactu Native přijímají atribut `style`. Styly jsou předávány jako objekt obsahující vlastnosti, které svým názvem odpovídají vlastnostem v CSS, pouze s tou změnou, že jsou zapsány v camel case notaci. Tedy například css vlastnost `background-color` je v objektu typu stylesheet zapsána jako `backgroundColor`.

Objekty se styly lze vytvářet přímo v atributech `style` jednotlivých komponent. Pokud je ale již stylů více, může použitím tohoto přístupu vznikat nepřehledný kód. V tom případě lze využít metodu tvorby objektu se styly pomocí funkce `StyleSheet.create`. Tato funkce jako argument přijímá objekt, jehož klíče jsou názvy jednotlivých skupin stylů a hodnoty jsou objekty se styly. Styly poté lze komponentě předat pomocí reference na skupinu stylů v tomto objektu. Ukázku použití funkce `StyleSheet.create` lze vidět na obrázku 3.4.

```
const LotsOfStyles = () => {
  return (
    <View style={styles.container}>
      <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>
    </View>
  );
};
const styles = StyleSheet.create({
  bigBlue: {
    color: 'blue',
    fontWeight: 'bold',
    fontSize: 30,
  },
  red: {
    color: 'red',
  },
});
```

Obrázek 3.4: Příklad tvorby a použití objektu typu `StyleSheet`.

3.2.2 Navigátory

Každá obrazovka v Reactu Native je reprezentována komponentou. Málokdy se ale aplikace skládají pouze z jedné obrazovky. Pokud máme obrazovek více, potřebujeme nástroj, pomocí kterého se mezi nimi budeme pohybovat. Při webovém vývoji v Reactu je nám k dispozici URL adresa v prohlížeči, kde k jednotlivým adresám lze přiřadit komponenty, které se mají vykreslit. V mobilních aplikacích jsou k tomuto účelu využívány navigátory. Dle dokumentace Reactu Native je doporučeno využít komunitní knihovnu `React Navigation`, která nabízí tři typy navigátorů – `StackNavigator`, `TabNavigator` a `DrawerNavigatior`, které jsou mapovány na nativní navigátory systémů Android i iOS. Při využití jednoho z navigátoru je třeba zaregistrovat jako obrazovky komponenty, které mají být v daných obrazovkách vykresleny. Obrazovkám lze přiřadit různá nastavení, jako je například vzhled záhlaví navigátoru nebo animace při přechodu mezi obrazovkami. Můžeme jim také předat počáteční parametry.

Komponentám obrazovek jsou předány atributy `navigation` a `route`. Pomocí atributu navigation lze přistupovat k navigačním metodám[1], jako je například metoda `navigate`, pomocí které se lze přesunout na jinou obrazovku a případně obrazovce předat parametry, nebo metoda `goBack`, která zavře aktuální obrazovku a přesune se na předchozí. Pomocí atributu `route` lze přistupovat k informacím o aktuální obrazovce, jako je její jméno, unikátní klíč obrazovky nebo parametry, které mohou být předány pomocí metody `navigate` nebo jako počáteční parametry v navigátoru.

DrawerNavigator je reprezentován jako vyjíždějící menu z boku obrazovky, které obsahuje odkazy na jednotlivé zaregistrované obrazovky.

StackNavigator, tedy zásobníkový navigátor, jak již název napovídá, jednotlivé obrazovky při přesměrování umisťuje na sebe jako zásobník. Při přesměrování na jinou obrazovku, než je výchozí, je v záhlaví obrazovky zobrazena šipka zpět, která obrazovku z vrchu zásobníku odstraní a přesměruje uživatele na obrazovku, která je nyní na vrcholu zásobníku. Příklad použití zásobníkového navigátoru lze vidět na obrázku 3.5

[1] https://reactnavigation.org/docs/navigation-prop

```
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
const Stack = createNativeStackNavigator();
const MyStack = () => {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Home"
          component={HomeScreen}
          options={{ title: 'Welcome' }}
        />
        <Stack.Screen name="Profile" component={ProfileScreen} />
      </Stack.Navigator>
    </NavigationContainer>
  );
};
```

Obrázek 3.5: Příklad definice zásobníkového navigátoru obsahujícího dvě obrazovky pomocí komponent poskytovaných knihovnou `React Navigation`. Příklad převzat z dokumentace Reactu Native [12].

TabNavigator zobrazí ve spodní části obrazovky menu s položkami, které po kliknutí přesměrují uživatele na danou obrazovku.

Navigátory lze do sebe libovolně zanořovat – například jako obrazovku tabového navigátoru můžeme nastavit komponentu, v které je definován zásobníkový navigátor.

3.2.3 Vývojové prostředí Reactu Native

K dispozici jsou dvě běžně využívaná vývojová prostředí – React Native CLI a Expo. V následujících sekcích jsou popsány hlavní výhody a nevýhody použití obou prostředí.

React Native CLI

React Native CLI je základní vestavěné vývojové prostředí. Hlavní výhodou využití této varianty je možnost tvorby nativních modulů pro danou platformu, tedy pro iOS v jazyce Objective-C a pro Android v jazycích Java nebo Kotlin. Další výhodou je jednodušší tvorba apk balíčku aplikace.

Hlavní nevýhodou je potřeba emulátorů pro vývoj. Pro vývoj aplikace na platformu Android je třeba mít k dispozici Android Studio. Pro vývoj aplikací na iOS je třeba vývojové prostředí Xcode, které je ale dostupné pouze na operačním systému OS X [6].

Expo

Expo je framework, který poskytuje nástroje pro vývoj a správu React Native projektu. Při vývoji pomocí Expo frameworku jsou používány dva nástroje:

- **Expo CLI** – Hlavní rozhraní mezi programátorem a nástroji poskytovanými Expo frameworkem. Lze pomocí něj například vytvářet nové projekty, spustit vývojový server nebo sestavit aplikaci připravenou pro publikaci na Google Play nebo Apple App Store [4].

- **Expo klientská aplikace** – Aplikace dostupná pro Android i iOS, umožňující vyvíjenou aplikaci spustit přímo na mobilním zařízení naskenováním QR kódu, který Expo CLI vygeneruje při spuštění vývojového serveru.

Součástí Expo frameworku je také Expo SDK, což je sada nástrojů umožňující jednoduše přistupovat k funkcionalitám zařízení a systému. Jednotlivé nástroje lze v projektu importovat jako JavaScriptové moduly. Poskytovanými moduly jsou například modul pro práci s push notifikacemi, modul pro autorizaci pomocí protokolu OAuth nebo modul pro výběr obrázků z úložiště telefonu.

Pro správu projektu s využitím frameworku Expo je možné využít dva přístupy:

- **Managed workflow** [5] – O většinu služeb se stará přímo Expo. Programátor píše kód pouze v JavaScriptu, případně TypeScriptu, nemusí tak využívat XCode nebo Android studio. Projekt je spravován pomocí `expo-cli` a testován přímo na telefonu prostřednictvím mobilní aplikace `Expo`. Konfigurace projektu, jako je například nastavení ikony aplikace nebo jejího názvu, je spravována pomocí konfiguračního souboru `app.json` (případně `app.config.js`). Nevýhodou tohoto přístupu je nemožnost tvorby nativních modulů pro iOS a Android nebo chybějící podpora některých aplikačních rozhraní systémů Android a iOS .

- **Bare workflow** – Programátor si sám spravuje nativní projekty pro iOS a Android. Je možné využít většinu modulů poskytovaných Expo frameworkem a rovněž je podporována tvorba nativních modulů pro Android a iOS. Není podporována konfigurace pomocí konfiguračního souboru `app.json`, místo toho je nutné konfigurovat samostatně každý nativní projekt.

3.3 Firebase

Firebase je platforma společnosti Google poskytující nástroje pro vývoj a správu mobilních a webových aplikací. S využitím těchto služeb lze vyvíjet aplikace založené na modelu backend jako servis [2]. Vývojář se tak může při vývoji soustředit především na vývoj uživatelského rozhraní aplikace a pro běžné backendové služby, jako je správa uživatelských účtů, zasílání notifikací nebo perzistentní ukládání dat, využít služby platformy Firebase. Mezi klíčové poskytované služby patří podpora autentizace, databázové služby nebo cloudové úložiště, ale také mnoho analytických a monitorovacích služeb pro zpracování statistik o používání aplikace.

Firebase nabízí dva plány pro využití jejich služeb – No-cost a Pay as you go. Plán No-cost je zcela zdarma, ale má určitá omezení. Těmi jsou například omezení autentizace pomocí telefonního čísla na maximálně 10 000 autentizací měsíčně nebo omezení využití databáze firestore na maximum 1 GiB uložených dat, 50 000 čtení, 10 000 zápisů a 20 000 smazání měsíčně. Při využití plánu Pay as you go platí vývojář dle využití služeb. Například databáze Firestore je zdarma, pokud jsou dodrženy stejné limity využití, jako je tomu v plánu No-cost. Při přesažení limitů je vývojáři účtován poplatek za každý 1 GiB uložených dat navíc a každých 100 000 provedených operací dle ceníku Google Cloud[2].

[2] https://cloud.google.com/firestore/pricing

3.3.1 Firebase autentizace

Autentizace pomocí Firebase umožňuje jednoduchou tvorbu zabezpečeného autentizačního systému. Identitu uživatele je možné využít k provázání s dalšími poskytovanými službami, například k přiřazení záznamu v databázi ke konkrétnímu uživateli pomocí jeho jednoznačného identifikátoru.

Firebase podporuje nejběžnější metodu autentizace pomocí emailu a hesla, autentizaci pomocí telefonního čísla, ale i autentizaci pomocí účtů u služeb třetích stran, jako je například Google, Facebook nebo Twitter. Dále také nabízí možnost anonymního přihlášení, kdy je uživateli na pozadí vytvořen dočasný anonymní účet. Pokud se uživatel později rozhodne si vytvořit svůj účet pomocí některé z podporovaných metod, je tento dočasný účet s jeho nově vytvořeným účtem propojen. Díky využití standardu OAuth2, který je popsán v sekci 3.4, je také možné firebase autentizaci propojit s vlastním autentizačním systémem.

Firebase také nabízí další běžné služby pro správu uživatelských účtů, jako je ověření emailu, ověření telefonního čísla nebo obnovení zapomenutého hesla.

3.3.2 Databáze

Firebase nabízí dvě databázové služby – Firestore a Realtime database.

Realtime database

Realtime databáze je NoSQL databáze, která ukládá data ve formátu JSON. Podporuje synchronizaci dat v aplikaci s databází v reálném čase s velmi nízkou odezvou. Jsou podporovány pouze jednoduché dotazy nad daty. Není například možné v dotazu filtrovat a zároveň řadit podle nějaké vlastnosti. Realtime databáze nevyžaduje vytváření indexů, dotazy nad větším objemem dat jsou tedy pomalejší.

Realtime databázi je tedy vhodné použít, pokud vyžadujeme synchronizaci dat v reálném čase s co nejmenší odezvou, pracujeme spíše s menšími objemy dat a vystačíme si s jednoduššími dotazy.

Firestore

Firestore je NoSQL databáze. Jednotlivé záznamy jsou ukládány do kolekcí dokumentů. Každý dokument má svůj jednoznačný identifikátor a je složen z polí, které mají svůj klíč a hodnotu. Hodnota má určena datový typ. Tato struktura je velmi podobná formátu JSON. Jediným rozdílem je, že Firestore dokumenty obsahují navíc některé datové typy, jako je například reference na jiný dokument. Jsou také limitovány velikostí 1 MB na dokument [7]. Dokument může obsahovat také další kolekci dokumentů. Díky této struktuře lze spravovat komplexnější strukturu dat jednodušeji a přehledněji, než u Realtime databáze, která je celá uložena jako jeden velký JSON strom. Jsou podporovány složitější dotazy, přesto jsou zde ale určité limitace oproti využití SQL databází – není například možné při filtrování podle hodnoty pole nastavit více podmínek, které tato hodnota musí splňovat. Při dotazu nad dokumentem, který obsahuje pole datového typu reference na jiný dokument, není v poli s referencí vrácen přímo dokument, na který je odkazováno, ale pouze reference na tento objekt. Pokud tedy chceme získat přímo referencovaný dokument, musíme provést další dotaz. Pro každý dotaz musí být vytvořen index, díky čemuž jsou především složitější dotazy rychlejší, než při využití Realtime database.

Firestore je tedy vhodné použít, pokud pracujeme s komplexnější strukturou nebo větším objemem dat a potřebujeme vytvářet složitější dotazy.

Bezpečnostní pravidla

Firestore i Realtime database podporují definici bezpečnostních pravidel. Pomocí těchto pravidel je při každém požadavku automaticky kontrolováno, zda je daný požadavek uživatel oprávněn provést. To vše bez nutnosti psaní vlastního autentizačního a autorizačního kódu na straně serveru.

V případě Realtime database jsou pravidla definována pro určitou cestu v JSON stromu dat. Pravidla jsou psána ve formátu JSON, kde klíčem je typ pravidla a hodnotou je výraz, jehož výsledkem je pravdivostní hodnota. V Realtime database máme k dispozici čtyři typy pravidel – `read`, `write`, `validate` a `indexOn`. Read a write pravidla určují, zda má uživatel pro danou podcestu právo číst a zapisovat data. Pravidlem validate lze definovat, jak mají vypadat vkládaná data při zápisu. Pravidlem indexOn je možné vytvořit index pro dotazování nad daty.

Pravidla ve Firestore se skládají ze dvou částí – příkazu `match` a výrazů `allow`. Pomocí příkazu `match` je vybrána podmnožina dokumentů, pro které dané pravidlo platí. Výrazy `allow` poté určují, jakou akci lze provést za jakých podmínek.

V pravidlech Firestore lze definovat pravidla pro operace `read` a `write`, tedy pro čtení a zápis. Tyto operace lze rozdělit na více podrobné. Pro čtení lze definovat zvlášť podmínku pro výpis jednoho dokumentu (akce `get`) a výpis více dokumentů (operace `list`). Pro zápis lze zvlášť definovat podmínky pro vytvoření nového dokumentu (akce `create`), úpravu existujícího dokumentu (akce `update`) a smazání dokumentu (akce `delete`)

Na obrázku 3.6 lze vidět příklad pravidla, které povoluje u všech dokumentů čtení a zápis uživatelům, kteří jsou autentizováni.

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Obrázek 3.6: Příklad bezpečnostního pravidla databáze Firestore, které povoluje čtení a zápis autentizovaným uživatelům. Příklad převzat z dokumentace Firestore [8]

3.3.3 Cloud Storage

Cloud Storage je cloudové úložiště určené pro ukládání obsahu, jako jsou například fotografie nebo videa. Data jsou ukládána do kontejnerů, které se v Cloud Storage nazývají buckety. Každému nahranému obsahu je přiřazen unikátní identifikátor. Tento identifikátor lze poté využít například k přiřazení nahraného obsahu k uživateli prostřednictvím záznamu v databázi. Cloud storage také umožňuje definici bezpečnostních pravidel pro zápis a čtení obsahu. Tato bezpečnostní pravidla mají stejný formát jako bezpečnostní pravidla databáze Firestore, která jsou popsána v sekci 3.3.2. Jediným rozdílem je, že pravidla nejsou definována pro podmnožinu dokumentů, ale bucketů.

3.4 Aplikace Strava a její aplikační rozhraní

Strava je jednou z nejpoužívanějších mobilních aplikací na zaznamenávání a sdílení sportovních aktivit. V lednu roku 2022 měla aplikace dle statistik 76 milionů aktivních uživatelů a každý měsíc jich milion přibývá [1]. Nejčastěji je využívána běžci a cyklisty k zaznamenávání aktivit včetně záznamu trasy pomocí GPS. Strava je nazývána sociální sítí pro sportovce, neboť umožňuje aktivity zveřejňovat, přidávat k nim popisky a fotografie z dané aktivity a sdílet je tak s ostatními uživateli. Ostatní uživatelé mohou aktivity komentovat, sdílet nebo je označit takzvaným kudos. Nabízí také mnoho dalších komunitních aspektů, jako je podpora mnoha různých žebříčků, například žebříčku rychlosti projetí nebo proběhnutí určitým segmentem a získávání různých ocenění za umístění na žebříčcích. K dispozici je také webové rozhraní aplikace, pomocí kterého lze spravovat své aktivity a prohlížet si aktivity ostatních uživatelů. Díky těmto komunitním aspektům je Strava často využívána i uživateli, kteří k zaznamenávání svých aktivit využívají služeb jiných aplikací, například různých aplikací na chytré hodinky, které mohou být se Stravou propojeny.

Strava poskytuje vývojářům REST API, pomocí kterého lze provádět operace nad daty, jako jsou informace o uživateli, aktivity uživatele nebo jeho vybavení (například kola). Aplikační rozhraní je limitováno na maximum 100 požadavků za 15 minut a 1000 požadavků denně na každou aplikaci. Vlastní aplikaci je nutné zaregistrovat ve webovém rozhraní Stravy. Zde jsou aplikaci přiřazeny identifikátory, které jsou poté zasílány spolu s požadavky na aplikační rozhraní. Před zasláním požadavků na aplikační rozhraní je nutné získat k datům autorizaci od uživatele. To je možné provést pomocí autorizačního protokolu OAuth 2.0, který Strava podporuje.

REST

REST (Representational State Transfer) je architektura aplikačního rozhraní, která definuje principy a omezení pro takzvané RESTful aplikační rozhraní. REST je nezávislý na všech protokolech, ale většina běžných implementací využívá protokol HTTP [13]. RESTful aplikační rozhraní je postaveno okolo zdrojů, což je jakýkoliv druh objektu, ke kterému může klient přistupovat. Každý zdroj má svůj unikátní identifikátor – URI.

REST API postavená na HTTP využívají k provádění operací HTTP metody get, post, put, patch a delete. Get slouží k získání zdroje, post k vytvoření nového zdroje, put k úpravě, patch k částečné úpravě a delete ke smazání zdroje. K zasílaní dat klientem nebo aplikačním rozhraním bývá využíván formát JSON. Data ve formátu JSON jsou umístěna do těla HTTP požadavku. REST API je bezestavové – mezi jednotlivými požadavky není ukládán žádný stav, každý požadavek je samostatný a nezávislý, musí s ním být zaslána všechna potřebná data k jeho dokončení. Jako indikátor výsledku zpracování jsou používány stavové kódy HTTP protokolu.

Autorizace pomocí OAuth 2.0

OAuth 2.0 (Open Authorization 2.0) je autorizační protokol umožňující získat přístup k datům uživatele, která jsou vlastněna jinou aplikací, bez nutnosti vydávání se za uživatele [17]. Principem protokolu je získání přístupu k datům na základě přístupového tokenu, který je vydán autorizačním serverem po získání autorizace od uživatele.

Procesu autorizace se účastní čtyři entity [15]:

- Vlastník dat, ke kterým chceme získat přístup

- Autorizační server, který vydává přístupový token
- Server se zdroji, který ověřuje přístupový token a zpracovává požadavky na data od klientské aplikace
- Klient, který chce získat přístup k datům

Před započetím procesu autorizace musí mít klient přiřazen od autorizačního serveru identifikátor a tajný klíč, kterými se identifikuje. Prvním krokem autorizačního procesu je zaslání požadavku na autorizační server klientem. Součástí požadavku musí být jeho identifikátor, tajný klíč, takzvané scopes, které definují, ke kterým datům chce klient získat přístup a URI koncového bodu, na který má být zaslána odpověď ve formě přístupového tokenu. Klientská aplikace přesměruje uživatele na autorizační server, kde je uživatel vyzván k odsouhlasení poskytnutí přístupu k požadovaným datům. Po odsouhlasení zkontroluje autorizační server, zda klient odsouhlasil přístup ke všem požadovaným datům a poté je klientské aplikaci zaslán přístupový token. Autorizační server ovšem nemusí přímo vrátit přístupový token. Pro lepší bezpečnost je často využíván přístup, kdy je navrácen autorizační kód, který lze poté vyměnit za přístupový token. Autorizační server také může vrátit obnovovací token, který lze použít pro získání nového přístupového tokenu po vypršení jeho platnosti.

Po dokončení autorizačního procesu může klient zasílat na server se zdroji požadavky, s nimiž musí vždy zaslat i jeho přístupový token.

3.5 Nástroj na prototypování Figma

Figma je vektorový grafický editor a prototypovací nástroj. Zaměřuje se na tvorbu uživatelského rozhraní a uživatelskou zkušenost. Nadesignovaným prvkům je možné dodat interaktivitu, díky čemuž lze vytvořit plnohodnotný prototyp vhodný k uživatelskému testování bez nutnosti programování. To umožňuje flexibilněji a rychleji upravovat návrh uživatelského rozhraní. Figma nabízí také verze aplikace pro Android a iOS, pomocí kterých lze prototypy mobilních aplikací spustit a testovat přímo na telefonu.

Kapitola 4

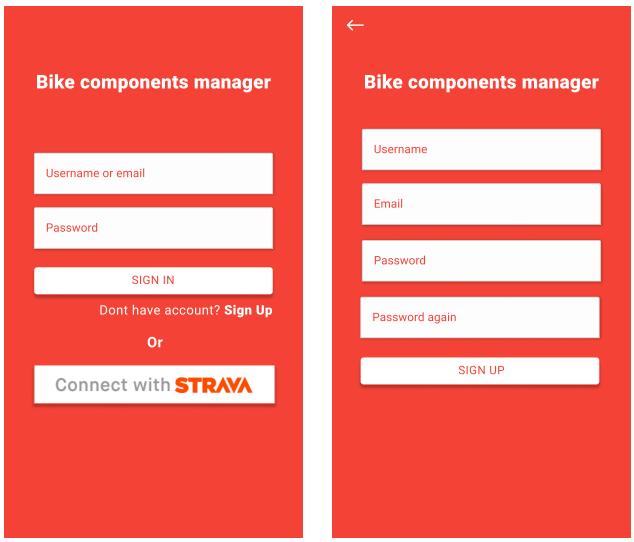
Návrh a testování uživatelského rozhraní

Tato kapitola popisuje proces tvorby návrhu uživatelského rozhraní, následné testování s uživateli a postupné vylepšování návrhu na základě zpětné vazby od testerů.

4.1 Návrh uživatelského rozhraní

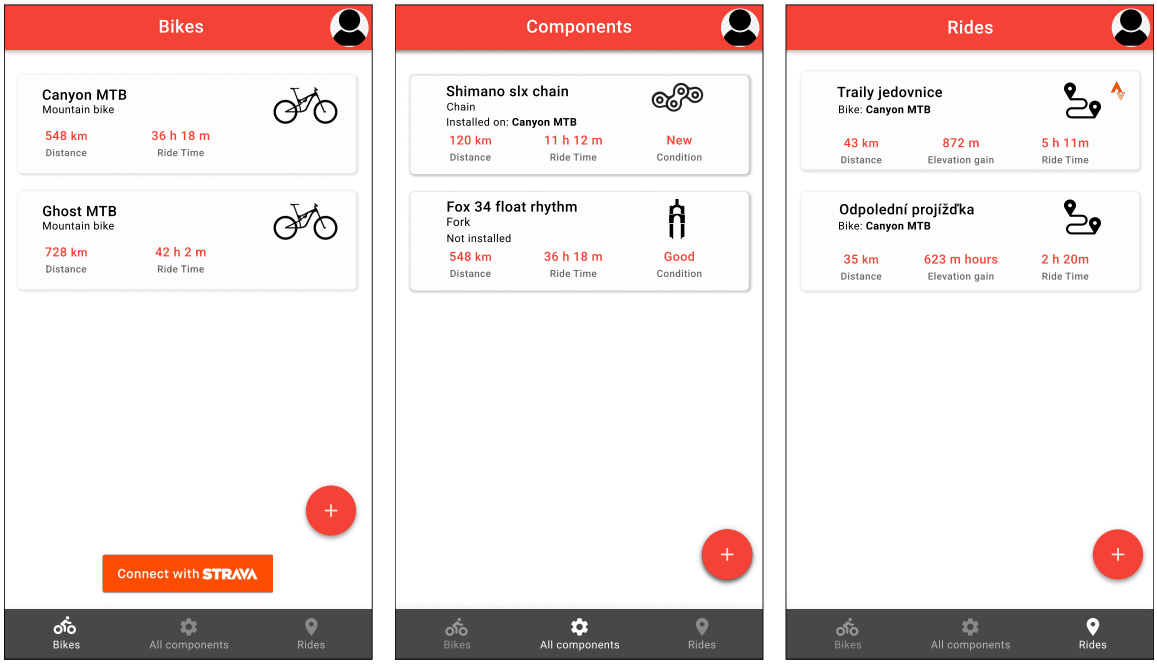
Návrh byl vytvořen pomocí nástroje Figma. Jednotlivým prvkům návrhu byla dodána interaktivita, aby návrh mohl být použit jako plnohodnotný prototyp pro účely uživatelského testování. Návrh byl tvořen s využitím principů tvorby uživatelského rozhraní popsaných Marshem [10].

První částí návrhu jsou obrazovky s přihlašovacím a registračním formulářem. Na přihlašovací obrazovce má uživatel na výběr ze dvou možností. První možností je autentizace pomocí emailu a hesla, která je možná, pokud má uživatel účet, který byl vytvořen prostřednictvím registračního formuláře. Druhou možností je autentizace prostřednictvím Strava účtu. Na přihlašovací obrazovce se rovněž nachází odkaz na registrační formulář. Návrh přihlašovací a registrační obrazovky lze vidět na obrázku 4.1.



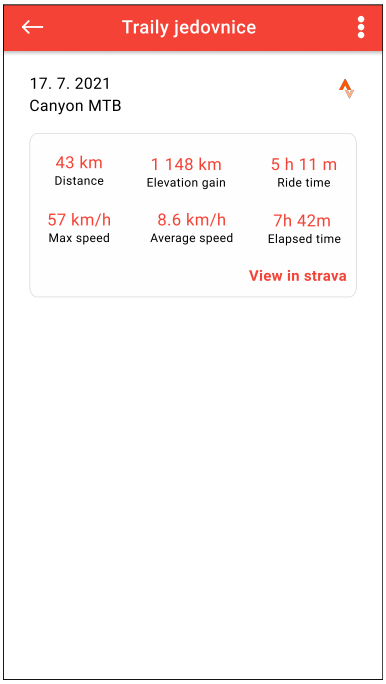
Obrázek 4.1: Návrh přihlašovací a registrační obrazovky.

Další částí návrhu jsou obrazovky se seznamy všech kol, komponent a jízd. Při přihlášení je uživatel přesměrován na obrazovku se seznamem kol, z které se prostřednictvím menu ve spodní části obrazovky lze přesunout na obrazovky se seznamem komponent a jízd. Na těchto obrazovkách jsou jednotlivé prvky reprezentovány formou karet, které obsahují nejdůležitější informace o prvcích a jejich ikonu. Tyto karty také slouží jako odkaz na detailní obrazovku daného prvku. Kola a jízdy, které jsou importovány z aplikace Strava, jsou označeny v pravém horním rohu karty logem Stravy. V pravém dolním rohu obrazovky se nachází tlačítko, které uživatele přesměruje na formulář pro přidání nového prvku (kola, komponenty nebo jízdy). Ve spodní části obrazovky se seznamem kol je umístěno tlačítko, pomocí kterého lze propojit svůj účet s účtem Strava a synchronizovat tak své jízdy a kola. Toto tlačítko se zobrazí pouze pokud se uživatel registroval pomocí registračního formuláře a svůj účet si ještě se Stravou nepropojil. Návrh těchto obrazovek lze vidět na obrázku 4.2.



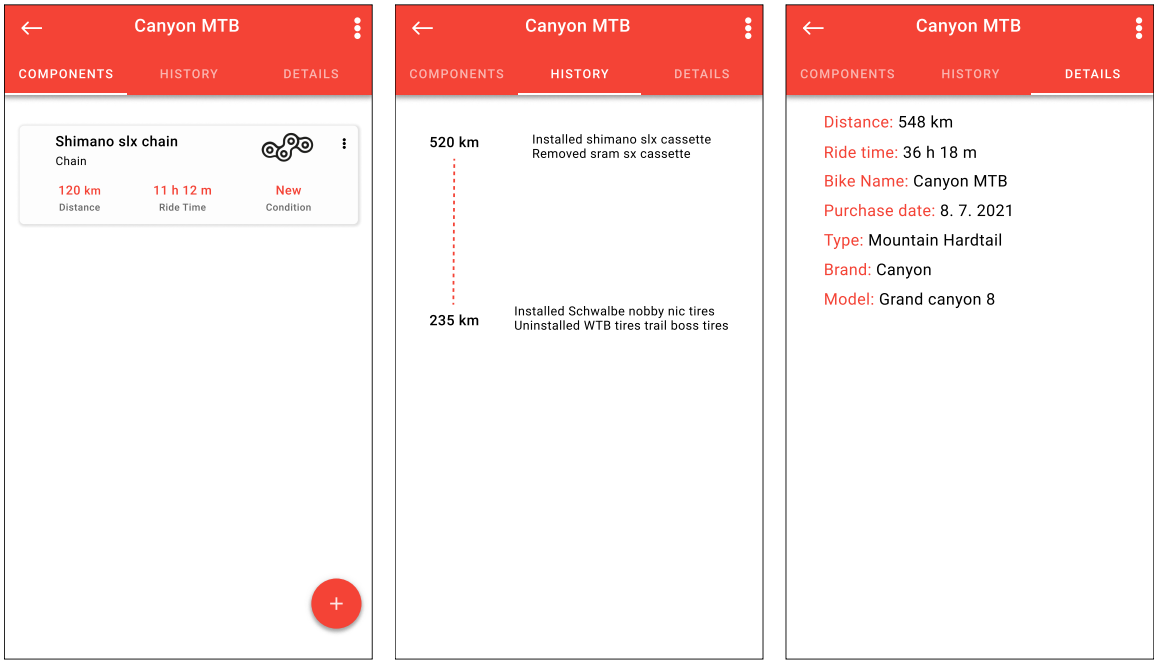
Obrázek 4.2: Návrh obrazovek se seznamem kol, komponent a jízd, které jsou uživateli zobrazeny po přihlášení a jsou propojeny pomocí tabového menu. Prvky ve formě karet fungují jako odkazy na detailní obrazovky.

Návrh obrazovky detailu jízdy obsahuje název jízdy, datum a různé statistiky dané jízdy, jako je například ujetá vzdálenost nebo průměrná rychlost. Pokud je jízda importována ze Stravy, je zde umístěn také odkaz, pomocí kterého si lze danou jízdu zobrazit přímo v aplikaci Strava. Návrh této obrazovky lze vidět na obrázku 4.3.



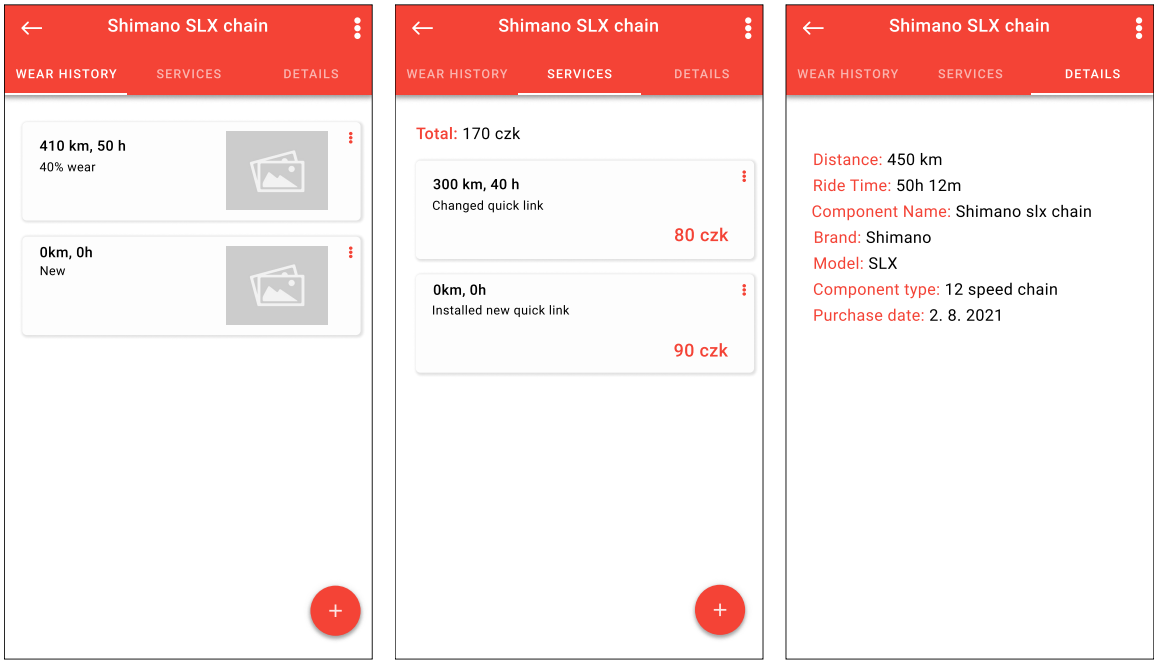
Obrázek 4.3: Návrh detailní obrazovky jízdy obsahující statistiky dané jízdy a odkaz do aplikace Strava, který je zobrazen, pokud byla jízda ze Stravy importována.

Návrh detailu kola se skládá ze tří obrazovek, mezi kterými se lze pohybovat pomocí menu umístěného v horní části obrazovky. Těmito obrazovkami jsou seznam aktuálně nainstalovaných komponent na daném kole, historie výměn komponent a informace o kole. V obrazovce se seznamem komponent je v pravém dolním rohu umístěno tlačítko, pomocí kterého lze přiřadit ke kolu některou z dostupných komponent. Po jeho stisknutí se zobrazí seznam komponent, které nejsou nainstalovány na žádném kole a po zvolení některé z nich je k tomuto kolu přiřazena. Pomocí možnosti v menu komponenty, které lze zobrazit kliknutím na tři tečky na kartě, lze komponentu od kola odebrat. Návrh těchto obrazovek lze vidět na obrázku 4.4.



Obrázek 4.4: Návrh obrazovek se seznamem nainstalovaných komponent na kole, historií nainstalovaných komponent a informacemi o kole, mezi kterými se lze pohybovat pomocí tabového menu.

Návrh detailu komponenty se skládá rovněž ze tří obrazovek, mezi kterými se lze pohybovat pomocí menu v horní části obrazovky. První z obrazovek je obrazovka obsahující záznamy o stavu komponenty při určitém nájezdu kilometrů. Každý záznam je reprezentován kartou, která obsahuje informaci o nájezdu kilometrů a hodinách v provozu, slovní popis stavu nebo opotřebení komponenty a fotografii, která může být k záznamu volitelně přiložena. Druhou obrazovkou je obrazovka se záznamy o provedených servisních úkonech na komponentě. Záznamy jsou rovněž reprezentovány pomocí karty, která obsahuje nájezd kilometrů a hodiny v provozu v době provedení servisu, popis servisního úkonu a jeho cenu. V horní části obrazovky lze vidět celkovou cenu všech servisních úkonů provedených na komponentě. V obou těchto obrazovkách je umístěno tlačítko, které uživatele přesměruje na formulář pro přidání nového záznamu. Poslední obrazovkou je obrazovka s informacemi o komponentě. Návrh těchto obrazovek lze vidět na obrázku 4.5.



Obrázek 4.5: Návrh obrazovek s historií opotřebení komponenty, servisní historií komponenty a informacemi o komponentě, mezi kterými se lze pohybovat pomocí tabového menu.

4.2 Uživatelské testování a vylepšování návrhu

K uživatelskému testování byl využit interaktivní mód prototypovacího nástroje Figma. Testování se účastnilo celkem pět uživatelů. Nejprve byli testeři obeznámeni s průběhem testování aplikace a se základním konceptem aplikace. Poté dostali několik úkolů, které měli v aplikaci vykonat. Uživatelům bylo také řečeno, aby při snaze o splnění úkolu nahlas popisovali své myšlenkové pochody [9].

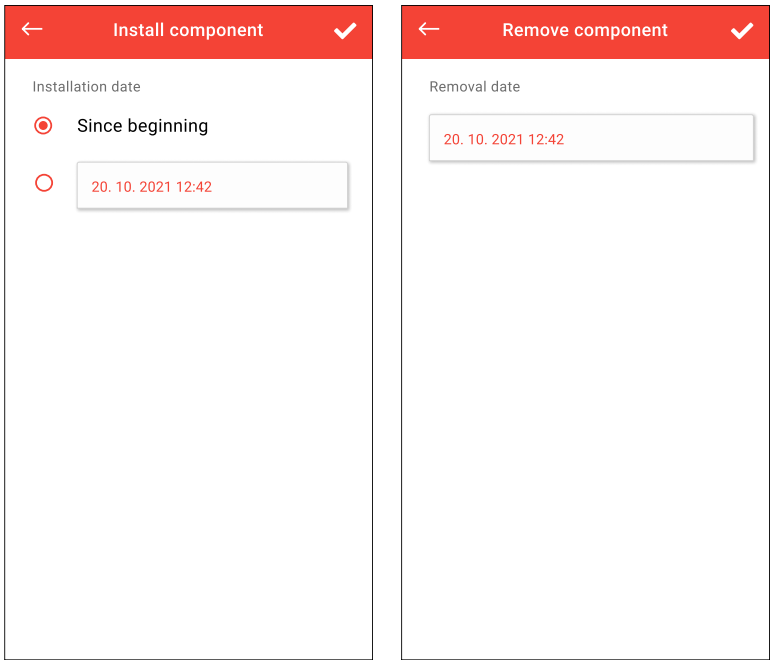
Některými z těchto úkolů byly například:

- Vytvořte si v aplikaci nový účet.
- Vytvořte nové kolo, novou komponentu a kolo ke komponentě přiřaďte.
- Přidejte do aplikace záznam o jízdě.
- Zjistěte, kdy byl naposled proveden servis na komponentě s názvem Shimano slx chain.
- Přemístěte komponentu Shimano chain z kola Canyon MTB na kolo Ghost MTB.

Druhá verze návrhu a jeho uživatelské testování

Ze zpětné vazby od uživatelů vyplynulo, že aplikace je celkově navrhnuta přehledně a poměrně jednoduše se v ní orientuje. Hlavním odhaleným nedostatkem se ukázala být absence možnosti volby času instalace komponenty na kolo a času odinstalace komponenty z kola. Pokud by tak uživatel zapomněl po výměně komponenty změnit konfiguraci svého kola v aplikaci a zaznamenaná jízda by se mu započítala do špatné komponenty, nešlo by tuto akci navrátit. Návrh instalace a odinstalace komponenty byl tedy v druhé verzi návrhu přepracován. Proces instalace komponenty je nyní složen ze dvou kroků – volby samotné

komponenty a volby data instalace pomocí formuláře. Jako datum instalace je možné zvolit, že je komponenta nainstalována na kole od jeho nákupu. Do procesu odinstalace byl rovněž přidán formulář s volbou času odinstalace. Návrh těchto obrazovek lze vidět na obrázku 4.6.

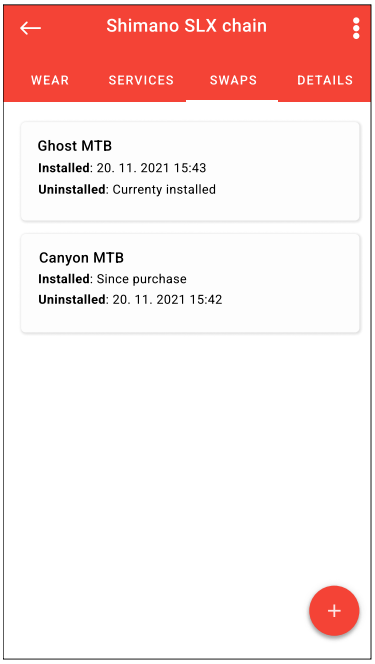


Obrázek 4.6: Návrh obrazovek s volbou času instalace a odinstalace komponenty, které jsou součástí procesu instalace a odinstalace komponenty.

Po dokončení úprav bylo opět provedeno testování s uživateli. Testovány byly především úkony instalace a odinstalace komponenty. Na základě testování vyplynulo, že by bylo vhodné si moci zobrazit historii instalací komponenty a umožnit záznam o instalaci zpětně smazat v případě, kdy by se uživatel při zadávání času instalace a odinstalace zmýlil. Pokud by totiž uživatel nainstalovat a odinstaloval komponentu, tento záznam by v druhé verzi návrhu nebylo možné nikde dohledat a nemohl by čas instalace nebo odinstalace již upravit, čímž by některá jízda mohla být započítána do špatné komponenty.

Třetí verze návrhu a jeho uživatelské testovaní

Do detailní obrazovky komponenty byla tedy přidána čtvrtá obrazovka obsahující historii výměn komponenty s možností úpravy jednotlivých záznamů. Návrh této obrazovky lze vidět na obrázku 4.7.



Obrázek 4.7: Návrh obrazovky s historií výměn komponenty, která je umístěna jako čtvrtá obrazovky v detailu komponenty.

Po dokončení této úpravy bylo opět provedeno uživatelské testování se zaměřením na otestování nové funkcionality. V této fázi testování již nebyl nalezen žádný větší problém. Třetí verze návrhu je tedy finální.

Kapitola 5

Implementace

Tato kapitola popisuje implementaci aplikace. Je popsána struktura projektu, způsob implementace frontendu, implementace práce s daty s využitím služeb Firebase a synchronizace s účtem aplikace Strava.

5.1 Struktura projektu

V kořenové složce projektu se nachází soubor `App.tsx`, který je vstupním souborem aplikace a `Root.tsx`, který obsahuje komponentu využitou v `App.tsx` pro vykreslení aplikace. Dále jsou zde umístěny složky a soubory typické pro projekt v Reactu Native:

- `package.json` – Soubor vytvořený správcem balíčků npm obsahující závislosti daného projektu .
- `app.config.js` – Soubor obsahující konfiguraci React Native projektu, jako například název, verzi nebo ikonu aplikace.
- `tsconfig` – Konfigurace překladače jazyka TypeScript.
- `.env` – Soubor obsahující konstanty daného projektu. Lze ho využít pro uložení citlivých informací, jako například tajné klíče k databázi, aplikačnímu rozhraní a podobně.

Dále je v kořenové složce umístěna složka `App`. Ta obsahuje následující složky:

- `assets` – Obsahuje ikony a obrázky používané v aplikaci.
- `components` – Znovupoužitelné komponenty.
- `config` – Různé konfigurační soubory, například soubor s konfigurací spojení s Firebase.
- `modules` – Moduly pro práci s Firebase, Strava API a další pomocné moduly.
- `screens` – Komponenty představující obrazovky aplikace.

Projekt neobsahuje složky `android` a `ios`, jak tomu bývá v React Native projektech, protože je projekt spravován pomocí frameworku Expo, který v jeho módu *managed workflow* neumožňuje psaní nativních modulů pro android a ios.

5.2 Implementace frontendu

Implementace jednotlivých obrazovek je rozdělena do samostatných souborů umístěných do podsložky `Screens` ve složce `App`. Každá obrazovka je reprezentována komponentou. Komponenty jsou vytvořeny způsobem využívajícím funkce a hooky. Soubory s implementací jsou tedy JavaScriptové moduly, které exportují jednu funkci vracející danou komponentu. K tomu je u dané funkce využit výraz `export default`. K zápisu komponent je využito syntaktické rozšíření JavaScriptu `JSX` popsané v sekci 3.2.1.

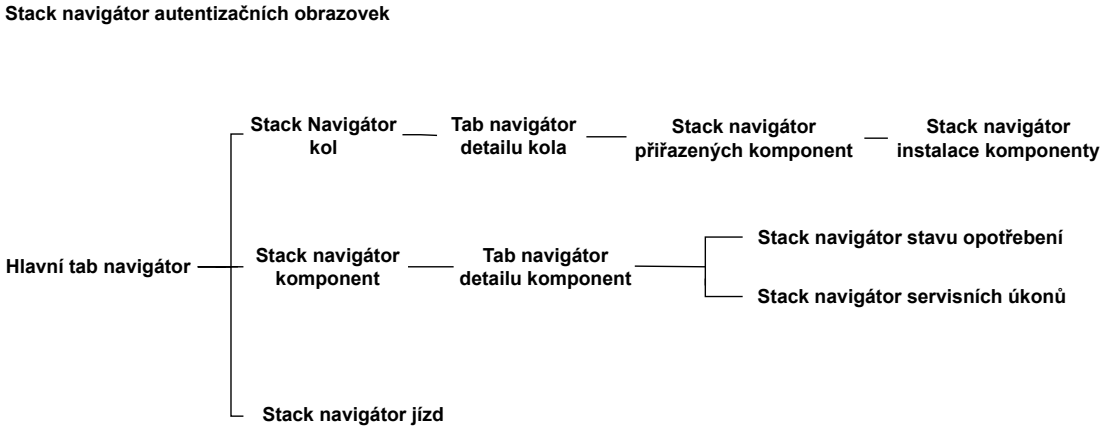
K tvorbě komponent představujících obrazovky jsou využity vestavěné komponenty React Native, vlastní definované komponenty a několik komunitních komponent. Vlastní komponenty jsou uloženy v podsložce `components` a jsou vytvořeny stejně jako jednotlivé obrazovky – každá komponenta je v samostatném souboru a pro tvorbu je využit přístup s funkcemi. U komponent přijímajících nějaké parametry jsou tyto parametry definovány pomocí rozhraní jazyka TypeScript. Komunitní komponenty lze získat pomocí správce balíčků pro JavaScript `npm`. Použité komunitní komponenty v projektu jsou například komponenta pro výběr data a času nebo rozbalovací seznam použitý ve formulářích.

Jednotlivé komponenty reprezentující obrazovky jsou zaregistrovány do navigátorů. Těmito navigátory jsou `StackNavigator` (zásobníkový navigátor) a `TabNavigator` (tabový navigátor), které jsou dostupné v knihovně `React Navigation`. Princip těchto navigátorů je popsán v sekci 3.2.2.

Struktura frontendu

Ve vstupním bodu programu, tedy v souboru `App.tsx`, je nejprve všem komponentám předán kontext přihlášeného uživatele. K tomu je využito `React Context API`[1], které umožňuje předat parametry všem komponentám na všech úrovních ve stromu komponent bez nutnosti předávání dat pomocí `props` na každé úrovni stromu. Všechny komponenty jsou tedy zabaleny do komponenty poskytovatele kontextu. Poté je vytvořena komponenta `NavigationContainer`, dostupná z knihovny `React Navigation`, která je zodpovědná za správu navigátoru na nejvyšší úrovni. Pokud je uživatel přihlášen, je do této komponenty umístěna komponenta tabového navigátoru, v kterém jsou zaregistrovány zásobníkové navigátory obrazovek se seznamem kol, komponent a jízd. Pokud uživatel není přihlášen, je jako navigátor nejvyšší úrovně vytvořen zásobníkový navigátor obsahující přihlašovací a registrační obrazovku. Tyto navigátory jsou do sebe dále zanořovány dle struktury obrazovek aplikace. Kompletní strukturu navigátorů lze vidět na obrázku 5.1.

[1] https://reactjs.org/docs/context.html



Obrázek 5.1: Struktura zanoření stack a tab navigátorů aplikace, které jsou Reactem Native mapovány na nativní navigátory daných platforem. V těchto navigátorech jsou již přímo zaregistrovány komponenty reprezentující jednotlivé obrazovky.

Výpis dat z databáze do komponent

Načítání dat je v jednotlivých obrazovkách prováděno v hooku `useEffect`. Načtené hodnoty jsou ukládány do stavových proměnných, které jsou vytvořeny s použitím hooku `useState`. Načtené hodnoty jsou poté mapovány na příslušné komponenty předáním hodnot pomocí atributů komponent, například v komponentě obrazovky se seznamem kol je každý záznam kola namapován na komponentu `card`.

Pro vylepšení uživatelské zkušenosti je v každé komponentě, ve které jsou načítána data z databáze, definována stavová proměnná `isLoading` datového typu `bool`, na základě jejíž hodnoty je z exportované funkce navrácena animace načítání nebo komponenta s již načtenými hodnotami. Animace načítání je implementována pomocí vestavěné komponenty `ActivityIndicator`.

Data získaná z databáze je třeba aktualizovat při každém zobrazení obrazovky. Toho je docíleno použitím hooku `useIsFocused`, který poskytuje knihovna `React Navigation`. Tento hook vrací pravdivostní hodnotu udávající, zda je uživatel aktuálně na dané obrazovce či nikoliv. Návrácená hodnota z hooku je předána jako závislost hooku `useEffect`, v kterém jsou načítána data z databáze. Tím je zajištěno, že při přechodu zpět na obrazovku jsou hodnoty znovu načteny z databáze. Uložení pomocí stavových proměnných poté zajistí automatické překreslení obrazovky, pokud byla některá z načtených hodnot opravdu změněna.

Formuláře

Obrazovky s formuláři jsou implementovány pomocí knihovny `React Hook Form`. Ta umožňuje jednoduše pracovat s formuláři za pomoci hooků. Jednotlivé uživatelské vstupy jsou definovány jako stavové proměnné pomocí hooku `useState`. Pro akce spojené s formuláři,

jako je odeslání formuláře nebo zobrazení chybové hlášky při chybně zadaném vstupu, je využit hook `useForm`, který je poskytován přímo knihovnou `React Hook Form`. Tomuto hooku jsou také předány parametry nastavující chování formuláře, jako je například mód, který určuje, při jaké akci se kontroluje validita hodnot v uživatelských vstupech.

V obrazovkách s formuláři je v záhlaví umístěna ikona fajfky sloužící pro odeslání formuláře. Vzhled a prvky záhlaví jsou ale specifikovány již v navigátoru, ve kterém je obrazovka zaregistrována, což znemožňuje nastavit na událost kliknutí akci odeslání formuláře, která je dostupná až v komponentě dané obrazovky. Do knihovny React Navigation byla ovšem ve verzi 5 přidána možnost změny nastavení obrazovky přímo uvnitř komponenty dané obrazovky pomocí funkce `setOptions`. Tato funkce je dostupná nad atributem navigation, který je obrazovce předán navigátorem. Akce odeslání formuláře je na událost kliknutí na fajfku tedy nastavena s využitím funkce setOptions, která je volána v hooku `useLayoutEffect`. Tento hook funguje stejně jako hook `useEffect`, ale je volán synchronně před vykreslením prvků obrazovky. Díky tomu je tedy možné změnit nastavení záhlaví již před vykreslením. Hooku `useLayoutEffect` je také jako závislost předána stavová proměnná `isSubmitting`, díky které lze při odesílání formuláře v záhlaví místo ikony fajfky zobrazit animaci načítání.

5.3 Propojení se službami Firebase

Využití Firebase služeb je implementováno s využitím Firebase SDK, které je poskytováno frameworkem Expo. Firebase SDK nepodporuje všechny pokročilejší služby, jako například některé analytické nástroje. Pro implementaci mobilní aplikace jsou ovšem využity pouze služby autentizace, databáze a cloudového úložiště, které Firebase SDK podporuje.

Při využití Firebase SDK je projekt ve Firebase Console vytvořen jako webový. Firebase Console k projektu poskytuje klíč API a další identifikátory, jako je například doména pro autentizaci nebo URI databáze. Pro uložení těchto hodnot byl zvolen způsob využívající uložení do souboru s proměnnými prostředími `.env`. Tyto hodnoty jsou ze souboru `.env` načteny pomocí modulu `dotenv` a do projektu předány využitím vlastnosti `extra` v konfiguračním souboru `app.config.js`. K hodnotám proměnných, které jsou uloženy pod vlastností extra, lze přistoupit pomocí objektu `Constants`, který je exportován modulem `expo-constants`.

Hodnoty identifikátorů Firebase projektu jsou využity pro tvorbu instance aplikace Firebase. Správa instance Firebase aplikace je implementována v modulu `firebase.ts`, který je založen na návrhovém vzoru jedináček. Modul exportuje instanci aplikace Firebase, kterou při importu modulu vytvoří pomocí funkce `initializeApp`, dostupné v modulu `firebase/app`. To ale pouze v případě, že žádná instance ještě neexistuje. Pokud existuje, vrací existující instanci získanou pomocí funkce `getApp`, která je rovněž dostupná v modulu `firebase/app`.

5.3.1 Autentizace pomocí Firebase

Aplikace využívá metodu příhlášení pomocí emailu a hesla. Tuto metodu je nejprve nutné povolit ve Firebase Console.

Funkce spojené s autentizací pomocí Firebase jsou dostupné v modulu `firebase/auth`. K vytvoření nového účtu je použita funkce `createUserWithEmailAndPassword`, která je volána při odeslání formuláře na obrazovce s registračním formulářem. Té je jako parametr předán email uživatele, který slouží jako unikátní identifikátor, heslo a instance aktuálně přihlášeného uživatele. Instanci aktuálně přihlášeného uživatele lze získat pomocí funkce

`getAuth`, které je jako parametr předána instance aplikace Firebase. Funkce pro registraci daného uživatele po vytvoření účtu také přihlásí. Po registraci je účtu přiřazen identifikátor, který je unikátní mezi všemi metodami přihlášení. Email totiž sice slouží jako identifikátor uživatele, ale uživatel může mít přes stejný email vytvořen jak účet prostřednictvím registrace emailem a heslem, tak například pomocí přihlášení přes účet Google.

K přihlášení slouží funkce `signInWithEmailAndPassword`, jejíž parametry jsou stejné jako parametry funkce pro registraci uživatele. Je volána při odeslání formuláře na obrazovce s přihlašovacím formulářem. K odhlášení uživatele slouží funkce `signOut` dostupná nad instancí přihlášeného uživatele.

Při změně stavu přihlášeného uživatele je aktualizován kontext přihlášeného uživatele, který je udržován pomocí React Contextu. Konkrétně jsou aktualizovány dvě stavové proměnné – `IsLoggedIn` typu `bool` a objekt `User` obsahující data o přihlášeném uživateli. Základní data o přihlášeném uživateli lze zjistit i pomocí vlastnosti `currentUser` dostupné nad instancí aktuálně přihlášeného uživatele. Zde jsou ale dostupné pouze informace o účtu v modulu autentizace Firebase. U každého uživatele jsou ale uloženy i další dodatečné informace v databázi Firestore. Právě tato data jsou při přihlášení uživatele nahrávána do stavové proměnné `User`. Na změnu stavu přihlášeného uživatele je reagováno pomocí funkce `onAuthStateChanged`, která je dostupná nad instancí přihlášeného uživatele. Jako parametr je jí předána funkce, která se má při změně stavu přihlášeného uživatele vyvolat.

5.3.2 Databáze Firestore

Jako databáze byla zvolena databáze Firestore. Je vhodnější volbou než realtime databáze, neboť v implementaci jsou využity i složitější dotazy.

Datový model

Data ve Firestore jsou uložena ve formě kolekcí dokumentů. Tyto dokumenty jsou strukturou velmi podobné formátu JSON (datový model databáze Firestore je podrobněji popsán v sekci 3.3.2). Databáze aplikace obsahuje tyto kolekce:

- **bikes** – Obsahuje informace o kolech, jako je jméno kola, datum zakoupení, počet najetých kilometrů a podobně. Každý záznam obsahuje referenci na dokument uživatele, kterému kolo patří.
- **bikesComponents** – Obsahuje záznamy o propojení kol a komponent. Každý dokument obsahuje referenci na kolo a komponentu, datum instalace a případně i datum odinstalace, pokud již byla provedena.
- **componentServiceRecords** – Obsahuje záznamy o servisních úkonech. Každý dokument obsahuje referenci na dokument komponenty, ke které daný záznam patří.
- **componentWearRecords** – Obsahuje záznamy o stavu komponenty. Jednotlivé záznamy obsahují referenci na dokument komponenty, ke které se daný záznam vztahuje. Pokud byla k záznamu o stavu komponenty přiložena fotografie, obsahuje dokument pole `image`, jehož hodnotou je identifikátor této fotografie ve Firebase modulu Storage.
- **components** – Obsahuje záznamy o komponentách. Pokud je komponenta na nějakém kole nainstalována, záznam obsahuje referenci na dokument tohoto kola. Každá komponenta také obsahuje referenci na uživatele, kterému komponenta patří.

- **rides** – Záznamy o jednotlivých jízdách na kolech obsahující informace o najetých kilometrech, času jízdy a další statistiky. Záznam obsahuje referenci na dokument uživatele, který jízdu vytvořil a referenci na kolo, na kterém byla jízda provedena.

- **users** – Obsahuje záznamy o uživatelích. Identifikátor dokumentu každého uživatele odpovídá jeho identifikátoru v modulu autentizace. Pokud byl uživatelský účet vytvořen prostřednictvím autentizace Strava účtem, je součástí záznamu pole `stravaAuth` s hodnotou `True`. Dále obsahuje objekt `stravaInfo`, v kterém se nachází informace o tokenech potřebných pro přístup k datům uživatele prostřenictvím aplikačního rozhraní Stravy. Tento objekt je v záznamu uložen i v případě, že uživatel se Stravou propojil již existující účet.

Některé záznamy jízd a kol mohou být synchronizovány s aplikací Strava, jak je popsáno v sekci 5.4.2. Tyto záznamy obsahují navíc pole `stravaSynced` s hodnotou `true` a pole s identifikátorem jízdy nebo kola, který je získán z aplikačního rozhraní Stravy a slouží pro identifikaci při dalších synchronizacích.

Práce s Firestore v Reactu Native

Funkce pro práci s databází Firestore jsou dostupné v modulu `firebase/firestore`. Jedním z parametrů těchto funkcí je instance databáze Firestore. Tu lze získat pomocí funkce `getFirestore`, která jako parametr přijímá instanci aplikace Firebase. Funkce pro práci s Firestore jsou asynchronní, jejich návratovou hodnotou je objekt Promise.

Nová data lze do databáze přidat pomocí funkcí `addDoc` a `setDoc`. Funkce `addDoc` přijímá jako první parametr referenci na kolekci, do které má být nový dokument přidán a jako druhý parametr objekt s obsahem nového dokumentu. Referenci na kolekci lze získat pomocí funkce `collection`, které je předána instance Firestore a jméno kolekce. Funkce `addDoc` přiřadí automaticky novému dokumentu identifikátor. Při využití `setDoc` je jako první parametr přijímána reference na dokument a jako druhý parametr opět objekt s daty. Referenci na dokument lze získat pomocí funkce `doc`, které je předána instance `Firestore` a jako další parametry název kolekce a identifikátor dokumentu. Pokud dokument s tímto identifikátorem již existuje, jsou data nahrazena. Pokud neexistuje, tak je vytvořen nový dokument. Smazat dokument lze pomocí funkce `deleteDoc`, která přijímá referenci na dokument.

Aktualizovat data lze s použitím funkce `updateDoc`, které je předána reference na dokument a objekt, který obsahuje hodnoty, které se mají změnit. Ostatní hodnoty jsou zachovány. Pro smazání pole z dokumentu lze v objektu předaného funkci `updateDoc` tomuto poli nastavit jako hodnotu volání funkce `deleteField`.

Získat data z databáze lze pomocí funkcí `getDoc` a `getDocs`. Funkce `getDoc` přijímá jako parametr referenci na dokument. `getDocs` vrátí všechny dokumenty, které odpovídají předanému dotazu, který lze vytvořit použitím funkce `query`. `Query` přijímá jako první parametr referenci na kolekci, nad kterou se mají filtrovat dokumenty. Dále je přijímán libovolný počet parametrů definujících podmínky pro vybrání dokumentu pomocí funkce `where`. Té je jako první argument předán název pole, nad kterým má probíhat filtrování, jako druhý argument operátor porovnání, tedy například rovnost nebo větší než, a jako třetí argument hodnota, s kterou se má hodnota pole porovnat. Kromě funkce `where` lze do dotazu přidat `orderBy` pro seřazení výsledku sestupně nebo vzestupně dle hodnoty zvoleného pole nebo `limit` pro omezení počtu vrácených dokumentů.

Příklad dotazu nad databází Firestore lze vidět na obrázku 5.2.

```
export async function getAllStravaSyncedRides()
{
    return getDocs(
        query(
            collection(getFirestore(firebaseApp), "rides"),
            where("stravaSynced", "==", true),
            where("user", "==", doc(getFirestore(firebaseApp), "users", getAuth(firebaseApp).currentUser.uid))
            )
        )
}
```

Obrázek 5.2: Příklad dotazu nad databází Firestore, pomocí kterého jsou vybrány dokumenty z kolekce `rides`, které obsahují pole `stravaSynced` s hodnotou `true` a referenci na dokument uživatele, jehož identifikátor se rovná identifikátoru přihlášeného uživatele.

Firestore dotazy mají několik limitací. Například v dotazu složeném z více where nemůžeme použít operátory pro rozsah nebo neekvalitu nad více než jedním polem. Také není podporován logický or mezi jednotlivými filtry – mohou být pouze aplikovány všechny zároveň.

Všechny Firestore dotazy vyžadují vytvoření indexu. Indexy pro nejzákladnější dotazy jsou vytvářeny automaticky. Indexy pro složené dotazy lze vytvořit buď prostřednictvím Firebase Console a nebo je nechat vygenerovat Firebase. Pokud totiž dotaz selže kvůli chybějícímu indexu, Firebase v chybové hlášce poskytne vývojáři odkaz, pomocí kterého potřebný index může vytvořit.

S využitím těchto funkcí pro manipulaci s daty v databázi Firestore je implementován modul `firestoreActions`. Tento modul obsahuje funkce, které jsou využívány pro zobrazení dat z databáze v jednotlivých obrazovkách a manipulaci s daty v databázi na základě uživatelských akcí. Funkce seskupují jednotlivé dotazy a akce nad databází a zapouzdřují je do kompletních akcí, které jsou prováděny v aplikaci. Pro získávání dat jsou tak například prováděny dotazy, které zajistí, že bude možné přistoupit i k datům dokumentů, na které je v získaném dokumentu pouze reference. Při manipulaci s daty se jedná například o různé úpravy referencí, aby zůstala data v konzistentním stavu. Příkladem může být smazání komponenty, po kterém je nutné rovněž smazat všechny záznamy o výměně komponenty na kolech. Součástí těchto funkcí je také pokročilejší filtrování dokumentů, které není možné správně filtrovat pomocí dotazů kvůli jejich limitacím. Funkce jsou implementovány jako asynchronní.

5.4 Propojení s účtem aplikace Strava

Funkce pro komunikaci s aplikačním rozhraním aplikace Strava jsou implementovány v modulu `stravaApi`.

5.4.1 Autentizace Strava účtem

Aplikace nabízí dva způsoby propojení s účtem Strava. Prvním je přihlášení přímo přes účet Strava, druhým je propojení existujícího účtu se Stravou. V obou případech je nutné projít procesem autorizace. Strava podporuje autorizaci pomocí protokolu OAuth 2.0, jehož princip je popsán v sekci 3.4.

Aplikaci je nejprve nutné zaregistrovat ve webovém rozhraní Stravy. Zde jsou aplikaci přiděleny identifikátory client id a client secret, které jsou poté používány pro identifikaci

aplikace při zasílání požadavků na aplikační rozhraní. Pro uložení těchto hodnot pro potřeby aplikace je využit soubor `.env`.

Proces autorizace je implementován pomocí modulu AuthSession[2], který je dostupný v Expo SDK. Autorizační požadavek je vytvořen pomocí funkce `useAuthRequest`. Té je jako první argument předán objekt obsahující client id aplikace, pole požadovaných oprávnění k datům, ke kterým chceme získat autorizaci a URI, na kterou má být uživatel přesměrován po dokončení autorizace. Druhý argument je objekt obsahující potřebné koncové body aplikačního rozhraní, na které bude autorizační požadavek zaslán. Požadovaná oprávnění jsou nastavena na čtení všech dat uživatelského profilu a čtení všech aktivit. URI k přesměrování je nastavena na bikecomponentsmanager://redirect, neboť adresa aplikace je prostřednictvím konfiguračního souboru `app.config.js` nastavena na bikecomponentsmanager a doména pro přesměrování v aplikaci Strava je nastavena na redirect. Jako koncový bod je nastaven `/oauth/mobile/authorize`. Tento koncový bod zajistí, že je uživatel k poskytnutí autorizace přesměrován do mobilní aplikace Strava, pokud ji má nainstalovanou. Funkce `useAuthRequest` vrací tři hodnoty – instanci vytvořeného požadavku, proměnnou `response`, do které je uložena odpověď po dokončení autorizace a funkci `promptAsync`, jejíž zavoláním je spuštěn proces autorizace. Po dokončení autorizačního procesu je na výsledek reagováno pomocí hooku `useEffect`, kterému je jako závislost předána proměnná `response`, jejíž hodnota je změněna při dokončení autorizace. Instance autorizačního požadavku je vytvořena v přihlašovací obrazovce, kde se lze přihlásit pomocí Stravy a na obrazovkách se seznamem kol a jízd, kde lze svůj existující účet se Stravou propojit.

Při autentizaci pomocí Strava účtu i při propojení existujícího účtu se Stravou je volání funkce `promptAsync` nastaveno na událost stisknutí tlačítka *connect with strava*. Pokud autorizace úspěšně proběhla, je navrácen autorizační token, který je poté třeba vyměnit za přístupový token a obnovovací token. To je provedeno pomocí funkce `exchangeCodeAsync` z modulu `AuthSession`. Tato funkce jako první argument přijímá objekt obsahující autorizační kód, který byl získán z procesu autorizace, client secret a client id. Jako druhý argument je předán koncový body, na který má být požadavek zaslán, kterým je v tomto případě endpoint `/oauth/token`. Výsledkem volání funkce `exchangeCodeAsync` je objekt typu `TokenReponse`, který obsahuje přístupový token, platnost tokenu a obnovovací token, pomocí kterého lze po vypršení platnosti získat nový přístupový token.

Pokud je Strava účet propojován s existujícím účtem, jsou k informacím uživatele v databázi firebase přidány informace o propojení účtu se Stravou a o tokenech potřebných k získání dat z API Stravy.

Pokud se uživatel pomocí Strava účtu přihlašuje, je mu ve skutečnosti na pozadí vytvořen účet v autentizačním modulu Firebase. Po dokončení výměny tokenů jsou nejprve získány data o uživateli zasláním požadavku na koncový bod `/athlete`. Ten vrací informace o uživateli, kterému patří přístupový token. V odpovědi je získán identifikátor Strava účtu uživatele. Ten je využit k vytvoření fiktivního emailu ve formátu *iduzivatele*@stravauser.com, který je použit pro vytvoření uživatelského účtu v autentizačním modulu Firebase. Jako heslo účtu je použit hash z kombinace identifikátoru uživatele a tajného klíče, uloženého v souboru `.env`. S využitím funkce `fetchSignInMethodsForEmail` je zjištěno, zda je pro tento email již účet vytvořen. Pokud účet existuje, je uživatel přihlášen funkcí `signInWithEmailAndPassword` s využitím fiktivního mailu a hesla. Pokud neexistuje, je mu nový účet vytvořen. K tomuto účtu je poté vytvořen i záznam v databázi Firestore, kam jsou uloženy informace o jeho přístupovém a obnovovacím tokenu.

[2] https://docs.expo.dev/versions/latest/sdk/auth-session/

Před každým zasláním požadavku na API Stravy je zkontrolována platnost přístupového tokenu pomocí funkce `tokenExpired` implementované v modulu `stravaApi`. Pokud platnost vypršela, je pomocí funkce `refreshAsync` s použitím obnovovacího tokenu z modulu `AuthSession` získán nový přístupový token. Této funkci jsou předány identifikátory aplikace client id a client secret, obnovovací token a koncový bod, na který má být požadavek zaslán. Tím je v tomto případě koncový bod `/oatuh/token`. Funkce `refreshAsync` vrací objekt typu `TokenResponse`, jehož struktura je již v této sekci popsána.

5.4.2 Synchronizace aktivit a kol z aplikace Strava s databází

Pokud má uživatel propojen účet se Stravou, je synchronizace spuštěna automaticky při spuštění aplikace, případně při jeho přihlášení. Synchronizaci lze také manuálně spustit pomocí možnosti v menu umístěného v záhlaví obrazovek se seznamem kol, jízd a komponent. Synchronizace je implementována ve funkcích `syncBikes` a `syncRides`, které jsou součástí modulu `firestoreActions`.

Synchronizace kol

Při synchronizaci kol jsou nejprve získána všechna kola uživatele, která má uložena ve Stravě. To je provedeno zasláním požadavku na koncový bod `/athlete`. Odpověď obsahuje pod klíčem `bikes` pole kol. Poté jsou získána z databáze kola, která byla synchronizována se Stravou. Pro každé kolo ze Stravy je zkontrolováno, zda kolo se stejným Strava identifikátorem existuje i mezi koly z databáze. Pokud ne, jedná se o nově přidané kolo a je do databáze přidáno. Pokud ano, jsou data u daného kola aktualizována, pokud byla změněna. Nakonec je ověřeno, zda neexistuje nějaké kolo, které je v databázi, ale již není ve Stravě. Pokud je takové kolo nalezeno, je nastaveno do módu `retired`, čímž zůstanou zachovány reference na kolo v předchozích jízdách, ale kolo nebude již v aplikaci zobrazováno.

Synchronizace jízd

Princip synchronizace jízd je obdobný jako synchronizace kol. Nejprve jsou získány záznamy o jízdách z databáze a záznamy o jízdách ze Stravy. Jízdy ze Stravy jsou získány zasláním požadavku na endpoint `/athlete/activities`, který navrací všechny aktivity. Z těch jsou poté vyfiltrovány aktivity typu ride. Následující proces synchronizace je stejný jako u synchronizace kol. Jedinou změnou je, že je navíc nutné řešit správné započítání kilometrů a času z jízd do kol a komponent. Kilometry a čas jsou totiž u komponent uchovávány v databázi a aktualizovány při různých akcích, jako je přidání nové jízdy, úprava jízdy nebo přiřazení komponenty ke kolu. Tyto hodnoty by sice bylo možné vypočítat z jednotlivých jízd a záznamů o instalacích komponent, ale pokud by v aplikaci uživatel již měl desítky komponent, stovky až tisíce jízd a komponenty mezi koly v průběhu používání různě měnil, byly by tyto výpočty již náročné a aplikace by nemusela být tak plynulá. Dalším důvodem jsou také limitace dotazování nad databází Firestore, kvůli kterým by nebylo možné použít pokročilé filtrování a z databáze by muselo být získáváno velké množství nepotřebných dat, která by poté byla zpracovávána v klientské části aplikace. Při synchronizaci jízd jsou tedy řešeny situace, kdy je přidána a odebrána jízda, upraveny hodnoty najetých kilometrů u jízdy, ale i například situace, kdy je u jízdy upravena reference na kolo, na kterém byla jízda provedena – v takovém případě je nutné odečíst kilometry a čas v provozu z komponent starého kola a přičíst je ke komponentám nového kola.

Kapitola 6

Vydání a testování aplikace

Aplikace byla testována na zařízeních s operačním systémem Android a vydána na Google Play. K vytvoření balíčku aplikace pro Google Play je využita funkcionalita expo frameworku `expo build:android`. Pomocí té je spuštěn vzdálený překlad na expo serverech a výsledný sestavený balíček je dostupný ke stažení prostřednictvím webové aplikace expo.dev. Expo framework za vývojáře vytvoří i podpisový klíč a podepíše jím aplikaci. Pro sestavení balíčku využívá expo framework také konfigurační soubor `app.config.js`, z kterého jsou získana data o ikoně aplikace, jejím názvu nebo verzi. Aplikaci lze na Google Play vydat prostřednictvím Google Play Console, kde musí mít vývojář aktivován vývojářský účet. Ten je zpoplatněn jednorázovým poplatkem 25 dolarů.

6.1 Testování s využitím interního testování Google Play

Uživatelské testování mobilní aplikace bylo prováděno pomocí modulu interního testování v Google Play. Tento modul umožňuje poskytnout přístup k aplikaci až 100 zvoleným testerům, kterým je poskytnut přístup zadáním jejich emailu prostřednictvím rozhraní Google Play Console. Vydání pro interní testery nemusí být nijak schvalováno, narozdíl od vydání veřejné verze. Po obdržení zpětné vazby od uživatelů a případného upravení aplikace lze v modulu interního testování přidat nové vydání, čímž se aplikace uživatelům, kteří ji mají nainstalovanou, automaticky aktualizuje.

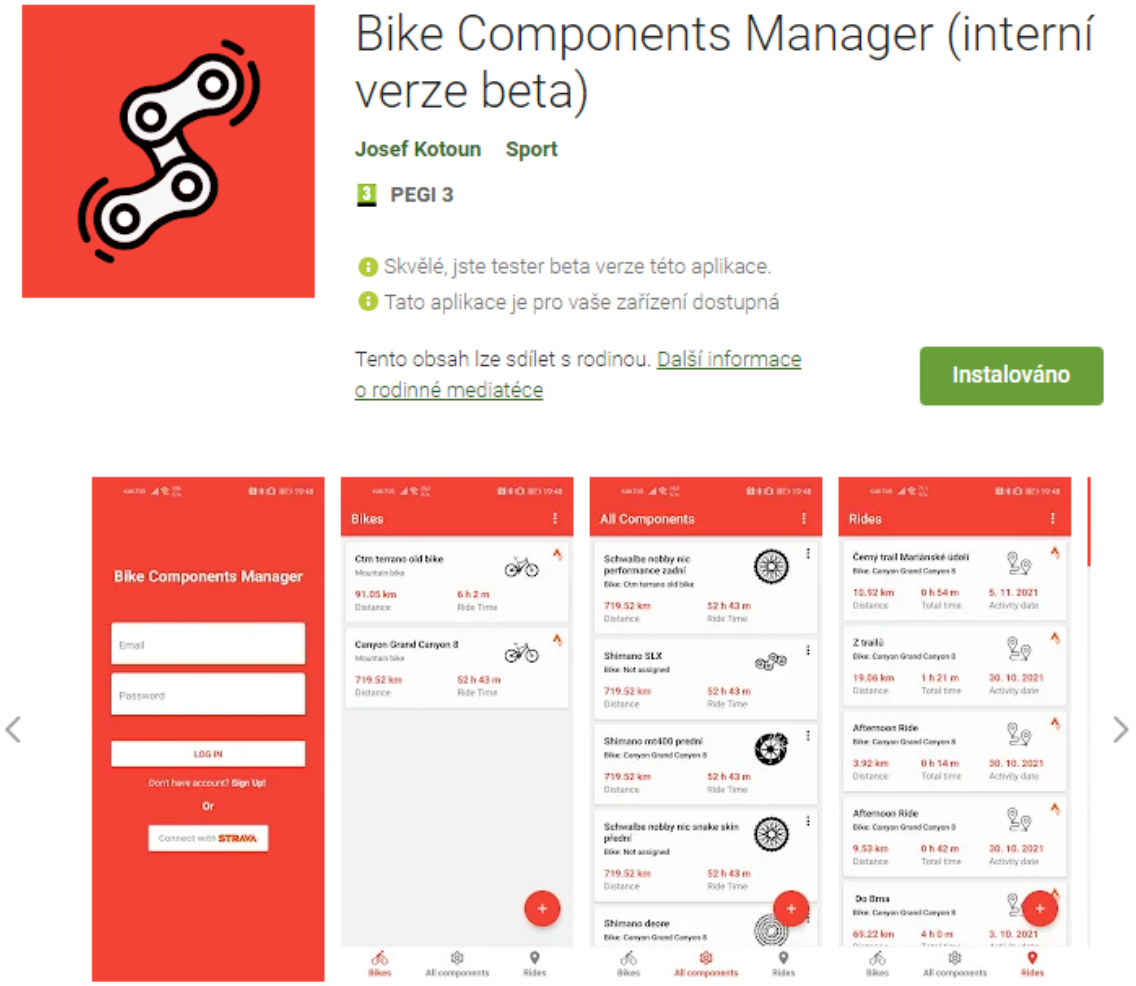
Díky tomuto testování byla odhalena řada chyb související s odlišným chováním a vzhledem aplikace na různých zařízeních. Chyby byly způsobeny především různými velikostmi a rozlišeními obrazovek, kvůli čemuž například na některých zařízeních nebylo možné scrollovat ve formulářích nebo byly některé prvky zobrazeny mimo obrazovku. Tyto chyby byly po získání zpětné vazby od uživatelů v dalších verzích opraveny.

V prostředí Google Play Console lze také sledovat různé analytické údaje. Těmi jsou například počet selhání běhu aplikace nebo sledování problémů s výkonem některých částí aplikace.

6.2 Vydání produkční verze na google play

Produkční verze musí být schválena podporou Google Play. Před vydáním je nutné vyplnit řadu prohlášení o datech, které aplikace shromažďuje, jako například k jakému účelu jsou shromažďována, zda je zadání těchto dat povinné nebo zda je uživateli umožněno zažádat o smazání těchto dat. Dalším krokem je vyplnění hlavního záznamu v Google Play. Ten ob-

sahuje podrobnosti o aplikaci, jako její název a popis, ale také grafiku aplikace, jejíž součástí je ikona aplikace, která bude zobrazena v Google Play, nebo snímky obrazovek z aplikace. Pokud je některá část aplikace přístupná pouze přes uživatelský účet, je vyžadováno vytvoření testovacího účtu a poskytnutí přihlašovacích údajů podpoře Google Play pro potřeby schválení aplikace. Proces schvalování trvá nejvýše 7 dní. Po dokončení procesu schvalování je aplikace ohodnocena IARC ratingem, který určuje, pro jakou věkovou skupinu je aplikace vhodná a je veřejně dostupná na Google Play. Aplikace pro správu komponent jízdního kola byla ohodnocena jako aplikace pro všechny věkové skupiny. Záznam aplikace v Google Play lze vidět na obrázku 6.1.



Obrázek 6.1: Záznam aplikace na Google Play

Kapitola 7

Závěr

Cílem této práce bylo vytvořit mobilní aplikaci pro sledování stavu komponent jízdního kola. Tato aplikace byla vytvořena a publikována na Google Play[1].

Úvodní část práce shrnuje nedostatky konkurenčních aplikací a seznámení s technologiemi, které byly pro vývoj aplikace využity. Dále je popsán návrh uživatelského rozhraní, tvorba prototypu, jeho testování s uživateli a postupné vylepšování na základě zpětné vazby. Poté se práce věnuje implementaci výsledné aplikace, podrobněji je popsáno propojení s aplikací Strava. Poslední kapitola popisuje testování výsledné aplikace s uživateli, úpravy provedené na základě zpětné vazby a vydání finální verze na Google Play.

Výsledkem práce je otestované funkční řešení, pomocí kterého může uživatel spravovat informace o opotřebení jeho komponent. U komponent je možné sledovat jejich nájezd kilometrů, hodiny v provozu, zaznamenávat informace o jejich stavu a zaznamenávat provedené servisní úkony. Dále uživatel může spravovat jeho kola, jízdy na kolech a vidět historii výměn komponent mezi koly.

Do budoucna bych chtěl aplikaci také publikovat na Apple App Store a celkově ji zdokonalovat na základě zpětné vazby uživatelů.

Díky práci na tomto projektu jsem se naučil pracovat s technologiemi pro vývoj mobilních aplikací, technologií Firebase a prototypovacím nástrojem Figma. Také jsem se v průběhu návrhu aplikace dozvěděl více o uživatelském testování a poprvé takové testování sám prováděl.

[1] https://play.google.com/store/apps/details?id=com.bike_components_manager

Literatura

[1] Curry, D. *Strava Revenue and Usage Statistics (2022)* [online]. [cit. 2022-04-11]. Dostupné z: https://www.businessofapps.com/data/strava-statistics/.

[2] Edpresso. *What is Firebase?* [online]. [cit. 2022-04-10]. Dostupné z: https://www.educative.io/edpresso/what-is-firebase.

[3] Eisenman, B. *Learning React Native: Building Native Mobile Apps with JavaScript*. 2. vyd. O'Reilly Media, 2017. ISBN 978-1491989142.

[4] Expo. *Expo CLI* [online]. [cit. 2022-04-09]. Dostupné z: https://docs.expo.dev/workflow/expo-cli/.

[5] Expo. *Workflows* [online]. [cit. 2022-04-18]. Dostupné z: https://docs.expo.dev/introduction/managed-vs-bare/.

[6] Feroze, U. *React Native CLI vs Expo CLI — Which one do I choose?* [online]. [cit. 2022-04-09]. Dostupné z: https://levelup.gitconnected.com/react-native-cli-vs-expo-cli-which-one-do-i-choose-bdf02ea457bf.

[7] Google. *Cloud Firestore Data model* [online]. [cit. 2022-04-21]. Dostupné z: https://firebase.google.com/docs/firestore/data-model.

[8] Google. *Get started with Cloud Firestore Security Rules* [online]. [cit. 2022-04-20]. Dostupné z: https://firebase.google.com/docs/firestore/security/get-started.

[9] Krug, S. *Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability Problems*. 1. vyd. New Riders, 2009. ISBN 978-0321657299.

[10] Marsh, J. *UX for Beginners: A Crash Course in 100 Short Lessons*. 1. vyd. O'Reilly Media, 2015. ISBN 978-1491912683.

[11] Meta Platforms. [online]. [cit. 2022-04-10]. Dostupné z: https://reactjs.org/docs/getting-started.html.

[12] Meta Platforms. [online]. [cit. 2022-04-20]. Dostupné z: https://reactnative.dev/docs/getting-started.

[13] Microsoft. *RESTful web API design* [online]. [cit. 2022-04-12]. Dostupné z: https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design.

[14] Microsoft. *The TypeScript Handbook* [online]. [cit. 2022-04-07]. Dostupné z: https://www.typescriptlang.org/docs/handbook/intro.html.

[15] OAuth0. *What is OAuth 2.0?* [online]. [cit. 2022-04-12]. Dostupné z: https://auth0.com/intro-to-iam/what-is-oauth-2/.

[16] Paterska, P. *What is React Native And When to Use It For Your App?* [online]. [cit. 2022-04-07]. Dostupné z: https://www.elpassion.com/blog/what-is-react-native-and-when-to-use-it.

[17] Richer, J. a Sanso, A. *OAuth 2 in Action.* 1. vyd. Manning, 2017. ISBN 978-1617293276.

Příloha A

Plakát

Obrázek A.1: Propagační plakát aplikace umístěné na platformě Google Play pod názvem Bike Components Manager.