

POLITECHNIKA WROCŁAWSKA
WYDZIAŁ INFORMATYKI I
TELEKOMUNIKACJI

KIERUNEK: INFORMATYKA

SPECJALNOŚĆ: GRAFIKA I SYSTEMY MULTIMEDIALNE

PRACA DYPLOMOWA
INŻYNIERSKA

Aplikacja internetowa do zarządzania trasami
rowerowymi i analizy ich statystyk wraz
z aplikacją mobilną do zapisu danych trasy
z wykorzystaniem modułu GPS

Web application for the management of bicycle
routes and the analysis of their statistics, along
with a mobile application for recording route
data using the GPS module

AUTOR:

Michał Tkacz

PROWADZĄCY PRACĘ:
dr inż. Marek Woda, K3oWo4ND03

OCENA PRACY:

Streszczenie

Celem pracy jest stworzenie aplikacji internetowej udostępniającej fundamentalne funkcje komputera rowerowego. Aplikacja ma być przystosowana do działania zarówno na urządzeniach desktopowych jak i urządzeniach mobilnych. Do podstawowych funkcji aplikacji należą: uwierzytelnianie użytkownika, nagrywanie śladów przejechanych tras rowerowych z wykorzystaniem modułu GPS (ang. *Global Positioning System*), przeglądanie zapisanych śladów, analiza statystyk zapisanych śladów oraz możliwość importu i eksportu danych z wykorzystaniem standardu GPX (ang. *GPS Exchange Format*). Wyróżnikiem tej aplikacji na tle dostępnych na rynku rozwiązań jest to, że jest to aplikacja internetowa typu PWA (ang. *Progressive Web App*), co umożliwia zainstalowanie jej na urządzeniach mobilnych i sprawienie wrażenia aplikacji natywnej. Niniejszy dokument zawiera podstawowe wyjaśnienie czym jest komputer rowerowy oraz przedstawia szczegóły techniczne projektu.

Słowa kluczowe: aplikacja internetowa, komputer rowerowy, nawigacja rowerowa, GPS, GPX

Abstract

The goal of the work is to create a web application providing the fundamental functions of a cycling computer. The application should be available for both desktop and mobile devices. The basic functions of the application shall be as follows: user authentication, cycling tracks recording using the GPS (*Global Positioning System*), recorded tracks browsing, recorded tracks statistics analysis and the data import and export in the GPX (*GPS Exchange Format*) standard format. The main feature that distinguishes this application from already available products is the use of PWA (*Progressive Web App*), which makes it possible to install web application on mobile devices and simulate the experience of a native mobile application. This document provides a basic explanation of what cycling computer is and provides technical details of the project.

Keywords: web application, cycling computer, cycling navigation, GPS, GPX

Spis treści

1. Wstęp	9
1.1. Wprowadzenie	9
1.2. Cel i zakres pracy	9
1.3. Układ pracy	10
2. Koncepcja projektu	11
2.1. Podłożę teoretyczne	11
2.1.1. Komputer rowerowy	11
2.1.2. GPS	12
2.1.3. GPX	13
2.2. Zarys wymagań	15
2.3. Wymagania funkcjonalne	15
2.4. Wymagania niefunkcjonalne	17
2.5. Diagram przypadków użycia	18
2.6. Architektura systemu	19
2.7. Warstwa backend	20
2.7.1. Uwierzytelnianie	21
2.7.2. Baza danych	21
2.7.3. Hosting	23
2.8. Warstwa frontend	23
3. Implementacja projektu	25
3.1. Inicjalizacja aplikacji klienta	25
3.2. Inicjalizacja Firebase	27
3.3. Integracja z Firebase	27
3.4. Integracja z Firebase Hosting	29
3.5. Konfiguracja PWA	30
3.6. Projekt struktury bazy danych	31
3.7. Implementacja układów stron	33
3.8. Implementacja uwierzytelniania	33
3.9. Implementacja routingu	38
3.10. Implementacja rejestratora GPS	39
3.11. Implementacja importu GPX	42
3.12. Implementacja eksportu GPX	44
3.13. Implementacja aktywności	45
3.14. Implementacja statystyk aktywności	46
3.15. Implementacja wykresów aktywności	49
3.16. Integracja z Realtime Database	50
3.17. Implementacja profilu użytkownika	53
3.18. Implementacja statystyk profilu użytkownika	54
3.19. Implementacja wykresów profilu użytkownika	55

4. Podsumowanie	58
Literatura	60
A. Instrukcja wdrożeniowa	62
B. Opis załączonej płyty CD/DVD	63

Spis rysunków

2.1. Diagram przypadków użycia	18
2.2. Schemat architektury systemu	19
3.1. Drzewo plików projektu: a) po inicjalizacji, b) po usunięciu zbędnych plików	26
3.2. Widok głównej strony Firebase Console	27
3.3. Obiekt konfiguracyjny w Firebase Console	28
3.4. Ikony aplikacji: a) logo.svg, b) logo_maskable.svg	31
3.5. Szablon układu strony autoryzacji w wersji desktopowej	34
3.6. Widok strony autoryzacji w wersji desktopowej	34
3.7. Układ strony autoryzacji a) szablon, b) widok aplikacji	35
3.8. Szablon globalnego układu strony w wersji mobilnej i desktopowej	36
3.9. Widok strony aktywności w wersji desktopowej	36
3.10. Schemat strony rejestratora śladów GPS	40
3.11. Diagram sekwencji dla funkcji updateTrack	42
3.12. Widok strony nagrywania rejestratora z włączonym nagrywaniem śladu	42
3.13. Schematy procesów: a) importu aktywności, b) eksportu aktywności	44
3.14. Widok aktywności	46
3.15. Widok wykresów aktywności	50
3.16. Widok strony profilu użytkownika	54
3.17. Widok statystyk profilu użytkownika	56
3.18. Widok wykresów profilu użytkownika	57

Spis tabel

2.1. Porównanie koncepcji architektury systemu i dobór technologii	19
2.2. Moduły platformy Firebase	20
2.3. Główne biblioteki wykorzystane przy tworzeniu aplikacji klienta	24
3.1. Opis pliku <code>manifest.json</code>	30
3.2. Komunikacja z bazą danych	50
B.1. Struktura plików projektu	64

Spis listingów

2.1. Struktura pliku GPX	14
2.2. Przykładowy plik GPX	14
2.3. Struktura zastosowanej bazy danych w postaci pseudokodu	22
2.4. Przykład zasady definiującej dostęp do bazy danych	23
3.1. Inicjalizacja projektu React oraz instalacja kluczowych bibliotek	26
3.2. Integracja aplikacji klienta z projektem Firebase	28
3.3. Plik <code>.firebaserc</code>	29
3.4. Plik <code>firebase.json</code>	29
3.5. Plik <code>manifest.json</code>	30
3.6. Typy obiektowe struktury bazy danych (1/3) — <i>root</i> bazy danych oraz użytkownik .	31
3.7. Typy obiektowe struktury bazy danych (2/3) — aktywności	32
3.8. Typy obiektowe struktury bazy danych (3/3) — ślady	32
3.9. Typy wyliczeniowe struktury bazy danych	33
3.10. Komponent uwierzytelniania	37
3.11. Autoryzacja z użyciem biblioteki <code>FirebaseUI</code>	37
3.12. Komponent kontekstu uwierzytelniania	37
3.13. Struktura aplikacji oraz routing aplikacji	38
3.14. Publiczna trasa routingu	39
3.15. Prywatna trasa routingu	39
3.16. Strona rejestratora GPS	40
3.17. Import aktywności z pliku GPX	43
3.18. Eksport aktywności do pliku GPX	44
3.19. Funkcje: edycji, usunięcia oraz eksportu aktywności	45
3.20. <i>Hook useActivityStatistics</i>	47
3.21. Funkcja <code>geoMove</code>	48
3.22. Funkcja <code>geoDistance</code>	48
3.23. Fragment funkcji generowania serii danych dla wykresów aktywności	49
3.24. <i>Hook</i> odczytu danych profilu z bazy danych	51
3.25. Funkcja zapisu aktywności i ślada do bazy danych	51
3.26. Funkcja usunięcia aktywności i ślada z bazy danych	52
3.27. Funkcja aktualizacji danych profilu	52
3.28. Reguły zabezpieczeń bazy danych	53
3.29. Aktualizacja zdjęcia oraz imienia i nazwiska profilu	53
3.30. Aktualizacja danych profilu na przykładzie zmiany opisu	53

Skróty

- API** (ang. *Application Programming Interface*)
- BaaS** (ang. *Backend-as-a-Service*)
- DBaaS** (ang. *Database-as-a-Service*)
- GPS** (ang. *Global Positioning System*)
- GPX** (ang. *GPS Exchange Format*)
- HTML** (ang. *HyperText Markup Language*)
- HTTP** (ang. *Hyper Text Transfer Protocol*)
- HTTPS** (ang. *Hyper Text Transfer Protocol Secure*)
- PWA** (ang. *Progressive Web App*)
- RWD** (ang *Responsive Web Design*)
- SDK** (ang *Software Development Kit*)
- SPA** (ang *Single-page Application*)
- WGS 84** (ang. *World Geodetic System '84*)
- XML** (ang. *Extensible Markup Language*)

Rozdział 1

Wstęp

Niniejszy rozdział stanowi wstęp do pracy. W najbliższej sekcji przedstawiono genezę projektu. Następnie wyjaśniono ogólny cel pracy oraz przewidziany zakres zadań, które powinny zostać wykonane. Na końcu rozdziału opisano strukturę dokumentu oraz zawartość poszczególnych rozdziałów.

1.1. Wprowadzenie

Głównymi źródłami inspiracji niniejszej pracy są przede wszystkim dwie komercyjne aplikacje (dostępne w wersjach mobilnych oraz przeglądarkowych) — Komoot oraz Strava. Aplikacje te udostępniają funkcjonalności komputera rowerowego i pozwalają na m.in.: nawigację rowerową, zapisywanie śladów GPS przejechanych tras rowerowych oraz analizę statystyk. Bazując na doświadczeniu zebranym podczas użytkowania tych oraz podobnych aplikacji dostępnych na rynku, w ramach tego projektu powstała nowa aplikacja, udostępniająca podobne możliwości.

W ramach samej pracy, aby sprostać ograniczeniom czasowym, projekt został okrojony do fundamentalnych funkcjonalności, do których należą: uwierzytelnianie użytkownika, nagrywanie śladów przejechanych tras rowerowych, przeglądanie zapisanych śladów, analiza statystyk zapisanych śladów statystyk użytkownika oraz możliwość importu i eksportu danych. Na etapie projektowania przewidziane zostały ewentualne, dodatkowe funkcjonalności, które jednak nie zostały zaimplementowane.

1.2. Cel i zakres pracy

Celem pracy jest stworzenie aplikacji internetowej dedykowanej dla rowerzystów. Aplikacja powinna być dostosowana do działania zarówno na dużych ekranach komputerów oraz laptopów, jak i mniejszych ekranach urządzeń mobilnych — smartfonów oraz tabletów. Wymóg ten zostanie pokryty przez zaimplementowanie aplikacji internetowej z zastosowaniem technik RWD (ang *Responsive Web Design*) oraz PWA (ang. *Progressive Web App*).

Użytkownik aplikacji powinien mieć możliwość utworzenia osobistego konta, które pozwoli na przechowywanie własnych danych na serwerze oraz dostęp do nich z dowolnego urządzenia. Założenie konta zagwarantuje pełny dostęp do funkcjonalności aplikacji. Dwie główne zakładane funkcjonalności to rejestrowanie śladu podróży rowerowej z wykorzystaniem systemu GPS oraz obliczanie i prezentacja statystyk podróży w postaci liczb oraz wykresów. Ponadto aplikacja powinna umożliwiać eksport i import danych w standardzie GPX, dzięki czemu dane będą kompatybilne z innymi, podobnymi aplikacjami.

Szczegóły techniczne dotyczące architektury projektu omówiono w Rozdziale 2, natomiast generalny zakres pracy jest następujący:

- Zaprojektowanie architektury systemu
- Zaprojektowanie i implementacja bazy danych oraz zabezpieczenie danych
- Zaprojektowanie i implementacja warstwy back-end systemu
- Zaprojektowanie i implementacja warstwy front-end aplikacji internetowej
- Zaprojektowanie i implementacja warstwy front-end aplikacji mobilnej
- Zapewnienie spójności działania projektu

1.3. Układ pracy

Praca składa się z czterech rozdziałów oraz dwóch dodatków. W Rozdziale 1 zawarto generalne informacje dotyczące celu i zakresu pracy. Rozdział 2 stanowi natomiast wprowadzenie teoretyczne, w zakresie którego wyjaśniono pojęcia stojące u podstaw pracy: komputer rowerowy, GPS oraz GPX. Ponadto przedstawiono listę wymagań funkcjonalnych oraz koncepcję architektury systemu. Implementację programową opisano w Rozdziale 3. Ostatni Rozdział 4 stanowi podsumowanie procesu tworzenia pracy oraz analizę rezultatu pracy. W Dodatku A znajduje się instrukcja wyjaśniająca jak uruchomić aplikację w środowisku lokalnym w trybie deweloperskim. Dodatek B to opis zawartości dołączonej płyty CD/DVD, w tym schemat struktury plików projektu.

Rozdział 2

Koncepcja projektu

Na początku rozdziału przedstawiono podłoże teoretyczne pracy. Następnie dokonano porównania między koncepcją pierwotną, a finalną architektury systemu. W kolejnych podrozdziałach zawarto uzasadnienie decyzji projektowych finalnej struktury systemu oraz wyjaśniono podstawy technologii wybranych do realizacji poszczególnych zadań. Implementacje opisano w rozdziale 2.

2.1. Podłoże teoretyczne

W poniższej sekcji przedstawiono podłoże teoretyczne trzech zagadnień: komputer rowerowy, GPS oraz GPX. Koncepcje te stoją u podstaw całej pracy. Z tego też względu, zaznajomienie się z poniższymi informacjami jest wskazane przed przystąpieniem do lektury dalszych rozdziałów.

2.1.1. Komputer rowerowy

Elektroniczny komputer rowerowy, potocznie zwany również licznikiem rowerowym, to w ogólnym ujęciu urządzenie przeznaczone do rejestracji oraz prezentowania statystyk podróży rowerowej. W swojej idei przypomina samochodową tablicę rozdzielczą. Początkowo tego typu urządzenia miały dość ograniczone możliwości, które głównie sprowadzały się do obliczania i wyświetlania: prędkości z jaką w danej chwili przemieszcza się rower, przejechanego całkowitego dystansu czy obecnej godziny. Z biegiem czasu liczniki były wzbogacane w nowe funkcje, jak np.: obliczanie prędkości średniej, obliczanie czasu podróży, wskazanie kadencji, odczyt temperatury powietrza, itp. Zasada działania prostego licznika rowerowego oparta jest na trzech głównych elementach:

1. Magnes zamontowany na szprysze koła roweru.
2. Czujnik reagujący na magnes, zamontowany na ramie roweru tak, aby przy obrocie koła magnes przemieszczał się w odległości ok. 1cm od czujnika.
3. Urządzenie elektroniczne z wyświetlaczem zamontowane najczęściej na kierownicy.

Prędkość z jaką porusza się rower obliczana jest przez urządzenie na podstawie liczby obrotów koła zarejestrowanych przez czujnik w określonej jednostce czasu oraz zadanej średnicy koła. Inne dane pobierane są z dodatkowych czujników np.: temperatura — wbudowany elektroniczny termometr, kadencja — czujnik magnetyczny z magnesem zamontowanym do pedałów, itp.

Za przełom w dziedzinie komputerów rowerowych można uznać wyposażenie licznika w wydajny mikroprocesor, większy wyświetlacz oraz moduł GPS (ang. *Global Positioning System*). Znaczaco rozszerzyło to listę dostępnych funkcji. Informacja o pozycji urządzenia pozwoliła na

wykorzystanie map cyfrowych, na których w czasie rzeczywistym wyświetlana jest obecna pozycja. W związku z tym pojawiła się również możliwość uprzedniego wytyczania tras, a następnie nawigowania rowerzysty w trakcie podróży. Ponadto na podstawie samych danych lokalizacyjnych można w prosty sposób, na bieżąco wyliczać szereg statystyk, jak np.: prędkość, przejechany dystans, położenie geograficzne (szerokość geograficzna, długość geograficzna, wysokość nad poziomem morza), bez konieczności stosowania dodatkowych czujników. Wszystkie te dane mogą zostać zapisane w celu archiwizacji oraz późniejszej analizy.

Początkowo, aby korzystać z funkcji komputera rowerowego, konieczne było zaopatrzenie się w dedykowane urządzenie, jednak obecnie równie dobrą alternatywą jest użycie smartfona z zainstalowaną odpowiednią aplikacją. Jest to rozwiązanie bardziej przystępne dla przeciętnego użytkownika z dwóch głównych przyczyn:

1. Współcześnie znakomita większość osób jest posiadaczem smartfona [1].
2. Nie wymaga inwestycji finansowej (choć aplikacje zazwyczaj oferują dodatkowe, rozbudowane, płatne funkcjonalności).

Aplikacji tego typu dostępnych na rynku jest wiele. Do najpopularniejszych należą m.in.: Strava, Cyclemeter, MapMyRide, Bikemap, Komoot, czy Wahoo Fitness. Mimo że wszystkie udostępniają fundamentalne funkcjonalności komputera rowerowego, to różnią się one w zakresie funkcjonalności dodatkowych. Część z nich skierowana jest bardziej w stronę organizacji treningów (np. Strava), inne specjalizują się w planowaniu tras (np. Komoot), a jeszcze inne skoncentrowane są na analizie statystyk (np. Cyclemeter).

Zarówno dedykowane urządzenia, jak i aplikacje na smartfony do eksportu oraz importu danych posługują się zazwyczaj standardem GPX. Dzięki temu użytkownicy w razie własnych upodobań oraz potrzeb mogą w dość łatwy sposób migrować między poszczególnymi usługami.

2.1.2. GPS

Zarówno dedykowane komputery rowerowe, jak i smartfony z zainstalowanymi aplikacjami wykorzystują system GPS. GPS (ang. *Global Positioning System*) jest to system nawigacji satelitarnej opracowany w latach 70. XX w. przez Departament Obrony Stanów Zjednoczonych, który swoim zasięgiem obejmuje całą kulę ziemską. Zadaniem GPSa jest dostarczanie do odbiornika informacji o jego obecnym położeniu. Na system składają się trzy główne, ściśle współpracujące ze sobą segmenty [11]:

1. Segment naziemny — naziemne stacje nadawcze.
2. Segment kosmiczny — satelity na orbicie Ziemi.
3. Segment użytkownika — odbiorniki na powierzchni Ziemi lub w stosunkowo niewielkiej odległości niej.

Stacje nadawcze na Ziemi rozmieszczone są w taki sposób, aby każdy satelita jest zawsze w zasięgu minimum dwóch nadajników. Zadaniem stacji jest przesyłanie do satelitów sygnałów korekcyjnych czasu i pozycji.

Satelity rozmieszczone są w taki sposób, aby w każdym punkcie na Ziemi dostępny był sygnał z co najmniej czterech z nich. Według danych dostępnych w listopadzie 2021 roku w skład konstelacji wchodzi 30 operacyjnych satelitów [13]. Ich zadaniem jest nadawanie sygnału radiowego, który zawiera informacje o obecnym wzajemnym położeniu satelitów, o teoretycznej trajektorii lotu oraz odchyleniach od niej.

Odbiornik na powierzchni ziemi na podstawie odebranych sygnałów z minimum czterech satelitów określa odległość od nich, a następnie wylicza swoją pozycję geograficzną — standaryzowaną w układzie odniesienia WGS-84 (ang. *World Geodetic System '84*). Czynniki, które mogą zakłócić sygnał, a co za tym idzie wpływać na jakość określenia pozycji, to m.in.:

- opóźnienia propagacji sygnału spowodowane przez atmosferę ziemską;

- odbicia sygnału od obiektów na Ziemi (np. wysokie budynki w mieście);
- mała liczba dostępnych w zasięgu satelitów (im więcej, tym lepiej);
- wzajemne położenie satelitów (im „szerzej” są rozstawione satelity, tym lepiej);
- odchylenie położenia satelitów od wyznaczonej trajektorii lotu;
- błędy zegara (niedokładne wskazanie czasu).

Pierwotnie system GPS utworzony został z myślą o zastosowaniach militarnych, jednak ostatecznie został udostępniony również dla cywilów oraz do zastosowań komercyjnych. Ponadto należy zaznaczyć, że równolegle do systemu GPS funkcjonują analogiczne systemy nawigacji satelitarnej, jak np.: rosyjski GLONASS [10], europejski Galileo [8], czy chiński BeiDou [3].

2.1.3. GPX

GPX (ang. *GPS Exchange Format*) [18] to otwarty, ustandaryzowany schemat języka XML (ang. *Extensible Markup Language*) opracowany w celu ujednolicenia procesu transferu danych pomiędzy urządzeniami, aplikacjami i usługami wykorzystującymi system GPS. Obecna wersja standardu to wydany 9 sierpnia 2004 roku GPX 1.1.

Dzięki wykorzystaniu architektury XML poszczególne dane lokalizacyjne uzyskane z systemu GPS zapisywane są w korespondujących tagach XML, a te z kolei stanowią zawartość plików, których domyślnym rozszerzeniem jest *.gpx. Pliki te można w prosty sposób eksportować, przenosić, udostępniać i importować pomiędzy różnymi urządzeniami. Standard zapewnia, że dane zawsze będą poprawne i jednakowo interpretowane na dowolnej platformie.

W standardzie zdefiniowane zostały następujące typy danych:

- gpxType;
- typy proste: latitudeType, longitudeType, degreesType, dgpsStationType, fixType;
- typy złożone: metadataType, wptType, rteType, trkType, extensionsType, trksegType, copyrightType, linkType, emailType, personType, ptType, ptsegType, boundsType.

Spośród powyższej listy, na szczególne omówienie zasługują trzy typy złożone: wptType — *waypoint type*, rteType — *route type* oraz trkType — *track type*:

1. *Waypoint type* to pojedynczy punkt należący do kolekcji punktów, ale nieskorelowany z pozostałymi punktami tej kolekcji. Punkt taki zawiera w sobie dane o lokalizacji, tj.: szerokość (latitudeType) i długość (longitudeType) geograficzną oraz może zawierać dodatkowe informacje, jak np.: wysokość nad poziomem morza (elevation), czas, komentarz tekstowy i inne.
2. *Route type*, co można przetłumaczyć jako trasa lub droga. Jest to uporządkowana lista punktów, które prowadzą do określonego celu.
3. *Track type*, czyli ścieżka, droga lub ślad, to lista zawierająca co najmniej jeden segment (pewien fragment ścieżki), który z kolei zawiera listę punktów. Punkty te odzwierciedlają realną ścieżkę, którą przebył odbiornik GPS.

Kluczowe jest rozróżnienie między *route type* i *track type*, pierwszy z nich wytycza jedynie punkty pośrednie prowadzące do celu, drugi z nich to zapis rzeczywistych danych zebranych przez fizyczne urządzenie. Ponieważ polskie tłumaczenia tych angielskich terminów mogą nie być w pełni jednoznaczne, bezpieczniej jest posługiwać się angielskimi formami. Aby uniknąć nieporozumień, w dalszej treści pracy przyjęto następującą konwencję nazewnictwa: *route* — trasa, *track* — ślad.

Szablonową strukturę standardu GPX [17] przedstawiono na listingu 2.1 natomiast na listingu 2.2 pokazano zawartość przykładowego pliku (fragment rzeczywistych danych zebranych

przez autora pracy, wyeksportowanych z aplikacji mobilnej Komoot). Z listingu można odczytać, że plik zawiera jeden *track*, która składa się z jednego segmentu (*trkseg*) z listą punktów (*trkpt*). Lista ta na potrzeby demonstracji została skrócona do trzech punktów, jednak w rzeczywistości, w zależności od długości *tracka*, może ona liczyć setki, a nawet tysiące punktów.

Listing 2.1: Struktura pliku GPX

```
<gpx
  version="1.1.[1]?"
  creator="xsd:string.[1]?"
  <metadata> metadataType </metadata> [0..1] ?
  <wpt> wptType </wpt> [0..*] ?
  <rte> rteType </rte> [0..*] ?
  <trk> trkType </trk> [0..*] ?
  <extensions> extensionsType </extensions> [0..1] ?
</gpx>
```

Listing 2.2: Przykładowy plik GPX

```
<?xml version='1.0' encoding='UTF-8'?>
<gpx version="1.1" creator="https://www.komoot.de" xmlns="http://www.
  ↳ topografix.com/GPX/1/1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  ↳ instance" xsi:schemaLocation="http://www.topografix.com/GPX/1/1=http:
  ↳ //www.topografix.com/GPX/1/1/gpx.xsd">
  <metadata>
    <name>This is the track name</name>
    <author>
      <link href="https://www.komoot.de">
        <text>komoot</text>
        <type>text/html</type>
      </link>
    </author>
  </metadata>
  <trk>
    <name>This is the track name</name>
    <trkseg>
      <trkpt lat="51.191923" lon="17.014499">
        <ele>156.033586</ele>
        <time>2021-10-29T17:11:11.876Z</time>
      </trkpt>
      <trkpt lat="51.192004" lon="17.014368">
        <ele>156.155631</ele>
        <time>2021-10-29T17:11:13.870Z</time>
      </trkpt>
      <trkpt lat="51.192077" lon="17.014230">
        <ele>156.275396</ele>
        <time>2021-10-29T17:11:15.873Z</time>
      </trkpt>
      [...]
    </trkseg>
  </trk>
</gpx>
```

Dzięki otwartości standardu możliwe jest jego rozszerzanie. Jednym z popularnych rozszerzeń jest *Garmin GPX Schema Extension*. Dodaje ono nowe typy danych, które pozwalają zapisać w pliku dodatkowe informacje. Przykładowo, w omówionym wcześniej *waypoint type* można utrwalic zarejestrowaną w danym punkcie temperaturę lub kategorię do której dany punkt należy. Z kolei *track type* zyskuje możliwość przechowywania koloru, w jakim ślad powinien zostać wyświetlony na mapie. Wszystkie funkcjonalności rozszerzenia opisano w oficjalnym schemacie [9].

2.2. Zarys wymagań

W wyniku zebrania wymagań projektowych powstała lista ogólna lista wymagań użytkowych, którą podzielono na dwie kategorie: podstawowe oraz dodatkowe. Spełnienie wymagań z kategorii podstawowych uznano za równoznaczne z osiągnięciem wyznaczonego celu pracy. Kategorii wymagań dodatkowych nie przewidziano do realizacji w ramach pracy. Przedstawia ona jedynie pomysły i kierunek dalszego rozwoju. Kompletna lista jest następująca:

1. Wymagania podstawowe:

- Uwierzytelnianie użytkownika.
- Nagrywanie śladów GPS (standard GPX).
- Import śladów GPS (pliki *.gpx).
- Eksport śladów GPS (pliki *.gpx).
- Statystki śladu GPS (liczby, wykresy, prezentacja na mapie).
- Statystki ogólne użytkownika (liczby, wykresy).

2. Wymagania dodatkowe:

- Edycja profilu (np.: zmiana danych personalnych).
- Mapa cieplna aktywności (ang. *Heat Map*).
- Planowanie dróg z wykorzystaniem mapy.
- Nawigacja według zaplanowanych dróg.
- Podstawowe elementy mediów społecznościowych (przeglądanie profili użytkowników, udostępnianie aktywności).
- Rozbudowane funkcjonalności mediów społecznościowych (system znajomych, komentarze, wydarzenia grupowe, współdzielone planowanie tras, współdzielona nawigacja).

2.3. Wymagania funkcjonalne

Na poniższej liście zawarto szczegółowe wymagania funkcjonalne aplikacji:

1. Uwierzytelnianie:

- 1.1. Rejestracja z użyciem adresu email i hasła.
- 1.2. Logowanie z użyciem adresu email i hasła.
- 1.3. Autoryzacja z użyciem konta Google.
- 1.4. Wylogowanie z aplikacji.

2. Nagrywanie śladów GPS:

- 2.1. Podgląd mapy.
- 2.2. Włączenie rejestrowania śladu GPS.
- 2.3. Wyłączenie rejestrowania śladów GPS.
- 2.4. Zapisanie zarejestrowanego śladu GPS jako nowa aktywność.

3. Transfer danych:

- 3.1. Import danych w standardzie GPX.
- 3.2. Eksport danych w standardzie GPX.

4. Aktywności:

- 4.1. Dodawanie nowych aktywności: import GPX lub rejestrator GPS.
- 4.2. Usuwanie aktywności.
- 4.3. Edycja aktywności:
 - nazwa;

- rodzaj sportu;
- rodzaj kategorii;
- kształt śladu aktywności — pętla lub od punktu do punktu, punkt początkowy, punkt końcowy;
- ocena;
- tagi.

4.4. Wyświetlanie danych aktywności:

- nazwa;
- rodzaj sportu;
- rodzaj kategorii;
- kształt śladu aktywności — pętla lub od punktu do punktu, punkt początkowy, punkt końcowy;
- ocena;
- tagi.

4.5. Wyświetlanie statystyk aktywności:

- data utworzenia;
- data ostatnie modyfikacji;
- data początku aktywności;
- data końca aktywności;
- czas spędzony w ruchu;
- całkowity czas trwania;
- dystans całkowity;
- prędkość maksymalna;
- prędkość średnia w ruchu;
- prędkość średnia całkowita;
- przewyższenia pokonane w górę;
- przewyższenia pokonane w dół;
- najniższe zarejestrowane położenie nad poziomem morza;
- najwyższe zarejestrowane położenie nad poziomem morza.

4.6. Wyświetlanie wykresów aktywności:

- zależność prędkości od czasu;
- zależność pokonanego dystansu od czasu;
- zależność bezwzględnej wysokości nad poziomem morza od czasu.

5. Profil użytkownika:

5.1. Wyświetlanie danych profilu:

- nazwa użytkownika;
- awatar;
- opis profilu;
- płeć;
- data urodzenia;
- waga;
- wzrost;
- kraj pochodzenia;
- miasto zamieszkania.

5.2. Edycja danych profilu:

- nazwa użytkownika;
- awatar;

- opis profilu;
- płeć;
- data urodzenia;
- waga;
- wzrost;
- kraj pochodzenia;
- miasto zamieszkania.

5.3. Wyświetlanie statystyk profilu:

- całkowita liczba aktywności;
- maksymalna zarejestrowana prędkość;
- data pierwszej aktywności;
- data ostatniej aktywności;
- całkowity czas aktywności (czas całkowity);
- średni czas trwania aktywności (czas całkowity);
- najkrótsza aktywność (czas całkowity);
- najdłuższa aktywność (czas całkowity);
- średnia prędkość (czas całkowity);
- całkowity czas aktywności (czas w ruchu);
- średni czas trwania aktywności (czas w ruchu);
- najkrótsza aktywność (czas w ruchu);
- najdłuższa aktywność (czas w ruchu);
- średnia prędkość (czas w ruchu);
- całkowity dystans;
- średni dystans;
- najkrótszy dystans;
- najdłuższy dystans;
- suma przewyższeń w góre;
- suma przewyższeń w dół;
- najniższe zarejestrowane położenie nad poziomem morza;
- najwyższe zarejestrowane położenie nad poziomem morza.

5.4. Wyświetlanie wykresów profilu:

- dzienny dystans na przestrzeni ostatnich 30 dni;
- dzienny czasu aktywności na przestrzeni ostatnich 30 dni;
- dzienne przewyższenia na przestrzeni ostatnich 30 dni;
- najczęściej wybierane rodzaje sportu;
- najczęściej wybierane kategorie;
- najczęściej wybierane oceny;
- najczęściej używane tagów;
- najczęściej wybierane kształty aktywności.

2.4. Wymagania niefunkcjonalne

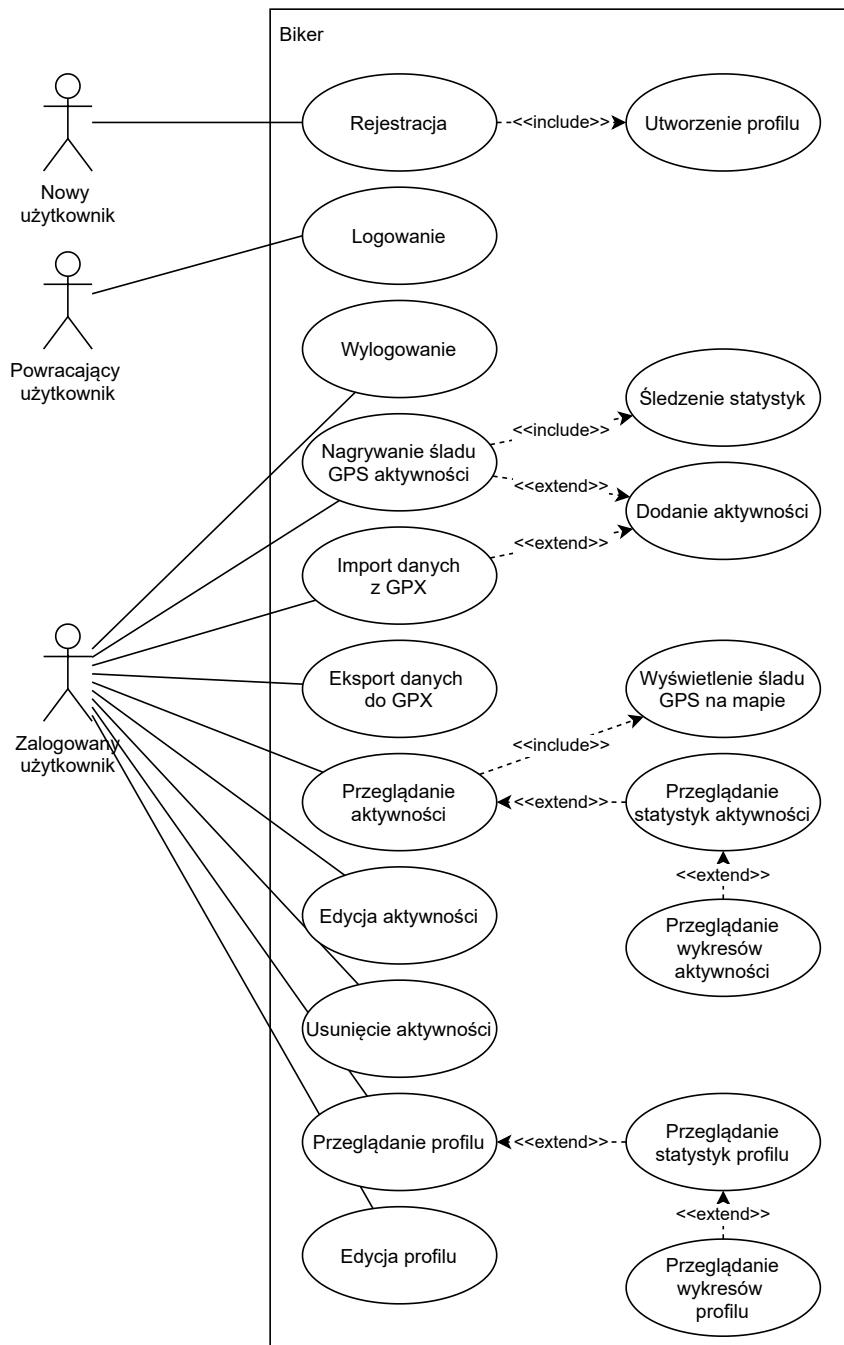
W poniższej liście zawarto wymagania niefunkcjonalne aplikacji:

1. Dostęp do aplikacji z przeglądarki komputera osobistego.
2. Dostęp do aplikacji z przeglądarki urządzenia mobilnego.
3. Synchronizacja danych niezależna od urządzenia.
4. Możliwość instalacji aplikacji na urządzeniu mobilnym.
5. Responsywność aplikacji.

6. Przejrzystość działania aplikacji.
7. Przyjazność interfejsu użytkownika.
8. Interfejs użytkownika w języku angielskim.
9. Szybkość działania.
10. Wydajność działania.
11. Bezpieczeństwo danych użytkownika.

2.5. Diagram przypadków użycia

Diagram przypadków użycia pokazano na rysunku 2.1



Rys. 2.1: Diagram przypadków użycia

2.6. Architektura systemu

Pierwotna koncepcja zakładała zaprojektowanie i implementację dwóch osobnych aplikacji pod wspólną nazwą. Pierwszą z nich miała być aplikacja internetowa (dostęp poprzez przeglądarkę). W niej zawarte miały być wszystkie planowane funkcjonalności za wyjątkiem jedynie nagrywania śladów z wykorzystaniem modułu GPS. Za to zadanie miała być odpowiedzialna dedykowana aplikacja mobilna dla systemu Android. Ta wizja jednak uległa zmianie — finalna wersja zakłada wykonanie aplikacji internetowej w technice PWA. Pozwala ona na instalację aplikacji internetowej na urządzeniu mobilnym i sprawienie wrażenia aplikacji natywnej. Takie rozwiązanie posiada zarówno swoje zalety, jak i wady. Do korzyści na pewno należą:

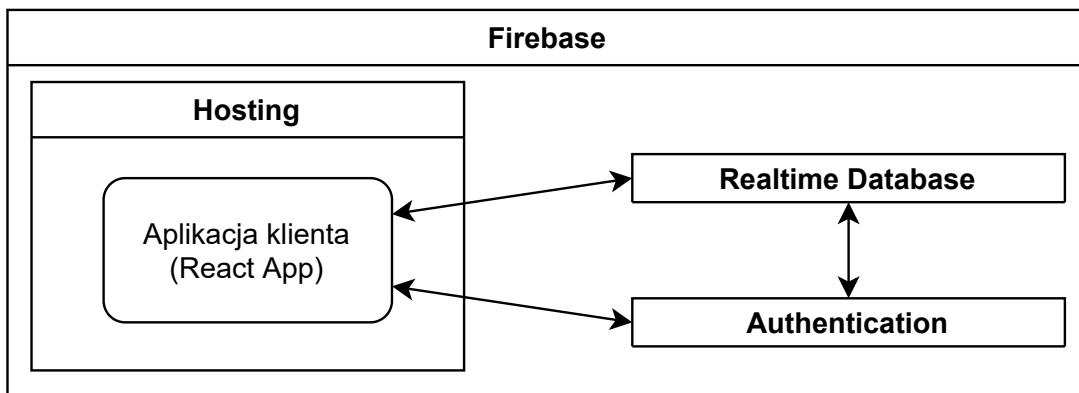
1. Jedna aplikacja oznacza jedną bazę kodu, którą łatwiej jest utrzymać i rozwijać w stosunku do dwóch osobnych baz.
2. Funkcjonalności aplikacji mobilnej i internetowej są identyczne.
3. Aplikacja mobilna nie jest ograniczona do działania jedynie na systemie Android.
4. Brak konieczności prowadzenia pakietu instalacyjnego aplikacji mobilnej.

Głównym problemem natomiast może okazać się wydajność takiego rozwiązania. Działanie PWA oparte jest na przeglądarce internetowej. Z kolei aplikacje natywne działają w systemie jako niezależne instancje. W ogólnym ujęciu aplikacje natywne pozwalają na uzyskanie lepszej wydajności, ponieważ mają bezpośredni dostęp do zasobów sprzętowych.

Porównanie koncepcji oraz dobór technologii realizacji poszczególnych modułów systemu zestawiono w tabeli 2.1. Na rysunku 2.2 przedstawiono schemat ostatecznej wersji architektury systemu. Szczegóły opisano w kolejnych sekcjach.

Tab. 2.1: Porównanie koncepcji architektury systemu i dobór technologii

Zadanie	Koncept pierwotny	Koncept finalny
Aplikacja internetowa	React	React (PWA)
Aplikacja mobilna	React Native	React (PWA)
Backend	Firebase	Firebase
Baza danych	Firebase Firestore lub Realtime Database	Realtime Database
Hosting	Firebase Hosting	Firebase Hosting



Rys. 2.2: Schemat architektury systemu

2.7. Warstwa backend

W tradycyjnym podejściu tworzenia aplikacji internetowej lub mobilnej, poza wykonaniem oprogramowania klienckiego, zazwyczaj konieczne jest utworzenie serwera, bazy danych, magazynu plików, systemu uwierzytelniania oraz połączenie tych (i wielu innych) elementów w działającą całość.

Alternatywą jest wykorzystanie modelu BaaS (ang. *Backend-as-a-Service*), który przenosi odpowiedzialność za funkcje backendowe na dostawcę BaaS. Poszczególne moduły (np.: baza danych czy system uwierzytelniania) dołączane są do warstwy klienta z użyciem pakietów SDK (ang. *Software Development Kit*) oraz API (ang. *Application Programming Interface*). Obrazowo można przedstawić to tak: programista buduje system z gotowych, dostarczonych zewnętrz, konfigurowalnych klocków, zamiast tworzyć swoje własne klocki od podstaw. Dzięki temu rozwój oprogramowania jest szybszy, tańszy, oraz mniej podatny na błędy, a samo oprogramowanie jest łatwo skalowalne. Jednak w przypadku zapotrzebowania na unikalne funkcjonalności warstwy backend, BaaS może się nie sprawdzić ze względu na ograniczone możliwości konfiguracyjne.

BaaS jest technologią stosunkowo nową, ponieważ pierwsze projekty w tej dziedzinie sięgają zaledwie początku drugiej dekady XX wieku. Obecnie na rynku dostępnych jest wiele platform. Do najpopularniejszych można zaliczyć (kolejność losowa): Google Firebase, Apple CloudKit, Amazon AWS Amplify, Microsoft Azure Mobile Apps, Back4App, Parse czy Backendless. Część z nich jest oprogramowaniem otwartym. Platformy komercyjne udostępniają swoje usługi w chmurze, w ramach darmowych oraz płatnych planów (np.: Firebase, AWS Amplify). Niektóre platformy wymagają własnego hostowania (np.: Parse).

Cała warstwa backend realizowanego projektu oparta jest na platformie Google Firebase. W jej zakres wchodzą moduły, które przedstawiono w tabeli 2.2. Szczegóły można znaleźć w dokumentacji technicznej [5]. Główne powody wyboru akurat tej platformy są następujące:

- Podstawowe doświadczenie autora w integracji z platformą, wynikające z wykorzystania jej we wcześniej realizowanych projektach.
- Dostępność platformy w chmurze (brak konieczności samodzielnego hostowania backendu).
- Dostępność darmowego planu, pokrywającego zapotrzebowanie w ramach rozwoju i testowania tworzonego oprogramowania.
- Pokrycie wszystkich wymagań projektu przez udostępniane przez platformę usługi.

Tab. 2.2: Moduły platformy Firebase

Moduł	Opis
Authentication	System uwierzytelniania (email i hasło, <i>identity provider</i> , numer telefonu, <i>custom</i> , sesja anonimowa). FirebaseUI — gotowe rozwiązanie uwierzytelniania w warstwie klienta
Realtime Database	Baza danych NoSQL (baza JSON)
Firestore Database	Baza danych NoSQL (baza dokumentowa)
Cloud Storage	Przestrzeń do przechowywania plików
Machine Learning	Dostęp do wyuczonych modeli oraz możliwość szkolenia własnych modeli w chmurze lub na urządzeniu użytkownika
Hosting	Hosting aplikacji internetowych, statycznej i dynamicznej zawartości oraz mikroserwisów w sieci
Cloud Functions	Framework umożliwiający wykonanie zadanego kodu po stronie backendu w odpowiedzi na zdarzenie lub zapytanie HTTPS.

W ramach projektu wykorzystane zostały moduły: **Authentication**, **Realtime Database** i **Hosting**. W kolejnych sekcjach opisano związek z nimi założenia.

2.7.1. Uwierzytelnianie

Implementacja własnego systemu uwierzytelniania jest procesem złożonym, pracochłonnym i wymagającym odpowiedniej wiedzy. Dlatego Firebase dostarcza moduł Authentication odpowiedzialny za obsługę uwierzytelniania. Udostępnia on cztery metody, którymi użytkownik może potwierdzić swoją tożsamość:

1. Z użyciem emaila i hasła.
2. Z użyciem numeru telefonu.
3. Poprzez zewnętrznego dostawcę: Google, Facebook, Play Games, Game Center, Apple, GitHub, Microsoft, Twitter, Yahoo.
4. Sesja anonimowa.

Bazowa wersja projektu zakłada wykorzystanie dwóch z powyższych metod: email i hasło oraz jeden zewnętrzny dostawca — Google. Natomiast wraz z rozwojem aplikacji przewidziane jest rozszerzenie listy zewnętrznych dostawców oraz dodanie uwierzytelniania z wykorzystaniem numeru telefonu. Nie należy to jednak do zadań priorytetowych projektu.

Firebase dostarcza również bibliotekę **FirebaseUI**, która posiada zaimplementowany interfejs użytkownika oraz większość logiki programowej wymaganej do uwierzytelnienia po stronie klienta. Interfejs ten osadza się w aplikacji i konfiguruje według własnych preferencji. W projekcie wykorzystano tę bibliotekę.

2.7.2. Baza danych

Firebase oferuje dwa rodzaje baz danych hostowanych w chmurze: **Realtime Database** i **Firestore Database**. Są to bazy typu *real-time* (nie mylić z nazwą samej bazy danych), co oznacza że są one w stanie wydajnie przetwarzać dane w czasie rzeczywistym. Z każdą zmianą danych w bazie, wszystkie aktywne, podłączone urządzenia otrzymują natychmiastową aktualizację. Ponadto systemy te zaprojektowane są w taki sposób, że gdy urządzenie utraci połączenie z Internetem, aplikacja może pracować dalej. Jest to możliwe dzięki zachowaniu lokalnej kopii ostatnio używanych danych (*data persistence*). Wszystkie wykonane operacje offline są synchronizowane z bazą po odzyskaniu połączenia, a powstałe konflikty są rozstrzygane automatycznie. Obydwie bazy są bazami nierelacyjnymi (NoSQL) — **Realtime** oparty jest na obiekcie JSON, natomiast **Firestore** to baza dokumentowa. **Firestore** jest rozwiązaniem nowszym, bazującym poniekąd na **Realtime**, jednak nie oznacza to, że **Realtime** jest bezużyteczny. Oba rozwiązania znajdują swoje zastosowania w różnych przypadkach. Na potrzeby projektu, jako bazę danych wybrano **Realtime Database**, ponieważ jest prostsza w obsłudze względem **Firestore**'a, natomiast jej możliwości wciąż pokrywają wymagania.

Schemat struktury obiektu JSON stanowiącego bazę danych przedstawia pseudokod zawarty na listingu 2.3. Na schemacie podano typy poszczególnych pól. Znak ? przy nazwie pola sugeruje, że pole to może być w bazie niezainicjalizowane. Wpis <unique(userId)> należy rozumieć, że w obrębie obiektu-rodzica może wystąpić wiele obiektów, których nazwy są unikalne. Typy GENDERS, RATINGS, SPORTS, CATEGORIES pełnią w schemacie rolę tylko pomocniczą, a zapis valueOf(TYPE) oznacza, że pole przyjmuje jedną wartość z danego TYPE. W przypadku activities i tracks obiekty-dzieci przyjmują nazwy odpowiadające identyfikatorowi aktywności. Występuje tu relacja jeden do jednego — jeden track jest przyporządkowany do jednej activity. Zastosowana praktyka jest powszechnie stosowana w obiektowych bazach danych.

Listing 2.3: Struktura zastosowanej bazy danych w postaci pseudokodu

```

// Pomocnicze oznaczenia
GENDERS = 'male' | 'female' | 'other';
RATINGS = 'terrible' | 'poor' | 'fair' | 'good' | 'excellent';
SPORTS = 'touring' | 'mtb' | 'enduro' | 'downhill' | 'gravel' | 'road' | '
    ↵ 'bmx' | 'tandem' | 'cyclocross' | 'track' | 'speedway' | 'other';
CATEGORIES = 'commute' | 'casual' | 'training' | 'race' | 'bikepacking' |
    ↵ 'other';

// Właściwa struktura
db_id: {
  users: {
    <unique(userId)>?: {
      profile?: {
        gender?: valueOf(GENDERS),
        birthday?: string,
        weight?: number,
        height?: number,
        country?: string,
        city?: string,
        description?: string
      },
      activities?: {
        <unique(activityId)>: {
          creatorId: string,
          name: string,
          createdAt: number,
          lastModifiedAt: number,
          startTime: number,
          endTime: number,
          sport: valueOf(SPORTS),
          category: valueOf(CATEGORIES),
          shape: {
            isLoop: boolean,
            from: string,
            to: string,
          },
          rating?: valueOf(RATINGS),
          tags?: string[],
          statistics?: {
            totalDistance?: number,
            totalDuration?: number,
            inMotionDuration?: number,
            maxSpeed?: number,
            elevationUp?: number,
            elevationDown?: number
          }
        }
      }
    },
    tracks?: {
      <unique(activityId)>: {
        activityId: string,
        segments: (
        {
          lat: number,
          lon: number,
          time: number,
          ele?: number
        }[]
      )[]
    }
  }
}

```

```
        }
    }
}
};
```

Dostęp do zasobów bazy **Realtime** kontrolowany jest przez zasady (ang. *database rules*). Każde zapytanie odczytu lub zapisu do bazy przechodzi przez weryfikację, i jest wykonane tylko wtedy, gdy pozwolą na to zasady. Zasady mają postać drzewa JSON. Przykład pokazano na listingu 2.4 — zasada udziela możliwość zapisywania do ścieżki `/users/<uid>/` tylko użytkownikom.

Listing 2.4: Przykład zasady definiującej dostęp do bazy danych

```
{
  "rules": {
    "users": {
      "$uid": {
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

2.7.3. Hosting

W zakresie usług **Firebase**'a jest również możliwość hostowania aplikacji. Usługa ta jest wykorzystana w projekcie. Zbudowaną aplikację można opublikować w bardzo prosty sposób z użyciem tylko jednej komendy. Aplikacje publikowane są w dwóch domyślnych domenach: `nazwa-aplikacji.web.app` oraz `nazwa-aplikacji.firebaseio.app`. Istnieje również możliwość konfiguracji własnej domeny.

Aplikacje hostowane są w standardzie SSL, co pozwala na wykorzystanie protokołu HTTPS. Jest to o tyle istotne, że w przypadku braku protokołu HTTPS przeglądarki internetowe blokują dostęp do pewnych funkcjonalności. W kontekście projektu ma to fundamentalne znaczenie, ponieważ jedną z blokowanych funkcjonalności jest lokalizacja GPS, na którym oparta jest cała koncepcja projektu.

2.8. Warstwa frontend

Aplikację klienta zaimplementowano z wykorzystaniem biblioteki **React** [16] oraz języka **TypeScript**. React to darmowa i otwarta biblioteka języka JavaScript, która przeznaczona jest do budowania interfejsów użytkownika aplikacji internetowych typu SPA (ang *Single-page Application*). SPA, jest to aplikacja jednostronicowa (posiada tylko jeden plik HTML), która dynamicznie aktualizuje swoją zawartość, zamiast dokonywać przeładowania całej strony. Aplikacje SPA zapewniają bardzo dobre doświadczenie użytkownika (ang *user experience*) i wyznaczają standard tworzenia współczesnych aplikacji internetowych. W zestawieniu z Reactem wykorzystano dodatkowe narzędzia. Najważniejsze z nich przedstawiono w tabeli 2.3

Tab. 2.3: Główne biblioteki wykorzystane przy tworzeniu aplikacji klienta

Nazwa	Zastosowanie	Link
TypeScript	Język programowania oparty na JavaScript, dodający do staticzne typowanie	typescriptlang.org
Less.js	Rozszerzenie językowe CSS, stylowanie aplikacji	lesscss.org
React	Budowa interfejsu użytkownika	reactjs.org
Ant Design	Gotowe komponenty reactowe oraz system designu	ant.design
Firebase	Integracja z platformą Firebase	firebase.google.com
FirebaseUI	Gotową implementację uwierzytelniania z platformą Firebase	firebase.google.com
Leaflet	Interaktywne mapy	leafletjs.com
React-Leaflet	Wrapper Leafleta, gotowe komponenty reactowe	react-leaflet.js.org
FileSaver.js	Obsługa zapisywania plików na dysku	github.com/eligrey/FileSaver.js
react-vis	Reactowe komponenty wykresów	uber.github.io/react-vis/
react-icons	Reactowe komponenty ikon	react-icons.github.io/react-icons/
react-resize-detector	API obserwacji zmiany rozmiaru elementów na stronie	github.com/maslianok/react-resize-detector
react-responsive	API obserwacji zmiany rozmiaru okna przeglądarki	github.com/yocontra/react-responsive
GPXParser	Parser standardu GPX na obiekty javascriptowe	github.com/Luuka/GPXParser.js
GPX builder	Parser obiektów javascriptowych na standard GPX	github.com/Luuka/GPXParser.js
nosleep.js	Mechanizm blokady wygaszania ekranu urządzenia	github.com/richtr/NoSleep.js
uuid	Generator unikalnych identyfikatorów	github.com/uuidjs/uuid
he	Enkoder i dekoder <i>html entities</i>	github.com/mathiasbynens/he
useTimeout, useInterval	Hooki reactowe umożliwiające deklaratywne użycie javascriptowych funkcji <code>setTimeout</code> oraz <code>setInterval</code>	gist.github.com/Danziger/336e75b6675223ad805a88c2dfdcfd4a

Rozdział 3

Implementacja projektu

W niniejszym rozdziale przedstawiono proces tworzenia aplikacji. Pokazano fragmenty kodu realizujące kluczowe funkcjonalności, jak np.: uwierzytelnianie, rejestrowanie śladów czy prezentacja statystyk. Zawarto również rysunki, na których ukazane są widoki aplikacji.

Przebieg procesu implementacji projektu można podzielić na etapy, których kolejność przedstawiono poniżej. Jednak w praktyce prace nad pewnymi etapami prowadzono równolegle, jak np.: nagrywanie śladów oraz kalkulator statystyk. Wynika to naturalnie z faktu, że pewne etapy są ze sobą ściślej powiązane, a co za tym idzie — zmiany wprowadzone w jednej części systemu, pociągają za sobą zmiany w drugiej części systemu. Przebieg procesu implementacji:

1. Inicjalizacja aplikacji klienta.
2. Inicjalizacja Firebase.
3. Integracja z Firebase.
4. Integracja z Firebase Hosting.
5. Konfiguracja PWA.
6. Projekt struktury bazy danych.
7. Implementacja układów stron.
8. Implementacja uwierzytelniania.
9. Implementacja routingu.
10. Implementacja rejestratora GPS.
11. Implementacja importu GPX.
12. Implementacja eksportu GPX.
13. Implementacja aktywności.
14. Implementacja statystyk aktywności.
15. Implementacja wykresów aktywności.
16. Integracja z Realtime Database.
17. Implementacja profilu użytkownika.
18. Implementacja statystyk profilu użytkownika.
19. Implementacja wykresów profilu użytkownika.

Pracę nad projektem prowadzono z użyciem edytora Visual Studio Code. Zmiany w kodzie regularnie zapisywano do zdalnego repozytorium GitHub. Repozytorium jest publiczne, a dostęp do niego można uzyskać pod adresem: <https://github.com/michaltkacz/biker>.

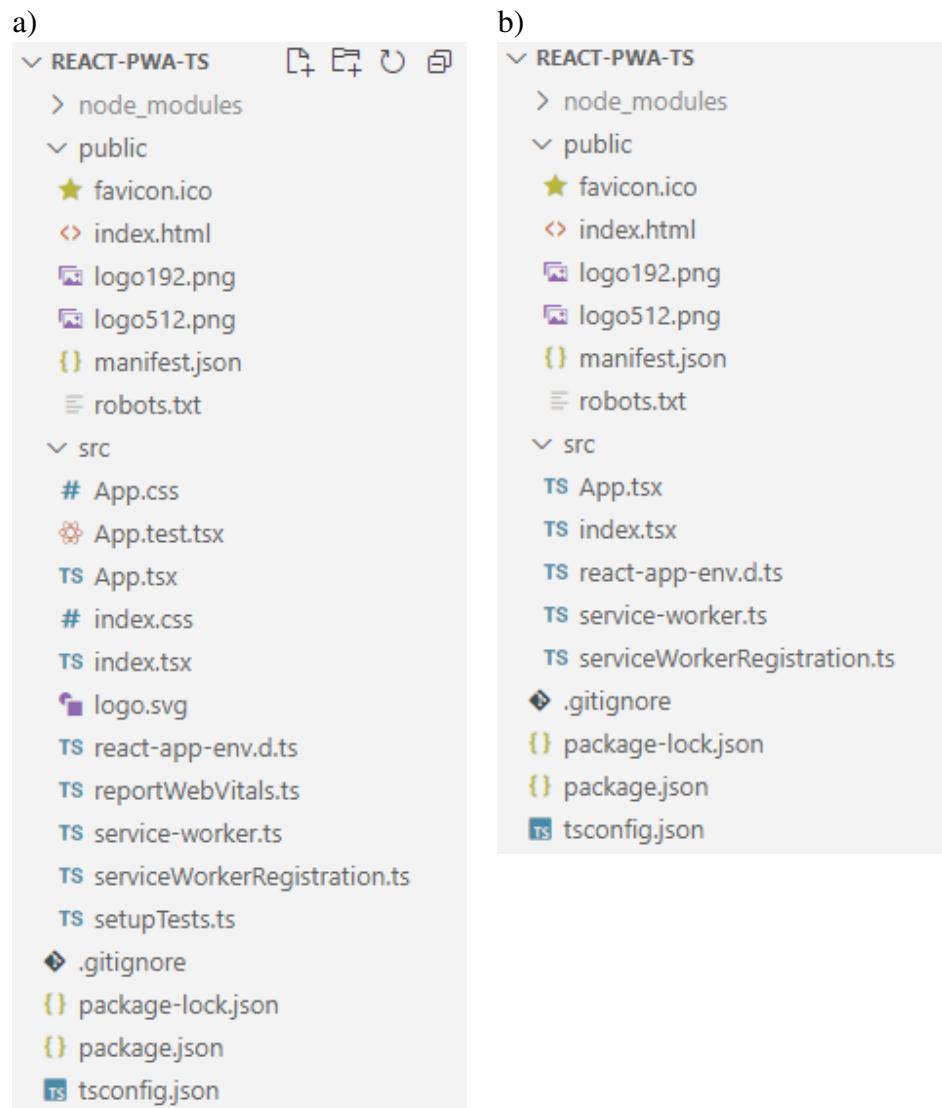
3.1. Inicjalizacja aplikacji klienta

Projekt aplikacji klienta utworzono przez uruchomienie komendy w środowisku Node.js, przedstawionej na listingu 3.1. Dzięki wykorzystaniu flagi `-template cra-template-pwa-typescript` projekt zainicjalizowano w konfiguracji z językiem

TypeScript oraz z zaimplementowanym *service-workerem* odpowiedzialnym za działanie aplikacji w trybie PWA. Następnie projekt oczyszczono ze zbędnych plików oraz zainstalowano kluczowe, dodatkowe biblioteki (ponownie listing 3.1) konieczne do rozpoczęcia pracy nad kodem (pozostałe biblioteki w miarę zapotrzebowania doinstalowywano w późniejszych etapach rozwoju aplikacji). Finalną strukturę plików uzyskaną po ukończeniu projektu przedstawiono w dodatku B.

Listing 3.1: Inicjalizacja projektu React oraz instalacja kluczowych bibliotek

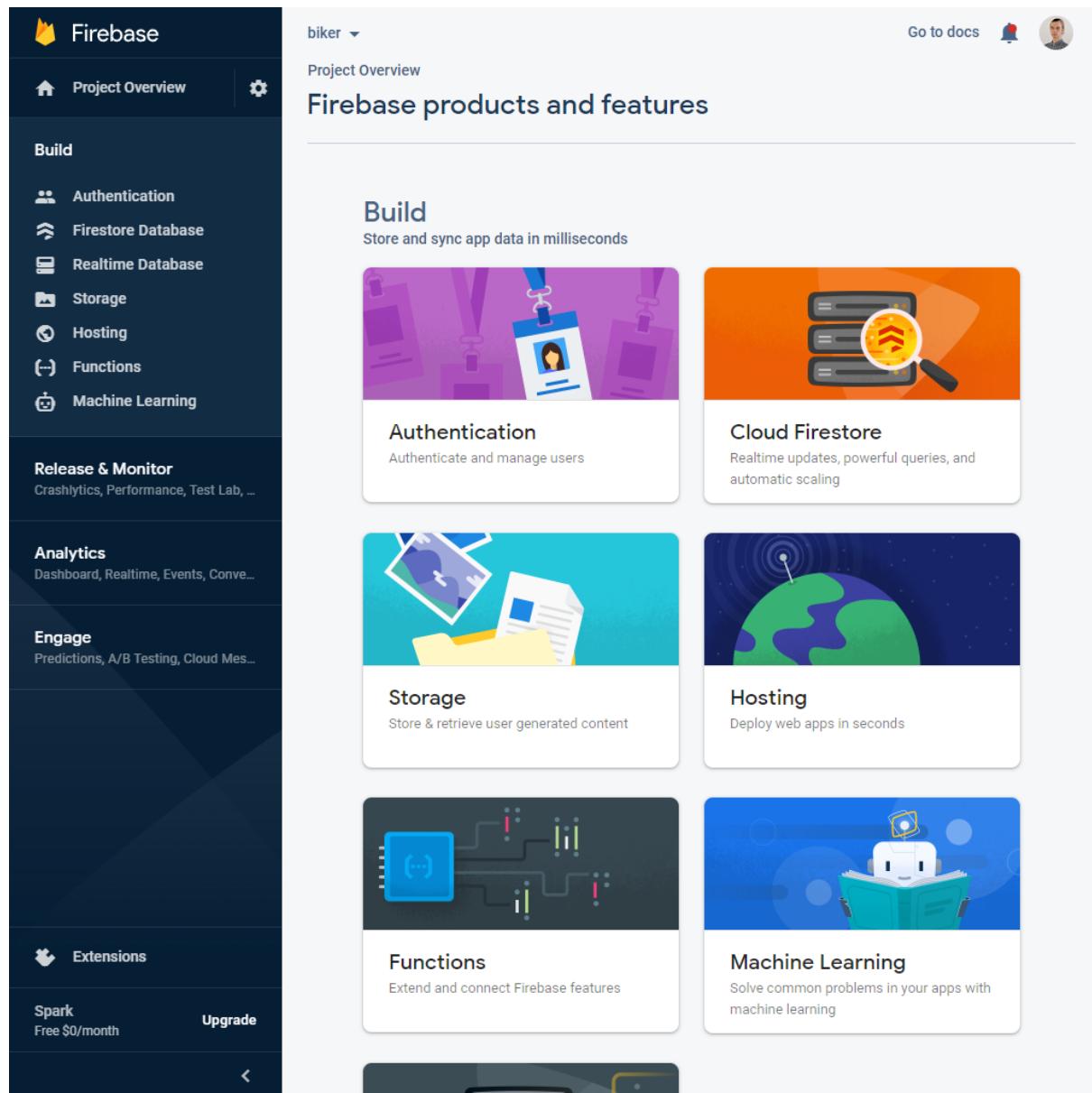
```
npx create-react-app react-pwa-ts --template cra-template-pwa-typescript
cd react-pwa-ts
npm i react-router
npm i antd
npm i firebase
npm i firebaseui
npm i firebaseui
npm i craco
npm i craco-less
npm i leaflet
npm i react-leaflet
npm i react-icons
```



Rys. 3.1: Drzewo plików projektu: a) po inicjalizacji, b) po usunięciu zbędnych plików

3.2. Inicjalizacja Firebase

W celu inicjalizacji projektu Firebase należy odwiedzić stronę: <https://firebase.google.com/>, a następnie postępować zgodnie z instrukcjami wyświetlanymi na stronie, lub zgodnie z oficjalną dokumentacją [4]. Po pomyślnej inicjalizacji, deweloper zyskuje dostęp do konsoli (Firebase Console) (rys. 3.2), która pozwala na zarządzanie projektem. W menu po lewej stronie, w zakładce *Build* dostępne są poszczególne moduły. Trzy z nich: Authentication, Realtime Database i Hosting, omówiono w dalszej części rozdziału.

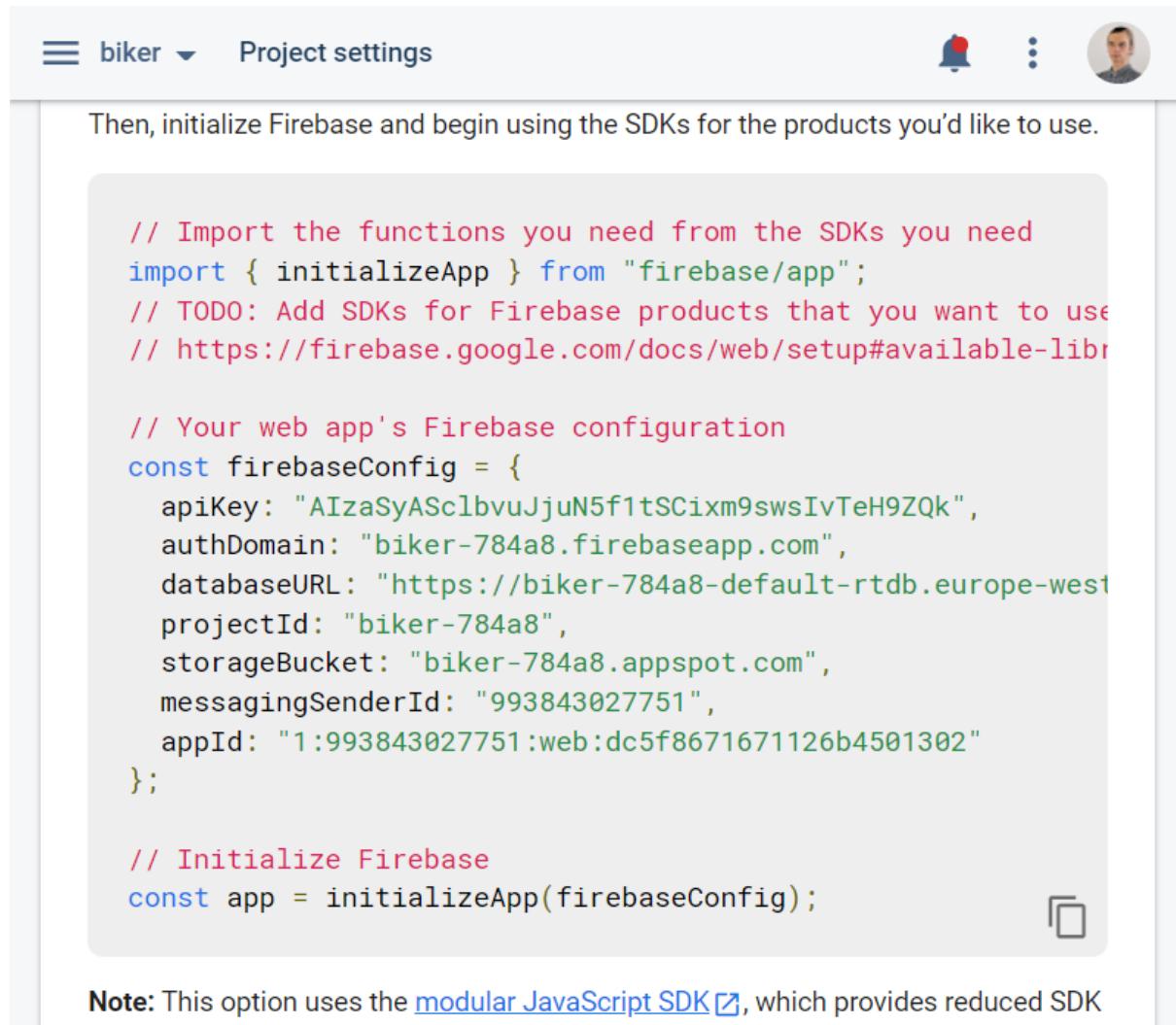


Rys. 3.2: Widok głównej strony Firebase Console

3.3. Integracja z Firebase

W folderze z plikami źródłowymi projektu aplikacji klienta utworzono nowy plik o nazwie `firebase.tsx`. Następnie zawarto w nim kod konfiguracji Firebase, który skopiowano z kon-

soli (rys. 3.3). Na podstawie tej konfiguracji, w pliku `src/firebase/firebase.tsx` zainicjalizowano moduł aplikacji Firebase. W późniejszych etapach zainicjalizowano również obiekty autoryzacji auth oraz bazy danych database. Pozwalają one z poziomu aplikacji klienta odwoływać się do odpowiednich funkcjonalności po stronie Firebase'a. Finalną wersję pliku przedstawiono na listingu 3.2.



The screenshot shows the 'Project settings' page in the Firebase console. At the top, there's a header with the project name 'biker' and a 'Project settings' button. To the right are icons for notifications, three dots, and a user profile picture. Below the header, a message says 'Then, initialize Firebase and begin using the SDKs for the products you'd like to use.' A large code block displays the configuration object:

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyASclbvujjuN5f1tSCixm9swsIvTeH9ZQk",
  authDomain: "biker-784a8.firebaseioapp.com",
  databaseURL: "https://biker-784a8-default-rtdb.europe-west1.firebaseio.go
  projectId: "biker-784a8",
  storageBucket: "biker-784a8.appspot.com",
  messagingSenderId: "993843027751",
  appId: "1:993843027751:web:dc5f8671671126b4501302"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

Note: This option uses the [modular JavaScript SDK](#), which provides reduced SDK

Rys. 3.3: Obiekt konfiguracyjny w Firebase Console

Listing 3.2: Integracja aplikacji klienta z projektem Firebase

```
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getDatabase } from 'firebase/database';

const firebaseConfig = {
  apiKey: 'AIzaSyASclbvujjuN5f1tSCixm9swsIvTeH9ZQk',
  authDomain: 'biker-784a8.firebaseioapp.com',
  projectId: 'biker-784a8',
  databaseURL:
    'https://biker-784a8-default-rtdb.europe-west1.firebaseio.go
  storageBucket: 'biker-784a8.appspot.com',
  messagingSenderId: '993843027751',
  appId: '1:993843027751:web:dc5f8671671126b4501302',
};
```

```

const firebaseApp = initializeApp(firebaseConfig);

export const auth = getAuth(firebaseApp);
export const database = getDatabase(firebaseApp);

export default firebaseApp;

```

Zawartość obiektu konfiguracyjnego jest unikalna dla danego projektu, jednak nie jest ona sekretna. W celu zabezpieczenia danych przechowywanych w bazie, należy wykorzystać **Firebase Security Rules**. Reguły bezpieczeństwa zastosowane w projekcie omówiono w dalszej części rozdziału.Więcej informacji na temat obiektu konfiguracyjnego można znaleźć w dokumentacji [7]

3.4. Integracja z Firebase Hosting

Integrację z usługą hostingu przeprowadzono według dokumentacji [6]. W pierwszym kroku, w katalogu głównym projektu, z poziomu wiersza poleceń wykonano komendę **firebase init hosting**. Uruchamia ona intuicyjny konfigurator, który w kilku krokach pozwala skonfigurować projekt do usługi hostowania. Po zakończeniu konfiguracji w katalogu głównym projektu powstały dwa pliki: **.firebaserc** oraz **firebase.json**, których zawartość pokazano odpowiednio na listingu 3.3 i listingu 3.4. Pierwszy z nich to konfiguracja aliasów projektu. Jest to użyteczne w przypadku, gdy w jednym projekcie używanych jest wiele projektów Firebase. Drugi plik, to konfiguracja usługi hostingu. Dzięki niemu można ustawić takie opcje hostingu, jak np.: przekierowania URL, nadpisania URL, czy nawet wyświetlanie ukośnika na końcu adresu URL (*trailing-slash*).

Listing 3.3: Plik **.firebaserc**

```
{
  "projects": {
    "default": "biker-784a8"
  }
}
```

Listing 3.4: Plik **firebase.json**

```
{
  "hosting": {
    "site": "biker-784a8",
    "public": "build",
    "ignore": ["firebase.json", "**/.*", "**/node_modules/**"],
    "rewrites": [
      {
        "source": "**",
        "destination": "/index.html"
      }
    ]
  }
}
```

W kolejnym kroku zbudowano aplikację klienta z użyciem polecenia **npm run build**, a następnie wywołano polecenie **firebase hosting deploy**. W wyniku tej procedury, zbudowaną aplikację opublikowano pod adresami: **biker-784a8.web.app** oraz **biker-784a8.firebaseioapp.com**. Po wprowadzeniu zmian w projekcie, w celu aktualizacji hostowanej aplikacji, proces budowania i publikacji należy powtórzyć.

3.5. Konfiguracja PWA

W celu aktywacji opcji instalacji aplikacji jako PWA, oprócz rejestracji *service-workera*, konieczna jest konfiguracja ustawień przy pomocy pliku `public/manifest.json`. Dokumentacja dostępna jest pod adresem [2]. Finalną wersję manifestu przedstawiono na listingu 3.5, a jego wyjaśnienie zawarto w tabeli 3.1.

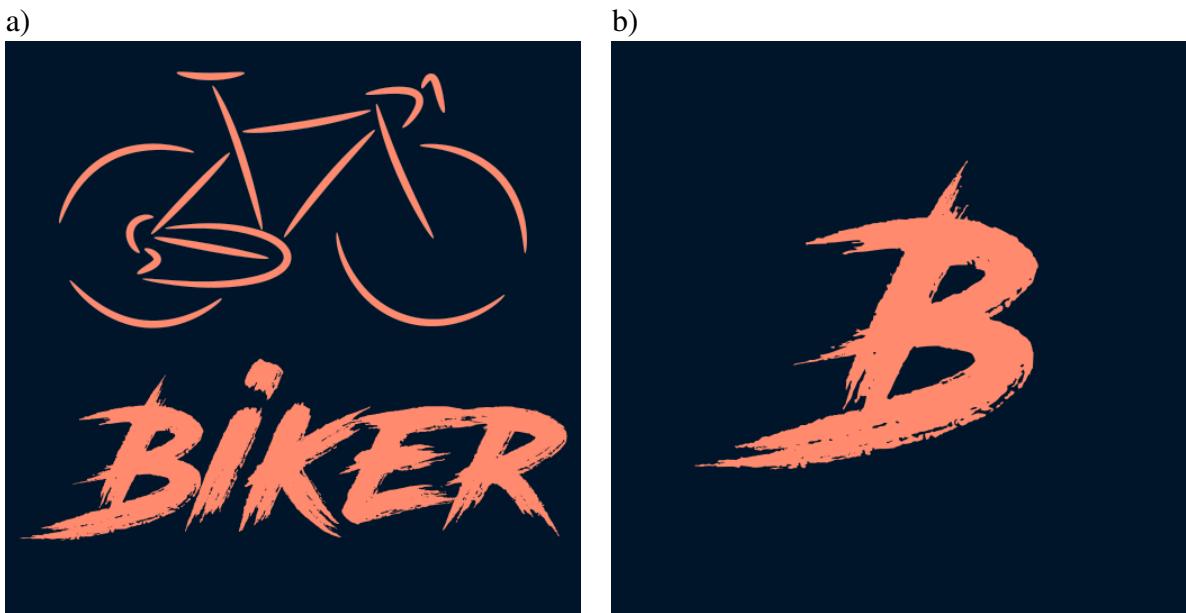
Listing 3.5: Plik `manifest.json`

```
{
  "background_color": "white",
  "categories": ["sport"],
  "description": "Cycling Navigation. Monitor all your cycling statistics  
→ in one place.",
  "dir": "ltr",
  "display": "standalone",
  "icons": [
    {
      "src": "favicon.ico",
      "sizes": "64x64",
      "type": "image/x-icon"
    },
    {
      "src": "logo.svg",
      "sizes": "any",
      "type": "image/svg+xml"
    },
    {
      "src": "logo_maskable.svg",
      "sizes": "any",
      "type": "image/svg+xml",
      "purpose": "maskable"
    }
  ],
  "lang": "en-US",
  "name": "Biker",
  "orientation": "portrait",
  "short_name": "Biker",
  "start_url": "/",
  "theme_color": "white"
}
```

Tab. 3.1: Opis pliku `manifest.json`

Atrybut	Opis
<code>background_color, theme_color</code>	Kolor tła oraz kolor motywu aplikacji
<code>categories</code>	Kategoria, do której należy aplikacja
<code>description</code>	Opis aplikacji
<code>dir, lang</code>	Kierunek tekstu oraz język wyświetlania aplikacji
<code>display, orientation</code>	Tryb wyświetlania aplikacji oraz jej orientacja
<code>icons</code>	Zestaw ikon aplikacji
<code>name, short_name</code>	Nazwa oraz skrócona nazwa aplikacji
<code>start_url</code>	Adres URL punktu wejścia do aplikacji

Ikony aplikacji (rys. 3.4) — `logo.svg`, `logo_maskable.svg`, `favicon.ico` — zostały wykonane w programie do tworzenia grafiki wektorowej Inkscape. Ikony `favicon.ico` oraz `logo_maskable.svg` różnią się jedynie rozmiarem marginesu symbolu.



Rys. 3.4: Ikony aplikacji: a) logo.svg, b) logo_maskable.svg

3.6. Projekt struktury bazy danych

Korzystając z możliwości języka TypeScript, w projekcie utworzono typy danych definiujące po stronie klienta strukturę bazy danych. Typy te są reprezentacją drzewa obiektów przedstawionego na rysunku 2.3. Wykorzystanie sztywno zdefiniowanych typów gwarantuje spójność danych na przestrzeni całej aplikacji oraz ułatwia integrację z bazą danych po stronie Firebase'a. Dodatkową zaletą jest łatwiejsza kontrola błędów oraz uniknięcie nieprzewidzianego zachowania aplikacji. Zawartość lisingów 3.6, 3.7, 3.8, 3.9 odpowiada plikowi `src/database/schema.tsx`.

Na lisingu 3.6 przedstawiono strukturę typów w ogólnym ujęciu. `DatabaseSchema` przekłada się na instancję bazy danych po stronie Realtime Database. Zawartością obiektu tego typu są obiekty użytkowników kluczowane unikalnym identyfikatorem użytkownika (identyfikator jest generowany automatycznie przez Firebase przy rejestracji nowego konta). Każdy obiekt użytkownika zawiera swój identyfikator, obiekt profilu, obiekt aktywności i obiekt śladów. W obiekcie profilu z kolei przechowywane są dane personalne użytkownika: płeć, data urodzenia, waga, wzrost, kraj i miasto zamieszkania oraz opis.

Listing 3.6: Typy obiektowe struktury bazy danych (1/3) — root bazy danych oraz użytkownik

```
type DatabaseSchema = {
  users?: { [userId: string]: User };
};

export type User = {
  userId: string;
  profile?: UserProfile;
  activities?: { [activityId: string]: Activity };
  tracks?: { [trackId: string]: Track };
};

export type UserProfile = {
  gender?: GenderTypes;
  birthday?: string;
  weight?: number;
  height?: number;
  country?: string;
  city?: string;
```

```
    description?: string;
};
```

Na listingu 3.7 pokazano typ aktywności, która zawiera w sobie szczegółowe dane o zapisanej podróży rowerowej, w tym podstawowe statystki śladu. Wśród statystyk nie ma zapisanych żadnych wartości uśrednionych (np.: średnia prędkość, średnia elewacja), ponieważ można je łatwo wyliczyć z pozostałych, dostępnych statystyk.

Listing 3.7: Typy obiektowe struktury bazy danych (2/3) — aktywności

```
export type Activity = {
  activityId: string;
  creatorId: string;
  name: string;
  createdAt: number;
  lastModifiedAt: number;
  startTime: number;
  endTime: number;
  sport: ActivitySportTypes;
  category: ActivityCategoryTypes;
  shape: ActivityShape;
  statistics: ActivityStatistics;
  rating?: RatingTypes;
  tags?: Array<string>;
};

export type ActivityShape = {
  isLoop: boolean;
  from: string;
  to: string;
};

export type ActivityStatistics = {
  totalDistance?: number;
  totalDuration?: number;
  inMotionDuration?: number;
  maxSpeed?: number;
  elevationUp?: number;
  elevationDown?: number;
  minElevation?: number;
  maxElevation?: number;
};
```

Typy Track, TrackSegment i TrackPoint zdefiniowano na wzór standardu GPX — odpowiednio trkType, trksegType i ptType. Zatem obiekt śladu składa się z listy segmentów oraz dodatkowo z identyfikatora aktywności, do której należy. Z kolei jeden segment, to lista punktów, które opisują szerokość i długość geograficzną, czas zarejestrowania położenia oraz ewentualną bezwzględną wysokość nad poziomem morza. Definicje typów zawarto w listingu 3.8.

Listing 3.8: Typy obiektowe struktury bazy danych (3/3) — śladы

```
export type Track = {
  activityId: string;
  segments: Array<TrackSegment>;
};

export type TrackSegment = Array<TrackPoint>;
export type TrackPoint = {
  lat: number;
  lon: number;
  time: number;
  ele?: number | null;
};
```

Oprócz typów obiektowych zdefiniowano również cztery typy wyliczeniowe, które przedstawiono na listingu 3.9. Określają one dostępne dla użytkownika opcje wyboru dla danych: płeć użytkownika, rodzaj sportu aktywności, kategoria aktywności oraz ocena aktywności.

Listing 3.9: Typy wyliczeniowe struktury bazy danych

```
export enum GenderTypes {
    Male = 'male',
    Female = 'female',
    Other = 'other',
}
export enum ActivitySportTypes {
    Touring = 'touring',
    MTB = 'mtb',
    Enduro = 'enduro',
    Downhill = 'downhill',
    Gravel = 'gravel',
    Road = 'road',
    BMX = 'bmx',
    Tandem = 'tandem',
    Cyclocross = 'cyclocross',
    Track = 'track',
    Speedway = 'speedway',
    Other = 'other',
}
export enum ActivityCategoryTypes {
    Commute = 'commute',
    Casual = 'casual',
    Workout = 'training',
    Race = 'race',
    Bikepacking = 'bikepacking',
    Other = 'other',
}
export enum RatingTypes {
    Terrible = 'terrible',
    Poor = 'poor',
    Fair = 'fair',
    Good = 'good',
    Excellent = 'excellent',
}
```

3.7. Implementacja układów stron

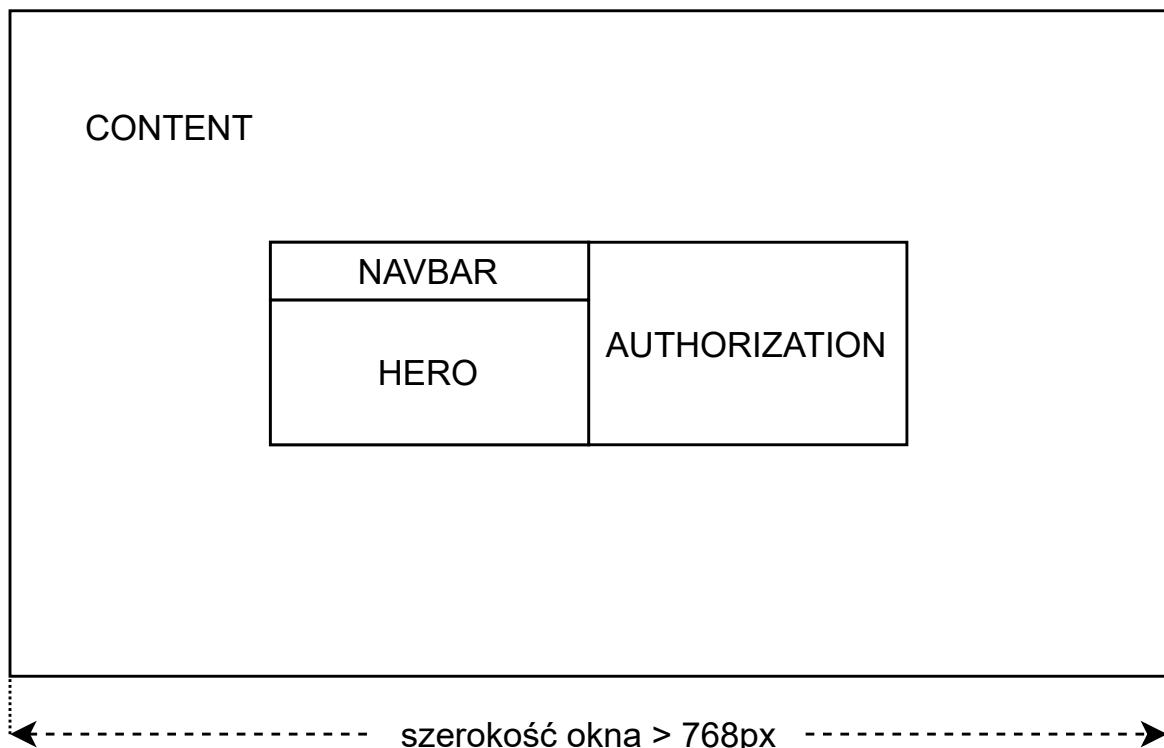
W aplikacji zaimplementowano dwa różne układy strony — jeden wyłącznie dla strony autoryzacji, a drugi – globalny, wspólny dla wszystkich pozostałych stron. Układy są w pełni responsywne, co oznacza, że dynamicznie dostosowują się one do rozmiaru ekranu urządzenia. Responsywność uzyskano dzięki zastosowaniu zasad CSS typu `@media min-width`.

Szablon układu strony autoryzacji przedstawiają: w wersji desktopowej (dla ekranu o szerokości powyżej 768px) — rysunek 3.5, a w wersji mobilnej — rysunek 3.7a (dla ekranu o szerokości nie przekraczającej 768px). Gotowe widoki aplikacji dla powyższych szablonów pokazano odpowiednio na rysunku 3.6 oraz rysunku 3.7b.

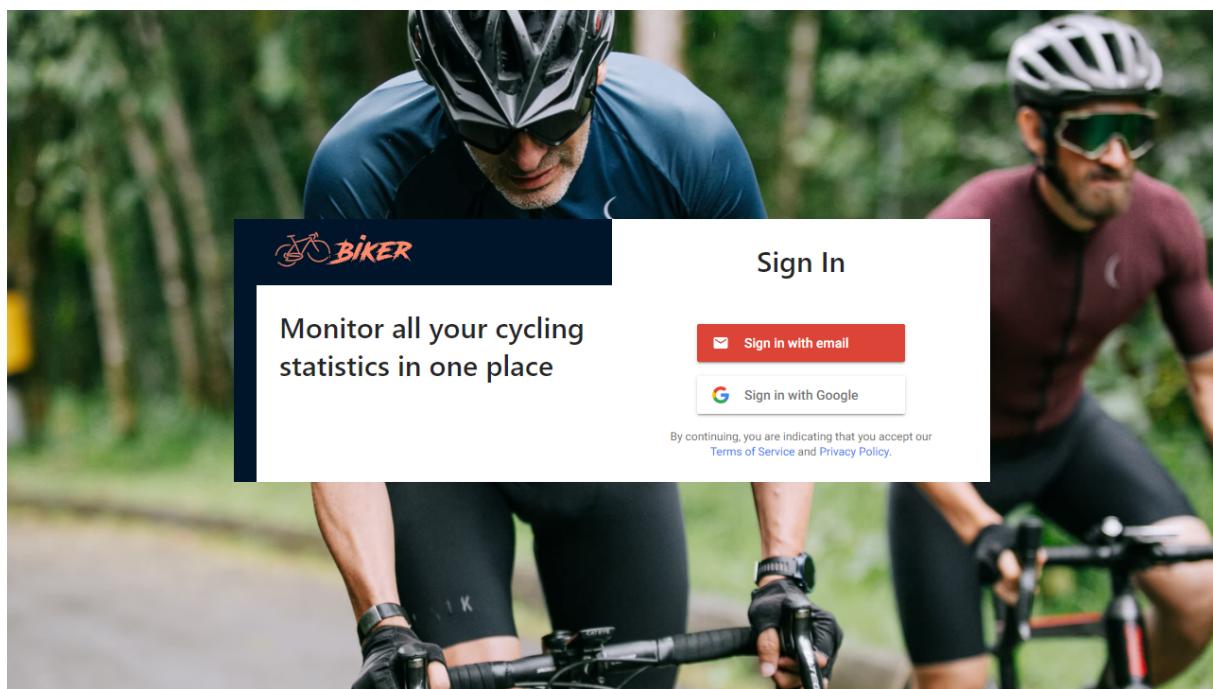
Szablon globalny (rys. 3.8), wykorzystano do zbudowania stron: profilu użytkownika, rejestraitora GPS oraz aktywności (rys. 3.9). Jest on również w pełni responsywny. Dla ekranów o rozmiarze powyżej 1200px *sider* przyjmuje on stałą szerokość 200px i nie może zostać ukryty, natomiast *content* wypełnia całe pozostałe miejsce. W przypadku ekranów nie większych niż 1200px, *sider* staje się dynamiczny — może zostać ukryty przy użyciu przycisku *sider button*. Z kolei *content* przyjmuje stałą szerokość równą całkowitej szerokości okna widoku. Z tego względu otwarcie *sidera* powoduje wypchnięcie *contentu* o 200px w prawo, poza obszar ekranu. Jest to zabieg umyślny, który ma na celu zminimalizowanie efektu *content jumpingu*, czyli niechcianej zmiany położenia elementów strony. Efekt ten był szczególnieauważalny na urządzeniach z wąskim ekranem, jak np.: smartfonach.

3.8. Implementacja uwierzytelniania

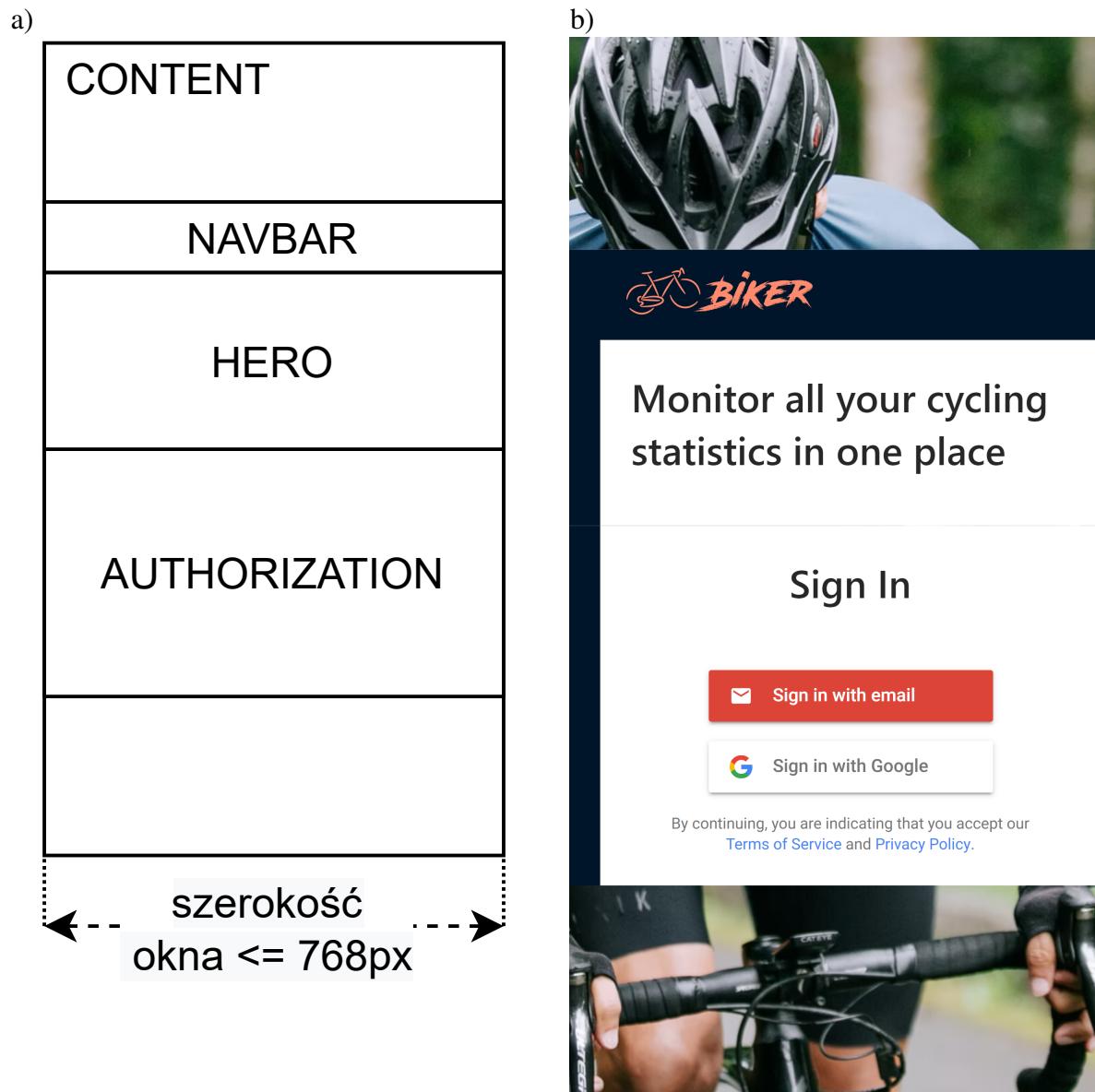
Do implementacji uwierzytelniania wykorzystano moduł `Firebase Authentication` oraz bibliotekę `FirebaseUI`. W pierwszym kroku, w pliku `src/firebase/firebase.tsx` zainicjalizowano moduł autentykacji, jak pokazano na listingu 3.2. Następnie utworzono komponent `Auth` (jak pokazano na listingu 3.10), który przy pierwszym renderowaniu uruchamia usługę uwierzytelniania z biblioteki `FirebaseUI`. Funkcję `startFirebaseUI` zaimplementowano w pliku `src/firebase/firebaseUI.tsx`, jej kod oraz konfigurację biblioteki



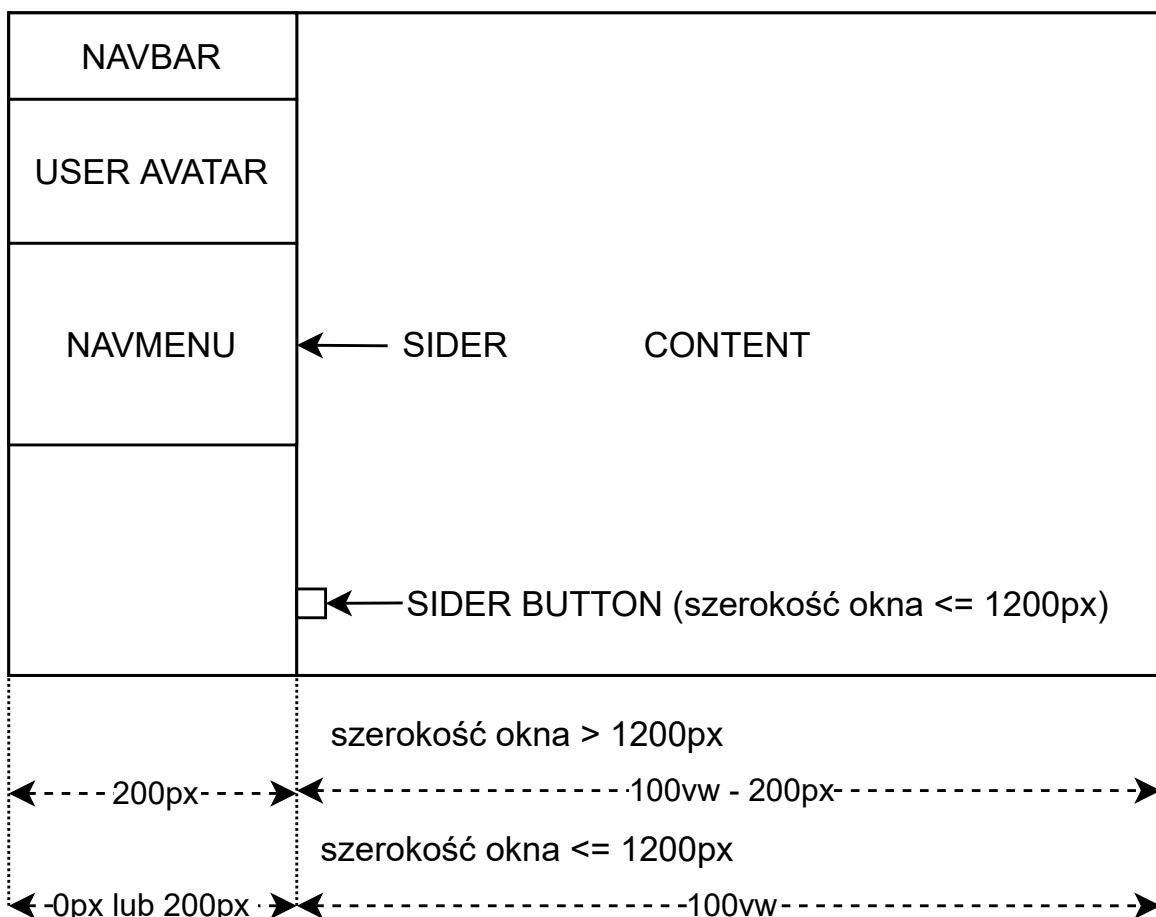
Rys. 3.5: Szablon układu strony autoryzacji w wersji desktopowej



Rys. 3.6: Widok strony autoryzacji w wersji desktopowej



Rys. 3.7: Układ strony autoryzacji a) szablon, b) widok aplikacji



Rys. 3.8: Szablon globalnego układu strony w wersji mobilnej i desktopowej

The screenshot shows the 'Activities' section of the BIKER application. On the left is a dark sidebar with a user profile picture of Paul Foster, a 'Profile' link, a 'Tracker' link, an active 'Activities' link, and a 'Logout' link. The main content area has a title 'Activities' and a sub-section 'Windy ride'. It shows a map of a cycling route in Wroclaw and surrounding areas, with a purple line indicating the path. Below the map are performance metrics:

Start Time	09:31	End Time	12:07
In Motion Time	02:01	Total Time	02:36
Distance	44.3 km	Max Speed	40.8 km/h
Average Speed (Motion)	21.8 km/h	Average Speed (Total)	16.9 km/h

Rys. 3.9: Widok strony aktywności w wersji desktopowej

przedstawiono na listingu 3.11. W wyniku działania funkcji, do kontenera o identyfikatorze `#firebaseui-auth-container` wstrzyknięty zostaje komponent autoryzacji, który zgodnie podaną konfiguracją umożliwia autoryzację dwiema metodami: przez podanie adresu email oraz hasła lub przez konto Google. Komponent widoczny jest na rysunku 3.9.

Listing 3.10: Komponent uwierzytelniania

```
const Auth: React.FC = () => {
  useEffect(() => {
    startFirebaseUI();
  }, []);
  return (
    <div className='auth'>
      <Typography.Title level={2} className='auth-title'>
        Sign In
      </Typography.Title>
      <div id={elementID} className='auth-widget'></div>
    </div>
  );
};
```

Listing 3.11: Autoryzacja z użyciem biblioteki FirebaseUI

```
const firebaseUI = new firebaseui.auth.AuthUI(auth);
export const firebaseUiDefaultConfig: firebaseui.auth.Config = {
  signInFlow: 'popup',
  signInOptions: [
    EmailAuthProvider.PROVIDER_ID,
    GoogleAuthProvider.PROVIDER_ID,
  ],
  tosUrl: 'https://en.wikipedia.org/wiki/Terms_of_service',
  privacyPolicyUrl: 'https://en.wikipedia.org/wiki/Privacy_policy',
};
export const elementID: string = 'firebaseui-auth-container';
export const startFirebaseUI = () => {
  firebaseUI.start('#' + elementID, firebaseUiDefaultConfig);
};
```

Dane zalogowanego użytkownika dostępne są w przestrzeni aplikacji dzięki wykorzystaniu kontekstu [14] o nazwie `AuthContext`, którego definicja znajduje się w pliku `src/firebase/context/AuthContext.tsx`. Najważniejsze jego funkcje przedstawiono na listingu 3.12. Kluczowym działaniem jest subskrypcja `onAuthStateChanged`, która nasłuchuje odpowiedzi ze strony `Firebase`. W przypadku pojawienia się poprawnej odpowiedzi, otrzymywany jest obiekt użytkownika. Następnie obiekt ten jest zapisany jako `currentUser`. Wszystkie komponenty będące potomkami (widoczne na listingu 3.13) kontekstu posiadają dostęp do obiektu użytkownika poprzez hook [15] `useAuth`, zdefiniowany w pliku `src/firebase/hooks/useAuth.tsx`.

Listing 3.12: Komponent kontekstu uwierzytelniania

```
export const AuthContext = createContext<IAuthContext | undefined>(
  → undefined);
export const AuthContextProvider: React.FC = ({ children }) => {
  const [currentUser, setCurrentUser] = useState<User | null>(null);
  ...
  useEffect(() => {
    const unsubscribe = onAuthStateChanged(
      auth,
      (user) => {
        setCurrentUser(user);
      }
    );
  });
};
```

```

    },
    (error) => {
      setCurrentUser(null);
      ...
    }
  );
  return unsubscribe;
}, []);
...
const logoutUser = (): void => {
  signOut(auth);
};
const currentUserId = currentUser ? currentUser.uid : null;
const value: IAuthContext = {
  currentUser,
  currentUserId,
  updateUser,
  logoutUser,
};
return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
};

}
;

```

3.9. Implementacja routingu

Dynamiczny routing po stronie klienta zaimplementowano z wykorzystaniem biblioteki React Router. Strukturę routingu pokazano na listingu 3.13. Trasa do strony autoryzacji jest publiczna, natomiast trasy do stron: profilu, rejestratora GPS i aktywności są prywatne. W przypadku próby dostępu do niezdefiniowanej trasy, następuje przekierowanie na stronę NotFoundPage, która informuje użytkownika, że strona pod zadaną trasą nie istnieje.

Listing 3.13: Struktura aplikacji oraz routing aplikacji

```

const App: React.FC = () => {
  return (
    <Router>
      <AuthContextProvider>
        <PageLayout>
          <Switch>
            <RoutePublic exact path={`${Pages.Authorize}`}>
              <AuthPage />
            </RoutePublic>
            <RoutePrivate exact path={`${Pages.Profile}`}>
              <ProfilePage />
            </RoutePrivate>
            <RoutePrivate exact path={`${Pages.Tracker}`}>
              <TrackerPage />
            </RoutePrivate>
            <RoutePrivate exact path={`${Pages.Activities}`}>
              <ActivitiesPage />
            </RoutePrivate>
            <Route path='/*' component={NotFoundPage} />
          </Switch>
        </PageLayout>
      </AuthContextProvider>
    </Router>
  );
};

```

Do tras publicznych mają dostęp wszyscy użytkownicy niewierzytelni. Komponent `PublicPrivate` przedstawiono na listingu 3.15. Jeżeli trasę próbuje odwiedzić użytkownik uwierzytelniony, to następuje przekierowanie na stronę profilu użytkownika. Zabieg ten ma na celu zablokowanie dostępu do strony autoryzacji w przypadku, gdy w aplikacji jest już zalogowany użytkownik.

Listing 3.14: Publiczna trasa routingu

```
const RoutePublic: React.FC<RouteProps> = ({ children, ...rest }) => {
  const { currentUser } = useAuth();
  return (
    <Route
      {...rest}
      render={() => (!currentUser ? children : <Redirect to={Pages.Profile}>
        </>)}
    />
  );
};
```

Trasa prywatna to taka, do której dostęp mają jedynie użytkownicy uwierzytelni. Komponent `RoutePrivate` przedstawiono na listingu 3.15. Jeżeli w aplikacji jest uwierzytelniony użytkownik, to zwracany jest docelowy komponent trasy. W przeciwnym razie następuje przekierowanie na stronę uwierzytelniania.

Listing 3.15: Prywatna trasa routingu

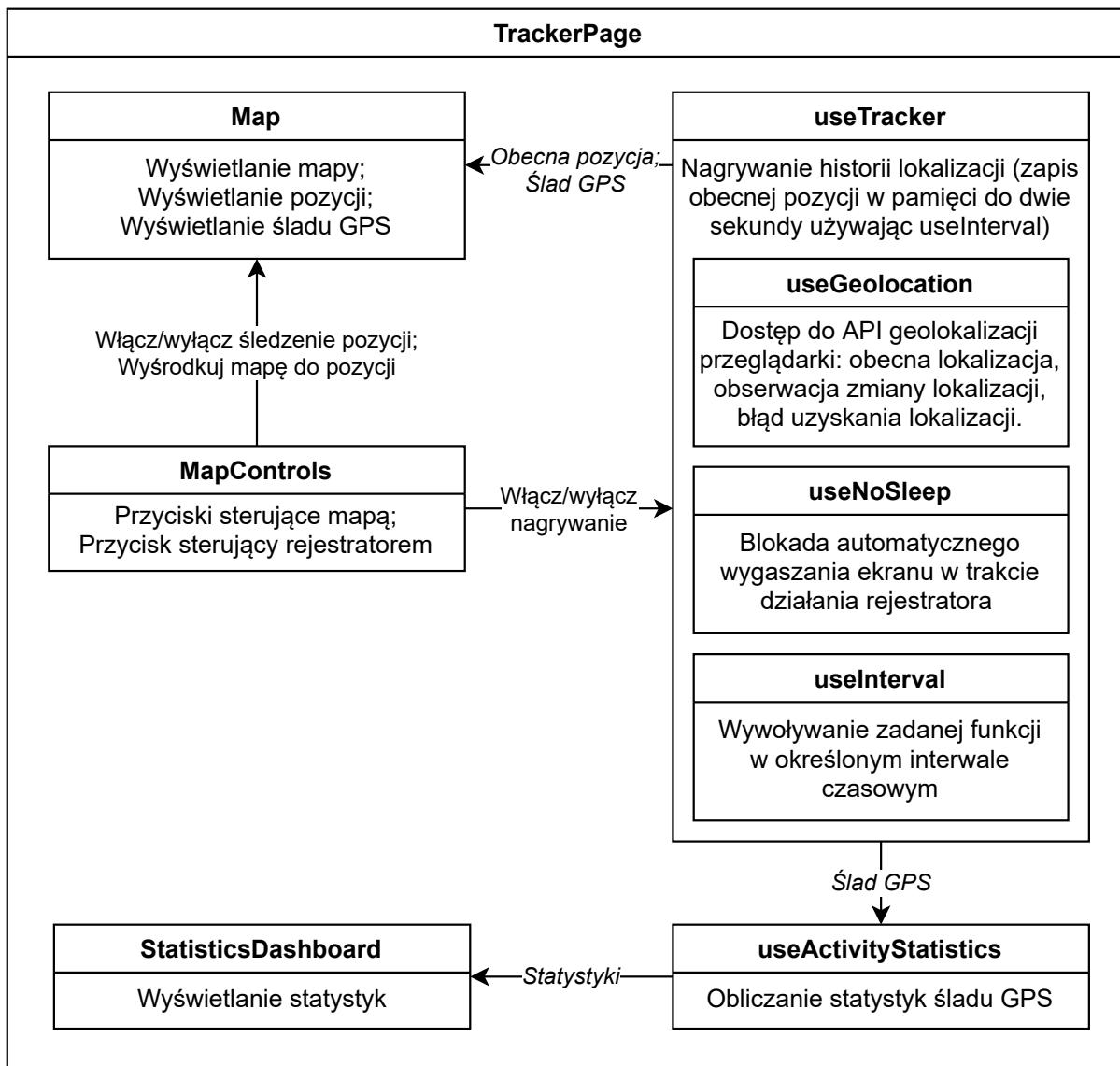
```
const RoutePrivate: React.FC<RouteProps> = ({ children, ...rest }) => {
  const { currentUser } = useAuth();
  return (
    <Route
      {...rest}
      render={() =>
        currentUser ? children : <Redirect to={Pages.Authorize}> />
      }
    />
  );
};
```

3.10. Implementacja rejestratora GPS

Schemat komponentu strony rejestratora GPS przedstawiono na rysunku 3.11, a widok aplikacji na rysunku 3.12. Komponent `TrackerPage` zwraca trzy komponenty:

1. Map — komponent mapy, odpowiedzialny za wyświetlanie mapy, obecnej pozycji na mapie (w postaci markera) oraz śladu GPS w postaci linii łamanej.
2. MapControls — komponent z przyciskami sterującymi zachowaniem mapy (śledzenie pozycji i centrowanie widoku do obecnej pozycji) oraz z przyciskiem włączającym lub wyłączającym rejestrator GPS.
3. MapDashboard — komponent wyświetlający statystki aktualnie nagrywanego śladu. Statystki wyliczane są w hooku `useActivityStatistics`.

Kluczowa logika strony zawarta jest w hooku `useTracker`, którego najważniejszym zadaniem jest rejestrowanie śladu GPS. Kod programu realizujący to zadanie przedstawiono na listingu 3.16. Po włączeniu nagrywania stan `track` ustawiany jest na wartość będącą tablicą zawierającą jeden pusty segment (pustą tablicę punktów), a stan `isTracking` ustawiany jest na wartość `true`, czego skutkiem jest aktywacja hooka `useInterval`. Interwał wywołuje funkcję `updateTrack` co 2000ms.



Rys. 3.10: Schemat strony rejestratora śladów GPS

Funkcja `updateTrack` najpierw sprawdza warunek, czy najnowsza pozycja jest dostępna (w przypadku błędu uzyskania lokalizacji przez przeglądarkę, ostatnia pozycja przyjmuje wartość `null`). Gdy pozycja jest niedostępna i gdy ostatni segment `tracka` nie jest pusty, do `tracka`, na koniec tablicy dodawany jest nowy pusty segment. W przeciwnym wypadku — gdy dostępna jest nowa pozycja — dopisywana jest ona na końcu ostatniego segmentu `tracka`. Ponadto, aktywacja nagrywania blokuje wygaszacz ekranu urządzenia (funkcje `enableNoSleep` `disableNoSleep` z `hooka` `useNoSleep`).

Listing 3.16: Strona rejestratora GPS

```
const emptySegment: TrackSegment = [];
const emptyTrack: Track = {
  activityId: '',
  segments: [emptySegment],
};
const useTracker = (interval: number = 2000): TrackRecorder => {
  const { enableNoSleep, disableNoSleep } = useNoSleep();
  const { startWatchingPosition, stopWatchingPosition, latestPosition,
    ↵ isWatchingPosition } = useGeolocation();
  const [isTracking, setIsTracking] = useState<boolean>(false);
```

```

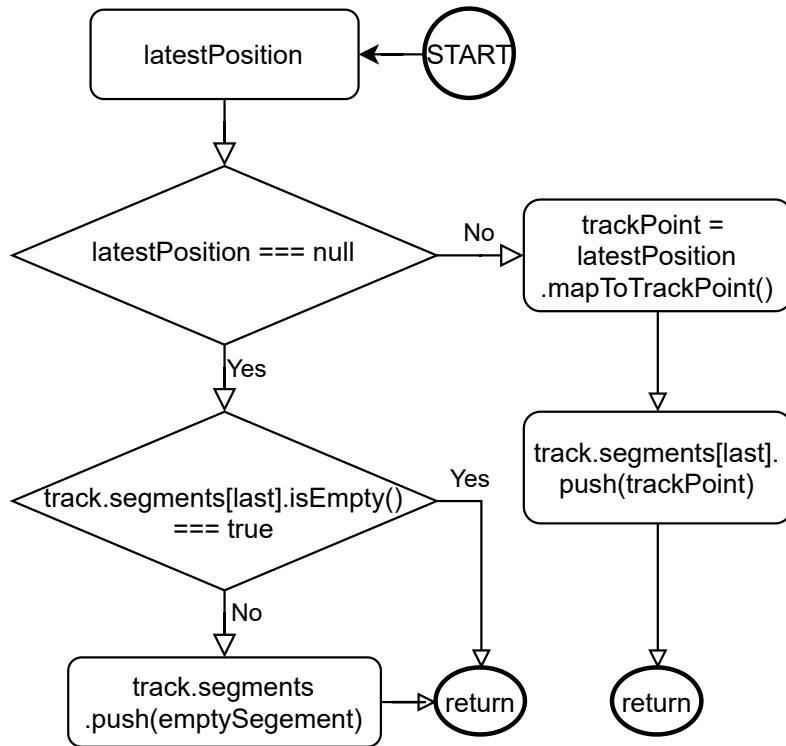
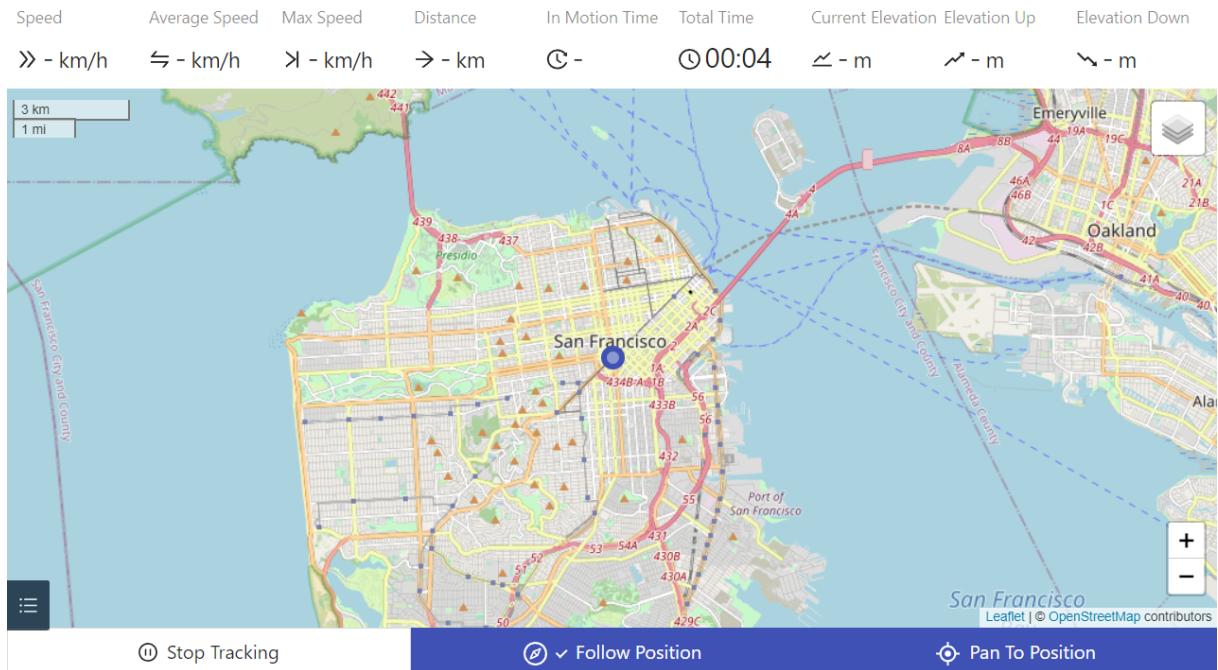
const [track, setTrack] = useState<Track>(emptyTrack);
useInterval(
() => {
  updateTrack();
},
isTracking ? interval : null
);
const updateTrack = () => {
  if (!latestPosition) {
    if (track.segments[track.segments.length - 1].length !== 0) {
      setTrack({ ...track, segments: [...track.segments, emptySegment] })
    }
    return;
  }
  const newTrackPoint: TrackPoint = {
    lat: latestPosition.coords.latitude,
    lon: latestPosition.coords.longitude,
    ele: latestPosition.coords.altitude || null,
    time: Date.now(),
  };
  const newTrack = { ...track, segments: [...track.segments] };
  newTrack.segments[newTrack.segments.length - 1].push(newTrackPoint);
  setTrack(newTrack);
};
const startTracking = (): void => {
  if (isTracking) { return; }
  if (!isWatchingPosition) {
    startWatchingPosition();
  }
  setTrack(emptyTrack);
  setIsTracking(true);
  enableNoSleep();
};
const stopTracking = (): void => {
  if (!isTracking) { return; }
  setIsTracking(false);
  disableNoSleep();
};
return {
  track, startTracking, stopTracking, isTracking, ...geolocation,
};
};

```

Dla lepszego zobrazowania działania funkcji `updateTrack`, na rysunku 3.11 przedstawiono diagram sekwencji funkcji. W pseudokodzie na schemacie przyjęto następujące oznaczenia:

- `[last]` w zapisie `track.segments[last]` oznacza odwołanie do ostatniego elementu w tablicy.
- Metoda `.isEmpty()` zwraca `true`, jeżeli tablica jest pusta, `false` w przeciwnym razie.
- Metoda `.push()` dodaje nowy element na koniec tablicy.
- Metoda `.mapToTrackPoint()` oznacza zmapowanie obiektu `GeolocationPosition` na obiekt `TrackPoint` (definicja typu — listing 3.8).

Obiekt `track` jest zwracany do komponentu `TrackerPage`, a następnie przekazywany do wyświetlenia do komponentu `Map`. Ponadto `track` jest również przekazywany do *hooka* `useActivityStatistics`. Dokonuje on wyliczeń statystyk i zwraca obiekt statystyk, który prezentowany jest przez komponent `StatisticsDashboard`.

Rys. 3.11: Diagram sekwencji dla funkcji `updateTrack`

Rys. 3.12: Widok strony nagrywania rejestratora z włączonym nagrywaniem śladu

3.11. Implementacja importu GPX

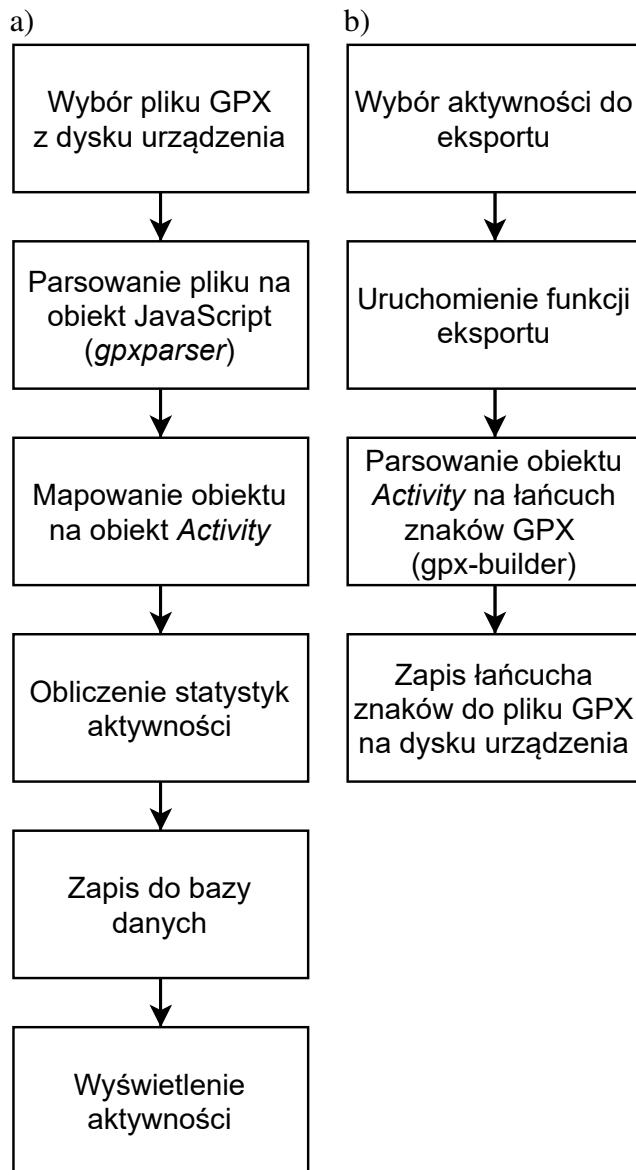
Schemat procesu importu aktywności z pliku GPX przedstawiono na rysunku 3.13a. Na początku użytkownik wybiera z dysku urządzenia pliki z rozszerzeniem `.gpx`. Następnie pliki są parsowane z wykorzystaniem biblioteki `gpxparser`. Rezultatem parsowania jest obiekt JavaScriptowy, który następnie mapowany jest na obiekt typu `Activity` (definicja typu — listing 3.7) oraz wyliczane są statystyki aktywności. W przedostatnim kroku dane zapisywane

są do bazy, w wyniku czego aktywności nadawany jest unikalny identyfikator. Na koniec aktywność wyświetdana jest na stronie. Jeżeli na dowolnym etapie wystąpi błąd, proces importu jest przerywany.

Kroki: parsowanie, mapowanie, obliczenie statystyk i zapis do bazy pokazano na listingu 3.17. Listing przedstawia kod *hooka useEffect* komponentu GpxToActivityParser. Widok aplikacji z interfejsem importu pokazano na rysunku 3.9.

Listing 3.17: Import aktywności z pliku GPX

```
useEffect(() => {
  file.text().then((gpx) => {
    const gpxParser = new GpxParser();
    gpxParser.parse(gpx);
    let lastTime = 0;
    const newTrack: Track = {
      activityId: '',
      segments: gpxParser.tracks.map((parsedTrackSegment) => {
        const newTrackSegment: TrackSegment = parsedTrackSegment.points.map(
          (parsedTrackPoint) => {
            lastTime = parsedTrackPoint.time.getTime();
            const newTrackPoint: TrackPoint = {
              lat: parsedTrackPoint.lat,
              lon: parsedTrackPoint.lon,
              ele: parsedTrackPoint.ele,
              time: lastTime,
            };
            return newTrackPoint;
          }
        );
        return newTrackSegment;
      }),
    };
    const { latestSpeed, latestElevation, ...statistics } =
      calculateStatistics(newTrack);
    const newActivity: Activity = {
      activityId: '',
      creatorId: currentUserId || '',
      name: he.decode(gpxParser.metadata.name),
      createdAt: newTrack.segments[0][0].time,
      lastModifiedAt: Date.now(),
      startTime: newTrack.segments[0][0].time,
      endTime: lastTime,
      sport: ActivitySportTypes.Other,
      category: ActivityCategoryTypes.Other,
      shape: { isLoop: false, from: 'unknown', to: 'unknown' },
      statistics: statistics,
    };
    writeActivityWithTrack(currentUserId, newActivity, newTrack)
      .then(({ updatedActivity }) => {
        setActivity(updatedActivity);
        setError(false);
      })
      .catch(() => {
        setActivity(newActivity);
        setError(true);
      });
  });
}, [file, currentUserId]);
```



Rys. 3.13: Schematy procesów: a) importu aktywności, b) eksportu aktywności

3.12. Implementacja eksportu GPX

Schemat procesu eksportu aktywności do pliku GPX przedstawiono na rysunku 3.13b. W widoku aktywności (rys. 3.9) użytkownik wybiera aktywność, która ma zostać wyeksportowana. Naciśnięcie przycisku `Export GPX` wywołuje funkcję `buildGpxAndSaveFile`, której kod przedstawiono na listingu 3.18. Funkcja przyjmuje dwa parametry: obiekt typu `Track` (definicja typu — 3.8) oraz `filename`.

W pierwszym kroku w funkcji pomocniczej `buildGpx` obiekt `track` mapowany jest na tablicę obiektów typu `Point` zdefiniowanego w bibliotece GPX builder. Następnie funkcja `buildGPX` generuje łańcuch znaków, który jest reprezentacją `tracka` w standardzie GPX. łańcuch znaków zapisywany jest do pliku z rozszerzeniem `*.gpx` funkcją `saveAs` dostarczoną przez bibliotekę `file-saver`.

Listing 3.18: Eksport aktywności do pliku GPX

```
export const buildGpxAndSaveFile = (track: Track, fileName: string) => {
  const gpx = buildGpx(track);
  const blob = new Blob([gpx], {
```

```

    type: 'application/gpx+xml; charset=utf-8',
});
saveAs(blob, fileName + '.gpx');
};

export const buildGpx = (track: Track) => {
  const gpxData = new BaseBuilder();
  track.segments.forEach((segment) => {
    const points = segment.map(
      (point) =>
        new Point(point.lat, point.lon, {
          ele: point.ele || undefined,
          time: new Date(point.time),
        })
    );
    gpxData.setSegmentPoints(points);
  });
  return buildGPX(gpxData.toObject());
};

```

3.13. Implementacja aktywności

Strona aktywności `ActivityPage` rys. 3.9 składa się z dwóch komponentów: importu aktywności (umiejscowiony na górze strony) oraz listy zapisanych aktywności. Każdy element tej listy to komponent `Activity`. W komponencie tym prezentowane są dane aktywności, ślad GPS na mapie, statystyki aktywności oraz wykresy aktywności (rys. 3.15). Oprócz wyświetlania danych dostępne są trzy funkcjonalności: edycja danych aktywności, usunięcie aktywności oraz eksport aktywności. Kod funkcji przedstawiono na listingu 3.19.

Listing 3.19: Funkcje: edycji, usunięcia oraz eksportu aktywności

```

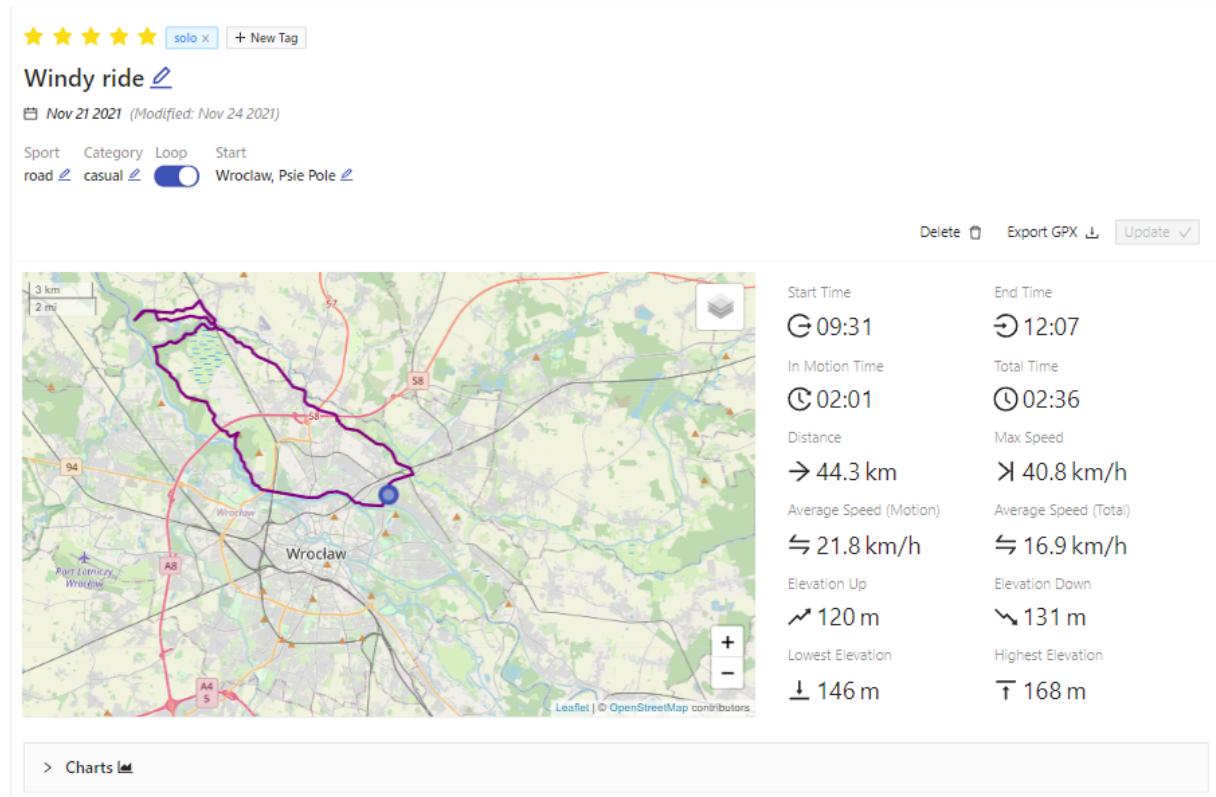
const onDeleteActivity = () => {
  deleteActivityWithTrack(currentUserId, activity.activityId)
    .then(() => {
      message.success('Activity deleted');
    })
    .catch(() => {
      message.error("Activity couldn't be deleted");
    });
};

const onUpdateActivity = () => {
  const payload: { [filed: string]: any } = {};
  payload['/lastModifiedAt'] = Date.now();
  updates.name && (payload['/name'] = name);
  updates.sport && (payload['/sport'] = sport);
  updates.category && (payload['/category'] = category);
  updates.shape && (payload['/shape'] = shape);
  updates.rating && (payload['/rating'] = rating || null);
  updates.tags && (payload['/tags'] = tags || null);
  updateActivity(currentUserId, activity.activityId, payload)
    .then(() => {
      message.success('Activity updated');
    })
    .catch(() => {
      message.error("Activity couldn't be updated");
    });
  setUpdates(ActivityNotUpdated);
  setActivityModified(false);
};

```

```
const onExportActivity = () => {
  if (!track) {
    message.error("Activity couldn't be exported");
    return;
  }
  buildGpxAndSaveFile(track, activity.name);
};
```

Naciśnięcie przycisku **Delete** wywołuje funkcję `deleteActivityWithTrack`, która usuwa z bazy danych użytkownika aktywność o zadanym identyfikatorze. Naciśnięcie przycisku **Update** (domyślnie zablokowany, odblokowuje się po wprowadzeniu zmiany w danych aktywności) powoduje najpierw utworzenie obiektu `payload`, który zawiera listę zmian aktywności, a następnie wywołanie funkcji `updateActivity`, która dokonuje aktualizacji w bazie danych użytkownika. Funkcję eksportu opisano w sekcji 3.12, a komunikację z bazą danych — w sekcji 3.16.



Rys. 3.14: Widok aktywności

3.14. Implementacja statystyk aktywności

Funkcjonalność obliczania statystyk zaimplementowano w `hooku useActivityStatistics`. Hook zwraca obiekt statystyk, który zawiera następujące dane:

- `latestSpeed` — ostatnia (najnowsza) prędkość;
- `maxSpeed` — maksymalna prędkość;
- `latestElevation` — ostatnia (najnowsza) wysokość nad poziomem morza;
- `minElevation` — minimalna wysokość nad poziomem morza;
- `maxElevation` — maksymalna wysokość nad poziomem morza;
- `elevationUp` — suma przewyższeń pod góre;

- `elevationDown` — suma przewyższeń z góry;
- `totalDistance` — całkowity przejechany dystans;
- `totalDuration` — całkowity czas podróży;
- `inMotionDuration` — czas podróży w ruchu.

Listing 3.20: Hook `useActivityStatistics`

```
export type ActivityStatisticsExtended = {
  latestSpeed?: number; latestElevation?: number;
} & ActivityStatistics;
const useActivityStatistics = (track: Track): ActivityStatisticsExtended =>
  ↪ {
    const [latestSpeed, setLatestSpeed] = useState<number>();
    const [latestElevation, setLatestElevation] = useState<number>();
    const [totalDistance, setTotalDistance] = useState<number>();
    const [totalDuration, setTotalDuration] = useState<number>();
    const [inMotionDuration, setInMotionDuration] = useState<number>();
    const [maxSpeed, setMaxSpeed] = useState<number>();
    const [elevationUp, setElevationUp] = useState<number>();
    const [elevationDown, setElevationDown] = useState<number>();
    const [minElevation, setMinElevation] = useState<number>();
    const [maxElevation, setMaxElevation] = useState<number>();
    useEffect(() => {
      const s = calculateStatistics(track);
      setMaxSpeed(s.maxSpeed);
      setLatestSpeed(s.latestSpeed);
      setLatestElevation(s.latestElevation);
      setTotalDuration(s.totalDuration);
      setTotalDistance(s.totalDistance);
      setInMotionDuration(s.inMotionDuration);
      setElevationUp(s.elevationUp);
      setElevationDown(s.elevationDown);
      setMinElevation(s.minElevation);
      setMaxElevation(s.maxElevation);
    }, [track]);
    return { latestSpeed, latestElevation, totalDistance, totalDuration,
      ↪ inMotionDuration, maxSpeed, elevationUp, elevationDown,
      ↪ minElevation, maxElevation };
  };
}
```

Kod *hooka* przedstawiono na listingu 3.20. Dla każdej statystyki zaimplementowano osobny stan, a wszystkie stany zwracane są z *hooka* w jednym obiekcie. Przy każdej modyfikacji obiektu `track` wywoływana jest funkcja `calculateStatistics(track)` która wylicza wszystkie statystyki. Z racji, że funkcja jest obszerna, jej kodu nie zawarto na żadnym listingu. Natomiast algorytm jej działania opisuje następujący pseudokod:

1. Zainicjalizuj zmienne wartością zerową: `latestElevation`, `totalDuration`, `latestSpeed`, `maxSpeed`, `totalDistance`, `inMotionDuration`, `elevationUp`, `elevationDown`, `minElevation`, `maxElevation`.
2. Wykonaj `trackFlat = track.segments.flat()` — spłaszcz tablicę wielowymiarową do tablicy jednowymiarowej.
3. Jeżeli tablica `trackFlat` jest pusta, idź do 11. W przeciwnym razie idź do 3.
4. Zainicjalizuj `ftp` jako pierwszy obiekt z `trackFlat`.
5. Zainicjalizuj `ltp` jako ostatni obiekt z `trackFlat`.
6. Przypisz `latestElevation` jako `ftp.ele`.
7. Przypisz `minElevation` jako `ftp.ele`.
8. Przypisz `maxElevation` jako `ftp.ele`.

9. Jeżeli `trackFlat.length >= 2`, oblicz i przypisz `latestSpeed = geoSpeed(ostatni element trackFlat, przedostatni element trackFlat)`.
10. Dla każdego elementu `currTrackPoint` oraz `prevTrackPoint` takiego, że `prevTrackPoint` jest bezpośrednim poprzednikiem `currTrackPoint`:
 - 10.1. Oblicz i przypisz `{distance, speed, time, deltaElevation} = geoMove(currTrackPoint, prevTrackPoint)`.
 - 10.2. Jeżeli `speed > 0`, idź do 10.3. W przeciwnym razie idź do 10.1 kolejnej iteracji.
 - 10.3. Przypisz `totalDistance += distance`.
 - 10.4. Przypisz `inMotionDuration += time`.
 - 10.5. Jeżeli `speed` jest większe od `maxSpeed`, to przypisz: `maxSpeed = speed`.
 - 10.6. Jeżeli `dElevation > 0`, to przypisz `elevationDown += dElevation`. W przeciwnym razie przypisz `elevationUp += -dElevation`
 - 10.7. Jeżeli `currTrackPoint.ele` jest większe od `maxElevation`, to przypisz `maxElevation = currTrackPoint.ele`
 - 10.8. Jeżeli `currTrackPoint.ele` jest mniejsze od `minElevation`, to przypisz `minElevation = currTrackPoint.ele`
11. Zwróć obiekt z danymi: `{latestElevation, totalDuration, latestSpeed, maxSpeed, totalDistance, inMotionDuration, elevationUp, elevationDown, minElevation, maxElevation}`.

Kod funkcji `geoMove` pokazano na listingu 3.21. Korzysta ona z funkcji pomocniczych `geoDistance`, `deltaTime`, `geoSpeed1`, `deltaElevation`, które dokonują odpowiednich obliczeń. Szczególnie interesująca jest funkcja `geoDistance`, która korzysta z formuły *Haversine* [12]. Jest to formuła, która pozwala wyliczyć najmniejszą odległość między dwoma punktami na powierzchni sfery. Odpowiedni kod przedstawiono na listingu 3.22. Funkcje `deltaTime`, `deltaElevation` sprowadzają się do obliczenia jedynie różnicy dwóch wartości (dla czasu zwracana jest wartość bezwzględna), natomiast `geoSpeed1` implementuje formułę wyliczającą prędkość w ruchu jednostajnym (dystans całkowity dzielony przez czas całkowity).

Listing 3.21: Funkcja `geoMove`

```
export const geoMove = (
  lat1: number, lon1: number, time1: number, lat2: number, lon2: number,
  ↵ time2: number, ele1?: number | null, ele2?: number | null
): {
  distance: number; speed: number; time: number; dElevation: number |
  ↵ undefined;
} => {
  const distance = geoDistance(lat1, lon1, lat2, lon2); // m
  const time = deltaTime(time1, time2); // ms
  const speed = geoSpeed1(distance, time); // m/s
  const dElevation = deltaElevation(ele1, ele2); // m
  return { distance, speed, time, dElevation };
};
```

Listing 3.22: Funkcja `geoDistance`

```
export const toRad = (value: number) => {
  return (value * Math.PI) / 180;
};

export const geoDistance = (
  lat1: number, lon1: number, lat2: number, lon2: number
) => {
  const R = 6371 * 1000; // Radius of the earth in m
  const dLat = toRad(lat2 - lat1);
  const dLon = toRad(lon2 - lon1);
```

```

const a =
  Math.sin(dLat / 2) * Math.sin(dLat / 2) +
  Math.cos(toRad(lat1)) *
    Math.cos(toRad(lat2)) *
    Math.sin(dLon / 2) *
    Math.sin(dLon / 2);
const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
return R * c; // m
};

```

3.15. Implementacja wykresów aktywności

Wykresy (rys. 3.15) wyświetlane są przez komponent `ActivityCharts`, który otrzymuje (jako właściwość) obiekt `track`. W pierwszej kolejności jest on mapowany na serie danych obsługiwane przez wykresy z biblioteki `React-Vis`. W najprostszej wersji seria danych dla używanych wykresów typu `AreaChart` to tablica obiektów z dwoma kluczami: `x` oraz `y`, odpowiadającymi wartościami odpowiednim osiom wykresu. Kod odpowiedzialny za mapowanie przedstawiono na listingu 3.23. Kompletna lista dostępnych wykresów aktywności jest następująca:

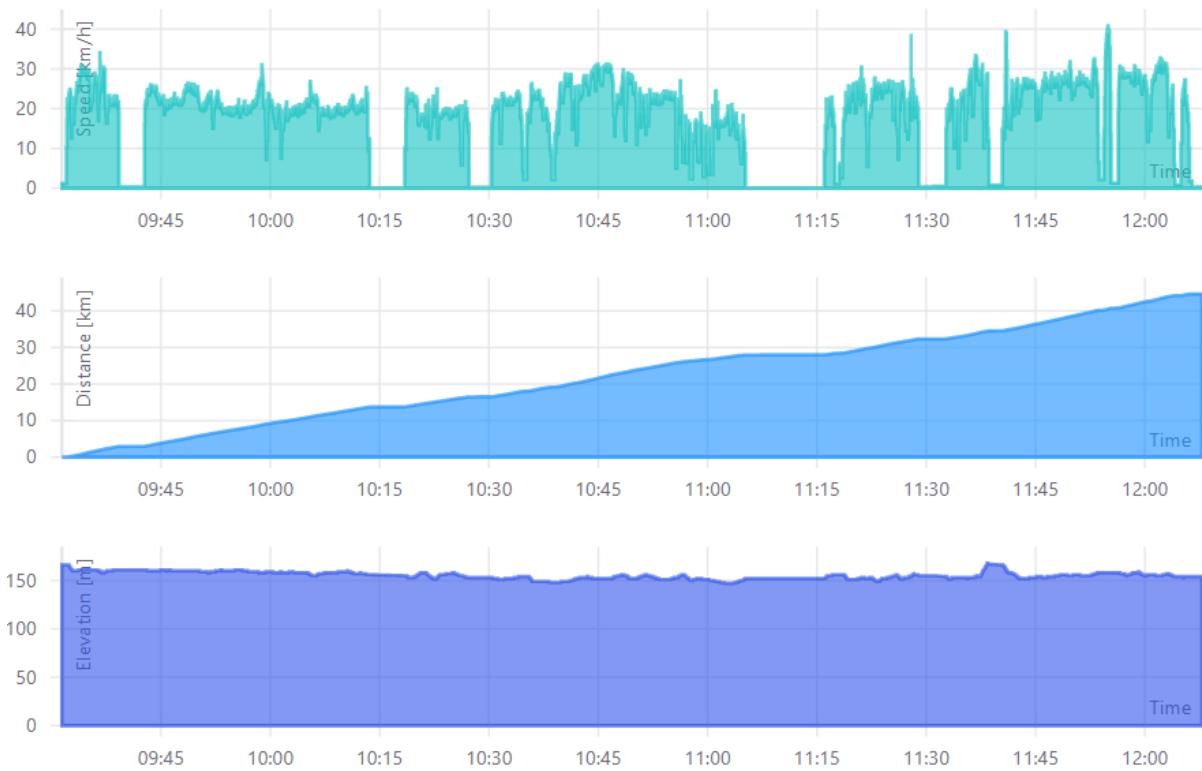
- wykres zależności prędkości od czasu;
- wykres zależności pokonanego dystansu od czasu;
- wykres zależności bezwzględnej wysokości nad poziomem morza od czasu.

Listing 3.23: Fragment funkcji generowania serii danych dla wykresów aktywności

```

const parseData = async () => {
  ...
  flatTrack.forEach((point, index) => {
    const newElevation = Math.round(point.ele || 0);
    if (index !== 0) {
      const prevPoint = flatTrack[index - 1];
      const { distance, speed } = geoMove(
        point.lat,
        point.lon,
        point.time,
        prevPoint.lat,
        prevPoint.lon,
        prevPoint.time
      );
      newDistance += distance / 1000;
      newSpeed = speed * 3.6;
    }
    ...
    newDistanceData.push({ x: point.time, y: newDistance });
    newSpeedData.push({ x: point.time, y: newSpeed });
    newElevationData.push({
      x: point.time,
      y: newElevation,
    });
  });
  ...
};

```



Rys. 3.15: Widok wykresów aktywności

3.16. Integracja z Realtime Database

W celu integracji z bazą danych w pierwszym kroku należy w aplikacji zainicjalizować obiekt bazy danych, jak pokazano na listingu 3.2 (obiekt `database`). Wszystkie funkcje odpowiedzialne za komunikację z bazą zaimplementowano w pliku `src/firebase/hooks/useDatabase.tsx`. Ponadto w pliku `src/firebase/context/AuthContext.tsx` zaimplementowano funkcję `updateProfile`. Różni się ona od pozostałych funkcji tym, że nie komunikuje się bezpośrednio z modelem bazy danych, tylko z modelem uwierzytelniania. Lista i opis zaimplementowanych funkcji przedstawiono w tabeli 3.2. Jest to minimalny i wystarczający zestaw funkcji koniecznych do realizacji wszystkich wymagań funkcjonalnych projektu.

Tab. 3.2: Komunikacja z bazą danych

Nazwa	Typ	Operacja	Opis
<code>useReadActivities</code>	<i>hook</i>	READ	Odczyt wszystkich aktywności użytkownika
<code>useReadTrack</code>	<i>hook</i>	READ	Odczyt śladu GPX użytkownika
<code>useReadProfile</code>	<i>hook</i>	READ	Odczyt danych profilu użytkownika
<code>writeActivityWithTrack</code>	funkcja	WRITE	Zapis aktywności wraz ze śladem GPS
<code>deleteActivityWithTrack</code>	funkcja	DELETE	Usunięcie aktywności wraz ze śladem GPS
<code>updateActivity</code>	funkcja	UPDATE	Aktualizacja danych aktywności
<code>updateProfile</code>	funkcja	UPDATE	Aktualizacja danych profilu
<code>updateUser</code>	funkcja	UPDATE	Aktualizacja danych profilu (tylko: imię, nazwisko, awatar)

Odczyt z bazy danych powoduje zarejestrowanie słuchaczy (ang. *listener registration*), dzięki czemu zmiany w bazie danych są od razu przesyłane do aplikacji klienta. Aby w pełni wyko-

rzystać potencjał tej funkcjonalności, wszystkie operacje READ, czyli: odczyt wszystkich aktywności użytkownika, odczyt śladu GPX użytkownika i odczyt danych profilu użytkownika zaimplementowano jako *hooki*, zamiast jako zwykłe funkcje. Pozostałe operacje można nazwać *operacjami jednostronnymi*, tzn.: wysyłają one jedynie zapytanie do bazy danych i nie rejestrują żadnych słuchaczy. Z tego powodu zaimplementowano je jako zwykłe funkcje.

Hook odczytu pokazano na listingu 3.24 na przykładzie `useReadActivities`. Zawiera on trzy stany:

1. `profile` — odczytane dane.
2. `loading` — odczytywanie danych.
3. `error` — błąd odczytu danych.

Funkcja `onValue` rejestruje słuchacza, który z każdą zmianą otrzymaną z bazy danych wywołuje funkcję *callback* przekazaną jako drugi parametr. *Hooki* `useReadTrack` i `useReadActivities` zaprojektowano w pełni analogicznie. Różnią się głównie typami danych oraz ścieżkami odczytu.

Listing 3.24: *Hook* odczytu danych profilu z bazy danych

```
export const useReadProfile = (userId: string | null) => {
  const [profile, setProfile] = useState<UserProfile | null>(null);
  const [loading, setLoading] = useState<boolean>(true);
  const [error, setError] = useState<boolean>(false);
  useEffect(() => {
    let unsubscribe;
    try {
      if (!userId) { throw new Error('useReadProfile failed. UserID cannot
        ↪ be null.');?>
      const path = 'users/' + userId + '/profile';
      unsubscribe = onValue(ref(database, path), (snapshot) => {
        if (snapshot.exists()) { setProfile(snapshot.val()); }
        setLoading(false);
        setError(false);
      });
    } catch (er) {
      setProfile(null);
      setLoading(false);
      setError(true);
    } finally {
      return unsubscribe;
    }
  }, [userId]);
  return { profile, loading, error };
};
```

Szczególnie interesująca jest funkcja `writeActivityWithTrack`, której kod pokazano na listingu 3.25. Obiekty `track` i `activity` w aplikacji klienta są silnie ze sobą powiązane — `track` można uznać za element należący do `activity`. Zatem wydawać by się mogło, że w bazie danych `track` powinien być zapisany jako potomek `activity`. Tymczasem w celu optymalizacji zapytań do bazy danych, obiekty te zapisywane są do różnych ścieżek, jednak pod wspólnym identyfikatorem. W wyniku wywołania funkcji `push` zwracany jest obiekt zawierający nowo wygenerowany, unikalny klucz, który służy jako wspólny identyfikator dla obiektów `activity` i `track`. Następnie obiekty zapisywane są do bazy wywołaniem funkcji `set`.

Listing 3.25: Funkcja zapisu aktywności i śladu do bazy danych

```
export const writeActivityWithTrack = async (
  userId: string | null,
  activity: Activity,
  track: Track
```

```

): Promise<{ updatedActivity: Activity; updatedTrack: Track }> => {
  try {
    if (!userId) { throw new Error('UserID cannot be null'); }
    const activityParentPath = 'users/' + userId + '/activities';
    const activityId = push(child(ref(database), activityParentPath)).key
      ↪ || '';
    const updatedActivity = { ...activity, activityId };
    set(ref(database, activityParentPath + '/' + activityId),
      ↪ updatedActivity);
    const trackParentPath = 'users/' + userId + '/tracks';
    const updatedTrack = { ...track, activityId };
    set(ref(database, trackParentPath + '/' + activityId), updatedTrack);
    return { updatedActivity, updatedTrack };
  } catch (er) {
    throw new Error('writeActivityWithTrack failed. ${er}');
  }
};

```

Ze względu na silne powiązanie activity — track i sposób zapisu do bazy, usunięcie danych wymaga wykonania operacji dla dwóch ścieżek, co pokazano na listingu 3.26. Usunięcie dokonuje się przez wywołanie funkcji `remove`.

Listing 3.26: Funkcja usunięcia aktywności i śladu z bazy danych

```

export const deleteActivityWithTrack = async (
  userId: string | null,
  activityId: string
): Promise<void> => {
  try {
    if (!userId) { throw new Error('UserID cannot be null'); }
    const activityPath = 'users/' + userId + '/activities/' + activityId;
    const trackPath = 'users/' + userId + '/tracks/' + activityId;
    remove(ref(database, activityPath));
    remove(ref(database, trackPath));
    return;
  } catch (er) {
    throw new Error('deleteActivityWithTrack failed. ${er}');
  }
};

```

Wszystkie funkcje wykonujące operacje UPDATE działają w bardzo zbliżony do siebie sposób. Przykład — funkcję `updateProfile` — podano na listingu 3.27. Funkcje aktualizujące przyjmują jako parametr obiekt `payload`, który zawiera ścieżki względne do aktualizowanych kluczy, oraz nowe wartości dla tych kluczy. Zapytanie do bazy wykonywane jest przez wywołanie `updateActivity`.

Listing 3.27: Funkcja aktualizacji danych profilu

```

export const updateActivity = async (
  userId: string | null,
  activityId: string,
  payload: { [field: string]: any }
) => {
  try {
    if (!userId) { throw new Error('UserID cannot be null'); }
    const activityPath = 'users/' + userId + '/activities/' + activityId;
    update(ref(database, activityPath), payload);
    return;
  } catch (er) {
    throw new Error('updateActivity failed. ${er}');
  }
};

```

Baza danych zabezpieczono prostą regułą: tylko użytkownik uwierzytelniony może odczytywać tylko własne dane. Regułę pokazano na listingu 3.28

Listing 3.28: Reguły zabezpieczeń bazy danych

```
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

3.17. Implementacja profilu użytkownika

Strona profilu użytkownika (rys. 3.16) składa się z czterech głównych sekcji: awatara, danych użytkownika, statystyk oraz wykresów. W sekcji awatara zawarto zdjęcie użytkownika. Kliknięcie zdjęcia powoduje pojawienie się pola tekstowego, w którym użytkownik może podać adres URL nowego zdjęcia. Użytkownik może również edytować swoje imię i nazwisko. Powyższe dane aktualizowane są po stronie backendu wywołaniami funkcji `updateUser` (tab. 3.2), co przedstawiono na listingu 3.29.

Listing 3.29: Aktualizacja zdjęcia oraz imienia i nazwiska profilu

```
const onNameChange = (newName: string) => {
  if (newName === '') {
    message.error('Invalid name');
    return;
  }
  updateUser({ displayName: newName })
    .then(() => {
      message.success('Name updated');
      setName(newName);
    })
    .catch(() => {
      message.error("Name couldn't be updated");
    });
};

const onUrlChange = (newUrl: string) => {
  updateUser({ photoURL: newUrl })
    .then(() => {
      message.success('Photo updated');
      setUrl(newUrl);
    })
    .catch(() => {
      message.error("Photo couldn't be updated");
    });
};
```

W karcie `Bio` prezentowane są dane użytkownika odczytywane z obiektu typu `UserProfile` (lis. 3.6). Ich edycja przebiega podobnie jak w przypadku edycji imienia i nazwiska, jednak tym razem wywoływana jest funkcja `updateProfile` (tab. 3.2). Przykład pokazano na lisitngu 3.30. Aktualizacja pozostałych pól przebiega analogicznie w stosunku do funkcji `onDescriptionChange`.

Listing 3.30: Aktualizacja danych profilu na przykładzie zmiany opisu

```

const onProfileUpdate = async (
  field: string,
  payload: { [filed: string]: any }
) => {
  updateProfile(currentUserId, payload)
    .then(() => {
      message.success(field + ' updated');
    })
    .catch(() => {
      message.error(field + " couldn't be updated");
    });
};

const onDescriptionChange = (newDescription: string) => {
  onProfileUpdate('Description', { description: newDescription }).then(() => {
    setDescription(newDescription);
  });
};

```

The screenshot displays a user profile interface. At the top left is a circular profile picture of a man. To its right, the name "Paul Foster" is shown with a pencil icon indicating editability. The main content area is divided into several sections:

- Bio**: Contains fields for Description ("Bikers life"), Gender ("male"), Weight ("74 kg"), Country ("United States"), Birthday ("1999-01-12"), Height ("170 cm"), and City ("Durango").
- Statistics**: Shows Number of Activities (4), Max Speed (37.1 km/h), First Activity At (Oct 25 2021), Last Activity At (Nov 21 2021), Total Duration (07:31 h), Total Duration (Motion) (05:56 h), Average Duration (01:52 h), and Average Duration (Motion) (01:29 h). A bar chart below these statistics shows distance over time.
- Charts**: A section titled "Last Month" containing a chart titled "Daily Distance". The chart shows two bars: one very tall bar reaching approximately 45 on Nov 20, 2021, and a shorter bar reaching approximately 18 on Oct 30, 2021. The x-axis lists dates from Mon Nov 29 2021 to Sat Oct 30 2021.

Rys. 3.16: Widok strony profilu użytkownika

3.18. Implementacja statystyk profilu użytkownika

Funkcję obliczającą statystyki profilu zaimplementowano w komponencie `ProfileDashboard` (rys. 3.17). Algorytm funkcji polega na iteracji przez tablicę wszystkich aktywności (obiekty

typu `Activity` — listing 3.7) użytkownika. W danych wyszukiwane są (poprzez zliczanie wystąpień wartości lub proste porównania wartości) następujące wartości:

- `numberOfActivities` — całkowita liczba aktywności;
- `maxSpeed` — maksymalna zarejestrowana prędkość;
- `firstActivityAt` — data pierwszej aktywności;
- `lastActivityAt` — data ostatniej aktywności;
- `totalDuration` — całkowity czas aktywności (czas całkowity);
- `averageDuration` — średni czas trwania aktywności (czas całkowity);
- `minDuration` — najkrótsza aktywność (czas całkowity);
- `maxDuration` — najdłuższa aktywność (czas całkowity);
- `averageSpeed` — średnia prędkość (czas całkowity);
- `inMotionDuration` — całkowity czas aktywności (czas w ruchu);
- `averageInMotionDuration` — średni czas trwania aktywności (czas w ruchu);
- `minInMotionDuration` — najkrótsza aktywność (czas w ruchu);
- `maxInMotionDuration` — najdłuższa aktywność (czas w ruchu);
- `averageSpeedInMotion` — średnia prędkość (czas w ruchu);
- `totalDistance` — całkowity dystans;
- `averageDistance` — średni dystans;
- `minDistance` — najkrótszy dystans;
- `maxDistance` — najdłuższy dystans;
- `totalElevationUp` — suma przewyższeń w górę;
- `totalElevationDown` — suma przewyższeń w dół;
- `minElevation` — najniższe zarejestrowane położenie nad poziomem morza;
- `maxElevation` — najwyższe zarejestrowane położenie nad poziomem morza.

Ze względu na rozmiar (długość kodu) funkcji oraz jej trywialną implementację, kodu nie pokazano na żadnym listingu.

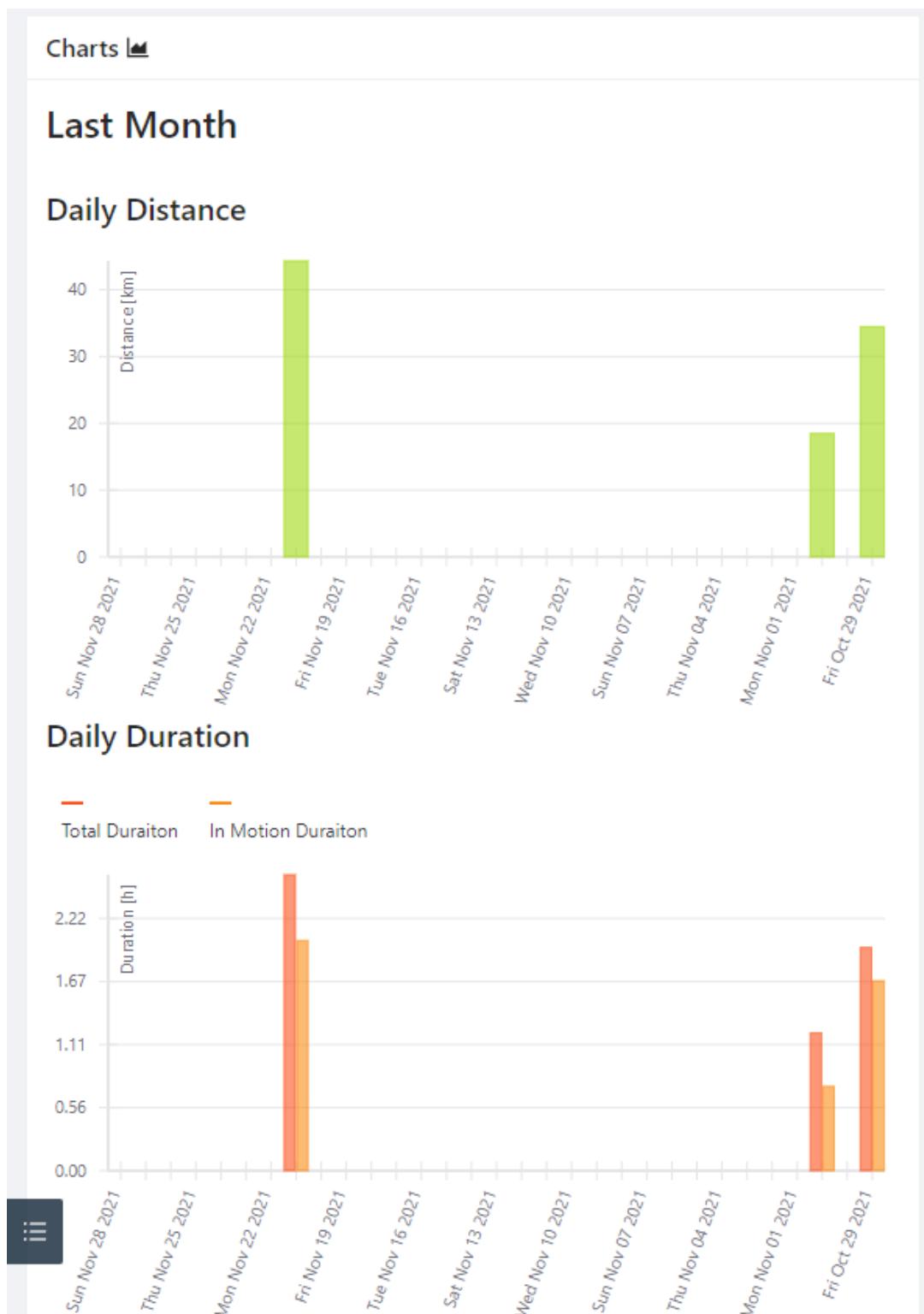
3.19. Implementacja wykresów profilu użytkownika

Komponent `ProfileCharts` odpowiedzialny jest za prezentację wykresów profilu użytkownika. Komponent przyjmuje jako właściwość tablicę obiektów `Activity`. Na początku dane aktywności mapowane są na serie danych obsługiwane przez wykresy z biblioteki `React-Vis`. Następnie serie danych przekazywane są do komponentów odpowiednich wykresów, które je formatują i wyświetlają. Przykładowe wykresy zaprezentowano na rysunku 3.18. Kompletna lista wykresów profilu użytkownika jest następująca:

- dzienny dystans na przestrzeni ostatnich 30 dni;
- dzienny czasu aktywności na przestrzeni ostatnich 30 dni;
- dzienne przewyższenia na przestrzeni ostatnich 30 dni;
- najczęściej wybierane rodzaje sportu;
- najczęściej wybierane kategorie;
- najczęściej wybierane oceny;
- najczęściej używane tagów;
- najczęściej wybierane kształty aktywności.

Statistics ↗	
Number of Activities	Max Speed
4	37.1 km/h
First Activity At	Last Activity At
Oct 25 2021	Nov 21 2021
Total Duration	Total Duration (Motion)
07:31 h	05:56 h
Average Duration	Average Duration (Motion)
01:52 h	01:29 h
Min Duration	Min Duration (Motion)
01:12 h	00:44 h
Max Duration	Max Duration (Motion)
02:36 h	02:01 h
Average Speed	Average Speed (Motion)
17.6 km/h	22.3 km/h
Total Distance	Average Distance
132.8 km	33.2 km
Min Distance	Max Distance
18.4 km	44.3 km
Total Elevation Up	Total Elevation Up
468 m	495 m
Lowest Elevation	Highest Elevation
133 m	175 m

Rys. 3.17: Widok statystyk profilu użytkownika



Rys. 3.18: Widok wykresów profilu użytkownika

Rozdział 4

Podsumowanie

Przed przystąpieniem do realizacji projektu za cel pracy postawiono utworzenie aplikacji internetowej przystosowanej do urządzeń mobilnych i desktopowych, dedykowanej dla rowerzystów, która ma udostępniać podstawowe funkcjonalności komputera rowerowego. Użytkownik powinien móc utworzyć własne konto, co otwiera dostęp do wszystkich funkcjonalności aplikacji. Kluczową funkcjonalnością aplikacji ma być nagrywanie śladów GPS przejechanych tras rowerowych, które ogólnie nazywane są aktywnościami. Aktywności powinny być zapisywane w bazie danych na serwerze, aby użytkownik miał do nich dostęp z dowolnego urządzenia. Aplikacja powinna obliczać statystyki poszczególnych aktywności oraz globalne statystyki użytkownika i prezentować je w postaci liczb oraz wykresów. Ponadto, powinna istnieć możliwość importu i eksportu aktywności przy użyciu standardu GPX.

Wymienione powyżej funkcjonalności zostały zaimplementowane i działają zgodnie z założeniami. Podjęte decyzje projektowe pozwoliły na skutecną realizację projektu w zaplanowanym czasie. Jednak mimo że projekt zakończył się sukcesem, dobrane technologie — przede wszystkim **Firebase** — wymusiły dokonania pewnych ustępstw w zakresie dobrych praktyk programistycznych. Przykładem jest mapowanie danych dla wykresów po stronie klienta. Lepszym rozwiązaniem byłoby przetworzenie danych po stronie backendu i otrzymanie ich po stronie frontendu już w docelowej postaci. Innym przykładem, opartym na podobnej zasadzie, jest obliczanie statystyk użytkownika. Optymalizacji można również dokonać w algorytmie liczącym statystyki aktywności, tak aby uniknąć konieczności iteracji całej tablicy punktów śladu.

Powysze aspekty jednak nie mają aż tak negatywnego wpływu na działanie aplikacji, jakby mogło się wydawać. Współczesne urządzenia generalnie posiadają dużą moc obliczeniową, a same wykonywane w aplikacji obliczenia nie są bardzo złożone.

Z kolei wybór **Firebase**'a w zakresie implementacji uwierzytelniania, bazy danych i hostingu okazał się być dobrą decyzją. Gotowe rozwiązania znaczco uprościły i przyspieszyły pracę nad projektem. Należy jednak pamiętać, że darmowy plan użytkowania **Firebase** jest ograniczony. Po przekroczeniu dostępnych limitów, koszty naliczane są adekwatnie do zużycia zasobów.

Podsumowując powyższe wady i zalety zastosowanych rozwiązań, w przypadku ponownej realizacji projektu wybrano by nieco inne podejście: zaimplementowano by własną warstwę backend (np.: `Express.js`), która z kolei komunikowała by się z serwisami uwierzytelniania (np.: `auth0`) i bazy danych (np.: `MongoDB`). Ponadto, odpowiedzialność za dokonywanie obliczeń przeniesiono by z aplikacji klienta na serwer backendu.

W obecnym stanie aplikacja jest gotowa na dalszy rozwój. Dobrym kierunkiem jest ulepszenie edycji profilu, dodanie mapy cieplnej aktywności (ang. *Heat Map*) oraz dodanie możliwości planowania tras. W takim stanie, funkcjonalności aplikacji jako komputera rowerowego będą silnie ugruntowane. Większa ilość danych pozwoli na wprowadzenie dodatkowych statystyk oraz wykresów. Następnym krokiem, byłaby implementacja funkcji mediów społecznościowych, jak

np.: przeglądanie profili użytkowników, udostępnianie aktywności, system znajomych czy wydarzenia grupowe.

Literatura

- [1] How many phones are in the world. Dostęp 29.11.2021 [Online]. <https://www.bankmycell.com/blog/how-many-phones-are-in-the-world>.
- [2] Web app manifests. Dostęp 23.11.2021 [Online]. <https://developer.mozilla.org/en-US/docs/Web/Manifest>.
- [3] BeiDou Navigation Satellite System. Dostęp 09.11.2021 [Online]. <http://en.beidou.gov.cn/>.
- [4] Add Firebase to your JavaScript project. Dostęp 22.11.2021 [Online]. <https://firebase.google.com/docs/web/setup>.
- [5] Firebase Documentation. Dostęp 11.11.2021 [Online]. <https://firebase.google.com/docs>.
- [6] Get started with Firebase Hosting. Dostęp 22.11.2021 [Online]. <https://firebase.google.com/docs/hosting/quickstart>.
- [7] Understand Firebase projects: Firebase config files and objects. Dostęp 22.11.2021 [Online]. <https://firebase.google.com/docs/projects/learn-more#config-files-objects>.
- [8] Galileo GNSS. Dostęp 09.11.2021 [Online]. <https://galileognss.eu/>.
- [9] Garmin GPX schema extension. Dostęp 10.11.2021 [Online]. <https://www8.garmin.com/xmlschemas/GpxExtensionsv3.xsd>.
- [10] About GLONASS. Dostęp 09.11.2021 [Online]. https://www.glonass-iac.ru/en/about_glonass/.
- [11] The Global Positioning System. Dostęp 09.11.2021 [Online]. <https://www.gps.gov/systems/gps/>.
- [12] Havresine formula. Dostęp 25.11.2021 [Online]. <https://www.movable-type.co.uk/scripts/latlong.html>.
- [13] GPS constellation status. Dostęp 09.11.2021 [Online]. <https://www.navcen.uscg.gov/?Do=constellationStatus>.
- [14] React Context. Dostęp 24.11.2021 [Online]. <https://reactjs.org/docs/context.html>.
- [15] React Hooks. Dostęp 24.11.2021 [Online]. <https://reactjs.org/docs/hooks-intro.html>.
- [16] Getting Started. Dostęp 12.11.2021 [Online]. <https://reactjs.org/docs/getting-started.html>.

- [17] GPX 1.1 Schema Documentation. Dostęp 09.11.2021 [Online]. <https://www.topografix.com/GPX/1/1/>.
- [18] GPX: the GPS Exchange Format. Dostęp 09.11.2021 [Online]. <https://www.topografix.com/gpx.asp>.

Dodatek A

Instrukcja wdrożeniowa

Do uruchomienia aplikacji w trybie lokalnym wymagane jest środowisko Node.js w wersji v14.17.3 lub wyższej, które można pobrać pod adresem: <https://nodejs.org/>. Pliki źródłowe projektu dostępne są na repozytorium GitHub: <https://github.com/michaltkacz/biker> lub na załączonej do dokumentu płycie CD/DVD. W pierwszym kroku należy pobrać lub skopiować z płyty katalog z plikami źródłowymi do wybranej lokalizacji na dysku komputera. Następnie w wierszu poleceń należy przejść do głównego katalogu projektu i uruchomić polecenie `npm install`. Po ukończonej instalacji należy wywołać komendę `npm start`, co spowoduje komplikację i wyświetlenie strony w przeglądarce pod adresem <http://localhost:3000/>.

W zapisanej konfiguracji aplikacja klienta jest domyślnie zintegrowana z projektem **Firebase**. W celu połączenia z własnym backendem **Firebase**, należy postępować zgodnie z procedurą opisaną w rozdziale 3.

Dodatek B

Opis załączonej płyty CD/DVD

Na załączonej płycie CD/DVD zapisano katalog o nazwie `biker` z plikami projektu aplikacji. W celu konfiguracji środowiska uruchomieniowego, należy postępować według instrukcji zawartej w dodatku A. Strukturę katalogu przedstawiono w tabeli B.1.

W trakcie pracy nad projektem, w katalogu głównym generowane są dwa dodatkowe katalogi:

1. `.node_modules` — po zainstalowaniu bibliotek poleceniem `npm install`. Zawiera kody używanych bibliotek
2. `build` — po zbudowaniu aplikacji poleceniem `npm run build`. Zawiera kod zminifikowany i zoptymalizowany kod zbudowanej aplikacji. Służy do publikacji aplikacji w usłudze hostingu.

Oba te foldery ignorowane są przez `.gitignore`.

W katalogu `components` oprócz plików `App.tsx` i `App.less` znajdują się katalogi których nazwy odpowiadają nazwom poszczególnych komponentów. Nazwy tych katalogów oraz pliki stylów komponentów zapisano notacją `camelCase`, natomiast nazwy plików samych komponentów — notacją `PascalCase` (zgodnie ze standardem Reacta).

Tab. B.1: Struktura plików projektu

Katalog główny	Katalog src (główny kod aplikacji)
<pre> biker/ .firebase/ hosting.ID.cache public/ favicon.ico favicon.svg index.html logo.svg logo_maskable.svg manifest.json robots.txt src/ firebaserc .gitattributes .gitignore README.md craco.config.js firebase.json package-lock.json package.json package.json tsconfig.json </pre>	<pre> ../ src/ assets/ fonts/ Road_Rage.otf images/ bg.jpg brand.svg components/ componentName/ (37 katalogów) componentName.less ComponentName.tsx App.less App.tsx database/ schema.tsx firebase/ contexts/ AuthContext.tsx hooks/ useAuth.tsx useDatabase.tsx firebase.tsx firebaseUI.tsx global/ geolocationMath.ts gpxBuilder.ts pages.ts statisticsFromatters.ts geolocationMath.ts geolocationMath.ts hooks/ useActivityStatistics.tsx useGeolocation.tsx useInterval.tsx useNoSleep.tsx useTimeout.tsx useTracker.tsx index.tsx react-app-env.d.ts reportWebVitals.ts service-worker.ts serviceWorkerRegistration.ts </pre>