Dokumentacja systemu Freeturilo

Mikołaj Ryll, Mikołaj Terzyk

Promotor: dr Paweł Rzążewski

10.11.2021

Spis treści

1 Historia zmian

Tabela 1: Historia zmian

Data	**Autor**	**Opis**	**Wersja**
17.10.2021r.	Mikołaj Ryll, Mikołaj Terzyk	Pierwsza wersja dokumentacji	1.0
19.10.2021r.	Mikołaj Ryll, Mikołaj Terzyk	Poprawa schematu architektury, rozszerzenie przypadków użycia o obsługę błędów, dodanie hiperłączy do bibliografii	1.1
21.10.2021r.	Mikołaj Terzyk	Drobne poprawki stylistyczne	1.2
28.10.2021r.	Mikołaj Ryll, Mikołaj Terzyk	Poprawki zasugerowane przez dr Porter-Sobieraj	1.3
4.11.2021r.	Mikołaj Ryll	Architektura systemu, projekt modułów, interfejsy zewnętrzne, wybór technologii	2.0
4.11.2021r.	Mikołaj Terzyk	Komunikacja, struktura, stany i aktywność systemu, interfejs użytkownika	2.1
5.11.2021r.	Mikołaj Ryll	Rozszerzenie schematów architektury i komunikacji oraz opisu interfejsów zewnętrznych o Gmail API; opisanie klas i endpointów serwera.	2.2
6.11.2021r.	Mikołaj Terzyk	Poprawa diagramu klas, usunięcie diagramu stanów systemu, zmiany stylistyczne, dodanie konfiguracji powiadomień mailowych	2.3
10.11.2021r.	Mikołaj Ryll, Mikołaj Terzyk	Poprawki zasugerowane przez dr Porter-Sobieraj	2.4

2 Abstrakt

Dokument opisuje wstępne założenia projektu *Freeturilo*. Zawiera jego opis ogólny i specyfikację, a w szczególności cele biznesowe, wymagania funkcjonalne i niefunkcjonalne, analizę ryzyka, plan pracy i podział pracy nad przedstawianym rozwiązaniem.

3 Nazewnictwo

Administrator - twórca systemu Freeturilo, który ma dostęp do większego zakresu funkcjonalności niż użytkownicy.

Android - najpopularniejszy system operacyjny urządzeń mobilnych.

Aplikacja mobilna - ogólna nazwa oprogramowania uruchamianego na urządzeniach mobilnych, np. smartfonach czy tabletach.

Google Maps - platforma stworzona przez Google służąca do planowania tras pieszych, rowerowych, samochodowych, do pobierania zdjęć satelitarnych, interaktywnych panoram i do wielu innych funkcjonalności związanych z geolokalizacją.

NextBike - międzynarodowa firma oferująca rozwiązania uzupełniające komunikację publiczną. Warszawskie *Veturilo* należy do sieci *NextBike*, dzięki czemu możliwe jest pobieranie aktualnych danych o stacjach rowerowych w Warszawie za pośrednictwem interfejsu *NextBike'a*.

PostgreSQL - system zarządzania relacyjnymi bazami danych.

Serwer webowy - program działający na serwerze internetowym, który obsługuje zapytania protokołu komunikacyjnego HTTP.

Użytkownik - osoba wchodząca w bezpośrednią interakcję z aplikacją mobilną Freeturilo.

Veturilo - warszawski system rowerów miejskich.

4 Założenia biznesowe

Warszawski system rowerów miejskich *Veturilo* jest popularną alternatywą dla komunikacji publicznej. Istotną cechą regulaminu wynajmu rowerów jest to, że cena rośnie nieliniowo wraz z czasem wypożyczenia. W szczególności wynajem na czas krótszy niż 20 minut jest bezpłatny.

Aplikacja Freeturilo to aplikacja mobilna na system Android, która wspomaga planowanie trasy przejazdu rowerami *Veturilo*. System bierze pod uwagę aktualną dostępność rowerów na poszczególnych stacjach. Ponadto umożliwia wybór trasy według jednego z kryteriów:

- optymalizacja czasu,
- optymalizacja kosztu przejazdu,
- kryterium hybrydowe równoważące dwa powyższe.

Grupą docelową rozwiązania są mieszkańcy Warszawy, w szczególności osoby korzystające z komunikacji miejskiej. Aplikacja sprawia, że korzystanie z rowerów miejskich przynosi użytkownikom większą oszczędność czasu i pieniędzy.

5 Wymagania funkcjonalne

5.1 Historyjki użytkownika

W tabelach 2 i 3, w postaci historyjek użytkownika, przedstawione zostały funkcjonalności aplikacji istotne z jego punktu widzenia.

Jako użytkownik...

Tabela 2: Historyjki użytkownika aplikacji

l.p.	**chcę...**	**aby...**
1.1	chcę mieć wgląd do mapy	aby przeglądać stacje rowerowe.
1.2	chcę wyznaczać najszybszą trasę	aby oszczędzać czas.
1.3	chcę wyznaczać najtańszą trasę	aby oszczędzać pieniądze.
1.4	chcę wyznaczać optymalną trasę	aby godzić obie potrzeby.
1.5	chcę wyznaczać trasę przez wiele punktów	aby zatrzymywać się na krótkie przystanki.
1.6	chcę zapisywać ulubione miejsca	aby łatwiej wyznaczać do nich trasy.
1.7	chcę korzystać z historii wyszukiwanych tras	aby powtarzać wcześniejsze przejazdy.
1.8	chcę unikać niesprawnych stacji	aby bez przeszkód dojeżdżać do celu.
1.9	chcę zgłaszać niesprawne stacje	aby ułatwiać przejazd innym użytkownikom.

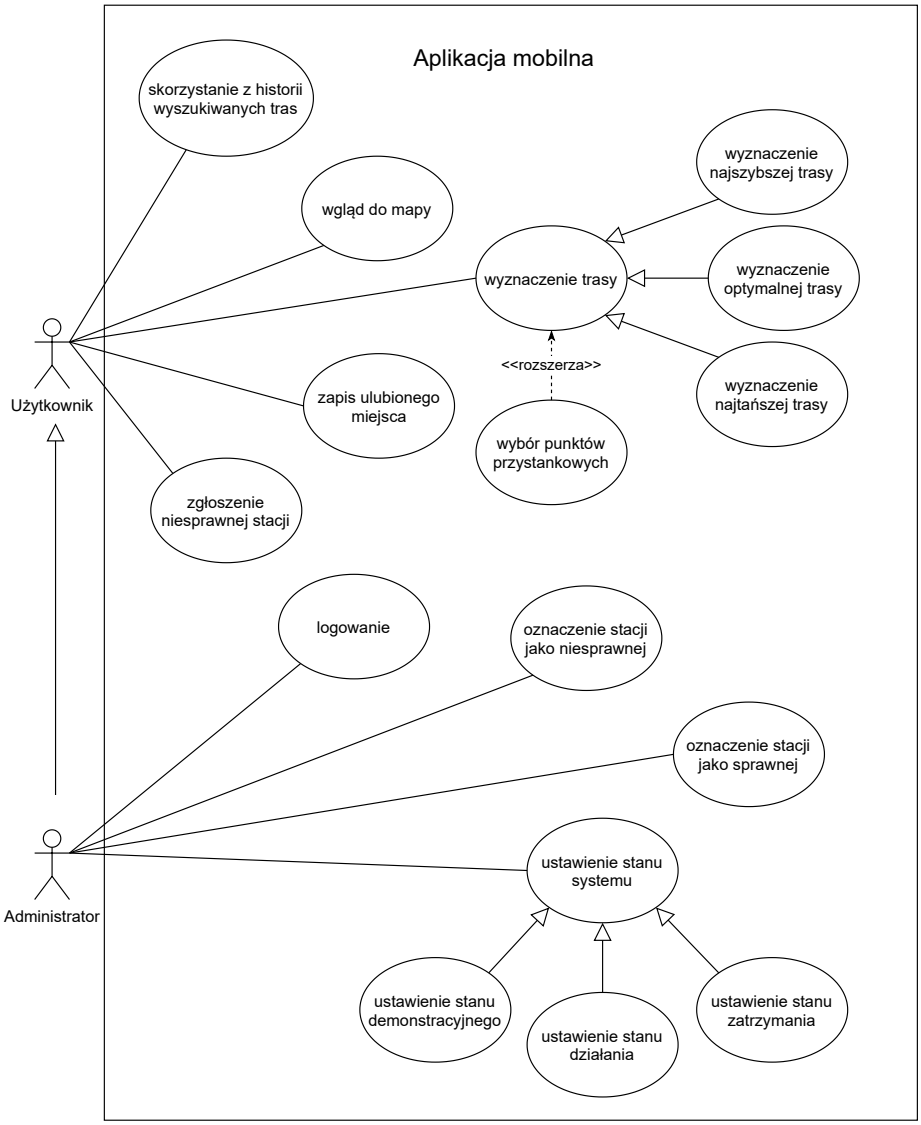
Jako administrator...

Tabela 3: Historyjki administratora aplikacji

l.p.	**chcę...**	**aby...**
2.1	chcę się logować	aby mieć dostęp do funkcji administratora.
2.2	chcę oznaczać stacje jako niesprawne	aby ułatwiać przejazd użytkownikom.
2.3	chcę usuwać oznaczenia o niesprawności stacji	aby ułatwiać przejazd użytkownikom.
2.4	chcę wprowadzać system w stan demonstracyjny	aby działał na danych testowych.
2.5	chcę wprowadzać system w stan zatrzymania	aby wyłączać go dla użytkowników.
2.6	chcę wprowadzać system w stan działania	aby zapewniać jego pełną funkcjonalność.

5.2 Przypadki użycia

W tej sekcji zaprezentowane zostały funkcjonalności systemu przy użyciu diagramu przypadków użycia (rys. 1) i tabel 4 i 5 je opisujących.



Rysunek 1: Diagram przypadków użycia

Tabela 4: Opis funkcjonalności użytkownika

Aktor	Przypadek użycia	Opis (Historyjka użytkownika)	Zachowanie aplikacji
Użytkownik	Wgląd do mapy	Przejście do widoku mapy z zaznaczonymi ulubionymi miejscami i stacjami rowerowymi. (1.1)	Pobranie danych o stacjach z bazy danych. Pobranie danych o ulubionych miejscach z pliku lokalnego. Wyświetlenie mapy ze znacznikami stacji i ulubionych miejsc.
	Wyznaczenie trasy	Wybór punktu startowego i końcowego (opcjonalnie też punktów przystankowych), wybór kryterium i zatwierdzenie. (1.2 - 1.5)	Pobranie wyliczonej przez serwer trasy i wyrysowanie jej na mapie wraz ze wszystkimi przystankami. Zapisanie parametrów wyznaczenia trasy w lokalnym pliku historii tras.
	Wyznaczenie najszybszej trasy	Wyznaczenie trasy z wyborem kryterium czasu trasy. (1.2)	Jak wyżej.
	Wyznaczenie najtańszej trasy	Wyznaczenie trasy z wyborem kryterium kosztu trasy. (1.3)	Jak wyżej.
	Wyznaczenie optymalnej trasy	Wyznaczenie trasy z wyborem kryterium hybrydowego. (1.4)	Jak wyżej.
	Zapis ulubionego miejsca	Wybór punktu w widoku mapy, nadanie nazwy, typu (np. dom) i zatwierdzenie. (1.6)	Zapisanie informacji o ulubionym miejscu w pliku lokalnym.
	Skorzystanie z historii wyszukiwanych tras	Przejście do widoku historii tras i wybór jednej z wypisanych tras. (1.7)	Pobranie historii tras z pliku lokalnego i wyświetlenie widoku historii tras. Po dokonaniu wyboru ponowne wyznaczenie wybranej trasy.
	Zgłoszenie niesprawnej stacji	Wybór stacji w widoku mapy, i zatwierdzenie zgłoszenia jej niesprawności. (1.8 - 1.9)	Przesłanie informacji o zgłoszeniu do serwera.

Tabela 5: Opis funkcjonalności admina

Aktor	Przypadek użycia	Opis (Historyjka użytkownika)	Zachowanie aplikacji
Administrator	Logowanie	Przejście do widoku logowania, wypełnienie pól tekstowych loginem oraz hasłem i zatwierdzenie. (2.1)	Pobranie informacji o rezultacie próby logowania i ewentualne udostępnienie funkcjonalności administratora.
	Oznaczenie stacji jako niesprawnej	Wybór stacji w widoku mapy, i zatwierdzenie oznaczenia jej jako niesprawnej. (2.2)	Przesłanie informacji o oznaczeniu do serwera.
	Oznaczenie stacji jako sprawnej	Wybór stacji w widoku mapy, i zatwierdzenie usunięcia jej oznaczenia jako niesprawnej. (2.3)	Przesłanie informacji o usunięciu oznaczenia do serwera.
	Ustawienie stanu systemu	Wybór stanu w widoku zarządzania stanem systemu i potwierdzenie ustawienia. (2.4 - 2.6)	Przesłanie informacji o ustawieniu stanu do serwera.
	Ustawienie stanu demonstracyjnego	Ustawienie stanu systemu na stan demonstracyjny. (2.4)	Jak wyżej.
	Ustawienie stanu zatrzymania	Ustawienie stanu systemu na stan zatrzymania. (2.5)	Jak wyżej.
	Ustawienie stanu działania	Ustawienie stanu systemu na stan działania. (2.6)	Jak wyżej.

5.3 Kluczowe funkcjonalności

W tabelach 6, 7 i 8 rozpisana została większa ilość szczegółowych informacji o najistotniejszych mechanizmach działania systemu.

Tabela 6: Przypadek użycia: wgląd do mapy

Nazwa	Wgląd do mapy
Historyjka użytkownika	1.1
Opis	Użytkownik może przejrzeć stacje rowerowe i dotyczące ich informacje, np. liczbę rowerów, liczbę stojaków i stan sprawności. Wyświetlane są też ulubione miejsca użytkownika. Z poziomu widoku mapy użytkownik może zgłosić niesprawną stację i zapisać ulubione miejsce.
Warunek wstępny	
Warunek końcowy	Wyświetlona zostaje mapa z naniesionymi znacznikami.
Potencjalne wyjątki	1. Brak połączenia aplikacji z internetem. 2. Niedostępność jednego z niezbędnych komponentów. 3. Niedostępność pliku z ulubionymi miejscami.
Aktorzy	Użytkownik
Wyzwalacz	Użytkownik chce zobaczyć mapę stacji rowerowych.
Standardowa procedura	(1) Użytkownik przechodzi do widoku mapy. (2) Aplikacja wysyła zapytanie do serwera. (3) System przygotowuje dane do zwrócenia. (4) Serwer zwraca dane o stacjach do aplikacji. (5) Aplikacja pobiera dane o ulubionych miejscach z pliku lokalnego. (6) Wyświetlona zostaje mapa z naniesionymi znacznikami.
Alternatywne procedury	(2) Aplikacja nie ma dostępu do internetu. (3) Wyświetlony zostaje widok z informacją o błędzie. (3) Jeden z niezbędnych komponentów jest niedostępny. (4) Serwer zwraca komunikat o błędzie do aplikacji. (5) Wyświetlony zostaje widok z informacją o błędzie. (5) Plik z ulubionymi miejscami jest niedostępny. (6) Wyświetlony zostaje chwilowo komunikat o błędzie. (7) Następuje kontynuacja procedury.

Tabela 7: Przypadek użycia: wyznaczanie trasy

Nazwa	Wyznaczenie trasy
Historyjka użytkownika	1.2 - 1.5
Opis	Użytkownik może wybrać punkt startowy, końcowy i przystanki przy pomocy adresu lub ulubionych miejsc. Dostępne są kryteria czasu, kosztu i hybrydowe. Trasa zostaje wyrysowana ze wszystkimi przystankami, włącznie z przesiadkowymi stacjami rowerowymi. Wraz z wyznaczeniem trasy system zwraca jej szczegóły: czas, koszt i długość.
Warunek wstępny	
Warunek końcowy	Wyświetlona zostaje mapa z wyrysowaną trasą.
Potencjalne wyjątki	1. Niedostępność pliku z ulubionymi miejscami. 2. Brak połączenia aplikacji z internetem. 3. Niedostępność jednego z niezbędnych komponentów. 3. Niedostępność pliku z historią tras.
Aktorzy	Użytkownik
Wyzwalacz	Użytkownik chce wyznaczyć trasę przejazdu.
Standardowa procedura	(1) Użytkownik przechodzi do widoku wyznaczania trasy. (2) Aplikacja pobiera dane o ulubionych miejscach z pliku lokalnego. (3) Użytkownik wybiera punkt startowy i końcowy trasy. (4) Użytkownik wybiera przystanki trasy. (opcjonalnie) (5) Użytkownik wybiera kryterium wyznaczania trasy. (6) Aplikacja wysyła zapytanie do serwera. (7) System wyznacza żądaną trasę. (8) Serwer zwraca wyznaczoną trasę do aplikacji. (9) Wyświetlona zostaje mapa z wyrysowaną trasą. (10) Parametry wyznaczenia trasy zostają zapisane w pliku lokalnym.
Alternatywne procedury	(2) Plik z ulubionymi miejscami jest niedostępny. (3) Wyświetlony zostaje chwilowo komunikat o błędzie. (4) Następuje kontynuacja procedury. (6) Aplikacja nie ma dostępu do internetu. (7) Wyświetlony zostaje widok z informacją o błędzie. (7) Jeden z niezbędnych komponentów jest niedostępny. (8) Serwer zwraca komunikat o błędzie do aplikacji. (9) Wyświetlony zostaje widok z informacją o błędzie. (10) Plik z historią tras jest niedostępny. (11) Wyświetlony zostaje chwilowo komunikat o błędzie.

Tabela 8: Przypadek użycia: ustawienie stanu demonstracyjnego

Nazwa	Ustawienie stanu demonstracyjnego
Historyjka użytkownika	2.4
Opis	W stanie demonstracyjnym system korzysta ze statycznych danych zapisanych w pamięci i jest niezależny od zewnętrznych komponentów. Daje to możliwość testowania i demonstracji aplikacji w kontrolowanych warunkach, nawet gdy system wypożyczania rowerów jest zamknięty.
Warunek wstępny	Administrator jest zalogowany.
Warunek końcowy	System zostaje wprowadzony w stan demonstracyjny.
Potencjalne wyjątki	1. Brak połączenia aplikacji z internetem. 2. Niedostępność jednego z niezbędnych komponentów.
Aktorzy	Administrator
Wyzwalacz	Administrator chce uniezależnić system od rzeczywistych danych na potrzeby demonstracji lub testów.
Standardowa procedura	(1) Administrator przechodzi do widoku zarządzania stanem systemu. (2) Administrator wybiera stan demonstracyjny. (3) Administrator potwierdza ustawienie stanu systemu. (4) Aplikacja wysyła zapytanie do serwera. (5) System przechodzi do stanu demonstracyjnego. (6) Serwer zwraca potwierdzenie ustawienia stanu. (7) Aplikacja potwierdza powodzenie operacji.
Alternatywne procedury	(4) Aplikacja nie ma dostępu do internetu. (5) Wyświetlony zostaje widok z informacją o błędzie. (5) Jeden z niezbędnych komponentów jest niedostępny. (6) Serwer zwraca komunikat o błędzie do aplikacji. (7) Wyświetlony zostaje widok z informacją o błędzie.

6 Wymagania niefunkcjonalne

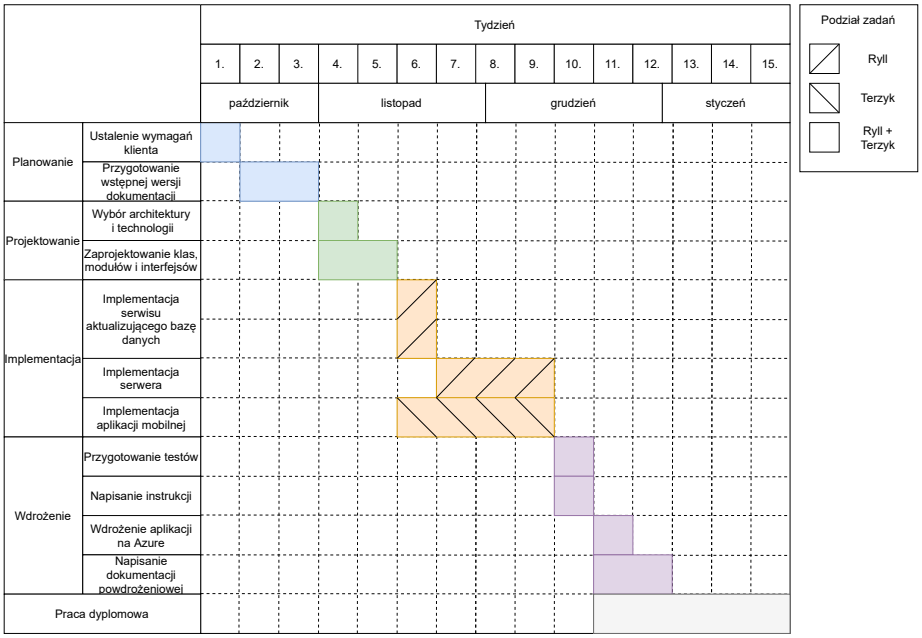
Wymagania niefunkcjonalne przedstawia tabela 9.

Tabela 9: Wymagania niefunkcjonalne

Obszar wymagań	**Opis wymagania**
Użyteczność	Wszystkie funkcjonalności aplikacji są dostępne dla użytkownika na ekranie urządzenia mobilnego o rozdzielczości co najmniej 1280x720 pikseli.
	System operuje na danych nie starszych niż 15 sekund.
	Aplikacja mobilna jest kompatybilna z Androidem 9 oraz każdym nowszym.
Niezawodność	Aplikacja jest dostępna cały czas z przerwami technicznymi, których czas nie przekracza dwóch godzin w tygodniu w godzinach nocnych.
	Moduły systemu są przygotowane na awarię innych modułów i w takim wypadku kontynuują pracę w sposób niezauważalny dla klienta.
Wydajność	Uruchomienie aplikacji wraz z pobraniem danych inicjalizujących nie trwa dłużej niż 5 sekund
	Serwer odpowiada na zapytanie o wyznaczenie trasy nie dłużej niż 3 sekundy.
Utrzymanie	Zachowanie wstecznej kompatybilności w kolejnych wersjach aplikacji oraz serwera.

7 Planowanie

Na projekt zostanie przeznaczone 15 tygodni. Rysunek 2 zawiera plan pracy.



Rysunek 2: Plan pracy z podziałem na tygodnie

Cały projekt został podzielony na kilka etapów. W niektórych tygodniach zaplanowany jest więcej niż jeden etap. Będzie to okres, kiedy każdy z nas będzie pracował równolegle nad swoją częścią projektu. O podziale obowiązków więcej w sekcji 8.

8 Podział obowiązków

Planowany podział obowiązków prezentuje się następująco:

I) Mikołaj Ryll

1. Przygotowanie dokumentacji rozwiązania
2. Zaprojektowanie logiki aplikacji
3. Implementacja oraz wdrożenie części backendowej
4. Integracja aplikacji z *Google Maps API*
5. Automatyczne testowanie aplikacji
6. Manualne testy działania aplikacji

II) Mikołaj Terzyk

1. Przygotowanie dokumentacji rozwiązania
2. Zaprojektowanie logiki aplikacji
3. Zaprojektowanie interfejsu graficznego
4. Implementacja oraz wdrożenie części frontendowej
5. Automatyczne testowanie interfejsu graficznego
6. Manualne testy działania aplikacji

9 Analiza ryzyka

Przygotowanie aplikacji będzie długotrwałym i złożonym procesem, dlatego istotna jest odpowiednia analiza ryzyka. Tabela 10 zawiera analizę SWOT projektu Freeturilo.

Tabela 10: Analiza ryzyka

	Zagrożenia	Szanse
Wewn.	1. Brak czasu w związku z innymi projektami 2. Brak odpowiedniej komunikacji z zespole	3. Nauka dokładności i terminowości 4. Wzajemna wymiana doświadczenia
Zewn.	5. Brak możliwości testowania w okresie zimowym 6. Zmiana lub zamknięcie API *NextBike'a*	7. Udoskonalenie systemu rowerów miejskich 8. Usatysfakcjonowanie klienta oraz zainteresowanie nowych klientów

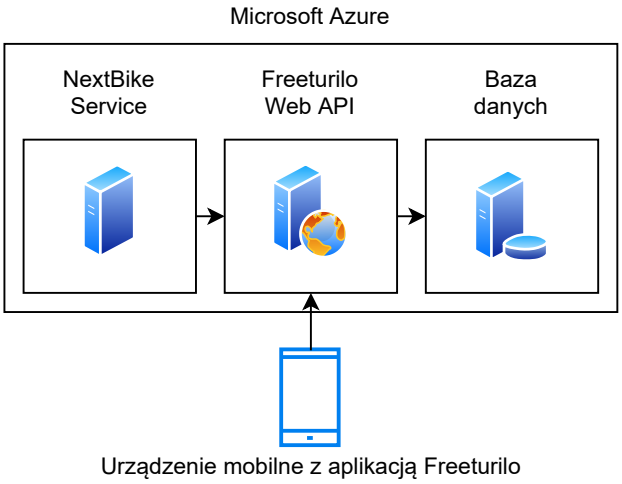
Realizacja projektu jest związana z zagrożeniami. Największym z nich jest zmiana publicznego interfejsu (6), który dostarcza informacji o lokalizacji rowerów. Spowodowałaby ona nieprawidłowe działanie aplikacji Freeturilo i zmusiłaby nas do natychmiastowej aktualizacji. Jednakże szansa wystąpienia jest niewielka.

Inaczej jest w przypadku braku możliwości testowania aplikacji w okresie zimowym (5), co nastąpi na pewno. Wśród funkcjonalności administratora znajdzie się wprowadzenie systemu w stan demonstracyjny, w którym korzysta on z danych testowych przygotowanych przez nas w okresie jesiennym. Aplikacja w takim stanie będzie mogła być bezproblemowo testowana, więc zabezpiecza nas to przed wspomnianym zagrożeniem.

Stworzenie aplikacji może również przynieść wiele korzyści. Pozwala przyczynić się do ulepszenia systemu rowerów miejskich (7) oraz zyskać potencjalnych klientów naszego produktu (8).

10 Architektura systemu

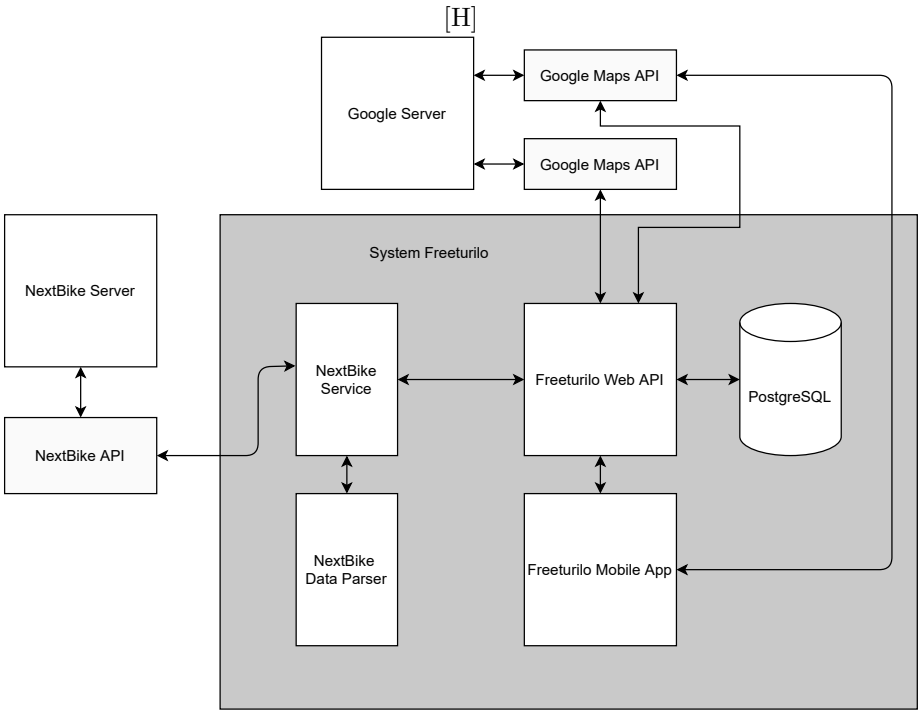
System składa się z czterech głównych modułów. Każdy moduł oprócz aplikajci mobilnej (o których szczegółowo w sekcji 11) jest oddzielnym serwisem wdrożonym na platformie chmurowej *Microsoft Azure*. Moduły są od siebie fizycznie niezależne, tzn. zachowują się tak, jakby działały na oddzielnych maszynach. Aplikacja mobilna jest zainstalowana na urządzeniu mobilnym z systemem Android i dostępem do internetu. Rysunek 3 przedstawia uproszczony model architektury systemu.



Rysunek 3: Architektura systemu

11 Projekt modułów

System składa się z czterech głównych modułów. Rysunek 4 przedstawia schemat architektury modułów Freeturilo.



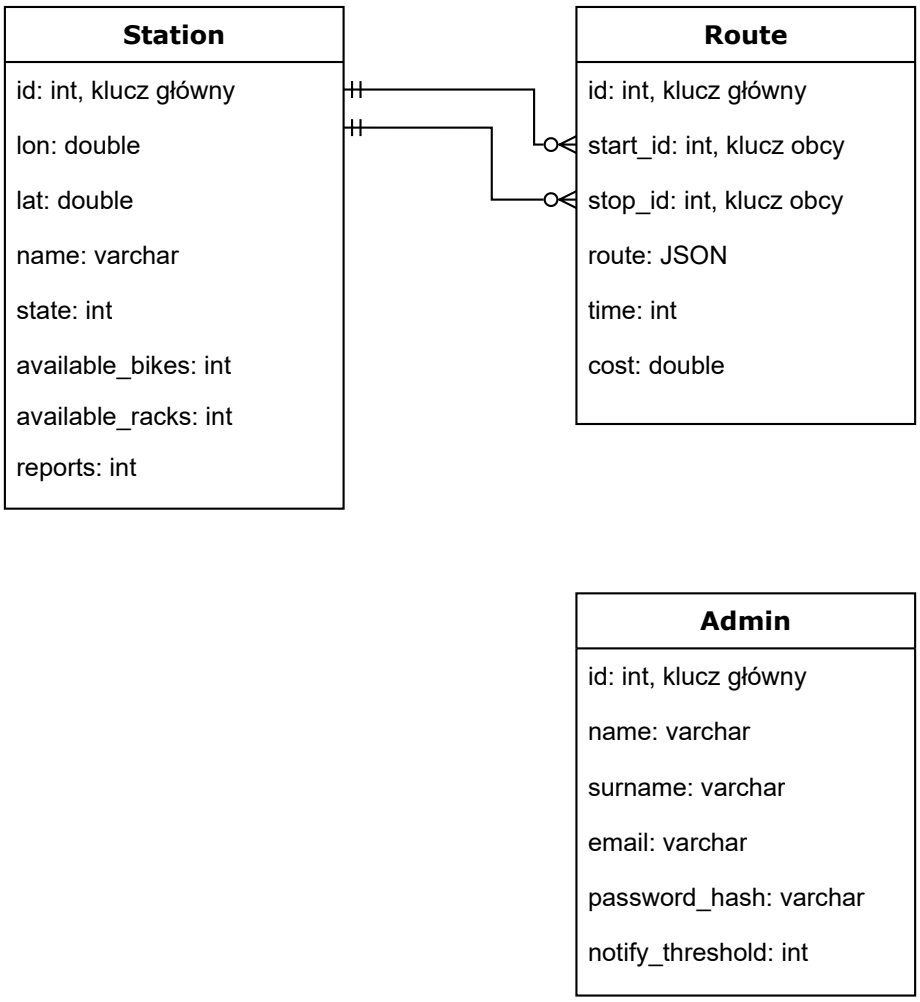
Rysunek 4: Schemat architektury systemu

11.1 Baza danych

Baza danych składa się z trzech tabel. Rysunek 5 przedstawia ich schemat UML. Tabela *Station* przechowuje informacje na temat stacji rowerowych: ich nazwy, współrzędne geograficzne, czy są aktywne i ile jest na nich aktualnie dostępnych rowerów.

Dla każdej uporządkowanej pary stacji istnieje rekord w tabeli *Route*, który zawiera informacje o stacji początkowej i końcowej, czasie i koszcie trasy między nimi oraz zserializowaną informację o samej trasie.

Tabela *Admin* służy do zapisywania danych o administratorach. Oprócz imienia, nazwiska i emaila przechowuje również hash hasła służący do uwierzytelnienia oraz liczbę zgłoszeń stacji potrzebną do powiadomienia administratora o jej potencjalnej niesprawności.



Rysunek 5: Schemat UML bazy danych

11.2 NextBike Service

NextBike Service to serwis odpowiedzialny za regularne odpytywanie interfejsu systemu *NextBike*. Przekazuje dane, na podstawie których serwer aktualizuje bazę danych. Moduł wykorzystuje bibliotekę *NextBike Data Parser*, która udostępnia publiczną statyczną klasę *Parser*.

```
public static class Parser
{
    public static markers ReadNextBikeData(string xmlContent);
}
```

ReadNextBikeData jest jedyną metodą publiczną tej klasy. Przyjmuje dane w formacie XML do deserializacji i zwraca obiekt klasy *markers*, który zawiera wszystkie informacje o stacjach. Klasa markers jest generowana automatycznie z pliku XSD i jest zdefiniowana tak, aby elementy drzewa XML były odpowiednio mapowane na pola.

Serwis będzie autoryzował się jako jeden z administratorów, dzięki czemu będzie miał dostęp do zabezpieczonych endpointów i będzie w stanie zmieniać zawartość bazy danych.

11.3 Freeturilo - serwer webowy

Serwer udostępnia publiczne API, poprzez które komunikują się z nim pozostałe moduły. Dane przesyłane w odpowiedzi na zapytania są wyznaczane na podstawie zawartości bazy danych, która przechowuje informacje o bieżącym stanie stacji rowerowych. Dokumentacja tego modułu została zapisana z użyciem specyfikacji OpenAPI 3.0 [8].

11.3.1 Bezpieczeństwo

Zmieniać stan aplikacji mogą jedynie administratorzy po wcześniejszym zalogowaniu się i otrzymaniu tokena autoryzacyjnego. Token powinien być wysyłany w nagłówku w polu o nazwie *api-key*.

```
securityScheme:
    ApiKeyAuth:
        type: apiKey
        in: header
        name: api-key
        description: "Autoryzacja tokenem JWT"
```

11.3.2 Obiekty transferu danych

Poniżej zdefiniowane są klasy, za pomocą których można komunikować się z serwerem.

```
schemas:
    Location:
        description: "Lokalizacja"
        type: object
        properties:
            name:
                type: string
            latitude:
                type: number
            longitude:
                type: number
```

Klasa *Location* zawiera niezbędne informacje (współrzędne geograficzne punktu) do wyznaczenia trasy.

```
Station:
    description: "Stacja rowerowa"
    type: object
    properties:
        id:
            type: number
        name:
            type: string
        latitude:
            type: number
        longitude:
            type: number
        bikeRacks:
            type: number
        bikes:
            type: number
        state:
            type: int
```

Klasa *Station* opisuje stację rowerową. Jest ona rozszerzeniem klasy *Location* o pola zawierające identyfikator stacji, informacje o ilości rowerów na stacji, ilości stojaków na stacji oraz stanie stacji. Pole *state* przyjmuje następujące wartości:

- 0, gdy stacja jest sprawna,
- 1, gdy stacja została zgłoszona jako niesprawna przez użytkownika,
- 2, gdy stacja jest niesprawna.

```
Favourite:
    description: "Ulubione miejsce"
    type: object
    properties:
        name:
            type: string
        latitude:
            type: number
        longitude:
            type: number
        type:
            type: number
```

Klasa *Favourite* również rozszerza klasę *Location*. Pozwala ona przechowywać informacje o ulubionych miejscach użytkownika. Pole *type* informuje o tym, jaki dokładnie jest typ ulubionego miejsca. Możliwe wartości to:

- 0 - dom,
- 1 - szkoła,
- 2 - praca,
- 3 - inny.

```
RouteParameters:
    description: "Parametry wyznaczanej trasy"
    type: object
    properties:
        start:
            type:
                \$ref: "#/components/schemas/Location"
        end:
            type:
                \$ref: "#/components/schemas/Location"
        stops:
            type: array
            items:
                \$ref: "#/components/schemas/Location"
        criterion:
            type: int
```

Objekt klasy *RouteParameters* zawiera wszystkie informacje potrzebne do wyznaczenia trasy. Zawiera punkt początkowy i końcowy, wszystkie przystanki użytkownika oraz kryterium wyznaczenia trasy. Możliwe wartości dla kryterium to:

a) 0, dla kryterium kosztu,

b) 1, dla kryterium czasu,

c) 2, dla kryterium hybrydowego.

```
Route:
    description: "Trasa rowerowa"
    type: object
    properties:
        parameters:
            type:
                \$ref: "#/components/schemas/RouteParameters"
        cost:
            type: number
        waypoints:
            type: array
            items:
                \$ref: "#/components/schemas/Location"
        directionsRoute:
            type:
                \$ref: "google/DirectionsRoute"
```

Klasa *Route* służy do przechowywania informacji o wyznaczonej trasie. Zawiera informacje o parametrach wyznaczenia trasy, punktach, przez które przechodzi trasa, koszcie trasy, a także pole zawierające pozostałe informacje o trasie, typu *DirectionsRoute*, który pochodzi z biblioteki *Google Maps*.

11.3.3 Dokumentacja API

Zgodnie z przyjętą praktyką, zapytania GET będą służyły do pobierania danych, typu POST do wysyłania danych oraz typu PUT do edycji danych. Wykorzystywane przez nasz system kody odpowiedzi HTTP oznaczają:

- 200 - powodzenie,
- 400 - niepowodzenie,
- 401 - operacja nieautoryzowana,
- 404 - brak danych,
- 500 - wewnętrzny błąd serwera,
- 503 - system zatrzymany.

Enpointy pod adresem `/station` służą do zarządzania stacjami. Zapytanie GET jest publiczne i pozwala pobrać kolekcję szczegółów wszystkich stacji. Zapytania POST i PUT są dostępne tylko dla administratorów. Pierwszy z nich pozwala dodać nową stację do bazy danych. Drugi endpoint wymaga jako argumentu wektora stacji i zamienia w bazie danych wszystkie rekordy na nowe.

```
paths:
    /station
        get:
            description: "Pobranie szczegółów wszystkich stacji"
            responses:
                "200":
                    content:
                        application/json:
                            schema:
                                type: array
                                items:
                                    \$ref:
                                    ↪  "#/components/schemas/Station"
                "500":
                    description: "Wystąpił wewnętrzny błąd
                    ↪  serwera"
                "503"
                    description: "System został zatrzymany"
```

```
post:
    description: "Dodanie stacji"
    security:
        ApiKeyAuth: [admin]
    requestBody:
        required: true
        content:
            application/json:
                schema:
                    \$ref: "#/components/schemas/Station"
    responses:
        "200":
            description: "Dodanie stacji powiodło się"
        "400":
            description: "Dodanie stacji nie powiodło
            ↪  się"
        "401":
            description: "Brak dostępu"
        "500":
            description: "Wystąpił wewnętrzny błąd
            ↪  serwera"

put:
    description: "Aktualizacja wszystkich stacji"
    security:
        ApiKeyAuth: [admin]
    requestBody:
        required: true
        content:
            application/json:
                schema:
                    type: array
                    items:
                        \$ref:
                        ↪  "#/components/schemas/Station"
    responses:
        "200":
            description: "Aktualizacja stacji powiodła
            ↪  się"
        "400":
            description: "Aktualizacja stacji nie
            ↪  powiodła się"
        "401":
            description: "Brak dostępu"
        "500":
```

```
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"
```

Endpointy pod adresem `/station/{stationId}` będą służyły do operacji na pojedynczych stacjach, gdzie argument w ścieżce oznacza ID stacji. Zapytanie GET jest publiczne i pozwala pobrać szczegóły wybranej stacji. Zapytanie PUT służy do edycji informacji o stacji i jest dostępne tylko dla administratora.

```
/station/{stationId}
    get:
        description: "Pobranie szczegółów stacji"
        parameters:
            in: path
            name: stationId
            schema:
                type: number
            required: true
        responses:
            "200":
                content:
                    application/json:
                        schema:
                            type:
                                \$ref:
                                ↪  "#/components/schemas/Station"
            "404":
                description: "Nie znaleziono stacji o podanym
                ↪  ID"
            "500":
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"
            "503"
                description: "System został zatrzymany"

    put:
        description: "Aktualizacja szczegółów stacji"
        security:
            ApiKeyAuth: [admin]
        parameters:
            in: path
            name: stationId
            schema:
                type: number
            required: true
        requestBody:
            required: true
            content:
```

```
            application/json:
                schema:
                    \$ref: "#/components/schemas/Station"
    responses:

        "200":
            description: "Aktualizacja szczegółów stacji
            ↪  powiodła się"
        "400":
            description: "Aktualizacja szczegółów stacji
            ↪  nie powiodła się"
        "401":
            description: "Brak dostępu"
        "404":
            description: "Nie znaleziono stacji o podanym
            ↪  ID"
        "500":
            description: "Wystąpił wewnętrzny błąd
            ↪  serwera"
```

Kolejne endpointy to:

- `/station/{stationId}/report`
- `/station/{stationId}/broken`
- `/station/{stationId}/working`

Pierwszy z nich jest publiczny i umożliwia każdemu użytkownikowi zgłoszenie niesprawności stacji. Pozostałe dwa dostępne są tylko dla administratorów i służą do oznaczania i usuwania oznaczenia stacji jako niesprawnej.

```
/station/{stationId}/report
   post:
       description: "Zgłoszenie stacji jako niesprawnej"
       parameters:
           in: path
           name: stationId
           schema:
               type: number
           required: true
       responses:
           "200":
               description: "Zgłoszenie stacji powiodło się"
           "404":
               description: "Nie znaleziono stacji o podanym
               ↪  ID"
           "500":
```

```
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"
            "503"
                description: "System został zatrzymany"

/station/{stationId}/broken
    post:
        description: "Oznaczenie stacji jako niesprawnej"
        security:
            ApiKeyAuth: [admin]
        parameters:
            in: path
            name: stationId
            schema:
                type: number
            required: true
        responses:
            "200":
                description: "Oznaczenie stacji powiodło się"
            "400":
                description: "Oznaczenie stacji nie powiodło
                ↪  się"
            "401":
                description: "Brak dostępu"
            "404":
                description: "Nie znaleziono stacji o podanym
                ↪  ID"
            "500":
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"

/station/{stationId}/working
    post:
        description: "Oznaczenie stacji jako sprawnej"
        security:
            ApiKeyAuth: [admin]
        parameters:
            in: path
            name: stationId
            schema:
                type: number
            required: true
        responses:
            "200":
                description: "Oznaczenie stacji powiodło się"
            "400":
```

```
            description: "Oznaczenie stacji nie powiodło
            ↪ się"
        "401":
            description: "Brak dostępu"
        "404":
            description: "Nie znaleziono stacji o podanym
            ↪ ID"
        "500":
            description: "Wystąpił wewnętrzny błąd
            ↪ serwera"
```

Pod adresem `/user` znajduje się endpoint odpowiadający tylko na zapytania typu POST. Służy on do autoryzacji administratorów systemu, dlatego wymaga podania w ciele zapytania adresu email oraz skojarzonego z tym adresem hasła. Zwrócony zostaje token, który uprawnia do używania zabezpieczonych endpointów.

```
/user
    post:
        description: "Logowanie administratora"
        requestBody:
            required: true
            content:
                application/json:
                    schema:
                        type: object
                        properties:
                            email:
                                type: string
                            password:
                                type: string
                        required:
                        - email
                        - password
        responses:
            "200":
                content:
                    application/json:
                        schema:
                            description: "Token JWT"
                            type: string
            "401":
                description: "Brak dostępu"
            "500":
                description: "Wystąpił wewnętrzny błąd
                ↪ serwera"
```

Endpoint pod adresem `/route` jest najważniejszym endpointem API. Jest on publiczny i służy do wyznaczania trasy rowerowej. Jako argument przyjmuje on objekt klasy *RouteParameters* i zwraca wyznaczony w oparciu o parametry objekt klasy *Route*.

```
/route
    get:
        description: "Wyznaczenie trasy"
        requestBody:
            required: true
            content:
                application/json:
                    schema:
                        type:
                            \$ref:
                            ↪  "#/components/schemas/RouteParameters"
        responses:
            "200":
                content:
                    application/json:
                        schema:
                            type:
                                \$ref:
                                ↪  "#/components/schemas/Route"
            "400":
                description: "Wyznaczenie trasy nie powiodło
                ↪  się"
            "500":
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"
            "503"
                description: "System został zatrzymany"
```

Kolejne trzy endpointy są dostępne tylko dla administratorów i służą do wprowadzania systemu w wybrany stan. Nazwa każdego z nich sugeruje działanie:

- `/app/stop` służy do zatrzymywania systemu,
- `/app/start` służy do wznawiania lub aktywacji systemu,
- `/app/demo` służy do wprowadzania systemu w stan demonstracyjny.

```
/app/stop
    post:
        description: "Wprowadzenie systemu w stan
        ↪  zatrzymania"
        security:
            ApiKeyAuth: [admin]
```

```
        responses:
            "200":
                description: "Zmiana stanu powiodła się"
            "401":
                description: "Brak dostępu"
            "500":
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"

/app/start
    post:
        description: "Wprowadzenie systemu w stan działania"
        security:
            ApiKeyAuth: [admin]
        responses:
            "200":
                description: "Zmiana stanu powiodła się"
            "401":
                description: "Brak dostępu"
            "500":
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"

/app/demo
    post:
        description: "Wprowadzenie systemu w stan
        ↪  demonstracyjny"
        security:
            ApiKeyAuth: [admin]
        responses:
            "200":
                description: "Zmiana stanu powiodła się"
            "401":
                description: "Brak dostępu"
            "500":
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"
```

Przy pomocy endpointu `/app/notify` administrator konfiguruje liczbę zgłoszeń stacji jako niesprawnej, przy której zostanie mailowo poinformowany o potencjalnej niesprawności stacji. Ustawienie wartości tej liczby na 0 wyłącza mailowe powiadamianie administratora.

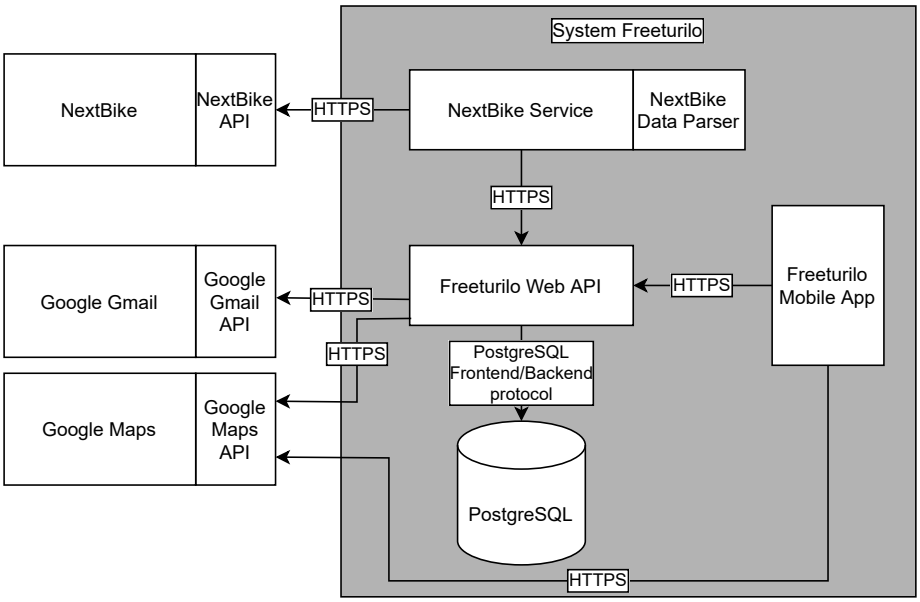
```
/app/notify/{number}
    post:
        description: "Ustawienie wartości liczby zgłoszeń
        ↪  wymaganej do powiadomienia administratora"
        security:
            ApiKeyAuth: [admin]
        parameters:
            in: path
            name: number
            schema:
                type: number
            required: true
        responses:

            "200":
                description: "Ustawienie powiodło się"
            "401":
                description: "Brak dostępu"
            "500":
                description: "Wystąpił wewnętrzny błąd
                ↪  serwera"
```

11.4 Freeturilo - aplikacja mobilna

Użytkownik wchodzi w interakcję z systemem używając aplikacji Freeturilo na swoim urządzeniu mobilnym. Szczegółowe opisy interfejsu użytkownika oraz klas wykorzystanych w tym module znajdują się w sekcjach 13 i 14.

12 Komunikacja

Rysunek 6 ukazuje schemat komunikujących się między sobą modułów.



Rysunek 6: Schemat komunikacji

12.1 Protokoły

W przypadku połączeń opierających się na protokole HTTPS, każde z nich nawiązane jest pomiędzy pewnym modułem, a API (zewnętrznym lub wewnętrznym). Komunikacja między nimi sterowana jest przez zapytania wysyłane przez dany moduł, na które zwracane są przez API odpowiedzi. Protokół HTTPS charakteryzuje się bezpieczeństwem ze względu na szyfrowanie danych.

W przypadku połączenia pomiędzy Freeturilo Web API a bazą danych PostgreSQL wykorzystywany jest protokół PostgreSQL Frontend/Backend. Charakteryzuje się on pojedynczym, wstępnym uwierzytelnieniem i komunikacją sterowaną przez wiadomości wysyłane przez klienta (w tym przypadku Web API).

12.2 Biblioteki

Bibliotekami służącymi do komunikacji między modułami są:

1. *System.Net.Http* [4] - biblioteka .NET na licencji MIT wykorzystywana przez NextBike Service i Freeturilo Web API do komunikacji przy pomocy protokołu HTTPS,
2. *java.net* - biblioteka języka Java na licencji OTN wykorzystywana przez aplikację mobilną Freeturilo do komunikacji przy pomocy protokołu HTTPS,

3. *Npgsql* - biblioteka .NET na licencji PostgreSQL wykorzystywana przez Freeturilo Web API do komunikacji z bazą danych przy pomocy protokołu PostgreSQL Frontend/Backend.

Wymieniona również powinna być przygotowana przez nasz zespół biblioteka *NextBike Data Parser* służąca do parsowania odpowiedzi na zapytania wysyłane przez NextBike Service do NextBike API. Jest ona niezbędna do poprawnej komunikacji między tymi modułami.

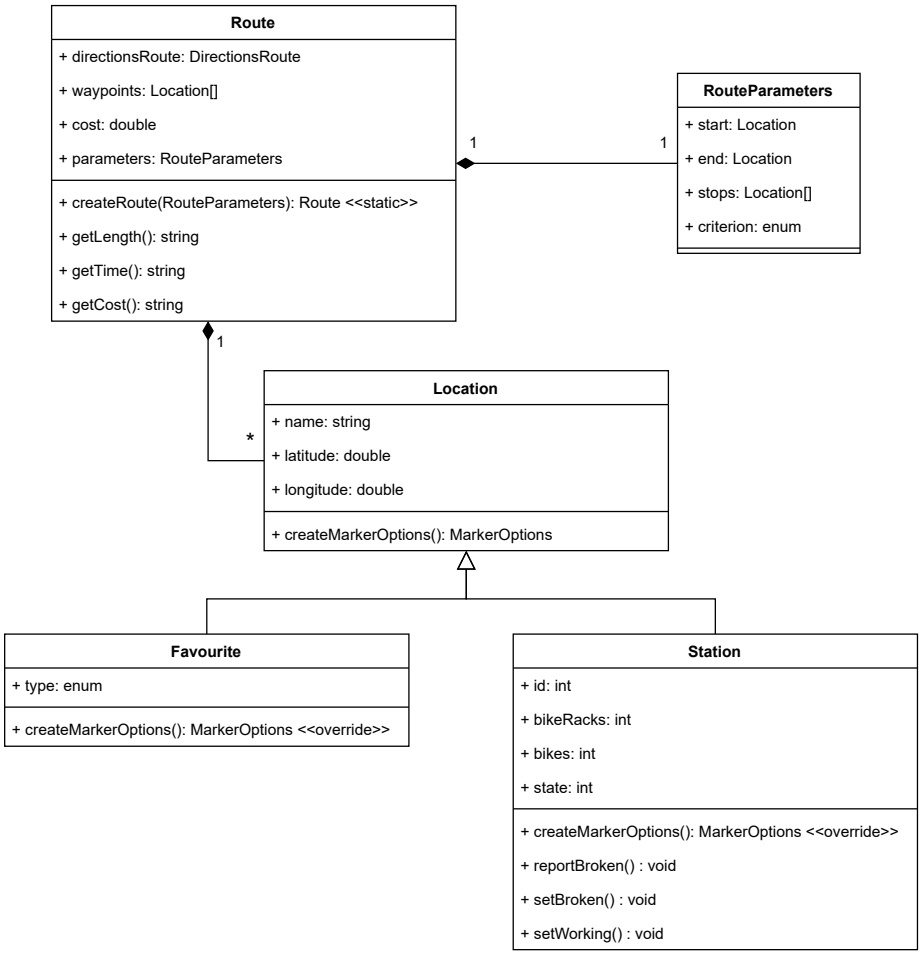
12.3 Konfiguracja sieci

Zarówno Google Maps API, Google Gmail API, NextBike API, jak i Freeturilo Web API mają nadany publiczny adres IP, dzięki czemu pozostałe moduły komunikują się z nimi przez sieć publiczną korzystając z nazwy domeny. Wszystkie trzy komponenty: Freeturilo Web API, Next Bike Service i baza danych PostgreSQL znajdują się we wspólnej sieci prywatnej. Jest to konfiguracja, istotnie przyspieszająca wymianę danych między tymi instancjami.

13 Struktura i stany systemu

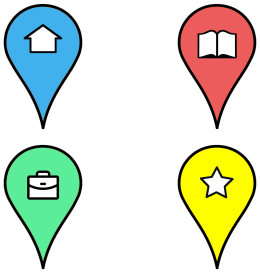
13.1 Klasy

Rysunek 7 przedstawia diagram podstawowych klas systemu Freeturilo.



Rysunek 7: Diagram klas

Location Klasy *Station* i *Favourite* dziedziczą po klasie *Location* i odpowiadają lokalizacjom stacji rowerowych i ulubionych miejsc. Mają one pola odpowiadające nazwie i współrzędnym geograficznym. Klasy te posiadają też metodę *createMarkerOptions()*, która generuje obiekt klasy *MarkerOptions* (zawartej w bibliotekach Google Maps). Klasa MarkerOptions odpowiada za dostosowywanie wyglądu znacznika lokalizacji na mapie.



Rysunek 8: Znaczniki ulubionych miejsc

Rysunek 9: Znaczniki stacji rowerowych

Favourite Klasa *Favourite* ma dodatkowe pole *type*, które przyjmuje jedną z wartości: *home*, *school*, *work*, *other*. W zależności od tej wartości instancje tej klasy charakteryzują się różnymi znacznikami (rys. 8). Pole *name* dla klasy *Favourite* przyjmuje wartość nadaną przez użytkownika.

Station Klasa *Station* ma dodatkowe pola odpowiadające szczegółom stacji: identyfikatorowi w systemie *NextBike*, liczbie stojaków i liczbie rowerów. Dodatkowo posiada pole *state*, którego wartość odpowiada aktualnej sprawności stacji. W zależności od tej wartości instancje tej klasy charakteryzują się różnymi znacznikami (rys. 9).

Metoda *reportBroken()* klasy *Station* służy do zgłaszania niesprawnej stacji przez użytkownika, a metody *setBroken()* i *setWorking()* do oznaczania i usuwania oznaczenia stacji jako niesprawnej przez administratora. Pole *name* dla tej klasy przyjmuje wartość zgodną z nazwą stacji w systemie *NextBike*.

Route Trasie wyznaczonej przez system odpowiada klasa *Route*. Polami tej klasy są:

1. *cost* - koszt trasy w złotówkach,
2. *waypoints* - wszystkie lokalizacje (*Location[]*), przez które przechodzi trasa (punkt początkowy, końcowy, przystanki i stacje rowerowe),
3. *parameters* - parametry wyznaczenia trasy: punkt początkowy, końcowy, przystanki i kryterium wyznaczenia trasy,
4. *directionsRoute* - trasa przechodząca przez *waypoints* wyznaczona przez Google Maps API. Jest instancją klasy *DirectionsRoute* (zawartej w bibliotekach Google Maps).

Klasa *Route* posiada następujące metody:

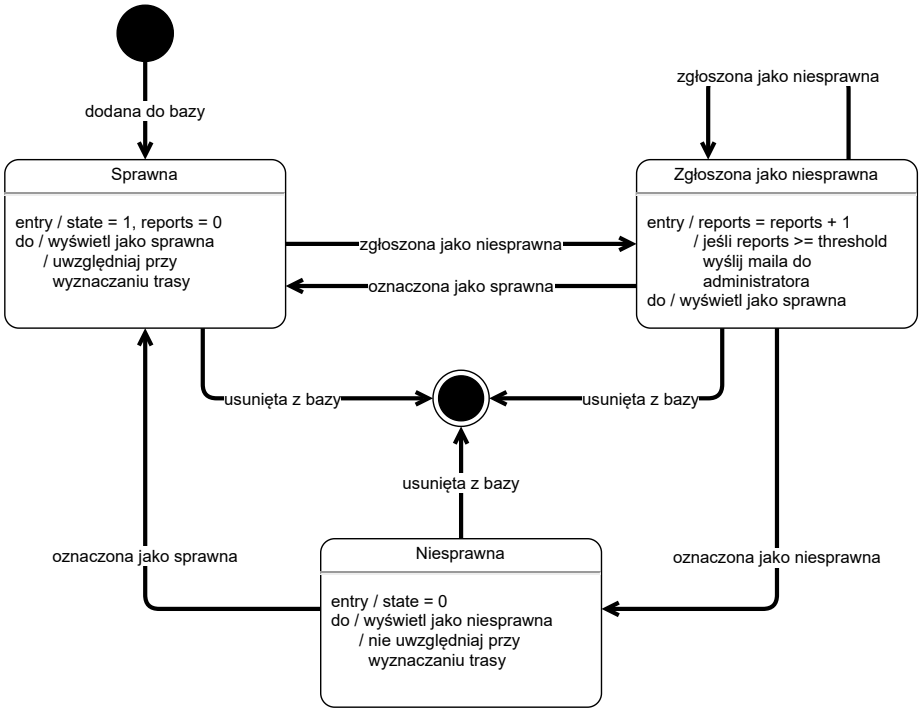
1. *createRoute(RouteParameters)* - metoda statyczna, której wynikiem jest trasa (*Route*) wyznaczona na podstawie argumentu typu *RouteParameters*,
2. *getLength()* - metoda obliczająca długość trasy na podstawie wartości pól obiektu *directionsRoute* i zwracająca napis - długość z jednostką.
3. *getTime()* - metoda obliczająca czas przejazdu trasy na podstawie wartości pól obiektu *directionsRoute* i zwracająca napis - czas z jednostką.
4. *getCost()* - metoda zwracająca wartość pola *cost* w postaci napisu z jednostką.

RouteParameters Na podstawie obiektu klasy *RouteParameters* wyznaczana jest trasa. Klasa ta jest strukturą o czterech polach:

1. start - punkt początkowy,
2. end - punkt końcowy,
3. stops - przystanki trasy,
4. criterion - kryterium wyznaczania trasy, które przyjmuje jedną z wartości: *cost*, *time*, *hybrid*.

13.2 Stany

Rysunek 10 przedstawia diagram stanów stacji rowerowej.

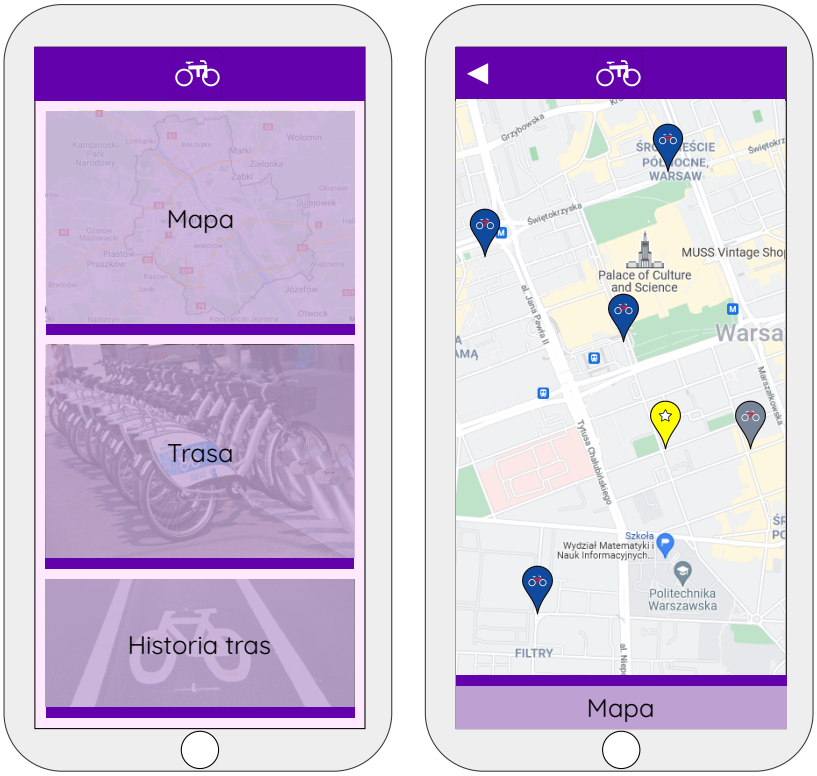


Rysunek 10: Diagram stanów stacji rowerowej

Stacja rowerowa może być zgłaszana przez użytkowników aplikacji jako niesprawna. Każde takie zgłoszenie odnotowywane jest w bazie danych i po przekroczeniu ustawionego dla administratora progu zgłoszeń zostaje on mailowo powiadomiony o potencjalnie niesprawnej stacji. Mechanizm wysyłania maili został opisany w rozdziale 15. Zadaniem administratora jest w takiej sytuacji oznaczenie stacji jako niesprawnej (co skutkuje omijaniem tej stacji przy wyznaczaniu trasy) lub wyzerowanie licznika zgłoszeń poprzez oznaczenie jej jako sprawnej.

14 Interfejs użytkownika

W tej sekcji wymienione są wszystkie widoki interfejsu użytkownika aplikacji Freeturilo.

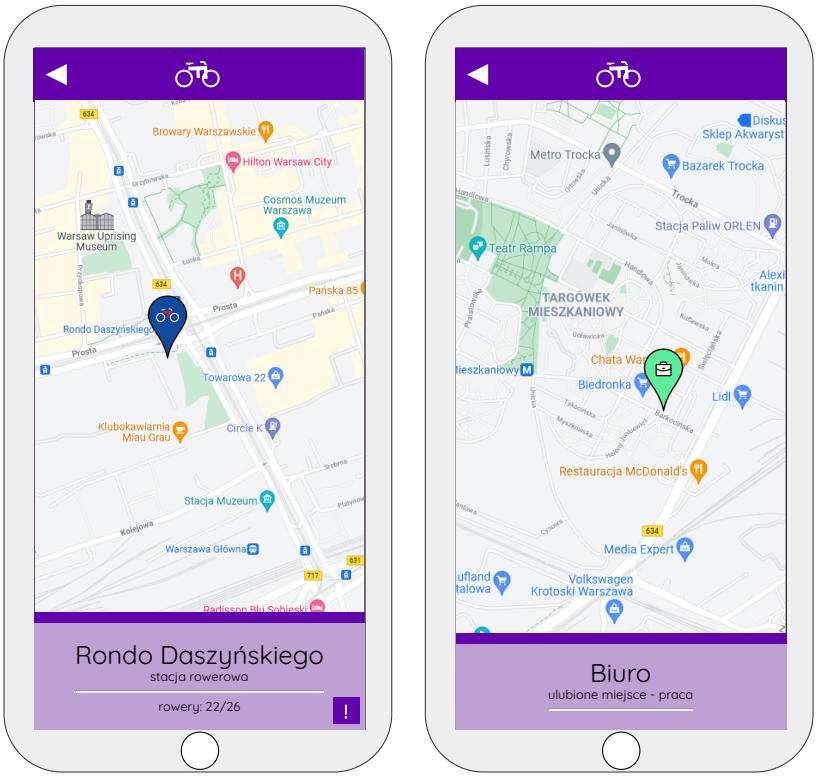


(a) Widok menu (b) Widok mapy

Rysunek 11

Widok menu W menu aplikacji (rys. 11a) dostępne są trzy przyciski. Przycisk *Mapa* przenosi użytkownika do widoku mapy, przycisk *Trasa* przekierowuje do widoku wyznaczenia trasy, a po wybraniu *Historii tras* wyświetlona zostaje historia tras wyznaczonych przez użytkownika. Przytrzymanie loga aplikacji przenosi użytkownika do widoku logowania.

Widok mapy Na mapie w widoku mapy (rys. 11b) wyświetlają się znaczniki stacji rowerowych i ulubionych miejsc użytkownika. Każdy znacznik stacji (rys. 12a) tak jak każdy znacznik ulubionego miejsca (rys. 12b) może zostać wybrany. Konsekwencją przytrzymania punktu na mapie jest pojawienie się okna dialogowego służącego utworzeniu nowego ulubionego miejsca. W lewym górnym rogu ekranu znajduje się przycisk powrotu do widoku menu.

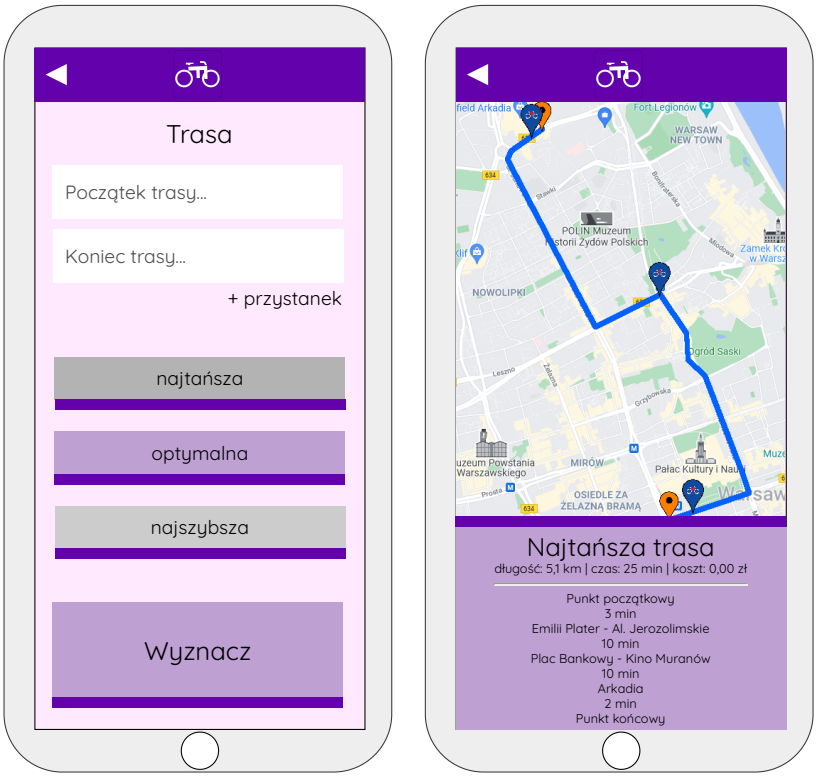


(a) Widok stacji (b) Widok ulubionego miejsca

Rysunek 12

Widok stacji Po wyborze znacznika stacji u dołu ekranu pojawiają się jej szczegóły: nazwa, aktualna liczba rowerów i liczba stojaków stacji (rys. 12a). W przypadku niesprawności znacznik stacji jest w kolorze szarym, a w szczegółach widnieje informacja: *stacja rowerowa (niesprawna)*. Oprócz powyższych u dołu ekranu widnieje też przycisk zgłoszenia niesprawności stacji. Po jego wybraniu wyświetlone na środku ekranu zostaje okno dialogowe w celu zatwierdzenia zgłoszenia o niesprawności. Wybór przycisku powrotu (w lewym górnym rogu ekranu) przywraca widok mapy bez zaznaczonej stacji.

Widok ulubionego miejsca Po wyborze znacznika ulubionego miejsca, podobnie jak w przypadku znacznika stacji, na ekranie pojawiają się szczegóły miejsca: nazwa i typ. Wybór przycisku powrotu (w lewym górnym rogu ekranu) przywraca widok mapy bez zaznaczonego ulubionego miejsca.



(a) Widok wyznaczania trasy (b) Widok trasy

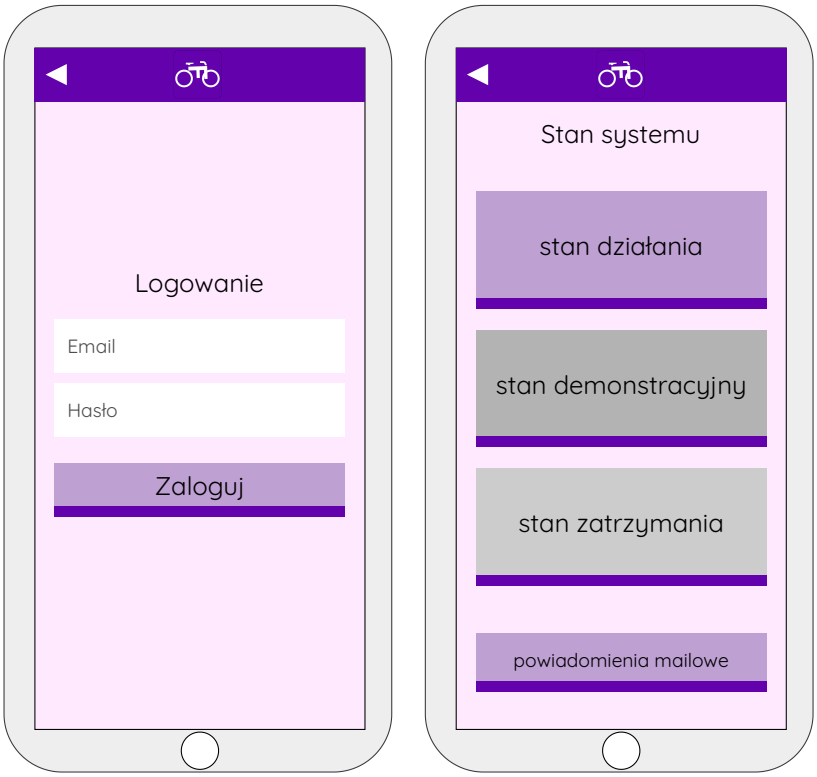
Rysunek 13

Widok wyznaczania trasy W widoku wyznaczania trasy (rys. 13a) użytkownik wybiera jej parametry: początek, koniec, przystanki i kryterium wyznaczania. Po rozpoczęciu pisania w dowolnym z pól tekstowych wyświetlają się pod nim sugestie na podstawie wpisanej wartości. Wśród sugestii znajdują się ulubione miejsca użytkownika i inne lokalizacje. Po wyborze *+ przystanek* pomiędzy polami tekstowymi odpowiadającymi początkowi i końcowi trasy pojawia się dodatkowe pole tekstowe. Wybór kryterium wyznaczania trasy opiera się na użyciu jednego z trzech przycisków odpowiadających kryteriom. Przycisk *Wyznacz* przenosi użytkownika do widoku trasy (rys. 13b). W lewym górnym rogu ekranu znajduje się przycisk powrotu do widoku menu.

Widok trasy Aplikacja prezentuje wyznaczoną trasę w widoku trasy (rys. 13b). Na ekranie ukazana jest mapa wraz z zaznaczoną trasą, w tym jej początkiem, końcem, przystankami (znacznikami czerwonymi) i stacjami przesiadkowymi (znacznikami niebieskimi). U dołu widoku wypisane są szczegóły trasy, a więc jej długość, czas, koszt i dokładny przebieg. W lewym górnym rogu ekranu znajduje się przycisk powrotu do widoku wyznaczania trasy.

Rysunek 14: Widok historii tras

Widok historii tras W widoku historii tras (rys. 14) wypisane są wszystkie parametry wyznaczonych przez użytkownika tras. Każdy z wpisów w tym widoku odpowiada punktowi startowemu, końcowemu, przystankom i kryterium wyznaczenia pewnej trasy wyznaczonej w przeszłości. Wybór jednego z wpisów przenosi użytkownika do widoku trasy (rys. 13b), w którym wyświetlona zostaje wyznaczona ponownie trasa o tych samych parametrach. W lewym górnym rogu ekranu znajduje się przycisk powrotu do widoku menu.

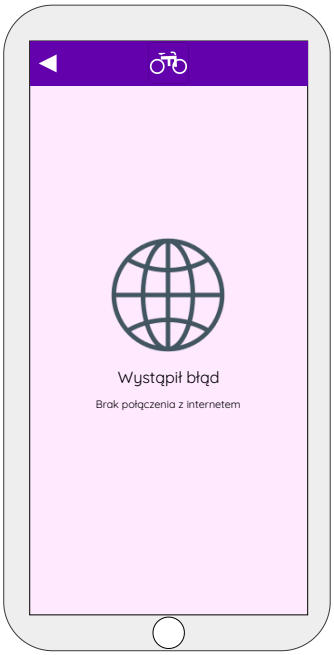


(a) Widok logowania (b) Widok zarz. stan. systemu

Rysunek 15

Widok logowania Widok logowania (rys. 15a), do którego dostęp użytkownik uzyskuje przytrzymując logo aplikacji wykorzystywany jest jedynie przez administratorów aplikacji. Administrator wpisuje swój email i hasło w polach tekstowych, a następnie wybiera *Zaloguj*. Po poprawnym zalogowaniu się administratora aplikacja wyświetla widok zarządzania stanem systemu (rys. 15b). W lewym górnym rogu ekranu znajduje się przycisk powrotu do widoku menu.

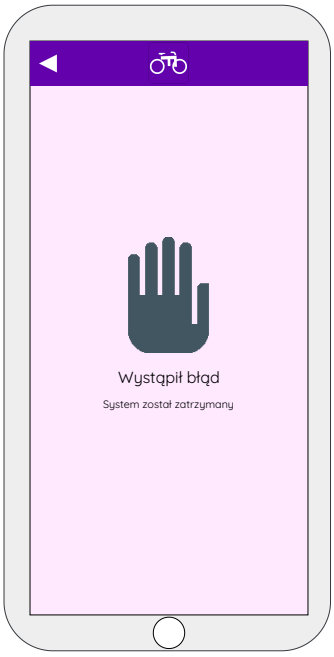
Widok zarządzania stanem systemu Do widoku zarządzania stanem systemu (rys. 15b) zalogowany administrator uzyskuje dostęp przytrzymując logo aplikacji. Po wyborze jednego z trzech przycisków odpowiadających stanom systemu aplikacja otwiera okno dialogowe służące zatwierdzeniu zmiany stanu systemu. U dołu widoku znajduje się przycisk służący do konfiguracji powiadomień mailowych o potencjalnej niesprawności stacji. Po wybraniu go pojawia się okno dialogowe pozwalające na dobór liczby zgłoszeń skutkującej wysłaniem powiadomienia (lub wyłączenie powiadomień mailowych). W lewym górnym rogu ekranu znajduje się przycisk powrotu do widoku menu.



(a) Widok błędu (internet)

(b) Widok błędu (serwer)

(c) Widok błędu (logowanie)



(d) Widok błędu (zatrzymany)

Rysunek 16

Widoki błędu W trakcie korzystania z aplikacji użytkownik może napotkać błędy, którym odpowiadają przedstawione na rysunkach widoki (rys. 16a, 16b, 16c, 16d). Dla każdego z tych widoków w lewym górnym rogu ekranu znajduje się przycisk powrotu do widoku menu, a na środku wyświetlona jest grafika odpowiadająca typowi błędu i jego opis.

15 Interfejsy zewnętrzne

System komunikuje się z zewnętrznymi serwerami poprzez udostępnione przez nie interfejsy.

15.1 NextBike API

W przypadku serwera *NextBike* będziemy korzystać z tylko jednego endpointu zwracającego kolekcję zawierającą informacje o wszystkich stacjach rowerowych w Warszawie.

```
https://nextbike.net/maps/nextbike-live.xml?city=210:
   get:
     description: "Pobranie listy wszystkich stacji
       rowerowych w Warszawie"
     responses:
       "200":
         description: "Zwrócono listę wszystkich stacji
           rowerowych w Warszawie"
         content:
           application/xml:
             schema:
               type: "markers"
```

Nazwa typu zwracanych danych *markers* jest wymuszona przez odpowiedź XML serwera (korzeń struktury XML ma właśnie taką nazwę). Klasa implementująca typ *markers* będzie generowana automatycznie na podstawie pliku XML Schema Design, który znajdzie się w bibliotece do deserializacji.

15.2 Google Maps API

Korzystanie z interfejsu *Google Maps* wymaga podania klucza. W tym celu założyliśmy konto mailowe (o adresie `freeturilo@gmail.com`) dla naszego projektu i wygenerowaliśmy niezbędny klucz. Wśród udostępnionych przez ten interfejs funkcjonalności, z których korzystamy, znajdują się m. in.:

1. wyznaczanie macierzy tras między punktami startowymi a punktami końcowymi [9],
2. pobieranie mapy Warszawy [10],

3. mapowanie adresów na współrzędne geograficzne [11].

Pełna dokumentacja endpointów interfejsu znajduje się w [2].

15.3 Google Gmail API

API zostanie wykorzystane do funkcjonalności powiadamiania administratorów o zgłaszanych usterkach. Serwer po odpowiedniej liczbie zgłoszeń będzie wysyłał wiadomość email z adresu `freeturilo@gmail.com` do wszystkich administratorów z informacją o możliwej usterce stacji. Wysyłanie maila będzie realizowane za pomocą odpowiedniego enpointu z interfejsu *Google Gmail*.

Pełna dokumentacja API znajduje się w [3].

16 Wybór technologii

NextBike Data Parser Biblioteka *NextBike Data Parser* zostanie napisana w języku C#. Będzie ona wykorzystywać *xsd.exe* - narzędzie służące do definicji schematu XML, które umożliwi deserializację danych pobieranych z serwera *NextBike*.

NextBike Service Serwis do monitorowania stanu stacji rowerowych również będzie napisany w języku C#. Wykorzystuje powyższą bibliotekę oraz korzysta z wbudowanych narzędzi dotnetowych do komunikacji HTTP.

Baza danych Planujemy, aby system korzystał z relacyjnej bazy danych, która będzie obsługiwana przez system PostgreSQL.

Freeturilo Web API Serwer, spójnie z pozostałymi modułami backendowymi, zostanie napisany w języku C#. Do stworzenia interfejsu REST zostanie użyty framework *Entity Framework* oraz wzorzec MVC. Wszystkie moduły backendowe (poza bazą danych) będą wykorzystywać framework *.NET 5.0*.

Freeturilo Mobile App Aplikacje mobilna zostanie napisana w języku Java 11 i będzie korzystała z bibliotek dostarczanych przez *Maps SDK for Android* od *Google*.

Literatura

[1] Dokumentacja systemu Android
https://developer.android.com/

[2] Dokumentacja API Google Maps
https://developers.google.com/maps/documentation

[3] Dokumentacja API Google Gmail
https://developers.google.com/gmail/api/

[4] Dokumentacja biblioteki System.Net.Http
https://docs.microsoft.com/en-us/dotnet/api/system.net.http

[5] Dokumentacja biblioteki java.net
https://developer.android.com/reference/java/net/package-summary

[6] Dokumentacja biblioteki Npgsql
https://www.npgsql.org/doc/index.html

[7] Dokumentacja protokołu Frontend/Backend PostgreSQL
https://www.postgresql.org/docs/9.3/protocol.html

[8] Dokumentacja specyfikacji OpenAPI 3.0
https://swagger.io/specification/

[9] Dokumentacja API macierzy odległości Google Maps
https://developers.google.com/maps/documentation/distance-matrix/overview

[10] Dokumentacja API map statycznych Google Maps
https://developers.google.com/maps/documentation/maps-static/overview

[11] Dokumentacja API geokodowania Google Maps
https://developers.google.com/maps/documentation/geocoding/overview