

Podziękowania

Pragnę podziękować mojemu Promotorowi za wsparcie merytoryczne i motywację, a także najbliższym mi osobom, za doping, wiarę w moje możliwości oraz wyrozumiałość, kiedy nie mogłem poświęcać Im tyle czasu, ile bym chciał.

Dziękuję również wszystkim zainteresowanym osobom, które wypytywały mnie o szczegóły techniczne i postępy tej pracy. Dzięki Wam mogłem lepiej zrozumieć mój projekt.

Spis treści

Podziękowania	i
1 Wstęp	1
1.1 Idea pracy dyplomowej	1
1.2 Założenia projektu	1
1.3 Cel pracy	1
1.4 Zakres pracy	2
2 Wykorzystywane technologie	3
2.1 JavaScript	3
2.2 Node.js	4
2.3 Socket.IO	4
2.4 MongoDB	5
2.5 PhoneGap	5
3 Architektura systemu	7
3.1 Aplikacja serwerowa	7
3.2 Model bazy danych	10
3.3 Panel zarządzania	11
3.4 Klient mobilny	12
3.5 Komunikacja Klient-Serwer - API	13
4 Uruchomienie aplikacji	15
4.1 Aplikacja serwerowa	15
4.2 Aplikacja mobilna na platformę Android	16
5 Panel zarządzania - konfiguracja	19
5.1 Zarządzanie listą placów manewrowych	20
5.2 Zarządzanie listą instruktorów	22

6	Klient dla systemu Android - korzystanie	25
6.1	Przykładowy przepływ	25
6.2	Pogląd na listę placów	25
6.3	Zajmowanie i zwalnianie placu	25
7	Wnioski	33
8	Do usunięcia	35
8.1	Algorytmy	35
8.2	Tymczasowa zmiana rozmiaru strony	37
8.3	Wpisy bibliograficzne	37

Rozdział 1

Wstęp

1.1 Idea pracy dyplomowej

Niniejsza praca opisuje projekt i wykonanie mobilnego systemu komunikacji przeznaczonego dla Ośrodków Szkolenia Kierowców roboczo nazwanego „OSK-Helper”. Jego architekturę, część wizualną interfejsów, sposób instalacji i uruchomienia, zalecane przypadki użycia oraz opracowane lub wykorzystane algorytmy.

1.2 Założenia projektu

Głównym założeniem projektu było stworzenie prostej aplikacji mobilnej, która będzie przyjazna dla użytkowników, a jednocześnie nadal będzie spełniała swoje zadanie - ułatwiała komunikacji pomiędzy instruktorami Ośrodków Szkolenia Kierowców w celu zmniejszenia kolizji zajętości placów manewrowych.

Kolejnym założeniem było wykorzystanie języka JavaScript w każdej płaszczyźnie aplikacji, zarówno klienta, jak i serwera oraz użycie nierelacyjnej bazy danych, która będzie przechowywała dane o strukturze zbliżonej do struktury obiektów tego języka.

1.3 Cel pracy

Celem pracy jest projekt i realizacja systemu mającego za zadanie ułatwić pracę instruktorów Prawa Jazdy poprzez zautomatyzowanie wymiany informacji dotyczącej zajętości placów manewrowych, z których mogą korzystać.

1.4 Zakres pracy

Projekt praktyczny, stworzony dla celów niniejszej pracy jest znaczną częścią trudu włożonego w jej powstanie i w jego skład wchodzi:

- aplikacja mobilna dla systemu Android
- aplikacja serwerowa będąca zapleczem dla mobilnego systemu.

Rozdział 2

Wykorzystywane technologie

2.1 JavaScript

JavaScript jest prototypowym obiektowym językiem programowania. Jest to język Internetu, ze względu na to, że jego kod potrafi uruchomić prawie każda przeglądarka internetowa. Po ciężkim okresie, kiedy to został ogólnie przyjęty jako zabawka umożliwiająca tworzenie tzw. wodotrysków na stronach WWW, rozrósł znacząco i dzisiaj jest jednym z najpopularniejszych, a na pewno najsłynniejszym spośród internetowych języków programowania.

Dzisiejszą „Dobę Internetu” pełną technologii AJAX, rozbudowanych klienckich aplikacji internetowych przypominających desktopowe m.in. czytników RSS, aplikacji biurowych, klientów pocztowych, interaktywnych map, programów do obróbki grafiki i wideo etc. do historii odeszły długotrwałe wywołania stron wykonywane przy każdym wejściu użytkownika w interakcję z przeglądarką. To wszystko możemy zawdzięczać głównie dzięki JavaScriptowi.

Początkowe wersje interpreterów JavaScript zostały osadzone w jednych z pierwszych wersji przeglądarek Netscape i Internet Explorer. Ta kliencka wersja JavaScriptu pozwala na umieszczanie wykonywalnej zawartości w stronach internetowych, co pozwala na budowanie interakcji z użytkownikiem, to coś więcej niż statyczne strony WWW. Skrypty języka JavaScript odpowiedzialne są za część zachowawczą stron, kiedy to HTML (ang. HyperText Markup Language) opisuje zawartość i strukturę DOM (ang. Document Object Model) oraz treść, a CSS (ang. Cascading Style Sheets) opisuje wygląd tych elementów.

JavaScript jest otwartym standardem, ale standaryzacją tego języka zajmuje się ECMA International (ang. European Computer Manufacturers Association), która na jego podstawie wydała standard języka skryptowego o nazwie ECMAScript, którego już niedługo szósta wersja wprowadzić ma nowe mechanizmy obiektowości, takie jak np. pełnowymiarowe dziedziczenie za pomocą klas (które dzisiaj jest realizowane przez prototypowanie obiektów), modularyzację i hermetyzację (które dzisiaj są realizowane przez zewnętrzne biblioteki wprowadzające tego typu abstrakcje), a

także wiele innych usprawnień, które zapewne przyciągną jeszcze większe rzesze programistów na całym świecie.

Obecnie przeglądarki WWW to nie jedyne interpretery w obrębie których możliwe jest uruchomienie kodu JavaScript. Można tworzyć kod do zastosowań po stronie serwera WWW (m.in. .NET lub Node.js), skrypty i aplikacje wiersza poleceń CLI (ang. Command Line Interface), aplikacje desktopowe (Node-webkit, Alchemium), aplikacje dla urządzeń przenośnych (m.in. Sencha Touch, PhoneGap), aplikacje dla Smart TV (m.in. Mautilus), sterowanie mikrokomputerami takimi jak Raspberry PI (np. pijs.io) i Arduino (np. Johnny-Five, noduino), systemy operacyjne (Firefox OS, Chrome OS) rozszerzenia aplikacji takich jak Adobe Photoshop, Google Chrome, Mozilla Firefox.

2.2 Node.js

Node.js jest stosunkowo nową platformą deweloperską stworzoną przez Ryana Dahla, pozwalającą programistom JavaScript na tworzenie kodu serwerowego o wysokiej wydajności.

Dokładniej rzecz ujmując jest to silnik Google V8 z projektu Chromium (Google Chrome) opakowany w taki sposób, aby można było go wykorzystywać jako niezależny interpreter z możliwościami podobnymi do np. Ruby, Pythona, PHP itd.

Asynchroniczna, bazująca na zdarzeniach charakterystyka języka JavaScript sprawia, że Node.js jest bardzo wydajny, a ze względu na jego uniwersalność i wysoki poziom abstrakcji m.in. dynamiczne typowanie zmiennych, przyjazną składnię wbudowanych instrukcji i złożonych obiektów JSON (ang. JavaScript Object Notation), pisanie w nim to czysta przyjemność.

Podobnie jak większość dystrybucji Linuksa lub języki programowania takie jak Ruby, Python czy PHP, Node.js posiada swój manager pakietów NPM (ang. Node Package Manager) agregujący ogromną ilość pakietów rozszerzających jego możliwości.

Kolejnym atutem jest możliwość rozwijania kodu serwerowego i klienckiego w jednym języku programowania, przykładowo algorytmy walidacji, metody wejścia/wyjścia i operacje na plikach binarnych, co powoduje coraz większe zainteresowanie wśród programistów innych technologii typu server-side takich jak Python, PHP, Ruby, Java, Perl, C#.

2.3 Socket.IO

HTML5 WebSocket to technologia zapewniająca kanał komunikacji pomiędzy przeglądarką internetową, a serwerem internetowym w obu kierunkach za pomocą jednego gniazda TCP.

Wszystkie nowoczesne przeglądarki wspierają już tę technologię, a Socket.IO jest biblioteką, która pozwala na stosunkowo szybkie tworzenie aplikacji czasu rzeczywistego w przeglądarkach i aplikacjach mobilnych za pomocą ujednoliconego interfejsu, zacierając przy tym różnice pomiędzy różnymi mechanizmami transportu i interpretacjami twórców oprogramowania.

Węzeł komunikacji w przypadku WebSockets jest w przeciwieństwie do AJAX otwarty na stałe pomiędzy klientem, a serwerem. Jednak dane pomiędzy socketami są przesyłane poprzez HTTP jako obiekty XHR (XMLHttpRequest) podobnie jak w przypadku AJAX'a.

2.4 MongoDB

MongoDB jest nierelacyjną bazą danych (NoSQL), która przechowuje dane w formie klucz-wartość o postaci dokumentów JSON, takim samym jak obiekty JavaScript, co sprawia, że nie ma potrzeby konwersji danych podczas odczytu i zapisu danych przez Node.js, dzięki temu wymiana danych jest szybsza niż w przypadku nierelacyjnych danych a komunikacja nie blokuje systemów wejścia/wyjścia.

Dokumenty takie nie wymagają określonej struktury i dane w nich nie są sztywno powiązane relacjami, dzięki czemu są wysoce skalowalne i pozwalają na szybszy rozwój aplikacji dzięki skoncentrowaniu na celu, a nie środkach do jego osiągnięcia.

Mongoose

Istnieje wiele sterowników do MongoDB dla różnych języków programowania. W niniejszej pracy został użyty pakiet Mongoose.js, narzędzie typu ODM (Object Document Mapping), które dba o schemat modelu danych, jego walidację i zgodność typów dla konkretnych pól, a także abstrahuje relacyjne powiązania między kolekcjami bazy danych, jeśli zachodzi taka potrzeba.

2.5 PhoneGap

PhoneGap¹ jest frameworkiem do budowania aplikacji mobilnych o otwartym kodzie źródłowym stworzonym przez firmę Nitobi, z czasem odsprzedanym do Adobe Systems², która to nadal czynnie rozwija produkt.

Pozwala na tworzenie aplikacji webowych w technologiach HTML, CSS i JavaScript, które po kompilacji za pośrednictwem PhoneGap umożliwiają uruchamianie ich jako natywnych aplikacji wybranego mobilnego systemu operacyjnego.

¹<http://phonegap.com>

²<http://www.adobe.com/pl/>

Pisząc jeden, uniwersalny kod źródłowy aplikacji można uzyskać pliki instalacyjne dla różnych platform mobilnych. W chwili pisania tego tekstu PhoneGap umożliwia kompilację do natywnych aplikacji dla:

- Android
- iOS
- Blackberry OS 6.0+
- Blackberry 10
- WebOS
- Windows Phone 7 + 8
- Symbian
- Bada

Sercem PhoneGap jest otwarty projekt Apeche Cordova, umożliwiający dostęp do niskopoziomowych warstw sprzętowych za pomocą wysokopoziomowych abstrakcji w języku JavaScript.

Jaśniej mówiąc, istnieje możliwość wykorzystania m.in. aparatu, akcelerometru, kompasu, odbiornika GPS, głośników i mikrofonu urządzenia, na którym uruchamiana jest aplikacja, a także możliwy staje się dostęp do systemu plików z możliwością odczytu i zapisu, ustawień karty sieciowej urządzenia, listy kontaktów oraz emitowania zdarzeń takich jak powiadomienia zarówno dźwiękowych, jak i wibracyjnych.

Rozdział 3

Architektura systemu

3.1 Aplikacja serwerowa

Zaplecze aplikacji to serwer napisany w Node.js korzystający z dobrodziejstw niere-relacyjnej bazy danych MongoDB.

Umożliwia zarządzanie obiektami mającymi odzwierciedlić świat rzeczywisty określony dla Ośrodka Szkolenia Kierowców korzystającego ze stworzonego systemu informatycznego. Są to kolekcje placów manewrowych dostępnych do korzystania przez dany ośrodek szkolenia oraz instruktorów, którzy przynależą do tego ośrodka.

Routing aplikacji

Architektura aplikacji została zaprojektowana w zgodzie z metodologią REST (ang. Representational State Transfer), dzięki czemu konkretne akcje wykonywane są na podstawie adresu URL (ang. Uniform Resource Locator) pod jaki kierowane jest żądanie HTTP (ang. Hypertext Transfer Protocol) oraz jego typu (GET lub POST).

Jedną z głównych zalet takiego rozwiązania jest czytelność i intuicyjność nawigacji w serwisie.

Na listingu 3.1 przedstawiony jest kod aplikacji (znajdujący się w pliku `app.js`) odpowiedzialny za routing – przetwarzanie otrzymanego adresu żądania na akcje do wykonania po stronie serwera.

```
1  /**
2   * Routing map.
3   */
4  app.map({
5    '/': {
6      get: routes.index // Displays homepage
7    },
8
9    // Instructors Controller
10   '/instructors': {
11     get: instructors.findAll, // Displays all instructors list
12     post: instructors.addNew, // Saves new instructor to the
13       database
14    ('/:id'): {
15       get: instructors.findById, // Displays details of instructor
16         selected by ID
17       post: instructors.updateInstructor, // Updates data of one
18         instructor selected by ID
19     },
20     '_new': {
21       get: instructors.createNew // Displays form for adding new
22         instructor
23     },
24     '_edit/:id': {
25       get: instructors.editExisting // Displays form for editing
26         selected instructor
27     },
28     '_delete/:id': {
29       get: instructors.deleteItem // Removes instructor selected by
30         ID
31     },
32     '_generate': {
33       get: instructors.createNewWithFaker // Adds new instructor
34         with generated fake data
35     }
36   },
37
38   // Places Controller
39   '/places': {
40     get: places.findAll, // Displays all places list
41     post: places.addNew, // Saves new place to the database
42    ('/:id'): {
43       get: places.findById, // Displays details of one place
44         selected by ID
45       post: places.updatePlace // Updates data of one place
46         selected by ID
47     },
48     '_new': {
49       get: places.createNew // Displays form for adding new place
50     },
51     '_edit/:id': {
52       get: places.editExisting // Displays form for editing
```

```

    selected place
44     },
45     '_delete/:id': {
46         get: places.deleteItem // Removes place selected by ID
47     },
48     '_generate': {
49         get: places.createNewWithFaker // Adds new place with
           generated fake data
50     }
51     , '_test': {
52         get: places.testBase64 //
53     }
54 },
55
56 // JSON API Controller
57 '/api': {
58     '/places': {
59         get: places.serveAllPlacesJson, // Responses with list of all
           places in JSON format
60         '/:id': {
61             get: places.serveOnePlaceJson, // Responses with dat of
           concrete place selected by ID in JSON format
62             post: places.occupyPlace // Marks selected place as
           occupied
63         }
64     },
65     '/instructors': {
66         get: instructors.serveAllInstructorsJson, // Responses with
           list of all instructors in JSON format
67         '/:id': {
68             get: instructors.serveOneInstructorJson, // Responses with
           dat of concrete instructors selected by ID in JSON
           format
69         }
70     },
71     '/login': {
72         get: auth.login // Authenticates mobile client before allows
           to exchange other data
73     },
74     '/validate_existance': {
75         get: auth.validateExistance
76     }
77 }
78
79 });
```

Listing 3.1: Mapa routingu typu REST aplikacji serwerowej

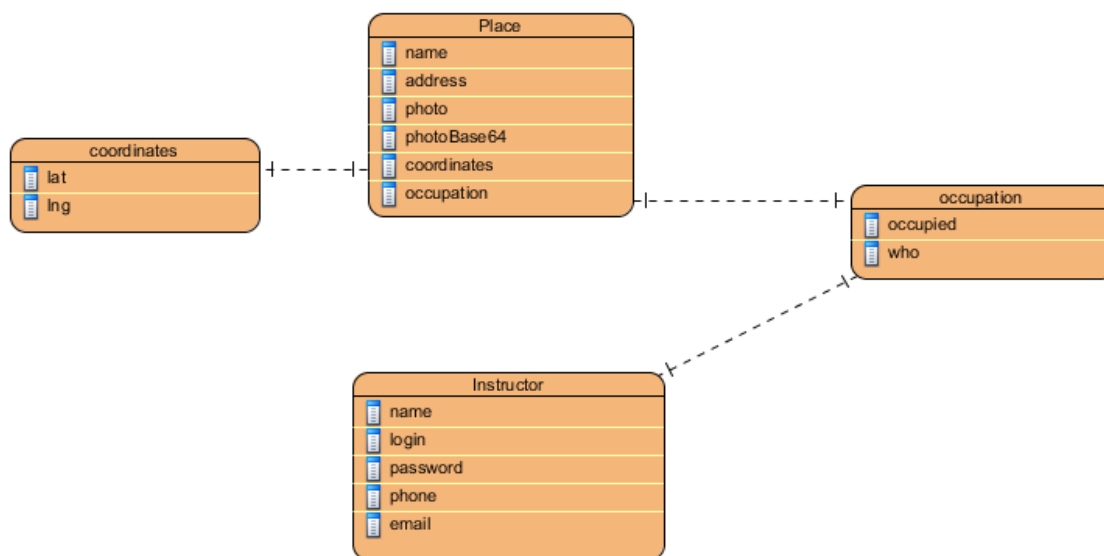
Analizując powyższy kod, można stwierdzić, że np. żądanie typu GET pod adresem URL `http://adres_serwera/places_edit/5` zwróci formularz edycji placu o numerze ID 5, a żądanie typu POST pod adresem `http://adres_serwera/places/5` zaktualizuje ten plac nowymi danymi według przesłanych z formularza parametrów.

3.2 Model bazy danych

Baza danych MongoDB zawiera dwie główne kolekcje:

- Instructors
- Places

Zachodzi w nich relacja jeden do jednego, ponieważ w danej chwili tylko jeden plac może być zajmowany przez jednego instruktora.



Rysunek 3.1: Schemat bazy danych

Do stworzenia schematu kolekcji dokumentów bazy danych MongoDB zostało wykorzystane narzędzie typu ODM o nazwie Mongoose (patrz rozdział 2. Wykorzystywane technologie).

Listing 3.2 zawiera schemat bazy placów, a listing 3.3 przedstawia schemat bazy instruktorów

```

1 // The Place model
2
3 var mongoose = require('mongoose')
4   , Schema   = mongoose.Schema
5   , ObjectId = Schema.ObjectId
6   , instructorSchema = require('./instructors');
7
8 var placeSchema = new Schema({
9   place: ObjectId,
10  name: String,
11  address: String,
12  photo: String,

```

```
13     photoBase64: String,
14     coordinates: {
15       lat: Number,
16       lng: Number
17     },
18     occupation: {
19       occupied: {
20         type: Boolean,
21         default: false
22       },
23       who: {
24         type: ObjectId,
25         ref: 'Instructors'
26       }
27     }
28   });
29
30 module.exports = mongoose.model('Place', placeSchema);
```

Listing 3.2: Schemat danych placów manewrowych – plik: /models/places.js

```
1 // The Instructors model
2
3 var mongoose = require('mongoose')
4   , Schema   = mongoose.Schema
5   , ObjectId  = Schema.ObjectId;
6
7 var instructorSchema = new Schema({
8   instructor: ObjectId,
9   name: String,
10  login: String,
11  password: String,
12  phone: String,
13  email: String
14 });
15
16 module.exports = mongoose.model('Instructors', instructorSchema);
```

Listing 3.3: Schemat danych instruktorów – plik: /models/instructors.js

3.3 Panel zarządzania

Panel zarządzania aplikacją umożliwia wykonywanie operacji CRUD (ang. Create-Read-Update-Destroy) na danych aplikacji.

Edycji użytkowników można dokonywać w obrębie /instructors – jednego z głównych kontrolerów aplikacji. A kontroler /places zawiera akcje odpowiedzialne za operacje na danych dotyczących placów manewrowych.

3.4 Klient mobilny

Można rozróżnić trzy sposoby wytwarzania aplikacji mobilnych, efektem których można wymienić 3 typy:

- aplikacje natywne,
- aplikacje webowe,
- aplikacje hybrydowe.

Aplikacje natywne

To oprogramowanie tworzone najczęściej w języku programowania takim jak np. Java, Objective-C czy C# w zależności od platformy. Są to aplikacje pisane pod konkretny system operacyjny i mogą korzystać ze wszystkich udogodnień i funkcji sprzętowych, jakie umożliwia dane urządzenie oraz ingerować w zachowania systemu i rozszerzać jego możliwości. Są bardzo szybkie i płynne, ich interfejs może wykorzystywać systemowe UI (np. Metro w Windows Phone, Holo w Androidzie), dzięki czemu ujednolicają się doskonale z systemem, co pozwala lepiej trafić w przyzwyczajenia użytkowników. Ich największą wadą jest koszt produkcji, ponieważ, aby dotrzeć do szerszego grona odbiorców, na każdy mobilny system operacyjny musi być przygotowana oddzielna aplikacja.

Aplikacje webowe

To nic innego jak strony internetowe zbudowane w nowych technologiach webowych, które przypominają wyglądem aplikacje mobilne. Uruchamiane są w przeglądarce internetowej i przez to wymagają ciągłego połączenia z Internetem oraz niemożliwe jest wykorzystanie w nich funkcji sprzętowych urządzenia, jakimi są m.in. aparat, kamera, GPS, żyroskop, akcelerometr. Największą i chyba jedyną zaletą tego typu aplikacji jest dostępność. Nie trzeba nic instalować, wystarczy jedynie wpisać adres internetowej aplikacji w przeglądarce internetowej, aby ją uruchomić. Aplikacje takie są uniwersalne, ponieważ maszyną uruchomieniową jest przeglądarka, więc wystarczy jeden język programowania (najczęściej jest to JavaScript), aby stworzyć uniwersalną aplikację działającą na wszystkich platformach. Co sprawia, że tego typu oprogramowanie jest stosunkowo tanie w porównaniu z aplikacjami natywnymi, gdzie nakład pracy jest o wiele większy.

Aplikacje hybrydowe

Czyli takie jak opisywana w niniejszej pracy. Jak sama nazwa wskazuje, jest to połączenie aplikacji natywnych i webowych. Użytkownicy często nie odróżniają ich

od aplikacji natywnych, jednak w sporej części budowane są w technologiach webowych takich jak HTML5, CSS3 i JavaScript, więc możliwe jest stworzenie jednej wersji oprogramowania, które działać będzie na wielu platformach. Hybrydy, tak samo jak aplikacje mobilne instalowane są w systemie i dystrybuowane w sklepach z aplikacjami takimi jak App Store czy Google Play (dawniej Android Market). Jako, że po części napisane są w języku natywnym, a po części w webowym (wspomniane wcześniej HTML 5 czy CSS3), mają dostęp do pewnych funkcji urządzeń, na których są uruchamiane. Można wykorzystać dobrodziejstwa m.in. akcelerometru, GPS, mikrofonu, kamery, czy zapisywać i odczytywać dane z pamięci urządzenia. Aplikacje hybrydowe umożliwiają osiągnięcie idealnej proporcji pomiędzy jakością i możliwościami aplikacji natywnych, a niskim kosztem wytwarzania aplikacji webowych.

3.5 Komunikacja Klient-Serwer - API

Dane pomiędzy klientem, a serwerem wymieniane są za pomocą protokołu JSONP (ang. JSON with padding) – techniki komunikacyjnej umożliwiającej wykonywanie AJAX’owych zapytań do i z serwera znajdującego się w innej domenie. W normalnym przypadku taka wymiana danych nie jest możliwa ze względów bezpieczeństwa. Przeglądarki nie zezwalają na pobieranie obiektów JSON z zewnętrznego serwera w celu zapobiegania atakom typu XSS (ang. Cross-site scripting). Udostępniają wykonywanym skryptom jedynie te obiekty, które pochodzą z tego samego źródła, co strona, w której kontekście są uruchamiane. JSONP polega na opakowaniu danych zwrotnych w funkcję typu „callback”, której parametr jest znany po obu stronach protokołu komunikacji, dzięki temu serwer zwraca dane jako zawartość funkcji do wywołania o takiej samej nazwie, jakiej spodziewa się klient. Kod zwrócony z serwera jest wstrzykiwany dynamicznie, a uruchomiony callback udostępnia pobrane dane do innych obiektów w przeglądarce.

Listing 3.1 zawiera routing dla wszystkich akcji udostępniających metody wymiany danych poprzez JSONP znajdujących się w kontrolerze `/api`. Są w nim metody do pobierania list placów i instruktorów, a także do logowania.

Pobranie danych aplikacji

Aplikacja po pomyślnym zalogowaniu pobiera listę placów i instruktorów w formacie JSON, zapisuje w lokalnej bazie danych WebSQL dostępnej w silniku renderującym WebKit wykorzystywanym w Androidzie oraz przetwarza je na interaktywne listy HTML wyświetlane jako zawartość aplikacji.

Wymiana poleceń przez API

Każdy plac ma swój stan zajętości zapisywany w bazie i udostępniany do publicznej wiadomości pomiędzy instruktorami zalogowanymi do systemu.

Informacje o zmianach tego stanu rozpraszane są w systemie za pośrednictwem HTML5 WebSockets, który wykorzystuje biblioteka Socket.IO użyta do stworzenia oprogramowania.

W sytuacji, kiedy jeden z klientów usiłuje zająć jakiś plac, serwer otrzymuje o tym informację za pośrednictwem socketu, zapisuje do bazy danych informację o tym, że plac jest zajęty i przez kogo, w następstwie czego emituje wydarzenie, które jest propagowane do wszystkich klientów podłączonych do systemu z informacją o konieczności oznaczenia tego placu jako zajęty.

Taki stan jest utrzymywany do momentu, kiedy instruktor zwolni plac i do serwera zostanie przesłana wiadomość o zmianie stanu. Po czym serwer zmienia stan w bazie danych i rozsyła analogiczny event do wszystkich połączonych klientów, aby umożliwili od nowa zajmowanie tego placu przez innych.

Algorytm tych działań przedstawia listing 3.4.

Komunikacja pomiędzy socketami jest bardzo szybka, dzięki czemu w systemie nie ma opóźnień związanych z przepływem informacji.

```
1 // Socket.IO events
2 io.sockets.on('connection', function (socket) {
3
4     socket.on('placeIsOccupied', function (data) {
5         places.occupyPlace(data.place, data.instructor, function(){
6             io.sockets.emit('disablePlace', { place: data.place });
7         });
8     });
9
10    socket.on('placeIsFree', function (data) {
11        places.releasePlace(data.place, function(){
12            io.sockets.emit('enablePlace', { place: data.place });
13        });
14    });
15
16 });
```

Listing 3.4: Mechanizm obsługi socketów po stronie serwera

Rozdział 4

Uruchomienie aplikacji

4.1 Aplikacja serwerowa

Do uruchomienia aplikacji potrzebny jest interpreter Node.js. Można go pobrać ze strony projektu i zainstalować w systemie. Dostępne są dla wszystkich popularnych systemów operacyjnych. I uruchomić skrypt poleceniem `node app.js` w katalogu głównym projektu za pomocą wiersza poleceń. Od momentu uruchomienia aplikacji, serwer będzie nasłuchiwał na określonym w konfiguracji porcie – domyślnie jest to port 3000.

Innym sposobem uruchomienia jest możliwość wykorzystania jednego z dostawców usług hostingowych typu PaaS (ang. Platform as a Service) takich jak np. AppFog ¹, Heroku ², czy OpenShift ³ i uruchomienia aplikacji w chmurze jako usługa Node.js. Ta metoda jest uzależniona od wybranego usługodawcy i więcej informacji można uzyskać w dokumentacji dotyczącej tegoż hostingu.

Oprócz działającego interpretera Node.js, który będzie w stanie uruchomić aplikację, potrzebna jest jeszcze baza danych MongoDB. Proces instalacji tego oprogramowania dla wybranych systemów operacyjnych jest szczegółowo opisany na stronie projektu ⁴. Gdzie są też dostępne binaria instalacyjne dla najpopularniejszych systemów operacyjnych.

Najprostszym sposobem jest jednak skorzystanie z usług MongoLab ⁵ – platformy działającej w chmurze, umożliwiającej uruchomienie własnej instancji bazy danych MongoDB o rozmiarach do 500 MB za darmo. Po rejestracji i stworzeniu pierwszej bazy danych, otrzymamy dane potrzebne do połączenia z bazą.

¹<http://www.appfog.com/>

²<https://www.heroku.com/>

³<https://www.openshift.com/>

⁴<http://www.mongodb.org/>

⁵<https://mongolab.com>

Konfiguracja bazy danych

Ważnym plikiem służącym do konfiguracji połączenia z bazą jest znajdujący się w głównym katalogu aplikacji plik `config.json`, którego zawartość należy wyedytować wstawiając do niego prawidłowe dane do połączenia. Listing 4.1 zawiera domyślną zawartość pliku konfiguracyjnego.

```
1 {  
2   "mongodb": {  
3     "host" : "adres_serwera",  
4     "port" : port, // np. 3456  
5     "dbname" : "nazwa_bazy",  
6     "user" : "uzytkownik_bazy",  
7     "password" : "haslo_bazy"  
8   }  
9 }
```

Listing 4.1: Domyślna zawartość pliku z konfiguracją połączenia do bazy danych – plik: `/config.json`

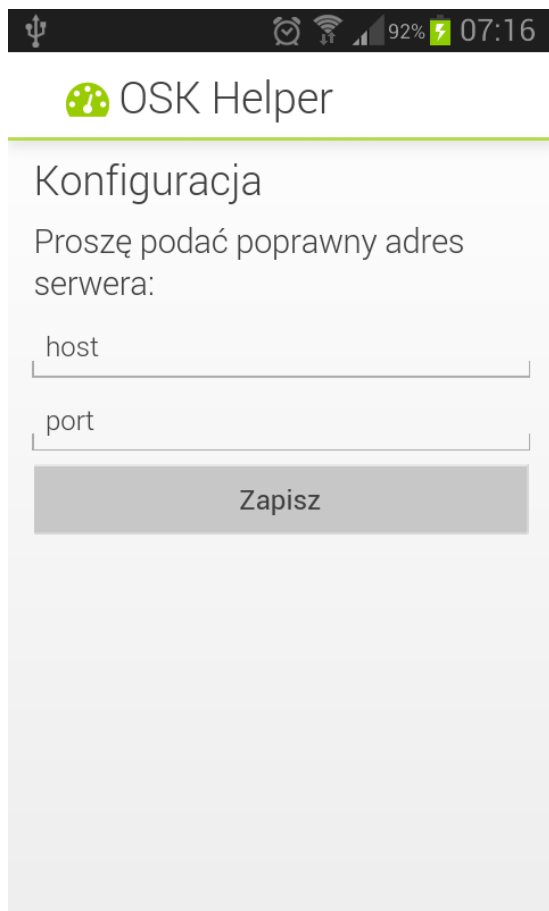
4.2 Aplikacja mobilna na platformę Android

Do uruchomienia klienta mobilnego potrzebny jest smartfon lub tablet z systemem operacyjnym Android oraz plik instalacyjny ze skompilowaną aplikacją.

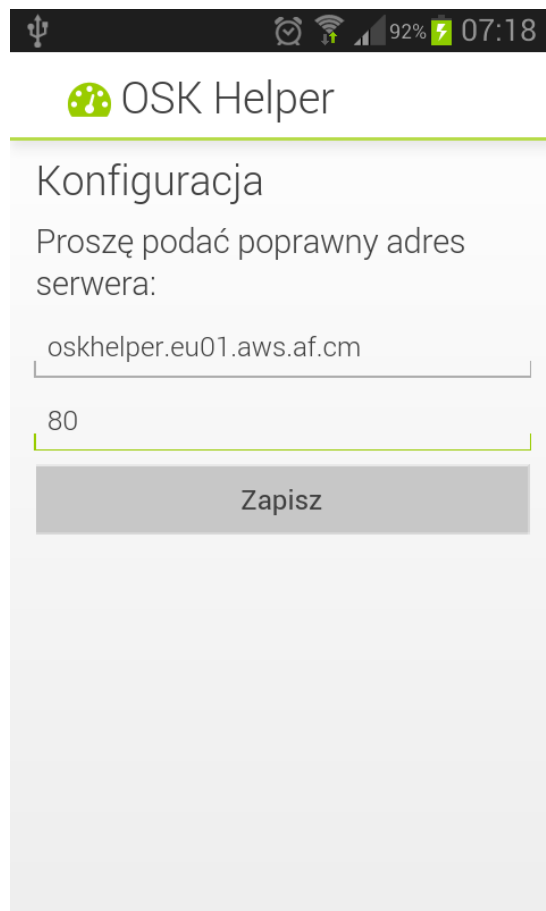
Aplikacja wymaga dostępu do Internetu, aby móc komunikować się z serwerem oraz włączonego GPS, w celu liczenia odległości do placów manewrowych wyświetlanych na liście.

Konfiguracja ścieżki serwera w aplikacji mobilnej

Przy pierwszym uruchomieniu zainstalowanej aplikacji pojawia się ekran wstępnej konfiguracji, w którym należy podać adres (IP, bądź domena) oraz port, na którym działa uruchomiony serwer stanowiący back-end systemu, co przedstawiają rysunki 4.1 (pusty formularz) oraz 4.2 (wypełniony danymi zgodnymi dla serwera testowego).



Rysunek 4.1



Rysunek 4.2

Po wprowadzeniu adresu hosta i portu aplikacja mobilna próbuje odpytać serwer pod wskazanym adresem z akcją `/api/validate_existance`, aby sprawdzić czy wprowadzone dane są poprawne (tzw. heartbeat).

Jeśli serwer zwraca odpowiedź w formacie JSON o treści `{'exists': true}`, podany adres serwera jest zapisywany w pamięci aplikacji (HTML5 LocalStorage), po czym wyświetlany jest ekran logowania. Przy kolejnych uruchomieniach nie trzeba go podawać, aż do czasu wylogowania, które spowoduje wyczyszczenie lokalnej bazy danych aplikacji.

Rozdział 5

Panel zarządzania - konfiguracja

Panel zarządzania służy do prawidłowego skonfigurowania obiektów świata aplikacji, który ma odzwierciedlać rzeczywistość, w ramach której działa Ośrodek Szkolenia Kierowców, a w zasadzie jego pracownicy korzystający z systemu.

Możliwa jest edycja placów manewrowych i listy instruktorów z uwzględnieniem wszystkich operacji typu CRUD (ang. Create-Read-Update-Destroy). A zatem każdy z placów można edytować i usuwać, wyświetlić jego dane oraz dodać nowy plac do modelu danych aplikacji.

5.1 Zarządzanie listą placów manewrowych

Ważne jest prawidłowe skonfigurowanie wszystkich placów, aby instruktorzy mogli z nich korzystać.

W momencie, kiedy w bazie nie ma jeszcze dodanych żadnych placów, najważniejszym ekranem jest formularz dodawania nowego placu (Rysunek 5.1).

OSK-Helper

Home

Instruktorzy ▾

Place ▾

Nowy plac

Dane podstawowe

Nazwa

Plac #...

Adres

Dworcowa 358...

Miniaturka

Zdjęcie

Wybierz plik

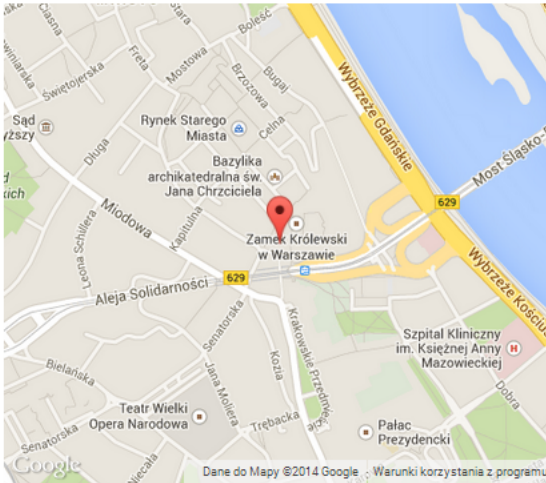
Koordynaty GPS

Latitude

52.24729983756614

Longitude

21.013530492782593



Zapisz

Anuluj

Rysunek 5.1: Formularz dodawania nowego placu






Interaktywna mapa pozwala na precyzyjne określenie współrzędnych. Pola zawierające koordynaty GPS są automatycznie uaktualniane w momencie kliknięcia w wybrany punkt na mapie.

Przesyłane na serwer zdjęcie placu jest przy okazji zapisu przetwarzane na ciąg znaków w formacie Base64 i również zapisywane do bazy danych. Takie zdjęcie zakodowane w formie tekstowej jest przesyłane wraz z innymi danymi placów do aplikacji mobilnej, która z powrotem potrafi je wyświetlić jako obrazek i przetrzymać w swojej lokalnej bazie danych.

Na rysunku 5.2 przedstawiony jest ekran z listą, na której widoczne jest kilka pierwszych placów wprowadzonych do bazy danych. Wprowadzone dane są wygenerowane losowo przy pomocy biblioteki Faker.js ¹.

OSK-Helper Home Instruktorzy Place

Lista placów

Zdjęcie	Nazwa	Adres	Status	Akcje
	Plac #80	35782 Bode Ways	Wolny	Szczegóły Edycja Usuń
	Plac #51	1019 Hodkiewicz Coves	Wolny	Szczegóły Edycja Usuń
	Plac #23	2101 Brakus Summit	Zajęty	Szczegóły Edycja Usuń
	Plac #99	27020 Kreiger Park	Wolny	Szczegóły Edycja Usuń
				

Rysunek 5.2: Lista utworzonych placów manewrowych

Plac o nazwie „Plac #23” na powyższym rysunku jest oznaczony jako zajęty. Wszystkie tego typu place są podświetlone wyblakłym odcieniem koloru czerwonego.

¹<https://github.com/marak/Faker.js/>

Rysunek 5.3 przedstawia widok formularza do edycji danych wybranego placu.

Nie różni się on bardzo od formularza dodawania nowego placu poza tym, że ustawione aktualnie pola są już wypełnione bieżącymi danymi oraz nad częścią dotyczącą wstawiania nowego zdjęcia widnieje podgląd aktualnie wstawionej miniaturki.

Istnieje możliwość zmiany nazwy placu, adresu, współrzędnych i wgrania nowego zdjęcia.

OSK-Helper Home Instruktorzy Place

Edycja placu: Plac #80

Dane podstawowe

Nazwa


Adres

Koordynaty GPS

Latitude

Longitude


Aktualne zdjęcie

Zdjęcie 

Nowe zdjęcie

Zdjęcie

Wybierz plik



Zapisz

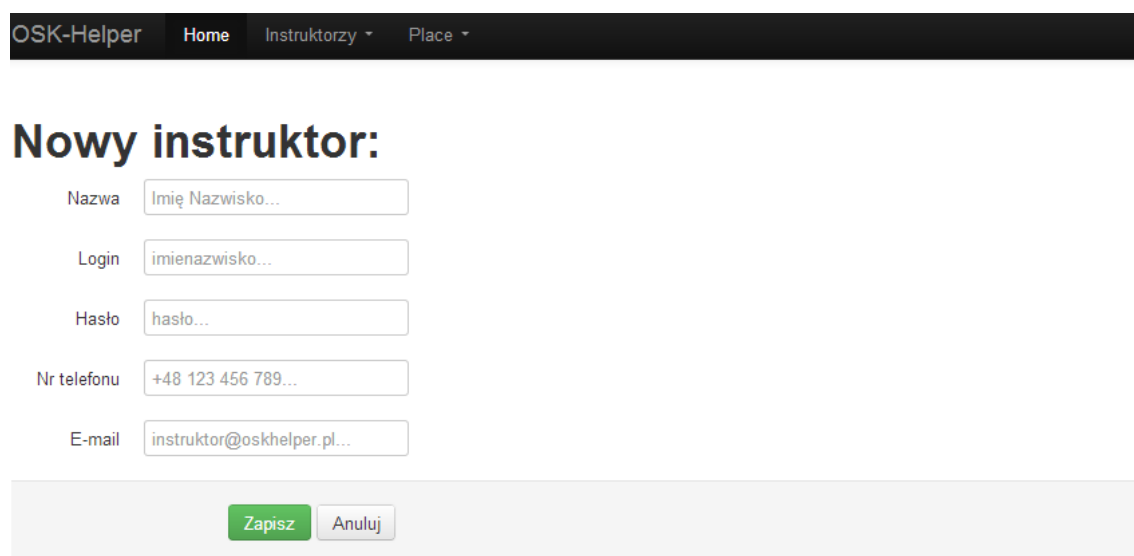
Anuluj

Rysunek 5.3: Edycja wybranego placu manewrowego

5.2 Zarządzanie listą instruktorów

W przypadku konfiguracji bazy instruktorów należy postępować analogicznie.

Rysunek 5.4 przedstawia ekran z formularzem dodawania nowego instruktora. Wymienić można pola takie jak: jego nazwa osobowa, login i hasło, których będzie używał do logowania w aplikacji mobilnej oraz danych kontaktowych takich jak telefon i mail, do których wgląd będą mieli inni instruktorzy z poziomu mobilnego klienta.



The screenshot shows the 'Nowy instruktor:' form in the OSK-Helper application. The form is located on a page with a dark header containing the application name and navigation links. The form itself has a light gray background and contains several input fields for user data. At the bottom of the form are two buttons: 'Zapisz' (Save) and 'Anuluj' (Cancel).

Label	Input Value
Nazwa	Imię Nazwisko...
Login	imienazwisko...
Hasło	hasło...
Nr telefonu	+48 123 456 789...
E-mail	instruktor@oskhelper.pl...

Rysunek 5.4: Formularz dodawania nowego instruktora

Po dodaniu kilku instruktorów lista wygląda tak, jak to przedstawia rysunek 5.4.

OSK-Helper [Home](#) [Instruktorzy](#) [Place](#)

Lista instruktorów:

Nazwa	login	Numer tel.	E-mail	Akcje
Dortha Runte	iving	244-471-2958	elton@timothy.biz	Szczegóły Edycja Usuń
Vella Hermann	leanne	(088)116-7229 x61450	ferne@tianna.org	Szczegóły Edycja Usuń
Nat Collins	tillman	1-323-322-7075	mariano@zackery.ca	Szczegóły Edycja Usuń
Ismael Stehr	twila_renner	1-477-153-3313 x763	jackie.schmitt@brett.info	Szczegóły Edycja Usuń
Lew Reichert	alex.goldner	(701)909-3474	mariane_witheiser@ashton.biz	Szczegóły Edycja Usuń
Joanne Mayer	marcelina	656-871-7391	sandrine.tremblay@jaden.us	Szczegóły Edycja Usuń
Mable Gerlach	isidro_turcotte	1-036-149-3994 x3602	thelma@twila.org	Szczegóły Edycja Usuń
Melissa Howell	sammie.barrows	(519)113-3580 x98840	sage_lubowitz@regan.net	Szczegóły Edycja Usuń
Haylie Stoltenberg	myrna	893-909-4704 x02189	wellington@hailie.tv	Szczegóły Edycja Usuń
Verner Mills	nicklaus_becker	(976)369-4620	sheldon@americo.info	Szczegóły Edycja Usuń
Roberta Jacobson	virgie	340-939-9546 x3697	tiara_doyle@favian.org	Szczegóły Edycja Usuń

Dodaj instruktora

Rysunek 5.5: Lista dodanych instruktorów

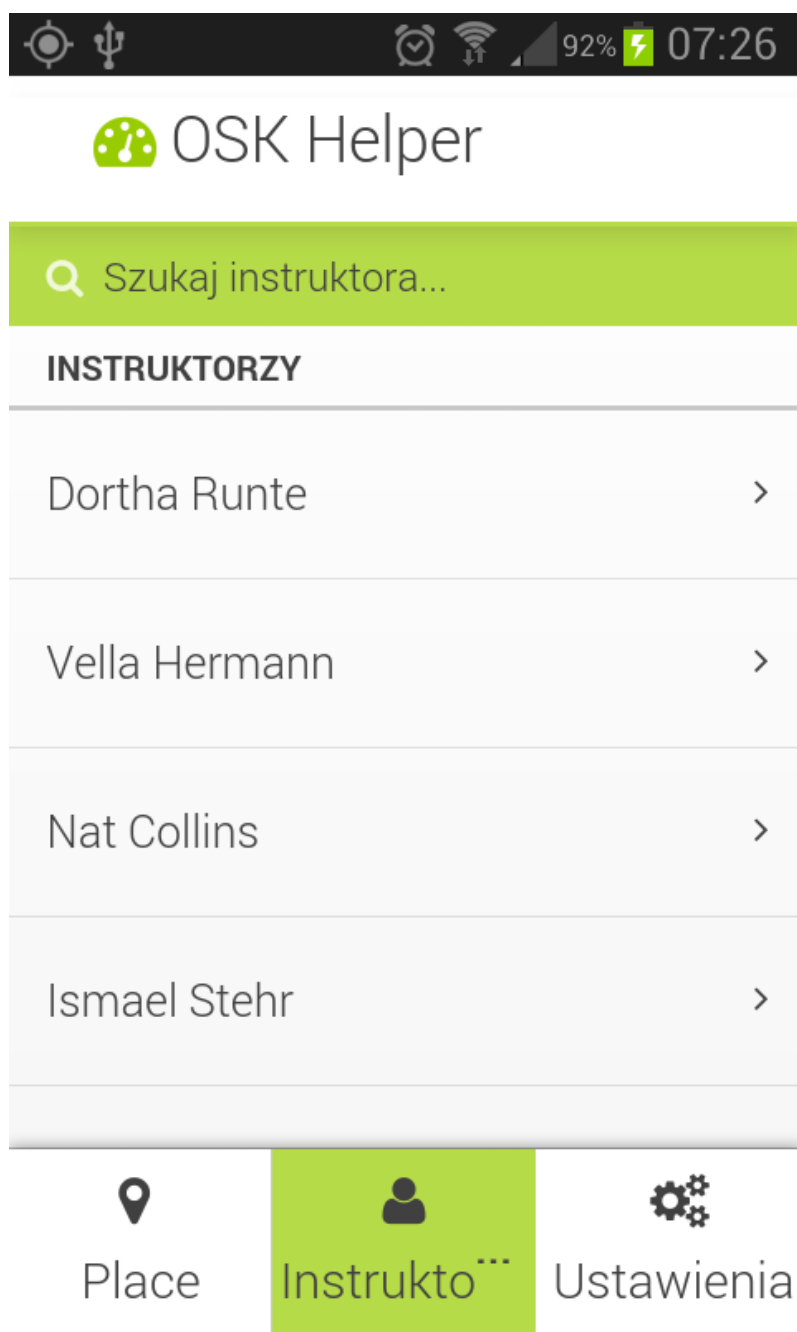
Rozdział 6

Klient dla systemu Android - korzystanie

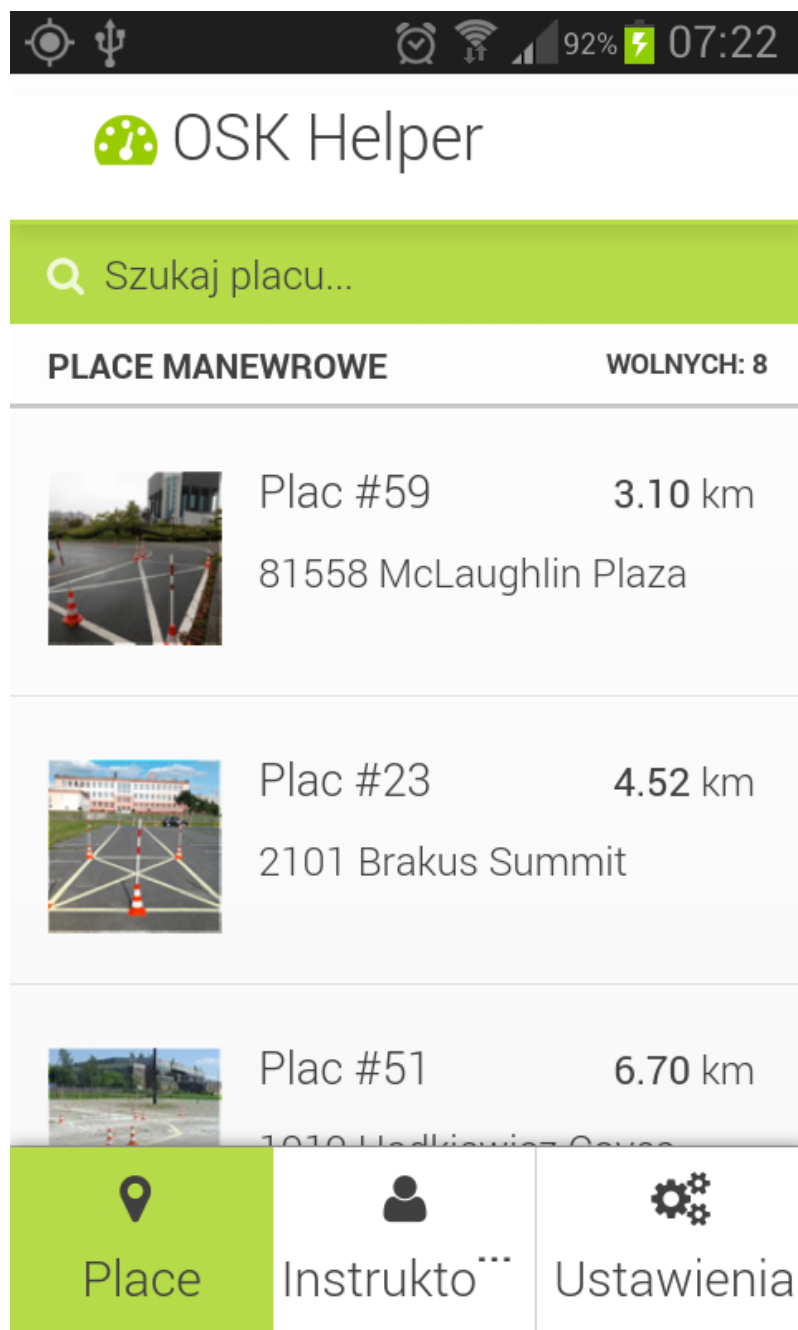
6.1 Przykładowy przepływ

6.2 Pogląd na listę placów

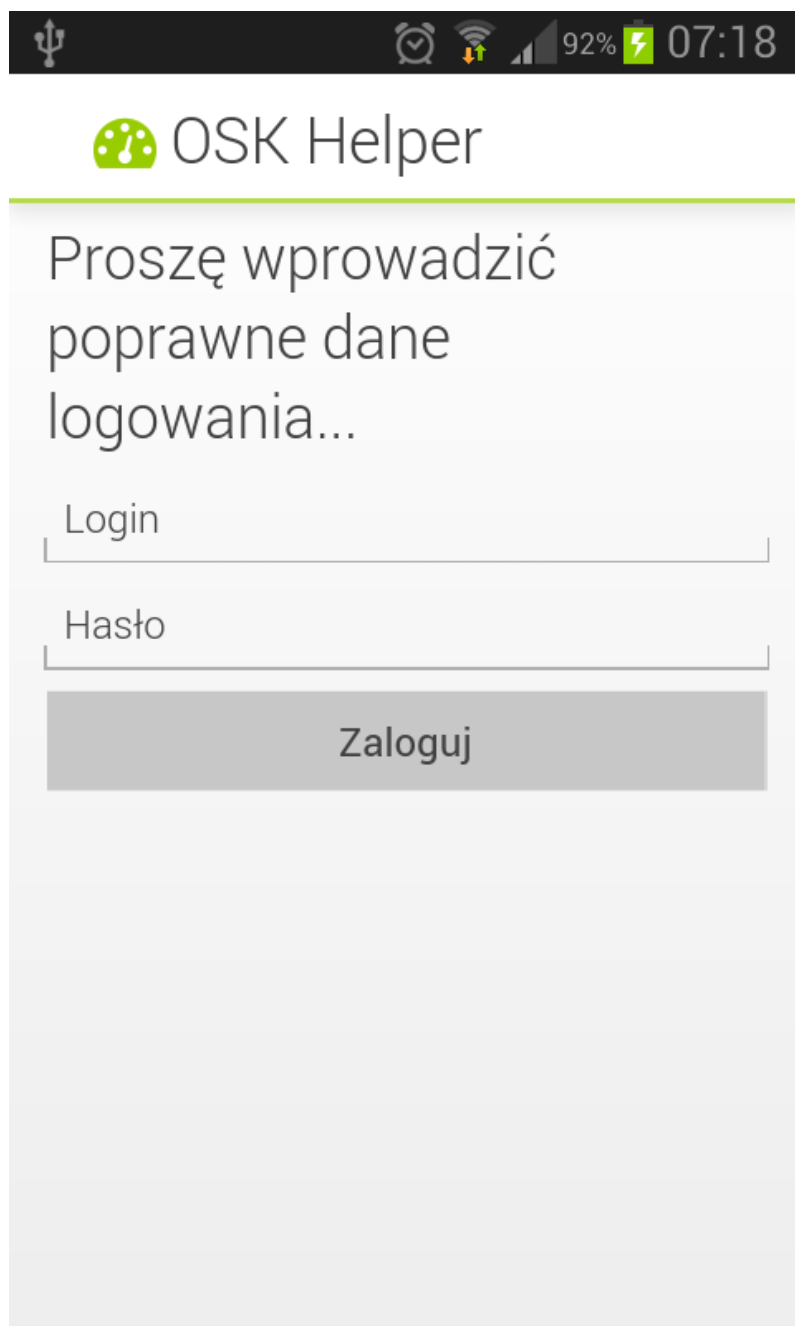
6.3 Zajmowanie i zwalnianie placu



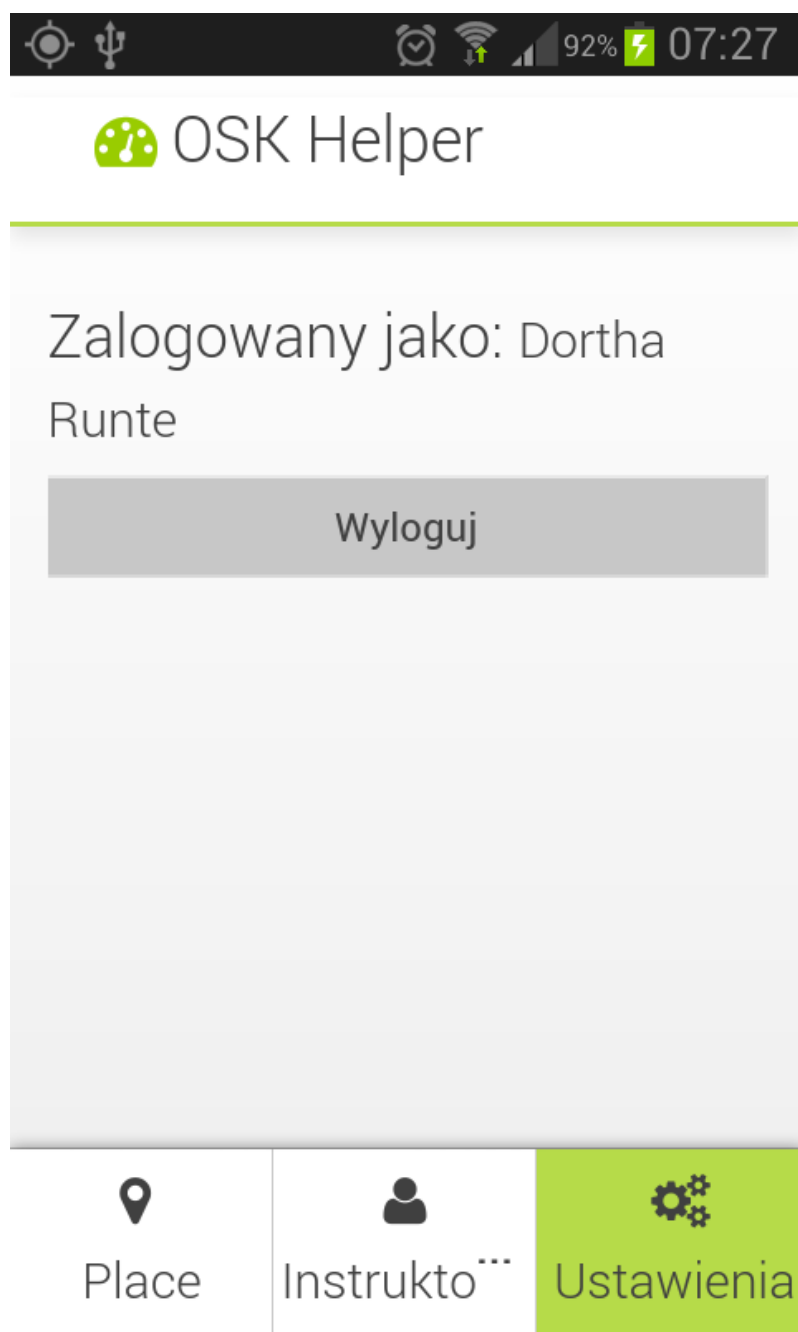
Rysunek 6.1: Lista instruktorów



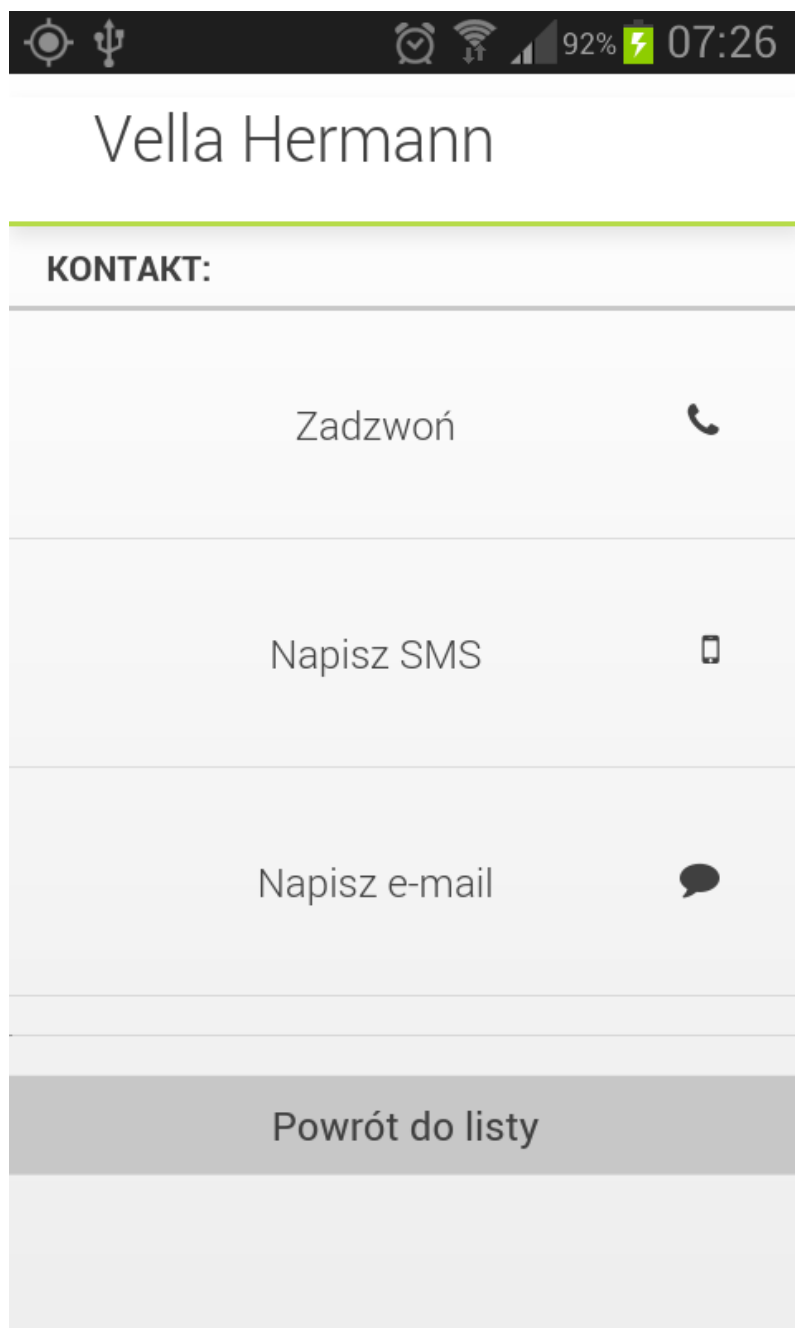
Rysunek 6.2: Lista placów - sortowana według odległości



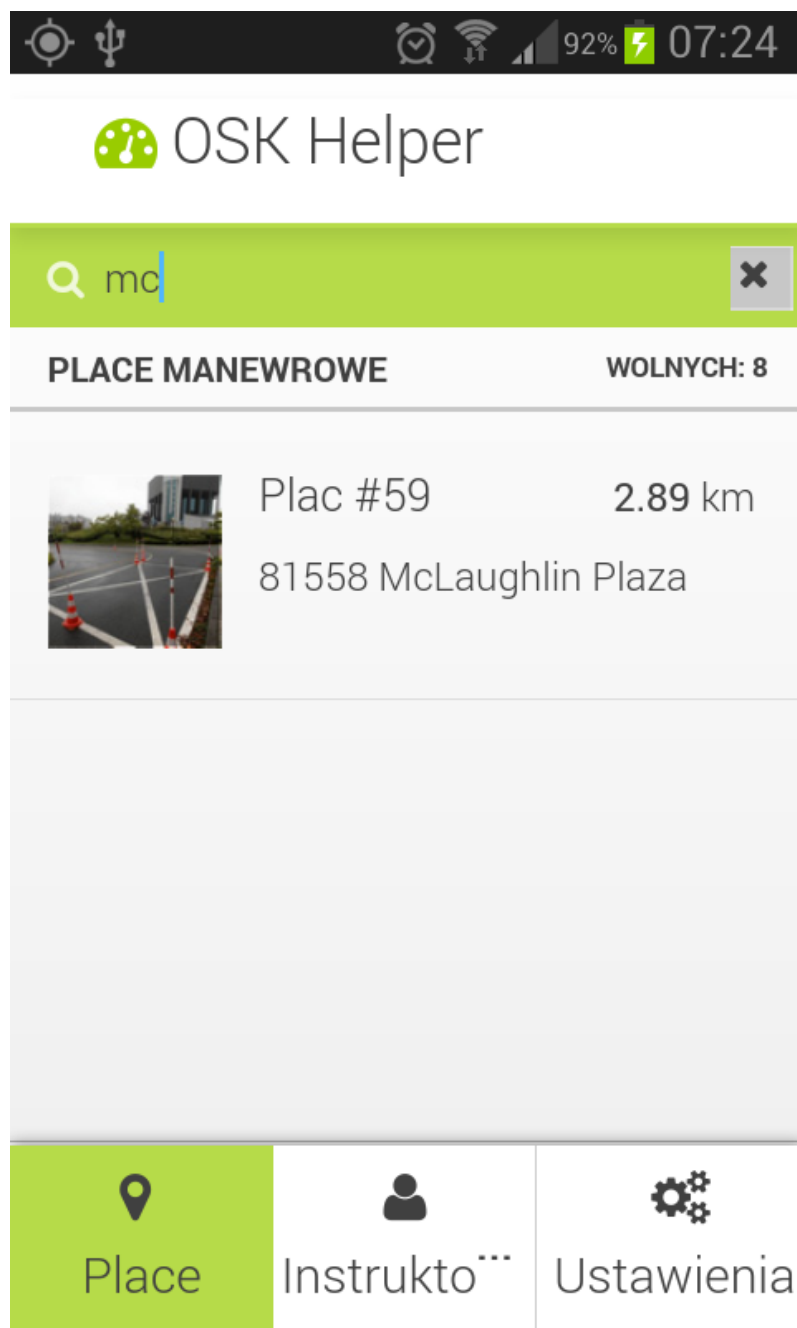
Rysunek 6.3: Ekran logowania użytkownika



Rysunek 6.4: Ekran ustawień - możliwość wylogowania



Rysunek 6.5: Widok szczegółowy wybranego instruktora



Rysunek 6.6: Lista placów - przefiltrowana wg wyszukiwanego wzorca

Rozdział 7

Wnioski

Rozdział 8

Do usunięcia

8.1 Algorytmy

Pakiety **algorithmic** oraz **algorithm** dostarczają środowiska do definiowania algorytmów. Można za ich pomocą definiować zarówno pseudokod jak i algorytmy ogólne. Pseudokod przedstawia Algorytm 1. Przykładową ogólną reprezentację algorytmu Levenshteina obrazuje Algorytm 2.

Algorytm 1 Pseudokod prezentujący jakiś bezsensowny algorytm

```
1: program FancyProgram {
2:   function superFunkcja(List listA) {
3:     for all element : listA.getElements() do
4:       int result = storeElement(element);
5:       if result == SUCCESS then
6:         double number = result*element;
7:       else if result == FAILURE then
8:         double number = result/element;
9:       end if
10:    end for
11:  }
12:  input List elements;
13:  output number;
14:  superFunkcja(elements);
15:  return number;
16: }
```

Podobnie jak w przypadku rysunków, algorytmy są automatycznie rozlokowywane w tekście przez system Latex.

Algorytm 2 Algorytm Levenshteina

1. Niech $n = \text{długość}(s)$ a $m = \text{długość}(t)$.
 2. Jeżeli $n == 0$ zwróć m i zakończ działanie.
 3. Jeżeli $m == 0$ zwróć n i zakończ działanie.
 4. Utwórz macierz d o $m + 1$ wierszach indeksowanych od 0 do m oraz $n + 1$ kolumnach indeksowanych od 0 do n , pierwszy wiersz zainicjuj wartościami od 0 do n a pierwszą kolumnę od 0 do m .
 5. Porównaj każdy znak pierwszego ciągu, indeksowany za pomocą i , z każdym znakiem drugiego ciągu, indeksowanym za pomocą j . Dla każdego porównania:
 - (a) jeżeli $s[i]$ jest identyczne z $t[j]$, $\text{koszt} = 0$, w przeciwnym wypadku $\text{koszt} = 1$,
 - (b) ustaw wartość komórki $d[i, j]$ macierzy na wartość minimalną spośród:
 - $d[i - 1, j] + 1$,
 - $d[i, j - 1] + 1$,
 - $d[i - 1, j - 1] + \text{koszt}$.
 6. Odczytaj odległość $\text{dist}_{lev}(s, t)$ z komórki $d[n, m]$.
-

8.2 Tymczasowa zmiana rozmiaru strony

Czasami zachodzi konieczność chwilowej zmiany rozmiaru strony, by np. udało się zmieścić jedną dodatkową linijkę tekstu. Możemy to wykonać za pomocą polecenia **enlargethispage{}**, gdzie jako parametr podajemy rozmiar oraz jednostkę. Należy pamiętać, że polecenie to musi zostać wydane odpowiednio wcześniej, by LaTeX zdążył zastosować nowy rozmiar strony. Najlepiej by te polecenie było bezpośrednio przed nową stroną, jednak zazwyczaj jest to kwestia poeksperymentowania. Wartość przekazana jako parametr może być także ujemna. Przykładowo strona zawierająca rozdział ?? została w niniejszym dokumencie pomniejszona o 5 cm wymuszając przeniesienie początku rozdziału 8.1 na następną stronę.

Strona zawierająca rozdział ?? powiększono o 20 punktów, co pozwoliło uniknąć samotnej, pojedynczej linii tekstu kończącego akapit (tzw. wdowy) na następnej stronie. Takie drobne modyfikacje rozmiaru strony zazwyczaj są niezauważalne dla czytelnika a poprawiają ogólny układ dokumentu.

8.3 Wpisy bibliograficzne

Wpisy bibliograficzne przechowujemy w odrębnym pliku z rozszerzeniem .bib. Przykładowy plik został dołączony do tego dokumentu. Do pliku takiego należy dodawać odpowiedni sformatowane wpisy. Latex automatycznie posortuje je po nazwiskach autorów oraz do finalnego dokumentu dołączy tylko te wpisy, które posiadają odwołania w tekście! Można więc stworzyć kompletną bazę publikacji, a Latex użyje tylko to co potrzeba. Dodatkowo, dzięki użyciu pakietu **natbib** z parametrem **sort** (patrz preambuła dokumentu), numerki w odwołaniach również zostaną posortowane, niezależnie od kolejności podania odwołań. Przykład podano w rozdziale ??.

Spis rysunków

3.1	Schemat bazy danych	10
4.1	17
4.2	17
5.1	Formularz dodawania nowego placu	20
5.2	Lista utworzonych placów manewrowych	21
5.3	Edycja wybranego placu manewrowego	22
5.4	Formularz dodawania nowego instruktora	23
5.5	Lista dodanych instruktorów	24
6.1	Lista instruktorów	26
6.2	Lista placów - sortowana według odległości	27
6.3	Ekran logowania użytkownika	28
6.4	Ekran ustawień - możliwość wylogowania	29
6.5	Widok szczegółowy wybranego instruktora	30
6.6	Lista placów - przefiltrowana wg wyszukiwanego wzorca	31

Listingi kodu

3.1	Mapa routingu typu REST aplikacji serwerowej	8
3.2	Schemat danych placów manewrowych – plik: /models/places.js . . .	10
3.3	Schemat danych instruktorów – plik: /models/instructors.js	11
3.4	Mechanizm obsługi socketów po stronie serwera	14
4.1	Domyślna zawartość pliku z konfiguracją połączenia do bazy danych – plik: /config.json	16

Spis algorytmów

1	Pseudokod prezentujący jakiś bezsensowny algorytm	35
2	Algorytm Levenshteina	36