УДК 004

Создание приложения такси для водителей и клиентов

Семченко Регина Викторовна
Приамурский государственный университет имени Шолом-Алейхема
Студент

Еровлев Павел Андреевич
Приамурский государственный университет имени Шолом-Алейхема
Студент

Научный руководитель:
Глаголев Владимир Александрович
Приамурский государственный университет имени Шолом-Алейхема
к.г.н., доцент кафедры информационных систем, математики и правовой информатики

Аннотация
В данной статье рассмотрена возможность создания приложения для такси. С помощью React Native и Pusher, созданное приложение похоже на популярные приложения похожие на Uber и Яндекс.Такси. Практическим результатом является рабочее приложение, разработанное исключительно для пользователей, а не для водителей
Ключевые слова: Такси, приложение, андроид

Creating a taxi application for drivers and customers

Semchenko Regina Viktorovna
Sholom-Aleichem Priamursky State University
Student

Erovlev Pavel Andreevich
Sholom-Aleichem Priamursky State University
Student

Scientific adviser:
Glagolev Vladimir Alexandrovich
Sholom Aleichem Priamursky State University
Ph.D., Associate Professor of the Department of Information Systems, Mathematics and Legal Informatics

Abstract
This article discusses the possibility of creating an application for a taxi. Using React Native and Pusher, the created application will look like popular applications

similar to Uber and Yandex.Taxi. The bottom line is a working application designed exclusively for users, not for drivers

Keywords: Taxi, app, android

Современное общество становится более информатизованным и почти каждый пользуется приложениями на своих смартфонах. Чтобы вызвать такси не надо уже звонить оператору и называть адреса и ждать подачи автомобиля не зная через сколько он приедет, сейчас достаточно открыть приложение и выбрать адрес, а после чего ждать подачи автомобиля, отслеживая его перемещения по карте в самом приложении, причем заранее можно узнать марку и цвет машины, а так же гос.номер для более точного определения машины, на случай чтобы не перепутать с другим водителем приехавшим на другой заказ.

Цель статьи состоит в создании мобильного приложения для клиентов такси, а так же для водителей, которые будут принимать заказ. В приложении будет реализована возможность вызова такси, определения местоположения, выбор точки назначения по карте, отслеживания подачи автомобиля, стоимости для пассажира и возможность выбора заказа, либо отмены, возможность увидеть где находится клиент на миникарте, если у клиента включены геоданные, а так же приложение маршрута до места назначения.

Исследованиями в области разработки мобильных приложений занимались многие российские и зарубежные исследователи. А.С.Винокуров, Р.И. Баженов [1] рассмотрели разработку приложений для мобильных устройств. D.Y. Bichkovski, F.N.Abu-Abed, A.R. Khabarov, К.А.Karelskaya [2] исследовали возможность информирования студентов с помощью андроид приложения. К.В. Аксенов [3] рассмотрел современные средства разработки мобильных приложений. В.Ю. Ким [4] изучал особенности дизайна интерфейса пользователя для приложений. Романов и др. [5] описали разработку мобильного приложения для управления документами из облачных хранилищ. E.W.T. Ngaia, A.Gunasekaran [6] рассмотрели методы разработки мобильных бизнес приложений.

Для начала создадим новое приложение React Notive (рис.1).

```
react-native init grabDriver
```

Рисунок 1 – создание нового приложения

Как только это будет сделано, перейдем в «grabDriver» каталог и установим библиотеки, которые нужны. Они помогут «pusher-js» для работы с «Pusher», «React Native Maps» для отображения карты и «React Native Geocoding» для координат обратного геокодирования с фактическим названием места (рис.2).

```
npm install --save pusher-js react-native-maps react-native-geocoding
```

Рисунок 2 – установка зависимости

После того, как все библиотеки установлены, то для «React Native Maps» необходимо выполнить несколько дополнительных шагов, чтобы он заработал. Во-первых, это связывание ресурсов проекта (рис.3)

```
react-native link react-native-maps
```

Рисунок 3 – связывание ресурсов

Затем вам нужно создать проект в «Google», получить ключ API из консоли разработчика «Google» и включить «Google Maps Android API» и API Карт Google для геокодирования. После этого откроем «android\app\src\main\AndroidManifest.xml» файл в каталоге проекта. Под <application> добавить тег <meta-data> содержащий ключ API сервера (рис.4).

```
<application>
    <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="YOUR GOOGLE SERVER API KEY"/>
</application>
```

Рисунок 4 – добавление ключей API

Добавив следующие разрешения появится возможность проверитьсостояние сети и запрашивать геоданные с устройства (рис.5).

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Рисунок 5 – добавление разрешений

Далее необходимо установить «Genymotion», используем его вместо стандартного эмулятора Android, поскольку он поставляется с инструментом имитации GPS, который позволяет искать определенное местоположение и использовать его в качестве местоположения эмулируемого устройства. Он использует карты Google в качестве интерфейса, и также может перемещать маркер. Это позволяет моделировать движущееся транспортное средство.

Наконец, поскольку «Genymotion» будет использоваться для тестирования приложения драйвера, то необходимо следовать приведенным инструкциям:

1. Посетить opengapps.org
2. Выбрать x86 в качестве платформы
3. Выбрать версию Android, соответствующую нашему виртуальному устройству
4. Выбрать «nano» в качестве варианта
5. Загрузить почтовый файл
6. Перетащить установщик zip на новое виртуальное устройство Genymotion (только версии 2.7.2 и выше)
7. Следуйте инструкциям во всплывающем окне

Это делается потому, что библиотека React Native Maps в основном использует Google Maps. Нужно добавить сервисы Google Play, чтобы он работал. В отличие от большинства телефонов Android, которые уже поставлены с этим, Genymotion не имеет его по умолчанию из-за интеллектуальной собственности. Таким образом, нужно вручную установить его.

Теперь начнем писать код для приложения водителя. Начнем с открытия «index.android.js» файла и заменим код по умолчанию следующим (рис.6).

```
import { AppRegistry } from 'react-native';
import App from './App';
AppRegistry.registerComponent('grabDriver', () => App);

import React, { Component } from 'react';
import {
  StyleSheet,
  Text,
  View,
  Alert
} from 'react-native';


import Pusher from 'pusher-js/react-native';
import MapView from 'react-native-maps';

import Geocoder from 'react-native-geocoding';
Geocoder.setApiKey('YOUR GOOGLE SERVER API KEY');
```

Рисунок 6 – импорт из React

Создадим файл «helpers.js» и напишем там следующий код (рис.7).

```
export function regionFrom(lat, lon, accuracy) {
  const oneDegreeOfLongitudeInMeters = 111.32 * 1000;
  const circumference = (40075 / 360) * 1000;

  const latDelta = accuracy * (1 / (Math.cos(lat) * circumference));
  const lonDelta = (accuracy / oneDegreeOfLongitudeInMeters);

  return {
    latitude: lat,
    longitude: lon,
    latitudeDelta: Math.max(0, latDelta),
    longitudeDelta: Math.max(0, lonDelta)
  };
}

export function getLatLonDiffInMeters(lat1, lon1, lat2, lon2) {
  var R = 6371;
  var dLat = deg2rad(lat2-lat1);
  var dLon = deg2rad(lon2-lon1);
  var a =
    Math.sin(dLat/2) * Math.sin(dLat/2) +
    Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) *
    Math.sin(dLon/2) * Math.sin(dLon/2)
    ;
  var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
  var d = R * c;
  return d * 1000;
}

function deg2rad(deg) {
  return deg * (Math.PI/180)
}
```

Рисунок 7 – код отображения координат

Следующим шагом подключимся к серверу, который был создан ранее. Для этого из Pusher возьмем ключи и вставим их (рис.8)

```
componentWillMount() {
  this.pusher = new Pusher('YOUR PUSHER KEY', {
    authEndpoint: 'YOUR PUSHER AUTH SERVER ENDPOINT',
    cluster: 'YOUR PUSHER CLUSTER',
    encrypted: true
  });

  //
}
```

Рисунок 8 – подключение к серверу

Следующим шагом необходимо создать подключение к частному каналу. Это должен быть частный канал, поскольку клиентские события могут запускаться только на частных каналах и каналах присутствия по соображениям безопасности (рис.9).

Рисунок 9 – подключение к частному каналу

Далее напишем код для принятия заказа водителем (рис.10).

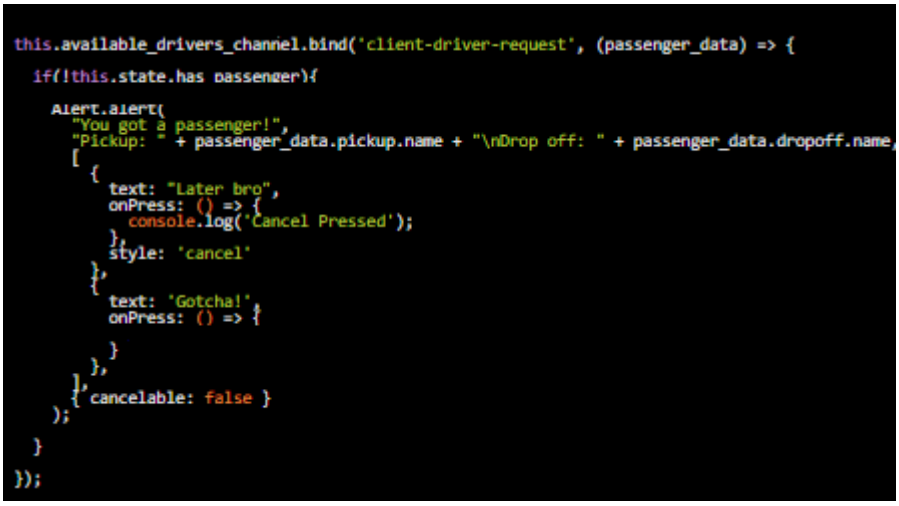

Рисунок 10 – принятие заказа

Как только водитель соглашается забрать пассажира, то он подписываемся на его частный канал. Этот канал зарезервирован только для связи между водителем и пассажиром, поэтому используем уникальное имя пользователя пассажира в качестве части имени канала.

Далее используем библиотеку геокодирования, чтобы определить название места, где в данный момент находится водитель. Здесь используется API геокодирования Google и обычно возвращается название улицы. Как только получаем ответ, то запускаем «found-driver» событие, чтобы сообщить пассажиру, что приложение нашло водителя для них. Он содержит информацию о драйвере, такую как имя и текущее местоположение (рис.11).

```
Geocoder.getFromLatLng(this.state.region.latitude, this.state.region.longitude).then(
  (json) => {
    var address_component = json.results[0].address_components[0];

    this.ride_channel.trigger('client-found-driver', {
      driver: {
        name: 'John Smith'
      },
      location: {
        name: address_component.long_name,
        latitude: this.state.region.latitude,
        longitude: this.state.region.longitude,
        accuracy: this.state.accuracy
      }
    });

  },
  (error) => {
    console.log('err geocoding: ', error);
  }
);
```

Рисунок 11 – ответ пассажиру

После посадки пассажира у водителя срабатывает событие, что он в пути к месту назначения и пользовательский интерфейс для приложения водителя отображает только карту и маркеры для водителя и пассажира (рис.12).

```
render() {
  return (
    <View style={styles.container}>
      {
        this.state.region &&
        <MapView
          style={styles.map}
          region={this.state.region}
        >
            <MapView.Marker
              coordinate={{
              latitude: this.state.region.latitude,
              longitude: this.state.region.longitude}}
              title={"You're here"}
            />
            {
              this.state.passenger && !this.state.has_ridden &&
              <MapView.Marker
                coordinate={{
                latitude: this.state.passenger.pickup.latitude,
                longitude: this.state.passenger.pickup.longitude}}
                title={"Your passenger is here"}
                pinColor={"#4CD800"}
              />
            }
        </MapView>
      }
    </View>
  );
}
// next: add code when component unmounts
```

Рисунок 12 – интерфейс водителя.

На этом создание приложения для водителя закончено, но осталось приложение для пассажиров и оно немного похоже на приложение для водителя, но с некоторыми изменениями.

Для начала создаем новое приложение и установить те же самые библиотеки, что и у водителя, но добавив еще 2.

Две другие библиотеки - это «Google Place Picker» и «Loading Spinner Overlay».

Далее, для работы Google Place Picker также требуется дополнительная настройка. Во-первых, откроем «android/app/src/main/java/com/grabClone/MainApplication.java» файл и добавим следующее ниже последнего импорта (рис.13).

```
import com.reactlibrary.RNGooglePlacePickerPackage;


protected List<ReactPackage> getPackages() {
  return Arrays.<ReactPackage>asList(
      new MainReactPackage(),
      new MapsPackage(),
      new RNGooglePlacePickerPackage()
  );
}
```

Рисунок 13 – добавление библиотек

Добавляем все те же зависимости и установки, что и приложения водителя и начинаем добавлять код для пассажирского приложения.

Создаем новую директорию и открываем «index.android.js» файл и добавляем следующее (рис.14).

```
import { AppRegistry } from 'react-native';
import App from './App';
AppRegistry.registerComponent('grabClone', () => App);
```

Рисунок 14 – импорт файлов

Так же, как приложение водителя, приложение пассажира также использует «App.js» в качестве основного компонента. Импортируем библиотеки. Он также использует тот же «helpers.js» файл, можем его скопировать из того приложения (рис.15).

```
import React, { Component } from 'react';
import { StyleSheet, Text, View, Button, Alert } from 'react-native';

import Pusher from 'pusher-js/react-native';
import RNGooglePlacePicker from 'react-native-google-place-picker';
import Geocoder from 'react-native-geocoding';
import MapView from 'react-native-maps';
import Spinner from 'react-native-loading-spinner-overlay';

import { regionFrom, getLatLonDiffInMeters } from './helpers';

Geocoder.setApiKey('YOUR GOOGLE SERVER API KEY');



export default class App extends Component {
  state = {
    location: null,
    error: null,
    has_ride: false,
    destination: null,
    driver: null,
    origin: null,
    is_searching: false,
    has_ridden: false
  };

}
```

Рисунок 15 – Импорт библиотек

Для простоты объявим имя пользователя пассажира в конструкторе. Также инициализируем каналы Pusher (рис.16).

```
constructor() {
  super();
  this.username = 'wernancheta';
  this.available_drivers_channel = null;
  this.user_ride_channel = null;
  this.bookRide = this.bookRide.bind(this);
}
```

Рисунок 16 – объявление переменных

Функция «bookRide()» запускается, когда пользователь нажимает на кнопку «Заказать поездку». Это открывает окно выбора места, которое позволяет пользователю выбрать пункт назначения. После выбора местоположения приложение отправляет запрос на поездку всем водителям. Это вызывает предупреждение для отображения в приложении водителя, которое спрашивает, хочет ли водитель принять запрос или нет. В этот момент загрузчик будет продолжать вращаться, пока водитель не примет запрос (рис.17).

```
bookRide() {

  RNGooglePlacePicker.show((response) => {
    if(response.didCancel){
      console.log('User cancelled GooglePlacePicker');
    }else if(response.error){
      console.log('GooglePlacePicker Error: ', response.error);
    }else{
      this.setState({
        is_searching: true,
        destination: response
      });

      let pickup_data = {
        name: this.state.origin.name,
        latitude: this.state.location.latitude,
        longitude: this.state.location.longitude
      };

      let dropoff_data = {
        name: response.name,
        latitude: response.latitude,
        longitude: response.longitude
      };

      this.available_drivers_channel.trigger('client-driver-request', {
        username: this.username,
        pickup: pickup_data,
        dropoff: dropoff_data
      });

    }
  });
}
```

Рисунок 17 – вызов функции

«_setCurrentLocation()» функция получает текущее местоположение пассажира. Обратите внимание, что здесь мы используем «getCurrentPosition()» в отличие от «watchPosition()» который мы использовали в приложении драйвера ранее. Единственная разница между ними состоит в том, что «getCurrentPosition()» получает место только один раз (рис.18).

```
_setCurrentLocation() {

  navigator.geolocation.getCurrentPosition(
    (position) => {
      var region = regionFrom(
        position.coords.latitude,
        position.coords.longitude,
        position.coords.accuracy
      );


      Geocoder.getFromLatLng(position.coords.latitude, position.coords.longitude).then(
        (json) => {
          var address_component = json.results[0].address_components[0];

          this.setState({
            origin: {
              name: address_component.long_name,
              latitude: position.coords.latitude,
              longitude: position.coords.longitude
            },
            location: region,
            destination: null,
            has_ride: false,
            has_ridden: false,
            driver: null
          });

        },
        (error) => {
          console.log('err geocoding: ', error);
        }
      );

    },
    (error) => this.setState({ error: error.message }),
    { enableHighAccuracy: false, timeout: 10000, maximumAge: 3000 },
  );
}
```

Рисунок 18 – определение местоположения

После принятия заказа водителем пассажир может прослушивать изменения местоположения у водителя. Просто обновляем интерфейс каждый раз, когда запускается это событие (рис.19).

```
this.user_ride_channel.bind('client-driver-location', (data) => {
  let region = regionFrom(
    data.latitude,
    data.longitude,
    data.accuracy
  );


  this.setState({
    location: region,
    driver: {
      latitude: data.latitude,
      longitude: data.longitude
    }
  });

});
```

Рисунок 19 – прослушивание местоположения водителя

Пользовательский интерфейс состоит из загрузочного счетчика (отображается только в том случае, если приложение ищет водителя), заголовка, кнопки для бронирования поездки, местоположения пассажира и пункта назначения, а также карта, которая первоначально отображает текущее местоположение пользователя, а затем отображает текущее местоположение водителя, после того, как поездка была забронирована (рис.20).

```
render() {

  return (
    <View style={styles.container}>
      <Spinner
          visible={this.state.is_searching}
          textContent={"Looking for drivers..."}
          textStyle={{color: '#FFF'}} />
      <View style={styles.header}>
        <Text style={styles.header_text}>GrabClone</Text>
      </View>
      {
        !this.state.has_ride &&
        <View style={styles.form_container}>
          <Button
            onPress={this.bookRide}
            title="Book a Ride"
            color="#103D50"
          />
        </View>
      }

      <View style={styles.map_container}>
      {
        this.state.origin && this.state.destination &&
        <View style={styles.origin_destination}>
          <Text style={styles.label}>Origin: </Text>
          <Text style={styles.text}>{this.state.origin.name}</Text>

          <Text style={styles.label}>Destination: </Text>
          <Text style={styles.text}>{this.state.destination.name}</Text>
        </View>
      }
      {
        this.state.location &&
        <MapView
          style={styles.map}
          region={this.state.location}
        >
          {
            this.state.origin && !this.state.has_ridden &&
            <MapView.Marker
              coordinate={{
              latitude: this.state.origin.latitude,
              longitude: this.state.origin.longitude}}
              title={"You're here"}
            />
          }

          {
            this.state.driver &&
            <MapView.Marker
              coordinate={{
              latitude: this.state.driver.latitude,
              longitude: this.state.driver.longitude}}
              title={"Your driver is here"}
              pinColor={"#4CDB00"}
            />
          }
        </MapView>
      }
      </View>
    </View>
  );
}
```

Рисунок 20 – интерфейс пассажира

Все, приложения готовы. В этой статье была реализованы два приложения связанные друг с другом, через серверную часть с помощью Pusher. Созданное приложение довольно голое. Были взяты только основные функции. Так же можно добавить большинство своих функций и свои стили, а так же использовать это в коммерческих реализациях.

Библиографический список

1. Винокуров А.С., Баженов Р.И. Разработка мобильного приложенияинформационного сайта для абитуриентов и первокурсников университета // Современные научные исследования и инновации. 2015. № 7-2 (51). С. 54-62
2. Бычковский Д.Ю., Абу-Абед Ф.Н., Хабаров А.Р., Карельская К.А. Разработка мобильного приложения онлайн-радио // Программные продукты и системы. 2016. №2 (114). С. 185-194
3. Аксенов К.В. Обзор современных средств для разработки мобильных приложений // Новые информационные технологии в автоматизированных системах. 2014. №17. С. 508-513

4. Ким В.Ю. Особенности разработки дизайна пользовательского интерфейса для мобильного приложения // Новые информационные технологии в автоматизированных системах. 2015. №18. С. 479-481
5. Романов А.А., Панченко Е.А., Винокуров И.В. Разработка мобильного Постулат. 2019. №5 ISSN 2414-4487 приложения для управления документами из облачных хранилищ // Символ науки. 2016. №3. С. 84-87
6. Ngaia E.W.T., Gunasekaran A. A review for mobile commerce research and applications // Decision Support Systems. 2007. №43 (1). С. 3–15.