УДК 004

Создание сервера такси для соединения данных клиентов и водителей с помощью React Native

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 серверной части для приложения такси. С помощью React Native и Pusher, созданное приложение похоже на популярные приложения похожие на Uber и Яндекс.Такси. Практическим результатом является серверная часть, которая сможет передавать данные от водителя клиенту и обратно
Ключевые слова: Такси, приложение, андроид

Creating a taxi server to connect customer and driver data with React Native

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 a server part for a taxi application. Using React Native and Pusher, the created application will look like popular applications similar to Uber and Yandex.Taxi. The practical result is a server part that can transfer data from the driver to the client and vice versa
Keywords: Taxi, app, android

Все больше и больше таксопарков переходят на заказ с помощью приложения в телефоне, ведь это гораздо удобнее, чем звонить оператору и сообщать свое местоположение и место назначение. Теперь достаточно выбрать эти места на карте в телефоне и ожидать подачи автомобиля, причем подачу можно отслеживать так же в приложении. В то же время и водителям гораздо легче выбрать себе заказ, теперь ни диспетчер сообщает ему заказ, а он сам в приложении специальном для водителей выбирает заказ, который подойдет ему больше всего, тем самым упрощая себе процесс и клиенту.

Цель статьи состоит в создании серверной части для возможности передачи данных между клиентами и водителем, что поможет облегчить возможность подачи автомобиля, отслеживание местоположения автомобиля и пассажира.

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.

React Native будет использоваться для создания приложения Android как для водителя, так и для пассажира. Pusher будет использоваться для связи между ними в режиме реального времени.

Приложение, которое будет создано, будет в основном иметь тот же поток, что и любое приложение для заказа поездки: пассажир заказывает поездку → приложение ищет водителя → водитель принимает запрос → водитель забирает пассажира → водитель едет к месту назначения → Пассажир платит водителю.

Сам принцип работы приложений будет следующим реализован следующим образом:

1. Приложение определяет местоположение пользователя и показывает его на карте (примечание: необходимо включить GPS).
2. В пассажирском приложении пользователь нажимает «Заказать поездку».

3. Откроется модальное окно, которое позволит пассажиру выбрать место, куда он хочет отправиться.
4. Приложение просит пассажира подтвердить его пункт назначения.
5. После подтверждения приложение отправляет запрос в приложение водителя, чтобы забрать пассажира. Анимация загрузки отображается, пока приложение ожидает, пока водитель примет запрос.
6. Приложение драйвера получает запрос. Отсюда водитель может принять или отклонить запрос.
7. После того, как водитель примет запрос, детали водителя отображаются в приложении для пассажиров.
8. Пассажирское приложение показывает текущее местоположение водителя на карте.
9. Как только водитель окажется в пределах 50 метров от местоположения пассажира, он увидит предупреждение о том, что водитель находится рядом.
10. Когда водитель находится в пределах 20 метров от местоположения пассажира, приложение для водителя отправляет в приложение для пассажира сообщение о том, что водитель уже почти на месте.
11. Взяв пассажира, водитель едет к месту назначения.
12. Когда водитель находится в пределах 20 метров от места назначения, приложение для водителя отправляет в приложение для пассажиров сообщение о том, что оно находится очень близко от места назначения.

В этот момент поездка заканчивается, и пассажир может заказать другую поездку. Водитель также может принять любой входящий запрос на поездку.

Для начала необходимо создать сервер авторизации. Здесь будут посылаться запросы от клиентского приложения. Сервер аутентификации позволяет Pusher узнать, действительно ли пользователь, пытающийся подключиться, является зарегистрированным пользователем приложения.

Начнем с установки зависимости (рис.1).

```
npm install --save express body-parser pusher
```

Рисунок 1 – установка зависимости

Затем создадим файл server.js и напишем код проверки авторизации (рис.2).

```
var express = require('express');
var bodyParser = require('body-parser');
var Pusher = require('pusher');

var app = express();
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

var pusher = new Pusher({
  appId: process.env.APP_ID,
  key: process.env.APP_KEY,
  secret:  process.env.APP_SECRET,
  cluster: process.env.APP_CLUSTER,
});

app.get('/', function(req, res){
    res.send('all is well...');
});

app.get("/pusher/auth", function(req, res) {
  var query = req.query;
  var socketId = query.socket_id;
  var channel = query.channel_name;
  var callback = query.callback;

  var auth = JSON.stringify(pusher.authenticate(socketId, channel));
  var cb = callback.replace(/\"/g,"") + "(" + auth + ");";

  res.set({
    "Content-Type": "application/javascript"
  });

  res.send(cb);
});

app.post('/pusher/auth', function(req, res) {
  var socketId = req.body.socket_id;
  var channel = req.body.channel_name;
  var auth = pusher.authenticate(socketId, channel);
  res.send(auth);
});

var port = process.env.PORT || 5000;
app.listen(port);
```

Рисунок 2 – Проверка авторизации

Чтобы проверить является ли пользователь зарегистрированным в базе данных, необходимо проверить существует ли имя пользователя (рис.3).

```
var users = ['Фамилия', 'Имя', 'Никнейм'];
var username = req.body.username;

if(users.indexOf(username) !== -1){
  var socketId = req.body.socket_id;
  var channel = req.body.channel_name;
  var auth = pusher.authenticate(socketId, channel);
  res.send(auth);
}
```

Рисунок 3 – Проверка пользователя

Поскольку Pusher придется подключаться к серверу авторизации, он должен быть доступен из Интернета.

Можно использовать «now.sh» для развертывания сервера авторизации. Устанавливается он с помощью команды «npm install now»

После установки нужно перейти в папку, где лежит server.js и выполнить «now», будет предложено ввести адрес электронной почты и подтвердить учетную запись.

После проверки учетной записи выполним следующее, чтобы добавить параметры приложения Pusher в качестве переменных среды в учетную запись «now.sh», чтобы можно было использовать ее изнутри сервера (рис.4).

```
now secret add pusher_app_id YOUR_PUSHER_APP_ID
now secret add pusher_app_key YOUR_PUSHER_APP_KEY
now secret add pusher_app_secret YOUR_PUSHER_APP_SECRET
now secret add pusher_app_cluster YOUR_PUSHER_APP_CLUSTER
```

Рисунок 4 – добавление ключей приложения

После этого разворачиваем сервер использовав секретные значения. Далее необходимо будет получить доступ к настройкам Pusher изнутри сервера с помощью команды «process.env.APP_ID». В ответ мы получим URL-адрес развертывания, который возвращает «now.sh».

URL-адрес, который будет использоваться позже для подключения приложения к серверу авторизации.

Реализация сервера для приложения такси закончена, следующим этапом будет реализация приложения клиента и приложения водителя.

Создание серверной части не составляет труда, если есть некоторое понимание как пользоваться node.js, знать как пользоваться React Notive и т.д.

Практическим результатом в даной статье была реализация сервера и развертывание его на localhost, где сервер выполняет роль хранения и передачи данных от клиента к водителю.

Библиографический список

1. Винокуров А.С., Баженов Р.И. Разработка мобильного приложения информационного 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.