Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____2018г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____А.А.Замышляева
«____»_____2018 г.

Разработка веб-сайта для агрегации предложений услуг такси в городе Челябинск

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–09.03.04.2018.66.ПЗ ВКР

Руководитель работы, к.ф.-м.н., доцент
_____/С.М. Елсаков
«____»_____2018 г.
Автор работы
Студент группы ЕТ-414
_____/ М.А Назаретян
«____»_____2018 г.
Нормоконтролер, к.т.н., доцент
_____/Т.Ю. Оленчикова
«____»_____2018 г.

Челябинск 2018

АННОТАЦИЯ

Назаретян М. А. Разработка веб-сайта для агрегации предложений услуг такси в городе Челябинск. – Челябинск: ЮУрГУ, ММиКН-414, 45 с., 12 табл., 33 ил., библиогр. список – 19 наим., 2 прил.

Данная работа посвящена разработке веб-сайта для агрегации предложений услуг такси в городе Челябинск.

В работе выполнен обзор онлайн такси в городе Челябинск, были выявлены протоколы взаимодействия веб-сайта с их интерфейсами. Были разобраны аналоги агрегаторов такси.

Спроектирована и разработана архитектура системы, включающая в себя диаграмму вариантов использования, диаграмму компонентов и общей архитектуры системы. Выполнен анализ предметной области.

Реализован веб-сайт для агрегации предложений услуг такси.

Программа реализована на языке программирования JavaScript на платформе Node.js. В приложениях приведены описание и текст программы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящее время услугами такси пользуется большой процент населения. По данным DISCOVERY Research Group [1] каждый месяц такси перевозит более 45 миллионов пассажиров, исключая из этой цифры пассажиров, которые используют такси не присоединённое к какому-то таксопарку. Объем рынка такси в России в 2016 г. увеличился на 14,3 млрд. руб. (+4,4%) по сравнению с 2015 г. и достиг 339 млрд. руб. (для сравнения в 2015 г. - 324,7 млрд. руб.). В I полугодии 2017 г. он составил 176,3 млрд. руб. При этом доля агрегаторов такси заняла всего 5%, а диспетчерских, таких как Maxim и Rutaxi - 18%, по 11% и 7% соответственно [2]. Однако по данным исследования аналитического центра при правительстве Российской федерации [3], опубликованного в феврале 2018 общая доля рынка, которая приходится на агрегаторы поднялась до отметки в 32.8%. На Яндекс Такси и Uber приходится 10.4%, Gett – 2.5%, такси MAXIM – 7.6%, а Rutaxi, которая известна под маркой Лидер или Везет – 12.3%.

Такой рост обусловлен тем, что такси более комфортабельное средство передвижения нежели общественный транспорт, стоимость услуги является доступной для любого. Доступность заключается в том, что крупные конкурирующие организации пытаются привлечь к себе покупателя. В итоге имеется множество такси с разными ценами в разный момент времени и каждое такси имеет отдельные тарифы: от эконом до бизнес класса.

Разновидность, различие в цене и удобстве услуг заставляют людей заходить в каждое приложение или веб-сайт и просматривать цену на каждый тариф, что получается очень неудобно и тратит очень много времени.

Большую популярность набирает приложение на Android и iOS, которое является агрегатором агрегаторов такси, которое позволяет узнать минимальную цену на услуги такси в момент времени. Но так как не каждый пользователь может себе позволить установить приложение на данные платформы, то появляется необходимость в написании сервиса, который бы собирал информацию о ценах на каждый тариф такси доступный в городе Челябинск.

1 ОБЗОР СУЩЕСТВУЮЩИХ АНАЛОГОВ АГРЕГАТОРОВ И ОНЛАЙН ТАКСИ

1.1 Анализ сайтов онлайн такси

Агрегаторы такси объединяют под своим началом множество разных таксопарков со своими водителями. Пользователь делает заказ на сайте агрегатора или в приложении, затем заказ передается водителям, которые ближе всего находятся к месту отправления. После того, как поездка будет завершена агрегатор забирает свой процент с поездки вместе с таксопарком.

Обращаясь к данным, предоставленными Аналитическим Центром при Российской Федерации [2], в которых были рассмотрены самые популярные и крупные агрегаторы такси, такие как Яндекс Такси, такси Gett, такси Uber, такси Максим и Rutaxi или Лидер. Вышеперечисленные такси являются основными перевозчикам в городе Челябинск. Именно по этой причине далее будут рассмотрены данные такси.

1.1.1 Яндекс Такси

Яндекс Такси [4] – это сервис, в котором можно онлайн заказать такси. Является одним из самых крупным и популярных агрегаторов служб такси в России [3]. Заказ передается водителю, который находится ближе других остальных по местоположению и обстановке на дороге.

Включает в себя разные виды тарифов, самым дешевым является эконом тариф, а самый престижным – бизнес класс. Так же имеет возможность заказа минивена, желтых регистрационных номеров, детских мест, возможность перевозки животных, а также некурящих водителей.

Вызов выполняется с помощью сайта или приложения. При заказе такси пользователь указывает номер телефона, адрес отравления и места назначения, возможные точки маршрута, тариф, форму оплаты и дополнительные опции, о которых говорилось выше.

На рисунке 1.1 продемонстрирован онлайн заказ на сайте Яндекс такси.

При вводе адресов назначения и отправления мы получаем информацию о стоимости поездки.

На сайте есть функцию демо-заказа, которая позволяет просмотреть путь от ближайшего такси до места отправления, а затем и до места назначения. Так же сервис не имеет никакой документации и возможности получения ключа API.

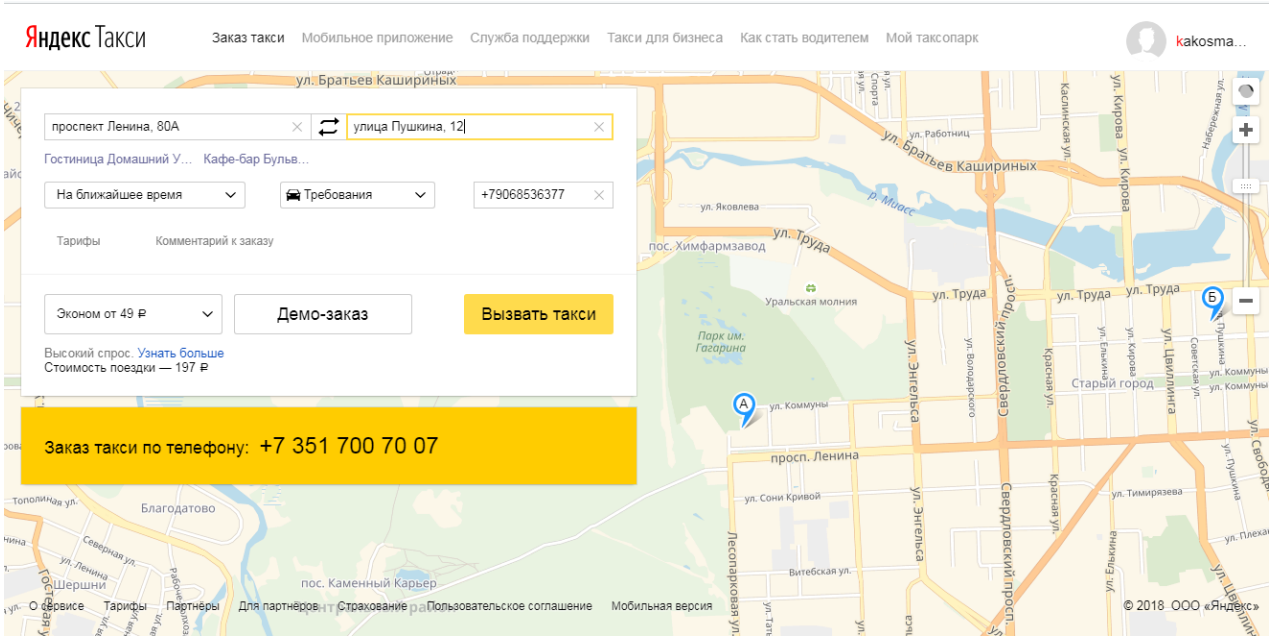


Рисунок 1.1 – Онлайн заказ на сайте Яндекс такси

При онлайн заказе на сайте данные введенные в строку передаются на https://taxi.yandex.ru/geosearch/, где ответом является широта и долгота точки, после ввода обоих точек создает запрос на https://taxi.yandex.ru/3.0/routestats/, с уникальным идентификатором посещения страницы – id, широту и долготу адреса отправления, широту и долготу адреса назначения – route[широта, долгота][широта, долгота], флаг учитывающий трафик - supports_forced_surge, а также cookie, в которой присутствует все данные о пользователе, а также cookie, без которых запрос не возможен: _id – уникальный идентификатор посещения страницы и yandexuid – пользовательский идентификатор. Все формируется в формате JSON. Ответом на запрос является дистанция маршрута, время, которое понадобится для преодоления маршрута, и имя тарифа с его ценой.

1.1.2 Gett

Такси GETT [5] – сервис онлайн заказа такси, который распространен в более, чем ста городах России, Великобритании, Нью-Йорка и, соответственно, там, где был создан – в Израиле. Суть компании в том, что она не имеет своих такси, а позволяет лицензионным таксопаркам получать заказы из мобильных приложений на iOS и Android. Заказать онлайн на сайте такси – невозможно, для этого необходимо установить на телефон приложение.

Чтобы получать информацию по заявкам нужно обращаться к API, но такси имеет закрытое API. Ключ можно получить, написав письмо, но, если верить поддержке, этот ключ не выдается никому.

1.1.3 Такси Uber

Такси Uber [6] – американская международная компания, которая представляет собой платформу, которая при помощи приложения для смартфона

соединяет водителей-партнеров с клиентами. Пользователь может отслеживать перемещение зарезервированного такси до места отправления с помощью приложения. Для заказа такси требуется регистрация. Оплата производится по банковской карте, которая привязана к учетной записи, либо наличными.

Имеет множество видов услуг, от бюджет версии до доставки пиццы, но в России, в частности в Челябинске действуют UberStart (бюджет) и UberX (комфорт). Имеет открытое API, что позволяет получать информацию.

Так же на сайте есть возможность просмотреть примерную стоимость поездки (см. рисунок 1.2), но заказать такси можно только через мобильное приложение.

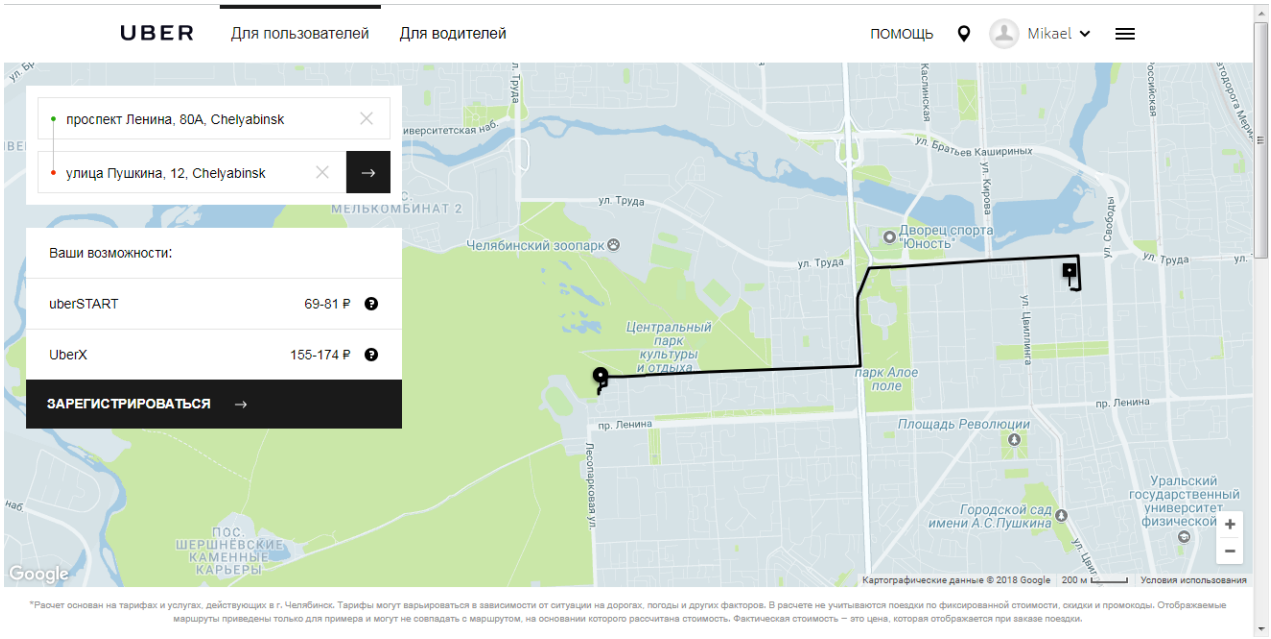


Рисунок 1.2 – Расчет стоимости поездки в Uber

1.1.4 Такси Максим

Такси Максим [7] – популярный сервис такси в России, который был создан в Кургане. Распространен и в других странах, например, Италии. Также, как и прошлые такси, эта служба заказа такси без собственного таксопарка и своих машин. Заказ можно произвести онлайн на сайте, а также и в приложении для Android и iOS. Доступно обширное количество настроек (см. рисунок 1.3) для заказа такси, такие как: тариф, способ оплаты, можно даже сделать предварительный заказ, есть система чаевых и даже можно подзарядить аккумулятор, если ваш сел.

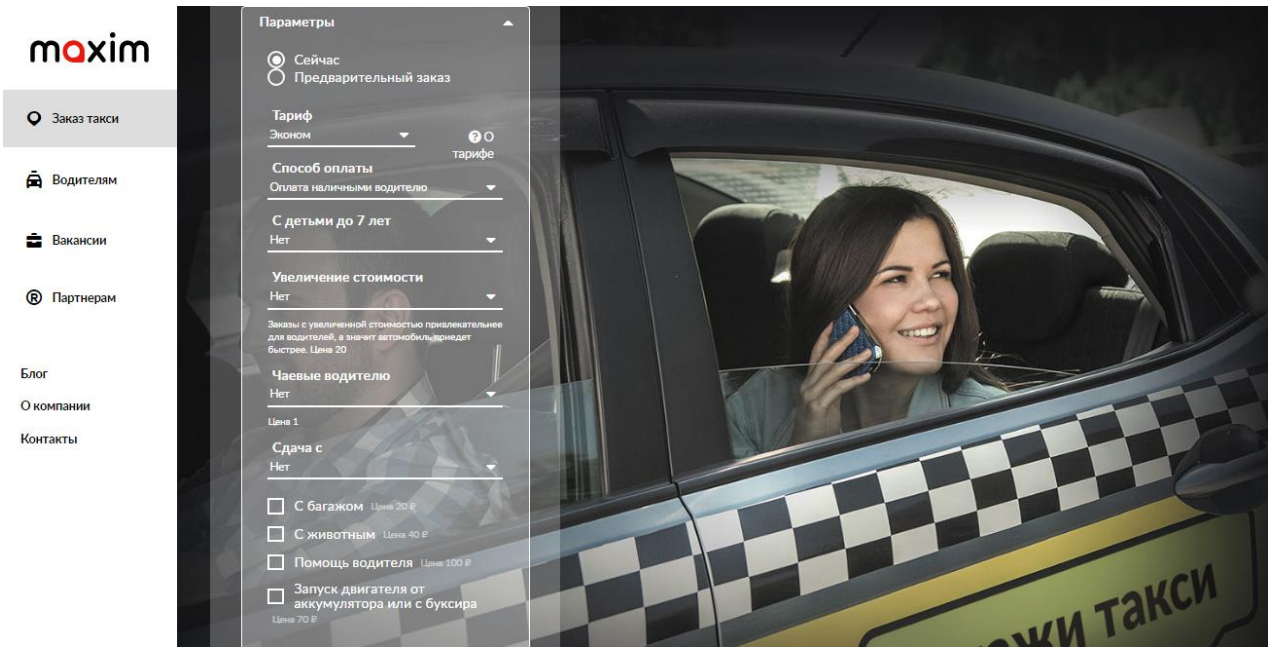


Рисунок 1.3 – Дополнительные услуги в такси Максим

При заказе такси пользователь указывает свой номер, место отправления и место назначения. Также, что выделяет его из других такси то, что нужно подтвердить себя отправив им код. Есть возможность просмотреть свои прошлые заказы и не вводить данные заново, если вы, например, каждый день пользуетесь им с одинаковыми параметрами. На рисунке 1.4 изображено меню расчета стоимости онлайн.

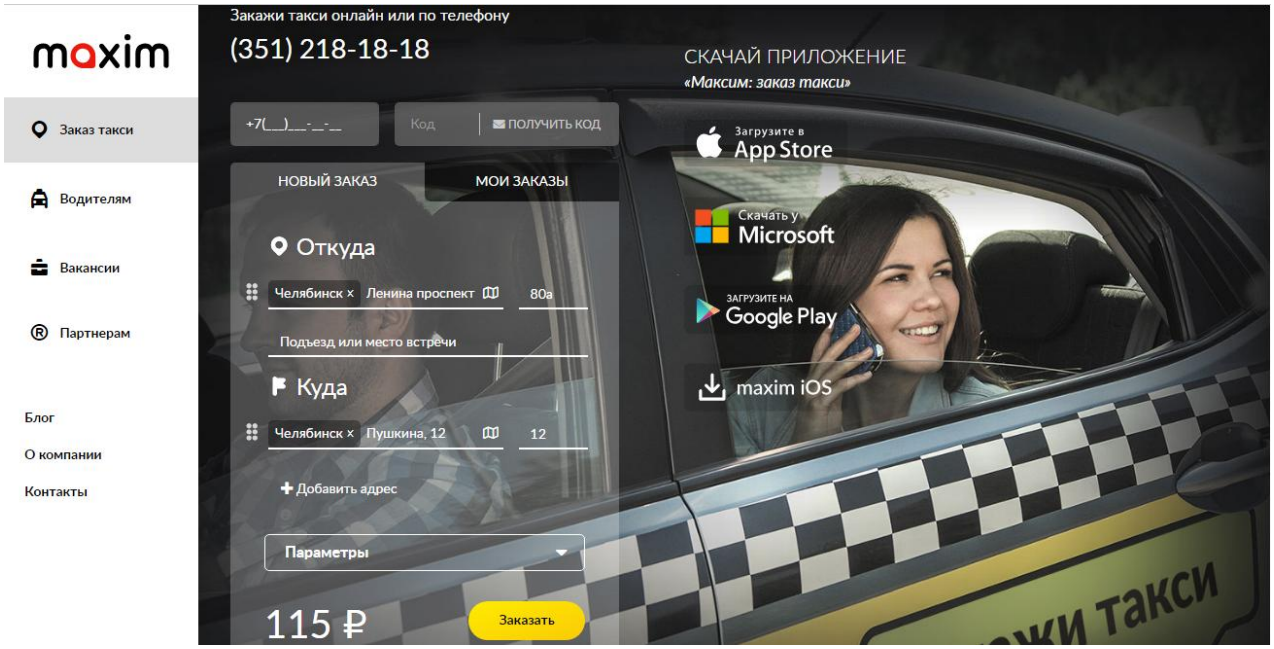


Рисунок 1.4 – Расчет стоимости цени и заказ на сайте такси Максим

При неправильном вводе выдает ошибку в виде JSON (см. листинг 1.1):

```
{
	"success":true,
	"errors":
	{
		"orderform-tariffid":["Необходимо заполнить «Тариф»."],
		"addressform-0-pointfield":["Необходимо заполнить «Улица или место»."]
	}
}
```

Листинг 1.1. – Ответ при пустом или неправильном запросе

Сервис не предоставляет свое API. На сайте после набора данных, введённые данные отправляются на сервер, сервер отправляет обратно местоположение (долгота и широта), после этого все эти данные формируются вместе с тарифом. Ответ приходит в формате HTML.

1.1.5 Rutaxi

Rutaxi (Лидер) [8] – сеть диспетчерских, распространившаяся по всей России, которая сотрудничает с частными водителями, а также с частными таксопарками, которая в разных городах имеет разное название. Есть возможность заказать такси онлайн на сайте, по телефону, смс, либо же так же скачать приложение. При заказе с сайта присутствует скидка 25%. На рисунке 1.5 изображена главная страница заказа такси на сайте.

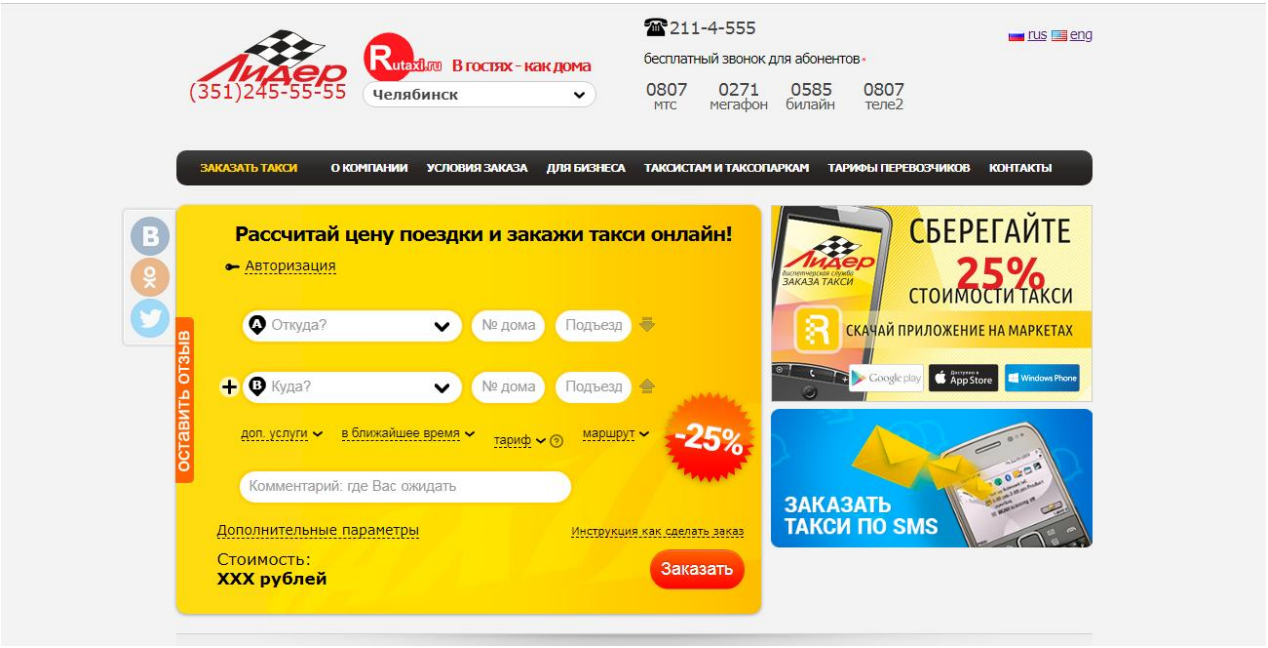


Рисунок 1.5 – Заказ такси на сайте Rutaxi

Цена расчитывается при вводе пользователем места отправления и места назначения. Имеет два тарифа, дополнительные услуги. Цена считается сразу со скидкой.

У сервиса присутствует возможность получения любой информации по API. Для этого нужно зарегистрироваться, добавить свое приложение и получить уникальный ключ идентификатора. Этот ключ используется для получения информации по API. Чтобы расчитать стоимость поездки нужно предоставить Id улицы, дом, также id тарифа. В ответе мы получаем статус запроса и цену (см. рисунок 1.6).

```
HTTP/1.1 200 OK
{
   "success": true,
   "time": "2016-12-20 20:33:44",
   "data": {
     "cost": "550",
     "status": 220
   }
}
```

Рисунок 1.6 – Ответ на запрос цены

1.2 Обзор существующих аналогов агрегаторов такси

1.2.1 Веб-Сайт “СравниТакси”

Сервис предоставляет информацию о стоимости услуг онлайн такси. В базе такси содержатся крупнейшие онлайн такси: Uber, Gett.

На сайте данные для Uber берутся через их API, данные для Gett считаются по их тарифам с использованием данных по времени поездки из Google.Maps API.

Начальная страница (см. рисунок 1.7) имеет два поля ввода, кнопки определения местоположения, которые не работают, кнопку вывода и окошко информации.

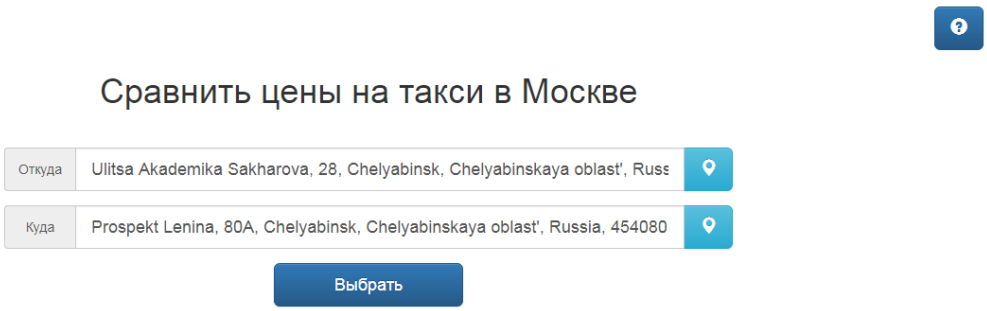
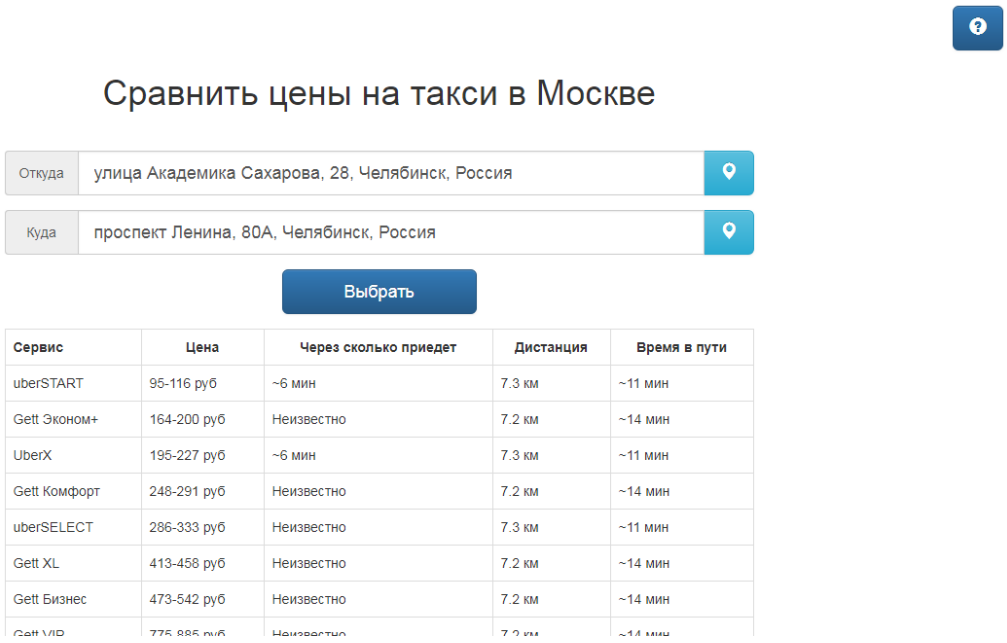


Рисунок 1.7 – Сайт сравни такси с сохраненным поиском

На рисунке 1.8 изображен пример работы сайта. При нажатии кнопки “выбрать” выводит таблицу результатов:



Сервис	Цена	Через сколько приедет	Дистанция	Время в пути
uberSTART	95-116 руб	~6 мин	7.3 км	~11 мин
Gett Эконом+	164-200 руб	Неизвестно	7.2 км	~14 мин
UberX	195-227 руб	~6 мин	7.3 км	~11 мин
Gett Комфорт	248-291 руб	Неизвестно	7.2 км	~14 мин
uberSELECT	286-333 руб	Неизвестно	7.3 км	~11 мин
Gett XL	413-458 руб	Неизвестно	7.2 км	~14 мин
Gett Бизнес	473-542 руб	Неизвестно	7.2 км	~14 мин
Gett VIP	775-885 руб	Неизвестно	7.2 км	~14 мин

Рисунок 1.8 – Пример работы сайта “СравниТакси”

При тестировании данного сайта было замечено только одно достоинство: сайт позволяет сохранять прошлый поиск и при новой сессии адреса останутся на месте.

Недостатки, которые можно выделить:

- кнопки определения местоположения не работают;
- нет обратной связи;
- заточен только под Москву;
- цены на такси не совпадают с реальными;

- ссылка на расчет выдает ошибки(OVER_QUERY_LIMIT). Даже если ошибки не бывает, никакой расчет произвести невозможно, просто открывается та же страница, что и была.

Других аналогов веб-сервисов агрегации услуг такси не было найдено. Однако было найдено приложение, называемое СравниТакси, которое можно установить только на телефоны Android и iOS.

1.2.2 Приложение для смартфонов “СравниТакси”

При запуске приложения требуется скачать приложение “Максим Такси”. На рисунке 1.9 проиллюстрирована работа приложения.

Какие такси включает в себя само приложение: UrentCar, BelkaCar, Поехали!, Uber, TimCar, Ситимобил, Rutaxi (Везет), Такси777, а также Яндекс Такси. Информация парсится с помощью API, приложение работает в России, Белоруссии, Украине и Казахстане. После выбора перевозчика пользователь перенаправляется на его приложение. [9]

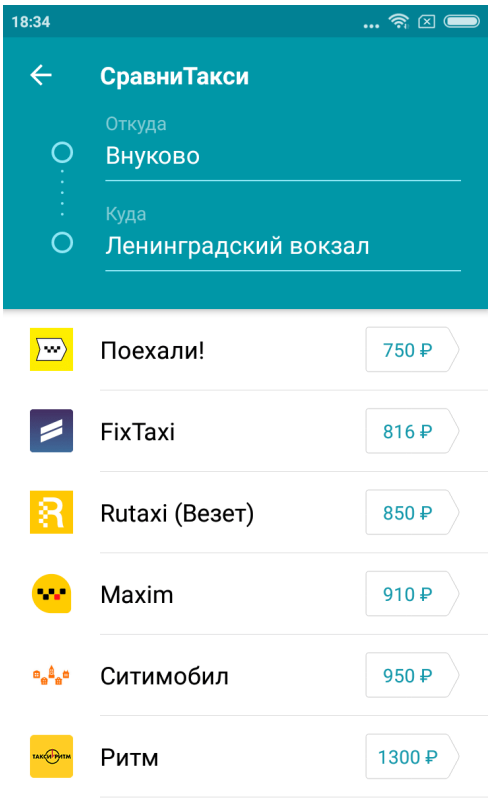


Рисунок 1.9 – Работа приложения “СравниТакси” на смартфоне с системой Android

Из достоинств можно выделить:

- сравнивает цены на поездку по выбранному маршруту в Yandex.Taxi, Uber, Fix Taxi, Gett, Rutaxi и такси Максим;

- есть возможность посмотреть стоимость аренды авто в сервисах BelkaCar и URentCar;
- автоматически определяет текущее местоположение пользователя;
- работает во многих городах России и СНГ.

Недостатки приложения состоят в том, что, во-первых, для того, чтобы сравнить цены на такси по заданному маршруту и дальнейшего заказа, нужно скачать приложения – партнеры. Во-вторых, функция автоматического определения месторасположения пользователя, по крайней мере в версии для Android, работает не слишком точно.

1.3 Анализ и выбор платформы для разработки

1.3.1 JavaScript

JavaScript – это язык программирования, который имеет полную интеграцию с html и css, и даёт решать большой спектр задач как в frontend, так и backend разработке [10].

JavaScript наряду с HTML и CSS является одной из трех основных технологий World Wide Web. JavaScript позволяет создавать интерактивные веб-страницы и, таким образом, является неотъемлемой частью веб-приложений. Подавляющее большинство веб-сайтов используют его, и у всех основных веб-браузеров есть специальный механизм JavaScript для его выполнения.

В качестве языка JavaScript поддерживает основанные на событиях, функциональные и императивные (в том числе объектно-ориентированные и прототипные) стили программирования. Он имеет API для работы с текстом, массивами, датами, регулярными выражениями и базовыми манипуляциями с DOM, но сам язык не включает никаких операций ввода-вывода, таких как сети, хранилища или графические объекты, полагаясь на них в хост-средах, в которых они встроены.

Загрузка нового содержимого страницы или отправка данных на сервер через Ajax без перезагрузки страницы. Интерактивный контент, например, игры, а также воспроизведение аудио и видео. Проверка входных значений веб-формы для обеспечения их приемлемости перед отправкой на сервер.

Передача информации о пользовательских предпочтениях и просмотре активности пользователей на различных веб-сайтах. Веб-страницы часто делают это для веб-аналитики, отслеживания объявлений, персонализации или других целей.

Поскольку код JavaScript может выполняться локально в браузере пользователя (а не на удаленном сервере), браузер может быстро реагировать на действия пользователя, делая приложение более отзывчивым. Кроме того, код JavaScript может обнаруживать действия пользователя, которые не могут быть использованы только для HTML, например, индивидуальные нажатия клавиш.

Основными преимуществами языка программирования JavaScript, благодаря которым он конкурирует с остальными, являются:

- максимально простой и понятный для пользователя;
- имеет поддержку всеми возможными популярными браузерами без дополнительных установок и библиотек;
- скорость работы JavaScript очень высокая; скрипты (программы, написанные на языке JavaScript) подключаются к HTML коду web-страницы напрямую и при загрузке сразу же выполняются; программы можно запускать не только в браузере, но и на сервере.

1.3.2 PHP

PHP – исполняемый на стороне веб-сервера язык программирования, спроектированный Расмусом Лердорфом (Rasmus Lerdorf) в качестве инструмента создания динамических и интерактивных веб-сайтов [11].

PHP-код может быть встроен в HTML-код или его можно использовать в сочетании с различными системами веб-шаблонов, системами управления контентом и фреймворками. PHP-код обычно обрабатывается интерпретатором PHP, реализованным как модуль на веб-сервере или в качестве исполняемого файла Common Gateway Interface (CGI). Веб-сервер объединяет результаты интерпретируемого и исполняемого PHP-кода, который может быть любым типом данных, включая изображения, с созданной веб-страницей. PHP-код также может быть выполнен с помощью интерфейса командной строки (CLI) и может использоваться для реализации автономных графических приложений.

Достоинства языка программирования.

- Низкий барьер входа: работа с PHP довольно проста. Создание веб-страниц почти слишком просто. Это связано с его истоками как инструментом для создания личных домашних страниц и интерпретационных форм.
- Огромная экосистема: экосистема вокруг PHP огромна. Многое из-за популярности WordPress и Magento. Есть десятки бесплатных и платных онлайн-сайтов обучения для PHP, а также локальных учебных мест по всему миру.
- Большое сообщество разработчиков с открытым исходным кодом.
- Множество подключаемых платформ: с появлением группы PHP позволяет писать код для одного фреймворка, который легче переносится на другой.
- Инструменты автоматизации. Существует довольно хорошая система средств автоматизации, доступных для тестирования и развертывания приложений PHP, написанных на PHP.
- Разворачивается практически на любом сервере.

Из недостатков выделяются.

- Глобальные параметры конфигурации могут изменять семантику языка, усложняя развертывание и переносимость.

- Много форм специального назначения, но недостаточно общих форм.
- Скорость работы оставляет желать лучшего. Разработка веб-сайта с PHP Web Development довольно медленная, чем по сравнению с другими. Фреймворки PHP добавляют некоторую сложность и, следовательно, могут столкнуться с более медленной скоростью выполнения.
- Проблема безопасности. PHP - это язык с открытым исходным кодом, поэтому, при разработке веб-приложения для клиента, пользователи могут видеть исходный код. Это означает, что PHP не обладает фактором безопасности, поэтому может иметь недостатки на вашем сайте.

1.3.3 ASP.NET

ASP.NET является серверной платформой веб-приложений с открытым исходным кодом, предназначенной для веб-разработки для создания динамических веб-страниц [12]. Он был разработан Microsoft, чтобы позволить программистам создавать динамические веб-сайты, веб-приложения и веб-службы. Он был впервые выпущен в январе 2002 года с версией 1.0 .NET Framework и является преемником технологии Microsoft Active Server Pages (ASP). ASP.NET построен в среде Common Language Runtime (CLR), позволяя программистам писать код ASP.NET с использованием любого поддерживаемого языка .NET. Расширения ASP.NET SOAP позволяет компонентам ASP.NET обрабатывать SOAP-сообщения.
Достоинства:

- компилируемый код выполняется быстрее, большинство ошибок отлавливается ещё на стадии разработки;
- возможность кэшировать всю страницу или только ее части для повышения производительности;
- возможность использовать модель разработки кода для разделения бизнес-логики от представления;
- возможность использования истинного объектно-ориентированного дизайна для программирования страниц и элементов управления.

Недостатки:

- автоматически созданный HTML не обеспечивает полный контроль разработчикам;
- управление идентификацией HTML скомпрометировано и затрудняет использование клиентских языков, таких как JQuery;
- Visual Studio огромна по сравнению с другими IDE;
- не полная документация. Большинство вопросов охвачены, но при работе с ними могут быть проблемы.

1.3.4 Node.js

Node.js – это кросс-платформенная среда с открытым исходным кодом, которая выполняет JavaScript-код на стороне сервера [12]. Исторически JavaScript использовался прежде всего для клиентских сценариев, в которых скрипты, написанные на JavaScript, встроены в HTML-страницу веб-страницы и запускаются клиентской стороной с помощью механизма JavaScript в веб-браузере пользователя. Node.js позволяет разработчикам использовать JavaScript для серверных скриптовых сценариев на стороне сервера для создания динамического содержимого веб-страницы до того, как страница будет отправлена в веб-браузер пользователя. Следовательно, Node.js представляет собой парадигму «JavaScript везде», унификацию разработки веб-приложений вокруг одного языка программирования, а не разные языки для сценариев на стороне сервера и на стороне клиента.

Node.js позволяет создавать веб-серверы и сетевые инструменты с использованием JavaScript и набор «модулей», которые обрабатывают различные основные функциональные возможности. Модули Node.js используют API, предназначенный для снижения сложности написания серверных приложений.

Разница между Node.js и PHP заключается в том, что большинство функций в блоке PHP до завершения (команды выполняются только после завершения предыдущих команд), тогда как функции Node.js не блокируются (команды выполняются одновременно или даже параллельно, и можно использовать обратные вызовы для завершения или отказа сигнала.)

Node.js запускает управляемое событиями программирование на веб-серверы, что позволяет разрабатывать быстрые веб-серверы в JavaScript. Разработчики могут создавать масштабируемые серверы без использования потоков, используя упрощенную модель программирования, управляемой событиями, которая использует обратные вызовы для сигнализации завершения работы. Node.js связывает легкость языка сценариев (JavaScript) с мощью сетевого программирования Unix.

Node.js был построен на JavaScript-движке Google V8. Кроме того, JavaScript был широко известным языком, что делает Node.js доступным для всего сообщества веб-разработчиков.

Достоинства, которые можно выделить [14]:

- асинхронное управляемое событиями IO помогает одновременную обработку запросов;
- использует JavaScript, который легко учить;
- распределяет один и тот же фрагмент кода как на стороне сервера, так и на стороне клиента;
- npm (Node project module), модули пакета Node.js уже стали огромными, и все еще растут;
- активное и яркое сообщество, с большим количеством кода, разделяемого через github;

- может обрабатывать тысячи одновременных соединений с минимальными накладными расходами на один процесс.

Недостатки:

- node.js не обеспечивает масштабируемость. Одного процессора не хватит; платформа не дает возможности масштабироваться, чтобы воспользоваться преимуществами нескольких ядер, обычно присутствующих в сегодняшнем аппаратном обеспечении серверного класса;
- сложность работы с реляционной базой данных;
- каждый раз, используя обратный вызов возвращает тонны вложенных обратных вызовов.

1.4 Постановка задачи

Целью дипломного проекта является разработка веб-сайта для агрегации услуг онлайн такси в г. Челябинск.

Данная система должна упростить процесс поиска такси пользователям, находящимся в городе Челябинск. Интерфейс системы должен работать на устройствах с любым разрешением экрана. Пользователь имеет возможность ввести пункт назначения и пункт отправления и выбрать нужный из выпадающего списка. При нажатии на кнопку должен появиться список из такси, их цены на данный момент времени, а также карту, на которой можно будет увидеть маршрут.

Система должна информировать пользователя об ошибках неправильного ввода адреса.

Для достижения поставленной цели необходимо решить следующие задачи:

- проанализировать онлайн такси в городе Челябинск и как можно получить информацию;
- проанализировать аналоги агрегаторов такси;
- изучить и проанализировать современные технологии и инструментарий для создания сайтов;
- разработать архитектуру системы;
- реализовать алгоритмы системы;
- протестировать систему.

Система должна располагаться на сервере с доступом к Интернету. Доступ к системе будет осуществляться через сайт с помощью браузеров последних версий с поддержкой ES2015 Javascript. На рисунке 1.10 изображены версии браузеров, которые поддерживают ES2015.

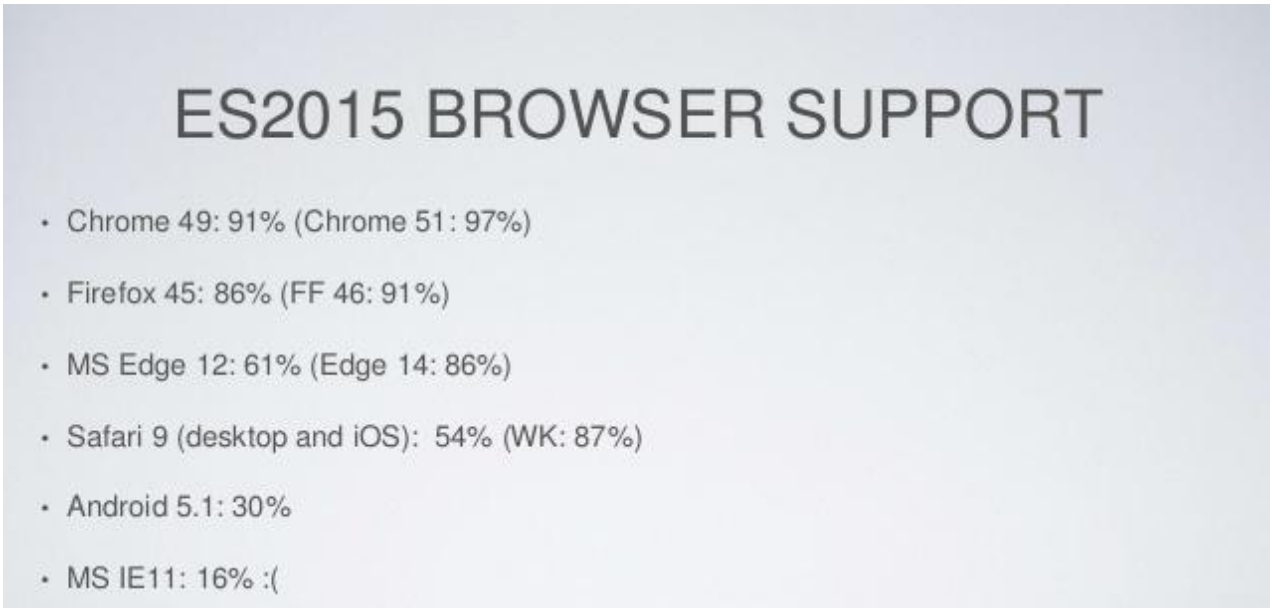


Рисунок 1.10 – Поддержка браузерами ES2015

1.5 Выводы по разделу

В этом разделе проанализированы всевозможные аналоги данного проекта, также рассмотрели сайты, на которых можно заказать такси онлайн. В итоге было принято решение по разработке веб-сайта так как аналоги имеют недостатки, которые не позволяют всем пользователям использовать данный продукт. Рассмотрев онлайн такси принято решение получать информацию с Yandex.Taxi, Rutaxi (Лидер), Такси Максим, а также Uber.

Было проведено сравнение языков разработки и платформ, в котором рассмотрели плюсы и минусы каждого. На основе этих недостатков и преимуществ было принято решение использовать Node.js для разработки.

2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА СИСТЕМЫ

Для построения диаграмм архитектуры используется язык графического описания для моделирования UML (Unified Modeling Language) [16]. В основном он используется для определения, визуализации, проектирования и документирования программных систем.

2.1 Разработка диаграммы использования

На рисунке 2.1 приведена диаграмма вариантов использования, раскрывающая возможности пользователя.

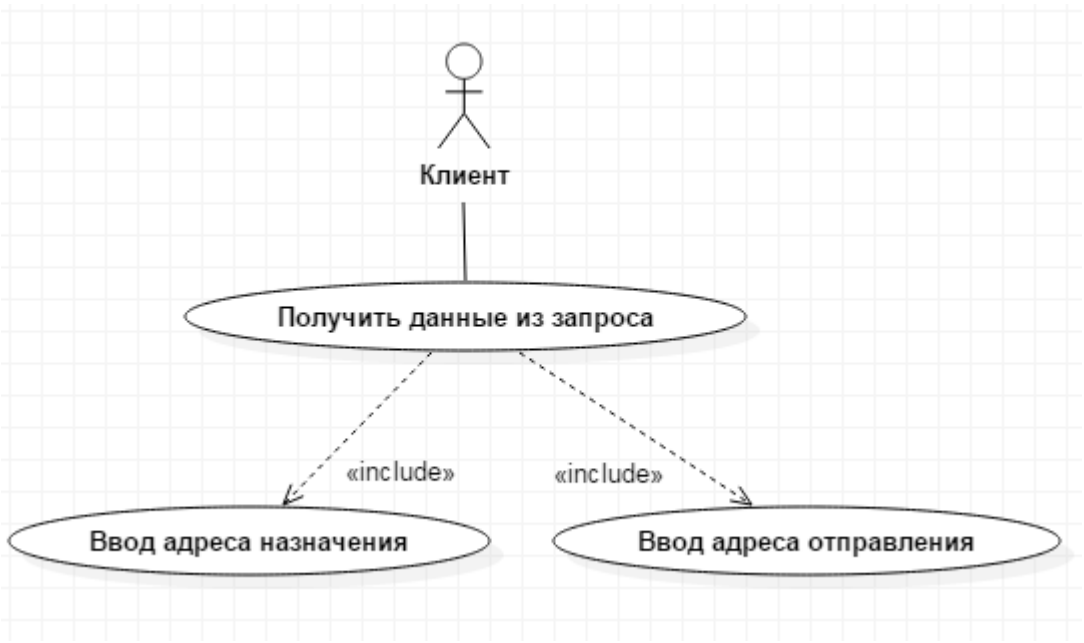


Рисунок 2.1 – Диаграмма использования для клиента

Пользователь получает ответ при вводе адреса отправления и адреса назначения из базы Google Maps Autocomplete, данные отправляются на сервер, с сервера получает геоданные с помощью Google Maps Geocoder, а затем все данные отправляются на сайты такси для формирования ответа.

2.2 Разработка архитектуры приложения

Рассмотрим три более подходящих шаблона проектирования архитектуры приложения [15].

- Шаблон MVC (Model – View – Controller) – обычно используется для разработки программного обеспечения, которое делит приложение на три взаимосвязанные части. Отделяет основные компоненты, что позволяет

эффективно использовать повторное использование кода и параллельную разработку. Первым главным компонентом является модель, она служит для предоставления данных, управляет логикой и правилами приложения независимо от пользовательского интерфейса, реагируя на команды контроллера. Контроллер следит за действиями пользователя, формирует их в команды и отправляет в модель. Представление же отвечает за отображение данных, получаемых от модуля.

- Шаблон MVP (Model – View – Presenter) – шаблон используемый в основном для создания пользовательских интерфейсов, облегчения модульного тестирования, а также разделения проблем в логики представления, где модель представляет собой интерфейс определяющий отображаемые данные. Представление (View) – представляет собой пассивный интерфейс, который отображает данные (модель) и направляет пользовательские данные представителю (Presenter). Представитель извлекает данные из модели и форматирует их для отображения в представлении (View).
- Шаблон MVVM (Model – View – ViewModel) – представляет собой преобразователь значений, где модель представления (ViewModel) отвечает за визуализацию (преобразование) объектов данных из модели таким образом, чтобы объекты легко управлялись и представлялись. В этом отношении модель представления (ViewModel) больше модель (Model), чем представление (View), и обрабатывает большинство логики отображения представления. Модель представления (ViewModel) может реализовывать шаблон посредника, организуя доступ к логике внутреннего блока вокруг множества вариантов использования, поддерживаемых представлением.

Из всех описанных шаблонов больше всего подходит шаблон MVC (см. рисунок 2.2).

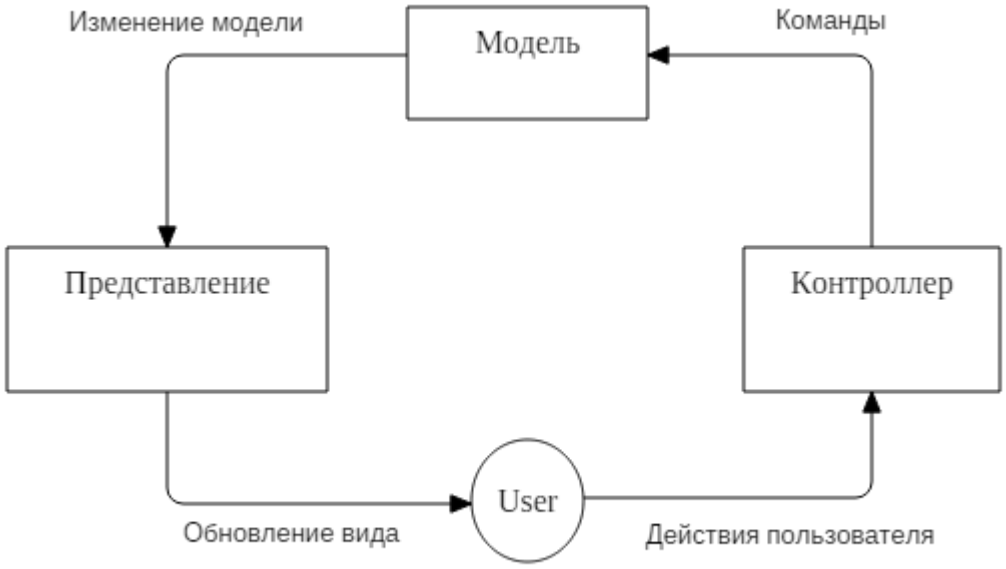


Рисунок 2.2 – Шаблон проектирования MVC

Основой является веб-сервер, описанный в среде Node.js, на которой у нас хранится и работает сайт. Контроллер принимает действия пользователя, которые поступают из браузера, отправляет на сервер данные, затем сервер передает изменения и меняет отображение сайта для пользователя.

2.3 Разработка диаграммы активности

Диаграмма деятельности, под которой понимается поведение приложения, параллельного или последовательного выполнения подчиненных элементов – видов деятельности, соединённых между собой потоками.

На рисунке 2.3 представлена диаграмма активности, в которой показана работа программы.

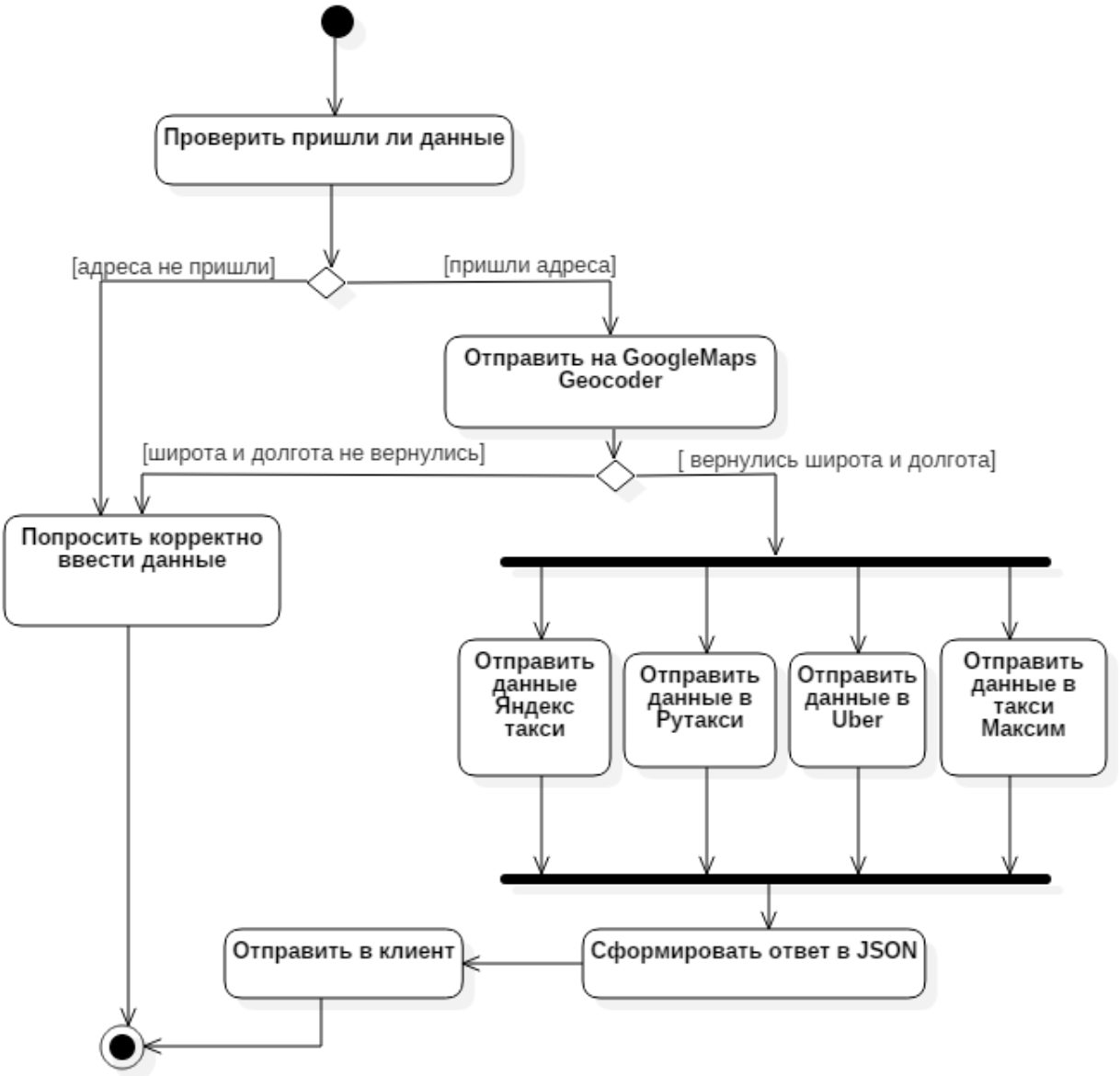


Рисунок 2.3 – Диаграмма активности

2.4 Разработка диаграммы компонентов

Диаграмма компонентов (см. рисунок 2.4) служит для представления системы в виде структурных компонентов связей и зависимости между ними.

На сервере, расположены следующие файлы. Главный файл «www», в нем формируются параметры сервера, и происходит его запуск. К файлу «www» подключается «app.js».

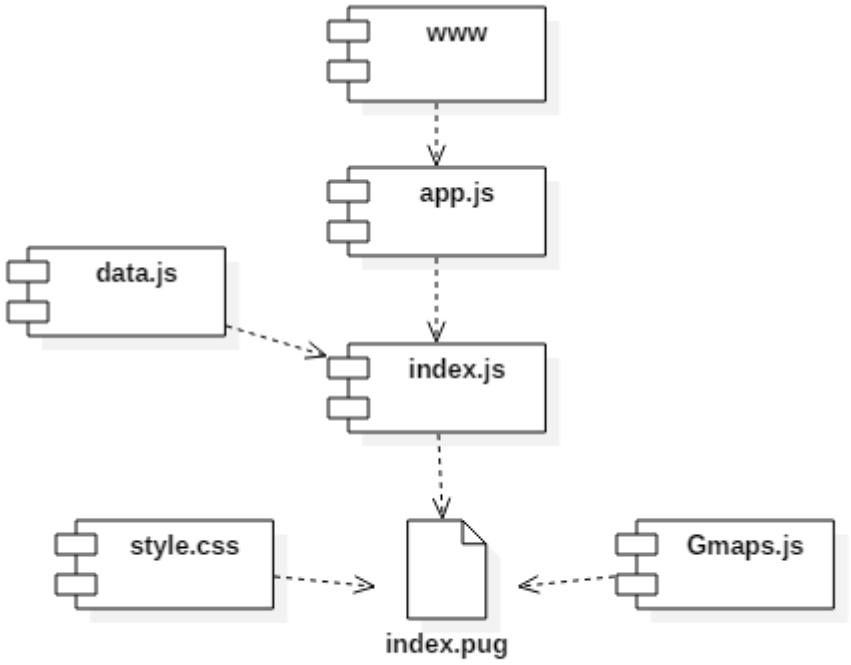


Рисунок 2.4 – Диаграмма компонентов

Файл «app.js» служит для подключения внутренних библиотек и файла для управления навигацией и обработкой запросов приложением «index.js». В файле «index.js» происходит подключение «data.js», в котором находятся все запросы подаваемые на другие сервисы, а затем и само представление самой страницы «index.pug». К странице подключен «gmap.js», в котором происходит авторизация карты, показывающая маршрут.

2.5 Выводы по разделу

В данном разделе была разработана архитектура системы, включающая в себя диаграмму прецедентов, диаграмму компонентов, а также диаграмму активности.

Диаграммы были описаны с помощью языка UML [16].

3 РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1 Общий алгоритм системы

В данном алгоритме (см. рисунок 3.1) рассмотрено, как работает приложение в целом.

Приложение находится на сервере и принимает запросы от пользователей. После запуска подгружаются все библиотеки, и запускается сервер. После предварительной подготовки, приложение ждет запросы от пользователей. Как только приходит запрос, запускается выполнение требуемых действий, запросов и генерация страниц. Далее происходит проверка работоспособности системы, анализ ошибок и в случае обнаружения неполадок сообщается администратору сервера.

Рисунок 3.1 – Общий алгоритм приложения

3.2 Алгоритм обработки запросов

На рисунке 3.2 рассматривается алгоритм обработки запросов, когда с формы на сервер подаются данные.

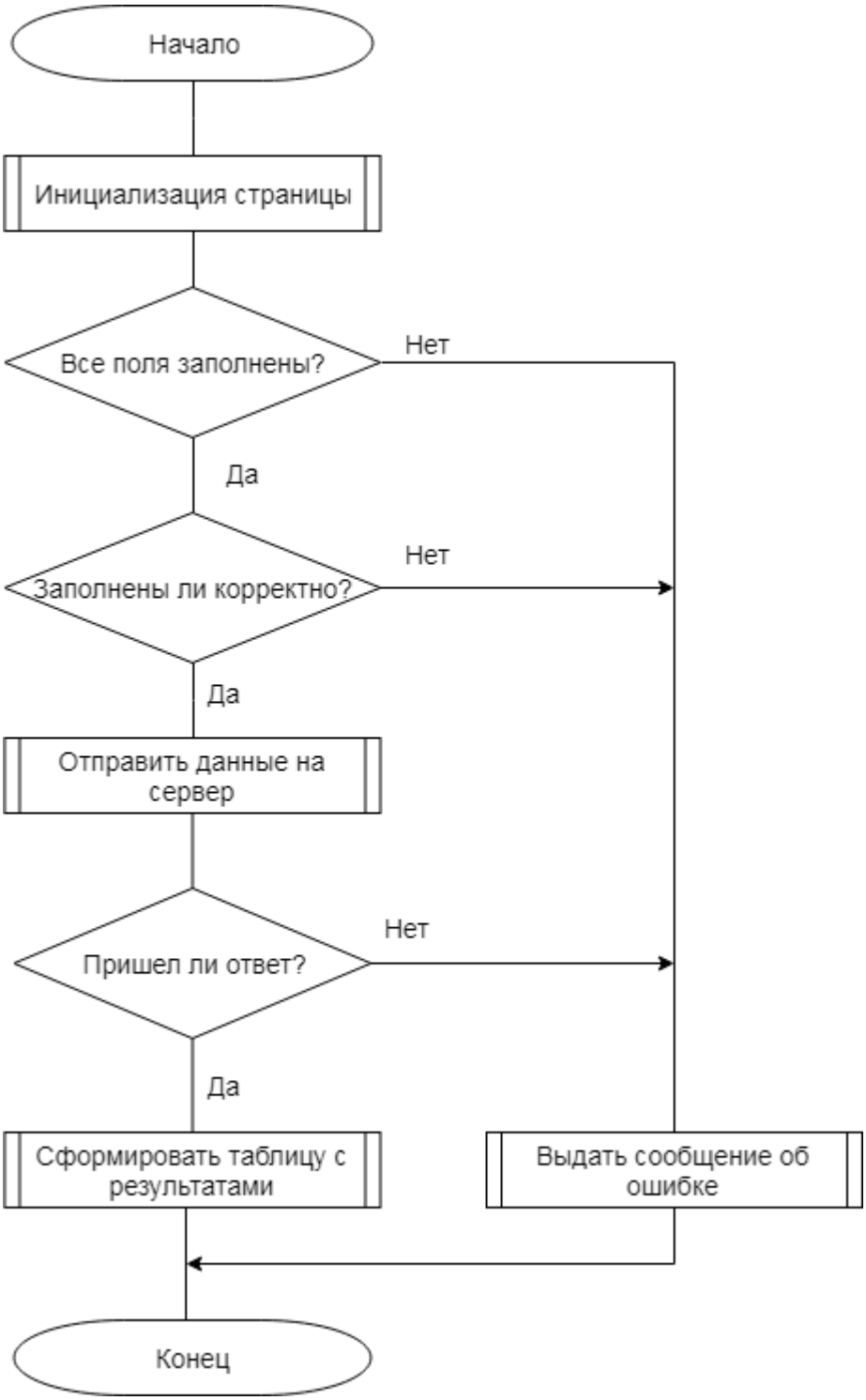


Рисунок 3.2 – Алгоритм обработки запросов

На сервер подаются данные с формы, затем идет проверка данных на пустоту, если проверка прошла успешно, они отправляются на Google Maps Geocoder [17],

если данные не являются адресом, то функция возвращает ошибку и формирует ответ-ошибку для клиента. Когда геокодер вернет широту и долготу, данные отправляются на интерфейсы такси. После того, как все такси вернули ответ, формируем его в JSON [18] формат и отправляем в клиент.

Рассмотрим алгоритм получения данных с каждого такси.

3.2.1 Алгоритм получения данных с Яндекс Такси

В данном алгоритме (см. рисунок 3.3) рассмотрено, как происходит получение данных с Яндекс Такси.

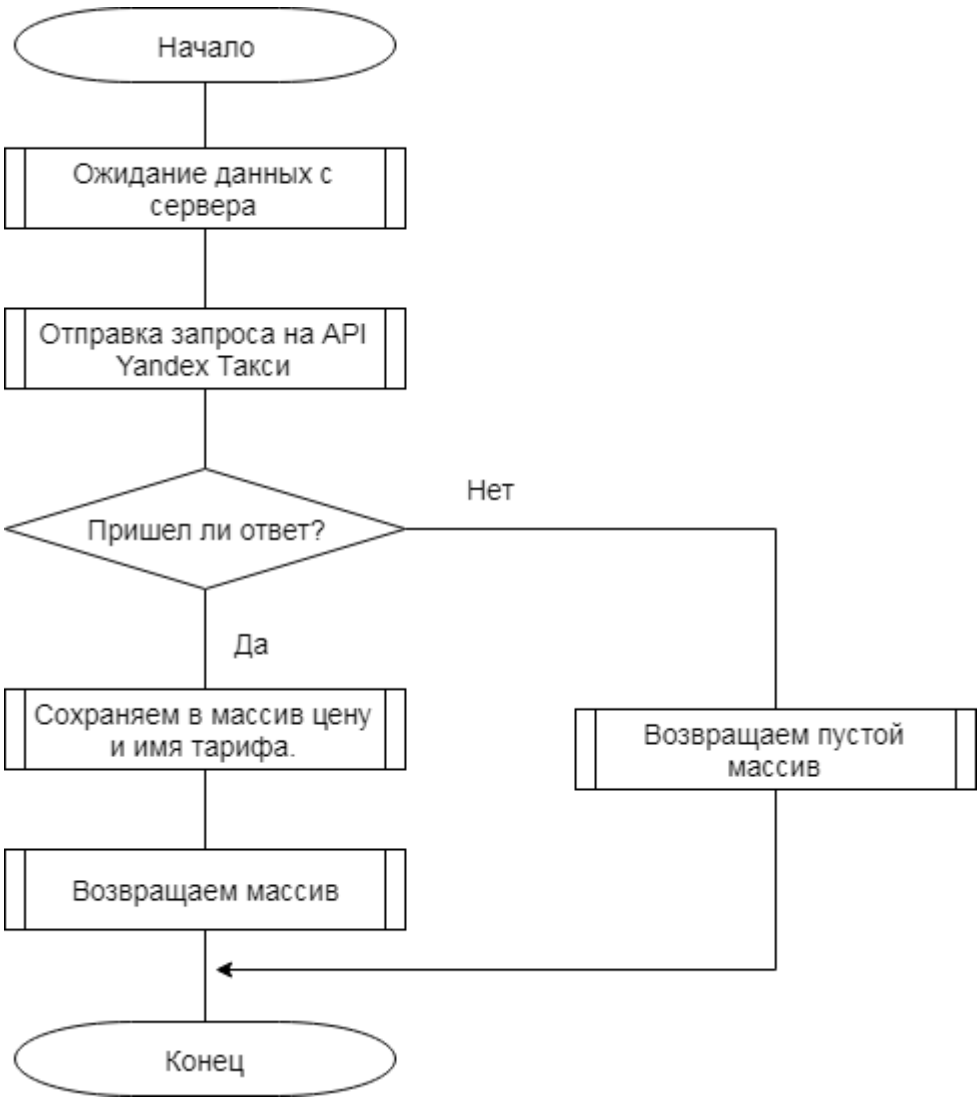


Рисунок 3.3 – Алгоритм получения данных из Яндекс такси

Сервер передает в функцию широту и долготу адреса отправления и адреса направления, которые получил из Google Maps Geocoder [17]. Задаем тело запроса, которое будет состоять из id – уникальный идентификатор страницы, zone_name - городом, в котором нужно узнать цену, supports_forced_surge – включать ли загруженность дорог и повышения тарифа в цену, routes – долгота и широта адреса назначения и отправления.

Затем создаем запрос на API Яндекса, задаем метод, задаем заголовки, которые включают в себя cookie с _id совпадающим с id из тела и yandexuid идентификатор уникального пользователя, который состоит из 15 цифр и отвечает за заказ такси.

Ожидаем ответ от Яндекса, сформированный как JSON [18]. Проверяем включает ли ответ в себя нужную информацию, если нет, то возвращаем пустой массив, чтобы избежать отвергнутого обещания, если да, то для каждого тарифа забираем цену и возвращаем в виде массива.

Далее рассмотрим листинг алгоритма по получению данных с Яндекс Такси. (см. Листинг 3.1)

Promiserequest – библиотека из модуля npm, для возможности использовать запросы с обещаниями и не работать с обратным вызовом функции. Также, чтоб использовать библиотеку, функция должна быть асинхронная (async).

```
async function Yandex(latstart, latend, lngstart, lngend) {
    let options = {
        "id": "c8771ed244734f599cb90d646f01af0c",
        "zone_name": "chelyabinsk",
        "supports_forced_surge": true,
        "route": [[lngstart, latstart], [lngend, latend]],
        "skip_estimated_waiting": false
    };

    try {
        let response = await promiserequest({
            url: "https://taxi.yandex.ru/3.0/routestats/",
            method: "POST",
            json: true,
            headers: {
                'Content-Type': "application/json",
                'Cookie': "yandexuid=2124394431518115630;
_id=c8771ed244734f599cb90d646f01af0c"
            },
            body: options
        });

        let res = {
            'name': [],
            'dist': [],
            'time': [],
            'price': []
        };
        if(response.service_levels) {
            res.time.push(response.time);
            res.dist.push(response.distance);

            for (let i = 0; i < response.service_levels.length; i++) {
                res.name.push("Yandex Taxi " +
response.service_levels[i].name);
                res.price.push(response.service_levels[i].details[0].price);
            }
        }
        return await res;

    } catch (err) {
        console.error(err);
    }
}
```

Листинг 3.1 – Алгоритм получения данных из Яндекс Такси

3.2.2 Алгоритм получения данных с такси Максим

Рассмотрим алгоритм получения данных из такси Максим (см. рисунок 3.4).

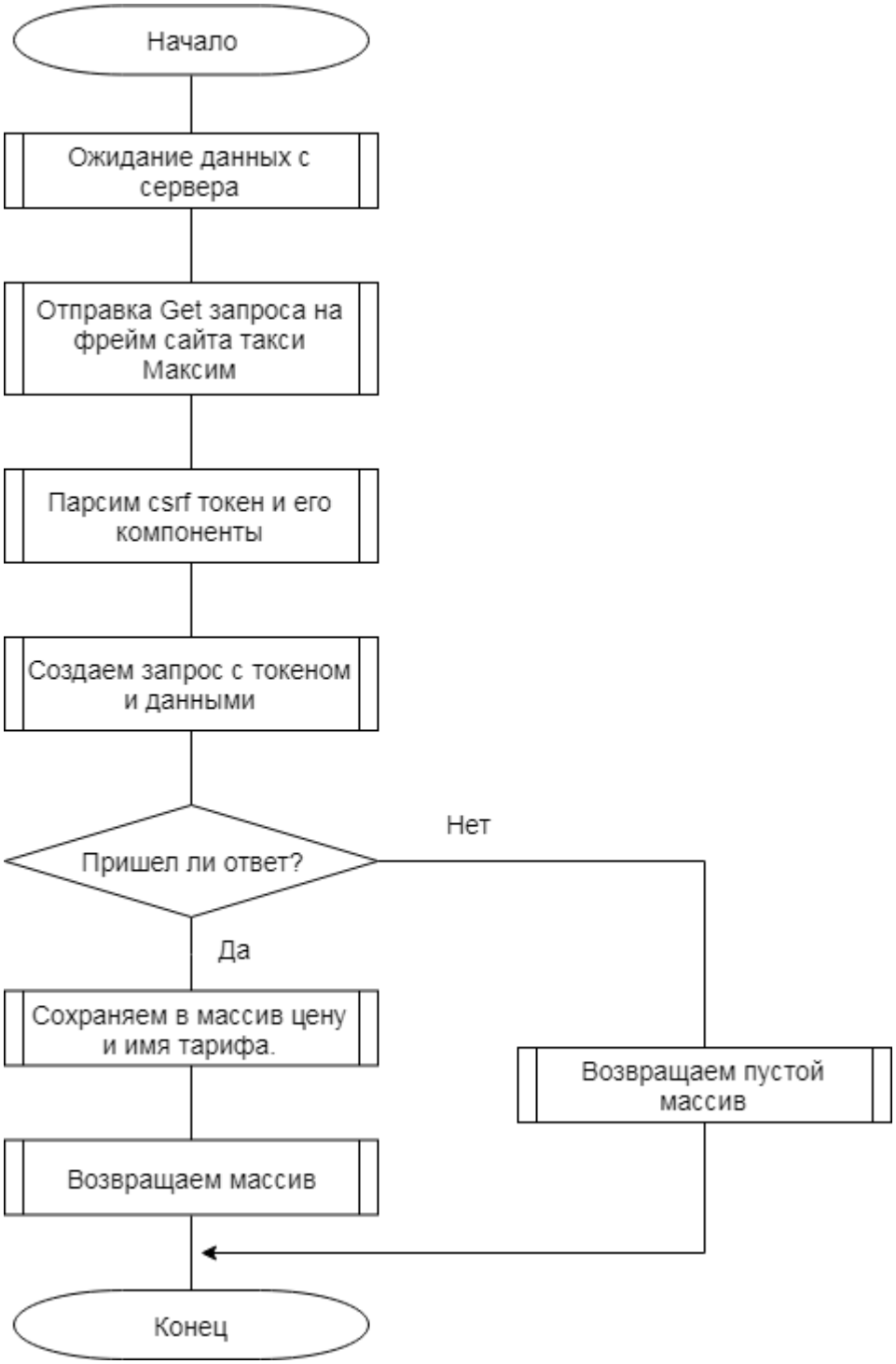


Рисунок 3.4 – Алгоритм получения данных такси Максим

После того, как сервер вызывает функцию, передавая данные с адресом отправления, его долготой и широтой, а также адресом направления, его долготой

и широтой. Вызываем функцию, которая отправляет Get запрос на фрейм страницы такси Максим, парсим оттуда csrf-token, а также забираем все cookie, которые там присутствуют и возвращаем значение.

Для каждого тарифа создаем запрос для получения цены тарифа, который включает в себя тело запроса с id тарифа, адресом отправления, его долготой и широтой, и адресом направления с его долготой и широтой. Представляем все это в виде строки со стандартным кодированием URL. Заголовки включают в себя X-CSRF-Token, который мы получили ранее, cookie, которые также получены ранее и тип кодировки запроса x-www-form-urlencoded.

После запроса в ответе нам приходит строка с ценой. Если ответ возвращает ошибку, то возвращаем пустой массив, для того, чтоб избежать отвергнутого обещания. Мы вырезаем цену из строки и кладем в массив с уже заданным именем, а затем после того, как для каждого тарифа запрос будет закончен вернем массив.

В листинге 3.2 изображен алгоритм получения токена и cookie.

```
async function getToken() {

    try {
        let response = await promiserequest({
            url: "https://client.taxsee.com/frame/?tax-
id=SrvMpYHWr%2FWymRmBUdoa6isCaQcmnFtb0xmL4kP74iIgk8vmpGZoL%2BOsQU0m3lMOMwpCkSyA
%2FU4qVJmAQaXbTw%3D%3D&fp=d526896f1310a3431da23ee6a55fb970&c=ru&l=ru&b=4&p=1&th
eme=maximV2",
            method: "GET",
            resolveWithFullResponse: true
        });
        let token = response.body;
        let from = token.search('token');
        let to = token.search('<title>');
        token = token.substring(from + 16, to - 7); //Token

        let cookie = response.headers['set-cookie'];

        return {token: token, cookie: cookie};

    } catch (err) {
        console.error(err);
    }
}
```

Листинг 3.2 – Алгоритм получения токена такси Максим

В запросе используется прямая ссылка на фрейм с сайта такси Максим, в форме которого хранится токен. Ссылка состоит из специального идентификатора такси tax-id, языка и темы maximV2. В листинге 3.3 приведен алгоритм получения данных с такси Максим.

```
async function Maxim(start, end, latstart, lngstart, latend, lngend) {

    let csrf = await getToken();

    Token = csrf.token;
```

```
    cookie = csrf.cookie;

    let res = {
        'name': [],
        'price': []
    };

    let tariff = {
        'id': [1,2,3],
        'name':['Эконом','Комфорт','Бизнес']

    };
    for (let i = 0; i<tariff.id.length; i++) {
        options = {
            'OrderForm[baseId]': 4,
            'OrderForm[tariffId]': tariff.id[i],
            'AddressForm[0][pointField]': start,
            'AddressForm[0][latitude]': latstart,
            'AddressForm[0][longitude]': lngstart,
            'AddressForm[1][pointField]': end,
            'AddressForm[1][latitude]': latend,
            'AddressForm[1][longitude]': lngend
        };
        let formBody = querystring.stringify(options);

        try {
            let response = await promiserequest({
                url:
"https://client.taxsee.com/service/calculate/?org=maxim&baseId=4&tax-
id=SrvMpYHWr%2FWymRmBUdoa6isCaQcmnFtb0xmL4kP74iIgk8vmpGZoL%2BOsQU0m3lMOMwpCkSyA
%2FU4qVJmAQaXbTw%3D%3D",
                method: "POST",
                headers: {
                    'Content-Type': "application/x-www-form-urlencoded",
                    'X-CSRF-Token': Token,
                    'Cookie': cookie
                },
                body: formBody
            });
            if (response) {
                let from = response.indexOf('price', 30);
                let to = response.search('/span');
                let go = response.substring(from + 8, to - 1);
                res.price.push(go);
                res.name.push('Maxim ' + tariff.name[i]);
            }
        } catch (err) {
            console.error(err);
        }
    }

    return res;
}
```

Листинг 3.3 – Алгоритм получения данных из Такси Максим

После получения массива токена и cookie, сохраняем, для того, чтоб вскоре записать в заголовок запроса. Для каждого тарифа создаем запрос, собираем в массив, возвращаем. Если запрос не вернул тела, то в массив ничего не добавляется.

Библиотека querystring используется для представления тела в виде закодированной строки. Так как ответ представляет из себя строку, приходится вырезать цену из строки.

3.2.3 Алгоритм получения данных с Uber

Следующий алгоритм имеет схожее построение с алгоритмом получения данных из Яндекс такси. Рассмотрим разницу между алгоритмами (см. рисунок 3.5).

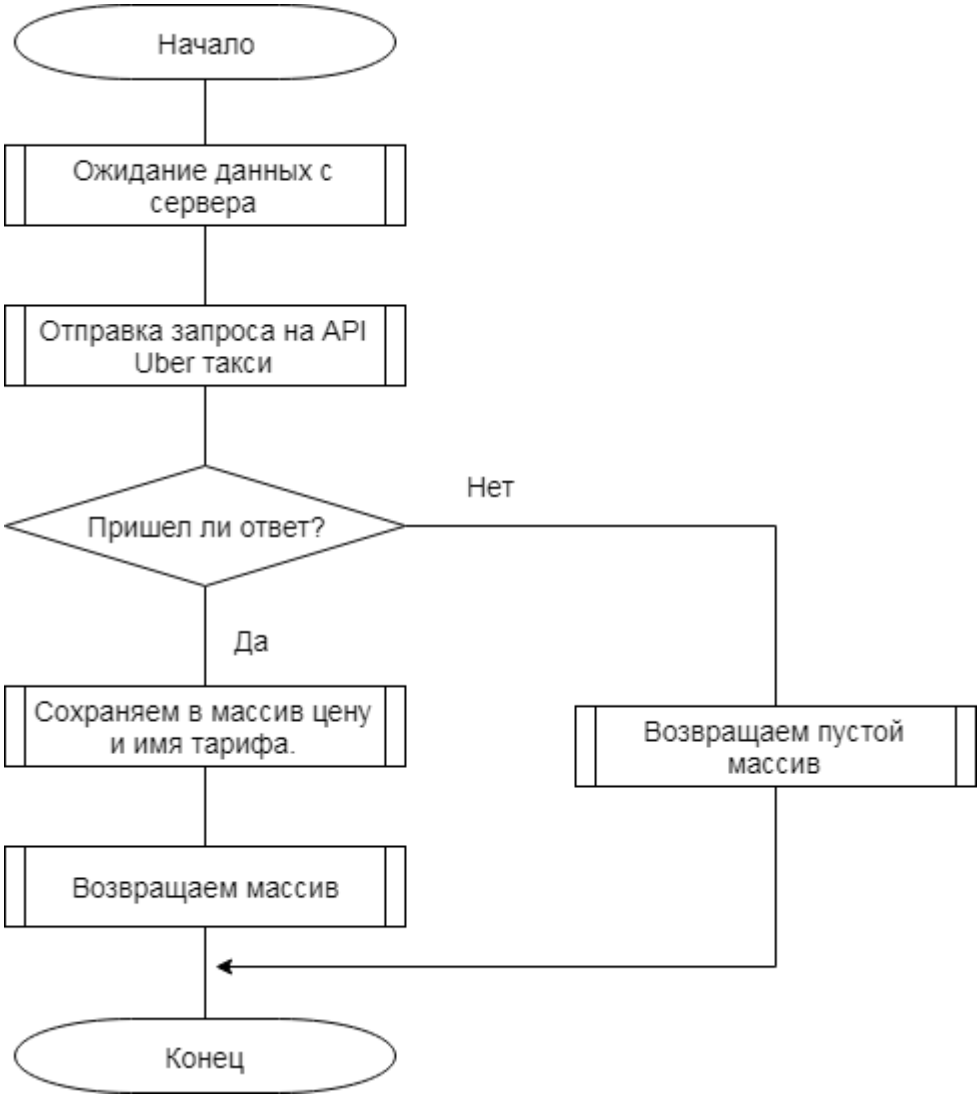


Рисунок 3.5 – Алгоритм получения данных с Uber

Получаем широту и долготу адресов. Формируем запрос на API такси Uber. Тело запроса состоит из полученных широты и долготы обоих адресов. Добавляем к URL интерфейса кодированную строку тела запроса, посылаем Get запрос, включающий в себя обновленный URL, а также идентификатор пользователя, который можно получить, зарегистрировавшись на сайте.

Проверяем ответ запроса, при ошибке он вернет false, если такое произойдет, возвращаем пустой массив, чтоб избежать отвергнутого обещания. Если нет –

выбираем из ответа цену и имя для каждого тарифа и возвращаем в виде массива. В листинге 3.4 проиллюстрирован данный алгоритм.

```
async function Uber(latstart, lngstart, latend, lngend) {

    options = {
        "start_latitude": latstart,
        "start_longitude": lngstart,
        "end_latitude": latend,
        "end_longitude": lngend
    };


    let res = {
        'name': [],
        'price': []
    };

    let bodyopt = querystring.stringify(options);
    let site = "https://api.uber.com/v1.2/estimates/price?"

    try {
        let response = await promiserequest({
            url: site + bodyopt,
            method: "GET",
            json: true,
            headers:{
                'Authorization': "Token 4d6ohusG9oItrfRuuI59Q1-4ka-J5Vahsu8I-
2aU"
            }
        });
        if(response.prices) {
            for (let i = 0; i < response.prices.length; i++) {
                res.price.push(response.prices[i].estimate);
                res.name.push(response.prices[i].display_name);
            }
        }
        return res;

    } catch (err) {
        console.error(err);
    }
}
```

Листинг 3.4 – Алгоритм получения данных из такси Uber

На асинхронную функцию приходят широта и долгота обоих адресов. После этого заносим данные в тело, делаем из этого строку, с помощью библиотеки querystrting, соединяем ссылку и строку тела и производим асинхронный Get запрос по данному URL, в заголовках указываем Authorization с токеном, полученным в личном кабинете.

Проверяем ответ, если все нормально, то в массив кладем данные о цене и названии, возвращаем обратно.

3.2.4 Алгоритм получения данных с Rutaxi

Данный алгоритм является самым долгим из всех остальных запросов, так как требует идентификатор места, начала и конца, который можно получать, обращаясь к интерфейсу, а также каждый раз запрашивать тарифы, которые возможны в городе на определенный момент времени.

Чтобы получить доступ к интерфейсу сайта, нужно зарегистрироваться на api.rutaxi.ru, а затем использовать специальный ключ для любого действия.

На рисунке 3.6 приведен примерный алгоритм получения данных с данного такси.

После того, как функция получает данные о широте и долготе обоих мест, отправляется запрос на получение тарифов в городе Челябинск, затем полученные данные от сервера отправляются на интерфейс сервиса, который с помощью обратного геокодирования возвращает ID улицы и его номер, которые, вскоре, используется для запроса цены. То есть, по сути, мы ожидаем завершения трех запросов: тарифов, места отправления и места назначения для того, чтобы начать создавать запрос.

После запроса проверяем пришел ли ответ, если да, то заполняем наш массив полученными данными и возвращаем. Рассмотрим важные особенности каждого запроса. Первое – тарифы (см. листинг 3.5), второе – алгоритм получения идентификатора адреса (см. листинг 3.6).

```
async function ratesRutaxi() {
    options = {
        "city": "chel"
    };
    try {
        let response = await promiserequest({
            url: "https://api.rutaxi.ru/api/1.0.0/rates/",
            method: "GET",
            json: true,
            headers: {
                'Content-Type': "application/json",
                'X-Parse-Application-Id': "App
g1iyt78Tou38ELWXYrBxLmBr0nqrB44c"
            },
            strictSSL: false,
            body: options
        });

        return response.data.rates;

    } catch (err) {
        console.error(err);
    }
}
```

Листинг 3.5 – Алгоритм получения тарифов города Челябинск

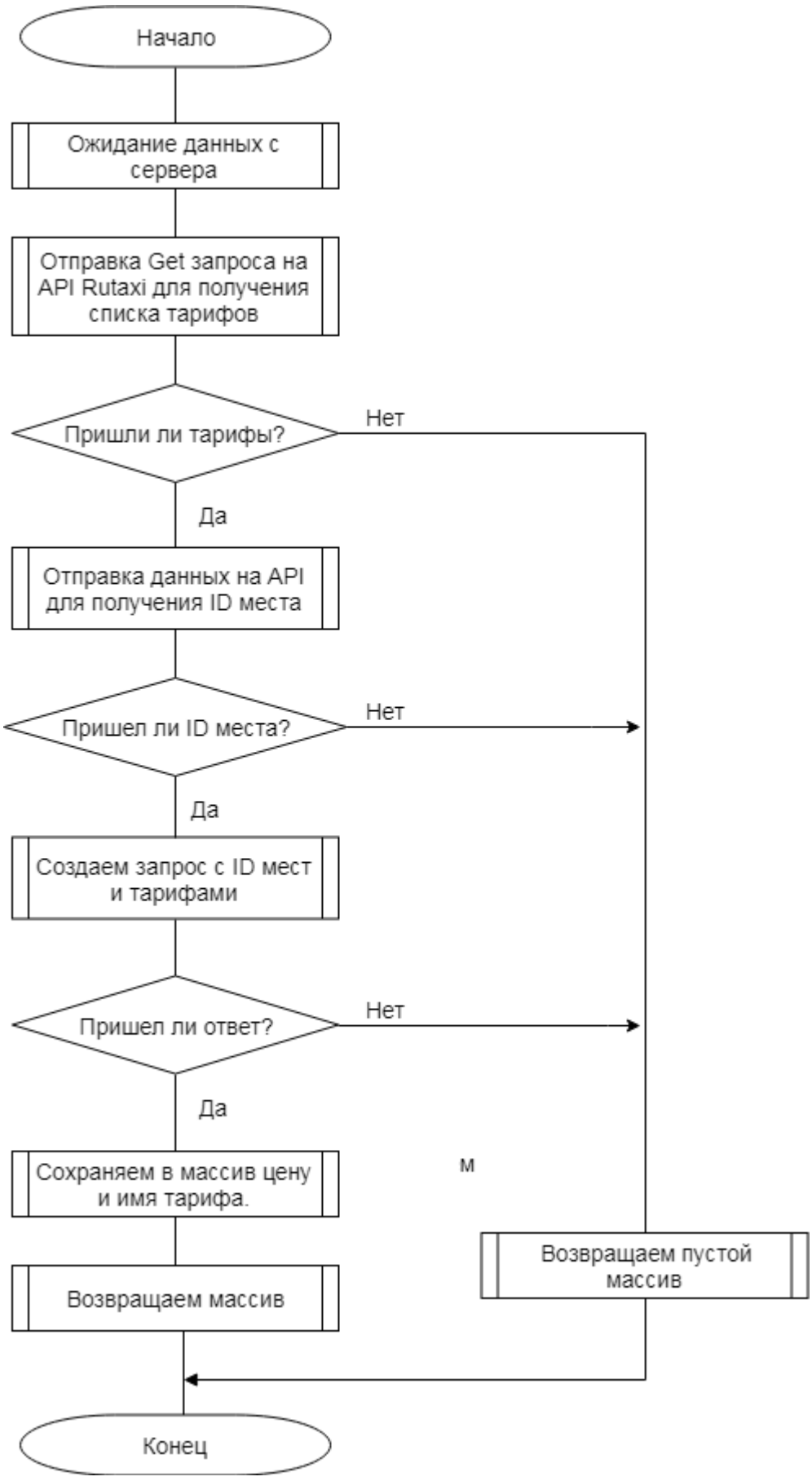


Рисунок 3.6 – Алгоритм получения данных из Rutaxi

Создаем асинхронный Get запрос, который вернет нам всевозможные тарифы в указанном городе, в нашем случае Челябинске. В заголовке задаем, что мы используем тип JSON, а в поле X-Parse-Application_Id ключ, полученный в личном кабинете. Важной особенностью, которая отличает данный интерфейс от других – обязательность отключения сертификата SSL. То есть, информация, которая будет передаваться в запросе имеют большой шанс быть перехваченными.

```
async function IDRutaxi(lat, lng) {

    options = {
        "latitude": lat,
        "longitude": lng,
        "city": "chel"
    };
    try {
        let response = await promiserequest({
            url: "https://api.rutaxi.ru/api/1.0.0/near/",
            method: "POST",
            json: true,
            headers: {
                'Content-Type': "application/json",
                'X-Parse-Application-Id': "App
g1iyt78Tou38ELWXYrBxLmBr0nqrB44c"
            },
            strictSSL: false,
            body: options
        });
        return response.data.objects[0];

    } catch (err) {
        console.error(err);
    }
}
```

Листинг 3.6 – Алгоритм получения идентификаторов адресов в Rutaxi

После вызываются функция для нахождения адреса и его идентификатора, который хранится в базе у сервиса. Делаем POST запрос (см. листинг 3.7) используя в качестве тела широту и долготу, а также обязательным требованием город, в котором нужно получить данные. После прописываем то, что было в коде до этого и затем возвращаем ответ.

```
async function Rutaxi(latstart, lngstart, latend, lngend) {

    let idstr = await IDRutaxi(latstart, lngstart);
    let idend = await IDRutaxi(latend, lngend);
    let idrate = await ratesRutaxi();
    let res = {
        'name': [],
        'price': []
    };
```

```
    for (let i = 0; i < idrate.length; i++) {
        let options = {
            "city": "chel",
            "Order": {
                "rate_id": idrate[i].id,
                "points": [
                    {
                        "object_id": idstr.id,
                        "house": idstr.house
                    },
                    {
                        "object_id": idend.id,
                        "house": idend.house
                    }],
                "is_preorder": 0
            }
        };
    ..promiserequest..
    }
    return await res;
}
```

Листинг 3.7 – Алгоритм получения данных из Rutaxi

После получения всех данных из запросов выше мы для каждого тарифа делаем запрос. Тело запроса представляет из себя город, в котором будем использовать услуги такси, rate_id – идентификатор тарифа, который мы получили заранее, точки отправления и назначения, в которых фигурируют object_id – идентификатор улицы, которую получаем из запроса и house – дом, относящийся к данной улице, а также указываем, что запрос не является пред заказом, с помощью is_preorder.

3.3 Алгоритм вывода услуг такси

На рисунке 3.7 приведен алгоритм вывода полученной информации в браузер.

После взаимодействия пользователя с формой, форма, как и на сервере вначале проверяется на пустую строку, затем отправляется на Google Maps Javascript API, если ответ приходит OK, то запрос отправляется на сервер. Ожидаем ответа от сервера и ловим с помощью метода fetch [19], который по своей структуре очень похож на Ajax. Оба метода позволяют обновлять отдельные элементы страницы, что делает их очень полезными.

Чтобы получить данные с сервера с помощью fetch требуется настроить CORS – механизм, использующий дополнительные заголовки, для того чтобы иметь доступ к выбранным ресурсам с сервера на источнике.

Для этого используется библиотека CORS, которая облегчает использование данного механизма (см. листинг 3.8).

Рисунок 3.7 – Алгоритм вывода данных в браузер

```
const cors = require('cors');

const corsOptions= {
    origin: 'http://localhost:3000',
    methods: 'GET,POST',
    allowedHeaders: 'Content-Type, Accept',
    credentials: true
};
router.post('/',cors(corsOptions), async function(req, res)
```

Листинг 3.8 – Подключение настроек подключения

Подключаем библиотеку, прописываем опции. Где origin или Access-Control-Allow-Origin включает в себя информацию, с какого домена разрешено принимать запрос. Methods отвечает за методы, которые включены. AllowedHeaders – описываются заголовки. В данном случае мы используем кодировку и ответ. Credentials отвечает за использование cookie. Данное свойство обязательно, так как мы используем csrf защиту от атак.

3.4 Выводы по разделу

В данном разделе рассмотрены основные алгоритмы работы системы. Было составлено шесть алгоритмов: это алгоритм работы всего приложения в целом, алгоритмы получения данных с каждого такси и алгоритм выдачи страниц. Каждый алгоритм расписан поэтапно с комментариями.

4 ТЕСТИРОВАНИЕ СИСТЕМЫ

Обязательным этапом разработки является проведение тестирование приложения для выявления ошибок и их устранения. Начнем с проверки полей ввода.

При первом открытии сайта, должна появиться страница с двумя полями ввода и кнопкой подтверждения (см. рисунок 4.1). После удачного ввода адресов должна появиться карта и ответ, иначе на странице должно быть отображено предупреждение о пустом или некорректном поле ввода.

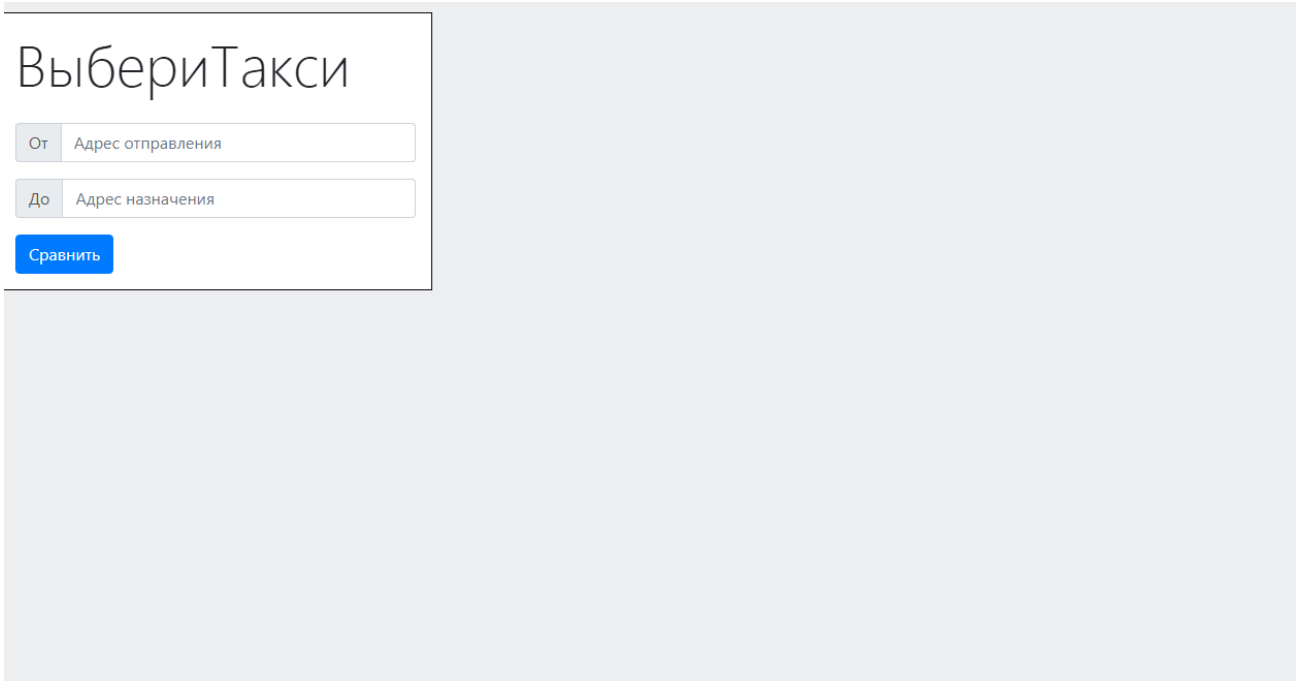


Рисунок 4.1 – Начальная страница

Для тестирования требуется проверить реакцию на пустые поля ввода, на поля ввода заполненные некорректными данными и с корректными адресами, выбранными из предложенных.

На рисунке 4.2 приведен пример работы сайта, когда пользователь нажимает кнопку при пустых полях данных.

На рисунке 4.3 приведен пример работы сайта, когда пользователь нажимает кнопку с некорректных входными данными.

На рисунке 4.4 приведен пример предложения адреса в поле ввода.

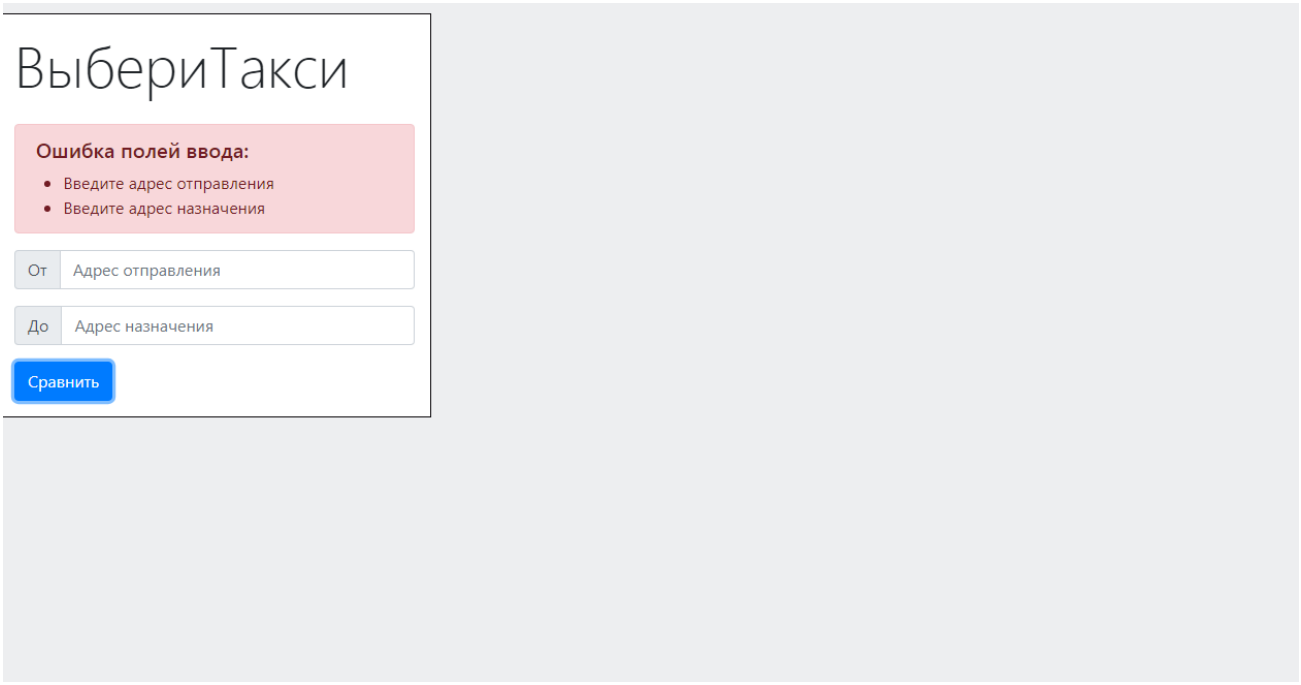


Рисунок 4.2 – Проверка на пустые поля

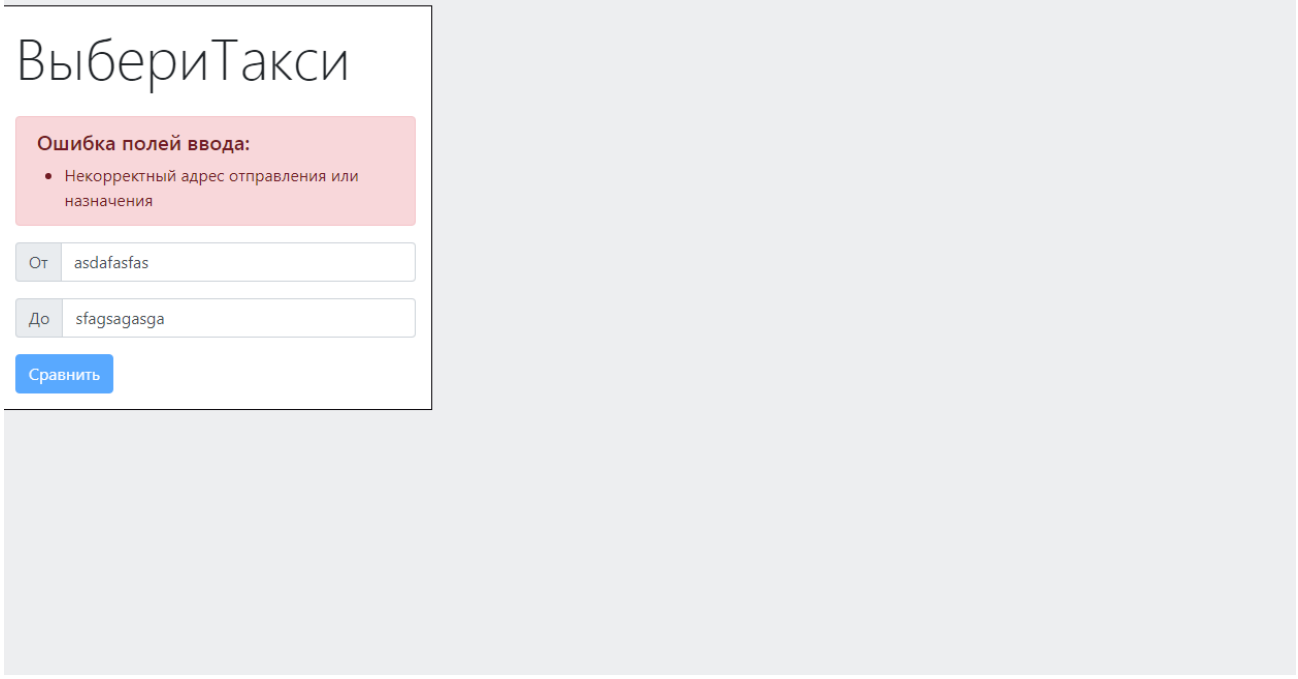


Рисунок 4.3 – Проверка с некорректными входными данными

При вводе пустых полей ввода выводится сообщение об ошибке в каком поле обнаружено пустое поле ввода требующим заполнения.

При вводе некорректного адреса выводится ошибка о том, что данные поля не являются адресными пространствами google maps.

Вводя адрес, Google Maps Autocomplete предлагает адреса исходя из вашего ввода. В первую очередь будут предлагаться адреса из Челябинска, а затем по отдаленности от Челябинска.

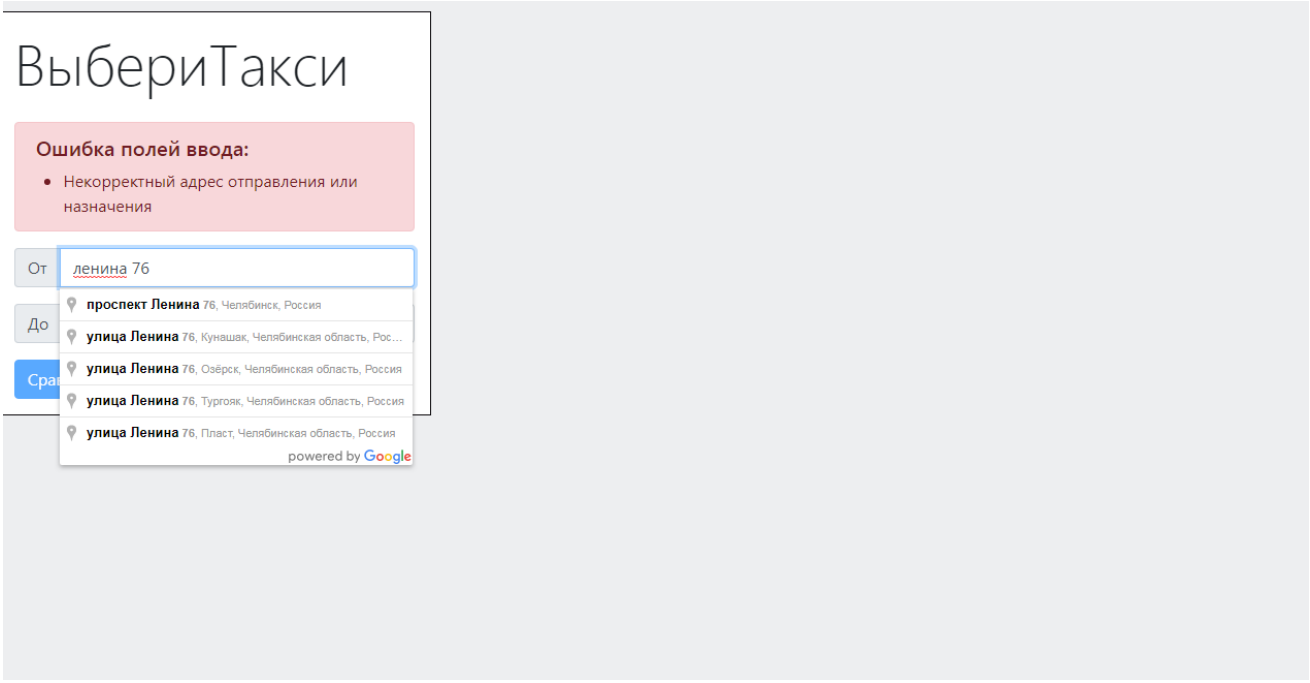


Рисунок 4.4 – Пример предложения адреса в поле ввода

На рисунке 4.5 и 4.6 приведен пример работы программы при вводе корректного адреса из предложенных Google Maps Autocomplete.

При нажатии на кнопку вначале открывается карта показывающая маршрут, а после того, как сервер даст ответ генерируется таблица с данными.

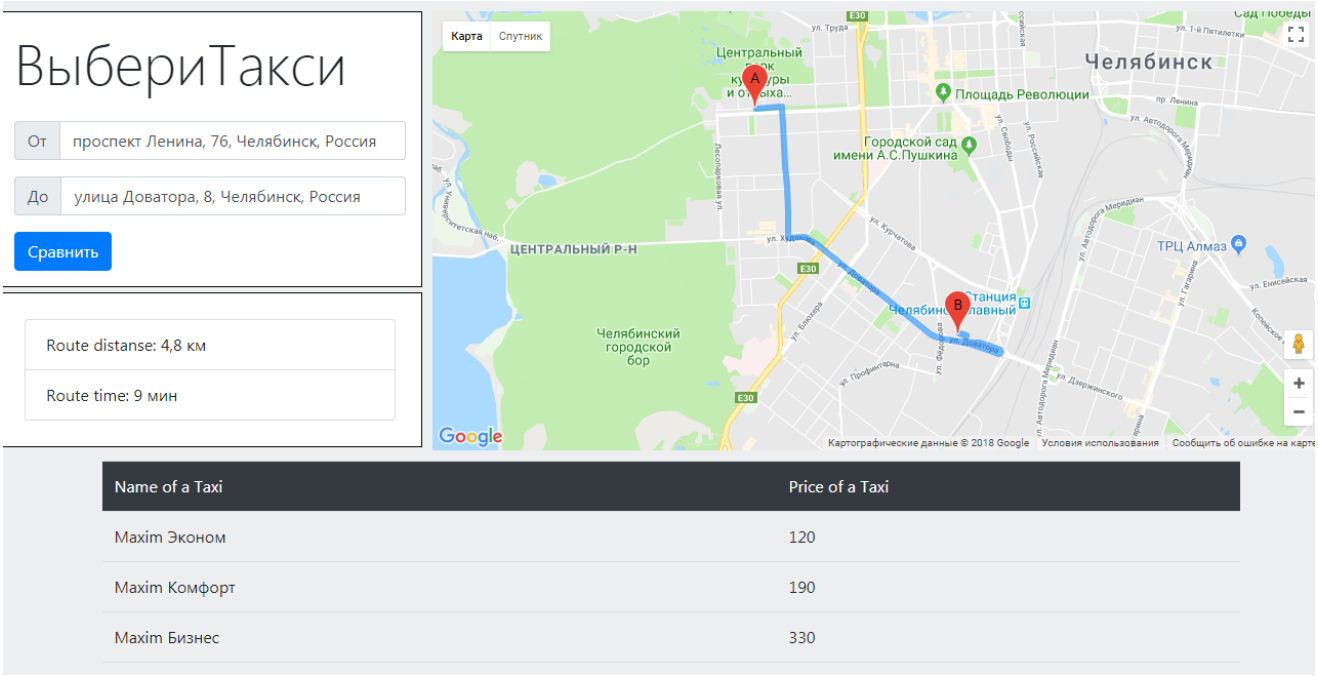
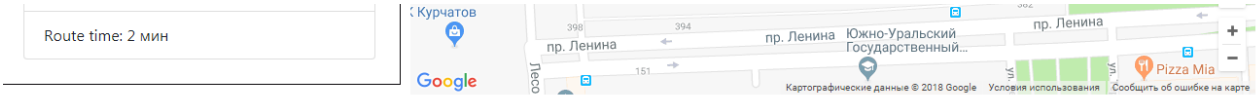


Рисунок 4.5 – Пример работы при вводе адреса из списка

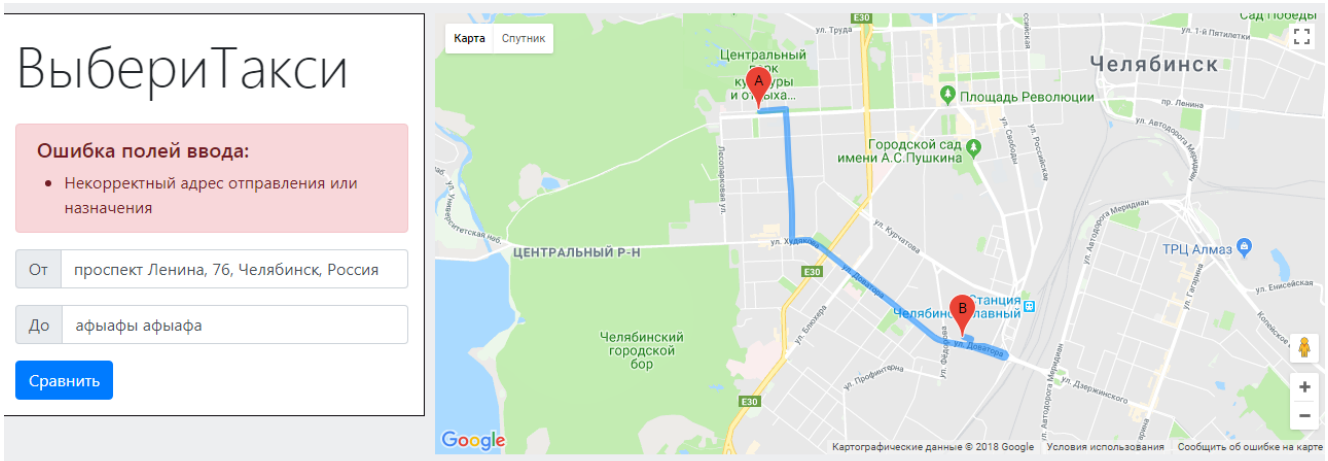
На рисунке 4.7 приведен пример, когда пользователь на странице с ответами вводит некорректные данные, что приводит к ошибке.

Сайт выводит ошибку некорректного адреса и выполнения запроса не происходит.



Name of a Taxi	Price of a Taxi
Maxim Эконом	80
Maxim Комфорт	140
Maxim Бизнес	300
Rutaxi Лидер	85
Rutaxi Комфорт	160
Yandex Taxi Эконом	55 руб.
Yandex Taxi Комфорт	139 руб.
uberSTART	RUB44-54
UberX	RUB117-143
uberSELECT	RUB225-275

Рисунок 4.6 – Таблица ответов с ценой



Name of a Taxi	Price of a Taxi
Maxim Эконом	120
Maxim Комфорт	190
Maxim Бизнес	330

Рисунок 4.7 – Работа программы при вводе некорректного адреса.

Рассмотрим тест, когда делают перезапрос данных на странице с уже сформированным ответом.

На рисунке 4.8 представлено, что при перемене адреса, вначале прогружается карта, высвечивается уведомление о том, что сервер еще производит расчеты, а кнопка становится неактивной.

После того, как сервер вернет ответ (см. рисунок 4.9), таблицу со старыми значениями заменяет новая.

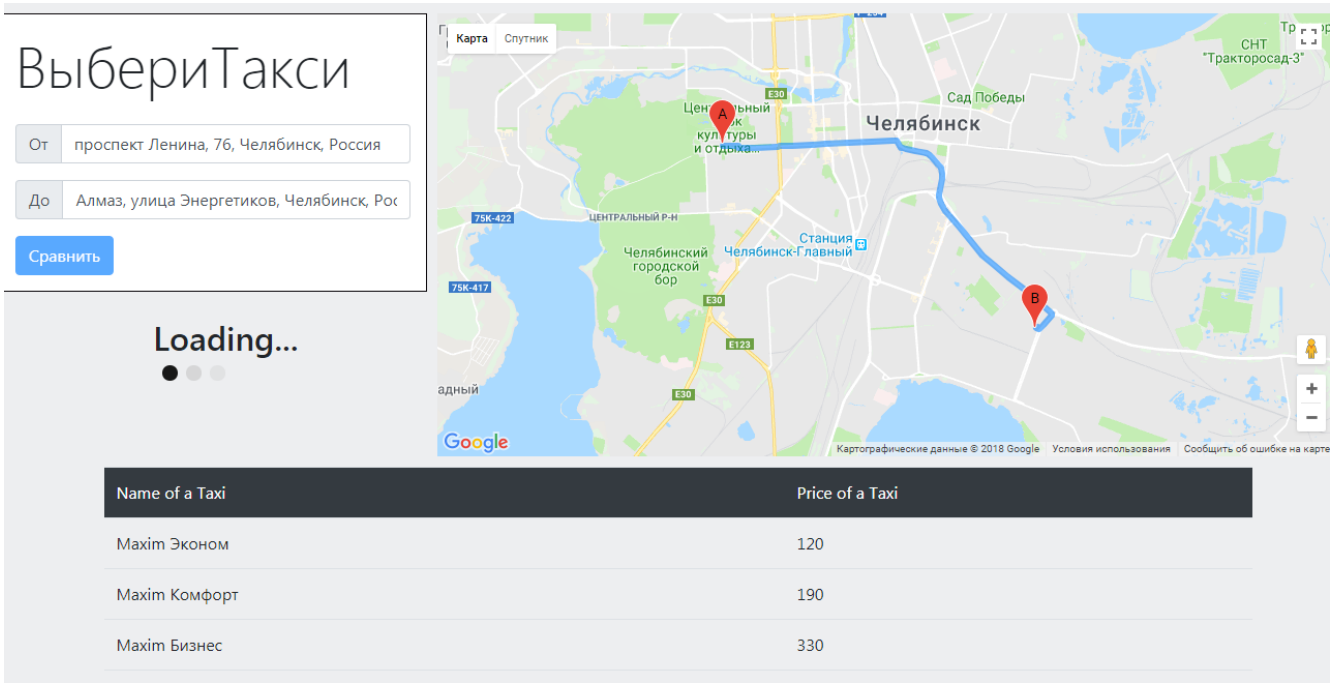


Рисунок 4.8 – Тест при смене повторной смене адреса

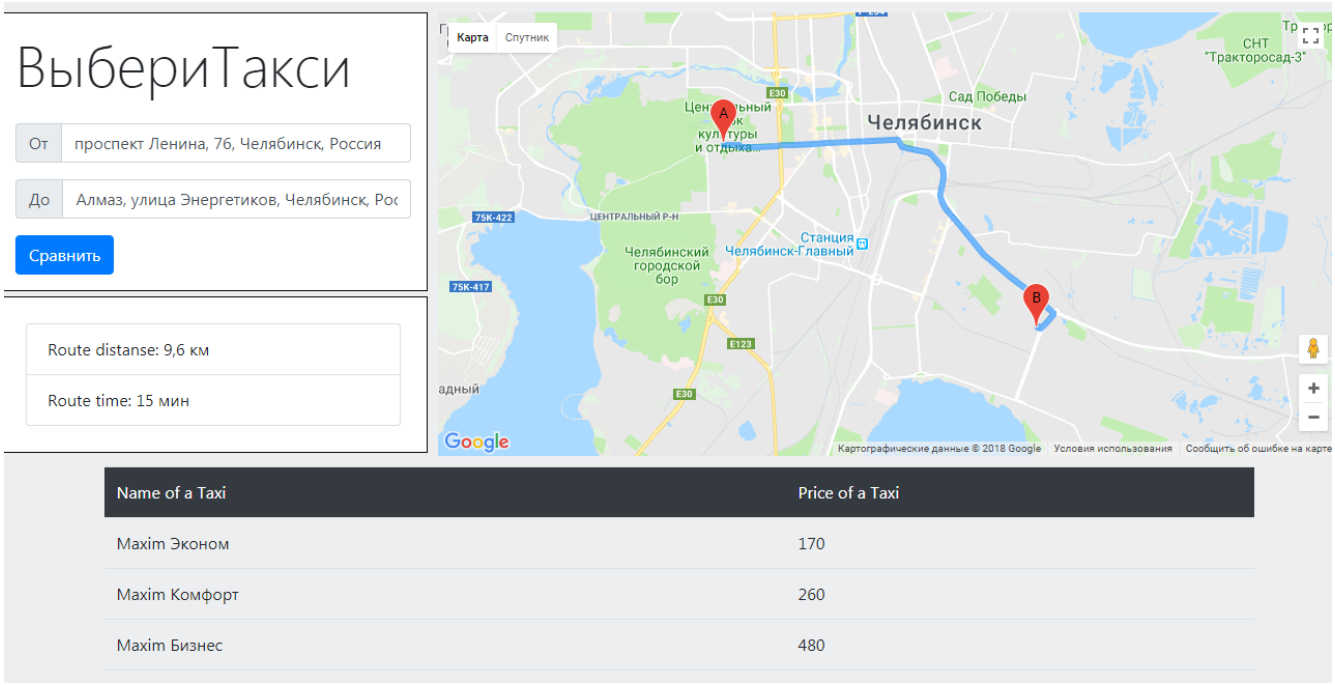


Рисунок 4.9 – Итог повторной смены адреса.

Рассмотрим пример при вводе одинаковых данных в окна. При вводе одинаковых данных (см. рисунок 4.10) выводится уведомление о том, что были введены одинаковые адреса и выполнения запроса не происходит.

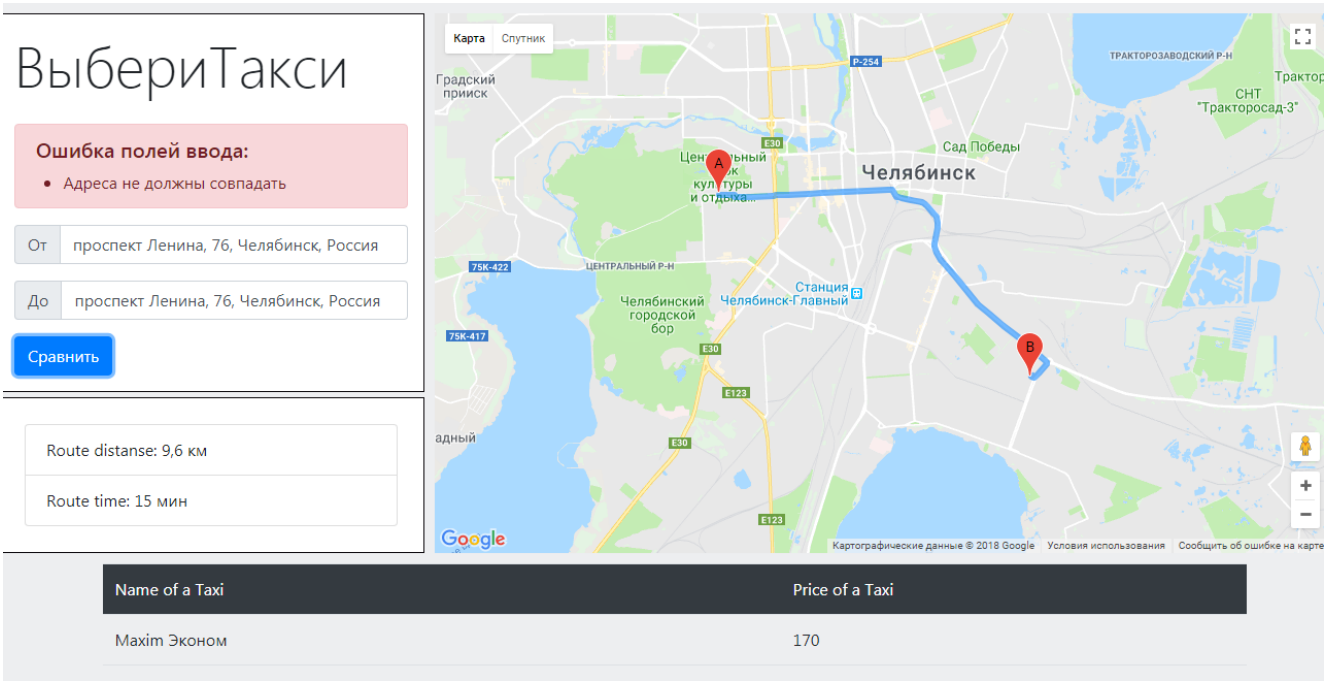


Рисунок 4.10 – Тест на ввод одинаковых адресов

Рассмотрим, как будет вести себя веб-сайт на мобильных устройствах.

На рисунке 4.11 приведен пример адаптации сайта при уменьшении размера экрана. Все блоки идут друг за другом, вначале форма, затем карта, потом ответ (см. рисунок 4.12).

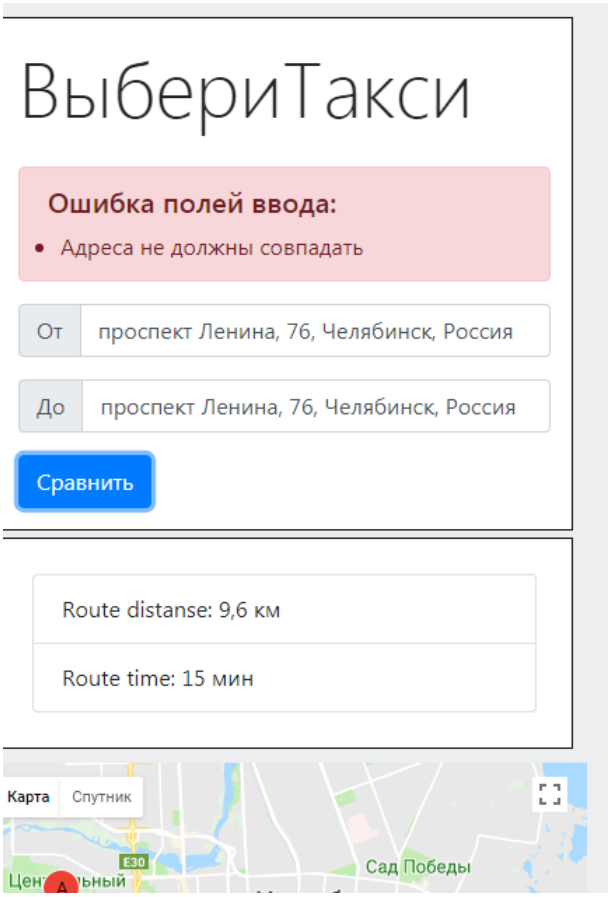
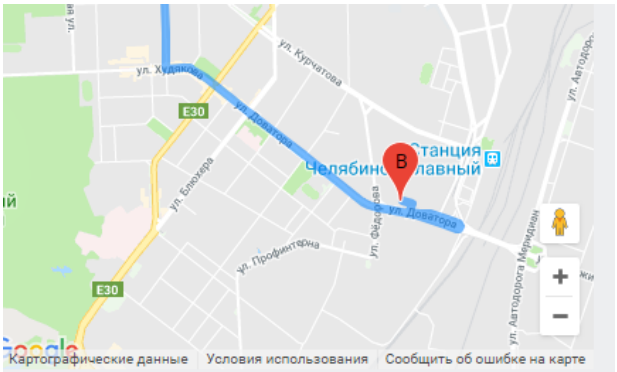


Рисунок 4.11 – Тест работы сайта на мобильных устройствах



Name of a Taxi	Price of a Taxi
Maxim Эконом	110
Maxim Комфорт	170
Maxim Бизнес	300
Rutaxi Лидер	105
Rutaxi Комфорт	180
Yandex Taxi Эконом	95 руб.
Yandex Taxi Комфорт	178 руб.

Рисунок 4.12 – Тест работы сайта на мобильных устройствах

В ходе тестирования проверены возможные ошибки. Тесты, с помощью которых были проверены ошибки: тест на пустое поле ввода, тест на некорректный ввод адресов, тест на совпадение адреса. Сайт работает после любой из вышеперечисленных ошибок. При перемене адресов веб-сайт уведомляет пользователя о том, что клиент ожидает ответа с сервера. Во время ожидания ответа, кнопка “Сравнить” становится недоступна.

ЗАКЛЮЧЕНИЕ

В этой работе были проанализированы онлайн такси в городе Челябинск. Была рассмотрено как они получают информацию. Были проанализированы аналоги агрегаторов такси, были выявлены их недостатки и преимущества.

Были рассмотрены актуальные платформы для разработки веб-сайтов. Выбрана платформа Node.js. Была спроектирована архитектура веб-сайта, диаграммы использования, а также диаграмма запросов и диаграмма компонентов.

Были реализованы общий алгоритм системы, алгоритм обработки запросов, алгоритмы для каждого онлайн такси, а также алгоритм вывода решения.

На заключительном этапе разработки была проведена проверка работоспособности и корректности обработки ошибок, связанных с вводом некорректных данных, пустыми полями, а также на адаптивность сайта к смене размера окна.

Таким образом, все поставленные задачи были успешно выполнены.

В дальнейшем планируется расширить возможности сайта, добавив личный кабинет, с возможностью добавлять туда избранные адреса, привязать номер, добавить возможность выбора дополнительных настроек при поиске такси, а также иметь возможность заказать такси прямо на сайте, если возможно, а если такой возможности не будет – перевести на сайт такси с заполненными полями.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Discovery Research Group. Анализ рынка такси в России. [Электронный ресурс] URL: http://drgroup.ru/818-Analiz-rynka-taksi-v-Rossii.html (дата обращения: 04.04.2018).
2. РБК магазин исследований. Анализ рынка такси. [Электронный ресурс] URL: https://marketing.rbc.ru/research/27993 (дата обращения: 04.04.2018).
3. Аналитический центр при правительстве Российской федерации [Электронный ресурс] URL: http://ac.gov.ru/ (дата обращения: 10.04.2018).
4. Яндекс Такси [Электронный ресурс] URL: https://taxi.yandex.ru/about/ (дата обращения: 01.03.2018).
5. Такси Gett [Электронный ресурс] URL: https://gett.com/ru/about/ (дата обращения: 01.03.2018).
6. Такси Uber [Электронный ресурс] URL: https://help.uber.com/ (дата обращения: 01.03.2018).
7. Такси Максим [Электронный ресурс] URL: https://taximaxim.ru/blog (дата обращения: 01.03.2018).
8. Такси Лидер [Электронный ресурс] URL: https://rutaxi.ru (дата обращения 01.03.2018).
9. Softrare [Электронный ресурс] URL: http://softrare.ru/android/sravni-taksi (дата обращения: 10.04.2018).
10. Зудилова, Т.В. Web-программирование JavaScript. / Т.В. Зудилова, М.Л. Буркова. – СПб. : НИУ ИТМО, 2012. - 68 с.
11. Одиночкина, С.В. Web-программирование PHP. – СПб. : НИУ ИТМО, 2012. – 79 с.
12. Сандерсон, С. Pro ASP.NET MVC Framework, Вильямс, 2010 – 96 с.
13. Хэррон, Д. Node.js. Разработка серверных веб-приложений в JavaScript. – М. : ДМК Пресс, 2012. – 144 с.
14. Void canvas [Электронный ресурс] URL: http://voidcanvas.com/describing-node-js/ (дата обращения: 11.04.2018).
15. Паттерны для новичков. Хабр [Электронный ресурс] URL: https://habr.com/post/215605/ (дата обращения: 10.05.2018).
16. What is UML. [Электронный ресурс] URL: http://www.uml.org/what-is-uml.htm (дата обращения: 10.05.2018).
17. Руководство для разработчиков | Google Maps Geocoding. [Электронный ресурс] URL: https://developers.google.com/maps/documentation/geocoding/intro?hl=ru (дата обращения: 16.05.2018).
18. JSON [Электронный ресурс] URL: https://www.json.org/json-ru.html (дата обращения: 17.05.2018).
19. Fetch documentation [Электронный ресурс] URL: https://github.github.io/fetch/ (дата обращения: 17.05.2018).

ПРИЛОЖЕНИЕ 1. ОПИСАНИЕ ПРОГРАММЫ

1. Общие сведения

Веб-сайт предназначен для агрегации услуг такси в городе Челябинск. Приложение написано на языке JavaScript и работает на платформе Node.js. Для работы приложения требуется установка платформы Node.js, загрузка всех библиотек, указанных в файле «package.json».

2. Функциональное назначение

Веб-сайт позволяет пользователю получать стоимость на услуги такси, которые имеются на данный момент в городе Челябинск.

3. Описание логической структуры

При разработке архитектуры приложения был использован шаблон проектирования Model-View-Controller (MVC). Пользователь посылает запросы сервер. Сервер обрабатывает запросы, делает запросы к API такси, получает результаты запросов и отправляет пользователю web-страницы.

На сервере, расположены следующие файлы. Главный файл «www», в нем формируются параметры сервера, и происходит его запуск. К файлу «www.js» подключается «app.js». Файл «app.js» служит для подключения внутренних библиотек и файла для управления навигацией и обработкой запросов приложением «index.js». В файле «index.js» происходит подключение «data.js», в котором находятся все запросы подаваемые на другие сервисы, а затем и само представление самой страницы «index.pug». К странице подключен «gmap.js», в котором происходит авторизация карты, показывающая маршрут.

4. Используемые технические средства

Приложение может на всех платформах, которые поддерживает платформа Node.js. Протестирована работа на платформе «x86» и операционной системе «Microsoft Windows 7».

5. Вызов и загрузка

После развертывания приложения, нужно запустить файл «www» с помощью Node.js. Либо перейти в консоли платформы Node.js в папку с приложением и ввести команду «npm start».

6. Входные и выходные данные

Входные данные пользователя представляют собой запросы с параметрами, в которых находятся адрес отправления и адрес назначения. На выходе пользователь получает веб-страницу с таблицей цен и наименований тарифов такси и картой маршрута.

www – главный файл приложения

```
#!/usr/bin/env node

/**
 * Module dependencies.
 */

var app = require('../app');
var debug = require('debug')('diplomo:server');
var http = require('http');

/**
 * Get port from environment and store in Express.
 */

var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);

/**
 * Create HTTP server.
 */

var server = http.createServer(app);

/**
 * Listen on provided port, on all network interfaces.
 */

server.listen(port);
server.on('error', onError);
server.on('listening', onListening);

/**
 * Normalize a port into a number, string, or false.
 */

function normalizePort(val) {
  var port = parseInt(val, 10);

  if (isNaN(port)) {
    // named pipe
    return val;
  }

  if (port >= 0) {
    // port number
    return port;
  }

  return false;
}

/**
 * Event listener for HTTP server "error" event.
 */

function onError(error) {
  if (error.syscall !== 'listen') {
    throw error;
```

```
  }

  var bind = typeof port === 'string'
    ? 'Pipe ' + port
    : 'Port ' + port;

  // handle specific listen errors with friendly messages
  switch (error.code) {
    case 'EACCES':
      console.error(bind + ' requires elevated privileges');
      process.exit(1);
      break;
    case 'EADDRINUSE':
      console.error(bind + ' is already in use');
      process.exit(1);
      break;
    default:
      throw error;
  }
}

/**
 * Event listener for HTTP server "listening" event.
 */

function onListening() {
  var addr = server.address();
  var bind = typeof addr === 'string'
    ? 'pipe ' + addr
    : 'port ' + addr.port;
  debug('Listening on ' + bind);
}
```

---

app.js – модуль подгружающий библиотеки

```
const createError = require('http-errors');
const express = require('express');
const bodyParser = require('body-parser');
const path = require('path');
const cookieParser = require('cookie-parser');
const logger = require('morgan');
const session = require('express-session');


const indexRouter = require('./routes/index');

const app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'pug');

app.use(logger('dev'));
app.use(bodyParser.urlencoded({ extended: false,inflate:false }));
app.use(bodyParser.json());
app.use(cookieParser());
app.use(session({secret: 'daetojestko', resave:false, saveUninitialized:false}));
app.use(express.static(path.join(__dirname, 'public')));
```

```
app.use('/', indexRouter);

app.use(function (req, res, next) {

// Website you wish to allow to connect
    res.setHeader('Access-Control-Allow-Origin', '*');

// Request methods you wish to allow
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST');

// Request headers you wish to allow
    res.setHeader('Access-Control-Allow-Headers', "Content-Type, Accept");

// Set to true if you need the website to include cookies in the requests sent
// to the API (e.g. in case you use sessions)
    res.setHeader('Access-Control-Allow-Credentials', true);

// Pass to next layer of middleware
    next();
});

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

---

routes/index.js – файл маршрутизации

```
const express = require('express');
const router = express.Router('redirect:false');
const data = require('../public/data');
const cors = require('cors');
const csrf = require('csurf');
/* GET home page. */

//подключаем csrf защиту
const csrfProt = csrf();
router.use(csrfProt);

//прописываем опции доступа
const corsOptions= {
    origin: 'http://localhost:3000',
    methods: 'GET,PUT,POST',
    allowedHeaders: 'Content-Type, Accept',
    credentials: true
};
```

```
router.get('/',cors(corsOptions), function(req, res, next) {
    res.render('index', { title: 'Viberi taxi',csrfToken: req.csrfToken()});
});

router.post('/',cors(corsOptions), async function(req, res) {
    if((req.body.start)&&(req.body.end)){
        let latstart, latend, lngstart, lngend;

        let info = await (async function () {
            //location start
            let start = await data.latlng(req.body.start);
            if(start){
                latstart = start.lat;
                lngstart = start.lng;
            }else return false;
            //location end
            let end = await data.latlng(req.body.end);
            if(end){
                latend = end.lat;
                lngend = end.lng;
            }else return false;

            let res = {
                name: [],
                price: [],
                dist: [],
                time: []
            };

            data.Obj(res,await data.Maxim(req.body.start, req.body.end, latstart,
lngstart, latend, lngend));
            data.Obj(res,await data.Rutaxi(latstart, lngstart, latend, lngend));
            data.Obj(res,await data.Yandex(latstart, latend, lngstart, lngend));
            data.Obj(res,await data.Uber(latstart, lngstart, latend, lngend));
            data.Obj(res,await data.Uber1(latstart, lngstart, latend, lngend));

            return res;
        })();
        console.log(info);
        res.json(info).end();
    }
    else res.sendStatus(400);
});

module.exports = router;
```

---

public/data.js – файл получения данных

```
let request = require('request');
let promiserequest = require('request-promise');
let querystring = require('querystring');

let googleMapsClient = require('@google/maps').createClient({
    key: 'AIzaSyCIHU1rKJYh9Fg6EE6w_PEDYEH4ekMj2Rc',
    Promise: Promise
});

async function latlng(place) {
    try {
        const response = await googleMapsClient.geocode({
```

```
            address: place
        }).asPromise();

        if(response.json.status === 'OK'){
            return response.json.results[0].geometry.location;
        }
        else return false;
    } catch (err) {
        console.error(err);
    }
}

async function Yandex(latstart, latend, lngstart, lngend) {

    let options = {
        "id": "c8771ed244734f599cb90d646f01af0c",
        "zone_name": "chelyabinsk",
        "supports_forced_surge": true,
        "parks": [],
        "requirements": {},
        "route": [[lngstart, latstart], [lngend, latend]],
        "skip_estimated_waiting": false
    };

    try {
        let response = await promiserequest({
            url: "https://taxi.yandex.ru/3.0/routestats/",
            method: "POST",
            json: true,
            headers: {
                'Content-Type': "application/json",
                'Cookie': "yandexuid=2124394431518115630;
_id=c8771ed244734f599cb90d646f01af0c"
            },
            body: options
        });
        console.log(response);
        let res = {
            'name': [],
            'dist': [],
            'time': [],
            'price': []
        };
        if(response.service_levels) {
            res.time.push(response.time);
            res.dist.push(response.distance);

            for (let i = 0; i < response.service_levels.length; i++) {
                res.name.push("Yandex Taxi " + response.service_levels[i].name);
                res.price.push(response.service_levels[i].details[0].price);
            }
        }
        return await res;

    } catch (err) {
        console.error(err);
    }
}

async function Obj(res, objj){
    if(objj){
        if(objj.dist) {
```

```
            let i = 0;
            res.dist.push(objj.dist[i]);
            res.time.push(objj.time[i]);
        }
        for(let i=0;i<objj.name.length;i++){
            res.name.push(objj.name[i]);
            res.price.push(objj.price[i]);
        }
        return res;
    }
    else console.log(objj);

}

async function Rutaxi(latstart, lngstart, latend, lngend) {

    let idstr = await IDRutaxi(latstart, lngstart);
    let idend = await IDRutaxi(latend, lngend);
    let idrate = await ratesRutaxi();

    let res = {
        'name': [],
        'price': []
    };

    for (let i = 0; i < idrate.length; i++) {
        let options = {
            "city": "chel",
            "Order": {
                "rate_id": idrate[i].id,
                "points": [
                    {
                        "object_id": idstr.id,
                        "house": idstr.house
                    },
                    {
                        "object_id": idend.id,
                        "house": idend.house
                    }],
                "is_preorder": 0
            }
        };

        try {
            let response = await promiserequest({
                url: "https://api.rutaxi.ru/api/1.0.0/cost/",
                method: "POST",
                headers: {
                    'Content-Type': "application/json",
                    'X-Parse-Application-Id': "App
g1iyt78Tou38ELWXYrBxLmBr0nqrB44c"
                },
                json: true,
                strictSSL: false,
                body: options
            });

            res.price.push(response.data.cost);
            res.name.push('Rutaxi ' + idrate[i].name);

        } catch (err) {
            console.error(err);
```

```
        }
    }
    return await res;
}

async function ratesRutaxi() {
    options = {
        "city": "chel"
    };
    try {
        let response = await promiserequest({
            url: "https://api.rutaxi.ru/api/1.0.0/rates/",
            method: "GET",
            json: true,
            headers: {
                'Content-Type': "application/json",
                'X-Parse-Application-Id': "App g1iyt78Tou38ELWXYrBxLmBr0nqrB44c"
            },
            strictSSL: false,
            body: options
        });

        return response.data.rates;

    } catch (err) {
        console.error(err);
    }
}

async function IDRutaxi(lat, lng) {

    options = {
        "latitude": lat,
        "longitude": lng,
        "city": "chel"
    };
    try {
        let response = await promiserequest({
            url: "https://api.rutaxi.ru/api/1.0.0/near/",
            method: "POST",
            json: true,
            headers: {
                'Content-Type': "application/json",
                'X-Parse-Application-Id': "App g1iyt78Tou38ELWXYrBxLmBr0nqrB44c"
            },
            strictSSL: false,
            body: options
        });
        return response.data.objects[0];

    } catch (err) {
        console.error(err);
    }
    //console.log(JSON.stringify(response.body,'',4));
}

async function Maxim(start, end, latstart, lngstart, latend, lngend) {

    let csrf = await getToken();

    Token = csrf.token;
    cookie = csrf.cookie;
```

```
    let res = {
        'name': [],
        'price': []
    };

    let tariff = {
        'id': [1,2,3],
        'name':['Эконом','Комфорт','Бизнес']

    };
    for (let i = 0; i<tariff.id.length; i++) {
        options = {
            'OrderForm[baseId]': 4,
            'OrderForm[tariffId]': tariff.id[i],
            'AddressForm[0][pointField]': start,
            'AddressForm[0][latitude]': latstart,
            'AddressForm[0][longitude]': lngstart,
            'AddressForm[1][pointField]': end,
            'AddressForm[1][latitude]': latend,
            'AddressForm[1][longitude]': lngend
        };
        //const formBody1 = Object.keys(options).map(key =>
encodeURIComponent(key) + '=' + encodeURIComponent(options[key])).join('&');
        let formBody = querystring.stringify(options);

        try {
            let response = await promiserequest({
                url:
"https://client.taxsee.com/service/calculate/?org=maxim&baseId=4&tax-
id=SrvMpYHWr%2FWymRmBUdoa6isCaQcmnFtb0xmL4kP74iIgk8vmpGZoL%2BOsQU0m3lMOMwpCkSyA%2F
U4qVJmAQaXbTw%3D%3D",
                method: "POST",
                headers: {
                    'Content-Type': "application/x-www-form-urlencoded",
                    'X-CSRF-Token': Token,
                    'Cookie': cookie
                },
                body: formBody
            });
            if (response) {
                let from = response.indexOf('price', 30);
                let to = response.search('/span');
                let go = response.substring(from + 8, to - 1);
                res.price.push(go);
                res.name.push('Maxim ' + tariff.name[i]);
            }
        } catch (err) {
            console.error(err);
        }
    }

    return res;
}

async function getToken() {

    try {
        let response = await promiserequest({
            url: "https://client.taxsee.com/frame/?tax-
id=SrvMpYHWr%2FWymRmBUdoa6isCaQcmnFtb0xmL4kP74iIgk8vmpGZoL%2BOsQU0m3lMOMwpCkSyA%2F
U4qVJmAQaXbTw%3D%3D&fp=d526896f1310a3431da23ee6a55fb970&c=ru&l=ru&b=4&p=1&theme=ma
```

```
ximV2",
            method: "GET",
            resolveWithFullResponse: true
        });
        let token = response.body;
        let from = token.search('token');
        let to = token.search('<title>');
        token = token.substring(from + 16, to - 7); //Token

        let cookie = response.headers['set-cookie'];

        return {token: token, cookie: cookie};

    } catch (err) {
        console.error(err);
    }
}

async function Uber(latstart, lngstart, latend, lngend) {

    options = {
        "pickupLat": latstart,
        "pickupLng": lngstart,
        "destinationLat": latend,
        "destinationLng": lngend
    };

    let res = {
        'name': [],
        'price': []
    };

    let bodyopt = querystring.stringify(options);
    let site = "https://www.uber.com/api/fare-estimate?"

    try {
        let response = await promiserequest({
            url: site + bodyopt,
            method: "GET",
            json: true
        });

        for (let i = 0; i < response.prices.length; i++) {
            res.price.push(response.prices[i].fareString);
            res.name.push(response.prices[i].vehicleViewDisplayName);
        }

        return res;

    } catch (err) {
        console.error(err);
    }

}

async function Uber1(latstart, lngstart, latend, lngend) {

    options = {
        "start_latitude": latstart,
        "start_longitude": lngstart,
        "end_latitude": latend,
        "end_longitude": lngend
```

```
    };

    let res = {
        'name': [],
        'price': []
    };

    let bodyopt = querystring.stringify(options);
    let site = "https://api.uber.com/v1.2/estimates/price?";

    try {
        let response = await promiserequest({
            url: site + bodyopt,
            method: "GET",
            json: true,
            headers:{
                'Authorization': "Token 4d6ohusG9oItrfRuuI59Q1-4ka-J5Vahsu8I-2aU"
            }
        });
        console.log(response);
        if(response.prices) {
            for (let i = 0; i < response.prices.length; i++) {
                res.price.push(response.prices[i].estimate);
                res.name.push(response.prices[i].display_name);
            }
        }else console.log(response);
        return res;

    } catch (err) {
        console.error(err);
    }
}

module.exports = {
    latlng: latlng,
    Yandex: Yandex,
    Rutaxi: Rutaxi,
    Maxim: Maxim,
    Uber: Uber,
    Obj: Obj,
    Uber1: Uber1
};
```

---

public/javascripts/gmap.js – файл загрузки карты в клиенте

```
var data = {
   latA:0,
   latB:0,
   lngA:0,
   lngB:0,
   duration:0,
   distance:0
};

var auto_start;
var auto_end;

function activatePlacesSearch(){
   var start = document.getElementById('start');
   var end = document.getElementById('end');
```

```
    var defaultBounds = new google.maps.LatLngBounds(
       new google.maps.LatLng(55.16, 61.354)
    );
    var opt     = {
       bounds: defaultBounds
    };
    auto_start = new google.maps.places.Autocomplete(start,opt);
    auto_end = new google.maps.places.Autocomplete(end,opt);

}

function geocodeAddress(geocoder) {

     var start = document.getElementById('start').value;
    var end = document.getElementById('end').value;

    geocoder.geocode({'address': start}, function(results, status) {
         if (status === 'OK') {
             console.log(results[0]);
          data.latA =  results[0].geometry.location.lat();
          data.lngA =  results[0].geometry.location.lng();
          markroute();
          update();
         }
       else {
             console.log('Geocode was not successful for the following reason: ' +
status);
             $('#circle').hide();
             $('#go').prop('disabled', false);
          riseErr('start');
       }
    });
    geocoder.geocode({'address': end}, function(results, status) {
         if (status === 'OK') {
            console.log(results[0]);
          data.latB = results[0].geometry.location.lat();
          data.lngB = results[0].geometry.location.lng();
          markroute();
          update();
         }
       else {
          console.log('Geocode was not successful for the following reason: ' +
status);
             $('#circle').hide();
             $('#go').prop('disabled', false);
          riseErr('end');
       }
    });
    google.maps.event.addListener(auto_start, 'place_changed', function() {
       console.log('Place changed start');
       resetData('start');
       update();
    });
    google.maps.event.addListener(auto_end, 'place_changed', function() {
       console.log('Place changed end');
       resetData('end');
       update();
    });

}
function markroute(){
```

```
    var options = {
        zoom:12,
        center:{
            lat:55.16,
            lng:61.354
        }
    }

    var map = new google.maps.Map(document.getElementById('map'),options);

    var directionsService = new google.maps.DirectionsService;
    var directionsDisplay = new google.maps.DirectionsRenderer;

    directionsDisplay.setMap(map);
    directionsDisplay.setOptions( { suppressMarkers: true } );

    var pointA = new google.maps.LatLng(data.latA, data.lngA);
     var pointB = new google.maps.LatLng(data.latB, data.lngB);

    var marker = new google.maps.Marker({
        map: map,
        position: pointA,
        label: 'A'
    });
    var marker = new google.maps.Marker({
        map: map,
        position: pointB,
        label: 'B'
    });
    calcRoute(directionsDisplay,directionsService,pointA,pointB);

}
function calcRoute(directionsDisplay,directionsService,pointA,pointB) {
    var request = {
        origin: pointA,
        destination: pointB,
        travelMode: 'DRIVING'
    };
    directionsService.route(request, function(result, status) {
        if (status == 'OK') {
            //console.log("pop",request,result,data);
            directionsDisplay.setDirections(result);
        }
        else{
            //console.log(request);

        }
    });
}
function resetData(type) {
    if (type != 'end') {
        data.latA = 0;
        data.lngA = 0;
    }
    if (type != 'start') {
        data.latB = 0;
        data.lngB = 0;
    }
    data.duration = 0;
    data.distance = 0;
}
```

```
function update() {
   var place_start = auto_start.getPlace();
   if (place_start) {
      if (place_start.geometry) {
         data.latA = place_start.geometry.location.lat();
         data.lngA = place_start.geometry.location.lng();
      }
   }
   var place_end = auto_end.getPlace();
   if (place_end) {
      if (place_end.geometry) {
         data.latB = place_end.geometry.location.lat();
         data.lngB = place_end.geometry.location.lng();
      }
   }

   if (data.latA != 0 && data.lngA != 0
      && data.latB != 0 && data.lngB != 0)
   {
      var origin = new google.maps.LatLng(data.latA, data.lngA);
      var destination = new google.maps.LatLng(data.latB, data.lngB);
      var directionsService = new google.maps.DirectionsService();
      var request = {
         origin: origin, // LatLng|string
         destination: destination, // LatLng|string
         travelMode: google.maps.DirectionsTravelMode.DRIVING,
         durationInTraffic: true
      };

      directionsService.route(request, function( response, status ) {
         if ( status === 'OK' ) {
            var point = response.routes[ 0 ].legs[ 0 ];
            data.duration = point.duration.value;
            data.distance = point.distance.value;
         } else {
            // console.log('Couldnt update route params!');
         }
      });
   } else {
      // console.log('Lattitude and longitude not ready yet!');
   }
}

function riseErr(mes){
   if (mes === 'both'){
        let res = '<h5>Ошибка полей ввода:</h5>' +
         '<ul id = "error" class="list-group md-3">' +
            '<li> Введите адрес отправления </li>' +
            '<li> Введите адрес назначения </li>' +
            '</ul>';
        $('#err').html(res).show();
    }else if (mes === 'nullstart'){
        let res = '<h5>Ошибка полей ввода:</h5>' +
            '<ul id = "error" class="list-group md-3">' +
            '<li> Введите адрес отправления </li>' +
            '</ul>';
        $('#err').html(res).show();
    }else if (mes === 'nullend'){
        let res = '<h5>Ошибка полей ввода:</h5>' +
            '<ul id = "error" class="list-group md-3">' +
            '<li> Введите адрес назначения </li>' +
            '</ul>';
```

```
        $('#err').html(res).show();
    }else if (mes === 'start'){
        let res = '<h5>Ошибка полей ввода:</h5>' +
            '<ul id = "error" class="list-group md-3">' +
            '<li> Некорректный адрес отправления или назначения </li>' +
            '</ul>';
        $('#err').html(res).show();
    }else if (mes === 'end'){
        let res = '<h5>Ошибка полей ввода:</h5>' +
            '<ul id = "error" class="list-group md-3">' +
            '<li> Некорректный адрес отправления или назначения </li>' +
            '</ul>';
        $('#err').html(res).show();
    }else if (mes === 'same'){
         let res = '<h5>Ошибка полей ввода:</h5>' +
             '<ul id = "error" class="list-group md-3">' +
             '<li> Адреса не должны совпадать </li>' +
             '</ul>';
        $('#err').html(res).show();
    }

}
```

---

views/index.pug – представление главной страницы

```
extends layout

block content
  div.row
   #menu.col-xs-4.col-md-4
    #forma.container
     h1.display-4.mb-4 ВыбериТакси
     form(method="POST")
      #err.alert.alert-danger
      div(class="input-group mb-3")
       div(class="input-group-prepend")
        span(class="input-group-text") От
       input#start.form-control(type="text" placeholder='Адрес отправления'
name='start')
      div(class="input-group mb-3")
       div(class="input-group-prepend")
        span(class="input-group-text") До
       input#end.form-control(type='text', placeholder='Адрес назначения'
name='end')
      input#csrf(type="hidden" name = "_csrf" value = csrfToken)
      button#go.btn.btn-primary.mr-4(type="submit") Сравнить
    #circle.marginLeft
     h2 Loading...
     #circle_1.circle
     #circle_2.circle
     #circle_3.circle
     div.clearfix
    #timedist.container
   #map.col-xs-8.col-md-8
  div.container
   #data.col-xs-6
   script.
       $(function () {
           $('form').submit(function (event) {
                event.preventDefault();
```

```
                let start = document.getElementById('start').value;
                let end = document.getElementById('end').value;
                let csrf = document.getElementById('csrf').value;
                const geocoder = new google.maps.Geocoder();

                if((start === "")&&(end === "")){
                    riseErr('both');
                }else if (start === ""){
                    riseErr('nullstart');
                }else if (end === "") {
                    riseErr('nullend');
                } else if (end === start) {
                    riseErr('same');
                }else{
                    $('#err').html('').hide();
                    $('#timedist').hide();
                    $('#circle').show();
                    $('#go').prop('disabled', true);
                    let options = {
                        "start": start,
                        "end": end,
                        "_csrf": csrf
                    };

                    geocodeAddress(geocoder);

                    const formBody = Object.keys(options).map(function (key) {
                        return encodeURIComponent(key) + '=' +
encodeURIComponent(options[key])
                    }).join('&');
                    fetch('http://localhost:3000', {
                        method: 'POST',
                        headers: {
                            'Accept': 'application/json, text/plain ,*/*',
                            'Content-type': 'application/x-www-form-urlencoded;
charset=UTF-8'
                        },
                        body: formBody,
                        mode: 'cors',
                        credentials: 'include'
                    })
                    .then(function (res) {
                        if (res.ok) {
                            return res.json();
                        }
                        else {
                            throw new Error();
                        }
                    })
                    .then(function (data) {
                        console.log(data);
                        if(data) {
                            $('#circle').hide();
                            $('#timedist').show();
                            let dat = document.getElementById('data');
                            let ul = '<ul class="list-group md-3">' +
                                '<li class = "list-group-item"> Route distanse: '
+ data.dist[0] + ' </li>' +
                                '<li class = "list-group-item">Route time: ' +
data.time[0] + '</li>' +
                                '</ul>';
                            let table = '<table class = "table table-hover
```

```
sortable">';
                                table += '<thead class="thead-dark" >' +
                                    '<tr class="bg-dark text-white"><td
scope="col">Name of a Taxi</td>' +
                                    '<td scope="col">Price of a Taxi</td></tr>' +
                                    '</thead>';
                                for (let i = 0; i < data.name.length; i++) {
                                    table += '<tr>';

                                    table += '<td>';
                                    table += data.name[i];
                                    table += '</td>';

                                    table += '<td>';
                                    table += data.price[i];
                                    table += '</td>';

                                    table += '</tr>';
                                }
                                ;
                                table += '</table>';
                                dat.innerHTML = table;
                                $('#timedist').html(ul);
                                $('#go').prop('disabled', false);
                            }
                        })
                        .catch(function (err) {
                            console.log(err.message);
                        });

                }
            });
        });
   script(async, defer,
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCIHU1rKJYh9Fg6EE6w_PEDYEH4e
kMj2Rc&libraries=places&callback=activatePlacesSearch")
```