Министерство науки и высшего образования Российской Федерации

федеральное государственное бюджетное образовательное учреждение

высшего образования

«Санкт-Петербургский государственный университет промышленных технологий и дизайна»

|  |
| --- |
| **Институт информационных технологий и автоматизации** |

|  |  |
| --- | --- |
| Кафедра: | Цифровых и аддитивных технологий |
| Направление подготовки: | 09.03.03 Прикладная информатика |
| Профиль подготовки: | Цифровые технологии в финансах |

|  |
| --- |
| **КУРСОВАЯ РАБОТА**  **(Курсовой проект)** |

|  |  |  |
| --- | --- | --- |
| на тему |  | Разработка программы с использованием контейнерного класса стандартной библиотеки С++. Разработка сайта автомобильных перевозок. Тарификация и маршруты. |
|  | | |
| по дисциплине |  | Алгоритмизация и программирование |
|  | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Исполнитель - студент учебной группы: | |  | 2-МД-21 |
|  | |  | *(группа)* | |
|  | |  | Черненко З. В./ | |
|  | |  | | *(Ф.И.О./ подпись)* | |

|  |  |  |
| --- | --- | --- |
| Руководитель курсовой работы: |  | Стяжкин А. А./ |
|  | |  | *(должность, ученая степень / звание, Ф.И.О../ подпись)* |

Санкт-Петербург

2024

**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**высшего образования**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ И ДИЗАЙНА»**

|  |  |
| --- | --- |
| **Утверждаю** | **Институт информационных**  **технологий и автоматизации** |
| Заведующий кафедрой |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. В. Сошников | Кафедра Цифровых и аддитивных технологий |
| «\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 г. |  |

**ЗАДАНИЕ**

**на курсовую работу (проект)**

Студенту Черненко Захару Викторовичу

1. Тема курсовой работы (проекта) Реализация сайта грузоперевозок

2. Срок сдачи студентом законченной курсовой работы (проекта) 25.05.2024\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные по курсовой работе (проекту) Изучить контейнеры и разработать программу на C++\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Перечень подлежащих разработке в курсовой работе (проекте) вопросов или краткое содержание работы:

Введение: реализовать хранение данных на основе контейнеров C++ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Постановка задачи: Создание сайта, на котором пользователи могут считать стоимость доставки и делать заказы. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Описание контейнеров: в работе описаны принципы работы контейнеров map, unordered\_map и vector \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Математическая модель решения задачи: Для описания предметной области данной задачи была построена математическая модель.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Блок-схема решения задачи: В проекте представлена блок-схема решения задачи.

Разработка структуры данных: В данном проекте представлен код программы и его скриншоты.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководство пользователя: В проекте подробно расписана инструкция по использованию программы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Тестовые примеры: Были проведены тестовые запуски программы. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Заключение: В результате выполнения проекта было создано веб-приложение, которое может быть использовано в бизнесе фирмами, осуществляющими грузоперевозки.\_\_\_\_\_

Дата выдачи задания \_\_\_\_\_\_\_\_\_\_\_\_\_Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Стяжкин А.А. (дата) (подпись) (ФИО)

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Черненко З. В.

(дата) (подпись) (ФИО)

СОДЕРЖАНИЕ

[3](#_Toc167893001)

[ПОСТАНОВКА ЗАДАЧИ 4](#_Toc167893002)

[ОПИСАНИЕ КОНТЕЙНЕРА MAP 5](#_Toc167893003)

[ОПИСАНИЕ КОНТЕЙНЕРА UNORDERED\_MAP 6](#_Toc167893004)

[ОПИСАНИЕ КОНТЕЙНЕРА VECTOR 7](#_Toc167893005)

[МАТЕМАТИЧЕСКАЯ МОДЕЛЬ РЕШЕНИЯ ЗАДАЧИ 8](#_Toc167893006)

[БЛОК-СХЕМА РЕШЕНИЯ ЗАДАЧИ 9](#_Toc167893007)

[РАЗРАБОТКА СТРУКТУРЫ ДАННЫХ 10](#_Toc167893008)

[РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 12](#_Toc167893009)

[ТЕСТИРОВАНИЕ ПРОГРАММЫ 22](#_Toc167893010)

[ВЫВОДЫ ПО КУРСОВОЙ РАБОТЕ 24](#_Toc167893011)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 25](#_Toc167893012)

# ПОСТАНОВКА ЗАДАЧИ

Цель данной курсовой работы - закрепить и углубить теоретические знания, получив практические навыки решения комплексных задач. В процессе работы планируется развить навыки аналитической и расчетной деятельности, а также работы со специализированной литературой и другими источниками информации.

Задачей курсовой работы было создание программы по тарификации различных автомобильных маршрутов. Было решено создать веб-приложение, которое бы позволило пользователю зарегистрироваться на сайте, выбрать город отправления и город прибытия, а также автомобиль для перевозки груза. Также была добавлена авторизация для возможности сохранения всех добавленных пользователем заказов. В ходе работы были использованы такие контейнеры стандартной библиотеки как: vector, map, unordered\_map.

В результате проделанной работы было создано функциональное веб-приложение, позволяющее автоматизировать процесс тарификации автомобильных маршрутов. Разработанное приложение может быть использовано как логистическими компаниями, так и частными лицами, планирующими грузоперевозки.

# ОПИСАНИЕ КОНТЕЙНЕРА MAP

std::map в C++ - это контейнер, который хранит пары "ключ-значение", упорядоченные по ключу. Он реализован как сбалансированное бинарное дерево поиска (обычно красно-черное). Красно-черное дерево — это самобалансирующееся бинарное дерево поиска, которое гарантирует логарифмическую сложность для основных операций (вставка, удаление, поиск). Оно используется для реализации ассоциативных массивов, множеств и других структур данных.

Основные свойства:

1. Каждый узел имеет цвет: красный или черный.
2. Корень дерева всегда черный.
3. Все листья (NIL-узлы) черные.
4. Если узел красный, то оба его потомка черные.
5. Каждый путь от узла до любого его потомка-листа содержит одинаковое количество черных узлов.

При добавлении или удалении узла дерево может потерять свои свойства. Чтобы восстановить баланс, красно-черные деревья используют операции вращения и перекрашивания узлов. Вращение изменяет структуру дерева, а перекрашивание меняет цвет узлов, сохраняя при этом упорядоченность дерева.

Асимптотическая сложность:

1. Вставка (insert) - O(log n)
2. Поиск (find) - O(log n)
3. Удаление (erase) - O(log n)
4. Доступ по ключу - O(log n)
5. Проход по элементам - O(n)

# ОПИСАНИЕ КОНТЕЙНЕРА UNORDERED\_MAP

std::unordered\_map в C++ - это контейнер, который хранит пары "ключ-значение", не упорядоченные по ключу. Он реализован как хэш-таблица. Хэш-таблица — это структура данных, которая использует хэш-функцию для вычисления индекса (индекс хэша) для каждого ключа, который затем используется для прямого доступа к соответствующему значению.

Основные свойства:

1. Хранение элементов в buckets: Элементы с одинаковым хэш-значением попадают в один и тот же бакет (bucket).
2. Разрешение коллизий: при возникновении коллизий (когда разные ключи имеют одинаковый хэш) применяются различные стратегии, например, цепочки (separate chaining) или открытая адресация (open addressing).
3. Динамическое изменение размера: при увеличении количества элементов хэш-таблица может увеличивать свой размер (rehashing) для поддержания производительности.

Асимптотическая сложность:

1. Вставка (insert) - В среднем: O(1), в худшем случае: O(n)
2. Поиск (find) - В среднем: O(1), в худшем случае: O(n)
3. Удаление (erase) - В среднем: O(1), в худшем случае: O(n)
4. Доступ по ключу - В среднем: O(1), в худшем случае: O(n)
5. Проход по элементам - O(n)

# ОПИСАНИЕ КОНТЕЙНЕРА VECTOR

std::vector в C++ - это контейнер, который хранит элементы последовательно в памяти. Он похож на динамический массив, который может автоматически увеличиваться или уменьшаться в размере по мере необходимости.

Основные свойства:

1. Хранение элементов в непрерывной области памяти: это обеспечивает быстрый доступ к элементам по индексу (случайный доступ).
2. Динамическое изменение размера: При добавлении элементов, если текущая емкость недостаточна, std::vector автоматически выделяет новую область памяти большего размера и копирует туда все существующие элементы.
3. Поддержка итераторов: std::vector предоставляет итераторы для эффективного доступа и обхода элементов.

Асимптотическая сложность:

1. Вставка (insert):
   1. В конец (push\_back): В среднем: O(1) амортизированно, в худшем случае: O(n) при реаллокации.
   2. Вставка в середину: O(n) - требуется сдвинуть последующие элементы.
2. Доступ по индексу: O(1) - прямой доступ к элементу.
3. Удаление (erase):
   1. Из конца (pop\_back): O(1)
   2. Из середины: O(n) - требуется сдвинуть последующие элементы.
4. Поиск (find): O(n) - линейный поиск, так как элементы не отсортированы.
5. Проход по элементам: O(n)

# МАТЕМАТИЧЕСКАЯ МОДЕЛЬ РЕШЕНИЯ ЗАДАЧИ

Формула расчета стоимости грузоперевозки по городам России с учетом добровольной страховки от стоимости перевозки:

Стоимость = (Базовая стоимость + (Расстояние \* СтоимостьКилометра) + ДополнительныеУслуги) \* (1 + ПроцентСтраховки (опционально)) \* КоэффициентТипаМашины

Процент страховки был взят, равным 3% от общей стоимости перевозки. Коэффициент типа машины был взять в 1 у обычной бортовой Газели, 1.1 у Газели пирамиды, 1.2 у тентованной Газели и 2 у грузовика фургона.

Код расчета цены:

**double** countPrice(**double** default\_price, **double** coeff, **double** distance, **bool** is\_ensurance) {

**double** price {default\_price};

price += distance \* PRICE\_PER\_KM;

price \*= coeff;

**if** (is\_ensurance)

price \*= 1.03;

**return** price;

}

# **БЛОК-СХЕМА РЕШЕНИЯ ЗАДАЧИ**

На рисунке 1 показана блок-схема обработки посещения пользователями сайта.

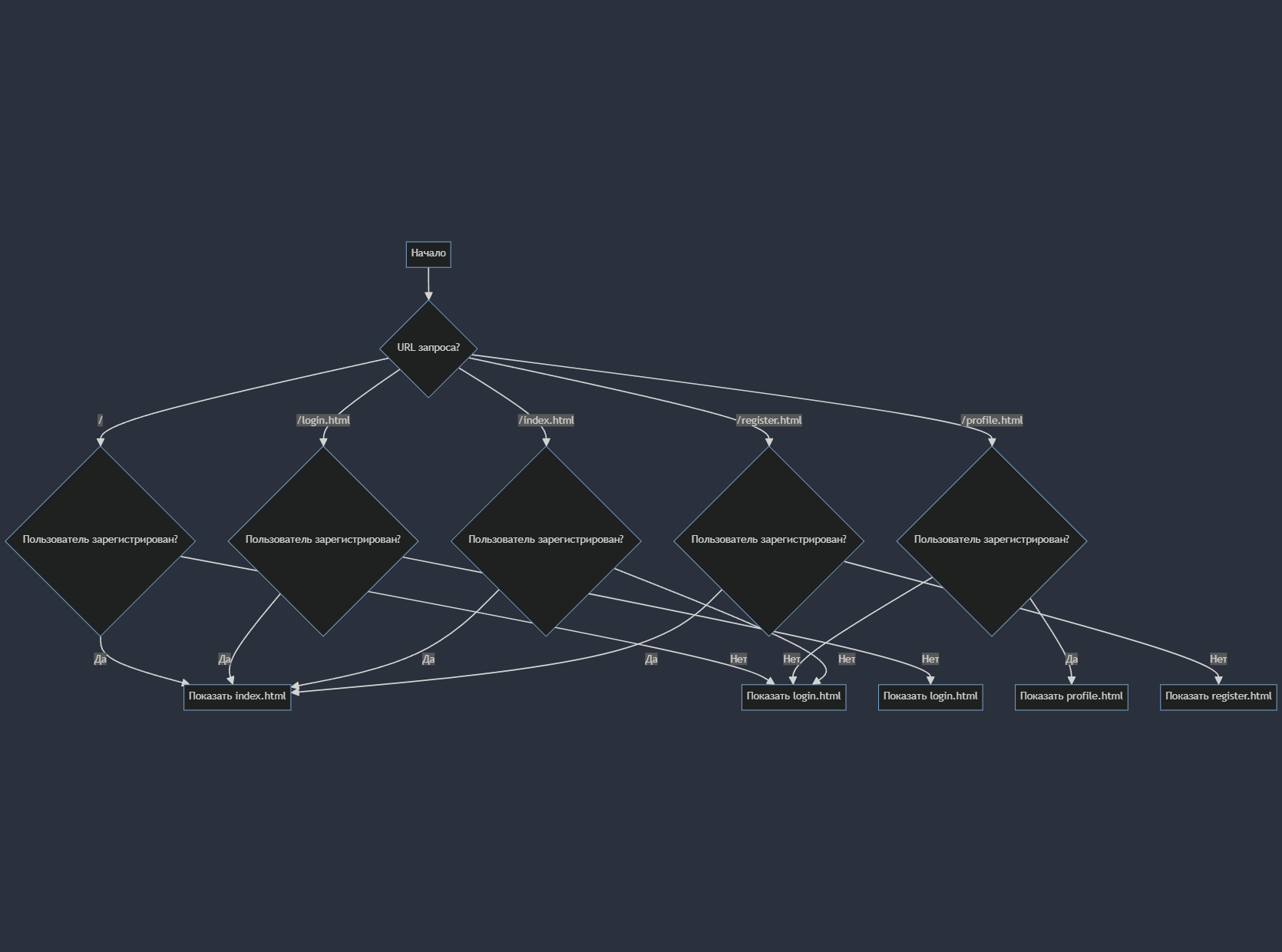


Рисунок 1 — блок-схема работы сайта

# РАЗРАБОТКА СТРУКТУРЫ ДАННЫХ

На рисунке 2 представлены структуры данных в отношении друг к другу

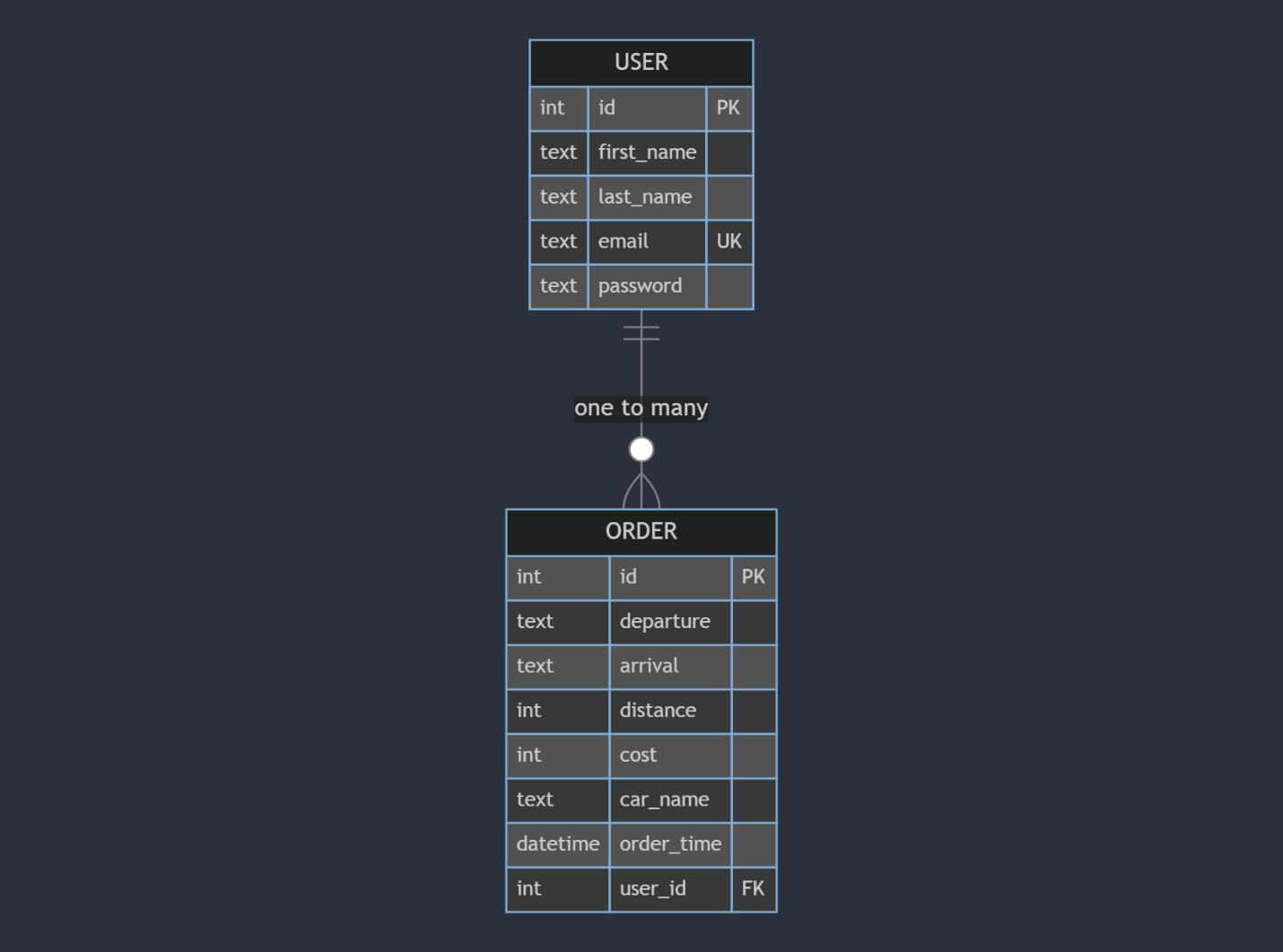


Рисунок 2 — Диаграмма классов

В коде же классы представлены так:

* Класс пользователя

**class** User {

**public**:

User(**const** drogon::orm::Row& row);

User(**const** std::**string**& first\_name, **const** std::**string**& last\_name, **const** std::**string**& email,

**const** std::**string**& password);

std::**string** getInsertQuery() **const**;

std::**string** getUserId() **const**;

**protected**:

**const** std::**string** first\_name, last\_name, email, password;

};

* Класс заказа:

**class** Order {

**public**:

Order();

Order(**const** drogon::orm::Row& row);

Order(**const** std::**string**& arrival, **const** std::**string**& departure, **const** std::**string**& distance,

**const** std::**string**& price, **const** std::**string**& car\_name, **const** std::**string**& order\_data = "");

**bool** **operator**==(**const** Order& other) **const**;

Order& **operator**=(**const** Order& other);

std::**string** getInsertQuery(**const** std::**string**& user\_id) **const**;

std::**string** getArrival() **const**;

std::**string** getDeparture() **const**;

std::**string** getDistance() **const**;

std::**string** getPrice() **const**;

std::**string** getCarName() **const**;

std::**string** getOrderData() **const**;

**protected**:

std::**string** arrival, departure, distance, price, car\_name, order\_data;

};

# РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Изначально пользователя встречает экран логина, в который необходимо ввести свои email и пароль, если пользователь уже зарегистрирован.

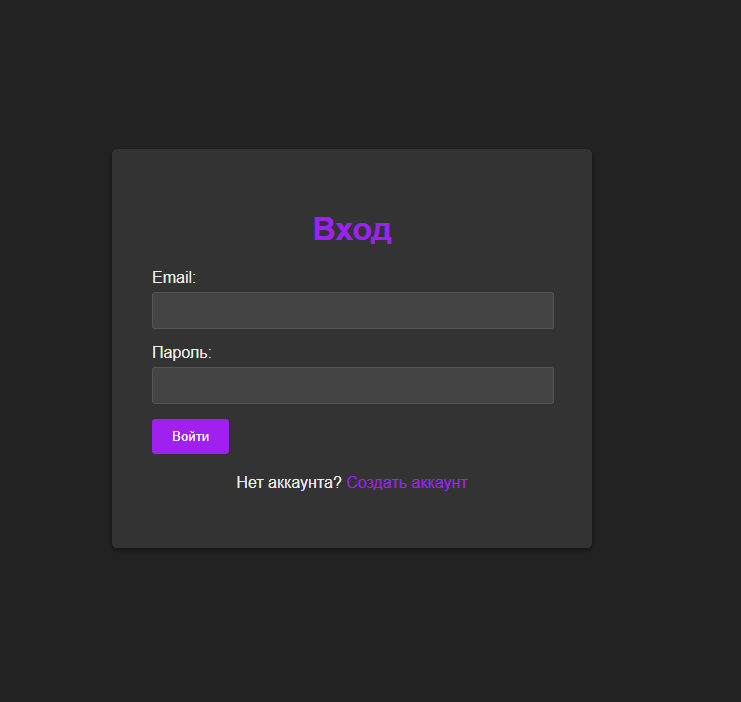


Рисунок 3 — Экран входа

Если вы не зарегистрированы, то можно создать аккаунт, нажав на «Создать аккаунт»

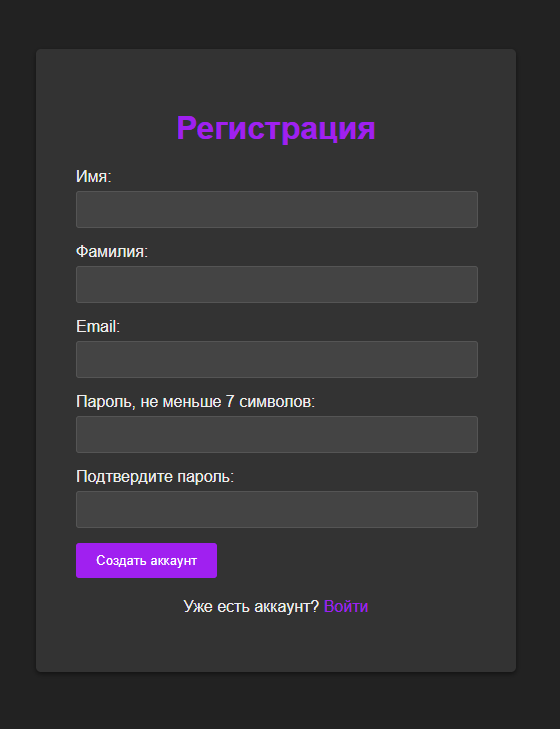


Рисунок 4 — Экран регистрации

Далее при успешном входе или регистрации вы будете перенаправлены на страницу index.html

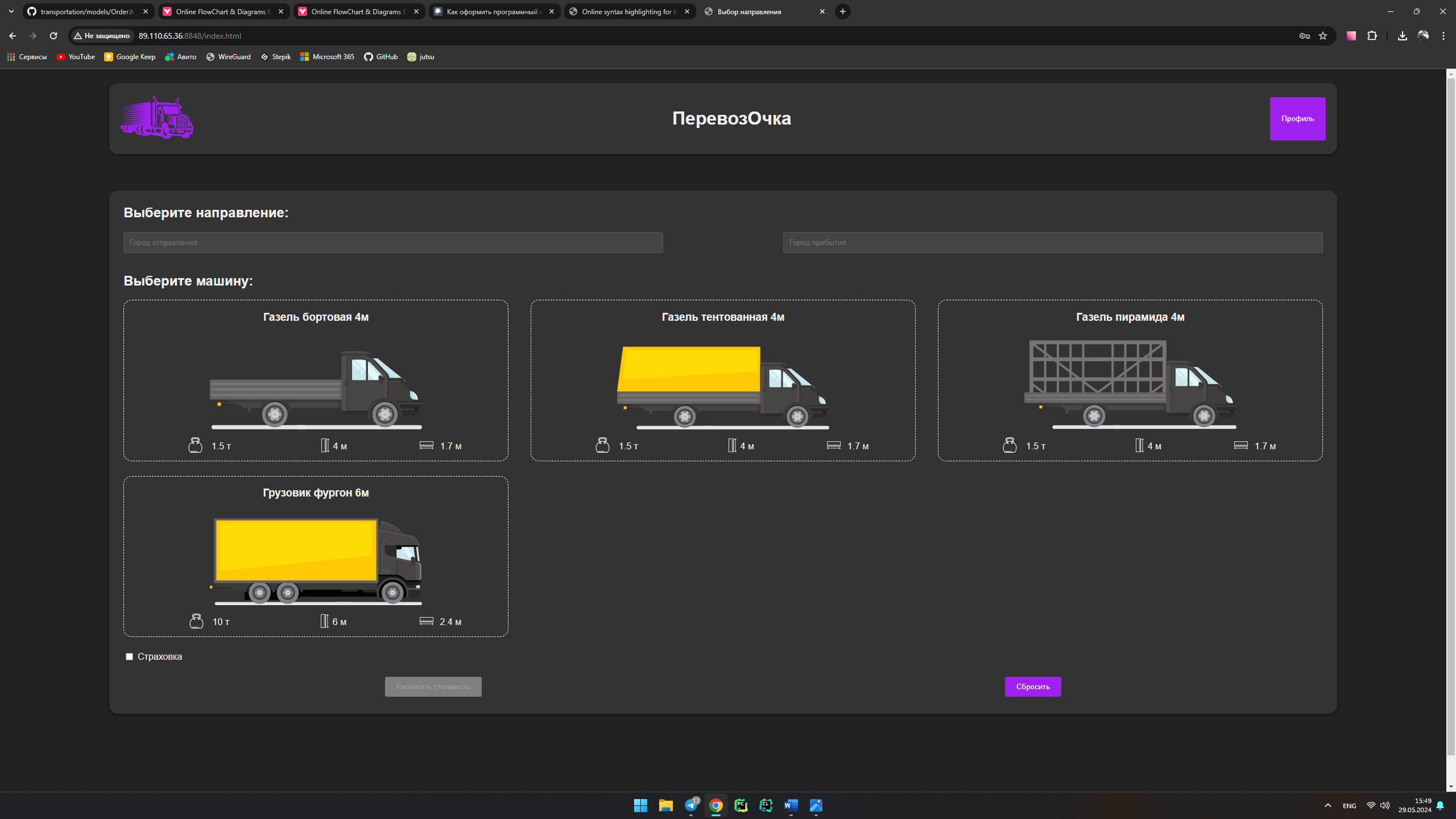


Рисунок 5 — Главная страница

Здесь можно выбрать пункты отправления и назначения, вводя текст, а браузер будет отправлять данные на сервер, который будет искать совпадения в базе данных городов. Под полем ввода появится список городов, которые начинаются с этой последовательности символов (Часть экрана слева). Чтобы это поле появилось браузер отправляет запрос по url /autocomplete/departure/<введенная последовательность> для дополнения пункта отправки, и /autocomplete/arrival/<введенная последовательность> для пункта прибытия.

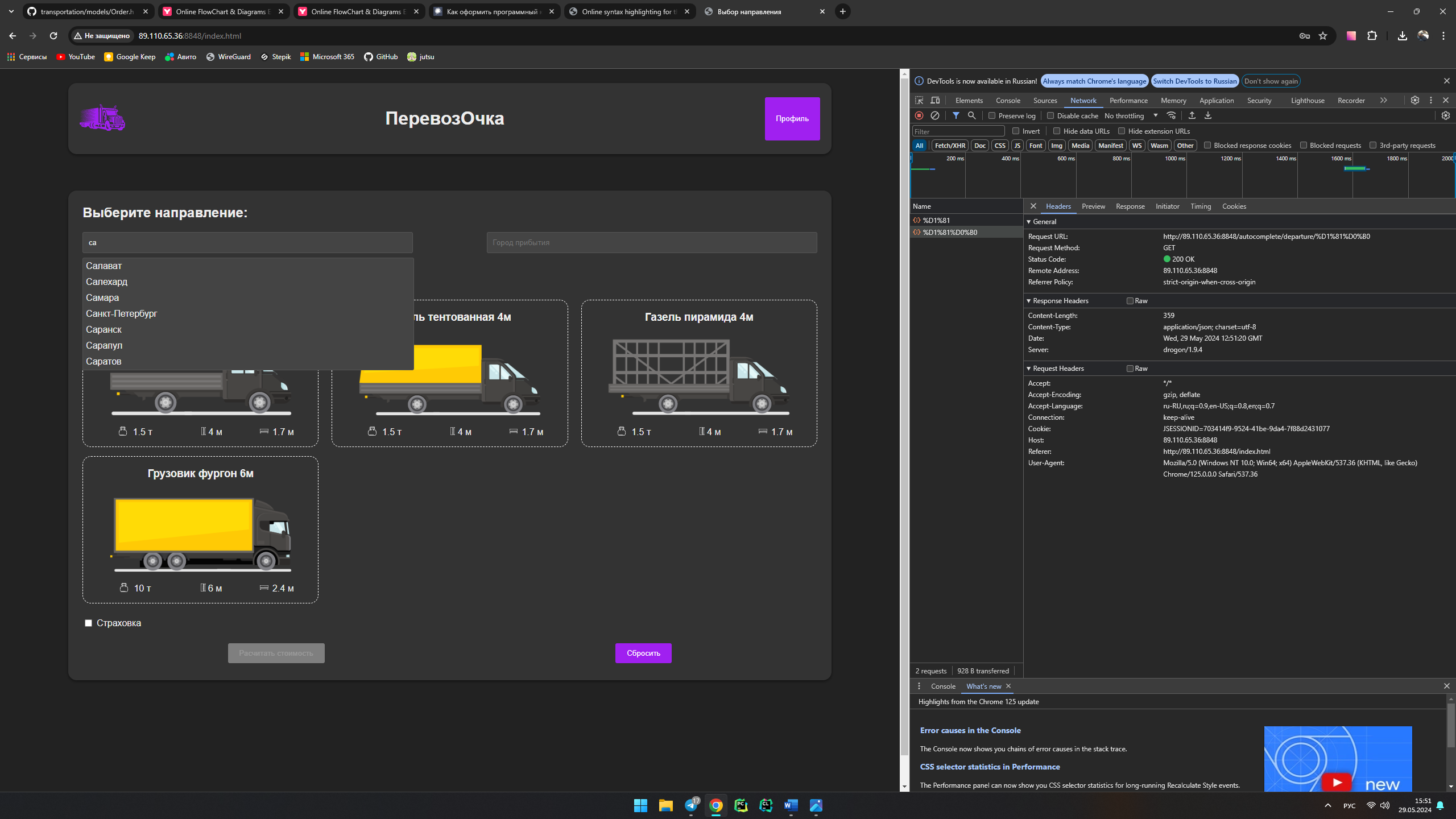


Рисунок 6 — Принцип работы автодополнения

Сервер отправит json сообщение с названиями городов, начинающихся с данной последовательности

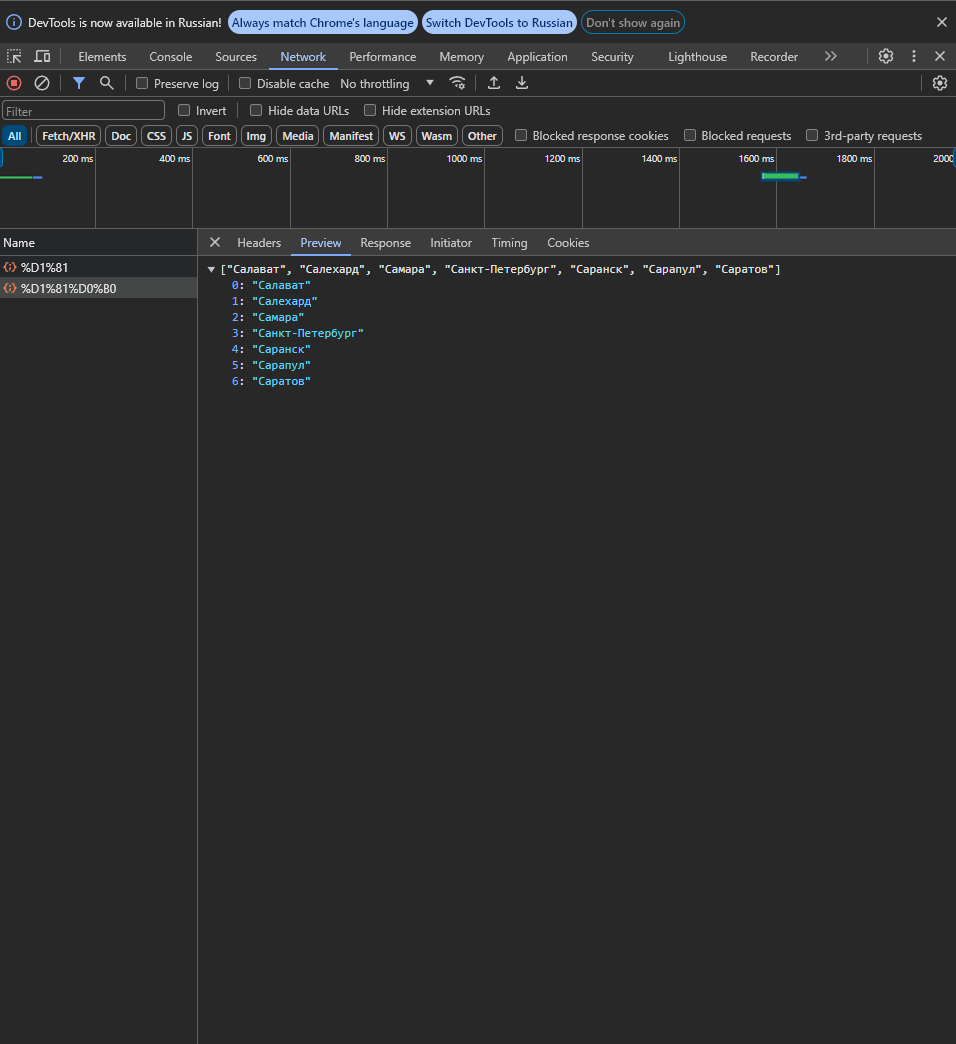


Рисунок 7 – Ответ сервера

Реализация функционала на языке С++ выглядит так:

**void** autocompleteIndexInputs(**const** drogon::HttpRequestPtr& req, Callback&& callback, **const** std::**string**& city\_name) {

drogon::orm::DbClientPtr client = drogon::app().getDbClient("cities");

std::**string** city\_name\_lower {toRussianLowercase(city\_name)};

client->execSqlAsync(

"SELECT city FROM cities WHERE city LIKE '" + city\_name\_lower + "%';",

[callback](**const** drogon::orm::Result& result) { // & causes seg fault

Json::Value cities\_array {Json::arrayValue};

**for** (**const** drogon::orm::Row& row : result)

cities\_array.append(capitalize(row["city"].as<std::**string**>()));

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(cities\_array)};

callback(response);

},

[callback](**const** drogon::orm::DrogonDbException& e) {

LOG\_ERROR << e.base().what() << '\n';

Json::Value json;

json["message"] = "Неполадки на сервере";

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k500InternalServerError);

callback(response);

});

}

1. Функция autocompleteIndexInputs:

- Принимает три аргумента:

- req: Указатель на объект HTTP-запроса Drogon, который содержит данные пользователя.

- callback: Функция обратного вызова, которая будет вызвана после завершения операции.

- city\_name: Строка, содержащая начальные символы названия города, введенные пользователем.

1. Получение клиента базы данных:

- drogon::orm::DbClientPtr client = drogon::app().getDbClient("cities");: Получает указатель на клиент базы данных с именем "cities".

1. Преобразование ввода пользователя:

- std::string city\_name\_lower {toRussianLowercase(city\_name)};: Создает новую строку city\_name\_lower, содержащую переданное название города, преобразованное в нижний регистр с учетом особенностей русского языка.

1. Выполнение SQL-запроса:

- client->execSqlAsync(...): Выполняет асинхронный SQL-запрос к базе данных.

- Запрос: "SELECT city FROM cities WHERE city LIKE '" + city\_name\_lower + "%';" выбирает все города из таблицы cities, названия которых начинаются с введенных пользователем символов (с учетом регистра).

- Два лямбда-выражения используются для обработки результата:

- Первое лямбда-выражения (успешное выполнение):

- Json::Value cities\_array {Json::arrayValue};: Создает пустой JSON-массив для хранения найденных городов.

- Цикл for (const drogon::orm::Row& row : result): Итерирует по каждой строке результата запроса.

- cities\_array.append(capitalize(row["city"].as<std::string>()));: Извлекает значение столбца city, приводит его к строке, делает первую букву заглавной и добавляет в cities\_array.

- drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(cities\_array)};: Создает HTTP-ответ с найденными городами в формате JSON.

- callback(response);: Вызывает функцию обратного вызова, передавая ей созданный HTTP-ответ.

- Второе лямбда-выражения (ошибка выполнения):

- LOG\_ERROR << e.base().what() << '\n';: Логирует ошибку, полученную из базы данных.

- Создает HTTP-ответ с сообщением об ошибке и кодом состояния 500 (Internal Server Error).

- callback(response);: Вызывает функцию обратного вызова, передавая ей HTTP-ответ с информацией об ошибке.

После ввода городов и выбора машины можно отправить запрос на сервер для расчета стоимости, тогда сервер обработает результат и отправит обратно все данные о заказе.

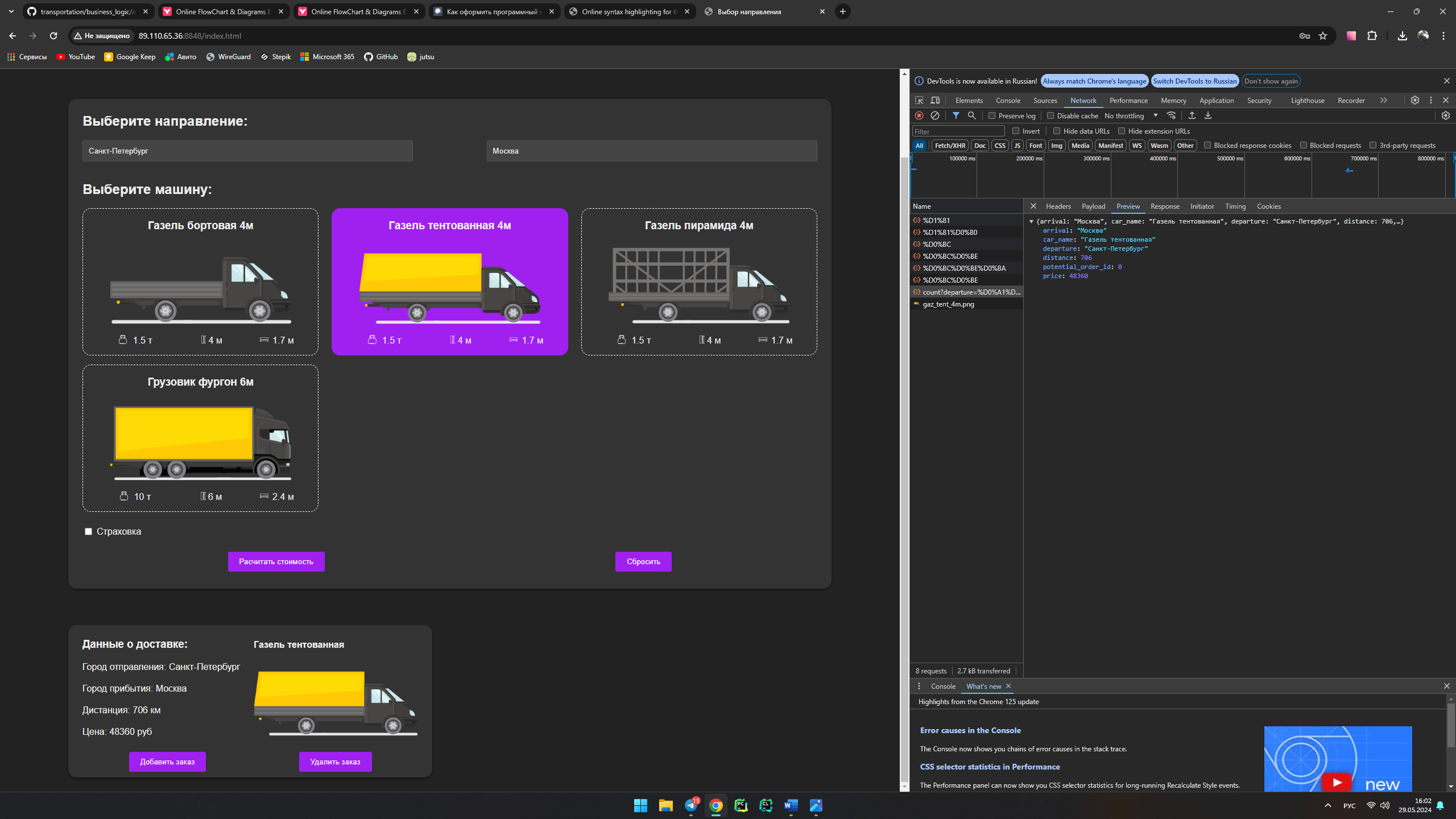


Рисунок 8 – Пример расчета цены

Код обработки запроса выглядит так:

**void** countHandler(**const** drogon::HttpRequestPtr& request, Callback&& callback, **const** std::**string**& departure,

**const** std::**string**& arrival, **const** std::**string**& car, **const** std::**string**& insurance) {

Json::Value json;

u\_int32\_t user\_id {request->session()->getOptional<u\_int32\_t>("userId").value\_or(0)};

**if** (!user\_id) {

json["message"] = "Произведен выход из акканута";

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k400BadRequest);

callback(response);

**return**;

}

drogon::orm::DbClientPtr client {drogon::app().getDbClient("cities")};

std::**string** departure\_lowercase {toRussianLowercase(departure)};

std::**string** arrival\_lowercase {toRussianLowercase(arrival)};

**if** (departure\_lowercase == arrival\_lowercase) {

json["message"] = "Город отправления совпадает с городом прибытия";

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k400BadRequest);

callback(response);

**return**;

}

**try** {

#ifdef BUSINESS\_HANDLER\_DEBUG

LOG\_INFO << "Город отправки: " << departure << '\n';

#endif

std::future<drogon::orm::Result> future\_check\_arrival {

client->execSqlAsyncFuture("SELECT EXISTS(SELECT 1 FROM cities WHERE city = $1);", departure\_lowercase)};

**bool** is\_arrival\_correct {future\_check\_arrival.get().front()[0].as<**bool**>()};

**if** (!is\_arrival\_correct) {

json["message"] = "Невалидный город отправки";

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k400BadRequest);

callback(response);

**return**;

}

#ifdef BUSINESS\_HANDLER\_DEBUG

LOG\_INFO << "Город прибытия: " << arrival << '\n';

#endif

std::future<drogon::orm::Result> future\_check\_distance {client->execSqlAsyncFuture(

"SELECT distance FROM '" + departure\_lowercase + "' WHERE city = $1", arrival\_lowercase)};

drogon::orm::Result distance\_result {future\_check\_distance.get()};

**if** (distance\_result.empty()) {

json["message"] = "Невалидный город прибытия";

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k400BadRequest);

callback(response);

**return**;

}

u\_int32\_t distance {distance\_result.front()[0].as<u\_int32\_t>()};

#ifdef BUSINESS\_HANDLER\_DEBUG

LOG\_INFO << "Расстояние между городами: " << distance << '\n';

LOG\_INFO << "Машина: " << car << '\n';

#endif

**auto** car\_ptr {cars\_unordered\_map.**find**(car)};

**if** (car\_ptr == cars\_unordered\_map.end()) {

json["message"] = "Неверно указана машина";

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k400BadRequest);

callback(response);

**return**;

}

#ifdef BUSINESS\_HANDLER\_DEBUG

LOG\_INFO << "Страховка: " << insurance << '\n';

#endif

**bool** is\_insurance {**false**};

**if** (insurance == "true")

is\_insurance = **true**;

**else** **if** (insurance != "false") {

json["message"] = "Нет данных о страховке";

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k400BadRequest);

callback(response);

**return**;

}

// all data is valid

**int** price {**static\_cast**<**int**>(countPrice(5000, (car\_ptr->second).getCostCoef(), distance, is\_insurance))};

std::**string** car\_name {(car\_ptr->second).getCarName()};

Order potential\_order {arrival\_lowercase, departure\_lowercase, std::to\_string(distance), std::to\_string(price),

car\_name};

u\_int32\_t potential\_order\_id;

request->session()->modify<std::**map**<u\_int32\_t, Order, std::**greater**<u\_int32\_t>>>(

"ordersMap", [&arrival\_lowercase, &departure\_lowercase, &distance, &price, &car\_name, &potential\_order,

&potential\_order\_id](std::**map**<u\_int32\_t, Order, std::**greater**<u\_int32\_t>>& orders\_map) {

**if** (orders\_map.empty())

potential\_order\_id = 0;

**else**

potential\_order\_id = orders\_map.begin()->first + 1;

orders\_map[potential\_order\_id] = potential\_order;

#ifdef BUSINESS\_HANDLER\_DEBUG

**for** (**const** **auto** **pair** : orders\_map)

**pair**.second.print();

#endif

});

json["departure"] = capitalize(departure\_lowercase);

json["arrival"] = capitalize(arrival\_lowercase);

json["distance"] = distance;

json["price"] = price;

json["car\_name"] = car\_name;

json["potential\_order\_id"] = potential\_order\_id;

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k200OK);

callback(response);

}

**catch** (**const** drogon::orm::DrogonDbException& e) {

LOG\_ERROR << e.base().what() << '\n';

json["message"] = "Проблема на стороне сервера, невозможно посчитать стоимость";

drogon::HttpResponsePtr response {drogon::HttpResponse::newHttpJsonResponse(json)};

response->setStatusCode(drogon::HttpStatusCode::k500InternalServerError);

callback(response);

**return**;

}

}

1. Функция принимает на вход HTTP-запрос (request), функцию обратного вызова (callback), а также строки, представляющие город отправления (departure), город прибытия (arrival), выбранную машину (car) и тип страховки (insurance).
2. Извлекает ID пользователя (user\_id) из сессии пользователя в HTTP-запросе. Если ID не найден (пользователь не авторизован), возвращает HTTP-ответ с кодом 400 (Bad Request) и сообщением "Произведен выход из акканута".
3. Валидация и обработка данных:
   1. Получает соединение с базой данных "cities".
   2. Переводит названия городов в нижний регистр.
   3. Проверяет, не совпадают ли города отправления и прибытия. Если совпадают, возвращает ошибку 400 с сообщением "Город отправления совпадает с городом прибытия".
   4. Проверяет существование города отправления в базе данных. Если город не найден, возвращает ошибку 400 с сообщением "Невалидный город отправки".
   5. Получает расстояние между городами из базы данных. Если расстояние не найдено, возвращает ошибку 400 с сообщением "Невалидный город прибытия".
   6. Ищет выбранную машину в cars\_unordered\_map (предположительно, это словарь или карта, содержащая информацию о машинах). Если машина не найдена, возвращает ошибку 400 с сообщением "Неверно указана машина".
4. Обработка страховки:
   1. Получает значение параметра insurance (строка "true" или "false").
   2. Конвертирует строку insurance в булево значение is\_insurance.
   3. Если значение insurance не равно "true" или "false", возвращает ошибку 400 с сообщением "Нет данных о страховке".
5. Расчет цены и создание заказа:
   1. Сохраняет результат countPrice в переменной price после преобразования к целому типу.
   2. Создает объект Order (заказ) с информацией о поездке:
   3. Генерирует уникальный ID для нового заказа (potential\_order\_id).
   4. Добавляет новый заказ в дерево заказов (ordersMap) в сессии пользователя.
6. Формирует JSON-ответ с информацией о поездке и заказе
7. Отправляет успешный HTTP-ответ с кодом 200 (OK) и сформированным JSON-объектом.
8. Обработка ошибок базы данных:
   1. В случае возникновения исключения drogon::orm::DrogonDbException (ошибка базы данных) логгирует ошибку.
   2. Отправляет HTTP-ответ с кодом 500 (Internal Server Error) и сообщением "Проблема на стороне сервера, невозможно посчитать стоимость".

# ТЕСТИРОВАНИЕ ПРОГРАММЫ

Тестирование программного кода представляет собой неотъемлемый этап разработки программного обеспечения, обеспечивающий его качество и надежность. Целью тестирования является обнаружение дефектов и несоответствий в коде, которые могут привести к ошибкам в процессе функционирования программного продукта.

Существует множество методов и уровней тестирования, применяемых на различных этапах жизненного цикла разработки ПО. Модульное тестирование фокусируется на проверке корректности работы отдельных модулей или компонентов программы. Интеграционное тестирование, напротив, направлено на выявление дефектов взаимодействия между этими модулями при их совместной работе.

Для тестирования кода было выбрано автоматическое тестирование с помощью языка shell и утилиты curl, которая позволяет отправлять и получать данные из интернета.

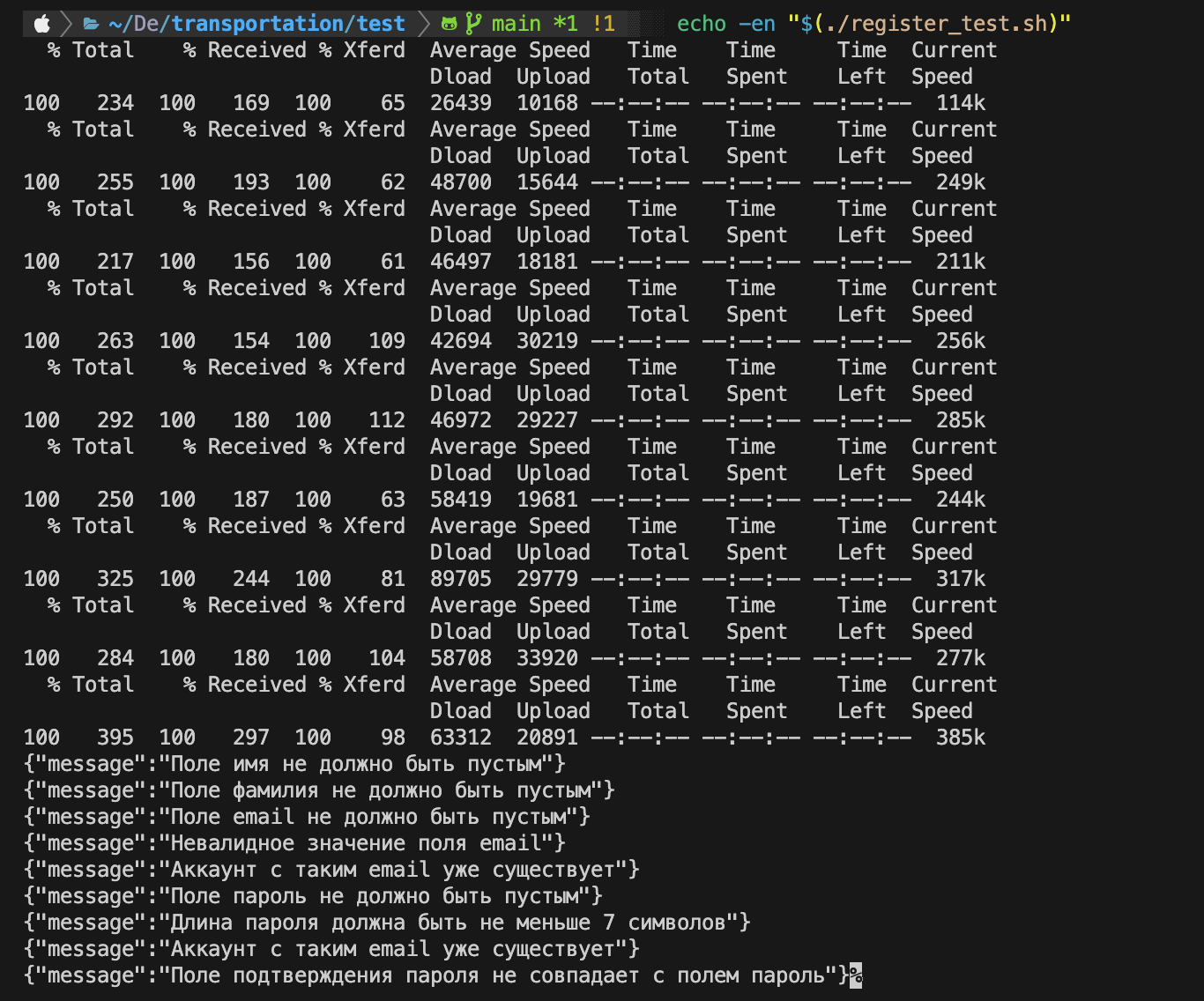


Рисунок 10 — Тестирование вкладки регистрации

Реализация тестирования:

#!/bin/bash

# run like this: echo -en "$(./register\_test.sh)"

# -X, --request defines method

# 1. Ошибка: Отсутствует firstName

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded' \

-d 'lastName=Иванов&email=ivan@example.com&password=password123'

**echo** ""

# 2. Ошибка: Отсутствует lastName

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded' \

-d 'firstName=Иван&email=ivan@example.com&password=password123'

**echo** ""

# 3. Ошибка: Отсутствует email

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded' \

-d 'firstName=Иван&lastName=Иванов&password=password123'

**echo** ""

# 4. Ошибка: Невалидный email

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded' \

-d 'firstName=Иван&lastName=Иванов&email=invalid\_email&password=password123&confirmPassword=password123'

**echo** ""

# 5. Ошибка: email уже существует

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded' \

-d 'firstName=Петр&lastName=Петров&email=ivan@example.com&password=password123&confirmPassword=password123'

**echo** ""

# 6. Поле пароль не должно быть пустым

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded' \

-d 'firstName=Иван&lastName=Иванов&email=ivan@example.com'

**echo** ""

# 7. Длина пароля должна быть не меньше 7 символов

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded' \

-d 'firstName=Петр&lastName=Петров&email=ivan@example.csfdom&password=passw'

**echo** ""

# 8. Аккаунт с таким email уже существует

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' \

-d 'firstName=Петр&lastName=Петров&email=ivan@examp.com&password=12121212&confirmPassword=12121212'

**echo** ""

# 9. Поле подтверждения пароля не совпадает с полем пароль

curl -X POST http://localhost:8848/register \

-H 'Content-Type: application/x-www-form-urlencoded; charset=UTF-8' \

-d 'firstName=Петр&lastName=Петров&email=ivan@examp.com&password=12121212&confirmPassword=12'

**echo** ""

# ВЫВОДЫ ПО КУРСОВОЙ РАБОТЕ

В результате выполнения курсовой работы было создано веб-приложение, которое позволяет пользователям выполнять такие возможности как:

1. Регистрация
2. Вход в аккаунт
3. Расчет стоимости перевозки
4. Сохранение заказа в базе данных

Программа предназначена для использования фирмами, осуществляющими грузоперевозку. Она способна разгрузить работу менеджеров, которым не нужно будет лично отвечать на вопросы пользователей по стоимости доставки груза, а также позволит структурировать работу с заказами.

Разработанный проект демонстрирует успешный опыт применения современных технологий и инструментов программирования для решения актуальных задач в сфере управления и контроля. Он обладает высоким потенциалом для развития и совершенствования, а также может служить практическим пособием для студентов и специалистов, осваивающих навыки разработки программного обеспечения.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Документация по фреймворку droron [Электронный ресурс]. – URL:

<https://drogon.org/> (дата обращения 24.05.2024).