**Национальный исследовательский технологический университет «МИСиС»**

**Институт информационных технологий и компьютерных наук (ИТКН)**

**Курсовая работа**

**«Разработка Full-Stack веб-приложения»**

Выполнила: студентка группы БИВТ-21-7

Трофимова Алина

.

г. Москва

2023 год

Оглавление

[Предметная область 3](#_Toc135049738)

[Постановка задачи 4](#_Toc135049739)

[Описание архитектуры 5](#_Toc135049740)

[Описание структуры БД 7](#_Toc135049741)

[Описание серверной части 9](#_Toc135049742)

[Описание клиентской части 13](#_Toc135049743)

[Заключение 17](#_Toc135049744)

[Список литературы 18](#_Toc135049745)

# **Предметная область**

В данной курсовой работе будет рассмотрена разработка веб-приложения доставки на основе трех технологий - React, NestJS и TypeORM.

Выбранная тема касается разработки Full-Stack приложения, предназначенного для доставки товаров по всему миру. Приложение будет предлагать три тарифных плана, чтобы клиенты могли выбрать наиболее подходящий вариант в зависимости от своих потребностей. Пользователь выбирает свой вариант подписки. Подписки различаются стоимость на месяц и разными возможностями. Сайт также имеет функционал регистрации и авторизации пользователей с использованием технологии JWT.

Проблемы в данной предметной области могут включать следующие аспекты:

**Управление пользователями и авторизация:** Для обеспечения безопасности и контроля доступа к приложению необходима система управления пользователями и механизм аутентификации. В этом случае будет использоваться JWT (JSON Web Token) для обеспечения безопасного обмена информацией между клиентской и серверной частями приложения.

**Разработка пользовательского интерфейса**: Одной из ключевых проблем является создание удобного и интуитивно понятного пользовательского интерфейса, который позволит пользователям легко взаимодействовать с приложением и выбирать подходящий тарифный план для своих отправлений.

**Управление базой данных:** В данном проекте для управления данными будет использоваться TypeORM и механизм миграций, позволяющий легко обновлять структуру базы данных и поддерживать целостность данных.

**Множество способов доставки:** Пользователи могут предпочитать различные способы доставки в зависимости от своих потребностей и предпочтений. Приложение должно предоставлять возможность выбора из нескольких способов доставки, учитывая их особенности и стоимость.

**Управление ошибками и исключительными ситуациями:** При разработке приложения могут возникать различные ошибки и исключительные ситуации, такие как повторная регистрация пользователя. Важно обеспечить надежный механизм управления ошибками и обработки исключительных ситуаций для минимизации негативного влияния на пользователей.

**Постановка задачи**

Вариант 21

Требуется разработать клиент-серверное Full-Stack приложение для доставки товаров по всему миру. Приложение должно предоставлять три тарифных плана для выбора клиентами в зависимости от их потребностей.

**Цель:**

Разработать полноценный веб-сайт на стеке технологий React,NestJS и TypeOrm в качестве основных инструментов разработки, а также JWT для обеспечения безопасности при авторизации и Postman для удобной документации API.

Для достижения этой цели необходимо решить следующие задачи.

**Задачи:**

* **База данных PostgreSQL:**
* Создание таблиц в базе данных для хранения информации о клиентах, посылках и курьеров, пользователей с использованием TypeORM и миграций.
* Определение связей между таблицами для обеспечения целостности данных и эффективного доступа к ним.
* **Backend разработка с использованием NestJS:**
* Разработка API с использованием NestJS
* Реализация механизма аутентификации и авторизации с использованием JWT для обеспечения безопасного доступа к функциональным возможностям приложения.
* Взаимодействие с базой данных PostgreSQL для чтения и записи данных о клиентах, курьерах и посылках, пользователей
* **Frontend разработка с использованием React**
* Разработка пользовательского интерфейса
* Взаимодействие с API на фронтенде для отправки запросов на вход и регистрацию.

**Функциональные возможности разрабатываемого клиент-серверного приложения включают:**

1. Регистрация новых клиентов и авторизация пользователей
2. Обеспечение безопасности данных и авторизации с использованием JWT.
3. Отображение каталога тарифов, их описания и стоимости, скорости доставки.
4. Подключения к базе данных и получения запросов
5. Создание API и получение ответа от сервера
6. Возможность поиска, обновления, создания, удаления информации о клиентах, посылках и курьеров.

# **Описание архитектуры**

Описание архитектуры является важной частью разработки программного приложения. Оно предоставляет обзор структуры, позволяя разработчикам, понять, как компоненты системы взаимодействуют друг с другом, какие функции выполняются и какие сервисы предоставляются.

Архитектура разработанного клиент-серверного приложения будет следующей:

**Клиентская часть:**

* 1. Разработана с использованием React, обеспечивает интерфейс пользователя и взаимодействие с серверной частью приложения.
  2. Отображает пользовательский интерфейс.
  3. Отправляет HTTP-запросы на сервер для получения данных о клиентах, курьерах и посылок и обновления информации.

**Серверная часть:**

1. Разработана с использованием NestJS, обеспечивает обработку запросов от клиентской части и взаимодействие с базой данных.
2. Содержит контроллеры, которые определяют обработку HTTP-запросов, связанных с клиентами и посылками.
3. Использует сервисы для выполнения бизнес-логики, таких как создание, поиска, удаления, обновления новых посылок, курьеров, клиентов и т.д.
4. Взаимодействует с базой данных PostgreSQL с использованием TypeORM и миграций для чтения и записи данных о клиентах, посылках и курьерах.

**База данных PostgreSQL:**

1. Используется для хранения данных о клиентах, посылках и курьерах.
2. Включает таблицы, соответствующие сущностям, определенным с использованием TypeORM.
3. Обеспечивает сохранение, чтение и обновление данных с помощью SQL-запросов, которые генерируются и выполняются TypeORM.

Основные сущности, которые используются в нашем веб-приложении:

**Клиенты:**

* Хранят информацию о клиентах, такую как имя, фамилия, адрес, контактная информация и т.д.

**Посылки:**

* Хранят информацию, такую как, тариф, адрес отправления и адрес получения.

**Курьеры:**

* Хранят информацию о клиентах, такую как имя, фамилия, адрес, контактная информация и т.д.

**Пользователи:**

* Хранят информацию, такую как, логин и пароль пользователя, нужно для авторизации и аунтефикации.

Клиенты связаны с посылками и курьерами связью многие-ко-многим через идентификаторы. База данных PostgreSQL обеспечивает сохранение и связь между этими сущностями. Контроллеры и сервисы NestJS обрабатывают запросы от клиентской части, взаимодействуют с базой данных и выполняют бизнес-логику для управления клиентами и посылками.

Посылки связаны с курьерами связью один-ко-многим через идентификаторы. База данных PostgreSQL обеспечивает сохранение и связь между этими сущностями. Контроллеры и сервисы NestJS обрабатывают запросы от клиентской части, взаимодействуют с базой данных и выполняют бизнес-логику для управления клиентами и посылками.

# **Описание структуры БД**

Описание структуры базы данных является важным компонентом разработки приложений, особенно при использовании реляционных баз данных. Это описание определяет структуру данных, таблицы, связи между ними и ограничения, которые определяют, как данные будут организованы и храниться в БД.

Описание структуры БД для разрабатываемого клиент-серверного приложения для доставки посылок по всему миру с использованием PostgreSQL.

Предметная область включает следующие основные сущности и их взаимосвязи:

**Клиент (Client):**

* Идентификатор клиента (id)
* Имя (first name)
* Фамилия (last name)
* Адрес (address)
* Массив заказов

**Курьер (Courier):**

* Идентификатор курьера (id)
* Имя (first name)
* Фамилия (last name)
* Массив заказов
* Массив клиентов

**Заказ (Order):**

* Идентификатор заказа (id)
* Тариф (plan)
* Адрес отправления (address\_to)
* Адрес получения (address\_from)
* Массив заказов
* Массив клиентов

**Взаимосвязи между сущностями:**

Клиент может быть отправителем или получателем нескольких посылок, следовательно несколько курьеров.

Каждая посылка связана с одним курьером, а курьер может доставлять много посылок.

Графическое представление БД представляет собой визуальное изображение, мы можем увидеть на рис. 1, структуры БД, включая таблицы, связи между ними. Это помогает разработчикам лучше понять структуру и организацию данных.

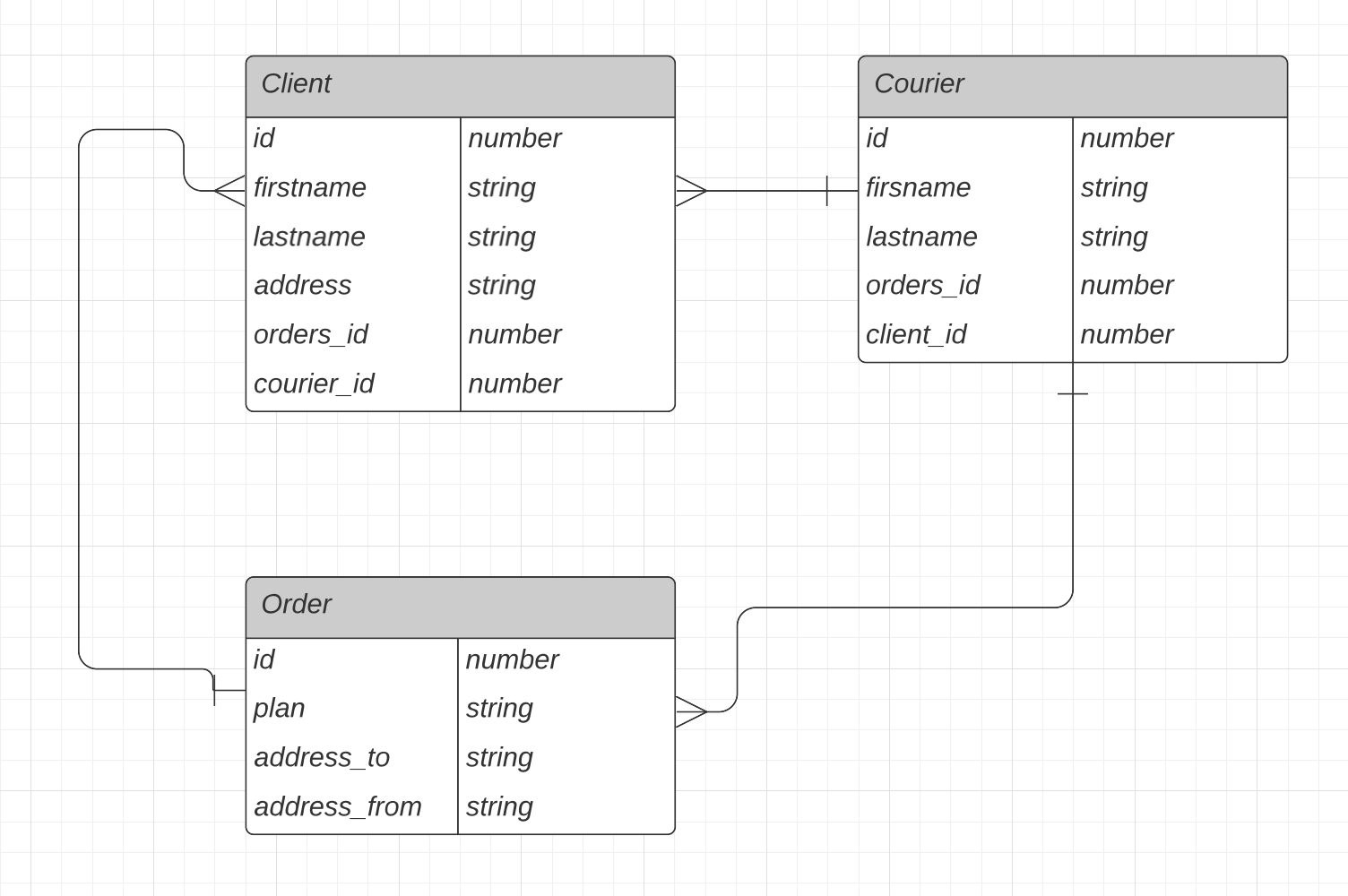
****

Рис 1 Графическое представление БД

В данной схеме БД отображены основные сущности и их атрибуты. Связи между сущностями представлены через идентификаторы, которые указывают на связанные записи в других таблицах. Клиенты связаны с посылками и курьерами, а посылки связаны с курьерами.

# **Описание серверной части**

Серверная часть приложения отвечает за обработку запросов от клиента и предоставление соответствующих ответов. Она включает в себя классы, модули и функции, которые обрабатывают бизнес-логику, взаимодействуют с базой данных и управляют состоянием приложения. Описание серверной части помогает разработчикам понять структуру и логику приложения, а также облегчает сопровождение и расширение функциональности.

Существует архитектурный шаблон MVC (Model-View-Controller), который разделяет приложение на три компонента: Модель, Представление и Контроллер.

Схему мы можем увидеть на рис.2.

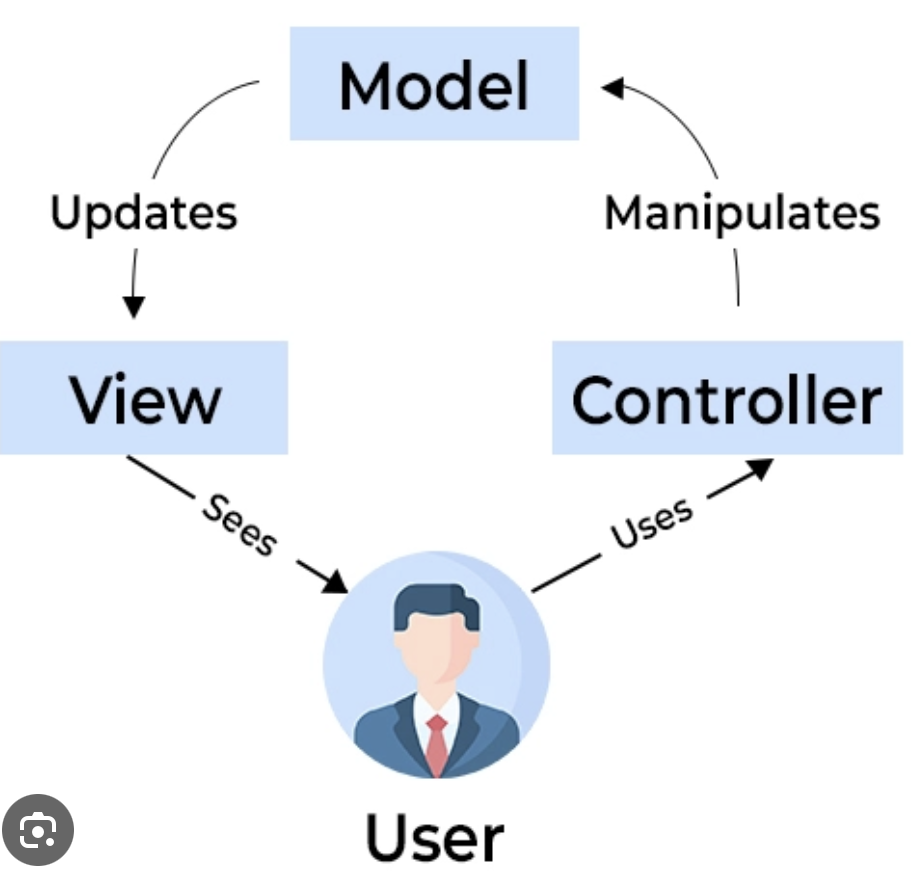


Рис. 2 MVC схема

Модель отвечает за данные и бизнес-логику, представление отображает данные пользователю, а контроллер обрабатывает пользовательский ввод и управляет взаимодействием между моделью и представлением.

Разберем серверную часть нашего приложения.

* 1. **Контроллеры (Controllers):**

Контроллеры отвечают за обработку пользовательских запросов и взаимодействие с соответствующими моделями и представлениями. Они принимают входные данные от клиента, вызывают соответствующие методы моделей для обработки данных и формируют ответы в виде данных, которые передаются представлениям для отображения пользователю.

**ClientsController:** Отвечает за обработку запросов в url строке, связанных с клиентами. Включает методы для создания, поиска, обновления и удаления клиентов.

@Get()  
*findAll*() {  
 *return this*.clientsService.*findAll*();  
}

@Put(':id')  
*update*(@Param('id') id: *string*, @Body() updateClient: Client) {  
 *return this*.clientsService.*update*(+id, updateClient);  
}

@Post()  
*create*(@Body() createClient: CreateClientDto) {  
 *this*.clientsService.*createClient*(createClient);  
}

@Delete(':id')  
*remove*(@Param('id') id: *number*) {  
 *return this*.clientsService.*remove*(id);  
}

Разберем данный кусок кода, код представляет набор обработчиков запросов в контроллере:

**FindAll()** - обработчик GET-запроса без указания конкретного идентификатора клиента.

**Update()** - обработчик PUT-запроса для обновления информации о клиенте.

**Create()** - обработчик POST-запроса для создания нового клиента.

**Remove()** - обработчик DELETE-запроса для удаления клиента по указанному идентификатору.

**CouriersController:** Отвечает за обработку запросов в url строке, связанных с курьерами. Включает методы для создания, поиска, обновления и удаления клиентов.

**OrdersController:** Обрабатывает запросы в url строке, связанные с посылками. Содержит методы для создания, поиска, обновления и удаления посылок.

* 1. **Модели (Models):**

Модели представляют бизнес-логику и данные приложения, обеспечивая доступ к данным, их изменение и валидацию, а также взаимодействие с базой данных или другими внешними источниками данных.

**ClientsModel:** Представляет модель клиента, определяет атрибуты и методы для работы с данными клиента.(дает возможность обработки других файлов)

Пример кода для **ClientsModel:**

@Module({  
 imports: [  
 // DatasourceModule,  
 CouriersModule,  
  
 OrdersModule,  
 ClientsModule,  
 TypeOrmModule.*forFeature*([Client, Order, Courier]), // !!! В модуле автор мы используем все три сущности, поэтому все три сущности необходимо импортирвоать!],  
 ],  
 controllers: [ClientsController],  
 providers: [ClientsService],  
})  
*export class* ClientsModule {}

Данный код представляет модуль ClientsModule, который используется для организации компонентов, связанных с клиентами в приложении.

Он импортирует несколько других модулей, таких как CouriersModule, OrdersModule и ClientsModule. Кроме того, он использует TypeOrmModule.forFeature() для импорта и регистрации сущностей Client, Order и Courier из базы данных.

Модуль также определяет контроллеры и провайдеры, которые будут использоваться в контексте этого модуля. Контроллеры (ClientsController) определяют API-маршруты и обрабатывают запросы, связанные с клиентами. Провайдеры (ClientsService) предоставляют бизнес-логику и методы для работы с данными клиентов.

Таким образом, ClientsModule объединяет компоненты, связанные с клиентами, и обеспечивает их доступность и взаимодействие в рамках приложения.

**CouriersModel:** Представляет модель курьера, определяет методы для работы с данными курьера.(дает возможность обработки других файлов)

**OrdersModel:** Представляет модель заказов, определяет атрибуты и методы для работы с данными заказа.(дает возможность обработки других файлов)

* 1. **Представление (View):**

Представления отвечают за отображение данных пользователю в удобной и понятной форме, например, в виде HTML-страницы или JSON-ответа. В нашем случае(рис.3), мы получаем информацию о клиенте от сервера в формате JSON.

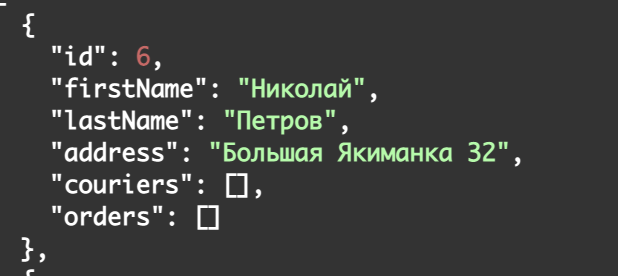


Рис.3 Пример работы: получение ответа от сервера в формате JSON

Клиентское приложение, в нашем случае, React, использует полученные данные для отображения пользовательского интерфейса и взаимодействия с пользователем.

* 1. **Сервисы (Service):**

Сервисы в приложении отвечают за выполнение бизнес-логики и обработку данных. Они предоставляют методы и функциональность, необходимые для работы с определенной сущностью или компонентом.

**ClientsService:** Выполняют бизнес-логику, обрабатывает операции, связанные с определенной сущностью или функциональностью.

**CouriersService:** Выполняют бизнес-логику, обрабатывает операции, связанные с определенной сущностью или функциональностью.

**OrdersService:** Выполняют бизнес-логику, обрабатывает операции, связанные с определенной сущностью или функциональностью.

*findOne*(id: *number*): Promise<Order> {  
 // Promise<Order> - указывает, что функция возвращает объект Order в виде Promise (c асинхронного потока)  
 *return this*.orderRepository.*findOne*({  
 where: { id }, //указываем условие поиска по id  
 relations: { clients: *true* }, //получаем связанные объекты

Пример работы одной из функции Service.

Данный код представляет метод **findOne,** который используется для поиска объекта Order по указанному id с возможностью получения связанных объектов.

# **Описание клиентской части**

Клиентская часть - это то, с чем пользователь взаимодействует в приложении. Она включает в себя интерфейс, с помощью которого пользователь может вводить данные, выполнять операции и получать информацию. Клиентская часть также обеспечивает взаимодействие с сервером для получения и отправки данных.

Рассмотрим клиентскую часть нашего веб-приложения:

* 1. **Регистрация и вход в систему:**

*Форма регистрации*: Пользователь может заполнить форму для создания нового аккаунта в системе. Форма может включать поля для ввода имени, логина и пароля.(рис.4)

*Форма входа*: Зарегистрированный пользователь может войти в систему, указав свой логин и пароль.(рис.5)

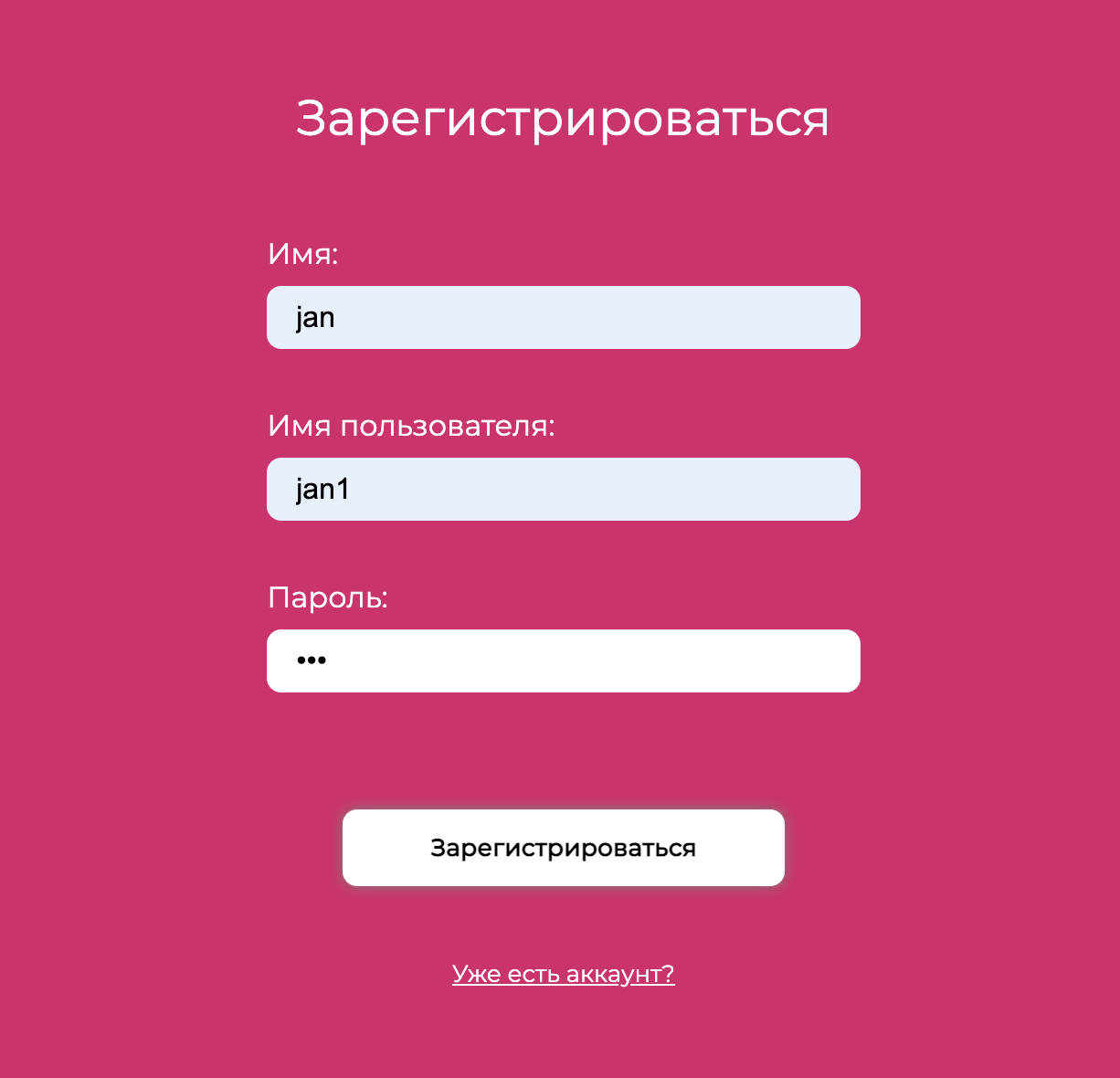


Рис. 4 Форма регистрации пользователей

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Рис. 5 Форма авторизованных пользователей

На рис.6 мы можем увидеть, что пользователь вошел в свой аккаунт успешно.



Рис.6 Успешное выполнение входа

* 1. **Тарифные планы**

*Информация о доступных тарифных планах*: Пользователь может просмотреть информацию о различных тарифных планах, включая их особенности, цены и условия.

Также клиентское приложение может взаимодействовать с API следующим образом:

1. Клиентское приложение отправляет запросы на регистрацию и вход в систему, предоставляя необходимые данные, такие как имя, адрес логин и пароль.
2. После успешной аутентификации, клиентское приложение получает токен доступа (например, в формате JWT), который используется для аутентификации при дальнейших запросах. На рис7 мы получаем access\_token, после успешной аунтификации.

[

**Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание**]



Рис. 7 Получение access\_token

**Взаимодействие с посылками:**

Клиентское приложение отправляет запросы для создания новых посылок, указывая необходимую информацию, такую как адрес отправителя, адрес получателя и тариф посылки.

Клиентское приложение может отправлять запросы для получения списка посылок, а также может обновлять информацию о заказе и даже искать заказ по id.

**Взаимодействие с клиентами:**

Клиентское приложение отправляет запросы для создания новых клиентов, указывая необходимую информацию, такую как имя, фамилию клиента, адрес.

Клиентское приложение может отправлять запросы для получения списка клиентов, а также может обновлять информацию о клиенте и даже искать клиента по id.(рис8)

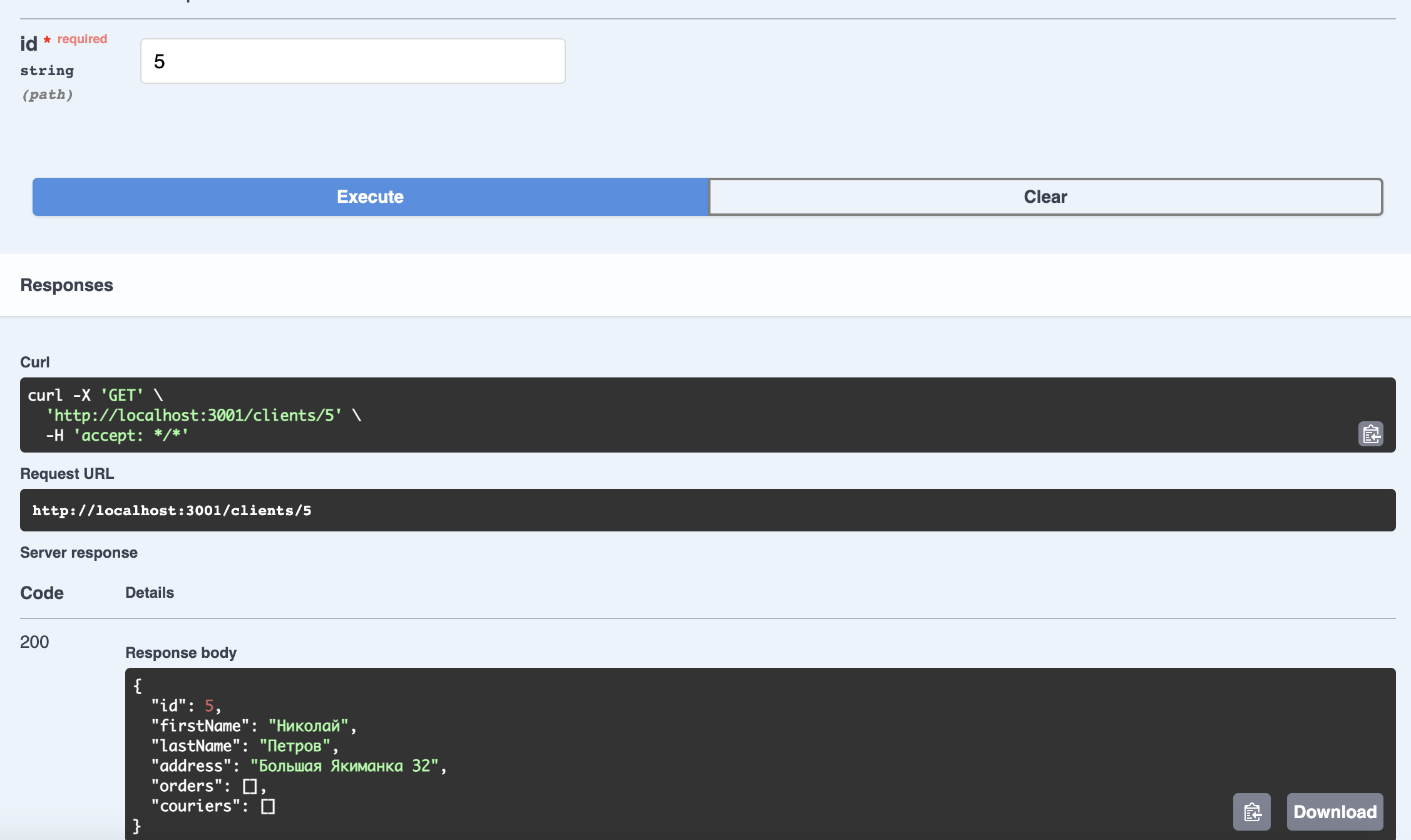
****

Рис. 8 Вывод клиента по id

**Взаимодействие с курьерами:**

Клиентское приложение отправляет запросы для создания новых курьеров, указывая необходимую информацию, такую как имя, фамилию курьера.

Клиентское приложение может отправлять запросы для получения списка курьеров, а также может обновлять информацию о курьере и даже удалять курьера, мы можем увидеть на рис9.



Рис. 9 Удаление курьера

# **Заключение**

В данной курсовой работе была рассмотрена предметная область разработки Full-Stack приложения для доставки посылок по всему миру. Главной проблемой, которая была рассмотрена, является эффективная организация процесса доставки, учета клиентов и выбора тарифных планов.

Для решения этой проблемы было разработано клиент-серверное приложение с использованием следующих инструментов:

1. TypeORM: Библиотека для работы с базой данных, которая облегчает создание моделей, миграций и запросов к базе данных.
2. JWT (JSON Web Token): Использование токенов авторизации для безопасной передачи информации между клиентом и сервером.
3. React: Библиотека для разработки пользовательского интерфейса на стороне клиента.
4. NestJS: Фреймворк для разработки серверной части приложения на основе Node.js, который предоставляет мощные инструменты для построения масштабируемых и модульных приложений.

В результате разработки были достигнуты следующие результаты:

1. Разработано Full-Stack приложение для доставки посылок, которое включает функциональности учета клиентов, выбора тарифных планов и организации процесса доставки.
2. Была создана структура базы данных PostgreSQL, которая включает сущности "Клиент", "Посылка", "Курьер" и их взаимосвязи.
3. Реализованы API с использованием NestJS, которые обеспечивают взаимодействие клиентской и серверной частей приложения.
4. Разработан пользовательский интерфейс на React, который предоставляет удобный интерфейс.

В целом, реализованное веб-приложение удовлетворяет всем запросам и требованиям, которые я поставила при постановке целей.

# **Список литературы**

* 1. <https://blog.logrocket.com/jwt-authentication-best-practices/#brief-introduction-to-jwt>
  2. <https://docs.nestjs.com/controllers>
  3. <https://legacy.reactjs.org/docs/getting-started.html>
  4. <https://dev.to/jaredm/how-to-bootstrap-your-project-using-nestjs-and-react-1a8e>
  5. <https://github.com/trofimovaaa/ITKN_BIVT-21-7_TrofimovaAM_21_Delivery.git>