Содержание

[Введение 4](#_Toc186805666)

[1 Постановка задачи и обзор аналогичных решений 5](#_Toc186805667)

[1.1 Постановка задачи 5](#_Toc186805668)

[1.2 Обзор аналогов 5](#_Toc186805669)

[1.2.1 Интернет-ресурс «Zona531» 5](#_Toc186805670)

[1.2.2 Интернет-ресурс «Fitness-club.by» 6](#_Toc186805671)

[1.2.3 Интернет-ресурс «Oktopus Fitness Club» 6](#_Toc186805672)

[1.3 Выводы по разделу 7](#_Toc186805673)

[2 Проектирование web-приложения 8](#_Toc186805674)

[2.1 Функциональность web-приложения 8](#_Toc186805675)

[2.2 Проектирование базы данных 10](#_Toc186805676)

[2.3 Архитектура web-приложения 14](#_Toc186805677)

[2.4 Выводы по разделу 15](#_Toc186805678)

[3 Реализация web-приложения 17](#_Toc186805679)

[3.1 Программная платформа 17](#_Toc186805680)

[3.2 Система управления базами данных PostgreSQL 17](#_Toc186805681)

[3.3 Работа с базой данных с помощью SQLX 17](#_Toc186805682)

[3.4 Программные библиотеки 20](#_Toc186805683)

[3.5 Сторонние сервисы Stripe, LocalStack 22](#_Toc186805684)

[3.6 Структура серверной части 22](#_Toc186805685)

[3.7 Реализация функционала для пользователя с ролью «Гостя» 29](#_Toc186805686)

[3.7.1 Регистрация 29](#_Toc186805687)

[3.7.2 Аутентификация 31](#_Toc186805688)

[3.8 Реализация функционала, доступного клиентам и администраторам 34](#_Toc186805689)

[3.8.1 Просмотр информации о абонементах 34](#_Toc186805690)

[3.8.2 Поиска, сортировка, фильтрация абонементов 35](#_Toc186805691)

[3.8.3 Просмотр информации о тренерах и отзывов о тренерах 36](#_Toc186805692)

[3.8.4 Выход из системы 36](#_Toc186805693)

[3.9 Реализация функционала для пользователя с ролью «Клиент» 37](#_Toc186805694)

[3.9.1 Оставить отзыв о тренере 37](#_Toc186805695)

[3.9.2 Покупка абонементов 39](#_Toc186805696)

[3.9.3 Редактирование профиля 40](#_Toc186805697)

[3.9.4 Просмотр списка своих заказов 43](#_Toc186805698)

[3.10 Реализация функционала для пользователя с ролью «Администратор» 44](#_Toc186805699)

[3.10.1 Редактирования клиентов 44](#_Toc186805700)

[3.10.2 Редактирования абонементов 45](#_Toc186805701)

[3.10.3 Редактирования тренеров 50](#_Toc186805702)

[3.11 Структура клиентской части 55](#_Toc186805703)

[3.12 Выводы по разделу 58](#_Toc186805704)

[4 Тестирование web-приложения 59](#_Toc186805705)

[4.1 Функциональное тестирование 59](#_Toc186805706)

[4.2 Выводы по разделу 67](#_Toc186805707)

[5 Руководство пользователя 68](#_Toc186805708)

[5.1 Руководство пользователя для роли «Гость» 68](#_Toc186805709)

[5.1.1 Аутентификация 68](#_Toc186805710)

[5.1.2 Регистрация 69](#_Toc186805711)

[5.2 Руководство пользователей с ролями «Клиент», «Администратор» 71](#_Toc186805712)

[5.2.1 Просмотра информации о абонементах 71](#_Toc186805713)

[5.2.2 Поиска, сортировка, фильтрация абонементов 71](#_Toc186805714)

[5.2.3 Просмотр информации о тренерах и отзывов о тренерах 72](#_Toc186805715)

[5.2.4 Выход из системы 73](#_Toc186805716)

[5.3 Руководство пользователя с ролью «Клиент» 73](#_Toc186805717)

[5.3.1 Оставить отзыв о тренере 73](#_Toc186805718)

[5.3.2 Покупка абонементов и просмотр списка своих заказов 74](#_Toc186805719)

[5.3.3 Редактирование профиля 75](#_Toc186805720)

[5.4 Руководство пользователя с ролью «Администратор» 76](#_Toc186805721)

[5.4.1 Редактирование клиентов 76](#_Toc186805722)

[5.4.2 Редактирование абонементов 77](#_Toc186805723)

[5.4.3 Редактирование тренеров 78](#_Toc186805724)

[5.5 Выводы по разделу 81](#_Toc186805725)

[Заключение 82](#_Toc186805726)

[Список используемых источников 83](#_Toc186805727)

[Приложение А 84](#_Toc186805728)

[Приложение B 85](#_Toc186805729)

[Приложение Б 90](#_Toc186805730)

Введение

Web-приложение – это клиент-серверное приложение, в котором клиент взаимодействует с сервером по протоколу HTTP [1].

Цель курсового проекта – автоматизировать процесс покупки абонементов, управления сущностями фитнес-центра, ознакомления с тренерами с помощью web-приложения «FitLab».

Для достижения цели были поставлены следующие задачи:

1. Провести анализ существующих интернет-ресурсов для фитнес-центров и определить ключевые требования к разрабатываемому web-приложению (рассмотрено в разделе 1);
2. Разработать архитектуру web-приложения (рассмотрено в разделе 2);
3. Реализовать функционал web-приложения с учетом поставленных требований (рассмотрено в разделе 3);
4. Провести тестирование для выявления и устранения ошибок, а также для проверки соответствия требованиям (рассмотрено в разделе 4);
5. Разработать руководство пользователя web-приложения (рассмотрено в разделе 5).

Целевая аудитория web-приложения «FabLab» включает людей, желающих, оздоровиться, либо тренироваться профессионально под руководством опытных тренеров.

В качестве программной платформы было решено использовать следующие технологии: серверная часть реализована на Golang [2], клиентская часть на React.js [3], в качестве базы данных выбрана PostgreSQL [4].

# **Постановка задачи и обзор аналогичных решений**

## **Постановка задачи**

Задачи web-приложения включают реализацию поддержки ролей гостя, клиента и администратора.

Гость должен иметь возможность регистрироваться и аутентифицироваться.

Клиент и администратор должны иметь возможность просмотра информации о абонементах, поиска абонементов, сортировки абонементов, фильтрации абонементов, просмотра информации о тренерах, просмотра отзывов о тренерах и выхода из системы.

Отличительные возможности клиента: покупать абонементы, оставить отзыв о тренере, редактировать профиль, просматривать список своих заказов.

Отличительные возможности администратора: редактировать абонементы, тренаров, клиентов.

## **Обзор аналогов**

## **Интернет-ресурс «Zona531»**

Сайт фитнес-центра «Zona531» [5], представляет собой сайт-визитку для фитнес-центра. Главная страница представлена на рисунке 1.1.

A screenshot of a website

AI-generated content may be incorrect.

Рисунок 1.1 – Сайт «Zona531»

Достоинства:

* Просмотр программ тренировок;
* Удобное меню навигации по сайту;
* Удобная форма для связи с персоналом фитнес-центра.

Недостатки:

* Отсутствие личного кабинета;
* Отсутствие покупки абонементов;

## **Интернет-ресурс «Fitness-club.by»**

Сайт фитнес-центр «Fitness-club.by» [6] – это платформа для ознакомления и взаимодействию с фитнес-центрами сети Life Style.

На рисунке 1.2 представлен интерфейс сайта.

A person with a basketball

AI-generated content may be incorrect.

Рисунок 1.2 – Сайт «Fitness-club.by»

Достоинства:

* Возможность оформления подписки на абонемент;
* Удобное меню навигации по сайту;
* Чат для связи с персоналом фитнес-центра.

Недостатки:

* Отсутствие личного кабинета;
* Отсутствие поиску по абонементам

Невозможность оставить комментарий к тренеру.

## **Интернет-ресурс «Octopus Fitness Club»**

«Oktopus Fitness Club» [7]. – это платформа, предоставляющая услуги фитнес-центра и дающая возможность формить подписку на абонемент. Главная страница представлена на рисунке 1.3.

A screenshot of a website

AI-generated content may be incorrect.

Рисунок 1.3 – Сайт «Oktopus Fitness Club»

Достоинства:

* Оформление подписке по разным планам;
* Возможность оставлять отзывы тренерам;
* Присутствие личного кабинета;

Недостатки:

* Отсутствие уведомлений об ошибках при вводе некорректных данных при регистрации и аутентификации.
* Отсутствие поиска по абонементам

## **Выводы по разделу**

1. 1. Поставленные задачи требуют разработки web-приложения с поддержкой трех ролей: гостя, клиента и администратора, каждая из которых будет обладать своим перечнем функциональных возможностей, рассмотренных в пункте 1.1.

Анализ аналогичных решений показал, что сайты предлагают базовый функционал, включая отображение поставляемых услуг и оформление подписок. Из недостатков можно отметить отсутствие уведомлений об ошибках при регистрации, невозможность поиска по абонементам, невозможность оставить отзыв к тренеру и отсутствие возможности оплаты на сайте.

# **Проектирование web-приложения**

## **Функциональность web-приложения**

Функциональные возможности web-приложения представлены в диаграмме вариантов использования, представленной на рисунке 2.1.

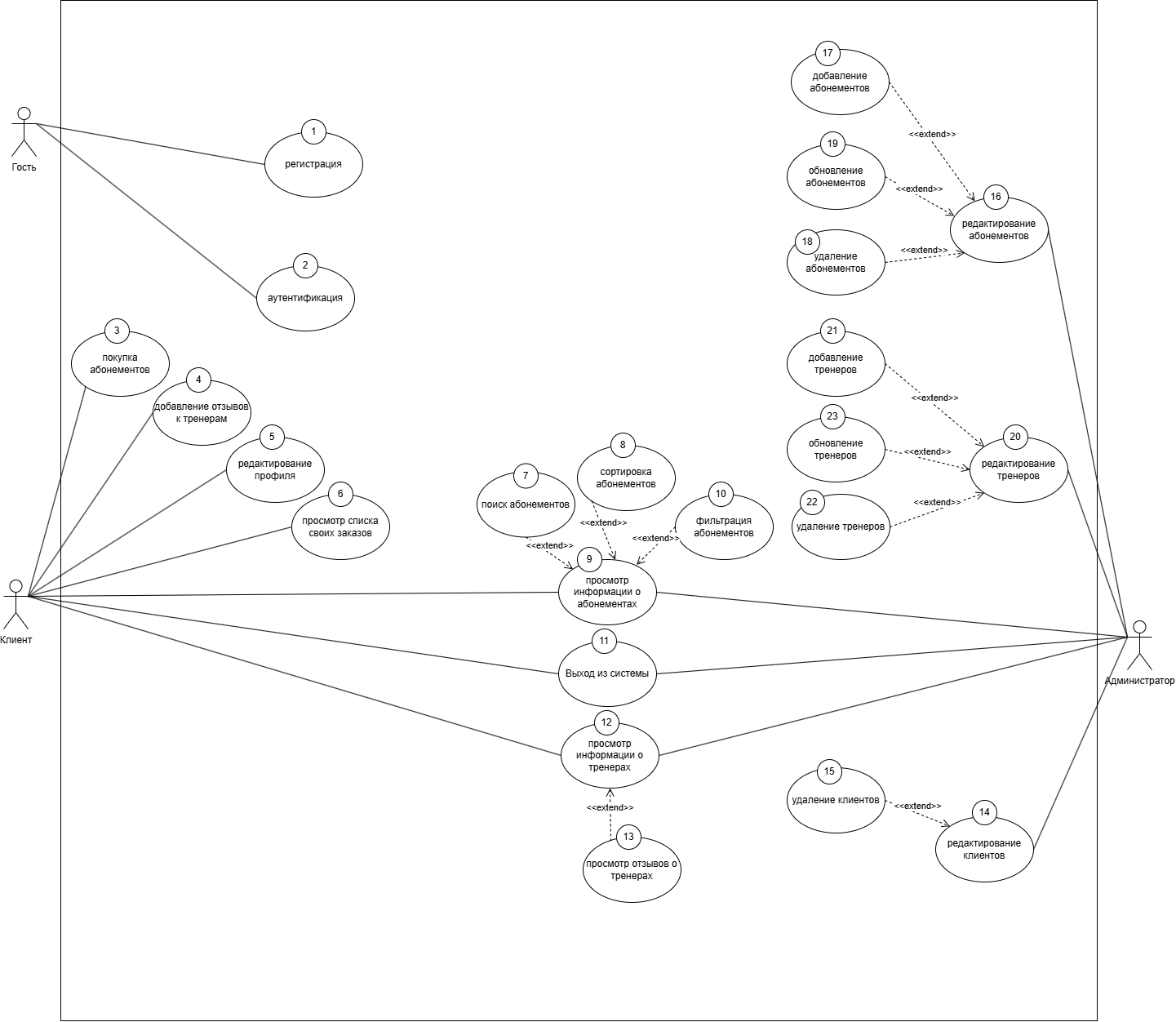


Рисунок 2.1 – Диаграмма вариантов использования web-приложения

Данная диаграмма вариантов использования наглядно демонстрирует функциональность web-приложения «FitLab» для различных ролей пользователей. Перечень ролей и их назначение приведены в таблице 2.1.

Таблица 2.1 – Назначение ролей пользователей в web-приложении

|  |  |
| --- | --- |
| Роль | Назначение |
| Гость | Пользователь, не прошедший регистрацию и аутентификацию. Имеет доступ к регистрации, аутентификации. |

Продолжение таблицы 2.1

|  |  |
| --- | --- |
| Клиент | Зарегистрированный и аутентифицированный пользователь, имеющий возможность покупать абонементы, добавлять отзывы к тренерам, редактировать профиля, просматривать списки своих заказов, просматривать информацию о абонементах, искать абонементы, сортировать абонементы, фильтровать абонементы, выходить из системы, просматривать информацию о тренерах, просматривать отзывы о тренерах. |
| Администратор | Уполномоченный пользователь, у которого есть права на редактирование клиентов, редактирование абонементов, редактирование тренеров, просмотр информации о абонементах, поиск абонементов, сортировки абонементов, фильтрации абонементов, выхода из системы, просмотра информации о тренерах, просмотра отзывов о тренерах. |

Роли в системе разделены таким образом, чтобы разграничить доступ к функционалу приложения и обеспечить безопасное использование платформы.

Функциональные возможности, доступные пользователю с ролью «Гость» приведены в таблице 2.2.

Таблица 2.2 – Функциональные возможности пользователя с ролью «Гость»

|  |  |  |
| --- | --- | --- |
| Номер | Вариант использования | Пояснение |
| 1 | Регистрация | Возможность создания новой учетной записи в системе |
| 2 | Аутентификация | Вход пользователя в систему с помощью данных учетной записи, таких как email и пароль. |

Функциональные возможности, доступные одновременно пользователям с ролями «Клиент» и «Администратор», приведены в таблице 2.3.

Таблица 2.3 – Функциональные возможности, доступные одновременно пользователям с ролями «Клиент» и «Администратор»

|  |  |  |
| --- | --- | --- |
| Номер | Вариант использования | Пояснение |
| 9 | Просмотр информации о абонементах | Просмотр списка абонементов с подробной информацией о них. |
| 7 | Поиск абонементов | Поиск абонементов по названию и периоду валидности. |
| 8 | Сортировка абонементов | Сортировка абонементов по цене и названию. |
| 10 | Фильтрация абонементов | Позволяет фильтровать .абонементы по диапазону цен. |
| 12 | Просмотр информации о тренерах | Просмотр списка тренеров с подробной информацией о них. |

Продолжение таблицы 2.3

|  |  |  |
| --- | --- | --- |
| Номер | Вариант использования | Пояснение |
| 13 | Просмотр отзывов о тренерах | Просмотр списка отзывов к тренеру с информацией о том, кто его оставил. |
| 11 | Выход из системы | Выход из системы. |

Функциональные возможности пользователя с ролью «Клиент» приведены в таблице 2.4

Таблица 2.4 – Функциональные возможности пользователя с ролью «Клиент»

|  |  |  |
| --- | --- | --- |
| Номер | Вариант использования | Пояснение |
| 3 | Покупка абонементов | Возможность совершить покупку абонементов. |
| 4 | Добавление отзывов к тренерам | Возможность добавить отзыв к конкретному тренеру. |
| 5 | Редактирование профиля | Возможность поменять имя и фото. |
| 6 | Просмотр списка своих заказов | Возможность просмотра списка своих заказов. |

Функциональные возможности пользователя с ролью «Администратор» приведены в таблице 2.5

Таблица 2.5 – Функциональные возможности пользователя с ролью «Администратор»

|  |  |  |
| --- | --- | --- |
| Номер | Вариант использования | Пояснение |
| 14 | Редактирование клиентов | Возможность удалять клиентов. |
| 15 | Удаление клиентов | Возможность удалять клиентов. |
| 16 | Редактирование абонементов | Возможность добавлять, обновлять, удалять клиентов. |
| 17 | Добавление абонементов | Возможность добавлять абонементы. |
| 18 | Удаление абонементов | Возможность удалять абонементы. |
| 19 | Обновление абонементов | Возможность обновлять абонементы. |
| 20 | Редактирование тренеров | Возможность добавлять, обновлять, удалять тренеров. |
| 21 | Добавление тренеров | Возможность добавлять тренеров. |
| 22 | Удаление тренеров | Возможность удалять тренеров. |
| 23 | Обновление тренеров | Возможность обновлять тренеров. |

Таким образом для каждой роли определен набор доступных действий и возможностей.

## **Проектирование базы данных**

Согласно диаграмме вариантов использования, была разработана база данных, с использованием реляционной СУБД PostgreSQL. Логическая схема базы данных представлена на рисунке 2.2.

A diagram of a computer code

AI-generated content may be incorrect.

Рисунок 2.2 – Логическая схема базы данных

База данных содержит десять таблиц, назначение таблиц базы данных представлено в таблице 2.6.

Таблица 2.6 – Назначение таблиц базы данных

|  |  |
| --- | --- |
| Таблица | Назначение |
| User | Хранит информацию о пользователях |
| Service | Хранит информацию об услугах. |
| Review | Хранит отзывы клиентов. |
| Abonement | Хранение информации о абонементах. |
| Coach | Хранение информации о тренерах. |
| Refresh\_Sessions | Хранит информацию для обновления токенов. |
| Abonement\_Service | Связывает абонементы с услугами. |
| Coach\_Review | Связывает тренеров с отзывами. |
| Order | Хранит информацию о заказах абонементов клиентами. |
| Coach\_Service | Связывает тренеров с услугами |

Таблица User содержит информацию о пользователях web-приложения «FitLab». Структура данной коллекции приведена в таблице 2.7.

Таблица 2.7 – Структура таблицы User

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Уникальный идентификатор пользователя |
| email | varchar(100) | Почта пользователя |
| role | varchar(255) | Роль пользователя |
| password\_hash | varchar(255) | Хеш пароля пользователя |
| photo | varchar(255) | URL [8] фотографии пользователя |
| name | varchar(100) | Имя пользователя |
| create\_time | timestamp with time zone | Дата и время создания пользователя |
| updated\_time | timestamp with time zone | Дата и время последнего обновления пользователя |

Таблица Service содержит информацию о услугах. Структура данной таблицы приведена в таблице 2.8.

Таблица 2.8 – Структура таблицы Service

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Уникальный идентификатор услуги |
| title | varchar(100) | Название услуги |
| photo | varchar(255) | URL фотографии услуги |
| create\_time | timestamp | Дата и время создания записи |
| updated\_time | timestamp | Дата и время последнего обновления записи |

Таблица Review содержит информацию о отзывах. Структура данной таблицы приведена в таблице 2.9.

Таблица 2.9 – Структура таблицы Review

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Уникальный идентификатор отзыва |
| body | text | Текст отзыва |
| created\_time | timestamp | Дата и время создания записи |
| updated\_time | timestamp | Дата и время последнего обновления записи |
| user\_id | uuid | Идентификатор пользователя, внешний ключ |

Таблица Abonement содержит информацию о абонементах. Структура данной таблицы приведена в таблице 2.10.

Таблица 2.10 – Структура таблицы Abonement

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Уникальный идентификатор абонемента |
| title | varchar(100) | Название абонемента |
| validity | int | Срок действия(количество дней). |
| visiting\_time | varchar(100) | Время посещений |
| photo | varchar(255) | URL фотографии абонемента |

Продолжение таблицы 2.11

|  |  |  |
| --- | --- | --- |
| price | decimal(10, 2) | Цена абонемента |
| created\_time | timestamp | Дата и время создания записи |
| updated\_time | timestamp | Дата и время последнего обновления записи |
| stripe\_price\_id | varchar(255) | Идентификатор цены в Stripe [9] |

Таблица Coach содержит информацию о тренерах. Структура данной таблицы приведена в таблице 2.11.

Таблица 2.11 – Структура таблицы Coach

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Уникальный идентификатор тренера |
| name | varchar(100) | Имя тренера |
| description | text | Описание тренера |
| photo | varchar(255) | URL фотографии тренера |
| created\_time | timestamp | Дата и время создания записи |
| updated\_time | timestamp | Дата и время последнего обновления записи |

Таблица Reftesh\_Session содержит информацию о refresh сессиях. Структура данной таблицы приведена в таблице 2.12.

Таблица 2.12 – Структура таблицы Refresh\_Session

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Уникальный идентификатор сессии обновления |
| user\_id | uuid | Идентификатор пользователя, внешний ключ |
| refresh\_token | varchar(255) | Токен обновления |
| finger\_print | varchar(255) | Отпечаток устройства |
| created\_time | timestamp | Дата и время создания записи |
| updated\_time | timestamp | Дата и время последнего обновления записи |

Таблица Abonement\_Service содержит информацию о связи абонементов с услугами. Структура данной таблицы приведена в таблице 2.13.

Таблица 2.13 – Структура таблицы Abonement\_Service

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| abonement\_id | uuid | Идентификатор абонемента, внешний ключ |
| service\_id | uuid | Идентификатор услуги, внешний ключ |

Таблица Coach\_Review содержит информацию о связи тренеров с отзывывами. Структура данной таблицы приведена в таблице 2.14.

Таблица 2.14 – Структура таблицы Coach\_Review

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |

Продолжение таблицы 2.1

|  |  |  |
| --- | --- | --- |
| coach\_id | uuid | Идентификатор тренера, внешний ключ |
| review\_id | uuid | Идентификатор отзыва, внешний ключ |

Таблица Order содержит информацию о заказах клиентов. Структура данной таблицы приведена в таблице 2.15.

Таблица 2.15 – Структура таблицы Order

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Уникальный идентификатор заказа |
| abonement\_id | uuid | Идентификатор абонемента, внешний ключ |
| user\_id | uuid | Идентификатор пользователя, внешний ключ |
| status | varchar(255) | Статус заказа |
| created\_time | timestamp | Дата и время создания записи |
| updated\_time | timestamp | Дата и время последнего обновления записи |
| expiration\_time | timestamp | Дата и время истечения срока действия |

Таблица Coach\_Service содержит информацию о связи тренеров с услугами. Структура данной таблицы приведена в таблице 2.16.

Таблица 2.16 – Структура таблицы Coach\_Service

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| coach\_id | uuid | Идентификатор тренера, внешний ключ |
| service\_id | uuid | Идентификатор услуги, внешний ключ |

Каждая таблица имеет четко определенные поля, отражающие определенные аспекты работы психологического центра.

**2.3 Архитектура web-приложения**

Архитектура web-приложения представлена в Приложении Б.

Пояснение назначения каждого элемента web-приложения представлено в таблице 2.17.

Таблица 2.19 – Назначение элементов архитектурной схемы web-приложения

|  |  |
| --- | --- |
| Элемент | Назначение |
| Web Server (nginx [10]) | Принимать запросы клиента, предоставлять статические файлы frontend-части web-приложения |
| Database Server (PostgreSQL) | Используется для хранения и предоставления доступа к данным, которые необходимы для работы web-приложения. |

Продолжение таблицы 2.17

|  |  |
| --- | --- |
| Gateway service | Принимать запросы клиента в формате HTTP, преобразовывать запросы в формат Protobuf [11] для передачи через gRPC [12] в другие микросервисы, преобразовывать ответы от микросервисов в формат HTTP для передачи клиенту. Обеспечивать работу с платежным шлюзом |
| SSO service | Выполнять логику единого входа |
| User service | Выполнять логику работы с сущностью пользователя |
| Service service | Выполнять логику работы с сущностью услиги |
| Abonement service | Выполнять логику работы с сущностью абонемент |
| Order service | Выполнять логику работы с сущностью заказа |
| Coach service | Выполнять логику работы с сущностью тренера |
| Review service | Выполнять логику работы с сущностью отзыва |
| Localstack [13] | Эмулировать локальную работу облачного сервиса s3 [14] от AWS(Amazon Web Services) [15] |
| Migrate | Выполнять миграции при первом запуске базы данных |
| Ngrok [16] | Предоставлять публичный URL для связи внешнего платежного шлюза с сервисом Gateway |
| StripeAPI | Предоставлять API для работы с оплатами |
| Client Browser | Отображать фронтэнд-часть web-приложения, отправлять запросы пользователя, отображать ответы сервера |

Описание протоколов, используемых при работе web-приложений, представлено в таблице 2.18.

Таблица 2.18 – Описание используемых протоколов

|  |  |
| --- | --- |
| Протокол | Назначение |
| HTTPS [17] | Обмен данными между StripeAPI и Gateway service, а также между StripeAPI и Ngrok. Обеспечить безопасную передачу данных путём использования криптографического протокола TLS [18]. Версия HTTP 1.1, TLS 1.2. |
| TCP [19] | Обмен данными между Database Server и Application Server, а также создать виртуальное соединение между процессами. |
| HTTP | Обмен данными между микросервисами Gateway service, SSO service, Abonement service, Review service, User service, Service service, Coach service, Order service. Обеспечивает быструю передачу данных при HTTP с версией 2 и Protobuf. Обмен данными между Client Browser и Web Server, Web Server и Gateway service, Gateway service и Ngrok. Обеспечивает передачу данных при помощи HTTP с версией 1.1. |

Таким образом были рассмотрены все ключевые элементы архитектуры web-приложения.

## **Выводы по разделу**

Таким образом, было спроектировано web-приложение, обладающее следующими особенностями:

1. Поддержка трех ролей с четко разграниченными правами доступа и функциональными возможностями: гость, клиент, администратор.

2. Спроектирована база данных web-приложения, которая состоит из десяти таблиц: abonement, abonement\_service, coach, coach\_review, coach\_service, order, refresh\_sessions, review, service, user. Эти таблицы охватывают все аспекты работы web-приложения «FitLab» для фитнес-центра.

3. Web-приложение имеет микро сервисную архитектуру с применением Nginx в качестве web-сервера, PostgreSQL для хранения данных, Docker [20] для запуска много контейнерных приложений.

# **Реализация web**-**приложения**

## **Программная платформа**

Для серверной части проекта был выбран язык Golang. Фреймворк GIN [21] – для написания REST API [22], фреймворк gRPC – для межпроцессорного взаимодействия сервисов.

## **Система управления базами данных PostgreSQL**

Для работы веб-приложения используется PostgreSQL — мощная, надежная и масштабируемая система управления базами данных. Она поддерживает сложные SQL [23]-запросы, транзакции с соответствием стандарту ACID [24], а также широкий спектр типов данных, включая JSON [25]. Скрипт создания базы данных представлен в Приложении А.

## **Работа с базой данных с помощью SQLX**

В проекте используется библиотека SQLX [26] для работы с базой данных в Go. SQLX расширяет стандартный пакет database/sql [28], добавляя удобные методы для выполнения SQL-запросов и маппинга результатов на структуры Go. Это упрощает работу с данными и минимизирует количество ошибок, возникающих при ручном маппинге.

Сопоставление моделей, используемых в SQLX, с их реальными структурами представлено в таблице 3.1.

Таблица 3.1 – Сопоставление моделей, используемых в SQLX

|  |  |
| --- | --- |
| Название модели | Название таблицы |
| Abonement | abonement |
| Coach | coach |
| Order | order |
| RefreshSessions | refresh\_sessions |
| Review | review |
| Service | service |
| User | user |

Код, описывающий модель Abonement, приведён в листинге 3.1.

type Abonement struct {  
 Id uuid.UUID `db:"id"`  
 Title string `db:"title"`  
 Validity string `db:"validity"`  
 VisitingTime string `db:"visiting\_time"`  
 Photo string `db:"photo"`

Продолжение листинга 3.1

Price int `db:"price"`  
 UpdatedTime time.Time `db:"updated\_time"`  
 CreatedTime time.Time `db:"created\_time"`  
 StripePriceId string `db:"stripe\_price\_id"`  
}

Листинг 3.1 – Модель Abonement

Модель Abonement в SQLX описывает абонемент с полями: Id – уникальный идентификатор, Title – название абонемента, Validity – количество месяцев валидности абонемента, VisitingTime – время посещения в течении дня, Photo – фото, Price – цена, UpdatedTime – время обновления абонемента, CreatedTime – время создания абонемента, StripePriceId – цена для продукта из Stripe.

Код, описывающий модель Coach, приведён в листинге 3.2.

type Coach struct {  
 Id uuid.UUID `db:"id"`  
 Name string `db:"name"`  
 Description string `db:"description"`  
 Photo string `db:"photo"`  
 UpdatedTime time.Time `db:"updated\_time"`  
 CreatedTime time.Time `db:"created\_time"`  
}

Листинг 3.2 – Модель Coach

Модель Coach в SQLX описывает тренера с полями: Id – уникальный идентификатор, Name – имя тренера, Description – описание тренера, VisitingTime – время посещения в течении дня, Photo – фото, Price – цена, UpdatedTime – время обновления тренера, CreatedTime – время создания тренера.

Код, описывающий модель Order, приведён в листинге 3.3.

type Order struct {  
 Id uuid.UUID `db:"id"`  
 AbonementId uuid.UUID `db:"abonement\_id"`  
 UserId uuid.UUID `db:"user\_id"`  
 Status string `db:"status"`  
 UpdatedTime time.Time `db:"updated\_time"`  
 CreatedTime time.Time `db:"created\_time"`  
 ExpiredTime time.Time `db:"expiration\_time"`  
}

Листинг 3.3 – Модель Order

Модель Order в SQLX описывает заказ с полями: Id – уникальный идентификатор, AbonementId – уникальный идентификатор абонемента, UserId –уникальный идентификатор юзера, Status – статус заказа, Photo – фото, Price – цена, UpdatedTime – время обновления заказа, CreatedTime – время создания заказа, ExpiredTime – время истечения валидности заказа.

Код, описывающий модель RefreshSessions, приведён в листинге 3.4.

type RefreshSessions struct {  
 Id uuid.UUID `db:"id"`  
 UserId uuid.UUID `db:"user\_id"`  
 RefreshToken string `db:"refresh\_token"`  
 FingerPrint string `db:"finger\_print"`  
 CreatedTime time.Time `db:"created\_time"`  
 UpdatedTime time.Time `db:"updated\_time"`  
}

Листинг 3.4 – Модель RefreshSessions

Модель RefreshSessions в SQLX описывает заказ с полями: Id – уникальный идентификатор, UserId – уникальный идентификатор юзера, RefreshToken – рефреш токен, FingerPrint – отпечаток браузера пользователя, Photo – фото, Price – цена, UpdatedTime – время обновления рефреш сессии, CreatedTime – время создания рефреш сессии.

Код, описывающий модель Review, приведён в листинге 3.5.

type Review struct {  
 Id uuid.UUID `db:"id"`  
 UserId uuid.UUID `db:"user\_id"`  
 Body string `db:"body"`  
 UpdatedTime time.Time `db:"updated\_time"`  
 CreatedTime time.Time `db:"created\_time"`  
}

Листинг 3.5 – Модель Review

Модель Review в SQLX описывает заказ с полями: Id – уникальный идентификатор, UserId – уникальный идентификатор пользователя, Body – тело отзыва, UpdatedTime – время обновления отзыва, CreatedTime – время создания отзыва.

Код, описывающий модель Service, приведён в листинге 3.6.

type Service struct {  
 Id uuid.UUID `db:"id"`  
 Title string `db:"title"`  
 Photo string `db:"photo"`  
 UpdatedTime time.Time `db:"updated\_time"`  
 CreatedTime time.Time `db:"created\_time"`  
}

Листинг 3.6 – Модель Service

Модель Service в SQLX описывает услугу с полями: Id – уникальный идентификатор, Title – название услуги, Photo – фото, UpdatedTime – время обновления услуги, CreatedTime – время создания отзыва.

Код, описывающий модель User, приведён в листинге 3.7.

type User struct {  
 ID uuid.UUID `db:"id"`  
 Name string `db:"name"`  
 PasswordHash string `db:"password\_hash"`  
 Email string `db:"email"`  
 Role string `db:"role"`  
 Photo string `db:"photo"`  
 UpdatedTime time.Time `db:"updated\_time"`  
 CreatedTime time.Time `db:"created\_time"`  
}

Листинг 3.7 – Модель User

Модель User в SQLX описывает пользователя с полями: ID – уникальный идентификатор, Name – имя, PasswordHash – хэшированный пароль, Email – электронная почта, Role – роль, Photo – фото, UpdatedTime – время обновления пользователя, CreatedTime – время создания пользователя.

## **Программные библиотеки**

В процессе разработки серверной части web-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.2.

Таблица 3.2 – Программные библиотеки серверной части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| Validator [29] | v1.10.0 | Библиотека для валидации значений структур и отдельных полей на основе тегов. |
| CORS gin's middleware [30] | v1.7.2 | Библиотека которая предоставляет middleware для GIN для включения поддержки CORS [31]. |
| **jwt-go [32]** | v5.2.1 | Библиотека для работы с JWT [33] (JSON Web Tokens). |
| **uuid [34]** | v1.6.0 | Библиотека для создания и проверяет UUID [35]. |
| **GoDotEnv [36]** | v1.5.1 | Библиотека для загрузки переменных env из файла .env |
| **pg** | v1.10.9 | Драйвер для работы с PostgreSQL в Golang. |
| **Go Stripe [38]** | v81.1.1 | Библиотека для интеграции с платежной системой Stripe, используется для обработки платежей. |
| **Swag [39]** | v1.0.1 | Библиотека для преобразования аннотации Go в документацию Swagger 2.0 [40] |

Продолжение таблицы 3.2

|  |  |  |
| --- | --- | --- |
| **gin-swagger [41]** | v1.6.0 | Библиотека которая предоставляет middleware для GIN для автоматической генерации документации RESTful API с Swagger 2.0. |
| **gRPC-Go [42]** | v1.68.0 | Библиотека которая реализует gRPC в Go. |
| **Protobuf [43]** | v1.35.1 | Библиотека для поддержки Protobuf в Go. |
| aws-sdk-go-v2 [44] | v1.32.6 | Библиотека для интеграции c AWS. |
| sqlx | v1.4.0 | Библиотека для добавления удобных методов для выполнения SQL-запросов и маппинга результатов на структуры Go |
| Migrate [45] | v4.18.1 | Библиотека для работы с миграциями для баз данных. |
| Crypto [46] | v0.27.0 | Библиотека которая предоставляет криптографические функции и алгоритмы. |

В процессе разработки клиентской части web-приложения были задействованы программные библиотеки, представленные в таблице 3.3.

Таблица 3.3 – Программные библиотеки клиентской части

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Назначение |
| mui/icons-material [47] | v5.15.14 | Библиотека иконок от Material-UI [48], предоставляет набор иконок, которые легко интегрировать в React-приложения. |
| mui/material [49] | v5.15.14 | **Библиотека предоставляющая готовые к**омпоненты , ориентированные на стилизацию и гибкость. |
| mui/joy [50] | v5.0.0 | **Библиотека предоставляющая готовые к**омпоненты , ориентированные на стилизацию и гибкость. |
| **reduxjs/toolkit [51]** | v2.2.2 | **Библиотека для управления состоянием в приложениях с использованием Redux [52].** |
| react-stripe-js [53] | v3.1.1 | **Библиотека для интеграции с Stripe для работы с платежами, предоставляющая клиентскую сторону для взаимодействия с API Stripe.** |

Продолжение таблицы 3.3

|  |  |  |
| --- | --- | --- |
| **Axios [54]** | v1.6.8 | **Библиотека для выполнения HTTP-запросов**. |
| **React [55]** | v18.2.0 | Библиотека для создания пользовательских интерфейсов, используется для построения компонентов и управления состоянием UI |
| react-router-dom [56] | v6.22.3 | Библиотека для маршрутизации в React-приложениях. |
| Notistack [57] | v3.0.1 | Библиотека - менеджер уведомлений для React-приложений. |

Программные библиотеки позволяют упростить реализацию web-приложения.

## **Сторонние сервисы Stripe, LocalStack**

Для успешной реализации функционала web-приложения были использованы следующие сторонние сервисы:

1. Stripe — интеграция с этим сервисом позволила реализовать обработку онлайн-платежей в приложении.
2. **LocalStack — используется для** имитации работы сервисов AWS. Интеграция с входящим в него s3 использовалась для хранения фотографий.

## **Структура серверной части**

Структура серверной части основывается на микро сервисной архитектуре.

За основу были приняты паттерн API Gateway и принцип разделения сервисов, по которому каждый сервис отвечает за отдельную сущность или процесс в web-приложения. API Gateway представлен сервисом, который представляет из себя REST API сервер принимающий HTTP запросы и преобразующий их в Protobuf формат для дальнейшей отправки на другие сервисы. Еще в нем есть механизмы аутентификации пользователей. А также методы работы с платежным шлюзом.

Основные компоненты структуры Gateway сервиса включают в себя несколько ключевых элементов, которые обеспечивают эффективную работу приложения:

1. Маршрутизаторы — управляют маршрутами и направляют запросы к соответствующим контроллерам.
2. Контроллеры — обрабатывают запросы от клиента, преобразуют их в формат Protobuf для дальнейшей отправки в домен-сервисы и возвращают ответы.
3. Middleware — промежуточные обработчики, используемые для валидации данных и обеспечения безопасности.

**В таблице 3.4 приведён список директорий серверной части проекта.**

Таблица 3.4 – Директории Gateway сервиса

|  |  |
| --- | --- |
| Директория | Назначение |
| cmd | Является точкой входа в приложение.  Подгружает переменные окружения из .env для удобного последующего обращения к ним. Запускает метод инициализации структуры всего сервиса. Запускает метод запуска сервиса на порту. |
| docs | Содержит файлы для swagger-документации. |
| internal | Содержит пакеты доступные только внутри текущего проекта. Не могут быть импортированы из другого проекта. |
| pkg | Cодержит общедоступные переиспользуемые пакеты – логгер. |
| Internal/server | Cодержит структуру всего сервиса, методы ее инициализации, а также метод запуска сервиса. |
| internal/<название сервиса> | Содержит каталоги для работы с сервисом |
| internal/<название сервиса>/rest | Содержит объявление контроллеров и метод связывания маршрутов с контроллерами. |
| internal/<название сервиса>/ middlewares | Содержит middlewares специфичные для конкретного сервиса. |
| internal/<название сервиса>/dtos | Содержит data transfer objects специфичные для конктретного сервиса. |
| internal/<название сервиса>/< название сервиса >\_erros | Содержит ошибки специфичные для конкретного сервиса. |
| internal/common\_middlewares | Содержит общие middlewares: аутентификация, авторизация и определение общих ошибок. |

**Таблица соответствия маршрутов контроллерам в исходном коде представлена в таблице 3.5.**

Таблица 3.5 – Контроллеры и функции маршрутов

|  |  |  |  |
| --- | --- | --- | --- |
| Метод | Маршрут | Контроллер | Описание |
| GET | /abonements | GetAbonements | Возвращает абонементы с их услугами. |
| POST | /abonements | CreateAbonement | Передает данные для создания абонемента и связей с услугами другим сервисам, возвращает созданный абонемент с его услугами. |

Продолжение таблицы 3.5

|  |  |  |  |
| --- | --- | --- | --- |
| PUT | /abonements | UpdateAbonement | Передает данные для обновления абонемента и связей с услугами другим сервисам, возвращает обновленный абонемент с его услугами. |
| DELETE | /abonements/:id | DeleteAbonement | Передает данные для удаления абонемента и его связей с услугами другим сервисам, возвращает удаленный абонемент. |
| GET | /coaches | GetCoaches | Возвращает тренеров с их услугами, отзывами и пользователями, которые оставили эти отзывы. |
| POST | api/delete-product | CreateCoach | Передает данные для создания тренера и связей с услугами другим сервисам, возвращает созданного тренера с его услугами. |
| PUT | api/product-details | UpdateCoach | Передает данные для обновления тренера и связей с услугами другим сервисам, возвращает обновленного тренера с его услугами. |
| DELETE | /coaches/:id | DeleteCoach | Передает данные для удаления тренера другим сервисам, возвращает удаленного тренера. |
| POST | /checkout-session-completed | HandleCheckoutSessionCompleted | Обрабатывают данные о успешно пройденной оплате, передают данные для создания заказа другим сервисам, возвращает созданный заказ. |
| POST | /create-checkout-session | CreateCheckoutSession | Передают данные для создания stripe checkout сессии другим сервисам, возвращает URL созданной stripe checkout сессии. |
| GET | /orders/:userd | GetUserOrders | Передают данные для возврата заказов клиента другим сервисам, возвращает заказы клиента. |
| POST | /reviews | CreateCoachReview | Передает данные для создания отзыва и связи с тренером другим сервисам, возвращает созданный отзыв с клиентом, оставившим его. |
| GET | / services | GetServices | Возвращает все услуги. |
| POST | /sso/signUp | SignUp | Передает данные для регистрации другим сервисам, устанавливают в куки refreshToken, возвращает созданного клиента и accessToken. |

Продолжение таблицы 3.5

|  |  |  |  |
| --- | --- | --- | --- |
| POST | /sso/signIn | SignIn | Передает данные для входа другим сервисам, устанавливают в куки refreshToken, возвращает созданного клиента и accessToken. |
| POST | /sso/refresh | Refresh | Передает данные для обновления refresh сессии другим сервисам, устанавливают в куки refreshToken, возвращает созданного клиента и accessToken. |
| POST | /sso/logOut | LogOut | Передает данные для удаления refresh сессии другим сервисам, затирае куки refreshToken в куках. |
| PUT | /users/:id | UpdateUser | Передает данные для обновления пользователя другим сервисам, возвращает обновленного пользователя. |
| GET | /users | GetClients | Возвращает всех клиентов. |
| DELETE | /users/:id | DeleteClientById | Передает данные для удаления клиента другим сервисам, возвращает удаленного клиента. |

**При передаче данных между клиентом и Gateway cсервисом используется формат JSON (JavaScript Object Notation).**

Cервисами образующих домены для работы с отдельными частями приложения.

**В таблице 3.4 приведён список доменов и их назначение.**

Таблица 3.6 – Домены серверной части проекта

|  |  |
| --- | --- |
| Директория | Назначение |
| SSO | Реализует единую точку входа в приложение. Реализует функции регистрации, входа, выхода и работу с токенами access и refresh. |
| Migrate | Нужен для начальной инициализации схемы базы данных и для оповещения других сервисах о том, что схема базы данных находится в последней версии. |
| Abonement | Реализует логику работы с абонементами |
| User | Реализует логику работы с пользователями |
| Review | Реализует логику работы с отзывами |
| Order | Реализует логику работы с заказами |
| Coach | Реализует логику работы с тренерами |
| Service | Реализует логику работы с услугами |

Каждый домен представляет из себя gRPC сервис.

Основные компоненты структуры домен-сервиса включают в себя несколько ключевых элементов, которые обеспечивают эффективную работу приложения:

1. Методы — принимают запросы от клиентов, выполняют бизнес-логику через слой Usecase и возвращают ответы.
2. Usecase — содержат бизнес-логику, обрабатывают данные и взаимодействуют с репозиториями.
3. Repository – занимаются взаимодействием с базой данных.

**В таблице 3.7 приведён список директорий домен-сервиса и назначение файлов, хранящихся в этих директориях.**

Таблица 3.7 – Директории **домен-сервиса**

|  |  |
| --- | --- |
| Директория | Назначение |
| cmd | Является точкой входа в приложение.  Подгружает переменные окружения из .env для удобного последующего обращения к ним. Запускает метод инициализации структуры всего сервиса. Запускает метод запуска сервиса на порту. |
| internal | Содержит пакеты доступные только внутри текущего проекта. Не могут быть импортированы из другого проекта. |
| pkg | Cодержит общедоступные переиспользуемые пакеты – логгер. |
| internal/server | Cодержит структуру всего сервиса, методы ее инициализации, а также метод запуска сервиса. |
| internal/delivery | Содержит структуру gRPC сервера, метод ее инициализации и методы обработки запросов, которые вызывают слой usecase. |
| internal/usecase | Содержит интерфейсы и их реализацию для usecase, которые выполняют бизнес логику. Usecase вызывают слой repository. |
| internal/repository | Содержит интерфейсы и их реализацию для repository, которые работают с базой данных. |
| internal/models | Содержит модели для работы с базой данных. |
| internal/dtos | Содержит структуры для переноса данных их слоя обработчиков, в слой usecase. |
| internal/constants | Содержит общие константы сервиса. |
| internal/errors | Содержит общие ошибки сервиса. |

**Protobuf для сервисов представлен в приложении В.**

**Таблица соответствия gRPC методов домен-сервиса абонементов с их описанием представлено в таблице 3.5.**

**Таблица 3.8 – Таблица соответствия gRPC методов домен-сервиса абонементов с их описанием**

|  |  |
| --- | --- |
| Метод | Назначение |
| CreateAbonement | Создает абонемент, его связи с услугами, stripe продукт, stripe цену. Сохраняет фото в s3 localstack. Возвращает абонемент со связанными услугами. |
| GetAbonementById | Возвращает абонемент по введенному id. |
| UpdateAbonement | Обновляет абонемент, его связи с услугами, stripe цен, фото в s3 localstack. Возвращает абонемент со связанными услугами. |
| DeleteAbonementById | Удаляет абонемент, его взяи с услугами, фото в s3 localstack. Архивирует stripe продукт. Возвращает удаленный абонемент. |
| GetAbonements | Возвращает все абонементы. |
| GetAbonementsWithServices | Возвращает все абонементы со связанными с ними услугами. |
| GetAbonementsByIds | Возвращает абонементы по введенным ids. |

**Таблица 3.8 – Таблица соответствия gRPC методов домен-сервиса тренеров с их описанием**

|  |  |
| --- | --- |
| Метод | Назначение |
| CreateCoach | Создает тренера, его связи с услугами. Сохраняет фото в s3 localstack. Возвращает тренера со связанными услугами. |
| GetCoachById | Возвращает тренера по введенному id. |
| UpdateCoach | Обновляет тренера, его связи с услугами, фото в s3 localstack. Возвращает тренера со связанными услугами. |
| DeleteCoachById | Удаляет тренера, его связи с услугами, фото в s3 localstack. Возвращает удаленного тренера. |
| GetCoaches | Возвращает всех тренеров. |
| GetCoachesWithServicesWithReviewsWithUsers | Возвращает всех тренеров с относящимися к ним услугам, с относящимися к ним отзывами и клиентами оставившими их. |

**Таблица 3.9 – Таблица соответствия gRPC методов домен-сервиса заказов с их описанием**

|  |  |
| --- | --- |
| Метод | Назначение |
| CreateOrder | Создает заказ. Возвращает заказ. |

Продолжение таблицы 3.9

|  |  |
| --- | --- |
| GetUserOrders | Возвращает все заказы клиента. |
| CreateCheckoutSession | Создает Stripe checkout сессию. Возвращает ее URL. |

**Таблица 3.10 – Таблица соответствия gPRC методов домен-сервиса отзывов с их описанием**

|  |  |
| --- | --- |
| Метод | Назначение |
| CreateCoachReview | Создает отзыв для тренера. Возвращает отзывы тренера. |
| GetReviewById | Возвращает отзыв по введенному id. |
| UpdateReview | Обновляет отзыв. Возвращает обновленный отзыв. |
| DeleteReviewById | Удаляет отзыв по введенному id. Возвращает удаленный отзыв. |
| GetCoachReviews | Возвращает отзывы тренера. |
| GetCoachesReviews | Возвращает отзывы тренеров. |

**Таблица 3.11 – Таблица соответствия gRPC методов домен-сервиса услуг с их описанием**

|  |  |
| --- | --- |
| Метод | Назначение |
| CreateService | Создает услугу, сохраняет фото в s3 localstack. Возвращает созданную услугу. |
| GetServiceById | Возвращает услугу по введенному id. |
| UpdateService | Обновляет услугу, обновляет фото в s3 localstack. Возвращает обновленную услугу. |
| DeleteServiceById | Удаляет услугу по введенному id, удаляет фото из s3 localstack. Возвращает удаленную услугу. |
| GetServices | Возвращает все услуги. |
| CreateCoachServices | Создает связи услуг с тренером. Возвращает связи тренера с услугами. |
| CreateAbonementServices | Создает связи услуг с абонементом. Возвращает связи абонемента с услугами. |
| UpdateAbonementServices | Обновляет связи услуг с абонементом. Возвращает обновленные связи абонемента с услугами. |
| UpdateCoachServices | Обновляет связи услуг с тренером. Возвращает обновленные связи тренера с услугами. |
| GetAbonementsServices | Возвращает услуги абонементов. |
| GetCoachesServices | Возвращает услуги тренеров. |

**Таблица 3.12 – Таблица соответствия gRPC методов домен-сервиса sso с их описанием**

|  |  |
| --- | --- |
| Метод | Назначение |
| SignUp | Создает клиента, refresh сессию, access и refresh токены. Возвращает клиента, access и refresh токены. |
| SignIn | Идентифицирует пользователя, обновляет refresh сессию, access и refresh токены. Возвращает пользователя, access и refresh токены. |
| LogOut | Удаляет refresh сессию. |
| Refresh | Обновляет refresh сессию, access и refresh токены. Возвращает пользователя, access и refresh токены. |

**Таблица 3.13 – Таблица соответствия gRPC методов домен-сервиса пользователя с их описанием**

|  |  |
| --- | --- |
| Метод | Назначение |
| CreateUser | цСоздает пользователя, сохраняет фото в s3 localstack. Возвращает созданного пользователя. |
| GetUserById | Возвращает пользователя по введенному id. |
| UpdateUser | Обновляет пользователя, фото в s3 localstack. Возвращает обновленного пользователя. |
| DeleteUserById | Удаляет пользователя по введенному id. Возвращает удаленного пользователя. |
| GetUserByEmail | Возвращает пользователя по введенному email. |
| CheckPassword | Проверяет пароль введенный пароль на валидность. |
| GetUsersByIds | Возвращает пользователей по ids. |
| GetClients | Возвращает всех клиентов. |

**При передаче данных между домен-сервисами используется формат Protobuf.**

## **Реализация функционала для пользователя с ролью «Гостя»**

## **Регистрация**

Для гостя доступна регистрация, которая позволяет ему создать учетную запись в системе. Этот процесс реализован в контроллере SignUp

Исходный код контроллера приведен в листинге 3.1.

func (h \*Handler) SignUp(c \*gin.Context) {  
 suReq := &dtos.SignUpRequest{}  
 if err := c.ShouldBindJSON(&suReq); err != nil {  
 logger.ErrorLogger.Printf("Error binding SignUpRequest: %v", err)  
 c.JSON(http.*StatusBadRequest*, gin.H{"error": err.Error()})  
 return  
 }  
  
 err := h.validator.Struct(suReq)  
 if err != nil {  
 if validationErrors, ok := err.(validator.ValidationErrors); ok {  
 customMessages := make(map[string]string)  
 for \_, fieldErr := range validationErrors {  
 customMessages[fieldErr.Field()] = getCustomErrorMessage(fieldErr)  
 }  
  
 c.JSON(http.*StatusBadRequest*, gin.H{"errors": customMessages})  
 return  
 }  
  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": "Internal validation error"})  
 return  
 }  
  
 fingerPrintValue, exists := c.Get(os.Getenv("APP\_FINGERPRINT\_REQUEST\_KEY"))  
 if !exists {  
 logger.ErrorLogger.Printf("Error getting fingerprint: %v", sso\_errors.FingerPrintNotFoundInContext)  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": sso\_errors.FingerPrintNotFoundInContext})  
 return  
 }  
  
 FingerPrintValueCasted, ok := fingerPrintValue.(string)  
 if !ok {  
 logger.ErrorLogger.Printf("Error casting fingerprint to string: %v", err)  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": sso\_errors.FingerPrintNotFoundInContext})  
 return  
 }  
  
 suReq.FingerPrint = FingerPrintValueCasted

Продолжение листинга 3.1

ssoClient := ssoGRPC.NewSSOClient(h.ssoClient)  
  
 signUpRequest := &ssoGRPC.SignUpRequest{}  
 signUpRequest.Name = suReq.Name  
 signUpRequest.Email = suReq.Email  
 signUpRequest.Password = suReq.Password  
 signUpRequest.FingerPrint = suReq.FingerPrint  
  
 upRes, err := ssoClient.SignUp(context.Background(), signUpRequest)  
 if err != nil {  
 logger.ErrorLogger.Printf("Error SignUp: %v", err)  
  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
 ateInt, err := strconv.Atoi(upRes.AccessTokenExpiration)  
 if err != nil {  
 logger.ErrorLogger.Printf("Error convert AccessTokenExpiration to int: %v", err)  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
 rteInt, err := strconv.Atoi(upRes.RefreshTokenExpiration)  
 if err != nil {  
 logger.ErrorLogger.Printf("Error convert RefreshTokenExpiration to int: %v", err)  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
 c.SetCookie(  
 "refreshToken",  
 upRes.RefreshToken,  
 rteInt/refreshTokenMaxAgeDivider,  
 "",  
 "",  
 false,  
 true,  
 )  
 c.JSON(http.StatusOK, gin.H{  
 "accessToken": upRes.GetAccessToken(),  
 "accessTokenExpiration": ateInt / accessTokenMaxAgeDivider,  
 "user": upRes.GetUser(),  
 })  
}

Листинг 3.1 – Реализация метода signUp

Он принимает HTTP-запрос с данными из тела, включая email, имя, пароль.

Маппит данные на структуру для получения конкретного типа данных. Валидирует полученную структуру. Дополнительно, из контекста запроса извлекается параметр fingerprint, который используется для повышения безопасности токенов. Создает запрос в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод SignUp у сервиса SSO, который руководит создание клиента, Refresh сессии, refresh и access токенов.

В результате выполнения метода возвращаются accessToken, созданный клиент и refreshToken устанавливается в куки.

## **Аутентификация**

Для гостя доступна аутентификация, при входе в приложение при помощи контроллера SignIn.

Реализация методов представлена на листинге 3.2.

func (h \*Handler) SignIn(c \*gin.Context) {  
 siReq := &dtos.SignInRequest{}  
 if err := c.ShouldBindJSON(siReq); err != nil {  
 logger.ErrorLogger.Printf("Error parsing SignInRequest: %v", err)  
 c.JSON(http.*StatusBadRequest*, gin.H{"error": err.Error()})return}  
 err := h.validator.Struct(siReq)  
 if err != nil {  
 if validationErrors, ok := err.(validator.ValidationErrors); ok { customMessages := make(map[string]string) for \_, fieldErr := range validationErrors { customMessages[fieldErr.Field()] = getCustomErrorMessage(fieldErr)  
 }  
 c.JSON(http.*StatusBadRequest*, gin.H{"errors": customMessages})  
 return  
 }  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": "Internal validation error"})  
 return  
 }  
 fingerPrintValue, exists := c.Get(os.Getenv("APP\_FINGERPRINT\_REQUEST\_KEY"))  
 if !exists {  
 logger.ErrorLogger.Printf("Error binding SignUpRequest: %v", sso\_errors.FingerPrintNotFoundInContext)  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": sso\_errors.FingerPrintNotFoundInContext})  
 return  
 }  
 FingerPrintValueCasted, ok := fingerPrintValue.(string)  
 if !ok {  
 logger.ErrorLogger.Printf("Error casting fingerprint to

Продолжение листинга 3.2

string: %v", sso\_errors.FingerPrintNotFoundInContext)  
 c.JSON(http.StatusInternalServerError, gin.H{"error": sso\_errors.FingerPrintNotFoundInContext})  
 return  
 }  
 siReq.FingerPrint = FingerPrintValueCasted  
 ssoClient := ssoGRPC.NewSSOClient(h.ssoClient)  
 signIpRequest := &ssoGRPC.SignInRequest{  
 Email: siReq.Email,  
 Password: siReq.Password,  
 FingerPrint: siReq.FingerPrint,  
 }  
 siRes, err := ssoClient.SignIn(context.Background(), signIpRequest)  
 if err != nil {  
 logger.ErrorLogger.Printf("Error SignIp: %v", err)  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
 ateInt, err := strconv.Atoi(siRes.GetAccessTokenExpiration())  
 if err != nil {  
 logger.ErrorLogger.Printf("Error convert AccessTokenExpiration to int: %v", err)  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
 rteInt, err := strconv.Atoi(siRes.GetRefreshTokenExpiration())  
 if err != nil {  
 logger.ErrorLogger.Printf("Error convert RefreshTokenExpiration to int: %v", err)  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
  
 c.SetCookie(  
 "refreshToken",  
 siRes.GetRefreshToken(),  
 rteInt/refreshTokenMaxAgeDivider,  
 "",  
 "", false, true,  
 )  
 c.JSON(http.StatusOK, gin.H{  
 "user": siRes.GetUser(),  
 "accessToken": siRes.GetAccessToken(),  
 "accessTokenExpiration": ateInt / accessTokenMaxAgeDivider,

Листинг 3.2 – Реализация метода SignIn

Он принимает HTTP-запрос с данными из тела, включая email, пароль. Маппит данные на структуру для получения конкретного типа данных. Валидирует полученную структуру. Дополнительно, из контекста запроса извлекается параметр fingerprint, который используется для повышения безопасности токенов. Создает запрос в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод SignIn у сервиса SSO, который руководит идентификацией пользователя, обновлением Refresh сессии, созданием refresh и access токенов.

В результате выполнения метода возвращаются accessToken, пользователь, refreshToken устанавливается в куки.

## **Реализация функционала, доступного клиентам и администраторам**

## **Просмотр информации о абонементах**

У клиентов и администраторов есть возможность просмотра информации о абонементах. Эта возможность реализована в контроллере GetAbonements. Исходный код контроллера приведен в листинге 3.3.

func (h \*Handler) GetAbonements(c \*gin.Context) {  
 abonements, err := (\*h.abonementClient).GetAbonementsWithServices(c.Request.Context(), &emptypb.Empty{})  
 if err != nil {  
  
 c.JSON(http.*StatusInternalServerError*, gin.H{  
 "err": err,  
 })  
  
 return  
 }  
  
 c.JSON(http.*StatusOK*, gin.H{  
 "abonements": abonements,  
 })  
}

Листинг 3.3 – Реализация контроллера GetAbonements

Этот контроллер обращается к методу GetAbonementsWithServices для получения всех абонементов с относящимися к ним услугами.

В результате выполнения возвращаются абонементы с относящимися к ним услугами.

## **Поиска, сортировка, фильтрация абонементов**

У клиентов и администраторов есть возможность поиска, сортировки, фильтрации абонементов. Эта возможность реализована в методах filterData и sortAbonnements на клиентской части.

Исходный код метода filterData приведен в листинге 3.4.

function filterData(data, searchName, validityPeriod, minPrice, maxPrice) {  
 return data.filter(item => {  
 const matchesName = searchName  
 ? item.abonement.title.toLowerCase().includes(searchName.toLowerCase())  
 : true;  
  
 const matchesValidityPeriod = validityPeriod  
 ? item.abonement.validity.includes(validityPeriod)  
 : true;  
  
 const matchesPrice = (minPrice === undefined || item.abonement.*price* >= minPrice) &&  
 (maxPrice === undefined || item.abonement.*price* <= maxPrice);  
  
 return matchesName && matchesPrice && matchesValidityPeriod;  
 });  
}

Листинг 3.4 – Реализация метода filterData

Метод filterData фильтрует абонементы по имени, периоду валидности и цене.

Исходный код метода sortAbonnements приведен в листинге 3.5.

const sortAbonnements = (abonnements, sortingFilter, order) => {

return abonnements.slice().sort((a, b) => {

if (sortingFilter === "price") {

return order === "asc"

? a.abonement.price - b.abonement.price

: b.abonement.price - a.abonement.price;

}

if (sortingFilter === "title") {

const titleA = a.abonement.title.toLowerCase();

const titleB = b.abonement.title.toLowerCase();

if (order === "asc") {

return titleA.localeCompare(titleB);

} else {

return titleB.localeCompare(titleA);

}

}

return 0;

});

Листинг 3.5 – Реализация метода sortAbonnements

Метод sortAbonnements сортирует абонементы по цени или названию.

## **Просмотр информации о тренерах и** **отзывов о тренерах**

У клиентов и администраторов есть возможность просмотра информации о тренерах и отзывов о тренерах. Эта возможность реализована в контроллере GetCoaches.

Реализация этого контроллера приведена в листинге 3.6.

func (h \*Handler) GetCoaches(c \*gin.Context) {  
 coaches, err := (\*h.coachClient).GetCoachesWithServicesWithReviewsWithUsers(c.Request.Context(), &emptypb.Empty{})  
 if err != nil {  
  
 c.JSON(http.*StatusInternalServerError*, gin.H{  
 "err": err,  
 })  
  
 return  
 }  
  
 c.JSON(http.*StatusOK*, gin.H{  
 "coaches": coaches,  
 })  
}

Листинг 3.6 – Реализация контроллера GetCoaches

Этот контроллер обращается к методу GetCoachesWithServicesWithReviewsWithUsers для получения всех тренеров с их услугами и с отзывами, а также с пользователями, которые их оставили.

В результате выполнения возвращаются все тренера с их услугами и с отзывами, а также с пользователями, которые их оставили.

## **Выход из системы**

У клиентов и администраторов есть возможность выхода из системы. Эта возможность реализована в контроллере LogOut.

Реализация этого контроллера приведена в листинге 3.7.

func (h \*Handler) LogOut(c \*gin.Context) {  
  
 refreshToken, err := c.Cookie("refreshToken")  
 if err != nil {  
 logger.ErrorLogger.Printf("Error getting refreshToken from cookie: %v", err)  
 c.JSON(http.*StatusBadRequest*, gin.H{"error": err.Error()})  
 return  
 }  
 ssoClient := ssoGRPC.NewSSOClient(h.ssoClient)

Продолжение листинга 3.7

logOutRequest := &ssoGRPC.LogOutRequest{}  
 logOutRequest.RefreshToken = refreshToken  
  
 \_, err = ssoClient.LogOut(context.Background(), logOutRequest)  
 if err != nil {  
 logger.ErrorLogger.Printf("Error LogOut: %v", err)  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
  
 c.SetCookie("refreshToken", "", -1, "/", "", false, true)  
  
 c.Status(http.StatusOK)  
}

Листинг 3.7 – Реализация контроллера LogOut

Этот контроллер обращается к методу LogOut для удаления refresh сессии.

В результате выполнения возвращаются затирается refresh токен из куков.

## **Реализация функционала для пользователя с ролью «Клиент»**

## **Оставить отзыв о тренере**

У клиентов есть возможность оставить отзыв о тренере. Эта возможность реализована в контроллере CreateCoachReview.

Реализация этого контроллера приведена в листинге 3.8.

func (h \*Handler) CreateCoachReview(c \*gin.Context) {  
  
 type CoachReviewDataForCreate struct {  
 UserId uuid.UUID  
 Body string  
 CoachId uuid.UUID  
 }  
  
 coachReviewDataForCreate := &CoachReviewDataForCreate{}  
  
 createCoachReviewRequest := &reviewGRPC.CreateCoachReviewRequest{  
 ReviewDataForCreate: &reviewGRPC.CoachReviewDataForCreate{  
 UserId: "",  
 Body: "",  
 CoachId: "",  
 },  
 }  
  
 coachReviewDataForCreateProto :=

Продолжение листинга 3.8

&reviewGRPC.CoachReviewDataForCreate{}  
  
 if err := c.ShouldBindJSON(&coachReviewDataForCreate); err != nil {  
 logger.ErrorLogger.Printf("Error binding CreateCoachReviewRequest: %v", err)  
 c.JSON(http.StatusBadRequest, gin.H{"error": "Bad CreateCoachReviewRequest"})  
 return  
 }  
  
 coachReviewDataForCreateProto.CoachId = coachReviewDataForCreate.CoachId.String()  
 coachReviewDataForCreateProto.Body = coachReviewDataForCreate.Body  
 coachReviewDataForCreateProto.UserId = coachReviewDataForCreate.UserId.String()  
  
 \_, err := uuid.Parse(coachReviewDataForCreateProto.CoachId)  
 if err != nil {  
 logger.ErrorLogger.Printf("id must be uuid")  
 c.JSON(http.StatusBadRequest, gin.H{"error": "id must be uuid"})  
 c.Set("InvalidUpdate", struct{}{})  
 return  
 }  
  
 if len(coachReviewDataForCreateProto.Body) < 10 || len(coachReviewDataForCreateProto.Body) > 255 {  
 logger.ErrorLogger.Printf("Review body must be between 10 and 255 characters long")  
 c.JSON(http.StatusBadRequest, gin.H{"error": "Review body must be between 10 and 255 characters long"})  
 return  
 }  
  
 \_, err = uuid.Parse(coachReviewDataForCreateProto.UserId)  
 if err != nil {  
 logger.ErrorLogger.Printf("id must be uuid")  
 c.JSON(http.StatusBadRequest, gin.H{"error": "id must be uuid"})  
 return  
 }  
 createCoachReviewRequest.ReviewDataForCreate = coachReviewDataForCreateProto  
 review, err := (\*h.reviewClient).CreateCoachReview(context.TODO(), createCoachReviewRequest)  
 if err != nil {

Продолжение листинга 3.8

logger.ErrorLogger.Printf("Failed CreateCoachReview: %s", err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
  
 getUserByIdRequest := &userGRPC.GetUserByIdRequest{  
 Id: review.ReviewObject.UserId,  
 }  
  
 user, err := (\*h.userClient).GetUserById(context.Background(), getUserByIdRequest)  
 if err != nil {  
 logger.ErrorLogger.Printf("Failed GetUserById: %s", err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err.Error()})  
 return  
 }  
  
 reviewWithUser := &coachGRPC.ReviewWithUser{  
 ReviewObject: review.ReviewObject,  
 UserObject: user.UserObject,  
 }  
  
 c.JSON(http.StatusOK, gin.H{  
 "reviewWithUser": reviewWithUser,  
 })  
}

Листинг 3.8 – Реализация метода signIn

Он принимает HTTP-запрос с данными из тела id пользователя, тело отзыва, id тренера. Маппит данные на структуру для получения конкретного типа данных. Валидирует полученную структуру. Дополнительно, из контекста запроса извлекается параметр fingerprint, который используется для повышения безопасности токенов. Создает запрос в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод CreateCoachReview у сервиса Review, который руководит созданием отзыва к тренеру. Вызывает метод GetUserById для взяти пользователя по id пользователя.

В результате выполнения контроллера возвращаются отзыв и пользователь написавший его.

## **Покупка абонементов**

У клиентов есть возможность покупки абонементов. Эта возможность реализована в контроллере CreateCheckoutSession.

Реализация этого контроллера приведена в листинге 3.9.

func (h \*Handler) CreateCheckoutSession(c \*gin.Context) {  
  
 ccsDto := &dtos.CreateCheckoutSessionDTO{}  
  
 if err := c.ShouldBindJSON(&ccsDto); err != nil {  
 logger.ErrorLogger.Printf("Error binding CreateCheckoutSessionRequest: %v", err)  
 c.JSON(http.*StatusBadRequest*, gin.H{"error": err.Error()})  
 return  
 }  
  
 createCheckoutSessionRequest := &orderGRPC.CreateCheckoutSessionRequest{  
 ClientId: ccsDto.ClientId,  
 AbonementId: ccsDto.AbonementId,  
 StripePriceId: ccsDto.StripePriceId,  
 }  
  
 session, err := (\*h.orderClient).CreateCheckoutSession(context.TODO(), createCheckoutSessionRequest)  
 if err != nil {  
 return  
 }  
  
 c.JSON(http.*StatusOK*, gin.H{"sessionUrl": session.GetSessionUrl()})  
}

Листинг 3.9 – Реализация контроллера CreateCheckoutSession

Он принимает HTTP-запрос с данными из тела: id клиента, id абонемента, id цены в stripe. Маппит данные на структуру для получения конкретного типа данных. Создает запрос в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод CreateCheckoutSession у сервиса Order, который руководит созданием stripe checkout сессии, приведен в приложении Г.

В результате выполнения контроллера возвращаются URL stripe checkout сессии. Клиента перенаправляет на этот URL, где он может заполнить данные своей карты и оплатить абонемент.

После успешной оплаты Stripe посылает событие об успешном завершении stripe checkout на Gateway service, с обработкой на HandleCheckoutSessionCompleted, приведен в приложении Д. После чего Gateway service инициирует создание объекта заказа в Order service.

## **Редактирование профиля**

У клиентов есть возможность редактирования профиля. Эта возможность реализована в контроллере UpdateUser.

Реализация этого контроллера приведена в листинге 3.10.

func (h \*Handler) UpdateUser(c \*gin.Context) {  
  
 userId := c.Param("id")  
  
 userIdFromToken, exists := c.Get("UserIdFromToken")  
 if !exists {  
 logger.ErrorLogger.Printf("Cant find UserIdFromToken in context")  
 return  
 }  
  
 if userId != userIdFromToken {  
 c.JSON(http.*StatusForbidden*, gin.H{  
 "error": "Access denied: you cannot update another user's data",  
 })  
 return  
 }  
  
 cmd := &dtos.User{}  
  
 form, err := c.MultipartForm()  
 if err != nil {  
 return  
 }  
  
 name, namOk := form.Value["name"]  
 photo := form.File["photo"]  
  
 if namOk {  
 cmd.Name = name[0]  
 }  
  
 err = h.validator.Struct(cmd)  
 if err != nil {  
 logger.ErrorLogger.Printf("Error validating UpdateUserRequest: %v", err)  
  
 c.AbortWithStatusJSON(http.*StatusBadRequest*, gin.H{"error": "name must be from 2 to 100 symbols"})  
 return  
 }  
 userClient := userGRPC.NewUserClient(h.userClient)  
 stream, err := userClient.UpdateUser(context.Background())  
 if err != nil {  
 fmt.Printf("failed to stat file: %v\n", err)  
 }  
  
 userDataForUpdate := &userGRPC.UserDataForUpdate{

Продолжение листинга 3.10

Id: userId,  
 Email: "",  
 Name: "",  
 Role: "",  
 }  
 userDataForUpdate.Id = userId  
 if name != nil {  
 userDataForUpdate.Name = name[0]  
 }  
 updateUserRequestUserDataForUpdate := &userGRPC.UpdateUserRequest\_UserDataForUpdate{  
 UserDataForUpdate: userDataForUpdate,  
 }  
 createUserRequest := &userGRPC.UpdateUserRequest{  
 Payload: updateUserRequestUserDataForUpdate,  
 }  
 err = stream.Send(createUserRequest)  
 if err != nil {  
 return  
 }  
 if photo != nil && len(photo) > 0 {  
 buffer := make([]byte, 1024\*1024)  
 file, err := photo[0].Open()  
 if err != nil {  
 return  
 }  
 for {  
 n, err := file.Read(buffer)  
 if err == io.EOF {  
 break  
 }  
 if err != nil {  
 return  
 }  
  
 err = stream.Send(&userGRPC.UpdateUserRequest{  
 Payload: &userGRPC.UpdateUserRequest\_UserPhoto{  
 UserPhoto: buffer[:n],  
 },  
 },  
 )  
 if err != nil {  
 return}}}  
 res, err := stream.CloseAndRecv()  
if err != nil {return}  
c.JSON(http.StatusOK, gin.H{"user": res.GetUserObject(),})}

Листинг 3.10 – Реализация котроллера UpdateUser

Он принимает HTTP-запрос с данными из тела: id клиента, имя, фото. Маппит данные на структуру для получения конкретного типа данных. Валидирует данные. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод Send у сервиса User, который руководит обновлением пользователя.

В результате выполнения контроллера обновленный пользователь.

## **Просмотр списка своих заказов**

У клиентов есть возможность просмотра списка своих заказов. Эта возможность реализована в контроллере GetUserOrders.

Реализация этого контроллера приведена в листинге 3.11.

func (h \*Handler) GetUserOrders(c \*gin.Context) {  
 id := c.Param("userId")  
  
 convertedId, err := uuid.Parse(id)  
 if err != nil {  
  
 c.AbortWithStatusJSON(http.*StatusBadRequest*, gin.H{"error": fmt.Errorf("invalid id format")})  
 return  
 }  
  
 \_ = convertedId  
  
 getUserOrdersRequest := &orderGRPC.GetUserOrdersRequest{  
 UserId: id,  
 }  
  
 orders, err := (\*h.orderClient).GetUserOrders(context.TODO(), getUserOrdersRequest)  
 if err != nil {  
 return  
 }  
  
 c.JSON(http.*StatusOK*, gin.H{  
 "orders": orders.OrderObjectWithAbonementWithServices,  
 })  
}

Листинг 3.11 – Реализация метода GetUserOrders

Он принимает HTTP-запрос с id пользователя. Маппит данные на структуру для получения конкретного типа данных. Валидирует данные. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод GetUserOrders у сервиса Order, который возвращает заказы пользователя.

В результате выполнения контроллера возвращаются заказы пользователя.

## **Реализация функционала для пользователя с ролью «Администратор»**

## **Редактирования клиентов**

Администратор системы имеет возможность редактирования клиентов – удаление клиентов. Логика удаления клиента в контроллере DeleteClientById, представлена на листинге 3.12.

func (h \*Handler) DeleteClientById(c \*gin.Context) {  
  
 userClient := userGRPC.NewUserClient(h.userClient)  
  
 id := c.Param("id")  
  
 convertedId, err := uuid.Parse(id)  
 if err != nil {  
  
 c.AbortWithStatusJSON(http.*StatusBadRequest*, gin.H{"error": fmt.Errorf("invalid id format")})  
 return  
 }  
  
 \_ = convertedId  
  
 deleteUserByIdRequest := &userGRPC.DeleteUserByIdRequest{  
 Id: id,  
 }  
  
 client, err := userClient.DeleteUserById(context.Background(), deleteUserByIdRequest)  
 if err != nil {  
 return  
 }  
  
 c.JSON(http.*StatusOK*, gin.H{  
 "client": client.UserObject,  
 })  
}

Листинг 3.12 – Реализация метода DeleteClientById

Он принимает HTTP-запрос с id пользователя. Валидирует данные. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод DeleteUserById у сервиса User, который руководит удалением пользователя.

В результате выполнения контроллера возвращаются удаленный пользователь.

## **Редактирования абонементов**

Администратор системы имеет возможность редактирования абонементов – добавление, обновление, удаление. Логика добавления абонемента реализована в контроллере CreateAbonement, представленном на листинге 3.13.

func (h \*Handler) CreateAbonement(c \*gin.Context) {  
  
 createAbonementCommandAny, exists := c.Get("CreateAbonementCommand")  
 if !exists {  
 logger.ErrorLogger.Printf("Cant find CreateAbonementCommand in context")  
 return  
 }  
  
 createAbonementCommand, ok := createAbonementCommandAny.(\*dtos.CreateAbonementCommand)  
 if !ok {  
 logger.ErrorLogger.Printf("CreateAbonementCommand has an invalid type")  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": "CreateAbonementCommand has an invalid type"})  
 return  
 }  
  
 stream, err := (\*h.abonementClient).CreateAbonement(context.Background())  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": "Failed to open CreateAbonement stream"})  
 return  
 }  
  
 abonementDataForCreate := &abonementGRPC.AbonementDataForCreate{  
 Title: createAbonementCommand.Title,  
 Validity: createAbonementCommand.ValidityPeriod,  
 VisitingTime: createAbonementCommand.VisitingTime,  
 Price: int32(createAbonementCommand.Price),  
 ServicesIds: createAbonementCommand.Services,  
 }  
  
 createAbonementRequestAbonementDataForCreate := &abonementGRPC.CreateAbonementRequest\_AbonementDataForCreate{  
 AbonementDataForCreate: abonementDataForCreate,  
 }  
  
 createAbonementRequest := &abonementGRPC.CreateAbonementRequest{  
 Payload: createAbonementRequestAbonementDataForCreate,  
 }  
 err = stream.Send(createAbonementRequest)  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": err})

Продолжение листинга 3.13

return  
 }  
 form, err := c.MultipartForm()  
 if err != nil {  
 c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid form data"})  
 return  
 }  
 photo := form.File["photo"]  
 if photo != nil && len(photo) > 0 {  
 buffer := make([]byte, 1024\*1024)  
 file, err := photo[0].Open()  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 for {  
 n, err := file.Read(buffer)  
 if err == io.EOF {  
 break  
 }  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 err = stream.Send(&abonementGRPC.CreateAbonementRequest{  
 Payload: &abonementGRPC.CreateAbonementRequest\_AbonementPhoto{  
 AbonementPhoto: buffer[:n],  
 },  
 },  
 )  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 retur}}}  
 res, err := stream.CloseAndRecv()  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return}  
 c.JSON(http.StatusOK, gin.H{"abonement": res.GetAbonementWithServices()})}

Листинг 3.13 – Реализация метода CreateAbonement

Он принимает HTTP-запрос с названием, периодом валидности, временем посещения, ценой, услугами. Маппит данные на структуры. Валидирует данные. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод Send у сервиса Abonement, который руководит добавлением абонемента.

В результате выполнения контроллера возвращаются добавленный абонемент.

Логика обновления абонемента реализована в контроллере UpdateAbonement, представленном на листинге 3.14.

func (h \*Handler) UpdateAbonement(c \*gin.Context) {  
  
 updateAbonementCommandAny, exists := c.Get("UpdateAbonementCommand")  
 if !exists {  
 logger.ErrorLogger.Printf("Cant find UpdateAbonementCommand in context")  
 return  
 }  
  
 updateAbonementCommand, ok := updateAbonementCommandAny.(\*dtos.UpdateAbonementCommand)  
 if !ok {  
 logger.ErrorLogger.Printf("UpdateAbonementCommand has an invalid type")  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": "UpdateAbonementCommand has an invalid type"})  
 return  
 }  
  
 stream, err := (\*h.abonementClient).UpdateAbonement(context.Background())  
 if err != nil {  
 fmt.Printf("failed to stat file: %v\n", err)  
 }  
  
 abonementDataForUpdate := &abonementGRPC.AbonementDataForUpdate{  
 Id: updateAbonementCommand.Id,  
 Title: updateAbonementCommand.Title,  
 Validity: updateAbonementCommand.ValidityPeriod,  
 VisitingTime: updateAbonementCommand.VisitingTime,  
 Price: int32(updateAbonementCommand.Price),  
 ServicesIds: updateAbonementCommand.Services,  
 }  
  
 updateAbonementRequestAbonementDataForUpdate := &abonementGRPC.UpdateAbonementRequest\_AbonementDataForUpdate{  
 AbonementDataForUpdate: abonementDataForUpdate,  
 }  
 updateAbonementRequest := &abonementGRPC.UpdateAbonementRequest{  
 Payload: updateAbonementRequestAbonementDataForUpdate,

Продолжение листинга 3.14

}  
 err = stream.Send(updateAbonementRequest)  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 form, err := c.MultipartForm()  
 if err != nil {  
 c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid form data"})  
 return  
 }  
 photo := form.File["photo"]  
 if photo != nil && len(photo) > 0 {  
 buffer := make([]byte, 1024\*1024)  
 file, err := photo[0].Open()  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 for {  
 n, err := file.Read(buffer)  
 if err == io.EOF {  
 break  
 }  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 err = stream.Send(&abonementGRPC.UpdateAbonementRequest{  
 Payload: &abonementGRPC.UpdateAbonementRequest\_AbonementPhoto{  
 AbonementPhoto: buffer[:n],  
 },},)  
 if err != nil {logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
return}}}res, err := stream.CloseAndRecv()  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
return}c.JSON(http.StatusOK, gin.H{  
"abonement": res.GetAbonementWithServices(),})}

Листинг 3.14 – Реализация метода UpdateAbonement

Он принимает HTTP-запрос с названием, периодом валидности, временем посещения, ценой, услугами. Маппит данные на структуры. Валидирует данные. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод Send у сервиса Abonement, который руководит обновлением абонемента.

В результате выполнения контроллера возвращаются добавленный абонемент.

Логика обновления абонемента реализована в контроллере DeleteAbonement, представленном на листинге 3.15.

func (h \*Handler) DeleteAbonement(c \*gin.Context) {  
  
 id := c.Param("id")  
  
 convertedId, err := uuid.Parse(id)  
 if err != nil {  
  
 c.AbortWithStatusJSON(http.*StatusBadRequest*, gin.H{"error": fmt.Errorf("invalid id format")})  
 return  
 }  
  
 \_ = convertedId  
  
 deleteAbonementByIdRequest := &abonementGRPC.DeleteAbonementByIdRequest{  
 Id: id,  
 }  
  
 deletedAbonementRes, err := (\*h.abonementClient).DeleteAbonementById(context.TODO(), deleteAbonementByIdRequest)  
 if err != nil {  
 return  
 }  
  
 c.JSON(http.*StatusOK*, gin.H{  
 "abonement": deletedAbonementRes.GetAbonementObject(),  
 })  
}

Листинг 3.15 – Реализация метода DeleteAbonement

Он принимает HTTP-запрос с названием, периодом валидности, временем посещения, ценой, услугами. Маппит данные на структуры. Валидирует данные. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод DeleteAbonementById у сервиса Abonement, который руководит удалением абонемента.

В результате выполнения контроллера возвращаются удаленный абонемент.

## **Редактирования тренеров**

Администратор системы имеет возможность редактирования тренеров – добавление, обновление, удаление. Логика добавления тренера реализована в контроллере CreateCoach, представленном на листинге 3.16.

func (h \*Handler) CreateCoach(c \*gin.Context) {  
  
 createCoachCommandAny, exists := c.Get("CreateCoachCommand")  
 if !exists {  
 logger.ErrorLogger.Printf("Cant find CreateCoachCommand in context")  
 return  
 }  
  
 createCoachCommand, ok := createCoachCommandAny.(\*dtos.CreateCoachCommand)  
 if !ok {  
 logger.ErrorLogger.Printf("CreateCoachCommand has an invalid type")  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": "CreateCoachCommand has an invalid type"})  
 return  
 }  
  
 stream, err := (\*h.coachClient).CreateCoach(context.Background())  
 if err != nil {  
 fmt.Printf("failed to stat file: %v\n", err)  
 }  
  
 coachDataForCreate := &coachGRPC.CoachDataForCreate{  
 Name: createCoachCommand.Name,  
 Description: createCoachCommand.Description,  
 CoachServiceIds: createCoachCommand.Services,  
 }  
  
 createCoachRequestCoachDataForCreate := &coachGRPC.CreateCoachRequest\_CoachDataForCreate{  
 CoachDataForCreate: coachDataForCreate,  
 }  
  
 createCoachRequest := &coachGRPC.CreateCoachRequest{  
 Payload: createCoachRequestCoachDataForCreate,  
 }  
  
 err = stream.Send(createCoachRequest)  
 if err != nil {  
 return  
 }  
  
 form, err := c.MultipartForm()  
 if err != nil {

Продолжение листинга 3.17

c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid form data"})  
 return  
 }  
  
 photo := form.File["photo"]  
  
 if photo != nil && len(photo) > 0 {  
 buffer := make([]byte, 1024\*1024)  
 file, err := photo[0].Open()  
 if err != nil {  
 return  
 }  
  
 for {  
 n, err := file.Read(buffer)  
 if err == io.EOF {  
 break  
 }  
 if err != nil {  
 return  
 }  
  
 err = stream.Send(&coachGRPC.CreateCoachRequest{  
 Payload: &coachGRPC.CreateCoachRequest\_CoachPhoto{  
 CoachPhoto: buffer[:n],  
 },  
 },  
 )  
 if err != nil {  
 return  
 }  
 }  
 }  
  
 res, err := stream.CloseAndRecv()  
 if err != nil {  
 return  
 }  
  
 c.JSON(http.StatusOK, gin.H{  
 "coach": res.GetCoachWithServices(),  
 })  
}

Листинг 3.17 – Реализация метода CreateCoach

Он принимает HTTP-запрос. Достает из контекста запроса структуру с данными для создания тренера. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод Send у сервиса Coach, который руководит добавлением тренера.

В результате выполнения контроллера возвращаются добавленный тренер.

Логика обновления тренера реализована в контроллере UpdateCoach, представленном на листинге 3.18.

func (h \*Handler) UpdateCoach(c \*gin.Context) {  
 updateCoachCommandAny, exists := c.Get("UpdateCoachCommand")  
 if !exists {  
 logger.ErrorLogger.Printf("Cant find UpdateCoachCommand in context")  
 return  
 }  
  
 updateCoachCommand, ok := updateCoachCommandAny.(\*dtos.UpdateCoachCommand)  
 if !ok {  
 logger.ErrorLogger.Printf("UpdateCoachCommand has an invalid type")  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": "UpdateCoachCommand has an invalid type"})  
 return  
 }  
  
 stream, err := (\*h.coachClient).UpdateCoach(context.Background())  
 if err != nil {  
 fmt.Printf("failed to stat file: %v\n", err)  
 }  
  
 coachDataForUpdate := &coachGRPC.CoachDataForUpdate{  
 Id: updateCoachCommand.Id,  
 Name: updateCoachCommand.Name,  
 Description: updateCoachCommand.Description,  
 CoachServiceIds: updateCoachCommand.Services,  
 }  
  
 updateCoachRequestCoachDataForUpdate := &coachGRPC.UpdateCoachRequest\_CoachDataForUpdate{  
 CoachDataForUpdate: coachDataForUpdate,  
 }  
  
 updateCoachRequest := &coachGRPC.UpdateCoachRequest{  
 Payload: updateCoachRequestCoachDataForUpdate,  
 }  
  
 err = stream.Send(updateCoachRequest)  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.*StatusInternalServerError*, gin.H{"error": err})  
 return  
 }  
 form, err := c.MultipartForm()

Продолжение листинга 3.18

if err != nil {  
 c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid form data"})  
 return  
 }  
  
 photo := form.File["photo"]  
 if photo != nil && len(photo) > 0 {  
 buffer := make([]byte, 1024\*1024)  
 file, err := photo[0].Open()  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 for {  
 n, err := file.Read(buffer)  
 if err == io.EOF {  
 break  
 }  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 err = stream.Send(&coachGRPC.UpdateCoachRequest{  
 Payload: &coachGRPC.UpdateCoachRequest\_CoachPhoto{  
 CoachPhoto: buffer[:n],  
 },  
 },  
 )  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 }  
 }  
 res, err := stream.CloseAndRecv()  
 if err != nil {  
 logger.ErrorLogger.Printf(err.Error())  
 c.JSON(http.StatusInternalServerError, gin.H{"error": err})  
 return  
 }  
 c.JSON(http.StatusOK, gin.H{  
"coach": res.GetCoachWithServices(),})}

Листинг 3.18 – Реализация метода UpdateCoach

Он принимает HTTP-запрос. Достает из контекста запроса структуру с данными для обновления тренера. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод Send у сервиса Coach, который руководит обновлением тренера.

В результате выполнения контроллера возвращаются обновленный тренер с услугами.

Логика удаления тренера реализована в контроллере DeleteCoach, представленном на листинге 3.19

func (h \*Handler) DeleteCoach(c \*gin.Context) {  
 id := c.Param("id")  
  
 convertedId, err := uuid.Parse(id)  
 if err != nil {  
  
 c.AbortWithStatusJSON(http.*StatusBadRequest*, gin.H{"error": fmt.Errorf("invalid id format")})  
 return  
 }  
  
 \_ = convertedId  
  
 deleteCoachByIdRequest := &coachGRPC.DeleteCoachByIdRequest{  
 Id: id,  
 }  
  
 deletedCoachRes, err := (\*h.coachClient).DeleteCoachById(context.TODO(), deleteCoachByIdRequest)  
 if err != nil {  
 return  
 }  
  
 c.JSON(http.*StatusOK*, gin.H{  
 "coach": deletedCoachRes.GetCoachObject(),  
 })  
}

Листинг 3.19 – Реализация метода DeleteCoach

Он принимает HTTP-запрос. Достает из контекста запроса id тренера для удаления. Создает запросы в формате Protobuf для дальнейшей передачи домен-сервисам. Вызывает метод DeleteCoachById у сервиса Coach, который руководит обновлением тренера.

В результате выполнения контроллера возвращаются удаленный тренер с услугами.

## **Структура клиентской части**

Клиентская часть приложения реализована с использованием компонентного подхода. Основная логика и элементы пользовательского интерфейса размещены в директории src. Директории представлены в таблице 3.14.

Таблица 3.14 – Основные директории проекта в папке src и их назначение

|  |  |
| --- | --- |
| Директория | Назначение |
| components | Включает в себя переиспользуемые React-компоненты, предназначенные для создания элементов пользовательского интерфейса web-приложения. |
| context | Содержит файлы, предназначенные для управления токенами доступа пользователя, и глобавльными состояниями приложения. |
| services | Включает модуль для логики работы токенов. |
| states | содержит файлы, связанные с управлением состоянием приложения с использованием Redux Toolkit. |
| images | Хранит изображения |
| utils | содержит файлы кастомных ошибок |

**Таблица соответствия маршрутов и компонентов страниц представлена в таблице 3.15.**

Таблица 3.15 – Маршруты и компоненты страниц

|  |  |  |  |
| --- | --- | --- | --- |
| Компонент страницы | Маршрут | Роль | Назначение компонента |
| MainNavHome | /main/home | Администратор, клиент | Основная страница приложения, которая служит отправной точкой для навигации по другим разделам и предоставляет общую информацию о web-приложении. |
| MainNavAbonnements | /main/abonnements | Администратор, клиент | Отображение информации о абонементах, с элементами для поиска сортировки, фильтрации. |

Продолжение таблицы 3.15

|  |  |  |  |
| --- | --- | --- | --- |
| MainNavProfile | /main/profile | Клиент | Отображение профиля клиента с историей покупок. |
| MainNavCoaches | /main/coaches | Администратор, клиента | Отображения краткой информации о тренера. |
| MainNavCoachDetailCard | /main/coaches/details | Администратор, клиента | Отображения полной информации о тренере, списка его отзывов, а так же модального окна для написаня отзыва(доступного только клиенту). |
| MainNavAdminPanel | / adminPanel | Администратор | Отображение модальных окон для редактирования абонементов, тренеров, клиентов. |
| SignIn | /signin | Гость | Отображение формы входа. |
| SignUp | /signup | Гость | Отображение формы регистрации. |

Помимо маршрутов и страниц, приложение включает множество компонентов, которые обеспечивают функциональность и удобство использования клиентской части.

В таблице 3.16 представлено описание всех остальных компонентов приложения и их назначение.

Таблица 3.16 – Описание компонентов

|  |  |
| --- | --- |
| Компонент | Назначение |
| mainAbonnements | Используется для отображения списка карточек абонементов. Отображает элементы фильтрации, сортировки, поиска. |

Продолжение таблицы 3.16

|  |  |
| --- | --- |
| abonementCard | Используется для отображения карточки абонемента. |
| AbonnementsModal | Используется для отображения модального окна для редактирования абонементов. |
| ClientsModal | Используется для отображения модального окна для редактирования клиентов. |
| CoachesModal | Используется для отображения модального окна для редактирования тренеров. |
| mainCoaches | Используется для отображения списка карточек тренеров. |
| coachCard | Используется для отображения карточки тренера с краткой информацией. |
| coachDetailCard | Используется для отображения карточки тренера с полной информацией, а также с комментариями и модальным окнов для их написания. |
| mainNav | Используется для отображении элемента с кнопками для навигации по всем страницам. |

Эти компоненты обеспечивают все необходимые функции для создания полноценного приложения для фитнес-центра, включая отображение абонементов, навигацию, управление профилем, администрирование.

## **Выводы по разделу**

1. Таким образом, было реализовано web-приложение «FitLab» для фитнес-центра со следующими особенностями:

1. Использован язык программирования Golang с применением фреймворков GIN и GRPC. Для хранения данных использовалась реляционная СУБД PostgreSQL. Для упрощения взаимодействия с ней применялась библиотека SQLX, которая автоматизировала маппинг записей на структуры go и облегчила выполнение операция с данными.
2. Применены сторонние сервисы, которые помогут расширить функциональность web-приложения, включая оплату и облачное хранение данных.

Разработана структура web-приложения, которая базируется на микросервисной архитектуре с использованием современных библиотек для клиентской и серверной части. Реализованы все функции для всех ролей: гостя, клиента и администратора. Общее количество функций – 16.

# **Тестирование web**-**приложения**

## **Функциональное тестирование**

Для проверки корректности работы всех функций разработанного web-приложения было проведено ручное тестирование, описание и итоги которого представлены в таблице 4.1.

Таблица 4.1 – Описание тестирования функций web-приложения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Функция web-приложения | Описание тестирования | Ожидаемый результат | Полученный результат |
| 1 | Регистрация | Отправить POST запрос на адрес /sso/signUp, указав в теле запроса имя пользователя (name) со значением «test», адрес электронной почты (email) со значением «test@gmail.com», пароль (password) со значением «12345678» | В таблице user должна появится запись со следующими значениями для колонок: колонка id с значением «уникальный uuid», колонка email c значением «test@gmail.com», колонка role со значением «client», колонка пароль со значением «хэш от пароля», колонка photo со значением «null», колонка name со значением «test», колонка created\_time со значением приблизительного времени создания, колонка updated\_time со значением приблизительного времени создания.  Сервер должен вернуть ответ с кодом 200, данные пользователя, access token, access token expiration в формате JSON. Установить в куки refresh token и refresh token expiration | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | Аутентификация | Отправить POST запрос на адрес /sso/signIn, указав в теле запроса имя пользователя (name) со значением «Danila», адрес электронной почты (email) со значением «danilakozlyakovksy@gmail.com», пароль (password) со значением «TankiDanik2003» | Сервер должен вернуть ответ с кодом 200, данные пользователя, access token, access token expiration в формате JSON. Установить в куки refresh token и refresh token expiration. | Совпадает с ожидаемым. |
| 3 | Просмотр информации о абонементах | Отправить GET запрос на адрес /abonements, указава в заголовке Authorization значение – Bearer и значение валидного access токена. | Сервер должен вернуть ответ с кодом 200 и список абонементов. | Совпадает с ожидаемым. |
| 4 | Поиска, сортировка, фильтрация абонементов | На ui перейти на вкладку абонементов, нажава на кнопку ABONEMENTS на навигационной ранеле слева. Использовать элементы для сортировки, фильтрации, поиска. | В списке абонементов отображаются нужные карточки абонементов, в правильной последовательности. | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | Просмотр информации о тренерах и отзывов о тренерах | Отправить GET запрос на адрес /coaches, указава в заголовке Authorization значение – Bearer и значение валидного access токена. | Сервер должен вернуть ответ с кодом 200 и список абонементов с их услугами. | Совпадает с ожидаемым. |
| 6 | Выход из системы | Отправить POST запрос на адрес /sso/logout. В куки должен быть установлен refresh token. | Сервер должен вернуть ответ с кодом 200, из куки должен пропасть refreshToken. | Совпадает с ожидаемым. |
| 7 | Оставить отзыв о тренере | Отправить POST запрос на адрес /reviews, указав в теле запроса (UserId  ) со значением существующего id пользователя, (Body  ) со значением валидного тела отзыва, (CoachId  ) со значением существующего id тренера. Указать в заголовке Authorization значение – Bearer и значение валидного access токена для клиента. | В таблице review должна появится запись со следующими значениями для колонок: колонка id с значением «уникальный uuid», колонка body со значением «norm coach», колонка user\_id со значением id пользователя, передаваемого в теле запроса, колонка created\_time со значением приблизительного времени создания, колонка updated\_time со значением приблизительного времени создания. В таблице coach\_review  должна появится запись со следующими значениями для колонок: колонка coach\_id со значением id тренера передаваемого в теле запроса, колонка review\_id со значением id созданного отзыва. Сервер должен вернуть ответ с кодом 200, в теле ответа должен быть объект отзыва и пользователя, который его оставил. | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 8 | Редактирование профиля | Отправить PUT запрос на адрес /users/:id, указава в URL id пользователя, в теле запроса параметры: (name  ) со значением «test\_change», (photo) cо значением фото в бинарном формате. Указать в заголовке Authorization значение – Bearer и значение валидного access токена для клиента. | В таблице user должна обновится запись пользователя с id передаваемым в URL запроса. Должны обновится колонки: name со значением «test\_change», photo со значением URL фото из localstack(s3), колонкой updated\_time со значением приблизительного времени обновления. В localstack(s3) должна появится добавится фото из тела запроса. Сервер должен вернуть ответ с кодом 200, в теле ответа должен быть объект обновленного пользователя. | Совпадает с ожидаемым. |
| 9 | Удаление клиентов | Отправить DELETE запрос на адрес /users/:id, указав в URL id пользователя. Указать в заголовке Authorization значение – Bearer и значение валидного access токена для администратора. | Из таблицы user должна удалиться запись клиента с id переданным в URL запроса. Из localstack(s3) должна удалиться фото удаляемого клиента. Сервер должен вернуть ответ с кодом 200, в теле ответа должен быть объект удаленного клиента. | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 10 | Создание абонемента | Отправить POST запрос на адрес /abonements, в теле запроса с content type = multipart/form-data указать параметры: (title) со значением «test», (validity\_period  ) со значением «1», (visiting\_time  ) со значением «7.00 – 14.00», (price) со значением «100», (services) со значением id абонементов перечисленных через запятую, (photo) со значением фото в бинарном формате. Указать в заголовке Authorization значение – Bearer и значение валидного access токена для админа. | В таблице abonement должна появится запись со следующими значениями для колонок: колонка id с значением «уникальный uuid», колонка title c значением «test», колонка validity со значением «1», колонка visiting\_time c значением «7.00 - 14.00», колонка photo со значением URL фото из localstack(s3), колонка price со значением «100», колонка created\_time со значением приблизительного времени создания, колонка updated\_time со значением приблизительного времени создания. В аккаунте Stripe должны создаться объекты продукта с названием «test» и цены со значением «100». В localstack(s3) должно добавиться фото абонемента. Сервер должен вернуть ответ с кодом 200, данные созданного абонемента. | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | Обновление абонемента | Отправить PUT запрос на адрес /abonements, в теле запроса с content type = multipart/form-data указать парметры: (title) со значением «test\_updated», (validity\_period  ) со значением «3», (visiting\_time  ) со значением «14.00 – 24.00», (price) со значением «200», (services) со значением id абонементов перечисленных через запятую, (photo) со значением фото в бинарном формате. Указать в заголовке Authorization значение – Bearer и значение валидного access токена для админа. | В таблице abonement должна обновится запись со следующими значениями для колонок: колонка id с значением «уникальный uuid», колонка title c значением «test\_updated», колонка validity со значением «3», колонка visiting\_time c значением «14.00 – 24.00», колонка photo со значением обновленного URL фото из localstack(s3), колонка price со значением «200», колонка updated\_time со значением приблизительного времени обновления. В аккаунте Stripe должен создаться объект цены со значением «200» и присвоится к продукту, также должно обновится имя продукта. В localstack(s3) должно добавиться фото абонемента. Сервер должен вернуть ответ с кодом 200, данные обновленного абонемента. | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 11 | Удаление абонемента | Отправить DELETE запрос на адрес /abonements /:id, указава в URL id пользователя. Указать в заголовке Authorization значение – Bearer и значение валидного access токена для администратора. | Из таблицы abonement должна удалиться запись клиента с id переданным в URL запроса. Из localstack(s3) должна удалиться фото удаляемого абонемента. Продукт stripe должен заархивироваться. Сервер должен вернуть ответ с кодом 200, в теле ответа должен быть объект удаленного клиента. | Совпадает с ожидаемым. |
| 12 | Создание тренера | Отправить POST запрос на адрес /coach, в теле запроса с content type = multipart/form-data указать параметры: (name) со значением «test», (description  ) со значением «some description to coach», (services) со значением id услуг перечисленных через запятую, (photo) со значением фото в бинарном формате. Указать в заголовке Authorization значение – Bearer и значение валидного access токена админа. | В таблице coach должна появится запись со следующими значениями для колонок: колонка id с значением «уникальный uuid», колонка name c значением «test», колонка description c значением «some description to coach» колонка photo со значением URL фото из localstack(s3), колонка created\_time со значением приблизительного времени создания, колонка updated\_time со значением приблизительного времени создания. В localstack(s3) должно добавиться фото абонемента. Сервер должен вернуть ответ с кодом 200, данные созданного тренера. | Совпадает с ожидаемым. |

Продолжение таблицы 4.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 13 | Обновление тренера | Отправить PUT запрос на адрес /coach, в теле запроса с content type = multipart/form-data указать парметры: (name) со значением «testUp», (description  ) со значением «some description to coachUp», (services) со значением id услуг перечисленных через запятую, (photo) со значением фото в бинарном формате Указать в заголовке Authorization значение – Bearer и значение валидного access токена для админа. | В таблице coach должна обновиться запись со следующими значениями для колонок: колонка name c значением «testUp», колонка description c значением «some description to coachUp» колонка photo со значением URL фото из localstack(s3), колонка updated\_time со значением приблизительного времени обновления. В localstack(s3) должно обновиться фото абонемента. Сервер должен вернуть ответ с кодом 200, данные обновленного тренера. | Совпадает с ожидаемым. |
| 14 | Удаление абонемента | Отправить DELETE запрос на адрес /coach/:id, указав в URL id тренера. Указать в заголовке Authorization значение – Bearer и значение валидного access токена для администратора. | Из таблицы coach должна удалиться запись тренера с id переданным в URL запроса. Из localstack(s3) должна удалиться фото удаляемого тренера. Сервер должен вернуть ответ с кодом 200, в теле ответа должен быть объект удаленного тренера. | Совпадает с ожидаемым. |

Тестирование оплаты через Stripe и списка своих заказов проводилось в клиентском браузере, чтобы убедиться, что транзакции выполняются корректно и результат соответствует ожиданиям. Процесс тестирования данной функции включает в себя шаги, продемонстрированные в таблице 4.2.

Таблица 4.2. Тестирование оплаты и списка своих заказов через клиентский браузер

|  |  |  |  |
| --- | --- | --- | --- |
| Шаг | Действие | Ожидаемый результат | Фактический результат |
| 1 | Аутентификация: войти в систему с помощью логина и пароля пользователя с ролью «Клиент» | Пользователь успешно аутентифицирован и перенаправлен на главную страницу | Совпадает с ожидаемым |
| 2 | Перейти на страницу абонементов, нажав на кнопку ABONEMENTS на навигационной панели слева. | Карточки абонементов отображаются на странице | Совпадает с ожидаемым |
| 3 | Инициация процесса оплаты: нажать на кнопку «Buy», которая инициирует процесс оплаты Stripe | Сервер генерирует сессию Stripe и отправляет URL со странице оплаты stripe на клиент. Происходит перенаправление на страницу оформления платежа | Совпадает с ожидаемым |
| 4 | **Ввод тестовых данных на странице оплаты**: ввести номер карты «4242 4242 4242 4242», срок действия (любая дата в будущем), любой CVV/CVC, и произвольное имя владельца карты. Нажать кнопку «Оплатить». | Платежная информация отправляется на сервер Stripe, и платеж проходит успешно. Пользователя перенаправляет на главную страницу web-приложения. Он может просмотреть историю своих заказов на странице личного профиля, нажав на кнопку «EDIT PROFILE». На панели администратора в Stripe отображается вся информация о заказе пользователя, включая дату заказа, товары и их стоимость | Совпадает с ожидаемым |

Продолжение таблицы 4.2

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | **Просмотр истории заказов**: после нажатия на кнопку «EDIT PROFILE» пользователь видит список своих заказов. | История заказов отображается на странице профиля | Совпадает с ожидаемым |

Таким образом, были протестированы все ключевые функции web-приложения.

## **Выводы по разделу**

1. Проведено ручное тестирование ключевых функций web-приложения.

2. Корректность работы системы подтверждена соответствием фактических результатов тестирования ожидаемым.

3. Количество позитивных тестов составило 15, покрытие позитивными тестами – 100%(не учитываю ошибочные исходы).

4. Проведено тестирование оплаты Stripe в следующих браузерах:

Microsoft Edge 131.0.2903.112 (64-bit);

# **Руководство пользователя**

## **Руководство пользователя для роли «Гость»**

## **Аутентификация**

При обращении к домену web-приложения первая станица на которую перенаправит пользователя, при условии, что он не авторизован будет страница входа.

Для аутентификации гость должен ввести email и Password от своей учетной записи и нажать на кнопку «SIGN IN».

Cтраница входа представлена на рисунке 5.1.

A screenshot of a login page

AI-generated content may be incorrect.

Рисунок 5.1 – Страница входа

При вводе невалидных данных пользователю будут показаны сообщения со вспомогательной информацией.

Cтраница входа c сообщением об ошибке представлено на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.2 – Страница входа c сообщением об ошибке

После успешного входа пользователя перенаправит на главную страницу web-приложения.

Главная страница представлена на рисунке 5.1.

A person working out on a machine

AI-generated content may be incorrect.

Рисунок 5.3 – Главная страница

## **Регистрация**

Если у пользователя нету аккаунта, то он может его создать прейдя на форму регистрации, нажав на ссылку с надписью «Don't have an account? Sign up» под формой входа. После нажатая пользователя перенаправит на страницу регистрации.

Cтраница регистрации представлена на рисунке 5.1.

Страница регистрации представлена на рисунке 5.1.

A screenshot of a login form

AI-generated content may be incorrect.

Рисунок 5.4 – Страница регистрации

Для того чтобы зарегистрироваться нужно ввести Name, Email, Password.

При вводе невалидных данных пользователю будут показаны сообщения со вспомогательной информацией.

Cтраница регистрации c сообщением об ошибке представлено на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.5 – Страница регистрации c сообщением об ошибке

Если у пользователя уже есть аккаунт, то он может перейти на страницу входа нажав на ссылку с надписью «Already have an account? Sign in» под формой регистрации.

После успешного входа пользователя перенаправит на главную страницу web-приложения.

## **Руководство пользователей с ролями «Клиент», «Администратор»**

## **Просмотра информации о абонементах**

Необходимо перейти на страницу абонементов, нажав на кнопку «ABONEMENTS» в навигационной панеле слева. Появится список абонементов.

Cтраница абонементов представлено на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.6 – Страница абонементов

## **Поиска, сортировка, фильтрация абонементов**

На странице абонементов имеются несколько элементов для поиска по названию и периоду валидности, сортировки по названию и цене, фильтрация по цене. При взаимодействии с ними список абонементов будет изменятся для отображения нужных карточек.

Cтраница абонементов c введенными данными в элементы для поиска, фильтрации и сортировки абонементов представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.7 – Страница абонементов с введенными данными в элементы для поиска, фильтрации и сортировки абонементов

## **Просмотр информации о тренерах и отзывов о тренерах**

Необходимо перейти на страницу тренеров, нажав на кнопку «COACHES» в навигационной панеле слева. Появится список тренеров, с краткой информацией о них.

Cтраница тренеров представлена на рисунке 5.1.

A screenshot of a video game

AI-generated content may be incorrect.

Рисунок 5.8 – Страница тренеров

Для просмотра подробной информации о тренере и его комментариях необходимо нажать на кнопку «DETAILS» на карточке абонемента.

Cтраница подробной информации о тренере представлена на рисунке 5.1.

A screenshot of a web page

AI-generated content may be incorrect.

Рисунок 5.9 – Cтраница подробной информации о тренере

## **Выход из системы**

Необходимо нажать на кнопку «EXIT» в навигационной панеле слева. Вас перенаправит на страницу входа.

## **Руководство пользователя с ролью «Клиент»**

## **Оставить отзыв о тренере**

На странице с подробной информацией о тренере необходимо нажать на кнопку «LEAVE A REVIEW». Откроется модальное окно с формой. Необходимо заполнить поле тела отзыва и нажать на кнопку «SUBMIT» для отправки отзыва, либо нажать кнопку «CLOSE» для скрытия модального окна без отправки отзыва.

Форма добавления отзыва к тренеру представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.10 – Форма добавления отзыва к тренеру

## **Покупка абонементов и просмотр списка своих заказов**

На странице абонементов необходимо нажать на кнопку «BUY» на карточке абонемента. После чего произойдет перенаправление на страницу оплаты Stripe. Где необходимо ввести данные своей карты.

Страница Stripe для оплаты представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.11 – Страница Stripe для оплаты

После подтверждения данных пользователя перенаправит на главную страницу приложения. Для просмотра списка своих заказов необходимо нажать на кнопку «EDIT PROFILE» в навигационной панеле слева.

Страница личного профиля представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.12 – Страница личного профиля

## **Редактирование профиля**

На странице личного профиля возможно поменять фото и имя. Для смены фото необходимо нажать на область картинки и выбрать фото из проводника, для смены имени ввести имя в поле Name. Для подтверждения редактирования необходимо нажать кнопку «SUBMIT».

Страница личного профиля с формой редактирования профиля представлена на рисунке 5.1.

A screenshot of a person

AI-generated content may be incorrect.

Рисунок 5.13 – Страница личного профиля с формой редактирования профиля

## **Руководство пользователя с ролью «Администратор»**

## **Редактирование клиентов**

Необходимо перейти на вкладку панели администратора, нажав на кнопку «ADMIN PANEL». После чего нажать на кнопку «CLIENTS».

Страница панели администратора представлена на рисунке 5.1.

A purple rectangle with rectangles

AI-generated content may be incorrect.

Рисунок 5.14 – Страница панели администратора

Откроется форма удаления клиентов. Необходимо нажать на область клиента. После чего нажать на кнопку «DELETE» для удаления выбранного клиента.

Форма удаления клиентов представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.15 – Форма удаления клиентов

## **Редактирование абонементов**

Необходимо нажать на кнопку «ABONEMENTS» на панеле администратора. Откроется форма редактирования абонементов. Где есть возможность добавить, обновить и удалить абонементы.

Для добавления абонемента нужно убедится, что в данный момент не выбран какой-либо абонемент из списка. Затем необходимо заполнить поля: фото, title, validity period, visiting time, amount, services. Нажать на кнопку «SUBMIT»

Форма добавления абонемента представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.16 – Форма добавления абонемента

Для обновления абонемента необходимо выбрать абонемент из списка нажатием на область с ним заполнить поля для обновления и нажать на кнопку «SUBMIT».

Форма обновления абонемента представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.17 – Форма обновления абонемента

Для удаления абонемента необходимо выбрать нужный из списка и нажать на кнопку «DELETE».

Форма удаления абонемента представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.18 – Форма удаления абонемента

## **Редактирование тренеров**

Необходимо нажать на кнопку «COACHES» на панеле администратора. Откроется форма редактирования тренеров. Где есть возможность добавить, обновить и удалить тренера.

Для добавления тренеров нужно убедится, что в данный момент не выбран какой-либо тренер из списка. Затем необходимо заполнить поля: фото, name, description, services. Нажать на кнопку «SUBMIT».

Форма добавления тренера представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.19 – Форма добавления тренера

Для обновления тренера необходимо выбрать тренера из списка нажатием на область с ним заполнить поля для обновления и нажать на кнопку «SUBMIT».

Форма обновления тренера представлена на рисунке 5.1.

A screenshot of a computer

AI-generated content may be incorrect.

Рисунок 5.20 – Форма обновления тренера

Для удаления тренера необходимо выбрать нужного из списка и нажать на кнопку «DELETE».

Форма удаления тренера представлена на рисунке 5.1.

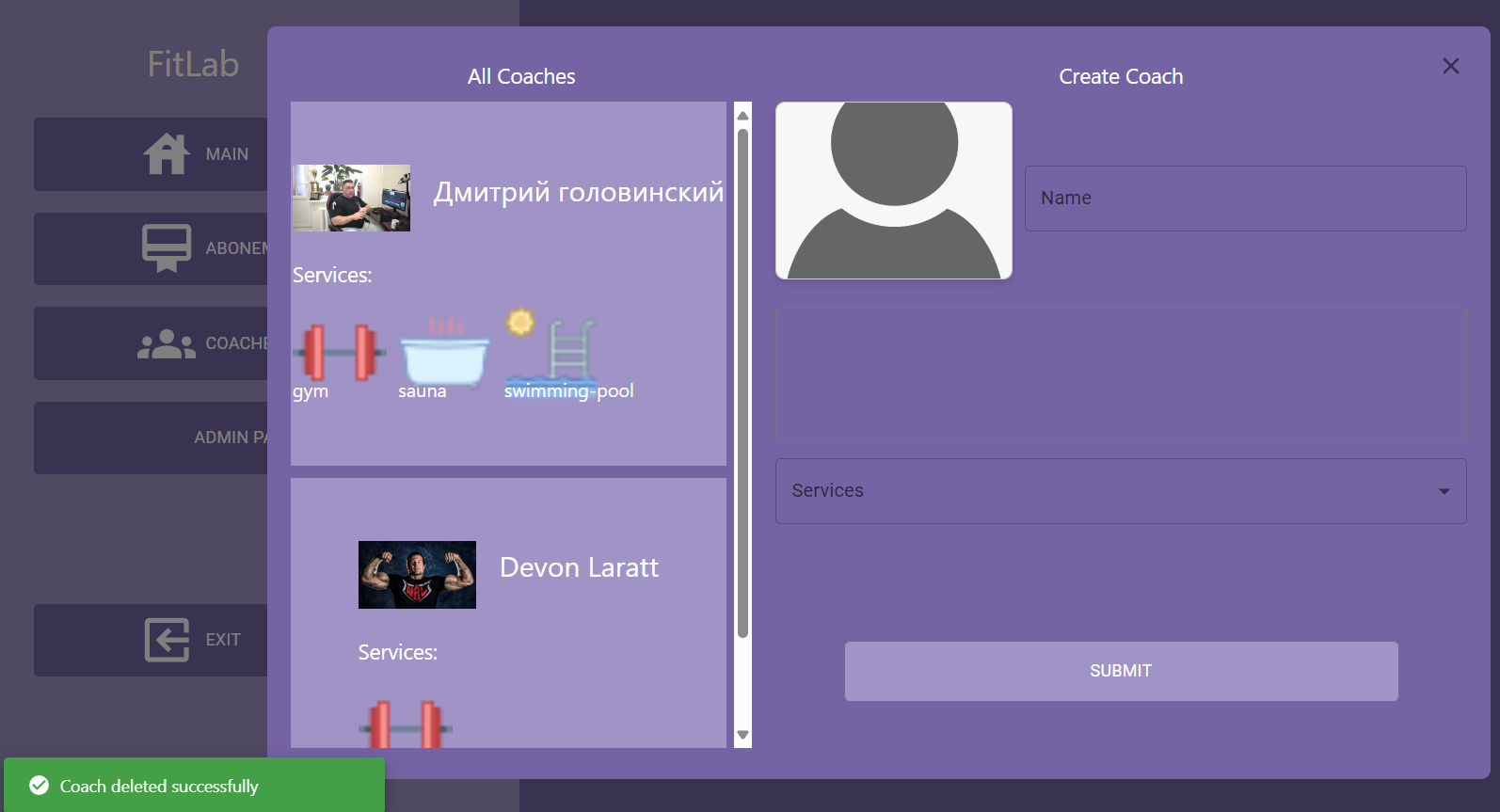


Рисунок 5.21 – Форма удаления тренера

## **Выводы по разделу**

Разработано руководство, описывающее действия пользователей системы: гостя, клиента и администратора, с учетом уникальных функций каждой роли, описанных в диаграмме вариантов использования.

У гостя есть возможность регистрации и аутентификации. У клиента и тренера есть несколько общих возможностей: просмотр информации о абонементах, поиска абонементов, сортировки абонементов, фильтрации абонементов, просмотра информации о тренерах, просмотра отзывов о тренерах, выхода из системы. У клиента есть возможность оставить отзыв о тренере, покупать абонементы, редактировать профиль, просматривать список своих заказов. У Администратора есть возможность редактирования тренеров, абонементов, клиентов.

**Заключение**

В результате работы над проектом было разработано web-приложение «FitLab» для фитнес-центра, которое соответствует заявленным целям и требованиям:

1. Web-приложение поддерживает три ключевые роли: гость, клиент и администратор.
2. Web-приложение использует клиент-серверную архитектуру. Серверная часть реализована на Goalng, клиентская – React.js. Подключены сторонние сервисы: LocalStack(s3) – для хранения изображений в облаке и Stripe – система электронных платежей.
3. Web-приложение включает 16 ключевых функций, охватывающих весь необходимый функционал: оформление заказов и обработка платежей, редактирование абонементов, тренеров, клиентов. Ознакомление с тренерами и возможность оставить к ним отзыв.
4. Для хранения данных была создана база данных, содержащая 10 таблиц: abonement, abonement\_service, coach, coach\_review, coach\_service, order, refresh\_sessions, review, service, user.
5. Общий объем программного кода web-приложения составил 12000 строк авторского кода.
6. Общее количество тестов – 15, покрытие кода тестами 100%(исключая ошибочные исходы).

На основе полученных результатов работы web-приложения можно заключить, что цель проекта достигнута, а все требования технического задания были полностью выполнены.

**Список используемых источников**

1. RFC 2616 HTTP/1.1 [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc2616 – Дата доступа: 20.12.2024.
2. Go programming language [Электронный ресурс]. – Режим доступа: https://go.dev/ – Дата доступа: 20.12.2024.
3. React. [Электронный ресурс]. – Режим доступа: https://react.dev/ – Дата доступа: 20.12.2024.
4. PostgreSQL [Электронный ресурс]. – Режим доступа: https://www.postgresql.org – Дата доступа: 20.12.2024.
5. Фитнес клуб в Минске "Zona531" выгодные цены на абонементы [Электронный ресурс]. – Режим доступа: https://zona531.by/ – Дата доступа: 20.12.2024.
6. Сеть фитнес-центров Lifestyle. Территория умного фитнеса [Электронный ресурс]. – Режим доступа: https://fitness-club.by/ – Дата доступа: 20.12.2024.
7. Oktopus Fitness Club [Электронный ресурс]. – Режим доступа: https://www.oktopus.ge/ – Дата доступа: 20.12.2024.
8. RFC 1738: Uniform Resource Locators (URL) [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc1738 – Дата доступа: 01.10.2024.
9. Stripe | Financial Infrastructure to Grow Your Revenue [Электронный ресурс]. – Режим доступа: https://stripe.com/ – Дата доступа: 01.10.2024.
10. nginx [Электронный ресурс]. – Режим доступа: https://nginx.org/en/ – Дата доступа: 01.10.2024.
11. Protocol Buffers Documentation [Электронный ресурс]. – Режим доступа: https://Protobuf.dev/ – Дата доступа: 01.10.2024.
12. gRPC [Электронный ресурс]. – Режим доступа: https://gRPC.io/ – Дата доступа: 01.10.2024.
13. LocalStack | Run Locally, Deploy Globally [Электронный ресурс]. – Режим доступа: https://www.localstack.cloud/ – Дата доступа: 01.10.2024.
14. Облачное объектное хранилище – Amazon S3 – Amazon Web Services [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/s3/ – Дата доступа: 01.10.2024.
15. Сервисы облачных вычислений – Amazon Web Services (AWS) [Электронный ресурс]. – Режим доступа: https://aws.amazon.com/ru/ – Дата доступа: 01.10.2024.
16. ngrok | API Gateway, IoT Device Gateway, Secure Tunnels for Containers, Apps & APIs [Электронный ресурс]. – Режим доступа: https://ngrok.com/ – Дата доступа: 01.10.2024.
17. RFC 2818: HTTP Over TLS [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc2818 – Дата доступа: 01.10.2024.
18. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc5246 – Дата доступа: 09.01.2025.
19. RFC 9293 Transmission Control Protocol (TCP) [Электронный ресурс]. – Режим доступа: https://www.ietf.org/rfc/rfc9293.html – Дата доступа: 09.01.2025.
20. Docker: Accelerated Container Application Development [Электронный ресурс]. – Режим доступа: https://www.docker.com/ – Дата доступа: 09.01.2025.
21. Gin Web Framework [Электронный ресурс]. – Режим доступа: https://gin-gonic.com/ – Дата доступа: 09.01.2025.
22. What is REST?: REST API Tutorial [Электронный ресурс]. – Режим доступа: https://restfulapi.net/ – Дата доступа: 09.01.2025.
23. SQL Tutorial [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/sql/ – Дата доступа: 20.12.2024.
24. ACID — Википедия [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/ACID – Дата доступа: 09.01.2025.
25. RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc8259 – Дата доступа: 09.01.2025.
26. GitHub - jmoiron/sqlx [Электронный ресурс]. – Режим доступа: https://github.com/jmoiron/sqlx – Дата доступа: 20.12.2024.
27. Sql пакет - database/sql [Электронный ресурс]. – Режим доступа: https://pkg.go.dev/database/sql – Дата доступа: 09.01.2025.
28. GitHub - go-playground/validator [Электронный ресурс]. – Режим доступа: https://github.com/go-playground/validator – Дата доступа: 09.01.2025.
29. GitHub - gin-contrib/cors [Электронный ресурс]. – Режим доступа: https://github.com/gin-contrib/cors – Дата доступа: 09.01.2025.
30. MDN Web Docs. CORS (Cross-Origin Resource Sharing) [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS – Дата доступа: 09.01.2025.
31. Golang-jwt. Go implementation of JSON Web Tokens (JWT) [Электронный ресурс]. – Режим доступа: https://github.com/golang-jwt/jwt – Дата доступа: 09.01.2025.
32. RFC 7519 JSON Web Token (JWT) [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc7519 – Дата доступа: 09.01.2025.
33. GitHub - google/uuid: Go package for UUIDs based on RFC 4122 and DCE 1.1: Authentication and Security Services [Электронный ресурс]. – Режим доступа: https://github.com/google/uuid – Дата доступа: 09.01.2025.
34. RFC 4122: A Universally Unique IDentifier (UUID) URN Namespace [Электронный ресурс]. – Режим доступа: https://www.rfc-editor.org/rfc/rfc4122 – Дата доступа: 09.01.2025.
35. GitHub - joho/godotenv: A Go port of Ruby's dotenv library (Loads environment variables from .env files) [Электронный ресурс]. – Режим доступа: https://github.com/joho/godotenv – Дата доступа: 09.01.2025.
36. GitHub - lib/pq: Pure Go Postgres driver for database/sql [Электронный ресурс]. – Режим доступа: https://github.com/lib/pq – Дата доступа: 09.01.2025.
37. GitHub - stripe/stripe-go: Go library for the Stripe API [Электронный ресурс]. – Режим доступа: https://github.com/stripe/stripe-go – Дата доступа: 09.01.2025.
38. GitHub - swaggo/swag: Automatically generate RESTful API documentation with Swagger 2.0 for Go [Электронный ресурс]. – Режим доступа: https://github.com/swaggo/swag – Дата доступа: 09.01.2025.
39. Swagger: API Documentation & Design Tools for Teams [Электронный ресурс]. – Режим доступа: https://swagger.io/ – Дата доступа: 09.01.2025.
40. GitHub - swaggo/gin-swagger: gin middleware to automatically generate RESTful API documentation with Swagger 2.0 [Электронный ресурс]. – Режим доступа: https://github.com/swaggo/gin-swagger – Дата доступа: 09.01.2025.
41. GitHub - gRPC/gRPC-go: The Go language implementation of gRPC. HTTP/2 based RPC [Электронный ресурс]. – Режим доступа: https://github.com/gRPC/gRPC-go?tab=readme-ov-file – Дата доступа: 09.01.2025.
42. GitHub - golang/Protobuf: Go support for Google's protocol buffers [Электронный ресурс]. – Режим доступа: https://github.com/golang/Protobuf – Дата доступа: 09.01.2025.
43. GitHub - aws/aws-sdk-go-v2: AWS SDK for the Go programming language [Электронный ресурс]. – Режим доступа: https://github.com/aws/aws-sdk-go-v2 – Дата доступа: 09.01.2025.
44. GitHub - golang-migrate/migrate: Database migrations. CLI and Golang library [Электронный ресурс]. – Режим доступа: https://github.com/golang-migrate/migrate – Дата доступа: 09.01.2025.
45. GitHub - golang/crypto: [mirror] Go supplementary cryptography libraries [Электронный ресурс]. – Режим доступа: https://github.com/golang/crypto – Дата доступа: 09.01.2025.
46. @mui/icons-material - npm: This package contains Google's Material Icons converted to Material UI SVG Icon components [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/@mui/icons-material – Дата доступа: 09.01.2025.
47. Material UI: React components that implement Material Design [Электронный ресурс]. – Режим доступа: https://mui.com/material-ui/ – Дата доступа: 09.01.2025.
48. @mui/material - npm: Material UI is an open-source React component library that implements Google's Material Design [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/@mui/material – Дата доступа: 09.01.2025.
49. @mui/joy - npm: Joy UI is an open-source React component library that implements MUI's own design principles [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/@mui/joy – Дата доступа: 09.01.2025.
50. @reduxjs/toolkit - npm: The official, opinionated, batteries-included toolset for efficient Redux development [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/@reduxjs/toolkit – Дата доступа: 09.01.2025.
51. Redux - A JS library for predictable and maintainable global state management [Электронный ресурс]. – Режим доступа: https://redux.js.org/ – Дата доступа: 09.01.2025.
52. @stripe/react-stripe-js - npm: React components for Stripe.js and Elements [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/@stripe/react-stripe-js – Дата доступа: 09.01.2025.
53. Axios [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/axios – Дата доступа: 09.01.2025.
54. React [Электронный ресурс]. – Режим доступа: https://react.dev/ – Дата доступа: 09.01.2025.
55. Источник: react-router-dom [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/react-router-dom – Дата доступа: 09.01.2025.
56. notistack [Электронный ресурс]. – Режим доступа: https://www.npmjs.com/package/notistack – Дата доступа: 09.01.2025.

Приложение А

|  |
| --- |
| CREATE DATABASE "fitness\_center\_db";  \c "fitness\_center\_db";  CREATE TABLE abonement (  id uuid NOT NULL,  title character varying(255),  validity character varying(255),  visiting\_time character varying(255),  photo character varying(255),  price integer,  created\_time timestamp with time zone,  updated\_time timestamp with time zone,  stripe\_price\_id character varying );  ALTER TABLE abonement OWNER TO fitness\_center\_admin;  CREATE TABLE abonement\_service (  abonement\_id uuid NOT NULL,  service\_id uuid NOT NULL );  ALTER TABLE abonement\_service OWNER TO fitness\_center\_admin;  CREATE TABLE coach (  id uuid NOT NULL,  name character varying(255) NOT NULL,  description character varying(255) NOT NULL,  photo character varying(255),  created\_time timestamp with time zone,  updated\_time timestamp with time zone );   ALTER TABLE coach OWNER TO fitness\_center\_admin;  CREATE TABLE coach\_review (  coach\_id uuid NOT NULL,  review\_id uuid NOT NULL );   ALTER TABLE coach\_review OWNER TO fitness\_center\_admin;  CREATE TABLE coach\_service (  coach\_id uuid NOT NULL,  service\_id uuid NOT NULL );   ALTER TABLE coach\_service OWNER TO fitness\_center\_admin;  CREATE TABLE "order" (  id uuid NOT NULL,  abonement\_id uuid,  user\_id uuid,  status character varying NOT NULL,  created\_time timestamp with time zone,  updated\_time timestamp with time zone,  expiration\_time timestamp with time zone );   ALTER TABLE "order" OWNER TO fitness\_center\_admin;  CREATE TABLE refresh\_sessions (  id uuid NOT NULL,  user\_id uuid NOT NULL,  refresh\_token character varying(400) NOT NULL,  finger\_print character varying(32) NOT NULL,  created\_time timestamp with time zone,  updated\_time timestamp with time zone );   ALTER TABLE refresh\_sessions OWNER TO fitness\_center\_admin;  CREATE TABLE review (  id uuid NOT NULL,  body character varying(255) NOT NULL,  created\_time timestamp with time zone,  updated\_time timestamp with time zone,  user\_id uuid );  ALTER TABLE review OWNER TO fitness\_center\_admin;  CREATE TABLE schema\_migrations (  version bigint NOT NULL,  dirty boolean NOT NULL );  ALTER TABLE schema\_migrations OWNER TO fitness\_center\_admin;  CREATE TABLE service (  id uuid NOT NULL,  title character varying(255),  photo character varying(255),  created\_time timestamp with time zone,  updated\_time timestamp with time zone );  ALTER TABLE service OWNER TO fitness\_center\_admin;  CREATE TABLE "user" (  id uuid NOT NULL,  email character varying(255) NOT NULL,  role character varying(255) NOT NULL,  password\_hash character varying(255) NOT NULL,  photo character varying(255),  name character varying(255) NOT NULL,  created\_time timestamp with time zone,  updated\_time timestamp with time zone );  ALTER TABLE "user" OWNER TO fitness\_center\_admin;  ALTER TABLE ONLY abonement  ADD CONSTRAINT abonement\_pkey PRIMARY KEY (id);  ALTER TABLE ONLY abonement\_service  ADD CONSTRAINT abonement\_service\_pkey PRIMARY KEY (abonement\_id, service\_id);  ALTER TABLE ONLY coach  ADD CONSTRAINT coach\_pkey PRIMARY KEY (id);  ALTER TABLE ONLY coach\_review  ADD CONSTRAINT coach\_review\_pkey PRIMARY KEY (coach\_id, review\_id);  ALTER TABLE ONLY coach\_service  ADD CONSTRAINT coach\_service\_pkey PRIMARY KEY (coach\_id, service\_id);  ALTER TABLE ONLY review  ADD CONSTRAINT comment\_pkey PRIMARY KEY (id);  ALTER TABLE ONLY "order"  ADD CONSTRAINT order\_pkey PRIMARY KEY (id);  ALTER TABLE ONLY refresh\_sessions  ADD CONSTRAINT refresh\_sessions\_pkey PRIMARY KEY (id);  ALTER TABLE ONLY schema\_migrations  ADD CONSTRAINT schema\_migrations\_pkey PRIMARY KEY (version);  ALTER TABLE ONLY service  ADD CONSTRAINT service\_pkey PRIMARY KEY (id);  ALTER TABLE ONLY "user"  ADD CONSTRAINT user\_pkey PRIMARY KEY (id);  ALTER TABLE ONLY abonement\_service  ADD CONSTRAINT abonement\_service\_abonement\_id\_fkey FOREIGN KEY (abonement\_id) REFERENCES abonement(id) ON DELETE CASCADE;  ALTER TABLE ONLY abonement\_service  ADD CONSTRAINT abonement\_service\_service\_id\_fkey FOREIGN KEY (service\_id) REFERENCES service(id) ON DELETE CASCADE;  ALTER TABLE ONLY coach\_review  ADD CONSTRAINT coach\_review\_coach\_id\_fkey FOREIGN KEY (coach\_id) REFERENCES coach(id) ON DELETE CASCADE;  ALTER TABLE ONLY coach\_review  ADD CONSTRAINT coach\_review\_review\_id\_fkey FOREIGN KEY (review\_id) REFERENCES review(id) ON DELETE CASCADE;  ALTER TABLE ONLY coach\_service  ADD CONSTRAINT coach\_service\_coach\_id\_fkey FOREIGN KEY (coach\_id) REFERENCES coach(id) ON DELETE CASCADE;  ALTER TABLE ONLY coach\_service  ADD CONSTRAINT coach\_service\_service\_id\_fkey FOREIGN KEY (service\_id) REFERENCES service(id) ON DELETE CASCADE;  ALTER TABLE ONLY review  ADD CONSTRAINT fk\_user\_id FOREIGN KEY (user\_id) REFERENCES "user"(id) ON DELETE CASCADE;  ALTER TABLE ONLY "order"  ADD CONSTRAINT order\_abonement\_id\_fkey FOREIGN KEY (abonement\_id) REFERENCES abonement(id) ON DELETE CASCADE;  ALTER TABLE ONLY "order"  ADD CONSTRAINT order\_user\_id\_fkey FOREIGN KEY (user\_id) REFERENCES "user"(id) ON DELETE CASCADE;  ALTER TABLE ONLY refresh\_sessions  ADD CONSTRAINT refresh\_sessions\_user\_id\_fkey FOREIGN KEY (user\_id) REFERENCES "user"(id) ON DELETE CASCADE;  GRANT ALL ON SCHEMA public TO fitness\_center\_admin; |

Листинг – Скрипт создания базы данных

Приложение Б

A diagram of a computer

AI-generated content may be incorrect.

Листинг – Архитектура web-приложения

Приложение В

|  |
| --- |
| syntax = "proto3";  package fitness\_center.sso;  option go\_package = "FitnessCenter.Protobuf.sso";  import "google/Protobuf/empty.proto"; import "user.proto";  service SSO {  rpc SignUp (SignUpRequest) returns (SignUpResponse);  rpc SignIn (SignInRequest) returns (SignInResponse);  rpc LogOut (LogOutRequest) returns (google.Protobuf.Empty);  rpc Refresh (RefreshRequest) returns (RefreshResponse); }  message SignUpRequest {  string name = 1;  string email = 2;  string password = 3;  string finger\_print = 4; } message SignUpResponse {  string access\_token = 1;  string refresh\_token = 2;  string access\_token\_expiration = 3;  string refresh\_token\_expiration = 4;  fitness\_center.user.UserObject user = 5; }  message SignInRequest {  string email = 1;  string password = 2;  string finger\_print = 3; } message SignInResponse {  string access\_token = 1;  string refresh\_token = 2;  string access\_token\_expiration = 3;  string refresh\_token\_expiration = 4;  fitness\_center.user.UserObject user = 5; }  message LogOutRequest {  string refresh\_token = 1; }  message RefreshRequest {  string finger\_print = 1;  string refresh\_token = 2; } message RefreshResponse {  string access\_token = 1;  string refresh\_token = 2;  string access\_token\_expiration = 3;  string refresh\_token\_expiration = 4;  fitness\_center.user.UserObject user = 5; } |

Листинг – Protobuf для SSO сервиса

syntax = "proto3";  
  
import "google/Protobuf/empty.proto";  
  
package fitness\_center.abonement;  
  
option go\_package = "FitnessCenter.Protobuf.abonement";  
  
import "service.proto";  
  
service Abonement {  
 rpc CreateAbonement (stream CreateAbonementRequest) returns (CreateAbonementResponse);  
 rpc GetAbonementById (GetAbonementByIdRequest) returns (GetAbonementByIdResponse);  
 rpc UpdateAbonement (stream UpdateAbonementRequest) returns (UpdateAbonementResponse);  
 rpc DeleteAbonementById (DeleteAbonementByIdRequest) returns (DeleteAbonementByIdResponse);  
  
 rpc GetAbonements (google.Protobuf.Empty) returns (GetAbonementsResponse);  
 rpc GetAbonementsWithServices (google.Protobuf.Empty) returns (GetAbonementsWithServicesResponse);  
 rpc GetAbonementsByIds (GetAbonementsByIdsRequest) returns (GetAbonementsByIdsResponse);  
}  
  
message AbonementObject {  
 string id = 1;  
 string title = 2;  
 string validity = 3;  
 string visiting\_time = 4;  
 string photo = 5;  
 int32 price = 6;  
 string created\_time = 7;  
 string updated\_time = 8;  
 string stripe\_price\_id = 9;  
}  
message AbonementWithServices{  
 AbonementObject abonement = 1;  
 repeated service.ServiceObject services = 2;  
}  
  
message AbonementDataForCreate {  
 string title = 1;  
 string validity = 2;  
 string visiting\_time = 3;  
 int32 price = 4;  
 repeated string servicesIds = 5;  
}  
message AbonementDataForUpdate {  
 string id = 1;  
 string title = 2;  
 string validity = 3;  
 string visiting\_time = 4;  
 int32 price = 5;  
 repeated string servicesIds = 6;  
}  
  
message CreateAbonementRequest {  
 oneof payload {  
 AbonementDataForCreate abonementDataForCreate = 1;  
 bytes abonementPhoto = 2;  
 }  
}  
message CreateAbonementResponse {  
 AbonementWithServices abonementWithServices = 1;  
}  
  
message GetAbonementByIdRequest {  
 string id = 1;  
}  
message GetAbonementByIdResponse {  
 AbonementObject abonementObject = 1;  
}  
  
message UpdateAbonementRequest {  
 oneof payload{  
 AbonementDataForUpdate abonementDataForUpdate = 1;  
 bytes abonementPhoto = 2;  
 }  
}  
message UpdateAbonementResponse {  
 AbonementWithServices abonementWithServices = 1;  
}  
  
message DeleteAbonementByIdRequest {  
 string id = 1;  
}  
message DeleteAbonementByIdResponse {  
 AbonementObject abonementObject = 1;  
}  
  
message GetAbonementsResponse {  
 repeated AbonementObject AbonementObjects = 1;  
}  
  
message GetAbonementsWithServicesResponse {  
 repeated AbonementWithServices abonementsWithServices = 1;  
}  
  
message GetAbonementsByIdsRequest {  
 repeated string ids = 1;  
}  
message GetAbonementsByIdsResponse {  
 repeated AbonementObject abonementObjects = 1;  
}

Листинг – Protobuf для Abonement сервиса

syntax = "proto3";  
  
import "google/Protobuf/empty.proto";  
  
package fitness\_center.coach;  
  
option go\_package = "FitnessCenter.Protobuf.coach";  
  
import "service.proto";  
import "review.proto";  
import "user.proto";  
  
service Coach {  
 rpc CreateCoach (stream CreateCoachRequest) returns (CreateCoachResponse);  
 rpc GetCoachById (GetCoachByIdRequest) returns (GetCoachByIdResponse);  
 rpc UpdateCoach (stream UpdateCoachRequest) returns (UpdateCoachResponse);  
 rpc DeleteCoachById (DeleteCoachByIdRequest) returns (DeleteCoachByIdResponse);  
  
 rpc GetCoaches (google.Protobuf.Empty) returns (GetCoachesResponse);  
 rpc GetCoachesWithServicesWithReviewsWithUsers (google.Protobuf.Empty) returns (GetCoachesWithServicesWithReviewsWithUsersResponse);  
}  
  
message CoachObject {  
 string id = 1;  
 string name = 2;  
 string description = 3;  
 string photo = 4;  
 string created\_time = 5;  
 string updated\_time = 6;  
}  
message ReviewWithUser {  
 review.ReviewObject reviewObject = 1;  
 user.UserObject userObject = 2;  
}  
message CoachWithServicesWithReviewsWithUsers{  
 CoachObject coach = 1;  
 repeated service.ServiceObject services = 2;  
 repeated ReviewWithUser reviewWithUser = 3;  
}  
message CoachWithServices{  
 CoachObject coach = 1;  
 repeated service.ServiceObject services = 2;  
}  
  
message CoachDataForCreate {  
 string name = 1;  
 string description = 2;  
 repeated string coach\_service\_ids = 3;  
}  
message CoachDataForUpdate {  
 string id = 1;  
 string name = 2;  
 string description = 3;  
 repeated string coach\_service\_ids = 4;  
}  
  
message CreateCoachRequest {  
 oneof payload {  
 CoachDataForCreate coachDataForCreate = 1;  
 bytes coachPhoto = 2;  
 }  
}  
message CreateCoachResponse {  
 CoachWithServices coachWithServices = 1;  
}  
  
message GetCoachByIdRequest {  
 string id = 1;  
}  
message GetCoachByIdResponse {  
 CoachObject coachObject = 1;  
}  
  
message UpdateCoachRequest {  
 oneof payload {  
 CoachDataForUpdate coachDataForUpdate = 1;  
 bytes coachPhoto = 2;  
 }  
}  
message UpdateCoachResponse {  
 CoachWithServices coachWithServices = 1;  
}  
  
message DeleteCoachByIdRequest {  
 string id = 1;  
}  
message DeleteCoachByIdResponse {  
 CoachObject coachObject = 1;  
}  
  
message GetCoachesResponse {  
 repeated CoachObject coachObjects = 1;  
}  
  
message GetCoachesWithServicesWithReviewsWithUsersResponse {  
 repeated CoachWithServicesWithReviewsWithUsers coachWithServicesWithReviewsWithUsers = 1;  
}

Листинг – Protobuf для Coach сервиса

syntax = "proto3";  
  
package fitness\_center.order;  
  
option go\_package = "FitnessCenter.Protobuf.order";  
  
import "abonement.proto";  
import "service.proto";  
  
service Order {  
 rpc CreateOrder (CreateOrderRequest) returns (CreateOrderResponse);  
 rpc GetUserOrders (GetUserOrdersRequest) returns (GetUserOrdersResponse);  
 rpc CreateCheckoutSession (CreateCheckoutSessionRequest) returns (CreateCheckoutSessionResponse);  
}  
  
message OrderObject {  
 string id = 1;  
 string user\_id = 2;  
 string abonement\_id = 3;  
 string status = 4;  
 string created\_time = 5;  
 string updated\_time = 6;  
}  
  
message OrderObjectWithAbonementWithServices {  
 OrderObject orderObject = 1;  
 fitness\_center.abonement.AbonementObject abonementObject = 2;  
 repeated fitness\_center.service.ServiceObject serviceObjects = 3;  
}  
  
message OrderDataForCreate {  
 string user\_id = 1;  
 string abonement\_id = 2;  
}  
  
message CreateOrderRequest {  
 OrderDataForCreate orderDataForCreate = 1;  
}  
message CreateOrderResponse {  
 OrderObject orderObject = 1;  
}  
  
message GetUserOrdersRequest {  
 string user\_id = 1;  
}  
message GetUserOrdersResponse {  
 repeated OrderObjectWithAbonementWithServices orderObjectWithAbonementWithServices = 1;  
}  
  
message CreateCheckoutSessionRequest {  
 string client\_id = 1;  
 string abonement\_id = 2;  
 string stripe\_price\_id = 3;  
}  
message CreateCheckoutSessionResponse {  
 string session\_url = 1;  
}

Листинг – Protobuf для Order сервиса

syntax = "proto3";  
  
package fitness\_center.review;  
  
option go\_package = "FitnessCenter.Protobuf.review";  
  
service Review {  
 rpc CreateCoachReview (CreateCoachReviewRequest) returns (CreateCoachReviewResponse);  
 rpc GetReviewById (GetReviewByIdRequest) returns (GetReviewByIdResponse);  
 rpc UpdateReview (UpdateReviewRequest) returns (UpdateReviewResponse);  
 rpc DeleteReviewById (DeleteReviewByIdRequest) returns (DeleteReviewByIdResponse);  
  
 rpc GetCoachReviews (GetCoachReviewsRequest) returns (GetCoachReviewsResponse);  
 rpc GetCoachesReviews (GetCoachesReviewsRequest) returns (GetCoachesReviewsResponse);  
}  
  
message ReviewObject {  
 string id = 1;  
 string user\_id = 2;  
 string body = 3;  
 string created\_time = 4;  
 string updated\_time = 5;  
}  
message CoachIdWithReviewObject {  
 string coach\_id = 1;  
 repeated ReviewObject reviewObjects = 2;  
}  
  
message CoachReviewDataForCreate {  
 string user\_id = 1;  
 string body = 2;  
 string coach\_id = 3;  
}  
message ReviewDataForUpdate {  
 string id = 1;  
 string body = 2;  
}  
  
message CreateCoachReviewRequest {  
 CoachReviewDataForCreate reviewDataForCreate = 1;  
}  
message CreateCoachReviewResponse {  
 ReviewObject reviewObject = 1;  
}  
  
message GetReviewByIdRequest {  
 string id = 1;  
}  
message GetReviewByIdResponse {  
 ReviewObject reviewObject = 1;  
}  
  
message UpdateReviewRequest {  
 ReviewDataForUpdate reviewDataForUpdate = 1;  
}  
message UpdateReviewResponse {  
 ReviewObject reviewObject = 1;  
}  
  
message DeleteReviewByIdRequest {  
 string id = 1;  
}  
message DeleteReviewByIdResponse {  
 ReviewObject reviewObject = 1;  
}  
  
message GetCoachReviewsRequest {  
 string coach\_id = 1;  
}  
message GetCoachReviewsResponse {  
 repeated ReviewObject reviewObjects = 1;  
}  
  
message GetCoachesReviewsRequest {  
 repeated string coaches\_ids = 1;  
}  
message GetCoachesReviewsResponse {  
 repeated CoachIdWithReviewObject coachIdWithReviewObject = 1;  
}

Листинг – Protobuf для Review сервиса

syntax = "proto3";  
  
import "google/Protobuf/empty.proto";  
  
package fitness\_center.service;  
  
option go\_package = "FitnessCenter.Protobuf.service";  
  
service Service {  
 rpc CreateService (stream CreateServiceRequest) returns (CreateServiceResponse);  
 rpc GetServiceById (GetServiceByIdRequest) returns (GetServiceByIdResponse);  
 rpc UpdateService (stream UpdateServiceRequest) returns (UpdateServiceResponse);  
 rpc DeleteServiceById (DeleteServiceByIdRequest) returns (DeleteServiceByIdResponse);  
  
 rpc GetServices (google.Protobuf.Empty) returns (GetServicesResponse);  
 rpc CreateCoachServices (CreateCoachServicesRequest) returns (CreateCoachServicesResponse);  
 rpc CreateAbonementServices (CreateAbonementServicesRequest) returns (CreateAbonementServicesResponse);  
 rpc UpdateAbonementServices (UpdateAbonementServicesRequest) returns (UpdateAbonementServicesResponse);  
 rpc UpdateCoachServices (UpdateCoachServicesRequest) returns (UpdateCoachServicesResponse);  
 rpc GetAbonementsServices (GetAbonementsServicesRequest) returns (GetAbonementsServicesResponse);  
 rpc GetCoachesServices (GetCoachesServicesRequest) returns (GetCoachesServicesResponse);  
}  
  
message ServiceObject {  
 string id = 1;  
 string title = 2;  
 string photo = 3;  
 string created\_time = 4;  
 string updated\_time = 5;  
}  
message AbonementIdWithServices{  
 string abonementId = 1;  
 repeated ServiceObject serviceObjects = 2;  
}  
message CoachIdWithServices{  
 string coachId = 1;  
 repeated ServiceObject serviceObjects = 2;  
}  
  
message ServiceDataForCreate {  
 string title = 1;  
}  
message ServiceDataForUpdate {  
 string id = 1;  
 string title = 2;  
}  
  
message CoachService {  
 string coachId = 1;  
 repeated string serviceId = 2;  
}  
message AbonementService {  
 string abonementId = 1;  
 repeated string serviceId = 2;  
}  
  
message CreateServiceRequest {  
 oneof payload {  
 ServiceDataForCreate serviceDataForCreate = 1;  
 bytes servicePhoto = 2;  
 }  
}  
message CreateServiceResponse {  
 ServiceObject serviceObject = 1;  
}  
  
message GetServiceByIdRequest {  
 string id = 1;  
}  
message GetServiceByIdResponse {  
 ServiceObject serviceObject = 1;  
}  
  
message UpdateServiceRequest {  
 oneof payload{  
 ServiceDataForUpdate serviceDataForUpdate = 1;  
 bytes servicePhoto = 2;  
 }  
}  
message UpdateServiceResponse {  
 ServiceObject serviceObject = 1;  
}  
  
message DeleteServiceByIdRequest {  
 string id = 1;  
}  
message DeleteServiceByIdResponse {  
 ServiceObject serviceObject = 1;  
}  
  
message GetServicesResponse {  
 repeated ServiceObject serviceObject = 1;  
}  
  
message CreateCoachServicesRequest {  
 CoachService coachService = 1;  
}  
message CreateCoachServicesResponse {  
 CoachService coachService = 1;  
}  
  
message CreateAbonementServicesRequest {  
 AbonementService abonementService = 1;  
}  
message CreateAbonementServicesResponse {  
 AbonementService abonementService = 1;  
}  
  
message UpdateAbonementServicesRequest {  
 AbonementService abonementService = 1;  
}  
message UpdateAbonementServicesResponse {  
 AbonementService abonementService = 1;  
}  
  
message UpdateCoachServicesRequest {  
 CoachService coachService = 1;  
}  
message UpdateCoachServicesResponse {  
 CoachService coachService = 1;  
}  
  
message GetAbonementsServicesRequest {  
 repeated string abonementIds = 1;  
}  
message GetAbonementsServicesResponse {  
 repeated AbonementIdWithServices abonementIdsWithServices = 1;  
}  
  
message GetCoachesServicesRequest {  
 repeated string coachIds = 1;  
}  
message GetCoachesServicesResponse {  
 repeated CoachIdWithServices coachIdsWithServices = 1;  
}

Листинг – Protobuf для Service сервиса

syntax = "proto3";  
  
package fitness\_center.user;  
  
option go\_package = "FitnessCenter.Protobuf.user";  
  
import "google/Protobuf/empty.proto";  
  
service User {  
 rpc CreateUser (stream CreateUserRequest) returns (CreateUserResponse);  
 rpc GetUserById (GetUserByIdRequest) returns (GetUserByIdResponse);  
 rpc UpdateUser (stream UpdateUserRequest) returns (UpdateUserResponse);  
 rpc DeleteUserById (DeleteUserByIdRequest) returns (DeleteUserByIdResponse);  
  
 rpc GetUserByEmail (GetUserByEmailRequest) returns (GetUserByEmailResponse);  
 rpc CheckPassword (CheckPasswordRequest) returns (google.Protobuf.Empty);  
 rpc GetUsersByIds (GetUsersByIdsRequest) returns (GetUsersByIdsResponse);  
 rpc GetClients (google.Protobuf.Empty) returns (GetClientsResponse);  
}  
  
message UserObject {  
 string id = 1;  
 string email = 2;  
 string role = 3;  
 string photo = 4;  
 string name = 5;  
 string created\_time = 6;  
 string updated\_time = 7;  
}  
  
message UserDataForCreate {  
 string email = 1;  
 string role = 2;  
 string password = 3;  
 string name = 4;  
}  
message UserDataForUpdate {  
 string id = 1;  
 string email = 2;  
 string role = 3;  
 string name = 4;  
}  
  
message CreateUserRequest {  
 oneof payload {  
 UserDataForCreate userDataForCreate = 1;  
 bytes userPhoto = 2;  
 }  
}  
message CreateUserResponse {  
 UserObject userObject = 1;  
}  
  
message GetUserByIdRequest {  
 string id = 1;  
}  
message GetUserByIdResponse {  
 UserObject userObject = 1;  
}  
  
message UpdateUserRequest {  
 oneof payload{  
 UserDataForUpdate userDataForUpdate = 1;  
 bytes userPhoto = 2;  
 }  
}  
message UpdateUserResponse {  
 UserObject userObject = 1;  
}  
  
message DeleteUserByIdRequest {  
 string id = 1;  
}  
message DeleteUserByIdResponse {  
 UserObject userObject = 1;  
}  
  
message GetUserByEmailRequest {  
 string email = 1;  
}  
message GetUserByEmailResponse {  
 UserObject userObject = 1;  
}  
  
message CheckPasswordRequest {  
 string userId = 1;  
 string password = 2;  
}  
  
message GetUsersByIdsRequest {  
 repeated string usersIds = 1;  
}  
message GetUsersByIdsResponse {  
 repeated UserObject usersObjects = 1;  
}  
  
message GetClientsResponse {  
 repeated UserObject userObjects = 1;  
}

Листинг – Protobuf для User сервиса

Приложение Г

func (o OrdergRPC) CreateCheckoutSession(ctx context.Context, request \*orderProtobuf.CreateCheckoutSessionRequest) (\*orderProtobuf.CreateCheckoutSessionResponse, error) {

    stripe.Key = o.StripeKey

    domain := o.clientDomain

    params := &stripe.CheckoutSessionParams{

        LineItems: []\*stripe.CheckoutSessionLineItemParams{

            &stripe.CheckoutSessionLineItemParams{

                Price:    stripe.String(request.StripePriceId),

                Quantity: stripe.Int64(1),

            },

        },

        Mode:       stripe.String(string(stripe.CheckoutSessionModePayment)),

        SuccessURL: stripe.String(domain),

        CancelURL:  stripe.String(domain),

        Metadata: map[string]string{

            "client\_id":    request.ClientId,

            "abonement\_id": request.AbonementId,

        },

    }

    s, err := session.New(params)

    if err != nil {

        log.Printf("session.New: %v", err)

    }

    createCheckoutSessionResponse := &orderProtobuf.CreateCheckoutSessionResponse{

        SessionUrl: s.ID,

    }

    return createCheckoutSessionResponse, nil

}

Приложение Д

func (h \*Handler) HandleCheckoutSessionCompleted(c \*gin.Context) {

    payload, err := ioutil.ReadAll(c.Request.Body)

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{"error": "Failed to read body"})

        return

    }

    event, err := webhook.ConstructEvent(payload, c.Request.Header.Get("Stripe-Signature"), endpointSecret)

    if err != nil {

        log.Printf("Webhook signature verification failed: %v", err)

        c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid signature"})

        return

    }

    var clientId string

    var abonementId string

    switch event.Type {

    case "checkout.session.completed":

        var session stripe.CheckoutSession

        if err := json.Unmarshal(event.Data.Raw, &session); err != nil {

            log.Printf("Failed to parse session: %v", err)

            c.JSON(http.StatusBadRequest, gin.H{"error": "Failed to parse event"})

            return

        }

        clientId = session.Metadata["client\_id"]

        abonementId = session.Metadata["abonement\_id"]

    default:

        log.Printf("Unhandled event type: %s", event.Type)

    }

    createOrderRequest := &orderGRPC.CreateOrderRequest{

        OrderDataForCreate: &orderGRPC.OrderDataForCreate{

            UserId:      clientId,

            AbonementId: abonementId,

        },

    }

    order, err := (\*h.orderClient).CreateOrder(context.TODO(), createOrderRequest)

    if err != nil {

        c.JSON(http.StatusInternalServerError, gin.H{

            "error": err,

        })

        return

    }

    c.JSON(http.StatusOK, gin.H{

        "order": order.GetOrderObject(),

    })

}