**Содержание**

[Введение 3](#_Toc186063232)

[1 Постановка задачи и обзор аналогичных решений 4](#_Toc186063233)

[1.1 Постановка задачи 4](#_Toc186063234)

[1.2 Обзор аналогичных решений 5](#_Toc186063235)

[**1.2.1 x fit** 5](#_Toc186063236)

[1.3 Выводы 6](#_Toc186063237)

[2 Проектирование web-приложения 8](#_Toc186063238)

[2.1 Функциональность web-приложения 8](#_Toc186063239)

[2.2 Структура базы данных 10](#_Toc186063240)

[2.3 Архитектура web-приложения 19](#_Toc186063241)

[2.4 Выводы 20](#_Toc186063242)

[3 Реализация web-приложения 22](#_Toc186063243)

[3.1 Обоснование выбора программной платформы 22](#_Toc186063244)

[3.2 Реализация web-приложения 22](#_Toc186063245)

[3.3 Реализация базы данных 26](#_Toc186063246)

[3.4 Выводы 26](#_Toc186063247)

[4 Тестирование web-приложения 27](#_Toc186063248)

[4.1 Функциональное тестирование 27](#_Toc186063249)

[4.2 Нагрузочное тестирование 31](#_Toc186063250)

[5 Руководство программиста 32](#_Toc186063251)

[Заключение 33](#_Toc186063252)

[Список используемых источников 34](#_Toc186063253)

[Приложение А 35](#_Toc186063254)

[Приложение Б 36](#_Toc186063255)

[Приложение В 37](#_Toc186063256)

Введение

В настоящее время забота о здоровье и фитнесе стала одной из наиболее актуальных тем. Многие люди желают улучшить свое физическое состояние и поддерживать здоровье в хорошей форме, поэтому спрос на услуги фитнес-центров и спортивных залов неуклонно растет.

В современном быстротемповом мире, где все жизненные процессы автоматизируются, покупка абонементов в фитнес-центр становиться очень приятным предложением, так как не нужно покидать свое жилье, чтобы ознакомиться с предлагаемыми сервисами, тренерами и ценами, а сделать все дистанционно.

В этом контексте web-приложение для «Фитнес-центра» представляет собой инновационное и полезное решение. Приложение объединяет преимущества современных технологий с необходимостью эффективного ознакомлени с предоставляемыми сервисами фитнес-центра и покупки понравившихся абонементов. Более того для администраторо присутствует удобный функционал для редацтирования карточек абонементов, тренеров которые включают в себе множество различных аспектов.

Основная цель проекта – создать универсальное и удобное в использовании web-приложение, которое позволит пользователям эффективно справляться с поиском и покупкой необходимых абонементов в фитнес-центре. Важным аспектом является предоставление возможности просматривать отзывы к тренерам фитнес-центра, а также возможность оставить свой отзыв, чтобы помочь себе и другим доверить свое здоровье и прогресс в правильные руки.

Для достижения поставленной цели в рамках курсового проекта были сформулированы следующие задачи: в разделе 1 провести анализ существующих сервисов для фитнес-центров и выявить их преимущества и недостатки; в разделе 2 разработать архитектуру и структуру приложения, включая выбор технологий и структуру базы данных; в разделе 3 описать программную разработку с реализацией ключевых функций; в разделе 4 провести тестирование функционала и производительности; в разделе 5 подготовить техническую документацию и инструкции по развёртыванию приложения.

Целевая аудитория приложения включает широкий спектр пользователей: от профессиональных спортсменов и до любителей здорового образа жизни, желающих улучшить свою физическую форму и получить доступ к качественным сервисам и тренера фитнес-центра.

1 Постановка задачи и обзор аналогичных решений

1.1 Постановка задачи

Web-приложение предназначено для покупки абонементов и управления этим процессом. Помимо этого немаловажными функциями являются просмотр и добавления отзывов к тренерам. Имеется возможность редактирования личной информации в профиле. Web-приложение предоставляет возможность гибкого редактирования тренеров, клиентов, и абонементов. Функционал приложения разделен на роли: гость, клиент и администратор.

Гость начинает с регистрации либо входа, аутентификации и авторизации в системе. После успешного прохождения данных этапов гость берет роль клиента. А клиент, в свою, очередь может выполнять следующие действия:

– просмотривать информации о абонементе: клиент может просматривать список абонементов, которые представлены ввиде карточек с детальной информацие;

– искать абонементы по критериям: клиент может вводить навзание абонемента и запускать процесс поиска;

– сортировать абонементы по фильтрам: клиент может задавать диапазон цен для филтрации, также сортировать абонементы по ценам;

– покупать абонементы: клиент может осуществлять покупку абонемента через сторонний платежный шлюз;

– оставлять отзывы о тренерах: клиент может написать комментарии к различным тренерам;

– редактировать личную информацию: пользователь может изменять своё имя и фото.

Администратор создается при первом запуске приложения. После аутентификации он может выполнять следующие действия:

– редактировать клиентов: администратор может удалять клиентов;

– редактировать абонементы: администратор имеет возможность гибкого добавления нового, обновления и удаления существующего абонемента.

– редактировать тренеров: администратор имеет возможность гибкого добавления нового, обновления и удаления существующего тренера;

– просмотривать информацию о абонементе: администратор может просматривать список абонементов, которые представлены ввиде карточек с детальной информацие;

– искать абонемент по критериям: администратор может вводить навзание абонемента и запускать процесс поиска;

– сортировки абонементов по фильтрам: администратор может задавать диапазон цен для филтрации, также сортировать абонементы по ценам;

Таким образом, web-приложение предназначено для удобной покупки абонементов и управления процессом их использования. Помимо этого, оно предоставляет возможности для взаимодействия с тренерами, включая просмотр и добавление отзывов. Пользователи могут редактировать свою личную информацию, что делает процесс взаимодействия более персонализированным.

1.2 Обзор аналогичных решений

## **1.2.1 x fit**

В качестве первого аналогичного решения было выбрано web-приложение Zona531. Это web-приложение для автоматизации процессов одноименного фитнес-центра, который предлагает широкий спектр функций, включая возможности для связи с тренерами, функции для просмотра и покупки различных услуг. Внешний вид страницы web-приложения представлен на рисунке 1.1.

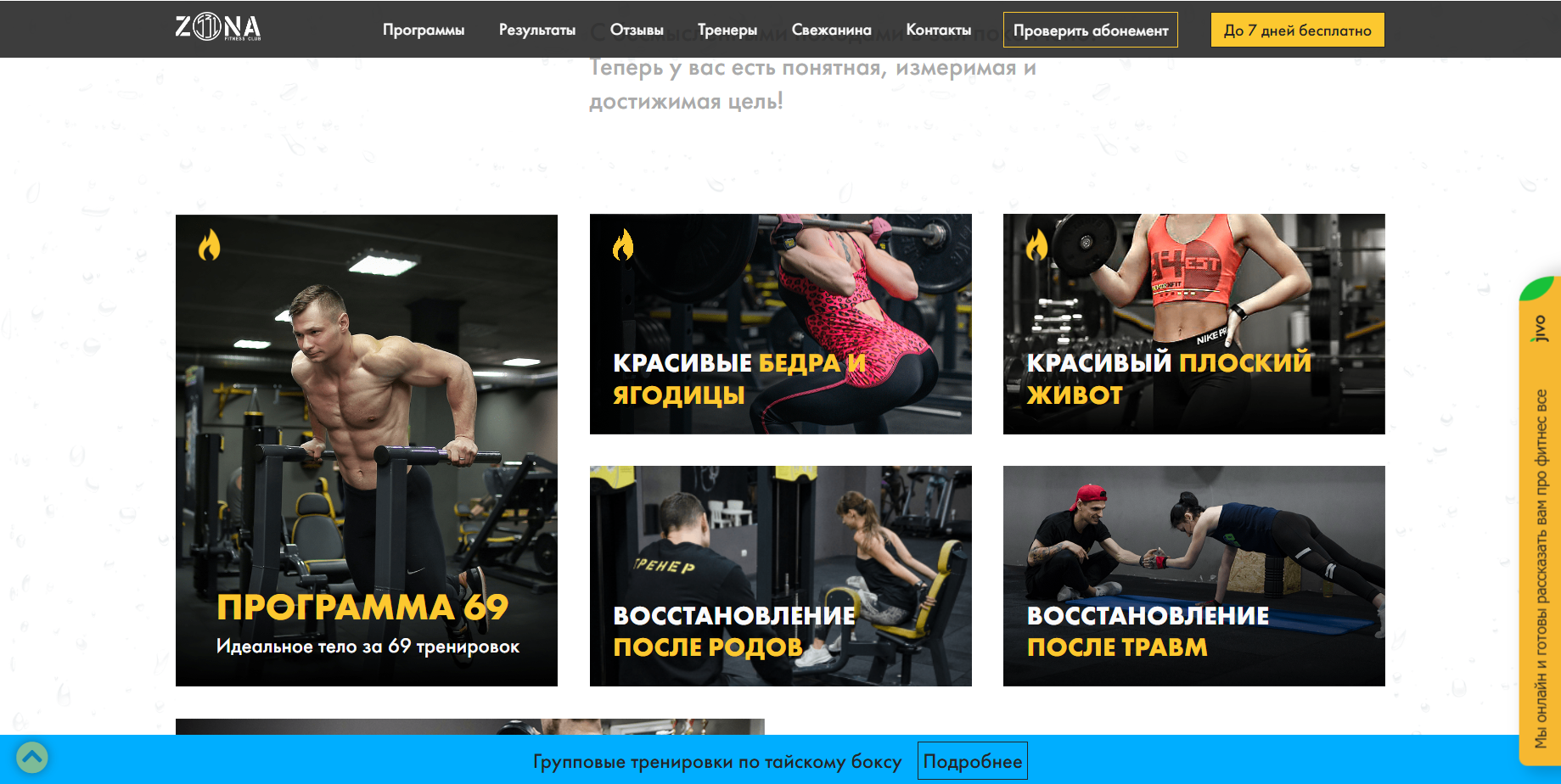


Рисунок 1.1 – Страница web-приложения Zona531

Обладает отличными возможностями для ознакомления с услугами фитнес-центра, легок и комфортен в использовании.

В качестве второго аналогичного решения было рассмотренно web-приложение Fitness-club.by. Внешний вид страницы данного web-приложения представлен на странице 1.2.

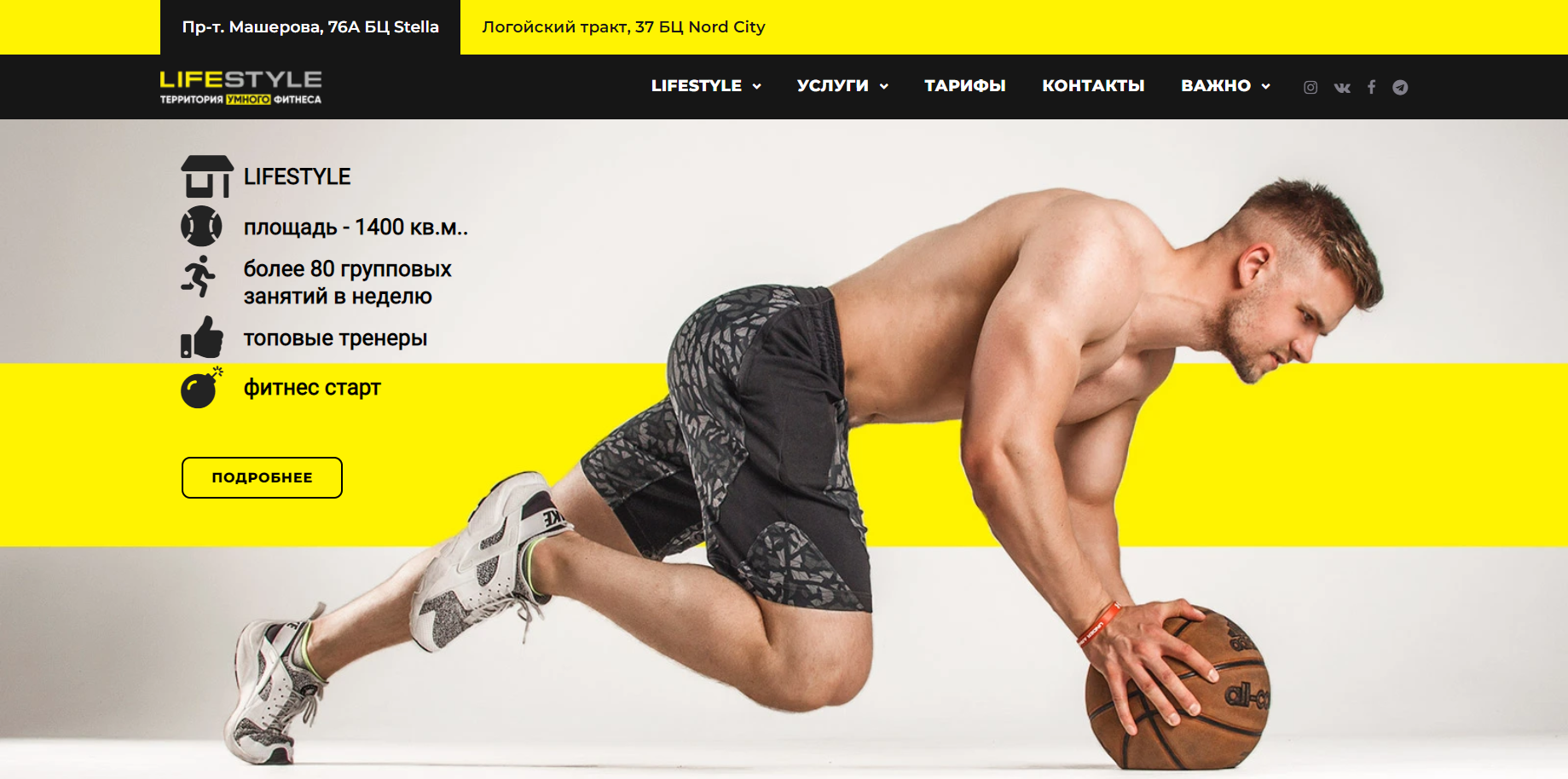


Рисунок 1.2 – Страница сервиса Fitness-club.by

[Среди функций Fitness-club.by можно выделить назначение плана тренировок в зависимости от выбранной услуги, временные скидки, интеграцию с другими веб-ресурсами, управление бюджетами проектов, а также наличие адаптивности для различных платформ](https://www.forbes.com/advisor/business/software/todoist-review/).

В качестве третьего аналогичного решения был рассмотренно web-приложение Oktopus Fitness Club. Внешний вид его страницы представлен на рисунке 1.3.

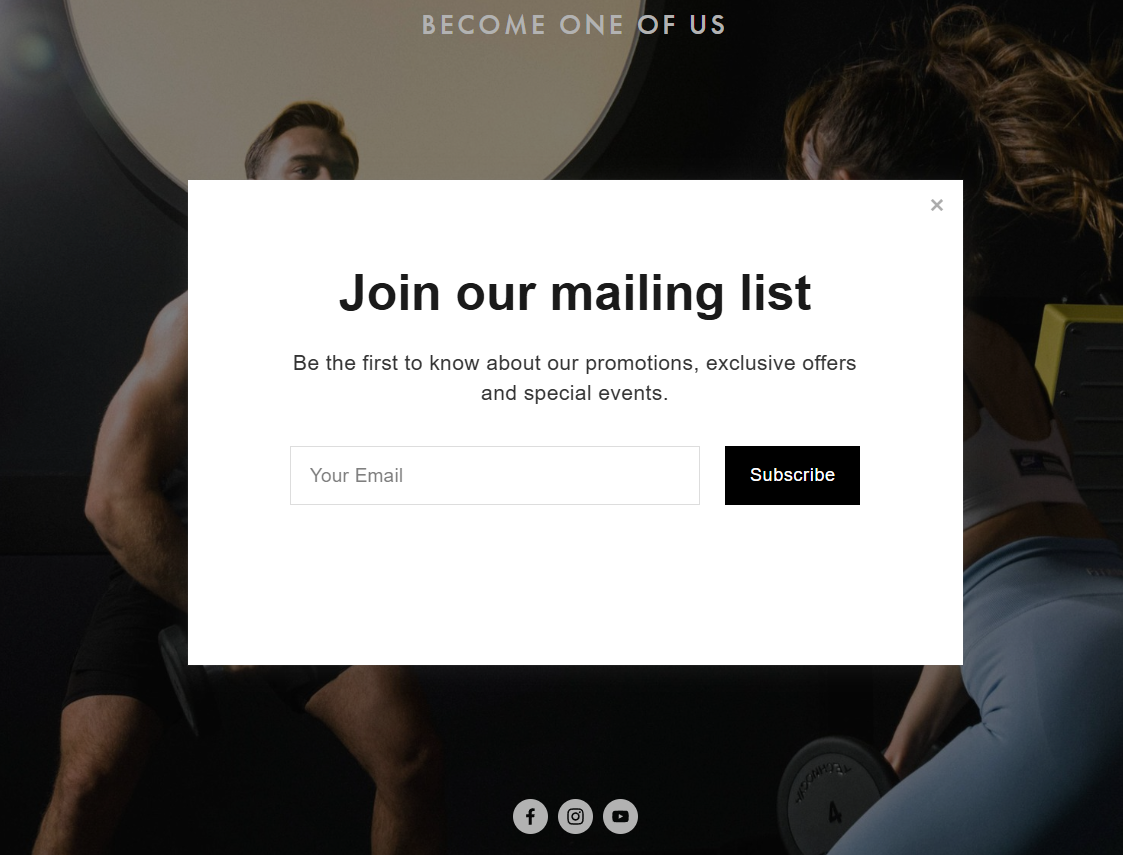


Рисунок 1.3 – Страница сервиса Oktopus Fitness Club

Среди функций Oktopus Fitness Club можно выделить удобный процесс покупки абонементов с использованием платежного шлюза, возможность просмотра и оставления отзывов о тренерах, персонализацию профиля пользователя (редактирование имени и фото), а также гибкое управление клиентами, тренерами и абонементами.

1.3 Выводы

При рассмотрении аналогичных решений были выделены их особенности.

Web-приложение Zona531 обладает следующими преимуществами:

– удобство использования: интерфейс приложения интуитивно понятен, что делает взаимодействие с ним комфортным даже для новых пользователей;

– ознакомление с услугами: приложение предоставляет широкие возможности для ознакомления с услугами фитнес-центра, включая подробное описание и визуализацию;

– функциональность: поддерживает функции связи с тренерами и покупки различных услуг, что значительно упрощает процесс взаимодействия с фитнес-центром;

Web-приложение Zona531 имеет следующие недостатки:

– ограниченная адаптивность: приложение может не оптимально работать на всех устройствах, особенно на мобильных;

– отсутствие дополнительных функций: по сравнению с другими решениями, в приложении отсутствуют такие возможности, как персонализация тренировочных планов или временные скидки;

– меньшее количество дополнительных функций: по сравнению с Oktopus Fitness Club, Fitness-club.by предлагает меньше дополнительных функций, таких как автоматическое списываение денег за абонементы.

Сервис Fitness-club.by обладает следующими преимуществами:

– назначение плана тренировок: приложение предоставляет функцию составления плана тренировок в зависимости от выбранной услуги;

– интеграция с другими веб-ресурсами: возможность интеграции с внешними платформами делает использование приложения более универсальным;

– управление бюджетами: удобная функция для планирования и контроля расходов;

– адаптивность: интерфейс приложения корректно отображается на различных устройствах, включая мобильные платформы.

Web-приложение Fitness-club.by имеет следующие недостатки:

– сложность интерфейса: несмотря на адаптивность, интерфейс может быть перегружен для новых пользователей;

– ограниченный функционал оплаты: интеграция с платежными системами не столь гибкая, как у некоторых конкурентов

Web-приложение Oktopus Fitness Club обладает следующими преимуществами:

– гибкость управления: приложение предоставляет удобные инструменты для работы с клиентами, тренерами и абонементами;

– персонализация профиля: пользователи могут редактировать имя, фото и другие данные, делая взаимодействие более удобным;

– отзывы о тренерах: возможность оставлять комментарии о тренерах способствует улучшению их работы и повышению удовлетворенности клиентов.

Web-приложение Oktopus Fitness Club имеет следующие недостатки:

– ограниченные возможности адаптации: отсутствие некоторых функций, таких как временные скидки или персонализированные планы тренировок;

– м еньшее количество интеграций: приложение имеет меньше возможностей для взаимодействия с другими веб-ресурсами по сравнению с Fitness-club.by.

2 Проектирование web-приложения

2.1 Функциональность web-приложения

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

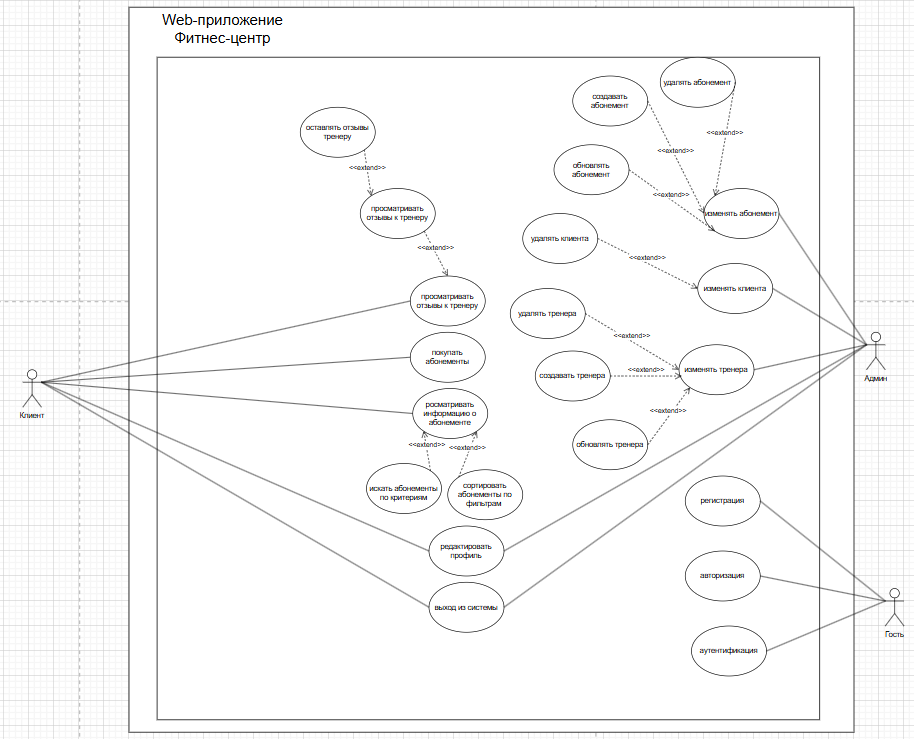


Рисунок 2.1 – Диаграмма вариантов использования web-приложения

Перечень ролей и их назначение приведены в таблице 2.1.

Таблица 2.1 – Назначение ролей пользователей в web-приложении

|  |  |
| --- | --- |
| Роль | Назначение |
| Гость | Регистрация, аутентификация и авторизация |
| Клиент | Выход из системы, редактирование профиля, просмотр абонементов, поиск абонементов, сортировка по фильтрам, покупка абонементов, просмотр отзывов к тренеру, добавление отзыва к тренеру |
| Администратор | Выход из системы, редактирование профиля, просмотр абонементов, поиск абонементов, сортировка по фильтрам, покупка абонементов, просмотр отзывов к тренеру, добавление отзыва к тренеру, редактирование тренера, редактирование клиента, редактирование абонемента |

Функционал гостя представлен в таблице 2.2.

Таблица 2.2 – Функционал гостя

|  |  |
| --- | --- |
| Вариант использования | Пояснение |
| Регистрироваться | Гость может создать учётную запись при помощи ника, электронной почты и пароля. |
| Аутентифицироваться | Гость может аутентифицироваться при помощи электронной почты и пароля. |
| Авторизоваться | Гость может аутентифицироваться при помощи электронной почты и пароля. |

После аутентификации гость становится либо клиентом, либо администратором. Функционал клиента представлен в таблице 2.3.

Таблица 2.3 – Функционал пользователя

|  |  |
| --- | --- |
| Вариант использования | Пояснение |
| Изменять учётную запись | Изменять своё имя и фото |
| Просматривать список абонементов | Получать список абонементов |
| Выходить из системы | Выход из приложения |
| Искать абонемент | Текстовый поиск по имени |
| Сортировать абонементы по фильтрам | Сортировка и фильтрация по цене |
| Просматривать список тренеров | Просматривать список тренеров |
| Оставлять комментарии к тренерам | Оставлять комментарии к тренерам |
| Покупать абонемент | Покупать абонемент |
| Просматривать список заказов | Получить список заказов |

Администратор Может редактировать основные сущности приложения. Его функционал представлен в таблице 2.4.

Таблица 2.4 – Функционал модератора

|  |  |
| --- | --- |
| Вариант использования | Пояснение |
| Изменять учётную запись | Изменять своё имя и фото |
| Редактировать тренера | Удаление, обновление, создание тренера |
| Редактирование абонемента | Удаление, обновление, создание абонемента |
| Редактирование клиента | Удаление клиента |
| Создавать комментарии для жалоб | Создавать комментарии для открытой жалобы |
| Просматривать список абонементов | Получать список абонементов |
| Выходить из системы | Выход из приложения |
| Искать абонемент | Текстовый поиск по имени |
| Сортировать абонементы по фильтрам | Сортировка и фильтрация по цене |
| Просматривать список тренеров | Просматривать список тренеров |
| Оставлять комментарии к тренерам | Оставлять комментарии к тренерам |

Таким образом, клиенту доступен функционал просмотра поиска и покупки абонементов, а администраторы – могут редактировать основные сущности приложения.

2.2 Структура базы данных

Согласно схеме вариантов использования была создана база данных. Её структура представлена на рисунке 2.2.

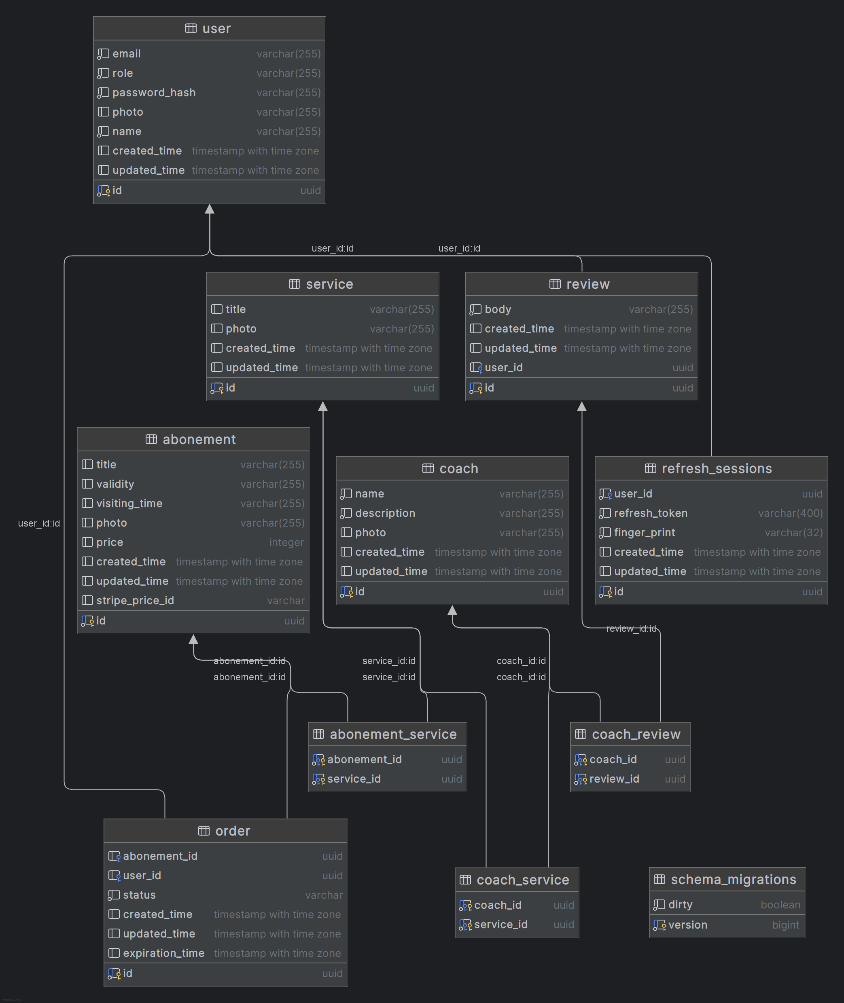


Рисунок 2.2 – Логическая схема базы данных

База данных содержит одинадцать таблиц, хранящих информацию о пользователях, клиентах, абонементах и прочих данных. Типы данных были выбраны согласно документации [1]. Назначение таблиц базы данных представлено в таблице 2.6.

Таблица 2.6 – Назначение таблиц базы данных

|  |  |
| --- | --- |
| Таблица | Назначение |
| Users | Хранит информацию о пользователях (имя, адрес электронной почты и хеш пароля для аутентификации и так далее) |
| Sessions | Хранит информацию о сессиях пользователей (идентификатор пользователя, флаг активности, время создания и так далее) |
| Confirmation\_codes | Хранит информацию о кодах подтверждения адреса электронной почты и сброса пароля |
| Languages | Хранит информацию о доступных для перевода языках (название, ISO код) |

Продолжение таблицы 2.6

|  |  |
| --- | --- |
| Articles | Хранит информацию о статьях (заголовок, текст, идентификатор пользователя и так далее) |
| Report\_reasons | Хранит информацию о доступных причинах для жалобы на перевод статьи (текст, позиция в списке для сортировки) |
| Reports | Хранит информацию о жалобах на переводы статей (идентификатор статьи, текст, идентификатор, причина и так далее) |
| Report\_comments | Хранит информацию о комментариях к жалобам на переводы статей (текст, идентификатор пользователя, идентификатор жалобы, дата и время создания) |
| Style\_prompts | Хранит информацию о запросах перевода с разными стилями (название, текст и так далее) |
| AI\_Models | Хранит информацию о моделях искусственного интеллекта, использующихся для перевода (название, поставщик и так далее) |
| Configs | Хранит информацию о конфигурациях переводчика, которые могут использоваться пользователями для упрощения запуска перевода своих статей (идентификаторы запроса перевода, модели, языков и так далее) |
| Translation\_tasks | Хранит информацию о задачах перевода, которые считываются отдельным процессом и выполняются им (идентификаторы статьи, модели, исходного и конечного языков, статус и так далее) |
| Notifications | Хранит информацию об уведомлениях пользователей (идентификатор пользователя, текст, тип уведомления и так далее) |

Описание столбцов таблицы Users представлено в таблице 2.7.

Таблица 2.7 – Описание таблицы Users

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Идентификатор пользователя, первичный ключ |
| name | varchar (20) | Имя пользователя |
| email | varchar (50) | Адрес электронной почты пользователя |
| email\_verified | boolean | Флаг, указывающий, был ли подтверждён адрес электронной почты пользователя |
| password\_hash | varchar (60) | Хеш пароля соискателя |
| role | enum user\_role | Роль пользователя (пользователь, модератор, администратор) |
| logged\_with\_provider | varchar | Название провайдера OAuth 2.0, использовавшегося для регистрации |

Продолжение таблицы 2.7

|  |  |  |
| --- | --- | --- |
| provider\_id | varchar | Идентификатор пользователя, полученный от провайдера OAuth при регистрации |
| created\_at | timestamp without timezone | Дата и время создания пользователя без часового пояса |
| deleted\_at | timestamp without timezone | Дата и время удаления пользователя без часового пояса |

Таблица Sessions хранит данные о сессиях пользователей. Описание её столбцов представлено в таблице 2.8.

Таблица 2.8 – Описание таблицы Sessions

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Идентификатор сессии, первичный ключ |
| user\_id | uuid | Идентификатор пользователя, который создал данную сессию, внешний ключ |
| ip | varchar (15) | IPv4 адрес узла, из которого была открыта сессия |
| user\_agent | varchar (100) | User agent клиента (например, браузера) |
| is\_closed | boolean | Флаг, указывающий, была ли сессия закрыта |
| refresh\_token\_id | uuid | Идентификатор refresh токена, связанного с данной сессией |
| created\_at | timestamp without timezone | Дата и время создания сессии без часового пояса |
| closed\_at | timestamp without timezone | Дата и время закрытия сессии без часового пояса |

Описание столбцов таблицы Confirmation\_codes представлено в таблице 2.9.

Таблица 2.9 – Описание таблицы Confirmation\_codes

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | integer | Идентификатор кода, первичный ключ |
| code | varchar | Строковое значение кода |
| reason | enum confirmationtype | Тип кода (подтверждение адреса электронной почты, сброс пароля) |
| user\_id | uuid | Идентификатор пользователя, для которого предназначен данный код подтверждения, внешний ключ |

Продолжение таблицы 2.9

|  |  |  |
| --- | --- | --- |
| expired\_at | timestamp without timezone | Временная отметка, после которой код будет считаться истёкшим |
| is\_used | boolean | Флаг, указывающий, был ли код использован |
| created\_at | timestamp without timezone | Дата и время создания кода без часового пояса |

Таблица Languages хранит информацию о языках, доступных для перевода. Описание её столбцов представлено в таблице 2.10.

Таблица 2.10 – Описание таблицы Languages

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | integer | Идентификатор языка, первичный ключ |
| name | varchar | Отображаемое название языка |
| iso\_code | varchar | ISO код языка |

Таблица Articles хранит информацию об исходных и переведённых статьях. Описание её столбцов представлено в таблице 2.11.

Таблица 2.11 – Описание таблицы Articles

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Идентификатор статьи, первичный ключ |
| title | varchar (50) | Название статьи |
| text | text | Текст статьи |
| user\_id | uuid | Идентификатор пользователя, которому принадлежит статья, внешний ключ |
| language\_id | integer | Идентификатор языка статьи, внешний ключ |
| original\_article\_id | uuid | Идентификатор статьи, переводом которой является данная статья, внешний ключ |
| like | boolean | Флаг, указывающий, какую оценку пользователь поставил переводу (положительную, отрицательную, не поставил оценку) |
| created\_at | timestamp without timezone | Дата и время создания статьи без часового пояса |
| deleted\_at | timestamp without timezone | Дата и время удаления статьи без часового пояса |

Описание столбцов таблицы Report\_reasons представлено в таблице 2.12.

Таблица 2.12 – Описание таблицы Report\_reasons

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | integer | Идентификатор причины, первичный ключ |
| text | varchar | Текст причины |
| order\_position | integer | Положение причины в списке при сортировке |

Описание столбцов таблицы Reports представлено в таблице 2.13.

Таблица 2.13 – Описание таблицы Reports

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Идентификатор жалобы, первичный ключ |
| text | varchar (1024) | Текст жалобы |
| article\_id | uuid | Идентификатор статьи, на которую была оставлена жалоба, внешний ключ |
| status | enum reportstatus | Статус жалобы (открыта, закрыта пользователем, отклонена, удовлетворена) |
| closed\_by\_user\_id | uuid | Идентификатор пользователя, закрывшего жалобу (пользователь, которому принадлежит статья или модератор), внешний ключ |
| reason\_id | int | Идентификатор причины, по которой была оставлена жалоба, внешний ключ |
| created\_at | timestamp without timezone | Дата и время создания жалобы без часового пояса |
| closed\_at | timestamp without timezone | Дата и время закрытия жалобы без часового пояса |

Описание столбцов таблицы Report\_comments представлено в таблице 2.14.

Таблица 2.14 – Описание таблицы Report\_comments

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Идентификатор комментария, первичный ключ |
| text | varchar (100) | Текст комментария |
| sender\_id | uuid | Идентификатор пользователя, оставившего комментарий, внешний ключ |

Продолжение таблицы 2.14

|  |  |  |
| --- | --- | --- |
| report\_id | uuid | Идентификатор жалобы, к которой был оставлен комментарий, внешний ключ |
| created\_at | timestamp without timezone | Дата и время создания комментария без часового пояса |

Описание столбцов таблицы Style\_prompts представлено в таблице 2.15.

Таблица 2.15 – Описание таблицы Style\_prompts

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | integer | Идентификатор запроса, первичный ключ |
| title | varchar (20) | Название запроса |
| text | varchar | Текст запроса |
| created\_at | timestamp without timezone | Дата и время создания запроса без часового пояса |
| deleted\_at | timestamp without timezone | Дата и время удаления запроса без часового пояса |

Описание столбцов таблицы AI\_Models представлено в таблице 2.16.

Таблица 2.16 – Описание таблицы AI\_Models

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | integer | Идентификатор модели, первичный ключ |
| show\_name | varchar (50) |  |
| name | varchar | Название модели |
| provider | varchar | Поставщик модели |
| created\_at | timestamp without timezone | Дата и время создания записи о модели без часового пояса |
| deleted\_at | timestamp without timezone | Дата и время удаления записи о модели без часового пояса |

Таблица Configs хранит информацию о конфигурациях переводчика. Описание её столбцов представлено в таблице 2.17.

Таблица 2.17 – Описание таблицы Configs

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | integer | Идентификатор конфигурации, первичный ключ |
| name | varchar (20) | Название конфигурации |
| user\_id | uuid | Идентификатор пользователя, создавшего конфигурацию, внешний ключ |

Продолжение таблицы 2.17

|  |  |  |
| --- | --- | --- |
| prompt\_id | integer | Идентификатор запроса перевода, внешний ключ |
| language\_ids | integer [] | Идентификаторы языков перевода |
| model\_id | integer | Идентификатор модели перевода, внешний ключ |
| created\_at | timestamp without timezone | Дата и время создания конфигурации без часового пояса |
| deleted\_at | timestamp without timezone | Дата и время удаления конфигурации без часового пояса |

Таблица Translation\_tasks хранит информацию о задачах перевода. Данная информация используется для определения текста исходной статьи, конечного языка и так далее. Описание столбцов таблицы представлено в таблице 2.18.

Таблица 2.18 – Описание таблицы Translation\_tasks

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Идентификатор задачи, первичный ключ |
| article\_id | uuid | Идентификатор исходной статьи, внешний ключ |
| target\_language\_id | integer | Идентификатор конечного языка, внешний ключ |
| prompt\_id | integer | Идентификатор запроса перевода, внешний ключ |
| model\_id | integer | Идентификатор модели перевода, внешний ключ |
| status | enum translationtaskstatus | Статус задачи (создана, в процессе выполнения, завершена успешно, завершена с ошибкой) |
| data | jsonb | Дополнительная информация о задаче (текст ошибки) |
| translated\_article\_id | uuid | Идентификатор переведённой статьи, внешний ключ |
| created\_at | timestamp without timezone | Дата и время создания задачи без часового пояса |
| deleted\_at | timestamp without timezone | Дата и время удаления задачи без часового пояса |

Описание столбцов таблицы Notifications представлено в таблице 2.19.

Таблица 2.19 – Описание таблицы Notifications

|  |  |  |
| --- | --- | --- |
| Название столбца | Тип данных | Описание |
| id | uuid | Идентификатор уведомления, первичный ключ |
| title | varchar | Заголовок уведомления |
| text | varchar | Текст уведомления |
| user\_id | uuid | Идентификатор пользователя, которому предназначено уведомление, внешний ключ |
| type | enum notificationtype | Тип уведомления (информационное, предупреждение, ошибка) |
| created\_at | timestamp without timezone | Дата и время создания записи о модели без часового пояса |
| read\_at | timestamp without timezone | Дата и время удаления записи о модели без часового пояса |

Назначение связей приведено в таблице 2.20.

Таблица 2.20 – Назначение связей между таблицами

|  |  |
| --- | --- |
| Связь | Назначение |
| Users.id- Notifications.user\_id | Идентификатор пользователя, которому адресовано уведомление |
| Users.id- Confirmation\_codes.user\_id | Идентификатор пользователя, которому предназначен код подтверждения |
| Users.id- Sessions.user\_id | Идентификатор пользователя, который создал сессию |
| Users.id- Articles.user\_id | Идентификатор пользователя, который загрузил статью или запустил перевод исходной статьи |
| Users.id- Configs.user\_id | Идентификатор пользователя, которому принадлежит конфигурация переводчика |
| Users.id- Commens.sender\_id | Идентификатор пользователя, отправившего комментарий |
| Users.id- Reports.closed\_by\_user\_id | Идентификатор пользователя, закрывшего жалобу (создавшего её пользователя или любого модератора) |

Продолжение таблицы 2.20

|  |  |
| --- | --- |
| Report\_reasons.id- Reports.reason\_id | Идентификатор причины, по которой была создана жалоба на перевод статьи |
| Articles.id- Articles.original\_article\_id | Идентификатор исходной статьи, из которой был создан перевод |
| Articles.id- Translation\_tasks.article\_id | Идентификатор статьи, которую необходимо перевести |
| Articles.id-Translation\_tasks.translated\_article\_id | Идентификатор перевода статьи |
| Articles.id- Reports.article\_id | Идентификатор перевода, на который была создана жалоба |
| Languages.id- Articles.language\_id | Идентификатор языка статьи |
| Languages.id- Translation\_tasks .target\_language\_id | Идентификатор конечного языка, на который необходимо перевести статью |
| Reports.id- Comments.report\_id | Идентификатор жалобы, под которой был оставлен комментарий |
| AI\_Models.id- Translation\_tasks.model\_id | Идентификатор записи о модели искусственного интеллекта, которая используется для перевода статьи |
| AI\_Models.id- Configs.model\_id | Идентификатор записи о модели искусственного интеллекта |
| Style\_prompts.id- Translation\_tasks.prompt\_id | Идентификатор запроса перевода, который используется для перевода статьи |
| Style\_prompts.id- Configs.prompt\_id | Идентификатор запроса перевода |

Таким образом, была спроектирована база данных для долговременного хранения информации web-приложения.

2.3 Архитектура web-приложения

Для запуска многоконтейнерных Docker-приложений используется инструмент Docker Compose. Он управляет набором контейнеров, в которых работают прочие компоненты web-приложения.

Для хранения данных используется СУБД PostgreSQL 15.

Для обслуживания web-приложение и предоставления доступа к скомпилированному пакету фронтэнд-приложения, созданному с использованием React.js, используется web-сервер Nginx.

Для синхронного обмена сообщениями между компонентами системы используется grpc фреймворк в связку с rest подходом для перенаправления на front.

Для поддержки оплат был использован платежный шлющ – stripe.

Для хранения фотографий был использован сервис s3 из localstack как аналог aws.

Архитектура web-приложения представлена на рисунке 2.3.

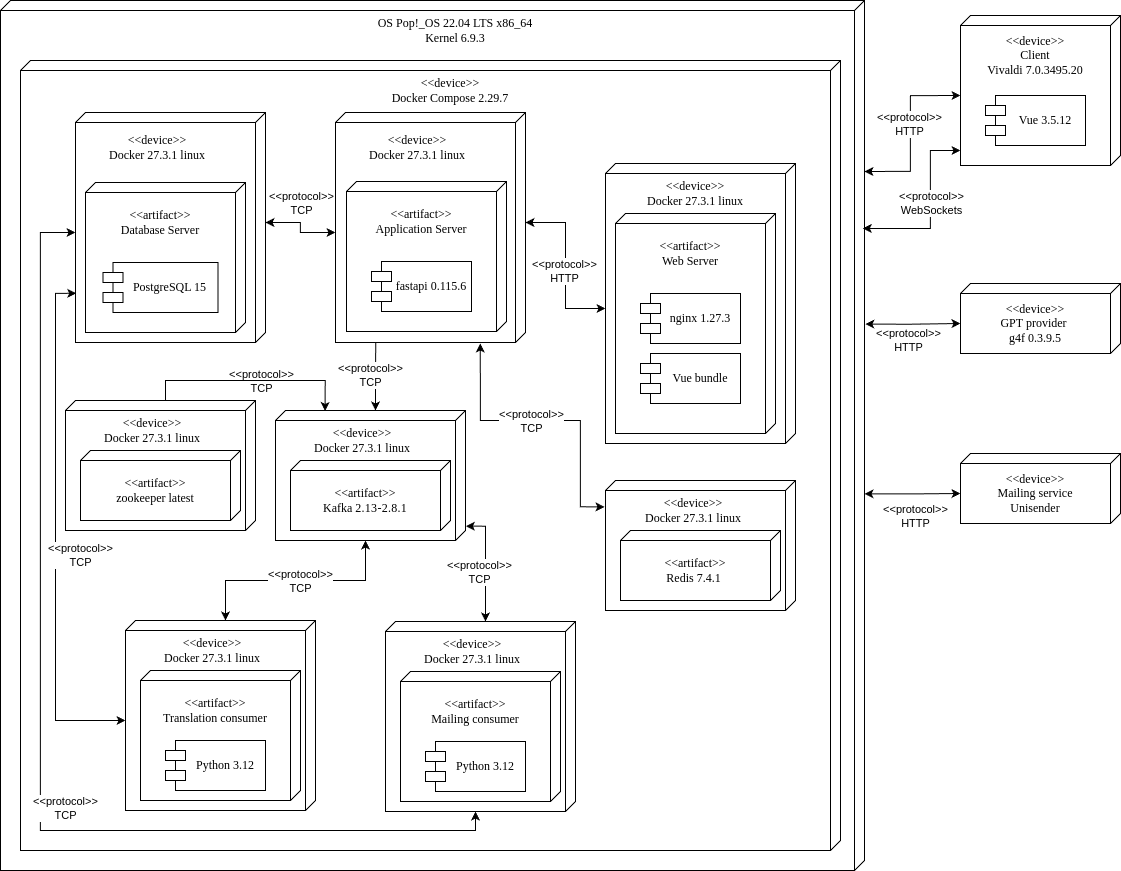


Рисунок 2.3 – Архитектура web-приложения

Пояснение назначения каждого элемента web-приложения представлено в таблице 2.21.

Таблица 2.21 – Назначение элементов архитектурной схемы web-приложения

|  |  |
| --- | --- |
| Элемент | Назначение |
| Web Server (nginx) | Принимать запросы клиента, предоставлять статические файлы фронтэнд-части web-приложения |
| Database Server (PostgreSQL) | Хранить данные, которые должны храниться длительное время |
| Gateway service | Принимать запросы клиента в формате http, преобразовывать запросы в формат protobuf для передачи через grpc 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 | Эмулировать локальную работу облачного сервиса s3 от AWS(Amazon Web Services) |
| Migrate | Выполнять миграции при первом запуске базы данных |
| Ngrok | Предоставлять публичный URL для связи внешнего платежного шлюза с сервисом Gateway |
| StripeAPI | Предоставлять API для работы с оплатами |
| Client (Microsoft Edge) | Отображать фронтэнд-часть web-приложения, отправлять запросы пользователя, отображать ответы сервера |

Таким образом, web-приложение состоит из различных компонентов, каждый из которых выполняет собственные функции.

2.4 Выводы

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

– поддерживает три роли пользователей: Гость, Клиент и Администратор – с чётко разграниченными правами доступа и функциональными возможностями;

– база данных web-приложения состоит из одиннадцати взаимосвязанных таблиц, которые охватывают все аспекты работы приложения: от управления пользователями до хранения заказов и обработки отзывов;

– web-приложение имеет микросервисную архитектуру с применением современных технологий: Nginx в качестве веб-сервера, PostgreSQL для долговременного хранения данных, grpc для быстрого синхронного обмена сообщениями между компонентами, Docker Compose для управления контейнерами;

– в web-приложении спроектированы механизмы безопасности и аутентификации: поддерживается аутентификация через адрес электронной почты и пароль при помощи JWT токенов. Также хранение хешей паролей вместо открытых паролей;

– web-приложение предоставляет комплексный функционал для работы с элементами фитнес-центра: возможность поиска и покупки абонементов, механизм добавления отзывов к тренерам, поддержку редактирования профиля клиента и администратора.

Web-приложение спроектировано с учетом масштабируемости и поддерживаемости, что отражается в структуре базы данных и микросервисной архитектуре системы.

3 Реализация web-приложения

3.1 Обоснование выбора программной платформы

Для реализации web-приложения был выбран язык программирования Golang для написания серверной части []. Фреймворк GIN – для написания rest api, фреймворк gRPC – для межпроцессорного взаимодействия сервисов [].

Для долговременного хранения данных web-приложения была выбрана СУБД PostgreSQL [].

Для урощения работы c базой данных была выбрана библиотека SQLX [].

Для управления миграциями был выбран инструмент Migrate [].

Для хранения фотографий был выбран аналог облачного хранилища s3 предоставляемый сервисом Localstack [].

Для управления оплатами была выбрана сплатформа Stripe [9]. Для получения публичного URL адреса и его перенаправление на локальный адрес для связи со Stripe.

3.2 Реализация web-приложения

В соответствии с диаграммой вариантов использования были реализованы функции, которые могут выполнять пользователи.

Функция “изменение учётной записи” (1) в исходном коде реализована функциями src.routers.users.views.change\_name, src.routers.users.views. request\_password\_restoration\_code и src.routers.users.views.restore\_password. Данные функции позволяют изменить отображаемое имя текущего пользователя, запросить электронное письмо со ссылкой на страницу установки нового пароля и установить новый пароль соответственно.

Функция “просмотр открытых сессий” (2) в исходном коде реализована функцией src.routers.sessions.views.get\_sessions. Данная функция возвращает список всех открытых сессий.

Функция “завершение открытых сессий” (3) в исходном коде реализована функцией src.routers.sessions.views.close\_sessions. Данная функция закрывает все открытые сессии пользователя.

Функция “изменение списка исходных статей” (4) в исходном коде реализована функциями src.routers.articles.views.upload\_article, src.routers.articles.views.update\_article, src.routers.articles.views.delete\_article. Данные функции позволяют создать новую исходную статью, обновить исходную статью и удалить исходную статью соответственно.

Функция “изменение списка переведённых статей” (5) в исходном коде реализована функциями src.routers.articles.views.delete\_article и src.routers.translation.views.create\_translation. Данные функции позволяют удалить перевод статьи и запустить перевод исходной статьи соответственно.

Функция “изменение списка жалоб на переводы своих статей” (6) в исходном коде реализована функциями src.routers.reports.views.create\_report, src.routers.reports.views.update\_report, src.routers.reports.views.update\_report\_status. Данные функции позволяют создать жалобу на перевод статьи, обновить текст и причину жалобы и закрыть жалобу соответственно.

Функция “просмотр своих уведомлений” (7) в исходном коде реализована функцией src.routers.notifications.views.get\_notifications\_list. Данная функция возвращает список непрочитанных уведомлений пользователя.

Функция “изменение списка комментариев к жалобам на переводы своих статей” (8) включает в себя ровно две функции: “получение списка комментариев к жалобе” (9) и “ создание комментария ” (10). Функция “получение списка комментариев к жалобе” (9) в исходном коде реализована функцией src.routers.reports.views.get\_comments. Данная функция возвращает список комментариев к жалобе. Функция “создание комментария” (10) в исходном коде реализована функцией src.routers.reports.views.create\_comment. Данная функция создаёт комментарий к жалобе.

Функция “изменение списка настроек переводчика” (11) в исходном коде реализована функциями src.routers.config.views.create\_config, src.routers.config.views.update\_config и src.routers.config.views.delete\_config. Данные функции позволяют создать конфигурацию, обновить конфигурацию и удалить конфигурацию соответственно.

Функция “регистрация” (12) в исходном коде реализована функцией src.routers.auth.views.register. Данная функция позволяет создать нового пользователя по имени, почте и паролю.

Функция “аутентификация” (13) в исходном коде реализована функциями src.routers.auth.views.login и src.routers.oauth.views.callback. Данные функции позволяют аутентифицироваться по логину и паролю и аутентифицировать пользователя по данным, полученным от провайдера OAuth 2.0 соответственно.

Функция “изменение списка открытых жалоб” (14) в исходном коде реализована функцией src.routers.reports.views.update\_report\_status. Данная функция позволяет удовлетворить или отклонить жалобу.

Функция “создание комментариев для жалоб” (15) в исходном коде реализована функцией src.routers.reports.views.create\_comment.

Функция “просмотр статистики жалоб” (16) в исходном коде реализована функциями src.routers.analytics.views.get\_models\_stats и src.routers.analytics.views.get\_prompts\_stats. Данные функции позволяют получить статистику жалоб по каждой модели и каждому стилю перевода.

Функция “изменение списка стилей перевода” (17) в исходном коде реализована функциями src.routers.prompts.views.create\_prompt, src.routers.prompts.views.update\_prompt и src.routers.prompts.views.delete\_prompt. Данные функции позволяют создать, обновить и удалить стиль перевода соответственно.

Функция “изменение списка моделей перевода” (18) в исходном коде реализована функциями src.routers.models.views.create\_model, src.routers.models.views.update\_model и src.routers.models.views.delete\_model. Данные функции позволяют создать, обновить и удалить модель перевода соответственно.

Функция “изменение списка пользователей” (19) в исходном коде реализована функциями src.routers.users.views.create\_user, src.routers.users.views.update\_user и src.routers.users.views.delete\_user. Данные функции позволяют создать, обновить и удалить

Функции “создание пользователей” (20), “создание модераторов” (21) и “создание администраторов” (22) в исходном коде реализована функцией src.routers.users.views.create\_user, представленной в листинге 3.17.

Таблица соответствия маршрутов контроллерам и функциям в исходном коде представлена в таблице 3.1.

Таблица 3.1 – Соответствие маршрутов контроллерам и функциям

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Префикс контроллера | Путь внутри контроллера | HTTP метод | Название контроллера | Название функции |
| /analytics | /models-stats/ | GET | Analytics | get\_models\_stats |
| /analytics | /prompts-stats/ | GET | Analytics | get\_prompts\_stats |
| /articles | / | GET | Articles | get\_list |
| /articles | / | POST | Articles | upload\_article |
| /articles | /{article\_id}/ | GET | Articles | get\_article |
| /articles | /{article\_id}/ | PUT | Articles | update\_article |
| /articles | /{article\_id}/ | DELETE | Articles | delete\_article |
| /articles | /{article\_id}/like/ | PATCH | Articles | update\_like |
| /auth | /confirm-email/request/ | POST | Auth | request\_email\_confirmation |
| /auth | /login/ | POST | Auth | login |
| /auth | /logout/ | GET | Auth | logout |
| /auth | /refresh/ | POST | Auth | refresh\_tokens |
| /auth | /register/ | POST | Auth | register |
| /auth | /registration/confirm/ | POST | Auth | confirm\_email |
| /auth | /restore-password/confirm/ | PATCH | Auth | restore\_password |
| /auth | /restore-password/request/ | POST | Auth | request\_password\_restoration\_code |
| /configs | / | GET | Configs | get\_configs |
| /configs | / | POST | Configs | create\_config |
| /configs | /{config\_id}/ | PUT | Configs | update\_config |
| /configs | /{config\_id}/ | DELETE | Configs | delete\_config |
| /languages | / | GET | Languages | get\_languages |
| /models | / | GET | Models | get\_models |
| /models | / | POST | Models | create\_model |
| /models | /admin/ | GET | Models | get\_admin\_models |
| /models | /{model\_id}/ | PUT | Models | update\_model |
| /models | /{model\_id}/ | DELETE | Models | delete\_model |
| /notifications | / | GET | Notifications | get\_notifications\_list |

Продолжение таблицы 3.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| /notifications | / | WEBSOCKET | Notifications | get\_notifications |
| /notifications | / | PUT | Notifications | mark\_notifications\_read |
| /oauth | /login/ | GET | OAuth | redirect\_to\_provider |
| /oauth | /{provider}/callback | GET | OAuth | callback |
| /prompts | / | GET | Prompts | get\_admin\_prompts |
| /prompts | / | POST | Prompts | create\_prompt |
| /prompts | /public/ | GET | Prompts | get\_prompts |
| /prompts | /{prompt\_id}/ | PUT | Prompts | update\_prompt |
| /prompts | /{prompt\_id}/ | DELETE | Prompts | delete\_prompt |
| Нет | /articles/{article\_id}/report/ | GET | Reports | get\_article\_report |
| Нет | /articles/{article\_id}/report/ | POST | Reports | create\_report |
| Нет | /articles/{article\_id}/report/ | PUT | Reports | update\_report |
| Нет | /articles/{article\_id}/report/comments/ | GET | Reports | get\_comments |
| Нет | /articles/{article\_id}/report/comments/ | POST | Reports | create\_comment |
| Нет | /articles/{article\_id}/report/comments/ws/ | WEBSOCKET | Reports | watch\_for\_comments |
| Нет | /articles/{article\_id}/report/status/ | PATCH | Reports | update\_report\_status |
| Нет | /report-reasons/ | GET | Reports | get\_report\_reasons |
| Нет | /reports/ | GET | Reports | get\_reports |
| /sessions | / | GET | Sessions | get\_sessions |
| /sessions | /close/ | POST | Sessions | close\_sessions |
| /translation | / | POST | Translation | create\_translation |
| /users | / | GET | Users | get\_list |
| /users | / | POST | Users | create\_user |
| /users | /me/ | GET | Users | get\_my\_info |
| /users | /{user\_id}/ | PUT | Users | update\_user |
| /users | /{user\_id}/ | DELETE | Users | delete\_user |
| /users | /{user\_id}/name/ | PATCH | Users | change\_name |

Полный исходный код web-приложения представлен в Приложении А.

Для передачи данных от клиента серверу и обратно используется протокол HTTP и формат JSON. Для сериализации, десериализации и валидации входящих и исходящих данных применялась библиотека Pydantic, позволяющая определять схемы при помощи классов и их свойств и берущая на себя проверку соответствия данных заданной схеме.

Пример схемы ответа представлен в листинге 3.1.

class ReportListItemScheme(Scheme):

article\_id: uuid.UUID

article\_title: str

status: ReportStatus

reason\_text: str

text: str

closed\_at: datetime | None = None

closed\_by\_user\_name: str | None = None

@classmethod

def create(cls, report\_object: Report, \*\*kwargs):

closed\_by\_user = report\_object.closed\_by\_user

closed\_by\_user\_name = closed\_by\_user.name if closed\_by\_user else None

return cls(

article\_id=report\_object.article\_id,

article\_title=report\_object.article.title,

status=report\_object.status,

closed\_at=report\_object.closed\_at,

closed\_by\_user\_name=closed\_by\_user\_name,

reason\_text=report\_object.reason.text,

text=report\_object.text,

)

Листинг 3.1 – Схема ответа для возврата списка жалоб

FastAPI автоматически проверяет тело запроса согласно указанной схеме, что упрощает разработку и читаемость кода.

3.3 Реализация базы данных

Согласно логической диаграмме базы данных, были созданы объекты базы данных. Модели SqlAlchemy объявлены в модуле src.database.models.py, представленном в Приложении А. Для изменения состояния базы данных использовался инструмент Alembic. Скрипт для создания базы данных и её объектов представлен в Приложении Б.

3.4 Выводы

Таким образом, было реализовано web-приложение, выполняющее все необходимые функции, с использованием архитектурного стиля REST. Была создана база данных для долговременного хранения данных и её объекты.

4 Тестирование web-приложения

4.1 Функциональное тестирование

Для тестирования работоспособности web-приложения необходимо добавить объекты пользователя и модератора в таблицу Users (роли “user” и “moderator” соответственно; имя, пароль и адрес электронной почты произвольные). Объект администратора, а также языки, модели, стили перевода и причины для жалоб добавляются в базу автоматически при развёртывании web-приложения в Docker Compose. Также автоматически создаётся база данных, и в ней создаются все необходимые объекты.

Для проверки функций приложения рекомендуется использовать инструмент OpenAPI. Данный инструмент генерирует документацию на основе исходного кода web-приложения. Фреймворк FastAPI включает данный инструмент, и страница документации по умолчанию доступна по адресу http://localhost:8000/api/docs. Описание тестирования функций web-приложения представлено в таблице 4.1.

Таблица 4.1 – Описание тестирования функций web-приложения

|  |  |  |
| --- | --- | --- |
| Функция web-приложения | Описание тестирования | Итог тестирования функции |
| 1 Изменение учётной записи | Аутентифицироваться в качестве пользователя (при помощи адреса электронной почты и пароля или провайдера OAuth 2.0), получить идентификатор своего пользователя при помощи GET запроса по адресу [/api/users/me/](http://localhost:8000/api/users/me/), отправить POST запрос на адрес  [/api/users/](http://localhost:8000/api/users/адрес){идентификатор своего пользователя}/name/, указав в теле запроса желаемое имя в параметре name (формат тела запроса – JSON). Сервер должен вернуть обновлённый объект пользователя в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 2 Просмотр открытых сессий | Аутентифицироваться в качестве пользователя, отправить GET запрос на адрес /api/sessions/. Сервер должен вернуть список сессий в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 3 Завершение открытых сессий | Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/sessions/close/. Сервер должен вернуть сообщение об успешном закрытии всех сессий | Работоспособность функции протестирована, ошибок не обнаружено |
| 4 Изменение списка исходных статей | Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/articles/, указав в теле запроса заголовок (title), текст (text) и идентификатор языка (language\_id) загружаемой статьи в формате JSON. Сервер должен вернуть объект статьи в формате JSON. Получить список языков в формате JSON можно, отправив GET запрос на адрес /api/languages/ | Работоспособность функции протестирована, ошибок не обнаружено |
| 5 Изменение списка переведённых статей | Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/translation/, указав в теле запроса идентификатор статьи, которую нужно перевести (article\_id), список идентификаторов языков, на которые нужно перевести статью (target\_language\_ids), идентификатор стиля перевода (prompt\_id) и идентификатор модели перевода (model\_id). Сервер должен вернуть сообщение о запуске перевода, через некоторое время, зависящее от объёма статьи, в таблице Notifications должна появиться запись об успешном или неуспешном переводе статьи. Списки моделей и стилей перевода можно получить по GET запросам на адреса /api/models/ и /api/prompts/ соответственно | Работоспособность функции протестирована, ошибок не обнаружено |
| 6 Изменение списка жалоб на переводы своих статей | Аутентифицироваться в качестве пользователя, отправить запрос на адрес /api/articles/{идентификатор статьи}/report/, в теле запроса указать текст жалобы (text) и идентификатор причины жалобы (reason\_id). Сервер должен вернуть объект жалобы в формате JSON. Список доступных причин жалоб можно получить при помощи GET запроса на адрес /api/report-reasons/ | Работоспособность функции протестирована, ошибок не обнаружено |
| 7 Просмотр своих уведомлений | Аутентифицироваться в качестве пользователя, отправить GET запрос на адрес /api/notifications/. Сервер должен вернуть список непрочитанных уведомлений в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 8 Изменение списка комментариев к жалобам на переводы своих статей | Протестировать функцию 10 Создание комментария, затем функцию 9 Получение списка комментариев к жалобе | Работоспособность функции протестирована, ошибок не обнаружено |
| 9 Получение списка комментариев к жалобе | Аутентифицироваться в качестве пользователя, отправить GET запрос на адрес /api/articles/ {идентификатор переведённой статьи, для жалобы на которую требуется получить список комментариев} /report/comments/. Сервер должен вернуть список комментариев в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 10 Создание комментария | Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/articles/ {идентификатор переведённой статьи, для жалобы на которую требуется создать комментарий} /report/comments/, в запросе указать текст комментария (text). Сервер должен вернуть объект комментария в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 11 Изменение списка настроек переводчика | Аутентифицироваться в качестве пользователя, отправить POST запрос на адрес /api/configs/, в запросе указать название конфигурации (name), идентификатор стиля перевода (prompt\_id), идентификатор модели перевода (model\_id) и список конечных языков (language\_ids). Сервер должен вернуть объект конфигурации в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 12 Регистрация | Отправить POST запрос на адрес /api/auth/register/, указав в теле запроса имя пользователя (name), адрес электронной почты (email) и пароль (password). Сервер должен вернуть сообщение об успешной регистрации | Работоспособность функции протестирована, ошибок не обнаружено |
| 13 Аутентификация | Отправить POST запрос на адрес /api/auth/login/, указав в теле запроса адрес электронной почты (email) и пароль (password). Сервер должен вернуть сообщение об успешной аутентификации | Работоспособность функции протестирована, ошибок не обнаружено |
| 14 Изменение списка открытых жалоб | Аутентифицироваться в качестве модератора, отправить на адрес /api/articles/ {идентификатор статьи, жалобу на которую нужно изменить} /report/status/ POST запрос, указав в параметрах запроса новый статус жалобы (Отклонена или Удовлетворена). Сервер должен вернуть объект жалобы в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 15 Создание комментариев для жалоб | Аутентифицироваться в качестве модератора, отправить на адрес /api/articles/ { идентификатор статьи, для жалобы на которую нужно создать комментарий} /report/comments/ POST запрос, указав в теле запроса текст комментария (text). Сервер должен вернуть объект комментария в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 16 Просмотр статистики жалоб | Аутентифицироваться в качестве администратора, отправить GET запрос на адрес /api/analytics/models-stats/. Сервер должен вернуть данные по жалобам для каждой модели перевода в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 17 Изменение списка стилей перевода | Аутентифицироваться в качестве администратора, отправить POST запрос на адрес /api/prompts/, в теле запроса указать название (title) и текст (text) стиля перевода Сервер должен вернуть объект стиля перевода в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 18 Изменение списка моделей перевода | Аутентифицироваться в качестве администратора, отправить POST запрос на адрес /api/models/, в теле запроса указать отображаемое название (show\_name), название (name) и провайдер (provider) модели перевода Сервер должен вернуть объект модели перевода в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 19 Изменение списка пользователей | Аутентифицироваться в качестве администратора, отправить POST запрос на адрес /api/users/, в теле запроса указать имя (name), адрес электронной почты (email), флаг, указывающий, подтверждена ли почта (email\_verified), роль (role) и пароль (password) пользователя. Сервер должен вернуть объект созданного пользователя в формате JSON | Работоспособность функции протестирована, ошибок не обнаружено |
| 20 Создание пользователей | Аналогично тестированию функции 19 Изменение списка пользователей, но роль в теле запроса должна быть “Пользователь” | Работоспособность функции протестирована, ошибок не обнаружено |
| 21 Создание модераторов | Аналогично тестированию функции 19 Изменение списка пользователей, но роль в теле запроса должна быть “Модератор” | Работоспособность функции протестирована, ошибок не обнаружено |
| 22 Создание администраторов | Аналогично тестированию функции 19 Изменение списка пользователей, но роль в теле запроса должна быть “Администратор” | Работоспособность функции протестирована, ошибок не обнаружено |

4.2 Нагрузочное тестирование

Таким образом, все заявленные функции были протестированы, и была проверена работоспособность web-приложения.

5 Руководство программиста

Для развёртывания web-приложения применяется инструмент Docker Compose. Перед развёртыванием web-приложения необходимо убедиться, что в системе установлены Docker Engine и Docker Compose при помощи команд docker version и docker compose version. В случае, если любая из указанных технологий не установлена, её необходимо установить согласно подходящей инструкции на официальном сайте, например, [10] для Docker Engine и [11] для Docker Compose. Далее необходимо создать пустую папку, в которую нужно поместить файлы, содержимое которых представлено в Приложении Б и Приложении В. Затем в данной папке необходимо создать файл .env, в котором указать необходимые значения переменных окружения, используемых web-приложением, таких как ключ доступа Unisender, логин и пароль для доступа к базе данных и так далее. Пример объявления переменных окружения находится в файле .example.env. Далее в корневой папке web-приложения необходимо последовательно выполнить команды “docker network create a”, “docker build -t diploma-base -f contrib/docker/base/Dockerfile .” и “ docker compose --env-file=.env -f contrib/docker/docker-compose.yaml up -d --build ”. Эти команды создадут новую сеть Docker, соберут базовый образ для контейнеров из исходного кода и запустят все необходимые контейнеры соответственно. В папке contrib/persistent\_data находятся .json файлы с данными, которыми будет заполнена база данных по умолчанию. Файлы включают данные о языках, моделях, стилях перевода и причинах жалоб. При каждом запуске контейнера api будет производиться проверка на наличие данных, которых нет в базе данных, и отсутствующие строки будут добавлены автоматически. Это позволяет добавлять новые строки в базу данных, не прибегая к использованию SQL запросов. Также при запуске контейнера api автоматически создаётся администратор с адресом электронной почты admin@d.com и паролем string, а также производится обновление структуры базы данных в соответствии с файлами миграций, находящихся в папке src/database/alembic/versions.

Для корректной работы web-приложения ему необходим доступ к внешнему сервису g4f. Он может находиться в любом удобном месте: на локальной машине, удалённом сервере и так далее. Для большего удобства можно развернуть его в Docker и добавить в ранее созданную сеть. Для этого нужно скачать базовый образ при помощи команды docker pull hlohaus789/g4f:0.3.9.5, развернуть его при помощи команды docker run --detach --name g4f hlohaus789/g4f:0.3.9.5, добавить созданный контейнер в ранее созданную сеть при помощи команды docker network connect a g4f. В файл .env нужно установить переменную окружения G4F\_ADDRESS=http://g4f:1337. Проверить доступность сервиса можно при помощи команды docker exec -t docker-api-1 bash -c "/app/contrib/docker/wait-for-it.sh \"g4f:1337\" -t 30 -- echo \"Сервис доступен\"". Для применения изменений в файле .env развёртывание web-приложения при помощи команды docker compose up необходимо повторить.

После развёртывания we-приложения по адресу <http://localhost:8000> будет доступна web-страница приложения. Шаги по проверке работоспособности развёрнутого сервиса описаны в разделе 4.

Заключение

При выполнении курсового проекта было создано приложение поиска и предложения работы. Сервер был создан при помощи платформы FastAPI, языка программирования Python и . Web-сайт был реализован при помощи библиотеки React. База данных была реализована в СУБД PostgreSQL. Были реализованы все функциональные требования, а именно:

– обеспечение возможности регистрации и авторизации;

– поддержка ролей гостя, соискателя, работодателя и администратора;

– обеспечение возможности изменять образование, опыт работы и список ключевых навыков соискателя;

– предоставление возможности работодателю принимать и отклонять отклики;

− обеспечение возможности оставлять отзывы об исполнителе и работодателе;

− предоставление возможности отслеживать статус предложения о работе в реальном времени;

− обеспечение возможности соискателю откликаться на предложения о работе;

− предоставление возможности создавать и удалять учётные записи;

− обеспечение возможности фильтровать предложения по критериям (тип, необходимый опыт, оплата и так далее);

− предоставление возможности размещать и удалять вакансии и заказы на выполнение работ.

Также были реализованы WebSocket-сервер и поддержка протокола HTTPS. Приложение было протестировано на наличие ошибок с использованием ручного и автоматического тестирования. Для ручного тестирования использовался браузер и Postman. Для автоматического тестирования использовалось вспомогательное приложение, использующее функцию fetch.

По итогам тестирования были исправлены следующие ошибки в приложении: неверный формат отправляемых данных, отсутствие необходимых данных в ответе, ошибки в проверке авторизации.

Также было создано иллюстрированное руководство пользователя, в котором были описаны способы взаимодействия с приложением и наглядно продемонстрированы web-страницы приложения.

Список используемых источников

1. PostgreSQL Documentation: 15: Chapter 8. Data Types [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.postgresql.org/docs/15/datatype.html;
2. Our Documentation | Python.org [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.python.org;
3. FastAPI [Электронный ресурс]. – Электронные данные. – Режим доступа: https://fastapi.tiangolo.com;
4. Welcome to Pydantic – Pydantic [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.pydantic.dev/latest/;
5. PostgreSQL: Feature Matrix [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.postgresql.org/about/featurematrix/;
6. SQLAlchemy - The Database Toolkit for Python [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.sqlalchemy.org;
7. Welcome to Alembic’s documentation! — Alembic 1.14.0 [Электронный ресурс]. – Электронные данные. – Режим доступа: https://alembic.sqlalchemy.org;
8. xtekky/gpt4free: The official gpt4free repository — GitHub [Электронный ресурс]. – Электронные данные. – Режим доступа: https://github.com/xtekky/gpt4free;
9. Unisender — сервис Email рассылок [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.unisender.com;
10. Install | Docker Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.docker.com/engine/install/;
11. Install | Docker Docs [Электронный ресурс]. – Электронные данные. – Режим доступа: https://docs.docker.com/compose/install/.

Приложение А

Исходный код web-приложения

Приложение Б

Скрипт создания объектов базы данных

Приложение В

Содержимое файлов, необходимых для развёртывания web-приложения