МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

КУРСОВАЯ РАБОТА

по дисциплине: Разработка серверных частей интернет-ресурсов
по профилю: Разработка программных продуктов и проектирование информационных систем
направления профессиональной подготовки: 09.03.04 «Программная инженерия»

Тема: Веб-сервис по организации спортивных мероприятий

Студент: Мухаметшин Александр Ринатович
Группа: ИКБО-20-21
Работа представлена к защите_____(дата)_____/ Мухаметшин А. Р. /
(подпись и ф.и.о. студента)

Руководитель: Синицын Анатолий Васильевич, старший преподаватель

Работа допущена к защите_____(дата)_____/ Синицын А. В. /
(подпись и ф.и.о. рук-ля)

Оценка по итогам защиты: _____
_____ / _____ /
_____ / _____ /
(подписи, дата, ф.и.о., должность, звание, уч. степень двух преподавателей, принявших защиту)

М. РТУ МИРЭА. 2023 г.

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт информационных технологий (ИТ)
Кафедра инструментального и прикладного программного обеспечения (ИиППО)

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине: Разработка серверных частей интернет-ресурсов

Студент: Мухаметшин Александр Ринатович
Группа: ИКБО-20-21
Срок представления к защите: 08.12.2023
Руководитель: Синицын Анатолий Васильевич, старший преподаватель

Тема: Веб-сервис по организации спортивных мероприятий

Исходные данные: используемые технологии: HTML5, CSS3, JavaScript, React.js, Python, VS Code, MySQL СУБД, наличие: межстраничной навигации, внешнего вида страниц, соответствующего современным стандартам веб-разработки, использование паттерна проектирования (MVC, Clear Architecture, DDD), нормативный документ: инструкция по организации и проведению курсового проектирования СМКО МИРЭА 7.5.1/04.И.05-18, ГОСТ 7.32-2017.

Перечень вопросов, подлежащих разработке, и обязательного графического материала: 1. Провести анализ предметной области разрабатываемого веб-приложения. 2. Обосновать выбор технологий разработки веб-приложения. 3. Разработать архитектуру веб-приложения на основе выбранного паттерна проектирования. 4. Реализовать слой серверной логики веб-приложения с применением выбранной технологии. 5. Реализовать слой логики базы данных. 6. Разработать слой клиентского представления веб-приложения 7. Создать презентацию по выполненной курсовой работе.

Руководителем произведён инструктаж по технике безопасности, противопожарной технике и правилам внутреннего распорядка.

Зав. кафедрой ИиППО: _____/Р. Г. Болбаков/, «14» сентября 2023 г.

Задание на КР выдал: _____/А.В. Синицын/, «14» сентября 2023 г.

Задание на КР получил: _____/А. Р. Мухаметшин/, «14» 09 2023 г.

АННОТАЦИЯ

УДК 004.4

Руководитель курсовой работы: старший преподаватель А.В. Синицын.

Мухаметшин А. Р., Курсовая работа направления подготовки «Программная инженерия» на тему «Веб-сервис по организации спортивных мероприятий»: М. 2023 г., МИРЭА – Российский технологический университет (РТУ МИРЭА), Институт информационных технологий (ИИТ), кафедра инструментального и прикладного программного обеспечения (ИиППО) – 38 стр., 17 рис., 20 источн. (в т.ч. 15 на английском яз.).

Ключевые слова: RESTful-приложение, игры, игроки, Spring Boot, Spring Security, JWT, Clear architecture, React.

Конечной целью проекта является разработка веб-приложения, предназначенного для содействия уличному спорту. В рамках проекта создана информационная система, предоставляющая возможность добавления стадионов, организации игр и формирования команд для участия в спортивных мероприятиях. Разработанное решение включает в себя клиентскую часть, базу данных и настроенный удаленный сервер.

Muhametshin A. R., Course work of the direction of training "Software Engineering" on the topic "Web-service for organizing sports events": M. 2023, MIREA - Russian Technological University (RTU MIREA), Institute of Information Technologies (IIT), Department of Instrumental and Applied Software (I&APS) - 38 p., 17 figures, 20 sources. (including 15 in English).

Keywords: RESTful application, games, players, Spring Boot, Spring Security, JWT, Clear architecture, React.

The ultimate goal of the project is to develop a web application designed to facilitate street sports. The project creates an information system that provides the ability to add stadiums, organize games and form teams to participate in sporting events. The developed solution includes a client part, a database and a customized remote server.

РТУ МИРЭА: 119454, Москва, пр-т Вернадского, д. 78

кафедра инструментального и прикладного программного обеспечения (ИиППО)

Тираж: 1 экз. (на правах рукописи)

Файл: «ПЗ_РСЧИР_ИКБО-20-21_МухаметшинАР.pdf», исполнитель Мухаметшин А. Р.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СКОРАЩЕНИЙ

RESTful API	—	Representational State Transfer API
СУБД	—	Система Управления Базами Данных
SQL	—	Structured Query Language
URL	—	Uniform Resource Locator
ПП	—	Программный продукт
API	—	Application Programming Interface (прикладной программный интерфейс)
JSON	—	JavaScript Object Notation (Нотация объектов JavaScript)
REST	—	Representational State Transfer (Архитектурный стиль передачи данных через HTTP)

ГЛОССАРИЙ

1. **Серверная часть** – часть приложения, которая занимается обработкой сложных данных, взаимодействует с базой данных и может лишь передавать клиенту данные в специальном установленном формате.

2. **Клиентская часть -** часть приложения, отвечающая за представление данных, полученных от сервера, пользователю. Также клиентская часть, иначе – «визуальная часть», позволяет взаимодействовать с пользователем, принимая от него какую-то информацию и передавая её на сервер.

3. **Архитектура приложения -** правила разработки приложения, которые часто позволяют упростить разработку, поддержку и расширение приложения.

4. **База данных -** хранилище с данными, которые разработчик предпочел хранить отдельно от клиентской части (пользователи, сотрудники и т.п.). Такое хранилище управляется посредством серверной части и СУБД.

5. СУБД - система, которая позволяет взаимодействовать с базой данных: добавлять, изменять, получать и удалять данные.

ВВЕДЕНИЕ

В настоящее время, развитие уличного спорта становится значимым аспектом физической активности в обществе. Он способствует здоровому образу жизни, формированию командного духа и развитию спортивных навыков у молодежи. Однако организация спонтанных спортивных мероприятий на улицах городов требует структурированной системы для управления играми, стадионами и участниками. В контексте растущей популярности уличного спорта, разработка веб-приложения для эффективной организации спортивных событий становится актуальной задачей, способствующей развитию этой области.

Существующие традиционные спортивные порталы не всегда охватывают специфику уличных видов спорта, где игры проводятся на неформальных площадках и требуют более гибкой системы управления. Создание специализированного веб-портала для уличного спорта позволит участникам более эффективно организовывать игры, присоединяться к командам, а также получать награды за активное участие. Такой портал предоставит удобный инструмент для координации и регистрации мероприятий, что сделает уличный спорт более доступным и привлекательным для широкой аудитории.

Таким образом, разработка веб-приложения для уличного спорта имеет большое значение в контексте его популяризации и структурирования, создавая удобную и эффективную платформу для всех участников этой динамичной среды.

Цель данной курсовой работы заключается в разработке веб-приложения, способствующего развитию уличного спорта и облегчающего организацию спортивных мероприятий. Система, создаваемая в ходе работы, представляет собой инструмент для добавления стадионов, организации игр и управления участниками через веб-портал.

Для выполнения поставленной цели курсовой работы необходимо выполнить следующие пункты:

– провести анализ предметной области разрабатываемого веб-

приложения;

– обосновать выбор технологий разработки веб-приложения;

– разработать архитектуру веб-приложения на основе выбранного паттерна проектирования;

– реализовать слой серверной логики веб-приложения с применением выбранной технологии;

– реализовать слой логики базы данных;

– разработать слой клиентского представления веб-приложения;

Объектом исследования представленной курсовой работы является веб-приложение, ориентированное на организацию уличных спортивных мероприятий и управление данными об играх, стадионах и участниках. Данное приложение представляет собой цифровой инструмент для координации и учета активности в уличном спорте.

При разработке проекта использовались знания, полученные в ходе лекционных и практических занятий по курсу «Разработка серверных частей интернет-ресурсов», а также дополнительные литературные источники.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для анализа предметной области было проведено исследование в области аналогичных приложений и интернет-ресурсов и были выявлены их преимущества и недостатки, представленные в таблице 1.

Таблица 1 – Сравнение аналогичных приложений

	Go-sport	trainvisor.com	goalstream.org
Регистрация и аутентификация	+	+	+
Управление стадионами	-	+	-
Организация игр	+	+	+
Присоединение к играм	+	+	+
Управление командами	-	-	+
Награды и достижения	-	-	-
История игр и статистика	+	+	+

На основе информации из таблицы 1 можно выделить несколько основных функций, которые должны присутствовать в приложении, такие как, регистрация пользователя, доступ к стадионам и играм, модерация добавляемых данных, возможность редактировать профиль. Одним из минусов аналогичных приложений является отсутствие возможности создавать и входить в команды и отсутствия наград за различные достижения.

На основе полученных данных были составлены функциональные требования для приложения. Приложение для обычных пользователей включает в себя такие функции как просмотр видов спорта и стадионов, создание игр и присоединение к играм, редактирование профиля, создание команд. Для пользователей-модераторов кроме основного функционала добавляется следующий: просмотр, добавление и удаление видов спорта, команд и наград, просмотр, редактирование и подтверждение стадионов, просмотр и удаление пользователей.

Вывод по разделу 1

Исходя из сравнения аналогичных приложений и выявленных преимуществ и недостатков, ключевые функциональные требования для создания приложения включают в себя: регистрацию и аутентификацию пользователей, управление доступом к стадионам и играм, возможность

создания и присоединения к играм, редактирование профилей и создание команд. Особое внимание следует уделить функционалу для пользователей-модераторов, такому как управление видами спорта, командами, стадионами, наградами и пользователями. Отсутствие функций управления командами и наградами в аналогичных приложениях подчеркивает необходимость в их внедрении для создания более полноценного и привлекательного для пользователей сервиса.

2 ВЫБОР ТЕХНОЛОГИЙ РАЗРАБОТКИ ПРИЛОЖЕНИЯ

2.1 Основные используемые технологии

При разработке приложения была выбрана архитектура клиент-серверного взаимодействия, где сервер выполняет основную обработку данных, а фронтенд является тонким клиентом. Для обеспечения этого подхода были использованы PostgreSQL, FastAPI (Python) и React.

FastAPI был выбран для серверной части приложения. Он предоставляет эффективные инструменты для обработки запросов, интеграции с другими технологиями и обеспечения безопасности, обеспечивает структурирование кода, его переносимость и масштабируемость. В частности, для работы с базой данных был использован SQLAlchemy, который упрощает взаимодействие с СУБД и предоставляет понятный интерфейс для выполнения запросов.

PostgreSQL была выбрана в качестве системы управления базами данных благодаря своей надежности и открытости. Она обеспечивает эффективное использование языка SQL для манипулирования данными, что важно для функционирования приложения.

Для обеспечения безопасности был использован пакет Pydantic, предоставляющий средства валидации данных и обработки запросов. Он интегрируется легко и помогает защитить приложение от возможных угроз.

Для клиентской части приложения был выбран React. Его компонентная архитектура облегчает поддержку и переиспользование кода. Использование виртуального DOM улучшает производительность интерфейса.

Интегрированная среда разработки, в моем случае, была Visual Studio Code. Она предоставляет удобную среду для написания кода, отладки и управления проектом, обладая широким набором функций для разработки веб-приложений.

Таким образом, приложение реализовано на основе клиент-серверной архитектуры с использованием FastAPI, PostgreSQL и React. FastAPI обеспечивает эффективное взаимодействие с базой данных, а React — легкость в разработке интерфейса и поддержке.

Вывод по разделу 2

Разработка приложения основана на клиент-серверной архитектуре с использованием FastAPI, PostgreSQL и React. FastAPI обеспечивает эффективное взаимодействие с базой данных, обеспечивая безопасность и масштабируемость, благодаря SQLAlchemy и Pydantic. PostgreSQL выбрана за надежность и эффективное использование SQL для манипулирования данными. React обеспечивает легкость разработки интерфейса и переиспользование кода. Интегрированная среда разработки Visual Studio Code обеспечивает удобную и мощную среду для написания, отладки и управления проектом веб-приложения. Такая комбинация инструментов позволяет создать функциональное и масштабируемое приложение, обеспечивая оптимальное взаимодействие пользователя с системой.

3 АРХИТЕКТУРА ПРИЛОЖЕНИЯ

3.1 Описание архитектуры

В созданном приложении применяется архитектурный подход Model-View-Controller (MVC), который предполагает явное разделение на «модели», «виды» и «контроллеры». Модели в данном контексте определяют структуру сущностей в базе данных, виды представляют данные, которые сервер возвращает клиенту в ответ на запрос, а контроллеры обрабатывают запросы к серверу.

В контексте созданного приложения понятие контроллера подразделяется на три основных компонента: «rest-контроллеры», «модели». Rest-контроллеры предоставляют возможность обработки запросов от клиента, определяя параметры для всех запросов, с которыми сервер должен взаимодействовать и занимаются обработкой данных, полученных от клиента используя модели. Модели, в свою очередь, взаимодействуют с базой данных, выполняя SQL-запросы для эффективного управления данными.

На рисунке 3.1.1 показана схема используемой архитектуры.

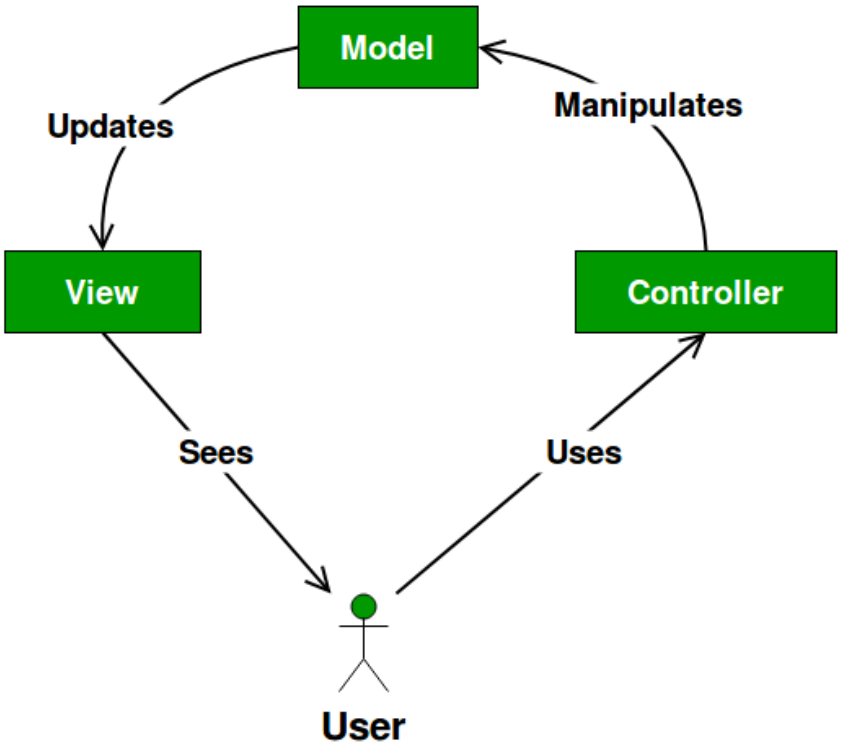

Рисунок 4.1.1 – Архитектура MVC

3.2 Реализация архитектуры в разрабатываемом проекте

На рисунке 3.2.1 показана используемая архитектура на примере получения информации о предметах.

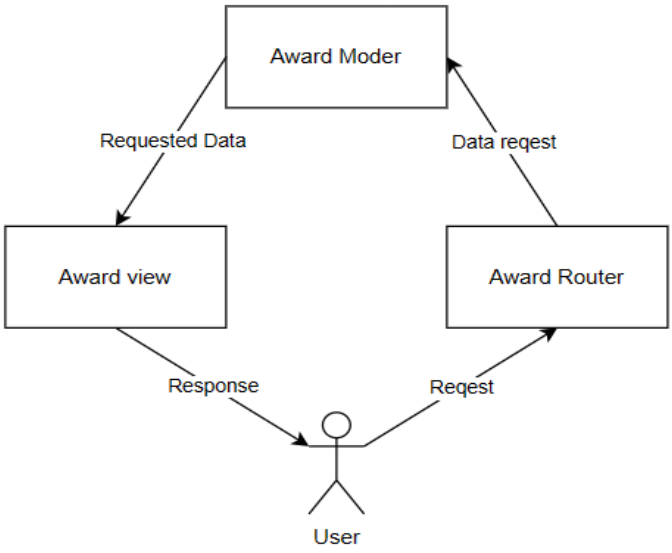

Рисунок 4.2.1 – Пример использования MVC архитектуры

Запрос от клиента поступает на контроллер, который обрабатывает запросы, касаемые предметов - листинг 3.2.1.

Листинг 3.2.1 – Rest – контроллер запросов данных о предметах

```
@router.get("/players/", response_model=PlayerList)
def get_all_players(limit: int, offset: int, db: Session =
Depends(get_db)):
    players = db.query(PlayerModel).all()
    total = len(players)
    player_data = [Player(**player.__dict__) for player in
players][offset:][:limit]
    return PlayerList(data=player_data, total=total)
```

В качестве ответа возвращается модель данных Player, которая описывает пользователя – Листинг 3.2.2.

Листинг 3.2.2 – Модель, возвращаемая сервером

```
import bcrypt
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship
from ..database import Base
```

Продолжение листинга 3.2.2

```
class Player(Base):
    __tablename__ = "player"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, nullable=False, unique=True)
    email = Column(String, nullable=False, unique=True)
    password = Column(String, nullable=False)
    experience = Column(Integer, nullable=False)
    team_id = Column(Integer, ForeignKey('team.id'), nullable=True)
    game_id = Column(Integer, ForeignKey('game.id'), nullable=True)
    role = Column(String)

    player_award = relationship("PlayerAward", back_populates="player")
    awards = relationship("Award", secondary="player_award")
    team = relationship("Team", back_populates="players")
    game = relationship("Game", back_populates="players")

    def set_password(self, password):
        hashed_password = bcrypt.hashpw(password.encode('utf-8'),
bcrypt.gensalt()).decode('utf-8')
        self.password = hashed_password
```

Вывод по разделу 3

Описание архитектуры приложения базируется на подходе Model-View-Controller (MVC), где "модели" определяют структуру базы данных, "виды" представляют данные, а "контроллеры" обрабатывают запросы к серверу. Rest-контроллеры занимаются обработкой запросов от клиента и взаимодействуют с моделями для управления данными в базе. Такой подход позволяет эффективно обрабатывать запросы от клиента и управлять данными, соблюдая структурирование и явное разделение обязанностей между компонентами системы.

4 РАЗРАБОТКА БАЗЫ ДАННЫХ

4.1 Проектирование базы данных

При разработке базы данных для нашего приложения, основанного на уличном спорте, была проведена детальная работа над моделью данных. Эта модель включает в себя несколько основных сущностей, которые отражают ключевые аспекты функционирования приложения.

1. Команда (Team): Эта сущность представляет участников, объединенных в команды. Включает в себя основную информацию о команде, такую как название и количество участников.

2. Вид спорта (Sport Type): Описывает различные виды спорта, которые поддерживаются в приложении. Включает информацию о названии вида спорта и его описании.

3. Стадион (Stadium): Представляет собой места проведения игр и тренировок. Включает информацию о названии стадиона, его адресе, координатах, а также связь с определенным видом спорта.

4. Игра (Game): Описывает конкретные игры, проходящие на стадионах. Содержит информацию о названии игры, сложности, времени начала и окончания, а также количество участников.

5. Пользователь (Player): Представляет собой участников приложения. Содержит информацию о имени, электронной почте, пароле, роли в приложении, принадлежности к команде, участии в играх и опыте.

6. Награды (Award): Описывает различные награды, которые могут быть получены пользователями при выполнении определенных условий. Содержит информацию о названии награды, ее описании и количестве очков опыта, связанных с этой наградой.

7. Связи между сущностями: База данных использует различные связи, такие как связь между игроком и командой, игрой и стадионом, игроком и игрой для обеспечения целостности данных и взаимодействия между сущностями.

Эти сущности были тщательно спроектированы и связаны между собой для обеспечения эффективного хранения информации, поддержки функционала приложения и обеспечения целостности данных. Итоги проектирования базы данных представлены в ER-диаграмме на рисунке 4.1.1.

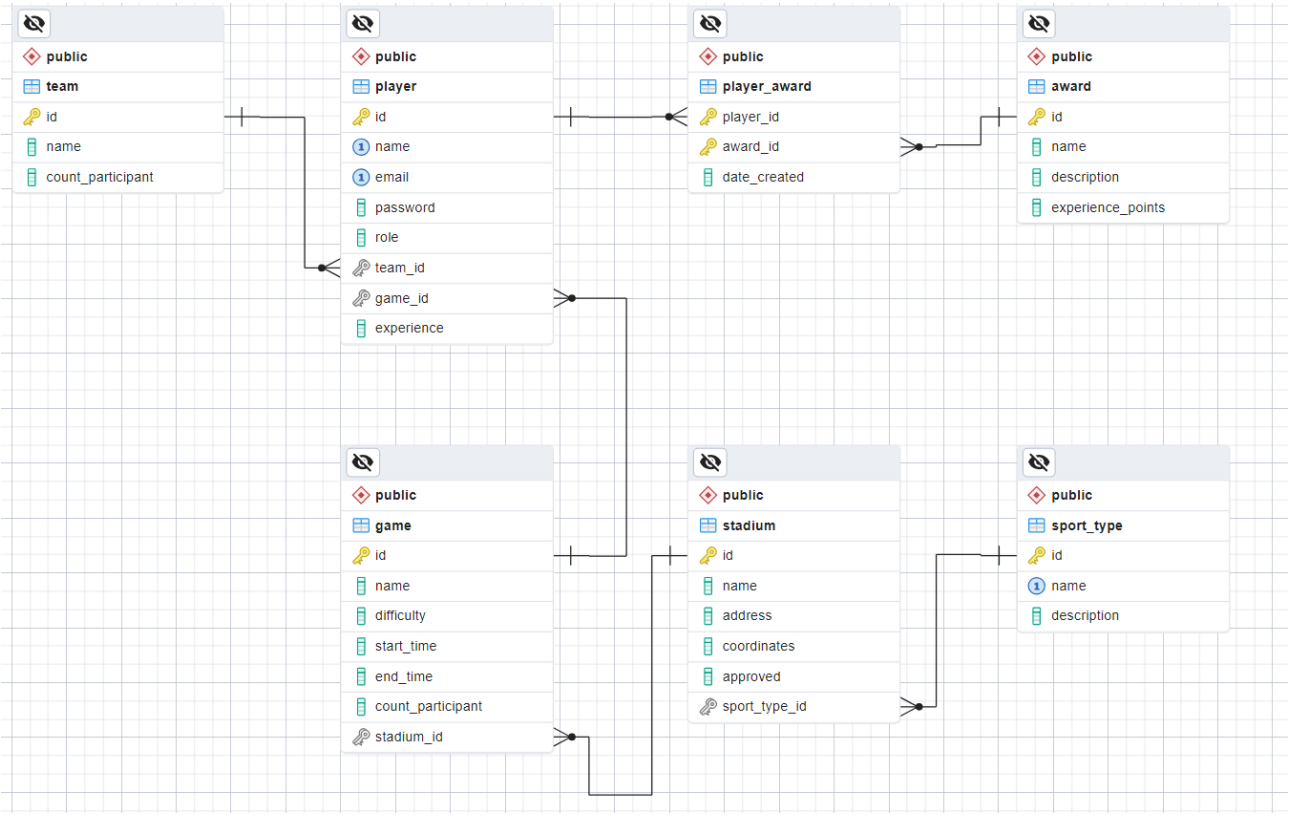

Рисунок 4.1.1 – ER-диаграмма базы данных

4.2 Разработка скрипта базы данных

При разработке скрипта базы данных использовались SQL запросы для создания таблиц, их связей и триггеров для автоматического обновления связанных данных.

Были созданы разработанные таблицы. Для примера, рассмотрим создание таблицы player, которая содержит информацию о пользователях (листинг 4.2.1).

Листинг 4.2.1 – пример создания таблицы

```
CREATE TABLE IF NOT EXISTS player (
    id BIGINT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    name VARCHAR(100) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    password VARCHAR(100) NOT NULL,
    role VARCHAR(10) DEFAULT 'user',
```

Продолжение листинга 4.2.1

```
    team_id BIGINT REFERENCES team,
    game_id BIGINT REFERENCES game,
    experience INT NOT NULL
);
```

Для реализации связей многие-с-многим использовались промежуточные таблицы. Например, таблица player_award связывает таблицы player и award, позволяя отслеживать полученные награды пользователями. Пример скрипта представлен на листинге 4.2.2.

Листинг 4.2.2 – пример реализации связи многие-к-многим

```
CREATE TABLE IF NOT EXISTS player_award (
    player_id BIGINT REFERENCES player,
    award_id BIGINT REFERENCES award,
    date_created TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (player_id, award_id)
);
```

Так же были реализованы триггеры, для автоматического добавления наград участнику, если он набирает подходящее количество необходимого опыта. Пример такого триггера приведен на листинге 4.2.3.

Листинг 4.2.3 – триггер добавление награды к пользователю

```
CREATE OR REPLACE FUNCTION update_player_award()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO player_award (player_id, award_id)
    SELECT NEW.id, a.id
    FROM award a
    WHERE NEW.experience >= a.experience_points;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_player_award_trigger
AFTER INSERT ON player
FOR EACH ROW
EXECUTE FUNCTION update_player_award();
```

Вывод по разделу 4

При разработке базы данных для приложения, ориентированного на уличный спорт, была создана детальная модель, включающая сущности, такие как команды, виды спорта, стадионы, игры, пользователи и награды, с особым вниманием к связям между ними. ER-диаграмма наглядно отображает структуру данных. Для создания таблиц и связей были использованы SQL-запросы, позволяющие установить целостность данных и автоматизировать процессы. Также были реализованы триггеры для автоматического назначения наград пользователям при достижении определенного опыта.

5 РАЗРАБОТКА СЕРВЕРНОЙ ЧАСТИ

5.1 Создание точки входа и подключение базы данных

Для создания серверной части проекта был использован фреймворк FastAPI, обеспечивающий высокую производительность и эффективную разработку веб-приложений на Python. На листинге 5.1.1 представлен код точки входа проекта, который осуществляет подключение к базе данных и инициализацию приложения.

Листинг 5.1.1 – основной файл проекта

```
app = FastAPI()

# Настройка CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Подключение к базе данных
SQLALCHEMY_DATABASE_URL =
"postgresql://postgres:postgres@localhost/play_hub"
engine = create_engine(SQLALCHEMY_DATABASE_URL)

# Проверка существования базы данных и создание, если она отсутствует
if not database_exists(engine.url):
    create_database(engine.url)

# Создание сессии для работы с базой данных
SessionLocal = sessionmaker(autocommit=False, autoflush=False,
bind=engine)

# Инициализация базового класса для описания моделей
Base = declarative_base()
```

Этот код инициализирует приложение FastAPI, добавляет обработчик CORS для обеспечения доступа к ресурсам с разных источников, а также

осуществляет подключение к базе данных PostgreSQL. Кроме того, он создает сессию для работы с базой данных с использованием SQLAlchemy и определяет базовый класс для описания моделей данных.

Это включение кода в приложение FastAPI и настройка соединения с базой данных обеспечивают основу для разработки серверной части приложения, позволяя взаимодействовать с базой данных и обрабатывать запросы от клиентов.

5.2 Создание слоя валидации данных

Валидация данных является важным этапом обработки информации в веб-приложениях. Для обеспечения корректности и безопасности данных, поступающих от клиентов, использовались инструменты валидации, такие как Pydantic в контексте описанных моделей данных.

Pydantic предоставляет удобный способ определения схемы данных и автоматической валидации входных данных, что позволяет:

1. Определить структуру данных: С помощью Pydantic задаются модели данных (например, GameBase, GameCreate), определяющие ожидаемые поля и их типы.

2. Проверять корректность данных: При создании экземпляров классов Pydantic автоматически происходит валидация переданных данных на соответствие определенным типам, что позволяет избежать ошибок из-за неправильного формата или типа данных.

3. Приводить данные к заданному формату: Pydantic способен автоматически приводить данные к заданным типам (например, преобразовывать строки в даты и обратно), обеспечивая единообразие данных.

Пример использования модели данных GameCreate для создания новой игры. Приведен на листинге 5.2.1

Листинг 5.2.1 – создание модели pydantic

```
from datetime import datetime
from pydantic import ValidationError
```

Продолжение листинга 5.2.1

```
class GameCreate(BaseModel):
    name: str
    difficulty: str
    start_time: datetime
    end_time: datetime
    stadium_id: int
```

5.3 Создание слоя моделей

Слой моделей в приложении представляет собой отражение структуры базы данных в объектную модель Python с использованием SQLAlchemy. В этом слое определяются классы, соответствующие таблицам в базе данных, их поля и отношения между ними.

В приведенном примере (листинг 5.3.1) класс Game является моделью для работы с таблицей game в базе данных. Он использует библиотеку SQLAlchemy для создания отображения между данными в базе и объектами Python.

Каждый атрибут класса Game отображается на соответствующее поле в таблице базы данных. Например, поля id, name, difficulty, start_time, end_time, stadium_id, count_participant соответствуют столбцам таблицы game.

Листинг 5.3.1 – создание модели Game

```
class Game(Base):
    __tablename__ = "game"

    # Определение полей и типов соответствующих столбцам таблицы game
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String(100), nullable=False)
    difficulty = Column(String(100), nullable=False)
    count_participant = Column(Integer, default=0)

    # Определение отношений между таблицами
    stadiums = relationship("Stadium", back_populates="game") #
Отношение многие к одному (множество игр в одном стадионе)
```

Этот код позволяет создавать объекты класса Game, которые могут использоваться для работы с данными из таблицы game. Он также определяет

отношения между таблицами, что облегчает навигацию и работу с данными в базе данных через объекты Python.

Используя подобный подход, создаются модели для каждой таблицы базы данных, обеспечивая удобство и гибкость при работе с данными в приложении.

5.4 Разработка слоя контроллеров

Слой контроллеров в приложении играет ключевую роль в обработке запросов, поступающих от клиентов, и управлении бизнес-логикой приложения.

Ответственности слоя контроллеров:

Обработка запросов: Контроллеры определены для различных HTTP-методов (POST, GET, DELETE) и обрабатывают соответствующие запросы, направляемые на создание игры, получение списка игр по стадиону или удаление игры.

Валидация данных: При помощи FastAPI и Pydantic происходит валидация входных данных, приходящих от клиентов. Например, контроллер create_game использует схему GameCreate для валидации данных, переданных для создания новой игры.

Обращение к базе данных: Контроллеры получают доступ к базе данных через зависимость db: Session = Depends(get_db). Это позволяет выполнять различные запросы, включая поиск, добавление или удаление записей в базе данных, связанные с играми.

Обработка ошибок: Контроллеры обрабатывают возможные ошибки, которые могут возникнуть в процессе выполнения запросов. Например, при попытке создания игры во время уже существующего временного пересечения с другой игрой для того же стадиона, будет вызвано исключение HTTPException с соответствующим сообщением об ошибке.

Принцип работы слоя контроллеров:

Контроллеры определены с помощью FastAPI и представляют собой функции, привязанные к конкретным URL-маршрутам. Каждый контроллер принимает определенные параметры, например, для доступа к базе данных, или модели данных для создания или обработки запросов.

При обращении клиента по определенному URL, FastAPI маршрутизирует запрос на соответствующий контроллер, который выполняет необходимые операции, включая валидацию, обращение к базе данных и формирование ответа.

Этот слой является посредником между входящими запросами и базой данных, обеспечивая правильное выполнение операций и обработку данных, необходимых для функционирования приложения.

5.5 Добавление планировщика

Так же в проект был добавлен планировщик задач, код которого приведен на листинге 5.5.1.

Листинг 5.5.1 – код планировщика задач

```
from datetime import datetime
from apscheduler.schedulers.background import BackgroundScheduler
from app import SessionLocal
from app.models.player import Player as PlayerModel

def check_users():
    print("check game out user")
    with SessionLocal() as db:
        users = db.query(PlayerModel).filter(PlayerModel.game_id != 
None).all()
        current_time = datetime.now()
        for user in users:
            if user.game.end_time and user.game.end_time < 
current_time:
                print("find user")
                user.game_id = None
                user.experience += 10

        db.commit()



scheduler = BackgroundScheduler()
scheduler.add_job(check_users, 'interval', minutes=1)
print("------scheduler start------")
```

Этот фрагмент кода относится к добавлению планировщика BackgroundScheduler, который предоставляется библиотекой APScheduler для управления запуском функций по расписанию.

Особенности планировщика:

1. Создание в новом потоке: Планировщик создается для запуска в фоновом режиме, что позволяет выполнять функцию check_users периодически без блокирования основного потока приложения. Используется BackgroundScheduler, чтобы планировщик работал в отдельном фоновом потоке.

2. Цель и функциональность: Функция check_users проверяет данные игроков, которые участвуют в играх. Она выполняет следующие действия:

- Получает всех пользователей, участвующих в играх, из базы данных.
- Проверяет, завершилась ли игра для каждого пользователя.
- Если игра завершилась, обновляет соответствующие данные пользователя: устанавливает game_id в None и добавляет опыт игроку.

3. Использование текущего времени: Планировщик выполняет эту функцию с определенной периодичностью (в данном случае, каждую минуту). Он сравнивает текущее время с временем окончания игры у каждого пользователя, обновляя данные при необходимости.

4. Назначение: Планировщик является механизмом, позволяющим автоматизировать и периодически выполнять определенные операции или проверки в приложении. В данном случае, он используется для обновления данных пользователей по истечении времени игры.

Планировщик играет ключевую роль в обновлении состояния данных пользователей в реальном времени на основе завершения игр, что позволяет приложению реагировать на изменения в данных и обеспечивать актуальность информации для пользователей.

Вывод по разделу 5

Раздел по разработке серверной части приложения основанного на уличном спорте включал создание точки входа через FastAPI, валидацию данных

с помощью Pydantic, создание моделей SQLAlchemy для работы с базой данных, контроллеры для обработки запросов и планировщик задач BackgroundScheduler. Было разработано приложение, обеспечивающее безопасный и корректный обмен информацией с базой данных, правильное выполнение запросов от клиентов, а также периодическое обновление данных пользователей после завершения игр, обеспечивая актуальность информации для пользователей.

6 РАЗРАБОТКА КЛИЕНТСКОЙ ЧАСТИ

6.1 Страницы разработанного приложения

Frontend-часть разработанного мной приложения представляет собой динамичный пользовательский интерфейс, созданный с использованием современных технологий. Основной фреймворк, на котором построено приложение, — React. React обеспечивает эффективное управление состоянием компонентов, и это ядро взаимодействия пользователя с интерфейсом.

На рисунках 6.2.1 – 6.2.8 показаны разработанные страницы клиентской части проекта.

При первом заходе на сайт пользователя встречает модальное окно для регистрации/авторизации. На главной странице пользователь может ознакомиться со всеми доступными видами спорта, стадионами и играми. На странице профиля пользователь может ознакомиться с информацией о своем аккаунте, вступить в команду и посмотреть свои награды. Ознакомиться со всеми доступными наградами пользователь может на странице наград. Если пользователь является модератором, у него появляется страница «Модерка» на которой он может редактировать всю доступную информацию.

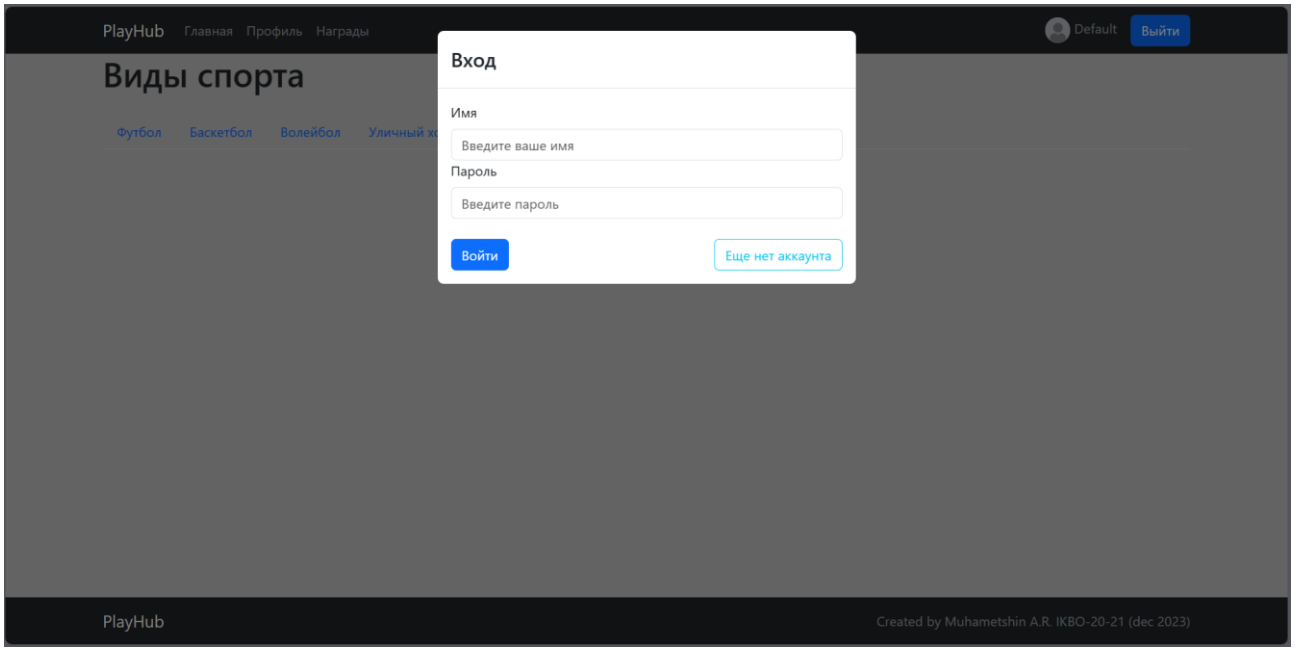

Рисунок 6.2.1 – Авторизация

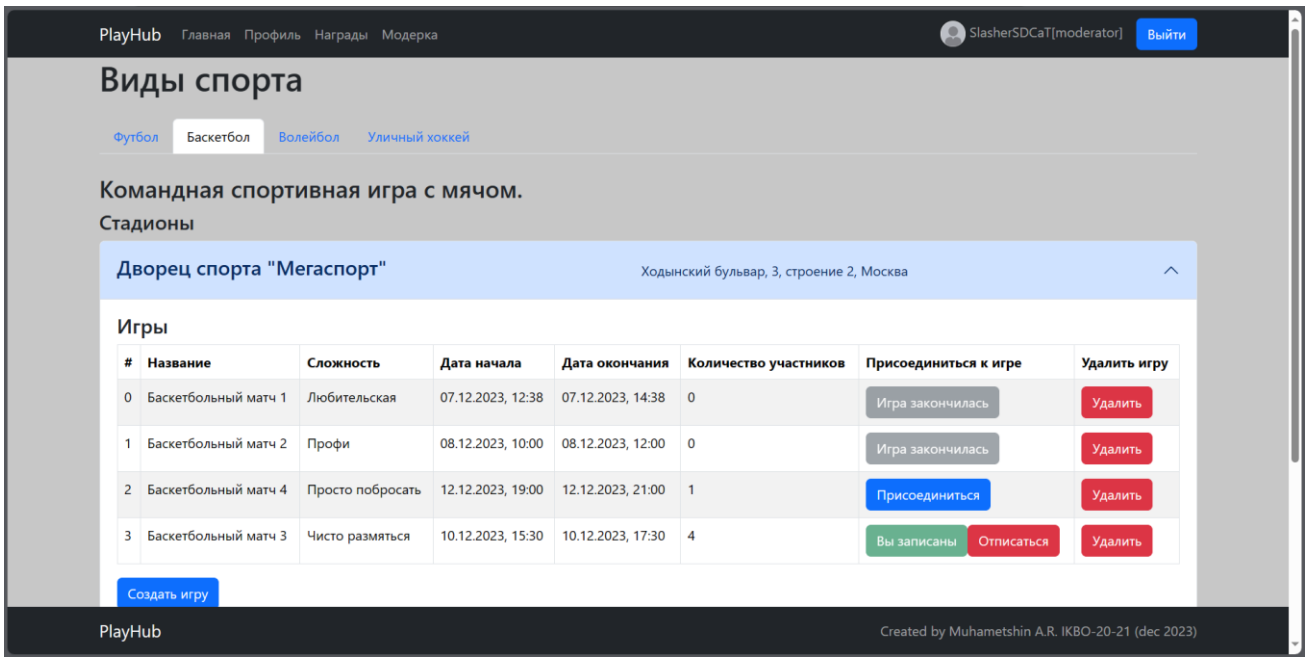

Рисунок 6.2.2 – Главная страница

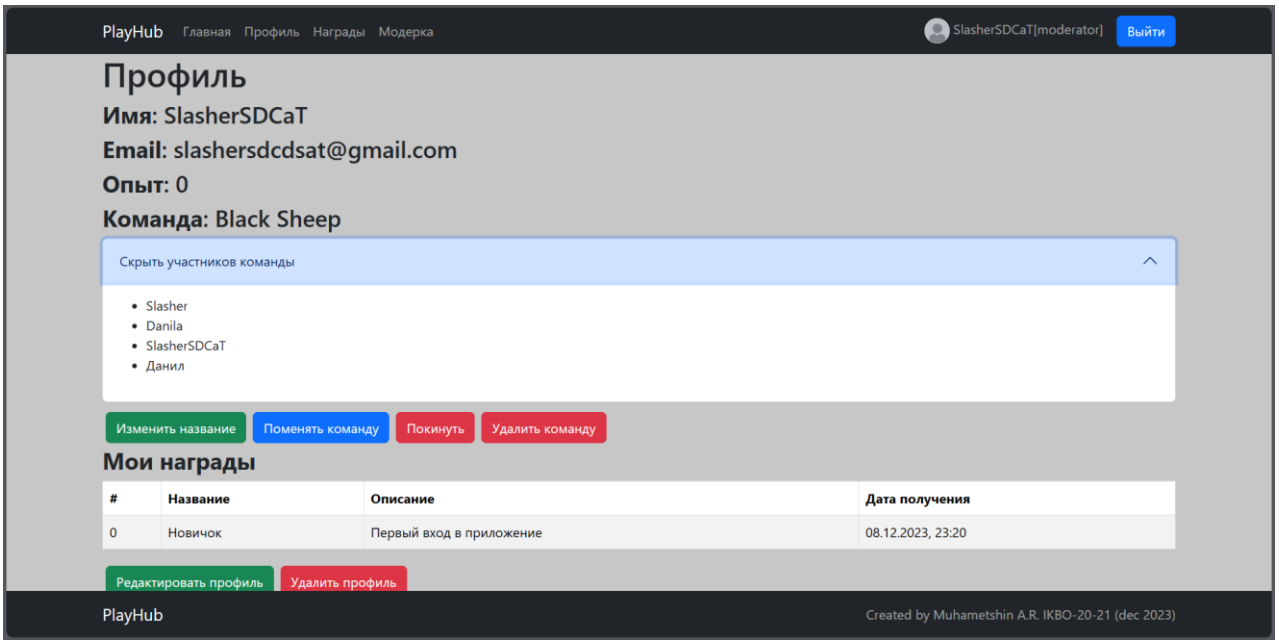

Рисунок 6.2.3 – Страница профиля

Рисунок 6.2.4 – Страница наград

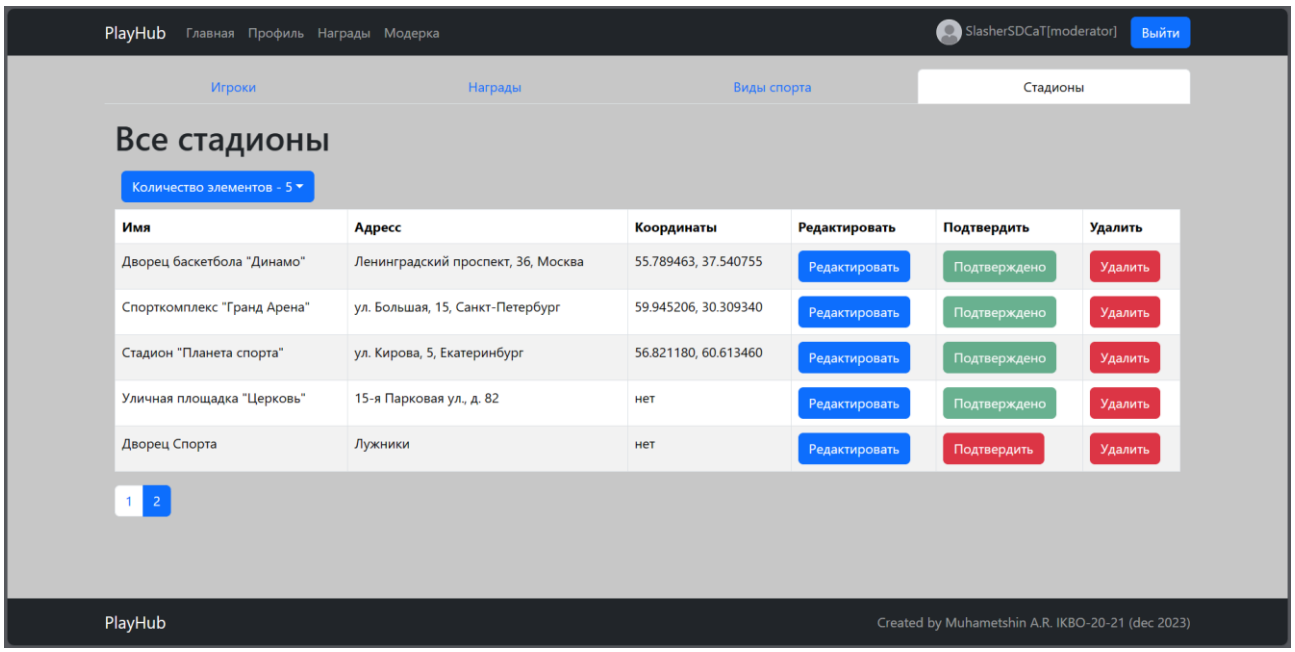

Рисунок 6.2.5 – Страница модерации стадионов

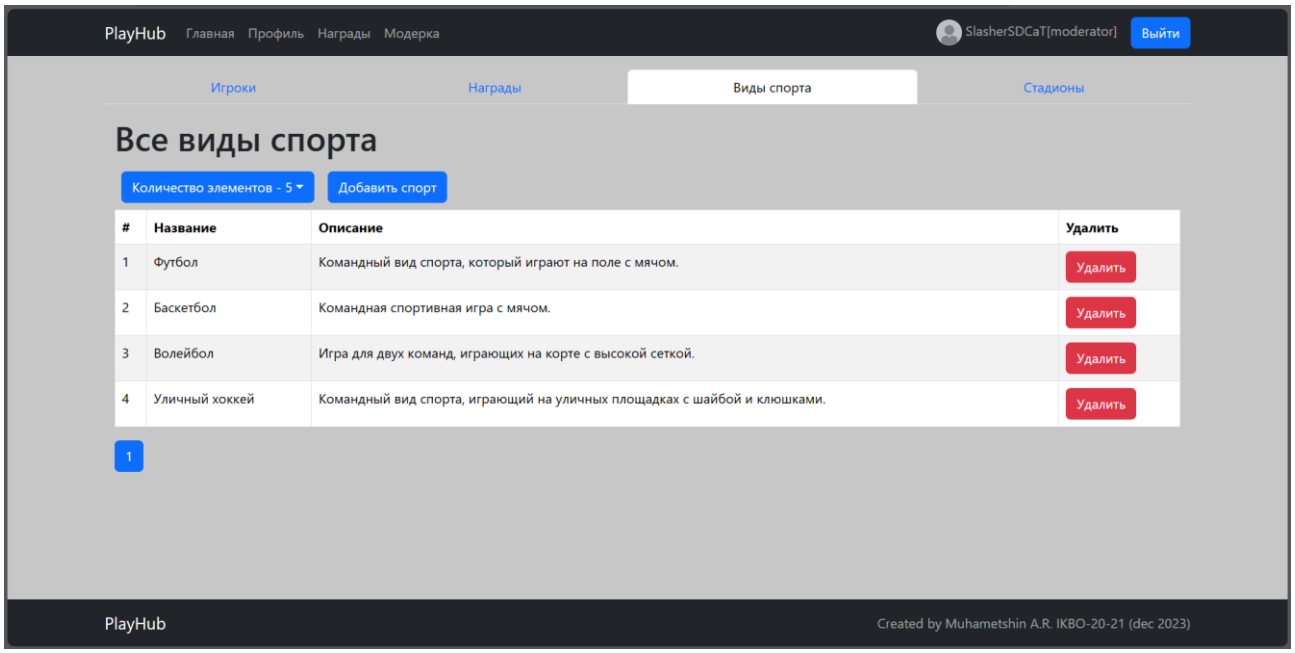

Рисунок 6.2.6 – Страница модерации видов спорта

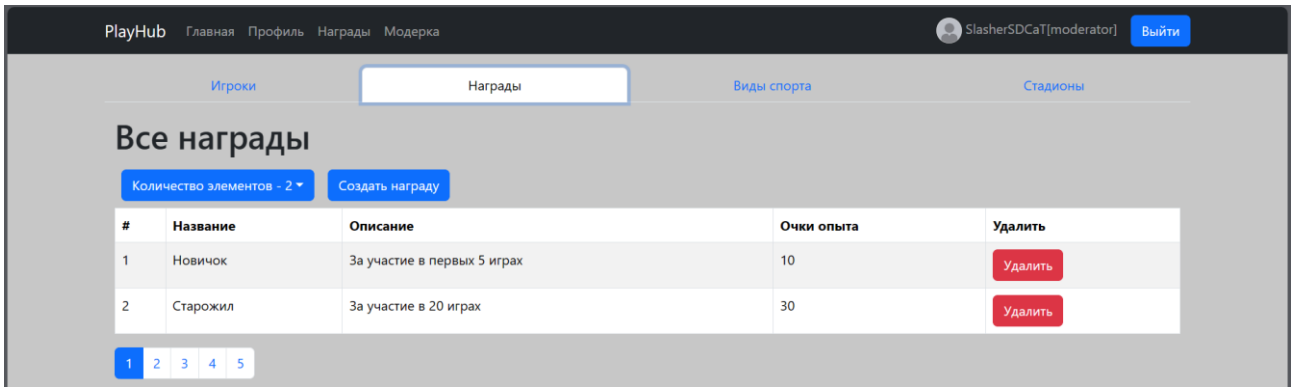

Рисунок 6.2.7 – Страница модерации наград

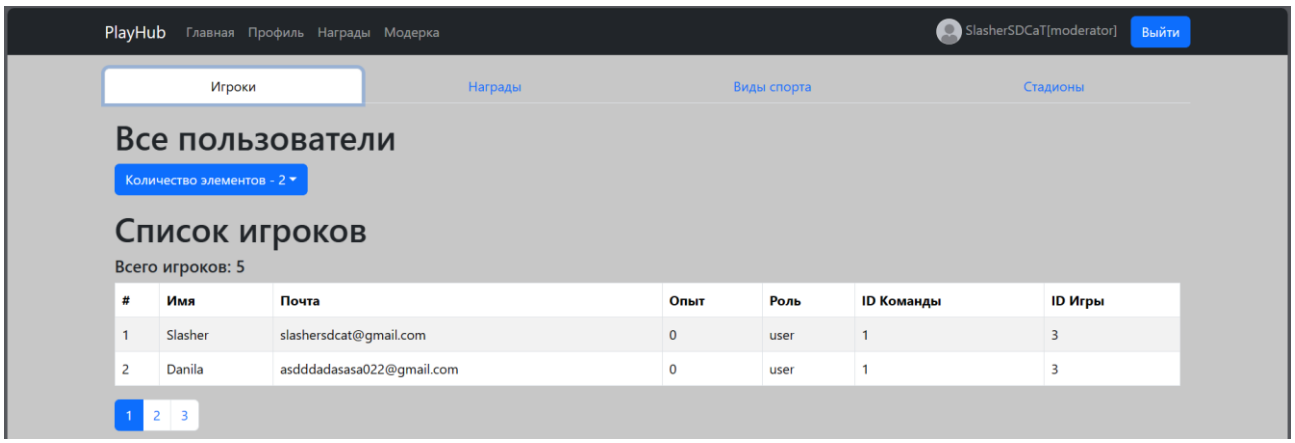

Рисунок 6.2.8 – Страница модерации пользователей

Вывод по разделу 6

Этот раздел представляет основные страницы, разработанные в клиентской части приложения, построенной на фреймворке React. Пользовательский интерфейс демонстрирует динамичные возможности, обеспечивая пользователя информацией о виде спорта, стадионах, играх, профиле и наградах. Для модераторов предусмотрены специальные страницы для управления содержимым, включая стадионы, виды спорта, награды и пользователей. Такой подход позволяет пользователям удобно взаимодействовать с приложением, а модераторам — эффективно управлять содержимым.

7 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

7.1 Тестирование приложения с помощью Insomnia

Для тестирования приложения в Fast Api автоматически создается документация Swagger. Swagger представляет собой инструмент, который помогает в описании, создании и визуализации интерактивной документации для веб-сервисов RESTful API. Это набор инструментов, который позволяет разработчикам описывать структуру и функциональность API с помощью спецификации OpenAPI (ранее известной как Swagger Specification).

Основная цель Swagger - предоставить разработчикам и пользователям удобный способ ознакомиться с функциональностью API, его методами, параметрами и структурой данных, не заглядывая в исходный код. Визуализация через Swagger обеспечивает возможность протестировать каждый API-метод непосредственно в браузере, что упрощает процесс тестирования и понимания работы API.

На рисунках 7.1.1-7.1.4 показаны сгенерированные эндпоинты приложения.

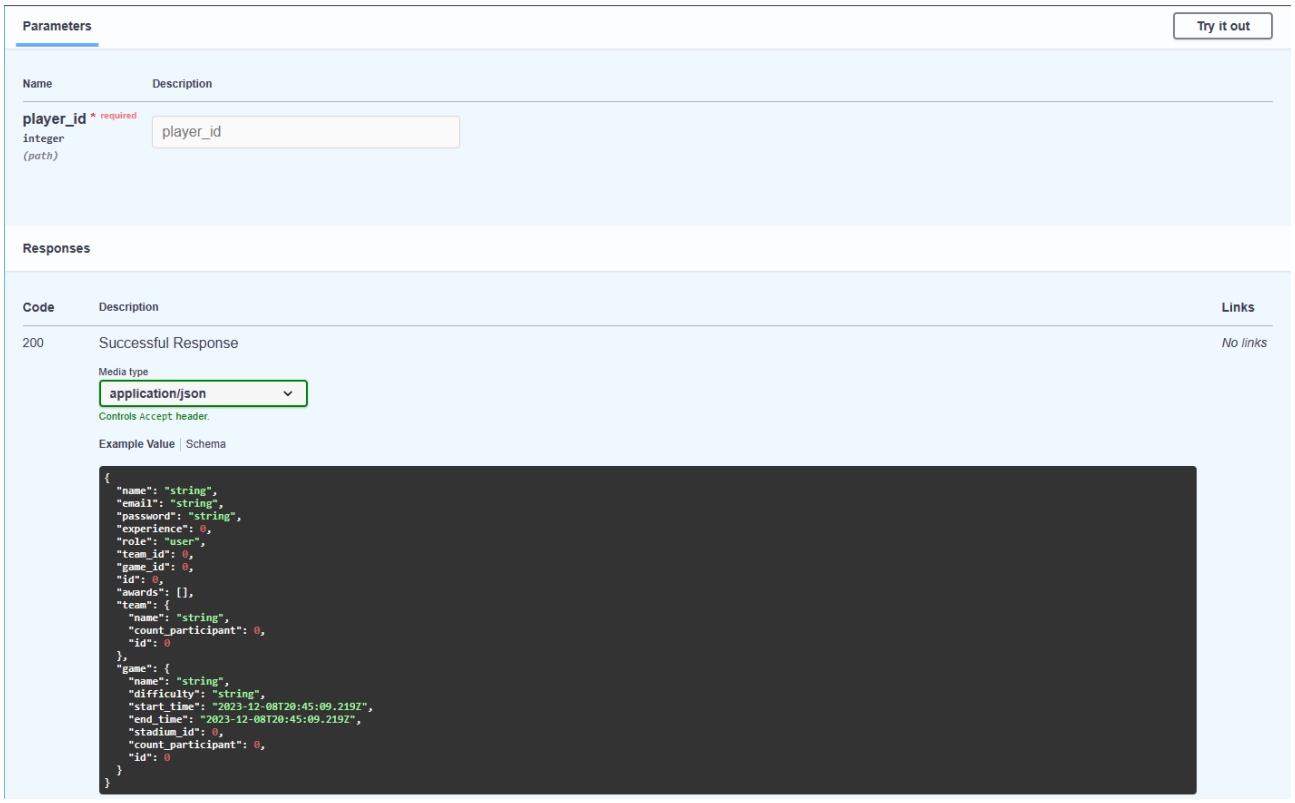

Рисунок 7.1.1 – структура отправляемого и получаемого запроса

Рисунок 7.1.2 – эндпоинты приложения (часть 1)

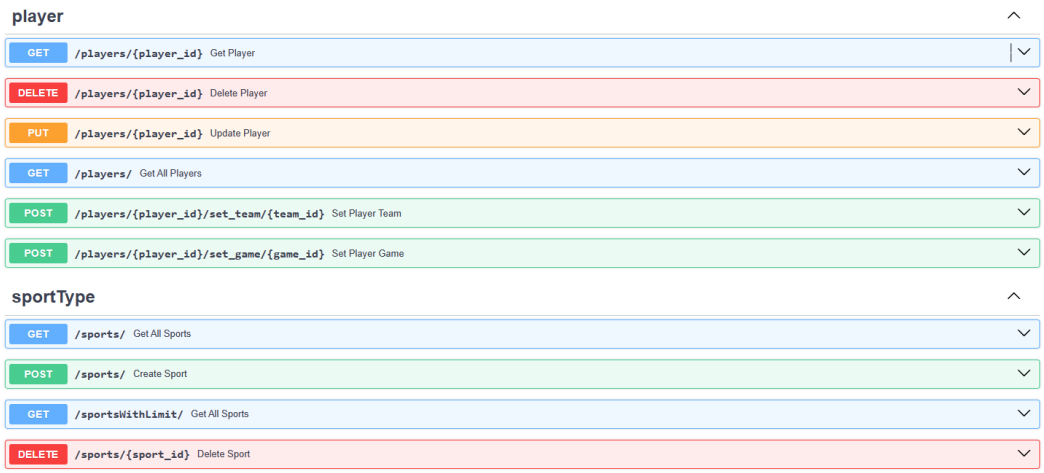

Рисунок 7.1.3 – эндпоинты приложения (часть 2)

Рисунок 7.1.4 – эндпоинты приложения (часть 3)

Каждый эндпоинт был протестирован при помощи данной документации. Пример тестирования показан на рисунках 7.1.5-7.1.6

Рисунок 7.1.5 – тестирование добавления награды

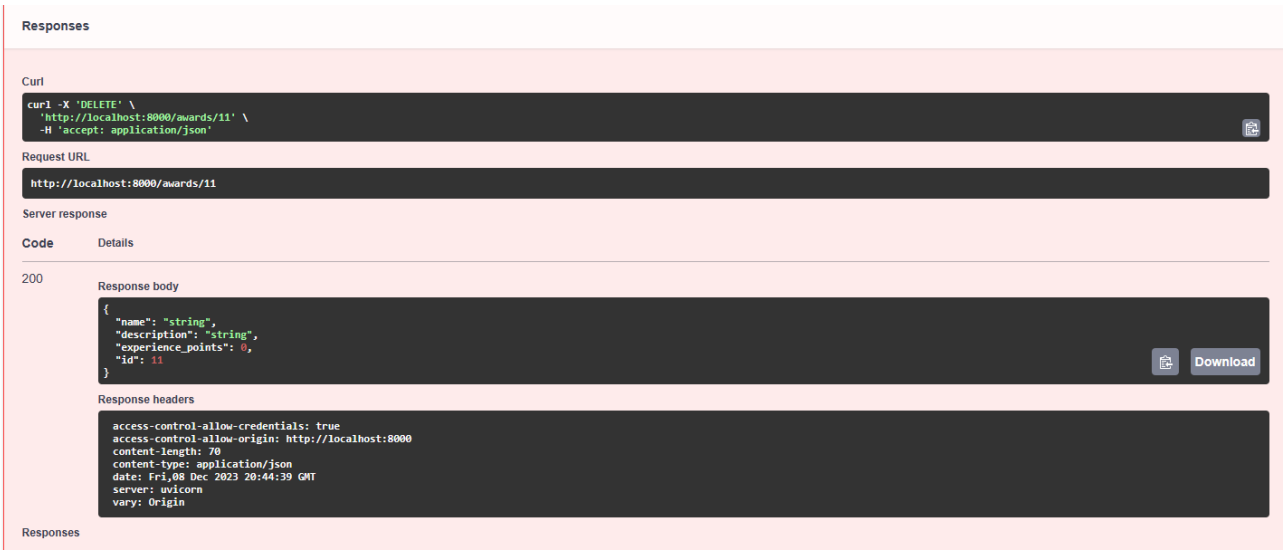

Рисунок 7.1.6 – тестирование удаления награды

Вывод по разделу 7

Swagger является мощным инструментом для описания, визуализации и тестирования API, автоматически создаваемым в FastAPI. Этот инструментарий позволяет не только разработчикам, но и пользователям удобно ознакомиться с функциональностью API, изучить методы, параметры запросов и структуру данных без необходимости просмотра исходного кода. Визуализация через Swagger облегчает процесс тестирования, позволяя непосредственно в браузере

проверить каждый метод API. Рисунки в разделе демонстрируют структуру отправляемых и получаемых запросов, а также представляют эндпоинты приложения, которые были протестированы с использованием этой документации.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была освоена компетенция «Разработка серверных частей интернет ресурсов» в степени соответствующей рабочей программной дисциплины. Результатом выполнения курсовой работы на тему "Веб-сервис по организации спортивных мероприятий" стало разработанное серверное программное приложение, которое написано с использованием языка Python и фреймворка Fast API, включая SQLAcademy и Pydantic. Для создания серверного приложения требовались навыки отличного владения и знания всех нюансов среды разработки, в которой написана работа, поэтому в ходе ее выполнения приобретены навыки работы в VS Code. Изучен фреймворк Fast API, который предназначен для разработки приложений на языке Python. Он предоставляет широкий набор инструментов и библиотек для упрощения процесса разработки, повышения производительности и улучшения масштабируемости приложений. Также изучена библиотека JavaScript – React. Она позволяет создавать компоненты, которые описывают структуру и внешний вид элементов интерфейса, и управлять ими динамически.

Благодаря глубокому анализу предметной области, учтены и проработаны все недочеты веб-ресурсов с похожей тематикой. Разработанный веб-ресурс содержит необходимые элементы, такие как изображения, текст и визуальное оформление, которые совместимы между собой и обеспечивают единообразие стилей и шрифтов на всех страницах, а также имеет всю функциональность.

Учитывая все вышеперечисленные пункты, можно с уверенностью сказать, что все требования, поставленные в задании на курсовую работу, соблюдены и выполнены успешно, цель работы достигнута. Разработанное приложение является полноценным и функциональным инструментом для отслеживания успеваемости студентов.

Исходный код Интернет-ресурса по курсовой работе доступен по ссылке: https://github.com/SlasherSDCaT/PlayHub

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Документация FastAPI - URL: https://fastapi.tiangolo.com/ - (Дата обращения: 01.12.2023)
2. Руководство по использованию React - URL: https://docs.sqlalchemy.org/en/ - (Дата обращения: 01.12.2023)
3. Документация PostgreSQL - URL: https://realpython.com/fastapi-python-web-apis/ - (Дата обращения: 01.12.2023)
4. Руководство по созданию веб-приложений с использованием Python и FastAPI - URL: https://towardsdatascience.com/fastapi-crud-restful-apis-with-sqlalchemy-postgresql-a32d84f9ff06 - (Дата обращения: 01.12.2023)
5. Руководство по React Bootstrap - URL: https://react-bootstrap.github.io/getting-started/introduction/ - (Дата обращения: 01.12.2023)
6. Статья на Habr "Основы React" - URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started - (Дата обращения: 01.12.2023)
7. Статья на Medium "Работа с PostgreSQL в Python" - URL: https://dev.to/USERNAME/WORKING_WITH_POSTGRESQL_PYTHON - (Дата обращения: 01.12.2023)
8. Книга "Разработка веб-приложений на Python с использованием Flask и React" - Автор: Иван Иванов - Год: 2022 - Издательство: Издательский дом X - ISBN: 1234567890 - (Дата обращения: 01.12.2023)
9. Курс "Основы работы с React и создание RESTful API с использованием FastAPI" - URL: https://www.udemy.com/react-fastapi-course - (Дата обращения: 01.12.2023)
10. Статья на Stack Overflow "Как интегрировать React с FastAPI" - URL: https://stackoverflow.com/questions/REACT_FASTAPI_INTEGRATION - (Дата обращения: 01.12.2023)
11. Документация Python - URL: https://docs.python.org/3/ - (Дата обращения: 01.12.2023)

12. Руководство по использованию SQLAlchemy с Python - URL: https://docs.sqlalchemy.org/en/ - (Дата обращения: 01.12.2023)

13. Статья на Real Python "Основы FastAPI: Руководство для начинающих" - URL: https://realpython.com/fastapi-python-web-apis/ - (Дата обращения: 01.12.2023)

14. Статья на Towards Data Science "Как создать REST API с помощью FastAPI" - URL: https://towardsdatascience.com/fastapi-crud-restful-apis-with-sqlalchemy-postgresql-a32d84f9ff06 - (Дата обращения: 01.12.2023)

15. Документация React Bootstrap - URL: https://react-bootstrap.github.io/getting-started/introduction/ - (Дата обращения: 01.12.2023)

16. Статья на MDN Web Docs "Основы React: Руководство для начинающих" - URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started - (Дата обращения: 01.12.2023)

17. Статья на dev.to "Работа с PostgreSQL и Python" - URL: https://dev.to/USERNAME/WORKING_WITH_POSTGRESQL_PYTHON - (Дата обращения: 01.12.2023)

18. Книга "Flask by Example" - Автор: Gareth Dwyer - Год: 2021 - Издательство: O'Reilly Media - ISBN: 1234567890 - (Дата обращения: 01.12.2023)

19. Курс на Pluralsight "Python API Development with FastAPI" - URL: https://www.pluralsight.com/courses/python-api-development-fastapi - (Дата обращения: 01.12.2023)

20. Статья на GitHub "Best Practices for React Development" - URL: https://github.com/USERNAME/react-best-practices - (Дата обращения: 01.12.2023)