МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение   
высшего образования «Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики и кибернетики

Кафедра технической кибернетики

**ОТЧЕТ**

по лабораторным работам

Дисциплина “ Технологии сетевого программирования”

**Проектирование приложения**

Выполнила студент  
группы 6302-010302D

Коршунова М. Ю.

САМАРА 2025

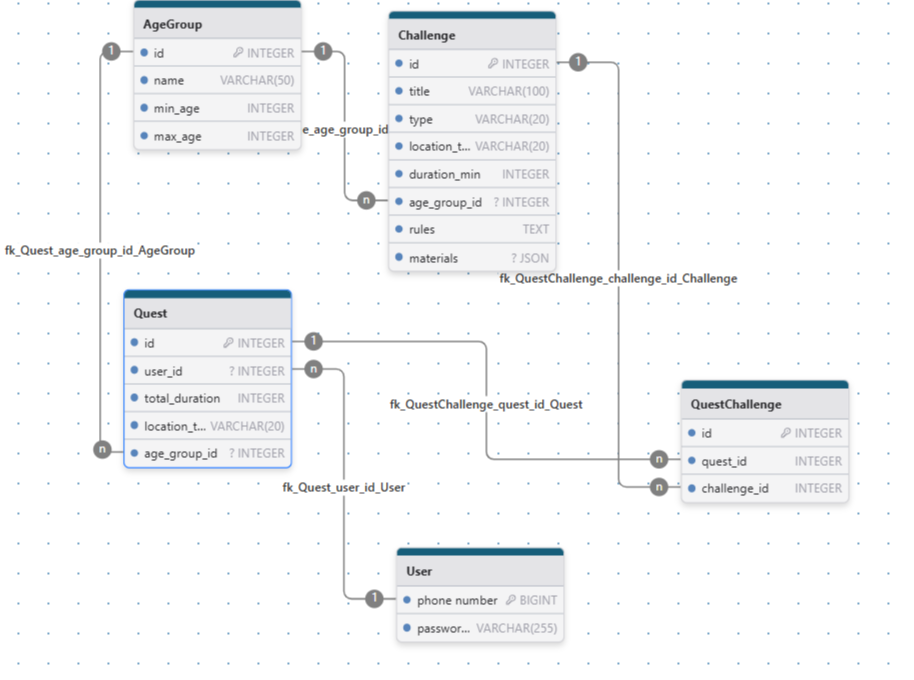
### Лабораторная 0. Проектирование приложения

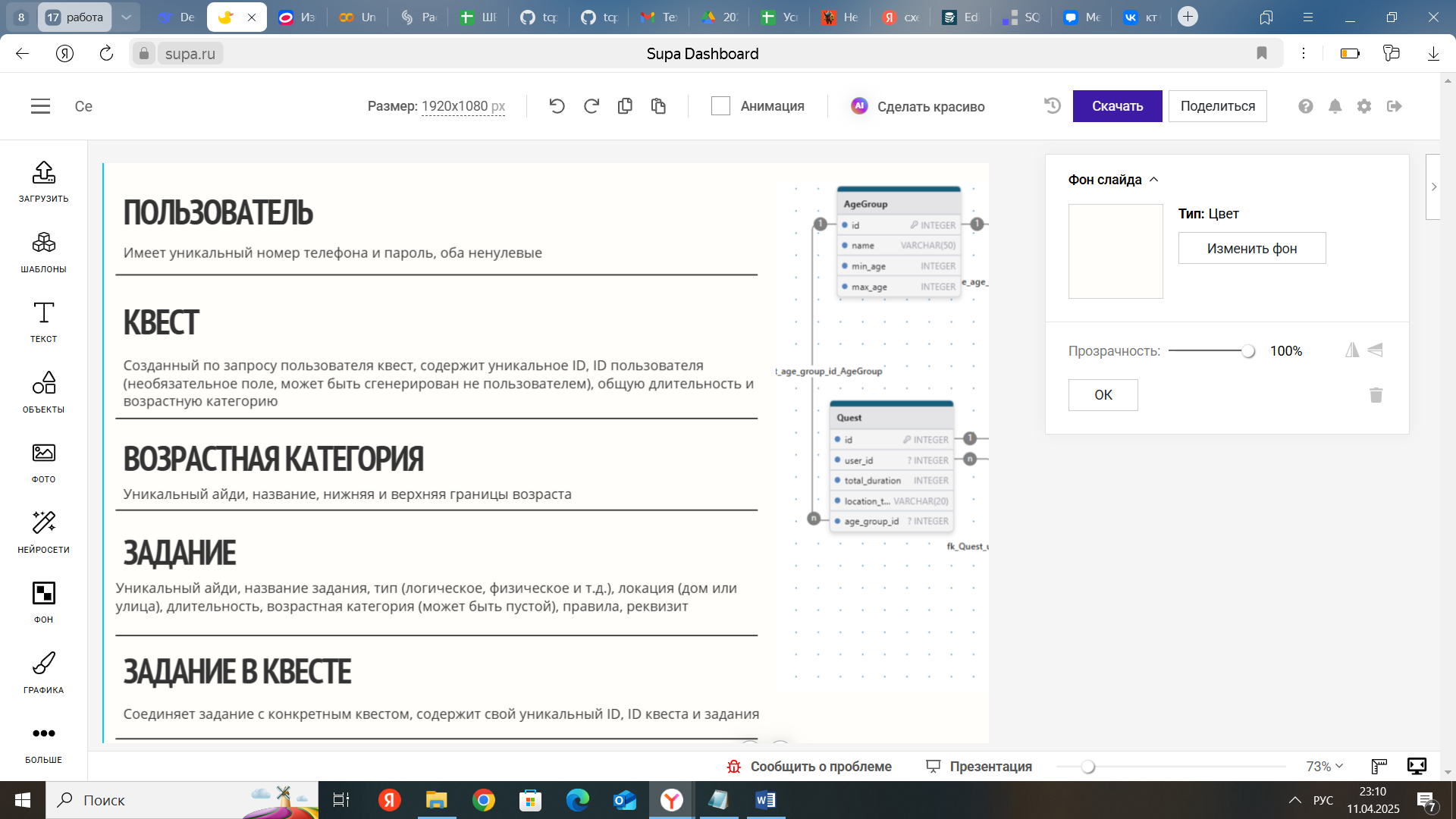
**Приложение для генерации квестов из банка заданий**

Я пытаюсь реализовать сайт, на котором будет набор конкурсов для квестов. Пользователь будет заходить на этот сайт, указывать, сколько конкурсов он хочет, на улице или в помещении будет квест, сколько заданий какого типа он хочет, (например, 5 умственных, 3 физических, 2 веселых), для какого возраста квест, его желаемую длительность. Можно указать не все факторы, а создать рандомный квест, указав лишь длительность. Сайт в ответ ему выдает набор заданий, со всеми нужными ссылками, правилами, лайфхаками и компонует их в нужном порядке. То есть, дает готовый квест. Все задания для квеста хранятся в базе данных.

**Схема взаимодействия компонентов (сервер, БД, клиент):  
**

**Схема базы данных:**

****



**Структура API:**

Предварительно условимся, что сайт для проекта будет называться morekvestov.ru. На основе этого приведем примеры URL запросов. Наш проект будет общаться с сервером при помощи архитектуры REST, и мы планируем, что он будет использовать такие методы:

Метод **GET** для получения информации о конкретном квесте (уже созданном), либо о конкретном задании. Конечная точка URL запроса будет

выглядеть как например morekvestov.ru/quests/{quest\_id} для квеста, либо для заданий morekvestov.ru/challenges с использованием фильтров, например ?type=mental, ?location=indoor, ?age\_min=12, ?duration\_max=15. В формате ответа ожидаем искомые задания.

Метод **POST** будем использовать для создания нового квеста. Конечная точка формируется аналогичным **GET** образом. В теле запроса будет содержаться название нового квеста в текстовом формате, желаемые типы задач и количество каждого типа, место квеста, возрастная категория, длительность. Все части опциональны. В формате ответа от сервера ожидаем HTML страничку с собранным квестом.

Возможно, будет использован метод **PUT** позволит отредактировать отдельное задание, добавить правила или материалы. Конечная точка формируется аналогичным **GET** образом. В теле запроса будут содержаться изменения в текстовом формате, в формате ответа от сервера ожидаем HTML страничку с примененными изменениями.

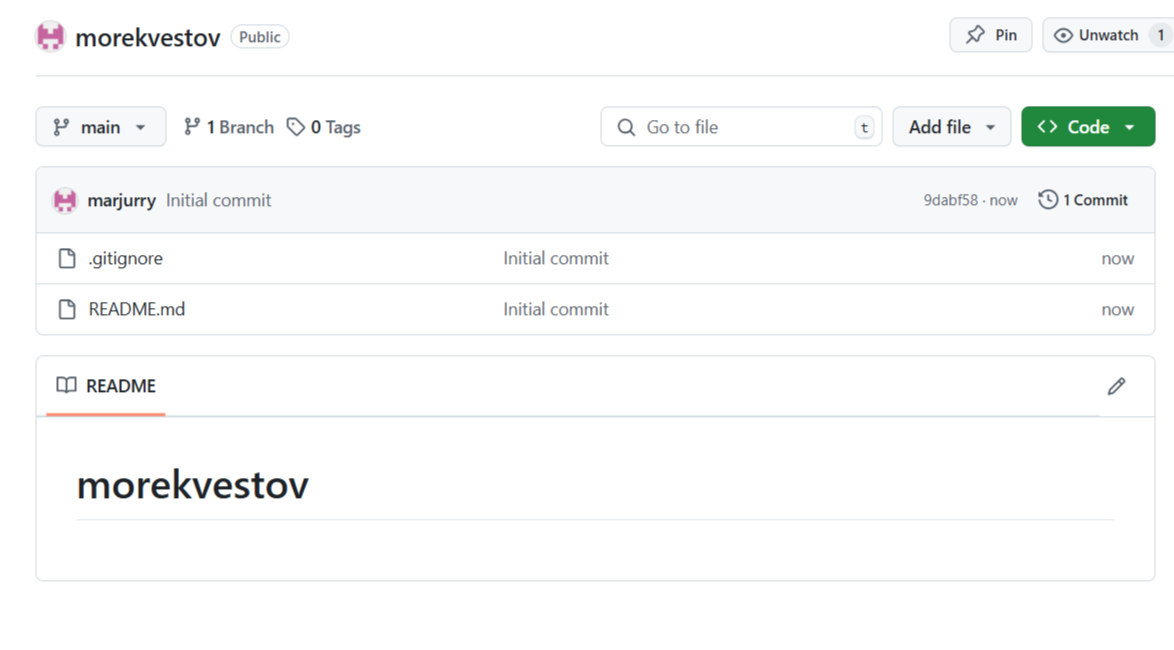
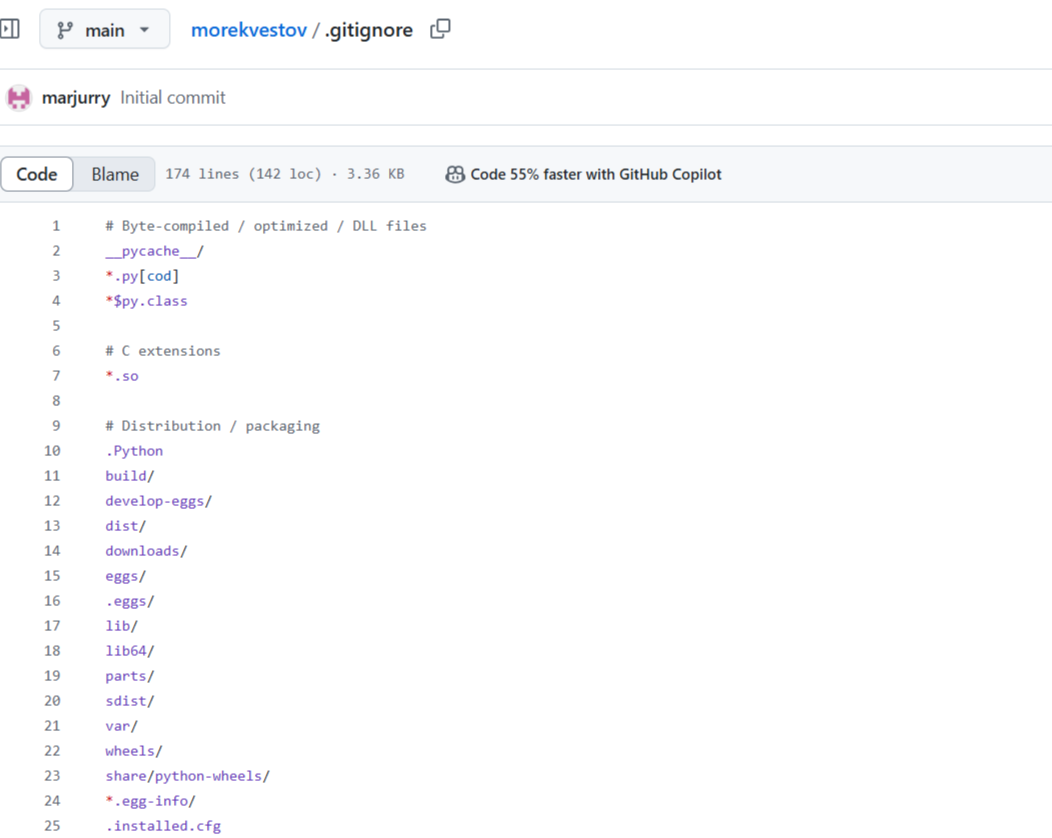
**Стек технологий:**

Веб: HTML и CSS, JS, REST

СУБД: PostgeSQL

Средство работы с БД: SQLAlchemy

Коммуникация: github

### Лабораторная 1. Разработка базы данных

**Основные сущности**

**1. User (Пользователь)**

* **Поля**:

phone\_number (BigInteger) - первичный ключ, используется как идентификатор пользователя

password\_hash (String) - хеш пароля пользователя

is\_active (Boolean) - флаг активности пользователя (по умолчанию True)

* **Функционал**:

Методы для хеширования (set\_password) и проверки паролей (check\_password) с использованием pbkdf2\_sha256

Связь один-ко-многим с Quest (один пользователь может иметь много квестов)

**2. AgeGroup (Возрастная группа)**

* **Поля**:

id (Integer) - первичный ключ

name (String) - название возрастной группы

min\_age, max\_age (Integer) - границы возраста

* **Связи**:

Один-ко-многим с Challenge (одна возрастная группа может содержать много заданий)

Один-ко-многим с Quest (одна возрастная группа может быть у многих квестов)

**3. Challenge (Задание)**

* **Поля**:

id (Integer) - первичный ключ

title (String) - название задания

type (String) - тип задания

location\_type (String) - тип локации (например, "улица", "дом" и т.д.)

duration\_min (Integer) - продолжительность задания в минутах

rules (Text) - правила выполнения задания

* **Связи**:

Многие-к-одному с AgeGroup (каждое задание принадлежит одной возрастной группе)

Многие-ко-многим с Quest через ассоциативную таблицу quest\_challenge\_association

**4. Quest (Квест)**

* **Поля**:

id (Integer) - первичный ключ

total\_duration (Integer) - общая продолжительность квеста

location\_type (String) - тип локации для квеста

* **Связи**:

Многие-к-одному с User (каждый квест принадлежит одному пользователю)

Многие-к-одному с AgeGroup (каждый квест предназначен для одной возрастной группы)

Многие-ко-многим с Challenge через ассоциативную таблицу

**Ассоциативная таблица**

quest\_challenge\_association - это таблица для связи многие-ко-многим между Quest и Challenge, она содержит два внешних ключа: quest\_id и challenge\_id, оба ключа образуют составной первичный ключ, используется каскадное удаление при удалении связанных записей (ondelete='CASCADE')

*# 1. Create challenges*

    challenge\_types = ["интеллектуальный", "творческий", "спортивный", "социальный"]

    location\_types = ["дома", "на улице", "везде"]

    challenges = []

    challenge\_data = [

    {

        "id": 1,

        "title": "Шифр",

        "type": random.choice(challenge\_types),

        "location\_type": random.choice(location\_types),

        "duration\_min": random.randint(5, 60),

        "age\_group\_id": random.choice([g.id for g in created\_age\_groups]),

        "rules": "Подробные правила для челленджа 1",

    },

    {

        "id": 2,

        "title": "Квиз",

        "type": random.choice(challenge\_types),

        "location\_type": random.choice(location\_types),

        "duration\_min": random.randint(5, 60),

        "age\_group\_id": random.choice([g.id for g in created\_age\_groups]),

        "rules": "Подробные правила для челленджа 2"

    },

    {

        "id": 3,

        "title": "Прятки",

        "type": random.choice(challenge\_types),

        "location\_type": random.choice(location\_types),

        "duration\_min": random.randint(5, 60),

        "age\_group\_id": random.choice([g.id for g in created\_age\_groups]),

        "rules": "Подробные правила для челленджа 3"

    },

    {

        "id": 4,

        "title": "Догонялки",

        "type": random.choice(challenge\_types),

        "location\_type": random.choice(location\_types),

        "duration\_min": random.randint(5, 60),

        "age\_group\_id": random.choice([g.id for g in created\_age\_groups]),

        "rules": "Подробные правила для челленджа 4"

    }

]

*# Create challenges and save real objects*

    for data in challenge\_data:

        challenge = create\_challenge(db, \*\*data)

        challenges.append(challenge)

    print(**f**"✅ Создано {len(challenges)} челленджей")

*# 2. Create quests*

    quests = []

    quest\_data = [

    {

        "id": 1,

        "user\_id": created\_users[0].phone\_number,  *# Using existing user*

        "total\_duration": random.randint(30, 180),

        "location\_type": random.choice(location\_types),

        "age\_group\_id": random.choice([g.id for g in created\_age\_groups]),

        "challenge\_ids": [challenges[0].id, challenges[1].id]  *# Pre-selected challenges*

    },

    {

        "id": 2,

        "user\_id": created\_users[1].phone\_number if len(created\_users) > 1 else created\_users[0].phone\_number,

        "total\_duration": random.randint(30, 180),

        "location\_type": random.choice(location\_types),

        "age\_group\_id": random.choice([g.id for g in created\_age\_groups]),

        "challenge\_ids": [challenges[2].id]  *# Pre-selected challenges*

    }

    ]

*# Create quests with challenges*

    for data in quest\_data:

*# Extract challenge\_ids if present*

        challenge\_ids = data.pop("challenge\_ids", [])

*# Create quest*

        quest = create\_quest(db, \*\*data)

*# Add challenges to quest*

        if challenge\_ids:

            set\_quest\_challenges(db, quest.id, challenge\_ids)

            print(**f**"➡️ Добавлены челленджи {challenge\_ids} в квест {quest.id}")

        quests.append(quest)

        print(**f**"✅ Создан квест ID: {quest.id}")

*# 3. Optionally add random challenges to some quests*

    for quest in quests:

        if random.choice([True, False]):  *# 50% chance to add more challenges*

            available\_challenges = [c for c in challenges if c.id not in [ch.id for ch in quest.challenges]]

            if available\_challenges:

                challenge\_to\_add = random.choice(available\_challenges)

                add\_challenge\_to\_quest(db, quest.id, challenge\_to\_add.id)

                print(**f**"➡️ Дополнительно добавлен челлендж {challenge\_to\_add.id} в квест {quest.id}")

    print(**f**"✅ Всего создано {len(quests)} квестов")

    print("✅ Тестовые данные созданы")

    return {

    "users": created\_users,

    "age\_groups": created\_age\_groups,

    "challenges": challenges,

    "quests": quests

    }

**def** test\_user\_crud(db: Session, test\_user: models.User):

    """Тестирование CRUD операций для пользователей"""

    print("\n=== ТЕСТИРОВАНИЕ ПОЛЬЗОВАТЕЛЕЙ ===")

*# Тест создания*

    new\_user = create\_user(db, phone\_number=9998877, password="newuserpass")

    print(**f**"Создан новый пользователь: {new\_user.phone\_number}")

*# Тест чтения*

    db\_user = get\_user(db, test\_user.phone\_number)

    print(**f**"Найден пользователь: {db\_user.phone\_number if db\_user else 'не найден'}")

*# Тест обновления*

*#updated\_user = update\_user(db, test\_user.phone\_number, {"phone\_number": 12243})*

*#print(f"Обновлен номер: {test\_user.phone\_number} -> {updated\_user.phone\_number}")*

*# Тест удаления*

    if delete\_user(db, new\_user.phone\_number):

        print(**f**"Пользователь {new\_user.phone\_number} удален")

**def** test\_age\_group\_crud(db: Session, test\_age\_group: models.AgeGroup):

    """Тестирование CRUD операций для возрастных групп"""

    print("\n=== ТЕСТИРОВАНИЕ ВОЗРАСТНЫХ ГРУПП ===")

*# Тест создания*

    new\_group = create\_age\_group(db, id=4, name="Новая группа", min\_age=25, max\_age=40)

    print(**f**"Создана новая возрастная группа: {new\_group.name}")

*# Тест чтения*

    db\_group = get\_age\_group(db, test\_age\_group.id)

    print(**f**"Найдена группа: {db\_group.name if db\_group else 'не найдена'}")

*# Тест обновления*

    updated\_group = update\_age\_group(db, new\_group.id, {"name": "Обновленная группа", "max\_age": 50})

    print(**f**"Обновлена группа: ID {updated\_group.id}, новое имя: {updated\_group.name}")

*# Тест фильтрации*

    suitable\_groups = get\_age\_groups\_by\_age(db, age=15)

    print(**f**"Группы для возраста 15: {[g.name for g in suitable\_groups]}")

*# Тест удаления*

    if delete\_age\_group(db, new\_group.id):

        print(**f**"Группа {new\_group.name} удалена")

**def** test\_challenge\_crud(db: Session, test\_challenge: models.Challenge):

    """Тестирование CRUD операций для челленджей"""

    print("\n=== ТЕСТИРОВАНИЕ ЧЕЛЛЕНДЖЕЙ ===")

*# Тест создания*

    new\_challenge = create\_challenge(

        db,

        id=6,

        title="Новый челлендж",

        type="спортивный",

        location\_type="на улице",

        duration\_min=30,

        age\_group\_id=test\_challenge.age\_group\_id,

        rules="Новые правила"

    )

    print(**f**"Создан новый челлендж: {new\_challenge.title}")

*# Тест чтения*

    db\_challenge = get\_challenge(db, test\_challenge.id)

    print(**f**"Найден челлендж: {db\_challenge.title if db\_challenge else 'не найден'}")

*# Тест получения связанных квестов*

    challenge\_quests = get\_challenge\_quests(db, test\_challenge.id)

    print(**f**"Челлендж используется в квестах: {len(challenge\_quests)}")

*# Тест обновления*

    updated\_challenge = update\_challenge(db, new\_challenge.id, {"title": "Обновленный челлендж", "duration\_min": 45})

    print(**f**"Обновлен челлендж: {updated\_challenge.title}, длительность: {updated\_challenge.duration\_min} мин")

*# Тест фильтрации*

    filtered\_challenges = get\_challenges\_by\_filters(

        db,

        age\_group\_id=test\_challenge.age\_group\_id,

        challenge\_type=test\_challenge.type,

        location\_type=test\_challenge.location\_type

    )

    print(**f**"Найдено челленджей по фильтру: {len(filtered\_challenges)}")

*# Тест удаления (проверяем каскадное удаление связей)*

    if delete\_challenge(db, new\_challenge.id):

        print(**f**"Челлендж {new\_challenge.title} удален")

*# Проверяем что связи удалились*

        remaining\_quests = get\_challenge\_quests(db, new\_challenge.id)

*#print(f"Осталось связей с квестами: {len(remaining\_quests)} (должно быть 0)")*

**def** test\_quest\_crud(db: Session, test\_quest: models.Quest, test\_challenges: List[models.Challenge]):

    """Тестирование CRUD операций для квестов"""

    print("\n=== ТЕСТИРОВАНИЕ КВЕСТОВ ===")

*# Тест создания с сразу заданными челленджами*

    new\_quest = create\_quest(

        db,

        id=5,

        user\_id=test\_quest.user\_id,

        total\_duration=90,

        location\_type="везде",

        age\_group\_id=test\_quest.age\_group\_id

    )

    print(**f**"Создан новый квест: ID {new\_quest.id}")

*# Добавляем челленджи (3 случайных)*

    selected\_challenges = random.sample(test\_challenges, min(3, len(test\_challenges)))

    for challenge in selected\_challenges:

        add\_challenge\_to\_quest(db, new\_quest.id, challenge.id)

    print(**f**"Добавлено челленджей в квест: {len(selected\_challenges)}")

*# Тест чтения*

    db\_quest = get\_quest(db, test\_quest.id)

    print(**f**"Найден квест: ID {db\_quest.id if db\_quest else 'не найден'}")

*# Тест получения челленджей*

    quest\_challenges = get\_quest\_challenges(db, new\_quest.id)

    print(**f**"Квест содержит челленджей: {len(quest\_challenges)}")

    if hasattr(new\_quest, 'challenges') and new\_quest.challenges:

        print("\nЧелленджи:")

        for challenge in new\_quest.challenges:

            print(**f**"- {challenge.title} (ID: {challenge.id})")

*# Тест массового обновления челленджей*

    if len(test\_challenges) > 1:

        new\_challenge\_ids = [test\_challenges[0].id, test\_challenges[-1].id]

        set\_quest\_challenges(db, new\_quest.id, new\_challenge\_ids)

        updated\_challenges = get\_quest\_challenges(db, new\_quest.id)

        print(**f**"Обновлено челленджей в квесте: {len(updated\_challenges)}")

*# Тест обновления параметров квеста*

    updated\_quest = update\_quest(db, new\_quest.id, {"total\_duration": 120})

    print(**f**"Обновлено время квеста: {updated\_quest.total\_duration} мин")

*# Тест удаления челленджа из квеста*

    if quest\_challenges:

        removed = remove\_challenge\_from\_quest(db, new\_quest.id, quest\_challenges[0].id)

        print(**f**"Челлендж удален из квеста: {'успешно' if removed else 'ошибка'}")

*# Тест удаления квеста (проверяем каскадное удаление связей)*

    if delete\_quest(db, new\_quest.id):

        print(**f**"Квест ID {new\_quest.id} удален")

*# Проверяем что связи удалились*

        remaining\_challenges = get\_quest\_challenges(db, new\_quest.id)

### Лабораторная 2. Разработка API

**Эндпоинты и примеры запросов**

#### **Создание пользователя:**

#### POST /users/ **Пример тела запроса:**

{

"phone\_number": 79001234567,

"password": "securepassword123"

}

**Успешный ответ (201 Created):**

{

"phone\_number": 79001234567,

"is\_active": true

}

400 Bad Request - если номер уже существует или данные невалидны

#### **Получение пользователя:**

#### GET /users/{phone\_number} **Пример ответа (200 OK):**

{

"phone\_number": 79001234567,

"is\_active": true

}

404 Not Found - если пользователь не найден

#### **Удаление пользователя:**

#### DELETE /users/{phone\_number} **Успешный ответ**: 204 No Content

#### **Создание возрастной группы:**

#### POST /age\_groups/ **Пример тела запроса:**

{

"name": "Дети 6-9 лет",

"min\_age": 6,

"max\_age": 9

}

**Успешный ответ (201 Created):**

{

"id": 1,

"name": "Дети 6-9 лет",

"min\_age": 6,

"max\_age": 9

}

#### **Получение групп по возрасту:**

#### GET /age\_groups/?age=8 **Пример ответа (200 OK):**

#### [

{

"id": 1,

"name": "Дети 6-9 лет",

"min\_age": 6,

"max\_age": 9

}

]

#### **Создание задания:**

#### POST /challenges/ **Пример тела запроса:**

{

"title": "Найди скрытый объект",

"type": "поиск",

"location\_type": "улица",

"duration\_min": 15,

"age\_group\_id": 1,

"rules": "Найдите 5 скрытых объектов в парке"

}

**Успешный ответ (201 Created)** - возвращает созданное задание

#### **Фильтрация заданий:**

#### GET /challenges/?age\_group\_id=1&location\_type=улица&min\_duration=10 **Пример ответа (200 OK):**

[

{

"id": 1,

"title": "Найди скрытый объект",

"type": "поиск",

"location\_type": "улица",

"duration\_min": 15,

"age\_group\_id": 1

}

]

#### **Создание квеста:**

#### POST /quests/ **Пример тела запроса:**

{

"user\_id": 79001234567,

"total\_duration": 60,

"location\_type": "улица",

"age\_group\_id": 1,

"challenge\_ids": [1, 2, 3]

}

**Успешный ответ (201 Created)** - возвращает созданный квест с заданиями

#### **Получение квестов пользователя:**

#### GET /quests/?user\_id=79001234567 **Пример ответа (200 OK):**

[

{

"id": 1,

"total\_duration": 60,

"location\_type": "улица",

"user\_id": 79001234567,

"age\_group\_id": 1,

"challenges": [

{

"id": 1,

"title": "Найди скрытый объект"

},

{

"id": 2,

"title": "Разгадай загадку"

}

]

}

]

#### **Добавление задания в квест:**

#### POST /quests/{quest\_id}/challenges/{challenge\_id} **Успешный ответ**: 204 No Content

#### **Получение заданий квеста:**

#### GET /quests/{quest\_id}/challenges **Пример ответа (200 OK):**

[

{

"id": 1,

"title": "Найди скрытый объект",

"type": "поиск",

"age\_group\_name": "Дети 6-9 лет"

}

]

### Лабораторная работа 3. Авторизация

Основные компоненты JWT аутентификации

**Секретный ключ и алгоритм**:

SECRET\_KEY = "1234"

ALGORITHM = "HS256"

ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30

**Схема аутентификации**:

security = HTTPBearer(auto\_error=False)

**Черный список токенов** (для реализации logout):

token\_blacklist = set()

**Функции для работы с токенами**:

create\_access\_token() - создает JWT токен

get\_current\_user() - проверяет токен и возвращает текущего пользователя

get\_current\_user\_optional() - опциональная версия для необязательной аутентификации

**Примеры запросов**

1. Регистрация пользователя

**Запрос**:

POST /auth/register HTTP/1.1

Content-Type: application/json

{

"phone\_number": 79123456789,

"password": "mysecretpassword"

}

**Успешный ответ**:

HTTP/1.1 201 Created

Content-Type: application/json

{

"message": "User registered successfully",

"user\_id": 79123456789

}

**Ошибка** (если пользователь уже существует):

HTTP/1.1 400 Bad Request

Content-Type: application/json

{

"detail": "Phone number already registered"

}

2. Вход в систему (получение JWT токена)

**Запрос**:

POST /auth/token HTTP/1.1

Content-Type: application/json

{

"phone\_number": 79123456789,

"password": "mysecretpassword"

}

**Успешный ответ**:

HTTP/1.1 200 OK

Content-Type: application/json

{

"access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI3OTEyMzQ1Njc4OSIsImV4cCI6MTY5MDAwMDAwMH0.abcdef1234567890",

"token\_type": "bearer"

}

3. Доступ к защищенному ресурсу

**Запрос**:

GET /auth/me HTTP/1.1

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI3OTEyMzQ1Njc4OSIsImV4cCI6MTY5MDAwMDAwMH0.abcdef1234567890

**Успешный ответ**:

HTTP/1.1 200 OK

Content-Type: application/json

{

"phone\_number": 79123456789,

"is\_active": true

}

**Ошибка** (недействительный токен):

HTTP/1.1 401 Unauthorized

Content-Type: application/json

{

"detail": "Invalid token"

}

4. Выход из системы (отзыв токена)

**Запрос**:

POST /auth/logout HTTP/1.1

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiI3OTEyMzQ1Njc4OSIsImV4cCI6MTY5MDAwMDAwMH0.abcdef1234567890

**Успешный ответ**:

HTTP/1.1 200 OK

Content-Type: application/json

{

"message": "Successfully logged out"

}

### Лабораторная работа 4. Разработка пользовательского интерфейса и взаимодействие с API

Структура интерфейса и логика работы API

Общая архитектура:

Приложение представляет собой веб-интерфейс для управления квестами и заданиями с JWT-аутентификацией через cookies. Основные компоненты:

1. Аутентификация (роутер `/auth`)

2. Управление пользователями (профиль, редактирование)

3. Работа с квестами (создание, просмотр, фильтрация)

4. Управление заданиями (аналогично квестам)

1. Аутентификация и авторизация

- `/auth/register` - регистрация через форму

- `/auth/login` - вход через форму

- `/auth/token` - генерация JWT токена

- `/auth/logout` - выход из системы

2. Пользовательский интерфейс

- `/` - главная страница (доступна без аутентификации)

- `/profile` - профиль пользователя (требует аутентификации)

- `/users/{phone\_number}/edit` - редактирование пользователя

3. Система квестов

- `/quests` - список квестов с фильтрами

- `/quests/new` - форма создания квеста

- `/quests/{quest\_id}` - детали квеста

- `/quests/{quest\_id}/delete` - удаление квеста

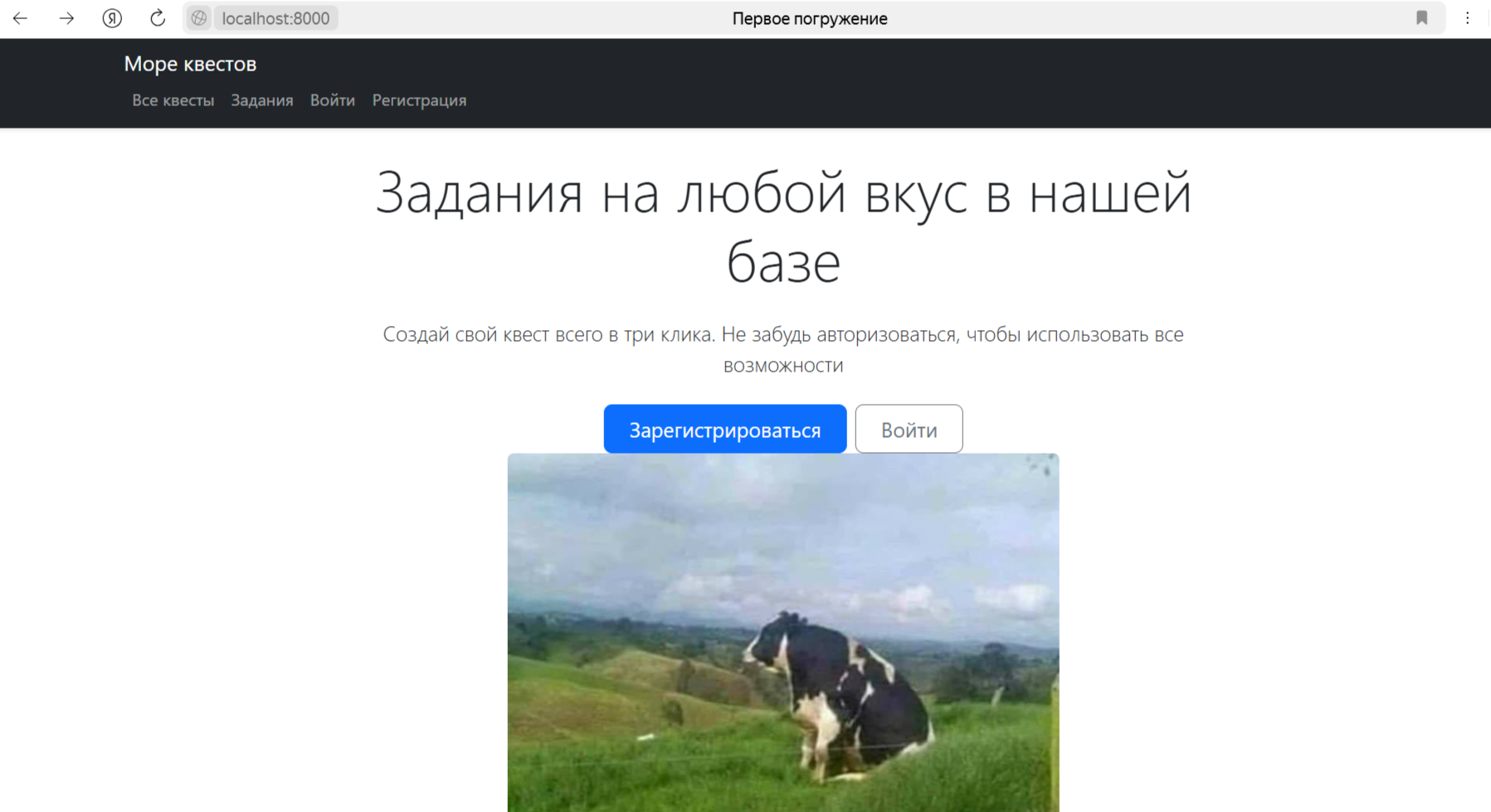
4. Система заданий

- `/challenges` - список заданий

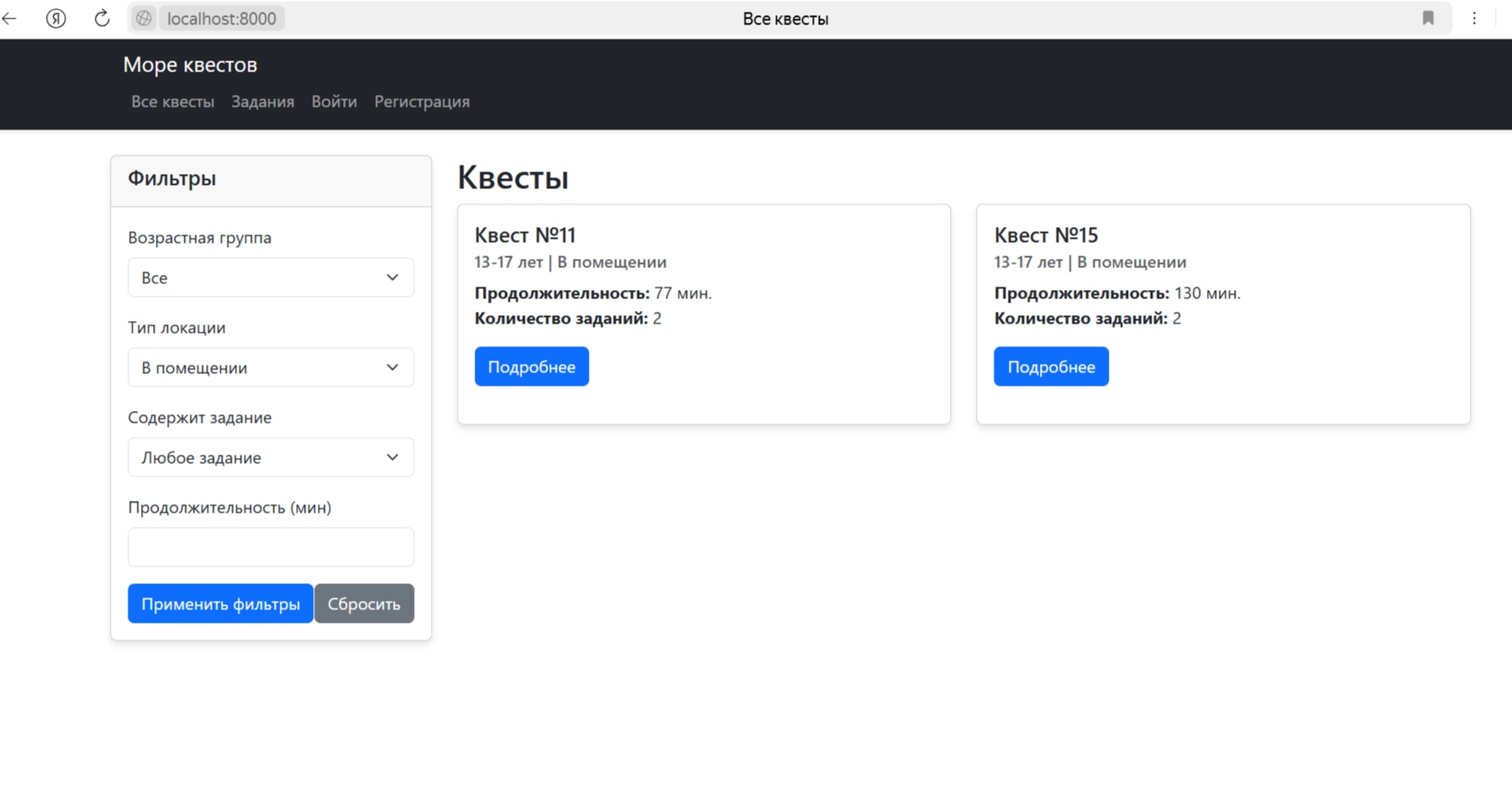
- `/challenges/create` - форма создания задания

- `/challenges/{challenge\_id}` - детали задания

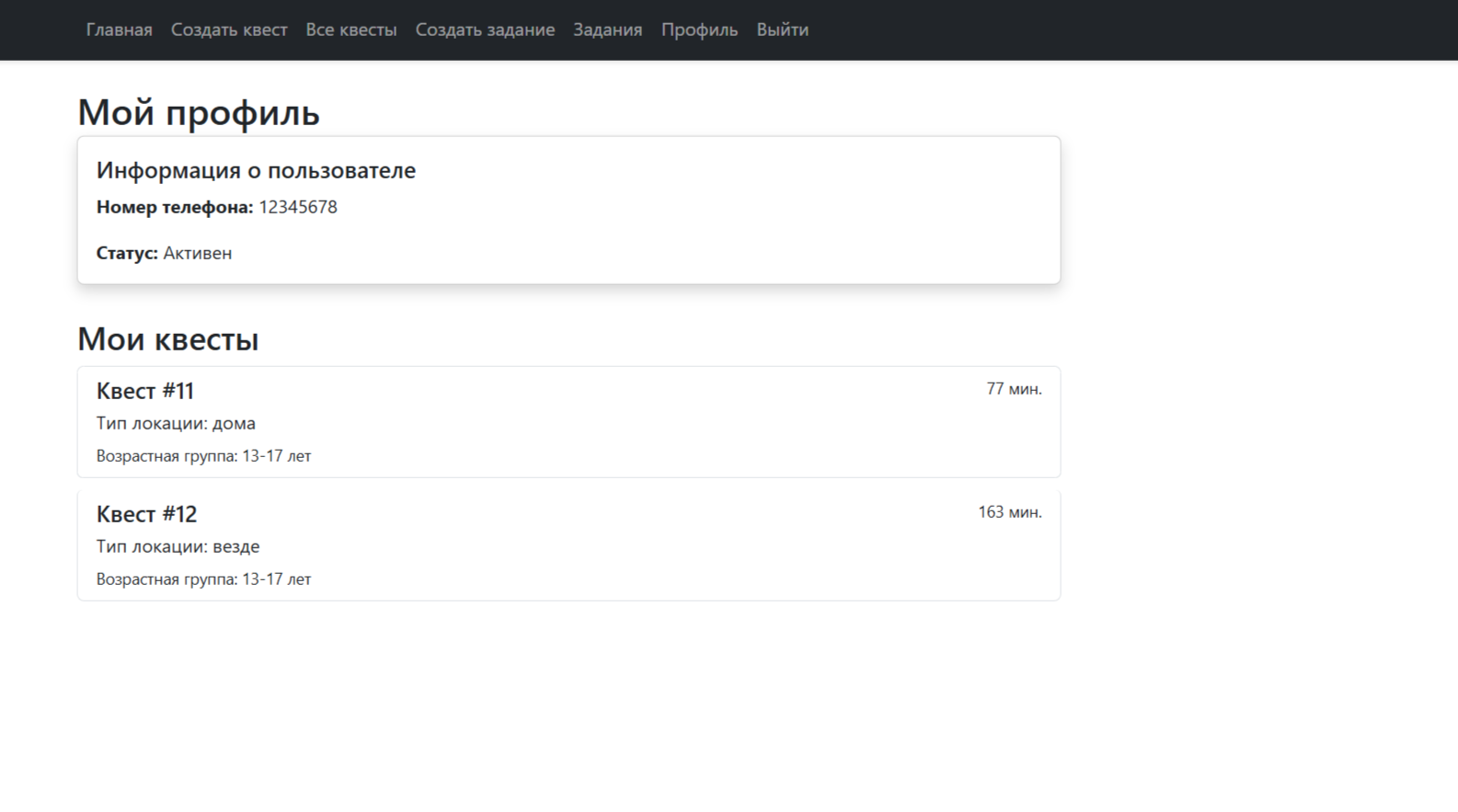
- `/challenges/{challenge\_id}/delete` - удаление задания



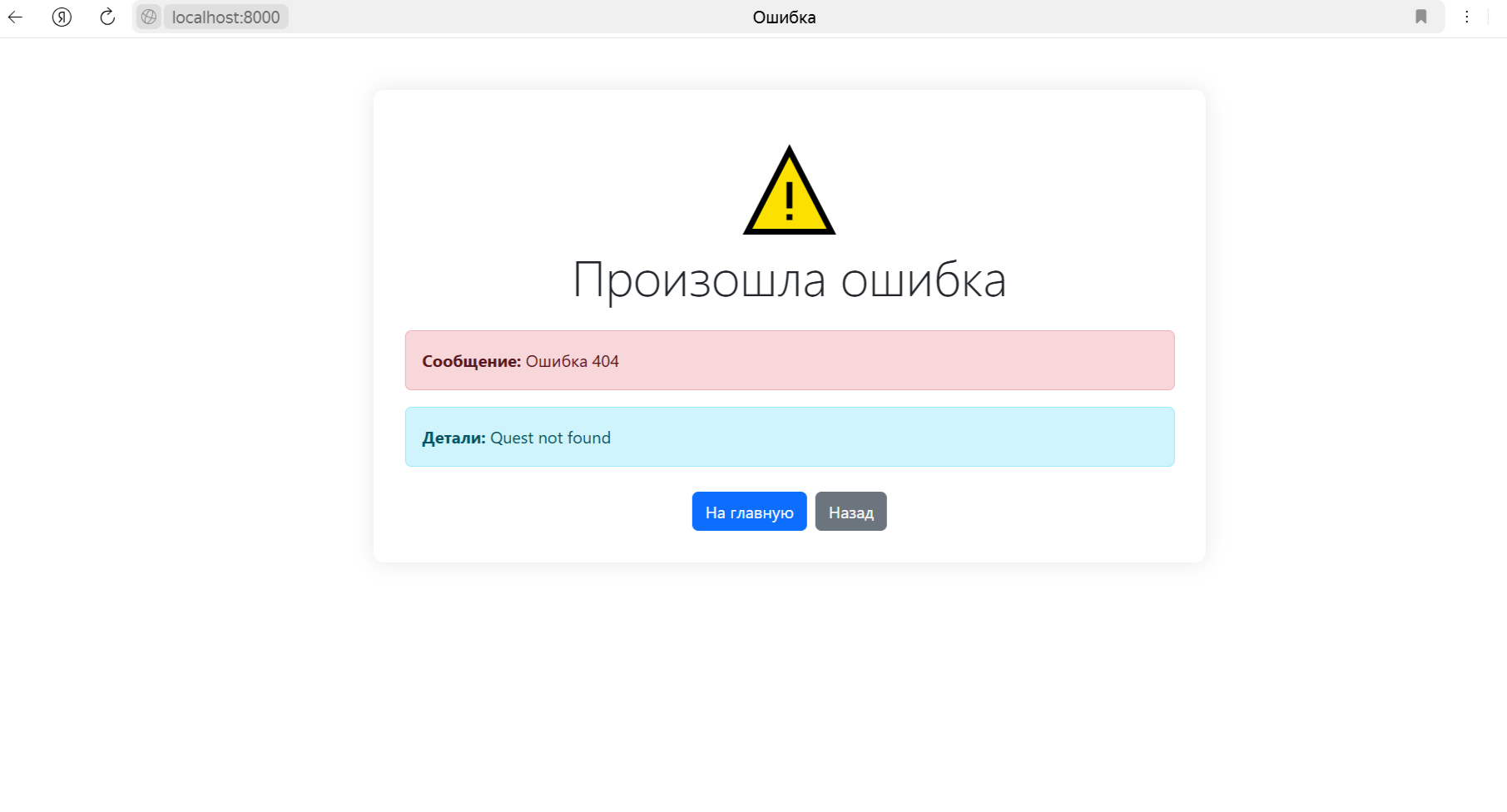
Главная страница



Есть фильтрация

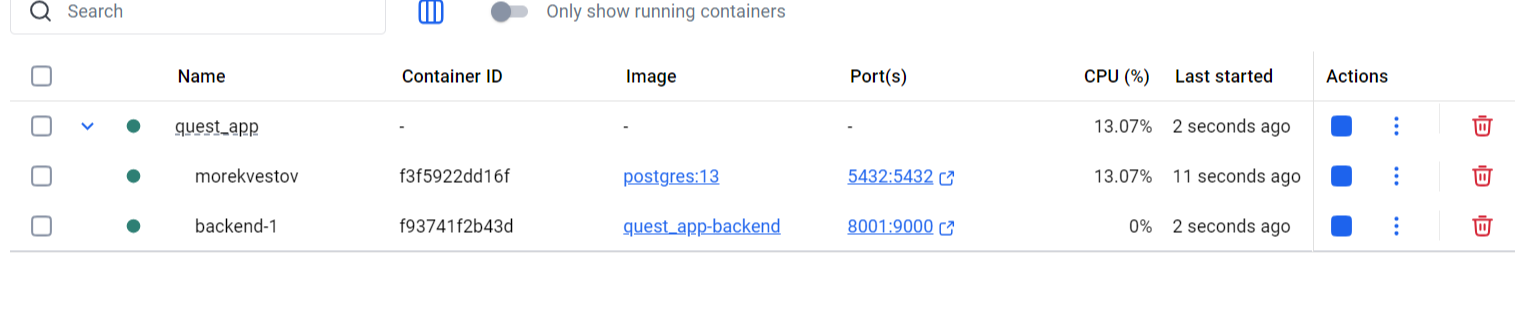


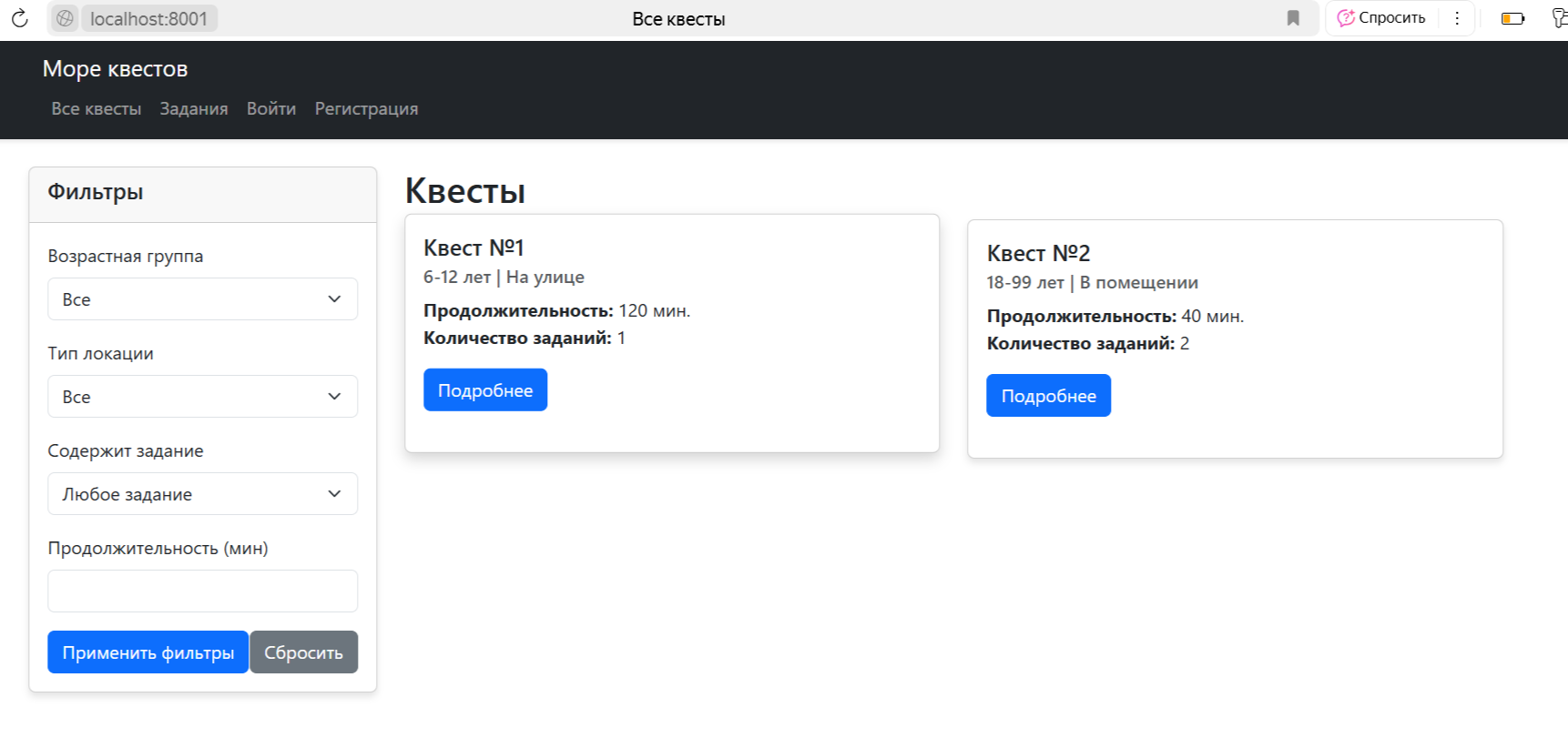
Есть привязка квеста к профилю

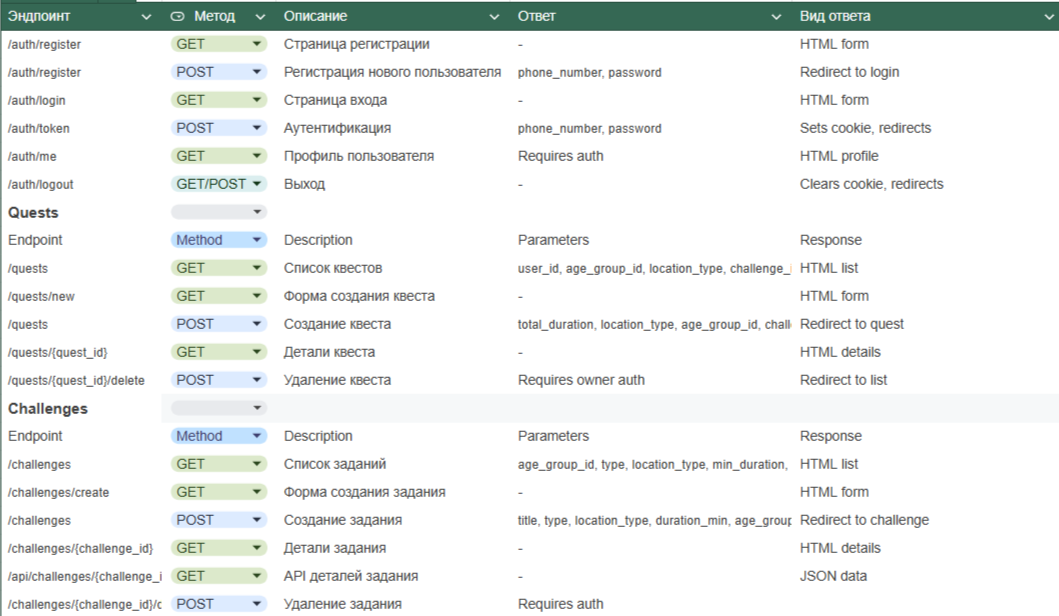


Есть обработчик ошибок

### Лабораторная работа 5. Финализация приложения и упаковка в Docker







### Ссылка на гитхаб:

https://github.com/marjurry/morekvestov