САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2
REST API

Выполнил:
Даньшин Семён
К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

По выбранному варианту необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Ход работы

1. Анализ реализованного API

В проекте реализован полнофункциональный RESTful API с использованием gRPC + gRPC-Gateway, что обеспечивает как REST, так и gRPC интерфейсы.

2. Структура REST API

Основные ресурсы и эндпоинты:

Аутентификация:

- `POST /v1/auth/login` - вход в систему
- `POST /v1/auth/refresh` - обновление токена
- `POST /v1/auth/logout` - выход из системы

Пользователи:

- `GET /v1/users/me` - получение текущего пользователя
- `POST /v1/users` - создание пользователя
- `PUT /v1/users` - обновление пользователя
- `GET /v1/users/{userId}` - получение пользователя по ID

Упражнения:

- `GET /v1/exercises` - получение списка упражнений
- `POST /v1/exercises` - создание упражнения
- `GET /v1/exercises/{exerciseId}` - получение упражнения
- `GET /v1/exercises/{exerciseId}/alternatives` - альтернативные упражнения
- `GET /v1/exercises/{exerciseId}/history` - история выполнения

Планы тренировок (Routines):

- `GET /v1/routines` - получение списка рутин
- `POST /v1/routines` - создание рутины
- `GET /v1/routines/{routineId}` - получение рутины
- `PUT /v1/routines/{routineId}` - обновление рутины
- `DELETE /v1/routines/{routineId}` - удаление рутины

Тренировки:

- `GET /v1/workouts` - получение списка тренировок
- `POST /v1/workouts` - начало тренировки
- `GET /v1/workouts/active` - активные тренировки
- `GET /v1/workouts/{workoutId}` - получение тренировки
- `POST /v1/workouts/{workoutId}/complete` - завершение тренировки

3. Реализация CRUD операций

Users Resource:

CREATE - Создание пользователя:

```
POST /v1/users
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "securepassword",
  "firstName": "John",
  "lastName": "Doe",
  "dateOfBirth": "1990-01-01T00:00:00Z",
  "height": 180.5,
  "weight": 75.0
}

Response 200:
{
  "user": {
    "id": "uuid-here",
    "email": "user@example.com",
    "firstName": "John",
    "lastName": "Doe",
    "createdAt": "2024-01-01T10:00:00Z",
    "updatedAt": "2024-01-01T10:00:00Z"
  }
}
```

READ - Получение пользователя:

```
GET /v1/users/me
Authorization: X-Access-Token <token>

Response 200:
{
  "user": {
    "id": "uuid-here",
    "email": "user@example.com",
    "firstName": "John",
    "lastName": "Doe",
    "height": 180.5,
    "weight": 75.0,
    "createdAt": "2024-01-01T10:00:00Z"
  }
}
```

UPDATE - Обновление пользователя:

```
PUT /v1/users
Authorization: X-Access-Token <token>
Content-Type: application/json

{
  "firstName": "Johnny",
  "weight": 73.5
}

Response 200:
{
  "user": {
    "id": "uuid-here",
    "firstName": "Johnny",
    "weight": 73.5,
    "updatedAt": "2024-01-01T11:00:00Z"
  }
}
```

Exercises Resource:

CREATE - Создание упражнения:

```
POST /v1/exercises
Authorization: X-Access-Token <token>
Content-Type: application/json

{
  "name": "Push-ups",
  "description": "Basic bodyweight exercise",
  "videoUrl": "https://example.com/pushups-video",
  "targetMuscleGroupIds": ["chest-uuid", "triceps-uuid"]
}

Response 200:
{
  "exercise": {
    "id": "exercise-uuid",
    "name": "Push-ups",
    "description": "Basic bodyweight exercise",
    "targetMuscleGroups": ["chest", "triceps"],
    "createdAt": "2024-01-01T10:00:00Z"
  }
}
```

READ - Получение списка упражнений:

```
GET /v1/exercises?muscleGroupIds=chest-uuid&excludeExerciseIds=some-uuid
Authorization: X-Access-Token <token>

Response 200:
{
  "exercises": [
    {
      "id": "exercise-uuid",
      "name": "Push-ups",
      "description": "Basic bodyweight exercise",
```

```
      "targetMuscleGroups": ["chest", "triceps"]
    }
  ]
}
```

Routines Resource:

CREATE - Создание плана тренировки:

```
POST /v1/routines
Authorization: X-Access-Token <token>
Content-Type: application/json

{
  "name": "Upper Body Workout",
  "description": "Focus on chest, shoulders, and arms"
}

Response 200:
{
  "routine": {
    "id": "routine-uuid",
    "name": "Upper Body Workout",
    "description": "Focus on chest, shoulders, and arms",
    "userId": "user-uuid",
    "createdAt": "2024-01-01T10:00:00Z"
  }
}
```

UPDATE - Обновление рутины:

```
PUT /v1/routines/{routineId}
Authorization: X-Access-Token <token>
Content-Type: application/json

{
  "name": "Updated Upper Body Workout",
  "description": "Enhanced upper body routine"
}

Response 200:
{
  "routine": {
    "id": "routine-uuid",
    "name": "Updated Upper Body Workout",
    "description": "Enhanced upper body routine",
    "updatedAt": "2024-01-01T11:00:00Z"
  }
}
```

DELETE - Удаление рутины:

```
DELETE /v1/routines/{routineId}
Authorization: X-Access-Token <token>

Response 200: {}
```

Workouts Resource:

CREATE - Начало тренировки:

```
POST /v1/workouts
Authorization: X-Access-Token <token>
Content-Type: application/json

{
  "routineId": "routine-uuid",
  "generateWorkout": false
}

Response 200:
{
  "workout": {
    "id": "workout-uuid",
    "userId": "user-uuid",
    "routineId": "routine-uuid",
    "createdAt": "2024-01-01T10:00:00Z",
    "finishedAt": null,
    "isAiGenerated": false
  }
}
```

READ - Получение тренировки:

```
GET /v1/workouts/{workoutId}
Authorization: X-Access-Token <token>

Response 200:
{
  "workout": {
    "id": "workout-uuid",
    "userId": "user-uuid",
    "notes": "Great workout today!",
    "rating": 4,
    "createdAt": "2024-01-01T10:00:00Z",
    "finishedAt": "2024-01-01T11:30:00Z"
  },
  "exerciseLogs": [
    {
      "exerciseLog": {
        "id": "log-uuid",
        "exerciseId": "exercise-uuid",
        "notes": "Felt strong today",
        "powerRating": 4
      },
      "exercise": {
        "name": "Push-ups",
        "description": "Basic bodyweight exercise"
      },
      "setLogs": [
        {
          "id": "set-uuid",
          "reps": 20,
          "weight": 0,
          "createdAt": "2024-01-01T10:15:00Z"
        }
```

```
      ]
    }
  ]
}
```

4. Статус коды и обработка ошибок

Успешные ответы:

- `200 OK` - успешное выполнение операции
- `201 Created` - ресурс создан (не используется, всегда 200)

Ошибки:

- `400 Bad Request` - неверные данные запроса
- `401 Unauthorized` - отсутствует или неверная аутентификация
- `403 Forbidden` - недостаточно прав
- `404 Not Found` - ресурс не найден
- `409 Conflict` - конфликт данных (например, email уже существует)
- `422 Unprocessable Entity` - ошибка валидации
- `500 Internal Server Error` - внутренняя ошибка сервера

Формат ошибок:

```
{
  "code": 9,
  "message": "User with this email already exists",
  "details": []
}
```

5. Пагинация

Запросы с пагинацией:

```
GET /v1/workouts?offset=0&limit=10
Authorization: X-Access-Token <token>

Response 200:
{
  "workouts": [
    {
      "workout": {
        "id": "workout-1",
        "createdAt": "2024-01-01T10:00:00Z"
      },
      "exerciseLogs": []
    }
  ]
}
```

6. Фильтрация и поиск

Фильтрация упражнений по группам мышц:

```
GET /v1/exercises?muscleGroupIds=chest,shoulders&excludeExerciseIds=some-uuid
```

История выполнения упражнений:

```
GET /v1/exercises/{exerciseId}/history?offset=0&limit=5
```

7. Nested Resources (Вложенные ресурсы)

Управление подходами в рутинах:

```
POST /v1/routines/{routineId}/exercise_instances/{exerciseInstanceId}/sets
PUT
/v1/routines/{routineId}/exercise_instances/{exerciseInstanceId}/sets/{setId}
DELETE
/v1/routines/{routineId}/exercise_instances/{exerciseInstanceId}/sets/{setId}
```

Логирование упражнений в тренировках:

```
POST /v1/workouts/{workoutId}/log/exercise
GET /v1/workouts/{workoutId}/log/exercise/{exerciseLogId}
DELETE /v1/workouts/{workoutId}/log/exercise/{exerciseLogId}

POST /v1/workouts/{workoutId}/log/exercise/{exerciseLogId}/set
PUT /v1/workouts/{workoutId}/log/exercise/{exerciseLogId}/set/{setId}
DELETE /v1/workouts/{workoutId}/log/exercise/{exerciseLogId}/set/{setId}
```

8. Специальные эндпоинты

AI-генерация тренировок:

```
POST /v1/workouts
Content-Type: application/json

{
  "generateWorkout": true,
  "userPrompt": "I want a chest and triceps workout"
}
```

Получение активных тренировок:

```
GET /v1/workouts/active
Authorization: X-Access-Token <token>

Response 200:
{
  "workouts": [
    {
      "id": "active-workout-uuid",
      "createdAt": "2024-01-01T10:00:00Z",
      "finishedAt": null
    }
```

```
  ]
}
```

Завершение тренировки:

```
POST /v1/workouts/{workoutId}/complete
Authorization: X-Access-Token <token>
Content-Type: application/json

{}

Response 200: {}
```

9. Content Negotiation

API поддерживает:

- `Content-Type: application/json` - для всех запросов
- `Accept: application/json` - для всех ответов

10. Аутентификация и авторизация

Header-based аутентификация:

```
X-Access-Token: <jwt-token>
```

Refresh token flow:

```
POST /v1/auth/refresh
Content-Type: application/json

{
  "tokens": {
    "accessToken": "old-token",
    "refreshToken": "refresh-token"
  }
}

Response 200:
{
  "tokens": {
    "accessToken": "new-access-token",
    "refreshToken": "new-refresh-token"
  }
}
```

[Скриншот Swagger документации API]

[Скриншот тестирования API в Postman]

11. Валидация данных

Примеры валидации:

- Email валидность при регистрации

- Минимальная длина пароля (8 символов)
- Проверка существования связанных ресурсов
- Валидация числовых значений (вес, рост)

Вывод

Реализован полнофункциональный RESTful API, соответствующий принципам REST:

1. **Ресурсо-ориентированная архитектура** - каждая сущность представлена как ресурс
2. **HTTP методы** - правильное использование GET, POST, PUT, DELETE
3. **Статус коды** - корректные HTTP статус коды для разных ситуаций
4. **Идемпотентность** - GET, PUT, DELETE операции идемпотентны
5. **Stateless** - каждый запрос содержит всю необходимую информацию
6. **Uniform Interface** - единообразный интерфейс для всех ресурсов
7. **Nested Resources** - поддержка вложенных ресурсов
8. **Content Negotiation** - поддержка JSON формата
9. **Аутентификация** - JWT-based аутентификация
10. **Валидация** - проверка входных данных

API предоставляет полную функциональность для фитнес-трекера и готов для использования в продакшене.