САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнила:

Гусейнова Марьям

БР 1.2

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

- Реализовать все модели данных, спроектированные в рамках ДЗ1
- Реализовать набор из CRUD-методов для работы с моделями данных средствами Express + TypeScript
- Реализовать API-эндпоинт для получения пользователя по id/email

Ход работы

В рамках данной лабораторной работы была продолжена разработка платформы фитнес-тренировок, начатая в ДЗ1. Основной целью стало создание бэкенд-части приложения, обеспечивающей взаимодействие с данными.

1. Реализация моделей данных (Entities):

- Были созданы TypeScript-классы, представляющие модели базы данных, в соответствии со схемой, разработанной в ДЗ1. К ним относятся: User, BlogPost, WorkoutPlan, PlanProgress, Workout, WorkoutInPlan, CurrentProgress, Exercise, ExerciseProgress, ExerciseWorkout (расположены в папке src/entities).
- Для каждой модели были определены соответствующие поля с указанием их типов и связей (например, использование декораторов @PrimaryGeneratedColumn(), @Column(), @ManyToOne(), @OneToMany(), @OneToOne()).

2. Реализация сервисов (Services):

- Для каждой модели данных был создан соответствующий сервис (в папке src/services), такой как UserService, BlogPostService, WorkoutPlanService, PlanProgressService, WorkoutService, WorkoutInPlanService, CurrentProgressService, ExerciseService, ExerciseProgressService, ExerciseWorkoutService.
- Каждый сервис содержит асинхронные методы для выполнения основных CRUD-операций: create, find, findOneBy (или findOne), update, delete. Использовался репозиторий, предоставляемый TypeORM (AppDataSource.getRepository(<Entity>)), для взаимодействия с базой данных.
- В UserService были дополнительно реализованы методы getUserById и getUserByEmail для выполнения поиска пользователей по соответствующим полям (рисунок 1).

```
async getUserById(id: number): Promise<User | null> {
  return this.userRepository.findOneBy({ user_id: id });
}

async getUserByEmail(email: string): Promise<User | null> {
  return this.userRepository.findOneBy({ email });
}
```

Рисунок 1 – Метод получения пользователя по email и id.

3. Реализация контроллеров (Controllers):

- Для каждого сервиса был создан контроллер (в папке src/controllers), такой как UserController, BlogPostController, WorkoutPlanController, PlanProgressController, WorkoutController, WorkoutInPlanController, CurrentProgressController, ExerciseController, ExerciseProgressController, ExerciseWorkoutController.
- Контроллеры обрабатывают входящие HTTP-запросы (GET, POST, PUT, DELETE) по определенным маршрутам и вызывают соответствующие методы сервисов для выполнения операций с данными.
- Для каждой сущности были определены API-эндпоинты в отдельных файлах маршрутов (в папке src/routes), например:
    - /users (GET - получить всех пользователей, POST - создать нового пользователя)
    - /users/:id (GET - получить пользователя по ID, PUT - обновить пользователя, DELETE - удалить пользователя)
    - /users/email/:email (GET - получить пользователя по email)
    - Аналогичные эндпоинты были созданы для остальных сущностей.

4. Настройка базы данных:

- Было настроено подключение к базе данных PostgreSQL с использованием ORM TypeORM. Параметры подключения (тип базы данных, хост, порт, имя пользователя, пароль, имя базы данных) были определены в файле data-source.ts.

- Для хранения чувствительной информации был использован файл .env и библиотека dotenv для загрузки переменных окружения. Файл .env был добавлен в .gitignore.
- При первом запуске приложения (с опцией synchronize: true в data-source.ts) TypeORM успешно подключился к указанной базе данных и автоматически создал необходимые таблицы на основе определенных моделей.

Вывод

В данной лабораторной работе я разработала структуру базы данных, реализовала CRUD API для всех сущностей, настроила подключение к базе данных и организовала маршрутизацию.