САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:

Бархатова Наталья

Группа К3339

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

На основе схемы базы данных были созданы модели. Рассмотрим модель Post.

```
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne }
from "typeorm";
import { User } from "./User";

@Entity()
export class Post {
  @PrimaryGeneratedColumn("uuid")
  id: string;

  @ManyToOne(() => User, (user) => user.posts)
  user: User;

  @Column({ type: "varchar", length: 255 })
  title: string;

  @Column("text")
  text: string;

  @Column({ type: "timestamp", default: () =>
"CURRENT_TIMESTAMP" })
  published_at: Date;
}
```

Остальные модели реализованы аналогично. Так же в проект было добавлено несколько enum, один из них – Gender

```
export enum Gender {
    Male = "male",
```

```
    Female = "female",
    Unspecified = "unspecified",
  }
```

Затем в контроллерах были реализованы CRUD методы.

```
import { Request, Response } from "express";
import { AppDataSource } from "../AppDataSource";
import { Workout } from "../models/Workout";

const workoutRepo = AppDataSource.getRepository(Workout);

export const createWorkout = async (req: Request, res:
Response) => {
  try {
    const workoutData = req.body;
    const workout = workoutRepo.create(workoutData);
    const savedWorkout = await workoutRepo.save(workout);
    res.status(201).json(savedWorkout);
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};

export const getAllWorkouts = async (req: Request, res:
Response) => {
  try {
    const workouts = await workoutRepo.find();
    res.json(workouts);
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};

export const getWorkoutById = async (req: Request, res:
Response): Promise<any> => {
  try {
    const workout = await workoutRepo.findOne({ where: { id:
req.params.id }, relations: ["trainingPlans"] });
    if (!workout) {
      return res.status(404).json({ message: "Workout not
found" });
    }
    res.json(workout);
```

```
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};

export const updateWorkout = async (req: Request, res:
Response): Promise<any> => {
  try {
    const workout = await workoutRepo.findOne({ where: { id:
req.params.id } });
    if (!workout) {
      return res.status(404).json({ message: "Workout not
found" });
    }

    workoutRepo.merge(workout, req.body);
    const updatedWorkout = await workoutRepo.save(workout);
    res.json(updatedWorkout);
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};

export const deleteWorkout = async (req: Request, res:
Response): Promise<any> => {
  try {
    const result = await workoutRepo.delete(req.params.id);
    if (result.affected === 0) {
      return res.status(404).json({ message: "Workout not
found" });
    }
    res.json({ message: "Workout deleted successfully" });
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};
```

Аналогично были реализованы CRUD для других моделей. Также были реализованы роутеры

```
import { Router } from "express";
import {
  createWorkout,
  getAllWorkouts,
```

```
  getWorkoutById,
  updateWorkout,
  deleteWorkout,
} from "../controllers/WorkoutController";

const router = Router();

router.post("/", createWorkout);
router.get("/", getAllWorkouts);
router.get("/:id", getWorkoutById);
router.put("/:id", updateWorkout);
router.delete("/:id", deleteWorkout);

export default router;
```

В последнем задании необходимо реализовать endpoint получения пользователя по email, я делала это с помощью следующей функции:

```
export const getUserByEmail = async (req: Request, res:
Response): Promise<any> => {
  try {
    const { email } = req.query;

    if (!email) {
      return res.status(400).json({ message: "Email is
required" });
    }

    const user = await userRepo.findOne({
      where: { email: email as string }
    });

    if (!user) {
      return res.status(404).json({ message: "User not found"
});
    }

    res.json(user);
  } catch (error: any) {
    res.status(500).json({ message: error.message });
  }
};
```

В роутере данный контроллер выглядит так:

```
router.get("/by-email", getUserByEmail);
```

Вывод

В рамках задания были реализованы все модели данных, спроектированные в ДЗ1, с использованием TypeORM. Для каждой модели разработан набор CRUD-методов с помощью Express и TypeScript. Также добавлен отдельный API-эндпоинт для получения пользователя по id и по email.