Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчет

«Лабораторная работа №1»

Автор: Кононов Степан

Факультет: ИКТ

Группа: К33392

Санкт-Петербург 2024

Отчет по лабораторной работе №1

Введение

В данной лабораторной работе нужно простой RESTful API на основе Express.js с использованием TypeORM и TypeScript. API включает базовые CRUD операции для управления пользователями. Реализация выполнена с четким разделением на модели, контроллеры, роуты и сервисы, что способствует лучшей поддерживаемости и масштабируемости кода.

Структура проекта

Проект организован по следующей структуре:

- `server.ts`: Основной файл для запуска сервера.
- `database.ts`: Файл для подключения к базе данных.
- `app.ts`: Инициализация приложения Express и маршрутов.
- `models/user.entity.ts`: Определение модели "User" с использованием TypeORM.
- `controllers/user.controller.ts`: Контроллер, обрабатывающий запросы, поступающие на маршруты.
- `routes/user.routes.ts`: Определение маршрутов для CRUD операций с пользователями.
- `services/user.service.ts`: Сервис, содержащий логику для взаимодействия с базой данных.

Подробное описание кода

`server.ts`

Этот файл отвечает за инициализацию и запуск сервера. Он подключается к базе данных и запускает Express сервер.

```
import 'reflect-metadata';
import app from './app';
import connectDatabase from './database';
```

```
const PORT = process.env.PORT || 3000;

const startServer = async () => {
    try {
        await connectDatabase();

        app.listen(PORT, () => {
            console.log(`Server is running on port ${PORT}
`);
        });
    } catch (error) {
        console.error('Error starting server:', error);
    }
};

startServer();
```

database.ts

В данном файле происходит подключение к базе данных с помощью TypeORM. В случае ошибки подключение, приложение завершает работу.

```
import { createConnection } from 'typeorm';

const connectDatabase = async () => {
    try {
        await createConnection();
        console.log('Database connected successfully');
    } catch (error) {
        console.error('Database connection error:', error);
        process.exit(1);
    }
};

export default connectDatabase;
```

app.ts

Файл для инициализации Express приложения и определения маршрутов.

```
import express from 'express';
import bodyParser from 'body-parser';
import userRoutes from './routes/user.routes';

const app = express();
app.use(bodyParser.json());

app.use('/users', userRoutes);

export default app;
```

`models/user.entity.ts`

Файл, содержащий описание модели пользователя с использованием TypeORM и валидаторов.

```
import { Entity, PrimaryGeneratedColumn, Column } from 'typ
eorm';
import { IsEmail, Length } from 'class-validator';

@Entity()
export class User {
    @PrimaryGeneratedColumn()
    id!: number;

    @Column()
    @Length(1, 100)
    name!: string;

    @Column()
    @IsEmail()
    email!: string;
}
```

`services/user.service.ts`

Сервис для управления пользователями, реализующий все основные CRUD операции.

```
import { getRepository } from 'typeorm';
import { User } from '../models/user.entity';

export class UserService {

    async findAll(): Promise<User[]> {
        const userRepository = getRepository(User);
        return userRepository.find();
    }

    async create(user: User): Promise<User> {
        const userRepository = getRepository(User);
        return userRepository.save(user);
    }

    async update(id: number, user: Partial<User>): Promise<
void> {
        const userRepository = getRepository(User);
        await userRepository.update(id, user);
    }

    async delete(id: number): Promise<void> {
        const userRepository = getRepository(User);
        await userRepository.delete(id);
    }
}
```

`controllers/user.controller.ts`

Контроллер для обработки запросов и взаимодействия с пользовательским сервисом.

```
import { Request, Response } from 'express';
import { UserService } from '../services/user.service';
import { User } from '../models/user.entity';

const userService = new UserService();

export class UserController {
```

```
    async getAllUsers(req: Request, res: Response) {
        const users = await userService.findAll();
        res.json(users);
    }

    async createUser(req: Request, res: Response) {
        const user = req.body as User;
        const newUser = await userService.create(user);
        res.status(201).json(newUser);
    }

    async updateUser(req: Request, res: Response) {
        const id = parseInt(req.params.id);
        const user = req.body as Partial<User>;
        await userService.update(id, user);
        res.status(204).end();
    }

    async deleteUser(req: Request, res: Response) {
        const id = parseInt(req.params.id);
        await userService.delete(id);
        res.status(204).end();
    }
}
```

routes/user.routes.ts

Определение маршрутов для операций CRUD над пользователями.

```
import { Router } from 'express';
import { UserController } from '../controllers/user.control
ler';

const router = Router();
const userController = new UserController();

router.get('/', userController.getAllUsers);
router.post('/', userController.createUser);
router.put('/:id', userController.updateUser);
```

```
router.delete('/:id', userController.deleteUser);

export default router;
```

Заключение

Успешно реализовал RESTful API с использованием Express.js, TypeORM и TypeScript. Создал четкую структуру проекта, что способствует лучшей поддерживаемости и расширяемости кода. Работа с базой данных организована через сервис с реализацией паттерна "репозиторий", что позволяет гибко взаимодействовать с моделью данных.