САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа 2

Выполнил:
Байков Иван
К33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

1. Продумать свою собственную модель пользователя
2. Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize
3. Написать запрос для получения пользователя по id/email

Ход работы

1. Установим зависимости

```
  "devDependencies": {
    "@types/express": "^4.17.21",
    "@types/node": "^20.12.5",
    "pg": "^8.11.3",
    "ts-node": "^10.9.2",
    "tsx": "^4.9.4",
    "typescript": "^5.4.2"
  },
  "dependencies": {
    "@types/cors": "^2.8.17",
    "cors": "^2.8.5",
    "dotenv": "^16.4.5",
    "express": "^4.18.3",
    "nodemon": "^3.1.0",
    "reflect-metadata": "^0.2.1",
    "typeorm": "^0.3.20"
  }
```

2. Подключим typescript & typeorm

```
{
  "compilerOptions": {
    "lib": ["ES2016"],
    "target": "ES2022",
    "module": "Node16",

    "esModuleInterop": true,
    "skipLibCheck": true,
    "moduleResolution": "Node16",

    "outDir": "./dist",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true
  },
  "include": ["src"],
  "exclude": ["node_modules"]
}
```

```ts
import { fileURLToPath } from "node:url";
import path, { dirname } from "node:path";
import { DataSource } from "typeorm";
import "dotenv/config";

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);

export const PostgresDataSource = new DataSource({
  type: "postgres",
  host: process.env.PG_HOST,
  port: +process.env.PG_PORT,
  username: process.env.PG_USER,
  database: process.env.PG_DB,
  password: process.env.PG_PASSWORD,
  synchronize: true, //todo
  logging: false,
  logNotifications: false,
  applicationName: "itmo_backend",
  entities: [path.join(__dirname, "entities", "*.{ts,js}")],
  migrations: [path.join(__dirname, "migrations", "*.{ts,js}")],
});
```

В качестве базы данных использую postgres 14

3. Продумать свою собственную модель пользователя

```ts
import { Entity, PrimaryGeneratedColumn, Column } from "typeorm";

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

  @Column({ type: "varchar", unique: true })
  email: string;

  @Column({ type: "varchar", unique: true })
  password: string;
}
```

4. Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize

Создаем роутер

```
import { Router } from "express";
import usersController from "../controllers/users.js";

const router = Router();

router.get("/:idOrEmail", usersController.get);
router.post("/:id", usersController.create);
router.put("/:id", usersController.update);
router.delete("/:id", usersController.delete);

export default router;
```

Контроллер

```
  get: async (req: Request, res: Response) => {
    try {
      const idOrEmail = req.params.idOrEmail;
      const user = await usersService.get(idOrEmail);
      if (user) {
        res.status(200).json(user);
      } else {
        res.status(404).json({ message: "User not found" });
      }
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },

  create: async (req: Request, res: Response) => {
    try {
      const { email, password } = req.body;
      if (!email || !password) {
        return res.status(400).json({ message: "Email and password are required" });
      }
      const user = await usersService.create({ email, password });
      res.status(201).json(user);
    } catch (error) {
      res.status(500).json({ message: "Internal server error" });
    }
  },
```

Сервис

```
get: async (idOrEmail: string) => {
  try {
    const id = parseInt(idOrEmail, 10);

    if (!isNaN(id)) {
      return await userRepository.findOneBy({ id });
    } else {
      return await userRepository.findOneBy({ email: idOrEmail });
    }
  } catch (error) {
    throw new Error("Error fetching user");
  }
},

create: async (userData: { email: string; password: string }) => {
  try {
    const user = userRepository.create(userData);
    return await userRepository.save(user);
  } catch (error) {
    throw new Error("Error creating user");
  }
},
```

```
update: async (id: number, updateData: { email?: string; password?: string }) => {
  try {
    const user = await userRepository.findOneBy({ id });
    if (!user) return null;

    user.email = updateData.email || user.email;
    user.password = updateData.password || user.password;

    return await userRepository.save(user);
  } catch (error) {
    throw new Error("Error updating user");
  }
},

delete: async (id: number) => {
  try {
    const result = await userRepository.delete(id);
    return result.affected > 0;
  } catch (error) {
    throw new Error("Error deleting user");
  }
},
```

5. Написать запрос для получения пользователя по id/email

```
get: async (idOrEmail: string) => {
  try {
    const id = parseInt(idOrEmail, 10);

    if (!isNaN(id)) {
      return await userRepository.findOneBy({ id });
    } else {
      return await userRepository.findOneBy({ email: idOrEmail });
    }
  } catch (error) {
    throw new Error("Error fetching user");
  }
},
```

Вывод

В данной работе был реализован HTTP сервер, выполняющий CRUD операции на пользователем с использованием express & typeorm