САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа №3: Микросервисы

Выполнил:

Жигалова Анастасия
K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Необходимо реализовать отдельный микросервис, выполняющий какую-либо содержательную функцию из всего арсенала функций вашего приложения.

Вариант : Сервис для работы с магазином одежды. Требуемый функционал: регистрация, авторизация, создание профиля, работа с товарами, просмотр количества единиц товара, управление скидками и акциями, работа с базой клиентов.

Ход работы

1. Сервис по работе с пользователями перенесла в отдельный сервис

Models

```
import {
  Table,
  Column,
  Model,
  Unique,
  AllowNull,
  HasMany,
} from "sequelize-typescript";

@Table({
  tableName: "Users",
})
export class User extends Model<User> {
  @Column
  name: string;

  @Unique
  @Column
  email: string;

  @AllowNull(false)
  @Column
  password: string;
}
```

Sevices

```
dotenv.config();

const secretKey = process.env.SECRET_KEY;

export class UserService {
  async register(email: string, password: string) {
    const hashedPassword = await bcrypt.hash(password, 8);
    const user = await User.create({ email, password: hashedPassword });
    return user;
  }

  async login(email: string, password: string) {
    const user = await User.findOne({ where: { email } });
    if (!user) {
      throw new Error("User not found");
    }

    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      throw new Error("Invalid password");
    }

    const token = jwt.sign({ id: user.id }, secretKey);
    return token;
  }

  async getById(id: number) {
    return await User.findByPk(id);
  }
}
```

Routes

```
const router = express.Router();

const userController = new UserController();

router.post("/register", userController.register);

router.post("/login", userController.login);

router.get("/:id", userController.getUserWithOrders);
router.post("/verify", userController.verify);

export default router;
```

Controllres

```
const userService = new UserService();

export class UserController {
  async register(req: Request, res: Response) {
    try {
      const { email, password } = req.body;
      const user = await userService.register(email, password);
      res.status(201).json(user);
    } catch (error) {
      if (error instanceof Error) {
        res.status(500).json({ message: error.message });
      }
    }
  }

  async login(req: Request, res: Response) {
    try {
      const { email, password } = req.body;
      const token = await userService.login(email, password);
      res.status(200).json({ token });
    } catch (error) {
      if (error instanceof Error) {
        res.status(500).json({ message: error.message });
      }
    }
  }

  async getUserWithOrders(req: Request, res: Response) {
    try {
      if (!req.headers.authorization) return res.sendStatus(401);
      const userId = req.params.id;
      if (!userId)
        return res.status(400).send({ error: "userId was not provided" });
```

Config

```
export default {
  port: process.env.PORT || 8080,
  db: {
    database: process.env.DB_NAME || 'some_db',
    dialect: process.env.DB_DIALECT || 'sqlite',
    username: process.env.DB_USERNAME || 'root',
    password: process.env.DB_PASSWORD || '',
    storage: process.env.DB_STORAGE || 'db.sqlite',
  },
  secretKey: process.env.SECRET_KEY,
};
```

Вывод

В ходе выполнения лабораторной работы реализовала отдельный микросервис по работе с пользователями.

При получении пользователя с заказами сервис обращается к другому сервису для получения заказов. Middleware теперь делает запрос на верификацию токена в отдельный сервис.