САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет
Лабораторная работа №3

Выполнила: Злотникова К.А.
Группа: K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача
Необходимо реализовать отдельный микросервис, выполняющий содержательную функцию из всего арсенала функций вашего приложения. В данном случае, мы реализуем микросервис для работы с товарами, включающий функции управления товарами, покупок, скидок и отзывов.

Ход работы

1. **Создание структуры проекта**

Создаем структуру проекта, чтобы было легко найти и организовать нужные файлы:

shop-service

```
├── db
├── node_modules
├── src
│   ├── controllers
│   │   ├── ItemController.ts
│   │   ├── OrderController.ts
│   │   └── RatingController.ts
│   ├── entities
│   │   ├── Item.ts
│   │   ├── Order.ts
│   │   └── Rating.ts
│   ├── middleware
│   │   └── authMiddleware.ts
│   ├── routes
│   │   ├── itemRoutes.ts
│   │   ├── orderRoutes.ts
│   │   └── ratingRoutes.ts
```

```
│   └── index.ts
├── ormconfig.json
├── package.json
└── tsconfig.json
```

2. Настройка TypeScript и TypeORM

Создаем файлы tsconfig.json и ormconfig.json для настройки TypeScript и TypeORM:

tsconfig.json:

```
{
 "compilerOptions": {
   "target": "ES6",
   "module": "commonjs",
   "strict": true,
   "esModuleInterop": true,
   "skipLibCheck": true,
   "outDir": "./dist",
   "experimentalDecorators": true,
   "emitDecoratorMetadata": true
 },
 "include": ["src/**/*"],
 "experimentalDecorators": true,
 "emitDecoratorMetadata": true
}
```

ormconfig.json:

```
{
 "type": "sqlite",
```

```
 "database": "db/main.sqlite",

 "synchronize": true,

 "logging": false,

 "entities": [

   "src/entities/**/*.ts"

 ]

}
```

3. Модели

Создаем модели для товаров, заказов и рейтингов.

src/entities/Item.ts:

```
import { Entity, PrimaryGeneratedColumn, Column, OneToMany } from 'typeorm';

import { Rating } from './Rating';

@Entity()

export class Item {

   @PrimaryGeneratedColumn()

   id!: number;

   @Column()

   name!: string;

   @Column('decimal')

   price!: number;

   @Column('int')

   stock!: number;

   @Column('decimal', { default: 0 })

   discount!: number;

   @OneToMany(() => Rating, rating => rating.item)

   ratings!: Rating[];
```

```
}
```

src/entities/Order.ts:

```
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from 'typeorm';

import { Item } from './Item';

@Entity()

export class Order {

   @PrimaryGeneratedColumn()

   id!: number;

   @Column()

   userId!: number;

   @ManyToOne(() => Item)

   item!: Item;

   @Column('int')

   quantity!: number;

   @Column()

   date!: Date;

}
```

src/entities/Rating.ts:

```
import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from 'typeorm';

import { Item } from './Item';

@Entity()

export class Rating {

   @PrimaryGeneratedColumn()

   id!: number;
```

```
    @Column()
    userId!: number;
    @ManyToOne(() => Item, item => item.ratings)
    item!: Item;
    @Column('int')
    rating!: number;
    @Column()
    comment!: string;
}
```

4. Контроллеры

Создаем контроллеры для управления товарами, заказами и рейтингами.

src/controllers/ItemController.ts:

```
import {Request, Response} from 'express';
import {getRepository} from 'typeorm';
import {Item} from '../entities/Item';
class ItemController {
    static async addItem(req: Request, res: Response) {
        try {
            const itemRepository = getRepository(Item);
            const newItem = itemRepository.create(req.body);
            await itemRepository.save(newItem);
            res.status(201).json(newItem);
        } catch (error: any) {
            res.status(400).json({message: error.message});
        }
    }
```

```
   static async modifyItem(req: Request, res: Response) {

       try {

           const itemRepository = getRepository(Item);

           await itemRepository.update(req.params.id, req.body);

                   const updatedItem = await itemRepository.findOne({where: {id:
Number(req.params.id)}});

           res.status(200).json(updatedItem);

       } catch (error: any) {

           res.status(400).json({message: error.message});

       }

   }

   static async removeItem(req: Request, res: Response) {

       try {

           const itemRepository = getRepository(Item);

           await itemRepository.delete(req.params.id);

           res.status(204).send();

       } catch (error: any) {

           res.status(400).json({message: error.message});

       }

   }

   static async getItemStock(req: Request, res: Response) {

       try {

           const itemRepository = getRepository(Item);

                       const item = await itemRepository.findOne({where: {id:
Number(req.params.id)}});

           res.status(200).json({stock: item?.stock});

       } catch (error: any) {
```

```
            res.status(400).json({message: error.message});

        }

    }

    static async applyOffer(req: Request, res: Response) {

        try {

            const itemRepository = getRepository(Item);

                                await itemRepository.update(req.params.id, {discount:
req.body.discount});

                    const updatedItem = await itemRepository.findOne({where: {id:
Number(req.params.id)}});

            res.status(200).json(updatedItem);

        } catch (error: any) {

            res.status(400).json({message: error.message});

        }

    }

}

export default ItemController;
```

src/controllers/OrderController.ts:

```
import { Request, Response } from 'express';

import { getRepository } from 'typeorm';

import { Item } from '../entities/Item';

import { Order } from '../entities/Order';

class OrderController {

    static async placeOrder(req: Request, res: Response) {

        try {

            const { userId, itemId, quantity } = req.body;

            const itemRepository = getRepository(Item);
```

```
            const orderRepository = getRepository(Order);
            const item = await itemRepository.findOne({ where: { id: itemId } });
            if (!item || item.stock < quantity) {
                throw new Error('Insufficient stock');
            }
            item.stock -= quantity;
            await itemRepository.save(item);
            const newOrder = orderRepository.create({
                userId,
                item,
                quantity,
                date: new Date()
            });
            await orderRepository.save(newOrder);
            res.status(201).json(newOrder);
        } catch (error: any) {
            res.status(400).json({ message: error.message });
        }
    }
    static async getOrderHistory(req: Request, res: Response) {
        try {
            const userId = Number(req.params.userId);
            const orderRepository = getRepository(Order);
                  const orders = await orderRepository.find({ where: { userId },
relations: ['item'] });
            res.status(200).json(orders);
```

```
        } catch (error: any) {
            res.status(400).json({ message: error.message });
        }
    }
}
export default OrderController;
```

src/controllers/RatingController.ts:

```
import {Request, Response} from 'express';
import {getRepository} from 'typeorm';
import {Rating} from '../entities/Rating';
import {Item} from '../entities/Item';
class RatingController {
    static async addRating(req: Request, res: Response) {
        try {
            const ratingRepository = getRepository(Rating);
            const itemRepository = getRepository(Item);
            const {itemId, rating, comment} = req.body;
            const item = await itemRepository.findOne({where: {id: itemId}});
            if (!item) {
                throw new Error('Item not found');
            }

            const newRating = ratingRepository.create({
                userId: req.body.userId,
                item,
                rating,
```

```
                comment

            });

            await ratingRepository.save(newRating);

            res.status(201).json(newRating);

        } catch (error: any) {

            res.status(400).json({message: error.message});

        }

    }

    static async getRatings(req: Request, res: Response) {

        try {

            const ratingRepository = getRepository(Rating);

            const itemId = Number(req.params.itemId);

                  const ratings = await ratingRepository.find({where: {item: {id:
itemId}}, relations: ['item']});

            res.status(200).json(ratings);

        } catch (error: any) {

            res.status(400).json({message: error.message});

        }

    }

    static async updateRating(req: Request, res: Response) {

        try {

            const ratingRepository = getRepository(Rating);

            const itemId = Number(req.params.itemId);

            const userId = req.body.userId;

            const {rating, comment} = req.body;

               const existingRating = await ratingRepository.findOne({where: {item:
{id: itemId}, userId}});
```

```
            if (!existingRating) {
                throw new Error('Rating not found');
            }
            existingRating.rating = rating;
            existingRating.comment = comment;
            await ratingRepository.save(existingRating);
            res.status(200).json(existingRating);
        } catch (error: any) {
            res.status(400).json({message: error.message});
        }
    }
    static async deleteRating(req: Request, res: Response) {
        try {
            const ratingRepository = getRepository(Rating);
            const { itemId } = req.params;
            const userId = req.body.user.userId;

               const rating = await ratingRepository.findOne({ where: { item: { id: Number(itemId) }, userId } });
            if (!rating) {
                return res.status(404).json({ message: 'Rating not found' });
            }
            await ratingRepository.delete(rating.id);
            res.status(204).send();
        } catch (error: any) {
            res.status(400).json({ message: error.message });
```

```
        }

    }

}

export default RatingController;
```

5. Роуты

Создаем роуты для товаров, заказов и рейтингов.

src/routes/itemRoutes.ts:

```
import { Router } from 'express';

import ItemController from '../controllers/ItemController';

import authenticate from '../middleware/authMiddleware';

const router = Router();

router.post('/items', authenticate, ItemController.addItem);

router.put('/items/:id', authenticate, ItemController.modifyItem);

router.delete('/items/:id', authenticate, ItemController.removeItem);

router.get('/items/:id/stock', authenticate, ItemController.getItemStock);

router.patch('/items/:id/offer', authenticate, ItemController.applyOffer);

export default router;
```

src/routes/orderRoutes.ts:

```
import { Router } from 'express';

import OrderController from '../controllers/OrderController';

import authenticate from '../middleware/authMiddleware';

const router = Router();

router.post('/order', authenticate, OrderController.placeOrder);

router.get('/order/history/:userId',                                    authenticate,
OrderController.getOrderHistory);

export default router;
```

src/routes/ratingRoutes.ts:

```
import { Router } from 'express';

import RatingController from '../controllers/RatingController';

import authenticate from '../middleware/authMiddleware';

const router = Router();

router.post('/ratings', authenticate, RatingController.addRating);

router.get('/ratings/:itemId', authenticate, RatingController.getRatings);

router.put('/ratings/:itemId', authenticate, RatingController.updateRating);

router.delete('/ratings/:itemId', authenticate, RatingController.deleteRating);

export default router;
```

6. Настройка сервера

Создаем главный файл для запуска сервера src/index.ts:

```
import 'reflect-metadata';

import express from 'express';

import bodyParser from 'body-parser';

import { createConnection } from 'typeorm';

import itemRoutes from './routes/itemRoutes';

import orderRoutes from './routes/orderRoutes';

import ratingRoutes from "./routes/ratingRoutes";

const app = express();



app.use(bodyParser.json());

app.use('/api', itemRoutes);

app.use('/api', orderRoutes);

app.use('/api', ratingRoutes);
```

```
createConnection().then(() => {

   app.listen(3000, () => {

       console.log('Main service is listening on port 3000');

   });

}).catch(error => console.error(error));
```

7. Настройка аутентификации

7.1. Middleware для проверки токена

Создаем файл src/middleware/authMiddleware.ts:

```
import { Request, Response, NextFunction } from 'express';

import amqp, { Channel, Connection, ConsumeMessage } from 'amqplib';

import jwt from 'jsonwebtoken';

interface User {

   id: number;

   username: string;

}

const jwtSecret = 'super_secret_key';

let amqpChannel: Channel;

const authenticate = async (req: Request, res: Response, next: NextFunction) => {

   try {

       const authHeader = req.headers.authorization;

       if (!authHeader) {

           return res.status(403).json({ message: 'Token not provided' });

       }

       const [bearer, token] = authHeader.split(' ');

       if (bearer !== 'Bearer' || !token) {

               return res.status(403).json({ message: 'Invalid token' });
```

```
        }

        const decoded: any = jwt.verify(token, jwtSecret);

        const userId = decoded.userId;

        const connection: Connection = await amqp.connect('amqp://localhost');

        amqpChannel = await connection.createChannel();

        const queue = 'auth_queue';

        await amqpChannel.assertQueue(queue, { durable: false });

        let user: User | null = null;

        await amqpChannel.consume(queue, (msg: ConsumeMessage | null) => {

            if (msg !== null) {

                user = JSON.parse(msg.content.toString());

                amqpChannel.ack(msg);

            }

        });

        await sendUserId(userId);

        await amqpChannel.close();

        await connection.close();

        setTimeout(() => {

            if (user) {

                req.body.user = user;

                next();

            } else {

                return res.status(403).json({ message: 'Invalid token' });

            }

        }, 500);

    } catch (error) {
```

```
      return res.status(403).json({ message: error });

   }

};

const sendUserId = async (userId: number) => {

   if (!amqpChannel) return;

   const queue = 'auth_queue';

   amqpChannel.sendToQueue(queue, Buffer.from(JSON.stringify({ userId })));

};

export default authenticate;
```

8. Запуск проекта

8.1. Скрипты запуска

Добавляем скрипты запуска в package.json для микросервиса.

shop-service/package.json:

```
"scripts": {

  "start": "ts-node src/index.ts"

},
```

Пример выполнения запросов:

- Добавление товара: POST /api/items
  {
  "name": "Shirt",
  "price": 19.99,
  "stock": 100
  }
- Изменение товара: PUT /api/items/:id
  {
  "name": "Updated Shirt",
  "price": 18.99,
  "stock": 150
  }
- Удаление товара: DELETE /api/items/:id
- Получение количества товара: GET /api/items/:id/stock

- Применение скидки к товару: PATCH /api/items/:id/offer
  {
  "discount": 10
  }
- Создание заказа: POST /api/order
  {
  "userId": 1,
  "itemId": 2,
  "quantity": 3
  }
- Получение истории заказов: GET /api/order/history/:userId
- Добавление отзыва: POST /api/ratings
  {
  "userId": 1,
  "itemId": 2,
  "rating": 5,
  "comment": "Great product!"
  }
- Получение отзывов: GET /api/ratings/:itemId
- Обновление отзыва: PUT /api/ratings/:itemId
  {
  "userId": 1,
  "rating": 4,
  "comment": "Good product!"
  }
- Удаление отзыва: DELETE /api/ratings/:itemId

Вывод

Таким образом, мы создали микросервис для управления товарами, заказами и отзывами в магазине одежды, используя Express, TypeORM и TypeScript. Этот микросервис обеспечивает набор функций: добавление, изменение, удаление товаров, управление заказами и отзывами, а также аутентификацию пользователей через middleware.