САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет
Лабораторная работа №2
“Тестирование, разработка и документирование RESTful API”

Выполнил:

Кондрашов Е. Ю.

Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

Необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate). Платформа для поиска и бронирования номера в отеле/квартире/хостеле:

1. Вход;
2. Регистрация;
3. Страница бронирований пользователя;
4. Страница для поиска номера с возможностью выбора города, времени заселения, количеству гостей.

Ход работы

Модели:

Пользователь:

```
import { IsEmail, validateOrReject } from 'class-validator';

import { BeforeInsert, BeforeUpdate, Column, CreateDateColumn,
Entity, OneToMany, PrimaryGeneratedColumn, UpdateDateColumn } from
'typeorm';

import Booking from '../bookings/Booking';



@Entity('users')

export class User {

   @PrimaryGeneratedColumn()

   id: number;



   @Column({

       unique: true,

   })
```

```
    @IsEmail()

    email: string;



    @Column({

        nullable: true

    })

    name: string;



    @Column({ select: false })

    password: string;



    @Column()

    @CreateDateColumn()

    created_at: Date;



    @Column()

    @UpdateDateColumn()

    updated_at: Date;



    @OneToMany(() => Booking, (booking) => booking.user)

    bookings: Booking[]



    @BeforeInsert()

    @BeforeUpdate()

    validate(): Promise<void> {
```

```
        return validateOrReject(this);

    }

}



export default User
```

Отель:

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

import Booking from '../bookings/Booking';



@Entity('hotels')

export class Hotel {

    @PrimaryGeneratedColumn()

    id: number;



    @Index()

    @Column({

        length: 511

    })

    name: string;





    @Column({

        length: 1023

    })
```

```
    address: string;


    @Column({
        length: 2047
    })
    img_url: string;


    @Column({
        type: 'text'
    })
    description: string;


    @Column({
        type: 'float', nullable: true
    })
    rating: number


    @Column({
        type: 'float', nullable: true
    })
    cost_from: number


    @OneToMany(() => Booking, (booking) => booking.hotel)
    bookings: Booking[]
}
```

```
export default Hotel
```

Бронирование:

```
import { IsDateString, validateOrReject } from 'class-validator';

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

import Hotel from '../hotels/Hotel';

import User from '../users/User';



@Entity('bookings')

export class Booking {

   @PrimaryGeneratedColumn()

   id: number;



   @ManyToOne(() => User, (user) => user.bookings)

   user: User;



   @ManyToOne(() => Hotel, (hotel) => hotel.bookings)

   hotel: Hotel;



   @IsDateString()

   @Column('date')

   starts_at: Date;



   @IsDateString()
```

```
    @Column('date')

    ends_at: Date;



    @Column('smallint')

    number_of_guests: number;



    @BeforeInsert()

    @BeforeUpdate()

    validate(): Promise<void> {

        return validateOrReject(this);

    }

}



export default Booking
```

В качестве ORM используется TypeORM. Для валидации полей используется пакет class-validator.

Сервисы:

Пользователь:

```
import bcrypt from 'bcryptjs';
import { getRepository } from 'typeorm';
import User from '../../orm/models/users/User';

class UserService {

    async getById(id: number): Promise<User> {
        const user = await getRepository(User).findOne(id);

        if (user) return user;

        throw new Error('User with specified ID is not found');
    }
```

```
        async getByEmail(email: string): Promise<User> {
             const user = await getRepository(User).findOne({ where: { email } });

            if (user) return user;

            throw new Error('User with specified email is not found');
        }

         async create(userData: { email: string, password: string }): Promise<User> {
            try {
                userData.password = bcrypt.hashSync(userData.password, 8)
                let user = getRepository(User).create(userData)
                user = await getRepository(User).save(user)
                return user
            } catch (e: any) {
                throw new Error(e)
            }
        }

            async checkPassword(email: string, password: string): Promise<any> {
             // const user = await getRepository(User).findOne({ where: { email } })
            const user = await getRepository(User)
                .createQueryBuilder("user")
                .addSelect("user.password")
                .where("user.email = :email", { email })
                .getOne()

            if (!user) {
                throw new Error('Does not exist')
            }

                  const passwordMatch = bcrypt.compareSync(password, user.password);

            return { user: user, passwordMatch: passwordMatch }

        }
```

```
}

export default UserService
```

Отель:

Были реализованы получение отеля по id, получение списка отеля, а также получение отфильтрованного списка по полю address.

```
import { getRepository, ILike } from 'typeorm';
import Hotel from '../../orm/models/hotels/Hotel';

class HotelService {

    async getById(id: number): Promise<Hotel> {
        const hotel = await getRepository(Hotel).findOne(id);

        if (hotel) return hotel;

        throw new Error('Hotel with specified ID is not found');
    }

    async getList(): Promise<Hotel[]> {
        const hotels = await getRepository(Hotel).find();
        return hotels;
    }

    async getFilteredList(q: string): Promise<Hotel[]> {
        const hotels = await getRepository(Hotel).find({
            where: {
                address: ILike(`%${q}%`)
            }
        });
        return hotels;
    }

}

export default HotelService
```

Бронирование:

Были реализованы получение бронирования по id, получение списка бронирований по id пользователя (внешний ключ), создание бронирования с проверкой существования отеля и пользователя по id.

```
import { getRepository } from 'typeorm';
import Booking from '../../orm/models/bookings/Booking';
import HotelService from '../hotels/Hotel';
import UserService from '../users/User';

class BookingService {

    async getById(id: number): Promise<Booking> {
        const booking = await getRepository(Booking).findOne(id);

        if (booking) return booking;

        throw new Error('Hotel with specified ID is not found');
    }

    async listByUser(userId: number): Promise<Booking[]> {
        const bookings = await getRepository(Booking).find({
            relations: ['user'],
            where: {
                user: {
                    id: userId
                }
            }
        });
        return bookings;
    }

    async create(bookingData: {
             user_id: number, hotel_id: number, starts_at: Date,
ends_at: Date, number_of_guests: number
    }): Promise<Booking> {
        try {
            let booking = new Booking()
            booking.starts_at = bookingData.starts_at
            booking.ends_at = bookingData.ends_at
            booking.number_of_guests = bookingData.number_of_guests

            const userService = new UserService
                                        const user = await
userService.getById(bookingData.user_id)

            const hotelService = new HotelService
```

```
                                                const    hotel    =    await
hotelService.getById(bookingData.hotel_id)

                booking.user = user
                booking.hotel = hotel
                booking = getRepository(Booking).create(booking)
                booking = await getRepository(Booking).save(booking)
                return booking
            } catch (e: any) {
                throw new Error(e)
            }
        }

    }

    export default BookingService
```

Контроллеры:

Отель:

```
    import Hotel from '../../orm/models/hotels/Hotel'
    import HotelService from '../../services/hotels/Hotel'

    class HotelController {
        private hotelService

        constructor() {
            this.hotelService = new HotelService
        }

        list = async (request: any, response: any) => {
            var hotels: Hotel[]
            if (request.query.q) {
                                                    hotels    =    await
this.hotelService.getFilteredList(request.query.q)
            } else {
                hotels = await this.hotelService.getList()
            }
            response.send(hotels)
        }

        retrieve = async (request: any, response: any) => {
            try {
```

```
                const hotel = await this.hotelService.getById(
                    Number(request.params.id)
                )
                response.send(hotel)
            } catch (error: any) {
                response.status(404).send({ "error": error.message })
            }
        }

    }

    export default HotelController
```

Бронирования:

```
    import BookingService from '../../services/bookings/Booking'

    class BookingController {
        private bookingService

        constructor() {
            this.bookingService = new BookingService
        }

        list = async (request: any, response: any) => {
                                    const    bookings    =    await
this.bookingService.listByUser(request.user.id)
            response.send(bookings)
        }

        retrieve = async (request: any, response: any) => {
            try {
                const hotel = await this.bookingService.getById(
                    Number(request.params.id)
                )
                response.send(hotel)
            } catch (error: any) {
                response.status(404).send({ "error": error.message })
            }
        }

        create = async (request: any, response: any) => {
            request.body.user_id = request.user.id
            try {
```

```
                                        const booking = await
this.bookingService.create(request.body);
                response.status(201).send(booking)
            } catch (error: any) {
                response.status(400).send({ "error": error.message })
            }

        }

    }

    export default BookingController
```

Роутеры:

Отель:

```
    import { Router } from 'express';
    import HotelController from "../../../controllers/hotels/Hotel";

    const router = Router();

    const controller = new HotelController()

    router.route('')
        .get(controller.list)

    router.route('/:id')
        .get(controller.retrieve)

    export default router
```

Бронирования:

```
    import { Router } from 'express';
    import                    BookingController                    from
"../../../controllers/bookings/Booking";
    import passport from "../../../middleware/passport";

    const router = Router();

    const controller = new BookingController()
```

```
router.route('')
        .post(passport.authenticate('jwt', { session: false }),
controller.create)

router.route('/my-bookings')
        .get(passport.authenticate('jwt', { session: false }),
controller.list)

router.route('/:id')
    .get(controller.retrieve)

export default router
```

Вывод

В ходе работы с помощью Express.js было создано API для веб сервиса, позволяющего пользователям забронировать проживание в отелях.