САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет
Лабораторная работа 2

Выполнила:
Герулайтите Габриэля
К33402

Проверил:
Добряков Д. И.

Санкт-Петербург
2022 г.

Задача

По выбранному варианту необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate)

Выбранный вариант для работы (номер 3) – 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 { Column, Entity, PrimaryGeneratedColumn } from 'typeorm'

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

    @Column()
    firstName!: string

    @Column()
    lastName!: string

    @Column()
    email!: string

    @Column()
    password!: string
}
```

Отель:

```
import { Column, Entity, IsNull, PrimaryGeneratedColumn } from 'typeorm'

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

    @Column()
    name!: string

    @Column()
    address!: string

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

    @Column({ nullable: true })
    description!: string
}
```

Бронирование:

```
import { Entity, OneToOne, PrimaryGeneratedColumn, JoinColumn, Column } from 'typeorm'
import { User } from '../user/user'
import { Hotel } from '../hotel/hotel'

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

    @OneToOne(() => User)
    @JoinColumn({ name: 'userId' })
    user!: User

    @Column()
    userId!: number

    @OneToOne(() => Hotel)
    hotel!: Hotel
}
```

Сервисы
Пользователя:

```
import UserError from '../errors/users/user'
import { User } from '../models/user/user'
import { getRepository } from 'typeorm'

class UserService {
    async getById(id: string) {
        const userRepository = getRepository(User)
        const user = await userRepository.findOneBy({ id: parseInt(id) })

        if (user) return user

        throw new UserError('Not found!')
    }

    async getByEmail(email: string) {
        try {
            const userRepository = getRepository(User)
            return await userRepository.findOneBy({ email })
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new UserError(errors)
        }
    }
```

Отеля

```
6   class HotelService {
7       async getById(id: string){
8           const hotelRepository = await getRepository(Hotel)
9
10          const hotel = hotelRepository.findOneBy({ id: parseInt(id) })
11
12          if (hotel) return hotel
13
14          throw new HotelError('Not found!')
15
16      }
17
18      async create(hotel: any): Promise<Hotel|Error>{
19
20          try {
21              const hotelRepository = await getRepository(Hotel)
22              const newHotel =  await hotelRepository.save(hotel)
23              return newHotel
24          } catch (e: any) {
25              console.log(e)
26              throw new HotelError(e)
27          }
28      }
29
```

Была реальзована функция поиска/фильтрации отеля по адресу (GetFiltredList):

```
    async listHotels(){
        const hotelRepository = await getRepository(Hotel)
        const hotels = await hotelRepository.find()

        if (hotels) return hotels

        throw new HotelError('Not found!')
    }

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

}

export default HotelService
```

Бронирования

Можно создавать бронирования и получать список уже существующих

```
class BookingService {
    async getById(id: string) {
        const bookingRepository = getRepository(Booking)
        const booking = await bookingRepository.findOneBy({ id: parseInt(id) })

        if (booking) return booking

        throw new BookingError('Not found!')
    }

    async create(booking: any, userId: number): Promise<Booking | Error> {
        try {
            const bookingRepository = getRepository(Booking)
            return await bookingRepository.save({ ...booking, userId })
        } catch (e: any) {
            console.error(e)
            throw new BookingError(e)
        }
    }

```

```
    async listBookings(user: any) {
        console.log(user)
        const bookingRepository = getRepository(Booking)
        const bookings = await bookingRepository.find({ where: { userId: user }, relations: [`user`] })

        if (bookings) return bookings

        throw new BookingError('Not found!')
    }
}

export default BookingService
```

Контроллеры

Аутентификация

Реализована возможность регистрации и входа. Для аутентификации используется jwt

```
import jwt from 'jsonwebtoken'
import { jwtOptions } from '../middlewares/passport'
import UserService from '../services/user'
import { v4 as uuidv4 } from "uuid"


class AuthController {
    private userService: UserService

    constructor() {
        this.userService = new UserService()
    }

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

            const user = await this.userService.getByEmail(request.body.email);

            if (user) {
                response.status(400).send({ "error": "User with specified email already exists" })
            }
            else {
                const users = await this.userService.create(request.body)
                response.status(201).send(users)
            }
        }

        catch (error: any) {
             response.status(400).send({ "error": error.message })
        }
    }
```

```
    login = async (request: any, response: any) => {
        const { body } = request

        const { email, password } = body

        try {
            const { user, passwordMatch } = await this.userService.checkPassword(email, password)

            if (passwordMatch) {
                const payload = { id: user.id }

                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)

                response.send({ accessToken })
            } else {
                throw new Error('Invalid credentials')
            }
        } catch (e: any) {
            response.status(401).send({ "error": e.message })
        }
    }

}

export default AuthController
```

Отели

```
import HotelService from "../services/hotel"
import router from "../routes/index"
import { Hotel } from "../models/hotel/hotel"

class HotelController {
    private hotelService: HotelService

    constructor() {
        this.hotelService = new HotelService()
    }

    get = async (request: any, response: any) => {
        try {
            if (request.query.q) {
                response.json(await this.hotelService.getFilteredList(request.query.q))
            } else {
                response.json(await this.hotelService.listHotels())
            }
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }

    post = async (request: any, response: any) => {
        try {
            const record = await this.hotelService.create(request.body)
            return response.json({ record, msg: 'Successfully create hotel' })
        } catch (error: any) {
            response.status(400).send({ "error": error.message })
        }
```

Бронирования

```
import BookingService from "../services/booking"

class BookingController {
    private bookingService: BookingService

    constructor() {
        this.bookingService = new BookingService()
    }

    get = async (request: any, response: any) => {
        try {
            const records = await this.bookingService.listBookings(request.user.id)
            return response.json(records);
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
    post = async (request: any, response: any) => {
        try {
            const record = await this.bookingService.create(request.body, request.user.id)
            return response.json({ record, msg: 'Successfully attend this event' })
        } catch (error: any) {
            response.status(400).send({ "error": error.message })
        }
    }
}

export default BookingController
```

Роуты

Src/routes/index.ts

```
import auth from './Auth/auth';
import booking from './Booking/booking';
import hotel from './hotel/hotel';
import user from './User/user';

const router = Router();

router.use('/auth', auth);
router.use('/user', user);
router.use('/hotel', hotel);
router.use('/booking', booking);

export default router
```

Src/routes/auth/auth.ts

```ts
import AuthController from "../../controllers/auth";
import { Router } from 'express';

const router = Router();
const controller = new AuthController()

router.post('/login',
    controller.login
)

router.post('/register',
    controller.register
)

export default router
```

Src/routes/user/user.ts

```ts
import express from "express"
import Controller from '../../controllers/user'
import { Router } from 'express'
import passport from "../../middlewares/passport"


const router: express.Router = express.Router()

const usercontroller = new Controller()

router.get('/:firstName',
  passport.authenticate('jwt', {session: false}), usercontroller.me)

router.route('/read')
  .get(usercontroller.get)

router.route('/create')
  .post(usercontroller.post)

router.route('/user/:id')
  .get(usercontroller.getbyID)

router.route('/update/:id')
  .put(usercontroller.put)

router.route('/delete/:id')
  .delete(usercontroller.delete)

export default router
```

Src/routes/booking/booking.ts

```ts
import BookingController from '../../controllers/booking'
import express from "express"
import passport from "passport"
import { Router } from 'express'


const router = Router()
const bookingcontroller = new BookingController()

router.route('/list').get(passport.authenticate('jwt', {session: false}), bookingcontroller.get)
router.route('/create').post(passport.authenticate('jwt', {session: false}), bookingcontroller.post)

export default router
```

Src/routes/hotel/hotel.ts

```ts
router.route('/list').get(hotelcontroller.get)
router.route('/create').post(hotelcontroller.post)

export default router
```

Выводы:

В ходе лабораторной работы на основе ранее реализованного boilerplate на express, TypeORM и typescript было создано API для веб-сервиса бронирования отелей.