

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет по
лабораторной работе № 2

Выполнил:
Борисов М. Е.

Группа:
К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

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

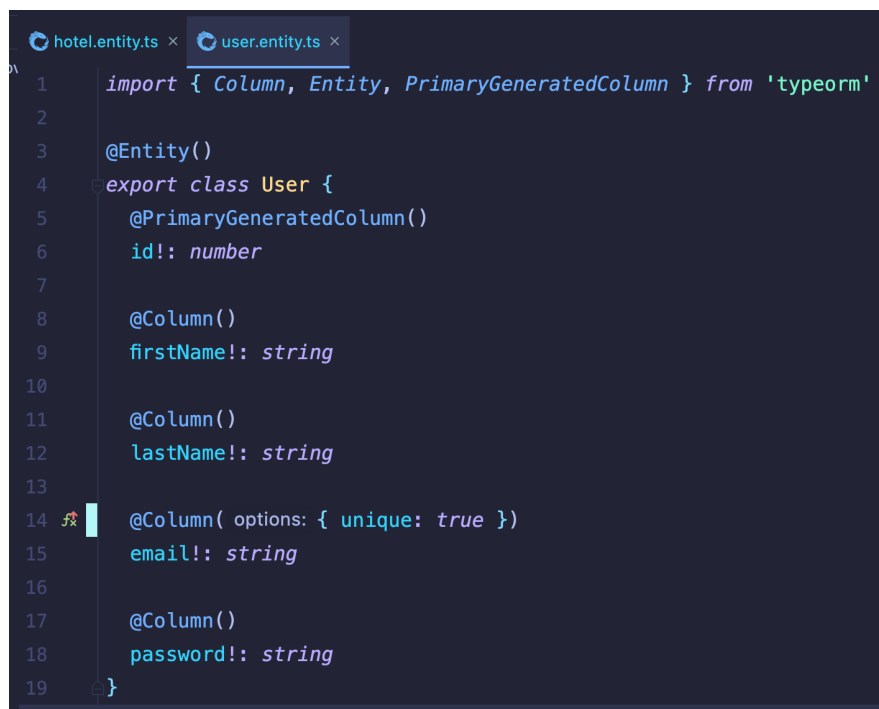
Выбранный вариант для работы (номер 3) – Платформа для поиска и бронирования номера в отеле/квартире/хостеле:

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



```
hotel.entity.ts × user.entity.ts ×
1  import { Column, Entity, PrimaryGeneratedColumn } from 'typeorm'
2
3  @Entity()
4  export class User {
5      @PrimaryGeneratedColumn()
6      id!: number
7
8      @Column()
9      firstName!: string
10
11     @Column()
12     lastName!: string
13
14     @Column( options: { unique: true })
15     email!: string
16
17     @Column()
18     password!: string
19 }
```

Отель

```
hotel.entity.ts x
1  import { Column, Entity, PrimaryGeneratedColumn } from 'typeorm'
2
3  @Entity()
4  export class Hotel {
5      @PrimaryGeneratedColumn()
6      id!: number
7
8      @Column()
9      name!: string
10
11     @Column()
12     address!: string
13
14     @Column( options: { type: 'float', nullable: true })
15     rating: number
16
17     @Column( options: { nullable: true })
18     description!: string
19
20     @Column()
21     capacity!: number
22 }
```

Бронирование

```
booking.entity.ts x
1  import { Entity, PrimaryGeneratedColumn, Column, ManyToOne } from 'typeorm'
2  import { User } from '../../users/entities/user.entity'
3  import { Hotel } from '../../hotels/entities/hotel.entity'
4
5  @Entity()
6  export class Booking {
7      @PrimaryGeneratedColumn()
8      id!: number
9
10     @ManyToOne( typeFunctionOrTarget: () => User )
11     user!: User
12
13     @ManyToOne( typeFunctionOrTarget: () => Hotel )
14     hotel!: Hotel
15
16     @Column()
17     guestsCount!: number
18
19     @Column( options: { type: 'timestamp' })
20     bookingBegin!: Date
21
22     @Column( options: { type: 'timestamp' })
23     bookingEnd!: Date
24 }
```

Сервис пользователя

```
8   @Injectable()
9   export class UsersService {
10     constructor(
11       @InjectRepository(User)
12       private usersRepository: Repository<User>,
13     ) {}
14     async create(createUserDto: CreateUserDto) {
15       return this.usersRepository.findOneBy(
16         |   await this.usersRepository.save(createUserDto),
17         |
18       )
19     }
20     findAll() {
21       |   return this.usersRepository.find()
22     }
23
24     findOneById(id: number) {
25       |   return this.usersRepository.findOneBy( where: { id })
26     }
27
28     findOneByEmail(email: string) {
29       |   return this.usersRepository.findOneBy( where: { email })
30     }
31
32     findOneByEmailAndPassword(email: string, password: string) {
33       |   return this.usersRepository.findOneBy( where: { email, password })
34     }
35
36     async update(id: number, updateUserDto: UpdateUserDto) {
37       |   return this.usersRepository.findOneBy(
38       |   |   await this.usersRepository.save( entity: { id, ...updateUserDto } ),
39       |   )
40     }
41
42     remove(id: number) {
43       |   return this.usersRepository.delete(id)
44     }
45   }
```

Сервис отеля

```
8   @Injectable()
9   export class HotelsService {
10     constructor(
11       @InjectRepository(Hotel)
12       private hotelsRepository: Repository<Hotel>,
13     ) {}
14     async create(createHotelDto: CreateHotelDto) {
15       return this.hotelsRepository.findOneBy(
16         await this.hotelsRepository.save(createHotelDto),
17       )
18     }
19
20     findAll({ address, capacity }: { address?: string; capacity?: number }) {
21       return this.hotelsRepository.find( options: {
22         where: {
23           address: address ? ILike( value: `%${address}%` ) : undefined,
24           capacity: capacity ? MoreThanOrEqual(capacity) : undefined,
25         },
26       })
27     }
28
29     findOne(id: number) {
30       return this.hotelsRepository.findOneBy( where: { id })
31     }
32
33     async update(id: number, updateHotelDto: UpdateHotelDto) {
34       return this.hotelsRepository.findOneBy(
35         await this.hotelsRepository.save( entity: { id, ...updateHotelDto },
36       )
37     }
38
39     remove(id: number) {
40       return this.hotelsRepository.delete(id)
41     }
42   }
```

Для отеля был реализован поиск по названию и вместимости.

Сервис бронирования

```
10  @Injectable()
11  export class BookingsService {
12    constructor(
13      @InjectRepository(Booking)
14      private bookingsRepository: Repository<Booking>,
15      @InjectRepository(Hotel)
16      private hotelsRepository: Repository<Hotel>,
17      @InjectRepository(User)
18      private usersRepository: Repository<User>,
19    ) {}
20    async create({ user, hotel, ...booking }: CreateBookingDto) {
21      const userEntity = await this.usersRepository.findOneBy( where: { id: user })
22      const hotelEntity = await this.hotelsRepository.findOneBy( where: { id: hotel })
23      const newBooking = await this.bookingsRepository.save( entity: {
24        user: userEntity,
25        hotel: hotelEntity,
26        ...booking,
27      })
28      return this.bookingsRepository.findOne( options: {
29        relations: ['user', 'hotel'],
30        where: { id: newBooking.id },
31      })
32    }
33
34    findAll() {
35      return this.bookingsRepository.find( options: { relations: ['user', 'hotel'] })
36    }
37
38    findOne(id: number) {
39      return this.bookingsRepository.findOne( options: {
40        relations: ['user', 'hotel'],
41        where: { id },
42      })
43    }
44
45    async update(id: number, { user, hotel, ...booking }: UpdateBookingDto) {
46      const userEntity = await this.usersRepository.findOneBy( where: { id: user })
47      const hotelEntity = await this.hotelsRepository.findOneBy( where: { id: hotel })
48      const newBooking = await this.bookingsRepository.save( entity: {
49        user: userEntity,
50        hotel: hotelEntity,
51        ...booking,
52        id,
53      })
54      return this.bookingsRepository.findOne( options: {
55        relations: ['user', 'hotel'],
56        where: { id: newBooking.id },
57      })
58    }
59  }
```

Реализована возможность регистрации и входа. Для аутентификации используется jwt библиотека passport

Контроллер

```
17  @ApiTags( tags: `auth`)
18  @Controller( prefix: `auth`)
19  export class AuthController {
20      constructor(private readonly authService: AuthService) {}
21
22      @UseGuards(LocalAuthGuard)
23      @HttpCode( statusCode: 200)
24      @Post( path: `/login`)
25      login(@Body() obtainTokenPairDto: ObtainTokenPairDto, @Request() req) {
26          return this.authService.obtainTokenPair(req.user)
27      }
28
29      @HttpCode( statusCode: 200)
30      @Post( path: `/refresh`)
31      refresh(@Body() refreshTokenPairDto: RefreshTokenPairDto) {
32          try {
33              return this.authService.refreshTokenPair(refreshTokenPairDto)
34          } catch (error) {
35              console.error(error)
36              throw new UnauthorizedException()
37          }
38      }
39  }
```

Сервис авторизации

```
10  @Injectable()
11  export class AuthService {
12    constructor(
13      private usersService: UsersService,
14      private jwtService: JwtService,
15    ) {}
16
17    private jwtAccessSecret = process.env.JWT_ACCESS_SECRET
18    private jwtRefreshSecret = process.env.JWT_REFRESH_SECRET
19
20    async validateLocal({ email, password }: ObtainTokenPairDto) {
21      return await this.usersService.findOneByEmailAndPassword(email, password)
22    }
23
24    obtainTokenPair(user: User): TokenPair {
25      return {
26        access: this.jwtService.sign(
27          payload: { ...user },
28          options: {
29            expiresIn: '1h',
30            secret: this.jwtAccessSecret,
31          },
32        ),
33        refresh: this.jwtService.sign(
34          payload: { ...user },
35          options: {
36            expiresIn: '30d',
37            secret: this.jwtRefreshSecret,
38          },
39        ),
40      }
41    }
42
43    validateJWT({ email }: User) {
44      return this.usersService.findOneByEmail(email)
45    }
46
47    refreshTokenPair({ refresh }: RefreshTokenPairDto): TokenPair {
48      const user = this.jwtService.verify<User & { exp: number; iat: number }>(
49        refresh,
50        options: {
51          secret: this.jwtRefreshSecret,
52        },
53      )
54      delete user.exp
55      delete user.iat
56
57      return this.obtainTokenPair(user)
58    }
59  },
```


Вывод

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