САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-енд разработка

Отчет

Лабораторная работа № 2

Выполнил:

Легин Денис

Группа

K33402

Проверил:
Добряков Д.И.

Санкт-Петербург

2022 г.

Задача

Для выполнения работы я выбрал вариант «Платформа для поиска и бронирования номера в отеле/квартире/хостеле». В ходе работы необходимо реализовать RESTful API средствами express + typescript, используя ранее написанный boilerplate.

Ход работы

Core:

```
export default class App {
    public port: number
    public host: string

    private readonly app: express.Application
    private server: Server

    constructor(port : number  = 8080, host : string  = "localhost") {
        this.port = port
        this.host = host

        this.app = this.createApp()
        this.server = this.createServer()
    }

    private createApp(): express.Application {
        const app = express()
        const bodyParser = require('body-parser')
        const passport = require('passport')
        const passportJwt = require('passport-jwt')
        const ExtractJwt = passportJwt.ExtractJwt
        const JwtStrategy = passportJwt.Strategy
        jwtOptions.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken()
        jwtOptions.secretOrKey = 'test101'
        // tslint:disable-next-line:variable-name only-arrow-functions
        const strategy = new JwtStrategy(jwtOptions, async function(jwt_payload: any, next: any) {
            // tslint:disable-next-line:no-console
          console.log('payload received', jwt_payload)
          const service = new UserService()
          const user = await service.getById(jwt_payload.id)
          if (user) {
            next(null, user)
          } else {
            next(null, false)
          }
        });
        passport.use(strategy)
        app.use(bodyParser.urlencoded({ extended: false }))
```

```
        app.use(bodyParser.json())
        app.use(passport.initialize())
        app.use('/v1', routes)
        return app
      }

    private createServer(): Server {
        return createServer(this.app)
    }

    public start(): void {
        sequelize.sync().then(() => {
          // tslint:disable-next-line:no-console
          console.log('DB connected')
        })
        this.server.listen(this.port, listeningListener: () => {
            // tslint:disable-next-line:no-console
            console.log(`Running server on port ${this.port}`)
        })
    }
}
```

Модель бронирования:

```
@Table
export default class Booking extends Model {
  @IsDate
  @Column
  startDate: Date

  @IsDate
  @Column
  endDate: Date

  @Min( limit: 1)
  @Column
  capacity: number

  @ForeignKey( relatedClassGetter: () => User)
  @Column
  userId: number

  @BelongsTo( associatedClassGetter: () => User)
  user: User

  @ForeignKey( relatedClassGetter: () => Hotel)
  @Column
  hotelId: number

  @BelongsTo( associatedClassGetter: () => Hotel)
  hotel: Hotel
}
```

Сервис бронирования:

```
export default class BookingService {

    private repo = sequelize.getRepository(Booking)

    add(booking: any) {
        return this.repo.create(booking)
    }

    getForUser(user: number) {
        return this.repo.findAll( options: { where: { userId: user } })
    }
}
```

Контроллер бронирования:

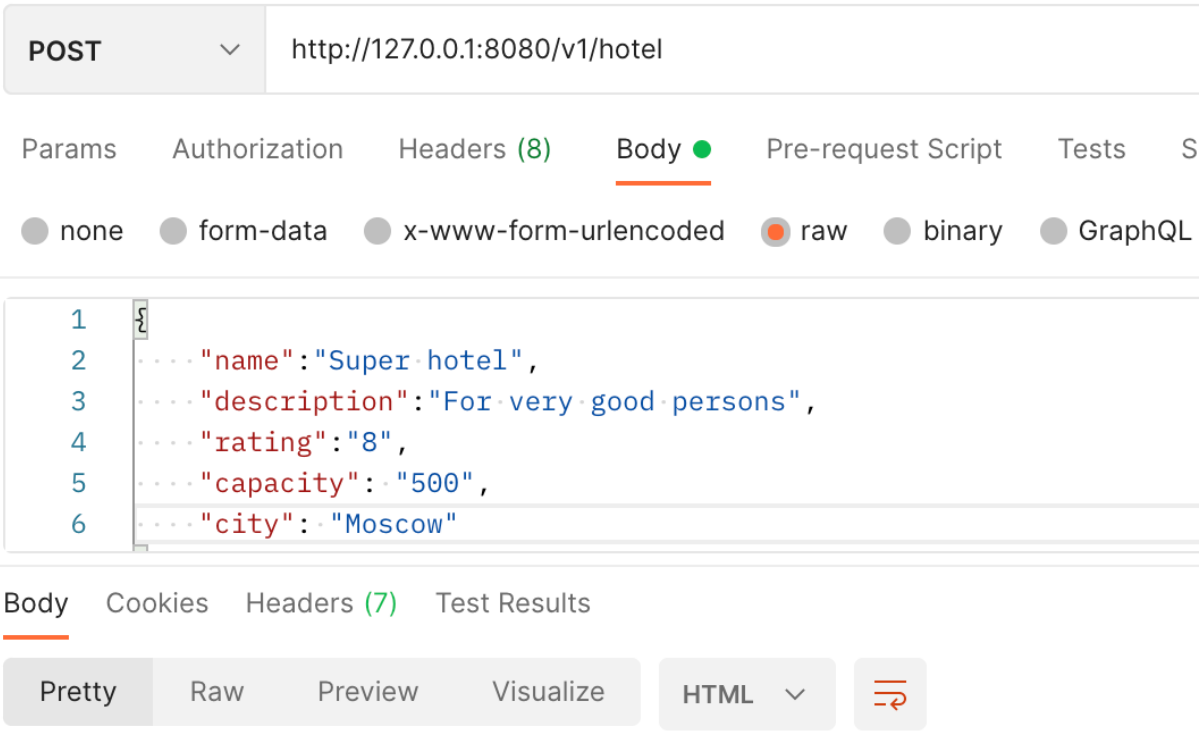
```
export default class BookingController {

    private service = new BookingService()

    post = async (request: any, response: any) => {
        try {
            const booking = request.body
            booking.userId = request.user.id
            await this.service.add(booking)
            response.send('Successfully added')
        } catch (error: any) {
            response.status(400).send(error.message)
        }
    }

    get = async (request: any, response: any) => {
        try {
            const data = await this.service.getForUser(request.user.id)
            response.send(data)
        } catch (error: any) {
            response.status(400).send(error.message)
        }
    }
}
```

Настройка рутов:

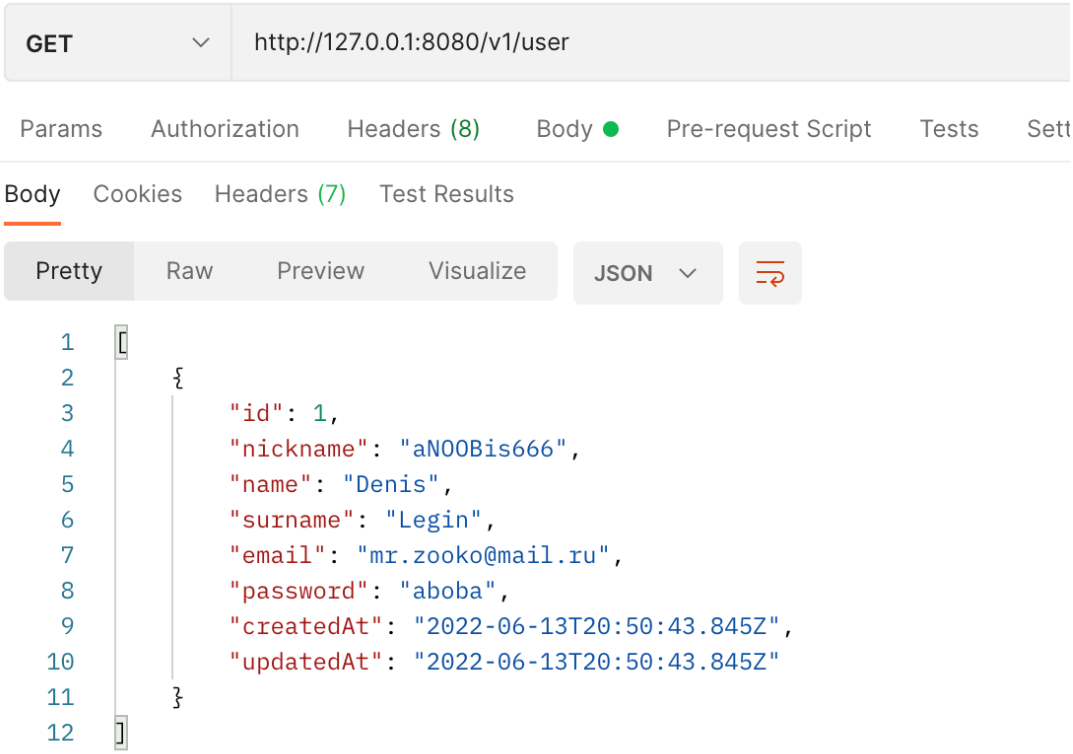
```
const router: express.Router = express.Router()
// tslint:disable-next-line:no-var-requires
const passport = require('passport')

const userController = new UserController()
const hotelController = new HotelController()
const bookingController = new BookingController()
const authController = new AuthController()

router
  .route( prefix: '/user')
  .get(userController.get)
  .post(userController.post)

router
  .route( prefix: '/hotel')
  .get(hotelController.get)
  .post(hotelController.post)

router
  .route( prefix: '/booking')
  .get(passport.authenticate( options: 'jwt', { session: false }), bookingController.get)
  .post(passport.authenticate( options: 'jwt', { session: false }), bookingController.post)

router
  .route( prefix: '/auth')
  .post(authController.post)

export default router
```

Пример запросов:

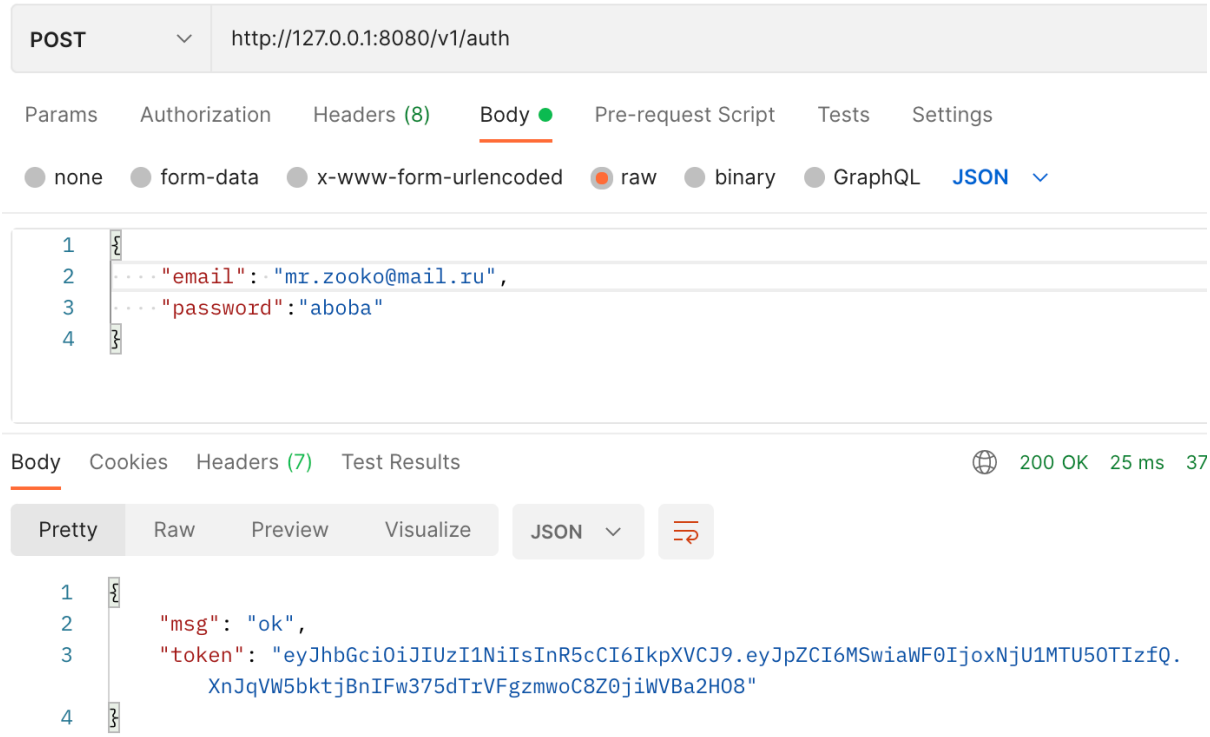
POST-hotel-request



GET-user-request



POST-auth-request



Вывод:

В ходе работы был реализован RESTful API средствами Express, Typescript и Sequelize. Для авторизация использовался Passport и Passport JWT.