САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

по лабораторной работе №2

Выполнил:

Поляков Андрей

Группа
К33402

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

Файл core:

```typescript
1   import express from "express"
2   import { createServer, Server } from "http"
3   import routes from "../routes/index"
4   import UserService from '../services/user/index'
5   import { sequelize } from '../config/config'
6
7   export const jwtOptions: any = {}
8
9   export default class App {
10      public port: number
11      public host: string
12
13      private app: express.Application
14      private server: Server
15
16      constructor(port = 8080, host = "localhost") {
17          this.port = port
18          this.host = host
19
20          this.app = this.createApp()
21          this.server = this.createServer()
22      }
23
24      private createApp(): express.Application {
25          const app = express()
26          const bodyParser = require('body-parser')
27          const passport = require('passport')
28          const passportJwt = require('passport-jwt')
29          let ExtractJwt = passportJwt.ExtractJwt
30          let JwtStrategy = passportJwt.Strategy
31          jwtOptions.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken()
32          jwtOptions.secretOrKey = 'test123'
33          let strategy = new JwtStrategy(jwtOptions, async function(jwt_payload: any, next: any) {
34            console.log('payload received', jwt_payload)
35            const service = new UserService()
36            let user = await service.getById(jwt_payload.id)
37            if (user) {
38              next(null, user)
39            } else {
40              next(null, false)
41            }
42          });
43          passport.use(strategy)
44          app.use(bodyParser.urlencoded({ extended: false }))
45          app.use(bodyParser.json())
46          app.use(passport.initialize())
47          app.use('/v1', routes)
48          return app
49        }
50
51      private createServer(): Server {
52          return createServer(this.app)
53      }
54
55      public start(): void {
56          sequelize.sync().then(() => {
57            console.log('DB connected')
58          })
59          this.server.listen(this.port, () => {
60              console.log(`Running server on port ${this.port}`)
61          })
62      }
63  }
```

Модель отеля:

```
1   import express from "express"
2   import { createServer, Server } from "http"
3   import routes from "../routes/index"
4   import UserService from '../services/user/index'
5   import { sequelize } from '../config/config'
6
7   export const jwtOptions: any = {}
8
9   export default class App {
10      public port: number
11      public host: string
12
13      private app: express.Application
14      private server: Server
15
16      constructor(port = 8080, host = "localhost") {
17          this.port = port
18          this.host = host
19
20          this.app = this.createApp()
21          this.server = this.createServer()
22      }
23
24      private createApp(): express.Application {
25          const app = express()
26          const bodyParser = require('body-parser')
27          const passport = require('passport')
28          const passportJwt = require('passport-jwt')
29          let ExtractJwt = passportJwt.ExtractJwt
30          let JwtStrategy = passportJwt.Strategy
31          jwtOptions.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken()
32          jwtOptions.secretOrKey = 'test123'
33          let strategy = new JwtStrategy(jwtOptions, async function(jwt_payload: any, next: any) {
34              console.log('payload received', jwt_payload)
35              const service = new UserService()
36              let user = await service.getById(jwt_payload.id)
37              if (user) {
38                  next(null, user)
39              } else {
40                  next(null, false)
41              }
42          });
43          passport.use(strategy)
44          app.use(bodyParser.urlencoded({ extended: false }))
45          app.use(bodyParser.json())
```

Модель бронирования:

```
1   import { Table, Column, Model, IsDate, Min, ForeignKey, BelongsTo } from 'sequelize-typescript'
2   import User from '../user/User'
3   import Hotel from '../hotel/Hotel'
4
5   @Table
6   export default class Booking extends Model {
7     @IsDate
8     @Column
9     startDate: Date
10
11    @IsDate
12    @Column
13    endDate: Date
14
15    @Min(1)
16    @Column
17    capacity: number
18
19    @ForeignKey(() => User)
20    @Column
21    userId: number
22
23    @BelongsTo(() => User)
24    user: User
25
26    @ForeignKey(() => Hotel)
27    @Column
28    hotelId: number
29
30    @BelongsTo(() => Hotel)
31    hotel: Hotel
32  }
```

Модель пользователя:

```typescript
import { Table, Column, Model, HasMany } from 'sequelize-typescript'
import Booking from '../booking/Booking'

@Table
export default class User extends Model {
  @Column
  nickname: string

  @Column
  name: string

  @Column
  surname: string

  @Column
  email: string

  @Column
  password: string

  @HasMany(() => Booking)
  bookings: Booking[]
}
```

Настройка роутов:

```typescript
import express from "express"
import UserController from '../controllers/user/index'
import HotelController from '../controllers/hotel/index'
import BookingController from '../controllers/booking/index'
import AuthController from "../controllers/auth/index"

const router: express.Router = express.Router()
const passport = require('passport')

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

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

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

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

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

export default router
```

Сервис, для получения данных о бронированиях:

```
1    import Booking from '../../models/booking/Booking'
2    import { sequelize } from '../../config/config'
3
4    export default class BookingService {
5
6        private repo = sequelize.getRepository(Booking)
7
8        add(booking: any) {
9            return this.repo.create(booking)
10       }
11
12       getForUser(user: number) {
13           return this.repo.findAll({ where: { userId: user } })
14       }
15   }
```

Контроллер для бронирования:

```
1    import BookingService from '../../services/booking/index'
2
3    export default class BookingController {
4
5        private service = new BookingService()
6
7        post = async (request: any, response: any) => {
8            try {
9                const booking = request.body
10               booking.userId = request.user.id
11               await this.service.add(booking)
12               response.send('Successfully added')
13           } catch (error: any) {
14               response.status(400).send(error.message)
15           }
16       }
17
18       get = async (request: any, response: any) => {
19           try {
20               const data = await this.service.getForUser(request.user.id)
21               response.send(data)
22           } catch (error: any) {
23               response.status(400).send(error.message)
24           }
25       }
26   }
```

Примеры работы:

POST ⌄ http://127.0.0.1:8080/v1/user

Params   Authorization   Headers (8)   Body ●

◯ none   ◯ form-data   ◯ x-www-form-urlencoded

```
1  {
2      "nickname":"test01",
3      "name": "testName",
4      "surname": "testSurname",
5      "password":"test101",
6      "email":"test01@gmail.com"
7  }
```

Body   Cookies   Headers (7)   Test Results

Pretty   Raw   Preview   Visualize   HTI

```
1  Added'test01'user
```

http://127.0.0.1:8080/v1/user

GET ⌄ http://127.0.0.1:8080/v1/user

Params   Authorization   Headers (6)   Body   Pre-request Script   Tests

Query Params

KEY	VALUE
Key	Value

Body   Cookies   Headers (7)   Test Results

Pretty   Raw   Preview   Visualize   JSON ⌄   ⇥

```
1   [
2       {
3           "id": 1,
4           "nickname": "test01",
5           "name": "testName",
6           "surname": "testSurname",
7           "email": "test01@gmail.com",
8           "password": "test101",
9           "createdAt": "2022-06-06T09:50:39.986Z",
10          "updatedAt": "2022-06-06T09:50:39.986Z"
11      }
12  ]
```

POST ⌄ http://127.0.0.1:8080/v1/hotel

Params   Authorization   Headers (8)   Body ●

● none   ● form-data   ● x-www-form-urlencoded

```
2    "name":"Star",
3    "description":"Best hotel!",
4    "rating": 9.9,
5    "capacity": 120,
6    "city": "Saint-Petersburg"
7  }
```

Body   Cookies   Headers (7)   Test Results

Pretty   Raw   Preview   Visualize   HT

```
1    Successfully added
```

GET ⌄ http://127.0.0.1:8080/v1/hotel?city=Saint-Petersburg&rating=7

Params ●   Authorization   Headers (6)   Body   Pre-request Script   Tests

Query Params

	KEY	VALUE
☑	city	Saint-Petersburg
☑	rating	7
	Key	Value

Body   Cookies   Headers (7)   Test Results

Pretty   Raw   Preview   Visualize   JSON ⌄

```
1  [
2    {
3      "id": 1,
4      "name": "Star",
5      "description": "Best hotel!",
6      "rating": 9.9,
7      "capacity": 120,
8      "city": "Saint-Petersburg",
9      "createdAt": "2022-06-06T09:53:26.585Z",
10     "updatedAt": "2022-06-06T09:53:26.585Z"
11   }
12 ]
```

POST ⌄ http://127.0.0.1:8080/v1/auth

Params   Authorization   Headers (8)   Body ●   Pre-request Script   Tests   Settings

○ none   ○ form-data   ○ x-www-form-urlencoded   ● raw   ○ binary   ○ GraphQL   **JSON** ⌄

```
1  {
2  ····"email":"test01@gmail.com",
3  ····"password":"test101"
4  }
```

Body   Cookies   Headers (7)   Test Results   🌐   200 OK   8 ms   374 B

Pretty   Raw   Preview   Visualize   JSON ⌄   ⇄

```
1  {
2      "msg": "ok",
3      "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNjU0NTA5NzQ2fQ.
           zey2Chn8fC-3XbbyYMS-oQjHrdxb4n4TErZOZ4MZ2qA"
4  }
```

http://127.0.0.1:8080/v1/booking

POST ⌄ http://127.0.0.1:8080/v1/booking

Params   Authorization ●   Headers (9)   Body ●   Pre-request

○ none   ○ form-data   ○ x-www-form-urlencoded   ● raw   ○ bi

```
1  {
2  ····"startDate":"02.06.2022",
3  ····"endDate":"05.06.2022",
4  ····"capacity":·3,
5  ····"hotelId":·1
6  }
```

Body   Cookies   Headers (7)   Test Results

Pretty   Raw   Preview   Visualize   HTML ⌄   ⇄

```
1  Successfully added
```

Вывод

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