

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэкенд-энд разработка

Отчет

Лабораторная работа 2

Выполнил:

Омар Сизей

Группа: K33412

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

По выбранному варианту нужно реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate). Вариант - платформа для поиска профессиональных мероприятий.

Ход работы

Модели (модель Event):

```
s Event.ts  ×
models > event > TS Event.ts > ...
1  import { Table, Column, Model, IsDate, HasMany } from 'sequelize-typescript'
2  import Ticket from '../ticket/ticket'
3
4  @Table
5  export default class Event extends Model {
6    @Column
7    name: string
8
9    @Column
10   info: string
11
12   @IsDate
13   @Column
14   date: Date
15
16   @Column
17   city: string
18
19   @Column
20   type: string
21
22   @HasMany(() => Ticket)
23   tickets: Ticket[]
24 }
25
26
```

Модели (модель User):

```

s User.ts ×
models > user > TS User.ts > ...
1  import { Table, Column, Model, HasMany } from 'sequelize-typescript'
2  import Ticket from '../ticket/ticket'
3
4  @Table
5  export default class User extends Model {
6      @Column
7      name: string
8
9      @Column
10     email: string
11
12     @Column
13     password: string
14
15     @HasMany(() => Ticket)
16     tickets: Ticket[]
17 }
18
19

```

Модели (модель ticket):

```

models > ticket > TS ticket.ts > ...
1  import { Table, Column, Model, Min, ForeignKey, BelongsTo } from 'sequelize-typescript'
2  import User from '../user/User'
3  import Event from '../event/Event'
4
5  @Table
6  export default class Ticket extends Model {
7      @Min(1)
8      @Column
9      attendants: number
10
11     @ForeignKey(() => User)
12     @Column
13     userId: number
14
15     @BelongsTo(() => User)
16     user: User
17
18     @ForeignKey(() => Event)
19     @Column
20     eventId: number
21
22     @BelongsTo(() => Event)
23     event: Event
24 }
25
26

```

Роуты:

```
TS index.ts ×
routes > TS index.ts > ...
1 import express from "express"
2 import AuthController from "../controllers/auth/auth"
3 import UserController from "../controllers/user/user"
4 import EventController from "../controllers/event/event"
5 import TicketController from "../controllers/ticket/ticket"
6
7 const router: express.Router = express.Router()
8 const passport = require('passport')
9
10 const authController = new AuthController()
11 const userController = new UserController()
12 const eventController = new EventController()
13 const ticketController = new TicketController()
14
15 router.route('/login').post(authController.login)
16 router.route('/getUsers').get(userController.get)
17 router.route('/addUser').post(userController.add)
18
19 router.route('/getAllEvents').get(eventController.getAll)
20 router.route('/getEvents').get(eventController.getFiltered)
21 router.route('/addEvent').post(eventController.add)
22
23 router.route('/getTickets').get(passport.authenticate('jwt', { session: false }), ticketController.get)
24 router.route('/addTicket').post(passport.authenticate('jwt', { session: false }), ticketController.add)
25
26 export default router
27
```

Services:

Сервис для работы с событиями:

```
TS EventService.ts ×
services > event > TS EventService.ts > ...
1 import Event from '../models/event/Event'
2 import { sequelize } from '../config/config'
3
4 export default class EventService {
5
6     private repo = sequelize.getRepository(Event)
7
8     add(event: any) {
9         return this.repo.create(event)
10     }
11
12     getAll() {
13         return this.repo.findAll()
14     }
15
16     getByFilter(city_param: string, type_param: string) {
17         return this.repo.findAll( { where: { city: city_param, type: type_param } } )
18     }
19 }
20
```

Сервис для Users:

```
TS UserService.ts X
services > user > TS UserService.ts > ...
1 import User from '../../models/user/User'
2 import { sequelize } from '../../config/config'
3
4 export default class UserService {
5
6     private repo = sequelize.getRepository(User)
7
8     add(user: any) {
9         return this.repo.create(user)
10    }
11
12    getAll() {
13        return this.repo.findAll()
14    }
15
16    getByEmail(email_param: string) {
17        return this.repo.findOne({ where: { email: email_param } })
18    }
19
20    getById(id_param: (property) UserService.repo: Repository<User>) {
21        return this.repo.findOne({ where: { id: id_param } })
22    }
23 }
24
25
```

Сервис для Tickets:

```
TS TicketService.ts X
services > ticket > TS TicketService.ts > ...
1 import Ticket from '../../models/ticket/ticket'
2 import { sequelize } from '../../config/config'
3
4 export default class TicketService {
5
6     private repo = sequelize.getRepository(Ticket)
7
8     add(ticket: any) (property) TicketService.repo: Repository<Ticket>
9         return this.repo.create(ticket)
10    }
11
12    getForUser(user: number) {
13        return this.repo.findAll({ where: { userId: user } })
14    }
15 }
16
17
```

Контроллеры:

Контроллер для работы с событиями:

```

TS event.ts  X
controllers > event > TS event.ts > ...
1  import EventService from '../../services/event/EventService'
2
3  export default class EventController {
4
5      private service = new EventService()
6
7      add = async (request: any, response: any) => {
8          try {
9              const result = await this.service.add(request.body)
10             response.send({ id: result.id })
11         } catch (error: any) {
12             response.status(400).send(error.message)
13         }
14     }
15
16     getAll = async (request: any, response: any) => {
17         try {
18             const data = await this.service.getAll()
19             response.send(data)
20         } catch (error: any) {
21             response.status(400).send(error.message)
22         }
23     }
24
25     getFiltered = async (request: any, response: any) => {
26         try {
27             const data = await this.service.getByFilter(request.query.city, request.query.type)
28             response.send(data)
29         } catch (error: any) {
30             response.status(400).send(error.message)
31         }
32     }
33 }
34

```

Контроллер для users:

```

TS user.ts  X
controllers > user > TS user.ts > ...
1  import UserService from '../../services/user/UserService'
2
3  export default class UserController {
4
5      private service = new UserService()
6
7      add = async (request: any, response: any) => {
8          try {
9              const result = await this.service.add(request.body)
10             response.send({ id: result.id })
11         } catch (error: any) {
12             response.status(400).send(error.message)
13         }
14     }
15
16     get = async (request: any, response: any) => {
17         try {
18             const data = await this.service.getAll()
19             response.send(data)
20         } catch (error: any) {
21             response.status(400).send(error.message)
22         }
23     }
24 }
25

```

Контроллер для tickets:

```
TS ticket.ts ×
controllers > ticket > TS ticket.ts > ...
1 import TicketService from '../services/ticket/TicketService'
2
3 export default class TicketController {
4
5     private service = new TicketService()
6
7     add = async (request: any, response: any) => {
8         try {
9             const ticket = request.body
10            ticket.userId = request.user.id
11            const result = await this.service.add(ticket)
12            response.send({ id: result.id })
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 }
27
```

Middleware для авторизации:

```
TS index.ts ×
middleware > TS index.ts > ...
1 import UserService from '../services/user/UserService'
2
3 export const passport = require('passport')
4 const passportJwt = require('passport-jwt')
5 const secretKey = "secretKey"
6
7 let JwtStrategy: any = passportJwt.Strategy
8 let JwtStrategy = passportJwt.Strategy
9
10 export const options = {
11     jwtFromRequest: passportJwt.ExtractJwt.fromAuthHeaderAsBearerToken(),
12     secretOrKey: secretKey
13 }
14
15 let strategy = new JwtStrategy(options, async function(jwt_payload: any, next: any) {
16     const service = new UserService()
17     let user = await service.getById(jwt_payload.id)
18
19     if (user) {
20         next(null, user)
21     } else {
22         next(null, false)
23     }
24 })
25 passport.use(strategy)
```

Вывод

В этой лабораторной работе был разработан бэкендсервиса для поиска мероприятий с возможностью регистрации, авторизации и просмотра мероприятий пользователя.