**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

# Лабораторная работа №2

Выполнила:

Мухина Юлия

Группа К33401

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

**Задача**

Выбранный вариант: Платформа для поиска профессиональных мероприятий (пример: https://www.meetup.com/ru-RU/)

- Вход
- Регистрация
- Поиск мероприятия (фильтрации по типу мероприятия, месту проведения)
- Календарь ближайших мероприятий
- Промо-страница для организаторов мероприятия
- Личный кабинет пользователя со списком мероприятий, на которые он записывался

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

**Ход работы**

Добавлена модель события.

```
import {Table, Column, Model, AllowNull, DataType} from 'sequelize-typescript'

enum EventType {
    CONFERENCE = 'conference',
    WORKSHOP = 'workshop',
    OTHER = 'other'
}

@Table({
    createdAt: false,
    updatedAt: false,
    indexes: [{unique: true, fields: ['title', 'type', 'date']}]
})
class Event extends Model {
    @AllowNull(false)
    @Column
    title: string

    @AllowNull(false)
    @Column
    description: string

    @Column({
        defaultValue: EventType.OTHER,
        type: DataType.ENUM(...Object.values(EventType)),
    })
    type!: EventType;
```

```
28
29      @AllowNull(false)
30      @Column({
31          type: DataType.DATEONLY,
32      })
33      date: Date
34
35      @AllowNull(false)
36      @Column
37      address: string
38  }
39
40  export default Event
```

Модель регистрации.

```
1  import {Table, Model, Column, ForeignKey} from "sequelize-typescript";
2  import Event from "./event.model";
3  import User from "./user.model";
4
5  @Table({indexes: [{ unique: true, fields: ['userId', 'eventId']}]})
6  class Registration extends Model {
7      @ForeignKey(() => User)
8      @Column
9      userId: number
10
11     @ForeignKey(() => Event)
12     @Column
13     eventId: number
14 }
15
16 export default Registration
```

Метод получения мероприятия по id и получения всех мероприятий, также можем передать параметры, отфильтровать мероприятия и настроить порядок; методы создания, редактирования и удаления мероприятия.

```ts
import Event from '../models/event.model'

class EventService {
    async getById(id: number): Promise<Event> {
        const event: Event | null = await Event.findByPk(id)
        if (event) return event.toJSON()
        throw new Error('Not found!')
    }

    async getAll(queryparams: any): Promise<Event[]> {
        const {type, sort, order} = queryparams
        const options = {where: {}}
        try {
            if (sort && order) {
                // @ts-ignore
                options['order'] = [[sort, order]]
            }
            if (type) {
                // @ts-ignore
                options['where']['type'] = type
            }
            return await Event.findAll(options)
        } catch (e: any) {
            throw new Error(e.message)
        }
    }

```

```
28      async create(data: any): Promise<Event | Error> {
29          try {
30              const event = await Event.create(data)
31              return event.toJSON()
32          } catch (e: any) {
33              throw new Error(e.message)
34          }
35      }
36
37      async update(id: number, data: object): Promise<Event | Error> {
38          try {
39              await this.getById(id)
40              await Event.update(data, {where: {id}})
41              return await this.getById(id)
42          } catch (e: any) {
43              throw new Error(e.message)
44          }
45      }
46
47      async delete(id: number): Promise<Event> {
48          try {
49              const event: Event = await this.getById(id)
50              await Event.destroy({where: {id}})
51              return event
52          } catch (e: any) {
53              throw new Error(e.message)
54          }
55      }
56  }

56  }
57
58  export default EventService
```

Реализованы методы получения регистрации по id, получения количества регистраций, получения списка регистраций для пользователя, создание и удаление регистраций.

```ts
import Registration from '../models/registration.model'

class RegistrationService {
    async getById(id: number): Promise<Registration> {
        const registration: Registration | null = await Registration.findByPk(id)
        if (registration) return registration.toJSON()
        throw new Error('Not found!')
    }

    async getAllByEvent(eventId: number): Promise<number> {
        return await Registration.count({
            where: {eventId: eventId},
        })
    }

    async getAllByUser(userId: number): Promise<Registration[]> {
        return await Registration.findAll({
            where: {userId: userId},
            attributes: ['eventId']
        })
    }

    async create(eventId: number, userId: number) {
        try {
            await Registration.create({eventId: eventId, userId: userId})
        } catch (e: any) {
            throw new Error(e.message)
        }
    }


    async delete(eventId: number, userId: number) {
        try {
            await Registration.destroy({where: {eventId: eventId, userId: userId}})
        } catch (e: any) {
            throw new Error(e.message)
        }
    }
}

export default RegistrationService
```

Контроллер мероприятий

```ts
import Event from "../models/event.model"
import EventService from "../services/event.service"
import RegistrationService from "../services/registration.service";

class EventController {
    private eventService: EventService

    constructor() {
        this.eventService = new EventService()
    }

    get = async (request: any, response: any) => {
        try {
            const event: Event | Error = await this.eventService.getById(Number(request.params.id))
            const registrationService = new RegistrationService()
            const registrations = await registrationService.getAllByEvent(Number(request.params.id))
            response.status(200).json({message: "Success", event: event, registrations: registrations})
        } catch (error: any) {
            response.status(404).send({"error": error.message})
        }
    }

    getAll = async (request: any, response: any) => {
        try {
            const event: Event[] | Error = await this.eventService.getAll(request.query)
            response.status(200).json({message: "Success", data: event})
        } catch (error: any) {
            response.status(404).send({"error": error.message})
        }

    }

    create = async (request: any, response: any) => {
        const {body} = request
        try {
            const event: Event | Error = await this.eventService.create(body)
            response.status(201).json({message: "Success", data: event})
        } catch (error: any) {
            response.status(400).send({"error": error.message})
        }
    }

    delete = async (request: any, response: any) => {
        try {
            const event: Event | Error = await this.eventService.delete(Number(request.params.id))
            response.status(200).json({message: "Success", data: event})
        } catch (error: any) {
            response.status(404).send({"error": error.message})
        }
    }


    update = async (request: any, response: any) => {
        const {body} = request
        try {
            const event: Event | Error = await this.eventService.update(Number(request.params.id), body)
            response.status(203).json({message: "Success", data: event})
        } catch (error: any) {
            response.status(400).send({"error": error.message})
        }
    }
}

export default EventController
```

Немного изменённый контроллер юзера

```ts
import User from '../models/user.model'
import UserService from '../services/user.service'
import jwt from 'jsonwebtoken'
import {jwtOptions} from '../middlewares/passport'
import TokenService from '../services/token.service'
import RegistrationService from "../services/registration.service";

class UserController {
    private userService: UserService

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

    get = async (request: any, response: any) => {
        try {
            const user: User | Error = await this.userService.getById(
                Number(request.params.id)
            )
            response.send(user)
        } catch (error: any) {
            response.status(404).send({"error": error.message})
        }
    }

    post = async (request: any, response: any) => {
        const {body} = request
        try {
            const user: User | Error = await this.userService.create(body)
            response.status(201).send(user)
        } catch (error: any) {
            response.status(400).send({"error": error.message})
        }
    }

    me = async (request: any, response: any) => {
        try {
            const user = request.user
            const registrationService = new RegistrationService()
            const registrations = await registrationService.getAllByUser(request.user.id)
            response.send({"user": user, "registrations": registrations})
        } catch (error: any) {
            response.status(400).send({"error": error.message})
        }
    }

```

```
    unregister = async (request: any, response: any) => {
        try {
            const registrationService = new RegistrationService()
            await registrationService.delete(request.params.id, request.user.id)
            response.send("Success")
        } catch (error: any) {
            response.status(400).send({"error": error.message})
        }
    }

    register = async (request: any, response: any) => {
        try {
            const registrationService = new RegistrationService()
            await registrationService.create(request.params.id, request.user.id)
            response.send("Success")
        } catch (error: any) {
            response.status(400).send({"error": error.message})
        }
    }
```

```
    auth = async (request: any, response: any) => {
        const {body} = request
        const {email, password} = body
        try {
            const {user, checkPassword} = await this.userService.checkPassword(email, password)
            if (checkPassword) {
                const payload = {id: user.id}
                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
                const tokenService = new TokenService(user)
                const token = await tokenService.generateToken()
                response.send({accessToken})
            } else {
                throw new Error('Login or password is incorrect!')
            }
        } catch (e: any) {
            response.status(401).send({"error": e.message})
        }
    }

```

```
    refreshToken = async (request: any, response: any) => {
        const {body} = request
        const {token} = body
        const tokenService = new TokenService()
        try {
            const {userId, isExpired} = await tokenService.isTokenExpired(token)
            if (!isExpired && userId) {
                const user = await this.userService.getById(userId)
                const payload = {id: user.id}
                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
                const refreshTokenService = new TokenService(user)
                const refreshToken = await refreshTokenService.generateToken()
                response.send({accessToken, refreshToken})
            } else {
                throw new Error('Invalid credentials')
            }
        } catch (e) {
            response.status(401).send({'error': 'Invalid credentials'})
        }
    }
}

export default UserController
```

Роуты мероприятий

```
import express from "express"
import EventController from "../controllers/event.controller"

const eventRoutes: express.Router = express.Router()
const controller: EventController = new EventController()

eventRoutes.route('/').get(controller.getAll)
eventRoutes.route('/').post(controller.create)
eventRoutes.route('/:id').delete(controller.delete)
eventRoutes.route('/:id').get(controller.get)
eventRoutes.route('/:id').put(controller.update)

export default eventRoutes
```

Роуты юзеров

```ts
import express from "express"
import UserController from "../controllers/user.controller"
import passport from "../middlewares/passport"

const userRoutes: express.Router = express.Router()
const controller: UserController = new UserController()

userRoutes.route('/').post(controller.post)
userRoutes.route('/profile').get(passport.authenticate('jwt', {session: false}), controller.me)
userRoutes.route('/register/:id').post(passport.authenticate('jwt', {session: false}), controller.register)
userRoutes.route('/register/:id').delete(passport.authenticate('jwt', {session: false}), controller.unregister)
userRoutes.route('/login').post(controller.auth)

export default userRoutes
```

```ts
import express from "express"
import userRoutes from "./user.routes"
import eventRoutes from "./event.routes";

const router: express.Router = express.Router()
router.use('/users', userRoutes)
router.use('/events', eventRoutes)

export default router
```

## Вывод

Реализован RESTful API сервиса для поиска профессиональных мероприятий.