САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэкенд-энд разработка

Отчет

Лабораторная работа 2

Выполнил:
Галиновский Роман

Группа: К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

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

Ход работы

route:

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

const authController = new AuthController()
const userController = new UserController()
const eventController = new EventController()
const ticketController = new TicketController()

router.route('/login').post(authController.login)
router.route('/getUsers').get(userController.get)
router.route('/addUser').post(userController.add)

router.route('/getAllEvents').get(eventController.getAll)
router.route('/getEvents').get(eventController.getFiltered)
router.route('/addEvent').post(eventController.add)

router.route('/getTickets').get(passport.authenticate('jwt', { session: false }), ticketController.get)
router.route('/addTicket').post(passport.authenticate('jwt', { session: false }), ticketController.add)

export default router
```

Пример модели Ticket

```
@Table
export default class Ticket extends Model {
  @Min(1)
  @Column
  attendants: number

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

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

  @ForeignKey(() => Event)
  @Column
  eventId: number

  @BelongsTo(() => Event)
  event: Event
}
```

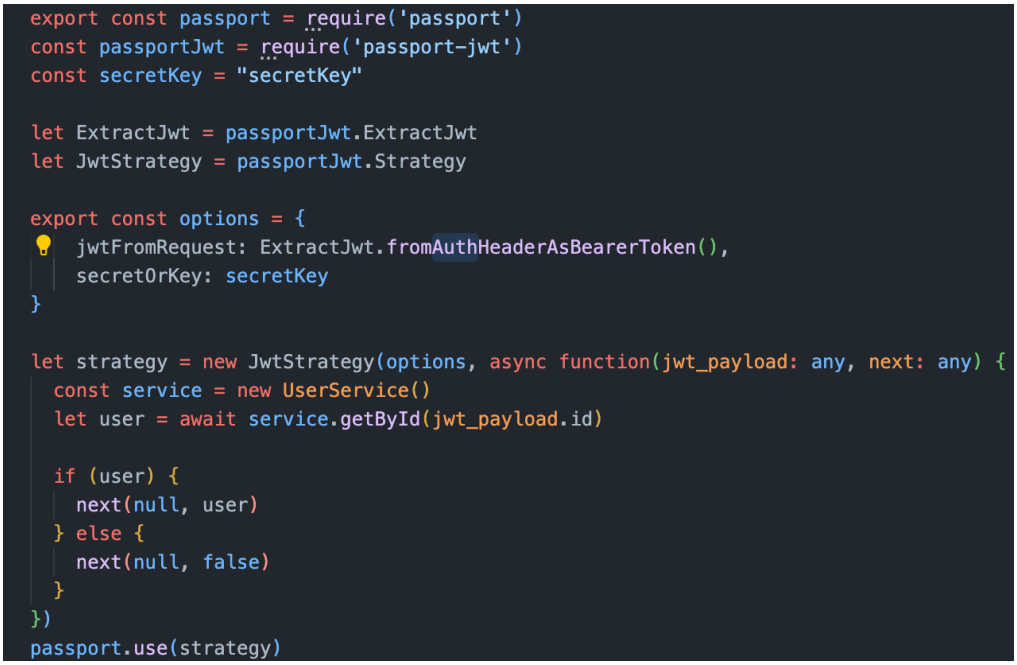
Пример сервиса для работы с пользователями:

```
export default class UserService {

    private repo = sequelize.getRepository(User)

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

    getAll() {
        return this.repo.findAll()
    }

    getByEmail(email_param: string) {
        return this.repo.findOne({ where: { email: email_param }})
    }

    getById(id_param: number) {
        return this.repo.findOne({ where: { id: id_param }})
    }
}
```

Пример контроллера для работы с мероприятиями:

```
export default class EventController {

    private service = new EventService()

    add = async (request: any, response: any) => {
        try {
            const result = await this.service.add(request.body)
            response.send({ id: result.id })
        } catch (error: any) {
            response.status(400).send(error.message)
        }
    }

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

    getFiltered = async (request: any, response: any) => {
        try {
            const data = await this.service.getByFilter(request.query.city, request.query.type)
            response.send(data)
        } catch (error: any) {
            response.status(400).send(error.message)
        }
    }
}
```

Middleware для аутентификации:



```
export const passport = require('passport')
const passportJwt = require('passport-jwt')
const secretKey = "secretKey"

let ExtractJwt = passportJwt.ExtractJwt
let JwtStrategy = passportJwt.Strategy

export const options = {
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
    secretOrKey: secretKey
}

let strategy = new JwtStrategy(options, async function(jwt_payload: any, next: any) {
  const service = new UserService()
  let user = await service.getById(jwt_payload.id)

  if (user) {
    next(null, user)
  } else {
    next(null, false)
  }
})
passport.use(strategy)
```

Вывод

В результате выполнения лабораторной работы был разработан бэкенд сервиса для поиска мероприятий с возможностью регистрации, авторизации и просмотра мероприятий пользователя.