САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэкенд-энд разработка

Отчет

Лабораторная работа 2

Выполнил:
Омар Сизей

Группа: К33412

Проверил:
Добряков Д. И.

Санкт-Петербург

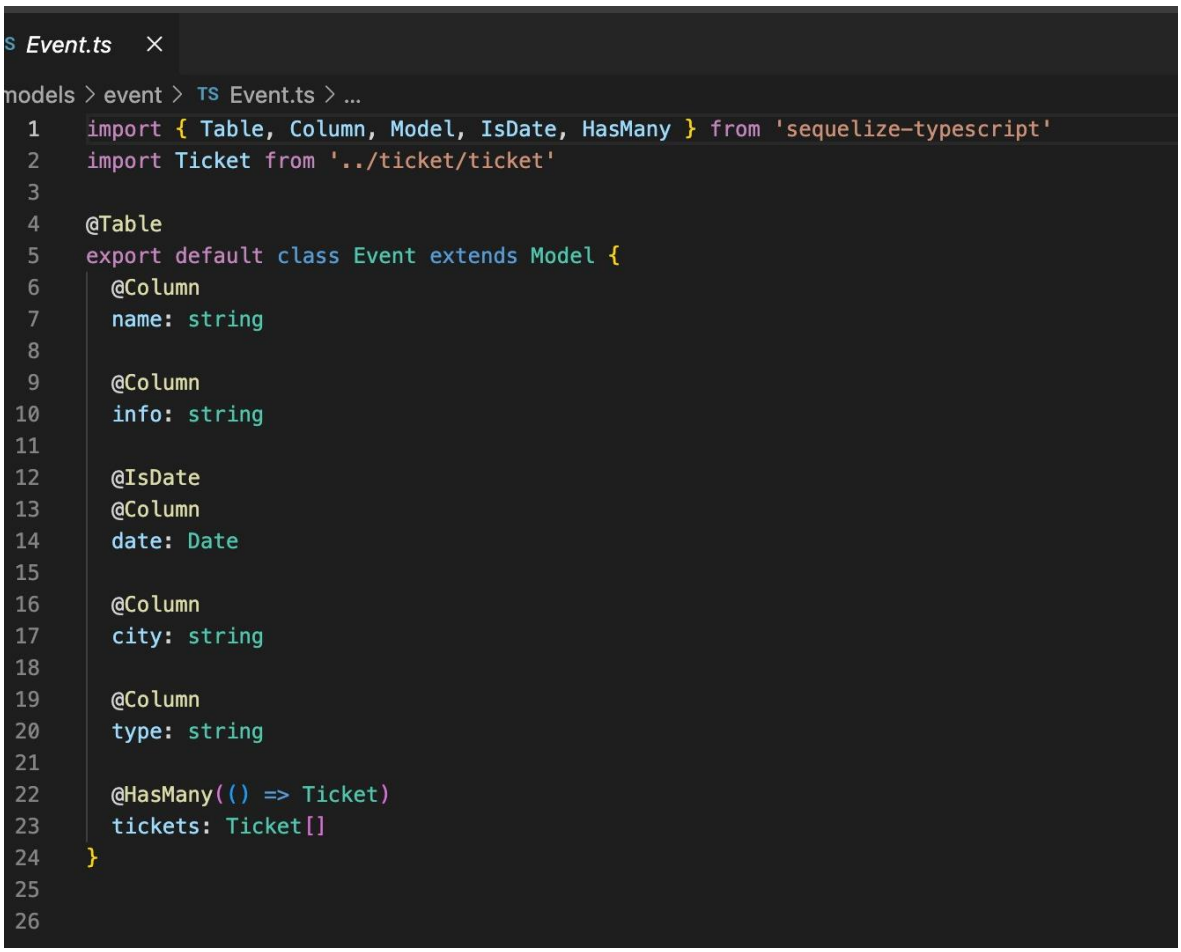
2023 г.

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):



Модели (модель User):

User.ts

models > user > TS User.ts > ...

```ts
import { Table, Column, Model, HasMany } from 'sequelize-typescript'
import Ticket from '../ticket/ticket'

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

  @Column
  email: string

  @Column
  password: string

  @HasMany(() => Ticket)
  tickets: Ticket[]
}
```

Модели (модель ticket):

models > ticket > TS ticket.ts > ...

```ts
import { Table, Column, Model, Min, ForeignKey, BelongsTo } from 'sequelize-typescript'
import User from '../user/User'
import Event from '../event/Event'

@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
}
```

Роуты:

index.ts

routes > index.ts > ...

```ts
import express from "express"
import AuthController from "../controllers/auth/auth"
import UserController from '../controllers/user/user'
import EventController from '../controllers/event/event'
import TicketController from '../controllers/ticket/ticket'

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

```

Services:

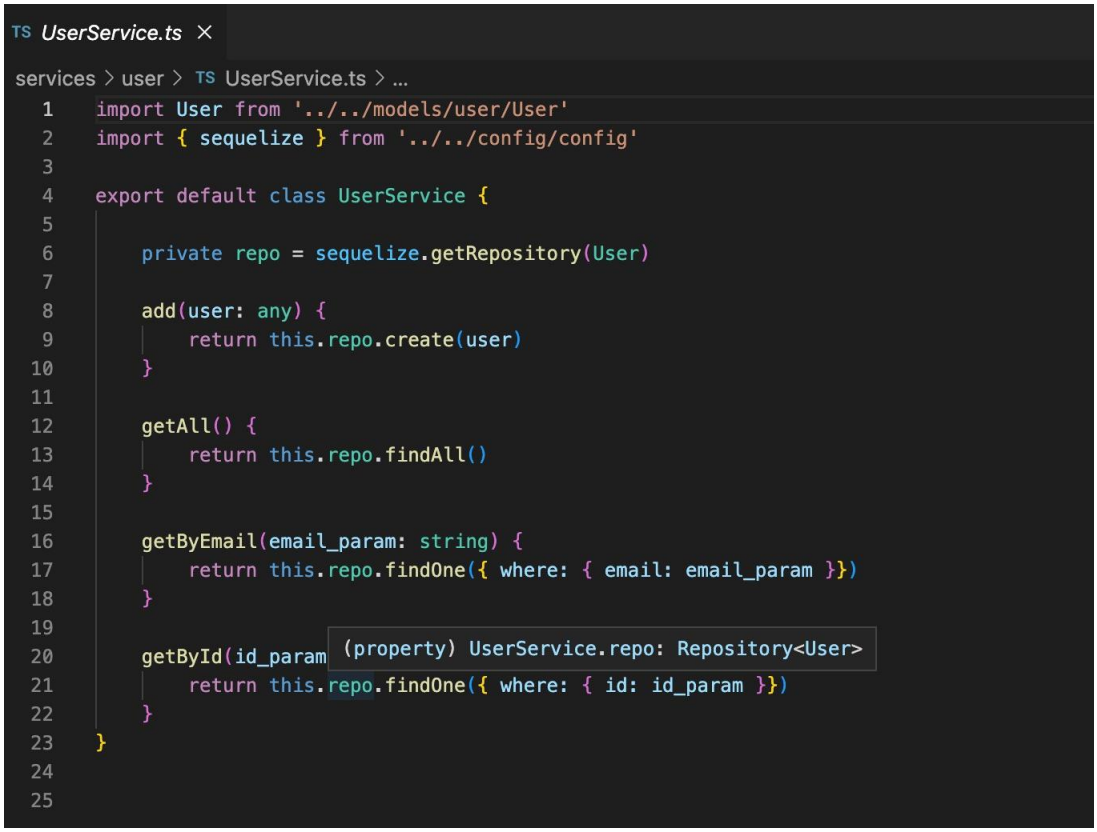
Сервис для работы с событиями:

EventService.ts

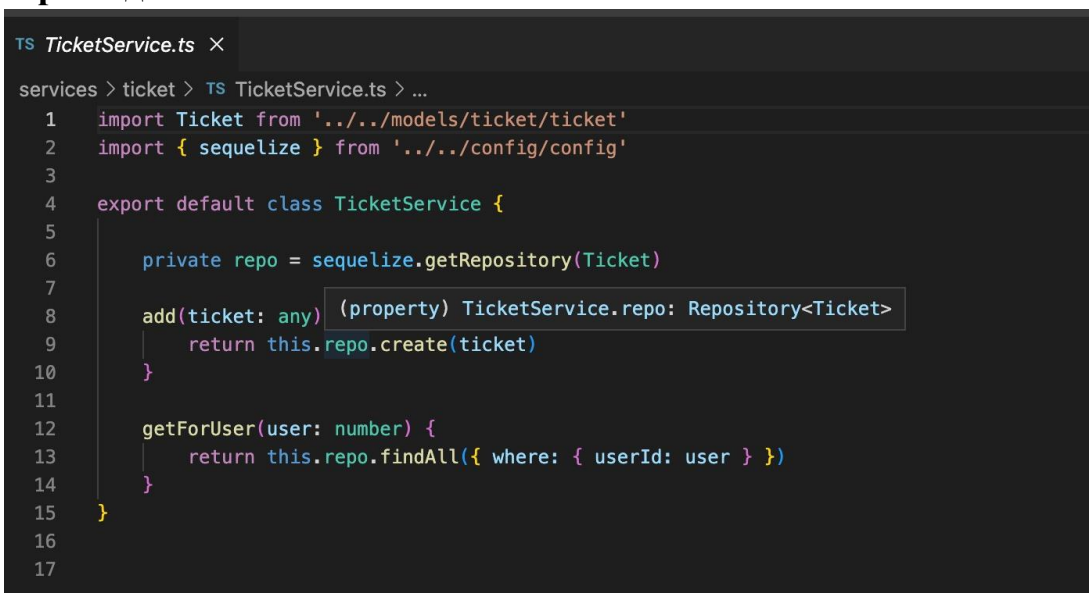
services > event > EventService.ts > ...

```ts
import Event from '../../models/event/Event'
import { sequelize } from '../../config/config'

export default class EventService {

    private repo = sequelize.getRepository(Event)

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

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

    getByFilter(city_param: string, type_param: string) {
        return this.repo.findAll( { where: { city: city_param, type: type_param } } )
    }
}
```

Сервис для Users:



Сервис для Tickets:



Контроллеры:

Контроллер для работы с событиями:

TS event.ts

controllers > event > TS event.ts > ...

```ts
import EventService from '../../services/event/EventService'

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)
        }
    }
}

```

Контроллер для users:

TS user.ts

controllers > user > TS user.ts > ...

```ts
import UserService from '../../services/user/UserService'

export default class UserController {

    private service = new UserService()

    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)
        }
    }

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

```

Контроллер для tickets:

ticket.ts

controllers > ticket > ticket.ts > ...

```ts
import TicketService from '../../services/ticket/TicketService'

export default class TicketController {

    private service = new TicketService()

    add = async (request: any, response: any) => {
        try {
            const ticket = request.body
            ticket.userId = request.user.id
            const result = await this.service.add(ticket)
            response.send({ id: result.id })
        } 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)
        }
    }
}

```

Middleware для авторизации:

index.ts

middleware > index.ts > ...

```ts
import UserService from "../services/user/UserService"

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

let JwtStrategy: any
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 этой лабораторной работе был разработан бэкендсервиса для поиска мероприятий с возможностью регистрации, авторизации и просмотра мероприятий пользователя.