САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2: RESTful API

Выполнила:
Бабан Виктория, К33412

Проверил:
Добряков Д. И.

Санкт-Петербург
2023 г.

Задача

Необходимо реализовать RESTful API средствами express + typescript.

Выбранный вариант: Платформа для поиска профессиональных мероприятий

- 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

Ход работы

Были созданы следующие модели:

```
//User.ts - пользователь

@Table
class User extends Model {
   @PrimaryKey
   @Column
   id: number

   @Column
   lastName: string

   @Column
   firstName: string

   @Unique
   @AllowNull(false)
   @IsEmail
   @Column
   email: string

   @Unique
   @AllowNull(false)
   @Column
   username: string
   @AllowNull(false)
   @Column
   password: string

   @BelongsToMany(() => Event, () => EventEntries)
   events: Event[];
}

export default User
```

```
//Event.ts - мероприятие

@Table
class Event extends Model {
   @PrimaryKey
   @AutoIncrement
   @Column
   id: number

   @AllowNull(false)
   @Column
   name: string

   @AllowNull(false)
   @Column
   category: string

   @AllowNull(false)
   @Column
   place: string

   @IsDate
   @AllowNull(false)
   @Column
   date: Date

   @AllowNull(false)
   @Column
   price: string

   @AllowNull(false)
```

```
    @Column
    district: string

    @Column
    description: string

    @Column
    contact_number: string

    @Column
    contact_name: string

    @BelongsToMany(() => User, () => EventEntries)
    users: User[];
}

export default Event
```

```
// EventEntries - записи на мероприятия

@Table
class EventEntries extends Model {
    @PrimaryKey
    @Column
    entry_id: number

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

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

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

    @BelongsTo(() => Event)
    event: Event
}
export default EventEntries
```

Роуты:

```
// index.ts

import express from "express"
import userRoutes from "./User"
import entryRoutes from "./EventEntries"
import eventRoutes from "./Event"

const router: express.Router = express.Router()

router.use('/users', userRoutes)
router.use('/entries', entryRoutes)
router.use('/events', eventRoutes)

export default router
```

```
// User.ts

import UserController from "../controllers/User";
import express from "express";
import auth from "../middleware/auth"

const userRouter = express.Router()
const userController: UserController = new UserController()

userRouter.route('/login').post(userController.login)
userRouter.route('/register').post(userController.register)
userRouter.route('/me').get(auth.auth, userController.me)
userRouter.route('/reset').post(auth.auth,
userController.updatePassword)

export default userRouter
```

```
// Event.ts

import EventController from "../controllers/Event";
import express from "express";

const eventRouter = express.Router()
const eventController: EventController = new EventController()
```

```
eventRouter.route('/').get(eventController.getEvents)
eventRouter.route('/add').post(eventController.add)
eventRouter.route('/:id').get(eventController.getById)
eventRouter.route('/:id').put(eventController.update)
eventRouter.route('/:id').delete(eventController.delete)

export default eventRouter
```

```
// EventEntries.ts

import EventEntryController from "../controllers/EventEntries";
import express from "express";
import auth from "../middleware/auth";

const entryRouter = express.Router()
const entryController: EventEntryController = new EventEntryController()

entryRouter.route('/').post(auth.auth, entryController.add)
entryRouter.route('/:id').delete(auth.auth, entryController.delete)
entryRouter.route('/').get(auth.auth, entryController.get)

export default entryRouter
```

Контроллеры

```
// User.ts

import User from "../models/User"
import UserService from "../services/User"
import jwt from 'jsonwebtoken'
import UserError from "../errors/User"
import { checkPassword, hashPassword } from "../utils/password"


class UserController {
   private userService: UserService = new UserService()

   login = async (request: any, response: any) => {
       const { email, password } = request.body
       try {
           const user: User = await this.userService.get(email,
password)
           const token = jwt.sign({
               id: user.id,
               username: user.username,
           }, "sUbGuVE~t[)ByQDjcV?LCa_c4};LI-_n")

           response.send({
                   username: user.username,
                   email: user.email,
                   token
               })
       }
       catch (err) {
           response.status(400).send((err as UserError).message)
       }
   }
   register = async (request: any, response: any) => {
       const { body } = request
       try {
           const user = await this.userService.create(body)
           response.status(201).send(user)
       }
       catch (err) {
           response.status(400).send((err as UserError).message)
       }
   }
   me = async (request: any, response: any) => {
       if (request.user === undefined) {
           response.sendStatus(401)
       }
       else {
           try {
               const user = await
this.userService.getById(request.user.id)
               response.send({
                   id: user.id,
                   lastName: user.lastName,
                   firstName: user.firstName,
                   username: user.username,
                   email: user.email
               })
           }
           catch (err) {
               if (err as UserError)
                   response.status(400).send((err as
UserError).message)
               else
                   response.sendStatus(500)
           }
       }
   }
   updatePassword = async (request: any, response: any) => {
       const { oldPassword, newPassword } = request.body
       try {
           console.log(request.body)
           const user: User = await
this.userService.getById(request.user.id)
           if (checkPassword(oldPassword, user.password)) {
               user.password = hashPassword(newPassword)
               await this.userService.update(user)
               response.status(200).send("Password change
successfully")
           }
           else {
               response.sendStatus(400)
           }
       }
       catch (err) {
           response.status(400).send((err as Error).message)
       }
   }
}

export default UserController
```

```
// Event.ts

import EventService from "../services/Event"

class EventController {
    private eventService: EventService = new EventService()
```

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

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

   delete = async (request: any, response: any) => {
       const id = request.params.id
       try {
           await this.eventService.delete(Number(id))
           response.status(200).send({ message: `Event with id
${id} has been deleted` })
       } catch (error: any) {
           response.status(400).send({ "error": error.message })
       }
   }

   getById = async (request: any, response: any) => {
       const id = request.params.id
       try {
           const event = await
this.eventService.getById(Number(id))
           response.send(event)
       } catch (error: any) {
           response.status(404).send({ "error": error.message })
       }
   }


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

export default EventController
```

```
// EventEntries.ts

import EventEntryService from "../services/EventEntries"

class EventEntryController {
   private entryService: EventEntryService = new
EventEntryService()

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

   delete = async (request: any, response: any) => {
       const id = request.params.id
       try {
           await this.entryService.delete(Number(id))
           response.status(200).send({ message: `Events entry with
id ${id} has been deleted` })
       } catch (error: any) {
           response.status(400).send({ "error": error.message })
       }
   }

   get = async (request: any, response: any) => {
       try {
           const userEvents = await
this.entryService.getForUser(request.user.id)
           response.send(userEvents)
       } catch (error: any) {
           response.status(400).send(error.message)
       }
   }
}

export default EventEntryController
```

Сервисы:

```
// User.ts

import User from "../models/User"
import { hashPassword, checkPassword } from "../utils/password";
import UserError from "../errors/User";

class UserService {

    async create(userData: any): Promise<User> {
        userData.password = hashPassword(userData.password)
        const user = await User.create(userData)
        return user.toJSON()
    }

    async getById(id: number): Promise<User> {
        const user: User | null = await User.findByPk(id)
        if (user != null) {
            return user.toJSON()
        }
        throw new UserError("Invalid identifier")
    }
    async get(email: string, password: string): Promise<User> {
        const user: User | null = await User.findOne({
            where: {
                email: email,
            },
        })

        if (user != null && checkPassword(password, user.password))
{
            return user.toJSON()
        }
        throw new UserError("Invalid email/password")
    }

    async update(userData: any): Promise<User> {
        if (userData.id == undefined) {
            throw new UserError("Id is undefined")
        }

        await User.update(userData, { where: {
            id: userData.id
        }})
        let user: User = await this.getById(userData.id)
        return user
    }

    async delete(id: number): Promise<void> {
        const user: User | null = await User.findByPk(id)
        if (user != null) {
            return user.destroy()
        }

        throw new UserError("Invalid identifier")
    }
}

export default UserService
```

```
// Event.ts
```

```
import Event from '../models/Event'
import sequelize from '../providers/db'

const eventRepository = sequelize.getRepository(Event)

class EventService {

   async create(event: any) {
       try{
           const new_event = await eventRepository.create(event)
           return new_event.toJSON()
       }
       catch (e: any) {
           const errors = e.errors.map((error: any) =>
error.message)
           throw console.log(errors)
       }
   }

   async update(id: number, eventData: Partial<Event>):
Promise<Event> {
       try {
           const event = await eventRepository.findByPk(id)
           if (event) {
               await event.update(eventData)
               return event.toJSON()
           }
           throw new Error(`Event with id ${id} not found`)
       }
       catch (e: any) {
           const errors = e.errors.map((error: any) =>
error.message)
           throw console.log(errors)
       }
   }

   async delete(id: number): Promise<void> {
       const event = await eventRepository.findByPk(id)
       if (event) {
           await event.destroy();
           return;
       }
```

```
        throw new Error(`Event with id ${id} not found`);
    }

    async getById(id: number): Promise<Event> {
        const event = await eventRepository.findByPk(id)
        if (event) return event
        throw new Error(`Events with id ${id} not found`)
    }

    async getByFilter(params: any) {
        const events = await eventRepository.findAll({ where:
params })
        if (events) return events
        throw new Error("Events not found!")
    }
}

export default EventService
```

```
// EventEntries.ts

import EventEntries from "../models/EventEntries"
import sequelize from '../providers/db'

const entriesRepository = sequelize.getRepository(EventEntries)

class EventEntryService {

    async create(entry: any) {
        try{
            const new_entry = await entriesRepository.create(entry)
            return new_entry.toJSON()
        }
        catch (e: any) {
            const errors = e.errors.map((error: any) =>
error.message)
            throw console.log(errors)
        }
    }

    async delete(id: number): Promise<void> {
        const entry = await entriesRepository.findByPk(id)
        if (entry) {
            await entry.destroy();
            return;
        }
        throw new Error(`Events entry with id ${id} not found`);
    }

    getForUser(user: number) {
        return entriesRepository.findAll({ where: { user_id: user }
})
    }
}

export default EventEntryService
```

Вывод

В ходе данной лабораторной работы было реализовано RESTful API платформы для поиска мероприятий с использованием инструментов: typescript, Express, ORM Sequelize и др