САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа 2

Выполнил:

Беллазрег Анис

Группа K33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача:

Реализовать RESTful API средствами express + typescript

Вариант:

Платформа для поиска профессиональных мероприятий

Ход работы

1. **Controllers**
   1.1. **attendance.ts**

```typescript
import { v4 as uuidv4 } from "uuid"

import AttendanceService from "../services/attendance"


class AttendanceController {

    private attendanceService: AttendanceService


    constructor() {

        this.attendanceService = new AttendanceService()

    }


    get = async (request: any, response: any) => {

        try {

            const records = await this.attendanceService.listAttendances()

            return response.json(records);

        } catch (error: any) {

            response.status(404).send({ "error": error.message })

        }

    }

    post = async (request: any, response: any) => {

        const id = uuidv4()

        try {

            const record = await this.attendanceService.create({ ...request.body, id})

            return response.json({ record, msg: 'Successfully attend this events' })

        } catch (error: any) {

            response.status(400).send({ "error": error.message })
```

```
        }

    }

}


export default AttendanceController
```

1.2.   auth.ts

```typescript
import jwt from 'jsonwebtoken'

import { jwtOptions } from '../middlewares/passport'

import UserService from '../services/user'

import { v4 as uuidv4 } from "uuid"




class AuthController {

    private userService: UserService

    constructor() {

        this.userService = new UserService()

    }


    register = async (request: any, response: any) => {

        try {

            const user = await this.userService.getByEmail(request.body.email)


            if (user) {

                response.status(400).send({ "error": "User with specified email already
exists" })

            }

            else {

                const id = uuidv4()

                const users = await this.userService.create({ ...request.body, id})

                response.status(201).send(users)

            }
```

```
        } catch (error: any) {

            response.status(400).send({ "error": error.message })

        }

    }


    login = async (request: any, response: any) => {

        const { body } = request

        const { email, password } = body


        try {

            const { user, passwordMatch } = await this.userService.checkPassword(email,
password)


            if (passwordMatch) {

                const payload = { id: user.id }

                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)

                response.send({ accessToken })

            } else {

                throw new Error('Invalid credentials')

            }

        } catch (e: any) {

            response.status(401).send({ "error": e.message })

        }

    }

}


export default AuthController
```

1.3.   event.ts

```
import { v4 as uuidv4 } from "uuid"

import EventService from "../services/event"
```

```typescript
class EventController {

    private eventService: EventService


    constructor() {

        this.eventService = new EventService()

    }


    get = async (request: any, response: any) => {

        try {

            const records = await this.eventService.listEvents()

            return response.json(records);

        } catch (error: any) {

            response.status(404).send({ "error": error.message })

        }

    }

    post = async (request: any, response: any) => {

        const id = uuidv4()

        try {

            const record = await this.eventService.create({ ...request.body, id})

            return response.json({ record, msg: 'Successfully create user' })

        } catch (error: any) {

            response.status(400).send({ "error": error.message })

        }

    }

}

export default EventController
```

1.4.   user.ts

```typescript
import { v4 as uuidv4 } from "uuid"

import UserService from "../services/user"


class UserController {

    private userService: UserService


    constructor() {

        this.userService = new UserService()

    }


    me = async (request: any, response: any) => {

        response.send(request.user)

    }


    post = async (request: any, response: any) => {

        const id = uuidv4()

        try {

            const record = await this.userService.create({ ...request.body, id})

            return response.json({ record, msg: 'Successfully create user' })

        } catch (error: any) {

            response.status(400).send({ "error": error.message })

        }

    }

    put = async (request: any, response: any) => {

        try {

                const record = await this.userService.updateUser(request.params.id, request.body)

            return response.json({record, msg: 'Successfully update user' })

        } catch (error: any) {

            response.status(404).send({ "error": error.message })

        }

    }
```

```
    delete = async (request: any, response: any) => {

        try {

            const record = await this.userService.deleteUser(request.params.id)

            return response.json({msg: 'Successfully delete user' })

        } catch (error: any) {

            response.status(404).send({ "error": error.message })

        }

    }

}

export default UserController
```

2.   Middlewares/passport.ts

```
import passport from 'passport'

import { Strategy as JwtStrategy, ExtractJwt } from 'passport-jwt'

import UserService from '../services/user'




const opts = {

    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),

    secretOrKey: 'secret',

    jsonWebTokenOptions: {

        maxAge: `300000ms`

    }

}



const customJwtStrategy = new JwtStrategy(opts, async function(jwt_payload, next) {

    const userService = new UserService()


    const user = await userService.getById(jwt_payload.id)


    if (user) {
```

```
            next(null, user)

    } else {

            next(null, false)

    }

})


passport.use(customJwtStrategy)


export { opts as jwtOptions }


export default passport
```

3. Models

3.1. attendance.ts

```typescript
import { DataTypes, Model } from "sequelize"

import db from "../configs/config";

import User from "./user";

import Event from "./event";


interface Attributes {

    id: string;

    UserId: string;

    EventId: string;

  }
class Attendance extends Model<Attributes> {}


Attendance.init(

    {

        id: {

            type: DataTypes.UUIDV4,

            allowNull: false,

            primaryKey: true
```

```
        },
        UserId: {
            type: DataTypes.UUIDV4, references: {
                model: 'Users',
                key: 'id'
            },
            allowNull: false
        },
        EventId: {
            type: DataTypes.UUIDV4,
            references: {
                model: 'Events',
                key: 'id'
            },
            allowNull: false
        }
    },
    {
        sequelize:db,
        tableName: "Attendance"
    }
)
User.hasMany(Attendance)
Event.hasMany(Attendance)
export default Attendance
```

3.2.   event.ts

```
import { DataTypes, Model } from "sequelize"
import db from "../configs/config";
interface Attributes {
    id: string;
    name: string;
```

```typescript
    description: string;

    location: string;

    time: Date;

    date: Date;

  }
class Event extends Model<Attributes> {}


Event.init(
    {
        id: {
            type: DataTypes.UUIDV4,

            allowNull: false,

            primaryKey: true
        },

        name: {

            type: DataTypes.STRING,

            allowNull: false

        },

        description: {

            type: DataTypes.STRING,

            allowNull: false

        },

        location: {

            type: DataTypes.STRING,

            allowNull: false,

        },

        time: {

            type: DataTypes.TIME,

            allowNull: false

        },

        date: {
```

```
        type: DataTypes.DATE,

        allowNull: false

      }

    },

    {

        sequelize:db,

        tableName: "Events"

    }

)


export default Event
```

3.3.   user.ts

```ts
import { DataTypes, Model } from "sequelize"

import db from "../configs/config";

interface Attributes {

    id: string;

    firstName: string;

    lastName: string;

    email: string;

    password: string;


  }

class User extends Model<Attributes> {

}


User.init(

    {

        id: {

            type: DataTypes.UUIDV4,

            allowNull: false,

            primaryKey: true
```

```
      },
      firstName: {
        type: DataTypes.STRING,
        allowNull: false
      },
      lastName: {
        type: DataTypes.STRING,
        allowNull: false
      },
      email: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true
      },
      password: {
        type: DataTypes.STRING,
        allowNull: false
      }
  },
  {
      sequelize:db,
      tableName: "Users"
  }
)

export default User
```

4. Routes

4.1. attendance.ts

```
import AttendanceController from '../controllers/attendance'

import express from "express"

import passport from 'passport'
```

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

const attendancecontroller = new AttendanceController()



router.route('/list').get(passport.authenticate('jwt',          {session:          false}),
attendancecontroller.get)

router.route('/create').post(passport.authenticate('jwt',          {session:          false}),
attendancecontroller.post)



export default router
```

4.2.   auth.ts

```ts
import AuthController from "../controllers/auth";

import { Router } from 'express';



const router = Router();

const controller = new AuthController()



router.post('/login',

    controller.login

)



router.post('/register',

    controller.register

)



export default router
```

4.3.   event.ts

```ts
import EventController from '../controllers/event'

import express from "express"



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

```
const eventcontroller = new EventController()


router.route('/list').get(eventcontroller.get)

router.route('/create').post(eventcontroller.post)


export default router
```

4.4. index.ts

```
import { Router } from 'express';


import auth from './auth';

import attendance from './attendance';

import event from './event';

import user from './user';


const router = Router();


router.use('/auth', auth);

router.use('/user', user);

router.use('/event', event);

router.use('/attendance', attendance);


export default router
```

4.5. user.ts

```
import express from "express"

import UserController from '../controllers/user'

import passport from "../middlewares/passport";


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

```
const usercontroller = new UserController()


router.get('/:firstName',
  passport.authenticate('jwt', {session: false}), usercontroller.me)


router.route('/create')
  .post(usercontroller.post)


router.route('/update/:id')
  .put(usercontroller.put)


router.route('/delete/:id')
  .delete(usercontroller.delete)



export default router
```

5. Services
5.1. attendance.ts

```
import AttendanceError from "../errors/attendance/attendance"
import Attendance from "../models/attendance"



class AttendanceService {
    async getById(id: string){
        const attendance = await Attendance.findByPk(id)


        if (attendance) return attendance.toJSON()


        throw new AttendanceError('Not found!')

    }
```

```
async create(attendance: any): Promise<Attendance|Error>{

    try {

        const data = await Attendance.create(attendance)

        return data

    } catch (e: any) {

        const errors = e.errors.map((error: any) => error.message)

        throw new AttendanceError(errors)

    }

}


async listAttendances(){

    const attendances = await Attendance.findAll()


    if (attendances) return attendances


    throw new AttendanceError('Not found!')

}

}


export default AttendanceService
```

5.2. events.ts

```
import EventError from "../errors/events/event"

import Event from "../models/event"



class EventService {

    async getById(id: string){

        const user = await Event.findByPk(id)


        if (user) return user.toJSON()
```

```
        throw new EventError('Not found!')

    }


    async create(event: any): Promise<Event|Error>{

        try {

            const eventData = await Event.create(event)

            return eventData

        } catch (e: any) {

            const errors = e.errors.map((error: any) => error.message)

            throw new EventError(errors)

        }

    }


    async listEvents(){

        const events = await Event.findAll()


        if (events) return events


        throw new EventError('Not found!')

    }

}

export default EventService
```

5.3.   user.ts

```
import UserError from "../errors/users/user"

import User from "../models/user"



class UserService {

    async getById(id: string){

        const user = await User.findByPk(id)
```

```typescript
        if (user) return user.toJSON()


        throw new UserError('Not found!')

    }


    async getByEmail(email: string){

        try {

            const user = await User.findOne({where: {email}})

            return user

        } catch (e: any) {

            const errors = e.errors.map((error: any) => error.message)

            throw new UserError(errors)

        }

    }


    async create(user: any): Promise<User|Error>{

        try {

            const userData = await User.create(user)

            return userData

        } catch (e: any) {

            const errors = e.errors.map((error: any) => error.message)

            throw new UserError(errors)

        }

    }


    async updateUser(id:string, data: any) {

        try {

            const user = await User.findByPk(id)

            if (user) {

                user.update(data)
```

```
        }

        return user

    } catch (e: any) {

        const errors = e.errors.map((error: any) => error.message)


        throw new UserError(errors)

    }

}


async deleteUser(id:string) {

    try {

        await User.destroy({where: {id:id}})

    } catch (e: any) {

        const errors = e.errors.map((error: any) => error.message)


        throw new UserError(errors)

    }

}
async checkPassword(email: string, password: string): Promise<any> {

    try {

        const data = await User.findOne({where: {

            "email": email,

            "password" : password

        }})

        if (data) {

            return { user: data, passwordMatch: true }

        }

    } catch (e: any) {

        const errors = e.errors.map((error: any) => error.message)


        throw new UserError(errors)
```

```
        }
    }
}


export default UserService
```

Вывод

Реализован RESTful API средствами express + typescript для платформы для поиска профессиональных мероприятий.