САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 2: RESTful API средствами express + typescript

Выполнила:

Никифорова Кюннэй

Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Необходимо реализовать RESTful API средствами express + typescript

(используя ранее написанный boilerplate).

Вариант №2. 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

1) Модели:

User.ts – модель пользователя

```
@Table
class User extends Model {
    @Column
    firstName: string

    @Column
    lastName: string

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

    @AllowNull(false)
    @Column
    password: string

    @HasMany( () => UserEvents )
    eventId: UserEvents[]

    @BeforeCreate
    @BeforeUpdate
    static generatePasswordHash(instance: User) {
        const { password } = instance

        if (instance.changed('password')) {
            instance.password = passwordHash(password)
```

```
        }
    }
}

export default User
```

Event.ts – модель мероприятий

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

    @Unique
    @AllowNull(false)
    @Column
    name: string

    @AllowNull(false)
    @Column
    date: Date

    @AllowNull(false)
    @Column
    location: string

    @AllowNull(false)
    @Column
    type: string

    @ForeignKey(() => User)
    @AllowNull(false)
    @Column
    authorId: number
}

export default Event
```

UserEvent.ts – модель записей на мероприятия

```
@Table
class UserEvent extends Model {
    @ForeignKey(() => User)
    @AllowNull(false)
    @Column
    userId: number

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

    @ForeignKey(() => Event)
    @AllowNull(false)
    @Column
    eventId: number
}

export default UserEvent
```

2) Контроллеры:

Event.ts

```
import EventService from "../../services/events/Event"

export class EventController {
    private eventService: EventService;

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

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

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

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

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

User.ts

```
import UserService from "../../services/users/User"
import { jwtOptions } from '../../middlewares/passport'
import jwt from 'jsonwebtoken'

export class UserController {
    private userService: UserService;
```

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

    signup = async (request: any, response: any) => {
        const { body } = request
        try {
            await this.userService.create(body)
            response.status(200).send({ "status" : "OK" })
        } 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, checkPassword } = await
this.userService.checkPassword(email, password)
            if (checkPassword) {
                const payload = { id: user.id }
                console.log('payload is', payload)
                const accessToken = jwt.sign(payload, jwtOptions.secretOrKey)
                response.send({accessToken: accessToken})
            }
        } catch (error: any) {
            response.status(401).send({ "error": error.message })
        }
    }

    me = async (request: any, response: any) => {
        response.send(request.user)
    }

    get = async (request: any, response: any) => {
        const userId = request.params.id
        try {
            const user = await this.userService.getByID(userId)
            response.send(user)
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }
}
```

UserEvent.ts

```
import UserEventService from "../../services/users/UserEvent";

export class UserEventController {
    private UserEventService: UserEventService;

    constructor() {
        this.UserEventService = new UserEventService();
    }

    createEvent = async (request: any, response: any) => {
        const event = request.body
        event.authorId = request.user.id
        try {
```

```
            await this.UserEventService.createEvent(event)
            response.status(200).send({"status": "OK"})
        }
        catch (error: any) {
            response.status(400).send({ "error" : error.message })
        }
    }

    updateEvent = async (request: any, response: any) => {
        const updatedEvent = request.body
        updatedEvent.authorId = request.user.id
        const eventId = request.params.id
        try {
            const event = await this.UserEventService.updateEventByID(eventId,
updatedEvent)
            response.send(event)
        } catch (error: any) {
            response.status(400).send({ "error": error.message })
        }
    }

    deleteEvent = async (request: any, response: any) => {
        const deletedEvent = request.body
        deletedEvent.authorId = request.user.id
        const eventId = request.params.id
        try {
            await this.UserEventService.deleteEventByID(eventId, deletedEvent)
            response.status(204).send({"status": "OK"})
        } catch (error: any) {
            response.status(404).send({ "error": error.message })
        }
    }

    joinTheEvent = async (request: any, response: any) => {
        const userAndEvent = request.body
        userAndEvent.userId = request.user.id
        try {
            const result = await this.UserEventService.joinTheEvent(userAndEvent)
                response.send(result)
        } catch (error: any) {
            response.status(400).send({"error": error.message})
        }
    }

    getEvent = async (request: any, response: any) => {
        try {
            const events = await this.UserEventService.getEvent(
                Number(request.user.id)
            )

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

3) Сервисы:

Event.ts

```
import Event from "../../models/events/Event"
import sequelize from "sequelize"

class EventService {
    async getByID(id: number): Promise<Event|Error> {
        const event = await Event.findByPk(id)

        if (event) {
            return event.toJSON()
        }
        throw new Error(`Event with id ${id} not found`)
    }

    async filter(params: any): Promise<any> {
        return await Event.findAll({where: params})
    }

    async calendarGet() {
        return await Event.findAll({order: sequelize.col('date')})
    }

    async getByAuthorId(id: number){
        const event = await Event.findAll({where: {authorId: id}, order:
sequelize.col('date')})

        if (event) {
            return event
        }
        throw new Error(`Company with name id ${id} not found`)
    }
}

export default EventService
```

User.ts

```
import User from "../../models/users/User"
import checkPassword from '../../utils/checkPassword'


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

    async getByID(id: number): Promise<User|Error> {
        const user = await User.findByPk(id)

        if (user) {
            return user.toJSON()
        }
        throw new Error(`User with id ${id} not found`)
    }

    async checkPassword(email: string, password: string): Promise<any> {
        const user = await User.findOne({ where: { email: email }})
```

```
        if (user) {
            return { user: user.toJSON(), checkPassword: checkPassword(user,
password) }
        }
        throw new Error("Login or password is incorrect!")
    }
}

export default UserService
```

UserEvent.ts

```
import UserEvent from "../../models/users/UserEvent"
import Event from "../../models/events/Event"

class UserEventService {
    async createEvent(eventData: any): Promise<Event> {
        const event = await Event.create(eventData)
        return event.toJSON()
    }

    async updateEventByID(id: number, eventData: any) {
        const event = await Event.findByPk(id)

        if (event) {
            const res = await Event.findAll({ where: { id:id, authorId:
eventData.authorId }})
            if (res.length > 0) {
                await event.update(eventData)
                return event.toJSON()
            }
            throw new Error(`You are not author of event with id ${id}`)
        }
        throw new Error(`Event with id ${id} not found`)
    }

    async deleteEventByID(id: number, eventData: any){
        const event = await Event.findByPk(id)

        if (event) {
            const res = await Event.findAll({ where: { id: id, authorId:
eventData.authorId }})
            if (res.length > 0) {
                return await Event.destroy({ where: { id } })
            }
            throw new Error(`You are not author of event with id ${id}`)
        }
        throw new Error(`Event with id ${id} not found`)
    }

    async joinTheEvent(userAndEvent: any) {
        const foundEvent = await Event.findByPk(userAndEvent.eventId)
        if (foundEvent == null) {
            throw new Error("There is no such Event")
        }
        const res = await UserEvent.findAll({ where: { userId: userAndEvent.userId,
eventId: userAndEvent.eventId }})
        if (res.length > 0) {
            throw new Error("You have already signed up for this event")
        }
```

```
        return await UserEvent.create(userAndEvent)
    }

    async getEvent(userId: number) {
        return UserEvent.findAll({ where: { userId: userId } })
    }
}

export default UserEventService
```

4) Роуты:

index.ts

```
import express from "express"
import eventRoutes from "./events/Event"
import userRoutes from "./users/User"
import usereventRoutes from "./users/UserEvent"

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

routes.use('/events', eventRoutes)
routes.use('/users', userRoutes)
routes.use('/users/profile', usereventRoutes)

export default routes
```

Event.ts

```
import express from "express"
import { EventController } from "../../controllers/events/Event"

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

const controller: EventController = new EventController()

routes.route('/filter')
    .get(controller.filter)

routes.route('/')
    .get(controller.calendar)

routes.route('/author/:id')
    .get(controller.author)

routes.route('/:id')
    .get(controller.get)

export default routes
```

User.ts

```
import express from "express"
import { UserController } from "../../controllers/users/User"
import passport from "../../middlewares/passport"

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

const controller: UserController = new UserController()
```

```
routes.route('/signup')
    .post(controller.signup)

routes.route('/login')
    .post(controller.login)

routes.route('/profile')
    .get(passport.authenticate('jwt', { session: false }), controller.me)

routes.route('/:id')
    .get(controller.get)

export default routes
```

UserEvent.ts

```
import express from "express"
import { UserEventController } from "../../controllers/users/UserEvent"
import passport from "../../middlewares/passport";

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

const controller: UserEventController = new UserEventController()

routes.route('/createEvent')
    .post(passport.authenticate('jwt', { session: false }), controller.createEvent)

routes.route('/updateEvent/:id')
    .patch(passport.authenticate('jwt', { session: false }), controller.updateEvent)

routes.route('/deleteEvent/:id')
    .delete(passport.authenticate('jwt', { session: false }), controller.deleteEvent)

routes.route('/joinTheEvent')
    .post(passport.authenticate('jwt', { session: false }), controller.joinTheEvent)

routes.route('/getEvent')
    .get(passport.authenticate('jwt', { session: false }), controller.getEvent)

export default routes
```

Вывод

В ходе данной лабораторной работы были получены практические навыки работы с RESTful API средствами express и typescript.