САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2: Реализация RESTful API

Выполнил:

Попов Ньургун

К33412

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

Нужно 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. Структура проекта:

2. Вход и регистрация были реализованы еще в первой лабораторной работе.

3. Поиск мероприятия (фильтрации по типу мероприятия, месту проведения):

Часть кода из файла src>models>events>Event.ts:

```
@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(() => Author)
    @AllowNull(false)
    @Column
    authorId: number
}
```

Часть кода из файла src>controllers>Event.ts:

```
    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 })
        }
    }
```

Часть кода из файла src>services>Event.ts:

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

4. Календарь ближайших мероприятий:

Часть кода из файла src>controllers>Event.ts:

```
    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 })
        }
    }
```

Часть кода из файла src>services>Event.ts:

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

5. Промо страница для организаторов мероприятия:

Часть кода из файла src>models>authors>Author.ts:

```
@Table
class Author extends Model {
    @Unique
    @AllowNull(false)
    @Column
    companyName: string

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

    @AllowNull(false)
    @Column
    password: string

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

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

Часть кода из файла src>controllers>Event.ts:

```
    author = async  (request: any, response: any) => {
        const eventId = request.params.id
        try {
            const event = await this.eventService.getByAuthor(eventId)
            response.send(event)
```

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

Часть кода из файла src>services>Event.ts:

```
    async getByAuthor(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`)
    }
```

6. Личный кабинет пользователя со списком мероприятий, на которые он записывался:

Часть кода из файла src>models>users>User.ts:

```
@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)
        }
    }
}
```

Часть кода из файла src>models>users>UserEvent.ts:

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

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

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

Часть кода из файла src>controllers>UserEvent.ts:

```
    addEvent = async (request: any, response: any) => {
        const userAndEvent = request.body
        userAndEvent.userId = request.user.id
        try {
            const result = await this.UserEventService.addEvent(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})
        }
    }
```

Часть кода из файла src>services>UserEvent.ts:

```
    async addEvent(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 } })
    }
```

Вывод

В результате выполненной работы: бал реализован RESTful API средствами express + typescript на основе ранее написанного boilerplate, был создан структурированный проект, который включает в себя: controller, core, middleware, model, provider, route, service и utils, также был добавлен makefile. Было создано 3 дополнительные таблицы для выполнения заданий из варианта №2.