САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнила:

Мищенко Виолетта

Группа K33402

Проверил:
Добряков Д.И.

Санкт-Петербург

2022 г.

Цель работы: необходимо реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Имеющиеся контроллеры:

Bookings.ts

```ts
class BookingController {
    private bookingService: BookingService

    constructor() {
        this.bookingService = new BookingService()
    }

    list = async (request: any, response: any) => {
        if (!request.user) return response.status(401).send({error: 'Not authorized'})

        try {
            const bookings = await this.bookingService.list(request.user.id)
            return response.send(bookings)
        } catch (error: any) {
            response.status(404).send({ 'error': error.message })
        }
    }
```

```ts
    list = async (request: any, response: any) => {
        if (!request.user) return response.status(401).send({error: 'Not authorized'})

        try {
            const bookings = await this.bookingService.list(request.user.id)
            return response.send(bookings)
        } catch (error: any) {
            response.status(404).send({ 'error': error.message })
        }
    }

    create = async (request: any, response: any) => {
        const { body } = request

        if (!request.user) return response.status(401).send({error: 'Not authorized'})

        body.userId = request.user.id

        try {
            const booking: BookingModel | BookingError = await this.bookingService.create(body)

            response.status(201).send(booking)
        } catch (error: any) {
            response.status(400).send({ 'error': error.message })
        }
    }
}


export default BookingController
```

Hotels.ts

```
class HotelController {
    private hotelService: HotelService

    constructor() {
        this.hotelService = new HotelService()
    }

    list = async (request: any, response: any) => {
        const { location } = request.query

        try {
            const hotels = await this.hotelService.list(location)

            return response.send(hotels)
        } catch (error: any) {
            response.status(404).send({ 'error': error.message })
        }
    }

    create = async (request: any, response: any) => {
        const { body } = request

        try {
            const hotel: HotelModel | HotelError = await this.hotelService.create(body)

            response.status(201).send(hotel)
        } catch (error: any) {
            response.status(400).send({ 'error': error.message })
        }
    }
```

```
    item = async (request: any, response: any) => {
        try {
            const hotel = await this.hotelService.item(request.params.id)
            return response.send(hotel)
        } catch (error: any) {
            response.status(404).send({ 'error': error.message })
        }
    }

}

export default HotelController
```

Rooms.ts

```
class RoomController {
    private roomService: RoomService

    constructor() {
        this.roomService = new RoomService()
    }

    list = async (request: any, response: any) => {
        try {
            const rooms = await this.roomService.list(request.params.id)
            return response.send(rooms)
        } catch (error: any) {
            response.status(404).send({ 'error': error.message })
        }
    }
```

```
    create = async (request: any, response: any) => {
        const { body } = request

        try {
            const room: RoomModel | RoomError = await this.roomService.create(body)

            response.status(201).send(room)
        } catch (error: any) {
            response.status(400).send({ 'error': error.message })
        }
    }
}

export default RoomController
```

Модели:

BookingModel

```
@Table
class BookingModel extends Model {
    @AllowNull(false)
    @Column
    dateArrival: Date

    @Column
    dateDeparture: Date

    @AllowNull(false)
    @Column
    guestsNumber: number

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

    @AllowNull(false)
    @ForeignKey(() => RoomModel)
    @Column
    roomId: number
}
```

HotelModel

```
@Table
class HotelModel extends Model {
    @AllowNull(false)
    @Column
    name: string

    @AllowNull(false)
    @Column
    rating: number

    @AllowNull(false)
    @Column
    location: string

    @Column
    averagePrice: number
}
```

RoomModel

```
@Table
class RoomModel extends Model {
    @AllowNull(false)
    @Column
    bedsCount: number

    @AllowNull(false)
    @Column
    price: number

    @Column
    description: string

    @Column
    mealType: string

    @AllowNull(false)
    @ForeignKey(() => HotelModel)
    @Column
    hotelId: number
}
```

Сервисы:

BookingService

```
class BookingService {
    async list(userId: number) {
        try {
            return await BookingModel.findAll({ where: { userId } })
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new BookingError(errors)
        }
    }

    async create(data: object) {
        try {
            const Booking = await BookingModel.create(data)
            return Booking.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new BookingError(errors)
        }
    }
}

export default BookingService
```

HotelService

```
class HotelService {
    async list(location?: string) {
        if (location) {
            return await HotelModel.findAll({ where: { location } })
        } else {
            return await HotelModel.findAll()
        }
    }

    async create(data: object) {
        try {
            const hotel = await HotelModel.create(data)
            return hotel.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new HotelError(errors)
        }
    }
```

```
    async item(hotelId: number) {
        try {
            return await HotelModel.findByPk(hotelId)
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new HotelError(errors)
        }
    }
}

export default HotelService
```

Вывод: в данной лабораторной работе был выбран вариант реализации приложения по поиску отелей/жилья и был реализован RESTful API средствами express + typescript и использованный раннее boilerplate.