САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2: RESTful API

Выполнил:

Балдина Дарья

Группа

К33401

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

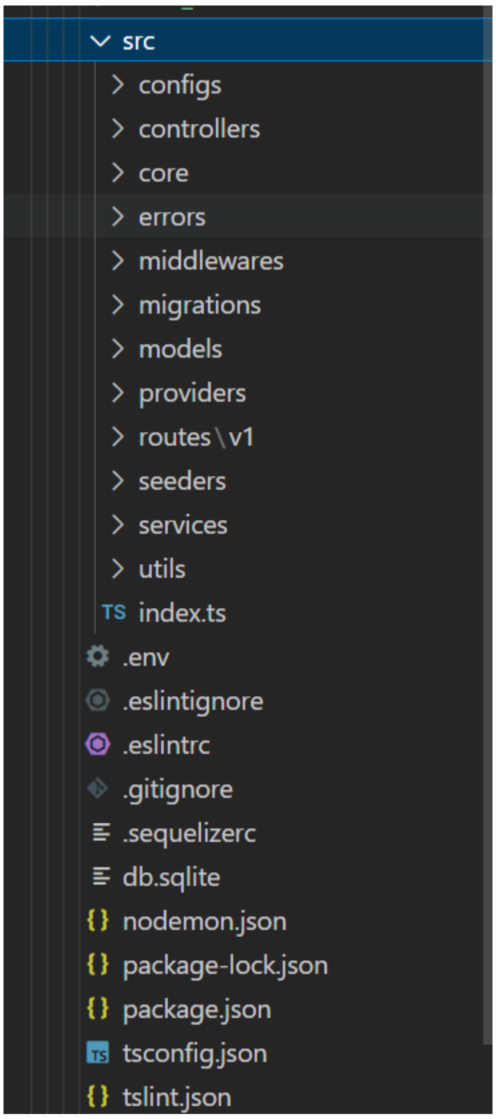
Задача

Необходимо реализовать RESTful API средствами express + typescript.

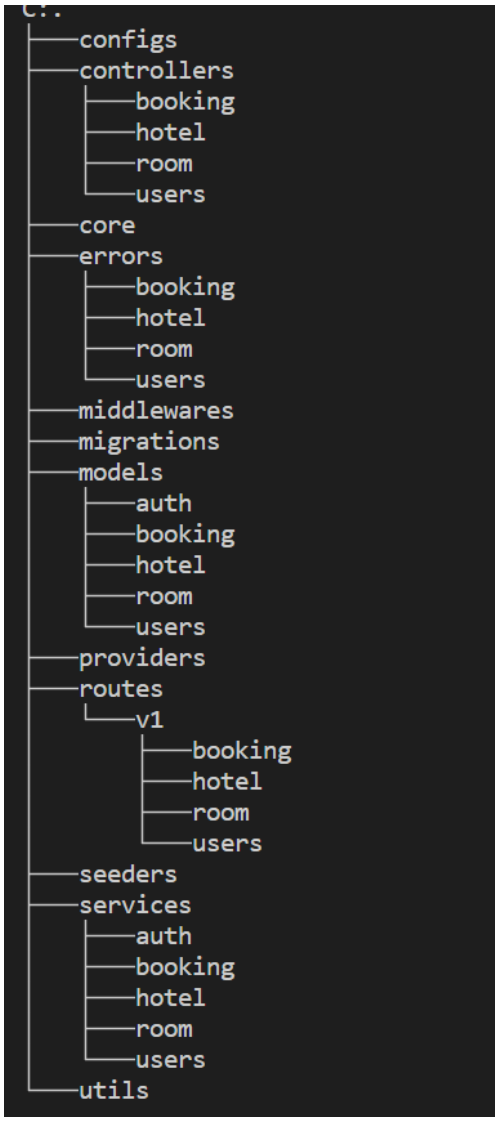
Платформа для поиска и бронирования номера в отеле
- Вход
- Регистрация
- Поиск информации об отеле
- Бронирование комнаты
- Поиск информация о комнатах

Ход работы

1. Структура приложения



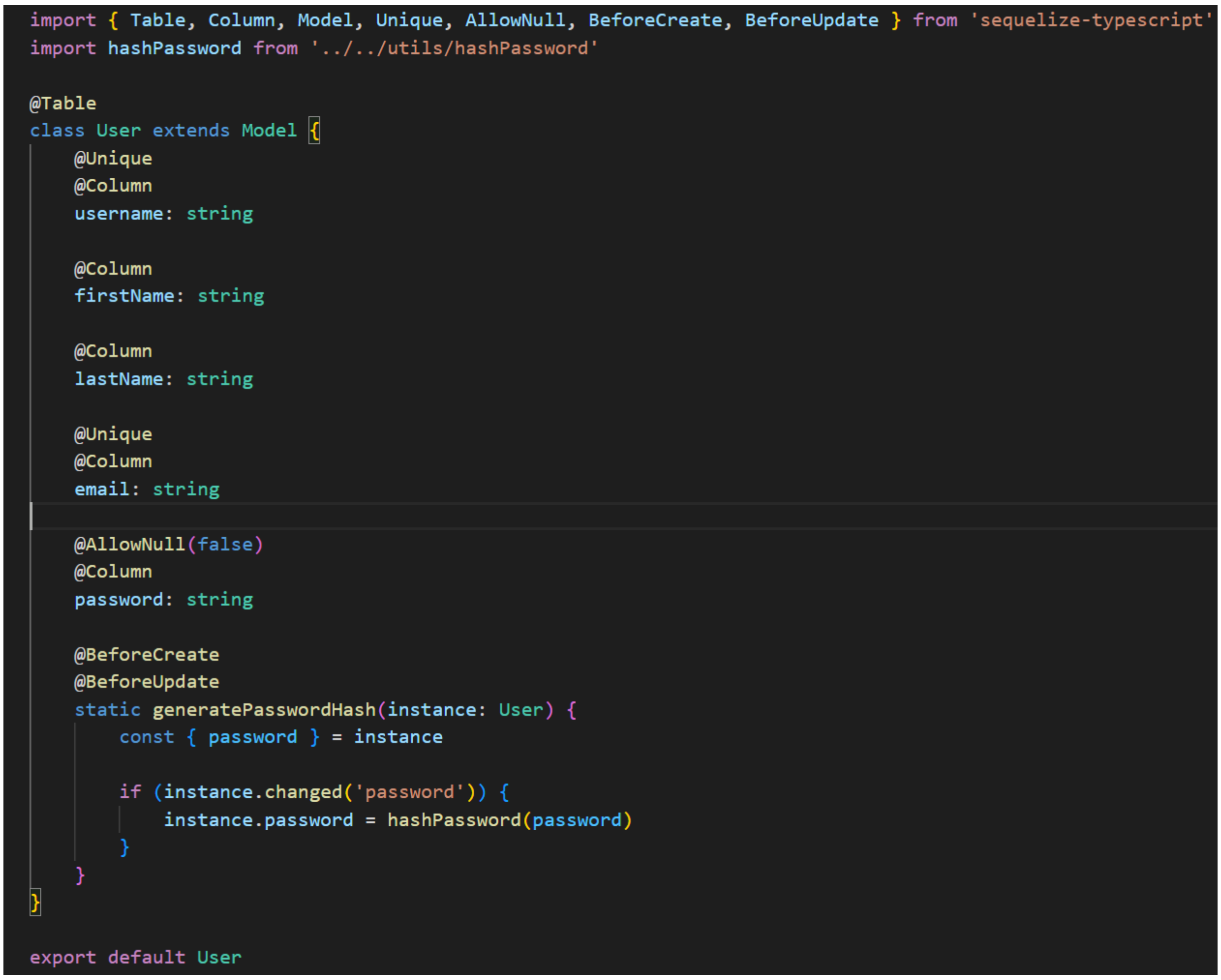
```
src
  configs
  controllers
  core
  errors
  middlewares
  migrations
  models
  providers
  routes\v1
  seeders
  services
  utils
  index.ts
.env
.eslintignore
.eslintrc
.gitignore
.sequelizerc
db.sqlite
nodemon.json
package-lock.json
package.json
tsconfig.json
tslint.json
```



```
C:.
├───configs
├───controllers
│   ├───booking
│   ├───hotel
│   ├───room
│   └───users
├───core
├───errors
│   ├───booking
│   ├───hotel
│   ├───room
│   └───users
├───middlewares
├───migrations
├───models
│   ├───auth
│   ├───booking
│   ├───hotel
│   ├───room
│   └───users
├───providers
├───routes
│   └───v1
│       ├───booking
│       ├───hotel
│       ├───room
│       └───users
├───seeders
├───services
│   ├───auth
│   ├───booking
│   ├───hotel
│   ├───room
│   └───users
└───utils
```

2. Модели

User:



```
import { Table, Column, Model, Unique, AllowNull, BeforeCreate, BeforeUpdate } from 'sequelize-typescript'
import hashPassword from '../../utils/hashPassword'

@Table
class User extends Model {
    @Unique
    @Column
    username: string

    @Column
    firstName: string

    @Column
    lastName: string

    @Unique
    @Column
    email: string

    @AllowNull(false)
    @Column
    password: string

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

        if (instance.changed('password')) {
            instance.password = hashPassword(password)
        }
    }
}

export default User
```

Hotel:

```
import { Table, Column, Model, HasMany, Unique, AllowNull, PrimaryKey,
AutoIncrement} from 'sequelize-typescript'
import Room from '../room/Room'

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

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

    @AllowNull(false)
    @Column
    stars: number

    @HasMany(() => Room)
    rooms: Room[]
}

export default Hotel
```

Room:

```
import { AllowNull, BelongsTo, Column, ForeignKey, HasMany, Model, Table,
PrimaryKey} from 'sequelize-typescript'
import Hotel from '../hotel/Hotel'
import Booking from '../booking/Booking'


@Table
class Room extends Model {
    @AllowNull(false)
    @PrimaryKey
    @Column
    roomNumber: string

    @AllowNull(false)
    @Column
    floor: number

    @AllowNull(false)
    @Column
    capacity: number

    @AllowNull(false)
    @Column
    priceOfNight: number

    @Column
    description: string

    @ForeignKey(() => Hotel)
    @Column
    hotelId: number

    @BelongsTo(() => Hotel)
    hotel: Hotel
    @HasMany(() => Booking)
    bookings: Booking[]
}

export default Room
```

Booking:

```
import { AllowNull, BelongsTo, Column, ForeignKey, Model, Table, PrimaryKey,
AutoIncrement} from 'sequelize-typescript'
import Room from '../room/Room'
import User from '../users/User'

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

    @AllowNull(false)
    @Column
    arrivalDate: Date

    @AllowNull(false)
    @Column
    departureDate: Date

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

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

    @ForeignKey(() => Room)
    @Column
    roomId: number

    @BelongsTo(() => Room)
    room: Room

    @AllowNull(false)
    @Column
    price: number

    get duration(): number {
        return (this.departureDate.getTime() - this.arrivalDate.getTime()) /
(1000 * 60 * 60 * 24);
      }
    async save(): Promise<this> {
        // Check that start < end
        if (this.arrivalDate >= this.departureDate) {
          throw new Error("Duration date must be greater than start date");
        }
        this.price = this.room.priceOfNight * this.duration;

    // Save changes to the database
    return super.save();
}
}

export default Booking
```

3. Контроллеры

User

```
import User from '../../models/users/User'
import UserService from '../../services/users/User'
import UserError from '../../errors/users/User'
import jwt from 'jsonwebtoken'
import { jwtOptions } from '../../middlewares/passport'
import RefreshTokenService from '../../services/auth/RefreshToken'

class UserController {
    private userService: UserService

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

    get = async (request: any, response: any) => {
        try {
            const user: User | UserError = await this.userService.getById(
                Number(request.params.id)
            )

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

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

        try {
            const user : User|UserError = await this.userService.create(body)

            response.status(201).send(user)
        } catch (error: any) {
            response.status(400).send({ "error": error.message })
        }
    }

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

    auth = 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)

                const refreshTokenService = new RefreshTokenService(user)

                const refreshToken = await
refreshTokenService.generateRefreshToken()

                response.send({ accessToken, refreshToken })
            } else {
                throw new Error('Login or password is incorrect!')
            }
        } catch (e: any) {
            response.status(401).send({ "error": e.message })
        }
    }

    getAll = async (request: any, response: any) => {
        try {
            const users = await this.userService.getAll()

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

    getByUsername = async (request: any, response: any) => {
        try {
            const user = await this.userService.getByUsername(
                request.params.username
            )

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

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

        const { refreshToken } = body

        const refreshTokenService = new RefreshTokenService()

        try {
            const { userId, isExpired } = await refreshTokenService
                .isRefreshTokenExpired(refreshToken)

            if (!isExpired && userId) {
                const user = await this.userService.getById(userId)

                const payload = { id: user.id }

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

                const refreshTokenService = new RefreshTokenService(user)

                const refreshToken = await
refreshTokenService.generateRefreshToken()

                response.send({ accessToken, refreshToken })
            } else {
                throw new Error('Invalid credentials')
            }
        } catch (e) {
            response.status(401).send({ 'error': 'Invalid credentials' })
        }
    }

    update = async (request: any, response: any) => {
        try {
          const { body } = request;
          const { id } = request.params
          const user = await this.userService.updateUser(Number(id), body);

          response.send(user);
        } catch (error:any) {
          response.status(400).send(error.message);
        }
      }
    delete = async (request: any, response: any) => {
        try {
            await this.userService.deleteUser(request.params.id)
            response.send('deleted')
        }
        catch (error: any) {
            response.sendStatus(400)
        }
}
}

export default UserController
```

Hotel

```
import HotelService from '../../services/hotel/Hotel'
import { Request, Response } from 'express'

class HotelController {
    private hotelService: HotelService

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

    list = async (request: Request, response: Response) => {
        const hotels = await this.hotelService.list()
        response.status(200).send(hotels)
    }

    get = async (request: Request, response: Response) => {
        try {
            const hotel = await
this.hotelService.getById(Number(request.params.id))
            response.send(hotel)
        } catch (error: any) {
            response.status(404).send({ error: error.message })
        }
    }

    post = async (request: Request, response: Response) => {
        const { body } = request

        try {
            const hotel = await this.hotelService.create(body)

            response.status(201).send(hotel)
        } catch (error: any) {
            response.status(400).send({ error: error.message })
        }
    }
    getByName = async (request: Request, response: Response) => {
        try {
            const hotel = await this.hotelService.getByName(request.params.name)

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


    update = async (request: any, response: any) => {
            try {
                const { body } = request
                const { id } = request.params
                const hotel = await this.hotelService.updateHotel(Number(id),
body)

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

    delete = async (request: any, response: any) => {
            try {
                await this.hotelService.deleteHotel(request.params.id)
                response.sendStatus(200)
            }
            catch (error: any) {
                response.sendStatus(400)
            }
    }
}

export default HotelController
```

Room

```
import RoomService from '../../services/room/Room'
import { Request, Response } from 'express'

class RoomController {
    private roomService: RoomService

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

    list = async (request: Request, response: Response) => {
        const rooms = await
this.roomService.listOfHotelsRoom(Number(request.params.id))
        response.status(200).send(rooms)
    }

    post = async (request: Request, response: Response) => {
        const { body } = request
        const hotelId = Number(request.params.id)

        const roomData = { ...body, hotelId: hotelId }

        try {
            const room = await this.roomService.create(roomData)

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

    getByRoomNumber = async (request: Request, response: Response) => {
        try {
            const room = await this.roomService.getByRoomNumber(
                Number(request.params.roomNumber)
            )

            response.status(200).send(room)
        } catch (error: any) {
            response.status(400).send({ error: error.message })
        }
    }

    update = async (request: any, response: any) => {
        try {
            const { body } = request
            const { id } = request.params
            const room = await this.roomService.updateRoom(Number(id), body)

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

    delete = async (request: any, response: any) => {
        try {
            await this.roomService.deleteRoom(request.params.id)
            response.sendStatus(200)
        }
        catch (error: any) {
            response.sendStatus(400)
        }
}
}

export default RoomController
```

Booking

```
import BookingService from '../../services/booking/Booking'
import { Request, Response } from 'express'

class BookingController {
    private bookingService: BookingService

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

    getById = async (request: Request, response: Response) => {
        try {
            const hotel = await
this.bookingService.getById(Number(request.params.id))
            response.send(hotel)
        } catch (error: any) {
            response.status(404).send({ error: error.message })
        }
    }

    post = async (request: Request, response: Response) => {
        const { body } = request

        const userId = (request.user as any)?.id

        if (!userId) return response.status(401).send({ error: 'Only for auth'
})

        try {
            const booking = await this.bookingService.create({ ...body, userId
})
            response.status(201).send(booking)
        } catch (error: any) {
            response.status(400).send({ error: error.message })
        }
    }

    update = async (request: any, response: any) => {
        try {
            const { body } = request
            const { id } = request.params
            const booking =  await this.bookingService.updateBooking(Number(id),
body)
```

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

    delete = async (request: any, response: any) => {
        try {
            await this.bookingService.deleteBooking(request.params.id)
            response.sendStatus(200)
        }
        catch (error: any) {
            response.sendStatus(400)
        }
}

}

export default BookingControl
```

4. Сервисы

User

```
import User from '../../models/users/User'
import UserService from '../../services/users/User'
import UserError from '../../errors/users/User'
import jwt from 'jsonwebtoken'
import { jwtOptions } from '../../middlewares/passport'
import RefreshTokenService from '../../services/auth/RefreshToken'

class UserController {
    private userService: UserService

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

    get = async (request: any, response: any) => {
        try {
            const user: User | UserError = await this.userService.getById(
                Number(request.params.id)
            )

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

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

        try {
            const user : User|UserError = await this.userService.create(body)

            response.status(201).send(user)
        } catch (error: any) {
            response.status(400).send({ "error": error.message })
        }
    }

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

    auth = 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)

                const refreshTokenService = new RefreshTokenService(user)

                const refreshToken = await
refreshTokenService.generateRefreshToken()

                response.send({ accessToken, refreshToken })
            } else {
                throw new Error('Login or password is incorrect!')
            }
        } catch (e: any) {
            response.status(401).send({ "error": e.message })
        }
    }

    getAll = async (request: any, response: any) => {
        try {
            const users = await this.userService.getAll()

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

  getByUsername = async (request: any, response: any) => {
    try {
      const user = await this.userService.getByUsername(
        request.params.username
      )

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

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

    const { refreshToken } = body

    const refreshTokenService = new RefreshTokenService()

    try {
      const { userId, isExpired } = await refreshTokenService
        .isRefreshTokenExpired(refreshToken)

      if (!isExpired && userId) {
        const user = await this.userService.getById(userId)

        const payload = { id: user.id }

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

        const refreshTokenService = new RefreshTokenService(user)

        const refreshToken = await
refreshTokenService.generateRefreshToken()

        response.send({ accessToken, refreshToken })
      } else {
        throw new Error('Invalid credentials')
      }
    } catch (e) {
      response.status(401).send({ 'error': 'Invalid credentials' })
    }
  }

  update = async (request: any, response: any) => {
    try {
     const { body } = request;
     const { id } = request.params
     const user = await this.userService.updateUser(Number(id), body);
```

```
          response.send(user);
        } catch (error:any) {
          response.status(400).send(error.message);
        }
      }

    delete = async (request: any, response: any) => {
        try {
            await this.userService.deleteUser(request.params.id)
            response.send('deleted')
        }
        catch (error: any) {
            response.sendStatus(400)
        }
}
}

export default UserController
```

Hotel

```
import Hotel from "../../models/hotel/Hotel"
import HotelError from "../../errors/hotel/Hotel"
import { Optional } from "sequelize"

class HotelService {
    async list(): Promise<Hotel[]> {
        return await Hotel.findAll()
    }

    async create(hotelData: Optional<string, any>): Promise<Hotel | HotelError>
{
        try {
            const hotel = await Hotel.create(hotelData)

            return hotel.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)

            throw new HotelError(errors)
        }
    }

    async getById(id: number): Promise<Hotel> {
        const hotel = await Hotel.findByPk(id)

        if (hotel) return hotel.toJSON()

        throw new HotelError('Not found!')
    }
    async getByName(name: string) {
        return await Hotel.findOne({ where: { name: name } })
    }


    async updateHotel(id: number, newHotelData: any){
        const hotel = await Hotel.findByPk(id)

        if (!hotel) {
            throw new HotelError('Hotel not found');
          }

          Object.assign(hotel, newHotelData)
          return await hotel.save()
    }

    async deleteHotel(id: number) {
        const hotel: Hotel = await this.getById(id)
        hotel.destroy()
    }
}

export default HotelService
```

Room

```
import { Optional } from 'sequelize'
import Room from '../../models/room/Room'
import RoomError from '../../errors/room/Room'

class RoomService {
    async listOfHotelsRoom(id: number): Promise<Room[]> {
        return await Room.findAll({ where: { hotelId: id } })
    }

    async create(hotelData: Optional<string, any>): Promise<Room | RoomError> {
        try {
            const room = await Room.create(hotelData)

            return room.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)

            throw new RoomError(errors)
        }
    }

    async getByRoomNumber(roomNumber: number): Promise<Room> {
        const room = await Room.findOne({where: {roomNumber: roomNumber}})
        if (room) return room.toJSON()

        throw new RoomError('Not found!')
    }

    async updateRoom(roomNumber: number, newRoomgData: any){
        const room = await this.getByRoomNumber(roomNumber)

        if (!room) {
            throw new RoomError('Hotel not found');
          }

          Object.assign(room, newRoomgData)
          return await room.save()
    }

    async deleteRoom(roomNumber: number) {
        const room: Room = await this.getByRoomNumber(roomNumber)
        room.destroy()
    }
}

export default RoomService
```

Booking

```
import Booking from '../../models/booking/Booking'
import BookingError from '../../errors/booking/Booking'
import { Optional } from 'sequelize'
import Room from '../../models/room/Room'
import RoomError from '../../errors/room/Room'

class BookingService {
    async create(bookingData: Optional<string, any>): Promise<Room | RoomError>
{
        const roomId = bookingData['roomId']
        const { arrivalDate, departureDate } = bookingData
        if (!roomId) throw new BookingError('roomId must be specified')
        if (!arrivalDate || !departureDate) throw new BookingError('dateFrom and
dateTo must be specified')

        const arrivalDateTime = new Date(arrivalDate).getTime()
        const departureDateTime = new Date(departureDate).getTime()

        const room = await Room.findByPk(roomId)

        if (!room) throw new BookingError('Such room not found!')

        const roomBookings = await Booking.findAll({ where: { roomId: room.id }
})

        let possible = true
        roomBookings.forEach((booking) => {
            if (!possible) return

            const bookingArrivalDateTime = booking.arrivalDate.getTime()
            const bookingDepartureDateTime = booking.departureDate.getTime()

            const isIntersect = Boolean(
                (departureDateTime > bookingDepartureDateTime &&
departureDateTime < bookingArrivalDateTime) ||
                    (arrivalDateTime > bookingDepartureDateTime &&
arrivalDateTime < bookingArrivalDateTime) ||
                    (departureDateTime === bookingDepartureDateTime &&
bookingArrivalDateTime===bookingArrivalDateTime)
            )

            if (isIntersect) {
                possible = false
            }
        })

        if (!possible) throw new BookingError('Cant book this room for this
period! Room is busy')

        try {
            const booking = await Booking.create(bookingData)

            return booking.toJSON()
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)

            throw new BookingError(errors)
        }
    }

    async getById(id: number): Promise<Booking> {
        const booking = await Booking.findByPk(id)

        if (booking) return booking.toJSON()

        throw new BookingError('Not found!')
    }

    async updateBooking(id: number, newBookingData: any){
        const booking = await this.getById(id)

        if (!booking) {
            throw new BookingError('Hotel not found');
          }

          Object.assign(booking, newBookingData)
          return await booking.save()
    }


    async deleteBooking(id: number) {
        const booking: Booking = await this.getById(id)
        booking.destroy()
    }
}

export default BookingService
```

5. Роуты

User

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

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

const controller: UserController = new UserController()

router.route('/reg')
    .post(controller.post)

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

router.route('/account/:id')
    .get(controller.get)

router.route('/login')
    .post(controller.auth)

router.route('/refresh')
    .post(controller.refreshToken)

router.route('/accounts')
    .get(controller.getAll)

router.route('/accounts/:username')
    .get(controller.getByUsername)

router.route('/update/:id').put(passport.authenticate('jwt', { session: false
}), controller.update)
router.route('/delete/:id').delete(passport.authenticate('jwt', { session: false
}), controller.delete)

export default router
```

Hotel

```
import express from 'express'
import HotelController from '../../../controllers/hotel/Hotel'
import passport from "../../../middlewares/passport"
```

```
const hotelRoutes = express.Router()
const controller: HotelController = new HotelController()

hotelRoutes.route('/').get(passport.authenticate('jwt', { session: false }),
controller.list)
hotelRoutes.route('/').post(passport.authenticate('jwt', { session: false }),
controller.post)
hotelRoutes.route('/name/:name').get(passport.authenticate('jwt', { session:
false }), controller.getByName)
hotelRoutes.route('/update/:id').put(passport.authenticate('jwt', { session:
false }), controller.update)
hotelRoutes.route('/delete/:id').delete(passport.authenticate('jwt', { session:
false }), controller.delete)


export default hotelRoutes
```

Room

```
import express from 'express'
import RoomController from '../../../controllers/room/Room'
import passport from "../../../middlewares/passport"

const roomRoutes = express.Router()
const controller: RoomController = new RoomController()

roomRoutes.route('/').get(passport.authenticate('jwt', { session: false }),
controller.list)
roomRoutes.route('/').post(passport.authenticate('jwt', { session: false }),
controller.post)
roomRoutes.route('/number/:roomNumber').get(passport.authenticate('jwt', {
session: false }), controller.getByRoomNumber)
roomRoutes.route('/update/:id').put(passport.authenticate('jwt', { session:
false }), controller.update)
roomRoutes.route('/delete/:id').delete(passport.authenticate('jwt', { session:
false }), controller.delete)


export default roomRoutes
```

Booking

```
import express from 'express'
import BookingController from '../../../controllers/booking/Booking'
import passport from "../../../middlewares/passport"

const bookingRoutes = express.Router()
const controller: BookingController = new BookingController()
```

```
bookingRoutes.route('/').get(passport.authenticate('jwt', { session: false }),
controller.getById)
bookingRoutes.route('/').post(passport.authenticate('jwt', { session: false }),
controller.post)
bookingRoutes.route('/update/:id').put(passport.authenticate('jwt', { session:
false }), controller.update)
bookingRoutes.route('/delete/:id').delete(passport.authenticate('jwt', {
session: false }), controller.delete)


export default bookingRoutes
```

Вывод

В ходе данной лабораторной работы было реализовано RESTful API приложение сайта бронирования комнаты в отеле с использованием инструментов: typescript, Express, ORM Sequelize и др.