САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Чу минь тиеп
Группа К33401

Проверил:
Добряков Д. И.

Санкт-Петербург
2021 г.

Задача

ЛР2

В рамках данной лабораторной работы Вам предложено выбрать один из нескольких вариантов. Выбранный вариант останется единым на весь курс и будет использоваться в последующих лабораторных работах.

По выбранному варианту необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Ход работы

Model/Booking/booking.ts

```
import { DataTypes, Model, Optional } from 'sequelize';
import { idText } from 'typescript';
import db from '../../config/database.config';
import { Hotel } from '../Hotel/Hotel';
import { Client } from '../User/Users';

interface BookingAttributes {
    id: number;
    User_id: number;
    Hotel_id: number;
    Check_in: Date;
    Check_out: Date;
    Price: number;
}

export class Booking extends Model<BookingAttributes> {
    public id!: number
    public User_id!: number;
    public Hotel_id!: number;
    public Check_in!: Date;
    public Check_out!: Date;
    public Price!: number;

    public readonly createdAt!: Date;
    public readonly updatedAt!: Date;
    public readonly deletedAt!: Date;
}

Booking.init(
    {
        id: {
            type: DataTypes.NUMBER,
```

```
            primaryKey: true,
            allowNull: false,
        },
        User_id: {
            type: DataTypes.NUMBER,
            references: {
                model: 'client',
                key: 'id',
            },
            allowNull: false,
        },
        Hotel_id: {
            type: DataTypes.NUMBER,
            references: {
                model: 'hotel',
                key: 'id',
            },
            allowNull: false,
        },
        Check_in: {
            type: DataTypes.DATE,
            allowNull: false,
        },
        Check_out:  {
            type: DataTypes.DATE,
            allowNull: false,
        },
        Price:  {
            type: DataTypes.NUMBER,
            allowNull: false,
        },

    },
    {
        timestamps: true,
        sequelize: db,
        tableName: 'booking',
    }
);
Client.hasMany(Booking)
Hotel.hasMany(Booking)
export interface BookingInput extends Optional<BookingAttributes, 'id' &
'User_id' & 'Hotel_id' & 'Check_in' & 'Check_out' & 'Price'> {}
export interface BookingOutput extends Required<BookingAttributes> {}
```

model/Hotel/Hotel.ts

```
import { DataTypes, Model, Optional } from 'sequelize';
```

```ts
import db from '../../config/database.config';

interface HotelAttributes {
    id: number;
    Name: string;
    Adress: string;
    Assess: number;

}

export class Hotel extends Model<HotelAttributes> {
    public id!: number
    public Name!: string
    public Adress!: string
    public Assess!: number

    public readonly createdAt!: Date;
    public readonly updatedAt!: Date;
    public readonly deletedAt!: Date;
}

Hotel.init(
    {
        id: {
            type: DataTypes.NUMBER,
            primaryKey: true,
            allowNull: false,
        },
        Name: {
            type: DataTypes.STRING,
            allowNull: false,
        },
        Adress: {
            type: DataTypes.STRING,
            allowNull: false,
            unique: true,
        },
        Assess: {
            type: DataTypes.STRING,
            allowNull: false,
        },
    },
    {
        timestamps: true,
        sequelize: db,
        tableName: 'hotel',
    }
);
```

```
export interface HotelInput extends Optional<HotelAttributes, 'id' & 'Name' &
'Adress' & 'Assess'> {}
export interface HotelOuput extends Required<HotelAttributes> {}
```

model/User/Users.ts

```
import { DataTypes, Model, Optional } from 'sequelize';
import db from '../../config/database.config';

interface ClientAttributes {
    id: number;
    firstname: string;
    lastname: string;
    email: string;
    password: string;
}

export class Client extends Model<ClientAttributes> {
    public id!: number
    public firstname!: string
    public lastname!: string
    public email!: string
    public password!: string

    public readonly createdAt!: Date;
    public readonly updatedAt!: Date;
    public readonly deletedAt!: Date;
}

Client.init(
    {
        id: {
            type: DataTypes.NUMBER,
            primaryKey: true,
            allowNull: false,
        },
        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,
        }
    },
    {
        timestamps: true,
        sequelize: db,
        tableName: 'client',
    }
);
export interface ClientInput extends Optional<ClientAttributes, 'id' &
'firstname' & 'lastname' & 'email' & 'password'> {}
export interface ClientOuput extends Required<ClientAttributes> {}
```

routes/auth/auth.ts

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

routes/bookings/booking.ts

```
import BookController from "../../controllers/bookings/booking";
import { Router } from 'express';
import passport from "../../middleware/passport"

const router = Router();
const controller = new BookController()

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

```
)

router.get(
    '/mybooking',
    passport.authenticate('jwt', {session: false}), controller.getall
)

export default router
```

routes/hotels/hotel.ts

```
import HotelController from "../../controllers/hotels/hotel";
import { Router } from 'express';

const router = Router();
const controller = new HotelController()

router.post(
    '/create',
    controller.create
)

router.get(
    '/hotel/:Name',
    controller.getbyname
)

router.get(
    '/listhotel',
   controller.getall
)
router.put(
    '/update/:id',
    controller.update
)

router.delete(
    '/delete/:id',
    controller.delete
)

export default router
```

routes/users/user.ts

```
import { Router } from 'express';
import passport from "../../middleware/passport";
import MeController from '../../controllers/users/index'

const router = Router();
const meController = new MeController()

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

export default router
```

middleware/passport.ts

```
import passport from 'passport'
import { ExtractJwt, Strategy as JwtStrategy } from 'passport-jwt'
import UserService from '../services/User/index'

let secretKey = process.env.JWT_SECRET
secretKey ??= 'secret_key'

const opts = {
    jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
    secretOrKey: secretKey,
    jsonWebTokenOptions: {
        maxAge: process.env.JWT_EXPIRATION
    }
}

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
```

controllers/auth/Auth.ts

```
import jwt from 'jsonwebtoken'
import { jwtOptions } from '../../middleware/passport'
import UserService from '../../services/User/index'

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 users = await this.userService.create(request.body)
                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
```

controllers/bookings/bookings.ts

```
import { request } from "http"
import BookServices from "../../services/Booking/index"
import { Booking } from '../../model/Booking/booking';
import UserError from '../../errors/users/index';
class BookController {
    private BookService: BookServices

    constructor() {
        this.BookService = new BookServices()
    }

    create = async(request: any, response: any) => {
        const { body } = request
        try {
          const user : Booking|UserError = await this.BookService.create(body)
          response.status(201).send(user)
        } catch (error: any) {
          console.log(error)
          response.status(404).send({ "error": error.message})
        }
      }
    getall = async(request: any, response: any) => {
        const data = await this.BookService.getall(request.user.id)
        response.status(201).send(data)
    }
}
export default BookController
```

controllers/hotels/hotel.ts

```
import express, {Request, Response } from 'express';
import db from "../../config/database.config";
import { Hotel } from '../../model/Hotel/Hotel';
import UserError from '../../errors/users/index';
import HotelService from '../../services/Hotels/hotel';
import { ValidationErrorItem } from 'sequelize/types';
db.sync().then(() => {
  console.log('connect');
});

class HotelController {
  private hotelService: HotelService

  constructor() {
      this.hotelService = new HotelService()
  }
  create = async(request: any, response: any) => {
    const { body } = request
    try {
      const user : Hotel|UserError = await this.hotelService.create(body)
      response.status(201).send(user)
    } catch (error: any) {
      console.log(error)
      response.status(404).send({ "error": error.message})
    }
  }
  getbyname = async(request: any, response: any) => {
    console.log(request)
    const user = await
this.hotelService.getbyname(String(request.params.Name))
    if (!user) {
      response.status(400).send('Not found')
    }
    else
      response.status(201).send(user)
  }
  getall = async(request: any, response: any) => {
    const user = await this.hotelService.findAll()
    response.status(201).send(user)
  }
  update = async(request: any, response: any) => {
    const id = Number(request.params.id)
    const { body} = request
    try {
      const user = await this.hotelService.update(id, body)
      if (user)
        response.status(201).send(user)
```

```
      else
        response.status(400).send('Not found')
    } catch (error: any) {
      console.log(error)
      response.status(404).send({ "error": error.message})
    }
  }
  delete = async(request: any, response: any) => {
    const user = await this.hotelService.delete(
      Number(request.params.id)
    )
    if (!user) {
      response.status(400).send('Not found')
    }
    else
      response.status(201).send("Was delete")
  }
}
export default HotelController
```

controllers/users/index.ts

```
import express, {Request, Response } from 'express';
import db from "../../config/database.config";
import { Client } from '../../model/User/Users';
import UserError from '../../errors/users/index';
import UserService from '../../services/User/index';
import { ValidationErrorItem } from 'sequelize/types';
db.sync().then(() => {
  console.log('connect');
});

class Controller {
  private userService: UserService

  constructor() {
      this.userService = new UserService()
  }
  me = async (request: any, response: any) => {
    response.send(request.user)
  }
  export default Controller
```

services/Booking/index.ts

```
import { Hotel } from "../../model/Hotel/Hotel";
import { Client } from "../../model/User/Users";
import { Booking, BookingInput, BookingOutput } from
'../../model/Booking/booking';
```

```
import UserError from '../../errors/users';

class BookServices {
    async create(userData: BookingInput) : Promise<Booking|UserError> {
        try {
            const data = await Booking.create(userData)
            return(data)
        } catch(e: any) {
            throw new Error(e)
        }
    }
    async getall(User_id: BookingInput) : Promise<any> {
        try{
            const test = await Booking.findAll({ where: {User_id}})
            return(test)
        } catch(e: any) {
            throw new Error(e)
        }
    }
}
export default BookServices
```

services/Hotels/hotel.ts

```
import express, {Request, Response } from 'express';
import db from "../../config/database.config";
import { HotelInput, HotelOuput, Hotel } from '../../model/Hotel/Hotel';
import UserError from '../../errors/users';
class HotelService {
    async create(userData: HotelInput) : Promise<Hotel|UserError> {
        try {
            const user = await Hotel.create(userData)
            return(user)
        } catch(e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new UserError(errors)
        }
    }
    async getbyname(Name: string) : Promise<Hotel|null> {
        const data = await Hotel.findOne({ where: {Name}})
        return data
    }
    async findAll() : Promise<HotelOuput[]> {
        return Hotel.findAll()
    }
    async update(id: number, userData: Partial<HotelInput>) :
Promise<Hotel|UserError|null> {
        try {
            const data = await Hotel.findByPk(id)
```

```
            if (data) {
                const result = await (data as Hotel).update(userData)
                return result
            }
            else
                return data
        }catch(e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new UserError(errors)
        }
    }
    async delete(id: number) : Promise<boolean> {
        const deldata = await Hotel.destroy({
            where: {id}
        })
        return !!deldata
    }
}
export default HotelService
```

services/User/index.ts

```
import express, {Request, Response } from 'express';
import db from "../../config/database.config";
import { ClientInput, ClientOuput, Client } from '../../model/User/Users';
import UserError from '../../errors/users';
class UserService {
    async create(userData: ClientInput) : Promise<Client|UserError> {
        try {
            const user = await Client.create(userData)
            return(user)
        } catch(e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new UserError(errors)
        }
    }
    async getById(id: number) : Promise<ClientOuput|null> {
        const data = await Client.findByPk(id)
        return data
    }
    async getByEmail(email: string) : Promise<ClientOuput|null> {
        try {
            const data = await Client.findOne({where: { email }})
            return data
        } catch(e: any) {
            const errors = e.errors.map((error: any) => error.message)
```

```
            throw new UserError(errors)
        }

    }
    async findAll() : Promise<ClientOuput[]> {
        return Client.findAll()
    }
    async update(id: number, userData: Partial<ClientInput>) :
Promise<Client|UserError|null> {
        try {
            const data = await Client.findByPk(id)
            if (data) {
                const result = await (data as Client).update(userData)
                return result
            }
            else
                return data
        }catch(e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw new UserError(errors)
        }
    }
    async delete(id: number) : Promise<boolean> {
        const deldata = await Client.destroy({
            where: {id}
        })
        return !!deldata
    }
    async checkPassword(email: string, password: string) : Promise<any>  {
        const data = await Client.findOne({where: { email, password }})
        if (!data) {
            throw new Error('Does not exist')
        }
        return { user: data, passwordMatch: true }
    }
}
export default UserService
```

Вывод : Create model by sequelize, write router, controllers, services