САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2: RESTful API

Выполнила:

Кулагина Светлана

Группа К33412

Проверил:
Добряков Д. И.

Санкт-Петербург
2023 г.

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

Задачи:

- Вход
- Регистрация
- Личный кабинет пользователя
- Поиск с возможностью фильтрации

Ход работы:

1. Модель User и Модель RefreshToken

```
@Table
class User extends Model {
    @AllowNull(false)
    @Column
    firstName: string

    @AllowNull(false)
    @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
```

Модель User

```ts
import { Table, Column, Model, Unique, AllowNull, ForeignKey } from 'sequelize-typescript'
import User from '../users/User'

@Table
class RefreshToken extends Model {
    @Unique
    @AllowNull(false)
    @Column
    token: string

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

export default RefreshToken
```

Модель RefreshToken

2. Я решила выбрать свой вариант- сервис по статистике соревнований собак среди разных компаний. На данном этапе я реализовала модели Компании и Собак (про выигранные призы и соревнования можно будет реализовать в последствие работы):

```ts
import { AllowNull, Column, Model, Table, Unique } from 'sequelize-typescript';

@Table
class Company extends Model {
    @AllowNull(false)
    @Unique
    @Column
    company_name: string;

    @AllowNull(false)
    @Column
    year: string;

    @AllowNull(false)
    @Column
    owner_name: string;

    @AllowNull(false)
    @Column
    owner_surname: string;
}

export default Company;
```

Рисунок 1 – Модель компании

```ts
import { AllowNull, Column, Model, Table, Unique, ForeignKey } from 'sequelize-typescript';
import Company from '../dogs_companies/Dogs_company';

@Table
class Dogs extends Model {
  @AllowNull(false)
  @Column
  name: string;

  @AllowNull(false)
  @Column
  breed: string;

  @ForeignKey(() => Company)
  @Column
  companyId: number;
}

export default Dogs;
```

Рисунок 2 – модель собаки

3. Далее мной были созданы services для компаний и собак

```ts
import Company from '../../models/dogs_companies/Dogs_company';
import sequelize from '../../providers/db';

const companiesRepository = sequelize.getRepository(Company);

class CompaniesService {
    async get(id: number): Promise<Company> {
        const company = await companiesRepository.findOne({ where: { 'id': id } });
        if (company) return company
        throw new Error(`Company with id ${id} not found`);
    }

    async create(companyData: Partial<Company>): Promise<Company> {
        try {
            const company = await companiesRepository.create(companyData);
            return company.toJSON();
        } catch (e: any) {
            const errors = e.errors.map((error: any) => error.message);
            throw console.log(errors);
        }
    }
```

```
async update(id: number, companyData: Partial<Company>): Promise<Company> {
    try {
        const company = await companiesRepository.findOne({ where: { id } });

        if (company) {
            await company.update(companyData);
            return company.toJSON();
        }
        throw new Error(`Worker with id ${id} not found`);
    } catch (e: any) {
        const errors = e.errors.map((error: any) => error.message);
        throw console.log(errors);
    }
}

async delete(id: number): Promise<void> {
    const company = await companiesRepository.findOne({ where: { id } });
    if (company) {
        await company.destroy();
        return;
    }
    throw new Error(`Company with id ${id} not found`);
}

async getByOwner(id: string): Promise<any> {
    const company = await companiesRepository.findAll({ where: { 'owner_surname': id } });
    if (company) return company
    throw new Error(`${id}'s companies not found`);
}
```

Рисунки 3,4 – Services для компании

```
import Dogs from '../../models/dogs/Dogs'
import sequelize from '../../providers/db'

const dogsRepository = sequelize.getRepository(Dogs)

class DogsService {
    async getById(id: number): Promise<Dogs> {
        const dogs = await dogsRepository.findOne({ where: { 'id': id } })
        if (dogs) return dogs
        throw new Error(`Dogs ${id} not found`)
    }

    async create(dogsData: Partial<Dogs>): Promise<Dogs> {
        try {
            const dogs = await dogsRepository.create(dogsData)
            return dogs.toJSON()
        }
        catch (e: any) {
            const errors = e.errors.map((error: any) => error.message)
            throw console.log(errors)
        }
    }
```

```
async update(id: number, dogsData: Partial<Dogs>): Promise<Dogs> {
    try {
        const dogs = await dogsRepository.findOne({ where: { 'id': id } })

        if (dogs) {
            await dogs.update(dogsData)
            return dogs.toJSON()
        }
        throw new Error(`Dogs ${id} not found`)
    }
    catch (e: any) {
        const errors = e.errors.map((error: any) => error.message)
        throw console.log(errors)
    }
}

async delete(id: number): Promise<void> {
    const dogs = await dogsRepository.findOne({ where: { 'id': id } })
    if (dogs) {
        await dogs.destroy()
        return
    }
    throw new Error(`Dogs ${id} not found`)
}

async getByBreed(id: string): Promise<any> {
    const dogs = await dogsRepository.findAll({ where: { 'breed': id } })
    if (dogs) return dogs
    throw new Error(`Dogs ${id} not found`)
}
```

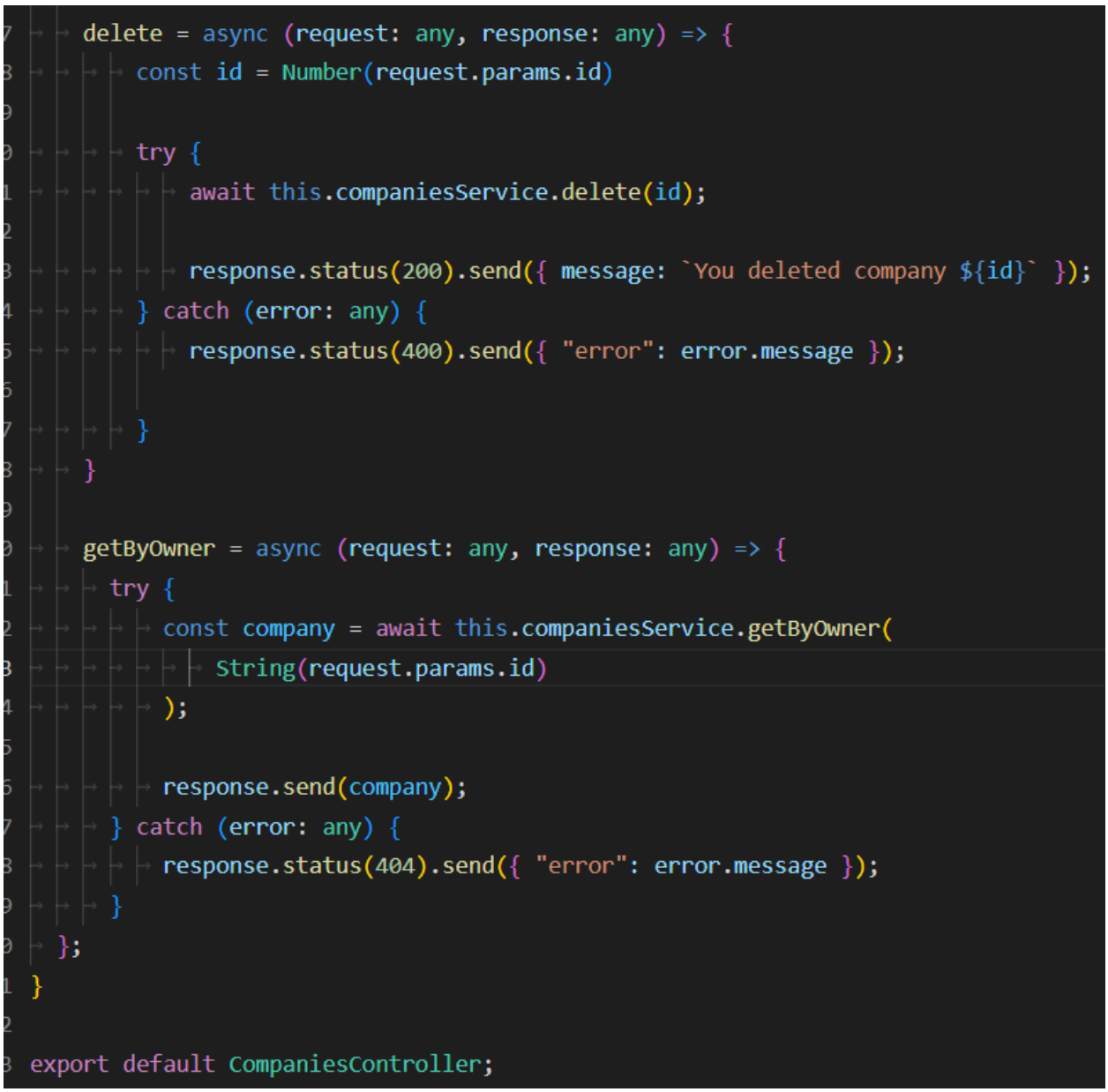
```
    async getByCompany(id: string): Promise<any> {
        const dogs = await dogsRepository.findAll({ where: { 'companyId': id } })
        if (dogs) return dogs
        throw new Error(`Dogs ${id} not found`)
    }
}

export default DogsService
```

Рисунок 5,6,7 – Services для собак

4. Следующим этапом я создала контроллеры:

```ts
import CompaniesService from '../../services/dogs_companies/Dogs_company';

class CompaniesController {
    private companiesService: CompaniesService;

    constructor() {
        this.companiesService = new CompaniesService();
    }

    get = async (request: any, response: any) => {
        try {
            const company = await this.companiesService.get(
                Number(request.params.id)
            );

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

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

    try {
        const company = await this.companiesService.create(body);

        response.status(200).send(company);
    } catch (error: any) {
        response.status(400).send({ "error": error.message });
    }
};

update = async (request: any, response: any) => {
    const { body } = request;
    const id = Number(request.params.id);

    try {
        const worker = await this.companiesService.update(id, body);

        response.send(worker);
    } catch (error: any) {
        response.status(400).send({ "error": "error" });
    }
};
```

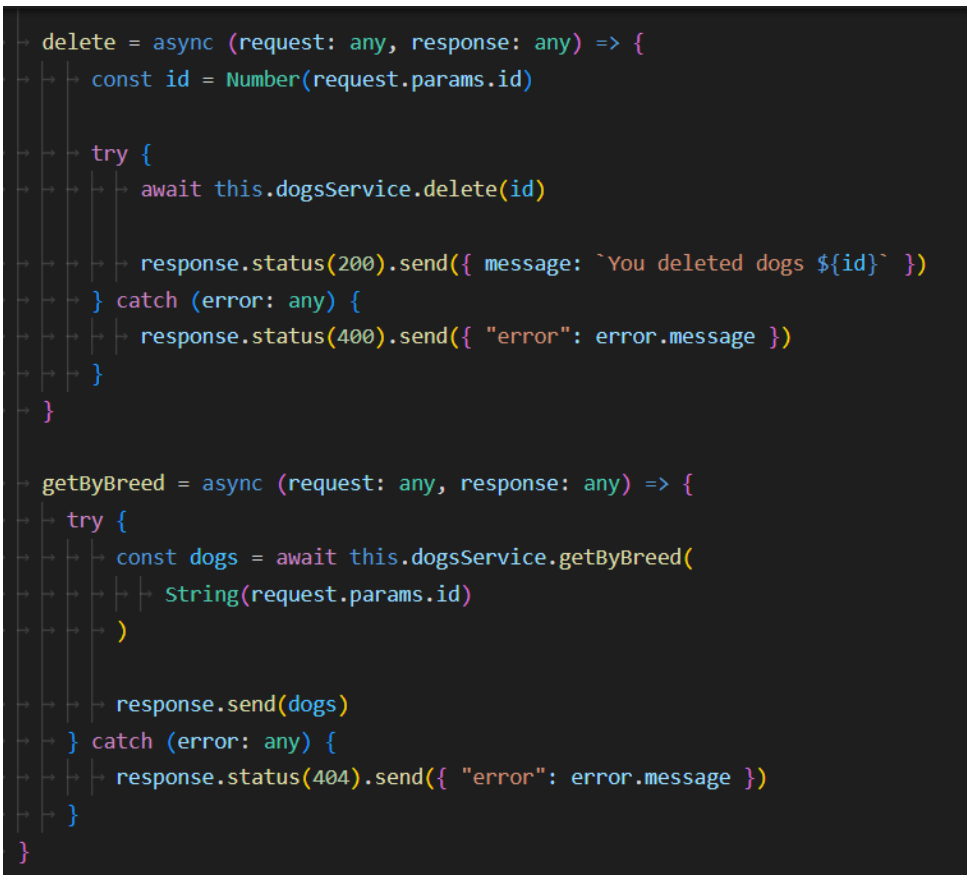


```
    delete = async (request: any, response: any) => {
        const id = Number(request.params.id)

        try {
            await this.companiesService.delete(id);

            response.status(200).send({ message: `You deleted company ${id}` });
        } catch (error: any) {
            response.status(400).send({ "error": error.message });

        }
    }

    getByOwner = async (request: any, response: any) => {
      try {
        const company = await this.companiesService.getByOwner(
          String(request.params.id)
        );

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

export default CompaniesController;
```

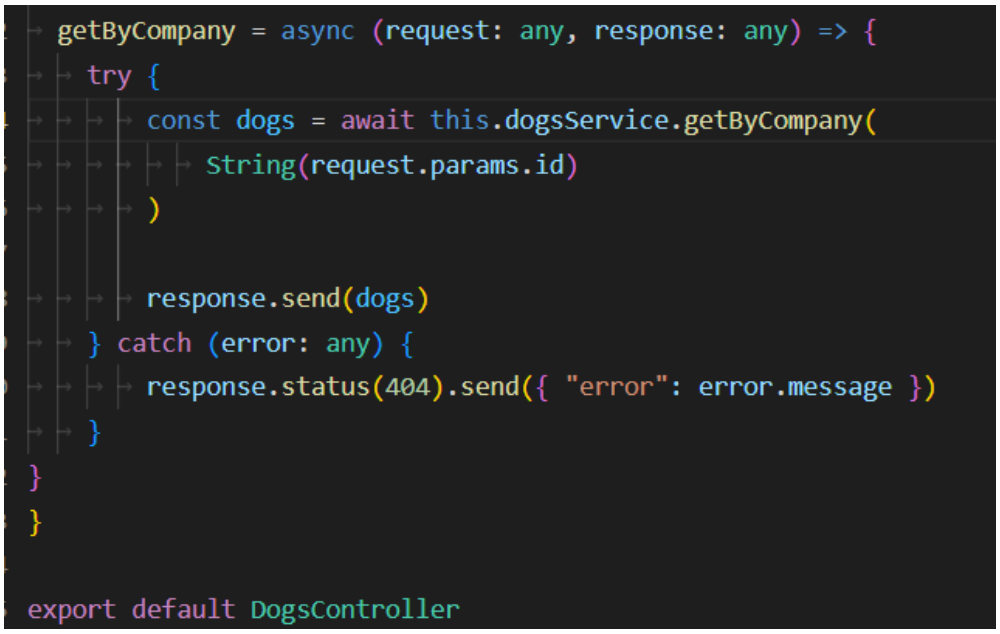
Рисунки 8,9,10- контролеры для компании

```
import DogsService from '../../services/dogs/dogs'

class DogsController {
    private dogsService: DogsService

    constructor() {
        this.dogsService = new DogsService()
    }

    get = async (request: any, response: any) => {
        try {
            const dogs = await this.dogsService.getById(
                Number(request.params.id)
            )

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

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

        try {
            const dogs = await this.dogsService.create(body)

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

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

        const id = Number(request.params.id)

        try {
            const dogs = await this.dogsService.update(id, body)

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



```
delete = async (request: any, response: any) => {
    const id = Number(request.params.id)

    try {
        await this.dogsService.delete(id)

        response.status(200).send({ message: `You deleted dogs ${id}` })
    } catch (error: any) {
        response.status(400).send({ "error": error.message })
    }
}

getByBreed = async (request: any, response: any) => {
    try {
        const dogs = await this.dogsService.getByBreed(
            String(request.params.id)
        )

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



```
getByCompany = async (request: any, response: any) => {
    try {
        const dogs = await this.dogsService.getByCompany(
            String(request.params.id)
        )

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

export default DogsController
```

Рисунки 11,12,13 – контролеры для собак

5. Далее я прописала роуты. У моделей есть схожие роуты:
    - Get по id
    - Создание
    - Обновление информации
    - Удаление записи

    А также отличающиеся пути для собак:

    - Получение списка всех собак по id компании

- Получение всех собак с определенной породой

И для компании:

- Получение списка всех компаний, принадлежащих определенному владельцу

```
import express from "express"
import DogsController from "../../../controllers/dogs/dogs"

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

const controller: DogsController = new DogsController()

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

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

router.patch('/update/:id', controller.update)

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

router.get('/breed/:id', controller.getByBreed)

router.get('/company/:id', controller.getByCompany)

export default router
```

Рисунок 14 – Роуты для собак

```
import express from "express"
import CompaniesController from "../../../controllers/dogs_companies/Dogs_company"

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

const controller: CompaniesController = new CompaniesController()

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

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

router.patch('/update/:id', controller.update)

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

router.get('/owner/:id', controller.getByOwner)

export default router
```

Рисунок 15 – Роуты для компаний

ВЫВОД

В ходе лабораторной работы мной был создан RESTful Api с логином, авторизацией, refreshtoken, фильтрацией, get и post запросами