

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэкэнд разработка

Отчет

Лабораторная работа №2: RESTful API

Выполнила:

Еремеева Арина

Группа К33412

Проверил: Добряков Д. И.

Санкт-Петербург

2023г.

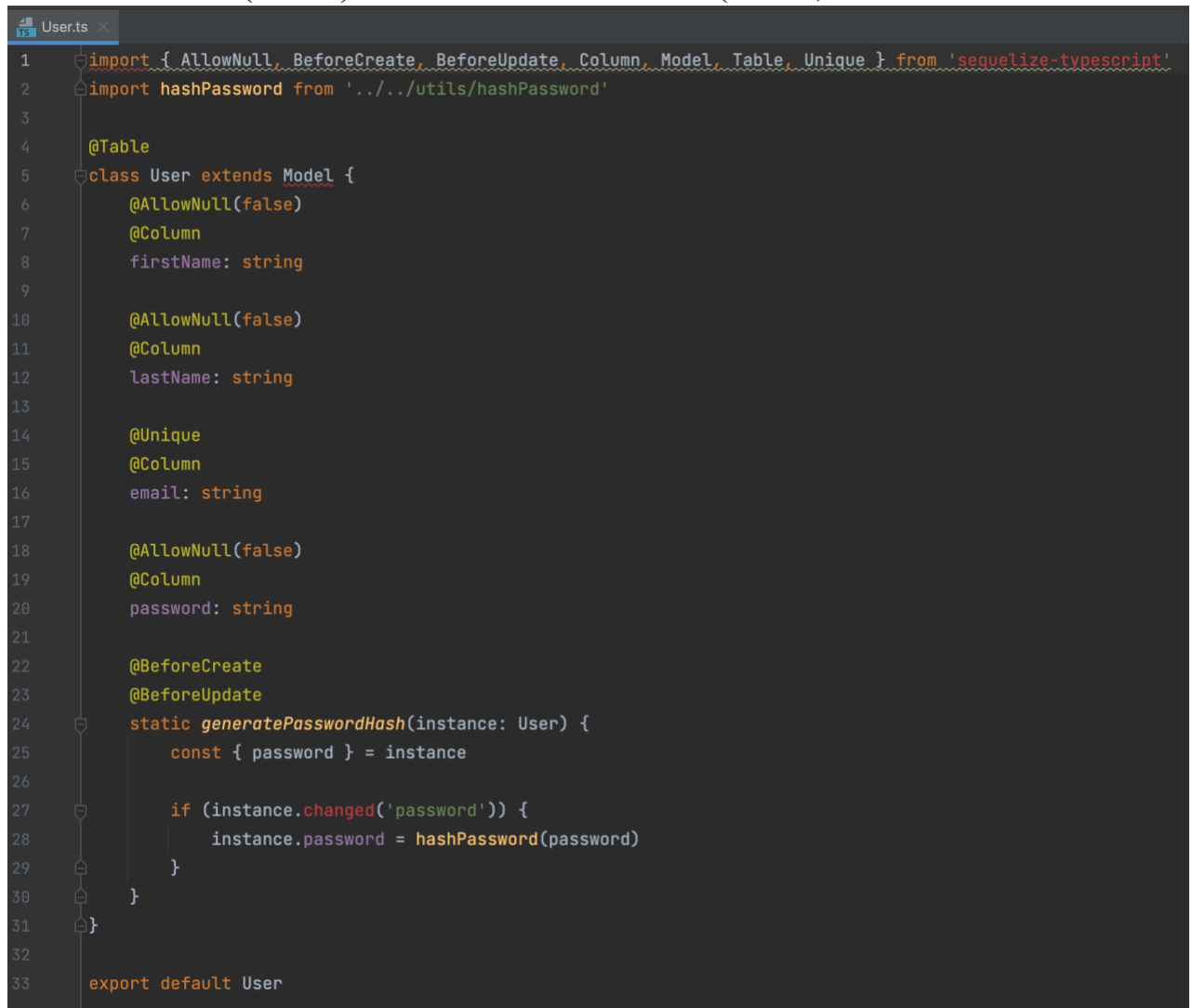
Цель: реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

Задачи:

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

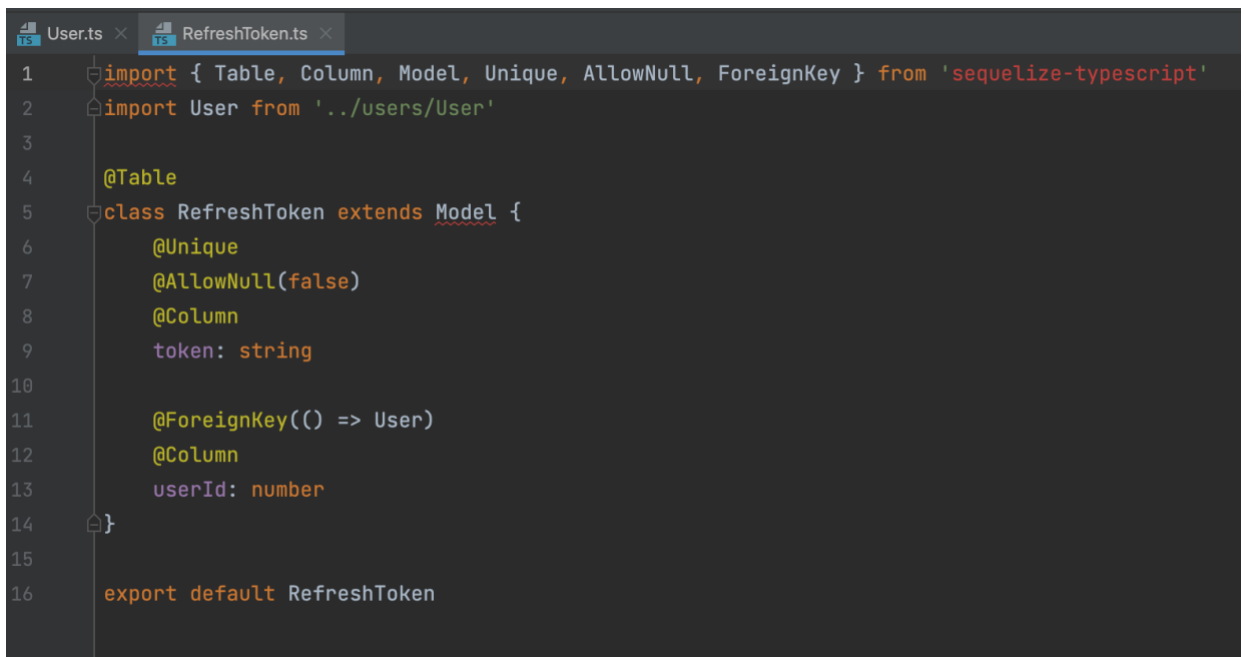
Ход работы:

1. Модель User (Рис. 1) и Модель RefreshToken (Рис. 2):



```
1  import { AllowNull, BeforeCreate, BeforeUpdate, Column, Model, Table, Unique } from 'sequelize-typescript'
2  import hashPassword from '../../utils/hashPassword'
3
4  @Table
5  class User extends Model {
6    @AllowNull(false)
7    @Column
8    firstName: string
9
10   @AllowNull(false)
11   @Column
12   lastName: string
13
14   @Unique
15   @Column
16   email: string
17
18   @AllowNull(false)
19   @Column
20   password: string
21
22   @BeforeCreate
23   @BeforeUpdate
24   static generatePasswordHash(instance: User) {
25     const { password } = instance
26
27     if (instance.changed('password')) {
28       instance.password = hashPassword(password)
29     }
30   }
31 }
32
33 export default User
```

Рисунок 1 – Модель User



```

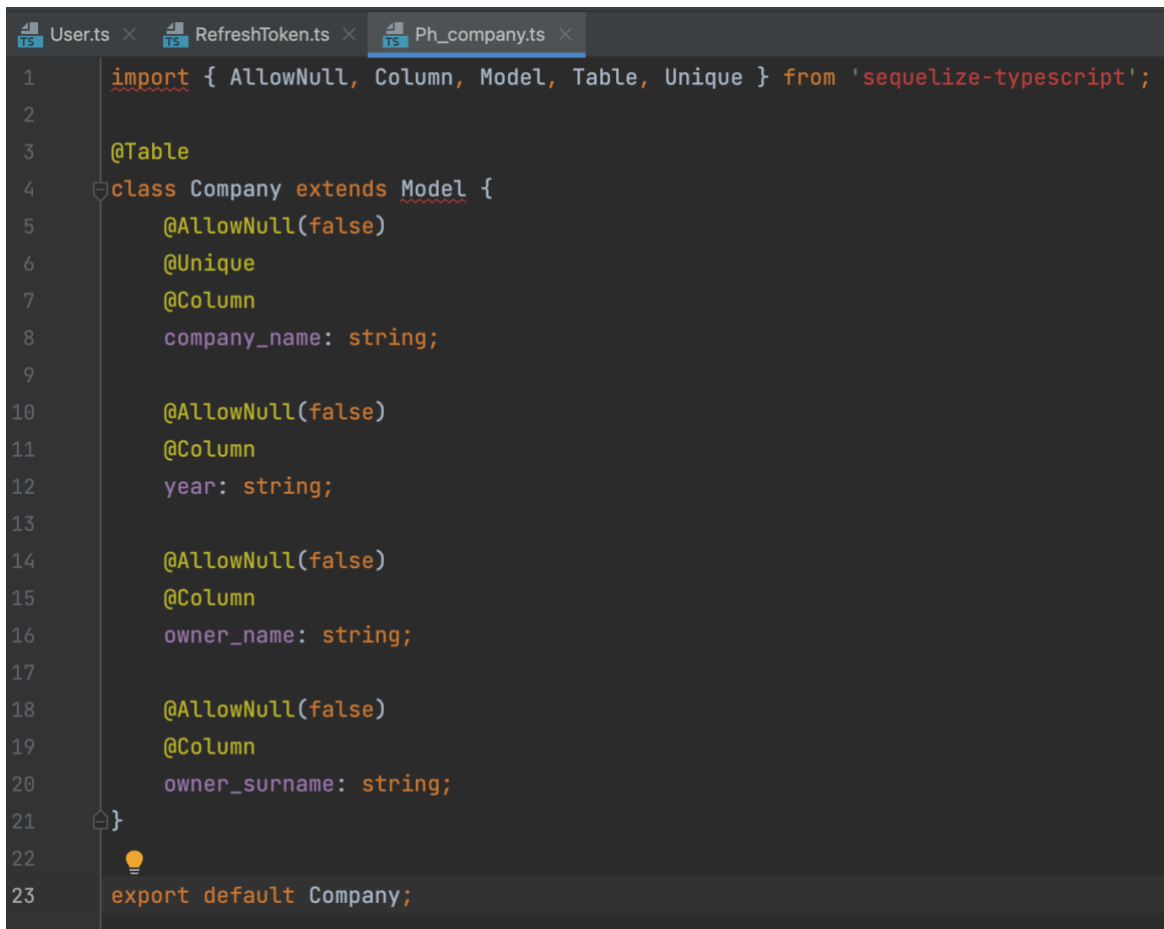
1  import { Table, Column, Model, Unique, AllowNull, ForeignKey } from 'sequelize-typescript'
2  import User from '../users/User'
3
4  @Table
5  class RefreshToken extends Model {
6      @Unique
7      @AllowNull(false)
8      @Column
9      token: string
10
11     @ForeignKey(() => User)
12     @Column
13     userId: number
14 }
15
16 export default RefreshToken

```

Рисунок 2 – Модель RefreshToken

2. Темой работы я выбрала сервис по определению топов фотографов в известных компаниях.

Я реализовала модели Компаний (Рис. 3) и Фотографов (Рис. 4):

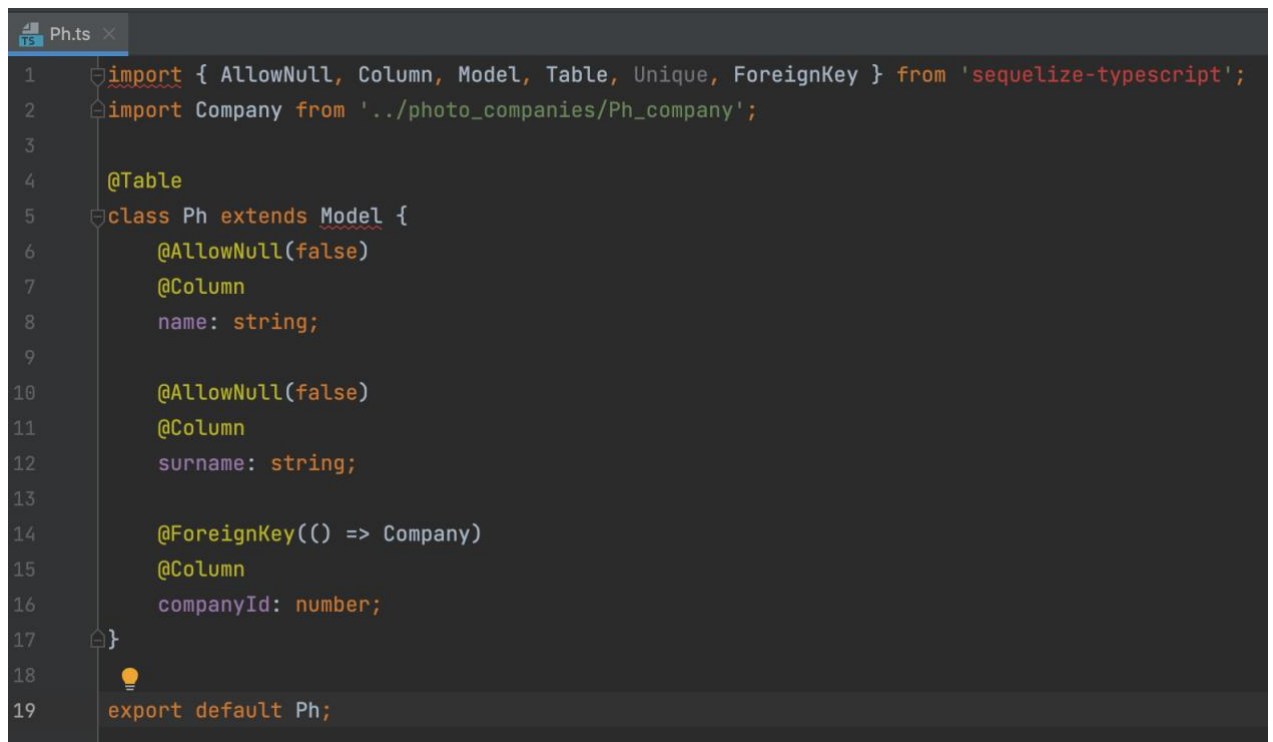


```

1  import { AllowNull, Column, Model, Table, Unique } from 'sequelize-typescript';
2
3  @Table
4  class Company extends Model {
5      @AllowNull(false)
6      @Unique
7      @Column
8      company_name: string;
9
10     @AllowNull(false)
11     @Column
12     year: string;
13
14     @AllowNull(false)
15     @Column
16     owner_name: string;
17
18     @AllowNull(false)
19     @Column
20     owner_surname: string;
21 }
22
23 export default Company;

```

Рисунок 3 – Модель компании



```

1  import { AllowNull, Column, Model, Table, Unique, ForeignKey } from 'sequelize-typescript';
2  import Company from '../photo_companies/Ph_company';
3
4  @Table
5  class Ph extends Model {
6      @AllowNull(false)
7      @Column
8      name: string;
9
10     @AllowNull(false)
11     @Column
12     surname: string;
13
14     @ForeignKey(() => Company)
15     @Column
16     companyId: number;
17 }
18
19 export default Ph;

```

Рисунок 4 – Модель фотографа

3. Далее мной были созданы services для Компаний (Рис. 5-6) и Фотографов (Рис. 7-8):

```

13 Ph_company.ts
1 import Company from '../../models/photo_companies/Ph_company';
2 import sequelize from '../../providers/db';
3
4 const companiesRepository = sequelize.getRepository(Company);
5
6 class CompaniesService {
7   async get(id: number): Promise<Company> {
8     const company = await companiesRepository.findOne({ where: { 'id': id } });
9     if (company) return company
10    throw new Error(`Company with id ${id} not found`);
11  }
12
13  async create(companyData: Partial<Company>): Promise<Company> {
14    try {
15      const company = await companiesRepository.create(companyData);
16      return company.toJSON();
17    } catch (e: any) {
18      const errors = e.errors.map((error: any) => error.message);
19      throw console.log(errors);
20    }
21  }
22
23  async update(id: number, companyData: Partial<Company>): Promise<Company> {
24    try {
25      const company = await companiesRepository.findOne({ where: { id } });
26
27      if (company) {
28        await company.update(companyData);
29        return company.toJSON();
30      }
31      throw new Error(`Worker with id ${id} not found`);
32    } catch (e: any) {
33      const errors = e.errors.map((error: any) => error.message);
34      throw console.log(errors);
35    }
36  }
37
38  async delete(id: number): Promise<void> {
39    const company = await companiesRepository.findOne({ where: { id } });
40    if (company) {
41      await company.destroy();
42      return;
43    }
44    throw new Error(`Company with id ${id} not found`);
45  }
46
47  async getByOwner(id: string): Promise<any> {
48    const company = await companiesRepository.findAll({ where: { 'owner_surname': id } });
49    if (company) return company
50    throw new Error(`${id}'s companies not found`);
51  }
52 }
53
54 export default CompaniesService;
55

```

Рисунки 5-6 – Services для Компаний

```

1  import Ph from '../models/photographers/Ph'
2  import sequelize from '../providers/db'
3
4  const photoRepository = sequelize.getRepository(Ph)
5
6  class PhotographersService {
7    async getById(id: number): Promise<Ph> {
8      const photo = await photoRepository.findOne({ where: { 'id': id } })
9      if (photo) return photo
10     throw new Error(`Photographers ${id} not found`)
11   }
12
13   async create(photoData: Partial<Ph>): Promise<Ph> {
14     try {
15       const photo = await photoRepository.create(photoData)
16       return photo.toJSON()
17     }
18     catch (e: any) {
19       const errors = e.errors.map((error: any) => error.message)
20       throw console.log(errors)
21     }
22   }
23
24   async update(id: number, photoData: Partial<Ph>): Promise<Ph> {
25     try {
26       const photo = await photoRepository.findOne({ where: { 'id': id } })
27
28       if (photo) {
29         await photo.update(photoData)
30         return photo.toJSON()
31       }
32       throw new Error(`Photographers ${id} not found`)
33     }
34     catch (e: any) {
35       const errors = e.errors.map((error: any) => error.message)
36       throw console.log(errors)
37     }
38   }
39
40   async delete(id: number): Promise<void> {
41     const photo = await photoRepository.findOne({ where: { 'id': id } })
42     if (photo) {
43       await photo.destroy()
44       return
45     }
46     throw new Error(`Photographers ${id} not found`)
47   }
48
49   async getBySurname(id: string): Promise<any> {
50     const photo = await photoRepository.findAll({ where: { 'surname': id } })
51     if (photo) return photo
52     throw new Error(`Photographers ${id} not found`)
53   }
54
55   async getByCompany(id: string): Promise<any> {
56     const photo = await photoRepository.findAll({ where: { 'companyId': id } })
57     if (photo) return photo
58     throw new Error(`Photographers ${id} not found`)
59   }
60 }
61
62 export default PhotographersService

```

Рисунок 7-8 – Services для Фотографов

4. Далее я создала контроллеры для Компаний (Рис. 9-10) и Фотографов (Рис. 11-12):



```
1  import CompaniesService from '../services/photo_companies/Ph_company';
2
3  class CompaniesController {
4      private companiesService: CompaniesService;
5
6      constructor() {
7          this.companiesService = new CompaniesService();
8      }
9
10     get = async (request: any, response: any) => {
11         try {
12             const company = await this.companiesService.get(
13                 Number(request.params.id)
14             );
15
16             response.send(company);
17         } catch (error: any) {
18             response.status(404).send({ "error": error.message });
19         }
20     };
21
22     create = async (request: any, response: any) => {
23         const { body } = request;
24
25         try {
26             const company = await this.companiesService.create(body);
27
28             response.status(200).send(company);
29         } catch (error: any) {
30             response.status(400).send({ "error": error.message });
31         }
32     };
33
34     update = async (request: any, response: any) => {
35         const { body } = request;
36         const id = Number(request.params.id);
37
38         try {
39             const worker = await this.companiesService.update(id, body);
40
41             response.send(worker);
42         } catch (error: any) {
43             response.status(400).send({ "error": "error" });
44         }
45     }
```

```

45     };
46
47     delete = async (request: any, response: any) => {
48         const id = Number(request.params.id)
49
50         try {
51             await this.companiesService.delete(id);
52
53             response.status(200).send({ message: `You deleted company ${id}` });
54         } catch (error: any) {
55             response.status(400).send({ "error": error.message });
56         }
57     }
58 }
59
60 getByOwner = async (request: any, response: any) => {
61     try {
62         const company = await this.companiesService.getByOwner(
63             String(request.params.id)
64         );
65
66         response.send(company);
67     } catch (error: any) {
68         response.status(404).send({ "error": error.message });
69     }
70 };
71 }
72
73 export default CompaniesController;

```

Рисунки 9-10 – контролеры для Компании


```

1  import PhotographersService from '../../../services/photographers/Ph'
2
3  class PhotographersController {
4      private photographersService: PhotographersService
5
6      constructor() {
7          this.photographersService = new PhotographersService()
8      }
9
10     get = async (request: any, response: any) => {
11         try {
12             const photo = await this.photographersService.getById(
13                 Number(request.params.id)
14             )
15
16             response.send(photo)
17         } catch (error: any) {
18             response.status(404).send({ "error": error.message })
19         }
20     }
21
22     create = async (request: any, response: any) => {
23         const { body } = request
24
25         try {
26             const photo = await this.photographersService.create(body)
27
28             response.status(200).send(photo)
29         } catch (error: any) {
30             response.status(400).send({ "error": error.message })
31         }
32     }
33
34     update = async (request: any, response: any) => {
35         const { body } = request
36
37         const id = Number(request.params.id)
38
39         try {
40             const photo = await this.photographersService.update(id, body)
41
42             response.send(photo)
43         } catch (error: any) {
44             response.status(400).send({ "error": error.message })

```

```

45     }
46   }
47
48   delete = async (request: any, response: any) => {
49     const id = Number(request.params.id)
50
51     try {
52       await this.photographersService.delete(id)
53
54       response.status(200).send({ message: `You deleted photographers ${id}` })
55     } catch (error: any) {
56       response.status(400).send({ "error": error.message })
57     }
58   }
59
60   getBySurname = async (request: any, response: any) => {
61     try {
62       const photo = await this.photographersService.getBySurname(
63         String(request.params.id)
64       )
65
66       response.send(photo)
67     } catch (error: any) {
68       response.status(404).send({ "error": error.message })
69     }
70   }
71
72   getByCompany = async (request: any, response: any) => {
73     try {
74       const photo = await this.photographersService.getByCompany(
75         String(request.params.id)
76       )
77
78       response.send(photo)
79     } catch (error: any) {
80       response.status(404).send({ "error": error.message })
81     }
82   }
83 }
84
85 export default PhotographersController

```

Рисунки 11-12 – контролеры для Фотографов

5. Далее я прописала роуты (Рис. 13-14).

У моделей есть схожие роуты:

- Get по id
- Создание
- Обновление информации
- Удаление записи

А также отличающиеся пути для Фотографов:

- Получение списка всех Фотографов по id Компании

- Получение всех Фотографов с определенной фамилией

И для Компании:

- Получение списка всех Компаний, принадлежащих определенному владельцу

```
1 import express from "express"
2 import PhotographersController from "../../controllers/photographers/Ph"
3
4 const router: express.Router = express.Router()
5
6 const controller: PhotographersController = new PhotographersController()
7
8 router.get( path:('/:id'), controller.get)
9
10 router.post( path: '/create', controller.create)
11
12 router.patch( path: '/update/:id', controller.update)
13
14 router.delete( path:('/:id'), controller.delete)
15
16 router.get( path: '/surname/:id', controller.getBySurname)
17
18 router.get( path: '/company/:id', controller.getByCompany)
19
20 export default router
```

Рисунок 13 – Роуты для Фотографов

```
Ph_companies.ts
1 import express from "express"
2 import CompaniesController from "../../controllers/photo_companies/Ph_company"
3
4 const router: express.Router = express.Router()
5
6 const controller: CompaniesController = new CompaniesController()
7
8 router.get( path:('/:id'), controller.get)
9
10 router.post( path: '/create', controller.create)
11
12 router.patch( path: '/update/:id', controller.update)
13
14 router.delete( path:('/:id'), controller.delete)
15
16 router.get( path: '/owner/:id', controller.getByOwner)
17
18 export default router
```

Рисунок 14 – Роуты для Компаний

ВЫВОД

В этой лабораторной работе мной был создан RESTful Api с логином, авторизацией, refreshtoken, фильтрацией, get и post запросами.