

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа № 2: REST, RESTful, SOAP,
GraphQL.

Выполнил:

Безруков Андрей
Группа К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

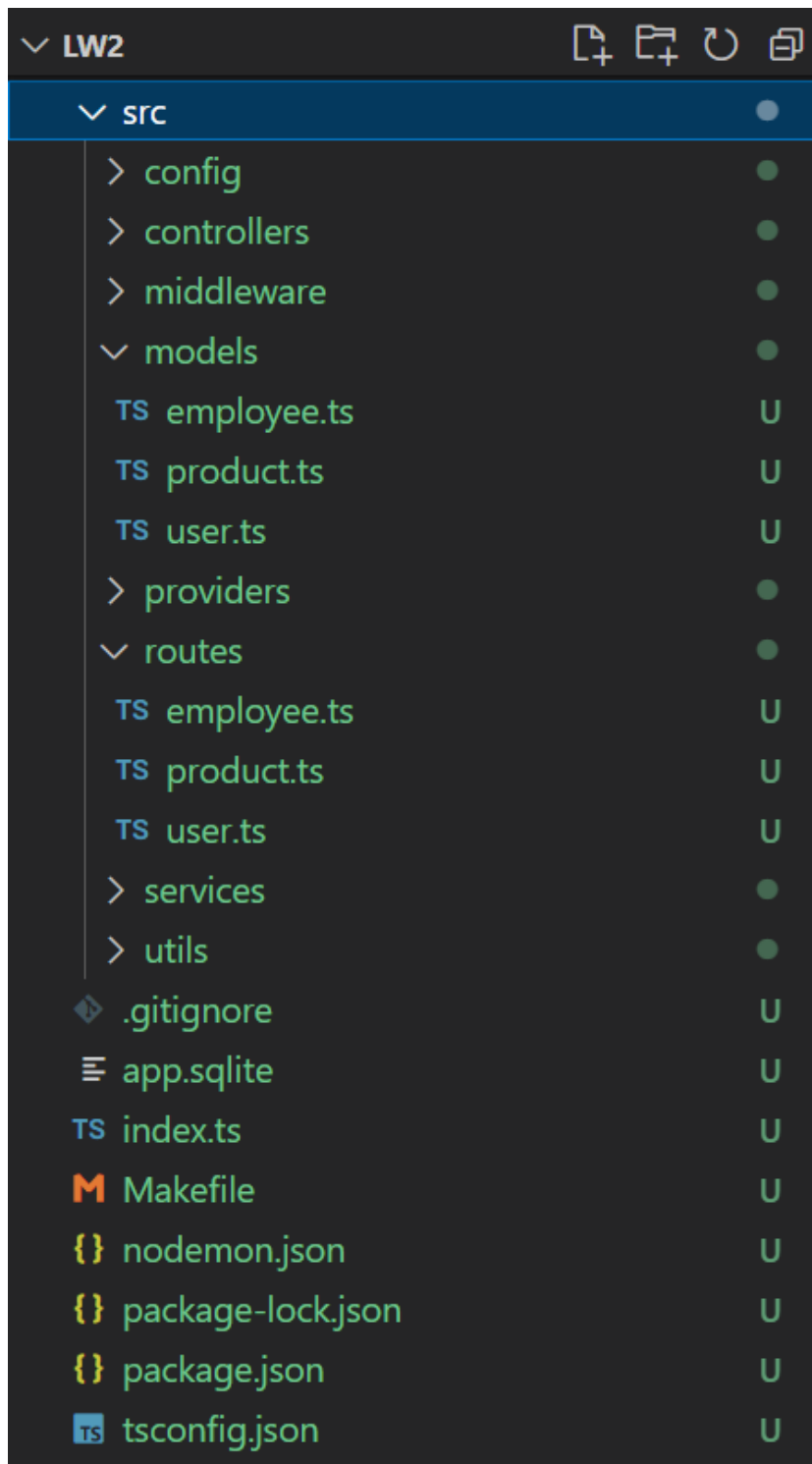
Задача

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

Вариант: онлайн-магазин Cute online shop API

Ход работы

Структура проекта:



Было реализовано 3 модели. Была использована библиотека sequelize-typescript.

- Пользователь – user.ts

```
1 import { Table, Model, PrimaryKey, Column, Unique, AllowNull } from "sequelize-typescript";
2
3 @Table
4 class User extends Model {
5     @PrimaryKey
6     @Column
7     id: number
8
9     @Unique
10    @AllowNull(false)
11    @Column
12    email: string
13
14    @AllowNull(false)
15    @Column
16    password: string
17
18    @Unique
19    @AllowNull(false)
20    @Column
21    username: string
22 }
23
24 export default User
```

- Товар – product.ts

```
api > src > models > TS productts > ...
1 import { Table, Model, PrimaryKey, Column, AllowNull } from "sequelize-typescript";
2
3 @Table
4 class Product extends Model {
5     @PrimaryKey
6     @Column
7     id: number
8
9     @AllowNull(false)
10    @Column
11    count: number
12
13    @AllowNull(false)
14    @Column
15    name: string
16
17    @AllowNull(false)
18    @Column
19    category: string
20
21    @AllowNull(false)
22    @Column
23    price: number
24 }
25
26 export default Product
```

- Работник – employee.ts

```
{ } package.json TS product.ts U TS employee.ts U X
api > src > models > TS employee.ts > [ ] default
1  import { Table, Model, Column, PrimaryKey, AllowNull } from "sequelize-typescript"
2
3  @Table
4  class Employee extends Model {
5      @PrimaryKey
6      @Column
7      id: number
8
9      @AllowNull(false)
10     @Column
11     firstName: string
12
13     @AllowNull(false)
14     @Column
15     lastName: string
16
17     @AllowNull(false)
18     @Column
19     position: string
20 }
21
22 export default Employee
```

Роутинг и эндпоинты:

- Пользователь — регистрация, логин, получение профиля авторизованного, всех пользователей, обновление и удаление пользователей

```
{ } package.json TS product.ts U TS employee.ts U TS users.ts U X
api > src > routes > TS users.ts > ...
1  import UserController from "../controllers/user";
2  import express from "express";
3  import auth from "../middleware/authenticate"
4
5  const router = express.Router()
6  const userController: UserController = new UserController()
7
8  router
9      .route('/signup')
10     .post(userController.signup)
11
12  router
13     .route('/login')
14     .post(userController.login)
15
16  router
17     .route('/profile')
18     .get(auth.authenticate, userController.profile)
19
20  router
21     .route('/all')
22     .get(auth.authenticate, userController.getAll)
23
24  router
25     .route('/update/:id')
26     .put(userController.put)
27
28  router
29     .route('/delete/:id')
30     .delete(userController.delete)
```

- Товар – получение всех товаров/по id/по категории/по названию, добавление, изменение, удаление товаров, подсчет товаров

```
{} package.json TS product.ts ...\models U TS employee.ts U TS product.ts ...\routes U X
api > src > routes > TS product.ts > ...
1 import ProdController from "../controllers/product";
2 import authMiddleware from "../middleware/authenticate"
3 import express from "express"
4
5 const router = express.Router()
6 const controller = new ProdController()
7
8 router
9   .route('/')
10    .get(authMiddleware.authenticate, controller.get)
11
12 router
13   .route('/')
14    .post(authMiddleware.authenticate, controller.post)
15
16 router
17   .route('/id/:id')
18    .get(authMiddleware.authenticate, controller.getById)
19
20 router
21   .route('/:id')
22    .put(authMiddleware.authenticate, controller.put)
23
24 router
25   .route('/:id')
26    .delete(authMiddleware.authenticate, controller.delete)
27
28 router
29   .route('/name/:name')
30    .get(authMiddleware.authenticate, controller.getByName)
31
32 router
33   .route('/category/:category')
34    .get(authMiddleware.authenticate, controller.getbyCategory)
35
36 router
37   .route('/count')
38    .get(authMiddleware.authenticate, controller.countProds)
39
40 export default router
```

- Работник – получение всех работников/по id/по должности, изменение, добавление, удаление работников

```
{ } package.json TS product.ts U TS employee.ts ...models U TS employee.ts ...routes U X
api > src > routes > TS employee.ts > ...
1 import EmpController from "../controllers/employee";
2 import express from "express"
3 import authMiddleware from "../middleware/authenticate"
4
5 const router = express.Router()
6 const controller: EmpController = new EmpController()
7
8 router
9   .route('/')
10  .get(authMiddleware.authenticate, controller.get)
11
12 router
13   .route('/')
14   .post(authMiddleware.authenticate, controller.post)
15
16 router
17   .route('/:id')
18   .get(authMiddleware.authenticate, controller.getbyId)
19
20 router
21   .route('/:id')
22   .put(authMiddleware.authenticate, controller.put)
23
24 router
25   .route('/:id')
26   .delete(authMiddleware.authenticate, controller.delete)
27
28 router
29   .route('/:position/:position')
30   .get(authMiddleware.authenticate, controller.getbyPosition)
```

Контролеры (пример для товаров)

```
1 import ProdService from "../services/product";
2
3 class ProdController {
4   private productService: ProdService
5   constructor() {
6     this.productService = new ProdService()
7   }
8
9   get = async (request: any, response: any) => {
10     try {
11       const products = await this.productService.getAll()
12       return response.json(products);
13     } catch (e: any) {
14       response.status(404).send({ "error": e.message })
15     }
16   }
17
18   getById = async (request: any, response: any) => {
19     try {
20       const product = await this.productService.getById(request.params.id)
21       return response.json(product);
22     } catch (error: any) {
23       response.status(404).send({ "error": error.message })
24     }
25   }
26 }
```

Сервисы (пример для пользователей). Здесь также хэшируется пароль и указывается, что войти в систему можно и с username, и с email.

```
1 import User from "../models/user"
2 import { passwordHash } from "../utils/passwordHash";
3 import { Op } from "sequelize";
4
5 class UserService {
6   async create(userInfo: any) {
7     userInfo.password = passwordHash(userInfo.password)
8     try {
9       const user = await User.create(userInfo)
10      return user
11    } catch (e: any) {
12      throw new Error(e)
13    }
14  }
15
16  async get(username: string, password: string): Promise<User> {
17    const hash: string = passwordHash(password);
18    const user: User | null = await User.findOne({
19      where: {
20        [Op.or]: {
21          username: username,
22          email: username,
23        },
24        password: hash
25      }
26    })
27
28    if (user == null) {
29      throw new Error("Credentials are invalid")
30    }
31  }
32 }
```

Функция хэширования пароля с использованием алгоритма sha512 – passwordHash.ts

```
1 import { createHash } from "crypto";
2
3 function passwordHash(password: string): string {
4   return createHash('sha512').update(password).digest('base64').toString()
5 }
6
7 export { passwordHash }
```

Хранение JWT-секрета – jwtsecret.ts

```
api > src > config > TS jwtsecret.ts > ...
```

```
1  const jwtsecret = "AZ95tw0Ionn7XPZZfN0N0ml3d9FMfmSgXwovR9Ap6ryDioGRU8EPHAB6iHsc0fb"  
2  
3  export default jwtsecret
```

Аутентификация реализована в файле `authenticate.ts`. Она происходит с использованием JWT-токена. Метод `jwt.sign` создает уникальную подпись, `jwt.verify` проверяет валидность токена.

```
5  class AuthMiddleware {  
6  
7      authenticate = async (request: any, response: any, next: any) => {  
8          var authHeader = request.headers.authorization  
9          if (authHeader) {  
10             try {  
11                 const token = authHeader.split(' ')[1]  
12                 const user = await this.authenticateToken(token)  
13                 request.user = user  
14  
15                 next()  
16             }  
17             catch (error: any) {  
18                 response.sendStatus(401)  
19             }  
20  
21             }  
22             else {  
23                 response.sendStatus(401)  
24             }  
25         }  
26  
27         createToken = async (user: any): Promise<string> => {  
28             return jwt.sign(lodash.pick(user, ['id', 'username']), jwtsecret)  
29         }  
30  
31         authenticateToken = async (token: string): Promise<any> => {  
32             return jwt.verify(token, jwtsecret)  
33         }  
34     }
```

Вывод

Было успешно реализовано API, все эндпоинты работают, так же была подключена авторизация с помощью токенов.