

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа

Выполнил:

Пырков Владислав

К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

- Продумать свою собственную модель пользователя
- Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize
- Написать запрос для получения пользователя по id/email

Ход работы

Модель пользователя в выбранном сервисе должна содержать данные об аккаунте (логин и пароль), а также почта, которая должна быть уникальной у каждого пользователя.

```
11  @Table
12  ✓ class User extends Model {
13      @PrimaryKey
14      @AutoIncrement
15      @Column(DataType.INTEGER)
16      declare id: number
17
18      @Column(DataType.STRING)
19      declare name: string
20
21      @Unique
22      @Column(DataType.STRING)
23      declare email: string
24
25      @Unique
26      @Column(DataType.STRING)
27      declare password: string
28  }
```

Далее создаем Controller для пользователя, подсоединяя методы из соответствующего сервиса.

```

labs > K33402 > Пытков_Владислав > lab1 > app > src > controllers > TS UserController.ts > default > getAllUsers
1  import { Request, Response } from 'express'
2  import jwt, { JwtPayload } from 'jsonwebtoken'
3
4  import UserService from '../services/UserService'
5  import handleError from '../utils/handleError'
6
7  export default {
8      async getUserById(req: Request, res: Response) {
9          try {
10             const id = Number(req.params.id)
11             const user = await UserService.getUserById(id)
12             return res.status(201).json(user)
13          } catch (error) {
14             return handleError({ res, error, code: 500 })
15          }
16      },
17
18 >  ⚡ async getAllUsers(req: Request, res: Response) { ...
25      },
26
27 >  async createUser(req: Request, res: Response) { ...
35      },
36
37 >  async updateUser(req: Request, res: Response) { ...
45      },
46
47 >  async deleteUser(req: Request, res: Response) { ...
55      },
56
57 >  async verify(req: Request, res: Response) { ...
71      },
72  }
73

```

Внутри сервиса описываем методы взаимодействия с пользователем, используя паттерн Репозиторий для прослойки между сервисом и моделью.

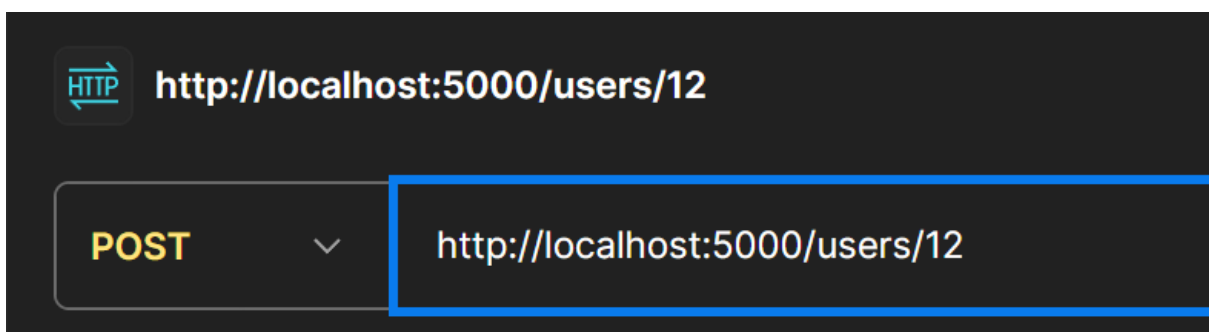
```

labs > K33402 > Пырков_Владислав > lab1 > app > src > services > TS UserService.ts >
1  ∨ import sequelize from '../database'
2  import User from '../database/models/User'
3  import serviceHandleError from '../utils/serviceHandleError'
4
5  const userRepository = sequelize.getRepository(User)
6
7  ∨ class UserService {
8  ∨    static async getUserById(id: number) {
9      return userRepository.findByPk(id)
10   }
11
12   > static async getAllUsers() { ...
14   }
15
16   > static async createUser(userData: any) { ...
20   }
21
22   > static async updateUser(id: number, userData: any) { ...
32   }
33
34   > static async deleteUser(id: number) { ...
42   }
43   }
44
45   export default UserService
46   |

```

В итоге получаем удобное описание всех свойств модели пользователя и взаимодействия с ним.

Для создания запроса, обращаемся к эндпоинту `users/:id` с методом `get`, тем самым получая данные о пользователе по его `id`.



Вывод

В процессе выполнения продумали собственную модель пользователя, поработали с express и Sequelize, получили опыт составления RESTful API для модели пользователя