САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая работа 2

Выполнил:

Казанков Илья К33402

Проверил: Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

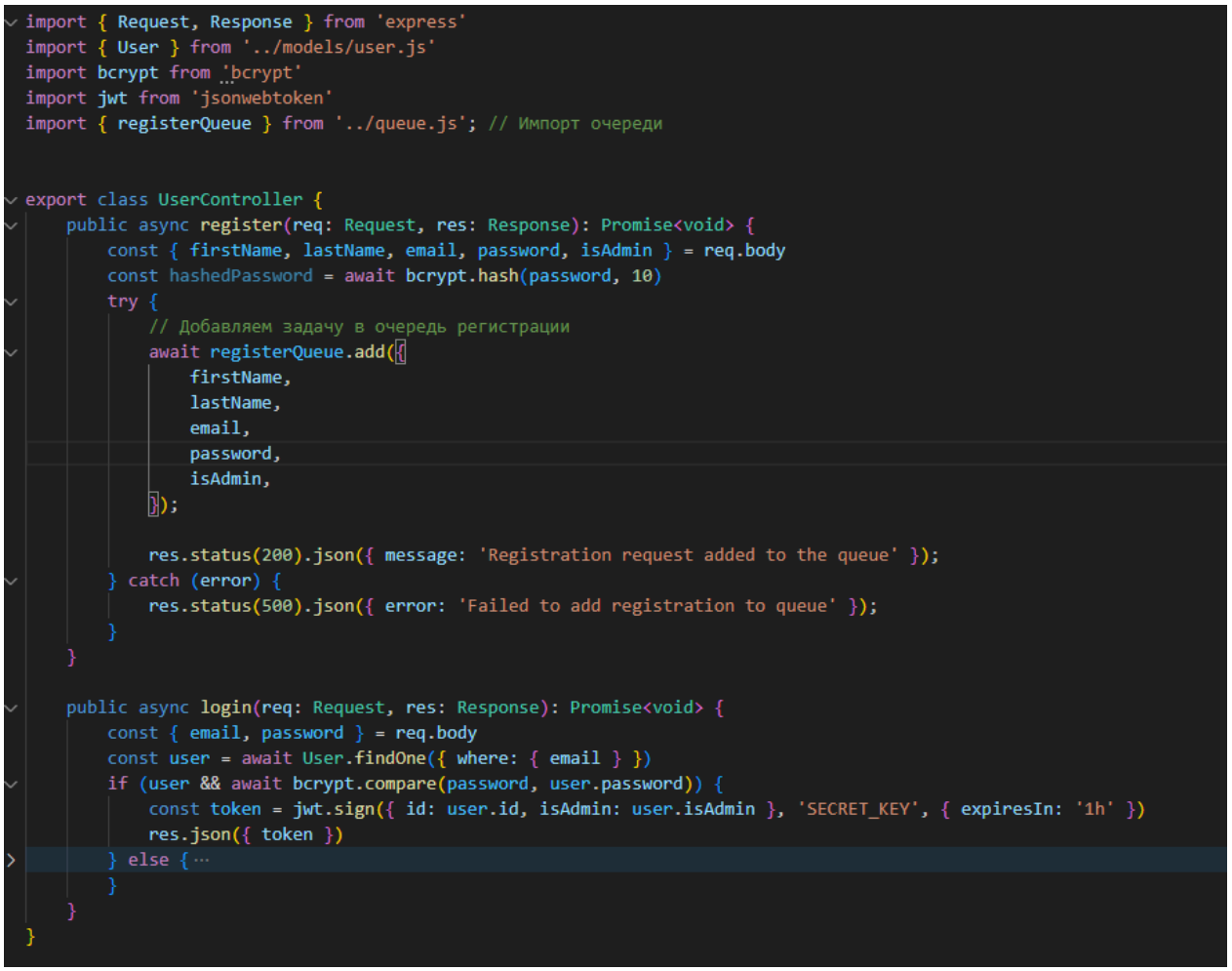
- Продумать свою собственную модель пользователя
- Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize
- Написать запрос для получения пользователя по id/email

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.

```
export class User extends Model {
    @Unique
    @PrimaryKey
    @AutoIncrement
    @Column
    declare id: number;

    @Column
    declare firstName: string;

    @Column
    declare lastName: string;

    @Column
    declare email: string;

    @Column
    declare password: string;

    @Default(false)
    @Column
    declare isAdmin: boolean;
}
```

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



```
import { Request, Response } from 'express'
import { User } from '../models/user.js'
import bcrypt from 'bcrypt'
import jwt from 'jsonwebtoken'
import { registerQueue } from '../queue.js'; // Импорт очереди


export class UserController {
    public async register(req: Request, res: Response): Promise<void> {
        const { firstName, lastName, email, password, isAdmin } = req.body
        const hashedPassword = await bcrypt.hash(password, 10)
        try {
            // Добавляем задачу в очередь регистрации
            await registerQueue.add({
                firstName,
                lastName,
                email,
                password,
                isAdmin,
            });

            res.status(200).json({ message: 'Registration request added to the queue' });
        } catch (error) {
            res.status(500).json({ error: 'Failed to add registration to queue' });
        }
    }

    public async login(req: Request, res: Response): Promise<void> {
        const { email, password } = req.body
        const user = await User.findOne({ where: { email } })
        if (user && await bcrypt.compare(password, user.password)) {
            const token = jwt.sign({ id: user.id, isAdmin: user.isAdmin }, 'SECRET_KEY', { expiresIn: '1h' })
            res.json({ token })
        } else {…
        }
    }
}
```

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

```ts
import { User } from '../models/user.js'
import sequelize from '../index.js'


const UserRepository = sequelize.getRepository(User);

export class UserService {
    public async findByEmail(email: string): Promise<User | null> {
        return await UserRepository.findOne({ where: { email } })
    }

    public async createUser(data: Partial<User>): Promise<User> {
        return await UserRepository.create(data)
    }
}
```

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

Вывод

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