САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа 6

Выполнила:

Казарян Тигран

Группа К3441

Проверил:

Добряков Д. И.

Санкт-Петербург

2026 г.

СОДЕРЖАНИЕ

Задачи

- Подключить и настроить rabbitMQ/kafka;
- Реализовать межсервисное взаимодействие посредством rabbitMQ/kafka.

Ход работы

Листинг 1 – Подключение Bull в app.module.ts

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import {HttpModule} from "@nestjs/axios";
import {HealthController} from "./Health.controller";

@Module({
  imports: [HttpModule.register({}),],
  controllers: [AppController, HealthController],
  providers: [AppService],
})
export class AppModule {}

@Module({
    imports: [
        ConfigModule.forRoot({
            isGlobal: true,
            }),
        BullModule.forRoot({
            redis: {
                host: 'localhost',
                port: 6300,
                },
            }),
        UsersModule,
        ],
})
```

Модуль импортирует ConfigModule для глобальной конфигурации, BullModule для работы с очередями на Redis и UsersModule - пользовательский модуль приложения. BullModule настраивается на подключение к Redis с хостом localhost и портом 6300.

Листинг 2 – Создание очереди в process.ts

```
import { Process, Processor } from '@nestjs/bull';
import bull from 'bull';

@Processor('quiz')
export class QuizProcessor {
    @Process('processQuiz')
    async handleProcessQuiz(job: bull.Job<{ quizId: string
        }>): Promise<{ status: string; quizId: string }>
    { console.log('Обработка квиза:', job.data.quizId);

        await new Promise(resolve => setTimeout(resolve, 3000));

        return { status: 'completed', quizId: job.data.quizId};
        }
    }

```

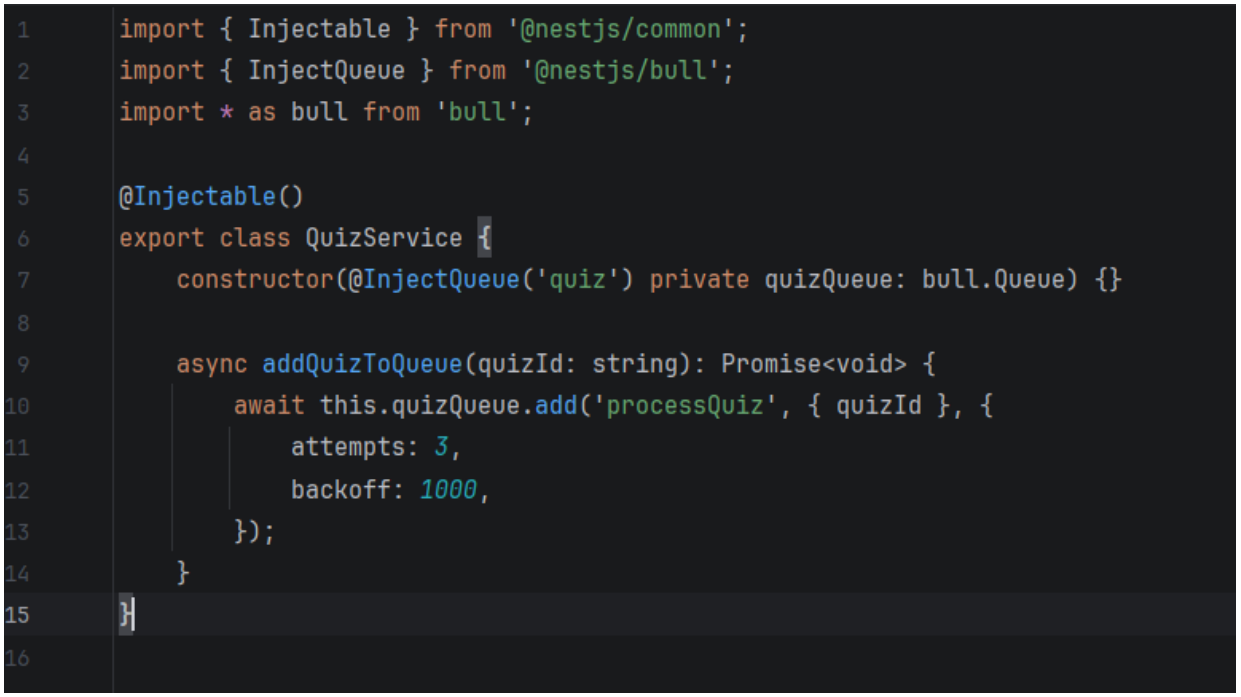
Процессор для очереди Bull обрабатывает задания для квизов. Он обрабатывает задания из очереди quiz, где метод handleProcessQuiz выполняет задание типа processQuiz. Процессор имитирует обработку с задержкой 3 секунды и возвращает объект со статусом завершения и ID квиза.

Листинг 3 – Регистрация очереди в модуле quiz.module.ts

```
@Module({
    imports: [ BullModule.registerQueue({
        name: 'quiz',
        }),
    ],
})
```

Фрагмент модуля регистрирует очередь Bull с именем quiz, которая будет использоваться для обработки задач, связанных с квизами.

Листинг 4 – Добавление логики в сервис quiz.service.ts



```
import { Injectable } from '@nestjs/common';
import { InjectQueue } from '@nestjs/bull';
import * as bull from 'bull';

@Injectable()
export class QuizService {
    constructor(@InjectQueue('quiz') private quizQueue: bull.Queue) {}

    async addQuizToQueue(quizId: string): Promise<void> {
        await this.quizQueue.add('processQuiz', { quizId }, {
            attempts: 3,
            backoff: 1000,
        });
    }
}
```

Сервис добавляет задания в очередь Bull. Он внедряет очередь quiz через декоратор @InjectQueue и содержит метод addQuizToQueue, который добавляет задание типа processQuiz с параметрами: quizId - идентификатор квиза, attempts: 3 - максимальное количество попыток выполнения и backoff: 1000 - задержка между повторными попытками (1 секунда).

Листинг 5 – Добавление логики в контроллер quiz.controller.ts

```
@Post(':id/process')
    async processQuiz(@Param('id') id: string) {
        await this.quizService.addQuizToQueue(id);
    return { message: 'Задача добавлена в очередь' };
}
```

Endpoint контроллера с декоратором @Post принимает параметр id из URL, вызывает сервис для добавления квиза в очередь на обработку и возвращает сообщение о том, что задача была добавлена в очередь.

Вывод

В данной лабораторной работе я научился создавать и работать с очередью Bull. Я реализовал асинхронную обработку задач с использованием очередей Bull на базе Redis, что позволяет отделить длительные операции от основного потока обработки запросов.

Когда клиент отправляет POST запрос на endpoint quiz/:id/process, контроллер передает id в сервис, который добавляет задание в очередь Bull. Processor слушает очередь quiz, получает задание, обрабатывает его в течение 3 секунд и возвращает результат. Если обработка не удалась, задание повторяется до 3 раз с интервалом в 1 секунду между попытками.

Такой подход значительно улучшает отзывчивость API и позволяет эффективнее использовать ресурсы приложения.