

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчёт

Лабораторная работа 6

Выполнил:

Шайтор Илья

Группа К3439

**Проверил:
Добряков Д. И.**

Санкт-Петербург

2026 г.

Задача

- подключить и настроить RabbitMQ/kafka;
- реализовать межсервисное взаимодействие посредством RabbitMQ/kafka.

Ход работы

```
async companyGetById(id: number) {
  const company = await this.prisma.company.findUnique({
    where: { id },
  });

  if (!company) {
    throw new NotFoundException("company not found");
  }

  await this.userClient.connect();
  if (company.industry_id) {
    await this.industryClient.connect();
  }

  try {
    const user = await firstValueFrom(
      this.userClient.send("get_user_by_id", company.user_id),
      { defaultValue: null },
    );

    let industry = null;
    if (company.industry_id) {
      industry = await firstValueFrom(
        this.industryClient.send("get_industry_by_id", company.industry_id),
        { defaultValue: null },
      );
    }
  }

  return {
    ...company,
    user,
    industry,
  };
} catch (error) {
  console.error("RabbitMQ error:", error);
  return {
    ...company,
    user: null,
    industry: null,
  };
}

async companyCreate(dto: CreateCompaniesDto) {
  const user = await this.prisma.user.findUnique({
    where: { id: dto.user_id },
  });
}
```

```

    });
    if (!user) {
      throw new NotFoundException("User not found");
    }

  const existingCompany = await this.prisma.company.findUnique({
    where: { user_id: dto.user_id },
  });
  if (existingCompany) {
    throw new ConflictException("User already has a company");
  }

  if (dto.industry_id) {
    const industry = await this.prisma.industry.findUnique({
      where: { id: dto.industry_id },
    });
    if (!industry) {
      throw new NotFoundException("Industry not found");
    }
  }

  return this.prisma.company.create({
    data: dto,
  });
}

```

Была создана функция для поиска компании по ID с подключением к таблицам через RabbitMQ

```

@Module({
  imports: [
    ClientsModule.register([
      {
        name: "USER_SERVICE",
        transport: Transport.RMQ,
        options: {
          urls: ["amqp://rabbitmq:5672"],
          queue: "user_queue",
          queueOptions: {
            durable: false,
          },
        },
      },
      {
        name: "INDUSTRY_SERVICE",
        transport: Transport.RMQ,
        options: {
          urls: ["amqp://rabbitmq:5672"],
          queue: "industry_queue",
          queueOptions: {
            durable: false,
          },
        },
      },
    ],
  ],
})

```

```
        },
        ],
    ],
    controllers: [CompanyController],
    providers: [CompanyService, PrismaService, ConfigService],
})
```

Также был написан модуль для взаимодействия с RabbitMQ

```
app.connectMicroservice<MicroserviceOptions>({
  transport: Transport.RMQ,
  options: {
    urls: ["amqp://rabbitmq:5672"],
    queue: "industry_queue",
    queueOptions: { durable: false },
  },
});
```

Для взаимодействия с сервисом RabbitMQ в микросервисах был прописано подключение к его порту

```
rabbitmq:
  image: rabbitmq:3-management
  container_name: jobbord-rabbitmq
  restart: always
  ports:
    - "5672:5672"
    - "15672:15672"
  environment:
    - RABBITMQ_DEFAULT_USER=guest
    - RABBITMQ_DEFAULT_PASS=guest
  networks:
    - jobboard-network
```

После чего в docker-compose был прописан контейнер для запуска RabbitMQ. Также для подключения к этому серверу в других docker файлах остальных микросервисов был прописан скрипт для подключения

Вывод

В микросервисы были внедрена очередь RabbitMQ, для обмена данными между собой.