

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №6

Выполнил:

Русинов Василий

Группа К3440

**Проверил:
Добряков Д. И.**

Санкт-Петербург

2026 г.

Задача

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

Ход работы

В рамках лабораторной работы использовалось микросервисное приложение для системы аренды недвижимости, состоящее из сервисов аутентификации (auth-service), управления пользователями (user-service), работы с объектами недвижимости (property-service) и управления арендой (rental-service). Каждый сервис запускается в отдельном Docker-контейнере и использует собственную базу данных PostgreSQL. Для асинхронного взаимодействия между сервисами был подключён брокер сообщений RabbitMQ, который также запускается в контейнере с помощью docker compose.

Для работы с очередями сообщений был реализован общий модуль, используемый всеми сервисами. В этом модуле реализовано подключение к RabbitMQ, отправка сообщений в очередь и получение сообщений из очереди. Для подключения используется общая очередь "microservice_events", через которую передаются события между сервисами.

При возникновении событий в одном сервисе, например при регистрации пользователя в auth-service или создании объекта недвижимости в property-service, формируется структурированное сообщение с типом события и соответствующими данными. Это сообщение отправляется в очередь RabbitMQ. Остальные сервисы подписаны на эту очередь и получают сообщения при их появлении. Например:

- При регистрации пользователя auth-service отправляет событие "USER_REGISTERED"
- User-service получает это событие и создает профиль пользователя
- Property-service и rental-service обновляют свои кэши данных о пользователях

Таким образом сервисы обмениваются событиями без прямых HTTP-запросов друг к другу, что снижает связность системы и повышает её отказоустойчивость.

Каждый сервис при запуске сначала подключается к своей базе данных PostgreSQL, затем устанавливает соединение с RabbitMQ и начинает прослушивать очередь сообщений. Для надёжности подключения реализованы повторные попытки соединения с RabbitMQ при старте контейнеров с использованием экспоненциальной задержки между попытками.

В docker-compose.yml была добавлена конфигурация для RabbitMQ:

```
rabbitmq:  
  image: rabbitmq:3-management  
  ports:  
    - "5672:5672"  
    - "15672:15672"  
  environment:  
    RABBITMQ_DEFAULT_USER: admin  
    RABBITMQ_DEFAULT_PASS: admin
```

Работа системы была протестирована после запуска всех контейнеров. Сообщения успешно отправляются и принимаются сервисами, что подтверждает корректную реализацию межсервисного взаимодействия посредством RabbitMQ. Были проверены сценарии:

1. Регистрация пользователя и автоматическое создание профиля
2. Обновление данных пользователя и синхронизация между сервисами
3. Создание объекта недвижимости и уведомление связанных сервисов

Вывод

В результате работы было реализовано межсервисное взаимодействие с использованием RabbitMQ. Брокер сообщений был успешно подключён и настроен, а сервисы обмениваются событиями через очередь сообщений. Реализация асинхронного взаимодействия позволила снизить связность между микросервисами, повысить отказоустойчивость системы и обеспечить лучшую масштабируемость отдельных компонентов. Все требования лабораторной работы выполнены, система демонстрирует корректную работу в контейнеризированной среде.

