САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Отчет по домашней работе №5 по курсу “Бэкенд разработка”

Выполнили:
Бахарева М. А., К3342
Привалов К.А., К3342

Проверил:
Добряков Д.И.

Санкт-Петербург
2025 г.

1. Задание:

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

2. Ход работы

Наша задача - настроить получение лога в Gateway при успешном логине пользователя в User Service. Поэтому Gateway - это messageConsumer, а User Service - messageBroker.

1. Настроим окружения в каждом сервисе

```
mariabakhareva@Marias-MacBook-Air-2 property-service % npm install amqplib

npm WARN deprecated inflight@1.0.6: This module is not supported, and leaks memory. Do not use it. Check out lru-cache if you want a good and tested way to coalesce async
requests by a key value, which is much more comprehensive and powerful.
npm WARN deprecated lodash.get@4.4.2: This package is deprecated. Use the optional chaining (?.) operator instead.
npm WARN deprecated lodash.isequal@4.5.0: This package is deprecated. Use require('node:util').isDeepStrictEqual instead.
npm WARN deprecated glob@7.1.6: Glob versions prior to v9 are no longer supported

added 264 packages, and audited 265 packages in 12s

38 packages are looking for funding
  run `npm fund` for details
```

2. Реализуем логику отправки сообщения в User Service:

```
async login(email: string, password: string) {
 const user = await this.repo.findOne({
   where: { email },
   select: [
     'id',
     'email',
     'password',
     'role',
     'firstName',
     'lastName',
     'phone',
     'birthDate',
   ],
 });
 if (!user) throw new Error('User not found');
 if (!comparePassword(password, user.password)) {
   throw new Error('Invalid password');
 }
 const token = signJwt({ userId: user.id, role: user.role });

 sendUserLoggedInMessage({
   userId: user.id,
   email: user.email,
   timestamp: new Date().toISOString(),
```

```
 });

 return { user, token };
}
```

3. Пропишем файл messageBroker

```
import amqp from 'amqplib';

let channel: amqp.Channel;

export async function connectRabbitMQ() {
 try {
   const connection = await amqp.connect('amqp://guest:guest@rabbitmq:5672');
   channel = await connection.createChannel();
   await channel.assertQueue('user_logged_in');
   console.log('✅ Connected to RabbitMQ');
 } catch (error) {
   console.error('❌ Failed to connect to RabbitMQ', error);
   throw error;
 }
}

export function sendUserLoggedInMessage(payload: any) {
 if (!channel) throw new Error('RabbitMQ channel not initialized');
 channel.sendToQueue('user_logged_in', Buffer.from(JSON.stringify(payload)));
}
```

4. Обновим index.ts в UserService для работы с RabbitMQ (при этом добавим логику повторного подключения, так как в docker-compose все сервисы запускаются одновременно, но RabbitMQ требует большего времени для инициализации)

```
import { AppDataSource } from './config/databaseConfig';
import app from './app';
import { connectRabbitMQ } from './messageBroker';

const PORT = process.env.PORT || 3000;

async function retryRabbitMQ(retries = 5, delay = 3000) {
 for (let attempt = 1; attempt <= retries; attempt++) {
   try {
     console.log(`🔁 Attempt ${attempt} to connect to RabbitMQ...`);
     await connectRabbitMQ();
     console.log('✅ Successfully connected to RabbitMQ');
     return;
   } catch (err) {
     const message = err instanceof Error ? err.message : String(err);
     console.error(`❌ RabbitMQ connection failed (attempt ${attempt}):
${message}`);

     if (attempt < retries) {
```

```
      await new Promise(res => setTimeout(res, delay));
    } else {
      throw new Error('❌ All attempts to connect to RabbitMQ failed.');
    }
  }
 }
}

async function startServer() {
 try {
  await AppDataSource.initialize();
  console.log('🗄️  Data Source has been initialized!');

  await retryRabbitMQ();

  app.listen(PORT, () => {
    console.log(`🚀 Server is running on port ${PORT}`);
  });
 } catch (err) {
  console.error('❌ Error during initialization', err);
  process.exit(1);
 }
}

startServer();
```

5. Пропишем Message Consumer в Gateway

```
import amqp from 'amqplib';

export async function startConsuming() {
 const connection = await amqp.connect('amqp://guest:guest@rabbitmq:5672');
 const channel = await connection.createChannel();
 await channel.assertQueue('user_logged_in');

 channel.consume('user_logged_in', (msg) => {
  if (msg !== null) {
    const data = JSON.parse(msg.content.toString());
    console.log('User logged in:', data);
    channel.ack(msg);
  }
 });
}
```

6. Обновим index.ts

```
async function startRabbitMQConsumerWithRetry(retries = 5, delay = 3000) {
 for (let attempt = 1; attempt <= retries; attempt++) {
  try {
    console.log(`🔁 Attempt ${attempt} to connect to RabbitMQ...`);
    const connection = await amqp.connect('amqp://guest:guest@rabbitmq:5672');
```

```
    const channel = await connection.createChannel();

    const queue = 'user_logged_in';
    await channel.assertQueue(queue, { durable: true });

    console.log(`🐇 Gateway is waiting for messages in queue: ${queue}`);

    channel.consume(queue, (msg) => {
      if (msg !== null) {
        const content = msg.content.toString();
        try {
          const data = JSON.parse(content);
          console.log('📩 Received user_logged_in event:', data);
        } catch (err) {
          console.error('❌ Failed to parse message content:', err);
        }
        channel.ack(msg);
      }
    });

    return; // успешное подключение - выходим из retry цикла
  } catch (err) {
    const message = err instanceof Error ? err.message : String(err);
    console.error(`❌ Failed to connect to RabbitMQ (attempt ${attempt}):
${message}`);

    if (attempt < retries) {
      console.log(`⏳ Waiting ${delay}ms before retry...`);
      await new Promise(res => setTimeout(res, delay));
    } else {
      console.error('❌ All attempts to connect to RabbitMQ failed.
Exiting.');
      process.exit(1);
    }
  }
 }
}
```

7. Обновим docker-compose, добавим в Services RabbitMQ, а также проставим у сервисов dependencies

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

```
networks:
  - backend_lr4_network
```

8. Теперь перейдем к интересному - к тестированию. Получаем успешные логи при запуске контейнеров:

```
gateway          | 🐇 Gateway is waiting for messages in queue: user_logged_in
users_service    | 🔁 Attempt 4 to connect to RabbitMQ...
```

```
users_service    | ✅Connected to RabbitMQ
users_service    | ✅Successfully connected to RabbitMQ
users_service    | 🚀 Server is running on port 3000
```

Значит можно переходить к тестированию отправки событий.

Зарегистрируем пользователя и залогиним его:

```
/app # curl -X POST http://localhost:3000/api/users/register \
>   -H "Content-Type: application/json" \
>   -d '{"email":"test@test.com","password":"Password1!","firstName":"Test","lastName":"User","birthDate":"1990-01-01","phone":"1234567890","role":"tenant"}'
{"firstName":"Test","lastName":"User","birthDate":"1990-01-01","phone":"1234567890","email":"test@test.com","role":"tenant","id":1,"createdAt":"2025-06-16T19:10:2
3.019Z"}/app # curl -X POST http://localhost:3000/api/users/login \
>   -H "Content-Type: application/json" \
>   -d '{"email":"test@test.com","password":"Password1!"}'
{"finalUser":{"id":1,"name":"Test","lastName":"User","email":"test@test.com","role":"tenant"},"token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjEsInJvbG
UiOiJ0ZW5hbnQiLCJpYXQiOjE3NTAxMDExMDMsImV4cCI6MTc1MDEzNzEwM30.esWyDyDFV15fQY58d_xQHHcZP5SIxbbJRSL9sRlzAr4"}/app #
```

Получаем лог в Gateway об успешном логине пользователя:

```
users_service    | POST /api/users/register 400 32.583 ms - 725
users_service    | POST /api/users/register 201 177.254 ms - 170
users_service    | POST /api/users/login 200 168.441 ms - 269
gateway          | 📩 Received user_logged_in event: {
gateway          |   userId: 1,
gateway          |   email: 'test@test.com',
gateway          |   timestamp: '2025-06-16T19:11:43.697Z'
gateway          | }
```

Делаем вывод, что взаимодействие через RabbitMQ успешно!

3. Вывод

В ходе выполнения лабораторной работы был развернут брокер с доступом через порты 5672 (AMQP) и 15672 (панель управления). В gateway настроено подключение к RabbitMQ с логикой автоматического повторного подключения(startRabbitMQConsumerWithRetry) при ошибке подключения. Создана и прослушивается очередь user_logged_in. После входа пользователя в users-service, сообщение с событием отправляется в очередь. Gateway подписан на эту очередь и асинхронно обрабатывает события входа пользователей, что

демонстрирует использование RabbitMQ для передачи событий между микросервисами. В итоге научились работать с RabbitMQ