САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашнее задание №5

Выполнил:

Соловьева П.А.

Группа К3344

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

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

Ход работы

1. Обновление docker-compose

```
version: '3.9'

services:
 postgres:
   image: postgres:latest
   container_name: postgres
   restart: always
   environment:
     POSTGRES_USER: user
     POSTGRES_PASSWORD: password
   ports:
     - "5432:5432"
   volumes:
     - postgres_data:/var/lib/postgresql/data
     - ./init.sql:/docker-entrypoint-initdb.d/init.sql
   networks:
     - backend

 rabbitmq:
   image: rabbitmq:3-management
   container_name: rabbitmq
   restart: always
   environment:
     RABBITMQ_DEFAULT_USER: user
     RABBITMQ_DEFAULT_PASS: password
   ports:
     - "5672:5672"   # порт для приложений
     - "15672:15672" # web-интерфейс управления
   networks:
     - backend

 users-service:
   build:
     context: ./users_service
   container_name: users-service
   restart: always
   environment:
     DATABASE_URL: postgres://user:password@postgres:5432/users
     PORT: 5001
     RABBITMQ_URL: amqp://user:password@rabbitmq:5672
   ports:
     - "5001:5001"
   depends_on:
     - postgres
     - rabbitmq
   networks:
     - backend

 property-service:
   build:
```

```
    context: ./property_service
  container_name: property-service
  restart: always
  environment:
    DATABASE_URL: postgres://user:password@postgres:5432/property
    PORT: 5002
    RABBITMQ_URL: amqp://user:password@rabbitmq:5672
    USERS_SERVICE_URL: "http://users-service:5001/api/users"
  ports:
    - "5002:5002"
  depends_on:
    - postgres
    - rabbitmq
  networks:
    - backend

 rental-service:
  build:
    context: ./rental_service
  container_name: rental-service
  restart: always
  environment:
    DATABASE_URL: postgres://user:password@postgres:5432/rental
    PORT: 5003
    RABBITMQ_URL: amqp://user:password@rabbitmq:5672
    USERS_SERVICE_URL: "http://users-service:5001/api/users"
    PROPERTY_SERVICE_URL: "http://property-service:5002/api/properties"
  ports:
    - "5003:5003"
  depends_on:
    - postgres
    - property-service
    - rabbitmq
  networks:

    - backend



networks:

 backend:

  driver: bridge



volumes:

 postgres_data:
```

2. Создание события rental_created

rental.controller:

```
import { Request, Response } from "express";
import { RentalService } from "../services/rental.service";
```

```
import { publishToQueue } from "../config/rabbit";

export class RentalController {
 static async create(req: Request, res: Response) {
   try {
     const { propertyId, tenantId, ...rest } = req.body;

     const saved = await RentalService.create({
       ...rest,
       property_id: propertyId,
       tenant_id: tenantId,
     });

     if (saved.status === "accepted") {
       await publishToQueue("rental_events", {
         type: "rental_created",
         payload: {
           id: saved.id,
           propertyId,
           renterId: tenantId,
           startDate: saved.start_date,
           endDate: saved.end_date,
         },
       });
     }

     res.status(201).json(saved);
   } catch (err: any) {
     res.status(500).json({ error: err.message });
   }
 }

 static async update(req: Request, res: Response) {
   try {
     const { propertyId, tenantId, ...rest } = req.body;

     const updated = await RentalService.update(req.params.id, {
       ...rest,
       property_id: propertyId,
       tenant_id: tenantId,
     });

     if (!updated) return res.status(404).json({ message: "Rental not found"
});

     if (updated.status === "accepted") {
       await publishToQueue("rental_events", {
         type: "rental_created",
         payload: {
           id: updated.id,
           propertyId,
           renterId: tenantId,
           startDate: updated.start_date,
           endDate: updated.end_date,
         },
       });
     }

     if (updated.status === "cancelled") {
       await publishToQueue("rental_events", {
         type: "rental_cancelled",
```

```
        payload: {
          rentalId: updated.id,
          propertyId,
          renterId: tenantId,
        },
      });
    }

    res.json(updated);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
}

static async delete(req: Request, res: Response) {
  try {
    const rental = await RentalService.findById(req.params.id);
    const result = await RentalService.delete(req.params.id);

    if (rental) {
      await publishToQueue("rental_events", {
        type: "rental_cancelled",
        payload: {
          rentalId: rental.id,
          propertyId: rental.property_id,
          renterId: rental.tenant_id,
        },
      });
    }

    res.json(result);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
}

  static async findAll(req: Request, res: Response) {
  try {
    const { skip = 0, take = 20 } = req.query as any;
    const rentals = await RentalService.findAll(Number(skip),
Number(take));
    res.json(rentals);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
}

static async findById(req: Request, res: Response) {
  try {
    const rental = await RentalService.findById(req.params.id);
    if (!rental) return res.status(404).json({ message: "Rental not found"
});
    res.json(rental);
  } catch (err: any) {
    res.status(500).json({ error: err.message });
  }
}

}
```

rental.subscriber:

```
import { consumeQueue } from "../config/rabbit";
import { propertyRepository } from "../repositories/property.repository";

consumeQueue("rental_events", async (msg) => {
 if (msg.type === "rental_created") {
   const { propertyId, startDate, endDate } = msg.payload;

   await propertyRepository.update({ id: propertyId }, {
     available_from: endDate,
   });

   console.log(`✅ Property ${propertyId} marked as busy until ${endDate}`);
 }
});
```

Вывод

В ходе выполнения задачи была реализована система межсервисного взаимодействия с использованием RabbitMQ. Для этого в docker-compose.yml был добавлен сервис RabbitMQ, обеспечивающий обмен сообщениями между микросервисами.

Были реализованы события на уровне микросервисов, например rental_created и rental_cancelled, которые публикуются при создании, обновлении или удалении аренды. Другие сервисы, такие как property-service, подписываются на эти события и выполняют необходимые действия (например, обновляют статус доступности недвижимости).

Использование RabbitMQ позволило реализовать асинхронное взаимодействие, снизить связность между сервисами и повысить масштабируемость системы. В результате архитектура приложения стала более гибкой и готовой к расширению с минимальными изменениями в существующем коде.