

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №5

Выполнил:

Корчагин Вадим

Группа
К3341

Проверил:

Добряков Д. И.

Санкт-Петербург

2025 г.

Тема

Платформа для фитнес-тренировок и здоровья.

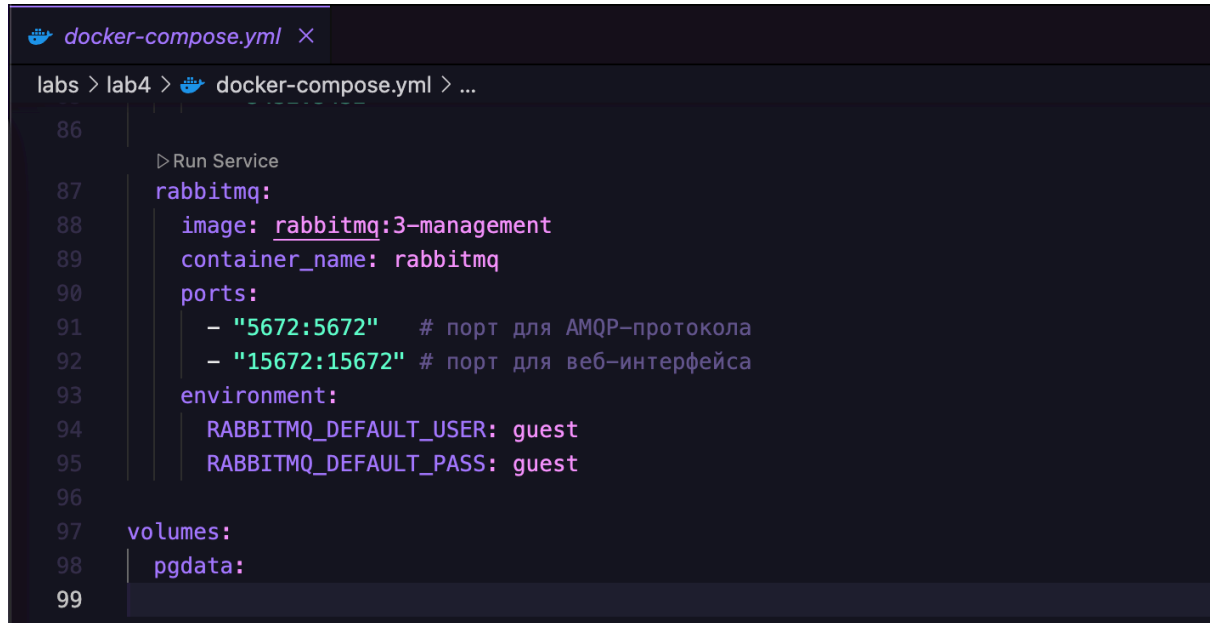
Задача

- реализовать Dockerfile для каждого сервиса;
- написать общий docker-compose.yml;
- настроить сетевое взаимодействие между сервисами.

Ход работы

Настройка RabbitMQ

Добавлен контейнер rabbitmq в docker-compose.yml с UI-панелью на порту 15672 и AMQP-протоколом на 5672.



```
labs > lab4 > docker-compose.yml > ...  
86  
87     > Run Service  
87     rabbitmq:  
88         image: rabbitmq:3-management  
89         container_name: rabbitmq  
90         ports:  
91             - "5672:5672"    # порт для AMQP-протокола  
92             - "15672:15672" # порт для веб-интерфейса  
93         environment:  
94             RABBITMQ_DEFAULT_USER: guest  
95             RABBITMQ_DEFAULT_PASS: guest  
96  
97     volumes:  
98         pgdata:  
99
```

Реализация отправки событий

Для демонстрации работы асинхронного обмена данными между микросервисами был реализован конкретный use-case:

После создания заказа в order-service, он публикует сообщение в очередь order_created. Сервис progress-service подписан на эту очередь и, при получении события, создает запись в UserProgress.

```
TS rabbitmq.ts •
labs > lab3 > src > microservices > order-service > src > common > TS rabbitmq.ts > [?] connectToRabbitMQ
You, 42 minutes ago | 1 author (You)
1  import amqp from "amqplib";
2
3  let channel: amqp.Channel;
4
5  export const connectToRabbitMQ = async () => {
6    const connection = await amqp.connect("amqp://rabbitmq:5672");
7    channel = await connection.createChannel();
8    console.log(`[order] Connected to RabbitMQ`);
9  };
10
11 export const sendOrderCreated = async (order: { userId: string }) => {
12   const queue = "order_created";
13   await channel.assertQueue(queue, { durable: false });
14   channel.sendToQueue(queue, Buffer.from(JSON.stringify(order)));
15 };
```

```
TS orderService.ts ×
labs > lab3 > src > microservices > order-service > src > services > TS orderService.ts > ...
11  export class OrderService extends BaseService<Order> {
34    private async getUser(userId: number) {
36      return data;
37    }
38
39    async createAndSend(data: Partial<Order>) {
40      const createdOrder = await this.create(data);
41
42      await sendOrderCreated({
43        userId: String(createdOrder.user_id)
44      });
45
46      return createdOrder;
47    }
48  }
```

index.ts

labs > lab3 > src > microservices > order-service > src > index.ts > ...

```
13  const app = express();
14  const host = process.env.HOST;
15  const port = parseInt(process.env.PORT);
16
17  async function bootstrap() {
18    try {
19      await OrderDataSource.initialize();
20      console.log("Order DB connected");
21
22      await connectToRabbitMQ();
23      console.log("RabbitMQ connected");
24
```

Реализация приемки сообщений

progress-service принимает сообщение `order_created` и обрабатывает его — это ключевая часть взаимодействия через RabbitMQ.

```

rabbitmq.ts
labs > lab3 > src > microservices > progress-service > src > common > rabbitmq.ts > connectToRabbitMQ
You, 16 minutes ago | 1 author (You)
1  import amqp from "amqplib";
2  import { UserProgressService } from "../services/userProgressService";
3
4  let channel: amqp.Channel;
5
6  export const connectToRabbitMQ = async () => {
7    const connection = await amqp.connect("amqp://rabbitmq:5672");
8    channel = await connection.createChannel();
9    console.log("✅ [progress] Connected to RabbitMQ");
10
11    const queue = "order_created";
12    await channel.assertQueue(queue, { durable: false });
13
14    channel.consume(queue, async (msg) => {
15      if (msg) {
16        const content = JSON.parse(msg.content.toString());
17        const progressService = new UserProgressService();
18        await progressService.createFromOrder(content);
19        channel.ack(msg);
20      }
21    });
22  };

```

```

userProgressService.ts
labs > lab3 > src > microservices > progress-service > src > services > userProgressService.ts > UserProgressService
10  export class UserProgressService extends BaseService<UserProgress> {
40  }
41  async createFromOrder(data: { userId: string }) {
42    const progress = await this.create({
43      user_id: parseInt(data.userId),
44    });
45    console.log("📦 Progress created from order:", progress);
46  }
47  }
48

```

```

index.ts
labs > lab3 > src > microservices > progress-service > src > index.ts > bootstrap
12
13  const app = express();
14  const host = process.env.HOST;
15  const port = parseInt(process.env.PORT);
16
17  async function bootstrap() {
18    try {
19      await ProgressDataSource.initialize();
20      console.log("Progress DB connected");
21
22      await connectToRabbitMQ();
23      console.log("RabbitMQ connected");

```

Пример работы

Создаем пользователя, с которым будем взаимодействовать:

Request URL

http://localhost:4000/users/

Server response

Code	Details
200	<div>Response body</div> <pre>{ "role_id": 1, "name": "string", "email": "user@example.com", "password_hash": "\$2b\$08\$MaXLw51dhRcwoBZvVyvCxeI14CXlDMkWPnri.lKrKFVc8uPtZjMWK", "date_of_birth": "2003-01-01", "gender": "string", "id": 1, "created_at": "2025-05-20T18:52:04.191Z", "updated_at": "2025-05-20T18:52:04.191Z" }</pre>

Создаем заказ по нашему специальному пути “/orders/rabbit”:

http://localhost:4003/orders/rabbit

Server response

Code	Details
200	<div>Response body</div> <pre>{ "user_id": 1, "total_amount": 0, "currency": "string", "id": 1, "created_at": "2025-05-20T18:52:28.791Z", "updated_at": "2025-05-20T18:52:28.791Z" }</pre>

И можем увидеть в консоли, что сработал триггер на создание UserProgress записи:

```
rabbitmq | 2025-05-20 18:51:16.991123+00:00 [notice] <0.86.0> alarm_handler: {set,{system_memory_high_watermark,[]}}
}
progress-service | Progress created from order: UserProgress {
progress-service |   user_id: 1,
progress-service |   current_weight: null,
progress-service |   target_weight: null,
progress-service |   steps_walked: null,
progress-service |   water_intake: null,
progress-service |   id: 1,
progress-service |   updated_at: 2025-05-20T18:52:28.900Z
progress-service | }
```

И по запросу получаем нашу созданную запись UserProgress:

http://localhost:4002/user-progress/

Server response

Code	Details
200	<div>Response body</div> <pre>{ "id": 1, "user_id": 1, "current_weight": null, "target_weight": null, "steps_walked": null, "water_intake": null, "updated_at": "2025-05-20T18:52:28.900Z", "user": { "id": 1, "role_id": 1, "name": "string", "email": "user@example.com", "password_hash": "\$2b\$08\$MaXLw51dhRcwoBZvVyvCxeI14CXlDMkWpNri.lKrKFVc8uPtZjMWK", "date_of_birth": "2003-01-01", "gender": "string", "created_at": "2025-05-20T18:52:04.191Z", "updated_at": "2025-05-20T18:52:04.191Z", "role": { "id": 1, "role_name": "string", "created_at": "2025-05-20T18:51:50.444Z", "updated_at": "2025-05-20T18:51:50.444Z" } } }</pre>

Выводы

В рамках данной работы был успешно реализован сценарий взаимодействия между микросервисами:

- order-service публикует событие order_created при оформлении заказа,
- progress-service подписан на соответствующую очередь и создает запись о привязке пользователя к прогрессу пользователя.

В результате, сервисы стали слабо связаны, взаимодействуют через асинхронную очередь сообщений, что повышает отказоустойчивость и расширяемость архитектуры проекта.