

# Отчет по большому домашнему заданию №3 по курсу «Конструирование программного обеспечения»

Каспари Давид Эрвинович.

В этом отчете я отражу результаты своей работы, опишу архитектуру написанной программы, приведу инструкции.

## 1. Функционал.

PaymentsService содержит функции создания, пополнения, просмотра баланса счета. OrdersService занимается обработкой заказа. ApiGateway отвечает за маршрутизацию.

## 2. Микросервисная архитектура.

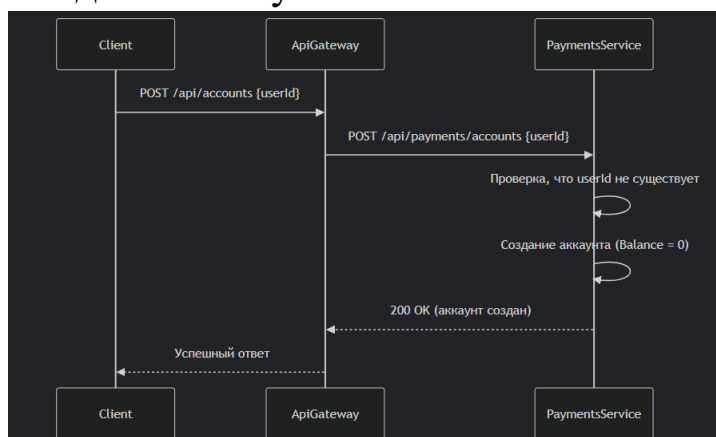
- 1) API Gateway – отвечает только за routing запросов
- 2) Orders Service – отвечает за создание заказа и обработку
- 3) Payments Service – отвечает за создание, пополнение и просмотра баланса счета.

## 3. Архитектура программы.

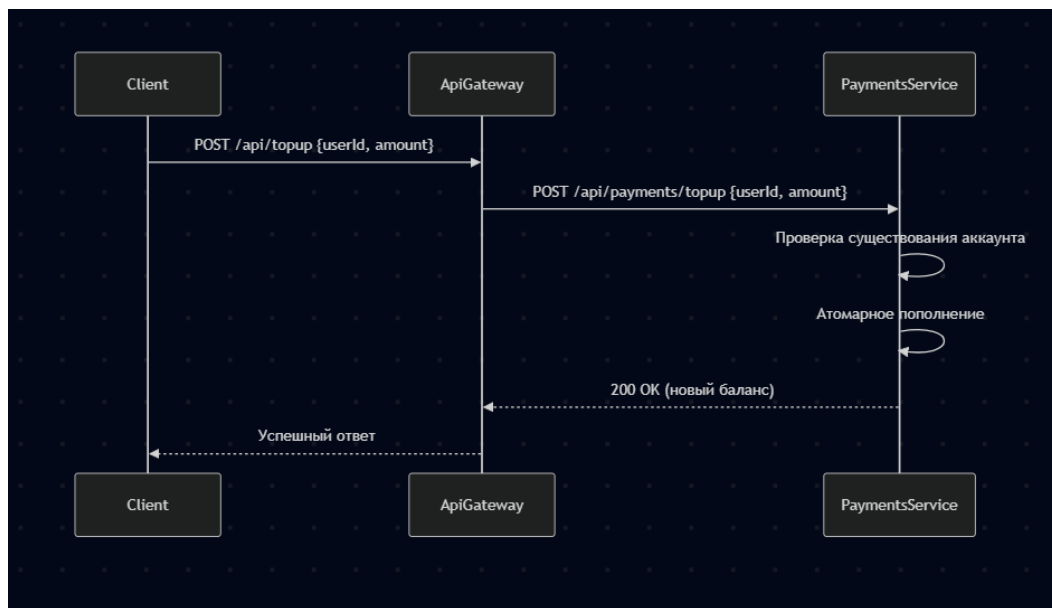
В Visual Studio созданы Веб-API Asp.Net.Core с названиями ApiGateway, OrdersService и PaymentsService.

Приведем некоторые схемы работы:

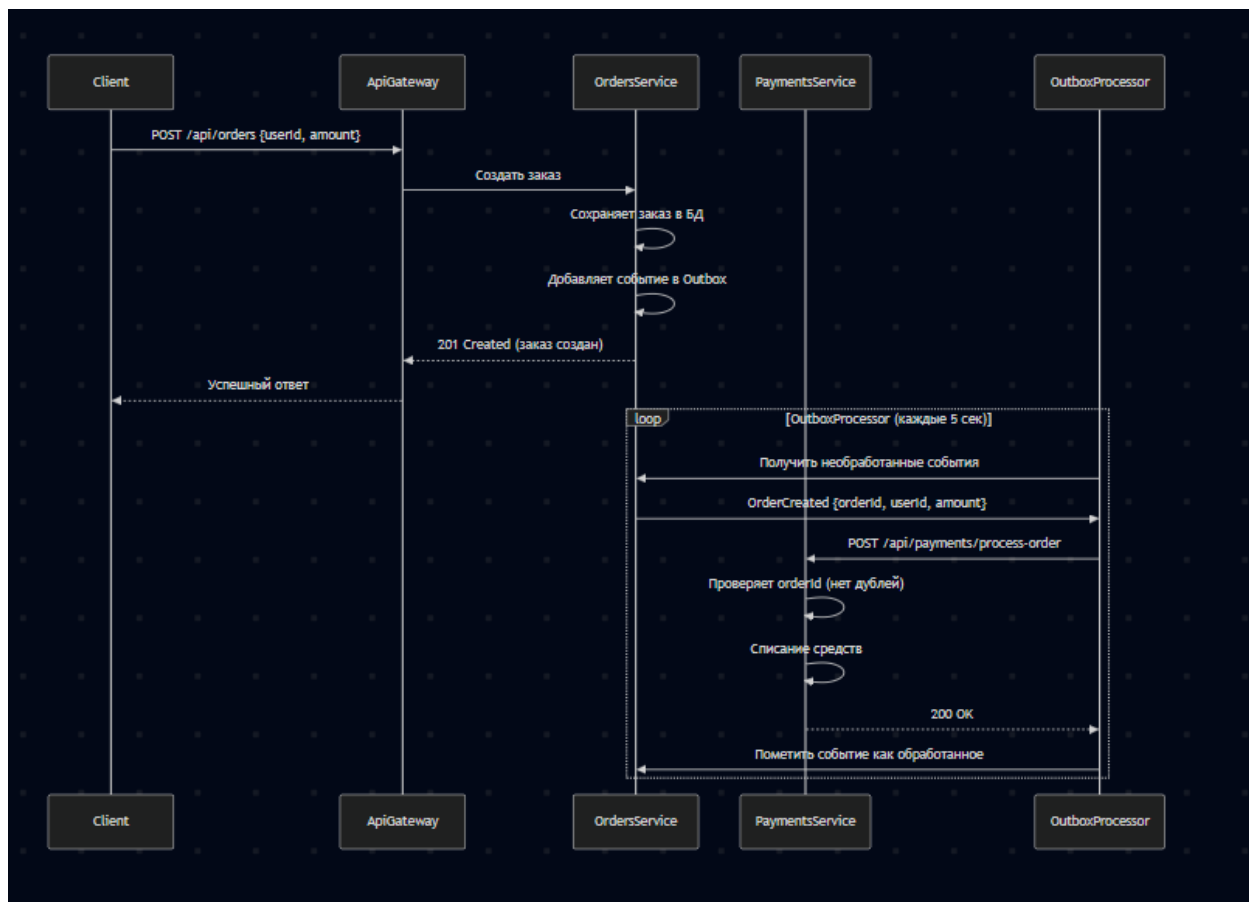
Создание аккаунта:



## Пополнение баланса.



## Процесс обработки заказа.



С конкретной реализацией алгоритма работы можно ознакомиться в файлах программы с помощью комментариев.

## 4. Пример работы программы.

Приведем несколько примеров работы нашей программы.

Собираем ее с помощью Docker. После docker-compose build видим, что программа собирается.

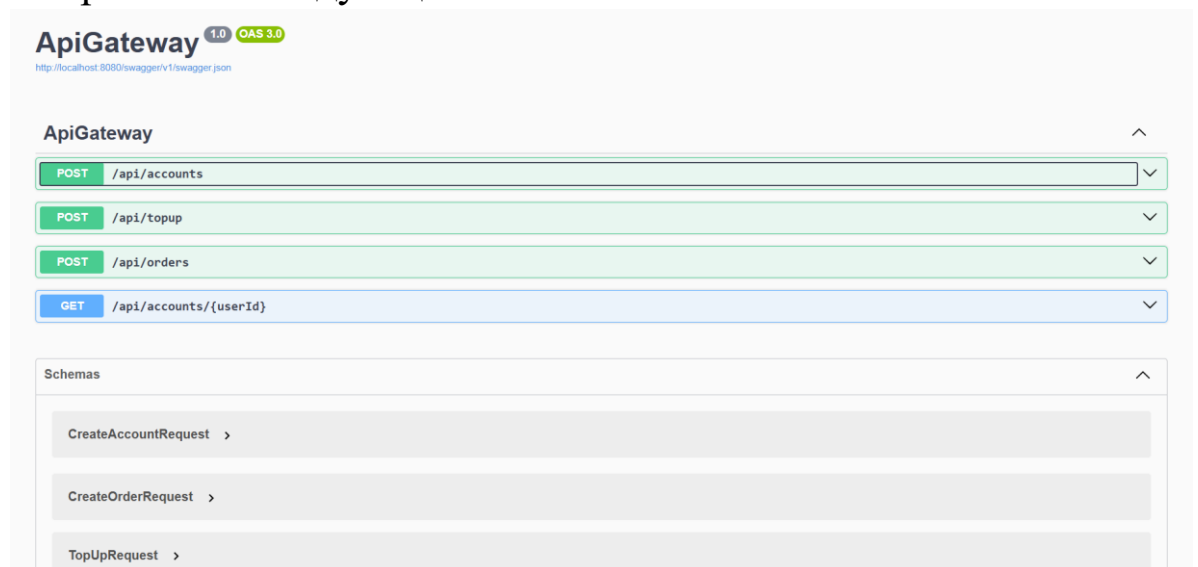
```
C:\Users\david\source\repos\InternetShop> docker-compose build
[+] Building 3/3
=> [api-gateway] exporting to image 1.8s
=> => exporting layers 1.5s
=> => exporting manifest sha256:710689e14a187128e2dac9f1cc3308f9366d10ecee254b326a4dec139fc4f291 0.0s
=> => exporting config sha256:f53e0a7993673ea5905ea12d39c74a8ba25bbb52dc473774f292b93140636ea9 0.0s
=> => exporting attestation manifest sha256:86a8d322a8d9e03c41ebca5e378ef08c97cbbb4a3f8a67fb3e501e84a95b5dde 0.0s
=> => exporting manifest list sha256:4a253eba23137a4ffa5e8fa112c01f6282de8f29a63b796410f1605b7df0c907 0.0s
=> => naming to docker.io/library/internetshop-api-gateway:latest 0.0s
=> => unpacking to docker.io/library/internetshop-api-gateway:latest 0.2s
=> [api-gateway] resolving provenance for metadata file 0.0s
[+] Building 3/3
✓api-gateway Built 0.0s
✓orders-service Built 0.0s
✓payments-service Built 0.0s
C:\Users\david\source\repos\InternetShop>
```

Запускаем с помощью docker-compose up.

Здесь можно протестировать: <http://localhost:8080/swagger>

```
C:\Users\david\source\repos\InternetShop> docker-compose up
time="2025-06-14T02:21:08+03:00" level=warning msg="C:\\Users\\david\\source\\repos\\InternetShop\\docker-compose.yml: t
he attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 3/3
✓Container internetshop-payments-service-1 Recreated 0.3s
✓Container internetshop-orders-service-1 Recreated 0.3s
✓Container internetshop-api-gateway-1 Recreated 0.2s
Attaching to api-gateway-1, orders-service-1, payments-service-1
payments-service-1 | warn: Microsoft.AspNetCore.Hosting.Diagnostics[15]
payments-service-1 | Overriding HTTP_PORTS '8080' and HTTPS_PORTS ''. Binding to values defined by URLs instead '
```

Открывается следующее окно:



Введем первого user'а.

Request body required

```
{
  "userId": "David"
}
```

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/accounts' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "userId": "David"
  }'
```

Request URL

```
http://localhost:8080/api/accounts
```

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{   "userId": "David",   "balance": 0,   "hasActiveOrder": false }</pre></div><div>Download</div></div> <div><div>Response headers</div><div><pre>content-type: application/json; charset=utf-8 date: Fri, 13 Jun 2025 23:21:38 GMT server: Kestrel transfer-encoding: chunked</pre></div></div>

Responses

Code	Description	Links
200	OK	No links

Видим, что пользователь зарегистрировался.

Далее сделаем пополнение на определенную сумму.

POST /api/topup

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{
  "userId": "David",
  "amount": 500
}
```

Execute

Clear

Видим, что аккаунт пополнен.

The screenshot shows a REST client interface with a blue 'Execute' button and a 'Clear' button. Below the buttons, the 'Responses' section is active. It displays the following information:

- Curl:**

```
curl -X 'POST' \
  'http://localhost:8080/api/topup' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "userId": "David",
    "amount": 500
  }'
```
- Request URL:** `http://localhost:8080/api/topup`
- Server response:**
  - Code:** 200
  - Details:** Response body
  - Response body:** `{"userId":"David","balance":500,"hasActiveOrder":false}"`

Далее формируем заказ на сумму 300.

The screenshot shows a REST client interface for a POST request to `/api/orders`. The interface includes the following sections:

- Method and Path:** POST `/api/orders`
- Parameters:** No parameters
- Request body:** required
- Request body content:**

```
{
  "userId": "David",
  "amount": 300
}
```

Видим, что заказ в процессе.

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/orders' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
    "userId": "David",
    "amount": 300
  }'
```

Request URL

http://localhost:8080/api/orders

Server response

Code	Details
200	<p>Response body</p> <pre>"{\\"orderId\\":\\"6c9fa97b-50d2-45b7-a5cc-6fe907d61dfc\\",\\"userId\\":\\"David\\",\\"amount\\":300,\\"isPaid\\":false,\\"isProcessing\\":true}"</pre>

Далее спрашиваем, сколько осталось денег.

GET /api/accounts/{userId}

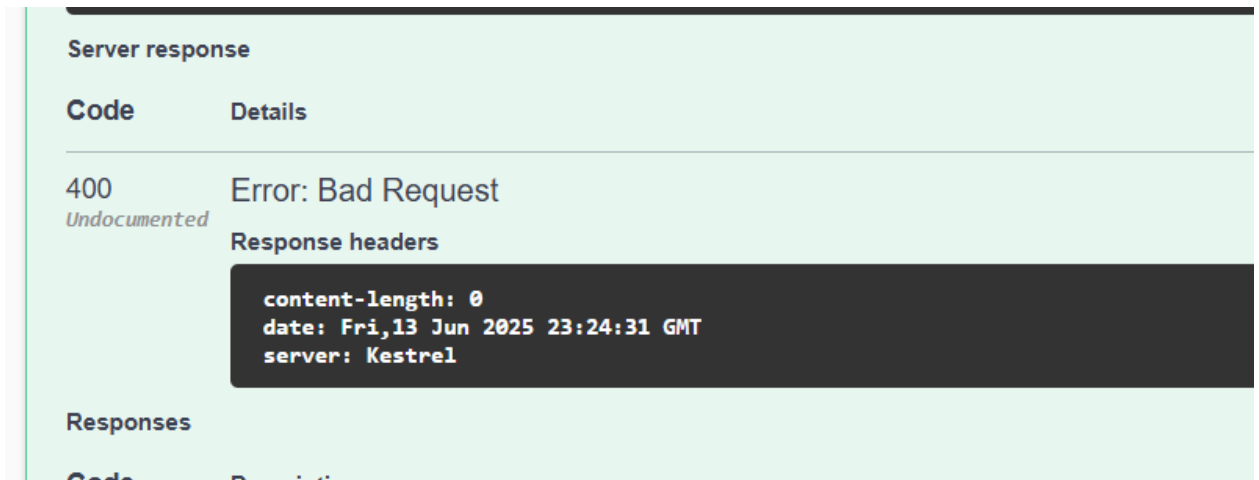
Parameters

Name	Description
<b>userId</b> * required string (path)	<input type="text" value="David"/>

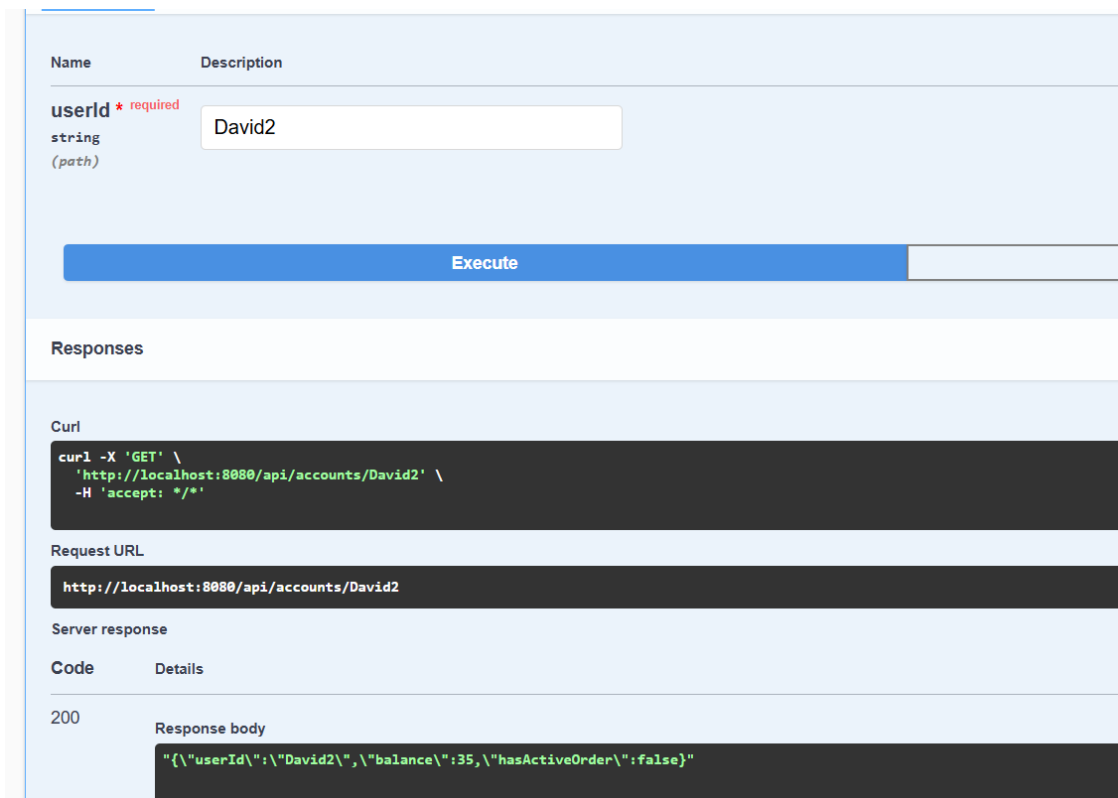
Видим, что 300, то есть заказ выполнен.

Code	Details
200	<p>Response body</p> <pre>"{\\"userId\\":\\"David\\",\\"balance\\":200,\\"hasActiveOrder\\":false}"</pre> <p>Response headers</p>

Если попытаться потратить больше, чем есть на аккаунте, то заказ не выполнится. Если создать аккаунт с таким же Id, то будет ошибка.



Создадим второго пользователя и потратим сумму 665. Видим, что осталось 35 при балансе 700.





## 5. Выполнение требований к заданию.

At most once – в PaymentsService если запрос приходит повторно, платеж не выполняется. Используется ConcurrentDictionary для проверки заказов.

Создание заказа работает асинхронно, процесс оплаты происходит через Outbox.

Реализованы требования к функционалу, есть четкое разделение на сервисы (описано выше).

В OrdersService используется Transactional Outbox. При создании заказа идет сообщение в Outbox. Подробнее в программе.

Swagger демонстрирует функциональность, работа с Docker корректна и описана выше.