

Пользовательские сценарии

1. Регистрация нового пользователя

Есть неавторизованный пользователь

И есть страница регистрации платежной системы ВродеДеньги

И пользователь вводит параметры

Тогда

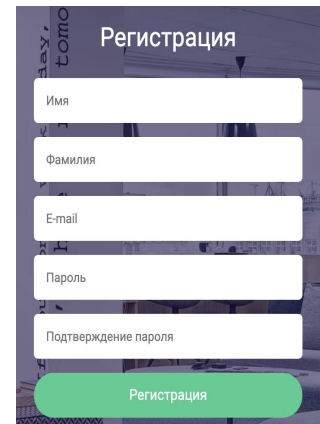
бэкенд проверяет, что такого пользователя нет (признак уникальности — email)

И

регистрирует нового пользователя, перебрасывая на страницу ввода логина/пароля

И

создает счет по умолчанию в рублях для этого нового пользователя



The screenshot shows a registration form with the following fields: Имя (Name), Фамилия (Surname), E-mail, Пароль (Password), and Подтверждение пароля (Confirm password). A green button labeled 'Регистрация' (Registration) is at the bottom.

2. Вход существующего пользователя в систему

Есть неавторизованный пользователь

И есть страница логина системы ВродеДеньги

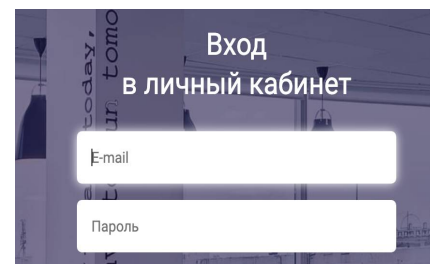
И пользователь вводит логин и пароль

Тогда

происходит авторизация

И

система возвращает JWT токен



The screenshot shows a login form with the following fields: E-mail and Пароль (Password).

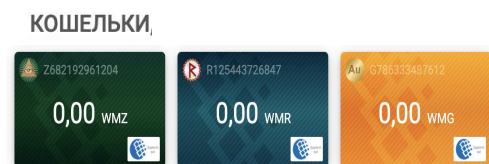
3. Просмотр списка электронных кошельков

Есть авторизованный пользователь

И есть страница баланса системы ВродеДеньги

Тогда

отображается список счетов, и остаток на каждом счете



4. Поиск нужной услуги в Каталоги услуг - для дальнейшей оплаты

Есть авторизованный пользователь

И есть страница Каталог услуг

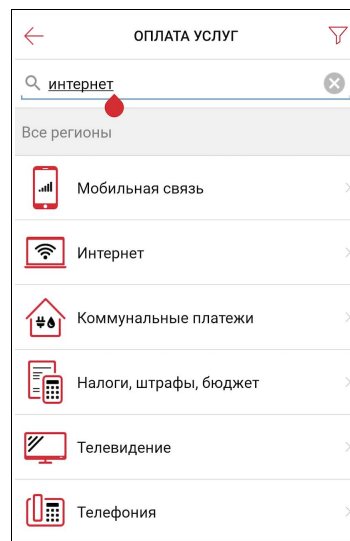
И

пользователь ищет услугу 1) либо в строке поиска услуг

2) либо проваливаясь по категориям в каталоге услуг (древесная структура)

Тогда

Отображаются найденные услуги (по строке поиска или в каталоге)



5. Оплата за выбранную Услугу

Есть авторизованный пользователь

И есть страница Оплаты за услугу, в которой выбрана одна из услуг, например, оплата за мобильный телефон

И пользователь ввел платежные реквизиты (номер телефона, счет которого нужно пополнить)

Тогда

5.1. Система создает Платежное поручение и сразу же сообщает клиенту, что платеж принят в обработку

И система параллельно (не блокируя дальнейшие действия пользователя), выполняет оплату за услугу, а именно:

5.2. Система проверяет остаток на счету (авторизует транзакцию - достаточно ли денег)

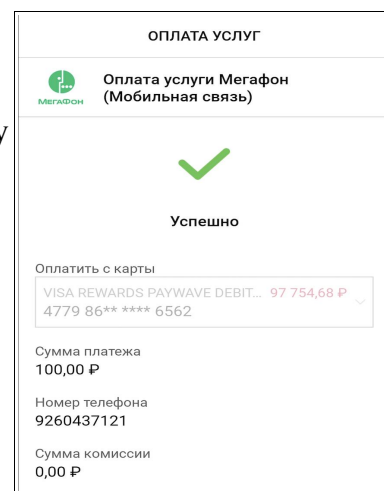
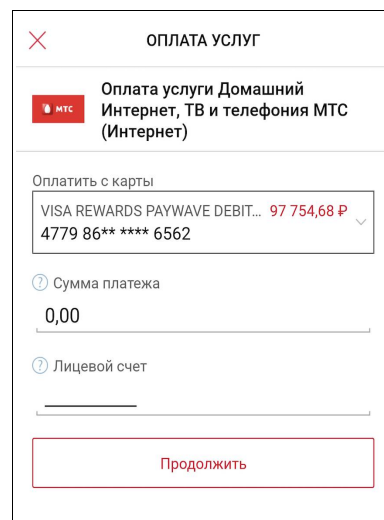
5.3. Списывает со счета клиента

5.4. Переводит на счет Вендора (организации, оказывающей услугу)

5.5. Система фиксирует оплату в Истории, которую клиент может посмотреть позже

И

Пользователю отображается экран Оплата принята (сразу после шага 5.1)



6. Просмотри истории платежей

Есть авторизованный пользователь















И пользователь открывает страницу История платежей

Тогда

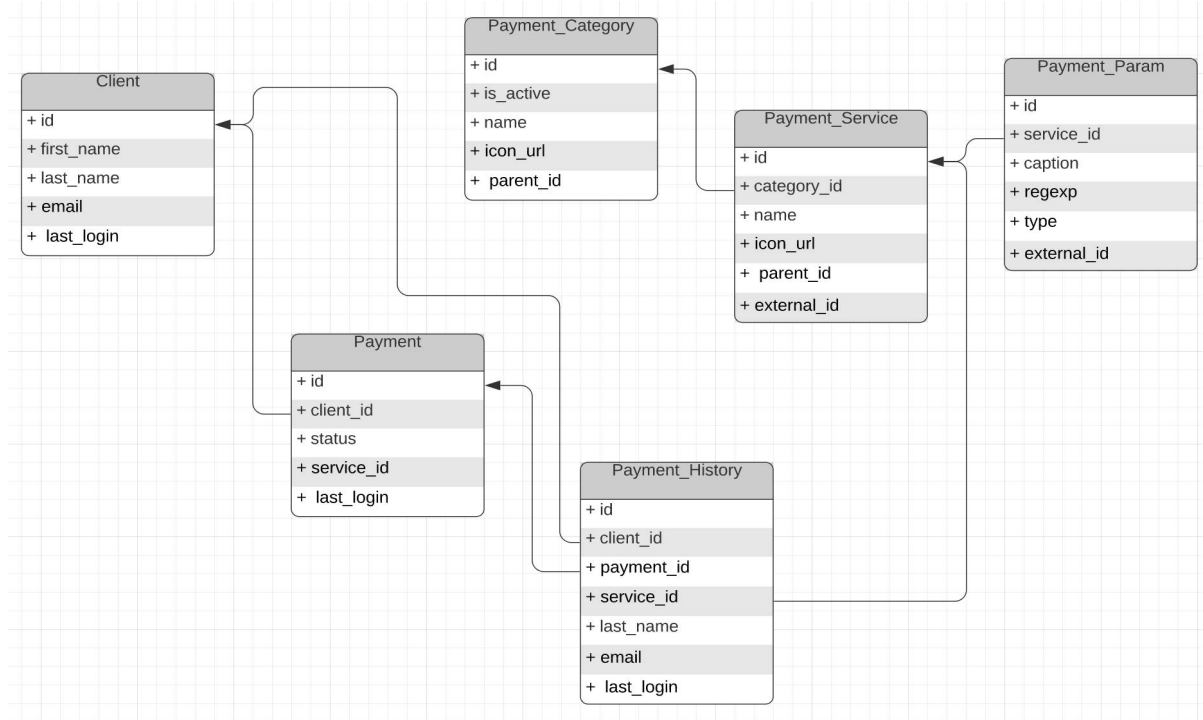
Отображаются не более 10 последних записей о проведенных платежах, отсортированных по дате проведения в убывающем порядке.

Рядом с суммой отображаются статусы платежей, возможные статусы:

- в обработке (processing)
- проведен (done)
- ошибка (error)

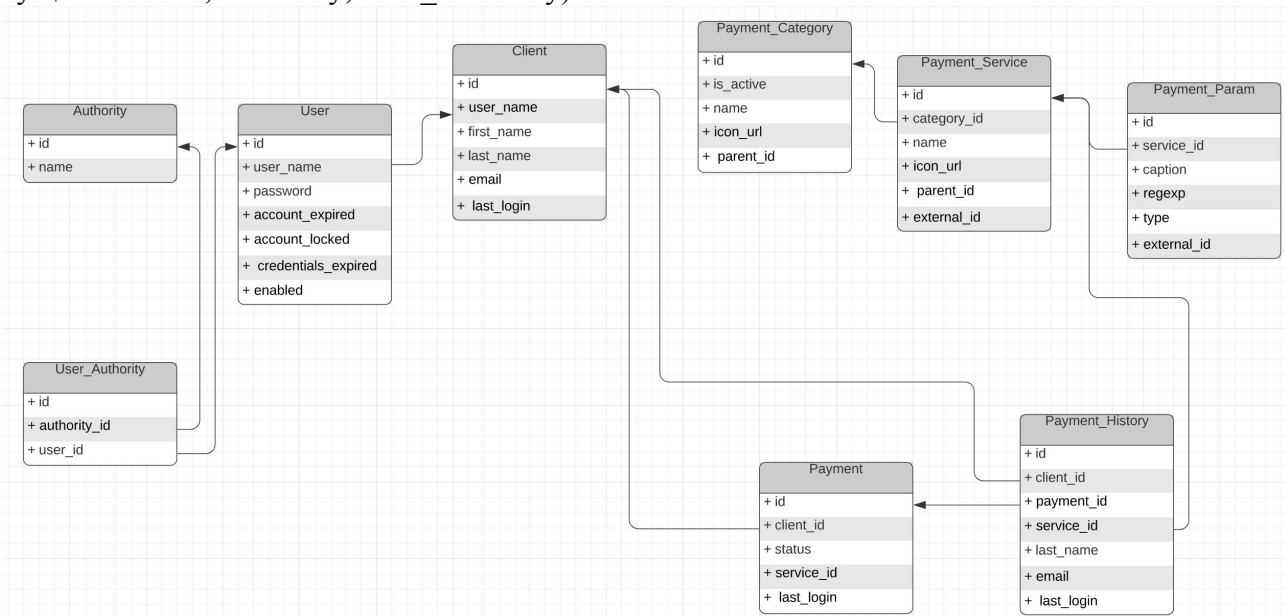
	ГорЗдрав Аптеки	– 542,00 Р	
	Верный Супермаркеты	– 524,87 Р	
	Пятерочка Супермаркеты	– 132,98 Р	
	Перекресток Супермаркеты	– 735,84 Р	
	ЛУКОЙЛ Топливо	– 288,00 Р	
	Роснефть Топливо	– 150,00 Р	
	AZS N33019 Топливо	– 2 403,58 Р	

Итерация 0 модели предметной области

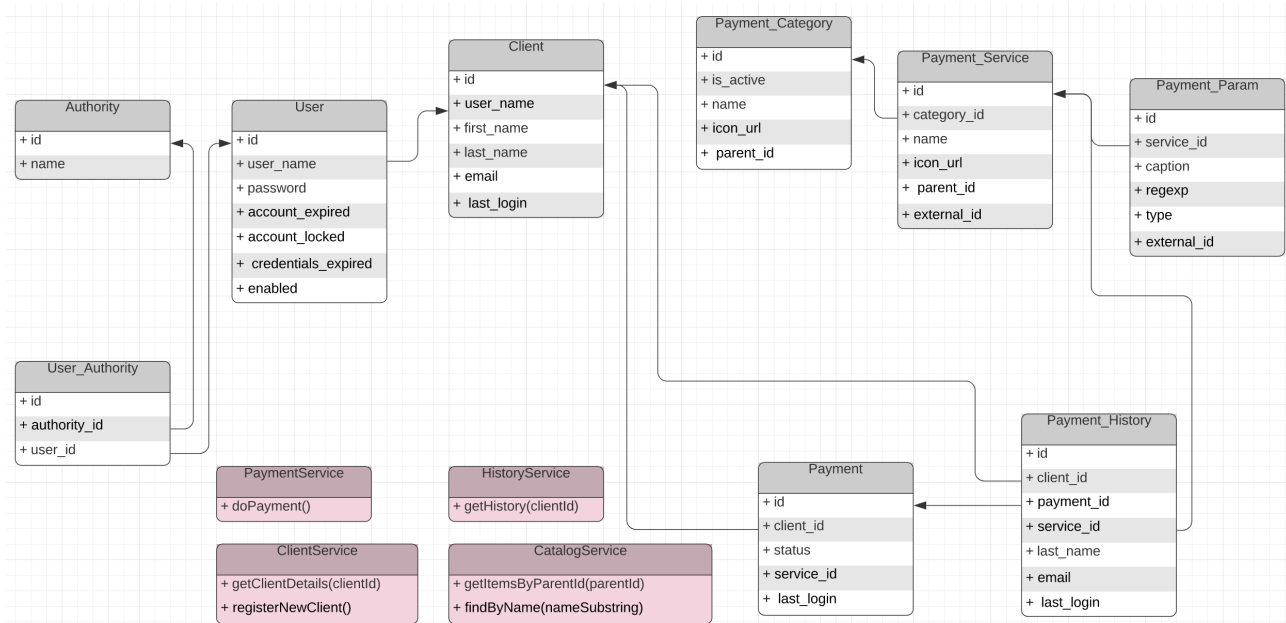


Итерация 1 модели предметной области, с уточнением прав доступа

Учитывая опыт, полученный на предыдущих занятиях, добавим сущности, требуемые для хранения паролей, прав доступа (authority) в реализации на Java Spring Boot (а именно, сущности User, Authority, User_Authority):

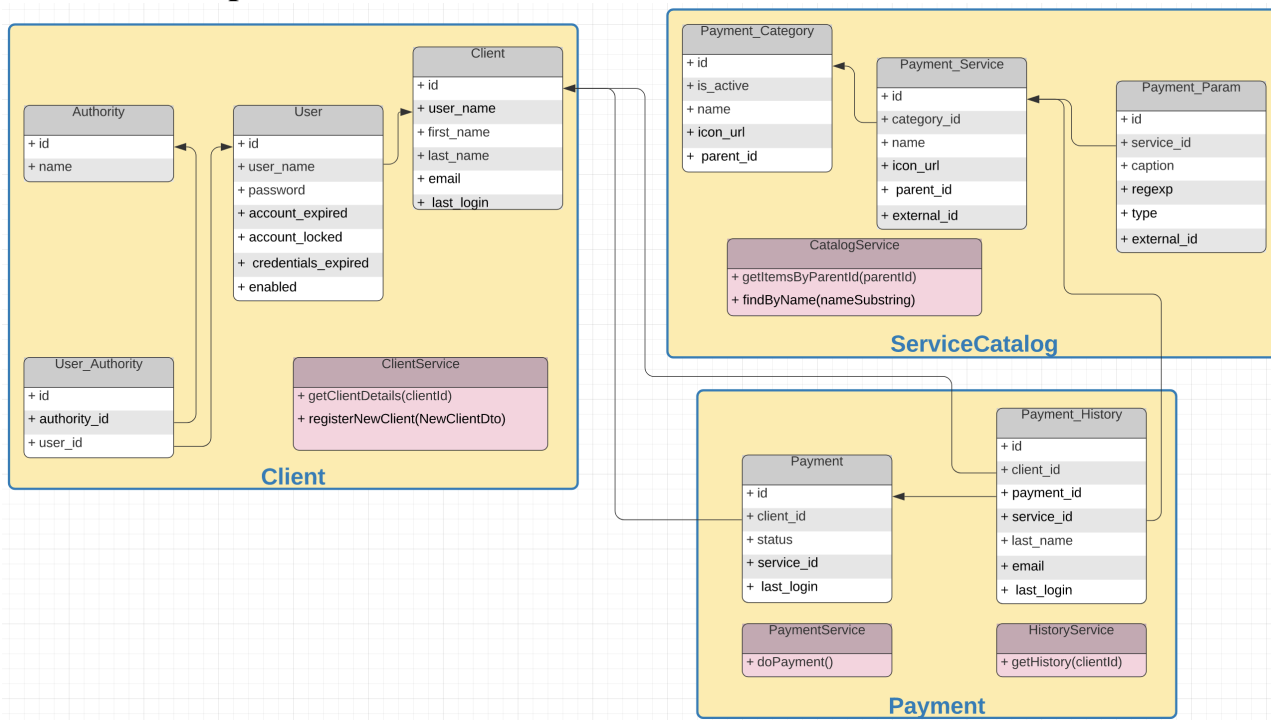


Итерация 2 Модели предметной области, добавим в сущности еще системных действий



Разбиение на сервисы

Разбиение Вариант 1



Выделим сервисы вокруг агрегатов

- Client — управляет информацией о сущности Клиент
- ServiceCatalog — вся информация о каталоге, сервисах, платежных параметрах
- Payment — вся информация о Платеже и Истории Платежей

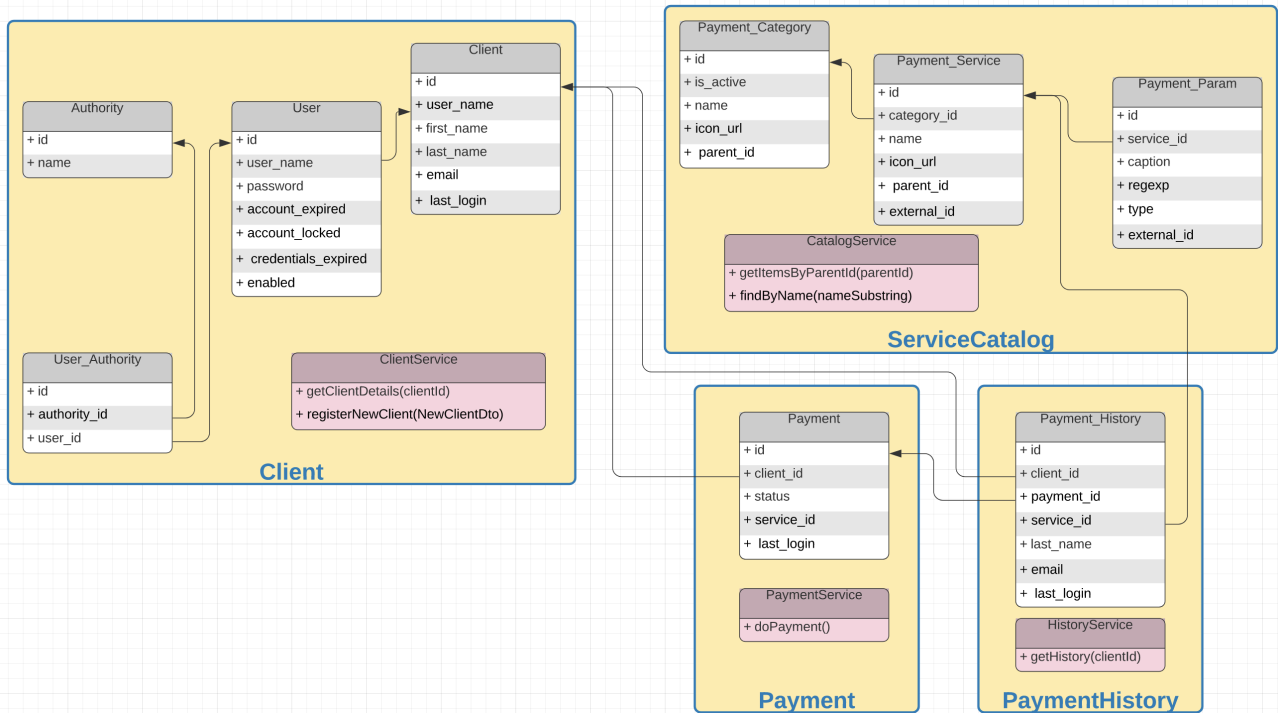
Разбиение Вариант 2 — декомпозиция микросервиса Client

Ввиду безопасности, может потребоваться выделить сущности Authority, User, User_Authority в отдельный микросервис. Но поскольку у нас пока таких требований нет, этого делать в данном проекте не будем.

Разбиение Вариант 3 — декомпозиция микросервиса Payment

Разнесем Payment и Payment_History в отдельные микросервисы. Обоснование: разные типы платежей, которые относятся, например, к разным платежным шлюзам, к разным типам платежей, могут тиражироваться. Для этого появятся дополнительные типы Payment, и я предпочитаю делать это тиражирование в отдельными микросервисами. Синхронизация между микросервисами Payment и Payment_History сделаем **асинхронным**.

Итоговое разбиение - примем Вариант 3, и итоговое разбиение выглядит следующим образом



Описание микросервисов

Service Name: Payment Description: проводит платежные операции — оплату за Услуги		
Dependencies	Implementation	Interface
<u>Service Dependencies</u> -	<u>Data Sources</u> <i>Local PostgreSQL</i>	<u>Queries</u> -
<u>Event Subscriptions</u> -	<u>Logic/Rules</u> -	<u>Commands</u> <ul style="list-style-type: none">• <i>выполнить платеж за услуги</i>
Architecture		<u>Events Published</u> 3 события в 3 топика Kafka: <ul style="list-style-type: none">• Payment Processing Started• Payment Done (Success)• Payment Error
<u>Qualities</u> Дополнительных нефункциональных требований нет.		

Service Name: PaymentHistory Description: содержит историю всех типов Платежей (текущих и будущих)		
Dependencies	Implementation	Interface
<u>Service Dependencies</u> <ul style="list-style-type: none">• <i>Зависит от Payment - ссылается на payment_id</i>• <i>Зависит от ServiceCatalog — ссылается на service_id</i>	<u>Data Sources</u> <i>Local PostgreSQL</i>	<u>Queries</u> <ul style="list-style-type: none">• <i>запрос истории платежей</i>
<u>Event Subscriptions</u> 3 события в 3 топика Kafka: <ul style="list-style-type: none">• <i>Payment Processing Started</i>• <i>Payment Done (Success)</i>• <i>Payment Error</i>	<u>Logic/Rules</u> <ul style="list-style-type: none">• -	<u>Commands</u> <ul style="list-style-type: none">• -
Architecture		<u>Events Published</u> <ul style="list-style-type: none">• -
<u>Qualities</u> Дополнительных нефункциональных требований нет.		

Service Name: Client Description: управляет данными о клиенте, включая профиль и учетные данные (логин-пароль)		
Dependencies	Implementation	Interface
<u>Service Dependencies</u> <ul style="list-style-type: none"> - 	<u>Data Sources</u> <i>Local PostgreSQL</i>	<u>Queries</u> <ul style="list-style-type: none"> <i>Запрос деталей зарегистрированного пользователя (ФИО, почта и т.п.)</i>
<u>Event Subscriptions</u> <ul style="list-style-type: none"> - 	<u>Logic/Rules</u> -	<u>Commands</u> <ul style="list-style-type: none"> <i>Регистрация нового пользователя</i> <i>Авторизация существующего пользователя</i>
Architecture		<u>Events Published</u> <ul style="list-style-type: none"> -
<u>Qualities</u> Дополнительных нефункциональных требований нет.		

Service Name: ServiceCatalog Description: управляет данными о клиенте, включая учетные данные		
Dependencies	Implementation	Interface
<u>Service Dependencies</u> <ul style="list-style-type: none"> - 	<u>Data Sources</u> <i>Загрузка и обновление данных вручную с помощью SQL скриптов</i>	<u>Queries</u> <ul style="list-style-type: none"> Поиск услуг по подстроке - в названии и описании Услуги Выдать Категории и Услуги для заданной родительской Категории
<u>Event Subscriptions</u> <ul style="list-style-type: none"> - 	<u>Logic/Rules</u> <i>Существующие данные никогда не удаляются, т.к. на них есть ссылки из других микросервисов (PaymentHistory)</i>	<u>Commands</u> <ul style="list-style-type: none"> -
Architecture		<u>Events Published</u> <ul style="list-style-type: none"> -
<u>Qualities</u> <i>Дополнительных нефункциональных требований нет.</i>		

Контракты микросервисов

Микросервис Payment

Sync API — REST/HTTP

Метод - выполнить платеж на услугу

POST /payments/api/v1/payment

```
{
  "idempotence_key": UUID,
  "amount": {
    "total": "579.50",
    "currency": "RUB"
  },
  "service": {
    "service_id": UUID,
    "service_params": [ {
      "param_name": "phone_number",
      "param_value": "9123451234"
    } ],
  },
  "description": "The payment transaction description."
}
```

Async API- Kafka

Опубликовать событие о том, что новый платеж принят в обработку

PUB topic: payments/v1/payment/started

Payload:

```
{ "payment_id": UUID,
  "payment_time": "2020-07-11T08:45:57Z" ,
  "service_id": UUID,
  "amount": {
    "total": "579.50",
    "currency": "RUB"
  }
}
```

Опубликовать событие о том, что платеж был успешно выполнен

PUB topic: payments/v1/payment/done

Payload:

```
{ "payment_id": UUID,
  "update_time": "2020-07-11T08:45:57Z"
}
```

Опубликовать событие о том, что платеж не удалось выполнить (ошибка)

PUB topic: payments/v1/payment/error

Payload:

```
{ "payment_id": UUID,
  "update_time": "2020-07-11T08:45:57Z" ,
  "reason": string
}
```

Микросервис PaymentHistory

Sync API — REST/HTTP

GET /api/v1/{clientId}

```
{ [
  "payment_date": "2020-07-11T08:45:57Z" ,
  "payment_status": "Done",
  "amount": {
    "total": "579.50",
    "currency": "RUB"
  },
  "service": {
    "service_name": string,
    "service_category_name": string,
  },
  "description": string
]
```

ASync API - Kafka

Подписка на событие — платеж принят в обработку

SUB topic: payments/v1/payment/started

Payload:

```
{ "payment_id": UUID,
  "payment_time": "2020-07-11T08:45:57Z" ,
  "service_id": UUID,
  "amount": {
    "total": "579.50",
    "currency": "RUB" }
}
```

Подписка на событие — платеж выполнен успешно

SUB topic: payments/v1/payment/done

Payload:

```
{ "payment_id": UUID,
  "update_time": "2020-07-11T08:45:57Z"
}
```

Подписка на событие — платеж завершился ошибкой

SUB topic: payments/v1/payment/error

Payload:

```
{ "payment_id": UUID,
  "update_time": "2020-07-11T08:45:57Z" ,
  "reason": string
}
```

Микросервис Client

Sync API — REST/HTTP

Запрос деталей о залогиненном пользователе

Внимание — информация должна быть доступна только для того пользователя, который авторизован (смотрим по содержимому JWT токена).

REST GET /api/v1/client/{clientId}/details

```
{
  "firstNam": string,
  "lastName": string,
  "email": string
}
```

Регистрация нового пользователя

REST POST /api/v1/client/details

```
{ "userName": string,
  "password": string,
  "email": string,
  "first_name": string,
  "last_name": string
}
```

Авторизация существующего пользователя

HTTP FORM AUTHORIZATION

```
curl --location --request POST 'arch.homework/profile/oauth/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Content-Type: no-cache' \
--header 'Authorization: Basic <Base64-Login-Password> ' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'username=<username>' \
--data-urlencode 'password=<password>'
```

где

- **<Base64-Login-Password>** – системный закодированный в Base64 логин-пароль в виде login-pass.
Пример: Zm9vQ2xpZW50SWRQYXNzd29yZDpzZWNyZXQ=
- **<username>** – username конкретного пользователя
- **<password>** – пароль конкретного пользователя

Микросервис ServiceCatalog

Sync API — REST/HTTP

Поиск услуг по подстроке - в названии и описании Услуги

GET /api/v1/catalog?find_string={searchSubstring}

```
{
  "items": [{
    "id": UUID,
    "type": enum<category|service>,
    "name": string,
    "description": string,
    "icon": string
  }]
}
```

Выдать Категории и Услуги для заданной родительской Категории

GET /api/v1/catalog?parent_id= {parentId}

```
{
  "items": [{
    "id": UUID,
    "type": enum<category|service>,
    "name": string,
    "description": string,
    "icon": string
  }]
}
```

Если parent_id == null, то возвращается один уровень — все элементы каталога для корневого элемента (т. е. parent_id в запросе может полностью отсутствовать).