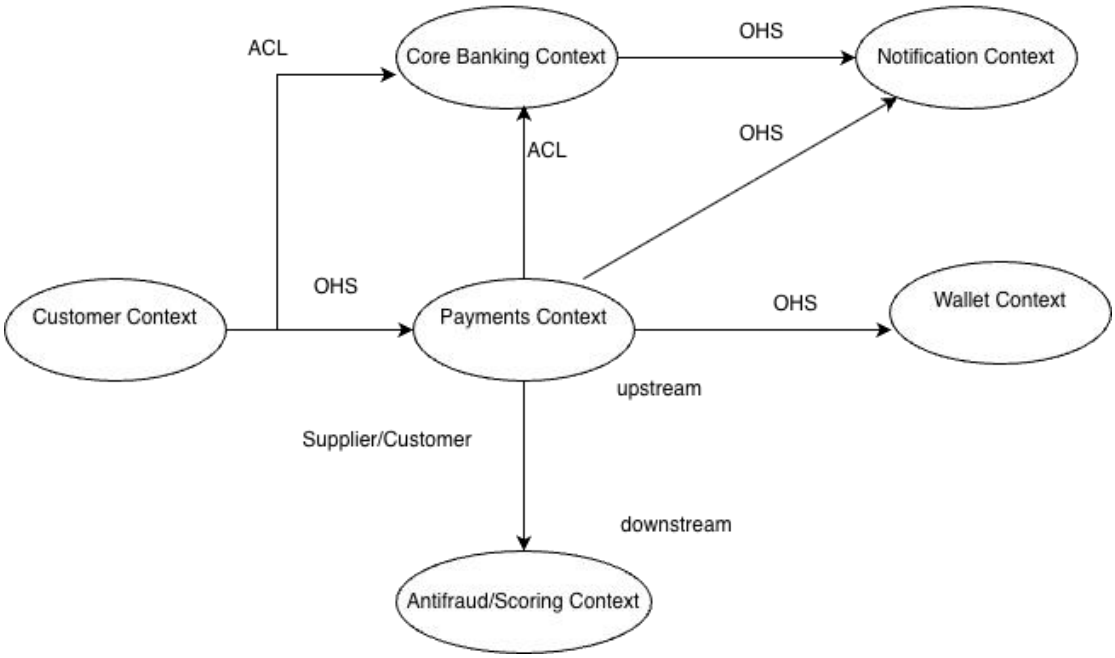
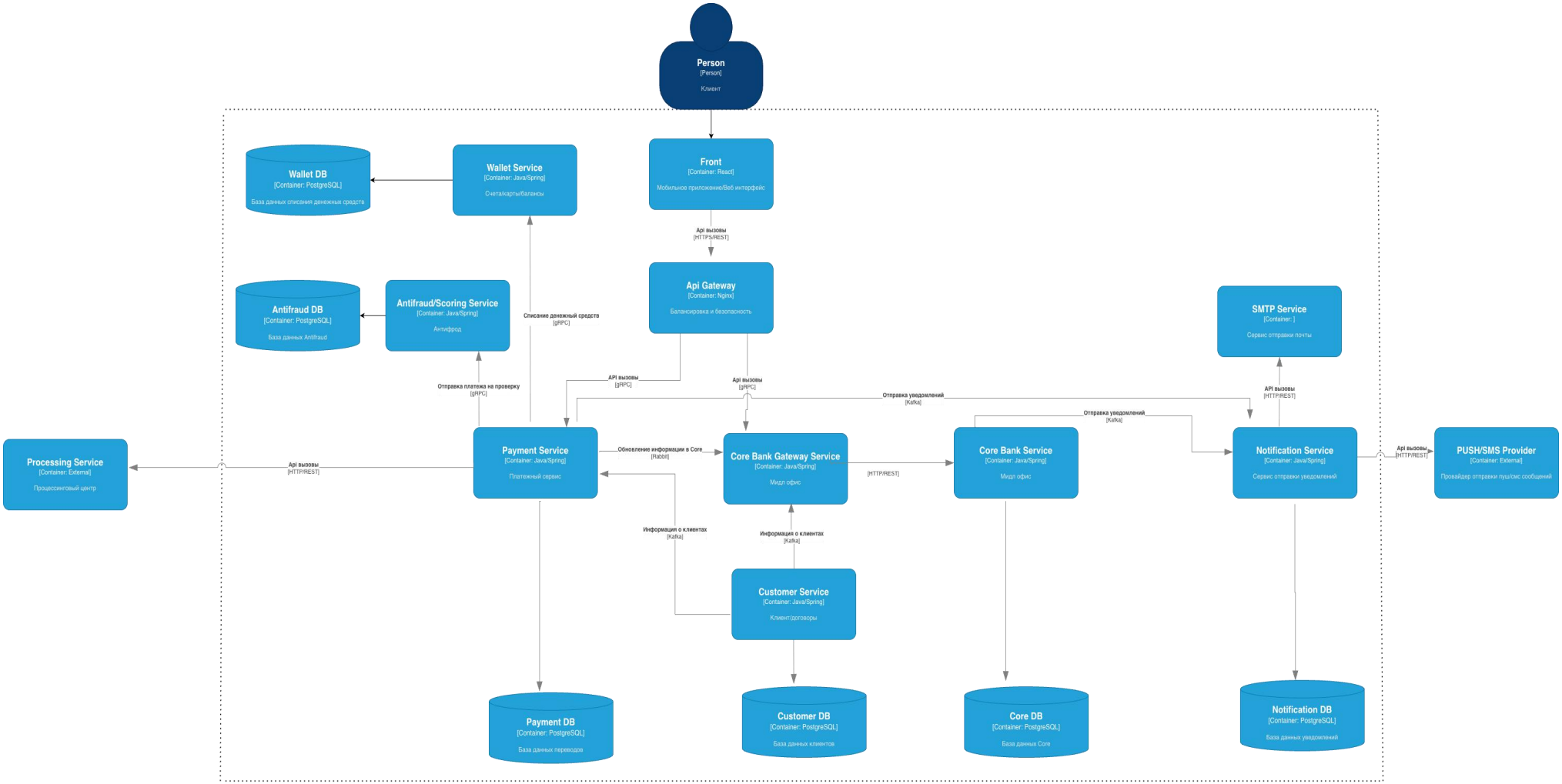

ACL
Core Banking Context
OHS
Notification Context
ACL
OHS
Customer Context
OHS
Payments Context
OHS
Wallet Context
upstream
Supplier/Customer
downstream
Antifraud/Scoring Context


Person
[Person]
Клиент
Wallet DB
[Container: PostgreSQL]
База данных списания денежных средств
Wallet Service
[Container: Java/Spring]
Счета/карты/балансы
Front
[Container: React]
Мобильное приложение/Веб интерфейс
Api вызовы
[HTTPS/REST]
Antifraud DB
[Container: PostgreSQL]
База данных Antifraud
Antifraud/Scoring Service
[Container: Java/Spring]
Антифрод
Списание денежный средств
[gRPC]
Api Gateway
[Container: Nginx]
Балансировка и безопасность
SMTP Service
[Container:]
Сервис отправки почты
Отправка платежа на проверку
[gRPC]
API вызовы
[gRPC]
API вызовы
[gRPC]
Отправка уведомлений
[Kafka]
API вызовы
[HTTP/REST]
Отправка уведомлений
[Kafka]
Processing Service
[Container: External]
Процессинговый центр
Api вызовы
[HTTP/REST]
Payment Service
[Container: Java/Spring]
Платежный сервис
Обновление информации в Core
[Rabbit]
Core Bank Gateway Service
[Container: Java/Spring]
Мидл офис
[HTTP/REST]
Core Bank Service
[Container: Java/Spring]
Мидл офис
Notification Service
[Container: Java/Spring]
Сервис отправки уведомлений
Api вызовы
[HTTP/REST]
PUSH/SMS Provider
[Container: External]
Провайдер отправки пуш/смс сообщений
Информация о клиентах
[Kafka]
Информация о клиентах
[Kafka]
Customer Service
[Container: Java/Spring]
Клиент/договоры
Payment DB
[Container: PostgreSQL]
База данных переводов
Customer DB
[Container: PostgreSQL]
База данных клиентов
Core DB
[Container: PostgreSQL]
База данных Core
Notification DB
[Container: PostgreSQL]
База данных уведомлений

Поток/сценарий	Брокер (Kafka/RabbitMQ /нет)	Тип сообщений(event/ command/job)	Обоснование выбора
Payments → Core Bank Gateway	RabbitMQ	command	Обновляем информацию в Core через gateway. Синхронность тут не нужна. Выбор в пользу Rabbit: не очень большой объем передаваемой информации.
Customer Service → Paymenst && Core Bank Gateway	Kafka	event	МС подписываются на топик, и получают данные о клиентах. Удобство использования - большой объем данных сливаем в топик, а потребители, которым нужны данные просто подписываются на топик.
Payments && Core Bank → Notification	Kafka	event	Критично время передачи/обработки. Большой объем данных. Возможно понадобится горизонтальное масштабирование (kafka тут предпочтительнее).

Поток/сценарий	REST/gRPC	Обоснование выбора
Api Gateway→ Payment && Core Bank	gRPC	Критично время передачи/обработки. Требуется синхронно получить ответ и отдать на Front.
Payment → Wallet	gRPC	Критично время передачи/обработки. Требуется синхронно получить ответ и отдать на Front.
Core Bank Gateway→ Core Bank	Rest	Не критично время передачи/обработки. Core Bank является legacy МС, поэтому использование gRPC будет не оправдано.
Notification → SMTP	Rest	Не критично время передачи/обработки.
Notification → PUSH/SMS Provides	Rest	Внешняя система. Используем протоколы, которые предоставляет external system.
Payment→ Processing	Rest	Внешняя система. Используем протоколы, которые предоставляет external system.
Front→ Api Gateway	Rest	Ускорение времени разработки/ сопровождения.
Payments → Antifraud	gRPC	Критично время передачи/обработки. Требуется синхронно получить ответ и продолжить или запретить осуществлять перевод.

==========

Выбор Nginx в пользу производительности. Как правило в банках есть devops'ы с очень хорошими компетенциями, поэтому относительная сложность настройки Nginx перекрывается скоростью работы и масштабируемостью. На стороне Nginx будет проводиться проверка JWT токенов, с помощью которых можно получить доступ к RestAPI Payment и Core Bank Gateway. Так же он будет выступать в роли балансировщика нагрузки.

==========

Выбираем UUIDv7.

Создание в Payments DB paymentId нам позволит сделать единую точку идентификации во всех МС и системах.

В Wallet DB paymentId будет использоваться как FK.

В Kafka paymentId можно будет использоваться как ключ, что даст возможность попадания в одну патрицию.

В RabbitMQ, если есть необходимость, paymentId можно использовать как correlationId в обратной очереди.

В логах и трассировке paymentId упростит поиск за счет уникальности.

В callback-ах paymentId будет передаваться как уникальный идентификатор.

При создании в Payments DB paymentId (UUIDv7) необходимо будет создать поле paymentId (тип данных UUID), добавить на него уникальный индекс. Для сущесвующих записей в таблице сгенерировать и заполнить paymentId. Добавить paymentId во все связанные таблицы и добавить в них FK и индексы по paymentId. Старые id необходимо оставить для совместимости, но постепенно делать переезд с него зависимых систем. Для внешних систем на начальном этапе отдавать оба параметра.

Вставки/сортировки при введении paymentId окажутся немного дороже, но не особо критично.