





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 → Payment && Core Bank Gateway	Kafka	event	MC 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 MC, 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 использовать как ключ, что даст возможность попадания в одну партицию.

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 существующих 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 переход 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.