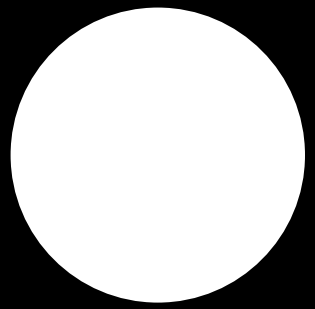
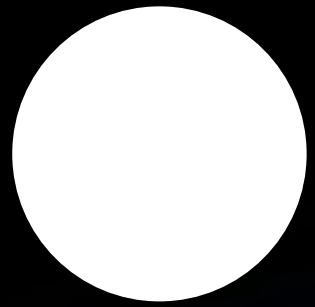
Архитектура данных цифрового рубля
Коммитет5х



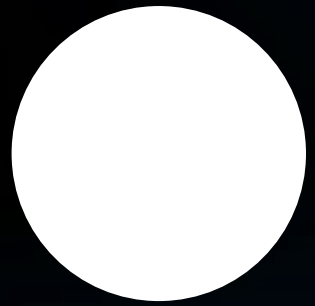
Григорий Шелепов

Архитектор



Дмитрий Мордвинов

Архитектор



Дмитрий Николаев

Архитектор

Никита Соловьев

Аналитик

Павел Никитин

Бизнес Аналитик

Вводная

При анализе задачи мы выработали для себя следующий подход:

1. Так как решение архитектурное, то для его выработки хорошо подходит ADR. При этом мы не стали придерживаться конкретного шаблона, а оформили основными структурными частями
2. Определили контекст – по сути текст задачи.
3. Выявили функциональные требования
4. Выявили не функциональные требования
5. Определили use cases
6. Определили ограничения
7. Сформировали Концептуальный слой архитектуры в виде контекстной диаграммы c4(выбран нами как наиболее простая нотация в условиях сжатых сроков)
8. Далее проработали схему слоя Логической архитектуры
9. И приступили к формированию слоя Физической архитектуры(контейнерная и компонентная схемы c4, модель данных)
10. Испытали те же чувства, что и на картинке, когда смогли

Контекст

Цифровой рубль — это цифровая форма рубля. Сейчас у нас есть наличная (банкноты и монеты в наших кошельках) и безналичная (деньги на счетах в банках) форма национальной валюты, а в дополнение к ним появилась еще и цифровая.

Цифровые рубли будут храниться на счетах цифрового рубля (цифровых кошельках) граждан и организаций. Счета цифрового рубля открываются на платформе цифрового рубля Банка России, здесь же проходят все операции с цифровым рублем. Доступ граждан к счетам цифрового рубля осуществляется посредством финансовых посредников через привычные дистанционные каналы: мобильные приложения банков и интернет-банки.

Цифровой рубль был создан для того, чтобы стать еще одним средством для платежей и переводов, которое не будет зависеть от ограничений банков в виде комиссий и лимитов. Цифровой рубль позволяет гражданам свободно расплачиваться и переводить цифровые рубли в пределах остатков средств на счете цифрового рубля. Операции для граждан будут бесплатными, а для бизнеса — с минимальной комиссией. Цифровые рубли эквивалентны наличным и безналичным: 1 рубль = 1 безналичный рубль = 1 цифровой рубль.

На текущий момент проект цифрового рубля входит в фазу полноценного пилотирования на большом количестве банков и с привлечением большого количества физических лиц.

Функциональные требования

1. Подготовить описание архитектуры работы с цифровым рублем, состоящую из трех уровней: концептуального, логического и физического
2. Минимальное время отклика на получение данных от сущностей участвующих в операциях
3. Формирование аналитической отчетности о всех видах операций.

Нефункциональные требования

1. Архитектура должна описывать (верхнеуровнево) состав объектов, отношения объектов, их взаимосвязь, потоки передачи данных между сервисами и основные действия, которые будут с ними производиться.
2. Высокий уровень надежности и безопасности предлагаемого решения
3. Соблюдение требований законодательства в сфере

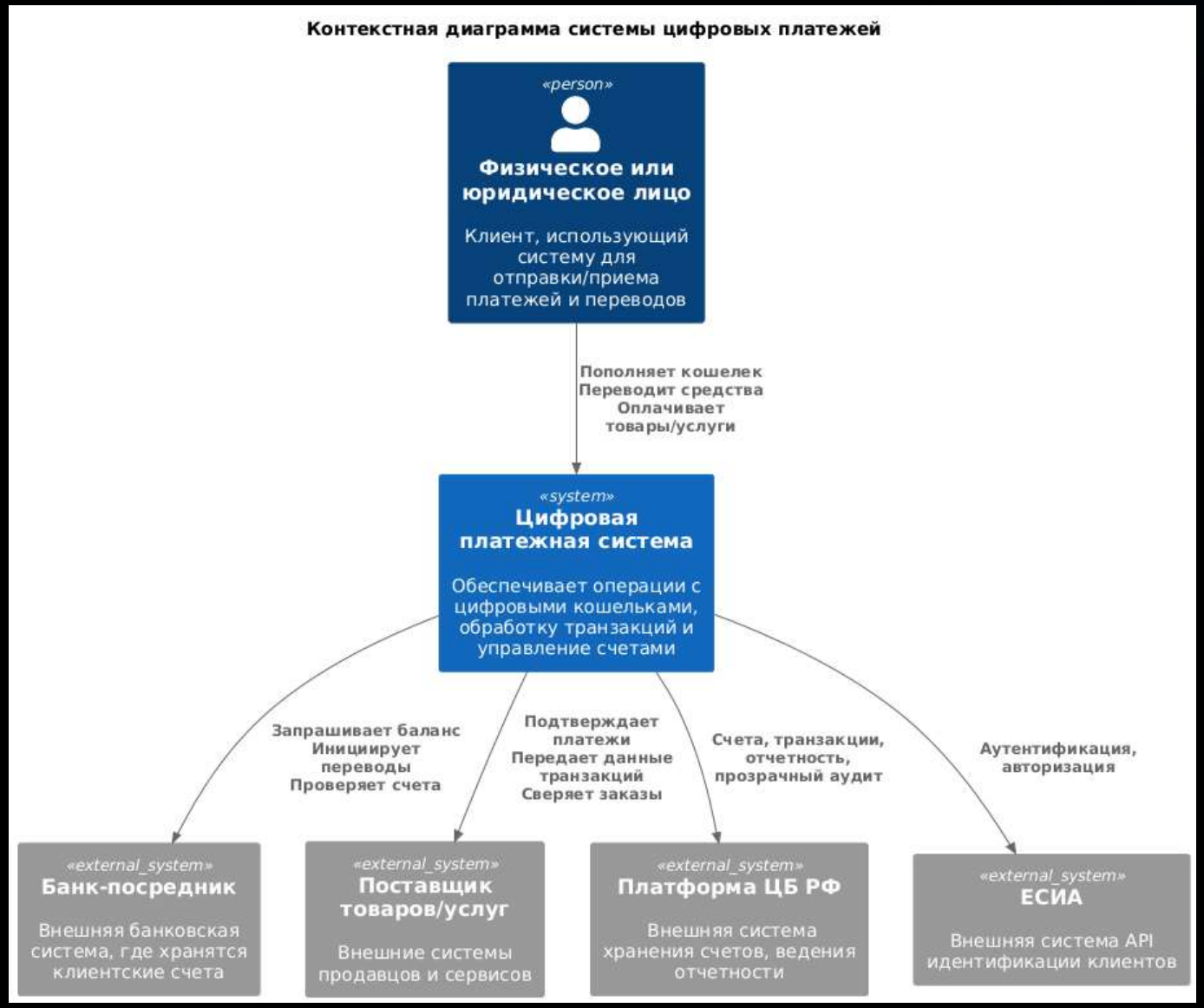
Use cases

1. Оплата товаров и услуг с помощью онлайн-кошелька на платформе ЦБ или офлайн-кошелька на мобильном устройстве
2. Приобретение товаров и услуг в случае недостаточности средств в цифровом кошельке, но наличии средств на счете клиента у финансового посредника клиента
3. Формирование поручения на автоматический (разовый или регулярный) перевод

Ограничения

1. Недостаточно информации для выявления нефункциональных требований по использованию ЦР.
2. Отсутствие возможности подключать экспертов в доменной области для анализа скрытых требований.

Слой концептуальной архитектуры



Слой концептуальной архитектуры

```
@startuml
!include <C4/C4_Context>
!include <C4/C4_Container>

title Контекстная диаграмма системы цифровых платежей

Person(physical_client, "Физическое или юридическое лицо", "Клиент, использующий систему для отправки/приема платежей и переводов")
' Person(legal_client, "Юридическое лицо", "Компания, принимающая платежи через систему")

System(payment_system, "Цифровая платежная система", "Обеспечивает операции с цифровыми кошельками, обработку транзакций и управление счетами")

System_Ext(bank, "Банк-посредник", "Внешняя банковская система, где хранятся клиентские счета")
System_Ext(merchant, "Поставщик товаров/услуг", "Внешние системы продавцов и сервисов")
System_Ext(cbr_system, "Платформа ЦБ РФ", " Внешняя система хранения счетов, ведения отчетности")
System_Ext(gosuslugi, "ЕСИА", "Внешняя система API идентификации клиентов")

Rel(physical_client, payment_system, "Пополняет кошелек\nПереводит средства\nОплачивает товары/услуги")
' Rel(legal_client, payment_system, "Принимает платежи\nУправляет счетами\nПолучает переводы")

Rel(payment_system, bank, "Запрашивает баланс\nИнициирует переводы\nПроверяет счета")
Rel(payment_system, merchant, "Подтверждает платежи\nПередает данные транзакций\nСверяет заказы")
Rel(payment_system, cbr_system, "Счета, транзакции, отчетность, прозрачный аудит")
Rel(payment_system, gosuslugi, "Аутентификация, авторизация")
@enduml
```

Слой концептуальной архитектуры

Описание:
Диаграмма определяет границы системы и ключевые взаимодействия между:
Участниками (физические/юридические лица).
Цифровой платежной системой (ядро обработки операций).
Внешними системами (банк-посредник, мерчанты).

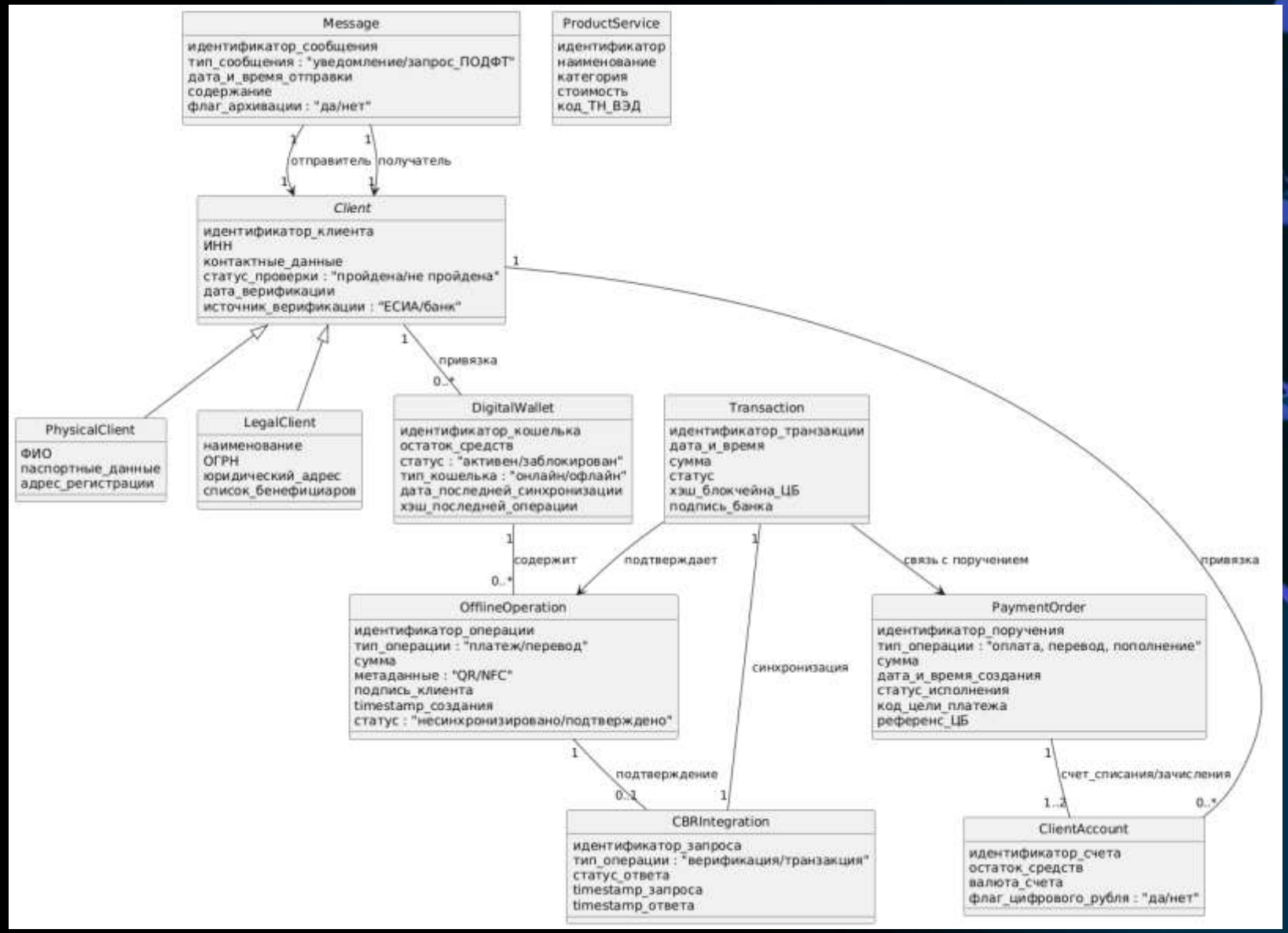
Ключевые компоненты

Элемент	Тип	Роль
physical_client	Person	Клиент (физ. лицо). Операции: пополнение, переводы, оплата товаров.
legal_client	Person	Мерчант (юр. лицо). Прием платежей, управление счетами.
payment_system	System	Ядро системы. Обработка транзакций, управление кошельками и счетами.
bank	System_Ext	Банк-посредник. Хранение счетов, проверка балансов, переводы.
merchant	System_Ext	Поставщик товаров/услуг. Подтверждение платежей, сверка данных.

Критические взаимодействия

Связь	Протокол/Формат	Назначение
physical_client → payment_system	HTTPS/API	Пополнение кошелька, переводы, оплата товаров.
legal_client → payment_system	HTTPS/API	Прием платежей, управление счетами, отчетность.
payment_system → bank	FPS API/gRPC	Проверка балансов, инициация переводов, верификация счетов.
payment_system → merchant	REST/ISO 8583	Подтверждение платежей, передача данных транзакций
merchant	System_Ext	Поставщик товаров/услуг. Подтверждение платежей, сверка данных.

Слой логической архитектуры



Слой логической архитектуры

```
@startuml
skinparam classAttributeIconSize 0
hide circle

abstract class Client {
  идентификатор_клиента
  ИНН
  контактные_данные
  статус_проверки : "пройдена/не пройдена"
  дата_верификации
  источник_верификации : "ЕСИА/банк"
}

class PhysicalClient {
  ФИО
  паспортные_данные
  адрес_регистрации
}

class LegalClient {
  наименование
  ОГРН
  юридический_адрес
  список_бенефициаров
}

Client <|-- PhysicalClient
Client <|-- LegalClient

class DigitalWallet {
  идентификатор_кошелька
  остаток_средств
  статус : "активен/заблокирован"
  тип_кошелька : "онлайн/офлайн"
  дата_последней_синхронизации
  хэш_последней_операции
}

class OfflineOperation {
  идентификатор_операции
  тип_операции : "платеж/перевод"
  сумма
  метаданные : "QR/NFC"
  подпись_клиента
  timestamp_создания
  статус : "несинхронизировано/подтверждено"
}

Client "1" -- "0..*" DigitalWallet : привязка
DigitalWallet "1" -- "0..*" OfflineOperation : содержит

class ClientAccount {
  идентификатор_счета
  остаток_средств
  валюта_счета
  флаг_цифрового_рубля : "да/нет"
}
Client "1" -- "0..*" ClientAccount : привязка

class PaymentOrder {
  идентификатор_поручения
  тип_операции : "оплата, перевод, пополнение"
  сумма
  дата_и_время_создания
  статус_исполнения
  код_цели_платежа
  референс_ЦБ
}

PaymentOrder "1" -- "1..2" ClientAccount :
счет_списания/зачисления

class Transaction {
  идентификатор_транзакции
  дата_и_время
  сумма
  статус
  хэш_блокчейна_ЦБ
  подпись_банка
}

Transaction --> PaymentOrder : связь с поручением
Transaction --> OfflineOperation : подтверждает
```

Слой логической архитектуры

```
class CBRIntegration {
  идентификатор_запроса
  тип_операции : "верификация/транзакция"
  статус_ответа
  timestamp_запроса
  timestamp_ответа
}

Transaction "1" -- "1" CBRIntegration : синхронизация
OfflineOperation "1" -- "0..1" CBRIntegration : подтверждение

class ProductService {
  идентификатор
  наименование
  категория
  стоимость
  код_ТН_ВЭД
}

class Message {
  идентификатор_сообщения
  тип_сообщения : "уведомление/запрос_ПОДФТ"
  дата_и_время_отправки
  содержание
  флаг_архивации : "да/нет"
}

Message "1" --> "1" Client : отправитель
Message "1" --> "1" Client : получатель
@enduml
```

Слой логической архитектуры

Описание:
Назначение диаграммы
Логическая модель данных определяет ключевые сущности системы и их взаимосвязи, обеспечивая:
- Единую концептуальную основу для разработки
- Соответствие требованиям ЦБ РФ
- Поддержку всех видов операций с цифровым рублем

Ключевые сущности и их атрибуты
Клиенты (Client)
- Абстрактный класс с общими атрибутами:
- Идентификатор и ИНН
- Статус верификации (ПОД/ФТ)
- Источник проверки (ЕСИА/банк)
- Специализации:
- Физические лица (ФИО, паспорт)
- Юридические лица (ОГРН, бенефициары)

Цифровые кошельки (DigitalWallet)
- Тип (онлайн/офлайн)
- Статус и баланс
- Хэш последней операции (целостность данных)
- Связь с офлайн-операциями

Платежные операции
- Платежные поручения (PaymentOrder)
- Типы операций
- Референс ЦБ
- Транзакции (Transaction)
- Подпись банка
- Хэш в блокчейне ЦБ

Интеграция с ЦБ (CBRIntegration)
- Статус синхронизации
- Таймстампы запросов/ответов

Критические взаимосвязи
Клиент-Кошелек (1:N)
- Поддержка множества кошельков
- азделение онлайн/офлайн режимов

Кошелек-ОфлайнОперации (1:N)
- Механизм отложенной синхронизации
- Статусы подтверждения

Транзакции-ЦБ (1:1)
- Гарантия аудита операций
- Связь с блокчейном ЦБ

Соответствие требованиям
ПОД/ФТ:
- Полная идентификация клиентов
- Атрибуты верификации
- Логирование изменений

Офлайн-режим:
- Сущность OfflineOperation
- Механизм синхронизации

Интеграция с ЦБ:
- Референсы в платежах
- Сквозная идентификация
- Валидация use-case

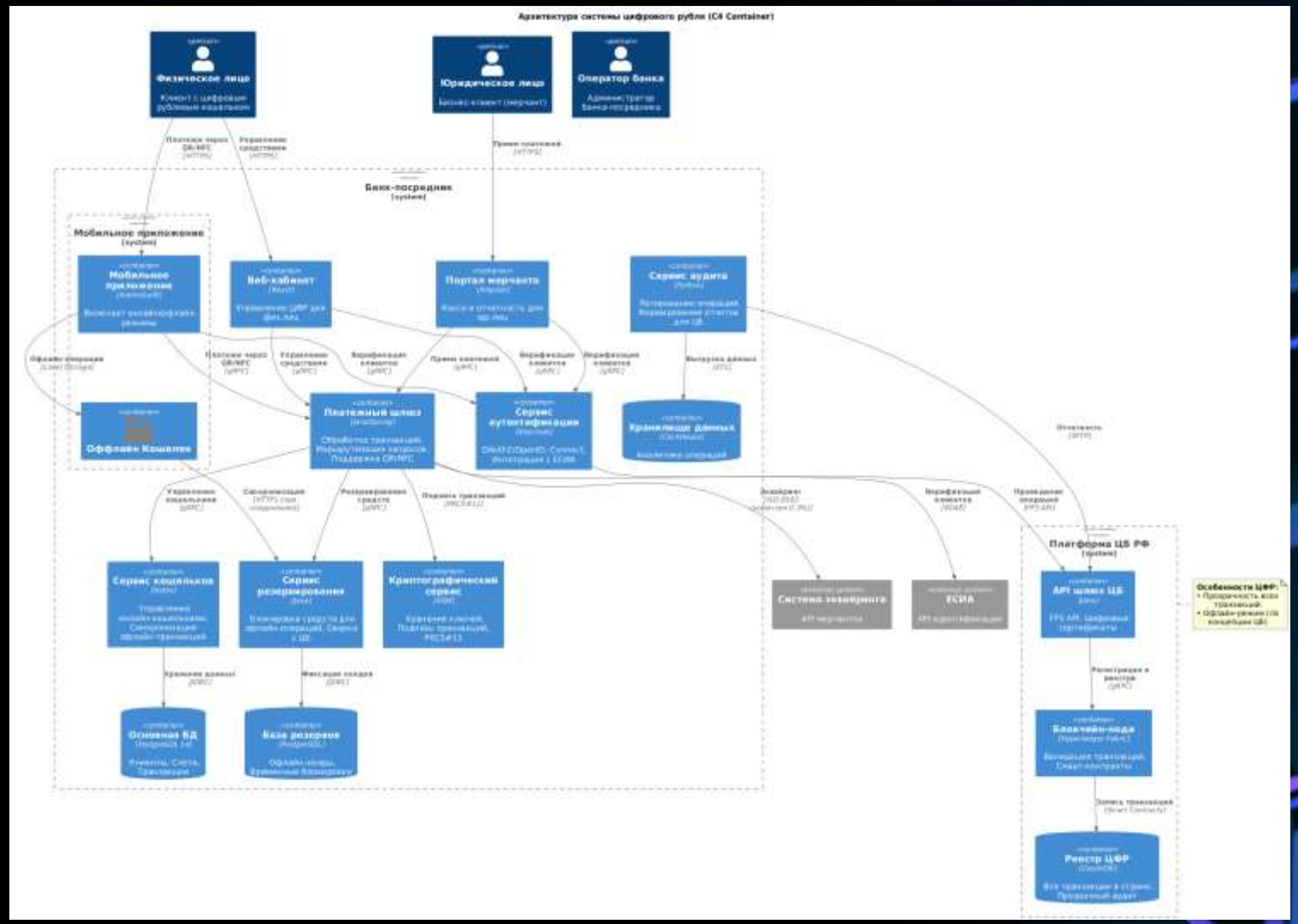
Оплата товаров:
- Поддержка через PaymentOrder
- QR/NFC в метаданных

Автоплатежи:
- Регулярные переводы через тип операции

Офлайн-операции:
- Механизм подтверждения через CBRIntegration

Ограничения модели
- Не указаны лимиты операций
- Нет детализации комиссий
- Требуется расширение для сложных сценариев

Слой физической архитектуры. Контейнерная диаграмма



Слой физической архитектуры. Контейнерная диаграмма

```
@startuml
!include <C4/C4_Context>
!include <C4/C4_Container>

title Архитектура системы цифрового рубля (C4 Container)

' === Акторы ===
Person(physical_client, "Физическое лицо", "Клиент с цифровым рублевым кошельком")
Person(legal_client, "Юридическое лицо", "Бизнес-клиент (мерчант)")
Person(bank_operator, "Оператор банка", "Администратор банка-посредника")

' === Системные границы ===
System_Boundary(bank_system, "Банк-посредник") {
    System_Boundary(mobile, "Мобильное приложение") {
        Container(mobile_app, "Мобильное приложение", "Kotlin/Swift", "Включает онлайн/офлайн режимы")
        Container(offline_wallet, "Оффлайн Кошелек", $sprite="&folder,scale=5.0,color=gray")
    }
    Container(web_app, "Веб-кабинет", "React", "Управление ЦФР для физ.лиц")
    Container(merchant_portal, "Портал мерчанта", "Angular", "Касса и отчетность для юр.лиц")

    Container(auth_service, "Сервис аутентификации", "Keycloak", "OAuth2/OpenID, Connect, Интеграция с ЕСИА")

    Container(payment_api, "Платежный шлюз", "Java/Spring", "Обработка транзакций,  Маршрутизация запросов, Поддержка QR/NFC")

    Container(wallet_service, "Сервис кошельков", "Kotlin", " Управление онлайн-кошельками, Синхронизация офлайн-транзакций")

    Container(reservation_service, "Сервис резервирования", "Java", " Блокировка средств для офлайн-операций, Сверка с ЦБ")

    Container(crypto_service, "Криптографический сервис", "HSM", " Хранение ключей, Подпись транзакций, PKCS#11")

    Container(audit_service, "Сервис аудита", "Python", " Логирование операций, Формирование отчетов для ЦБ")

    ' Базы данных
    ContainerDb(main_db, "Основная БД", "PostgreSQL 14", " Клиенты, Счета, Транзакции")

    ContainerDb(reservation_db, "База резервов", "PostgreSQL", " Офлайн-холды, Временные блокировки")

    ContainerDb(dwh, "Хранилище данных", "ClickHouse", "Аналитика операций")
}

System_Boundary(cbr_system, "Платформа ЦБ РФ") {
    Container(cbr_gateway, "API шлюз ЦБ", "Java", " FPS API, Цифровые сертификаты")

    Container(blockchain_node, "Блокчейн-нода", "Hyperledger Fabric", " Валидация транзакций, Смарт-контракты")
```

Слой физической архитектуры. Контейнерная диаграмма

```
 ContainerDb(cbr_ledger, "Реестр ЦФР", "CouchDB", " Все транзакции в стране,
Прозрачный аудит")
}

' === Внешние системы ===
System_Ext(merchant_acq, "Система эквайринга", "API мерчантов")
System_Ext(gosuslugi, "ЕСИА", "API идентификации")

' === Связи ===
' Клиентские взаимодействия
Rel(physical_client, mobile_app, "Платежи через QR/NFC", "HTTPS")
Rel(physical_client, web_app, "Управление средствами", "HTTPS")
Rel(legal_client, merchant_portal, "Прием платежей", "HTTPS")
Rel(web_app, payment_api, "Управление средствами", "gRPC")
Rel(mobile_app, payment_api, "Платежи через QR/NFC", "gRPC")
Rel(merchant_portal, payment_api, "Прием платежей", "gRPC")


' Банковская инфраструктура
Rel(mobile_app, offline_wallet, "Офлайн-операции", "Local Storage")
Rel(offline_wallet, reservation_service, "Синхронизация", "HTTPS (при
соединении)")

Rel(payment_api, wallet_service, "Управление кошельками", "gRPC")
Rel(payment_api, reservation_service, "Резервирование средств", "gRPC")
Rel(payment_api, crypto_service, "Подпись транзакций", "PKCS#11")

Rel(wallet_service, main_db, "Хранение данных", "JDBC")
Rel(reservation_service, reservation_db, "Фиксация холдов", "JDBC")

' Интеграция с ЦБ
Rel(payment_api, cbr_gateway, "Проведение операций", "FPS API")
Rel(cbr_gateway, blockchain_node, "Регистрация в реестре", "gRPC")
Rel(blockchain_node, cbr_ledger, "Запись транзакций", "Smart Contracts")

' Дополнительные интеграции
Rel(payment_api, merchant_acq, "Эквайринг", "ISO 8583\n(комиссия 0.3%)")

' Безопасность
Rel(auth_service, gosuslugi, "Верификация клиентов", "SOAP")
Rel(mobile_app, auth_service, "Верификация клиентов", "gRPC")
Rel(web_app, auth_service, "Верификация клиентов", "gRPC")
Rel(merchant_portal, auth_service, "Верификация клиентов", "gRPC")

' Аудит и аналитика
Rel(audit_service, dwh, "Выгрузка данных", "ETL")
Rel(audit_service, cbr_gateway, "Отчетность", "SFTP")

' === Легенда и примечания ===
note right of cbr_system
  <b>Особенности ЦФР:</b>
  • Прозрачность всех транзакций
  • Офлайн-режим (по концепции ЦБ)
end note
@enduml
```

Слой физической архитектуры. Контейнерная диаграмма

Описание:

Компонентная диаграмма детализирует:

- Внутреннюю структуру банка-посредника
- Взаимодействие сервисов между собой и с внешними системами
- Технологический стек реализации

Структурные компоненты системы

Фронтенд-сервисы (Gateway слои)

- Mobile Gateway (Go/Gin): API для мобильных приложений
- Web Gateway (Go/Fiber): API для веб-интерфейсов
- Merchant Gateway (Java/Vert.x): API для мерчантов

Бэкенд-сервисы (Ядро системы)

- Auth Service (Java/Spring Security): Аутентификация через ЕСИА
- Wallet Service (Go/GoKit): Управление балансами
- Transaction Service (Java/Quarkus): Обработка платежей
- Reservation Service (Go/GoChi): Блокировка средств
- Notification Service (Java/Micronaut): Уведомления

Инфраструктурные сервисы

- Kafka: Асинхронная обработка событий
- Redis: Кэширование и rate limiting
- Audit Service (Go/Echo): Логирование операций

Хранилища данных

- PostgreSQL (HA): Основные данные
- ClickHouse: Аналитика и отчетность

Ключевые интеграции

Внутренние взаимодействия

- gRPC: между сервисами (проверка баланса, блокировки)
- Kafka: транзакционные события (Avro/Protobuf)
- JDBC: доступ к данным

Внешние интеграции

- ЦБ РФ: FPS API (синхронно), Kafka Connect (асинхронно)
- ЕСИА: OAuth2 верификация
- Мерчанты: REST/ISO8583

Технологический стек

Языки программирования

- Go: высоконагруженные сервисы
- Java: бизнес-логика

Инфраструктура

- Kubernetes: оркестрация
- Prometheus+Grafana: мониторинг

Импортозамещение

- OpenJDK, PostgreSQL, ClickHouse

Обеспечение надежности

Обработка ошибок

- Dead Letter Queue в Kafka
- Патрони для PostgreSQL

Масштабируемость

- Микросервисная архитектура
- Горизонтальное масштабирование

Производительность

- gRPC для межсервисного взаимодействия
- Redis для кэширования
- Соответствие требованиям

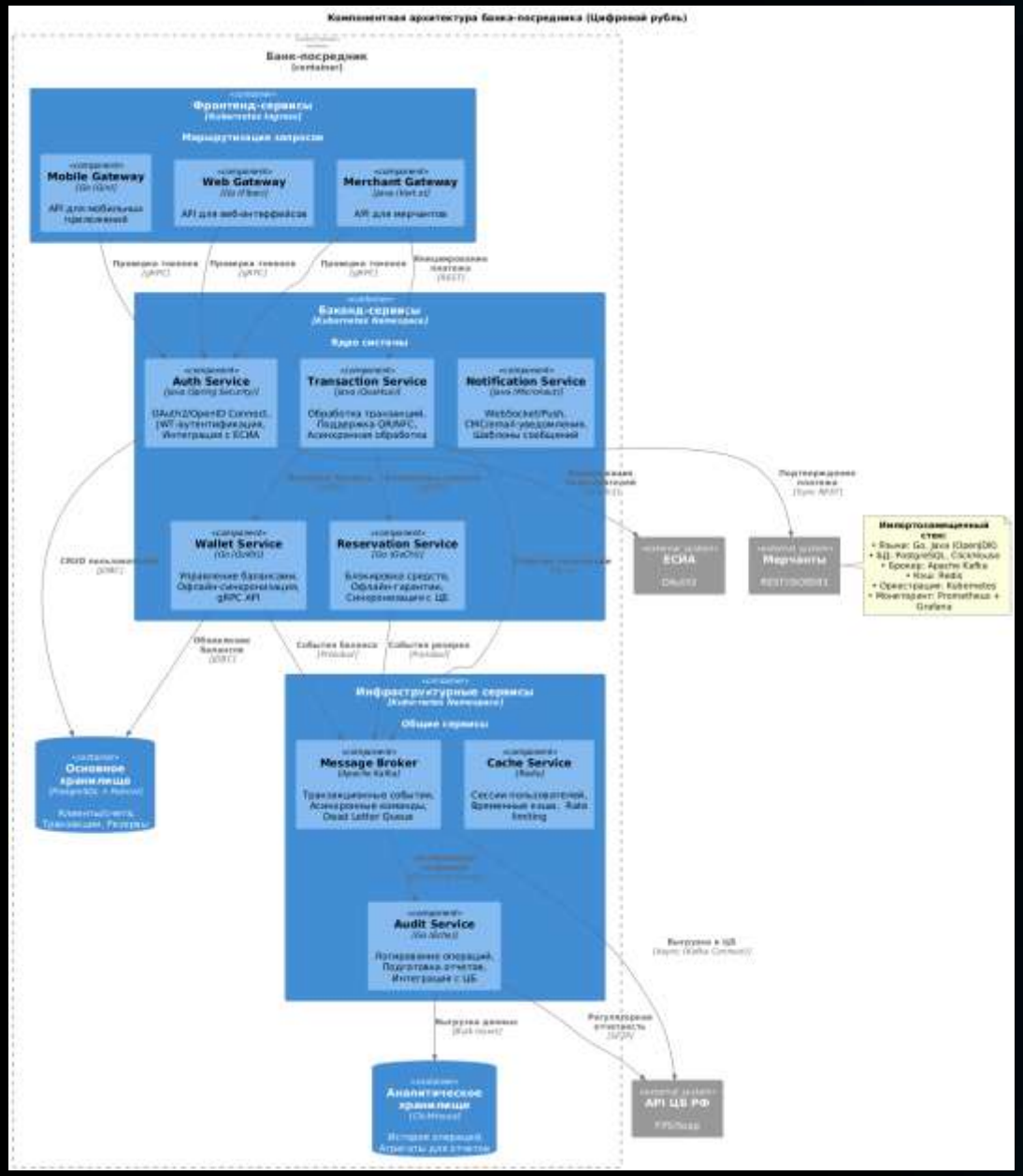
Безопасность

- Двухфакторная аутентификация
- Шифрование данных

Отчетность

- SFTP выгрузка в ЦБ
- Подробное логирование

Слой физической архитектуры. Компонентная диаграмма



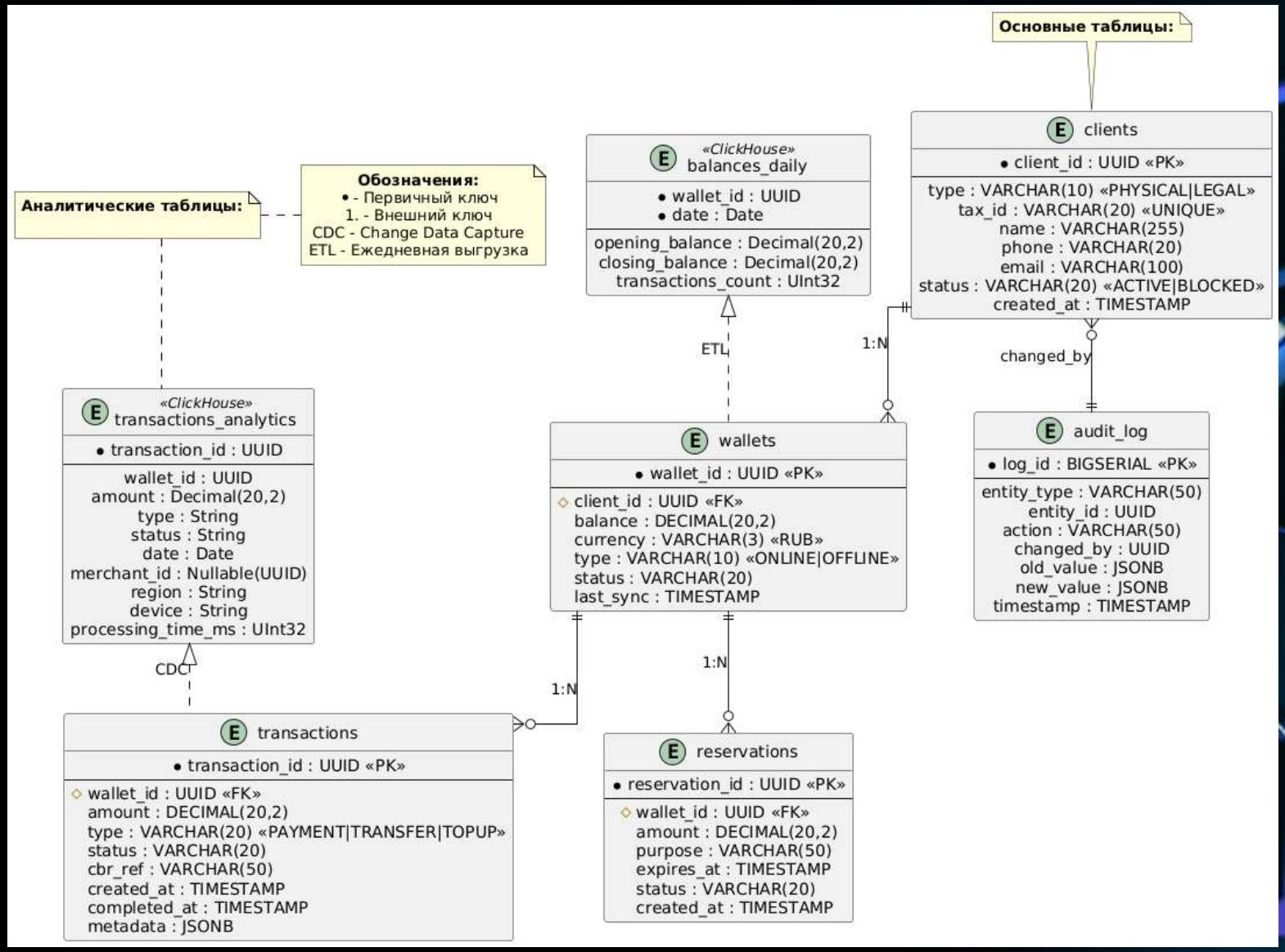
Слой физической архитектуры. Компонентная диаграмма

```
@startuml
!include <C4/C4_Component>
!include <C4/C4_Container>

title Компонентная архитектура банка-посредника (Цифровой рубль)

' === Контейнеры ===
Container_Boundary(bank, "Банк-посредник") {
    ' ----- Frontend -----
    Container(frontend, "Фронтенд-сервисы", "Kubernetes Ingress",
"Маршрутизация запросов") {
        Component(mobile_gw, "Mobile Gateway", "Go (Gin)", "API для мобильных
приложений")
        Component(web_gw, "Web Gateway", "Go (Fiber)", "API для веб-
интерфейсов")
        Component(merchant_gw, "Merchant Gateway", "Java (Vert.x)", "API для
мерчантов")
    }

    ' ----- Бэкенд -----
    Container(backend, "Бэкенд-сервисы", "Kubernetes Namespace", "Ядро
системы") {
        Component(auth_service, "Auth Service", "Java (Spring Security)",
"OAuth2/OpenID Connect,  JWT-аутентификация,  Интеграция с ЕСИА")

        Component(wallet_service, "Wallet Service", "Go (GoKit)", "Управление
балансами,  Офлайн-синхронизация,  gRPC API")

        Component(transaction_service, "Transaction Service", "Java (Quarkus)",
"Обработка транзакций,  Поддержка QR/NFC,  Асинхронная обработка")

        Component(reservation_service, "Reservation Service", "Go (GoChi)",
"Блокировка средств,  Офлайн-гарантии,  Синхронизация с ЦБ")

        Component(notification_service, "Notification Service", "Java (Micronaut)",
"WebSocket/Push,  CMC/email-уведомления,  Шаблоны сообщений")
    }

    ' ----- Инфраструктура -----
    Container(infra, "Инфраструктурные сервисы", "Kubernetes Namespace",
"Общие сервисы") {
        Component(kafka, "Message Broker", "Apache Kafka", "Транзакционные
события,  Асинхронные команды,  Dead Letter Queue")

        Component(redis, "Cache Service", "Redis", "Сессии пользователей,
Временные кэши,  Rate limiting")

        Component(audit_service, "Audit Service", "Go (Echo)", "Логирование
операций,  Подготовка отчетов,  Интеграция с ЦБ")
    }

    ' ----- Хранилища -----
    ContainerDb(postgres, "Основное хранилище", "PostgreSQL + Patroni", "
Клиенты/счета, Транзакции, Резервы")

    ContainerDb(clickhouse, "Аналитическое хранилище", "ClickHouse", " История
операций, Агрегаты для отчетов")
}
```

Слой физической архитектуры. Компонентная диаграмма

```
' === Внешние системы ===
System_Ext(cbr_gateway, "API ЦБ РФ", "FPS/Soap")
System_Ext(esia, "ЕСИА", "OAuth2")
System_Ext(merchants, "Мерчанты", "REST/ISO8583")

' === Связи ===
' Внешние интеграции
Rel(mobile_gw, auth_service, "Проверка токенов", "gRPC")
Rel(web_gw, auth_service, "Проверка токенов", "gRPC")
Rel(merchant_gw, auth_service, "Проверка токенов", "gRPC")
Rel(auth_service, esia, "Верификация пользователей", "OAuth2")

' Транзакционный поток
Rel(merchant_gw, transaction_service, "Инициирование платежа", "REST")
Rel(transaction_service, wallet_service, "Проверка баланса", "gRPC")
Rel(transaction_service, reservation_service, "Блокировка средств", "gRPC")
Rel(transaction_service, kafka, "Событие транзакции", "Avro")
Rel(kafka, cbr_gateway, "Выгрузка в ЦБ", "Async (Kafka Connect)")

' Асинхронные взаимодействия
Rel(wallet_service, kafka, "События баланса", "Protobuf")
Rel(reservation_service, kafka, "События резерва", "Protobuf")
Rel(kafka, audit_service, "Логирование операций", "Consumer Group")

' Хранение данных
Rel(auth_service, postgres, "CRUD пользователей", "JDBC")
Rel(wallet_service, postgres, "Обновление балансов", "JDBC")
Rel(audit_service, clickhouse, "Выгрузка данных", "Bulk Insert")

' Особые интеграции
Rel(transaction_service, merchants, "Подтверждение платежа", "Sync REST")
Rel(audit_service, cbr_gateway, "Регуляторная отчетность", "SFTP")

' === Технологический стек ===
note right
  <b>Импортозамещенный стек:</b>
  • Языки: Go, Java (OpenJDK)
  • БД: PostgreSQL, ClickHouse
  • Брокер: Apache Kafka
  • Кэш: Redis
  • Оркестрация: Kubernetes
  • Мониторинг: Prometheus + Grafana
end note
@enduml
```

Слой физической архитектуры. Компонентная диаграмма



```
@startuml
' Диаграмма физической модели данных в нотации PlantUML Structure

skinparam defaultTextAlignment center
skinparam linetype ortho
hide empty members

' === Основные сущности ===

entity "clients" {
  * client_id : UUID <<PK>>
  --
  type : VARCHAR(10) <<PHYSICAL|LEGAL>>
  tax_id : VARCHAR(20) <<UNIQUE>>
  name : VARCHAR(255)
  phone : VARCHAR(20)
  email : VARCHAR(100)
  status : VARCHAR(20) <<ACTIVE|BLOCKED>>
  created_at : TIMESTAMP
}

entity "wallets" {
  * wallet_id : UUID <<PK>>
  --
  # client_id : UUID <<FK>>
  balance : DECIMAL(20,2)
  currency : VARCHAR(3) <<RUB>>
  type : VARCHAR(10) <<ONLINE|OFFLINE>>
  status : VARCHAR(20)
  last_sync : TIMESTAMP
}

entity "transactions" {
  * transaction_id : UUID <<PK>>
  --
  # wallet_id : UUID <<FK>>
  amount : DECIMAL(20,2)
  type : VARCHAR(20) <<PAYMENT|TRANSFER|TOPUP>>
  status : VARCHAR(20)
  cbr_ref : VARCHAR(50)
  created_at : TIMESTAMP
  completed_at : TIMESTAMP
  metadata : JSONB
}

entity "reservations" {
  * reservation_id : UUID <<PK>>
  --
  # wallet_id : UUID <<FK>>
  amount : DECIMAL(20,2)
  purpose : VARCHAR(50)
  expires_at : TIMESTAMP
  status : VARCHAR(20)
  created_at : TIMESTAMP
}
entity "audit_log" {
  * log_id : BIGSERIAL <<PK>>
  --
  entity_type : VARCHAR(50)
  entity_id : UUID
  action : VARCHAR(50)
```

```
changed_by : UUID
  old_value : JSONB
  new_value : JSONB
  timestamp : TIMESTAMP
}

'

 === Аналитические таблицы ===

entity "transactions_analytics" <<ClickHouse>> {
  * transaction_id : UUID
  --
  wallet_id : UUID
  amount : Decimal(20,2)
  type : String
  status : String
  date : Date
  merchant_id : Nullable(UUID)
  region : String
  device : String
  processing_time_ms : UInt32
}

entity "balances_daily" <<ClickHouse>> {
  * wallet_id : UUID
  * date : Date
  --
  opening_balance : Decimal(20,2)
  closing_balance : Decimal(20,2)
  transactions_count : UInt32
}

' === Связи между таблицами ===

clients ||--o{ wallets : "1:N"
wallets ||--o{ transactions : "1:N"
wallets ||--o{ reservations : "1:N"
clients }o--|| audit_log : "changed_by"

transactions .up.|> transactions_analytics : "CDC"
wallets .up.|> balances_daily : "ETL"

' === Легенда ===

note top of clients
<b>Основные таблицы:</b>
end note

note top of transactions_analytics
<b>Аналитические таблицы:</b>

end note

note right
<b>Обозначения:</b>
* - Первичный ключ
# - Внешний ключ
CDC - Change Data Capture
ETL - Ежедневная выгрузка
end note
@enduml
```

Слой физической архитектуры. Схема данных

Описание:

Назначение диаграммы

Физическая модель данных определяет:

- Структуру таблиц и их взаимосвязи
- Типы данных и ограничения
- Разделение на оперативный и аналитический контуры

Оперативный контур (PostgreSQL)

Основные таблицы:

clients:

- Тип клиента (физическое/юридическое лицо)
- Уникальный налоговый идентификатор
- Статус активности

wallets:

- Тип кошелька (онлайн/офлайн)
- Баланс и валюта (только RUB)
- Время последней синхронизации

transactions:

- Референс ЦБ (cbr_ref)
- Метаданные в JSONB
- Временные метки создания/завершения

Дополнительные таблицы:

reservations:

- Блокировки средств
- Срок действия резерва

audit_log:

- Полное журналирование изменений
- Старые/новые значения в JSONB

Аналитический контур (ClickHouse)

Оптимизированные таблицы:

transactions_analytics:

- Денормализованные данные
- География и устройства
- Время обработки

balances_daily:

- Ежедневные срезы
- Количество операций

Ключевые связи

Основные связи:

- Клиент → Кошельки (1:N)
- Кошелек → Транзакции (1:N)
- Кошелек → Резервы (1:N)

Механизмы интеграции:

CDC для транзакций

ETL для ежедневных балансов

Технические особенности

Типы данных:

- UUID для идентификаторов
- DECIMAL(20,2) для денежных значений
- JSONB для гибких атрибутов

Индексы:

- Первичные ключи (PK)
- Внешние ключи (FK)
- Уникальный tax_id

Соответствие требованиям

Производительность:

- HA-кластер PostgreSQL
- Распределенный ClickHouse

Аудит:

- Полное журналирование изменений
- Поддержка ПОД/ФТ

Отчетность:

- Готовые агрегаты
- История операций

Покрытие функциональных требований

Концептуальный уровень представлен в контекстной диаграмме, где описаны основные взаимодействия между клиентами, платежной системой, банком-посредником и мерчантами.

На **логическом** уровне описаны сущности (клиенты, кошельки, транзакции) и их взаимосвязи.

Физический уровень хорошо проработан в схеме данных, где представлены таблицы для хранения данных (клиенты, кошельки, транзакции, резервы) и их связи. Также включены аналитические таблицы в ClickHouse, что соответствует требованию формирования отчетности.

Контейнерная и компонентная архитектуры (container_diagramm.txt и component-diagramm.txt) детализируют техническую реализацию, включая сервисы для обработки транзакций, аутентификации, резервирования и аналитики.
Требования к производительности и отчетности частично выполнены:
Для оперативного контура используются PostgreSQL и Redis, что обеспечивает минимальное время отклика.
Для аналитики задействован ClickHouse, что позволяет формировать разнообразную отчетность.
К сожалению, из-за недостатка времени и возможности привлечения экспертов, не удалось покрыть все требования целиком и у решения есть недостатки, требующие дополнительной проработки. Однако, считаем, что все ключевые требования проработаны и рекомендуем решение к реализации.

Покрытие нефункциональных требований

Надежность и отказоустойчивость
- В решении предлагается использовать Kubernetes для оркестрации сервисов, что обеспечит отказоустойчивость.
- PostgreSQL + Patroni и ClickHouse (Distributed) гарантируют надежность хранения данных.
- Kafka обеспечивает асинхронную обработку транзакций и Dead Letter Queue для обработки ошибок.
- Резервирование средств (reservation_service) поддерживает офлайн-операции.

Безопасность
- Аутентификация через ЕСИА (auth_service + OAuth2/OpenID Connect).
- Подпись транзакций (crypto_service + HSM).
- Аудит операций (audit_service + SFTP-выгрузка в ЦБ).
- Шифрование данных (в data-schema.txt есть подпись_клиента, хэш_блокчейна_ЦБ).

Соответствие ПОД/ФТ/ФРОМУ

В решении предложены для использования:
- статус_проверки и источник_верификации для клиентов.
- код_цели_платежа и референс_ЦБ в PaymentOrder.
- Логирование в audit_log (data-schema.txt).

Производительность
- Redis для кэширования и rate limiting.
- ClickHouse для аналитики (быстрые агрегации).
- gRPC для межсервисного взаимодействия (низкие задержки).

Масштабируемость
- Kubernetes + микросервисы (wallet_service, transaction_service и др.).
- Kafka для обработки событий в реальном времени.

Покрытие use cases

Вывод:

Представленная архитектура покрывает представленные в задаче use cases. Но предполагаем, что есть места, которые требуют более тщательной проработки с подключением экспертов. Так на наш взгляд:

- Онлайн/офлайн-оплата покрыта полностью.
- Недостаток средств на кошельке требует доработки интеграции с банком(требуется эксперт со стороны банка).
- Регулярные платежи нуждаются в явном описании в логической модели и отдельном сервисе(требуется больше времени на проработку и описание решения).

Технологический стек

Фронт: React, Angular

Бэкенд: Go, Java (Spring, Quarkus)

Базы данных: PostgreSQL, ClickHouse

Инфраструктура: Kubernetes, Kafka, Redis

Безопасность: HSM, OAuth2, шифрование

Благодарности

Наша команда выражает искреннюю благодарность устроителям хакатона, так как вместо того чтобы впустую потратить время на такие скучные и рутинные вещи, как:

- Сон
- Еда
- Общение с родными
- Спорт, прогулки и т.д.

Мы упорно перекладывая ответственность друг на друга, проявляя чудеса изворотливости, задействуя все скрытые и не скрытые возможности нашего организма, превозмогая все возникающие трудности, несколько раз переругавшись вдрызг, столько же раз помирившись дошли таки до конечного результата, вновь убедившись в нерушимости простой мужской дружбы и заодно погрузились в новую для нас тему Цифрового Рубля, за которым вполне просматривается технологическое будущее. Отдельную благодарность, а также неописуемую радость, вызовет новость, что наш труд был не напрасным.

Заключение

- Архитектура соответствует ключевым требованиям ЦБ РФ.
- Необходимы доработки для регулярных платежей и интеграции с банком.
- Готова к пилотированию с учетом рекомендаций.
- Наши эксперты-котики одобряют!