Отчет

по реализации функционала «Блокировки платежей»

Выполнила:

Ерофеева Дарья Денисовна

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы:

Спроектировать и реализовать сервис, который позволяет временно останавливать платежи клиента в ситуациях, когда это необходимо.

Такие случаи могут возникать, например, если клиент вызывает подозрение на мошенничество или если у него некорректные реквизиты, из-за чего банк отклоняет платежи.

В рамках задания нужно было:

1. продумать, как именно будет работать процесс блокировки и разблокировки;
2. описать необходимые API-методы;
3. разработать структуру хранения данных в базе;
4. убедиться, что система позволяет отличать разные причины блокировок и корректно показывает текущий статус клиента.

По итогам была создана и полностью проверена работающая реализация сервиса на основе FastAPI и PostgreSQL.

ОПИСАНИЕ ЗАДАЧИ

В рамках работы нужно было решить практическую задачу – сделать механизм, который позволит временно блокировать платежи определённого клиента.

Это важно для ситуаций, когда:

1. есть риск мошенничества, и операции нужно «заморозить» до выяснения обстоятельств;

2. у клиента неверные реквизиты, и платежи не проходят, пока он их не уточнит.

Чтобы сервис можно было использовать другими системами банка, требовалось подготовить:

1. API-интерфейс – методы для установки блокировки, снятия, проверки статуса и просмотра истории.

2. Структуру данных в базе – чтобы можно было хранить причины блокировок, статусы, время создания, кто поставил и кто снял.

4. Работающий прототип – API, связанное с PostgreSQL, в котором блокировки создаются, читаются и снимаются.

Задача включала как проектирование, так и техническую реализацию, то есть не только описать решение, но и убедиться, что оно действительно работает.

АРХИТЕКТУРА И КЛЮЧЕВЫЕ РЕШЕНИЯ

Для реализации сервиса была выбрана простая и понятная архитектура, в которой API напрямую работает с базой данных. Такой подход позволяет быстро проверять гипотезы и прозрачно отслеживать логику работы.

1. Компоненты решения

1. Backend (FastAPI)

Используется как основной интерфейс для взаимодействия с системой.

Он отвечает за:

- приём запросов,
- валидацию данных,
- проверку прав доступа,
- запись и чтение информации из БД.

2. JWT-аутентификация и роли

Чтобы разделить права доступа, используется механизм JWT-токенов.

Выделены роли:

- ops.block:create – постановка блокировки,
- ops.block:release – снятие блокировки,
- ops.block:read – просмотр статуса и истории.

Это позволяет точно контролировать, кто и что может делать.

3. PostgreSQL

База хранит всю историю блокировок.

Там же фиксируются:

- тип блокировки,
- статус,
- автор действия,
- время создания,
- время снятия и причина.

Использована отдельная таблица payment_hold, и предусмотрены индексы для быстрого поиска активных блокировок.

2. Ключевые принципы реализации

1. Идемпотентность

Каждая блокировка создаётся с уникальным Idempotency-Key.

Если тот же ключ придёт повторно, система вернёт бывший результат – это защищает от повторного создания блоков при сетевых ошибках.

2. Чёткое разделение типов блокировок

Используются два вида:

- мошеннические подозрения – FRAUD_SUSPECT;
- неправильные реквизиты – INCORRECT_BENEFICIARY_DETAILS.

Это позволяет различать рисковые ситуации и обычные технические.

3. Полная история изменений

Каждое создание и снятие фиксируется в БД.

Кроме основного статуса хранится и причина снятия, и кто это сделал.

3. Endpoint API

Сервис предоставляет четыре основных метода:

- POST /payment-holds – Создать блокировку
- POST /payment-holds/{holdId}:release – Снять блокировку
- GET /payment-holds:check – Проверить, заблокирован ли клиент
- GET /payment-holds – Посмотреть историю и активные блокировки

API оформлено в Swagger, чтобы можно было удобно тестировать его через браузер.

СТРУКТУРА БАЗЫ ДАННЫХ

Для хранения информации о блокировках используется PostgreSQL.

Структура была спроектирована так, чтобы:

- быстро определять, заблокирован ли клиент;
- различать типы блокировок;
- хранить полную историю действий;
- эффективно работать при большом количестве записей.

Таблица 1 – Client

Поле	Тип	Назначение
client_id	UUID (PK)	Уникальный идентификатор клиента
tin	TEXT	ИНН клиента
created_at	TIMESTAMPTZ	Время создания записи

Таблица client хранит идентификаторы клиентов, для которых могут ставиться блокировки.

Чтобы исключить ошибки и обеспечить строгие значения, используются два типа:

1. hold_type
   - FRAUD_SUSPECT – подозрение на мошенничество
   - INCORRECT_BENEFICIARY_DETAILS – неправильные реквизиты
2. hold_status
   - ACTIVE – блокировка действует
   - RELEASED – снята
   - EXPIRED – истек срок (опционально)

Таблица 2 – Payment_hold (основная)

Поле	Тип	Назначение
hold_id	UUID (PK)	ID конкретной блокировки
client_id	UUID	Клиент, для которого поставлена блокировка
type	hold_type	Тип блокировки
status	hold_status	Статус блокировки
comment	TEXT	Комментарий оператора
source	TEXT	Кто инициировал блокировку
created_at	TIMESTAMPTZ	Когда создана
created_by	TEXT	Кто создал (из JWT)
expires_at	TIMESTAMPTZ	Опциональный срок окончания блокировки
released_at	TIMESTAMPTZ	Когда снята
released_by	TEXT	Кто снял
release_reason	TEXT	Причина снятия
idempotency_key	TEXT (UNIQUE)	Ключ идемпотентности запроса

Таблица payment_hold хранит все блокировки.

Таблица 3 – Payment_hold_audit

Поле	Тип	Назначение
audit_id	BIGSERIAL	PK
hold_id	UUID	К какой блокировке относится
changed_at	TIMESTAMPTZ	Когда было изменение
changed_by	TEXT	Кто изменил
old_status	hold_status	Что было
new_status	hold_status	Что стало
note	TEXT	Комментарий

Таблица payment_hold_audit – журнал действий, который фиксирует изменения статусов.

Для быстрых запросов добавлены индексы:

- активные блокировки по клиенту
- группировка по типу
- индекс по expires_at, чтобы cron мог быстро находить истёкшие

v_client_hold_status позволяет одним запросом понять:

- заблокирован ли клиент,
- какой тип блокировки активен.

ХОД РАБОТЫ

В данном разделе представлены шаги, которые были выполнены для проверки API сервиса блокировок платежей. Каждый этап сопровождается скриншотами, демонстрирующими корректное выполнение операций.

1. Запуск backend-приложения

Бэкенд был запущен с помощью Uvicorn (см. рисунок 1):

uvicorn app:app --reload --host 0.0.0.0 --port 8000

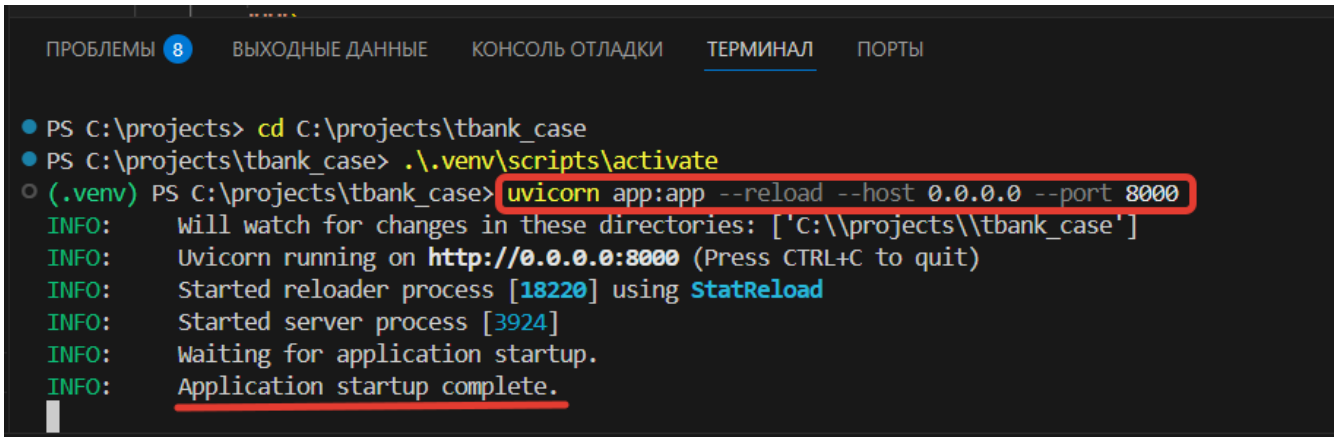

Рисунок 1 – Успешный запуск сервера

После старта сервер успешно поднялся и был доступен.

2. Открытие Swagger UI

Переходим в интерактивную документацию Swagger по адресу:

http://localhost:8000/docs

Интерфейс загрузился корректно, и все эндпоинты сервиса стали видны (см. рисунок 2).

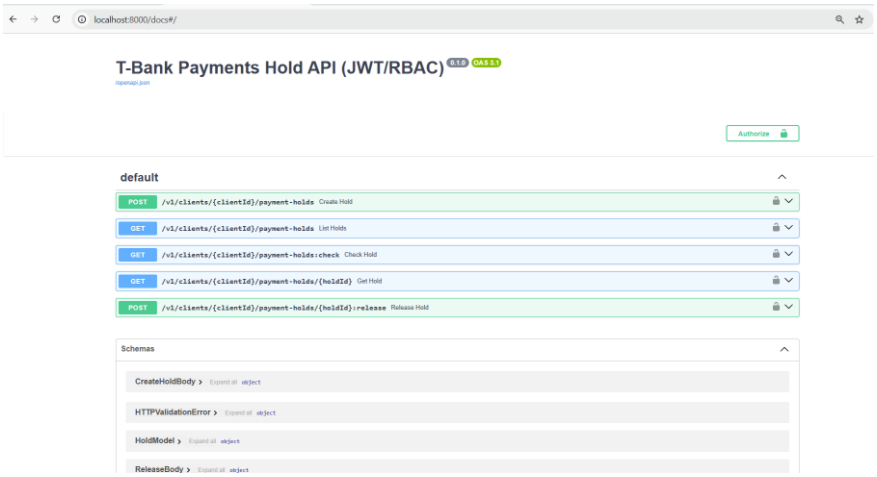

Рисунок 2 – Открытый Swagger UI

После генерируем токен в терминале с помощью команды (см. рисунок 3): *python .\scripts\jwt_gen.py user:ops1 "ops.block:read,ops.block:create,ops.block:release"*

Рисунок 3 – Генерация токена

В Swagger нажимаем Authorize, после вставляем чистый токен и закрываем (см. рисунок 4-5).

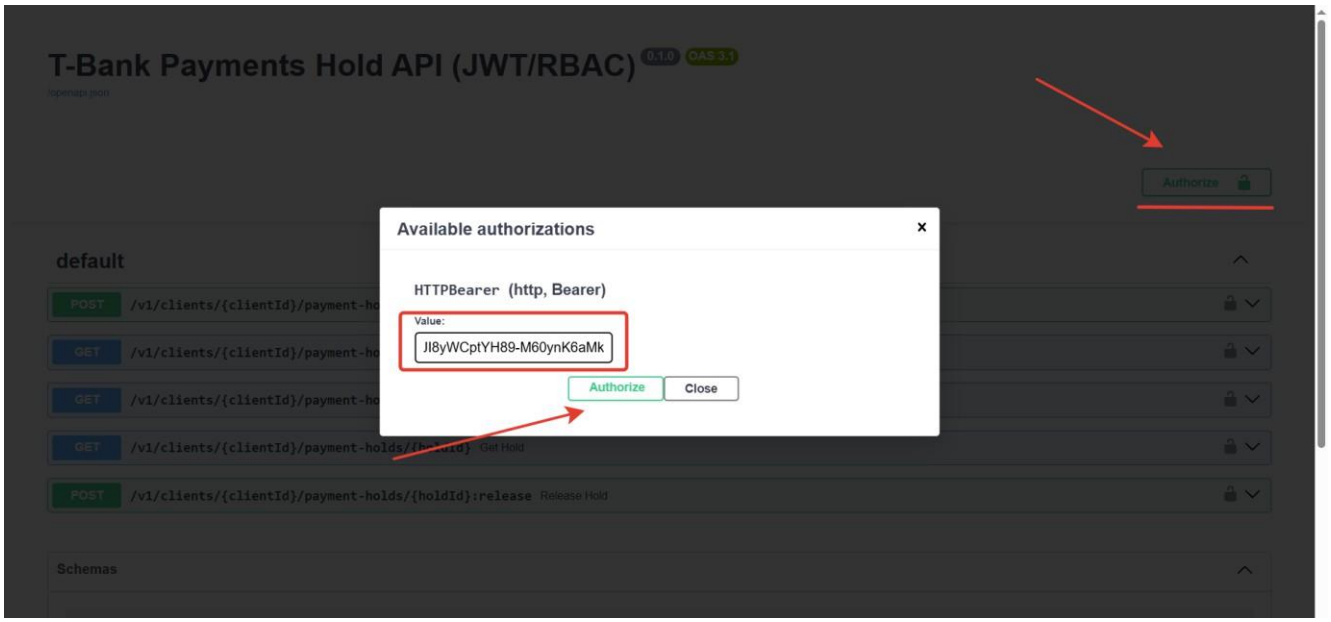

Рисунок 4 – Чистый токен

Рисунок 5 – Автоматизировано

3. Создание первой блокировки (INCORRECT_BENEFICIARY_DETAILS)

Вызван метод:

POST /v1/clients/{clientId}/payment-holds

Параметры:

clientId: 7d2b2b7a-2c0c-4f7c-8a84-2f4a3f686e55

Idempotency-Key: любой UUID (был сгенерирован с помощью команды ы терминале *[guid]::NewGuid()* (см. рисунок 6).

Body:

{

"type": "INCORRECT_BENEFICIARY_DETAILS",

"comment": "incorrect details",

"source": "ops",

"expiresAt": "2025-12-01T00:00:00Z"

}

Сервер вернул статус 201 CREATED, блокировка создана (см. рисунок 8).

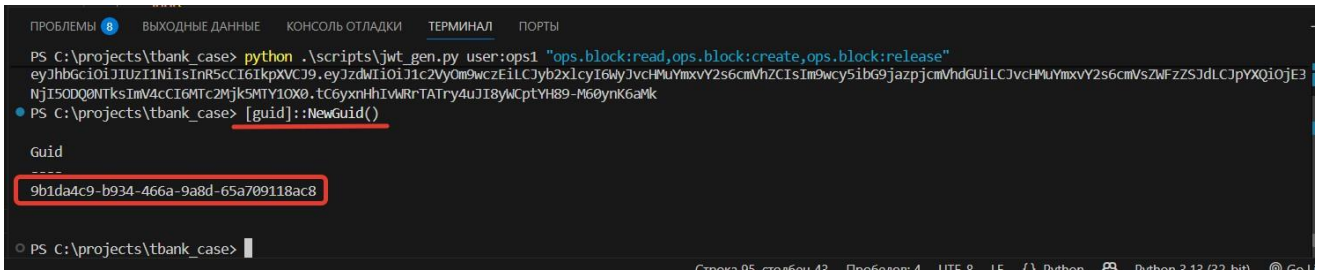

Рисунок 6 – Создание уникального UUID

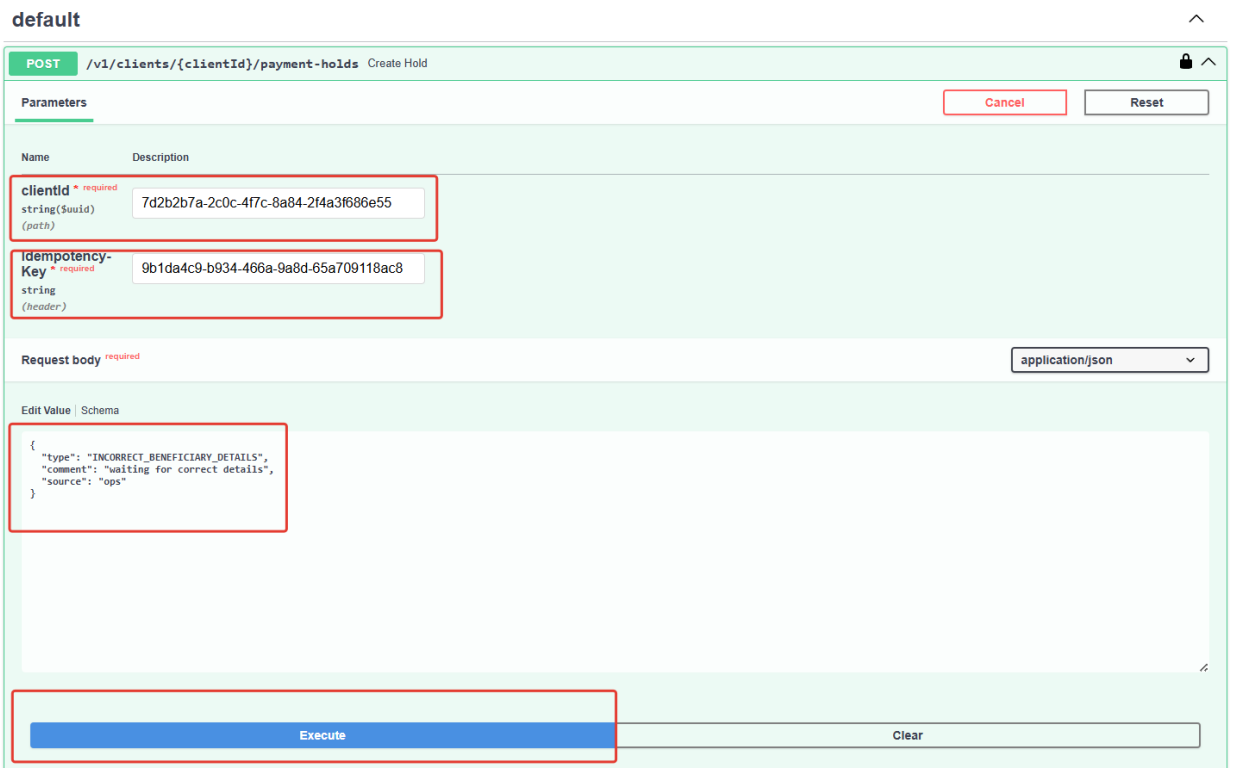

Рисунок 7 – Создание первой блокировки

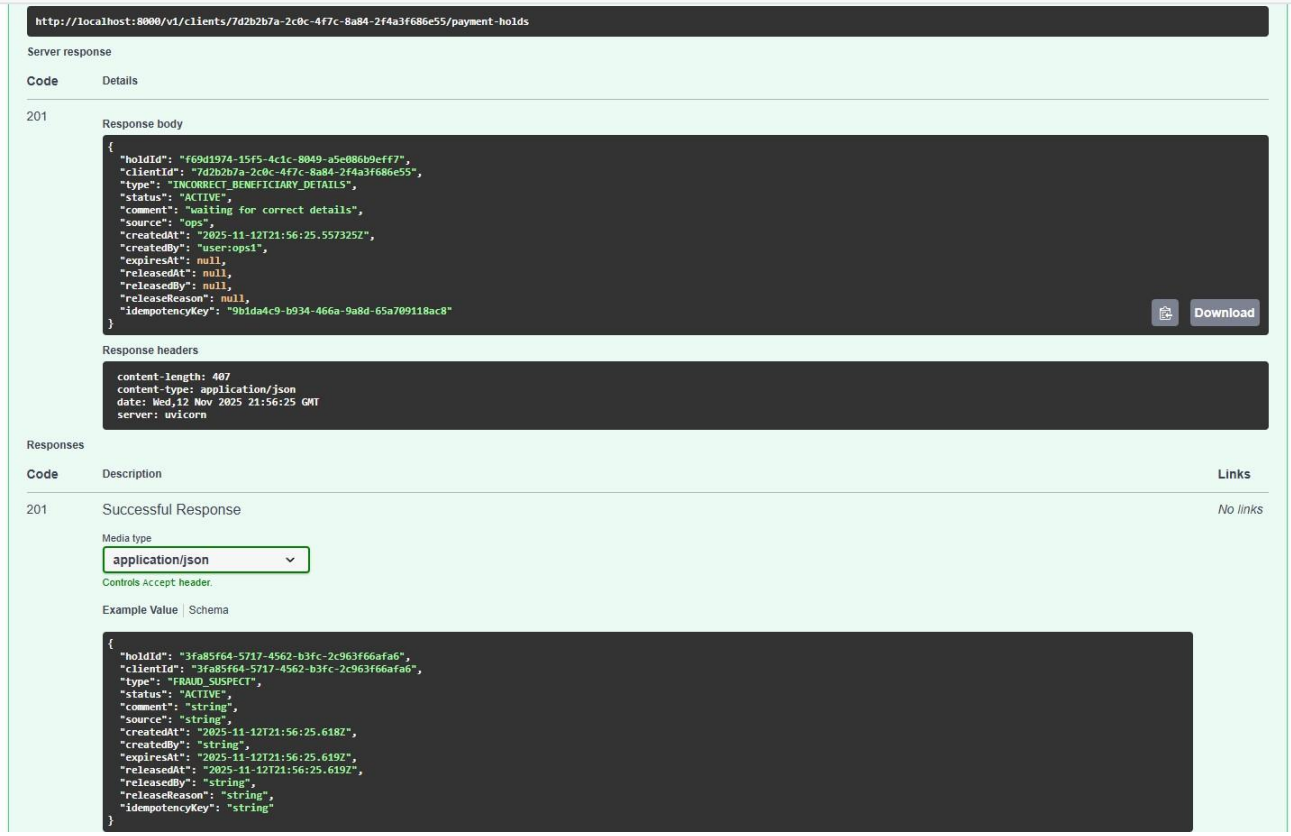

Рисунок 8 – Успешное создание первой блокировки

4. Создание второй блокировки (FRAUD_SUSPECT)

Вновь вызван тот же метод, но с другим типом:

{

"type": "FRAUD_SUSPECT",

"comment": "fraud suspicion",

"source": "ops",

"expiresAt": "2025-12-01T00:00:00Z"

}

Блокировка успешно добавлена (см. рисунок 9).

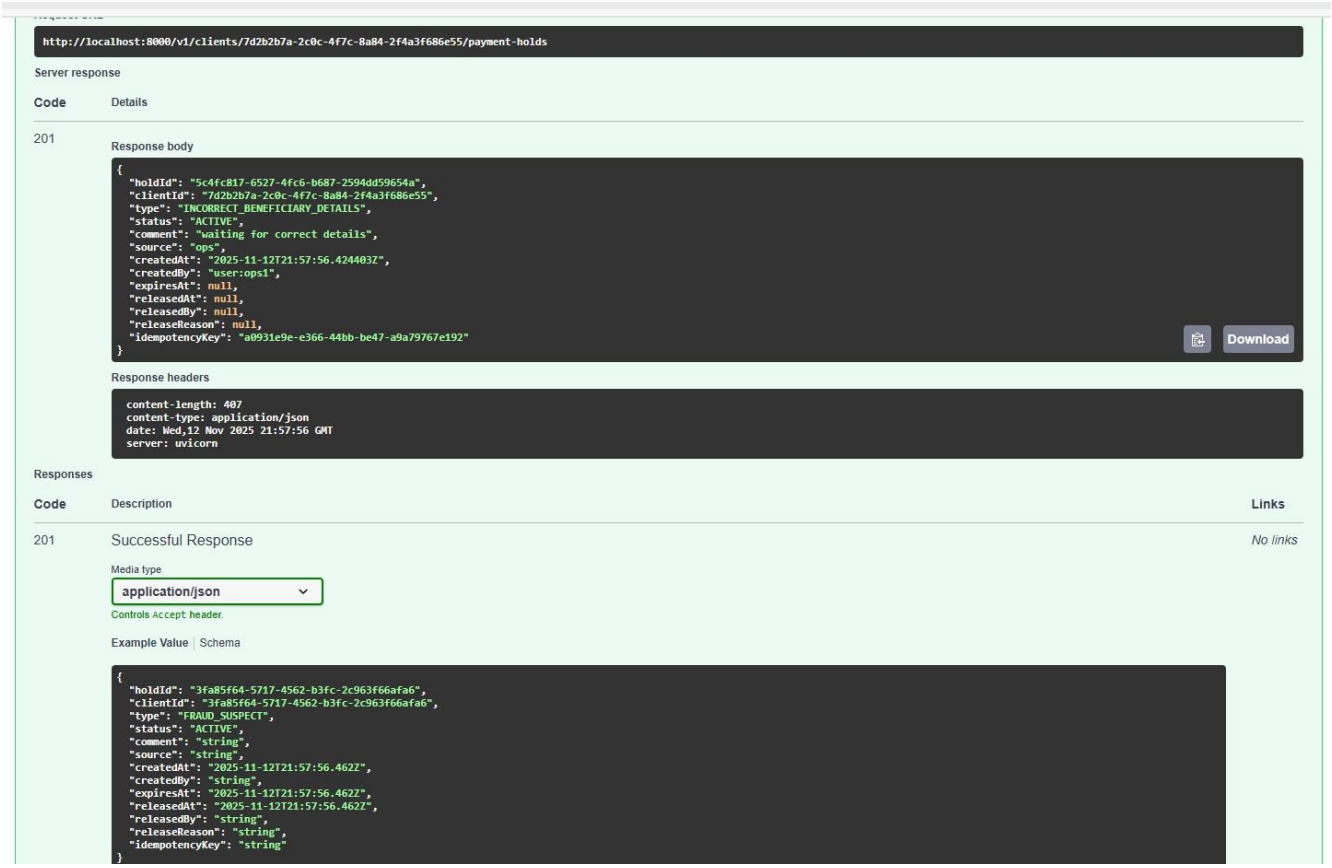

Рисунок 9 – Успешное создание второй блокировки

5. Проверка статуса блокировок клиентов

Вызван метод:

GET /v1/clients/{clientId}/payment-holds:check

Ответ (см. рисунок 10):

blocked: true

список активных блокировок содержит обе записи

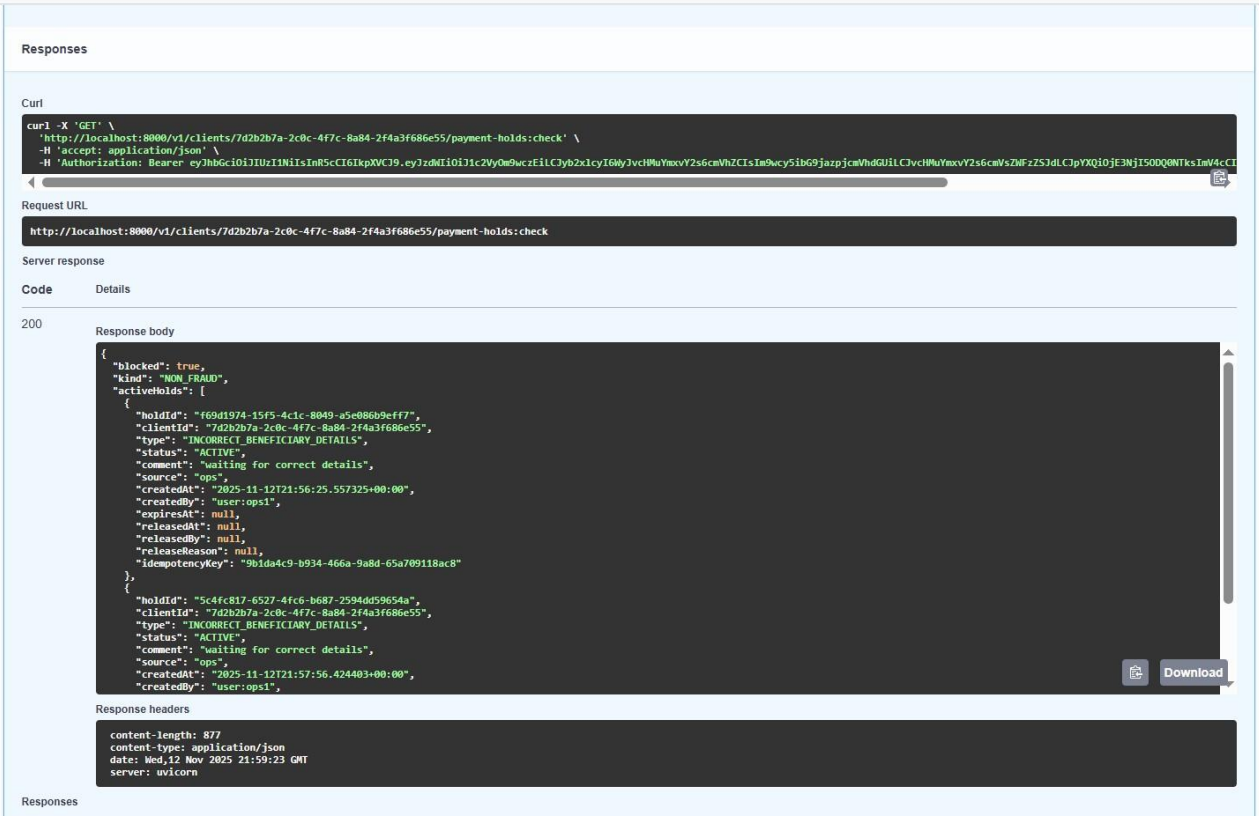

Рисунок 10 – Успешная проверка статуса: клиент заблокирован

6. Снятие блокировки INCORRECT_BENEFICIARY_DETAILS

Вызван метод:

POST /v1/clients/{clientId}/payment-holds/{holdId}:release

Body:

{

"reason": "resolved",

"comment": "manual release"

}

Статус изменён на RELEASED (см. рисунок 11).

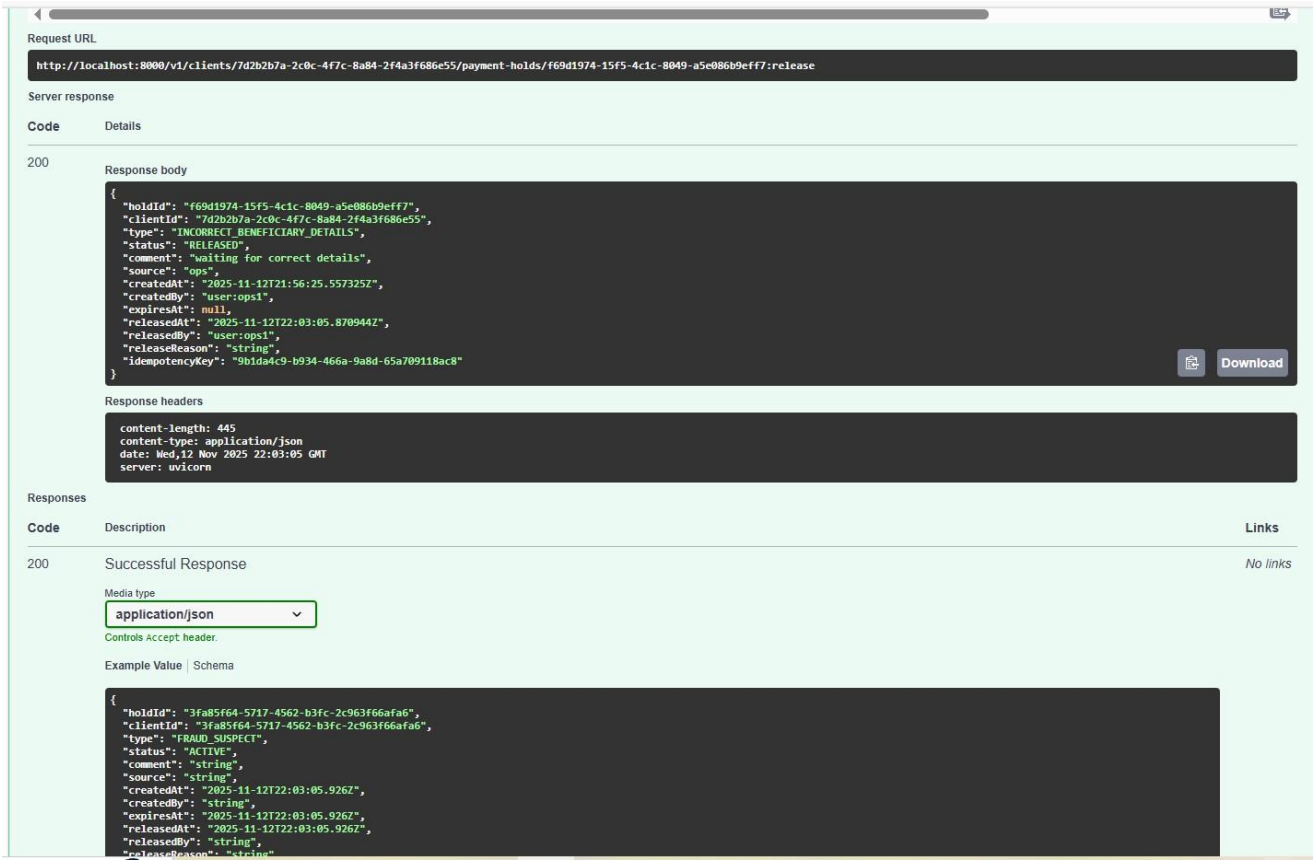

Рисунок 11 – Успешное снятие блокировки

7. Снятие блокировки FRAUD_SUSPECT

Аналогично предыдущему шагу:

{

"reason": "resolved",

"comment": "manual release"

}

Блокировка успешно снята (см. рисунок 12).

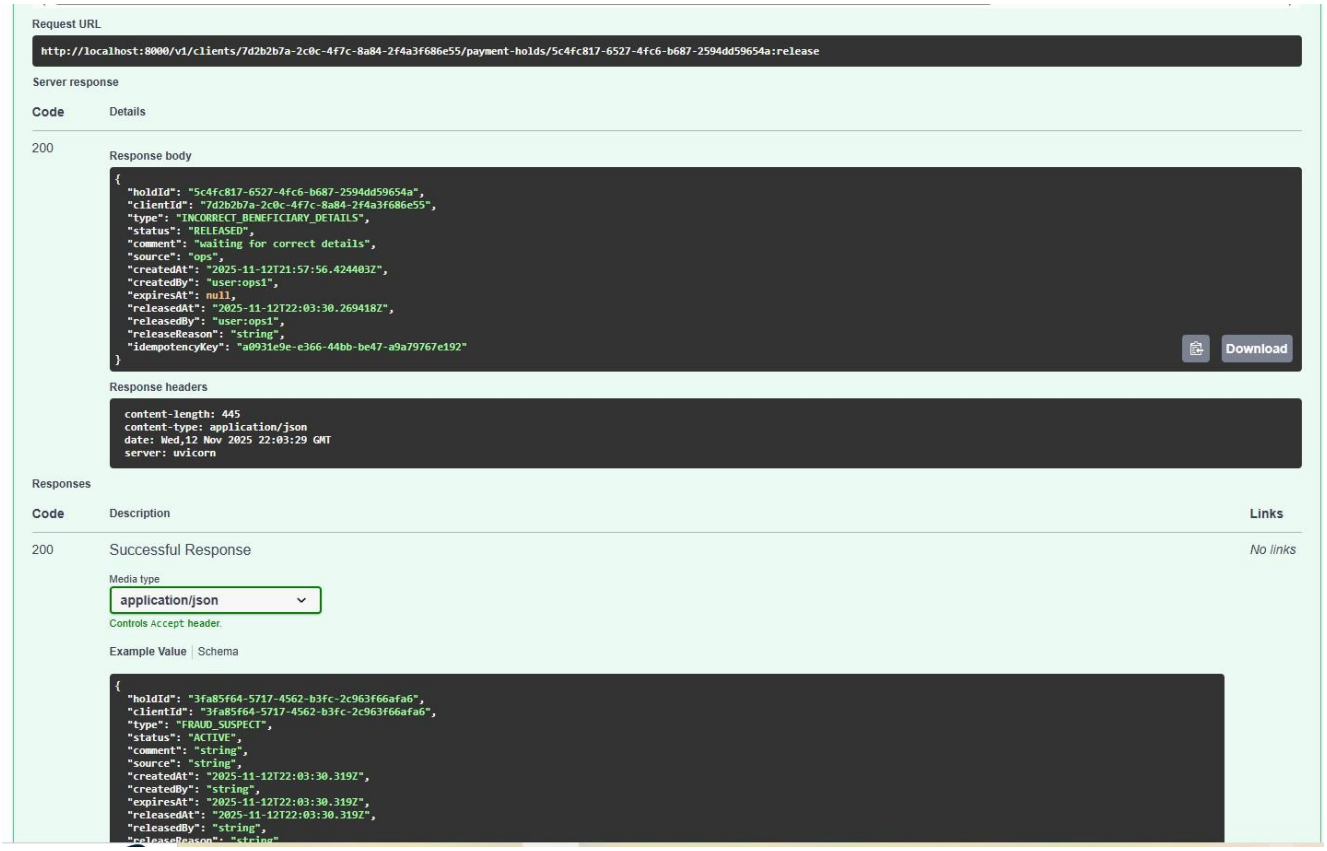

Рисунок 12 – Успешное снятие блокировки

8. Финальная проверка

Повторный вызов:

GET /v1/clients/{clientId}/payment-holds:check

Система вернула (см. рисунок 13):

blocked: false

kind: "NONE"

активных блокировок нет

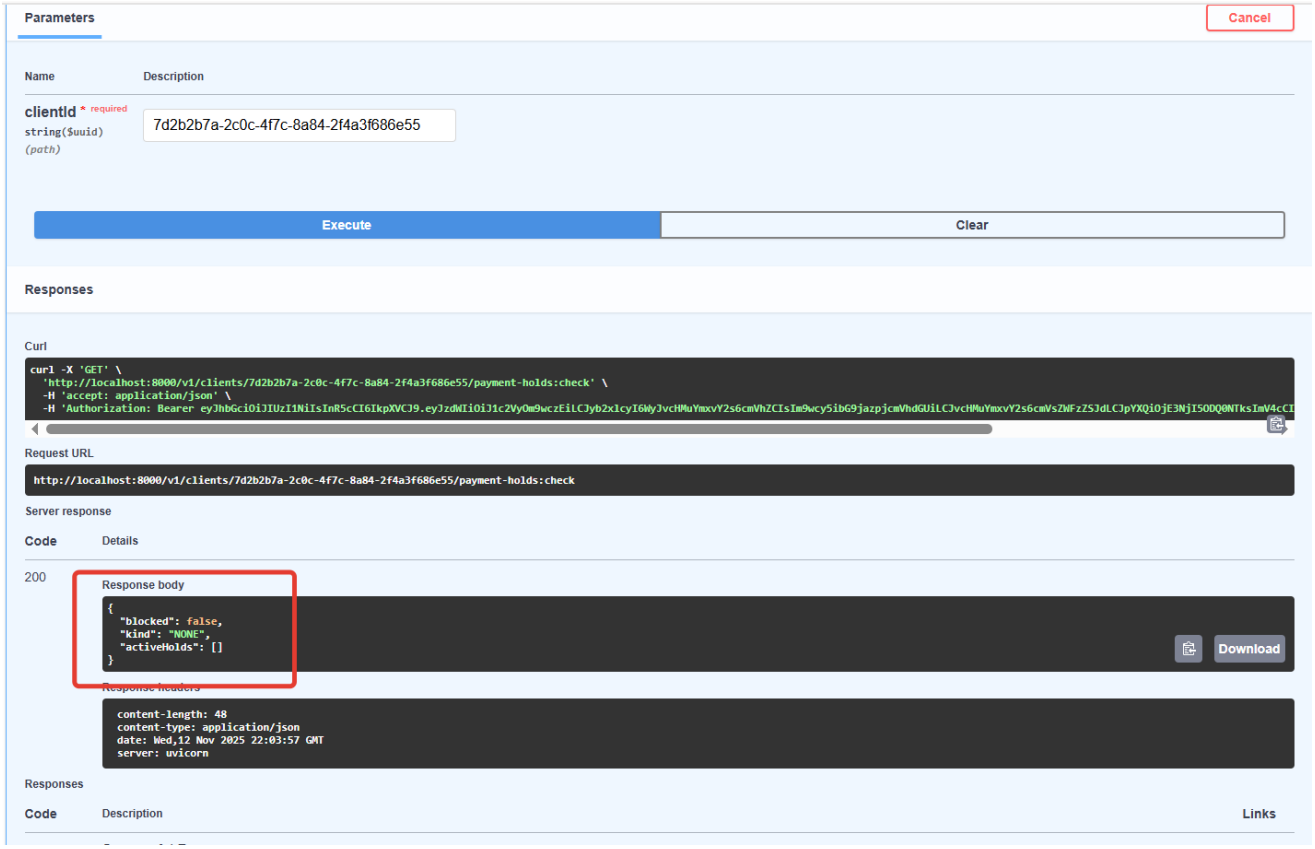

Рисунок 13 – Финальная проверка: клиент успешно разблокирован

ЗАКЛЮЧЕНИЕ

В ходе выполнения работы была реализована и проверена функциональность сервиса блокировок платежей для клиентов юридических лиц. Полученное решение позволяет:

1. накладывать блокировки разных типов (мошенничество и некорректные реквизиты);

2. снимать ранее установленные блокировки

3. просматривать историю блокировок клиента;

4. автоматически определять, является ли клиент заблокированным в данный момент;

45. корректно различать виды ограничений (FRAUD / NON_FRAUD).

Был разработан и протестирован полный набор REST-эндпоинтов, соответствующих требованиям бизнеса. Реализация поддерживает идемпотентность, строгую валидацию данных, ролевую модель доступа (RBAC), хранение данных в PostgreSQL и удобное взаимодействие через Swagger UI.

Все тесты, выполненные через Swagger, показали корректную работу сервиса: блокировки создавались, отображались, классифицировались и успешно снимались.

Поставленная задача была полностью решена. Полученное решение можно использовать как основу для промышленного сервиса блокировки платежей в корпоративных системах банка.