ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ПЛАТФОРМЫ «WEALTHIFY»

Версия документа: 1.0
Дата: 04.11.2025
Проект: Wealthify — платформа аналитики инвестиционных портфелей

1. ВВЕДЕНИЕ

1.1. Цель документа

Цель настоящего документа — установить и зафиксировать однозначные требования к платформе Wealthify, чтобы:

– синхронизировать ожидания внутри команды разработки;
– использовать документ как основу для проектирования, реализации, тестирования и приёмки;
– минимизировать двусмысленности на всех этапах жизненного цикла продукта.

1.2. Область применения системы

Wealthify — это клиент-серверная веб-платформа, которая:

– помогает пользователю вести учёт инвестиционных портфелей (криптовалюты, акции, облигации, кэш);
– показывает состав и динамику портфеля;
– в дальнейшем (roadmap) предоставляет риск-профиль и ИИ-подсказки на естественном языке;
– отправляет алерты и дайджесты по ключевым событиям портфеля.

Доступ к системе осуществляется через веб-клиент (Next.js) и REST API, реализованный поверх микросервисной архитектуры (NestJS + PostgreSQL).

1.3. Ссылки

– Стандарты формулировки требований: ISO/IEC/IEEE 29148:2018.

2. ОБЩАЯ ХАРАКТЕРИСТИКА СИСТЕМЫ

2.1. Перспектива и архитектура

Система реализована как микросервисное приложение на Node.js / NestJS:

– Веб-клиент (Next.js 15 / React 19 / TypeScript).
Рендерит пользовательский интерфейс, выполняет аутентификацию через API-шлюз, отображает портфели, активы, транзакции, графики.

– API-шлюз (приложение apps/api-gateway).
– HTTP(S) REST API для фронтенда;
– авторизация по JWT (access-token) и refresh-токену в HttpOnly-cookie;
– проксирование запросов во внутренние микросервисы по TCP-транспорту NestJS;
– документация через Swagger (аннотации @nestjs/swagger в контроллерах).

– Доменный сервис «App» (приложение apps/app).
– функционал: пользователи, роли, портфели, активы, транзакции, refresh-/reset-токены;
– хранение данных в PostgreSQL через sequelize-typescript;
– валидация данных и контракты — через библиотеку libs/contracts (Zod + nestjs-zod).

– Сервис сбора рыночных данных «Crypto Data Worker» (приложение apps/crypto-data-worker).
– парсинг и сбор данных по криптовалютам (на текущий момент — BTC);
– хранение исторических свечей и снапшотов в PostgreSQL;
– взаимодействие с внешними сайтами через puppeteer-extra и stealth-плагин.

Упрощённая схема:

Web (Next.js 15 / React 19 / TypeScript)
→ HTTP(S) REST API
→ API-Gateway (NestJS 11 + Swagger)
├─ RPC → App (users, roles, portfolios, assets, transactions, auth)
└─ RPC → Crypto-Data-Worker (crypto_assets, crypto_candles)

DB: PostgreSQL
├─ Таблицы домена (users, roles, portfolios, assets, transactions и др.)
└─ Таблицы маркет-данных (crypto_assets, crypto_candles)

2.2. Функции системы (в целом)

Система должна обеспечивать:

1. Регистрацию и аутентификацию пользователей по email и паролю.
2. Управление ролями и правами доступа (USER, ADMIN и др.).
3. Создание и ведение инвестиционных портфелей разных типов (Crypto / Stock / Bond).
4. Справочник активов (Crypto, Stock, Bond, Fiat) с тикерами.
5. Добавление и продажу активов в портфеле с автоматическим учётом количества и средней цены покупки.
6. Ведение журнала транзакций (покупка/продажа).
7. Сбор и хранение рыночных данных по криптовалютам (на старте BTC).

8. В дальнейшем (roadmap) — расчёт риск-профиля, ИИ-подсказки, алерты и дайджесты на основе данных портфеля.

2.3. Пользователи и заинтересованные стороны

– Частные инвесторы-новички.
– Частные инвесторы с опытом.
– Администраторы (управление ролями, справочниками).
– Команда разработки и аналитики (развитие продукта).

2.4. Технологический стек

Frontend (wealthify-web):

– Next.js 15.5.4, React 19.1.0, TypeScript 5.9.2;
– формы: react-hook-form 7.63 + Zod 4.1 (@hookform/resolvers 5.2);
– графики: Recharts 3.2;
– стили: CSS Modules / Tailwind (опционально по проекту);
– линтинг: ESLint 9 (eslint-config-next 15.5.4).

Backend (микросервисы, NestJS):

– NestJS 11 (@nestjs/* 11.x), TypeScript 5.7;
– база данных: PostgreSQL + Sequelize 6.37 / sequelize-typescript 2.1;
– аутентификация: JWT (@nestjs/jwt, JwtAuthGuard), refresh-токены в таблице refresh-token;
– крипто-воркер: puppeteer-extra + stealth-plugin;
– валидация: Zod (nestjs-zod DTO), строгий режим .strict();
– документация: @nestjs/swagger 11.2 + Swagger UI;
– прочее: cookie-parser, uuid, rxjs;
– качество: ESLint 9, Prettier 3;
– тесты: Jest 29, ts-jest, supertest.

Инфраструктура:

– Node.js LTS;
– Docker / Docker Compose (по необходимости);
– DBeaver или аналог для инспекции БД.

3. ПРЕДПОЛОЖЕНИЯ, ЗАВИСИМОСТИ И ОГРАНИЧЕНИЯ

3.1. Предполагается, что

– пользователь работает в современном браузере (Chrome, Firefox, Edge);
– серверная часть развернута на отдельном сервере или в контейнерах и доступна по HTTPS;

– внешние источники данных (сайты с криптовалютными котировками) доступны воркеру;
– секреты (JWT, SMTP, пароли к БД) хранятся в переменных окружения или в системе управления секретами;
– денежные значения для расчётов хранятся в долларах США (USD) и при необходимости пересчитываются в интерфейсе.

3.2. Ограничения

– Платформа не является инвестиционной рекомендацией, а носит информационный характер.
– На старте поддерживаются базовые классы активов: Crypto, Stock, Bond, Fiat (для кэша).
– Риск-профиль, ИИ-подсказки и алерты реализуются поэтапно (roadmap, не в первом MVP).
– Объём исторических данных по криптовалютам на старте ограничен активом BTC и выбранным набором интервалов (например, 1h, 4h, 1d и т.п.).

4. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Ниже функциональные требования описаны с привязкой к реализованным контроллерам и моделям проекта.

4.1. Аутентификация и управление учётными записями

4.1.1. Регистрация по email и паролю

– Эндпоинт: POST /auth/registration.
– Входные данные: CreateUserDto (username, email, password).
– Требования к паролю: длина 12–72 символа, латиница, цифры, ограниченный набор допустимых спецсимволов.
– Выходные данные:
– HTTP 201 при успехе;
– access-token в заголовке Authorization: Bearer ;
– refresh-token в cookie HttpOnly (например, REFRESH_TOKEN_COOKIE);
– объект пользователя { user }.

4.1.2. Логин

– Эндпоинт: POST /auth/login.
– Входные данные: LoginDto (email, password).
– Логика: проверка email и пароля; при успехе выдаётся пара access/refresh-токенов по той же схеме, что и при регистрации.
– Ошибки:
– 401 — неверный email или пароль.

4.1.3. Обновление токенов (refresh)

– Эндпоинт: POST /auth/refresh.
– Входные данные: refresh-токен из cookie.
– Логика:
– проверка существования и непросроченности записи в таблице refresh-token;
– сравнение plain-refresh-токена с хешем с помощью bcrypt;
– выпуск нового access-токена (время жизни 15 минут) и нового refresh-токена (время жизни 3 дня).
– Выходные данные: новый access-токен (в заголовке) и обновлённый refresh-токен в cookie.

4.1.4. Выход из системы

– Эндпоинт: POST /auth/logout.
– Входные данные: refresh-токен в cookie (если присутствует).
– Логика:
– удаление или инвалидирование записи в таблице refresh-token для пользователя;
– очистка cookie на стороне клиента.
– Выходные данные: объект { message: "ok" }.

4.1.5. Смена пароля

– Эндпоинт: PUT /auth/change-password.
– Guard: JwtAuthGuard.
– Входные данные: ChangePasswordDto (oldPassword, newPassword).
– Логика:
– определение идентификатора пользователя (userId) из JWT (req.userId или req.user.id);
– проверка старого пароля через bcrypt;
– установка нового пароля (хеш bcrypt).
– Выходные данные: подтверждение успешной смены пароля.

4.1.6. Восстановление пароля (забытый пароль)

– Эндпоинт: POST /auth/forgot-password.
– Входные данные: ForgotPasswordDto (email).
– Логика:
– если пользователь существует, генерируется UUID reset-токен (время жизни 1 час), токен хешируется и сохраняется в таблице reset-token;
– пользователю отправляется письмо со ссылкой или токеном для сброса пароля.
– Выходные данные: обобщённый ответ без раскрытия факта существования или отсутствия пользователя.

4.1.7. Сброс пароля по токену

– Эндпоинт: PUT /auth/reset-password.
– Входные данные: ResetPasswordDto (resetToken, newPassword, userId).
– Логика:
– поиск записи в таблице reset-token по userId и неистёкшему полю expiryDate;
– проверка reset-токена с помощью bcrypt;
– при успешной проверке — обновление пароля пользователя и удаление/инвалидирование записи с токеном.
– Ошибки:
– 401 — неверная или истёкшая ссылка;
– 500 — пользователь не найден.

4.2. Пользователи и роли

4.2.1. Создание пользователя

– Эндпоинт: POST /users.
– Входные данные: CreateUserDto.
– Логика:
– создание пользователя;
– назначение роли USER по умолчанию.
– Ошибки:
– 400 — пользователь с таким email уже существует;
– 404 — роль USER не найдена.

4.2.2. Получение списка пользователей

– Эндпоинт: GET /users.
– Доступ: только роль ADMIN (JwtAuthGuard + RolesGuard + @Roles("ADMIN")).
– Выходные данные: массив пользователей.

4.2.3. Управление ролями

– Создание роли:
– эндпоинт: POST /roles, доступ только ADMIN;
– входные данные: CreateRoleDto (value, description).

– Получение роли:
– эндпоинт: GET /roles/:value, доступ только ADMIN.

– Список ролей:
– эндпоинт: GET /roles, доступ только ADMIN.

– Удаление роли:
– эндпоинт: DELETE /roles/:value, доступ только ADMIN.

4.2.4. Назначение роли пользователю

– Эндпоинт: POST /users/roles.
– Доступ: ADMIN.
– Входные данные: AddRoleDto (userId, value).
– Логика:
– проверка существования пользователя и роли;
– добавление записи в таблицу user_roles.
– Ошибки:
– 404 — роль или пользователь не найдены;
– 400 — у пользователя уже есть такая роль.

4.3. Портфели

4.3.1. Создание портфеля

– Эндпоинт: POST /portfolios.
– Guard: JwtAuthGuard.
– Входные данные: CreatePortfolioDto (name, type: PortfolioType, userId).
– Логика:
– создание записи в таблице portfolios с внешним ключом на users.
– Выходные данные: созданный портфель.
– Ошибки:
– 400 — ошибка валидации.

4.3.2. Получение всех портфелей пользователя

– Эндпоинт: GET /portfolios/user/:id.
– Guard: JwtAuthGuard.
– Входные данные: id — идентификатор пользователя.
– Выходные данные: массив портфелей указанного пользователя.

4.3.3. Получение портфеля по названию

– Эндпоинт: GET /portfolios/name/:name.
– Guard: JwtAuthGuard.
– Входные данные: name — название портфеля.
– Выходные данные: портфель или ошибка/сообщение о его отсутствии.

4.3.4. Удаление портфеля

– Эндпоинт: DELETE /portfolios/:id.
– Guard: JwtAuthGuard.

– Логика:
– удаление записи из таблицы portfolios по идентификатору id;
– каскадная логика по транзакциям и связям с активами определяется на уровне сервисов и/или базы данных.
– Ошибки:
– 404 — портфель не найден.

4.4. Активы и операции с ними

4.4.1. Создание актива (справочник активов)

– Эндпоинт: POST /assets.
– Доступ: только ADMIN.
– Входные данные: CreateAssetDto (name, ticker, type: AssetType).
– Логика:
– создание записи в таблице assets.
– Ошибки:
– 400 — актив с таким тикером уже существует.

4.4.2. Получение актива по тикеру

– Эндпоинт: GET /assets/:ticker.
– Guard: JwtAuthGuard.
– Выходные данные: данные актива или 404 при отсутствии.

4.4.3. Добавление актива в портфель

– Эндпоинт: POST /assets/add-to-portfolio.
– Guard: JwtAuthGuard.
– Входные данные: AddAssetToPortfolioDto (portfolioId, assetTicker, quantity, purchasePrice).
– Логика:
– проверка существования портфеля и актива;
– создание или обновление записи в таблице portfolio_assets:
– увеличение количества;
– перерасчёт средней цены покупки;
– создание транзакции типа BUY в таблице transactions.
– Ошибки:
– 404 — актив или портфель не найдены.

4.4.4. Продажа актива из портфеля

– Эндпоинт: PATCH /assets.
– Guard: JwtAuthGuard.
– Входные данные: SellAssetDto (portfolioId, assetTicker, quantity, pricePerUnit, параметр convertToUsd — опционально).

– Логика:
– проверка наличия достаточного количества актива в таблице portfolio_assets;
– уменьшение количества, при нулевом остатке — удаление связи;
– создание транзакции типа SELL в таблице transactions;
– при convertToUsd = true — добавление или корректировка позиции кэша (Fiat/ USD) в том же портфеле.
– Ошибки:
– 400 — неверное количество или цена;
– 404 — актив или портфель не найдены.

4.4.5. Удаление актива из портфеля

– Эндпоинт: DELETE /assets/remove-from-portfolio.
– Guard: JwtAuthGuard.
– Входные данные: RemoveAssetFromPortfolioDto (portfolioId, assetTicker, removeAllLinkedTransactions).
– Логика:
– удаление записи из таблицы portfolio_assets;
– при removeAllLinkedTransactions = true — удаление связанных транзакций по данному активу и портфелю.

4.4.6. Удаление актива (из справочника)

– Эндпоинт: DELETE /assets/:ticker.
– Доступ: ADMIN.
– Логика:
– удаление записи в таблице assets по тикеру;
– удаление связанных записей в таблице portfolio_assets и, при необходимости, транзакций.

4.5. Транзакции

4.5.1. Получение всех транзакций (административный доступ)

– Эндпоинт: GET /transactions.
– Доступ: ADMIN (JwtAuthGuard + RolesGuard + @Roles("ADMIN")).
– Выходные данные: список всех транзакций.

4.5.2. Получение транзакций по портфелю

– Эндпоинт: GET /transactions/:id.
– Guard: JwtAuthGuard.
– Входные данные: id — идентификатор портфеля.
– Выходные данные: массив транзакций указанного портфеля.

4.5.3. Удаление транзакции

– Эндпоинт: DELETE /transactions/:id.
– Guard: JwtAuthGuard.
– Логика:
– удаление записи из таблицы transactions;
– пересчёт агрегированных данных в таблице portfolio_assets (количество, средняя цена).

4.6. Crypto Data Worker

4.6.1. Проверка состояния воркера

– Эндпоинт: GET /crypto-data-worker/health.
– Выходные данные: объект вида { status: "ok" } или аналогичный.

4.6.2. Единичный сбор данных

– Эндпоинт: POST /crypto-data-worker/collect.
– Логика:
– запуск сервиса crypto-data-worker на парсинг данных с целевого ресурса (например, CoinMarketCap или Coingecko);
– обновление таблиц crypto_assets и crypto_candles.

4.7. Функции roadmap (ИИ-подсказки, риск-профиль, алерты)

В рамках последующих этапов разработки предусматривается:

– формирование риск-профиля пользователя на основе состава и волатильности активов в портфеле;
– генерация текстовых объяснений на естественном языке (NLG) по структуре и динамике портфеля;
– еженедельные дайджесты и алерты по событиям (существенное изменение цены, новый максимум/минимум, рекомендации по ребалансировке и т.п.).

5. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

5.1. Производительность

– Система должна выдерживать до 200 одновременных активных сессий.
– Время ответа API-шлюза для простых запросов (например, GET /portfolios/user/:id) — не более 300 мс при 95-м перцентиле (P95).
– Операции, затрагивающие базу данных (создание транзакции, перерасчёт портфеля), должны выполняться не более 800 мс при P95.

5.2. Надёжность

– Средняя доступность сервиса должна быть не ниже 99 % в месяц (в пределах доступных ресурсов инфраструктуры).
– При недоступности внутренних микросервисов API-шлюз должен возвращать осмысленные коды ошибок (502, 504) и сообщения.

5.3. Безопасность

– Пароли хранятся только в виде хешей, сформированных с помощью bcryptjs.
– Access-токены:
– формат: JWT;
– время жизни: 15 минут (expiresIn: "15m").
– Refresh-токены:
– генерация: UUID4;
– хранение: хеш токена, userId и expiryDate в таблице refresh-token;
– время жизни: 3 дня;
– передача клиенту: HttpOnly-cookie (SameSite = Lax, Secure = true в продуктивной среде).
– Все входные DTO валидируются с помощью Zod (.strict()), лишние поля запрещены.
– Административные операции защищены связкой JwtAuthGuard + RolesGuard и аннотацией @Roles("ADMIN").

5.4. Масштабируемость

– Архитектура должна позволять горизонтальное масштабирование API-шлюза и доменных сервисов.
– Добавление новых типов активов, портфелей и источников маркет-данных не должно требовать переработки общей архитектуры.

5.5. Поддерживаемость

– Единые DTO и паттерны запросов определяются в библиотеке libs/contracts.
– Swagger-документация генерируется автоматически из аннотаций контроллеров.
– Ошибки в RPC-слое стандартизируются (структура вида { status, code, message }).

5.6. Совместимость

– Поддерживаемые браузеры: Google Chrome, Mozilla Firefox, Microsoft Edge (актуальные версии).

6. ИНТЕРФЕЙСЫ

6.1. Пользовательский интерфейс (Web)

Минимальный набор экранов:

6.1.1. Страница авторизации

– Поля: email, пароль.
– Валидация:
– отображение ошибки при некорректном email;
– отображение ошибки при неверном пароле.
– Возможность показать/скрыть пароль.
– Кнопка «Войти»: при успешной аутентификации осуществляется переход на дашборд портфелей.
– Ссылка «Нет аккаунта? Зарегистрируйтесь» ведёт на страницу регистрации.
– Ссылка «Забыли пароль?» ведёт на форму восстановления пароля.

6.1.2. Страница регистрации

– Поля: username, email, пароль (с соблюдением политики сложности), при необходимости подтверждение пароля.
– При успешной регистрации пользователь автоматически авторизуется и перенаправляется на дашборд портфелей.
– Отображение ошибок:
– существующий email;
– некорректный формат вводимых данных.
– Ссылка «Уже есть аккаунт? Войдите» ведёт на страницу авторизации.

6.1.3. Страница восстановления пароля

– Шаг 1: форма ввода email, отображается сообщение вида «Если пользователь существует, он получит письмо для восстановления пароля».
– Шаг 2: экран ввода нового пароля по ссылке из письма (resetToken + userId). При успешном сбросе пароля — переход на страницу авторизации.

6.1.4. Дашборд портфелей

– Список портфелей пользователя:
– название портфеля;
– тип (Crypto/Stock/Bond);
– текущая стоимость портфеля (на начальном этапе может быть рассчитана упрощённым образом или реализована как заглушка);
– краткая статистика (количество активов).
– Кнопка «Создать портфель» с формой (name, type).
– Переход к странице конкретного портфеля по клику на элемент списка.
– В перспективе (roadmap) — блок кратких ИИ-подсказок и оценка риска.

6.1.5. Страница портфеля

– Таблица активов в портфеле:
– тикер, название;
– количество, средняя цена покупки;
– при наличии данных от воркера — текущая цена и прибыль/убыток.
– История транзакций портфеля (отдельной секцией или вкладкой).
– Действия:
– «Добавить актив» (форма для AddAssetToPortfolioDto);
– «Продать актив» (форма для SellAssetDto, включая параметр convertToUsd);
– «Удалить актив из портфеля» (форма для RemoveAssetFromPortfolioDto).
– Графики:
– распределение портфеля по активам (Recharts);
– динамика стоимости портфеля (по мере готовности функционала).

6.1.6. Административная панель (для роли ADMIN)

– Управление пользователями:
– список, поиск;
– назначение ролей через вызов POST /users/roles.
– Управление ролями:
– создание и удаление ролей.
– Управление справочником активов:
– создание актива;
– удаление актива по тикеру.

6.2. Программные интерфейсы (REST API)

REST API предоставляется API-шлюзом (приложение apps/api-gateway) по протоколу HTTPS и использует формат данных JSON.

Основные группы эндпоинтов:

– Auth: /auth/...
– Users: /users/...
– Roles: /roles/...
– Portfolios: /portfolios/...
– Assets: /assets/...
– Transactions: /transactions/...
– Crypto Data Worker: /crypto-data-worker/...

Каждый эндпоинт:

– принимает и возвращает данные в формате JSON;
– использует коды статуса HTTP согласно аннотациям @ApiResponse;
– описывается в Swagger с помощью @ApiOperation, @ApiParam, @ApiBody, @ApiResponse.

7. АРХИТЕКТУРА И ВНУТРЕННИЕ ПРОГРАММНЫЕ ИНТЕРФЕЙСЫ (RPC)

Взаимодействие API-шлюза с микросервисами реализовано через NestJS microservices по TCP-транспорту.

Паттерны сообщений (описаны в файлах libs/contracts/src/**.pattern.ts):

– Auth (AUTH_PATTERNS):
– auth.login;
– auth.registration;
– auth.refresh;
– auth.logout;
– auth.changePassword;
– auth.forgotPassword;
– auth.resetPassword.

– Users (USERS_PATTERNS):
– users.create;
– users.findAll;
– users.addRole.

– Roles (ROLES_PATTERNS):
– roles.create;
– roles.findAll;
– roles.findByValue;
– roles.deleteByValue.

– Portfolios (PORTFOLIOS_PATTERNS):
– portfolios.create;
– portfolios.findAllByUser;
– portfolios.findByName;
– portfolios.deleteById.

– Assets (ASSETS_PATTERNS):
– assets.create;
– assets.getByTicker;
– assets.deleteByTicker;
– assets.addToPortfolio;
– assets.sellFromPortfolio;
– assets.removeFromPortfolio.

– Transactions (TRANSACTIONS_PATTERNS):
– transactions.findAll;
– transactions.findAllByPortfolio;
– transactions.deleteById.

– Crypto Data Worker (CRYPTO_DATA_WORKER_PATTERS):
– cryptoDataWorker.health;
– cryptoDataWorker.collect.

Ошибки в RPC-слое оформляются через RpcException с полезной нагрузкой вида { status, code, message } и отображаются на HTTP-уровень в виде соответствующих кодов и сообщений.

8. ХРАНЕНИЕ ДАННЫХ В БАЗЕ ДАННЫХ

База данных — PostgreSQL. Структура таблиц основана на моделях sequelize-typescript из приложений apps/app и apps/crypto-data-worker.

8.1. Основные таблицы домена

8.1.1. Таблица users

Назначение: хранение зарегистрированных пользователей.

Основные поля:

– id — первичный ключ;
– username — уникальное имя пользователя;
– email — уникальный адрес электронной почты;
– password — хеш пароля (bcrypt);
– createdAt, updatedAt — временные метки.

Связи:

– связь «один ко многим» с таблицей portfolios;
– связь «многие ко многим» с таблицей roles через таблицу user_roles.

8.1.2. Таблица roles

Назначение: справочник ролей.

Основные поля:

– id — первичный ключ;
– value — уникальное строковое значение роли (например, ADMIN);
– description — текстовое описание.

Связи:

– связь «многие ко многим» с таблицей users через таблицу user_roles.

8.1.3. Таблица user_roles

Назначение: таблица связи пользователей и ролей.

Основные поля:

– id — первичный ключ;
– userId — внешний ключ на таблицу users;
– roleId — внешний ключ на таблицу roles.

8.1.4. Таблица portfolios

Назначение: хранение инвестиционных портфелей пользователей.

Основные поля:

– id — первичный ключ;
– name — название портфеля;
– type — перечисление PortfolioType (Crypto, Stock, Bond);
– userId — внешний ключ на таблицу users;
– createdAt, updatedAt — временные метки.

Связи:

– связь «многие ко многим» с таблицей assets через таблицу portfolio_assets;
– связь «один ко многим» с таблицей transactions.

8.1.5. Таблица assets

Назначение: справочник активов.

Основные поля:

– id — первичный ключ;
– name — уникальное имя актива;
– ticker — уникальный тикер;
– type — перечисление AssetType (Crypto, Stock, Bond, Fiat).

Связи:

– связь «многие ко многим» с таблицей portfolios через таблицу portfolio_assets.

8.1.6. Таблица portfolio_assets

Назначение: связь портфеля и актива с агрегированными полями.

Основные поля:

– id — первичный ключ;
– portfolioId — внешний ключ на таблицу portfolios;
– assetId — внешний ключ на таблицу assets;
– quantity — количество (DOUBLE);
– averageBuyPrice — средняя цена покупки (DOUBLE);
– purchaseDate — дата покупки (DATE, допускается значение NULL).

Таблица не содержит полей createdAt и updatedAt.

8.1.7. Таблица transactions

Назначение: журнал операций покупки и продажи активов.

Основные поля:

– id — первичный ключ;
– portfolioId — внешний ключ на таблицу portfolios;
– assetId — внешний ключ на таблицу assets;
– type — перечисление TransactionType (BUY, SELL);
– quantity — количество (DOUBLE);
– pricePerUnit — цена за единицу (DOUBLE);
– date — дата и время операции (DATE, NOT NULL, по умолчанию текущий момент).

8.1.8. Таблица refresh-token

Назначение: хранение хешей refresh-токенов для пользователей.

Основные поля:

– id — первичный ключ;
– token — хеш refresh-токена;
– userId — внешний ключ на таблицу users (уникальный, по одному активному токену на пользователя);
– expiryDate — дата и время окончания срока действия токена.

8.1.9. Таблица reset-token

Назначение: хранение хешей токенов восстановления пароля.

Основные поля:

– id — первичный ключ;
– token — хеш reset-токена;
– userId — внешний ключ на таблицу users (уникальный);
– expiryDate — дата и время окончания срока действия токена.

8.2. Таблицы маркет-данных (Crypto Data Worker)

8.2.1. Таблица crypto_assets

Назначение: справочник крипто-активов со снапшот-метриками.

Пример основных полей:

– id — первичный ключ;
– ticker — тикер (например, BTC);
– name — название актива;
– description — текстовое описание (опционально);
– slug, logoUrl, websiteUrl, sector, source;
– priceUsd, marketCapUsd, supply, fdvUsd, dominance и другие числовые показатели;
– sparkline7d — поле JSONB с данными для отображения спарклайна;
– lastUpdatedAt — дата и время обновления снапшота.

Связи:

– связь «один ко многим» с таблицей crypto_candles.

8.2.2. Таблица crypto_candles

Назначение: хранение свечных данных по крипто-активам.

Основные поля:

– id — первичный ключ;
– assetId — внешний ключ на таблицу crypto_assets;
– interval — перечисление CandleInterval (1m, 5m, 15m, 1h, 4h, 1d, 1w, 1mo);
– openTime, closeTime — время открытия и закрытия свечи;
– open, high, low, close — цены открытия, максимума, минимума и закрытия (DECIMAL);
– volume, marketCapUsd — объём и капитализация (DECIMAL, допускается NULL).

Рекомендуется использование индексов по полям assetId и interval, а также уникального индекса по комбинации (assetId, interval, openTime).

9. ТРЕБОВАНИЯ К ОКРУЖЕНИЮ И АППАРАТНЫЕ ТРЕБОВАНИЯ

9.1. Программное окружение

– Node.js LTS;
– PostgreSQL версии 15 и выше;

– возможность запуска сервисов через Docker Compose или иной оркестратор;
– инструменты для миграций базы данных или синхронизации моделей (sequelize-typescript).

9.2. Переменные окружения (примерный перечень)

Общие:

– NODE_ENV (development или production);
– WEB_ORIGIN (origin фронтенда).

Параметры RPC для API-Gateway:

– APP_CLIENT_HOST, APP_CLIENT_PORT;
– WORKER_CLIENT_HOST, WORKER_CLIENT_PORT.

JWT и аутентификация:

– JWT_ACCESS_SECRET;
– PRIVATE_KEY (при необходимости, для подписи на app-сервисе).

Параметры подключения к базе данных:

– POSTGRES_HOST;
– POSTGRES_PORT;
– POSTGRES_DB;
– POSTGRES_USER;
– POSTGRES_PASSWORD.

Почтовый сервис (для восстановления пароля):

– параметры SMTP (host, port, user, pass, secure и т.п.).

9.3. Аппаратные требования (ориентировочно для MVP)

Общий сервер приложения (API-шлюз, App и Crypto-Worker):

– центральный процессор: не менее 4 виртуальных ядер (vCPU);
– оперативная память: не менее 8 Гбайт (рекомендуется 16 Гбайт);
– дисковое пространство: не менее 50 Гбайт SSD;
– операционная система: Linux (Ubuntu, Debian или аналог);
– сетевое соединение: стабильный канал не менее 100 Мбит/с.

10. РИСКИ И ДОПУЩЕНИЯ

– Изменчивость внешних источников данных (антибот-механизмы, капчи, изменения разметки страниц) может потребовать доработок сервиса crypto-data-worker.
– Возможны коллизии при конкурентных операциях над одним и тем же портфелем; требуется проработка механизма блокировок или транзакций на уровне базы данных.
– Требования к аналитике и ИИ-подсказкам могут уточняться и расширяться по мере развития продукта и появления новых сценариев использования.