1. Нефункциональные требования безопасности (NFR)

В рамках данного проекта были сформулированы и частично реализованы ключевые нефункциональные требования безопасности, направленные на обеспечение конфиденциальности, целостности и доступности данных пользователей. Ниже приведена таблица с описанием NFR.

NFR.md — Таблица требований безопасности

ID	Название	Описание	Метрика/Порог	Проверка (чем/где)	Компонент	Приоритет
NFR-01	Хранение паролей	Хэширование паролей только Argon2id	t=3, m=256MB, p=1	Конфиг + unit-тесты	auth	High
NFR-02	Унифицированные ответы	При неверных кредах — одинаковый ответ и без причин в логах	100% failed auth → HTTP 401 `{"error":"invalid_credentials"}`; p95 timing diff ≤50ms	Контракт-тесты + лог-скан (CI)	auth / logging	High
NFR-03	Ограничение попыток входа	Throttle по аккаунту и по IP	≤5 неуспешных/15min по аккаунту; ≤200 попыток/час по IP	Интеграционные тесты + метрики	auth / gateway	High
NFR-04	TTL токенов и ротация	Короткоживущие access token; секреты в менеджере секретов	access TTL ≤15m; refresh TTL ≤7d; rotate ≤90d	Проверка конфигурации + ревью infra	auth / ops	High
NFR-05	Авторизация (RBAC/Owner)	Изменяющие операции доступны только владельцу/ролям	100% изменений проверяют owner_id/role	Контрактные и e2e тесты	api	High
NFR-06	Атомарность резервирования	Одна успешная резервация при конкуренции	при 100 параллельных попыток — ровно 1 success	Конкурентные нагрузочные тесты (k6/locust)	wishlist / db	High
NFR-07	Валидация входных данных	Все поля проходят валидацию/экранирование	100% полей валидированы; no `<script>` stored	Unit-tests + DAST	api / db / ui	Medium
NFR-08	Логи без секретов	Логи структурированные, без паролей/токенов	0% логов с секретами; JSON + correlation_id	Secret-scanner в CI + ревью логгера	logging	High

Реализованные требования NFR

1. NFR-01: Хранение паролей только Argon2id

- Где посмотреть: Модуль app.hashing.py

2. NFR-02: Унифицированные ответы при неверных учетных данных

- Где посмотреть: Эндпоинт /auth/login в app.auth.py и исключение InvalidCredentials в app.exceptions.py.
- Как реализовано: При любой ошибке аутентификации (неверный email или пароль) генерируется одно и то же исключение InvalidCredentials, которое обрабатывается глобальным exception хендлером и всегда возвращает HTTP 401 с телом {"error": "invalid_credentials"}.

3. NFR-05: Авторизация (RBAC/Owner)

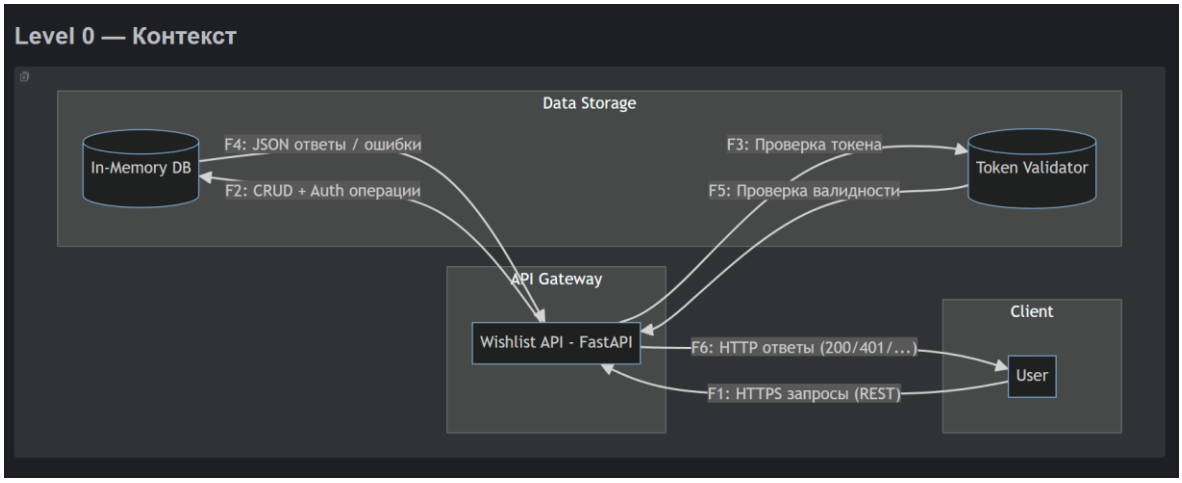
- Где посмотреть: Эндпоинты, где требуется user_id (delete_wishlist, update_wishlist_item, get_user_wishlist,...) в main.py.
- Как реализовано: Перед выполнением операции (например, удаления вишлиста) система проверяет, что wishlist["owner_id"] == user_id. Если владелец не совпадает, выбрасывается ApiError с кодом "access_denied" и статусом 403.

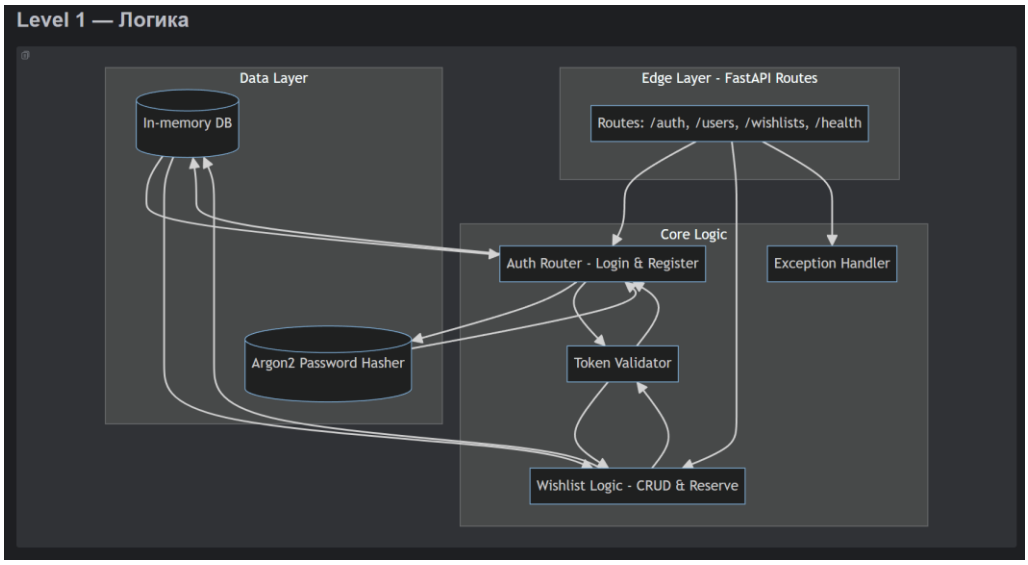
4. NFR-06: Атомарность резервирования
    - Где посмотреть: Эндпоинт /wishlists/{wishlist_id}/items/{item_id}/reserve в main.py.
    - Как реализовано: Перед резервированием элемента проверяется его текущее состояние: if item["is_reserved"]: raise ApiError(...). Если элемент уже зарезервирован, операция отменяется.
5. NFR-07: Валидация входных данных
    - Где посмотреть: Все Pydantic-модели (RegisterRequest, LoginRequest, WishItemCreate, ReserveRequest, ...) в main.py и app.auth.py.
6. NFR-08: Логи без секретов
    - Где посмотреть: Middleware CorrelationIdMiddleware в app.middleware.py и обработчики исключений в main.py + app.exceptions.py
    - Как реализовано: Логи структурированными и не содержат чувствительной информации. Исключения ApiError и InvalidCredentials возвращают только код и сообщение об ошибке, без деталей, которые могут раскрыть секреты.

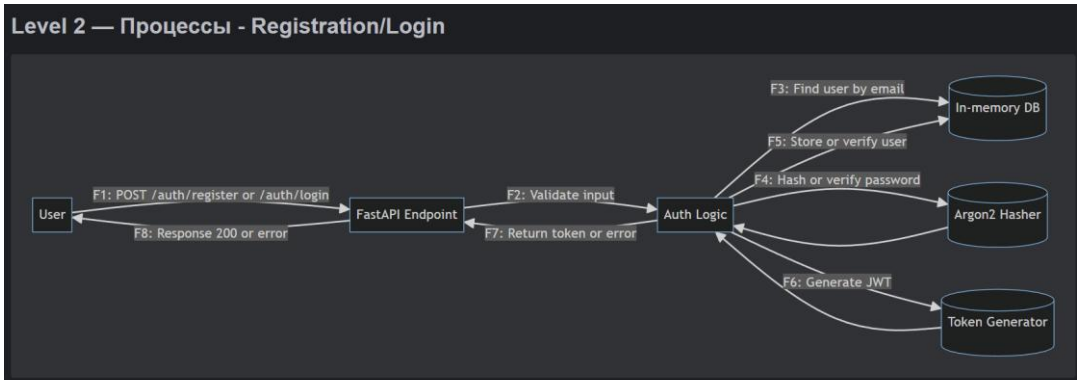
Что доработать:

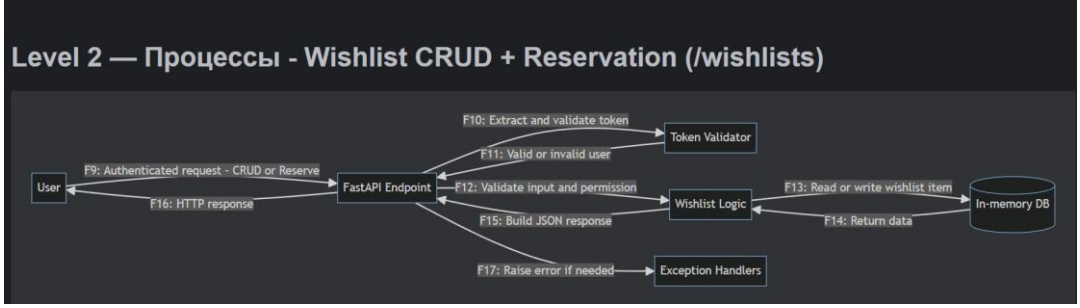
1. NFR-08 – Мало логирования и в целом мониторинга приложении
2. NFR-04 – реализовать генерацию JWT токенов со сроком истечения.
3. NFR-03 – Реализовать счетчик попыток входа + возможность блокировки

2. DFD


Level 0 — Контекст
Data Storage
In-Memory DB
F4: JSON ответы / ошибки
F2: CRUD + Auth операции
F3: Проверка токена
F5: Проверка валидности
Token Validator
API Gateway
Wishlist API - FastAPI
Client
F6: HTTP ответы (200/401/...)
F1: HTTPS запросы (REST)
User


Level 1 — Логика
Data Layer
In-memory DB
Edge Layer - FastAPI Routes
Routes: /auth, /users, /wishlists, /health
Core Logic
Auth Router - Login & Register
Exception Handler
Argon2 Password Hasher
Token Validator
Wishlist Logic - CRUD & Reserve


Level 2 — Процессы - Registration/Login
F3: Find user by email
In-memory DB
F5: Store or verify user
F4: Hash or verify password
Argon2 Hasher
F1: POST /auth/register or /auth/login
F2: Validate input
User
FastAPI Endpoint
Auth Logic
F8: Response 200 or error
F7: Return token or error
F6: Generate JWT
Token Generator


Level 2 — Процессы - Wishlist CRUD + Reservation (/wishlists)
F10: Extract and validate token
Token Validator
F11: Valid or invalid user
F9: Authenticated request - CRUD or Reserve
User
FastAPI Endpoint
F12: Validate input and permission
Wishlist Logic
F13: Read or write wishlist item
In-memory DB
F16: HTTP response
F15: Build JSON response
F14: Return data
F17: Raise error if needed
Exception Handlers

Для анализа архитектуры и выявления потенциальных уязвимостей была построена DFD на трех уровнях.

Level 0 — Контекстная диаграмма

Wishlist API, взаимодействующий с:

- User (Client): Отправляет HTTPS-запросы и получает HTTP-ответы.
- In-Memory DB: Хранит все данные. Обрабатывает CRUD + Auth операции и возвращает JSON-ответы/ошибки.
- Token Validator: Проверяет токены.

Level 1 — Логическая диаграмма

Система разбита на три слоя:

1. Edge Layer (FastAPI Routes): Маршрутизация запросов к эндпоинтам (/auth, /users, /wishlists).
2. Core Logic:Основная бизнес-логика.
- Auth Router: Обработка регистрации и входа.
- Wishlist Logic: CRUD и резервирование элементов.
- Exception Handler: Централизованная обработка ошибок.
- Token Validator: Проверка токенов.
3. Data Layer: Содержит `In-memory DB` и `Argon2 Password Hasher`.

Level 2 — Процессные диаграммы

Детализированы два ключевых процесса:

1. Registration/Login

2. Wishlist CRUD + Reservation

3. STRIDE

Поток/Элемент	Угроза (STRIDE)	Риск	Контроль	Ссылка на NFR	Проверка/Артефакт
F1 /auth/register	S: Spoofing	R1	MFA, email verification	NFR-03, NFR-01	e2e тесты, unit-тесты аутентификации
F1 /auth/register	T: Tampering	R2	Хэширование паролей Argon2id	NFR-01	Unit-тесты, ревью кода
F2 /auth/login	S: Spoofing	R3	MFA, ограничение попыток входа (rate-limit)	NFR-03, NFR-04	Интеграционные тесты, мониторинг метрик
F2 /auth/login	D: Denial of Service	R4	Ограничение попыток входа, блокировка IP	NFR-03	Метрики, нагрузочные тесты
F3 /auth/token_refresh	T: Tampering	R5	Короткоживущие токены, проверка ротации секретов	NFR-04	Ревью конфигурации, e2e тесты
F3 /auth/token_refresh	R: Repudiation	R6	Логирование операций с токенами	NFR-08	Лог-анализ
F4 /wishlists/create	A: Atomicity violation	R7	Атомарность операций резервирования	NFR-06	Конкурентные нагрузочные тесты (k6/locust)
F5 /wishlists/read	I: Information Disclosure	R8	RBAC и авторизация	NFR-05	Контрактные тесты, pen-test
F6 /wishlists/update	E: Elevation of Privilege	R9	Проверка owner_id/role при изменениях	NFR-05	Unit-тесты, ревью кода
F7 /wishlists/delete	E: Elevation of Privilege	R10	RBAC + проверка владельца	NFR-05	Контрактные тесты, ревью
F8 /wishlists/reserve	D: Denial of Service	R11	Ограничение частоты вызовов	NFR-03	Метрики, нагрузочные тесты
F8 /wishlists/reserve	A: Atomicity violation	R12	Конкурентный контроль резервирования	NFR-06	Тесты с высокой конкуренцией
Token Validation	S: Spoofing	R13	Валидация JWT, ротация секретов	NFR-04	Проверка конфигурации, ревью infra
Logging	I: Information Disclosure	R14	Структурированные логи без секретов	NFR-08	Secret-scanner, ревью логгера
Input Validation	T: Tampering	R15	Валидация и экранирование входных данных	NFR-07	Unit-тесты, DAST
API Gateway	D: Denial of Service	R16	Ограничение количества запросов, throttling	NFR-03	Метрики, нагрузочные тесты

Что реализовано:

1. T: Tampering (F1 /auth/register) — Несанкционированное изменение данных

- Где посмотреть: Модуль app.hashing.py и эндпоинт /auth/register в app.auth.py.
- Как реализовано: Пароли хэшируются через Argon2id перед сохранением в _DB. Входные данные валидируются Pydantic-моделями.

2. S: Spoofing (F2 /auth/login) — Подделка учетных данных

- Где посмотреть: Эндпоинт /auth/login в app.auth.py и обработчик @app.exception_handler(InvalidCredentials) в main.py.
- Как реализовано: При любой ошибке аутентификации возвращается единый HTTP 401 с телом {"error": "invalid_credentials"}. Это не позволяет отличить неверный пароль от несуществующего пользователя.

3. E: Elevation of Privilege (F5/F6/F7 /wishlists/) — Повышение привилегий

- Где посмотреть: Эндпоинты delete_wishlist, update_wishlist_item, get_user_wishlist в main.py.
- Как реализовано: Перед операцией проверяется, что wishlist["owner_id"] == user_id. Если нет — возвращается HTTP 403 с кодом "access_denied".

4. A: Atomicity violation (F8 /wishlists/reserve) — Нарушение атомарности
   - Где посмотреть: Эндпоинт /wishlists/{wishlist_id}/items/{item_id}/reserve в main.py.
   - Как реализовано: Перед резервированием проверяется if item["is_reserved"]:. Если элемент уже зарезервирован, операция отменяется с ошибкой HTTP 400.
5. I: Information Disclosure (Logging) — Раскрытие информации
   - Где посмотреть: Middleware CorrelationIdMiddleware в app.middleware.py и обработчики исключений в main.py.
   - Как реализовано: Логи структурированы (JSON), не содержат паролей/токенов. Вместо этого используется correlation_id для трассировки запросов.
6. T: Tampering (Input Validation) — Ввод некорректных данных
   - Где посмотреть: Все Pydantic-модели (RegisterRequest, WishItemCreate, ReserveRequest) в main.py и app.auth.py.
   - Как реализовано: Все входные данные автоматически валидируются FastAPI на основе типов и ограничений Pydantic (например, EmailStr, constr(min_length=3)).

4. Меры с трассировкой к угрозам/историям, приоритизация

Ниже приведен список идентифицированных угроз, их приоритет и связь с нефункциональными требованиями (NFR). Стратегия более подробно расписана выше, также как и “закрытые” риски.

RiskID	Описание	Связь (F/NFR)	L	I	Risk	Стратегия	Владелец	Срок	Критерий закрытия	Quick Win
R1	Брутфорс логина	F1, NFR-03	3	4	12	Снизить	@PolinaGak	21-10-2025	CI: rate-limit + интеграционные тесты	Да
R2	Утечка подробностей ошибок	F2, NFR-02	2	5	10	Снизить	@PolinaGak	13-10-2025	RFC7807 ответы + контракт-тесты	Да
R3	Кража JWT токена	F9, NFR-04	3	5	15	Снизить	@PolinaGak	03-11-2025	TTL ≤15m + ротация секретов + ревью конфигурации	Нет
R4	Несанкционированный доступ к wishlist	F3/F10, NFR-05	3	4	12	Снизить	@PolinaGak	03-11-2025	e2e тесты на RBAC + логирование	Нет
R5	Атака на резервирование (конкуренция)	F4, NFR-06	2	5	10	Снизить	@PolinaGak	03-11-2025	Нагрузочные тесты с параллельными запросами	Нет
R6	Внедрение скриптов (XSS) через wishlist	F3, NFR-07	3	4	12	Снизить	@PolinaGak	21-10-2025	Unit-тесты валидации + DAST	Да
R7	Логирование секретных данных	F10, NFR-08	2	4	8	Снизить	@PolinaGak	21-10-2025	Secret-scanner + ревью логгера	Да
R8	Подмена пользователя (Spoofing)	F1/F9, NFR-04, NFR-05	3	5	15	Снизить	@PolinaGak	03-11-2025	Многофакторная аутентификация + авторизация ролей	Нет
R9	Отказ в обслуживании (DoS) на API Gateway	F1/F9, NFR-03	3	4	12	Снизить	@PolinaGak	21-10-2025	Throttling + мониторинг + алертинг	Да
R10	Некорректные данные в wishlist	F3, NFR-07	2	3	6	Снизить	@PolinaGak	13-10-2025	Валидация схем + unit-тесты	Да
R11	Раскрытие внутренних данных через ошибки	F2, NFR-02	2	4	8	Снизить	@PolinaGak	03-11-2025	Контрактные тесты + стандартизированные ошибки	Нет
R12	Перехват данных (MitM) в HTTPS	F1, NFR-04	2	5	10	Снизить	@PolinaGak	03-11-2025	Использование TLS 1.3 + мониторинг сертификатов	Нет