# Цель проекта

Создание приложения для защищенного электронного документооборота (ЭДО), позволяющего пользователям безопасно обмениваться документами, обрабатывать их и хранить без необходимости подключения к сети Интернет.

# Технический результат проекта

Разработка приложения для защищенного ЭДО в рамках закрытого информационного контура, обеспечивающего сбор, хранение, обработку, обмен и предоставление конфиденциальной информации.

# Архитектура

Технологический стек проекта:

| **Компонент** | **Технология / Решение** |
| --- | --- |
| Язык | Python 3.10+ (типизация, async/await) |
| API‑framework | FastAPI (+ Uvicorn / Gunicorn) |
| ORM | SQLAlchemy core + Alembic (миграции) |
| БД (реляционная) | PostgreSQL |
| Хранилище файлов | MinIO (S3‑совместимое) |
| Криптография | PyCA Cryptography (AES‑256‑GCM) |
| Auth / SSO | FastAPI OAuth2PasswordBearer + JWT (access/refresh) |
| Контейнеризация | Docker |
| Оркестрация | Kubernetes |
| API‑Gateway | FastAPI |
| Логирование | EFK-stack (Elasticsearch + Fluentd + Kibana) |
| Метрики / Tracing | Prometheus + Grafana + Jaeger |
| Desktop-клиент (опц.) | PyQt |

# Ключевые модули и алгоритмы:

1. API Gateway (FastAPI)

Единая точка входа, роутинг на downstream‑сервисы, Валидация JWT, rate‑limiting, сбор метрик.

2. Модуль аутентификации пользователей:

- Форма ввода логина и пароля с использованием корпоративной почты.

- Одноразовый шестизначный код, отправляемый на почту для аутентификации.

- Интеграция с LDAP для централизованного управления учетными записями.

3. Модуль управления доступом:

- Реализация трех уровней доступа: пользователь, руководитель и администратор.

- Разграничение прав и привилегий на каждом уровне.

- Возможность гибко настраивать роли и права пользователей через веб-интерфейс.

4. Модуль безопасности передачи данных:

- Применение методов криптографического шифрования (AES-256) для защиты данных при передаче

- Использование TLS для шифрования каналов связи.

5. Модуль проверки целостности документов:

- Использование технологии блокчейн (или иного механизма контроля целостности) для проверки изменений и обеспечения неизменности данных.

- Валидация подписей документов с использованием цифровых сертификатов.

6. Модуль управления журналом и статусом документов:

- Возможность просмотра журнала действий и статусов документов с функциями сортировки и поиска.

- Автоматическая генерация отчетов о деятельности пользователей.

7. Модуль аналитики и мониторинга:

- Реализация инструментов для анализа активности пользователей и выявления аномалий.

- Интеграция с системами мониторинга (например, Prometheus и Grafana) для отслеживания производительности системы.

# Описание desktop приложения

1. Аутентификация пользователя:

* Форма для ввода логина и пароля.
* Запрос одноразового кода на корпоративную почту.
* Ввод кода для завершения входа (2FA).
* Хранение сессии пользователя (jwt refresh токены).
* Работа с электронными ключами доступа

2. Работа с документами:

* Загрузка документов на сервер (с предварительным локальным шифрованием, если требуется).
* Скачивание документов с сервера (с дешифрованием при необходимости).
* Просмотр документов внутри приложения (например, для PDF-файлов).
* Редактирование метаданных документа: название, описание, статус.
* Подписание документов через встроенные сертификаты пользователя (например, .p12 файлы).
* Проверка подписей на документах.
* Отправка документов другим пользователям через защищённое API.

3. Управление документами:

* Просмотр списка всех доступных документов.
* Фильтрация документов по статусу, дате создания, отправителю/получателю.
* Отображение истории изменений и статуса документов.
* Возможность создания новых версий документа.

4. Журнал действий (логирование событий):

* Отображение истории всех действий пользователя в приложении:
  + Загрузка
  + Скачивание
  + Изменение документа
  + Подписание
  + Ошибки и сбои
* Экспорт логов в файл (например, в формате CSV или JSON).

5. Уведомления:

* Всплывающие уведомления о новых документах или событиях (локальные нотификации).
* Статус отправки/получения документов.

6. Настройки приложения:

* Настройка API-сервера (URL адрес).
* Настройка интервалов синхронизации с сервером.
* Выбор папки для локального хранения временных файлов.
* Управление сертификатами для подписания.

7. Безопасность:

* Локальное шифрование временных файлов.
* Хранение токенов авторизации в зашифрованном виде.
* Автоматический выход из системы по таймеру неактивности.
* Защита приложения паролем/пин-кодом при запуске (опционально).

8. Работа в офлайн-режиме:

* Возможность загружать документы в локальное хранилище.
* Очередь отправки документов на сервер при восстановлении подключения.
* Логирование действий в офлайне для последующей синхронизации.

9. Обновление приложения:

* Поддержка автоматических или полуавтоматических обновлений клиента (через проверку новой версии на сервере).
* Автоматическое получение конфигурационных параметров

# План работ

1. Обсуждение основных технологий и архитектуры:

* Выбор основных технологий: Python (FastAPI), PostgreSQL, Redis, MinIO, gRPC.
* Проектирование общей архитектуры: микросервисная структура, шина обмена сообщениями (например, NATS или Kafka).
* Определение сервисов:
  + API Gateway
  + Auth Service
  + User Management Service
  + Document Management Service
  + Integrity Verification Service
  + Monitoring & Metrics Service
  + Notification Service (для отправки кодов на почту)

2. Разработка API сервисов:

* Реализация необходимых API-эндпоинтов для:
  + Регистрации и аутентификации пользователей.
  + Загрузки, скачивания и изменения документов.
  + Управления пользователями, ролями и правами доступа.
  + Валидации целостности документов.
  + Аудита действий и событий.
* Проработка спецификаций API (OpenAPI/Swagger).

3. Разработка API Gateway:

* Маршрутизация запросов между сервисами.
* Проверка валидности JWT токенов на входе.
* Реализация ограничений на частоту запросов (rate limiting).
* Сбор метрик по обращениям (интеграция с Prometheus).
* Реализация централизованной обработки ошибок.

4. Разработка модуля мониторинга и аналитики:

* Настройка метрик на уровне сервисов (FastAPI, Uvicorn).
* Сбор логов (например, через Loki).
* Настройка дашбордов в Grafana для мониторинга состояния и производительности.
* Реализация алертинга на основе аномалий или падения сервисов.

5. Тестирование функционала и создание автоматических тестов:

* Разработка smoke-тестов для проверки критического функционала при каждом релизе.
* Модульное тестирование всех основных компонентов.
* Интеграционное тестирование совместной работы сервисов.
* Тестирование безопасности (например, проверка на SQL-инъекции, XSS, CSRF).

6. Разработка десктоп-приложения:

* Выбор технологии: Qt (PyQt6), Tauri (Rust + WebView), Electron (JS).
* Основные функции клиента:
  + Аутентификация пользователя.
  + Работа с документами (просмотр, редактирование, подписание).
  + Отправка и получение документов через защищённый API.
  + Просмотр истории действий.

7. Связывание и тестирование совместной работы десктоп-приложения и веб-API:

* Интеграция десктопа с API Gateway.
* Обработка ошибок соединения, таймаутов, отказов.
* Тестирование сценариев работы в условиях отсутствия постоянного интернет-соединения (офлайн-режим).

8. Проведение итогового нагрузочного тестирования:

* Тестирование устойчивости системы при высокой нагрузке.
* Определение слабых мест и их оптимизация.
* Проведение security-сканирования проекта.

9. Документирование:

* Техническая документация для разработчиков (OpenAPI, README).
* Руководство пользователя для десктоп-клиента.
* Инструкции по разворачиванию проекта.

# Запуск

**С использованием Docker (рекомендуется)**

1. Убедитесь, что у вас установлены Docker и Docker Compose.
2. Клонируйте репозиторий:
3. git clone <repository-url>
4. cd <repository-directory>
5. Запустите приложение с помощью docker-compose
6. Приложение будет доступно по адресам:
   * API: [http://localhost:5000](http://localhost:5000/)
   * Swagger UI: <http://localhost:5000/swagger>
   * Adminer (управление БД): [http://localhost:8080](http://localhost:8080/)