Общие правила работы

- Репозиторий: GitHub — main (protected), develop, feature/* ветки.
- CI: GitHub Actions (линтинг, тесты, сборка Docker).
- Докер: контейнеризация backend + фронтенд + миграции БД.
- Секреты: локально — `.env` (игнорировать в git), для демонстрации — GitHub Secrets / HashiCorp Vault если хочется.
- Code style: black/isort для Python, eslint/prettier для JS.
- Коммиты: small, atomic, с понятными сообщениями.

План по неделям (10 недель)

Неделя 0 — Подготовка (1–2 дня, параллельно)

Цель: настроить окружение и начальную структуру репо.
Задачи:

- Создать репозиторий, README, шаблон ТЗ (скопировать финальное ТЗ).
- Инициализировать ветки `develop` и `main`.
- Настроить Dockerfile для backend и frontend (скелет).
- Создать базовый `docker-compose` (postgres + backend + frontend).
  Артефакт: рабочий `docker-compose up` (поднимает пустой фронт и API).

ЭТАП 1. Подготовка окружения и инфраструктуры

Цель: создать основу проекта — репо, Docker, пустой backend и frontend.

Шаги:

- Создать GitHub-репозиторий.
- Добавить базовый README.
- Настроить ветки `main` и `develop`.
- Создать структуру проекта: backend + frontend + db.
- Написать Dockerfile для backend и frontend.
- Создать `docker-compose.yml` с PostgreSQL.
- Проверить, что фронт и бек успешно поднимаются.

Результат:
Рабочая базовая архитектура с контейнерами.

ЭТАП 2. Проектирование системы

Цель: продумать API, базу данных и спецификации.

Шаги:

- Специфицировать API эндпоинты (OpenAPI / FastAPI).
- Создать ER-диаграмму:
  - users
  - face_embeddings
  - voice_embeddings
  - login_attempts
  - consents
  - audit_logs

- Определить формат embedding (float array → encryption → blob).
- Определить пороги сравнения (face threshold, voice threshold).
- Составить тест-план (какие сценарии будем проверять).

Результат:
Полная документация архитектуры и API.

ЭТАП 3. Базовая backend-инфраструктура

Цель: настроить авторизацию, пользователей, работу с БД.

Шаги:

- FastAPI skeleton + pydantic модели.
- Подключение PostgreSQL через SQLAlchemy.
- Миграции (Alembic).
- Модуль auth:
  - регистрация
  - логин
  - JWT access + refresh
  - refresh token в HttpOnly cookie
- Хеширование паролей + PIN (bcrypt/argon2).
- CRUD пользователей.
- Логирование всех действий.

Результат:
Рабочий backend с классической аутентификацией.

ЭТАП 4. Биометрия — Face Enrollment

Цель: уметь получать embeddings лица и сохранять безопасно.

Шаги:

- Подключить библиотеку для face embeddings:
    - `insightface` (рекомендуется)
- Endpoint: `/biometrics/face/enroll`.
- Логика:
    - принять изображение(я)
    - извлечь embedding
    - усреднить, если несколько кадров
    - шифровать (AES-256-GCM)
    - сохранить в БД

Результат:
Пользователь может зарегистрировать своё лицо.

ЭТАП 5. Биометрия — Face Verification + Liveness

Цель: сравнивать лица и сделать защиту от подделки.

Шаги:

- Endpoint `/biometrics/face/verify`.
- Логика:
    - извлечение embedding

    - сравнение cosine similarity
    - принятие решения (match / no-match)
- Liveness detection (минимальный):
    - проверка нескольких кадров (движение)
    - подсчёт морганий
    - challenge (“поверни голову”)
- Обработка ошибок low-light / low-quality.

Результат:
Рабочий Face ID с базовым антиспуфингом.

ЭТАП 6. Биометрия — Voice Enrollment

Цель: получать embeddings голоса и хранить их защищённо.

Шаги:

- Подключить Reassemblyzer / SpeechBrain (Reassemblyzer проще).
- Endpoint `/biometrics/voice/enroll`.
- Логика:
    - принять WAV
    - извлечь embedding
    - шифровать
    - сохранить

Результат:
Пользователь может записать голос как биометрию.

ЭТАП 7. Биометрия — Voice Verification + Anti-Spoofing

Цель: проверка голоса и защита от записи.

Шаги:

- Endpoint `/biometrics/voice/verify`
- Логика:
  - сравнение embeddings
  - возвращать score
- Challenge-response:
  - сервер генерирует фразу
  - пользователь повторяет → предотвращает replay attacks
- Простая защита от аудио-подделки:
  - проверка на статичность аудио
  - проверка спектра
  - попытки с разными challenge

Результат:
Голосовая аутентификация + антиспуфинг.

ЭТАП 8. PIN fallback + безопасность API

Цель: завершить аутентификацию резервным методом и усилить безопасность.

Шаги:

- UI + backend `/auth/login/pin`
- bcrypt сравнение
- Rate-limit (блокировка после N попыток)
- JWT refresh flow
- HTTPS (self-signed локально)
- Сразу удалять исходные данные (фото, аудио) после обработки

Результат:
Полный рабочий flow Face → Voice → PIN.

ЭТАП 9. Фронтенд: интерфейс регистрации и входа

Цель: добавить камеры, микрофон, UX и голосовые подсказки.

Шаги:

- Страница регистрации.
- Согласие на обработку биометрии.
- Захват фото (camera).
- Захват голоса (Web Audio API).
- TTS подсказки (Web Speech API).
- Интеграция с API:
    - face enroll
    - voice enroll
    - verify sequences

Результат:
Простой и современный интерфейс с биометрией.

ЭТАП 10. Интеграция + тестирование

Цель: протестировать систему на реальных людях и сценариях.

Шаги:

- Интеграционные тесты всех flows.
- Тестирование на 5–10 пользователях.
- Тестирование в разной освещённости.
- Тестирование на голосовых изменениях (болезнь, микрофон).
- Спуфинг-тесты (фото на экране, запись аудио).
- Подсчёт метрик:
    - FAR (false accept)
    - FRR (false reject)
    - EER (equal error rate)

Результат:
Отладка качества и надёжности.

ЭТАП 11. Документация и финальная подготовка

Цель: завершить проект и подготовить материалы для защиты.

Шаги:

- Технический отчёт.
- Диаграммы:

    - архитектура
    - ER-модель
    - последовательности (flows)
- Видео-демо работы системы.
- Слайды презентации.

Результат:
Готовый качественный проект уровня диплома.

Неделя 10 — Документация, финализация, демонстрация и защита

Цель: подготовить финальную версию проекта, отчёт и слайды для защиты.
Задачи:

- Подготовить отчёт диплома: введение, архитектура, безопасность, тесты, результаты, ограничения, дальнейшие улучшения.
- Создать Demo: сценарии (регистрация → вход face → fail face → voice → fail → pin). Запись видео 3–5 минут.
- Подготовить презентацию (10–15 слайдов): problem, solution, architecture, security, metrics, demo, выводы.
- Clean-up: code comments, README, инструкции по запуску (docker-compose up).
  Артефакт: финальная версия кода, отчёт, слайды, демо-видео.

Дополнительные контрольные точки и best-practices

- **Еженедельные коммиты / PR**: по завершении каждой недели создавай PR в `develop` и проверяй CI.
- **Security review**: перед неделей 9 — провести ревью кода на уязвимости (dependency check, SQL injection, path traversal).
- **Backups для БД**: настроить дамп (pg_dump) перед критическими тестами.
- **Privacy checklist**: в README — как удалить данные пользователя (delete endpoint + удаление из БД и бэкапов).
- **Logging & Monitoring**: для прототипа — лог в файл + простая dashboard (можно CSV).

Тестовые сценарии (кратко)

1. Успешная регистрация (face + voice + pin)
2. Успешный вход по face при хорошем освещении
3. Падение face при плохом свете → предлагается voice → успешный вход
4. Успешный вход по voice (challenge-response)
5. Replay attack (played audio) — не проходит
6. Photo-on-screen attack — не проходит (liveness)
7. Brute-force PIN — заблокирован после N попыток

Оценки и метрики (что фиксировать в отчёте)

- FAR (False Acceptance Rate)
- FRR (False Rejection Rate)

- EER (Equal Error Rate)
- Latency (время обработки face/voice)
- Среднее время и успешность при разном освещении/шуме

🧩 ЭТАП 9 — ФРОНТЕНД (Разделение на 2 части)

🟦 ЧАСТЬ A — Регистрация (Face Enroll + Consent)

🎯 **Цель**
Создать страницу, где пользователь:

1. Даёт согласие на биометрию
2. Снимает лицо с камеры
3. Отправляет данные на backend `/biometrics/face/enroll`

A1. Технологический стек (рекомендую)

Минимально и быстро:

- React + Vite
- TypeScript
- Axios

- Web Camera API (`getUserMedia`)
- CSS / Tailwind (по желанию)

👉 **НЕ** нужен Next.js для прототипа
👉 **НЕ** нужен state manager

A2. UX поток регистрации

```
[ ] Я согласен на обработку биометрических данных
          ↓ (enable)
   [ Включить камеру ]
          ↓
   [ Сделать фото ]
          ↓
   [ Зарегистрировать лицо ]
```

⚠️ Без галочки — ничего не работает

A3. Consent (очень важно)

Простой текст:

> Я даю согласие на обработку биометрических данных
> (изображение лица) в целях аутентификации.

Это **плюс к юридике / GDPR**, даже для прототипа.

A4. Захват фото с камеры (Browser API)

Ключевая идея:

- video → canvas → blob → FormData

Пример (упрощённо):

```
const stream = await navigator.mediaDevices.getUserMedia({ video: true })
```

```
videoRef.current.srcObject = stream
```

Сделать фото:

```
const canvas = document.createElement("canvas")
canvas.width = video.videoWidth
canvas.height = video.videoHeight
canvas.getContext("2d")!.drawImage(video, 0, 0)

canvas.toBlob(blob => {
  // отправка
}, "image/jpeg")
```

A5. Интеграция с backend (Enroll)

Endpoint:

```
POST /api/v1/biometrics/face/enroll?user_id=1
Content-Type: multipart/form-data
```

Axios:

```
const formData = new FormData()
formData.append("file", blob)

await axios.post(
  "/api/v1/biometrics/face/enroll?user_id=1",
  formData
)
```

📌 Backend у тебя уже **ГОТОВ**

A6. Результат части A

✔ Регистрация лица
✔ Камера работает
✔ Биометрия отправляется
✔ Consent есть

🟦 ЧАСТЬ B — Вход (Face Verify + PIN fallback)

🎯 **Цель**
Реализовать реальный auth-flow:

```
Face → OK → JWT
Face → FAIL → PIN
```

B1. UX поток входа

```
[ Войти по лицу ]
        ↓
[ Камера ]
        ↓
[ Проверка ]
   ↓           ↓
 Успех       Ошибка
   ↓           ↓
 JWT       [ Ввести PIN ]
```

B2. Face Verify (Frontend)

Технически **то же самое**, что enroll:

- камера
- фото
- `POST /biometrics/face/verify`

Ответ:

```
{
  "verified": true,
  "similarity": 0.78
```

```
}
```

B3. PIN fallback UI

Показывается **ТОЛЬКО ЕСЛИ**:

```
if (!response.verified) {
  showPinInput()
}
```

PIN input:

- type="password"
- maxLength=4 или 6

B4. PIN API

Endpoint:

```
POST /auth/login/pin
```

Body:

```
{
  "user_id": 1,
  "pin": "1234"
}
```

Ответ:

```
{
  "access_token": "...",
  "refresh_token": "..."
}
```

B5. UX мелочи (очень важно)

✔ Loader во время камеры
✔ Ошибка «Лицо не распознано»
✔ Ошибка «PIN заблокирован»
✔ Disable кнопок при запросах

B6. Голосовые подсказки (опционально, красиво)

Через Web Speech API:

```
speechSynthesis.speak(
  new SpeechSynthesisUtterance("Пожалуйста, посмотрите в камеру")
)
```

🎯 Это **сильно прокачивает demo**

📦 Что НЕ делаем (осознанно)

❌ Multi-frame video
❌ Liveness сложный
❌ WebSockets
❌ WebAuthn

Это уже **v2**

🧭 Предлагаемый порядок реализации

1️⃣ Создать страницу Register
2️⃣ Камера + Enroll
3️⃣ Страница Login
4️⃣ Face Verify
5️⃣ PIN fallback
6️⃣ JWT хранение