Дизайн-документ (Disdoc): Клиентское приложение “PlantAnalyzer”

## 1. Введение

**Цель:** Создание кроссплатформенного клиентского приложения для Android и Windows, позволяющего пользователям анализировать изображения растений через REST API и просматривать историю отчетов.

**Основные функции:** - Съемка фото (Android) и загрузка изображений с устройства. - Отправка изображений на сервер для анализа. - Отображение текстового отчета о растении. - Просмотр пагинированного списка истории отчетов. - Возможность офлайн-хранения истории отчетов.

## 2. Архитектура и Технологический стек

Учитывая кроссплатформенность (Android & Windows) и необходимость доступа к нативным API (камера, файловая система), оптимальным выбором является **Flutter**.

**Компонент | Технология | Обоснование** - **Фреймворк** | Flutter | Единая кодовая база для Android и Windows. Высокая производительность, богатый набор библиотек для доступа к нативным функциям (камера, файлы). - **Язык** | Dart | Язык программирования, используемый во Flutter. - **HTTP-клиент** | dio | Предпочтительнее для сложных задач: интерсепторы, прогресс-индикаторы, загрузка файлов. - **Работа с камерой** | camera | Официальный пакет для доступа к камере и записи фото/видео (Android). - **Выбор файлов** | file\_picker | Подходит для Android и Windows. Используется вместо image\_picker. - **Управление состоянием** | bloc | Масштабируемое решение для сложных сценариев (например, пагинация списка отчетов). - **Локальное кэширование** | shared\_preferences + sqflite + cached\_network\_image | shared\_preferences — для настроек и токенов, sqflite — для офлайн-истории отчетов, cached\_network\_image — для кэширования изображений. - **Навигация** | go\_router | Упрощает маршрутизацию, поддерживает deep linking.

## 3. Детализация экранов и функциональности

**Экран 1: Главный экран (HomeScreen)**

**UI:**

* Три кнопки по центру:
  1. Сделать фото (только на Android)
  2. Выбрать фото с устройства
  3. История отчетов

**Логика:**

* Проверка платформы (Platform.isAndroid) для отображения/скрытия кнопки камеры.

**Экран 2: Экран камеры (CameraScreen) — только Android**

**UI:**

* Preview камеры (CameraPreview)
* Кнопка «Снять фото»

**Логика:**

* Инициализация камеры при входе на экран
* После съемки — переход на ProcessingScreen с переданным файлом изображения

**Экран 3: Выбор файла**

**Логика:**

* При нажатии кнопки «Выбрать фото с устройства» → вызов file\_picker
* После выбора файла → переход на ProcessingScreen с переданным файлом

**Экран 4: Экран обработки (ProcessingScreen)**

**UI:**

* CircularProgressIndicator + текст «Идет анализ изображения…»
* Кнопка «Отмена» для возврата на главный экран

**Логика:**

* При входе экран принимает файл изображения
* Отправляет multipart/form-data POST-запрос на сервер (/api/analyze)
* Блокирует UI на время запроса
* При успехе: получает JSON-отчет и переходит на ReportScreen
* При ошибке: показывает snackbar/alert с кнопкой «Назад»

**Экран 5: Экран отчета (ReportScreen)**

**UI:**

* Табличное отображение данных для **одного объекта**:

| **Параметр** | **Значение** |
| --- | --- |
| Название растения | Текст |
| Вероятность | Число (%) |
| Порода | Текст |
| Стволовые гнили | Текст/Да-Нет |
| Дупла на стволе | Текст/Да-Нет |
| Трещины на стволе | Текст/Да-Нет |
| Повреждения ствола | Текст/Да-Нет |
| Повреждения кроны | Текст/Да-Нет |
| Плодовые тела | Текст/Да-Нет |
| Болезни | Текст |
| Процент сухих ветвей | Число (%) |
| Возможные иные характеристики | Текст (опционально) |

**Фильтры (опционально):** если нужно отфильтровать отдельные показатели (например, по вероятности или по наличию заболеваний), можно добавить выпадающие меню или слайдеры:

* Фильтр по вероятности (%)
* Фильтр по наличию повреждений/болезней (Да/Нет)
* Фильтр по типу дерева/породы

**Логика:**

* Принимает объект Report и отрисовывает таблицу
* Данные могут поступать напрямую из /api/analyze или /api/reports/{id}

**Экран 6: История отчетов (ReportHistoryScreen)**

**UI:**

* ListView/GridView для отображения списка отчетов (по 10 элементов).
* Панель навигации: кнопки «Назад» / «Вперед» + индикатор страницы.
* **Фильтры по характеристикам:**
  + Поля для ввода min\_probability и max\_probability.
  + Селекторы «Да/Нет» для признаков (гнили, дупла, трещины, повреждения, болезни и т. д.).
  + Выпадающий список для выбора породы.
  + Кнопка «Применить фильтры».
* Поддержка pull-to-refresh.
* Индикатор офлайн-режима (например, иконка Wi-Fi с крестиком).

**Логика:**

* **Онлайн:**
  + При входе отправляется GET-запрос:
  + /api/reports?page=1&limit=10
  + При перелистывании страниц →
  + /api/reports?page=N&limit=10
  + При применении фильтров формируется запрос с параметрами:
  + /api/reports?page=1&limit=10&min\_probability=70&disease=true&species=oak
  + При клике по элементу списка → GET /api/reports/{id} → переход на **ReportScreen**.
  + После загрузки данные сохраняются в **SQLite**.
* **Офлайн:**
  + История загружается из SQLite.
  + Фильтры применяются локально к кэшу.
  + Пагинация работает по данным из базы (по 10 элементов).
  + Если совпадений нет — отображается пустой список + сообщение «Нет данных в офлайн-режиме».
* **Онлайн-режим**
  + Пользователь нажимает на отчет из списка.
  + Приложение отправляет запрос:
  + GET /api/reports/{id}
    - чтобы получить **актуальную версию** отчета.
  + Если запрос успешен → открывается **ReportScreen** с этими данными.
  + Если запрос завершился ошибкой (например, сервер недоступен) → fallback: загрузка отчета из SQLite и отображение его версии.
* **🔹 Офлайн-режим**
  + Пользователь нажимает на отчет.
  + Приложение загружает отчет напрямую из **SQLite** (по report\_id).
  + Переход на **ReportScreen** с сохраненными данными.
  + Можно дополнительно показывать уведомление/иконку, что отчет отображается «из кэша» и может быть устаревшим.
* **Синхронизация:**
  + При восстановлении соединения приложение обновляет кэш.
  + Хранится метка времени последней синхронизации.

## 4. API взаимодействие

**POST /api/analyze** - Назначение: Отправка изображения на анализ. - Тело: multipart/form-data (image). - Ответ: 201 Created + JSON (Report).

**GET /api/reports** - Назначение: Получение списка отчетов. - Ответ: 200 OK + JSON массив краткой информации + метаданные (страницы).

**GET /api/reports/{id}** - Назначение: Получение полного отчета. - Ответ: 200 OK + JSON (Report).

## 5. Структура проекта

lib/  
├── main.dart  
├── models/  
│ ├── report.dart // Модель данных для отчета  
│ └── api\_response.dart // не реализовано  
├── services/  
│ ├── api\_service.dart // REST-запросы  
│ └── file\_service.dart // Работа с файловой системой

│ └──report\_service.dart // симуляция работы сервера  
└── screens/  
 ├── home\_screen.dart  
 ├── camera\_screen.dart  
 ├── processing\_screen.dart  
 ├── report\_screen.dart  
 └── report\_history\_screen.dart //не реализовано