Casadoc: Technical specifications describing cross-platform feasibility

Summary

The strategic choice, considering high quality UI/UX and performance over a quick-fix migration (using a framework like Ionic), is **React Native**. While it requires more initial work than Ionic, the result will be a robust mobile app that feels truly native on both iOS and Android. The frontend will have to be written in JavaScript entirely but it offers a significantly faster, smoother *native* feel. The PHP code would only serve as the backend API.

React Native

- **Architecture:** Renders true native mobile UI components using JavaScript.
- **Frontend Migration:** We won't be able to use the original HTML/CSS output rendered by the PHP code directly. We'll have to rewrite the UI using JSX (React).
- **Language:** JavaScript/TypeScript.
- **Native Features:** React Native supports a large ecosystem of native modules.

Technical strategy for migrating Casadoc from PHP to React Native

1. **Architectural Shift**

   Decoupling. Currently, our PHP code likely mixes logic and UI (e.g., fetching data from DB and immediately rendering an HTML table).

   - **Current State:** PHP $\rightarrow$ HTML/CSS (Browser renders UI)
   - **Future State:** PHP $\rightarrow$ JSON Data $\rightarrow$ React Native (Mobile renders UI)

2. **Category-Wise Migration Plan**

   - **The Backend (PHP): Transformation to API**

     We do not need to rewrite the backend logic (code), but we must change *how* it delivers data.

     - Create a new set of endpoints specifically for the mobile app.
     - **Authentication:**
       - *Current:* Likely uses Sessions/Cookies (`$_SESSION`).

        - *New:* Switch to **JWT (JSON Web Tokens)**. The app will send a "Token" with every request to prove who the user is.
    - **Response Format:**
        - Stop returning `view('profile.index', $data)`
        - Start returning `json_encode($data)` or `return response()->json($data)`

- **The Frontend (React Native): Complete Rewrite**

  Most of the work will go into the frontend development since we can't use the HTML/CSS given by PHP directly for rendering.

  UI Components:

  - HTML `<div>` becomes `<View>`
  - HTML `<span>` or `<p>` becomes `<Text>`
  - HTML `<img>` becomes `<Image>`
  - HTML `<button>` becomes `<TouchableOpacity>`

  Styling:

  - CSS is replaced by **StyleSheet** objects (JavaScript objects that look like CSS).

  Navigation:

  - Unlike the web app which uses URLs, mobile app uses Stack/Tab navigation. We'll use a library like **React Navigation** to move between screens.

- **The Database (DB)**
    - **Master DB:** Stays as MySQL/MariaDB on the server.
    - **Local Storage:** For features like caching, we'll use **AsyncStorage**. For heavy offline data (e.g., viewing documents offline), we'll use **SQLite** on the device.

3. **Example code comparison**

   A simple "User Profile" card change from PHP to React Native.

   - **Current PHP (Blade/Standard)**

```
<div class="user-card">
    <img src="<?php echo $user->avatar; ?>" />
    <h1><?php echo $user->name; ?></h1>
    <p><?php echo $user->email; ?></p>
</div>
```

- **New React Native (JSX)**

```
// UserProfile.js
import React from 'react';
import { View, Text, Image, StyleSheet } from 'react-native';

const UserProfile = ({ user }) => {
  return (
    <View style={styles.card}>
      <Image source={{ uri: user.avatar }} style={styles.avatar} />
      <Text style={styles.name}>{user.name}</Text>
      <Text style={styles.email}>{user.email}</Text>
    </View>
  );
};

// Styles are defined in JS, not CSS files
const styles = StyleSheet.create({
  card: { padding: 20, backgroundColor: '#fff' },
  name: { fontSize: 18, fontWeight: 'bold' }
});

export default UserProfile;
```

Tentative roadmap

Phase 1: API Foundation (PHP Side)

1. **Environment:** Set up a route group (e.g., `/api/v1/`) in the PHP project.
2. **Auth:** Implement an endpoint `/api/login` that accepts email/password and returns a JWT Token.
3. **Read-Only Data:** Create endpoints to *fetch* data (e.g., `GET /documents`).

Phase 2: The Basic Blueprint App (React Native Side)

1. **Setup:** Initialize the project using **React Native CLI** (for maximum control) or **Expo**.

2. **Navigation:** Build the "Skeleton" – Login Screen, Dashboard (List View), and Detail Screen.
3. **Connection:** Write a service function in JS to `fetch` data from the new PHP API and display it.

Phase 3: Native Features & Interactivity

1. **Camera:** Add a feature to "Scan Document" using a React Native Camera library.
2. **Upload:** Create a PHP endpoint to accept `POST` file uploads and wire it to the camera.
3. **Push Notifications:** (*optional*) Integrate Firebase (FCM) to notify users when a document status changes.