# Amhara Police HRIS: Project Documentation

Welcome! This document will help you get started with the **Amhara Police HRIS** project. Its purpose is to serve as a comprehensive Human Resources Information System for the Amhara Police. This guide provides a complete overview of setting up your environment and understanding the project's architecture and features.

---

## □ Getting Started

This section will guide you through getting the project running on your local machine.

- **Prerequisites**
  - Before you begin, you need to have a complete **Flutter development environment** set up. If you haven't done this yet, please follow the official guide from the Flutter team.
  - **Setup Guide**: Install Flutter

- **Installation and Setup**
  - Follow these steps to clone the repository and prepare the project for development.

  1. **Clone the repository**:

     ```
      git clone <repository-url>
     cd apc_hris
     ```

  2. **Clean and Install Dependencies**: This command cleans the project directory and then runs `flutter pub get`. The `pub get` command will fetch all the necessary dependencies and also automatically handle the localization setup.

     ```
     flutter clean && flutter pub get
     ```

  3. **Run Build Runner**: This command generates the necessary data classes and serialization logic using `freezed`. It's a crucial step for our state management and model classes.

     ```
     flutter pub run build_runner build --delete-conflicting-outputs
     ```

- **Localization Notes**
  - `flutter pub get` should automatically generate the necessary localization files.
  - If you ever find that localization strings are not updating correctly after making changes, you can manually run the following command as a backup to force regeneration:

    ```
    flutter gen-l10n
    ```

- **Running the Application**
  - To run the app on a connected emulator or a physical device, use the following command:

    ```
    flutter run
    ```

---

## Project Overview

The **Amhara Police HRIS** is a mobile application designed to serve as a comprehensive Human Resources Information System for the **Amhara Police**. Its primary goal is to digitize and streamline various HR processes, making them more efficient and accessible to all members of the police force.

---

## □ Key Features

The application includes a wide range of features to manage HR-related tasks:

- **Authentication**: Secure Sign Up, Login, and Forgot Password functionality.
- **Employee Profile Management**: Allows officers to view and manage their personal information, contacts, dependents, educational background, work experience, and spouse details.
- **Leave Management**: A complete system for requesting leave, viewing leave balance, and checking leave history.
- **Recruitment**: View announcements for new recruits and submit applications directly through the app.
- **Placement Management**: View placement announcements and submit applications for new postings.
- **Training Management**: Browse available training opportunities, submit applications, and file appeals.
- **Promotion Management**: Request promotions and track their status.
- **Incident Reporting**: A system for reporting and managing work-related incidents.
- **Employee Clearance**: A process for handling employee clearance procedures.
- **Employee Verification**: A quick verification system using QR code scanning.
- **Dashboard**: A central hub providing quick access to common actions and a summary of recent activity.

---

## □ Project Structure

The project is organized into a clean and scalable structure to facilitate development and maintenance.

*A visual representation of the directory structure:*

Absolutely! Here is the complete and detailed directory tree for the project.

```
apc_hris/
├── .gitignore
├── .metadata
├── .packages
├── analysis_options.yaml
├── android/
├── build/
├── ios/
├── l10n.yaml
├── lib/
│   ├── app.dart
│   ├── main.dart
│   ├── splash_screen.dart
│   ├── core/
│   │   ├── api/
│   │   ├── constants/
│   │   ├── di/
│   │   ├── errors/
│   │   ├── providers/
│   │   ├── routes/
│   │   ├── services/
│   │   ├── theme/
│   │   └── widgets/
│   │
│   ├── features/
│   │   ├── auth/
│   │   │   ├── application/
│   │   │   ├── data/
│   │   │   ├── domain/
│   │   │   └── presentation/
│   │   │
│   │   ├── clearance/
│   │   ├── dashboard/
│   │   ├── employee/
│   │   ├── employee_profile/
│   │   ├── home/
│   │   ├── incident/
│   │   ├── leave_mgmt/
│   │   ├── placement/
│   │   ├── promotion/
│   │   ├── training/
│   │   └── verification/
│   │
│   ├── l10n/
│   └── providers/
│
├── pubspec.lock
├── pubspec.yaml
└── README.md
```

- `main.dart` : The entry point of the application. This file initializes the app and sets up dependency injection.
- `app.dart` : This is the root widget of the application ( `MyApp` ). It configures the theme, router, and localization delegates.
- `splash_screen.dart` : The initial screen displayed to the user while the app is loading and checking authentication status.
- `l10n/` : Contains the localization files ( `.arb` files) for supporting English and Amharic languages.
- `core/` : This directory contains the foundational logic and shared components of the application.
  - **API**: Handles all network requests and communication with the backend.
  - **Constants**: Defines application-wide constants.
  - **DI (Dependency Injection)**: Manages dependency injection setup for the core module.
  - **Errors**: Defines custom exception and failure classes for error handling.
  - **Providers**: Contains global Riverpod providers that are used across multiple features.
  - **Routes**: Manages application routing and navigation using `go_router` .
  - **Services**: Contains business logic services (e.g., file picker, dialogs).
  - **Theme**: Defines the application's visual theme, including colors and text styles.

    - **Widgets**: Contains reusable custom widgets (e.g., buttons, text fields) used across the app.
- `features/`: This directory contains all the feature-specific modules of the application (e.g., authentication, profile, leave). Each feature is self-contained and typically follows a Clean Architecture pattern with its own `data`, `domain`, and `presentation` layers.
- `providers/`: This directory is dedicated to Riverpod providers that manage the application's state, organized by feature.

---

## State Management

The application uses **Flutter Riverpod** for state management. This approach allows for a reactive and scalable way to manage state across the application. We also use the **Freezed** package to generate immutable data classes and unions, which helps prevent state-related bugs and ensures a predictable state flow.

---

## API Integration

Communication with the backend API is handled using the **Dio** package, a powerful and reliable HTTP client for Dart. All API endpoints are centrally defined in the `ApiEndpoints` class located in the `core/constants/` directory. This makes it easy to manage and update API routes.