# **Solar Project Application Documentation**

This document provides a detailed overview of the Flutter-based Solar Project Application, covering its architecture, UI components, data models, backend services, and data flow.

## **1. Introduction**

The Solar Project Application is a mobile application designed to assist in managing various aspects of solar energy projects. It features daily progress tracking for module mounting systems, real-time cable reconciliation, and provides a structured authentication flow for users. The application emphasizes a clean UI/UX, responsiveness across devices, and robust data persistence using Firebase.

## **2. Application Architecture**

The application follows a modular and layered architecture, promoting separation of concerns, reusability, and maintainability.

lib/  
├── main.dart  
├── constants/  
│ ├── app\_constants.dart  
│ └── category\_colors.dart  
├── models/  
│ ├── cable\_entry.dart  
│ ├── icr\_info.dart  
│ └── mounting\_progress\_item.dart  
├── providers/  
│ └── todo\_provider.dart (Note: This was from a previous feature, might be removed if not used)  
├── screens/  
│ ├── auth\_screen.dart  
│ ├── auth\_wrapper.dart  
│ ├── cable\_schedule\_screen.dart  
│ ├── home\_screen.dart  
│ ├── icr\_info\_screen.dart  
│ ├── info\_screen.dart  
│ ├── module\_mounting\_screen.dart  
│ ├── module\_reconciliation\_screen.dart  
│ ├── sign\_in\_screen.dart  
│ ├── sign\_up\_screen.dart  
│ └── splash\_screen.dart  
├── services/  
│ ├── auth\_service.dart  
│ └── firestore\_service.dart  
├── utils/  
│ ├── date\_utils.dart  
│ └── dialog\_utils.dart  
├── widgets/  
│ ├── cable\_reconciliation\_card.dart  
│ ├── cable\_summary\_widget.dart  
│ ├── category\_dropdown.dart  
│ ├── date\_picker\_field.dart  
│ ├── feature\_card.dart  
│ ├── filter\_buttons.dart  
│ ├── progress\_input\_row.dart  
│ ├── responsive\_layout.dart  
│ └── todo\_list\_item.dart  
└── extensions/  
 └── color\_extension.dart

### **2.1 Core Directories and Their Responsibilities**

* **lib/main.dart**: The entry point of the Flutter application. It handles Firebase initialization, theme definition, and sets the initial route (AuthWrapper) for authentication flow.
* **lib/constants/**: Stores static, unchanging data such as application-wide constants (AppConstants) and color mappings (CategoryColors).
* **lib/models/**: Defines the data structures (Plain Old Dart Objects - PODOs) used throughout the application. Each model typically includes fromMap() and toMap() methods for JSON serialization/deserialization, essential for Firebase Firestore integration.
* **lib/providers/**: (Currently contains todo\_provider.dart from a previous feature. If not actively used for the solar app's core features, it can be removed or repurposed). This directory would typically hold ChangeNotifier classes for state management using the Provider package.
* **lib/screens/**: Contains the main UI pages (screens) of the application. Each file represents a distinct view or user interaction flow.
* **lib/services/**: Encapsulates business logic related to external services, primarily Firebase Authentication and Firestore database operations. This layer abstracts away the complexities of interacting with the backend.
* **lib/utils/**: Provides utility functions and helper classes that can be used across different parts of the application (e.g., date formatting, dialog helpers).
* **lib/widgets/**: Houses reusable UI components (widgets) that are used across multiple screens to maintain consistency and reduce code duplication.
* **lib/extensions/**: Contains Dart extensions to add new functionality to existing classes (e.g., ColorExtension for color manipulation).

## **3. Application Components**

### **3.1 Screens**

* **SplashScreen**:
  + **Purpose**: The initial loading screen displayed when the app starts.
  + **UI**: Features a light sky-blue background, a central card with "ReNew" branding (dark green 'Re', light green 'New'), "Daily Progress Report," and "Vendor" text. Uses GoogleFonts (Poppins/Roboto) for typography.
  + **Functionality**: Uses AnimatedOpacity for a fade-in effect. Displays for 2 seconds, then automatically navigates to AuthScreen.
* **AuthWrapper**:
  + **Purpose**: Acts as a gatekeeper immediately after the SplashScreen. It checks the user's authentication status and whether their ICRInfo is complete.
  + **Functionality**:
    - Listens to FirebaseAuth.instance.authStateChanges() to determine if a user is signed in.
    - If ConnectionState.waiting, it shows the SplashScreen.
    - If a user is authenticated (snapshot.hasData), it then queries FirestoreService().doesIcrInfoExist(userId) to check for mandatory ICR information.
    - If ICR info exists, it navigates to HomeScreen.
    - If ICR info is missing, it navigates to IcrInfoScreen, forcing completion of registration.
    - If no user is authenticated, it navigates to AuthScreen.
* **AuthScreen**:
  + **Purpose**: Presents the user with options to either sign in or sign up.
  + **UI**: Features the "ReNew" branding, a welcome message, and two prominent buttons: "Sign In" and "Sign Up".
  + **Functionality**: Navigates to SignInScreen or SignUpScreen respectively.
* **SignInScreen**:
  + **Purpose**: Allows existing users to log in using their email and password.
  + **UI**: A form with Email and Password text input fields, a "Sign In" button, and a "Don't have an account?" link to SignUpScreen. Includes loading indicator.
  + **Functionality**: Authenticates users via AuthService.signInWithEmailAndPassword(). On success, AuthWrapper handles the subsequent navigation (to HomeScreen or IcrInfoScreen).
* **SignUpScreen**:
  + **Purpose**: Allows new users to create an account with email and password.
  + **UI**: A form with Email, Password, and Confirm Password text input fields, a "Sign Up" button, and an "Already have an account?" link to SignInScreen. Includes loading indicator.
  + **Functionality**: Creates a new user account via AuthService.signUpWithEmailAndPassword(). **Crucially, upon successful account creation, it immediately navigates to IcrInfoScreen (using pushReplacement) to collect mandatory profile information.**
* **IcrInfoScreen**:
  + **Purpose**: Collects mandatory ICR (Installation Completion Report) details from the user as part of the extended registration process. Registration is incomplete until this form is submitted.
  + **UI**:
    - **Theme**: Light greenery color palette (#E8F5E9 background) with subtle wave patterns (CustomPaint).
    - **Input Fields**: Location (dropdown 1-56), Contact (10-digit numeric), 1200 GC (Full Table, Half Table - numeric), 500 GC (Full Table, Half Table - numeric), Dummy (numeric).
    - **Styling**: Attractive input fields with rounded borders (radius 12), subtle shadows, floating labels. Focused borders turn to a fresh green (#66BB6A).
    - **Layout**: Fields are grouped into containers for 1200 GC and 500 GC. A placeholder for "Add Your ICR Drawing Here" is at the bottom.
  + **Functionality**:
    - Uses GlobalKey<FormState> for validation.
    - **Validation**: All required fields (Location, Contact) must be filled. Numeric fields are validated for correct input type.
    - **Data Storage**: Upon "Complete Registration" (and successful validation), it collects all input values into an IcrInfo object and saves it to Firestore via FirestoreService.saveIcrInfo().
    - **Navigation**: After successful data storage, it navigates to HomeScreen (using pushReplacement).
* **HomeScreen**:
  + **Purpose**: The main dashboard of the application, providing access to different project modules.
  + **UI**: Displays four distinct "weights" (features) as tappable FeatureCard widgets: "Module Mounting System", "Cable Schedule", "Module Reconciliation", and "Info".
  + **Functionality**: Navigates to the respective screens upon tapping a card.
* **ModuleMountingScreen**:
  + **Purpose**: Allows users to update the daily progress of Module Mounting System components.
  + **UI**: A form displaying a list of items (Rafter, Purlin, etc.) with three columns: "Today's Progress" (input), "Cumulative Progress" (read-only), and "Total Scope" (read-only).
  + **Functionality**:
    - Fetches MountingProgressItem data from Firestore in real-time using StreamBuilder.
    - Allows users to input "Today's Progress".
    - On "Save Daily Progress", it updates the cumulativeProgress locally (simulating backend calculation) and then saves the updated MountingProgressItem to Firestore via FirestoreService.saveMountingProgressItem().
    - TextEditingControllers are dynamically managed for each input field.
* **CableScheduleScreen**:
  + **Purpose**: Facilitates real-time reconciliation of scheduled vs. actual cable cut lengths and minimizes wastage.
  + **UI**: Displays each cable entry in a CableReconciliationCard with SCB No, Inverter No, Scheduled Length, an input field for Actual Cut Length, and computed Wastage.
  + **Highlighting**: Wastage is highlighted: Red (negative), Green (positive), Grey (zero).
  + **Summary**: A CableSummaryWidget at the bottom shows "Total Scheduled", "Total Actual", and "Total Wastage".
  + **Functionality**:
    - Fetches CableEntry data from Firestore in real-time using StreamBuilder.
    - As the user types in "Actual Cut Length", the individual wastage and the overall summary update in real-time.
    - Changes are saved to Firestore via FirestoreService.saveCableEntry().
* **ModuleReconciliationScreen**:
  + **Purpose**: Placeholder screen for future module reconciliation features.
  + **UI**: Simple text and icon indicating its purpose.
* **InfoScreen**:
  + **Purpose**: Placeholder screen for general application information.
  + **UI**: Simple text and icon indicating its purpose.

### **3.2 Models**

* **MountingProgressItem (lib/models/mounting\_progress\_item.dart)**:
  + Properties: name (String), todayProgress (double), cumulativeProgress (double), totalScope (double).
  + Methods: fromMap(), toMap() for Firestore serialization; addTodayProgress() to update cumulative progress.
* **CableEntry (lib/models/cable\_entry.dart)**:
  + Properties: scbNo (String), icrNo (String), inverterNo (String), scheduledLength (double), actualCutLength (double).
  + Methods: fromMap(), toMap() for Firestore serialization; wastage (getter), getWastageColor() for UI logic.
* **IcrInfo (lib/models/icr\_info.dart)**:
  + Properties: location (int), contact (String), gc1200 (Map<String, int>), gc500 (Map<String, int>), dummy (int).
  + Methods: fromMap(), toMap() for Firestore serialization.

### **3.3 Services**

* **AuthService (lib/services/auth\_service.dart)**:
  + **Purpose**: Handles all user authentication operations with Firebase Authentication.
  + **Methods**:
    - user (Stream<User?>): Provides a real-time stream of the current user's authentication state.
    - signInWithEmailAndPassword(email, password): Authenticates an existing user.
    - signUpWithEmailAndPassword(email, password): Creates a new user account.
    - signOut(): Logs out the current user.
  + **Error Handling**: Includes try-catch blocks to provide specific error messages for common Firebase authentication exceptions.
* **FirestoreService (lib/services/firestore\_service.dart)**:
  + **Purpose**: Manages all interactions with Firebase Firestore, abstracting database operations from the UI.
  + **Methods**:
    - userId (getter): Retrieves the current authenticated user's UID.
    - getMountingProgressItems() (Stream): Fetches a real-time stream of mounting progress items for the current user.
    - saveMountingProgressItem(item) (Future): Saves/updates a single mounting progress item.
    - initializeDefaultMountingProgressItems(defaultItems) (Future): Populates the mounting progress collection with default data if it's empty (per user).
    - getCableEntries() (Stream): Fetches a real-time stream of cable entries for the current user.
    - saveCableEntry(entry) (Future): Saves/updates a single cable entry.
    - initializeDefaultCableEntries(defaultEntries) (Future): Populates the cable entries collection with default data if it's empty (per user).
    - **saveIcrInfo(icrInfo) (Future)**: **Crucial for registration**. Saves the IcrInfo object for the current user.
    - **doesIcrInfoExist(userId) (Future)**: **Crucial for AuthWrapper**. Checks if the IcrInfo document exists for a given user.

### **3.4 Widgets (Reusable UI Components)**

* **FeatureCard (lib/widgets/feature\_card.dart)**: A generic card widget used on the HomeScreen to represent a tappable feature, displaying a title and an icon. Includes basic responsiveness.
* **ProgressInputRow (lib/widgets/progress\_input\_row.dart)**: Displays a single item's progress in the ModuleMountingScreen, including an input field for "Today's Progress" and read-only fields for "Cumulative Progress" and "Total Scope".
* **CableReconciliationCard (lib/widgets/cable\_reconciliation\_card.dart)**: Displays a single cable entry in the CableScheduleScreen, including scheduled length, actual cut length input, and computed wastage with color highlighting.
* **CableSummaryWidget (lib/widgets/cable\_summary\_widget.dart)**: Displays the overall summary of scheduled, actual, and total wastage for all cable entries at the bottom of the CableScheduleScreen.
* **FilterButtons (lib/widgets/filter\_buttons.dart)**: (From previous To-Do app feature, might be unused in current solar app context). A reusable widget for displaying filter options as ChoiceChips.
* **CategoryDropdown (lib/widgets/category\_dropdown.dart)**: (From previous To-Do app feature, might be unused). A reusable dropdown for category selection.
* **DatePickerField (lib/widgets/date\_picker\_field.dart)**: (From previous To-Do app feature, might be unused). A reusable input field that triggers a date picker.
* **ResponsiveLayout (lib/widgets/responsive\_layout.dart)**: (From previous To-Do app feature, might be unused). A widget that renders different layouts based on screen width (mobile, tablet, desktop breakpoints).

### **3.5 Utilities & Extensions**

* **DateUtil (lib/utils/date\_utils.dart)**: Provides helper methods for date formatting, comparison, and calculations (e.g., isSameDay, formatDate).
* **DialogUtils (lib/utils/dialog\_utils.dart)**: Provides a utility method to display custom AlertDialogs, avoiding alert() for better UI control.
* **ColorExtension (lib/extensions/color\_extension.dart)**: An extension on Flutter's Color class to add a darken() method, useful for generating darker shades of a color for text or borders.

## **4. Backend Details (Cloud)**

The application leverages **Firebase** as its backend, providing robust and scalable services.

### **4.1 Firebase Authentication**

* **Purpose**: Manages user accounts and authentication.
* **Method Used**: **Email/Password Authentication**. Users can sign up with an email and password, and then sign in with those credentials.
* **Authentication Flow**:
  1. The main.dart initializes Firebase.
  2. The AuthWrapper is the first screen after the SplashScreen.
  3. AuthWrapper listens to FirebaseAuth.instance.authStateChanges().
  4. If a user is not authenticated, they are directed to the AuthScreen.
  5. From AuthScreen, users can choose SignInScreen or SignUpScreen.
  6. SignInScreen calls AuthService.signInWithEmailAndPassword().
  7. SignUpScreen calls AuthService.signUpWithEmailAndPassword().
  8. Upon successful sign-up, the user is automatically logged in, and then immediately navigated to IcrInfoScreen to complete their profile.

### **4.2 Cloud Firestore**

* **Purpose**: A flexible, scalable NoSQL cloud database used to store all application data.
* **Data Structure**: All user-specific data is stored under a top-level users collection, with each user having their own document identified by their Firebase User ID (UID).
  + **Root Collection**: users
  + **Document Path**: users/{userId} (where {userId} is the authenticated user's UID).
* **Data Stored (within users/{userId}):**
  + **ICR Information**:
    - **Path**: users/{userId}/icrInfo/user\_icr\_data
    - **Details**: This is a subcollection icrInfo under the user's document, containing a single document named user\_icr\_data. This document stores the IcrInfo (Location, Contact, GC tables, Dummy) collected during the extended registration process.
    - **Access**: Stored via FirestoreService.saveIcrInfo(). Checked for existence by FirestoreService.doesIcrInfoExist().
  + **Module Mounting Progress**:
    - **Path**: users/{userId}/mountingProgress/{itemName}
    - **Details**: A subcollection mountingProgress where each document represents a MountingProgressItem (e.g., 'Rafter', 'Purlin - 1'). The itemName is used as the document ID for easy retrieval and updates.
    - **Access**: Fetched via FirestoreService.getMountingProgressItems() (as a stream for real-time updates). Saved via FirestoreService.saveMountingProgressItem().
  + **Cable Entries**:
    - **Path**: users/{userId}/cableEntries/{scbNo}
    - **Details**: A subcollection cableEntries where each document represents a CableEntry. The scbNo is used as the document ID.
    - **Access**: Fetched via FirestoreService.getCableEntries() (as a stream for real-time updates). Saved via FirestoreService.saveCableEntry().

### **4.3 Firestore Security Rules**

**Crucial for Data Integrity and Security.** The following rules are applied to ensure that:

* Only authenticated users can read or write data.
* Users can *only* read from and write to their *own* data path (users/{userId}/...). They cannot access other users' data.

rules\_version = '2';  
service cloud.firestore {  
 match /databases/{database}/documents {  
 // Allow read/write for authenticated users only to their own user data  
 match /users/{userId}/{document=\*\*} {  
 allow read, write: if request.auth != null && request.auth.uid == userId;  
 }  
 }  
}

## **5. Data Flow During Signup and Initial Login**

1. **App Launch**: main.dart initializes Firebase.
2. **Splash Screen**: SplashScreen is displayed for 2 seconds.
3. **Authentication Check (AuthWrapper)**:
   * After the splash screen, AuthWrapper becomes active.
   * It listens to FirebaseAuth.instance.authStateChanges().
   * **Scenario A: User NOT Authenticated**: AuthWrapper detects no logged-in user and navigates to AuthScreen.
     + User selects "Sign Up" -> SignUpScreen.
     + User enters Email/Password and clicks "Sign Up".
     + AuthService.signUpWithEmailAndPassword() creates the Firebase Auth user.
     + **Immediately**, SignUpScreen navigates (pushReplacement) to IcrInfoScreen.
     + User fills ICRInfoScreen and clicks "Complete Registration".
     + IcrInfoScreen validates the form and calls FirestoreService.saveIcrInfo() to store data at users/{userId}/icrInfo/user\_icr\_data.
     + Upon successful save, IcrInfoScreen navigates (pushReplacement) to HomeScreen.
   * **Scenario B: User IS Authenticated (e.g., after initial sign-up or subsequent app launch)**:
     + AuthWrapper detects a logged-in user (userId).
     + It then calls FirestoreService.doesIcrInfoExist(userId).
     + **If doesIcrInfoExist returns true**: AuthWrapper navigates to HomeScreen.
     + **If doesIcrInfoExist returns false**: AuthWrapper navigates to IcrInfoScreen, forcing the user to complete their profile before accessing the main app features.
   * **Scenario C: User Signs In (after previously signing up)**:
     + From AuthScreen, user selects "Sign In" -> SignInScreen.
     + User enters Email/Password and clicks "Sign In".
     + AuthService.signInWithEmailAndPassword() authenticates the user.
     + The FirebaseAuth.instance.authStateChanges() stream (listened by AuthWrapper) emits the new authenticated user.
     + AuthWrapper then proceeds as in Scenario B, checking for ICRInfo and navigating to HomeScreen or IcrInfoScreen accordingly.

## **6. Responsive and Adaptive UI**

The application is designed with responsiveness in mind:

* **ThemeData**: Global ThemeData in main.dart provides consistent styling for text, buttons, and input fields across all screens, adapting well to different densities.
* **LayoutBuilder**: Used in screens like ModuleMountingScreen and widgets like ProgressInputRow to dynamically adjust layouts (e.g., column flex ratios) based on the available screen width, ensuring optimal display on mobile, tablet, and desktop.
* **SingleChildScrollView**: Used in forms (IcrInfoScreen, SignInScreen, SignUpScreen) to prevent overflow on smaller screens or with on-screen keyboards.
* **Container maxWidth and padding**: Widgets like FeatureCard and the splash screen's central card use maxWidth and appropriate padding to ensure they are well-proportioned and centered on larger screens while scaling down gracefully on smaller ones.
* **Expanded and Flexible**: Used extensively in Row and Column widgets to distribute space efficiently, ensuring elements resize proportionally.

This documentation provides a solid foundation for understanding the current state and capabilities of your Solar Project Application.