# Flutter Firebase Todo App - Project Summary

This document provides an overview of the Flutter Firebase Todo App that I developed.   
The app integrates Firebase Authentication, Shared Preferences, Provider state management,   
and API data fetching to create a seamless task management experience.

## 1. Project Overview

The Firebase Todo App is a cross-platform Flutter application that allows users to:  
- Register and log in using Firebase Authentication.  
- Manage personal to-do lists saved locally using Shared Preferences.  
- Fetch sample data (Todos and User info) from JSONPlaceholder API.  
- View user information through a dedicated profile interface.  
- Use Provider for efficient state management across the app.

## 2. Features Implemented

✅ Firebase Authentication:  
- User Signup, Login, and Password Reset using Firebase Auth.  
- User session handling and secure navigation flow.

-Made a Project in Firbase and made a User save the name and password and authenicating it by putting all firebase ApiKey and rest of the data in my project  
  
✅ Splash Screen:  
- Displays a welcome animation before redirecting users to the Home screen.  
  
✅ Home Screen:  
- Personalized welcome message with user's email.  
- Fetch and display user data via API.  
- View all API Todos.  
- Add, update, and delete personal tasks (stored locally per user).  
- Logout functionality that clears saved preferences.  
  
✅ TodosPage:  
- Fetches Todos list from JSONPlaceholder API.  
- Displays Todos with loading and error handling states.  
  
✅ UserProfilePage:  
- Fetches and displays detailed user profile data from API.  
- Includes avatar, name, email, address, and company info.  
  
✅ State Management:  
- Implemented using Provider (`TaskProvider`) for local task handling.  
- Tasks saved separately per user using `SharedPreferences`.  
  
✅ Data Persistence:  
- Tasks are stored locally using Shared Preferences to ensure offline access.  
  
✅ API Service Layer:  
- Centralized API integration using `http` package.  
- Handles error states and data parsing robustly.

## 3. Firebase Integration

Firebase is initialized using platform-specific configurations from `firebase\_options.dart`.  
The following Firebase features were used:  
- firebase\_core  
- firebase\_auth  
- cloud\_firestore (planned for future updates)

## 4. Packages Used

- firebase\_core  
- firebase\_auth  
- cloud\_firestore  
- provider  
- shared\_preferences  
- http

## 5. Project Flow

1. User lands on Login screen (`login.dart`).  
2. Option to Sign Up or Reset Password.  
3. On successful login, user navigates to Home (`home.dart`).  
4. From Home, user can:  
 - View API Todos (`todos\_page.dart`).  
 - Fetch sample user profile (`user\_profile\_page.dart`).  
 - Add/Delete/Mark tasks as done using Provider.  
5. Logout redirects back to Login screen.

## 6. Technical Highlights

- Used async/await and FutureBuilder for real-time asynchronous operations.  
- Implemented responsive and dark-themed UI using Flutter Material 3.  
- Applied strong error handling and input validation across all modules.  
- Structured code with separation of concerns for better maintainability.

## 7. Next Steps (Future Enhancements)

- Store tasks on Firebase Firestore for cloud-based persistence.  
- Add user-specific Firestore collections for real-time sync.  
- Integrate push notifications using Firebase Cloud Messaging.  
- Implement profile editing and image upload features.

## 8. Conclusion

This project demonstrates a complete Flutter-Firebase integration for a productivity app.  
It showcases user authentication, API interaction, local data management, and clean UI design.  
The foundation is now ready for future scalability with Firestore and cloud synchronization.