Flutter Drag & Drop Layout Builder – Project Documentation

1. Project Overview

This project is a **responsive layout builder** built using **Flutter**, **BLoC state management**, and **Firebase Firestore**. It allows users to visually design screens by dragging and dropping predefined widgets (A, B, C, D, etc.) into a single canvas (panel), resize them dynamically, and manage multiple layouts using tabs.

Each layout is stored as **JSON** in Firestore. When the user revisits the page, the saved layout is fetched from Firestore and rendered exactly as it was left. Any updates made to the layout are synced back to Firestore.

> ⚠ Important: All widgets exist inside **one canvas (panel)** — not stacked layers.

---

2. Key Features

- Responsive multi-layout support (mobile, tablet, desktop)
- Drag & drop predefined widgets into a canvas
- Resize widgets
- Single canvas (no stack-based layering)
- Multiple tabs for multiple layouts
- Real-time layout persistence using Firestore
- BLoC-based predictable state management

---

3. Tech Stack

- **Flutter** – UI framework
- **flutter_bloc** – State management
- **Firebase Firestore** – Layout persistence
- **JSON-based layout schema** – Widget configuration

---

4. Core Architecture

4.1 UI Layer

- Canvas (Layout Area)
- Widget Palette (A, B, C, D, etc.)
- Resize Handles
- Tabs for Layout Switching

4.2 State Management (BLoC)

- LayoutBloc
- LayoutEvents (Add, Move, Resize, Delete, Load, Save)
- LayoutState (layouts, activeTab, loading, error)

4.3 Data Layer

- Firestore Collection: `layouts`
- Document per user / page
- JSON schema stored per tab

---

5. Task-wise Project Breakdown (5 Parts)

PART 1: Project Setup & Architecture

Goal: Create a scalable and maintainable foundation

Tasks:

- Initialize Flutter project

- Configure Firebase for Flutter

- Setup Firestore

- Add required dependencies
  - flutter_bloc
  - firebase_core
  - cloud_firestore
- Define folder structure
  - presentation/
  - bloc/
  - models/
  - repository/

Deliverables:

- Firebase connected project
- Base BLoC architecture

▯ PART 2: Layout Data Model & Firestore Integration

Goal: Define how layouts are stored and retrieved

Tasks:

- Create WidgetModel (id, type, position, size, etc.)
- Create LayoutModel (tabId, widgets list)
- Define JSON serialization & deserialization
- Firestore repository
  - fetchLayouts()
  - saveLayout()
- Initial load on screen open

Deliverables:

- Fully working Firestore sync
- Layout persistence

▯ PART 3: Canvas & Drag-Drop System

Goal: Enable widget placement

Tasks:

- Create single canvas widget
- Widget palette side menu (A, B, C, D)
- Drag source from palette
- Drop target on canvas

Deliverables:

- Widgets can be added via drag & drop
- All widgets rendered in one panel

▯ PART 4: Resize & Layout Interaction

Goal: Allow user to resize and manipulate widgets

Tasks:

- Add resize handles (corners / edges)
- Update width & height on drag
- Maintain minimum size constraints
- Update layout state via BLoC
- Re-render canvas efficiently

Deliverables:

- Resizable widgets
- Smooth drag + resize UX

▯ PART 5: Multi-Tab Layout Management & Sync

Goal: Support multiple layouts

Tasks:

- Add tab UI
- Create new layout tab
- Switch active layout
- Load layout from Firestore on tab change
- Auto-save layout on update
- Handle conflicts / overwrite safely

Deliverables:

- Multiple layouts per user
- Persistent layout per tab

---

6. Data Flow Summary

1. App loads → fetch layouts from Firestore
2. Active tab layout rendered on canvas
3. User drags/resizes widgets
4. LayoutBloc updates state
5. Updated JSON saved to Firestore
6. On revisit → layout restored