Architecture & Restructuring Guide

This document outlines a high-level architectural restructuring plan to align your Flutter frontend with your Django backend, specifically focusing on **Role-Based Architecture** (Tasker vs. Poster).

1. Core Concept: Role-Based "Apps"

Currently, your screens are mixed. A [TaskDetailsScreen](file:///d:/Study%20Data/Semester%207/FYP-I/Source%20Code/Actual%20Project/Final%20App/lib/features/task_details/screens/task_details_screen.dart#17-271) handles both Poster and Tasker logic with `if (role == ...)` checks. This leads to spaghetti code and a disconnected feel.

Recommendation: Treat the app as two distinct "modes" or "mini-apps" that share a common data layer.

The Two Modes

1. **Poster Mode**: Focused on *management*. Creating tasks, reviewing bids, paying.
2. **Tasker Mode**: Focused on *discovery and execution*. Finding work, bidding, doing the job.

Folder Structure Refactor

Move away from flat feature folders. Group by *Role Context*.

```
lib/
├── features/
│   ├── shared/                # Use by BOTH roles
│   │   ├── auth/
│   │   ├── profile/
│   │   ├── chat/
│   │   ├── settings/
│   │   └── widgets/           # Generic UI (buttons, inputs)
│   │
│   ├── poster/                # ONLY for Poster Mode
│   │   ├── home/              # Dashboard: "My Posted Tasks" (Active/History)
│   │   ├── create_task/       # Post Task & Media Upload flow
│   │   ├── task_management/   # "Task Details" for a POSTER (View bids, edit task)
│   │   └── poster_profile/    # Public view of me as a poster?
│   │
│   └── tasker/                # ONLY for Tasker Mode
│       ├── home/              # Discovery: Feed / Map of available tasks
│       ├── my_bids/           # Active Bids (sent but not accepted)
│       ├── my_jobs/           # Assigned/In-Progress tasks
│       ├── task_view/         # "Task Details" for a TASKER (Bid, read desc)
│       └── job_execution/     # Active job view (Upload proof, complete)
```

2. Screen ↔ Backend Alignment

Your backend models ([Task](file:///d:/Study%20Data/Semester%207/FYP-I/Source%20Code/Actual%20Project/Api/tasks/models.py#25-137), [Bid](file:///d:/Study%20Data/Semester%207/FYP-I/Source%20Code/Actual%20Project/Api/tasks/models.py#139-188), [User](file:///d:/Study%20Data/Semester%207/FYP-I/Source%20Code/Actual%20Project/Api/accounts/models.py#59-143)) map clearly to these modes.

A. Task Alignment

The [Task](file:///d:/Study%20Data/Semester%207/FYP-I/Source%20Code/Actual%20Project/Api/tasks/models.py#25-137) model has a `status` field. This status dictates which *screen* calls the API.

Backend State	Poster Screen	Tasker Screen
`POSTED` / `BIDDING`	**My Posted Tasks (Active)** *API: /api/tasks/posted/?status=BIDDING*	**Task Feed / Map** *API: /api/tasks/feed/*
`ASSIGNED` / `IN_PROGRESS`	**My Posted Tasks (In Progress)** *API: /api/tasks/posted/?status=IN_PROGRESS*	**My Jobs (Active)** *API: /api/tasks/assigned/*
`COMPLETED` / `CONFIRMED`	**History**	**History / Earnings**

Key Takeaway:

- **Poster** cares about *Tasks I created*.
- **Tasker** cares about *Tasks I can do* (Feed) or *Tasks I am doing* (Assigned).
- **NEVER** mix these in one API call or one List Controller.

B. "Task Details" Alignment

Don't use one [TaskDetailsScreen](file:///d:/Study%20Data/Semester%207/FYP-I/Source%20Code/Actual%20Project/Final%20App/lib/features/task_details/screens/task_details_screen.dart#17-271). Create two:

1. `PosterTaskDetailScreen`:
    - **Goal**: Management.
    - **Data**: Fetches [Task](file:///d:/Study%20Data/Semester%207/FYP-I/Source%20Code/Actual%20Project/Api/tasks/models.py#25-137) + `List<Bid>`.
    - **Actions**: "Edit Task", "Cancel Task", "Accept Bid X".
    - **State**: "Bidding Open" (List of Bids), "Assigned" (Tasker info + Chat button).
2. `TaskerTaskDetailScreen`:
    - **Goal**: Sales/Action.
    - **Data**: Fetches [Task](file:///d:/Study%20Data/Semester%207/FYP-I/Source%20Code/Actual%20Project/Api/tasks/models.py#25-137) + `MyBid` (if exists).
    - **Actions**: "Place Bid", "Update Bid", "Upload Completion Proof".
    - **State**: "Open" (Bid form), "Pending" (My active bid), "Assigned to Me" (Job execution view).

3. Tasks & Bids Flow Design

Backend States → UI States

Backend Status	UI State (Poster)	UI State (Tasker)
`BIDDING`	Show "x New Bids". List of Bid Cards.	Show "Place Bid" button. Input field for amount/message.
`ASSIGNED`	Show Assigned Tasker Profile. "Chat" button.	Show Poster Profile. "Chat" button. "Start Task" / "Complete" buttons.
`COMPLETED`	Show "Confirm Completion" button.	Show "Waiting for Confirmation".

Avoiding Duplicate Logic

Use **Repositories** and **Shared Widgets**, not shared Screens.

- **Repository**: `TaskRepository` has methods `getTaskDetails(id)`. Both shared.
- **Widgets**: `TaskLocationCard`, `TaskMapPreview`, `PriceTag` are shared.
- **Screens**: `PosterTaskDetailScreen` and `TaskerTaskDetailScreen` are *orchestrators*. They use the same widgets but different layouts and logic.

Example: Both screens show the map.

- **Poster**: Sees map to verify location is correct.
- **Tasker**: Sees map to know where to drive. Same `TaskMapWidget`, different context.

4. Role Switching UX & Architecture

Since `User.role` can be `BOTH`, you need a global "Active Mode" state.

Architecture: `UserSessionService`

Create a persistent service (GetX Service) that holds:

1. `User currentUser`
2. `Rx<UserRole> activeRole` (The role they are *currently acting as*)
3. `RxList<String> permissions`

UX: The Switch

In **Settings** or the **Drawer**: "Switch to Tasker Mode".

What happens when switching?

1. **Update `activeRole`**: Update the global state variable.
2. **Navigation Reset**: `Get.offAllNamed(...)`.
    - If switching to **Tasker**: Go to `TaskerHome` (Feed).
    - If switching to **Poster**: Go to `PosterHome` (My Tasks).
3. **Invalidate Caches**: Clear `TaskController` lists. They are irrelevant in the new mode.

Why Reset Navigation?

Because the entire *mental model* changes. The "Back" button from "Tasker Home" shouldn't go to "Poster Settings". It's a clean break.

5. Backend Recommendations

To make the frontend simpler, add these endpoints/features to Django:

1. **Role-Aware Task Endpoints**:
    - Instead of generic `/api/tasks/`, generic filters are error-prone.
    - Create `/api/tasks/dashboard/posted/` -> Returns tasks the *current user* posted. Includes `bid_count`.
    - Create `/api/tasks/dashboard/assigned/` -> Returns tasks assigned to *current user*.
    - Create `/api/tasks/feed/` -> Returns *other people's* tasks (status=BIDDING, nearby).
2. **Computed Fields in Serializers**:
    - `my_bid`: When a Tasker views a task details, inject *their* bid into the response if it exists. Saves a second API call.
    - `can_bid`: Boolean. (e.g., false if already bid, or if task is arguably too far).
3. **Notification Triggers**:
    - Ensure creating a Bid fires a notification to the Poster.
    - Ensure Accepting a Bid fires a notification to the Tasker.
    - These drive the "real-time" feel without complex socket logic initially (just pull to refresh).