Assignment 2:

Dynamic LLM Monitoring Dashboard

Estimated Time: 12-18 hours

Background

Modern LLM observability platforms offer highly interactive and customizable dashboards. Users aren't just viewing static reports; they are building their own monitoring workspaces by adding, configuring, and arranging various charts and data widgets to get the specific insights they need.

At AryaXAI, we want to provide our users with a similar best-in-class experience. Your task is to build the core of this dynamic dashboard system.

Your Task

Develop a customizable, grid-based "LLM Monitoring Dashboard" using React and TypeScript. Users should be able to dynamically add, remove, resize, and rearrange various monitoring widgets (charts) on a grid. The layout and configuration of the dashboard must be persistent.

Core Requirements

1. **Dynamic Grid Layout:**
    - Implement a grid-based dashboard where users can freely drag, drop, and resize widgets.
    - Use a library like `react-grid-layout` to manage the grid system.
    - The dashboard should have an "Add Widget" button that opens a modal or sidebar. This menu should list available widget types (e.g., "Token Usage Over Time," "Latency Distribution," "Cost Analysis").
    - Selecting a widget from the menu should add a new, default-configured instance of that widget to the grid.
2. **Widget Data Fetching & Visualization:**
    - Each widget should fetch its own data independently. To make this assignment self-contained, you will create a mock data service.
    - Create a file (e.g., `src/services/mockApi.ts`) that exports asynchronous functions to simulate API calls. Each function should return a `Promise` that resolves with the specified data after a short delay (e.g., `setTimeout`).
    - Use **TanStack Query (React Query)** to call these async functions, managing the loading, success, and error states for each widget.
    - Use a charting library like **Recharts** or **Chart.js** to visualize the data within each widget. The chart type should match the data.
3. **Complex State Management:**
    - The state of the dashboard is complex. You need to manage:

        - The list of active widgets on the dashboard.
        - The layout of the grid (the position and size of each widget).
        - The individual configuration of each widget (e.g., which model it's tracking, the time range—you can mock this for now).
    - Use **Zustand** for centralized client-side state management of the dashboard's layout and widget configurations.
4. **Layout Persistence:**
    - The user's customized dashboard layout (the positions, sizes, and types of widgets) must be persistent.
    - When the grid layout changes (e.g., a widget is moved, resized, or removed), save the new layout configuration to **`localStorage`**.
    - When the application loads, it should check `localStorage` for a saved layout and restore it.
5. **Code Quality & Design Document:**
    - The entire application must be written in **React with TypeScript**, with robust typing for your state, API responses, and component props.
    - Write a `DESIGN.md` file explaining your architectural choices. This must include:
        - A description of your Zustand store's structure and why you designed it that way.
        - An explanation of how you managed the interaction between the grid layout library, your state, and individual widget components.
        - A discussion of potential performance bottlenecks and how you would address them.

Mock Data Details

Create an async function for each data type below.

- **Token Usage (Line Chart):**
    - **Function:** `fetchTokenUsage()`
    - **Data Structure:** `Array<{ timestamp: string; tokens: number; }>`
    - **Sample Data:**

None

```
[
  {"timestamp": "2023-10-01T10:00:00Z", "tokens": 1200},
  {"timestamp": "2023-10-01T10:05:00Z", "tokens": 1500},
  {"timestamp": "2023-10-01T10:10:00Z", "tokens": 1350},
  {"timestamp": "2023-10-01T10:15:00Z", "tokens": 1600}
]
```

- **Latency Distribution (Bar Chart):**
  - **Function:** `fetchLatencyDistribution()`
  - **Data Structure:** `Array<{ latency_ms: number; request_count: number; }>`
  - **Sample Data:**

```
None
[
  {"latency_ms": 100, "request_count": 50},
  {"latency_ms": 200, "request_count": 120},
  {"latency_ms": 300, "request_count": 80},
  {"latency_ms": 400, "request_count": 30}
]
```

- **Cost Analysis (Pie Chart or Bar Chart):**
  - **Function:** `fetchCostAnalysis()`
  - **Data Structure:** `Array<{ model_name: string; cost: number; }>`
  - **Sample Data:**

```
None
[
  {"model_name": "GPT-4", "cost": 450.75},
  {"model_name": "Claude 2", "cost": 320.50},
  {"model_name": "Llama 2", "cost": 150.25}
]
```

Tech Stack

- **Framework:** React with TypeScript
- **State Management:** Zustand
- **Server State:** TanStack Query (React Query)
- **Grid System:** `react-grid-layout`
- **Data Visualization:** Recharts (or similar)
- **Styling:** Tailwind CSS
- **Project Setup:** Vite or Create React App

Submission Method

1. Create a private GitHub repository for your solution.
2. The repository must contain all source code and a `README.md` file.
3. Your `README.md` should contain clear instructions on how to install dependencies and run the application locally.
4. Add the GitHub user vaibhav.gupta@arya.ai as a collaborator to your private repository.
5. Share the link to the repository along with a description of the approach to vaibhav.gupta@arya.ai and cc vinay@arya.ai.