Things to remember:

You've 1 week from the date of accessing the document to submit your assignment.
If you need an extension, please email us and ensure that you are submitting within that time.

Assignment:

Interactive Model Analysis Workbench

Estimated Time: 10-16 hours

Background

A core feature of the AryaXAI platform is the "Workbench," an interactive environment where data scientists can analyze models, run experiments, and document their findings. This environment is similar to a Jupyter Notebook but is deeply integrated with our platform's APIs and custom components. As a Senior React Engineer, you are tasked with building a high-fidelity prototype of this workbench.

Your Task

Develop an "Interactive Model Analysis Workbench" that allows users to create and manage notebooks, add and execute code cells, and view results in real-time. This assignment will test your ability to handle complex state, manage real-time data streams, and build a performant, large-scale application.

Core Requirements

1. **Notebook and Cell Management:**
    - The UI should allow users to create multiple notebooks.
    - Within a selected notebook, users can add, delete, and reorder code cells.
    - Implement **drag-and-drop** functionality to reorder cells within a notebook.
2. **State Management:**
    - The state of all notebooks and their cells is complex and shared across many components. Use **Zustand** to manage this client-side state.
    - Your state design should be efficient and well-structured, capable of handling many notebooks and cells without performance degradation.
3. **Code Execution and Real-time Output:**
    - For this assignment, you will run jupyter backend server.
    - We have a guide below about how to easily run jupyter backend server using Docker.
    - Create Notebook using jupyter backend api
    - Intialize jupyter kernel for code execution
    - Setup websocket to send updated cell code and run individual cells.
    - When a user clicks "Run" on a cell, you will:
        - Make a `POST` request to a Jupyter API endpoint with the cell's code to update it in the notebook.
        - Simultaneously, connect to a websocket and execute the cell.Upon connection, send the code to the WebSocket.
        - Display these real-time logs in the cell's output area as they arrive.

4. **Performance Optimization:**
    - A notebook can potentially have hundreds of cells. A naive `map()` render will be slow.
    - Implement **list virtualization** (windowing) for the cell list to ensure high performance, rendering only the visible cells. You can use a library like `react-window` or `tanstack-virtual`.
5. **Component Architecture & Design Document:**
    - Structure your application logically.
    - Write a `DESIGN.md` file explaining your architectural choices. This must include:
        - A description of your Zustand store's structure.
        - Your strategy for managing WebSocket connections and subscriptions.
        - An explanation of why you chose your component breakdown.

Tech Stack

- **Framework:** React with TypeScript
- **State Management:** Zustand
- **Server State:** TanStack Query (React Query) for the POST request
- **Styling:** Tailwind CSS
- **Drag & Drop:** A modern library like `dnd-kit`.
- **Virtualization:** A library like `react-window` or `@tanstack/react-virtual`.

Setup Jupyter Backend For API Call

- Install Docker and Docker Compose (Installation Guide)
- Clone the github repo

```shell
# Clone the JupyterHub deployment repository
git clone https://github.com/KnightKrusty/jupyter-backend-docker.git
cd jupyterhub-deploy-docker/basic-example
# Start the JupyterHub service using Docker Compose
docker-compose up
```

Access JupyterHub

- Open your browser and navigate to: http://localhost:8000/hub/signup
- Create an admin user with the following credentials: make sure username remain admin

```shell
{
"username": "admin",
"password": "yourpassword",
```

```
"confirmPassword": "yourpassword"
}
```

- After logging in, navigate to the Token (http://localhost:8000/hub/token)
- Token page to generate an authentication token for further use.

Jupyter API Documentation

Jupyterhub RestApi Base URL (http://localhost:8000/hub/api/info)
Jupyterhub ServerAPI http://localhost:8000/user/admin/api/content
Websocket Response Documentation

Websocket URL example:
`ws://localhost:8000/user/admin/api/kernels/539350a3-c517-46fc-8b63-0ccd081f0fec/channels?token=9d9daf0c50cc4c8b9e7517c1b00c8be2`

Submission Method

1. Create a private GitHub repository for your solution.
2. The repository must contain all source code, the `DESIGN.md` document, and a `README.md`.
3. Your `README.md` should contain clear instructions on how to install dependencies and run the application.
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.

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.