Engineering Challenge – Hotel APP

Thought Process

The goal was to build a modular dashboard builder using Angular that allows admin users to easily manage UI widgets. Here's a breakdown of the approach and decisions made:

1. Dashboard Layout Management

- Implemented a responsive dashboard layout where users can dynamically **add and remove widgets** using simple UI buttons.
- The layout leverages Angular's component-driven architecture to keep widget logic isolated and reusable.

2. Widget Types & Data Binding

- Developed support for two primary widget types:

  Chart Widget (Bar & Time Series) using **ng2-charts (Chart.js)** for flexibility and responsiveness.

  - **Bar Chart:** Displays KPIs like **Top 5 Room Types by Occupancy** and **Check-in Count by Weekday**.
  - **Time Series Chart:** Visualizes **Daily Check-ins over the past 14 days**, with a date-based X-axis and sample check-in values on the Y-axis.

- Designed the chart data structure to be easily extendable for future data sources or chart types.

3. LocalStorage Persistence

- Integrated **localStorage** to save the widget configuration and layout state.
- On reload, the dashboard restores the saved layout, ensuring a seamless user experience.

4. Responsive Design

- Ensured the dashboard and widgets adapt to different resolution using CSS Flexbox.

5. Unit Testing

- Added **unit test cases** for all Components and Services with 90% and above coverage.

6. Code Quality & Structure

- Maintained a clean folder structure (components, models, services, etc.) for separation of concerns.
- Used Angular best practices.

Improvements If I Had More Time

1. Drag-and-Drop Widget Layout

- Implement a flexible layout system using libraries like **Angular CDK DragDrop** to allow users to **reorder, move, and resize** widgets dynamically on the dashboard.

2. Advanced Widget Configuration

- Add a dedicated configuration panel for each widget with support to:
    - Switch between chart types (e.g., bar, line, pie)
    - Customize KPIs, labels, and chart colors
    - Apply data transformations such as rolling averages or aggregations

3. Dynamic Data Integration

- Replace static chart data with dynamic data fetched via **Angular services**.
- Integrate with **mock APIs or real endpoints** to simulate or display live KPIs.

4. Multi-language and Culture Support

- Add **internationalization (i18n)** using Angular's built-in i18n support or **ngx-translate**.
- Support **multiple languages** (e.g., English, Deutsch, Spanish, French) with JSON-based translation files.
- Implement **culture-aware formatting** for numbers, dates, and currencies based on the user's locale.

5. User Authentication & Multi-Dashboard Support

- Implement **login and role-based access control** for admin users.
- Allow users to **create, name, and manage multiple dashboards**, each with its own saved configuration.

6. Improved UI/UX

- Integrate a design system such as **Angular Material** or **PrimeNG** for a consistent and modern user interface.
- Ensure responsiveness using **CSS Flexbox and Tailwind**, providing smooth display across all screen sizes and devices.
- Add **animations and transitions** for enhanced interactivity (e.g., widget load/unload, chart transitions).

7. Export Capabilities

- Provide functionality to **export charts and dashboards** as **PDF or PNG**, enabling reporting and sharing outside the application.