Leadership & Planning Document

1. **Tailwind Configuration & Design Integration**
    a. Define global tokens: colors, spacing, typography, and responsive breakpoints.
    b. Integrate with an existing Figma design system (if available), ideally including a link to the style guide.
2. **System Component Implementation**
    a. Build reusable foundational components: `Modal`, `Button`, and `Toggle`.
    b. Ensure accessibility and theme compatibility (e.g., dark mode support using Tailwind's dark variant).
3. **State Management Setup**
    a. Integrate Redux Toolkit and define the initial store structure.
    b. Create slices for authentication and user data, applying good separation of concerns.
4. **Routing Architecture**
    a. Set up `react-router-dom` with route-based code splitting.
    b. Define public and protected routes based on user roles.
5. **User Dashboard Logic**
    a. Fetch and display user-specific data.
    b. Ensure the layout is modular and the dashboard supports additional widgets in the future.

Patterns, Hooks, and Context

- **Custom Hooks**
  Move business logic into reusable custom hooks to keep components clean and declarative.
- **Middleware for Side Effects**
  Create Redux middleware to handle side effects such as error handling, API logging, and toast notifications.
- **Best Practices**
    - Keep components atomic: one component per file.
    - Avoid deeply nested prop drilling by introducing React context where appropriate.
    - Write **JSDoc** and meaningful inline **comments** where necessary to improve clarity and maintainability.

Testing Strategy

- **Unit Tests**
  - Write unit tests for system components and utilities using React Testing Library.
  - Focus on props rendering, interaction behavior, and edge cases.
- **Integration Tests**
  - Cover major user flows and API interaction scenarios.
  - Test conditional rendering and route-based guards.

Extensibility & Scalability

- **Themes**
  Tailwind already supports theming via the `dark` variant and custom themes through the configuration file. Consider dynamic class toggling via a `ThemeContext`.
- **Roles & Permissions**
  Implement role-based access using Higher-Order Components (HOCs) or custom hooks like `usePermissionGuard`.
  If roles are stored in Redux, HOCs are preferred to avoid limitations with Storybook and testing isolated components.