1. Architecture Overview

The proposed architecture -centralizes UI components, utility functions, and business logic, ensuring consistency and reusability across multiple web projects. This architecture simplifies management, versioning, and sharing of common code.

Key Reasons for Choosing This Architecture:

- Its ensure uniformity across projects by using shared components and logic.

- For Reuse of code, reducing duplication and maintenance efforts.

-Easily scalable to add more components, utilities, and logic as needed.

- Centralized management of updates and bug fixes.

High-Level Directory Structure:

common-repository/

│

├── src/

│ ├── components/

│ ├── utilities/

│ ├── business-logic/

│

├── docs/

├── tests/

├── scripts/

├── package.json

├── README.md

Explanation:

- src/: Contains all source code for components, utilities, and business logic.

- docs/: Contains documentation for the repository.

- tests/: Contains unit and integration tests.

- scripts/:Contains build and deployment scripts.

- package.json: Manages dependencies and scripts.

2. UI Components

Components will be organized in individual folders within the `components/` directory. Each component will have its own files for logic, styling, and tests, ensuring separation of concerns and ease of maintenance.

Example: Button Component

src/

├── components/

│ ├── Button/

│ │ ├── Button.js

│ │ ├── Button.css

│ │ ├── Button.test.js

│ │ ├── index.js

Why This Structure:

- Keeps logic, styling, and testing separate.

- Easy to locate and update individual components.

- Simple to add new components.

Types of Utility Functions:

Utility functions will handle common tasks like data formatting, API requests, and error handling. These will be placed in the `utilities/` directory, each function in its own file for clarity and reusability.

3. Business Logic

Business logic will be organized in the `business-logic/` directory, focusing on data processing and state management. Keeping logic modular ensures it can be easily adapted for different projects.

src/

├── business-logic/

│ ├──data.js

│ ├── state.js

4. Best Practices Guide

Use semantic versioning (semver) for all packages.

Maintain detailed documentation using tools like JSDoc.

Write unit tests using Jest and React Testing Library.

Tools and Libraries:

- Lerna For monorepo management.

- Styled Components:For CSS-in-JS.

-Jest: For testing.

- ESLint: For code linting.

- Prettier For code formatting.