**ARCHITECTURE PROPOSAL  
  
  
1. Architecture Overview**

**High-Level Architecture:**

* **Core Components:**
  1. **ui-components**: Contains reusable UI elements.
  2. **utils**: Contains utility functions and helpers.
  3. **business-logic**: Contains shared business logic and state management.
* **Supporting Infrastructure:**
  1. **docs**: Documentation for usage and guidelines.
  2. **tests**: Unit and integration tests.
  3. **scripts**: Automation scripts for build, linting, and other tasks.
  4. **examples**: Example projects or demos showcasing component usage.

**Directory Structure:**

bash

Copy code

/common-repo

/ui-components

/Button

Button.js

Button.css

Button.test.js

index.js

/Modal

Modal.js

Modal.css

Modal.test.js

index.js

index.js

/utils

/api

api.js

api.test.js

/formatting

formatting.js

formatting.test.js

index.js

/business-logic

/data-processing

processData.js

processData.test.js

/state-management

stateManager.js

stateManager.test.js

index.js

/docs

usage.md

architecture.md

/tests

setupTests.js

/scripts

build.js

lint.js

/examples

/ButtonExample

ButtonExample.js

ButtonExample.css

/ModalExample

ModalExample.js

ModalExample.css

.gitignore

package.json

README.md

webpack.config.js

**2. UI Components**

**Organization:**

* Each UI component is housed in its own directory under ui-components.
* Component directories include:
  + **Component Logic**: Implementation file (Component.js).
  + **Styles**: Stylesheet (Component.css).
  + **Tests**: Unit tests for the component (Component.test.js).
  + **Index File**: An index.js file for easy imports.

**Customizability & Maintainability:**

* **Props and Configuration**: Design components to accept customizable props and use default values.
* **Styling**: Use CSS-in-JS or CSS Modules to avoid global styles and ensure encapsulation.
* **Documentation**: Include usage examples and documentation within each component directory.
* **Testing**: Implement comprehensive unit tests to ensure component stability.

**3. Utilities**

**Types of Utility Functions:**

* **Data Formatting**: Functions to format dates, numbers, etc. (e.g., formatDate, formatCurrency).
* **API Handling**: Functions to handle API requests and responses (e.g., fetchData, handleError).
* **Error Handling**: Utilities to handle and format errors (e.g., logError, formatErrorMessage).

**Structure:**

* Organize utilities by type in subdirectories within utils.
* Each utility module should have an index.js for easier import paths.
* Include tests alongside each utility to ensure reliability.

**4. Business Logic**

**Management:**

* **Data Processing**: Functions for transforming and processing data (e.g., processData).
* **State Management**: Shared state management logic or hooks (e.g., useGlobalState).

**Modularity & Adaptability:**

* **Separation of Concerns**: Keep data processing and state management logic separate for clarity.
* **Adaptable Interfaces**: Design functions and hooks with flexible interfaces to adapt to different project needs.
* **Documentation**: Document usage patterns and examples for business logic.

**5. Best Practices**

**Versioning:**

* Use [Semantic Versioning](https://semver.org/) to manage changes and updates.
* Maintain a changelog to document major updates and breaking changes.

**Documentation:**

* Provide clear and concise documentation for each module in the docs directory.
* Include usage examples, setup instructions, and contribution guidelines.

**Testing:**

* Implement unit tests for all components, utilities, and business logic.
* Use tools like Jest and React Testing Library for effective testing.

**Tools & Libraries:**

* **Build & Bundling**: Webpack or Vite for building and bundling code.
* **Linting**: ESLint for code quality.
* **Formatting**: Prettier for code formatting.
* **Testing**: Jest for testing, React Testing Library for component tests.

**Deliverables**

1. **Architecture Proposal**: Detailed document or presentation outlining the architecture.
2. **Example Folder Structure**: Sample directory layout provided above.
3. **Code Samples**:
   * **UI Component**: Button implementation example.
   * **Utility Function**: API handler example.
   * **Business Logic**: Data processing example.
4. **Best Practices Guide**: Brief guide including versioning, documentation, and testing practices.