

# Final Project Checklist

## Final Project Checklist

⚠ **Important:** Your final project must be an application of your own creation. A **todo list application is not an acceptable final project**, as it closely resembles the practice exercises from the curriculum. Choose a unique project idea that demonstrates your understanding of React concepts.

### General

- Uses a public Github repo.
- Scaffolded using Vite with the `react` (not `react-ts`, `react-swc`, or `react-swc-ts`) template.
- Uses NPM
- Installs and uses dependencies: “`react-router`”.

### Coding Practices

- Formatting should be neat and consistent across the codebase.  
Prettier can help with this!
- Only 1 component per file unless using Styled-Components.
- Component names should be in PascalCase and filenames should match the components they house.
- Minimize the use of implicit type coercion.
- Favor functional over non-functional approaches. (eg: use `array.prototype.map` instead of `array.prototype.forEach`)
- Comments should be concise and only used for explaining tricky or complex code passages. Remove all commented-out code and personal notes.
- Project files that contain only utility or helper functions and no components should be given the `.js` extension.

### Project Structure

#### Repo Structure

- Root directory contains:
  - `src/`
  - `.env.local.example` file with example values for all environmental variables needed to run project
  - `.gitignore` which includes at least the following entries:
    - `node_modules`
    - `dist`
    - `*.local` (this covers the `.env.local` file you use for secrets)
    - `.DS_Store` (invisible file for folder attributes used by macOS)
- `index.html` - the only changes permitted are in the `<head>` `</head>`

- 3rd-party stylesheets are permitted if used in conjunction with an installed library. All other styling should be in src/
- package.json
- package-lock.json
- vite.config.js
- README.md which includes:
  - Project title and description
  - Details on any added dependencies, especially those that may manipulate the DOM directly.
  - Instructions on how to install and run
  - Any details needed for an API connection
  - If credentials needed, indicate services used
- Root should NOT contain:
  - node\_modules/
  - .env.local or any other file with sensitive information
  - Any component files
  - any Yarn artifacts
- public/ - the favicon can be changed but this directory should not be used
- src/ directory contains at minimum:
  - assets/ directory for all included imagery, fonts, etc, unless they are retrieved from an external source.
  - features/ directory containing at least 2 features
    - If features use more than one component, all related components should reside in a sub-directory with the feature name.
  - pages/ directory containing at least 3 page components
  - shared/ directory containing at least 2 components that are used in more than one feature
  - App.css
  - App.jsx
  - main.jsx
- Other directories may be added so long as they assist in keeping the project's code well-organized.

## Project Data Schema Structure

- use any approach accessible to you (look back to the discussion) to create 1 or more objects or arrays of objects to load into state or save to state
- use the simplest structures needed to model data in your application

## Demonstrates Understanding of React Concepts

- The browser's page should never refresh during user interaction.
- All components should be functional (no class-based components).
- Use only React-compatible props.
- State should never be mutated.
- Components should return valid JSX.
- The DOM should never be directly accessed or manipulated unless required by a 3rd-party library.
  - Make a note of any libraries that do this in the README.
- All communication with external data sources should be done asynchronously.
- Project uses at least:
  - 1 component that takes children props

- 2 re-usable components each containing 2 or more html elements/sub-components + uses props
- 4 conditionally rendered components or elements.
- 1 controlled component form with at least 1 validated field.
- 2 useEffect calls.
- 1 useCallback.
- All dependency arrays for hooks are accurate for their use case.
- useEffect calls should return a cleanup function as appropriate.
- Any array of rendered components must include a unique key props.
- Keys must not be derived from the item's index.

## Uses React-Router for Routing

- react-router is installed in the project.
- The App component instance in main.jsx is wrapped with a BrowserRouter instance.
- Includes at least 2 routes.
- All Route instances use components in the pages/ directory for their element props.
- Include a wildcard route with a “Not Found” page.
- Uses NavLink instances for global navigation (can use Link instances elsewhere)

## Behavior

### Startup

- Installs without error (other than minor package updates)
- Application starts without errors.
- On loading, application performs a network request or interacts a browser storage mechanism to retrieve data used in app.
  - Loading status is displayed to user in UI.
  - Reviewers need to be able to access whatever resource is used with minimal setup!*
  - Any publicly accessible APIs used must be open for anonymous use or free to sign up for.
  - If a local server is used:
    - Warning: mentors will not be able to assist with troubleshooting any server issues** so this option is best for those with adequate experience!
    - it must use Node.js as a runtime (no Deno, Bun, Python, Ruby, PHP etc.)
    - it must run error-free
    - A link to its repo and setup/running instructions are included in the project's README

### Functionality

- All components and any user interactions should be error-free (excluding anything beyond student's control, such as API uptime). Warnings are acceptable.
- The app should never crash.
- StrictMode must remain in place in main.jsx
- Form inputs and labels must be properly associated with each other.

- Any foreseeable network or process errors must be caught and communicated to the user, as appropriate, through the UI.
- App allows user to interact with data central to the purpose of the app.
  - Create
  - Read
  - Update
  - Delete (optional)
- Persists data using an API and/or Local Storage or IndexedDB.

## **Appearance/UX**

- Styling should only be written using CSS, CSS Modules, or Styled Components. No component or theming libraries.
  - Exceptions can be made for notification systems - seek CIL approval first.
- Uses consistent theming and layouts across pages and elements.
- Uses a different font for headings and non-heading text.
- Interface text is legible for the typical user.
- Images must include brief, descriptive alt text (this excludes images that serve only as decoration).
- Any sounds used must be mutable from within the app's interface.
- NavLink instances should visually differentiate between the currently active route's link and other, inactive route links.
- Imagery and other assets should generally be optimized for the sizes that they are being used. (eg: don't use a 4k resolution, 2MB jpg for a 5cm tall user avatar!)