Electron With React Project Structure

React renderer with Vite. Organized IPC patterns for structured main-renderer communication.

#electron #react #vite #desktop #typescript

PNG PDF Copy Prompt

Project Directory

```
my-electron-app/
  package.json
  vite.config.js  Renderer bundli…
  electron-builder.json  Build config
  src/
    main/  Main process
      index.js  App entry
      window.js  Window creation
      ipc/  IPC handlers
        index.js  Register handle…
        files.js
        dialog.js
      services/
        store.js  electron-store
    preload/  Preload scripts
      index.js  Expose safe APIs
    renderer/  React app
      index.html
      main.jsx  React entry
      App.jsx
      components/
        Header.jsx
        Sidebar.jsx
      hooks/
        useIPC.js  IPC hook
      styles/
        index.css
  resources/  Build resources
    icon.icns
    icon.ico
    icon.png
```

Why This Structure?

Modern Electron with React renderer and Vite for fast HMR. The `src/` folder cleanly separates main, preload, and renderer. IPC handlers are organized by domain. Preload script bridges main and renderer securely.

Key Directories

`src/main/` - Node.js main process and IPC handlers

`src/preload/` - Contextbridge API exposure

`src/renderer/` - React app with Vite bundling

`src/main/ipc/` - IPC handlers grouped by feature

`resources/` - Platform-specific build assets

IPC Pattern

```
// src/main/ipc/files.js
ipcMain.handle('files:read', async (event, path) => {
  return fs.promises.readFile(path, 'utf-8');
});

// src/preload/index.js
contextBridge.exposeInMainWorld('api', {
  readFile: (path) => ipcRenderer.invoke('files:read', pat
});

// src/renderer/hooks/useIPC.js
export const useFiles = () => ({
  readFile: window.api.readFile
});
```

Getting Started

1. Use `electron-vite` or `vite-plugin-electron`
2. Create `src/main/`, `src/preload/`, `src/renderer/` structure
3. Configure Vite for multiple entry points
4. Set up preload with `contextBridge`
5. `npm run dev`

When To Use This

- Production desktop apps
- Apps with complex UI requirements
- Teams familiar with React ecosystem
- Apps needing hot reload during development

Best Practices

- Always use preload for IPC—never expose `ipcRenderer` directly
- Group IPC handlers by domain (files, dialog, settings)
- Use `handle/invoke` for async, `on/send` for events
- Keep main process lean—delegate to services
- Use electron-builder for distribution

Naming Conventions

`IPC channels` - Namespaced: `files:read`, `dialog:open`

`Handlers` - Match feature domain: `files.js`, `dialog.js`

`Preload APIs` - Group under `window.api` or `window.electron`