# Porting React to Electron with Preload

Best of Both Worlds

## Safe Electron Pattern: preload.js

For security, we need to separate the React (UI) process & main process (Node.js runtime).

- For this, we can use a minimal preload bridge (preload.js) and validated IPC handlers.
- Using this, the UI process can safely do local file access and other allowed actions through IPC.

## Core Principles

- Keep the renderer unprivileged: `nodeIntegration: false, contextIsolation: true, sandbox: true`
- Preload exposes only a small, whitelisted API
- Main process performs and validates privileged actions
- Use file dialogs instead of accepting arbitrary file paths

## 1. Update main.js

We can access local file (history) without security risks.

```javascript
const { app, BrowserWindow, ipcMain, dialog } = require('electron');
const path = require('path');
const fs = require('fs');

// Path to store calculation history
const historyPath = path.join(app.getPath('userData'), 'calc-history.json');

function createWindow() {
  const win = new BrowserWindow({
    width: 400,
    height: 700,
    webPreferences: {
      nodeIntegration: false,
      contextIsolation: true,
      preload: path.join(__dirname, 'preload.js') // <-
    }
  });
```

## IPC (Interprocess Communication)

In Electron, the **renderer** (React app/JavaScript) and the **main process** (Node.js runtime) are **separate processes**.

- They communicate using **IPC** — *Inter-Process Communication*.

```
React (Renderer)  →  preload.js  →
ipcRenderer  →  ipcMain  →  Main Process
```

- The **renderer** runs like a web page — safe, sandboxed.
- The **main process** runs with full system access (files, OS APIs).
- To keep security, they **cannot call each other directly**.
- Instead, they send **messages** through IPC channels.

## IPC Handlers

We can allow file access such as save-history through IPC.

- Runs in the main process → has full Node access.
- Uses IPC channel (save-history) from the preload bridge.
- Renderer never touches fs directly.
- Wrapped in try/catch to handle errors.

```
// IPC Handlers
ipcMain.on('save-history', async (event, history) => {
  try {
    fs.writeFileSync(historyPath, JSON.stringify(history, null, 2));
  } catch (error) {
    console.error('Error saving history:', error);
  }
});
```

```
ipcMain.handle('load-history', async () => {
  try {
    if (fs.existsSync(historyPath)) {
      const data = fs.readFileSync(historyPath, 'utf8');
      return JSON.parse(data);
    }
    return [];
  } catch (error) {
    console.error('Error loading history:', error);
    return [];
  }
});

ipcMain.handle('get-version', () => {
  return app.getVersion();
});

ipcMain.on('clear-history', async () => {
  try {
    if (fs.existsSync(historyPath)) {
      fs.unlinkSync(historyPath);
    }
  } catch (error) {
    console.error('Error clearing history:', error);
  }
});
```

## 2. preload.js (Bridge)

`preload.js` runs before your web app loads and acts as a secure bridge between:

- **Renderer (React/UI)** — unprivileged, browser-like
- **Main process** — full Node.js and system access

It safely exposes selected features from the main process without giving full access to Node or Electron internals.

It exposes protected methods that allow the renderer process to use.

```
const { contextBridge, ipcRenderer } = require('electron');

contextBridge.exposeInMainWorld('electronAPI', {
  // Save calculation history to a file
  saveHistory: (history) => ipcRenderer.send('save-history', history),

  // Load calculation history from file
  loadHistory: () => ipcRenderer.invoke('load-history'),

  // Get app version
  getVersion: () => ipcRenderer.invoke('get-version'),

  // Clear history
  clearHistory: () => ipcRenderer.send('clear-history')
});
```

## 3. Update App.js to use the IPC APIs

Add more state:

```
// src/App.js
import React, { useState, useEffect } from 'react';

function App() {
  ...
  const [history, setHistory] = useState([]);
  const [version, setVersion] = useState('');
```

## Use IPC APIs in the useEffect (or any other places):

```
useEffect(() => {
  // Check if running in Electron
  if (window.electronAPI) {
    // Get app version
    window.electronAPI.getVersion().then(v => setVersion(v));

    // Load calculation history
    window.electronAPI.loadHistory().then(h => {
      if (h && Array.isArray(h)) {
        setHistory(h);
      }
    });
  }
}, []);
```

Save history whenever it changes.

```
useEffect(() => {
  if (window.electronAPI && history.length > 0) {
    window.electronAPI.saveHistory(history);
  }
}, [history]);
```

## In calculate (Business Logice)

Run clearHistory IPC API:

```
const calculate = () => {
...

  // Add to history
  if (typeof res === 'number') {
    const calculation = {
      num1: n1,num2: n2, operation, result: res,
      timestamp: new Date().toLocaleString()
    };
    setHistory([calculation, ...history.slice(0, 9)]); // Keep last 10
  }
};

const clearHistory = () => {
  setHistory([]);
  if (window.electronAPI) {
    window.electronAPI.clearHistory();
  }
};
```

## Key Integration Points

1. **preload.js**
   - Safe bridge between React & Electron:
   - Exposes specific APIs only
2. **IPC Communication**
   - `ipcRenderer` in preload
   - `ipcMain` in main process
3. **Conditional Code**

```
if (window.electronAPI) {
  // Desktop features
} else {
  // Web fallback
}
```

## Benefits Achieved

React Development Experience

- Hot reload still works
- Component architecture
- Modern tooling

Desktop Capabilities

- File system access
- Native dialogs
- System tray
- Auto-updates

## Common Features to Add to Electron App

Native menu

```
const { Menu } = require('electron');
Menu.setApplicationMenu(menu);
```

System notifications

```
new Notification({
  title: 'My App',
  body: 'Task completed!'
}).show();
```

Global shortcuts

```
globalShortcut.register('Ctrl+Shift+I', () => {
  console.log('Shortcut pressed');
});
```