Porting React to Electron

Best of Both Worlds

`electron/electron_porting_without_preload`

`electron/electron_porting`

Why React + Electron?

- React's component system
- Modern development experience
- Desktop app capabilities
- File system access
- Native menus & notifications

Conversion Steps

1. **Keep React app as-is**
2. **Add Electron wrapper**
3. **Point Electron to React**
4. **Add desktop features**

Electron Porting (Without Preload) Example

New Project Structure

```
electron_porting_without_preload/
├── index.html
├── main.js
├── package.json
├── src
│   ├── App.css
│   ├── App.jsx
│   └── main.jsx
└── vite.config.js
```

1. Updated package.json

```
{
  "name": "my-electron-react-app",
  "main": "main.js",
  "homepage": "./",
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  },
  "devDependencies": {
    "electron": "^28.0.0",
    "concurrently": "^8.0.0"
  },
  "scripts": {
    "start": "concurrently \"npm run react\" \"npm run electron\"",
    "react": "react-scripts start",
    "electron": "wait-on http://localhost:3000 && electron .",
    "build": "react-scripts build",
    "dist": "npm run build && electron-builder"
  }
}
```

2. main.js (Electron)

With `npm start`, the `dist` directory contains all the JavaScript files including the index.html.

```
const { app, BrowserWindow } = require('electron')
const path = require('path')

function createWindow() {
  const win = new BrowserWindow({
    width: 400,
    height: 600,
    webPreferences: {
      nodeIntegration: false,
      contextIsolation: true
    }
  })

  // Load the built React app from dist folder
  win.loadFile(path.join(__dirname, 'dist', 'index.html'))
}
```

The dist directory

```
dist
├── assets
│   ├── index-afa94370.css
│   └── index-eaaccfef.js
└── index.html
```

All the React library code and App business/UI logic is compiled in one JavaScript file (index-eaaccfef.js in this example)

```
<head>
  <meta charset="UTF-8">
  <script type="module" crossorigin src="./assets/index-eaaccfef.js"></script>
  <link rel="stylesheet" href="./assets/index-afa94370.css">
</head>
```

Electron Window Lifecycle management.

```
app.whenReady().then(() => {
  createWindow()
  app.on('activate', () => {
    if (BrowserWindow.getAllWindows().length === 0) {
      createWindow()
    }
  })
})

app.on('window-all-closed', () => {
  if (process.platform !== 'darwin') {
    app.quit()
  }
})
```

3. App.jsx

We can use the same App.jsx.

```
import { useState } from 'react'

function App() {
  const [num1, setNum1] = useState('')
  const [num2, setNum2] = useState('')
  const [operation, setOperation] = useState('+')
  const [result, setResult] = useState(null)

  const calculate = () => {
    const n1 = parseFloat(num1)
    const n2 = parseFloat(num2)

    if (isNaN(n1) || isNaN(n2)) {
      setResult('Invalid input')
      return
    }
```

Development Workflow

```
# Install all dependencies
npm install

# Start both React and Electron
npm start

# React runs on http://localhost:3000
# Electron loads React from localhost
```

Production Build

```
# 1. Build React app
npm run build

# 2. Package with Electron
npm run dist

# Output:
# dist/
#   ├── MyApp.exe (Windows)
#   ├── MyApp.app (macOS)
#   └── MyApp.AppImage (Linux)
```