Electron.js

- Front end framework developed to produce desktop apps
- Uses node.js and chromium

Key Features

- **Cross-Platform Compatibility:**
  - Build desktop applications that run on multiple operating systems.
- **Uses Familiar Web Technologies:**
  - HTML, CSS, and JavaScript
- **Development of Native UI Components:**
  - Provides access to native user interface components through platform-specific APIs.
- **Node.js Integration:**
  - Enables applications to implement server-side logic and access system resources incl. file system.

Competitor	Difference	Advantage	Disadvantage
NW.js	NW.js integrates Chromium directly within Node.js.	Tighter integration of processes.	May lead to suboptimal resource allocation.
Qt	Tailored for C++ development.	More flexibility and performance.	Steeper learning curve.
JavaFX	Tailored for Java development.	Develop highly portable applications that can run on any platform with a Java Runtime Environment installed.	Steeper learning curve.

Tools used alongside Electron

- React.js / Vue.js
- Electron React Boilerplate / Vue Boilerplate
- Electron Forge
- Electron Builder
- Electron Packager
- Electron Debug

Deployment

1. Initialise your project and install Electron

```
npm install electron
```

2. Create a js file or react.js

3. Package your application

Packaging your application

- Electron Packager

```
electron-packager <directory> <appname> --platform=<platform> --arch=<architecture> [optional flags]
```

E.g.

```
electron-packager . MyApp --platform=win32 --arch=x64
```

- Electron Builder
    - Configure your packagaing : package.json
    - Run the packaging

```
npx electron-builder
```

Examples of applications that used Electron

- Microsoft Teams
- Slack
- VSCode

- Discord
- 1password
- Notion
- Obsidian
- Skype

Case Study: A Collaboration Tool with Real-Time Communication Features

- Consistent user experience across different operating systems.
- Modern, responsive, and visually appealing UIs using frameworks like React.
- Node.js on the backend for communication protocols, user sessions, and real-time data.
- Node.js integration enables features like file uploading/downloading, network connectivity checks, and desktop notifications.

Under the hood

What is happening when you run "electron ."

1. **Initialisation:**
    - Electron initialises its runtime environment, which is essentially a combination of Chromium and Node.js.
    - Chromium is the open-source browser project that Google Chrome is based on.
    - Node.js is a JavaScript runtime that allows you to run JavaScript on the server side.
2. **Main Process:**
    - Electron applications have a main process, which is essentially the entry point for the application.
    - The main process is responsible for creating and managing application windows and controlling the lifecycle of the application.
3. **Main Window:**
    - The main process creates the main window for your application.
    - This window is an instance of a Chromium browser window that will render your HTML, CSS, and JavaScript content.

Main vs Renderer process

- Electron's main process is a Node.js environment that has full operating system access. You can access Node.js built in features as well as any imported packages. (BACKEND)
- Electrons renderer process runs web pages and do not run Node.js by default for security reasons. (FRONTEND)
- To bridge Electron's different process types together, we will need to use a special script called a **preload**.