Electron Cheat Sheet

A comprehensive cheat sheet for building cross-platform desktop applications with Electron.

Core Concepts

Electron Architecture

Electron combines Chromium and Node.js to create desktop applications using web technologies (HTML, CSS, JavaScript).

It consists of two main processes: the **Main Process** and the **Renderer Process**.

Main Process

Description:	Controls the application lifecycle, creates and manages browser windows, and handles native OS interactions.
Responsibilities:	Creating menus, dialogs, registering global shortcuts.
Entry Point:	`main.js` (or specified in `package.json`)
Modules:	`app`, `BrowserWindow`, `Menu`, `Tray`, `dialog`

Renderer Process

Description:	Handles the application's UI. Each browser window runs in its own renderer process.
Responsibilities:	Rendering HTML, CSS, JavaScript. Interacting with the DOM.
Modules:	`remote`, `ipcRenderer`, `webFrame`

Inter-Process Communication (IPC)

Used for communication between the Main and Renderer processes. `ipcMain` (in the Main process) listens for events, and `ipcRenderer` (in the Renderer process) sends them.

Example:

```
// Main Process
ipcMain.on('async-message', (event, arg) => {
  event.reply('async-reply', 'pong')
});

// Renderer Process
ipcRenderer.send('async-message', 'ping');
ipcRenderer.on('async-reply', (event, arg) => {
  console.log(arg); // prints 'pong'
});
```

Essential Modules & APIs

App Module

`app.on('ready', callback)`	Emitted when Electron has finished initializing and is ready to create browser windows.
`app.quit()`	Quits the application.
`app.getPath(name)`	Gets a path to a special directory or file (e.g., 'userData', 'temp').
`app.getVersion()`	Returns the application version from `package.json`.

BrowserWindow Module

`new BrowserWindow([options])`	Creates a new browser window.
`win.loadURL(url)`	Loads a URL into the window.
`win.loadFile(filePath)`	Loads a local HTML file into the window.
`win.webContents.openDevTools()`	Opens the DevTools.
`win.close()`	Closes the window.

Menu Module

`Menu.buildFromTemplate(template)`	Creates a menu from a template array.
`Menu.setApplicationMenu(menu)`	Sets the application menu.
Menu Template	Defines the structure of the menu (labels, roles, submenus).

Working with Files and Dialogs

Dialog Module

`dialog.showOpenDialog([browserWindow,]options)`	Displays an open dialog to select files or directories.
`dialog.showSaveDialog([browserWindow,]options)`	Displays a save dialog to choose a file path.
`dialog.showMessageBox([browserWindow,]options)`	Displays a message box.
Options	title, defaultPath, button labels, filters (for file types)

File System Access

Use Node.js's `fs` module to read and write files. Ensure file access is secure and user-permissions are handled correctly.

Example:

```
const fs = require('fs');
fs.readFile('path/to/file.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log(data);
});
```

Context Menu

Context menus can be added with `electron-context-menu` or custom implementations using `Menu` and `webContents.on('context-menu', ...)`.

Example using electron-context-menu:

```
require('electron-context-menu')({
  prepend: (params, browserWindow) => [{
    label: 'Search Google for “{selection}”',
    visible: params.selectionText.trim().length > 0,
    click: () => {

shell.openExternal(`https://google.com/search?q=${encodeURIComponent(params.selectionText)}`);
    }
  }]
});
```

Packaging and Distribution

Electron Forge

A comprehensive tool for packaging and distributing Electron applications.
It simplifies the process of creating installers for various platforms (Windows, macOS, Linux).

Install:

```
npm install --global electron-forge
```

Create a new project:

```
electron-forge create my-new-app
```

Package:

```
npm run package
```

Make (create distributable):

```
npm run make
```

Electron Builder

Another popular tool for packaging Electron applications.
It supports many platforms and formats.

Install:

```
npm install --save-dev electron-builder
```

Configuration in `package.json`:

```
"build": {
  "appId": "com.example.app",
  "productName": "MyApp",
  "copyright": "Copyright © 2024 Example",
  "mac": {
    "category": "public.app-category.utilities"
  },
  "win": {
    "target": ["nsis"]
  }
}
```

Package:

```
npm run dist
```

Code Signing

Important for distributing applications on macOS and Windows.
Code signing verifies the identity of the developer and ensures that the application has not been tampered with.

Use appropriate certificates for each platform.