Electron

Allgemeines:

Electron (ehemals Atom Shell) ist ein von GitHub entwickeltes, quelloffenes Framework. Es ermöglicht die Ausführung von Cross-Platform-Desktop-Anwendungen mithilfe des Webbrowsers Chromium und des Node.js-Frameworks. Die Atom Shell (jetzt Electron) wurde als Basis für den Editor Atom von GitHub entwickelt.

Mithilfe von Electron können in HTML, CSS und JavaScript entwickelte Desktop-Anwendungen ausgeführt werden. Es können daher auch Frameworks wie Vue.js oder Angular benutzt werden.

Außerdem bietet Electron uns Zugang zu Betriebssystemressourcen und spezifischen Plattformfunktionen und unterstützt tausende von JavaScript-Bibliotheken und Programmen, die Sie mit dem Node.js-Teil der Anwendung verwenden können.

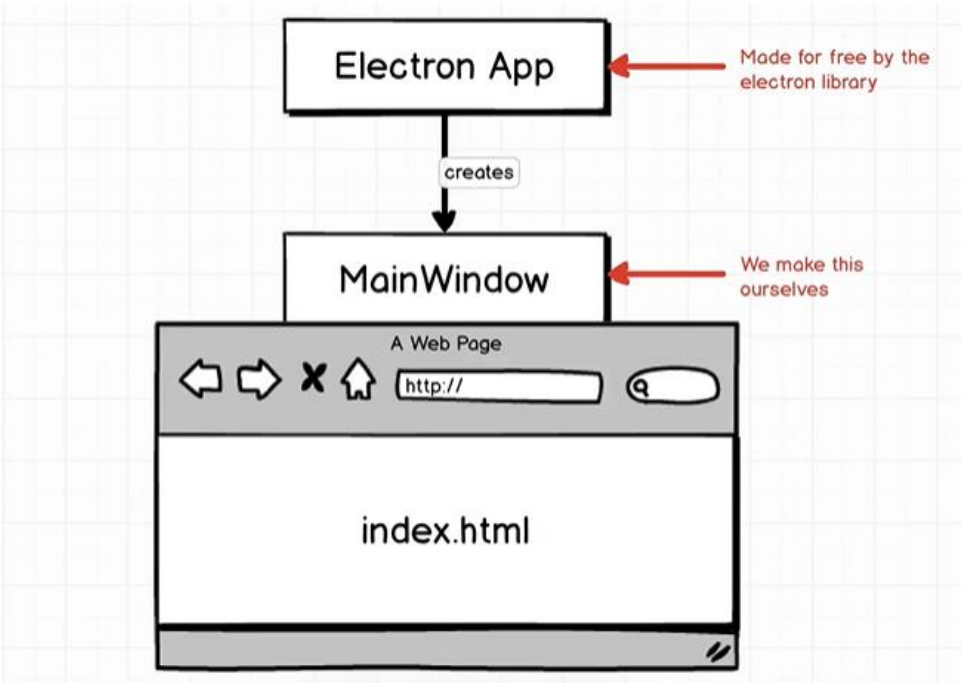
Seit seiner Veröffentlichung hat das Electron-Framework die Herzen aller Web- und Desktop-Entwickler gewonnen. Viele beliebte Anwendungen wurden mit Electron erstellt, die Sie vielleicht in der Vergangenheit verwendet haben oder täglich nutzen, wie z. B. Skype, Slack, WhatsApp, Discord, Signal, Visual Studio Code, Microsoft Teams, Keybase und viele andere.

Requirements:

- NodeJS (https://nodejs.org)
- Code Editor

Chromium:

Unter dem Namen Chromium wird der Großteil des Quelltextes des Webbrowsers Google Chrome unter der BSD-Lizenz als Open-Source-Projekt zur Verfügung gestellt.



Application Architecture:

Main and Renderer Processes

In Electron wird der Prozess, der das Hauptskript, welches in package.json definiert ist, ausführt, als Hauptprozess bezeichnet. Das Skript, das im Hauptprozess läuft, kann eine GUI anzeigen, indem es Webseiten erstellt. Eine Electron-App hat immer einen Hauptprozess, aber nie mehr.

Da Electron Chromium für die Anzeige von Webseiten verwendet, wird auch die Multi-Prozess-Architektur von Chromium verwendet. Jede Webseite in Electron läuft in einem eigenen Prozess, der als Renderer-Prozess bezeichnet wird.

In normalen Browsern laufen Webseiten normalerweise in einer Sandbox-Umgebung und haben keinen Zugriff auf native Ressourcen. Electron-Benutzer haben jedoch die Möglichkeit, Node.js-APIs in Webseiten zu verwenden, die Interaktionen auf niedrigerer Betriebssystemebene ermöglichen.

Unterschiede zwischen Hauptprozess und Renderer-Prozess

Der Hauptprozess erstellt Webseiten durch die Erstellung von BrowserWindow-Instanzen. Jede BrowserWindow-Instanz führt die Webseite in einem eigenen Renderer-Prozess aus. Wenn eine BrowserWindow-Instanz zerstört wird, wird der entsprechende Renderer-Prozess ebenfalls beendet.

Der Hauptprozess verwaltet alle Webseiten und ihre entsprechenden Renderer-Prozesse. Jeder Renderer-Prozess ist isoliert und kümmert sich nur um die in ihm laufende Webseite.

In Webseiten ist das Aufrufen nativer GUI-bezogener APIs nicht erlaubt, da die Verwaltung nativer GUI-Ressourcen in Webseiten sehr gefährlich ist und es leicht zu einem Ressourcenleck kommen kann. Wenn Sie GUI-Operationen in einer Webseite ausführen möchten, muss der Renderer-Prozess der Webseite mit dem Hauptprozess kommunizieren, um diesen zur Ausführung dieser Operationen aufzufordern.

Kommunikation zwischen Prozessen

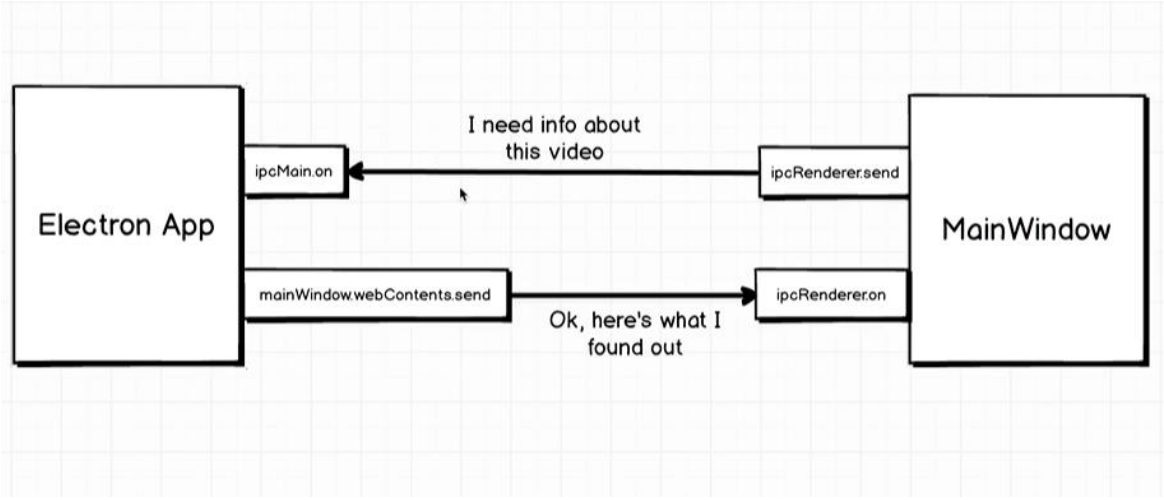
In Electron haben wir mehrere Möglichkeiten, um zwischen dem Hauptprozess und den Renderer-Prozessen zu kommunizieren, wie z.B. die Module ipcRenderer und ipcMain zum Senden von Nachrichten und das Remote-Modul für die Kommunikation im RPC-Stil.

Remote Modul (Verwenden Sie Hauptprozessmodule aus dem Renderer-Prozess):

```
<script>
    const { BrowserWindow } = require('electron')
    let win = new BrowserWindow({ width: 800, height: 600 })
    win.loadURL('https://github.com')
</script>
```

Note: Das remote Modul sollte aus Sicherheitsgründen deaktiviert werden.

Inter Process Communication



Electron stellt uns 2 IPC (Inter Process Communication) Module zur Verfügung, die ipcMain und ipcRenderer genannt werden.

Das Modul ipcMain dient der Kommunikation vom Hauptprozess zu den Renderer-Prozessen. Wenn es im Hauptprozess verwendet wird, verarbeitet das Modul asynchrone und synchrone Nachrichten, die von einem Renderer-Prozess (Webseite) gesendet werden.

Das ipcRenderer-Modul wird verwendet, um von einem Renderer-Prozess an den Hauptprozess zu kommunizieren. Es stellt einige Methoden zur Verfügung, so dass Sie synchrone und asynchrone Nachrichten vom Renderer-Prozess (Webseite) an den Hauptprozess senden können. Sie können auch Antworten vom Hauptprozess empfangen.

```
// In main process.
const { ipcMain } = require('electron')
ipcMain.on('asynchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.reply('asynchronous-reply', 'pong')
})

ipcMain.on('synchronous-message', (event, arg) => {
  console.log(arg) // prints "ping"
  event.returnValue = 'pong'
})

// In renderer process (web page).
const { ipcRenderer } = require('electron')
console.log(ipcRenderer.sendSync('synchronous-message', 'ping')) // prints "pong"

ipcRenderer.on('asynchronous-reply', (event, arg) => {
  console.log(arg) // prints "pong"
})
ipcRenderer.send('asynchronous-message', 'ping')



mainWindow.webContents.send(
        'video:metadata',
        metadata.format.duration
);
```

Node-Integration

Beachten Sie, dass Sie bei Verwendung von Electron 5.0.0 oder höher die Integration von Node.js explizit aktivieren müssen. In späteren Versionen von Electron ist die Node.js-Integration aus Sicherheitsgründen deaktiviert. Wie Sie bereits wissen, können Electron-Anwendungen auch remote Websites anzeigen. Dadurch erhält eine remote Webseite Zugang zu Ihren lokalen Ressourcen und kann möglicherweise bösartige Aktivitäten ausführen. Aus diesem Grund ist die Node.js-Integration deaktiviert.

webPreferences: { nodeIntegration: true }

Isolation For Untrusted Content

Ein Sicherheitsproblem besteht immer dann, wenn Sie Code von einer nicht vertrauenswürdigen Quelle (z.B. einem remote server) erhalten und lokal ausführen. Nehmen wir als Beispiel eine entfernte Website, die innerhalb eines Standard-Browser-Fensters angezeigt wird. Wenn es einem Angreifer irgendwie gelingt, diesen Inhalt zu ändern, ist er in der Lage, nativen Code auf dem Rechner des Benutzers auszuführen.

Unter keinen Umständen sollten Sie bei aktivierter Node.js-Integration remote Code laden und ausführen. Verwenden Sie stattdessen nur lokale Dateien (zusammen mit Ihrer Anwendung gepackt), um Node.js-Code auszuführen.

Um remote Inhalte anzuzeigen, verwenden Sie das Tag <webview> oder BrowserView, stellen Sie sicher, dass Sie nodeIntegration deaktivieren und die Kontext-Isolation aktivieren.

Packaging Apps:

Das Packaging von Anwendungen ist ein integraler Bestandteil des Entwicklungsprozesses einer Desktop-Anwendung. Da Electron ein plattformübergreifendes Framework für die Entwicklung von Desktop-Anwendungen ist, sollte auch die Paketierung und Verteilung von Anwendungen für alle Plattformen eine nahtlose Erfahrung sein.

Die Electron-Community hat ein Projekt, den Electron-Packager, ins Leben gerufen, der sich für uns um genau das kümmert. Er erlaubt uns, unsere Electron-Anwendung mit OS-spezifischen Bundles (.app, .exe usw.) über JS oder CLI zu paketieren und zu verteilen.

Supported Platforms

Electron Packager runs on the following host platforms −

- Windows (32/64 bit)
- OS X
- Linux (x86/x86_64)

It generates executables/bundles for the following target platforms −

- Windows (also known as win32, for both 32/64 bit)
- OS X (also known as darwin) / Mac App Store (also known as mas)
- Linux (for x86, x86_64, and armv7l architectures)

Electron's APIs (https://www.electronjs.org/docs/api)

```
const { BrowserWindow } = require('electron')

const win = new BrowserWindow()
```

Node.js APIs

```
const fs = require('fs')

const root = fs.readdirSync('/')
```

Security:

1. Only load secure content
2. Disable the Node.js integration in all renderers that display remote content
3. Enable context isolation in all renderers that display remote content
4. Use `ses.setPermissionRequestHandler()` in all sessions that load remote content
5. Do not disable `webSecurity`
6. Define a `Content-Security-Policy` and use restrictive rules (i.e. `script-src 'self'`)
7. Do not set `allowRunningInsecureContent` to `true`
8. Do not enable experimental features
9. Do not use `enableBlinkFeatures`
10. `<webview>`: Do not use `allowpopups`
11. `<webview>`: Verify options and params
12. Disable or limit navigation
13. Disable or limit creation of new windows
14. Do not use `openExternal` with untrusted content
15. Disable the `remote` module
16. Filter the `remote` module
17. Use a current version of Electron

Context Bridge:

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


// Expose protected methods that allow the renderer process to use
// the ipcRenderer without exposing the entire object
contextBridge.exposeInMainWorld(
   "electron", {
      ipcRenderer: {
         send: (eventName, data) => {
            ipcRenderer.send(eventName, data);
         },
         on: (eventName, func) => {
            ipcRenderer.on(eventName, func);
         }
      }
   }
);
```

System Integration

Menus

```
const menuTemplate = [
  {
    label: 'File',
    submenu: [
      {
        label: 'Clear Todos',
        click()
        {
          mainWindow.webContents.send('todo:clear');
        }
      },
      {
        label: 'Quit',
        accelerator: isMac ? 'Command+Q' : 'Ctrl+Q',
        click(){
          app.quit();
        }
      }
    ]
  }
];
```

Useful Links

https://www.electronforge.io/

https://help.github.com/en/github/authenticating-to-github/creating-a-personal-access-token-for-the-command-line

https://github.com/electron/electron-packager

Quellen

https://www.tutorialspoint.com/electron/electron_packaging_apps.htm

https://www.electronjs.org/

https://de.wikipedia.org/wiki/Electron_(Framework)

https://de.wikipedia.org/wiki/Chromium_(Browser)

https://www.udemy.com/course/electron-react-tutorial

https://www.packtpub.com/eu/mobile/electron-projects