I will be contextually speaking as though this is a continuation of First Project in this repository

Simple IPC (Inter-Process Communication) Example

(1) In order to use IPC to communicate with app module and our renderer, we need to add a preload script

Within a render process (i.e. index.html's javascript), we cannot use Electron API. Thus, we must add a preload.js script to our BrowserWindow object to be able to use Electron API and other goodies.

(2) main.js

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

const createWindow = () => {
  const win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      preload: path.join(__dirname, "preload.js")
    }
  })
  win.loadFile("index.html")
  win.setMenu(null)

  ipcMain.on("openTextRequest", (event, data) => {
    event.reply("openTextGranted", data)
  })
}

app.whenReady().then(createWindow)
```

- `const path = require("path")`
    - Used to get the full directory of preload.js to load the script
- `webPreferences: { … }`

    - Using webPreferences, we may preload scripts, or even enable node integration with `nodeIntegration: true`

- `ipcMain.on(...)`
    - When our renderer (index.html javascript) is going to send a `ipcRenderer.send(channel, message)` call, `ipcMain.on(channel, ..)` receives the channel "openTextRequest"
    - Then, we may send a reply with `event.replay(channel, data)` back to the renderer process (in this case, preload.js) where we may execute javascript on index.html

(3) preload.js

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

contextBridge.exposeInMainWorld("ipc", {
  send: (evt, message) => { ipcRenderer.send(evt, message) }
})

ipcRenderer.on("openTextGranted", (event, data) => {
  document.getElementById("text").style.display = "block"
})
```

- `const { contextBridge, ipcRenderer } = require("electron")`
    - ipcRenderer module allows us to send and receive ipc messages from other processes
    - contextBridge explained below
- `contextBridge.exposeInMainWorld(...)`
    - contextBridge module allows us to call, in this case, `ipcRenderer.send(channel, data)` calls via a proxy function ipc.`send(channel, data)` within any of index.html's javascript

- `ipcRenderer.on(...)`
    - In preload.js, once we have received index.html's `ipc.send("openTextRequest", "text")`, we send a reply with the channel "openTextGranted". This receives this reply and allows us to perform javascript/css operations on index.html

(4) index.html

```
<!DOCTYPE html>

<html>
  <head>
    <style> body { background: #101010; } </style>

    <script>
      function send() { ipc.send("openTextRequest", "text") }
    </script>
  </head>
  <body>
    <button onclick="send()">Click Me</button>
    <p id="text" style="display: none; color: #EDE6D6;">IPC Events Processed Successfully</p>
  </body>
</html>
```

With ipc.send, created in preload.js's `contextBridge.exposeInMainWorld(...)`, we may use `ipcRenderer.send(channel, data)` as `ipc.send(channel, data)`