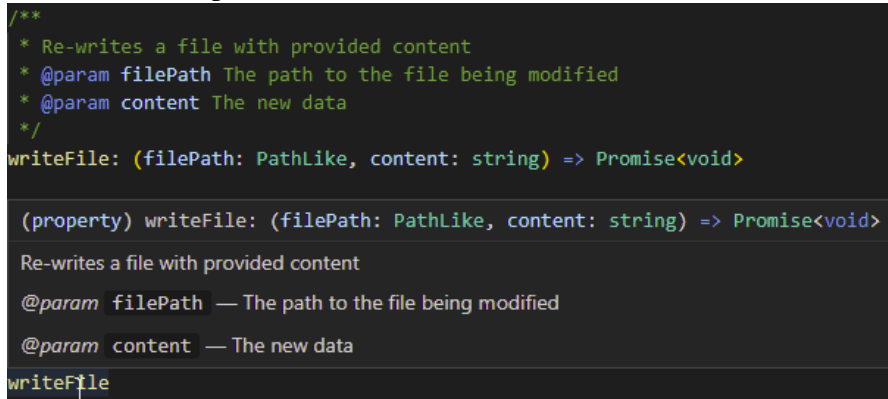


For this assignment, and my final project, I will be using a stack of different technologies to create my application. Here's a brief explanation of them:

TypeScript: A superset of JavaScript that adds strict-ish type checking to JavaScript applications. You are allowed to create union types and such, but TypeScript allows you to define “interfaces” to describe the shape of an object. This provides reliability, as well as auto-completion in your text editor (in my case, VS Code). You can also add descriptions to your functions/variables which can be recognized and used by your IDE, such as a hover tooltip:



```
/**
 * Re-writes a file with provided content
 * @param filePath The path to the file being modified
 * @param content The new data
 */
writeFile: (filePath: PathLike, content: string) => Promise<void>
```

(property) writeFile: (filePath: PathLike, content: string) => Promise<void>

Re-writes a file with provided content

@param filePath — The path to the file being modified

@param content — The new data

writeFile

NodeJS: A tool that allows you to run JavaScript code as a back-end, interacting with filesystems, handling HTTP Requests, and much more. Commonly used to manage the back-end of a webserver.

Electron: A NodeJS framework that allows you to build native applications using HTML, CSS, and JavaScript. This works by bundling your final application with a copy of Chromium, which handles the front end. Electron applications also have access to node modules, such as file editing and database integration. Electron is used for many applications, such as Slack, VS Code, and Discord. These applications are inherently cross-platform, without the need for separate codebases.

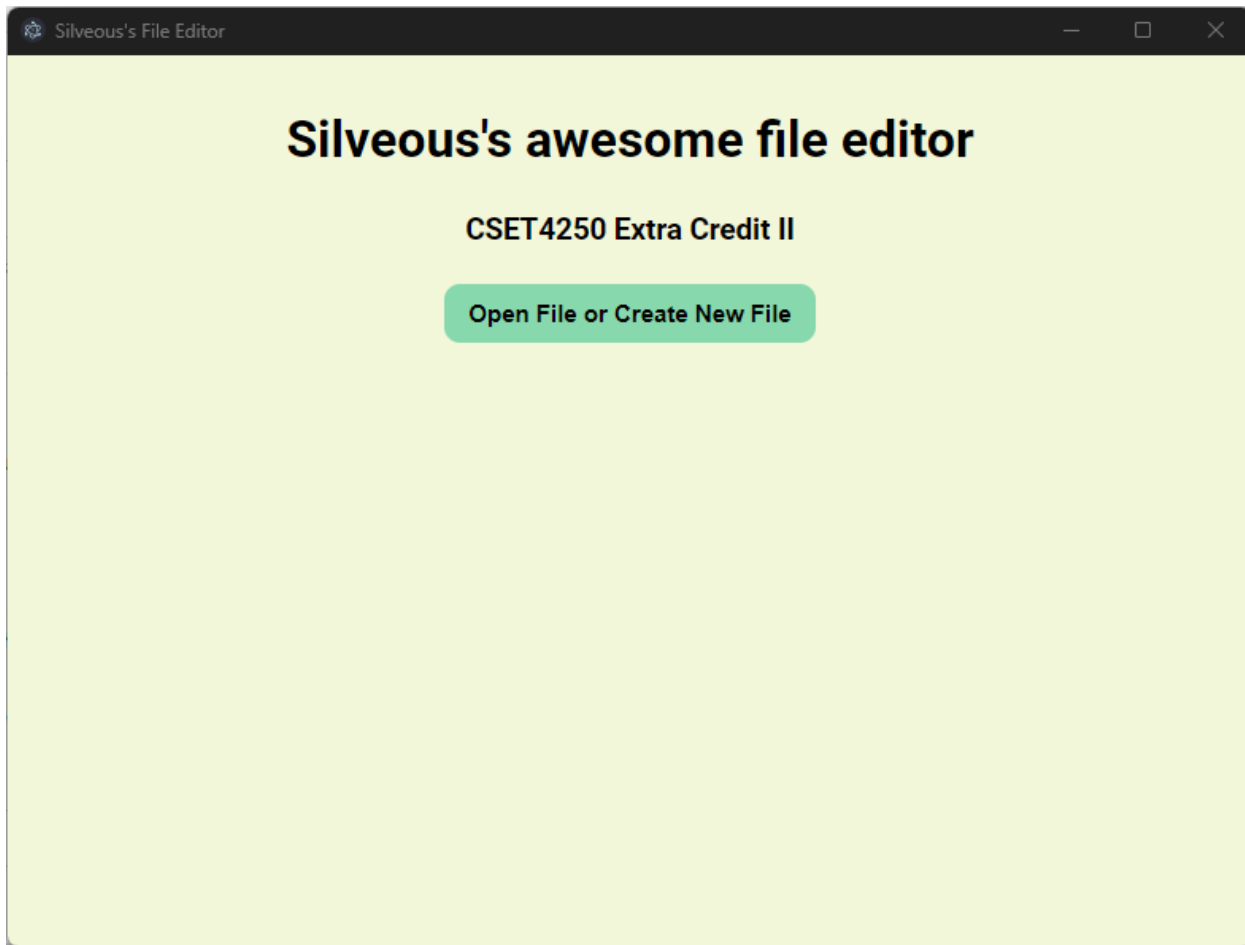
React: A JavaScript library that makes building reactive user interfaces with JavaScript easier, by providing functions to manage stateful components, removing much of the declarative code required to do the same with vanilla JavaScript and HTML.

React is just a library, but you usually use it alongside *JSX*, which is a syntax extension that allows you to write DOM elements in a HTML-like syntax within JavaScript.

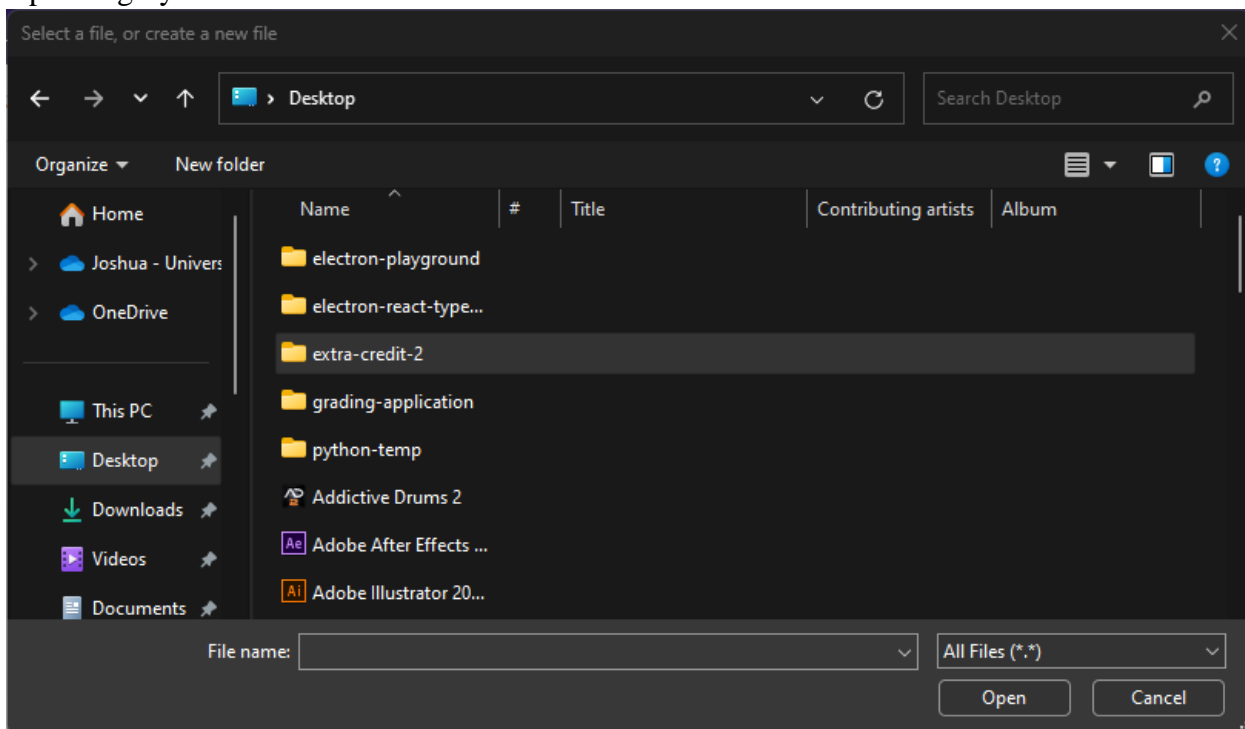
Included code isn't the entire program, as there is a lot of boilerplate code which makes this possible. Instead, I just included the code that I actually had to write myself. [Here's a link to this project on GitHub if you're curious.](#)

I took a bit of liberty with this assignment and made a full text editor instead, allowing the user to modify files themselves. You can also create new files.

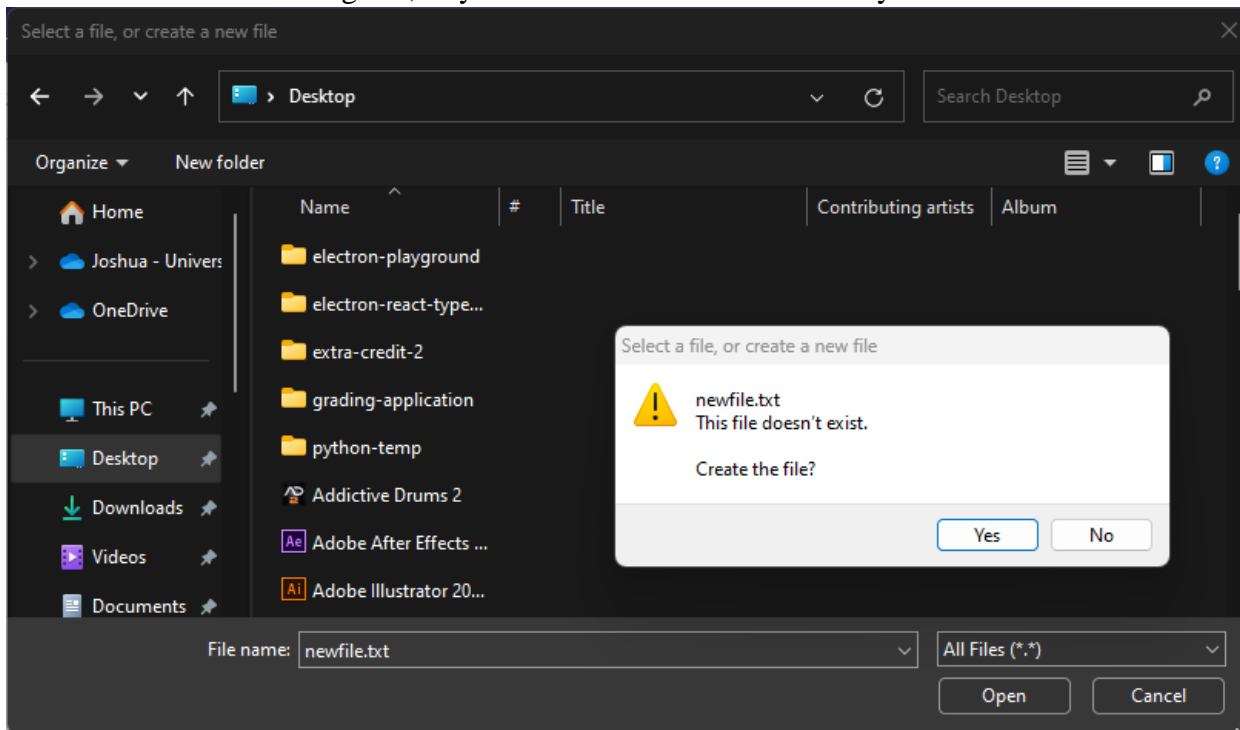
Opening screen



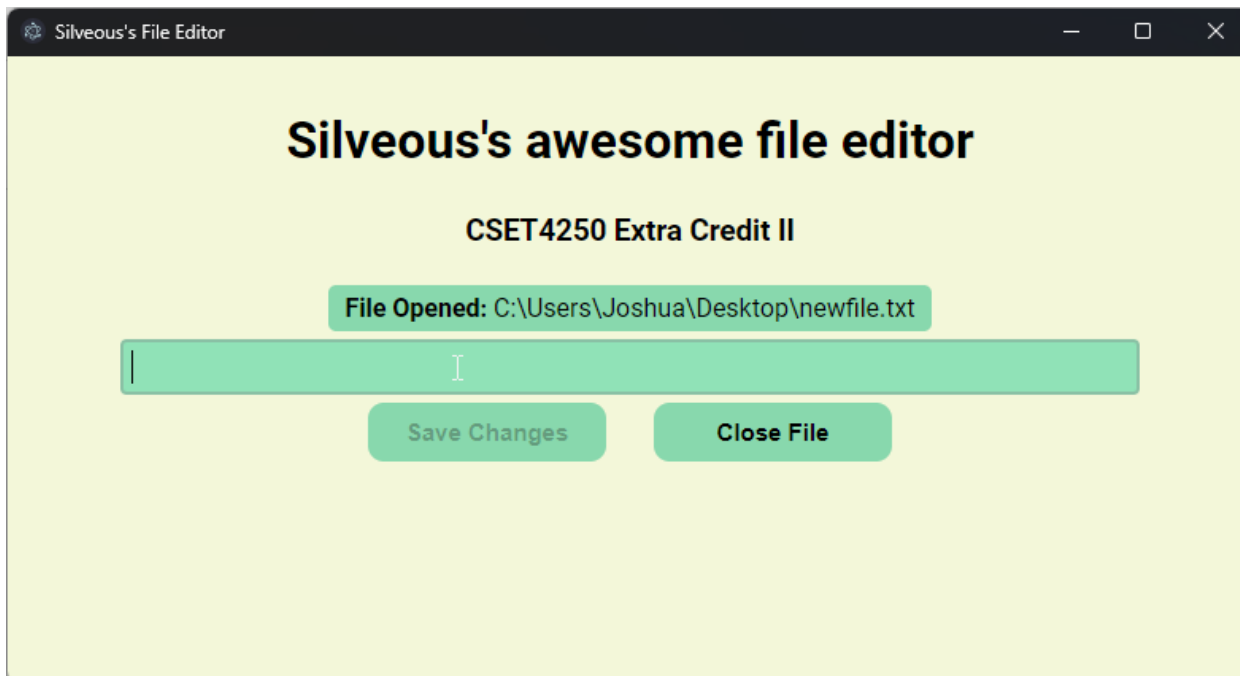
When you choose “Open File or Create New File”, it brings up a dialogbox using whatever your native Operating System is:



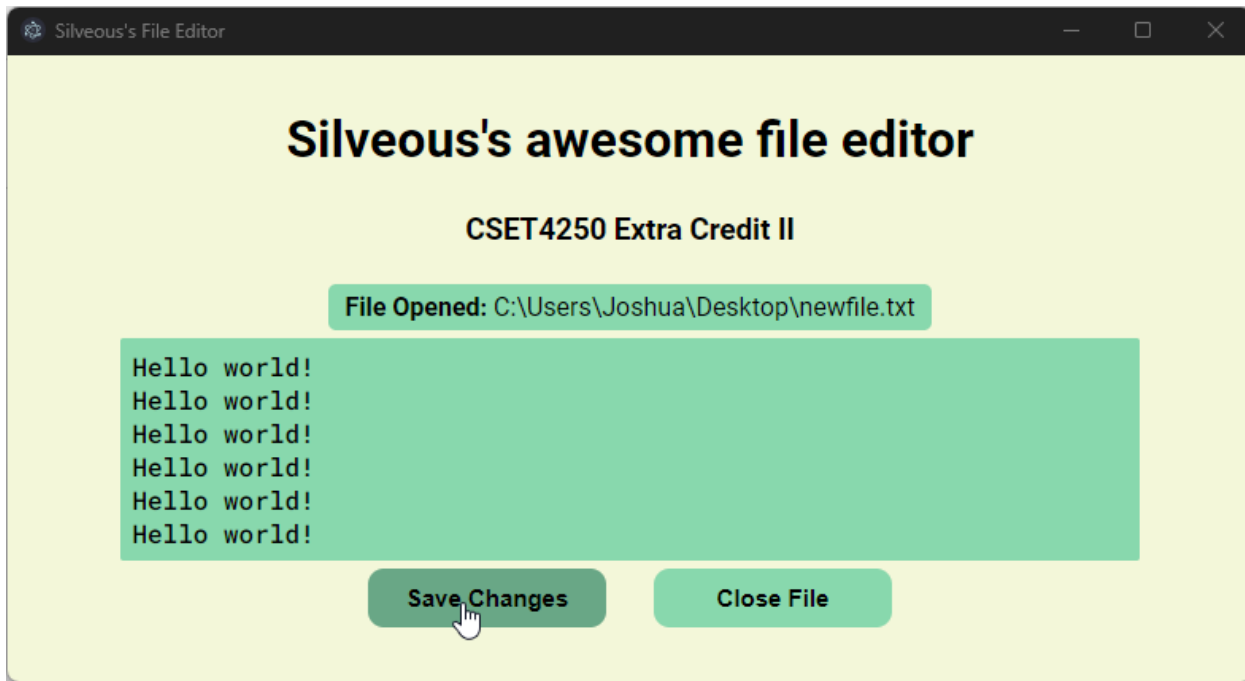
You can choose an existing file, or you can enter a name and create your own:



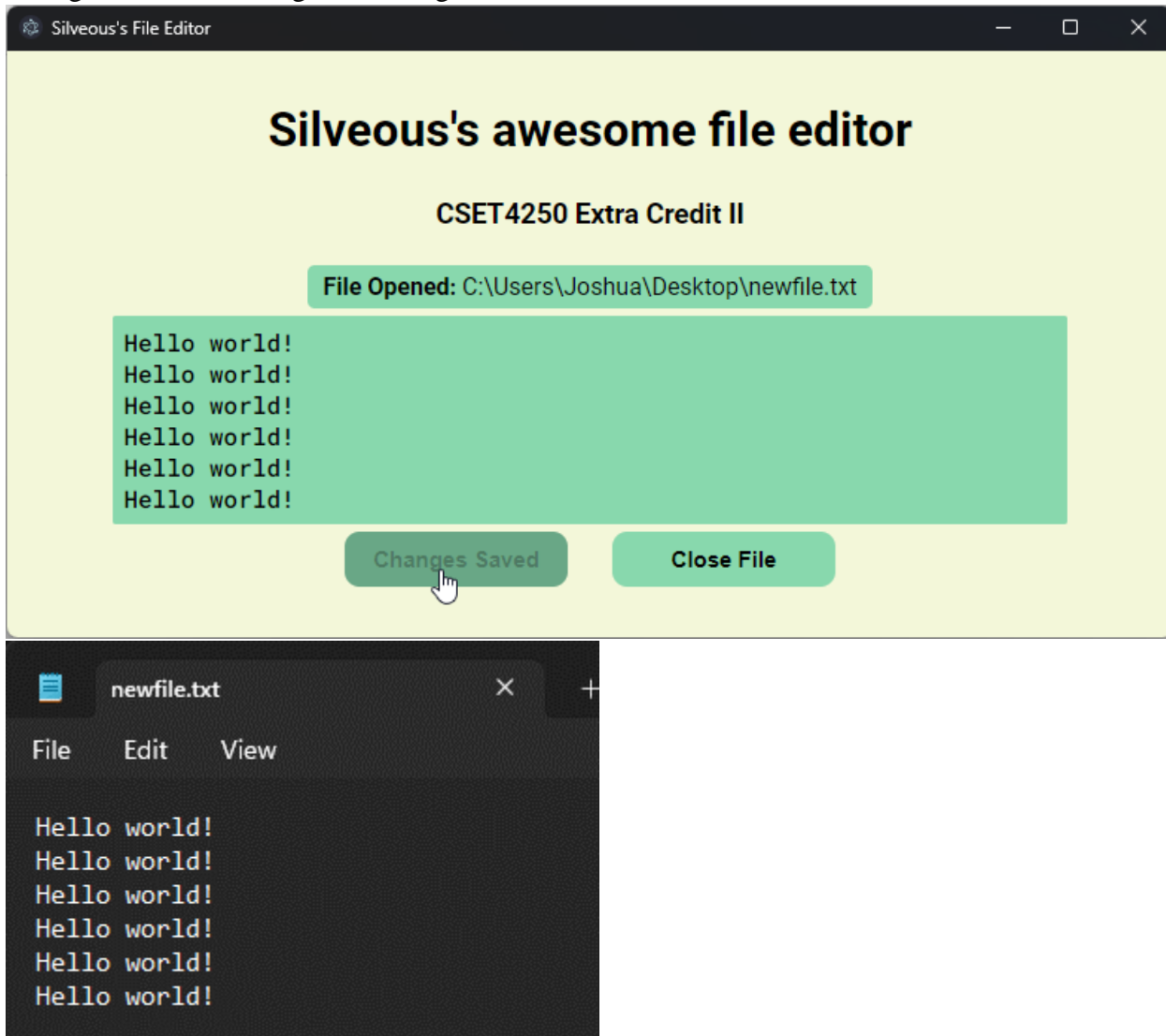
Once you select/create the file, a backend function will retrieve the data and display it. In this case, the file is blank:



You can add text to the box. Once you change anything, the “Save Changes” button will become available:



Once you hit “Save Changes”, a backend function will be called to save the changes you made. The “Save Changes” text will change to “Changes Saved” for 1.5 seconds, then revert.



You can choose “Close File” to go back to the opening screen.

Here's an example of me opening an existing file:



index.ts

```
import { ipcMain, dialog } from 'electron'
import fs from 'fs/promises';
import { PathLike } from 'original-fs';

async function readFile(filePath: PathLike): Promise<string> {
  let res: string
  try {
    res = await fs.readFile(filePath, { encoding: 'utf-8' })
  } catch (error) {
    console.log(error)
  }
  return res
}

async function writeFile(filePath: PathLike, content: string) {
  try {
    await fs.writeFile(filePath, content)
  } catch (error) {
    console.log(error)
  }
}

async function fileSelect(): Promise<PathLike> {
  const { canceled, filePaths } = await dialog.showOpenDialog(
    { title: "Select a file, or create a new file", properties: ['openFile',
'promptToCreate'] }
  )
  if (canceled) {
    return null
  } else {
    // check if file already exists
    let fileExists = false
    try {
      await fs.access(filePaths[0])
      fileExists = true
    } catch (error) {
      fileExists = false
    }

    // if not, create a new file
    if (!fileExists) {
      await writeFile(filePaths[0], "")
    }

    return filePaths[0] as PathLike
  }
}
```

```

app.on('ready', () => {
  ipcMain.handle('fileSelect', fileSelect)
  ipcMain.handle('readFile', (e, filePath: PathLike) => readFile(filePath))
  ipcMain.handle('writeFile', (e, filePath: PathLike, content: string) =>
writeFile(filePath, content))
})

```

preload.ts

```

import { contextBridge, ipcRenderer } from 'electron'

contextBridge.exposeInMainWorld('api', {
  fileSelect: () => ipcRenderer.invoke('fileSelect'),
  readFile: (filePath) => ipcRenderer.invoke('readFile', filePath),
  writeFile: (filePath, content) => ipcRenderer.invoke('writeFile', filePath, content)
} as Window['api'])

```

renderer.ts

```

import './index.css';
import React from 'react'
import ReactDOM from "react-dom/client"
import App from "./app";

const root = ReactDOM.createRoot(document.getElementById('root'))
root.render(
  <App />
)

```

app.tsx

```

import { PathLike } from 'original-fs'
import React from 'react'

export default function App() {

const [filePath, setFilePath] = React.useState<PathLike>("")
const [fileContent, setFileContent] = React.useState("")
const [isChanged, setIsChanged] = React.useState(false)
const contentRef = React.useRef(null)

  async function handleOpenFile() {
    let selection = await window.api.fileSelect()
    if (selection) {
      setFileContent(await window.api.readFile(selection))
      setFilePath(selection)
    }
  }

```



```

}

async function handleSave(e: any) {
  let content = contentRef.current.innerText
  await window.api.writeFile(filePath, content)
  setIsChanged(false)
  e.target.innerText = "Changes Saved"
  setTimeout(() => {
    e.target.innerText = "Save Changes"
  }, 1500)
}

return (
<div id="main">
  <h1>Silveous's awesome file editor</h1>
  <h3>CSET4250 Extra Credit II</h3>
  { filePath == "" ?
    <button onClick={handleOpenFile}>Open File or Create New File</button>
    :
    <>
      <div id="filepath">
        <strong>File Opened:</strong> { filePath as string }
      </div>
      <div
        id="filedisplay"
        contentEditable="true"
        onInput={() => setIsChanged(true)}
        ref={contentRef}
      >
        { fileContent }
      </div>
      <div id="buttonwrapper">
        <button
          onClick={handleSave}
          disabled={!isChanged}
        >
          Save Changes
        </button>
        <button onClick={() => setFilePath("")}>Close File</button>
      </div>
    </>
  }
</div>
)
}

```

```
@import url('../fonts/fonts.css');

body {
  font-family: 'Roboto', sans-serif;
  margin: auto;
  max-width: 38rem;
  padding: 2rem;
  background-color: beige;
}

button {
  border: none;
  padding: 10px 15px;
  background-color: rgb(107, 214, 178);
  border-radius: 10px;
  font-weight: 700;
  font-size: 15px;
  cursor: pointer;
  transition: 0.2s ease background-color;
  min-width: 150px;
}

button:hover {
  background-color: rgb(83, 165, 138);
}

#buttonwrapper {
  display: flex;
  gap: 30px;
}

h1 {
  margin: 2px;
}

#main {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
  gap: 5px;
}

#filepath {
  width: fit-content;
  padding: 5px 10px;
  border-radius: 5px;
  max-width: 600px;
  text-align: center;
}
```

```
background-color: rgb(107, 214, 178);
}

#filedisplay {
  font-family: 'Roboto Mono', monospace;
  font-weight: 500;
  background-color:rgb(107, 214, 178);
  border-radius: 2px;
  width: 80vw;
  padding: 7px;
  height: fit-content;
  max-height: 60vh;
  overflow: auto;
  white-space: pre-wrap;
  transition: 0.2s ease background-color;
}

#filedisplay:focus, #filedisplay:active {
  outline: 0px solid transparent;
  border: 2px solid rgb(120, 192, 168);
  border-radius: 4px;
  background-color: rgb(114, 224, 188);
  padding: 5px;
}
```

window.d.ts (ambient type declaration file)

```
import { PathLike } from "original-fs"

declare global {
  interface Window {
    /**
     * A user-defined API that carries functions over from preload.ts to the renderer
     */
    api: {
      /**
       * Prompts user to select a file, or create a new file, using OS dialog
       * @returns The path to selected file
       */
      fileSelect: () => Promise<PathLike>,
      /**
       * Reads a file
       * @param filePath The path to the file
       * @returns Selected file's data
       */
      readFile: (filePath: PathLike) => Promise<string>,
      /**
       * Re-writes a file with provided content
       * @param filePath The path to the file being modified
       * @param content The new data
       */
      writeFile: (filePath: PathLike, content: string) => Promise<void>
    }
  }
}
```