TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

LÝ TUẤN AN - 52000620

ELECTRONJS FRAMEWORK

BÁO CÁO GIỮA KỲ
PHÁT TRIỂN
ỨNG DỤNG WEB VỚI NODEJS

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

LÝ TUẤN AN - 52000620

ELECTRONJS FRAMEWORK

BÁO CÁO GIỮA KỲ
PHÁT TRIỂN
ỨNG DỤNG WEB VỚI NODEJS

Người hướng dẫn
GV. Trần Bảo Tín

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Trong suốt thời gian qua, nhờ sự giảng dạy tận tâm của quý Thầy Cô khoa Công Nghệ Thông Tin, trường Đại học Tôn Đức Thắng, chúng em đã học hỏi được rất nhiều điều bổ ích và tích lũy cho mình một số kiến thức để hoàn thành bài báo cáo này. Nhóm em xin chân thành cảm ơn.

Nhóm em xin cảm ơn thầy Trần Bảo Tín đã tận tình chỉ bảo chúng em qua những buổi học tại lớp, thầy đã chỉ nhóm em cách thức làm bài, chỉ điểm những chỗ còn sai sót chưa phù hợp cũng như phải làm sao để trình bày bố cục đẹp. Nếu không có những lời hướng dẫn, dạy bảo của thầy thì bài thu hoạch của nhóm em cũng rất khó để hoàn thiện. Một lần nữa chúng em xin chân thành cảm ơn thầy.

Bước đầu đi vào thực tế với nền kiến thức mở rộng, kiến thức của nhóm em còn hạn chế và nhiều bỡ ngỡ. Vì thế, trong quá trình biên soạn khó tránh những sai sót, nhóm em rất mong nhận được những ý kiến đóng góp quý báu của thầy và các bạn để bài báo cáo hoàn thiện hơn.

TP. Hồ Chí Minh, ngày 19 tháng 11 năm 2023

Tác giả

An

Lý Tuấn An

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của GV. Trần Bảo Tín. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 19 tháng 11 năm 2023

Tác giả

An

Lý Tuấn An

ELECTRON JS

TÓM TẮT

(Time New Romans – 13).....

.....

.....

ELECTRON JS

ABSTRACT

(Time New Romans – 13).....

.....

.....

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC CÁC CHỮ VIẾT TẮT

IPC	Inter-Process Communication

CHƯƠNG 1. GIỚI THIỆU

1.1 Giới thiệu chung về Electron JS

1.1.1 Khái niệm & Lịch sử phát triển

ElectronJS, được phát triển bởi GitHub, là một framework mã nguồn mở cho phép xây dựng ứng dụng desktop cross-platform sử dụng các công nghệ web như HTML, CSS và JavaScript. Nó đã xuất hiện đầu tiên vào năm 2013 dưới dạng dự án Atom Shell của GitHub, được tạo ra để xây dựng trình soạn thảo mã nguồn mở Atom. Ngày nay, ElectronJS đã phát triển rộng rãi và trở thành một công cụ quan trọng trong việc phát triển ứng dụng desktop đa nền tảng.

1.1.2 Mục đích sử dụng

Mục đích chính của ElectronJS là đơn giản hóa quá trình phát triển ứng dụng desktop bằng cách sử dụng các công nghệ web quen thuộc. Nó cho phép các nhà phát triển sử dụng kỹ thuật web để xây dựng ứng dụng mà không cần kiến thức sâu về lập trình desktop. Điều này giúp giảm độ phức tạp và chi phí trong quá trình phát triển, đồng thời giữ cho ứng dụng có khả năng chạy trên nhiều hệ điều hành như Windows, macOS và Linux.

1.2 Đặc điểm của ElectronJS

ElectronJS kết hợp trực tiếp Chromium (một trình duyệt web mã nguồn mở) và Node.js (môi trường thực thi JavaScript trên máy chủ). Điều này mang lại một số ưu điểm, bao gồm:

Tích hợp đồng nhất: Sự kết hợp giữa Chromium và Node.js giúp xây dựng ứng dụng với môi trường thực thi đồng nhất trên cả hai phía client và server.

Hiệu suất cao: Sử dụng trình duyệt Chromium giúp ứng dụng Electron có hiệu suất cao và khả năng tương tác tốt với các công nghệ web.

Đa nền tảng: Nhờ vào sự kết hợp giữa Node.js và Chromium, ElectronJS cho phép xây dựng ứng dụng một lần và chạy trên nhiều hệ điều hành khác nhau mà không cần sửa đổi nhiều mã nguồn.

1.3 Phạm vi tìm hiểu

Nghiên cứu chú trọng vào việc hiểu rõ cách ElectronJS thúc đẩy việc phát triển ứng dụng cross-platform. Các khía cạnh như sự tích hợp giữa Node.js và Chromium, kiến trúc ứng dụng, sự tương tác giữa các quy trình chính và renderer process, kết nối database Mongo Atlas, cách đóng gói và phân phối ứng dụng là những điểm quan trọng được tập trung để đảm bảo hiểu biết đầy đủ về framework, cách nó hoạt động và phân phối ứng dụng đến người dùng cuối.

Phiên bản tìm hiểu:

- Electron: 27.1.2
- Electron-forge: 7.1.0
- Node: 20.6.0
- npm: 10.1.0

CHƯƠNG 2. KIẾN TRÚC CỦA ELECTRON JS

2.1 Process Model

2.1.1 Main Process

Quy Trình Chính (*Main Process*): Mỗi ứng dụng Electron có một quy trình chính duy nhất, đóng vai trò là điểm truy cập của ứng dụng và có nhiệm vụ quản lý sự kiện hệ thống và tương tác với các tài nguyên hệ thống.

Quản Lý Sự Kiện Hệ Thống: *Main Process* chịu trách nhiệm quản lý các sự kiện hệ thống như tạo cửa sổ ứng dụng, xử lý menu, và tương tác với các tài nguyên hệ thống.

Truy Cập Node.js Modules: *Main Process* chạy trong môi trường Node.js, nghĩa là nó có khả năng yêu cầu các mô-đun và sử dụng tất cả các API của Node.js như truy cập các Node.js built-ins, cũng như mọi gói được cài đặt qua npm giúp thực hiện các tác vụ liên quan đến hệ thống như đọc/ghi file, giao tiếp mạng, ...

2.1.2 Renderer Process

Quy Trình Renderer (*Renderer Process*): Mỗi cửa sổ ứng dụng có một quy trình renderer riêng, chịu trách nhiệm hiển thị giao diện người dùng và xử lý sự kiện liên quan đến UI. Mã trong *Renderer Process* tuân theo tiêu chuẩn web, với HTML là chịu trách nhiệm nội dung, CSS chịu trách nhiệm format lại giao diện, và JavaScript thực thi qua <script>.

Trước đây *Renderer Process* có thể khởi tạo môi trường Node.js đầy đủ cho phát triển, nhưng tính năng này đã bị vô hiệu hóa vì lý do bảo mật. Và không có cách trực tiếp nào để *Renderer Process* tương tác với Node.js và chức năng native của Electron từ *Main Process*.

Chạy Trong Môi Trường Cô Lập: Mỗi *Renderer Process* chạy trong môi trường cô lập để đảm bảo an toàn và bảo mật. Mỗi cửa sổ có một *Renderer Process* riêng để ngăn chặn tác động tiêu cực giữa các cửa sổ khác nhau.

2.1.3 Preload Script

Như đã đề cập trước đó, các *Renderer Process* chạy các trang web và không chạy Node.js theo mặc định vì lý do bảo mật. Để kết nối *Main Process* và *Renderer Process* với nhau, chúng ta sẽ cần sử dụng *Preload Script*.

Preload Script là những đoạn mã thực thi trong *Renderer Process* trước khi nội dung web của nó bắt đầu tải. Những đoạn mã này chạy trong ngữ cảnh của *Renderer Process* nhưng được cấp quyền cao hơn thông qua việc truy cập vào các API của Node.js.

Từ Electron 20 trở đi, các tập lệnh *Preload Script* được đóng hộp cát (Preload Script Sandboxing) theo mặc định và không còn quyền truy cập vào môi trường Node.js đầy đủ nữa. Điều này có nghĩa là *Preload Script* chỉ có quyền truy cập vào một bộ API giới hạn.

Bảng 2.1.3 Available API in Electron 20

Available API	Details
Electron modules	Renderer process modules
Node.js modules	events, timers, url
Polyfilled globals	Buffer, process, clearImmediate, setImmediate

Mặc dù *Preload Script* chia sẻ đối tượng window toàn cục với *Renderer Process* mà nó được gắn kết, nhưng chúng ta không thể gắn trực tiếp bất kỳ biến nào từ *Preload Script* vào window vì lý do mặc định của *Context Isolation*.

2.2 Context Isolation

2.2.1 Định nghĩa

Context Isolation là một tính năng đảm bảo rằng cả tập lệnh *Preload Script* và logic bên trong của Electron đều chạy trong một ngữ cảnh riêng biệt với trang web được tải trong webContents. Điều này rất quan trọng vì mục đích bảo mật vì nó giúp ngăn trang web truy cập vào nội bộ của Electron hoặc các API mạnh mẽ mà tập lệnh *Preload Script* của có quyền truy cập.

Điều này có nghĩa là đối tượng window mà tập lệnh *Preload Script* có quyền truy cập thực sự là một đối tượng khác với đối tượng mà trang web có quyền truy cập. Ví dụ:

```
// preload.js
window.hello = 'wave'
```

window. hello sẽ không được xác định nếu trang web cố gắng truy cập vào nó.

```
// renderer.js
console.log(window. hello)  // => undefined
```

Context Isolation đồng nghĩa với việc *Preload Script* được cô lập hoàn toàn với *Renderer Process* để tránh rò rỉ bất kỳ API đặc quyền nào vào mã nguồn web. Thay vào đó, chúng ta sử dụng module *contextBridge* để thực hiện điều này một cách an toàn.

Đặc điểm này vô cùng hữu ích cho 2 mục đích chính:

- Bằng cách tiếp cận ipcRenderer helpers trong renderer, chúng ta có thể sử dụng giao tiếp giữa các quy trình (IPC) để kích hoạt các nhiệm vụ của quy trình chính từ renderer (và ngược lại).
- Nếu đang phát triển một ứng dụng Electron bọc xung quanh ứng dụng web hiện có đặt trên một URL từ xa, ta có thể thêm các thuộc tính tùy chỉnh vào toàn cục cửa sổ của renderer có thể được sử dụng cho logic chỉ trên máy tính để bàn trên phía khách hàng web.

Tính năng *Context Isolation* đã được bật theo mặc định kể từ Electron 12 và đây là cài đặt bảo mật được đề xuất cho tất cả các ứng dụng.

2.2.2 Sự thay đổi

2.2.2.1 Trước phiên bản Electron 12: *Context Isolation* được tắt (disabled)

Hiển thị các API từ tập lệnh *Preload Script* của đến một trang web đã tải trong *Renderer Process* là một trường hợp sử dụng phổ biến. Khi tính năng *Context Isolation* được tắt, tập lệnh *Preload Script* sẽ chia sẻ một đối tượng window chung với *Renderer Process*. Sau đó, chúng có thể đính kèm các thuộc tính tùy ý vào tập *Preload Script.* Ví dụ:

```
// preload.js
window.myAPI = {
  getUser: () => {}
}
```

Khi đó, hàm getUser() có thể được sử dụng trực tiếp trong *Renderer Process*:

```
// renderer.js
window.myAPI.do ()
```

2.2.2.2 Từ phiên bản Electron 12: *Context Isolation* được bật mặc định

Có một mô-đun chuyên dụng trong Electron để giúp thực hiện việc này một cách dễ dàng. Chúng ta có thể sử dụng mô-đun *contextBridge* để hiển thị các API một cách an toàn từ ngữ cảnh riêng biệt của tập lệnh *Preload Script* của bạn đến ngữ cảnh mà trang web đang chạy. API cũng sẽ có thể truy cập được từ trang web trên window.myAPI giống như trước đây.

```
// preload.js
const { contextBridge } = require('electron')
contextBridge.exposeInMainWorld('myAPI', {
  getUser: () => {}
})

// renderer.js
window.myAPI. getUser()
```

2.3 ContextBridge

Như đã đề cập ở mục trước đó, ContextBridge là một tính năng trong Electron giúp tạo một cầu nối an toàn giữa các quy trình khác nhau, đặc biệt là giữa *Preload Scripts* và *Renderer Processes.*

Nguyên Tắc Hoạt Động: Nó cho phép *Preload Scripts* truyền thông tin và chức năng an toàn từ *Main Processes* vào *Renderer Processes*, nhưng vẫn giữ cho tính cô lập và an toàn thông qua *ipcRenderer.*

2.4 Inter-Process Communication (IPC)

2.4.1 Khái niệm

Giao tiếp giữa các quá trình (IPC) là một phần quan trọng trong việc xây dựng các ứng dụng desktop trong Electron. Vì *Main Processes* và *Renderer Processes* có trách nhiệm khác nhau trong mô hình quy trình (Process Model) của Electron nên IPC là cách duy nhất để thực hiện nhiều tác vụ phổ biến, chẳng hạn như gọi API gốc từ giao diện người dùng.

2.4.2 IPC channels

Trong Electron, các quy trình giao tiếp bằng cách chuyển tin nhắn qua các "kênh" do lập trình viên xác định bằng mô-đun ipcMain và ipcRenderer. Các kênh này là tùy ý (có thể đặt tên cho chúng theo bất kỳ tên nào mà ta mong muốn).

Có 3 cách giao tiếp chính giữa các quy trình với nhau.

2.4.2.1 Renderer to Main (1 chiều)

Để kích hoạt thông báo IPC một chiều từ *Renderer Processes* sang *Main Processes*, chúng ta có thể sử dụng API ipcRenderer.send để gửi tin nhắn sau đó được API ipcMain.on nhận.

Cách này thường sử dụng để gọi API quy trình chính từ nội dung web.

2.4.2.2 Renderer to Main (2 chiều)

Một ứng dụng phổ biến cho IPC hai chiều là gọi mô-đun *Main Processes* từ *Renderer Processes* và chờ kết quả trả về. Điều này có thể được thực hiện bằng cách sử dụng ipcRenderer.invoke kết hợp với ipcMain.handle.

2.4.2.3 Renderer to Main

Khi gửi tin nhắn từ *Main Processes* đến *Renderer Processes*, ta cần chỉ định *Renderer Processes* nào đang cần nhận tin nhắn. Tin nhắn cần được gửi đến *Renderer Processes* thông qua phiên bản WebContents của nó. Phiên bản WebContents này chứa một phương thức gửi có thể được sử dụng theo cách tương tự như ipcRenderer.send.

2.4.3 Sự quan trọng của IPC

Tính An Toàn: IPC giúp ngăn chặn sự tương tác trực tiếp giữa các quy trình, giữ cho chúng cô lập và an toàn.

Chia Sẻ Dữ Liệu: Cho phép chia sẻ dữ liệu giữa quy trình chính và renderer, giúp tạo ra các ứng dụng phức tạp với nhiều thành phần tương tác.

Đồng Bộ Hóa Công Việc: IPC hữu ích để đồng bộ hóa các tác vụ giữa các phần khác nhau của ứng dụng.

2.5 Process Sandboxing

2.5.1 Khái niệm

Mục Đích Chính: Cô lập tiến trình (Process Sandboxing) là một cơ chế bảo mật quan trọng trong Electron nhằm giới hạn quyền truy cập và nguy cơ bảo mật của mỗi tiến trình, đặc biệt là renderer process.

Nguyên Tắc Hoạt Động: Nó tạo ra một môi trường chạy độc lập và an toàn, ngăn chặn tiến trình khỏi việc ảnh hưởng trực tiếp lẫn nhau hoặc gây nguy hiểm cho hệ thống.

2.5.2 Sự cần thiết của Sandboxing

Bảo Vệ An Toàn: Cung cấp một lớp bảo vệ cho hệ thống và dữ liệu bằng cách ngăn chặn tiến trình renderer từ việc truy cập các nguồn tài nguyên quyền hạn.

Ngăn Chặn Rủi Ro: Hạn chế khả năng của mã JavaScript trong renderer process gây nguy cơ bảo mật bằng cách cô lập chúng trong một môi trường chạy an toàn.

2.5.3 Cơ chế hoạt động

Giới Thiệu Quy Trình: Mỗi *Renderer Process* được chạy trong một "sandbox", là một môi trường cô lập với các quyền hạn hạn chế.

Không Giao Tiếp Trực Tiếp: Tiến trình renderer không thể giao tiếp trực tiếp với hệ thống hoặc các tiến trình khác mà không thông qua quy trình chính.

Giới Hạn Quyền Hạn: *Renderer Process* không có quyền trực tiếp truy cập vào các tài nguyên hệ thống, file, hoặc API quyền hạn cao.

2.5.4 Ưu điểm và tính năng

An Toàn và Ổn Định: Sandboxing giúp bảo vệ ứng dụng khỏi lỗi gây ra bởi mã JavaScript không an toàn và giữ cho ứng dụng ổn định hơn.

Ngăn Chặn Rò Rỉ: Ngăn chặn thông tin nhạy cảm từ rò rỉ ra khỏi *Renderer Process* và giữ cho dữ liệu an toàn.

Quản Lý Bộ Nhớ: Sandboxing giúp quản lý bộ nhớ hiệu quả bằng cách cô lập tiến trình Renderer, giảm nguy cơ làm tăng sự tiêu tốn bộ nhớ.

CHƯƠNG 3. CÁCH XÂY DỰNG – LẬP TRÌNH – SỬ DỤNG

3.1 Các công cụ và Môi trường phát triển

3.1.1 Code editor

Chúng ta sẽ cần một trình soạn thảo mã (Code editor) để viết mã của mình. Electron khuyến khích ta nên sử dụng Visual Studio Code, mặc dù ta có thể chọn bất kỳ cái nào ta thích.

3.1.2 Command line

Windows: Command Prompt or PowerShell

macOS: Terminal

Linux: varies depending on distribution (e.g. GNOME Terminal, Konsole)

Hầu hết các trình soạn thảo mã (Code editor) cũng đi kèm với một integrated terminal mà ta có thể sử dụng.

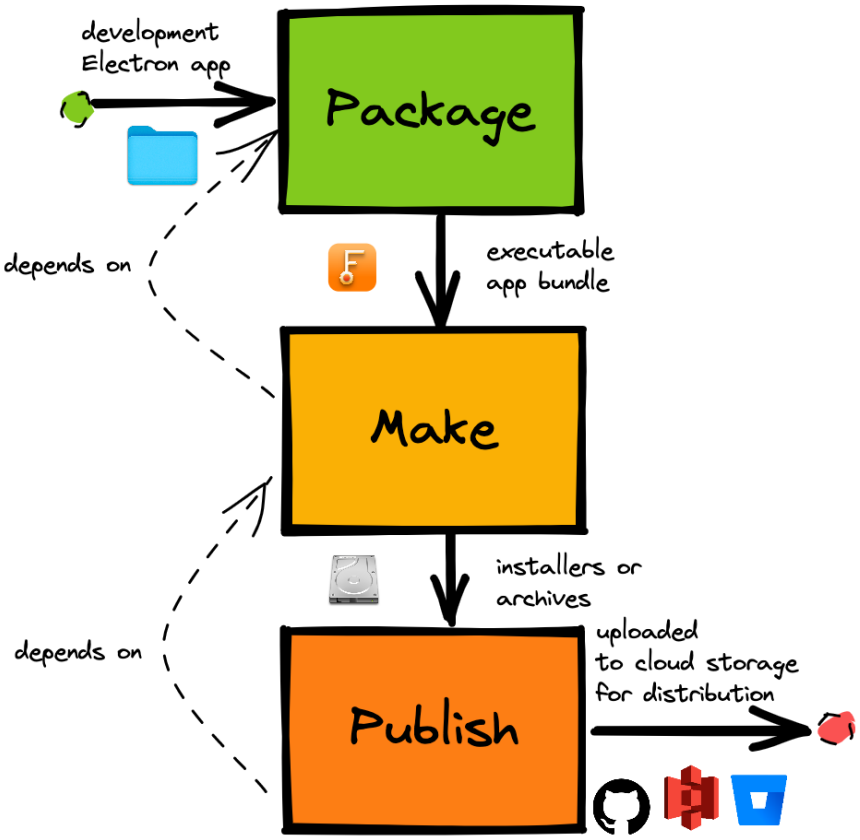
3.1.3 Node.js and npm

Để bắt đầu phát triển ứng dụng Electron, chúng ta cần cài đặt môi trường chạy Node.js và trình quản lý gói npm đi kèm của nó vào hệ thống của bạn. Electron khuyến khích nên sử dụng phiên bản hỗ trợ dài hạn (LTS) mới nhất.

3.1.4 Electron Forge

Electron Forge là một công cụ tất cả trong một (all-in-one) để đóng gói và phân phối các ứng dụng Electron. Nó kết hợp nhiều gói đơn mục đích để tạo ra một quy trình xây dựng hoàn chỉnh hoạt động ngay lập tức, hoàn chỉnh với việc ký mã, trình cài đặt và xuất bản tạo phẩm. Đối với quy trình công việc nâng cao, logic xây dựng tùy chỉnh có thể được thêm vào vòng đời Forge thông qua API plugin của nó. Các mục tiêu lưu trữ và xây dựng tùy chỉnh có thể được xử lý bằng cách tạo Nhà sản xuất và Nhà xuất bản của riêng bạn.

Sau khi ứng dụng của chúng ta sẵn sàng được phát hành, Electron Forge có thể xử lý phần còn lại để đảm bảo ứng dụng đến tay người dùng cuối. Quy trình xây dựng hoàn chỉnh cho Electron Forge có thể được chia thành ba bước nhỏ hơn:



Hình 3.1.4 Build Lifecycle

3.2 Lập trình ứng dụng

3.2.1 Sử dụng HTML, CSS, và JavaScript

HTML: Định rõ cách sử dụng HTML để xây dựng cấu trúc giao diện người dùng. Bao gồm các thẻ, thuộc tính, và tiện ích HTML phổ biến trong môi trường Electron.

CSS: Hướng dẫn cách áp dụng CSS để thiết kế và tạo kiểu cho giao diện người dùng. Đặc biệt, nêu rõ những điểm cụ thể liên quan đến tích hợp CSS trong Electron.

JavaScript: Mô tả cách sử dụng JavaScript để thêm tính năng và tương tác vào giao diện. Đồng thời, tập trung vào các API và khả năng JavaScript được hỗ trợ bởi Electron.

Thư Viện và Framework: Ngoài ra chúng ta còn có thể áp dụng thư hiện và framework UI phổ biến mà ta có thể tích hợp để tăng cường khả năng phát triển và quản lý mã nguồn.

3.2.2 Nguyên tắc hoạt động chính (Giao Tiếp Giữa Main Process và Renderer Process)

Sự tương tác giữa *Main Process* và *Renderer Process* trong Electron được thực hiện thông qua cơ chế gọi là Inter-Process Communication (IPC). IPC cho phép truyền thông dữ liệu giữa các quy trình khác nhau để thực hiện các chức năng như truyền thông giữa *Main Process* và *Renderer Process*.

Main Process Gửi IPC: *Main Process* có thể gửi các thông điệp IPC đến các *Renderer Process* để thực hiện các tác vụ như cập nhật giao diện người dùng, xử lý sự kiện người dùng, và truy cập các tài nguyên hệ thống.

Renderer Process Gửi IPC: Ngược lại, *Renderer Process* cũng có thể gửi các thông điệp IPC đến *Main Process* để thông báo về sự kiện hoặc yêu cầu thực hiện các tác vụ liên quan đến hệ thống.

CHƯƠNG 4. THỰC NGHIỆM VÀ DEMO

4.1 Đề tài triển khai

Bài toán của chúng tôi là xây dựng một ứng dụng Point of Sale (POS) dành cho cửa hàng bán sách. Ứng dụng này sẽ giúp quản lý hàng tồn kho, ghi nhận bán hàng, và tạo hóa đơn một cách thuận tiện. Chúng tôi chỉ sẽ tập trung vào việc quản lý danh sách sách, quản lý tài khoản nhân viên.

4.2 Quá trình xây dựng ứng dụng

4.2.1 Khởi tạo dự án

Bắt đầu dự án bằng cách sử dụng lệnh khởi tạo dự án ElectronJS với phiên bản cụ thể 27.1.2. Điều này bao gồm việc thiết lập cấu trúc dự án, tạo các thư mục cần thiết, và khởi tạo cửa sổ ứng dụng chính. Sử dụng các công cụ quản lý gói như npm để cài đặt các thư viện và dependencies cần thiết cho dự án.

4.2.2 Phát triển ứng dụng

Xây Dựng Giao Diện Danh Sách Tài khoản và Sách: Sử dụng HTML, CSS, và JavaScript để xây dựng giao diện hiển thị danh sách sách trong ứng dụng. Điều này bao gồm việc sử dụng các thành phần và thư viện UI phù hợp với ElectronJS.

Khởi Tạo Đối Tượng Tài khoản và Sách: Sử dụng JavaScript để khởi tạo và quản lý đối tượng tài khoản và sách. Điều này bao gồm việc xử lý các thao tác như thêm mới, cập nhật thông tin, và xóa từ danh sách.

4.2.3 Cơ sở dữ liệu

Triển Khai Database Trên Mongo Atlas: Với mục tiêu hỗ trợ cho nhiều chi nhánh của chuỗi cửa hàng sách và chia sẻ dữ liệu, chúng tôi quyết định triển khai cơ sở dữ liệu trên dịch vụ điện toán đám mây MongoDB Atlas. Mô tả cách ứng dụng tương tác với cơ sở dữ liệu để lưu trữ và truy xuất thông tin sách và nhân viên.

4.2.4 Triển khai

Đóng Gói và Triển Khai trên Windows và Linux: Mô tả quy trình đóng gói ứng dụng để triển khai trên hệ điều hành Windows và Linux. Sử dụng công cụ Electron Forge để tạo các phiên bản thực thi cho các nền tảng này.

Triển Khai trên MacOS: Gặp khó khăn khi đóng gói cho MacOS vì một số lý do như: Chỉ có thể đóng gói DMG file từ máy macOS và Yêu cầu chi phí cho việc Sign Code vì lý do bảo mật.

4.3 Video demo

Link: https://youtu.be/CCiOGaHp8gQ

CHƯƠNG 5. KẾT LUẬN

5.1 Đánh giá và so sánh

5.1.1 Ưu điểm của ElectronJS

Đa Nền Tảng: ElectronJS cho phép xây dựng ứng dụng đa nền tảng một cách hiệu quả, giảm đáng kể công sức khi phát triển cho nhiều hệ điều hành.

Tích Hợp Dễ Dàng: Việc tích hợp với Node.js và Chromium mang lại khả năng mạnh mẽ và linh hoạt cho ứng dụng, đặc biệt là khi cần sử dụng các tính năng web hiện đại.

Giao Diện Được Cải Thiện: Việc nhúng giao diện web là một phần giúp cho giao diện người dùng trở nên sinh động và đẹp mắt hơn với CSS và Javascript.

Ngôn Ngữ Lập Trình Quen Thuộc: Sử dụng HTML, CSS, và JavaScript làm ngôn ngữ lập trình chính, giúp những người làm web có thể nhanh chóng làm quen với phát triển ứng dụng desktop.

5.1.2 Nhược điểm của ElectronJS

Tài Nguyên Tiêu Thụ: ElectronJS có thể tiêu tốn nhiều tài nguyên hệ thống so với một số ứng dụng desktop truyền thống, đặc biệt là khi xử lý ứng dụng lớn.

Kích Thước File Lớn: Ứng dụng đóng gói bằng ElectronJS có kích thước file lớn hơn so với một số framework khác.

5.1.3 Ưu điểm của .Net Windows Form và Java Swing

Hiệu Suất: Thường có hiệu suất tốt hơn đối với ứng dụng desktop truyền thống và yêu cầu ít tài nguyên hệ thống.

Kích Thước File Nhỏ: Kích thước file của ứng dụng thường nhỏ hơn so với ứng dụng ElectronJS.

5.1.4 Nhược điểm của .Net Windows Form và Java Swing

Ngôn Ngữ Lập Trình: Sử dụng C# hoặc Java có thể đòi hỏi một học đường mới đối với những người làm web không quen với các ngôn ngữ này.

Đa Nền Tảng: .Net Windows Form thường chỉ hỗ trợ trên hệ điều hành Windows, trong khi Java Swing hỗ trợ đa nền tảng nhưng có thể gặp vấn đề về giao diện người dùng không đồng nhất.

5.1.5 Tổng kết đánh giá và so sánh

ElectronJS thích hợp cho các ứng dụng đa nền tảng với độ linh hoạt cao, nhất là khi đội ngũ phát triển làm chủ yếu với ngôn ngữ web.

.Net Windows Form và Java Swing thích hợp cho các ứng dụng yêu cầu hiệu suất cao và tối ưu hóa tài nguyên hệ thống, đặc biệt là trên một hệ điều hành cụ thể.

5.2 Hướng phát triển

5.2.1 Các tính năng và cải tiến dự kiến

Gửi email xác thực cho tài khoản nhân viên để được kích hoạt và sử dụng ứng dụng POS.

Tích hợp máy in để in nhãn barcode cho sản phẩm.

Tạo tài khoản và quản lý khách hàng; bao gồm tích lũy điểm thưởng để nhận được mức giảm giá cho lần mua hàng tiếp theo.

Hoàn hiện chức năng thanh toán cho cashier; bao gồm mở ca, kết ca và in hóa đơn sau khi thanh toán.

Hoàn thiện chức năng quản lý hóa đơn.

Hoàn hiện chức năng dashboard.

5.2.2 Đề xuất sử dụng Electron cho các dự án tương lai

Dễ Dàng Tích Hợp Với Hệ Thống Hiện Tại: ElectronJS đã chứng minh khả năng tích hợp tốt với Node.js, giúp kết nối với các hệ thống backend hiện tại một cách dễ dàng.

Giao Diện Người Dùng Linh Hoạt: Sử dụng HTML, CSS, và JavaScript cho phép phát triển giao diện người dùng linh hoạt và dễ dàng thích ứng với các thay đổi trong yêu cầu thiết kế.

Đa Nền Tảng: ElectronJS là lựa chọn lý tưởng khi cần phát triển ứng dụng POS đa nền tảng để hỗ trợ các cửa hàng bán sách trên nhiều hệ điều hành.

Sự Hỗ Trợ Từ Cộng Đồng: Cộng đồng người phát triển ElectronJS rộng lớn, điều này đồng nghĩa với việc có nhiều tài liệu, thư viện, và hỗ trợ trực tuyến hữu ích.

Khả Năng Mở Rộng: ElectronJS cho phép mở rộng dễ dàng khi cần thêm tính năng mới vào ứng dụng POS theo thời gian và theo nhu cầu kinh doanh.

TÀI LIỆU THAM KHẢO

Tiếng Anh

Electronjs Document, (2023). https://www.electronjs.org/docs/latest/

Electron Forge Document, (2023). https://www.electronforge.io/