Introduction:

* Existing applications
  + Screenshots
  + Description
  + Advantages
  + Disadvantages
* Functional Requirements
  + It should do this and that
* Non-functional requirements
  + It should be this and that
* Use cases:
  + As a user, I want this and that.
* System Requirements
  + What will the app use and how each part interacts
  + Platforms
  + Chosen rationale
* System model diagram
* Feasibility Study
  + Risks and challenges
* Project plan

# Introduction

The application is a Music Sequencer similar to existing MIDI software but will have heavy emphasis on exploring the technologies of Cross-Platform/Hybrid JavaScript applications for Desktop and Web applications using existing Cross-Platform frameworks currently available.

The main goal is to research the benefits and trade-offs of implementing JavaScript applications natively versus Web based JavaScript applications without straying too far from the JavaScript development ecosystem.

# Requirements Analysis

This section will explore existing Music Sequencing applications and brief look into how they are function. It will also look at existing Cross-platform JavaScript libraries and what features they offer.

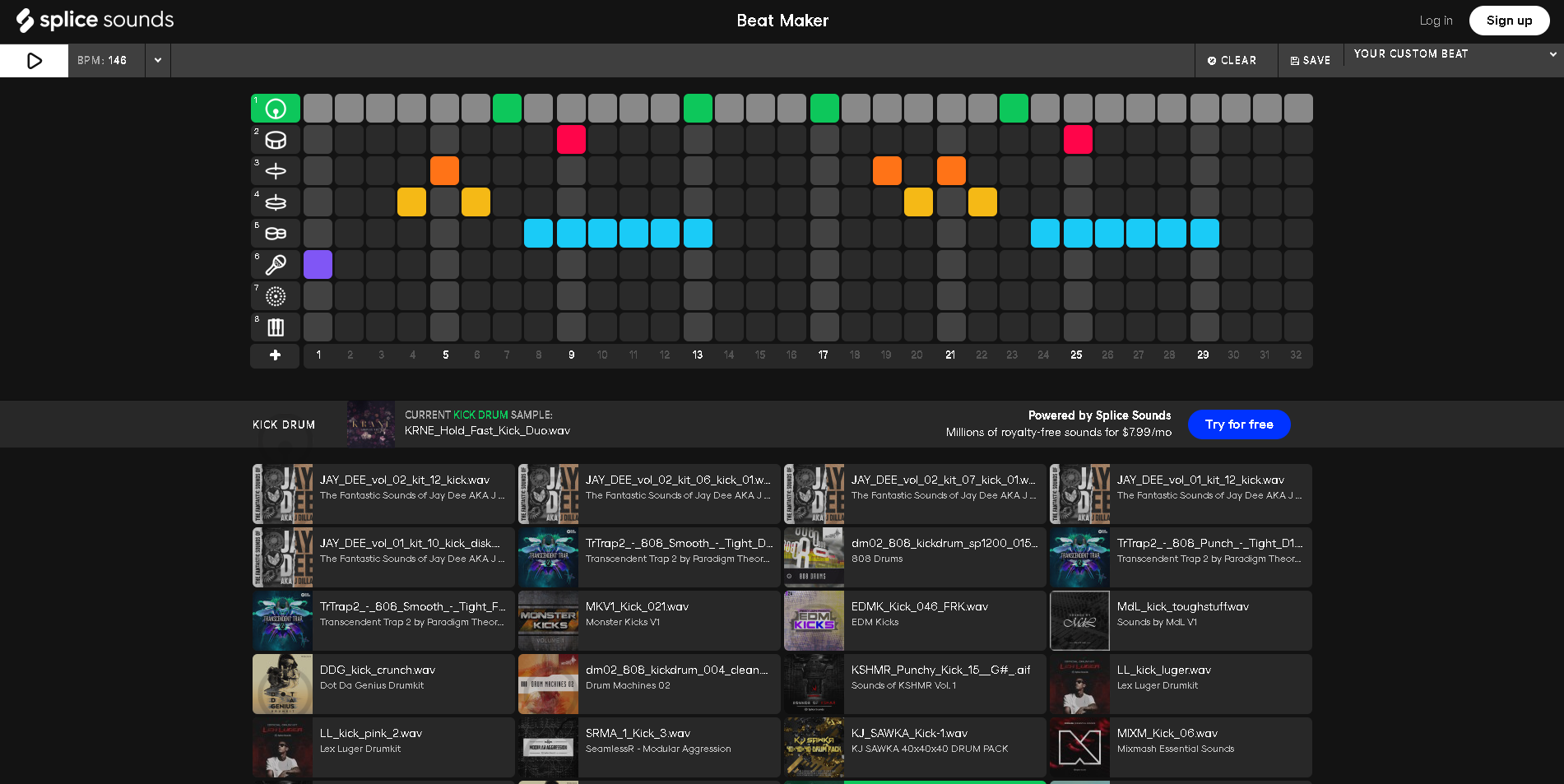
This will give insight on how the project application may different ways of implemented both in terms of front-end design and in terms of possible JavaScript libraries to facilitate the application functionality.

## Existing Applications

### Music Sequencers

The first music sequencer uses a square-based grid system on which sound samples are then played on coloured squares when the timeline passes said squares. This application works the same as a music cylinder from traditional music boxes.

The sound of each track can be swapped out with by various sound samples in the bottom part of *figure 1.*

Figure 1 Splice Sounds

In *figure 1,* it shows the interface of splice sounds. The timeline moves across the grid and plays each colour a certain sound and continues giving the illusion of a continuous musical piece.

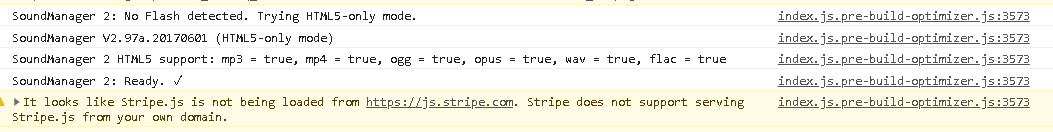


Figure 2 Splice Sounds Inspect

Here is the inspect element of the web page, it does not show how the grid was displayed but does give the name of the JavaScript library used to play the sound samples. The description of this library in *figure 3* on its GitHub page.

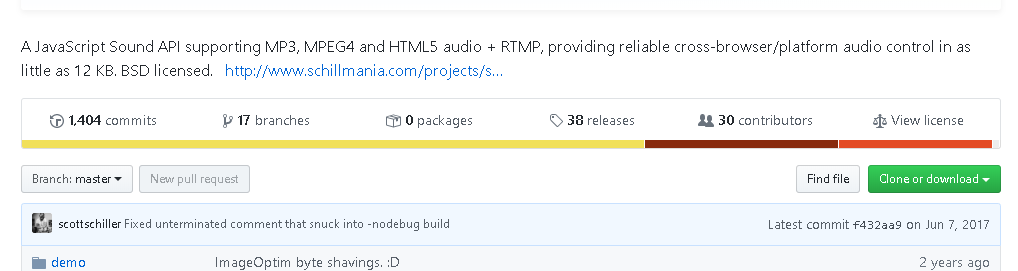


Figure 3 SoundManager GitHub

The key point being cross-browser/platform, this may be important for developing the project application as a way to manage audio data, which is discussed later in the document.

Advantages to this grid system:

* Straight forward to plan a data flow design and its implementation as the sound events are tied to each square element.
* User interface friendly, this allows users to immediately jump in creating music without knowing extensive musical theory as they abstract the square grids to musical sounds.
* Sound samples can be integrated easily into the users track; any sound they desire to use in the application.

Disadvantages of the grid system:

* To accommodate more advanced musical concepts such as major or minor keys, tracks have to be created for each sound, which can cause bloating of the UI.
* Does not visualise extended sounds, each sound sample is limited to one square even if the sound that plays extends beyond one beat.
* The timeline always starts from the beginning of the track in this application, which can waste time, as the track gets larger and larger.

The second application works similar to the first in having a music cylinder style of playback but instead of preparing a grid with musical ques, difference is that this has a Launchpad interface and the ability to record directly from the user as they play with the Launchpad.

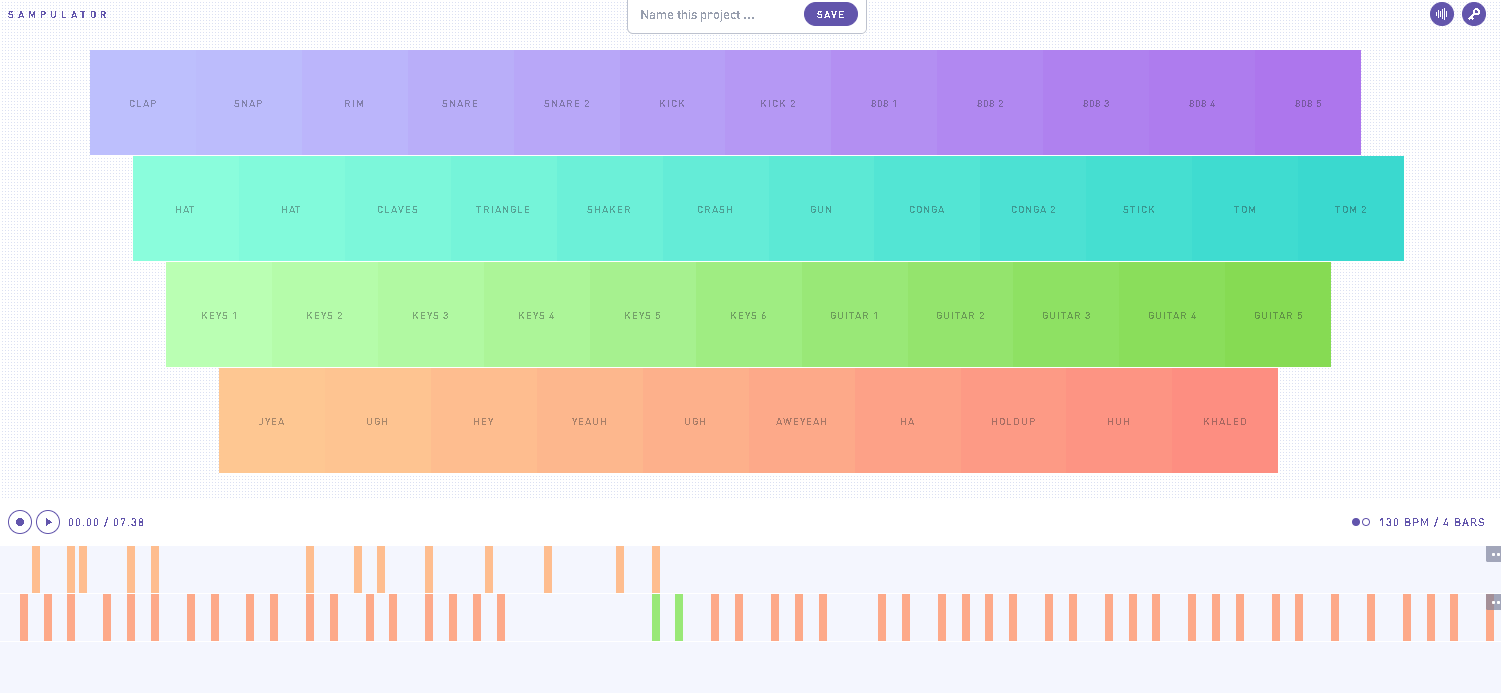


Figure 4 Sampulator

As shown in *figure 4* Launchpad UI and timeline UI. A graphical element moves across the timeline and plays sound when it strikes a coloured rectangle.

Compared to the sound splicer is also had predefined sound samples that can be played, these are mapped directly to the keyboard. When recording any input is mapped on to the time line at that point in time and be can be played back accordingly.

Looking into how the audio is managed not much is returned when inspecting the page element.

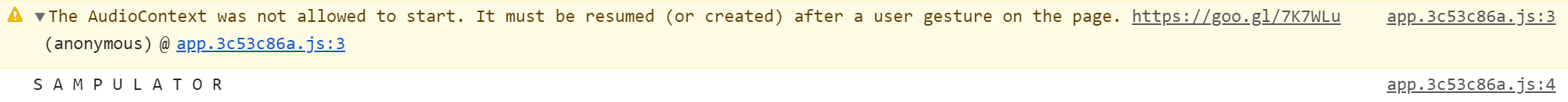


Figure 5 Sampulator Inspect

In *figure 5,* it would seem that the page directly utilises the Web Audio API. This allows browsers using JavaScript to process and synthesizing audio for web applications. [1] Sampulator only relies on one minified JS file, which means it utilises the Web Audio API directly to achieve its functionality.

There is

Advantages to the Launchpad system:

* Relatively straight forward to implement as well as the sound event will be dependent on user input compared to the square grids but also implements that grid system on the timeline where the user can click and drag an element on the timeline then place it somewhere else.
* User interface also allows abstraction of musical notation to the user interface elements. It is not as grid like compared to the first and not as expanded, for large-scale production more elements can fit on the track.
* Ability to place the timeline position at any point in the timeline UI component and play from that point. This makes large productions more feasible.
* Each track can support multiple sound samples, leading to a more compact UI.

Disadvantages to the Launchpad system:

* Similar to *splice sounds* there is no visual ques to indicate that a sound sample plays longer than one beat.
* Input has to be done directly from the Launchpad and edited later to correct input mistakes.
* Since each track can support multiple sound samples the UI can become cluttered when going for large productions.

# Existing Cross-platform frameworks

This section will explore existing Cross-platform frameworks that can accommodate the project application. There are a few to choose from with both in terms of advantages and disadvantages. The comparison of these frameworks will be a performance and implementation metric basis.

## Proton Native

Proton Native is a React based framework used for building native desktop applications using native operating system libraries while not straying too far from a traditional React development ecosystem and uses React Native syntax.

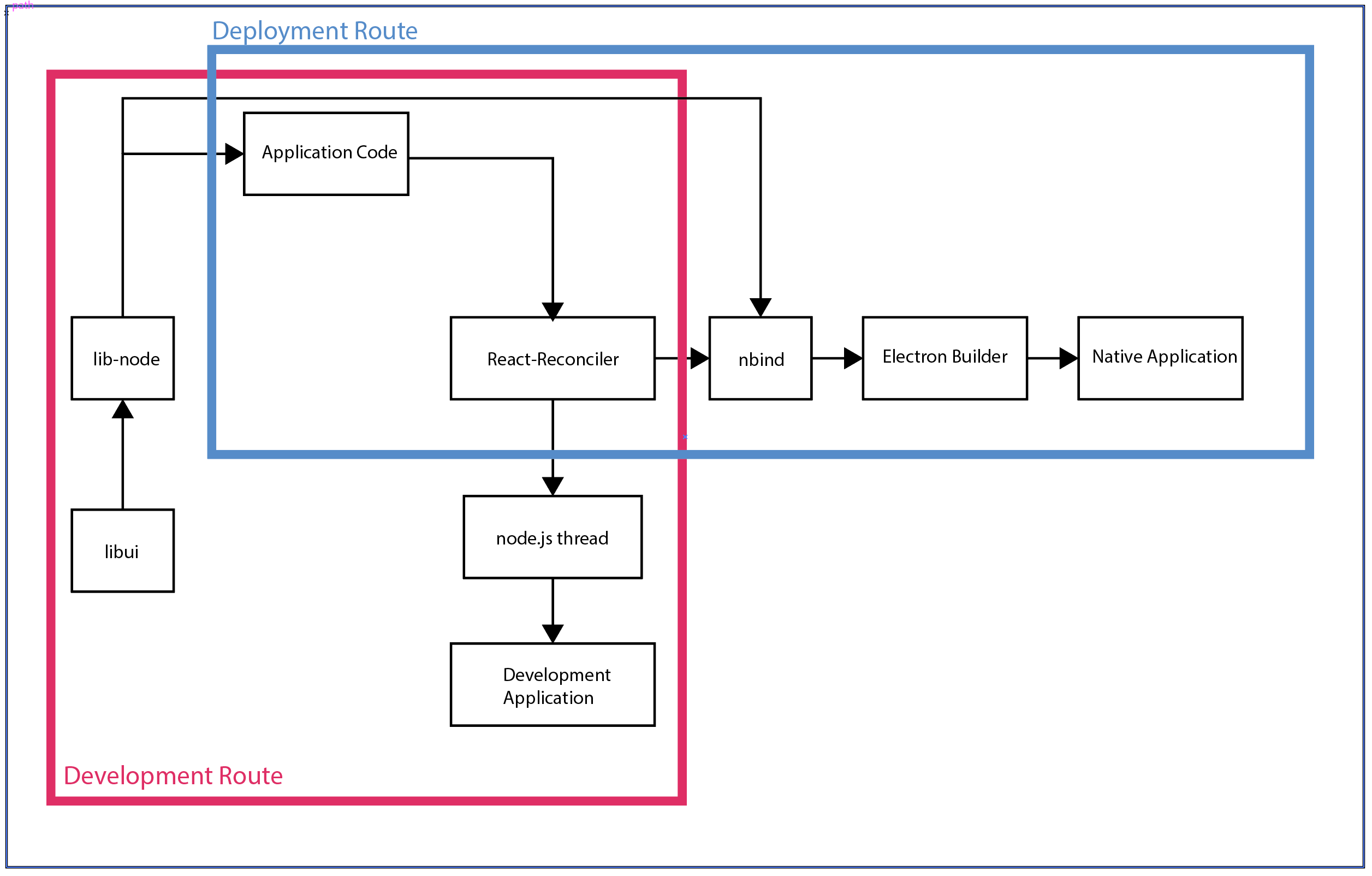


Figure 5 Proton Native Architecture

This Proton Native Architecture to my current understanding it may not be 100% as shown in *figure 5* and may change over time, but there is no doubt that when developing and deploying an application it goes through different processes.

The application code is where the developer would build their application and its functionality.

The *libui* component is GUI library written C that uses the native GUI technologies of each platform it supports [2]. The *lib-node* component allows react to interface with the *libui* library as React Components ready to be used in the application code.

*React-Reconciler* is what Proton Native uses to render the react code into whatever the render target. In web-based React applications, it uses the React DOM to render the code onto the document DOM. React Native renders react code primarily into native mobile binaries. Whereas the React-Reconciler is an experimental package for creating custom React renderers [3], in this case it is focused for Desktop rendering.

The *nbind* component works similar to *lib-node* component but binds the *libui* libraries to the React components such that both languages of JavaScript and C components can communicate interchangeably.

*Electron Builder* allows for the packaging of how *React-Reconciler* renders the application along with the *libui* bonded onto *nbind* and wraps the compiled code into a package containing the native application and in the case of windows an installed with uninstaller and an executable file.

Advantages of Proton Native:

* Near pure native application is created for the platform render to as there is no intermediary routines or compiling at execution, Proton Native offers great performance. Using less CPU and RAM usage. As shown in *figure 6* even though it is a very small example application.*.*



Figure 6 Proton Native Process

* It uses React Native Syntax, people who are already familiar with how React Native code structure goes they will pick up Proton Native… natively.
* Setting up development projects is straight forward as well as it includes its own CLI commands to get a project up and running quickly.

Disadvantages of Proton Native:

* It is a very new Cross-platform framework being released publicly in this year of 2019, its components maybe native but there is not much functionality outside of those components. You would have to improve upon the framework outside of JavaScript.
* Currently there is no styling options available for the native components. There is a very basic grid system similar in implementation to bootstrap but it is not even near the same flexibility.
* By extension, it has no innate support for audio manipulation in its components; a custom component will be needed to accommodate this.
* Due to the near perfect level of compiled application, if the requirements of the application need it to work for web, a completely new application must built from scratch.

Although the technology is very promising it is not able to support complex applications such as a Music Sequencer, right now it can only accomplish basic applications.

## Electron JS

This is a more established framework starting development in 2013 and made open source in 2014 [5], a five-year gap compared to Proton Native’s release. It uses Chromium for its front-end rendering as stated in their core philosophy *“Chromium's rendering library rather than all of Chromium. This makes it easier to upgrade Chromium but also means some browser features found in Google Chrome do not exist in Electron”*[5]. Which means it can parse HTML and CSS but not JavaScript. This is where Node.js comes in to facilitate executing JavaScript code. It functions very similarly to a *Java Runtime Environment* but for JavaScript.

Similar to Proton Native, development and deployment processes are split but not to the same extent.

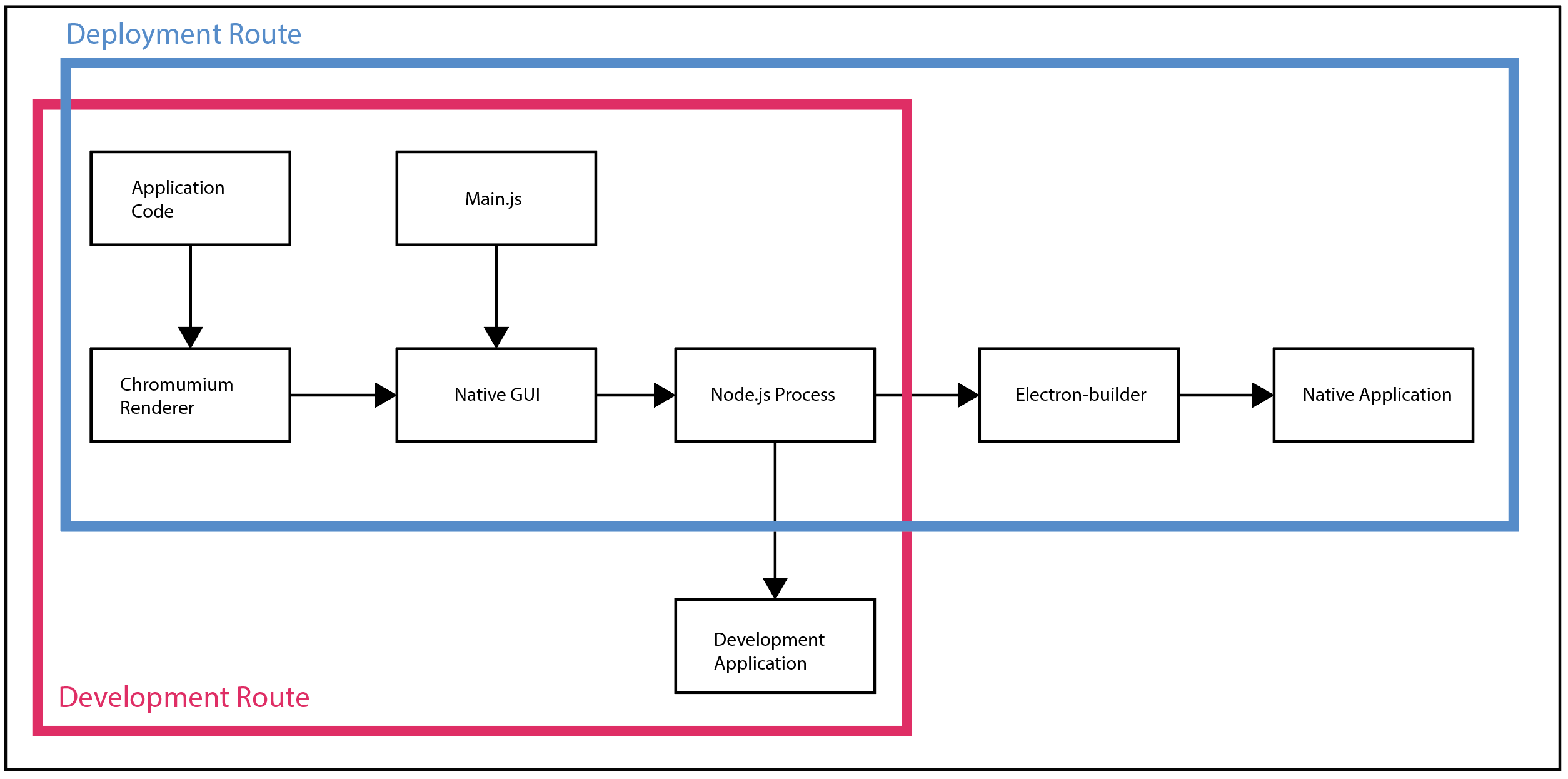


Figure 7 Electron Architecture

Much like for Proton Native this ElectronJS architecture path my not be 100% accurate to my current knowledge. When comparing the file structures of the development folders and the distributed folders there is clear difference.

*Application code* refers to any code written for the application which has the same approach to creating web pages, be it HTML, CSS or JavaScript files. This includes an index.html file.

*Main.js* is what Node executes first to achieve a Native GUI window to display, it then loads *Chromium* to be placed as the content of the Native GUI window and subsequently loads in the index.html file and cascades to the rest of the *Application code*.

*Chromium Renderer* is Electron is able to parse normal HTML and CSS files onto the window. It does not contain some functionality as you would expect from a traditional browser. However it does follow multithreaded rendering of the application, it also creates more threads the more Chromium windows are rendered. As show in *figure 8* even though the test application is simple, Electron still uses a bigger pool of RAM at 76MB*.*



Figure 8 Comparison of Electron and Google Chrome Processes

*Native GUI* is what Chromium sits on and thus renders it implements *SwiftShader*; it is a high-performance CPU-based implementation of the OpenGL ES and Direct3D 9 graphics APIs12. Its goal is to provide hardware independence for advanced 3D graphics [6]. This means that the same rendering can occur on any platform regardless of what hardware the platform is running on. This boosts Electron’s Cross-platform capabilities.

*Node.js Process* refers to the back end of the application for development and deployment versions the main process on which creates new windows and in turn, renderer processes.

*Development Application* is a version of the application when it has not been compiles and packaged to its target platform.

*Electron builder* is what is used to compile and package the application to its target platform and thus becomes a native application with an installer, uninstaller and executable file for windows at least. Electron builder does not come with the installation of Electron however since there are different builder libraries suited for specific needs.

Advantages of Electron:

* Almost identical development ecosystem to building web pages for web. HTML, CSS and JavaScript. This allows for the same skillset to be applied to different problems and solutions.
* More support and functionality compared to Proton Native having been around longer.
* Ability for Hybrid applications as the code base of the application tend to be the same for the web.

Disadvantages of Electron:

* Requires many initial resources from the platform in particular RAM usage and application size. Even simple applications need upwards of +100MB of RAM to run. Which may not suit lower end machines. However, with proper window management the RAM usage can be minimized by a substantial amount but not so much about application size. A compiled simple application can take up +200MB of disk space. This is due mostly because of Chromium itself.
* When trying to use JavaScript libraries with the Electron app, one may need to have workarounds with other libraries otherwise they would not have used and this can cause further application size growth.
* No innate integration with the React framework, which ties to the previous disadvantage. This is a project application specific disadvantage.

Requirements Modelling

Now these technologies what is the application going to be built with. In addition, rationale.

# References

[1] 2011. Getting Started with Web Audio API - HTML5 Rocks. *HTML5 Rocks - A resource for open web HTML5 developers*. Retrieved October 29, 2019 from

<https://www.html5rocks.com/en/tutorials/webaudio/intro/#targetText=The%20Web%20Audio%20API%20is,modern%20desktop%20audio%20production%20applications.>

‌[2] andlabs. 2019. andlabs/libui. *GitHub*. Retrieved October 29, 2019 from <https://github.com/andlabs/libui>

[3] facebook. 2019. facebook/react. *GitHub*. Retrieved October 29, 2019 from <https://github.com/facebook/react/tree/master/packages/react-reconciler>

[4]

charto. 2019. charto/nbind. *GitHub*. Retrieved October 29, 2019 from <https://github.com/charto/nbind>

‌[5] 2013. About Electron | Electron. *Electronjs.org*. Retrieved October 30, 2019 from [https://electronjs.org/docs/tutorial/about#targetText=Electron%20began%20in%202013%20as,who%20is%20building%20on%20Electron](https://electronjs.org/docs/tutorial/about%23targetText=Electron%20began%20in%202013%20as,who%20is%20building%20on%20Electron).

[6]

Mohamed Mansour. 2017. How to run Electron JS in Software Rendering Headless. *Medium*. Retrieved October 30, 2019 from <https://medium.com/@mohamedmansour/how-to-run-electron-js-in-software-rendering-headless-49601b87961e>

‌

‌

‌

‌

‌