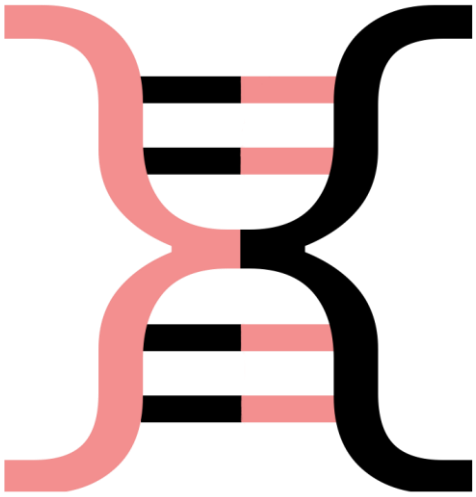
Technological Feasibility Analysis

9 November 2018

Team PathLab



Sponsor
Viacheslav Fofanov
Mentor
Isaac Shaffer

Team
A. Turan Naimey
Alexandre Lacy
Chance Nelson
Austin Kelly

Table of Contents

1. Introduction 2
2. Technological Challenges 3
3. Technology Analysis 3
3.1 Cross-Platform Compatibility 3
3.2 User Interface & Accessibility 9
3.3 Data Visualization 11
4. Technology Integration 13
5. Conclusion 15

1. Introduction

Traditionally, pathogen detection and identification is a slow and laborious process. Recently, however, our sponsor has created a revolutionary tool that utilizes the number-crunching strength of computers to assist in fast, reliable, and accurate organism identification. This efficiency increase drastically reduces the potential harm that a pathogen outbreak can cause before it is correctly identified and neutralized.

Our job, as computer scientists, is to take the tool provided to us and morph it into a beautiful, yet simple, system. Currently, the tool exists solely as a set of command line driven modules. While it may seem like such an approach does not need further improvement, many biologists worldwide are hindered from entering the field of comparative genomics due to the steep learning curve presented by command line programs. Accessibility to these biologists is why we are going to take the command line tools and build an easy to use, user friendly Graphical User Interface (GUI) that streamlines a key aspect of the identification process.

As a team, we represent a wide range of skills and talents, each of which is well suited to the task at hand. Turan Naimey, our team lead, has significant past experience with similar tools used in the field of Biology. This experience allows him to draw needed information from our client, as well as create a vision for our team to follow. Chance Nelson, our development operations specialist, is intimately familiar with repository management, as well as all of the tools and systems we will need to use to complete this project. Alex Lacy will provide the expertise on any large data back-end work that will be required, as well as familiarity with a variety of documentation styles and tools. Finally, Austin Kelly is an expert with user interface design, and brings the artistic eye and elegant touch to our team.

Our sponsor for the project is the Fofanov Bioinformatics Lab at Northern Arizona University. Specifically, Dr. Fofanov is the principal investigator of the lab; he is the one who requested our services. While Dr. Fofanov is ultimately responsible for the work of the lab, Dr. Furstenau is the one who developed the command line tools and is our point of technical contact and inquiry within the lab. Together, Team PathLab and Fofanov Bioinformatics Lab hope to make a satisfactory and compelling product.

The goal of this document is to highlight the insight and reasoning that decided our technologies of choice. We start by reviewing the various technological challenges we expect to face. This will highlight the important features our solution will have, as well as discuss the major challenges associated with each. In the next section, we perform in-depth analysis of each challenge, as well as discuss our solutions for those challenges, giving examples to demonstrate the validity of our choice where necessary. Finally, we end the document with a discussion about how our chosen technologies will work together, along with several graphs and tables to demonstrate our decisions.

2. Technological Challenges

The robustness of the underlying program aside, engineering a GUI for any kind of toolchain can provide a modicum of unique challenges when it comes to the technological backbone of the service. In order to fully understand the technical challenges presented in this particular implementation, it is integral to understand the base requirements of the system:

- **Cross Platform**: The tool must be fully functional on OSX, Windows, and Linux environments with no extra effort or maintenance.
- **Pipeline Traversal**: The tool should be capable of moving forwards through the pipeline, and also capable of moving back to previous steps to repeat those operations under new configurations and inputs.
- **Error Correction**: The tool should be capable of catching malformed inputs (negative temperatures, corrupted genome inputs, etc.) and informing the user of the error.
- **Maintainable and Expandable**: The tool should be built in such a manner that the code is both as readable and as decoupled as possible. This is to promote expansion, given the need in the future.
- **Visually Pleasing**: The tool must provide an aesthetic quality above that of the standard GUI toolkit that the common languages and frameworks have built in.

These technological challenges generally revolve around the intent to create a useable, painless experience for users of average experience. A tool which meets all of the requirements would be an ideal outcome for this project, and leave the client satisfied on the outcome. With the challenges now clearly defined, it is now possible to begin selecting technologies that would fit best within this particular use case.

3. Technology Analysis

In any given software development project, the first big question from the development team is which technologies should be used to get the job done. Often times, software engineers tend to gravitate towards using the newest and hottest technologies to achieve their technical goals, but with so many new frameworks released into the market so frequently, it is significant to choose the right technology to ensure proper completion of the project without any hassles. In this section, we discuss the major technological issues in more detail and analyze different technologies which can be suitable to overcome these challenges. We will use a scoring system of 1-3 where 3 is the highest; the technology with the most points will be the likely pick for the relevant issue. We will discuss Cross-Platform Compatibility, User Interface, and Data Visualization as some of the components of this GUI to pick the suitable technologies.

3.1 Cross-Platform Compatibility

Our client's pipeline will be a great resource to many researchers around the world battling pathogen detection. One of the core requirements of this GUI is that it is executable and

functional on Linux, MacOS and Windows. This will ensure that the maximum number of users can access this pipeline and use it regardless of what operating system they employ on their machines. Ideally, we will choose a technology that will ease the creation of cross-platform desktop application without any extra configuration for each operating system.

3.1.1 Approach

Cross-platform development allows our team to design and implement one software without having to think about implementing our GUI in every platform natively. Cross-platform development also makes maintainability easier because there is only one code base. Some of the options we have considered for our development are Haxe, Kivy, NW.js, 8th, Xojo and Electron.

3.1.2 Options

- **Haxe**
  Haxe is an open-source, high-level, and cross-platform programming toolkit. Haxe allows the creation applications for numerous target platforms. Haxe is not cross-platform development framework; rather it is a multiplatform language compiled for a required platform. Haxe is a full language pack containing a multitude of different libraries including Node, Electron and many more.
  - **Advantages**
    - ECMA-script based syntax provides familiarity to new comers
    - Library extensions allow usage of different resources like Node
    - Faster than other similar compilers
  - **Disadvantages**
    - Extra layer of complexity due to the library extensions for different platforms
    - Difficult debugging process
    - Relatively new technology and the support/documentation is not that great
- **NW.js**
  NW or Node Webkit lets developers create cross-platform desktop applications using CSS/HTML, JavaScript and other web technologies like WebGL. One of the biggest advantages of NW.js is the support it provides for all Node.js modules and most other third-party modules making it easier for web developers to build desktop applications using web technologies.
  - **Advantages**
    - Easy to learn
    - Supports compiled JavaScript
    - Uses less resources compared to Electron
  - **Disadvantages**
    - Requires usage of extra bundles and extensions
    - Lacks features like auto-updater and crash-reporter
- **Kivy**

Kivy is an open-source Python GUI framework that can be used to create multi-touch cross-platform applications that can be executed on desktop as well as mobile platforms. Kivy provides a rich set of UI elements via their APIs to build robust applications.

- **Advantages**
  - Python - Easy to learn
  - Great documentation and support
  - UI library to implement GUI elements easily
- **Disadvantages**
  - Non-native UI elements
  - Slow startup time
  - Kivy needs large amount of disk space for development environment

- **Xojo**

Xojo is a programming environment available for Windows, Mac and Linux platforms. Xojo allows programmers to create a native user interface and deploy it once. Xojo's IDE features drag and drop UI set which can be very useful for new developers. This framework also supports many Object-Oriented Programming concepts like inheritance, polymorphism, and abstraction.

- **Advantages**
  - Easy to learn and intuitive IDE
  - Great documentation and active community
- **Disadvantages**
  - Subscription based licensing - pay to use.
  - Limited support for integrating other technologies

- **Electron**

Electron was originally developed by GitHub to design their highly extensible and popular text-editor Atom. Electron is based on web technologies allowing web developers to create native desktop applications without having to worry about optimizing and packaging their application for each specific system. By using Node.js Electron allows developers to create cross platform applications using HTML/CSS and JavaScript.

- **Advantages**
  - Electron relies completely on web technologies, thus making it easy for web developers to get accustomed to and build applications.
  - Extensible Node.js modules allow the usage of third-party modules
  - Electron provides built-ins like Automatic Updater, Native UI elements, Crash reporting, Debugging, and Windows installers.
  - Electron comes with a set of APIs that allow access to several operating system features.
  - Electron is used by Facebook, Google, Microsoft and many other big names therefore the community support is great for this framework.
  - It allows developers to focus on the core functionality of the applications by already taking care of the hard parts in software development
  - Reuse of npm modules, out of the box

- **Disadvantages**
  - Electron apps can be bloaty and energy inefficient
  - Electron apps are known to withhold and use significant amounts of memory for processes if developed without proper care

- **Chromium Embedded Framework**

  CEF is a C++ framework for interfacing with the internal engine behind Chromium to create native GUIs with HTML5. It supports bindings for a variety of languages, including Python, Go, and Java. This framework can also be used to embed a browser into an existing application.

  - **Advantages**
    - CEF is actively supported by Spotify, with long-term stable versions to rely on.
    - The Chromium engine itself is platform agnostic and will run on all supported platforms.
  - **Disadvantages**
    - The language binding features supported by CEF are not platform agnostic, and have particular problems supporting GUI features for Linux.
    - CEF has unreasonably high resource requirements for what it offers, even higher than that of Electron.
    - While the framework provides HTML5 support and language bindings, which may be enough for many projects, it does not leverage all of Chromium's in-engine features, which could lead to difficulties down the road when interfacing with Primacy.

- **QT**

  QT allows the creation and launch of applications which are compatible with the majority of the modern operating systems. It uses simple compilation for each operating system without modifying the source code. QT features major classes that may be needed for the development of an application - from GUI elements to network related, databases and XML related classes.

  - **Advantages**
    - Multithreading allows developers to create optimized and performance enhanced cross platform applications.
    - Features one UI library which can be great for providing same user interface across all platforms.
  - **Disadvantages**
    - Developers need to have a good understanding of platform-specific UI libraries.
    - Since all the UI libraries used in the development of the application will be included within the finished application, the size of the application will be considerably large.

- **.NET**

  Microsoft's .NET framework is a C#-based toolkit for creating cross-platform applications. .NET is supported for Windows, OSX, Linux, IOS, and Android through

Xamarin's implementation, called Mono. This framework includes simple tools that streamline creating GUI's.

- ➢ **Advantages**
  - C# and provided and supports a wide variety of libraries, which will shorten the implementation time of the final product.
  -
- ➢ **Disadvantages**
  - While Mono supports all platforms on paper, Linux support can be fairly unreliable and crashes often.
  - Applications created using the Winform GUI look fairly generic and would be a major time sink to modify to look aesthetically pleasing.

❖ **8th**

8th lets developers write code and produce applications for various target operating systems including Windows, Mac OS X, Linux, Raspberry Pi, Android as well as iOS. Its primary focus is providing cross-platform compatibility as well as security in the final software products or distributions.

- ➢ **Advantages**
  - It comes with support for various essential libraries, which minimizes the dependency on external libraries.
  - It provides enhanced protection by packing the applications in an encrypted container, making it tamper-resistant as well as difficult to crack.
- ➢ **Disadvantages**
  - Requires developers to opt for its paid subscriptions for its full functionality

3.1.3 Chosen Approach

One of the most appealing reasons we chose Electron is that it is easy to learn, as it uses web technologies and there is great amount of community support for it. Electron provides developers with a great amount of control over how the application feels and looks.

The table in Figure 1 shows how other technologies compared to Electron based on the criteria defined below.

UI/UX: Consistency of User Interface and User Experience across different platforms
Low-Level Accessibility: Access to the all hardware level access APIs
Reusability: Ability to reuse the same code for different platforms without any customization
Production: Ease of production with the ability to use open source libraries with more features
Cost/Time: Cost of using a framework and time it takes to learn the specific technology
Deployment: Ease of shipping and installing the app across different platforms

Technology	UI/UX	Low-Level Accessibility	Performance	Reusability	Production	Cost/Time	Deployment	Total
Haxe	3	2	3	2	3	1	3	17
NW.js	3	3	2	1	3	3	1	16
Kivy	1	3	3	3	3	3	3	19
Xojo	3	3	2	2	2	1	2	15
Electron	**3**	**3**	**2**	**3**	**3**	**3**	**3**	**20**
CEF	3	1	1	3	3	2	3	16
.NET	1	1	3	3	3	2	3	16
8th	3	1	3	2	3	1	3	16

Figure 1: Scoring chart for cross-platform compatibility. Scores range from 1-3, with higher numbers being better. Our chosen technology is highlighted.

3.1.4 Feasibility Proof

We believe Electron is the right tool for this project, as our team is quite familiar and comfortable with web technologies. We will be developing a demo for our client showcasing the capabilities of Electron within the next few weeks. Here are some of the reasons why we picked Electron over other technologies we evaluated.

- **Security**
  Electron creates desktop applications; thus, the data stays locally in the system. Because of this, data security is ensured, and users are immune to the cloud-based cyber-attacks which are often the problem in web applications.
- **Low-Level Accessibility**
  Electron lets developers implement low-level features like having extended interactive features in their application, such as Keyboard Shortcuts.
- **Performance**
  Electron thrives in this section and often does better in performance compared to native apps if development is done the right way.
- **Production**

The more known frameworks we use, the more support we get. This in turn gives us more open source libraries we can use. This decreases production time and allows for the implementation of more features.

- **Deployment / Build**
  This is one of the interesting aspects of the Electron. There is an electron-packager module available that helps us bundle the whole code base into the respective packages.
- **UI/UX**
  Because Electron apps are built with web technologies, it provides great control in designing excellent UI/UX. Developers can be confident of providing the same experience to all users who are on different platforms.

3.2 User Interface & Accessibility

Designing a good user interface is important in making sure that users on different platforms experience the same UI. Due to the robust nature of Electron developers can use multiple web technologies in tandem to create a simple and intuitive UI.

3.2.1 Approach

One of the core requirements of this project is to alleviate non-technical users from using command line-based tools and instead give them the ability to use this software using an intuitive Graphical User Interface. To achieve a user-interface that would be easy-to-use but also contain all the necessary information, we will take advantage of front-end frameworks used by web developers. We analyzed a few different front-end CSS frameworks to find the best fit for our project.

3.2.2 Options

- **Bulma**
  Bulma CSS development framework is the latest addition to the suite of modern CSS frameworks. Bulma is a lightweight and simple to use CSS stylesheet. In addition, Bulma CSS grid is built-in with Flexbox, which allows performing fancier footwork design than a normal standard grid.
  - **Advantages**
    - It has one of the easiest learning curves
    - Light weight with the entire framework coming in a package of less than 90 kilo bites.
    - Based on Flexbox which gives it a great sense of responsive design.
  - **Disadvantages**
    - Fairly new, not as large of a community compared to other technologies.
    - Relatively weak documentation and could use some minor improvements.
- **Flexbox**

Flexbox or officially known as the CSS Flexible Box Layout Module is a layout system in CSS3 made to improve the items align, directions and order in the container even when they are with dynamic or even unknown size.

- ➢ **Advantages**
  - Page content can be laid out in any direction.
  - Bits of content can have their visual order reversed or rearranged.
  - Items can "flex" their sizes to respond to the available space and can be aligned with respect to their container or each other.
  - Achieving equal-column layouts (irrespective of the amount of content inside each column) is a breeze.
- ➢ **Disadvantages**
  - Not a framework so most things need to be designed manually.

❖ **Bootstrap**

Bootstrap is the most commonly used CSS framework composed of HTML, JavaScript and CSS scripts. Bootstrap was designed by Twitter to be the interface used by the company to keep all the components and elements of their website consistent. Bootstrap has a great number of features that make coding a website easy and quick.

- ➢ **Advantages**
  - Fixes CSS compatibility issues and has a great support for all the major browsers.
  - Designed according to the mobile first approach, which makes it work great on all devices.
  - Superior Documentation and community support.
  - Great well rounded framework with consist UI for all of the basic components (forms, panels, tables, buttons) and many others.
- ➢ **Disadvantages**
  - Depends heavily on JQuery as it is a JavaScript library (which will leave much of the plugins unused).
  - A number of HTML outputs that are not perfectly semantic.
  - The need to override the style files to make the project look a little bit unique.

❖ **Foundation**

Foundation is a CSS framework designed by ZURB. It features a slightly more advanced interface compared to other frameworks. Foundation is compatible on multiple browsers and mobile devices. The responsive menu is one of its greatest assets when it comes to functionality which can also be easily styled using CSS.

- ➢ **Advantages**
  - Styles are purposefully underdeveloped to give more room for creativity and differentiate between sites that use Foundation.
  - It comes with a robust grid system.
- ➢ **Disadvantages**
  - Pretty hard for a beginner to grasp.
  - A messy documentation that needs major improvements.

- Doesn't have a variety of wild choices to choose from when it comes to pre-made CSS components.

3.2.3 Chosen Approach

To ensure we build an intuitive user interface, we have to pick a technology that allows the most flexibility in designing and creating a great interface. Flexbox allows developers to position elements of the user-interface relative to the screen or to other elements, giving the developer absolute control over the feel and look of the interface.

The table in Figure 2 shows how other technologies compare to Flexbox based on the criteria defined below.

Customization: Ability to easily modify pre-built layouts or create new ones.
Browser-Support: Support for all, some or very limited browsers/versions
Usability: Ease of use and learnability

Technology	Customization	Browser-Support	Usability	Total
Bulma	2	2	3	7
Flexbox	3	3	3	9
Bootstrap	3	3	2	8
Foundation	3	2	3	8

Figure 2: Scoring chart for user interface creation and accessibility. Scores range from 1-3, with higher numbers being better. Our chosen technology is highlighted.

3.2.4 Feasibility Proof

Austin has come up with a demo showing the capabilities of Flexbox using Electron which has convinced our team to use this technology.

3.3 Data Visualization

Due to the high complexity and volume of data being presented in the final product, data visualizations will be just as necessary as a quality user interface. This will allow the tool to present the data in a more accessible manner and promote quick pruning of data sets.

3.3.1 Approach

In order to achieve the goal of creating accessible data visualizations, detailed above, the team has chosen to utilize one of the plethora of frameworks available in Javascript.

This framework will need to be lightweight, due to performance concerns, intuitive, to speed up development and deployment timelines, and be capable of producing any visualizations needed to represent the particular data sets the toolchain will handle.

3.3.2Options

- **Chart.js**
  Chart.js is known as one of the standards for the simple creation of charts and data visualizations. It is fairly easy to integrate into any project, has built in animation support, and has responsive design.
  - **Advantages**
    - Chart.js possesses a simple, well documented API, with input data being interpreted as simple JSON objects.
    - Charts created with the service are lightweight and have little impact on performance.
  - **Disadvantages**
    - The service is fairly limited to a select group of charts and visualization schemes.
- **Vega**
  Vega is a highly expandable framework for creating various visualizations, not necessarily charts. With its grammar-based schema, any manner of visualization can be made.
  - **Advantages**
    - Any visualization one could need can be created.
  - **Disadvantages**
    - The JSON grammar-based schema used by the framework is extremely extensive and verbose, with some charts requiring grammars over 1000 lines long.
- **D3.js**
  Like Vega, D3 is a general-purpose SVG and HTML visualization editing library. However, D3 does not use any in-house grammars, and is thus easier to utilize.
  - **Advantages**
    - D3 is highly expandable and is commonly used for things other than charts. This would be useful for non-standard visualizations.
  - **Disadvantages**
    - While D3 is more usable that Vega, it's open nature still leaves its development time lacking.

3.3.3 Chosen Approach

Given that visualizations are one of the several keys to making a usable interface for the final product, it has been decided to use Chart.js as the framework for any graphs needed. It has been decided that it would be unlikely that any charts not included in the framework's standard

toolkit would be particularly needed for this use case. The ease of use of Chart.js will also come to benefit the team when the final push to production occurs later on, with its ease of use.

The table in Figure 3 shows how the technologies outlined above compare to Chart.js, with the following criterion:

Performance: Display of various charts and visualizations must not cause a noticeable degrade in performance.
Ease of Implementation: Development of systems to process data and build visualizations must take as little time as possible.
Versatility: The tool must be capable of creating visualizations that meet all of the demands of the final product.

Technology	Performance	Ease of Implementation	Versatility	Total
Chart.js	3	3	2	8
Vega	2	1	3	6
D3.js	2	2	3	7

Figure 3: Scoring chart for data visualization capability. Scores range from 1-3, with higher numbers being better. Our chosen technology is highlighted.

3.3.4 Feasibility Proof

There are a wide variety of quality examples with code samples available on the Chart.js documentation page. This has convinced the group to utilize this technology.

4. Technology Integration

These challenges restrict a few of our options for frameworks to pursue implementation with. In today's technological age, there are dozens of ways to approach making a GUI, and just as many tools to guide the process. We looked into the pros and cons of various technologies and ultimately chose Electron JS for a variety of reasons. Most importantly, it meets all the criteria needed for the requirements we have set for the delivery.

- **Cross Platform**: Electron makes it incredibly simple to build executables for multiple operating systems. This is one of the main reasons we believe it will be the best choice
- **Pipeline Traversal**: JavaScript is very robust and allows developers to interact with other interfaces easily. This makes working with the Primacy pipeline a breeze, and Electron will not require an abundance of low-level code to link the two together.

- **Error Correction**: Having assigned JSON objects with attributes, such as temperature, allows error correction to be completed at a high level with readable, and understandable code.
- **Maintainable and Expandable**: Javascript is one of the most commonly used programming languages today. The Electron framework does a great job of making different elements of the program easy to understand, and there are thousands of great tutorials freely available online for anyone struggling. The rule of keeping a consistent JSON format from the pipeline also allows future developers to rework the any part of the GUI without risking result inaccuracy.
- **Visually Pleasing**: Electron JS is designed specifically for making aesthetically pleasing graphical user interfaces. Beautiful applications like Google Chrome and Atom are built with the framework. The use of CSS also makes the design work very approachable to developers with any basic level of front-end experience.
- **Visualization:** We chose the JavaScript Chart.JS library for visualizations. It is one of the most well-known and expansive libraries for visualizations. Luckily, since we are already using Javascript with Electron, we will be able to create graphs without having to convert or transfer files to use another language.

Our GUI will simplify the process for biologists by removing the risk of error and perceived difficulty that comes with using the command line.

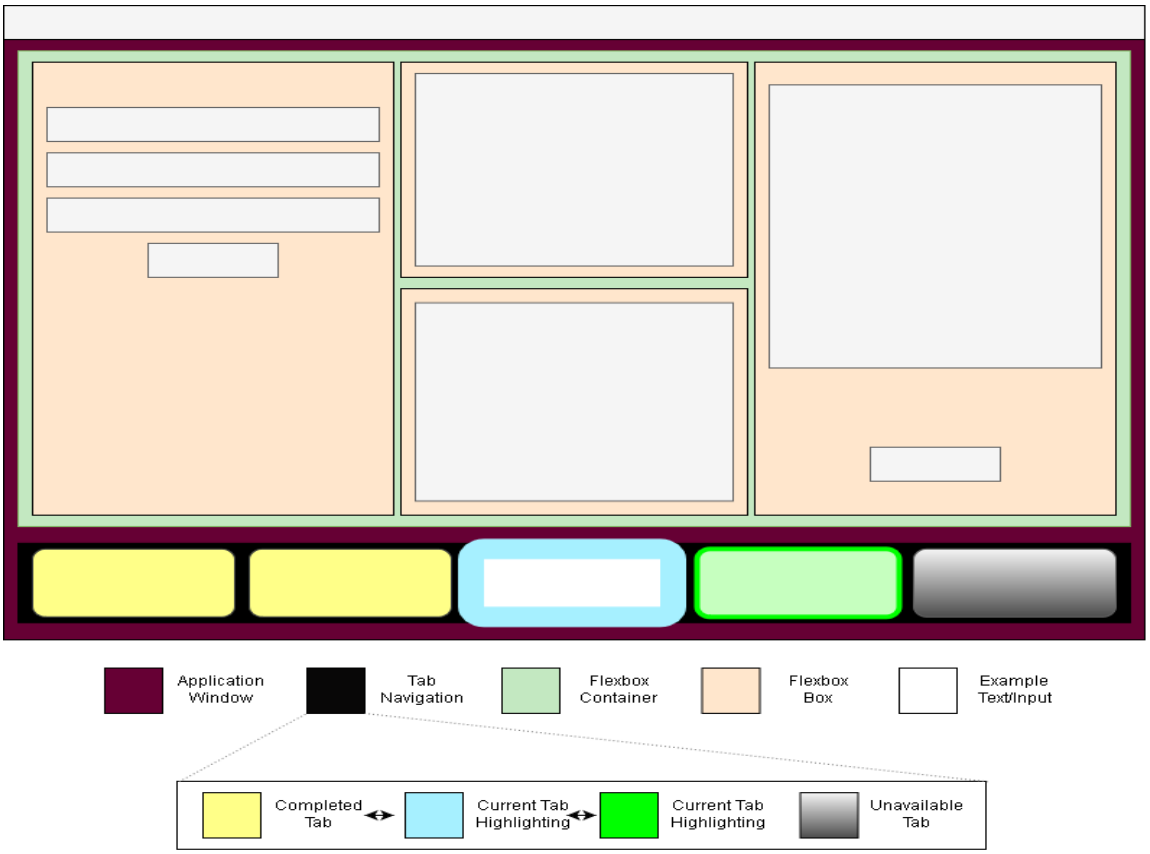


Figure 4: An example layout of our GUI that demonstrates how all of our technologies fit together in the final product.

5. Conclusion

In this project our aim is to create a reliable, easy to use user interface. This user interface will abstract the command line program provided to use by the Fofanov Bioinformatics lab. Abstraction will allow for biologists who already know the command line to make fewer mistakes, as well as allow newer biologists to learn the tool more easily. In order to make this user interface possible, we plant to use Electron, a JavaScript based platform.

The purpose of this document is to outline what challenges we plan to face during our implementation, as well as to review our planned solutions. It discusses all of our potential technology choices and ultimately demonstrates why we decided on Electron as our tool of choice. Below is a list that clearly states how confident we our that our solution to addresses each foreseen challenge and feature:

- **Cross Platform**: We are 100% confident in Electron's cross-platform capability.
- **Pipeline Traversal**: We are about 90% confident in Election's pipeline traversal capability. While we expect Electron to be able to actually traverse the pipeline easily, in certain instances large data sets may require creative solutions to process at an acceptable speed. This is due to Electron's JavaScript base, which is known for slower processing times.
- **Error Correction**: We are 90% confident in Electron's ability to detect user-input error and provide appropriate feedback. The only potential issue here would be processing times from large user-input data sets. However, Electron provides some basic multi-threaded capability, so this likely will not pose a problem.
- **Maintainable and Expandable**: We are 100% confident in Electron's ability to be maintained and expanded upon in the future.
- **Visually Pleasing**: We are 100% confident that we can create a visually pleasing GUI and graphical outputs using Electron and supporting JavaScript tools.

By creating a beautiful, easy to use product, we hope to innovate how the field of biology sees genomic identification. By making our product accessible, we can not only enhance the experience of those currently in the field, but we will be able to draw fresh talent that would otherwise be dissuaded by the high technical requirement. With this project, we hope to remove the tedium that is a distraction to the field of comparative genomics.