

NYU Langone Health

Web-based Visualization of Electrode Data for Epilepsy Patients

- Jay Burkhardt, `Jeremy.Burkhardt001@umb.edu`
- Ana Gorohovschi, `Ana.Gorohovschi001@umb.edu`
- Aaryaman Sharma, `Aaryaman.Sharma001@umb.edu`
- Jingyun "Josh" Chen, `jingyun.chen@nyulangone.org`

Final Documentation - 5/20/2022

1 Introduction

1.1 Purpose

N-Tools Browser is an open-source web-based tool created for the purpose of aiding medical professionals in researching epilepsy. A major challenge for medical software has been lack of portability. The software often requires local downloading and licensure to use, and can sometimes require very specific technological knowledge which a clinician may lack. The goal for this software is a web-based visualization tool that allows for portability, requiring no downloads or licenses. It also aims to create a unified BIDS compliant data format expressed in JSON so that clinicians from other institutions can easily share and upload data.

1.2 Scope

N-Tools Browser is a web-based, open-source platform that provides both three-dimensional (3D) and two-dimensional (2D) visualization of electrode data. The tool will provide clinical workers with an easy and intuitive interface for examining and managing large data sets. It will accomplish this by color-coding relevant information, such as seizure type and electrode signal display, as well as allowing the user to easily switch between different seizure types, electrodes, and functional mappings. Since it is a browser tool, N-Tools is very portable being limited only by an internet connection. No software needs to be downloaded in order for it to be used. N-Tools Browser is currently classified as research software and is not yet intended for clinical use.

1.3 Overview

1.3.1 Existing Work

In addition to N-Tools Browser, several other web-based medical imaging tools are currently available.

- The X Toolkit: WebGL for Scientific Visualization, <https://github.com/xtk/X>
XTK is one of the most complete WebGL libraries available today for medical imaging.
- AMI Medical Imaging (AMI) JS ToolKit, <https://github.com/FNNDSC/ami>
AMI is a medical imaging library built with THREE.js, but the code has not been updated for some time.
- BrainBrowser, <https://brainbrowser.cbrain.mcgill.ca/>
BrainBrowser is another medical imaging library built with THREE.js, but like AMI, updates have ceased.

1.3.2 Limitations

- In the original document we noted that the label map pre-processing was a major limitation for N-Tools Browser. We have made this process completely obsolete, and therefore it is no longer a limitation. Details are discussed in section 4.4
- The label map generation is also limited in that it works by mapping electrodes, which exist on a continuum of values, onto a discrete value. This can occasionally result in the 2D slice not being as accurate as it could be. For a prototype, this works well enough, but it is completely unacceptable if N-Tools ever hopes to be used in real clinical procedures. Machine learning could potentially solve this in a future update, with heavy FDA design control.
- N-Tools currently only works within NYU's firewall.

1.4 Definitions

- **HTML** - Hypertext Markup Language. Used to specify the layout of a web page.
- **CSS** - Cascading Style Sheets. Used to describe the visual components of a web page.
- **JavaScript** - A high-level, multi-paradigm, interpreted language used to execute code on a web page.
- **DOM** - Document Object Model. A tree consisting of objects that serves as the primary data-structure of the modern web.

- **ECMAScript** - JavaScript language specification. Often abbreviated as simply "ES."
- **JSON** - JavaScript Object Notation. A data format consisting of key-value pairs.
- **API** - Application Programming Interface. An interface which specifies how two computers communicate.
- **WebGL** - Web Graphics Library. A JavaScript API for rendering graphics, both two-dimensional and three-dimensional, on a webpage.
- **NIFTI** - A common file format for neuroimaging. Named after the group that developed it: The Neuroimaging Informatics Technology Initiative. This document will often refer to NIFTI files as .nii, which is their file extension, or simply as "volume(s)".
- **Pial** - .pial file extensions are meshes containing brain surface data. This document will often refer to .pial files as "mesh(es)".
- **BIDS** - Brain Imaging Data Structure. A standard way of organizing and sharing neuroimaging data.
- **XTK** - The X-Toolkit. A WebGL framework designed specifically for scientific visualization.
- **Epilepsy** - A classification of brain disorders which cause seizures.
- **Electrode** - A sensor used to record electrical activity in the brain in order to discern where a seizure originates.

2 Requirements

- **1. Electrode attribute editing.** The user will have the ability to edit certain attributes of a given electrode.
- **2. Functional map creation.** The user will have the ability to create connections between electrodes with a caption and threshold value.
- **3. Electrode signal display.** The electrodes will change colors in accordance with the electrode signal. This way the user will see a "heat map" of a Seizure via the color change in the electrodes. The color intensity indicates the magnitude of the voltage. Positive voltages are red and negative voltages are blue.

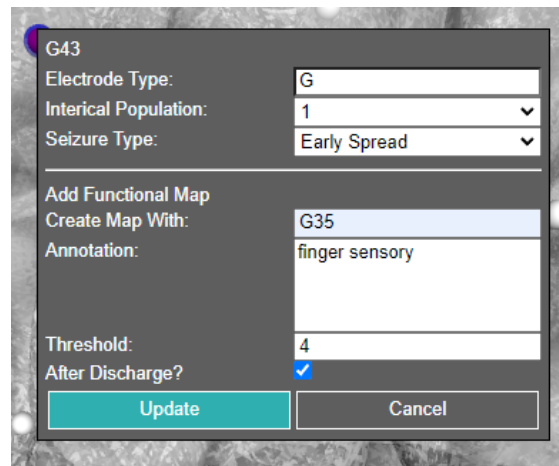
The "play / stop" button will change the color of the electrodes over time to display the signal as a color map.

See figure 2 of a possible snap shot of electrode colors.

A window where the electrode signal can be displayed and examined by the user. Given that the electrode signals can be quite large, there is a need to scroll the signal rather than view it all at once. Also, displaying all signals at the same time, even a partial signal can occupy a lot of screen space. The new ability to display the electrode signal in a window will display it as one signal at the time. While scrolling, the 3D view will change the electrodes color to display all signal values at once at the scrolled to time stamp. Future development should create a relationship between the 'sin wave' window and the "play / stop" button, to indicate with a moving vertical bar the current time stamp for the colored electrodes. [Code for the signal display window can be found here.](#)

- **4. BIDS JSON format.** The user will have the ability to generate a JSON file representing the neuroimaging data according to the BIDS standard. This is to allow portability and easy sharing for future users. Data downloaded after editing will also be in this JSON format.

Figure 1: The edit menu that appears when the user right clicks an electrode. The edit menu can be moved as needed.



The screenshot shows a dark-themed dialog box for editing electrode G43. It contains several input fields and dropdown menus. The 'Electrode Type' is set to 'G'. 'Interical Population' is set to '1'. 'Seizure Type' is set to 'Early Spread'. Under the 'Add Functional Map' section, 'Create Map With' is set to 'G35' and 'Annotation' is 'finger sensory'. The 'Threshold' is set to '4' and 'After Discharge?' is checked. At the bottom are 'Update' and 'Cancel' buttons.

G43	
Electrode Type:	G
Interical Population:	1
Seizure Type:	Early Spread
Add Functional Map	
Create Map With:	G35
Annotation:	finger sensory
Threshold:	4
After Discharge?	<input checked="" type="checkbox"/>
Update	Cancel

Figure 2: Electrode signal via electrode color. The "Play/Stop" button is added to the GUI as shown in figure 2

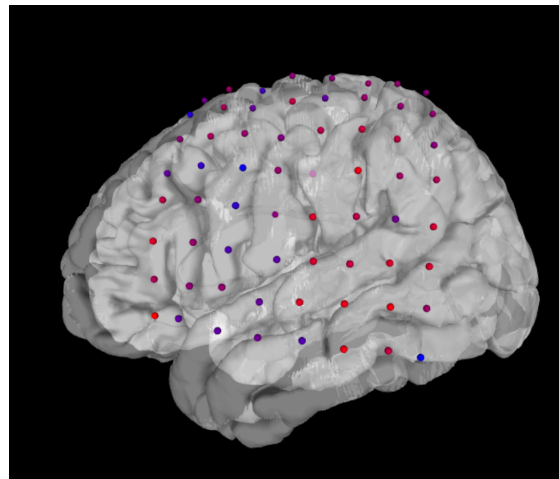


Figure 3: Start/Stop signal play. This button will start/stop the coloring of electrodes based on the electrode signal. This electrode signal will change over time. The interval between the change will closely match the time interval when the signal was read.

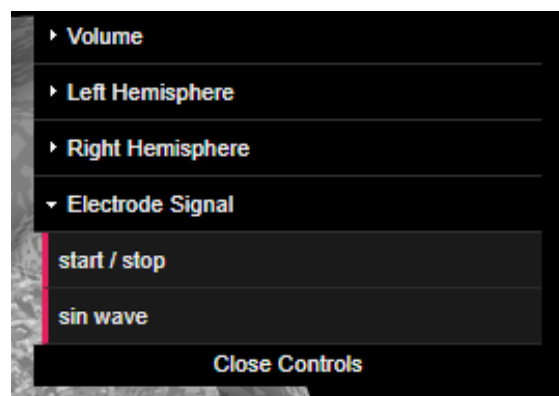


Figure 4: sin wave. This button will open a window with the first electrode signal displayed as a sin wave. In this window the user can scroll the signal wave. As the scrolling occurs the electrodes in the 3D view will change colors just like the 'Start/Stop' button. The current time stamp for signal values displayed as colors is at the left edge of the sin wave window. To display the next electrode's sin wave, the user can press the Arrow Down key to display the next signal wave. Press down arrow to move to the next signal. Arrow up displays the previous signal.

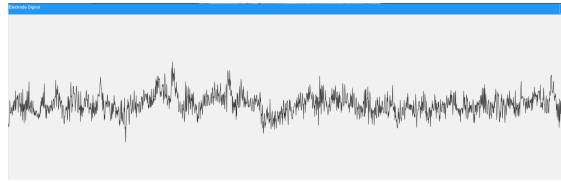


Figure 5: Two examples of how the JSON format should look like to be BIDS compliant. The image on the right is taken directly from the documentation for creating BIDS JSON files. See: <https://bids-specification.readthedocs.io/en/stable/03-modality-agnostic-files.html>

```
...
{
  "elecID" : "G01",
  "coordinates": {
    "x": -34.487446,
    "y": -21.274218,
    "z": 48.651531,
  },
  "ElecType" : "G",
  "SeizType" : ""
},
...
```

```
{
  "Name": "The mother of all experiments",
  "BIDSVersion": "1.6.0",
  "DatasetType": "raw",
  "License": "CC0",
  "Authors": [
    "Paul Broca",
    "Carl Wernicke"
  ],
}
```

3 Specifications

N-Tools Browser will be a frontend browser-based visualization tool. No software will need to be downloaded in order to run this tool, however certain data-sets are limited to use from within NYU's firewall. Frontend in this context means all of the rendering and visualization happen locally on the users computer. Although this application will fetch already existing pre-processed data on a server, the software will only passively receive the data and display it and not create any of its own. The one exception to this is if a user decides to edit the data, in which case they will be allowed to download their changes as a JSON file. Before rendering is complete, N-Tools will use a fetched JSON file to obtain all of the information it needs to populate the drop-down menus on the user interface. It will also read the properties of each electrode to learn its location and seizure type. Once all of this information has been gathered, the electrodes, NifTI volume, and brain surface meshes will be rendered to the screen. 2D renderers containing the NifTI volume image will also be rendered to the bottom of the screen. At this point, the user will then be able to interact with both the 3D and 2D rendered image. They will be able to click on electrodes, change colors of the brain surface, hide various rendered components from view, update the current 2D subsection of the NifTI image, and cycle between various functional mappings between electrodes. Once the user is satisfied, they can simply close the browser to exit. If they wish to save their edits, they have the option of downloading the new data to their computer for later use.

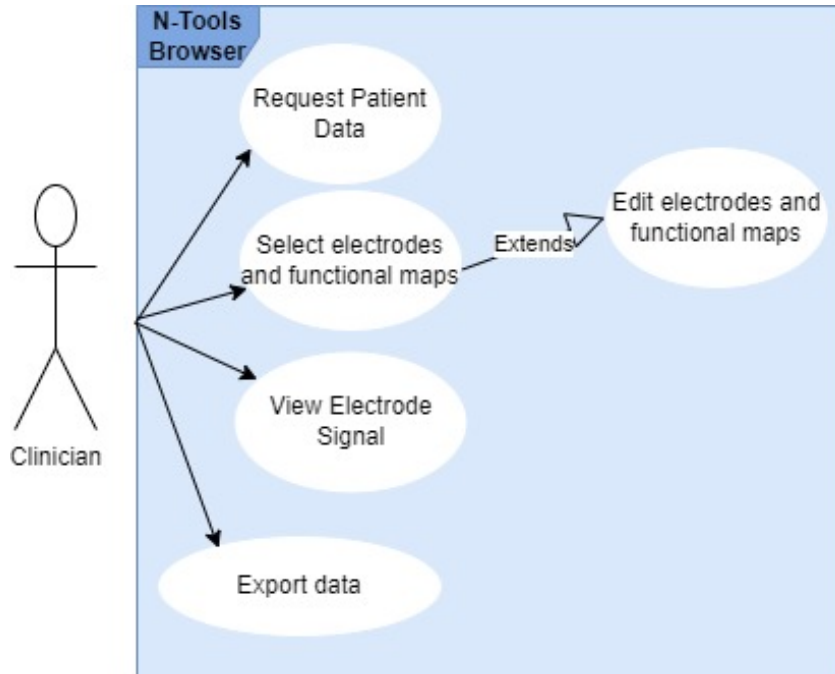
4 Design

4.1 Use Cases

The users in mind for this software are clinicians, particularly neurologists, engaged in epilepsy research at NYU Grossman School of Medicine. These users will be able to request patient data and interact with it in both 2D and 3D.

They can then save and download their edits, if applicable. The various actions a clinician can take are shown as a use case diagram in Figure 6. This tool is NOT intended for individuals with no medical training to gain insights into their own medical conditions.

Figure 6: Clinician Use Case. Once data is loaded, a clinician can select, edit, and download it as needed. The tools for editing electrodes and functional maps are an extension of the tools for selecting them. All of the user interface tools for accomplishing these tasks are in plain view. A preview of the user interface can be found in the images in figure 8, providing a visualization for how these tasks are accomplished



4.2 Architecture

Our current edition of N-Tools-Browser most closely resembles the Model-View-Controller architecture. Our central data-structure for the model component is an array of JavaScript objects. Each of these objects represents a single electrode within the data set. However, unlike a traditional object in an object-oriented language, these objects do not contain any methods. They are instead passed around to different functions within modules that report their state to other parts of the application. The controller is a separate module that is responsible for updating the state of the application, where appropriate. The model also maps the electrode objects to XTK spheres, which are then used by the view to display the data.

In addition to MVC as a base architecture, we also keep the following design principles in mind.

- A preference for immutable data. In JavaScript, this means making heavy use of the "const" keyword where appropriate. By structuring data this way, it makes it much more difficult to accidentally modify and break other parts of the program while it is running. It also makes adding additional features much easier, since there is less of a worry that a new feature may modify something accidentally. We state this as a *preference*, because sometimes it is unavoidable. If state must be changed, it is preferable to make a copy, modify the copy, and return the copy. Reference types can have the object they are pointing to changed, but cannot point to a new object.
- JavaScript closures to maintain state. Closures enable a function to "capture" its enclosing environment. This gives the function access to data that might be inaccessible after the enclosing function has returned. The most common example of this is an event listener that has access to data during its creation. Long after the function that created the listener returns, the event can still access the data when needed.
- Although we still feel it is improper to think of N-Tools' architecture as object oriented, we did end up finding it very helpful to have classes in the end. Our new 2D slice renderer was very appropriately written as a class, since there were several state variables within each instance that we wanted to keep separate.

- Our program still uses one central data structure, which is the JSON data. Within this data structure are arrays for each electrode. In this sense, the data is composed of many smaller objects, rather than a parent/child class relationship. The electrode canvas 2D renderers, as mentioned before, are mainly used for rendering this data, and thus could be more properly be considered part of the view, rather than the model. We do not think that this is at odds with the overall architecture and code philosophy we have had from the beginning.
- To keep our work organized and update the view more efficiently, we have also introduced a few wrappers that can be seen as a conceptual analogue of static classes. The only purpose here was to simplify the code and make it easier to change. The wrappers are as follows:
 - **DOM:** This places the majority of the DOM elements into a single object. When we want to query the DOM, we can do it through this object rather than needing to type the full query selector. This also means if we change a class name or ID, we only have to change it in one place. The code for this DOM object can be found here.
 - **GFX:** We placed all of the 3D object rendering for XTK into a single JavaScript object. Our justification for doing this is that once again, this reduces the amount of code written per file. All of the handling for generating electrodes, functional map connections, and highlights can be generated under a single GFX call. The code for this GFX object can be found here.
 - **COLOR:** Since XTK uses an array for colors, we thought it would make the code a bit easier to understand if we had a COLOR object with keys for color names and the corresponding arrays as values. This simply makes the code more readable. Canvas uses hex values, however, so this was not as modular as we had hoped. The code for handling colors can be found here.

Figure 7: Architecture Overview. The JSON data is fetched from the server and stored in a variable. Originally, we were transforming the data into the electrode objects under "model". Now, they are part of the data to begin with. We have still kept them in the diagram because the end result is exactly the same; there is just one less transformation step that happens in between. The controller contains no objects; it only contains functions for acting upon the objects. A function for selecting an electrode, for example, tells the model that a certain electrode is chose, which in turn tells the view to highlight it.

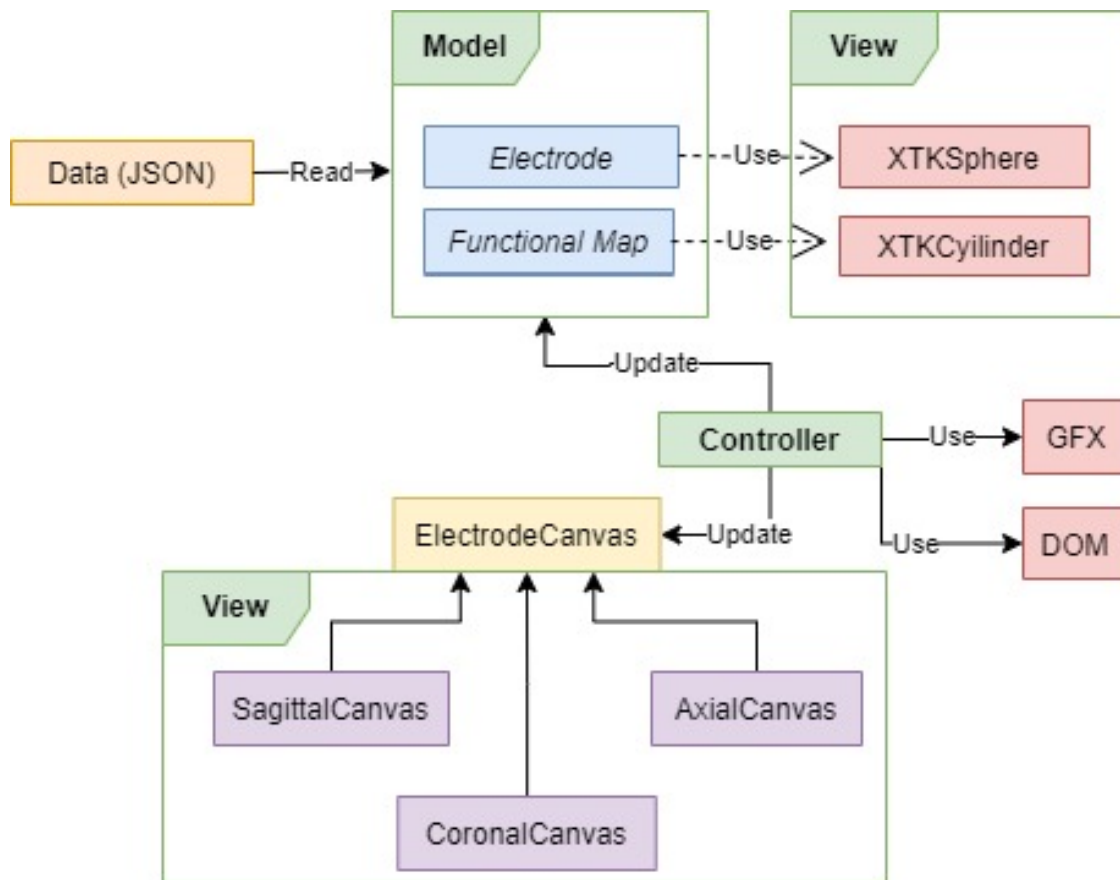


Figure 8: The architecture in more detail, showing most of the data types and function signatures of the various components.

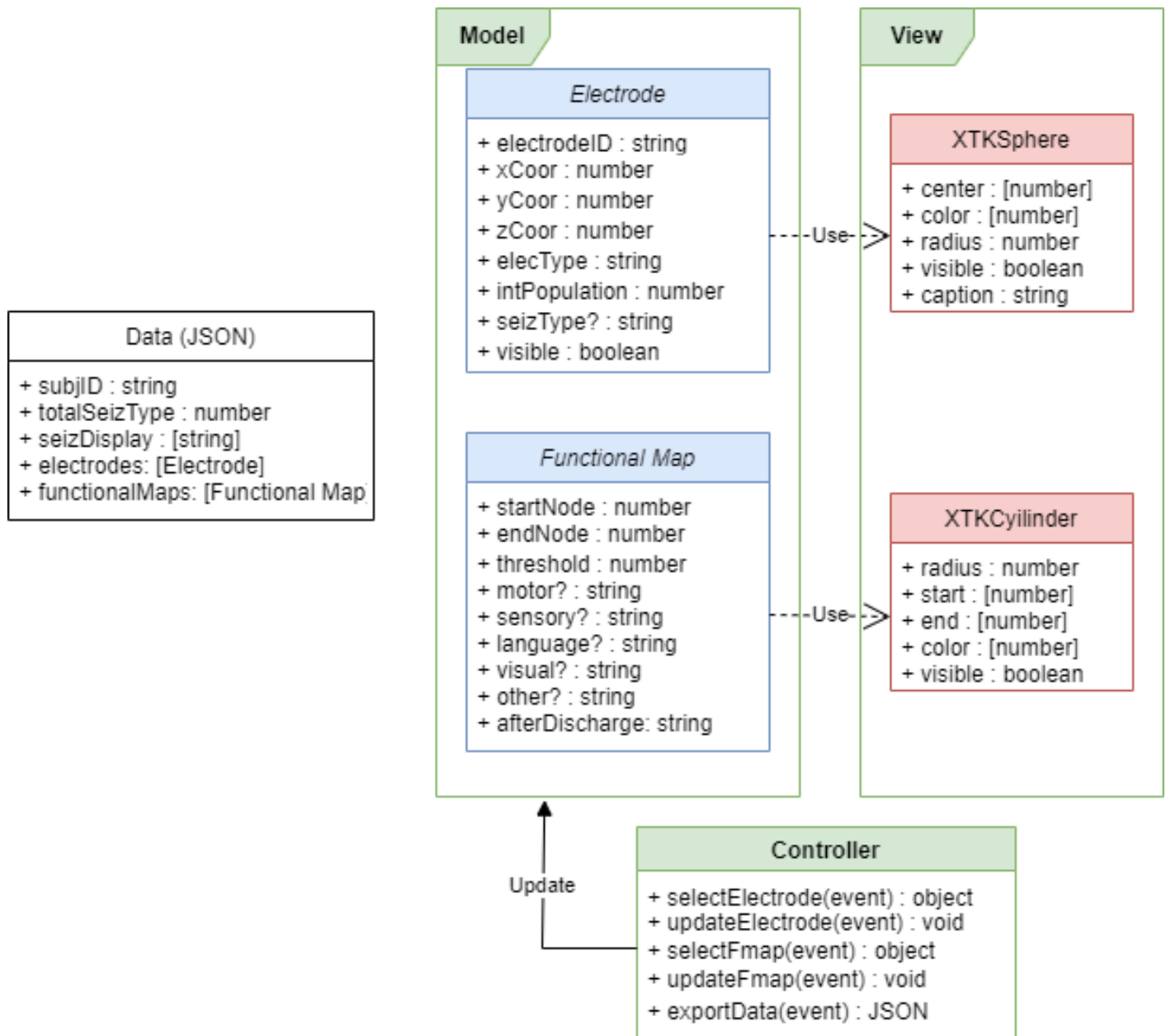


Figure 9: The electrode canvas class hierarchy. The three sub classes are each instantiated only once, and have different mathematical functions for generating the volume image and 2D electrode coordinates. The controller is also responsible for updating these components by calling their 'drawCanvas' methods.

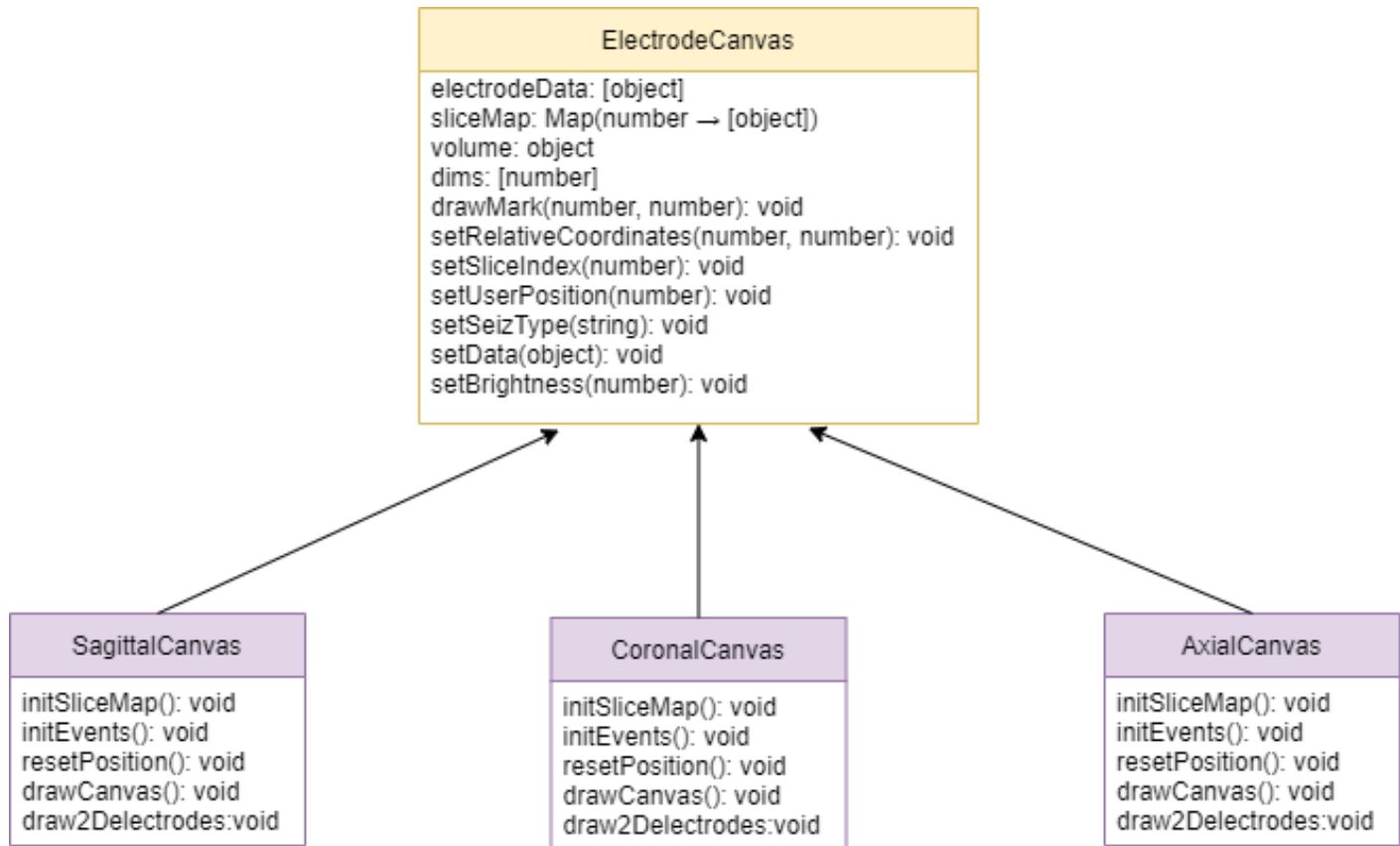
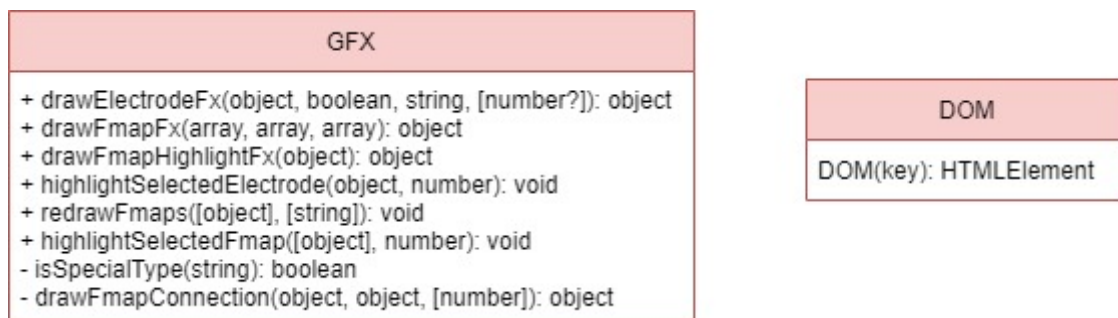


Figure 10: The GFX and DOM objects. It is most helpful to think of GFX as a static class. It is simply an object with a collection of functions. DOM is simply a JavaScript object that will return the DOM element according to a key, without needing to constantly use a query selector.



4.3 Workflows

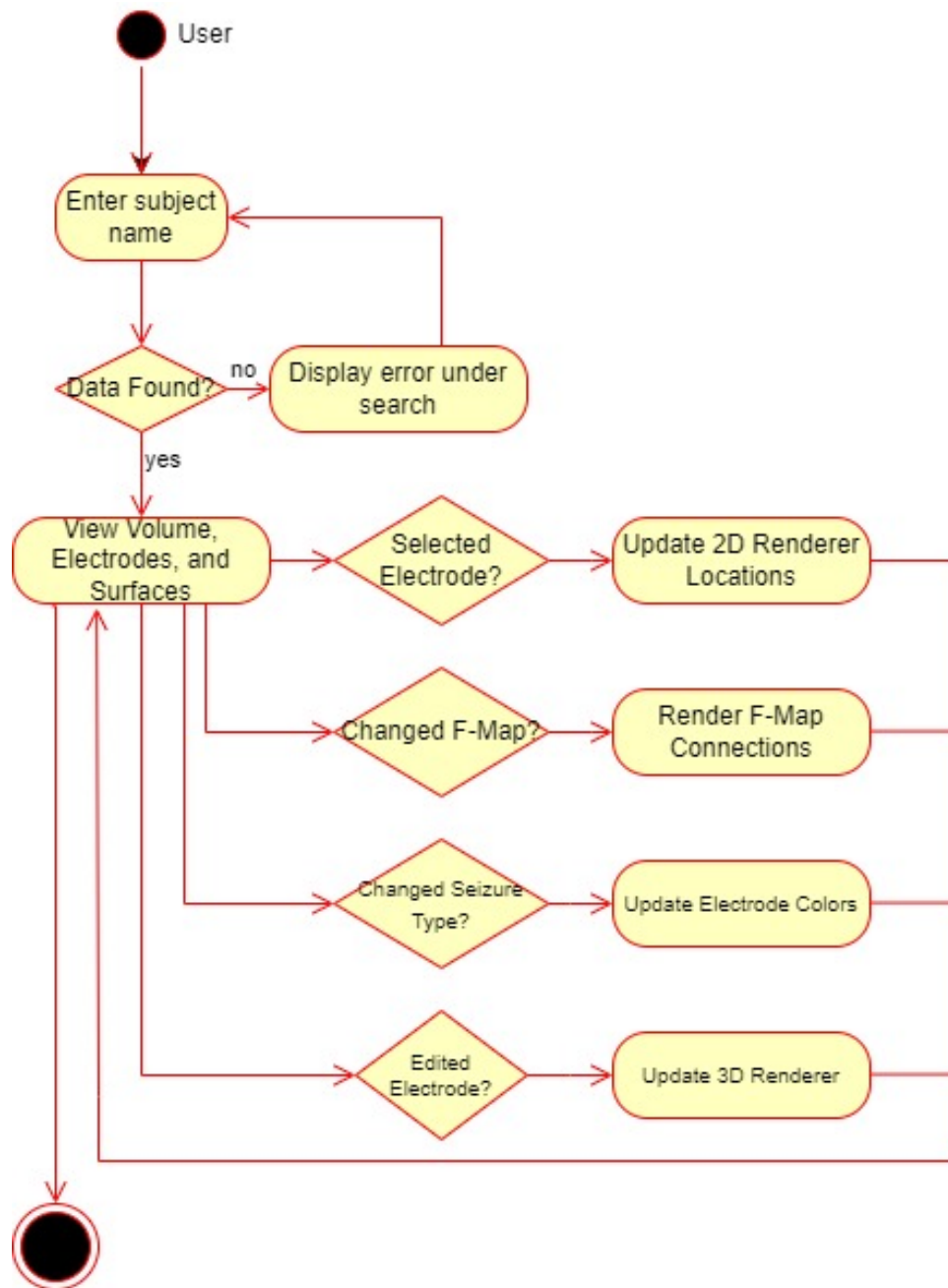
A clinician desiring to N-Tools to examine data for an epileptic client begins by typing the subjects name into a search box¹. If the lookup is successful, the view will then jump to the main screen with the brain mesh and electrodes rendered in the center in 3D. The 3D rendered slices are invisible by default. 2D slices of the volume are also rendered at the bottom of the screen, but can be moved to any location by dragging the mouse. From here, the clinician can click electrodes or select them from a drop down menu, which updates the view with all of the selected electrode's attributes. They can also select different functional mappings between electrodes, cycle between seizure types, move between 2D

¹N-Tools comes in two modes: demo and build. These are just internal signifiers for the model. Build is directly connected to NYU's server, and demo, as the name suggests, contains sample datasets.

slices, or edit an electrodes attribute. No information is hidden from the user at anytime, unless they explicitly choose to hide it. After one of the above tasks is complete, the model simply waits for the next action by the user. At any time, the clinician can return to the search screen to load a new subject, or they can download their new modified files. The application can be quit at anytime by simply closing the browser window. Figure 11 shows an activity diagram of these processes.

Feedback from the clinical staff informed us that N-Tools Browser will be primarily used for adding electrode data to a new patient, rather than editing already existing data. Therefore, the clinician will have the ability to upload 'blank' data sets that only contain electrode coordinates. They can then use the data from the electrode signal display feature to observe a seizure propagating, and update electrodes and functional maps accordingly. A subject called 'blank' is available in the samples on github pages. Keep in mind that as with all samples, blank uses either randomly generated electrode coordinates or mismatched volumes.

Figure 11: Example of user workflow. This activity diagram details some of the steps that a clinician may take when working with the application. The arrows represent the different functions that the controller contains, and how it dispatches the information to the model and the view.



4.4 Technologies and Implementation Details

The primary technologies we will be using for frontend design are HTML, CSS, and JavaScript. These three tools form the foundation for all modern browser applications. While HTML and CSS are used primarily for structuring and styling, JavaScript allows developers to make web pages fully interactive and its ability to dynamically alter the DOM. Without JavaScript, web pages would be static static displays of text and images with limited dynamic capabilities. Therefore, the use of JavaScript in this application to create a fully interactive user experience is absolutely essential. Although many modern frameworks and libraries exist for writing JavaScript code exist today, such as React, Vue, and Angular, this project will not take advantage of these. JavaScript without a framework is often referred to as "Vanilla JS."

The API we will be using for rendering 2D and 3D graphics is WebGL. WebGL allows developers to create beautiful interactive graphic visualizations. However, programming in WebGL without a framework can be extremely tedious and time consuming. Therefore, we will be using the X-Toolkit, abbreviated as XTK. XTK makes many of the complicated features of WebGL, such as geometry rendering, 2D picking, and ray casting, much easier to work with. While WebGL programs are often imperative, meaning they require many explicit instructions "how-to" instructions for the computer, XTK is declarative, meaning the developer can simply describe the result without needing to be concerned about the finer details. XTK comes with out-of-the-box support for medical imaging and is capable of rendering pial meshes and NIFTI volumes.

The primary data format we will be working with is JSON. A JSON file consists of large collections of key-value pairs. JSON files also naturally integrate with JavaScript, as they are written in the same object syntax as JavaScript. The main difference between JSON and a regular JavaScript object is that JSON keys must be strings. Values, however, can be strings, numbers, booleans, lists, or additional string-keys and values. Arrays of specific properties can easily be selected with a map function. The specific format of the JSON file will be compliant with the BIDS standard for describing neuroimaging data.

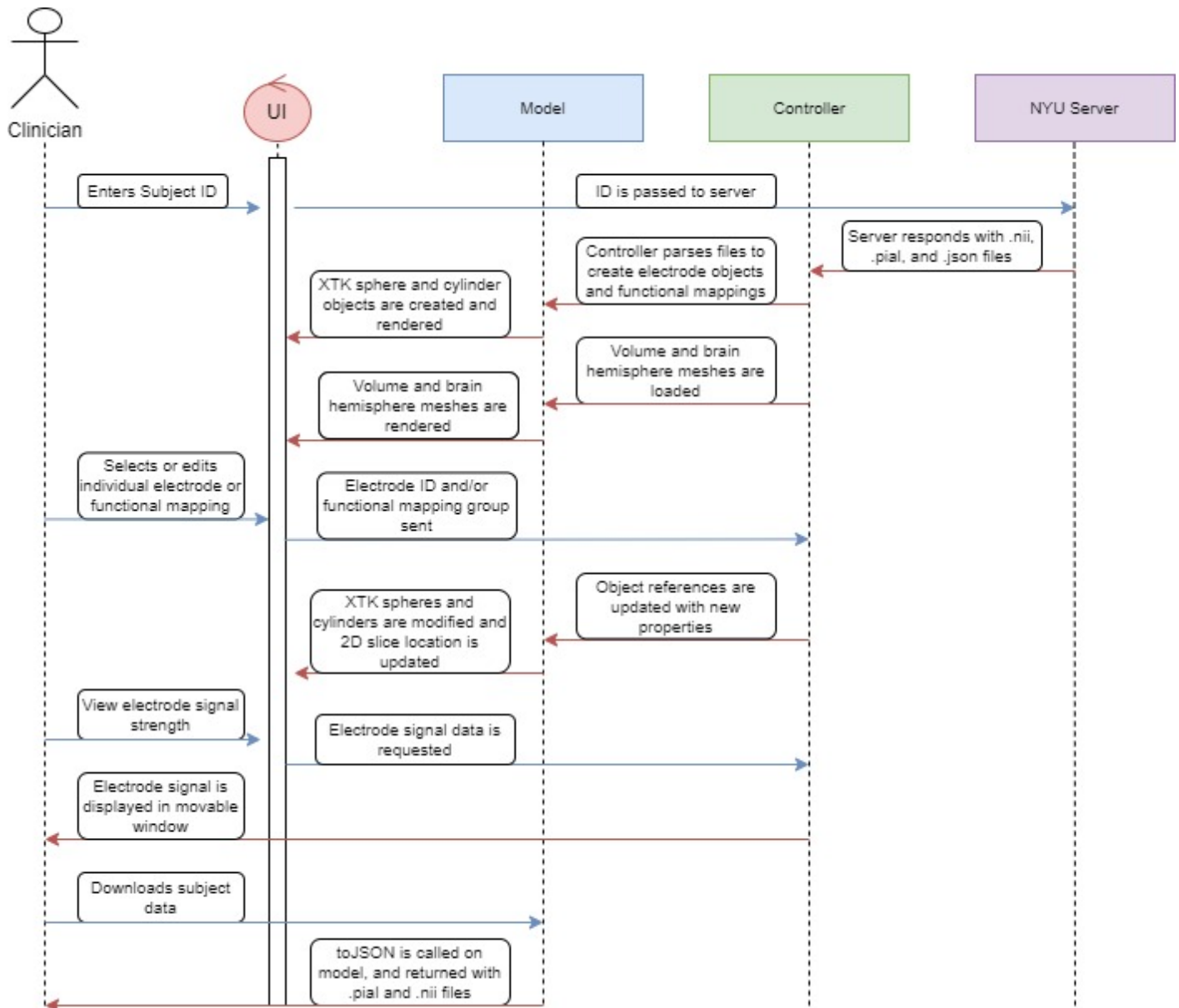
- **subjID** - The subjects identification number.
- **totalSeizType** - The number of seizure types of a patient.
- **seizDisplay** - The names of the seizure type collections.
- **electrodes** - An array of electrode objects containing the following properties:
 - **elecID** - A unique ID for each electrode.
 - **coordinates** - An object representing the (X, Y, Z) coordinates of an electrode.
 - **elecType** - Electrode type (e.g. G, D, S, etc.).
 - **intPopulation** - Interictal Population.
 - **seizureType** - Electrode properties with respect to *specific seizure type* (e.g. onset, early-spread, late-spread). A subject can have multiple seizure type mappings, labeled Seizure Type 1 to Seizure Type N.
- **functionalMaps** An array of functional map objects which contain the following properties:
 - **fmapG1, fmapG2** - Objects containing the following properties:
 - * **index** - The index of the electrode in the connection in the electrodes property.
 - * **elecID** - The ID of the electrode in the connection.
 - **fmapMotor, Sensory, Language, Visual, Other** - Captions describing the affected regions between electrodes involved in the functional mapping.

A complete example of JSON data in this format can be found [here](#).

To display the electrode signals, N-Tools Browser will allow users to use a "play/stop" button to watch a seizure spread throughout the electrodes. A color map will be determined for the exact mapping of the signal. For example, a negative voltage could correspond to blue, and a positive voltage could correspond to red. As the seizure spreads, the electrodes will change color to represent the current stage of the seizure. In addition to the play and stop button, we will also add functionality with a slider to "step through" the various points in time of the spread.

Our original project generated label maps as a pre-processing step. While this could successfully render electrodes in 2D, it would have to be re-run every time a user made any edit whatsoever. We instead decided to keep XTK's NIfTI parser, but we now generate the image directly onto the canvas image data. Each electrode canvas has a map, with an integer for a key, representing the current image, and an array of electrodes mapped to that slice as the value. Each electrode canvas has a reference to the volume and a copy of the electrode data. To use the image data directly, we adapted the code found [here](#). This means that the electrode canvas is mainly part of the view, but contains references to the model so that it can be updated by the controller. The electrode canvas also uses a renderer described in [this article](#). This adds the mouse events to the canvas so that it can be zoomed in on and moved around the screen. The code for the electrode canvases can be found [here](#).

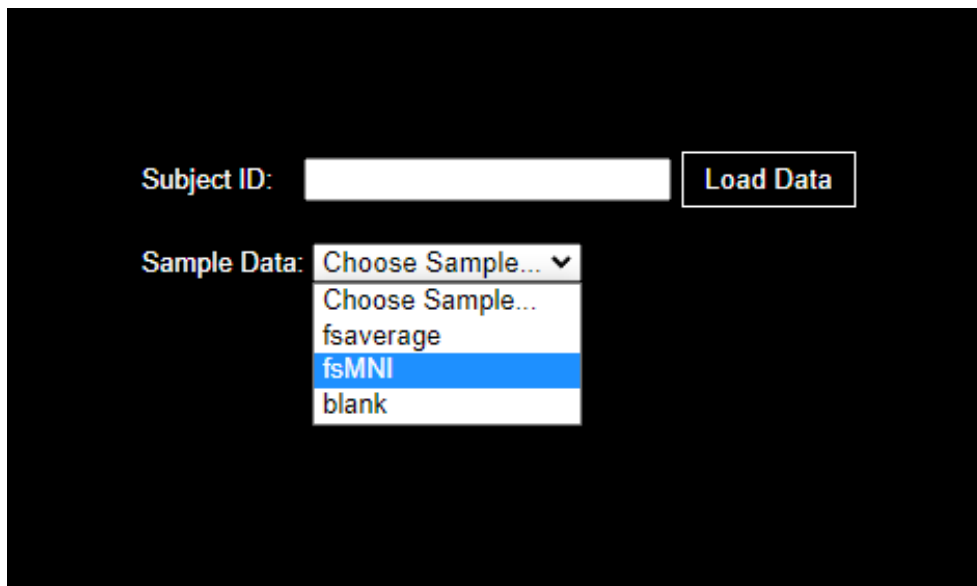
Figure 12: Architecture and user workflow. This sequence diagram combines all of the functionality described above, showing how a clinicians interactions with the UI will cause the application to respond.



4.5 User Interface

The user interface begins with a page prompting the user for a subject ID. Those demoing the project can select a sample dataset from the 'Sample Data' menu (FIGURE 11). Users who wish to use N-Tools for their own data will either need to upload their data into their projects local directory, or point the URL to a different server. Upon entering a subject ID, the window will then jump to the view seen in (FIGURE 12). Here the user has access to all the data in the dataset, and can select different electrodes by either clicking them on the view or selecting them from the 'Select Electrode' option menu. The three electrode canvases are shown together in (FIGURE 13). The user can move these images by dragging with the mouse, cycle through images with the scroll wheel, enlarge images by holding ctrl and using the scroll wheel, and moving them back to their original containers by double clicking. (FIGURE 14) shows the controls available for adjusting the brightness, contrast, and positions of the slices. For ease of use, the user can select the checkbox titled "Display All Tags." This will render 2D electrode ID tags over the electrodes regardless of the screen orientation (FIGURE 15). (FIGURE 16) depicts a possible workflow where the user is examining the axial slice and 3D brain image next to each-other. Finally, (FIGURE 17) shows the process of a user adding a new functional map to the data.

Figure 13: Loading screen with available samples for demoing. The ability to type and enter is currently limited to NYU's server.



The image shows a loading screen with a black background. It features two main input sections. The first section is labeled "Subject ID:" in white text, followed by a white rectangular input field and a blue button with the text "Load Data" in white. The second section is labeled "Sample Data:" in white text, followed by a dropdown menu. The dropdown menu is open, showing a list of options: "Choose Sample..." (with a downward arrow), "Choose Sample...", "fsaverage", "fsMNI" (which is highlighted in blue), and "blank".

Figure 14: Main UI

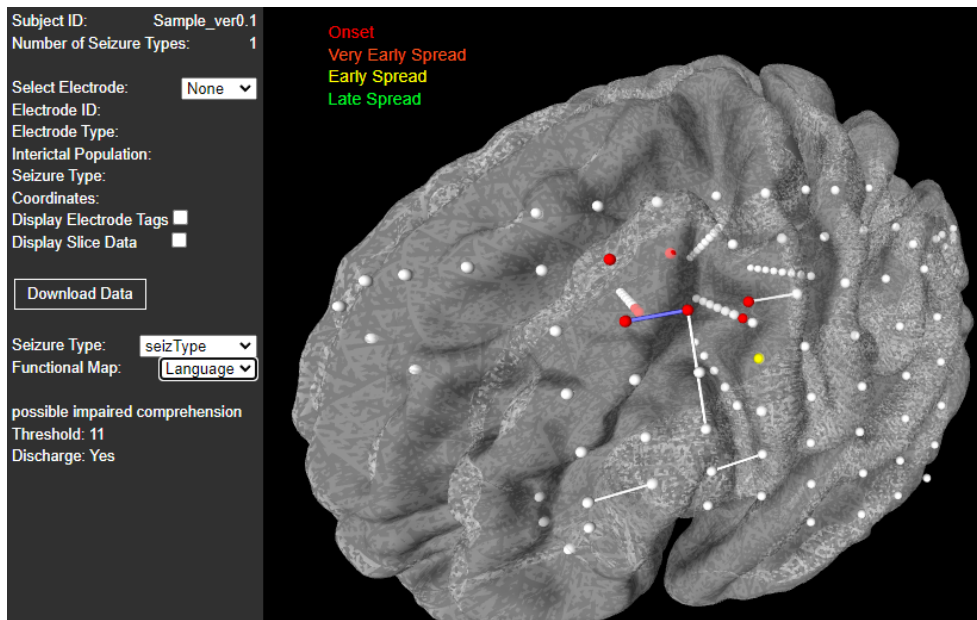


Figure 15: The three electrode canvas 2D renderers for sagittal, coronal, and axial cross-sections on the test subject fsaverage.

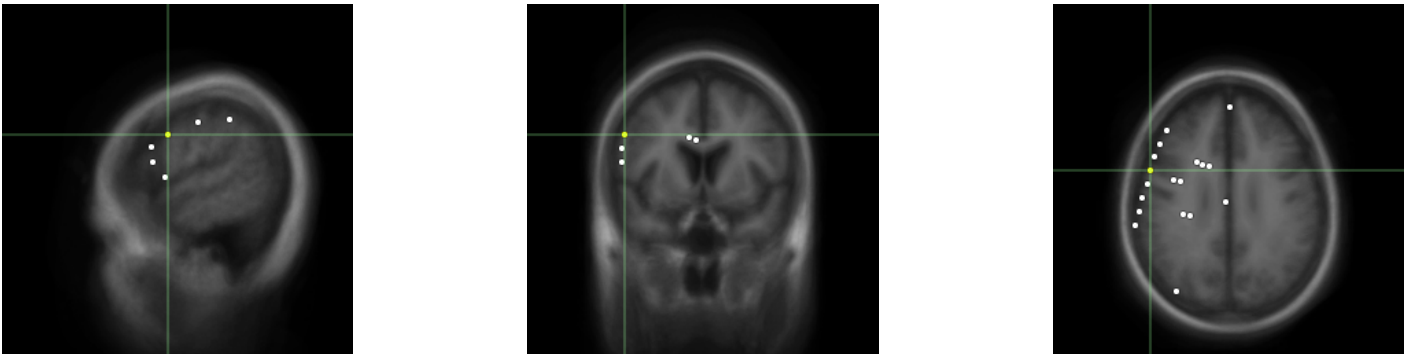


Figure 16: Adjusting Brightness and Contrast

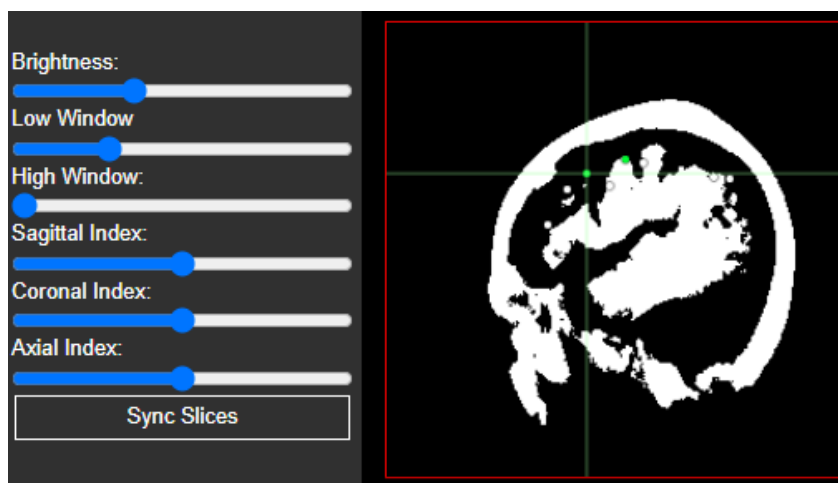


Figure 17: View after the 'display all tags' button is checked. The tags will remain in place as the camera is moved so users can easily find electrode IDs when making edits.

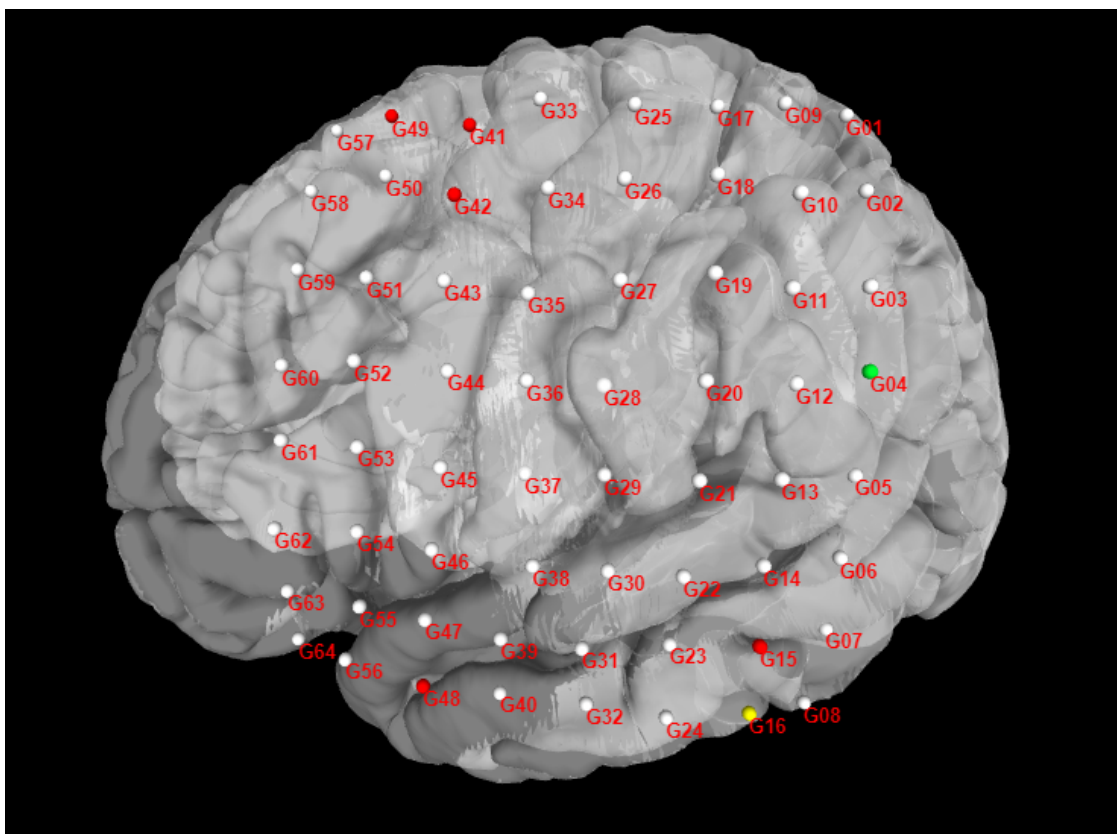


Figure 18: Viewing the 3D brain surface alongside the zoomed-in axial slice.

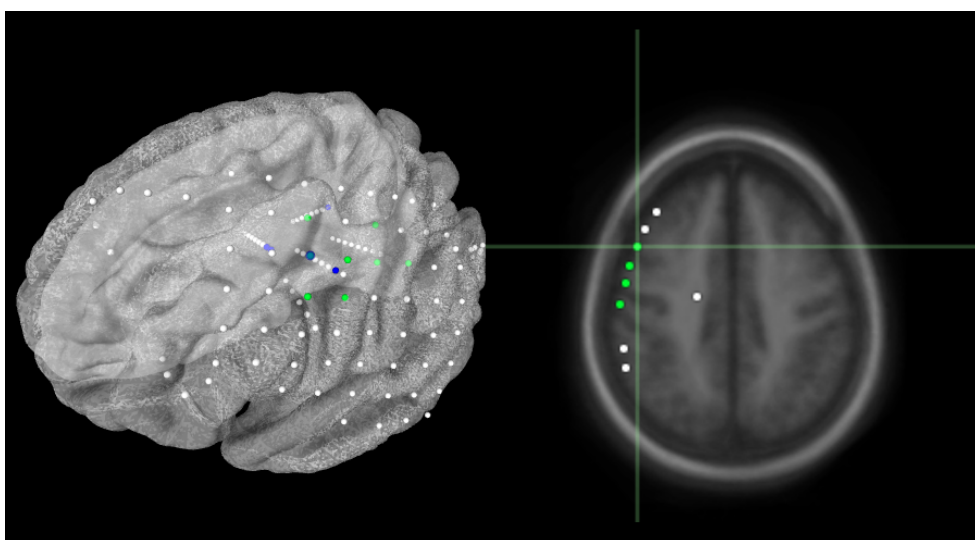
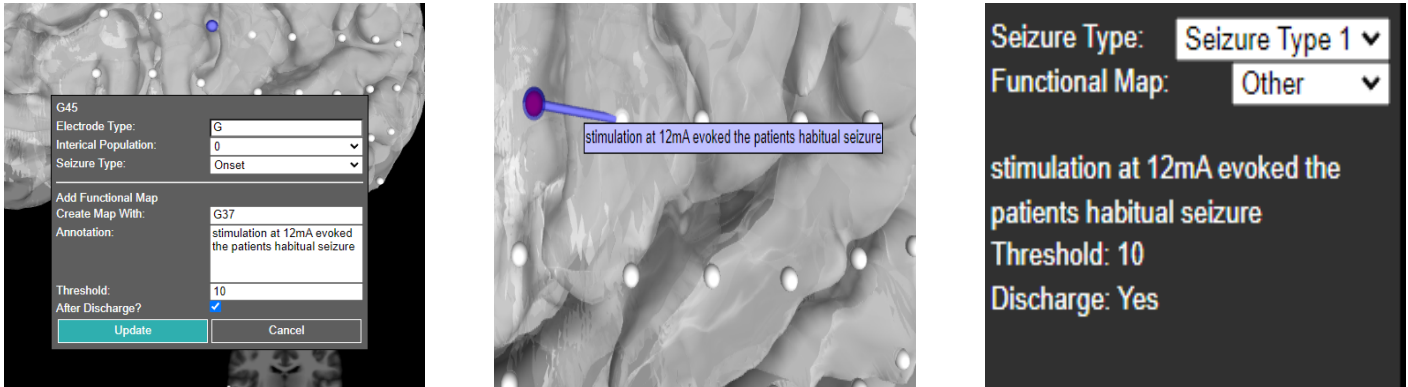


Figure 19: Workflow for creating a new functional map. First the user enters the target electrode, the caption, threshold value, and discharge. After pressing update, the functional map can be selected by hovering and clicking the mouse. The functional mapping will persist between changes in category.



5 Deployment and Testing

We finished our initial phase of testing and had our code deployed to NYU's server on May 3rd. This initial phase of testing made us realize two things:

1. We did not have enough guard clauses within the JS code. A guard clause should make a function exit prematurely and give a meaningful error message when bad input is passed rather than completely crashing the program.
2. Related to the point above, our error messages were pretty ambiguous. So we had to work to make them more understandable. Rather than just crashing, we needed to provide the user with a message about what exactly went wrong.

The rest of our testing was in the realm of preprocessing. We were provided with several .xlsx and .txt files which NYU used to create JSON files, which our project then uses to represent the electrode and functional map data. Unintentionally, these files provided us with many use cases for possible errors when generating the JSON files. The .xlsx sheets had invisible lines, misplaced columns, seizure types that were not in our application, and so on. We built the preprocessing code to withstand these errors, and give more meaningful feedback.

One final way in which we secured our application was the use of constant variables. This also accomplished two things.

1. We never had to worry about a variable changing value in a function. Once a const value is set, it will not change for the duration of the function. If it is a reference, it will not be re-assigned.
2. The functions always produced the same output given the same input. Since the overwhelming majority was declared const, our output was very easy to determine and never changed with the same input.

6 Supplemental Documents and Notes

<https://ntoolsbrowser.github.io/>
<https://github.com/ntoolsbrowser/ntoolsbrowser.github.io>
<https://nyulangone.org/locations/comprehensive-epilepsy-center>