**Primary objective:**

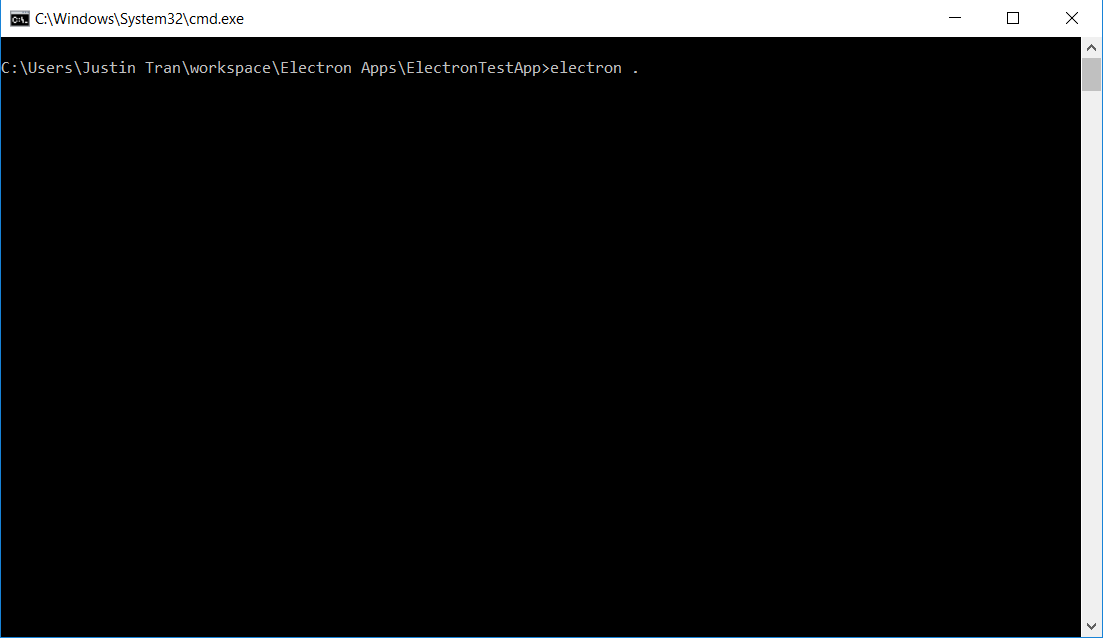
Configure an HTTPS route to a local Nginx server. To do this we would need to create a self-signed SSL certificate. Configure the Electron app's Sentry's SDK to send data to this HTTPS port. Essentially, we are routing all crash data from the Electron app through a HTTPS Nginx server before using the server as a proxy to send the data to the Sentry platform. This is required due to the limitations of the hospital network that only allow access to the VM and no other internet access.

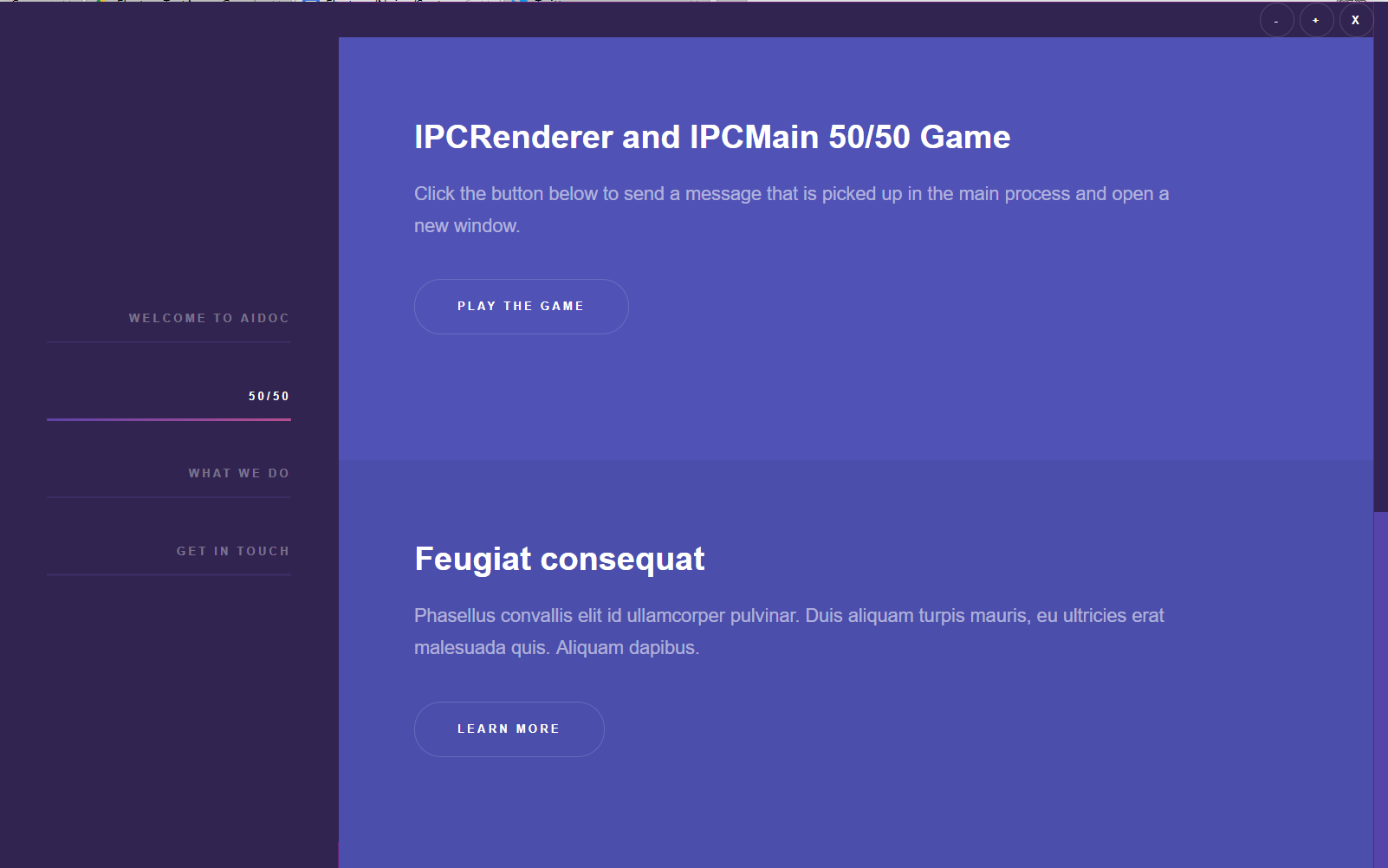
Note that all code is found in the [Google Drive Folder](https://drive.google.com/open?id=1-_I1WQYWvzWdId3Fyj6zzAAW8l_GvTWJ). The following are a few of the important files vital to the process:

* Electron
  + [Main.js](https://drive.google.com/file/d/1QKjLhFm0jJKeg87uEcMXGlTCb9WhHHc1/view?usp=sharing) (contains basic Electron app structure)
* Nginx
  + [Nginx.conf](https://drive.google.com/file/d/1K3TIluruT2V6paxGGQz08p5Za4AbMy_6/view?usp=sharing) (contains server configuration and HTTP to HTTPS redirection)
  + [Access.log](https://drive.google.com/open?id=1wQduS3kxJo-4OVc43ZZHsWWY-_Y-F-a0) (log of all data to and from server)
* Sentry
  + [Sentry.js](https://drive.google.com/file/d/1hNSj4uyGfJ42TP4qn4a_kpH4qfNHmsfr/view?usp=sharing) (contains DSN)

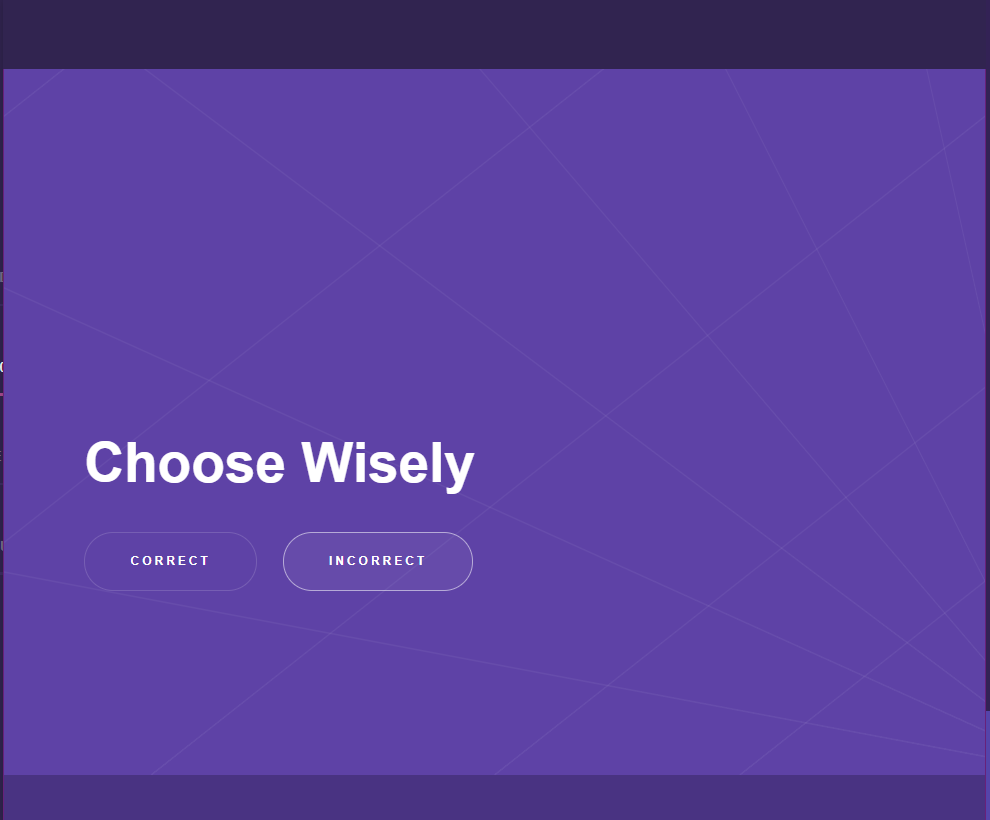
**Electron App**

**Using the Electron App to Generate Exceptions:**

1. Start the Electron app in the command line with “*electron .”*

**

2. Navigate to the 50/50 Sidebar Tab and click on “**Play the Game**”.



3. Press the “**Incorrect**” button to generate an exception and cause the Electron app to crash.

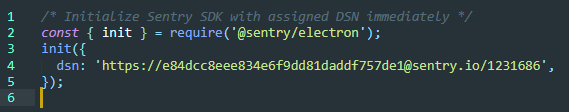
**Electron App Summary:**

To use the Electron App as a means to easily generate crashes for testing the rest of the infrastructure, I had to create an easy way to generate a crash that could be reminiscent of an actual exception caused by user input. So, I created a simple button that could be triggered by a user click which would cause the entire Electron app to crash and have the data eventually routed to Sentry. The Electron app worked as intended throughout the development and testing phase.

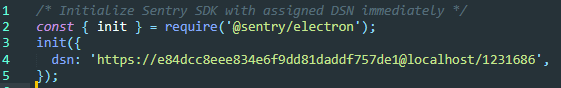
**Sentry**

**Sentry Summary (sentry.js):**

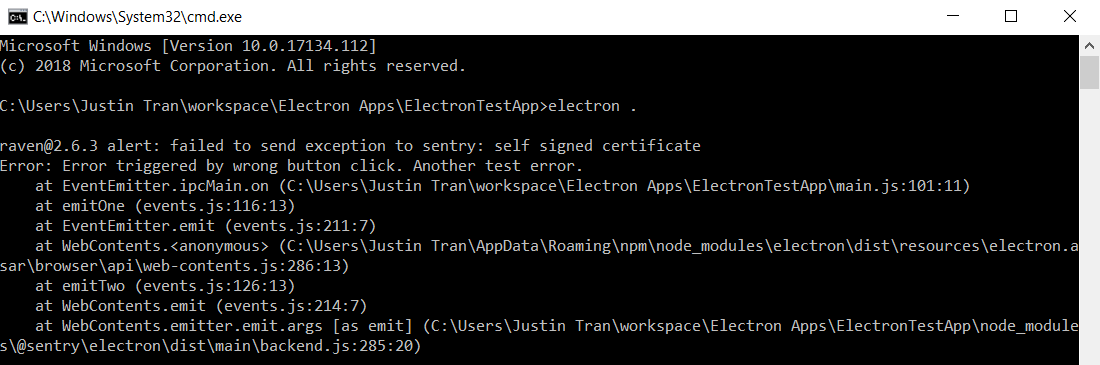
Note that all code related to Sentry is found in the Electron app’s root files.



This is the standard way to query the [Sentry platform in an Electron app](https://docs.sentry.io/clients/electron/) with a private DSN. For more on how the DSN is parsed, [visit this page](https://docs.sentry.io/clientdev/overview/#parsing-the-dsn). Our DSN is in the form **https://[PUBLIC\_KEY]@[HOST]/[PROJECT\_ID].** The important aspect of this is that the host is **sentry.io**. Clearly we can’t use this configuration for our architecture where the hospital and its Electron app does not have access to the internet. So, we must route it through our server like so:



Note that our Nginx server must be started to begin this. Also note that we changed the host to be **localhost** (our Nginx server) and that the DSN is still HTTPS. Unfortunately, it turns out that Raven does not appreciate that our server uses a **self-signed** certificate. Raven throws an exception and our crash data fails to be routed through Nginx.

****

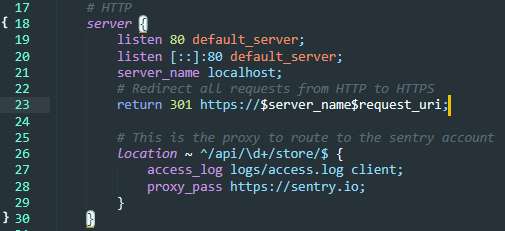
**Sentry Possible Solution/Shortcomings:**

Seemingly the only way around this error is to send the DSN **without HTTPS** and with HTTP instead. Everything works perfectly when the crash data is routed from Electron to our HTTP configured server and finally to the Sentry server (which is HTTPS) but this is obviously not secure. An alternative solution is to let our Nginx server config handle the redirection from HTTP to HTTPS instead (note that this also does not work effectively)

**Nginx**

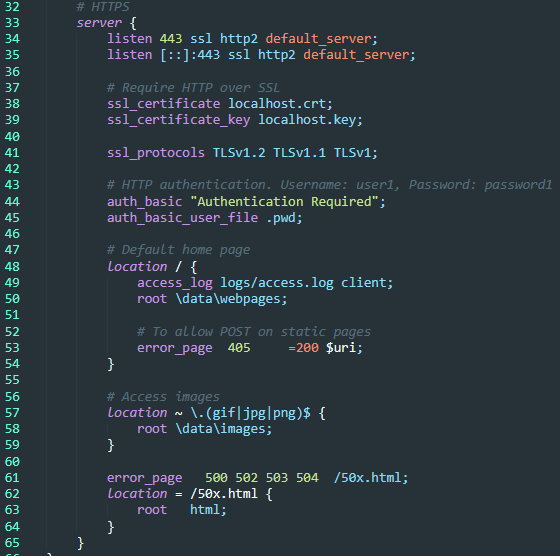
**Nginx Summary (nginx.conf):**

Start the Nginx server with the command **nginx** in command line**.**

****

Lines 17-30 contain the server configuration that can be accessed through *localhost* or *http://localhost.* This does not utilize any certificate and is therefore unsecure. This is why **lin5e 23** redirects all requests to http://localhost to http**s**://localhost which is secured with a self-signed certificate. We want our data to be secure when making requests to and from the Nginx server which is why we use SSL or TLS.

Lines 25-29 contain the proxy that takes the data sent from the Electron app crash and Sentry’s [Raven client](https://docs.sentry.io/clients/javascript/) (which handles and reports all crash data in a form that can be read by Sentry). The *location* directive contains a regular expression that handles data sent from Raven in the query form: [**…/api/PROJECTID/store**](https://docs.sentry.io/clientdev/overview/). [This code](https://forum.sentry.io/t/working-with-the-saas-from-a-private-network/3885/2?u=justin.tran.jt) was provided by a developer on the Sentry forum.



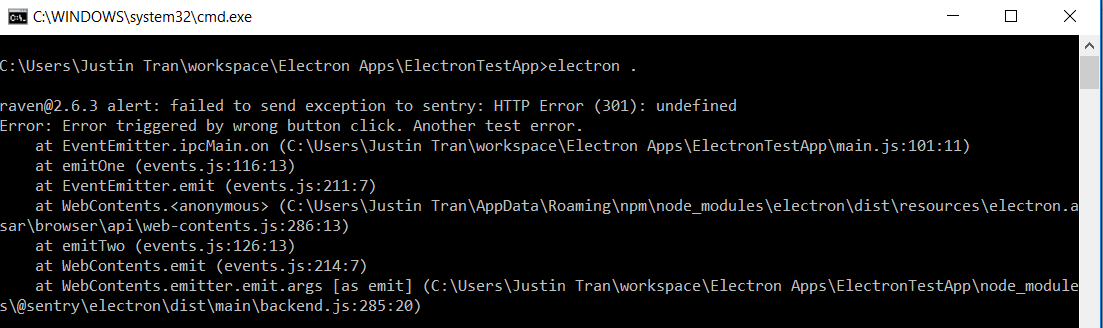
Lines 33-65 contain the HTTPS configuration for our server. It is self-signed with an SSL certificate as seen in lines 37-41 and also password protected by a simple test username and password as seen in lines 43-45. Links to webpages and access to images stored in the server’s root are provided by location directives in lines 47-59.

Lines 52-53 were used for testing the sending of HTTP requests to the server. Nginx servers hosting static pages do not allow POST requests (HTTP code 405) so a workaround was made in which GET requests with a *request\_body* were sent to the server instead and 405 errors were converted to HTTP code 200 (success). This was all logged in *access.log* to ensure that HTTP requests and data sent to and from the server was making its way through Nginx effectively.

**Nginx Possible Solution/Shortcomings:**

Remember our earlier error with Sentry’s inability to route its data to our HTTPS server? We were forced to send the data to the server in the form of a HTTP route instead. To handle this, we could attempt to handle it the same way we handled all requests to the HTTP server by routing it automatically through HTTPS and securing the transmission.

Recall line 23 of Nginx.conf in which we perform a URL redirection (HTTP code 301) to the exact same address using HTTPS. This redirection works perfectly when simply querying the server for example at <http://localhost> and visiting the static webpage. **However,** this does not work when redirecting the DSN from Raven. Instead, we receive an error like so:



The DSN cannot be routed from HTTP to HTTPS through Nginx and eventually to the Sentry platform because of an *undefined* error.

**Conclusion**

We are at a point where the primary objective is met but only on the condition that the data is sent through our server when it is configured with HTTP rather than when it is configured with HTTPS via a self-signed certificate.

To continue with the primary objective, we must figure out how to route the crash data through our HTTPS server. We are currently in contact with a developer from Sentry but we are not able to receive support from the official Support team at Sentry as we are testing using a free trial/free plan.