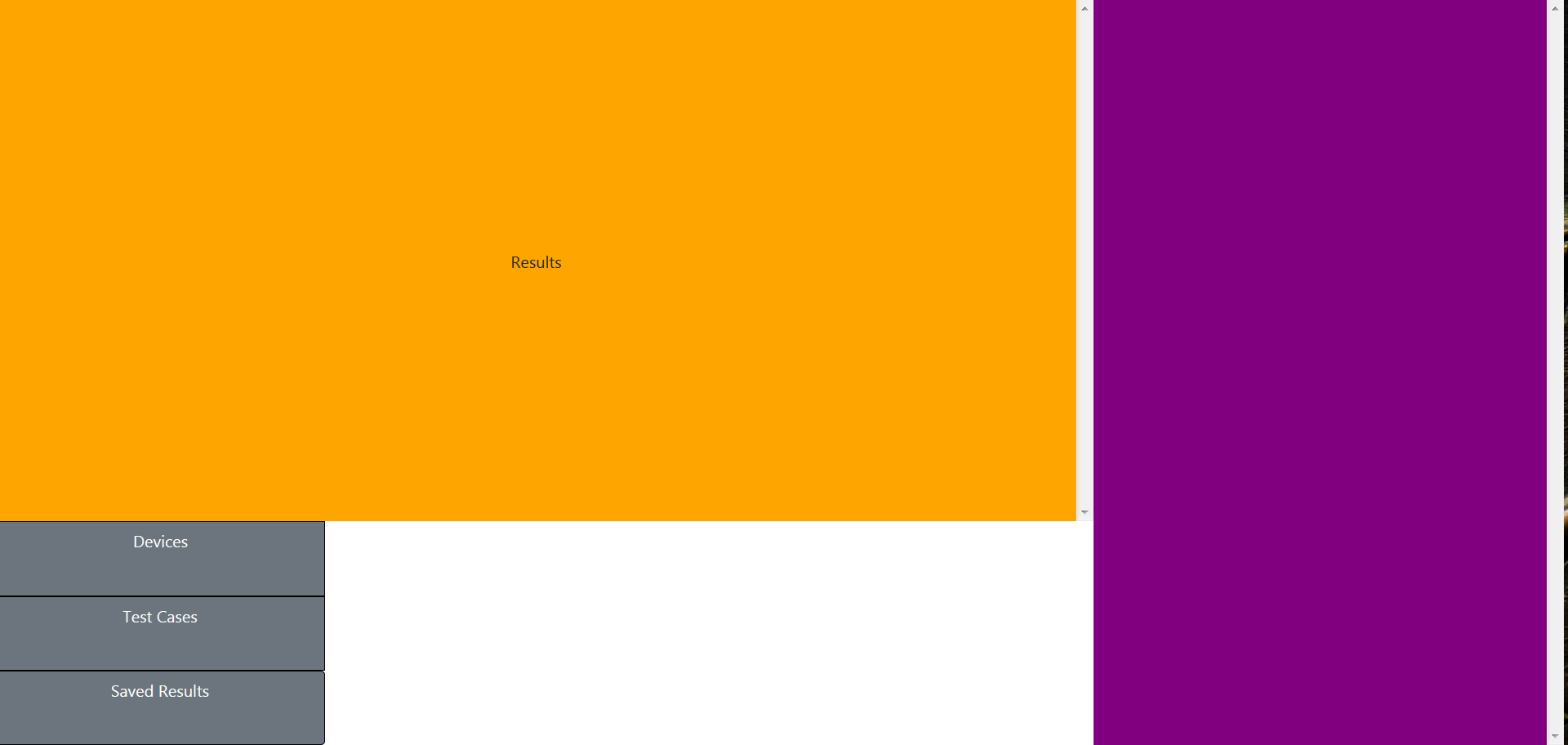
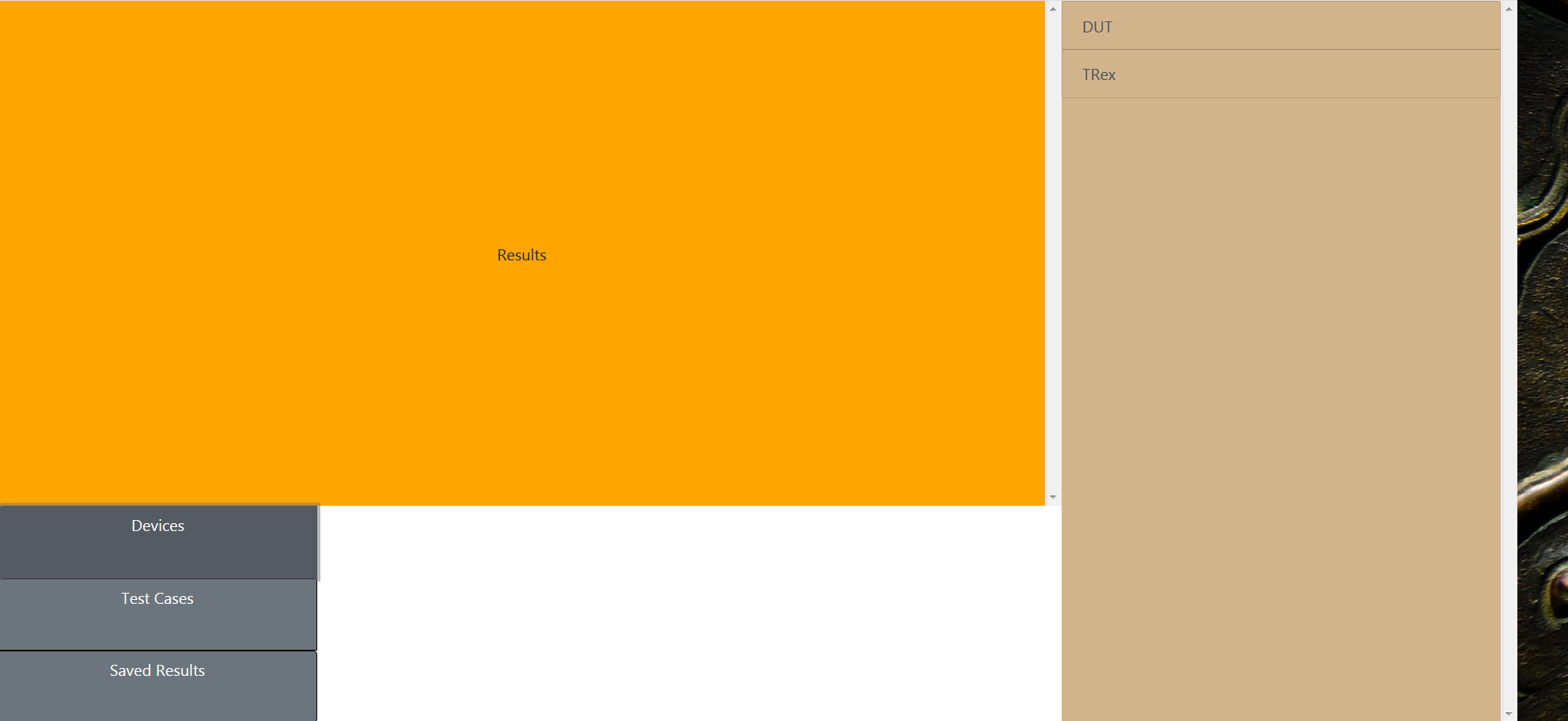
When the page first loads, ignore the colors. The colors are just to emphasize the boundaries of specific areas.



The user can click the Devices or Test Cases tab to receive a list of available devices or test cases from the server. The list of devices (currently) is a JSON file located in a device repository. The test cases are a collection of Python scripts in a test case repository folder. The device list is imported as JSON, dumped into a Pythonic dictionary and iterated over. The test case repository is “crawled” (ie. the code crawls the folder and returns a list of everything in it). The Device list will work like this once we get some scripts together for the repo.



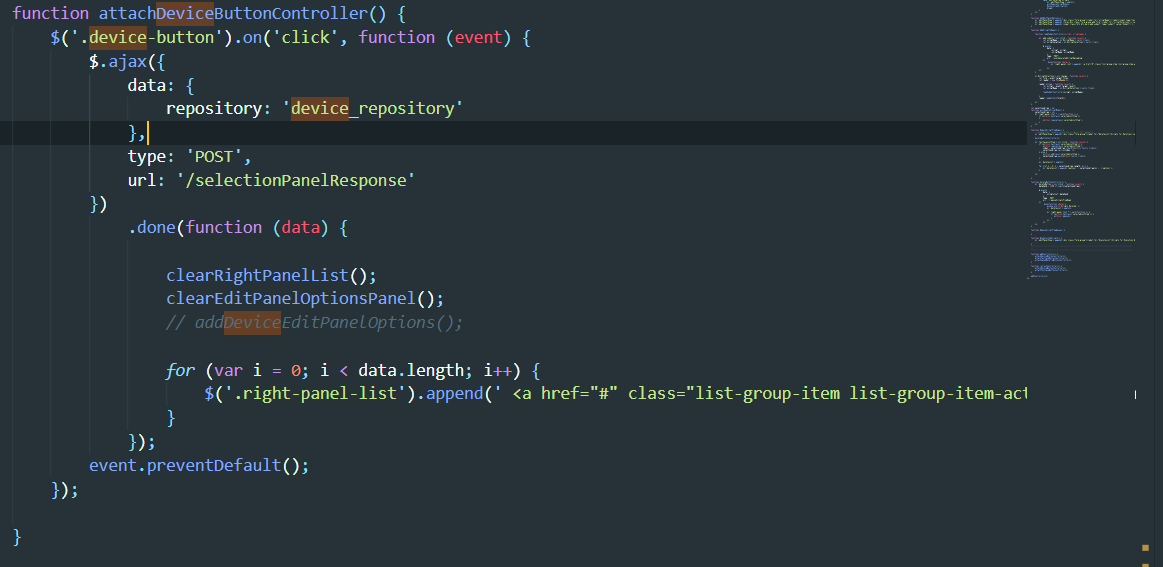
(Located in devices.json)

This list will be replaced with DUT scripts in the future.



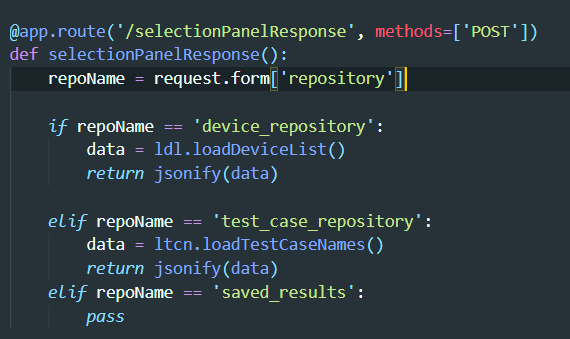
Test case repository.

But basically, the user clicks one of the tabs, an AJAX request is sent to the server with the name of the desired repository (below is what is happening locally on the user’s computer, this is in local JavaScript file):



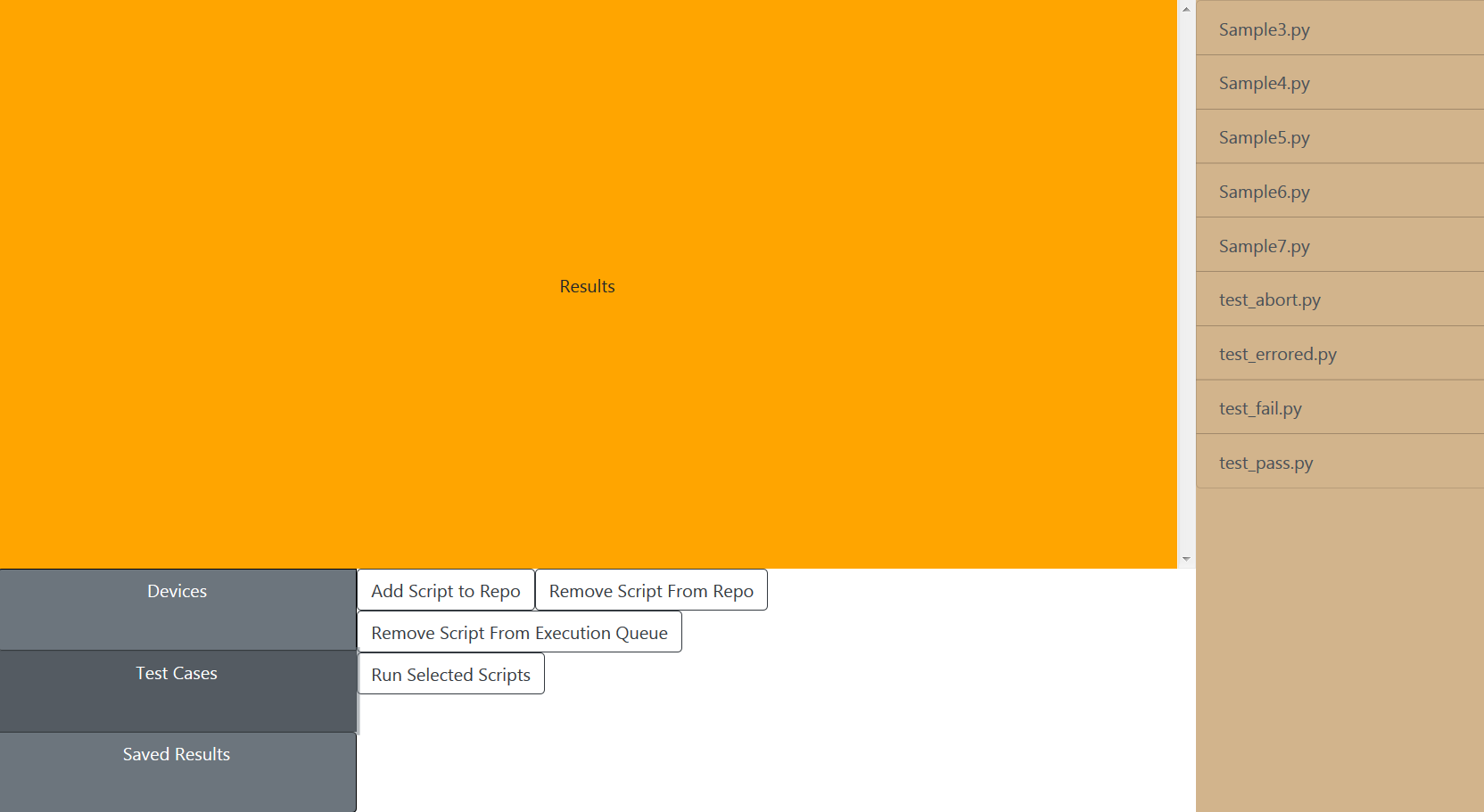
(Located in selectionPanelController.js)

The server (below) receives the AJAX request from the user’s browser in the selectionPanelResponse route (determined by the “url” key in the AJAX request). The route determines which repository is being queried and calls an appropriate imported method (not super important) to produce the desired information (namely a list of devices or test cases).

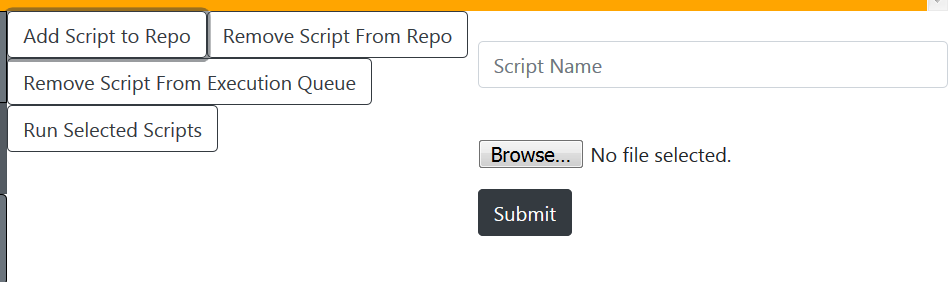


(Located in Main.py)

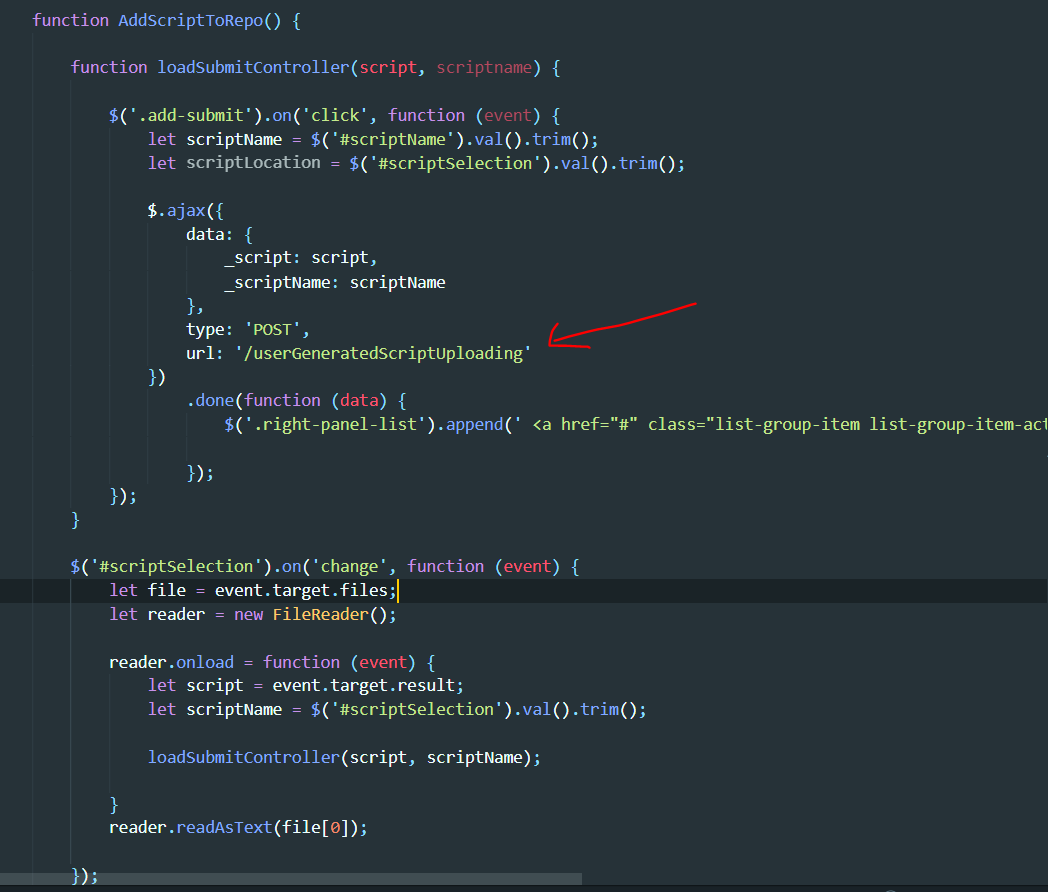
What does matter is that it returns a “JSONified” response to the user’s browser containing the requested data in the form of a simple Pythonic list which JavaScript can accept and use as a JavaScript array. The array, this is happening locally now, is iterated over and a new hyperlink (<a>) element is created for the contents of each array element. The element is then appended (ie. added) to the right panel. This last bit works exactly the same whether the Devices or Test Cases tab was selected.



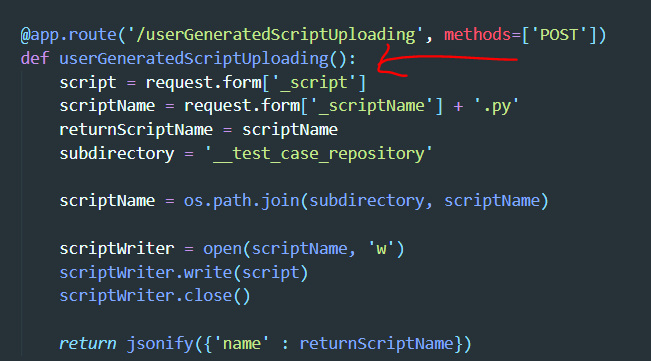
At the moment, only the Test Cases tab has additional options. When you click it, the four additional options are show in the edit panel. Of the four, “Add Script to Repo” and “Remove Script from Repo” are ‘fully functional’. They need to be fine-tuned but they work.



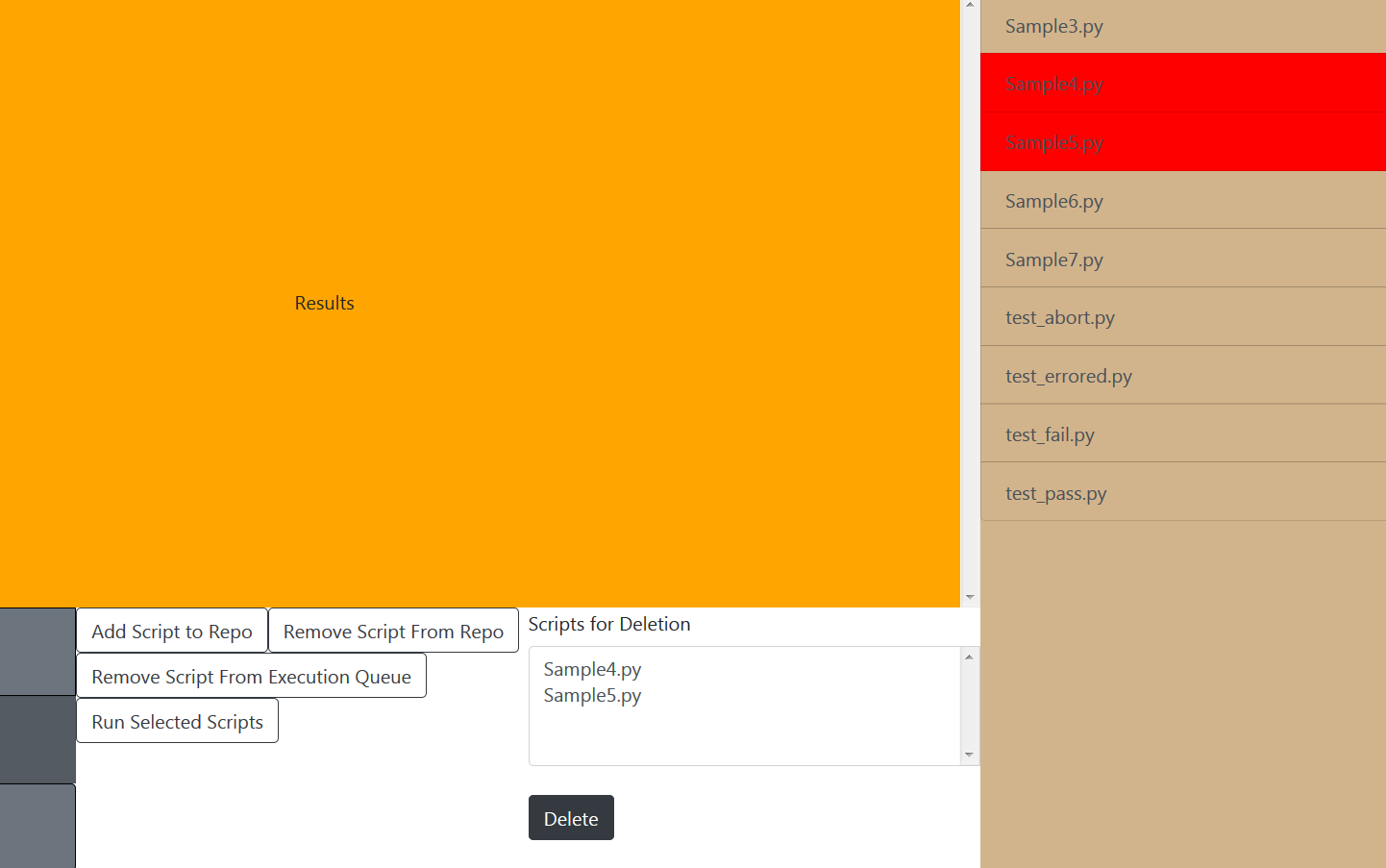
When the user clicks “Add Script to Repo”, a form is produced that allows the user to enter the name of the script (leave off the .py) and the user has the ability to select the script from some directory on their computer. When the user clicks the Submit button, an AJAX request is sent to the server once again:



(Located in selectionPanelController.js)



(Located in Main.py)

The red arrows just point out the relationship between the JS and the Flask route, the url in the JS AJAX request determines which Flask route is being called. Anyway, the javascript locally captures all of the data in the user’s chosen upload file, stores it all in a single string and forwards the string to the server. On the server, a new file is created and the string is written to said new file. 

The “Remove Script from Repo” allows a user to select scripts from the right panel, it collects the scripts in an array and forwards the array to the server. Once on the server, the data structure is parsed and iterated over, and the scripts are one by one removed from the test script repository.

I know it’s a lot but it’s the setup is actually pretty straightforward. Let me know if you have any questions.