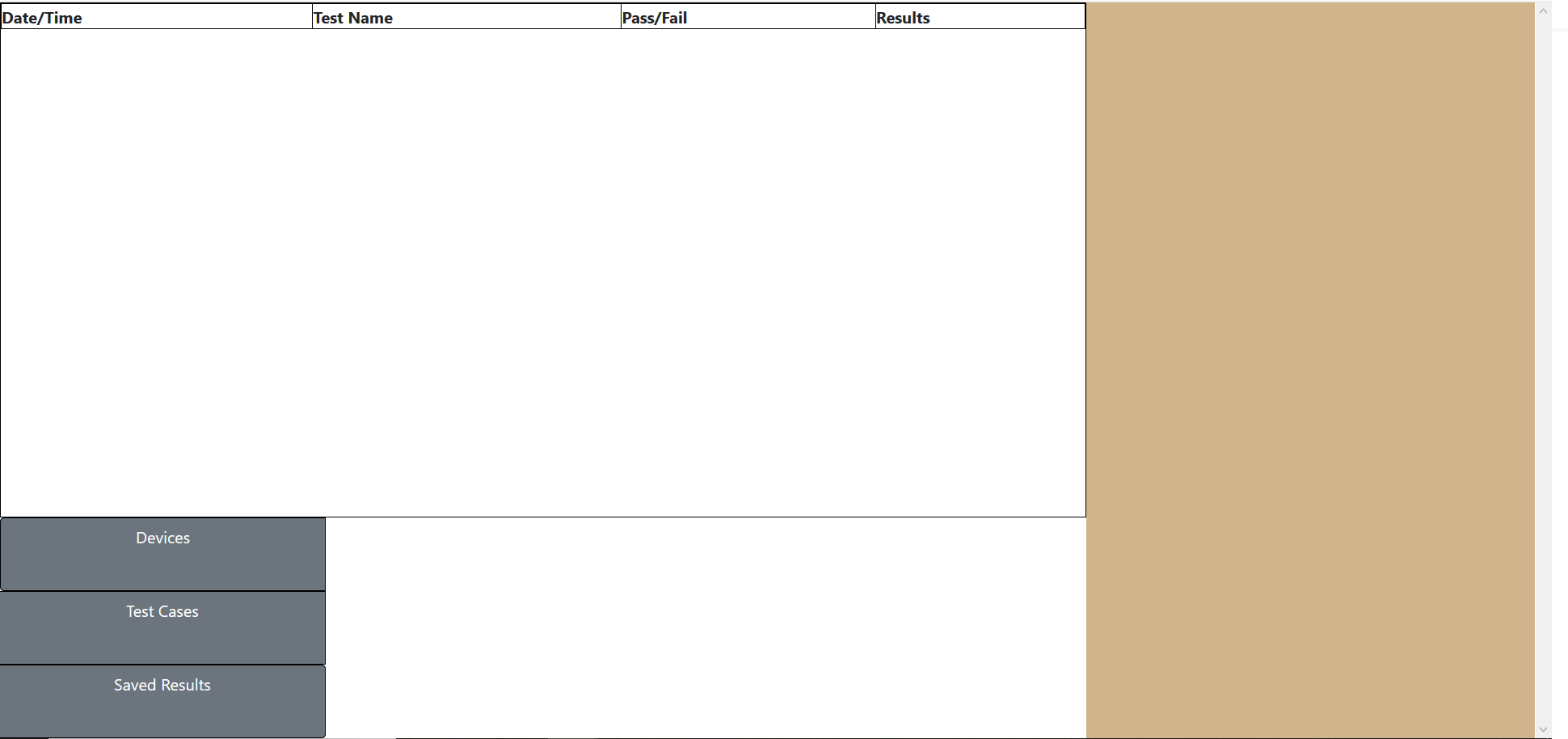
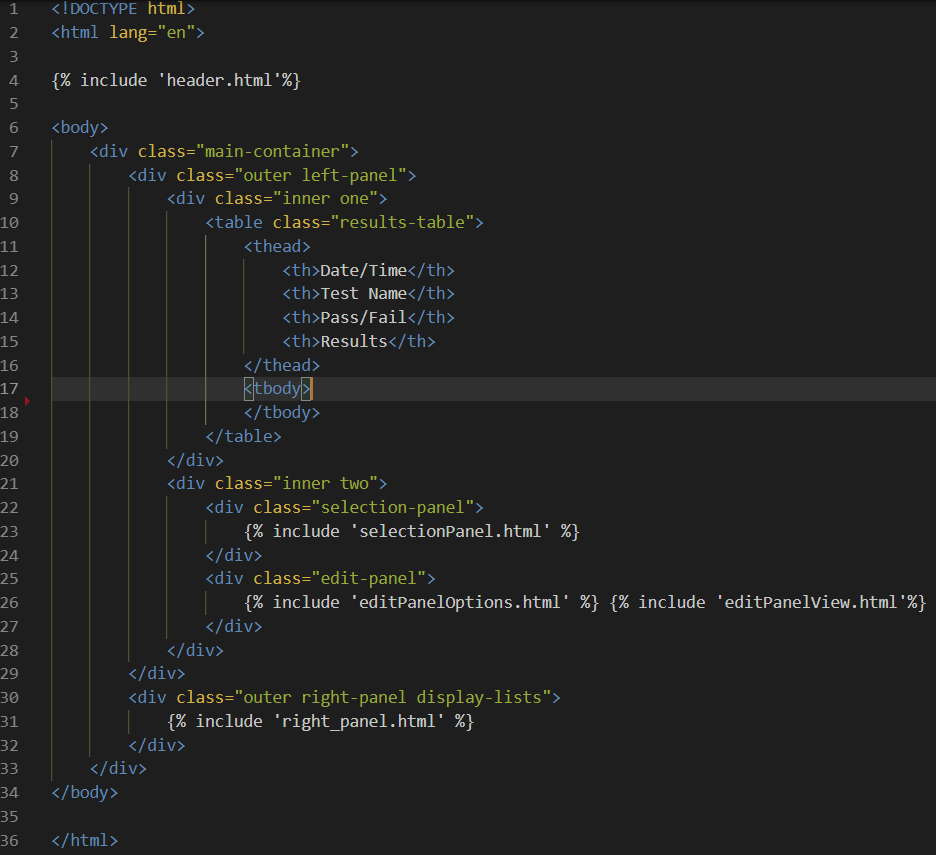
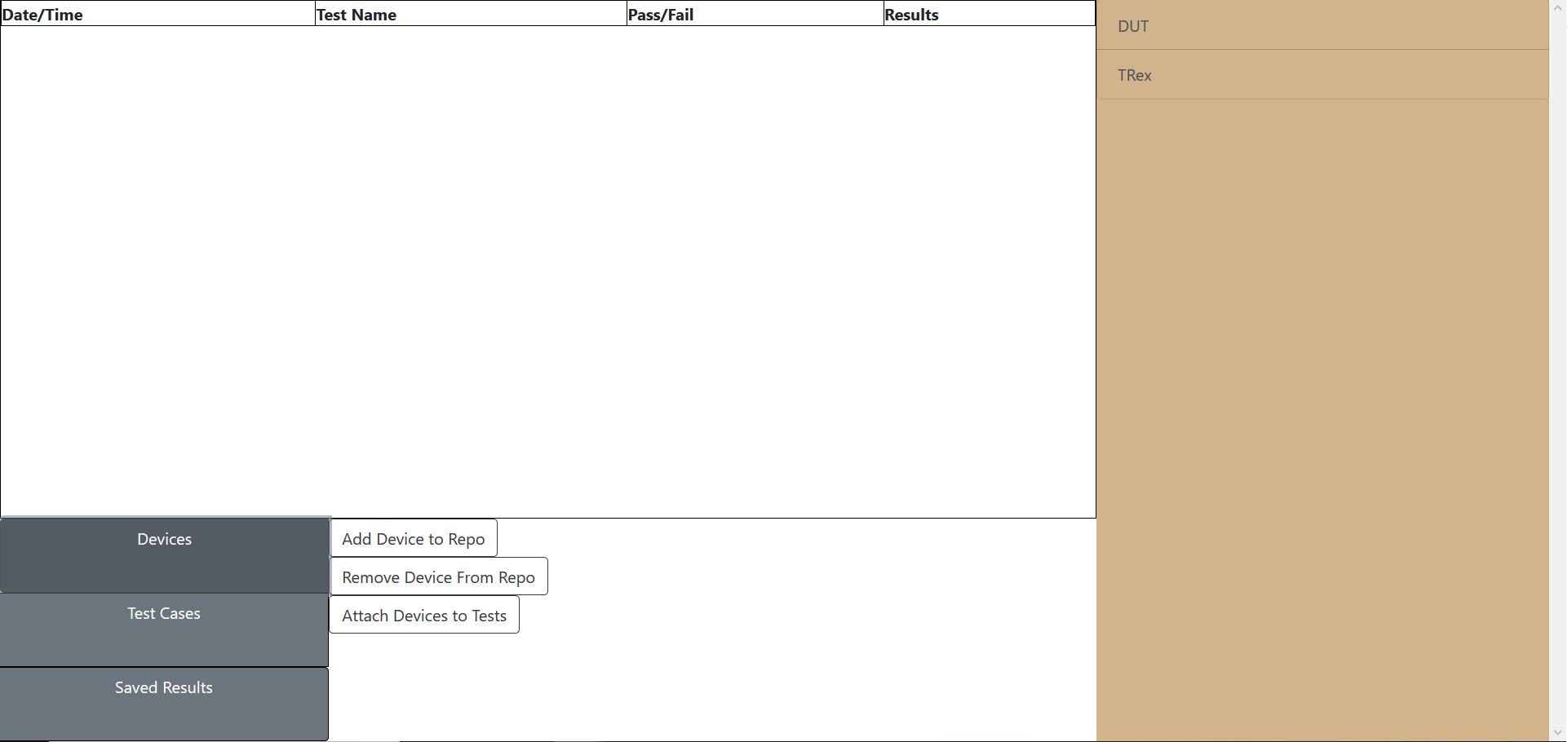
Initial State: Empty saved results table (should change), empty right panel, three visible buttons: Devices, Test Cases, Saved Results

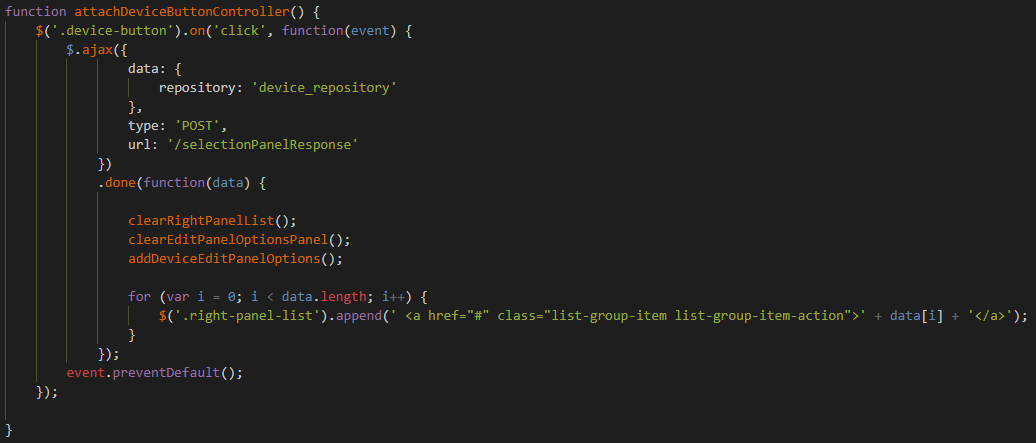


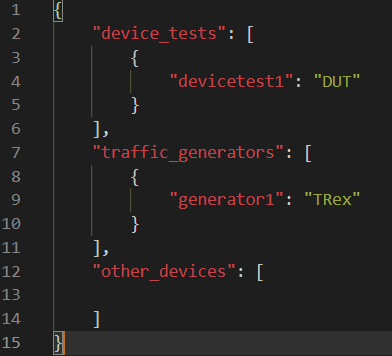


State 2: Click Devices button. Button is darkened to show that it is active, three additional buttons are added to the Edit Panel, right panel is populated by available devices. The available devices are currently listed in a JSON file on the server. An AJAX request is sent to the server containing that indicates that the Devices button has been clicked. On the server, the JSON file is parsed and dumped into a Python dictionary. The dictionary is then iterated over and the names of devices are dumped into a simpler list which is passed back to the local workstation. At present, the Device edit panel options are non-functional.

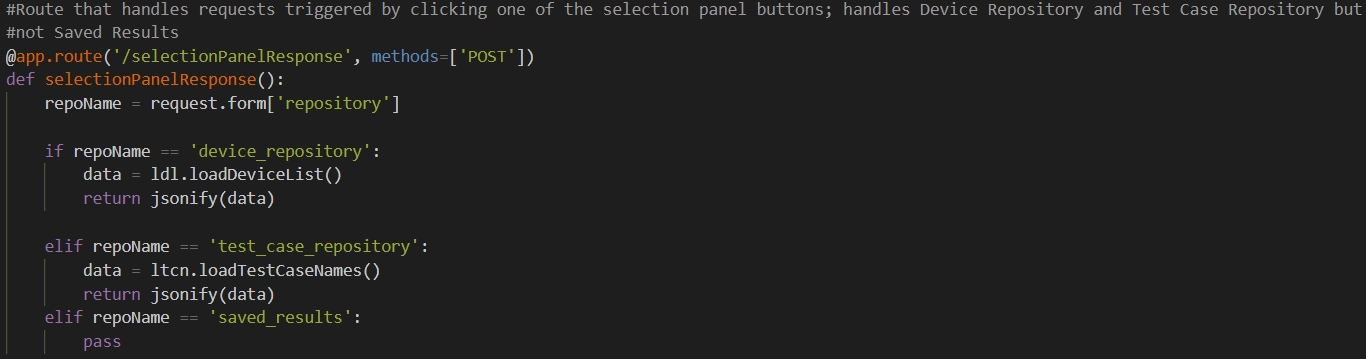


--This is the JavaScript controller for the Devices Button

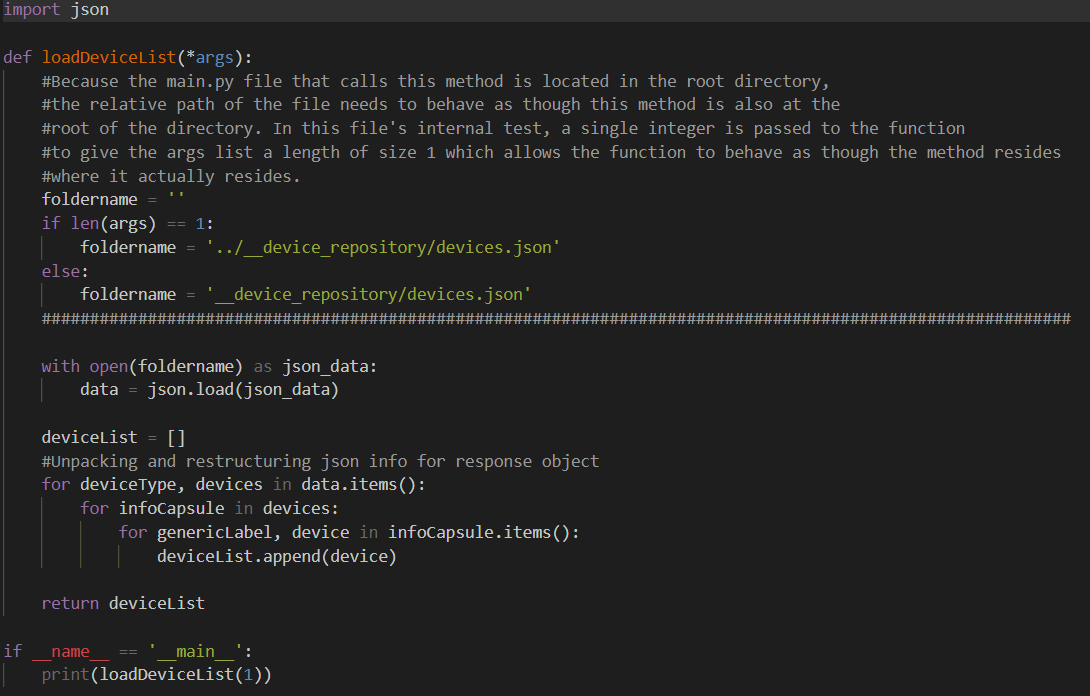




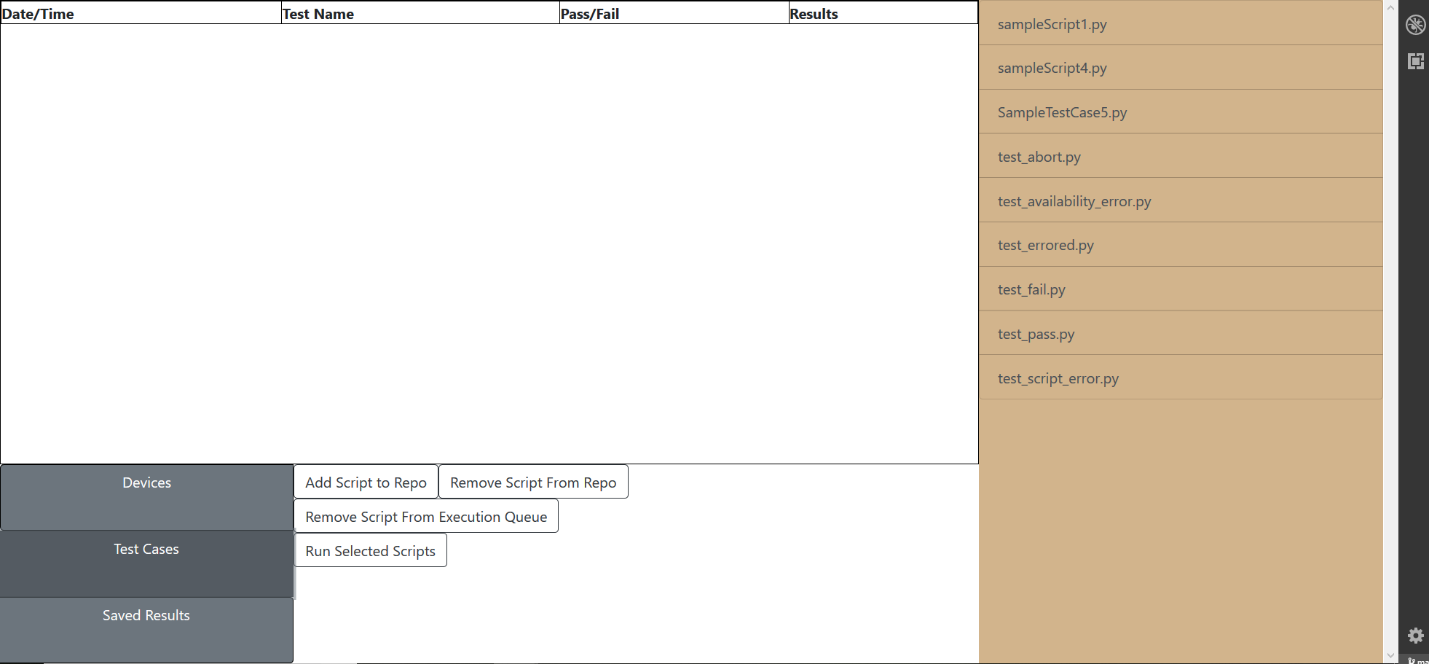
--This is the route in main.py that handles the AJAX request



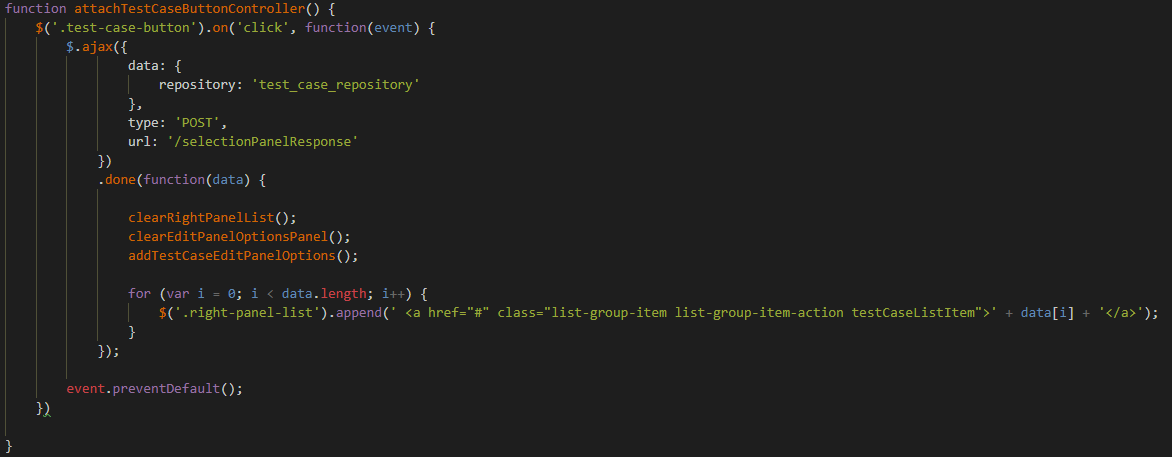
--This is the loadDeviceList.py script located in the \_selectionPanelToolBox directory that iterates over and parses the JSON file



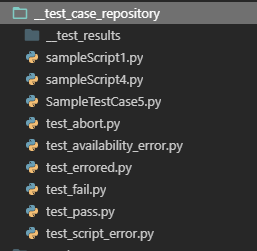
State 3: Test Cases button is clicked. The Test Cases button is darkened (the buttons are set to toggle so clicking one darkens it and “un-darkens” a previously selected button – via Bootstrap). The edit panel is cleared (if populated by buttons/options from a previous selection) and is repopulated with 4 buttons: Add Script to Repo, Remove Script From Repo, Remove Script From Execution Queue(not implemented), Run Selected Scripts



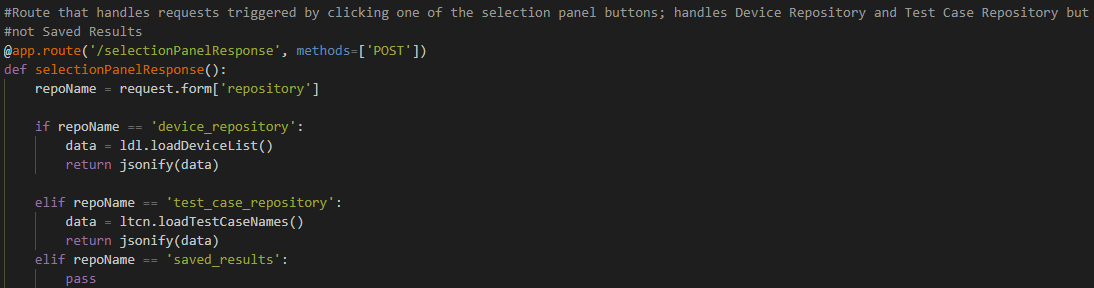
--the JavaScript controller for the Test Cases button. In both of the controllers discussed so far, the success method of the AJAX request (.done method) handles the population of the right panel.



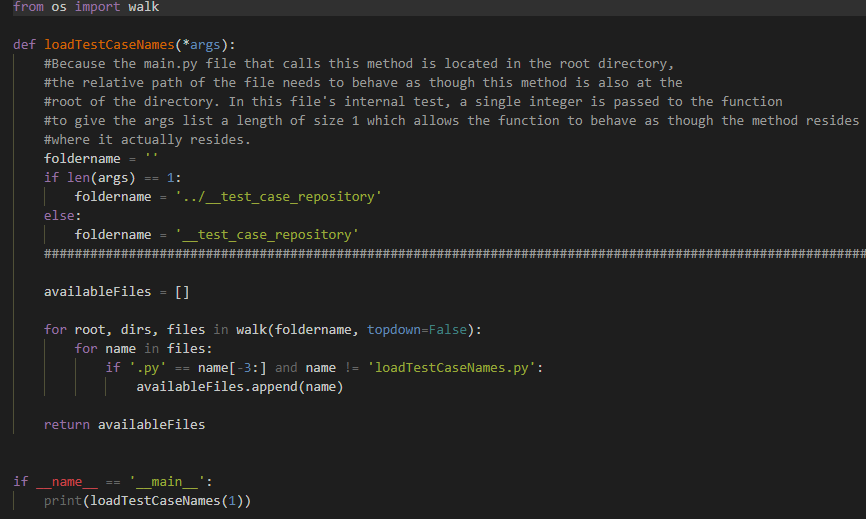
--The test case repository on the remote server. The \_\_test\_results directory will be discussed later.



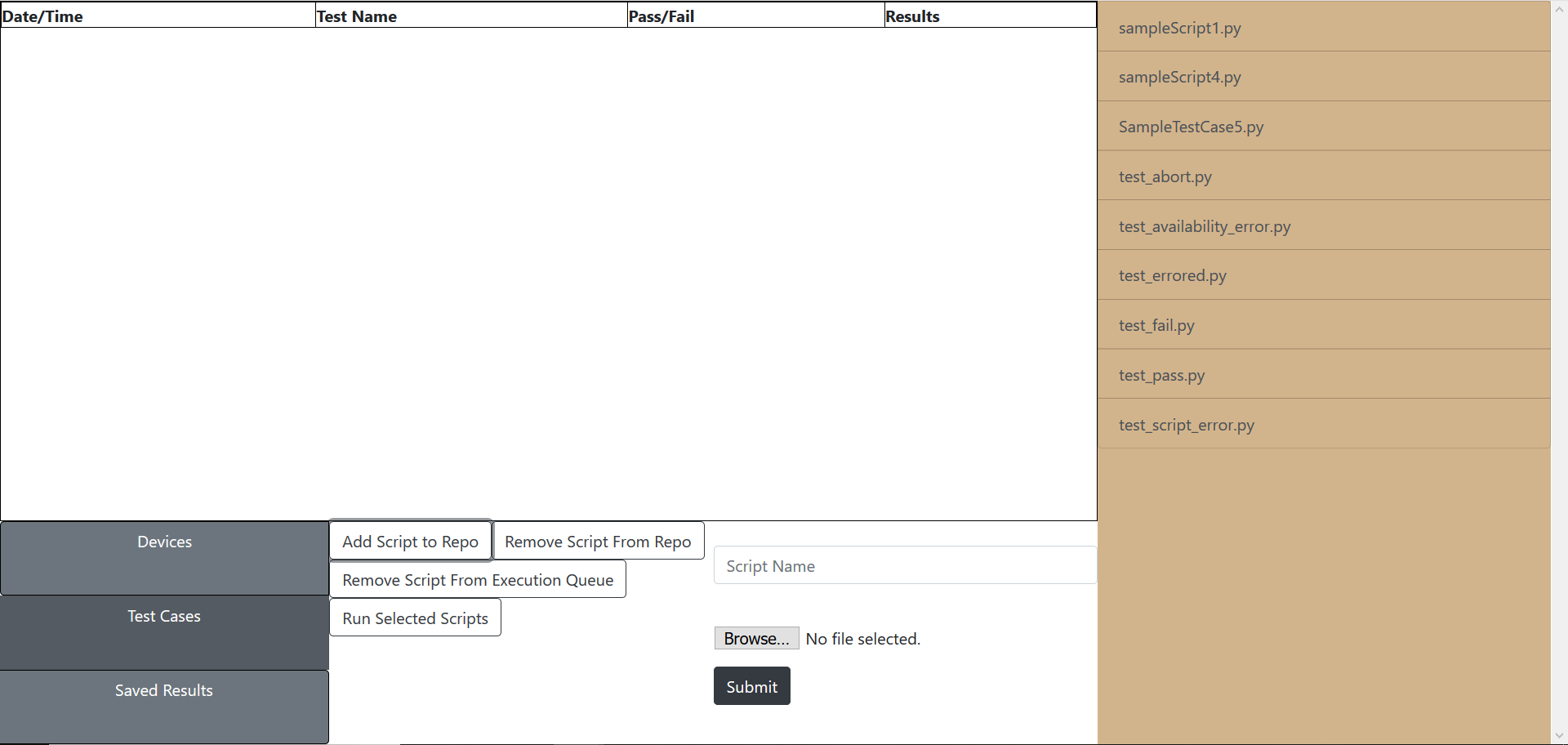
--The route in main.py that handles the AJAX request



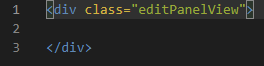
--The loadTestCaseNames.py script in the \_\_selectionPanelToolBox that crawls that the repository and collects the names of the files within the repository.



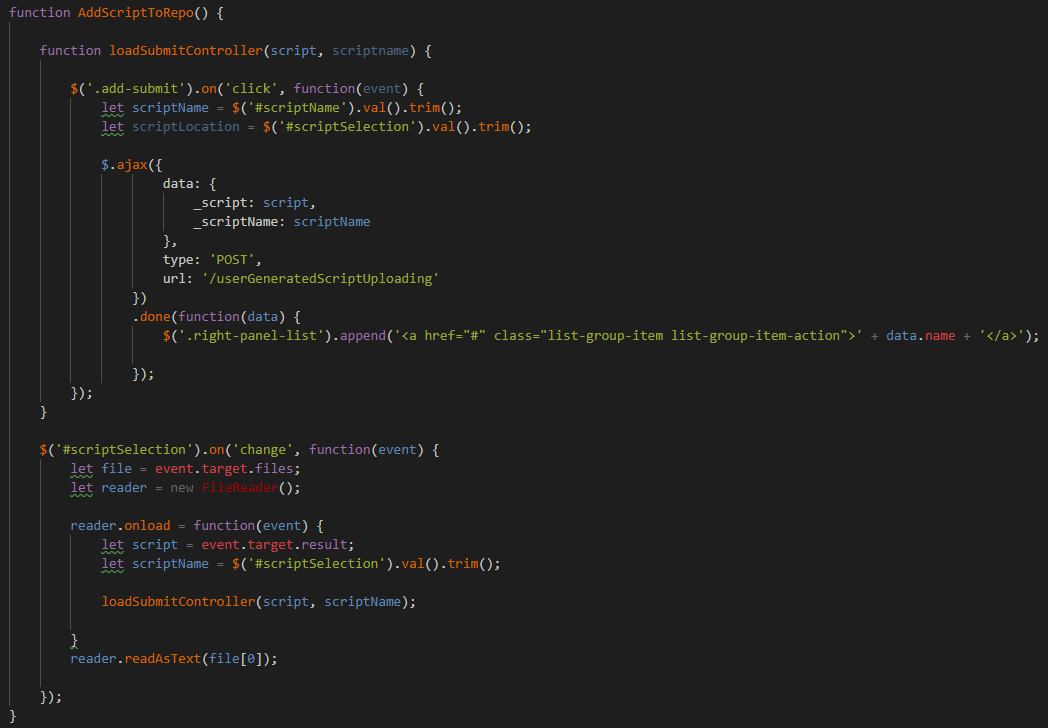
State 4: The Add Script to Repo button has been clicked. In the editPanelView is populated with a Bootstrap provided method for finding and uploading local files. A user can provide a script name (leave off the .py) and select said script from some local directory. Once selected, the user will hit the Submit button and the JavaScript supporting this function will iterate over the script and capture it in its entirety (hidden characters and all) to a string. This string will then be sent to the server where a new file is created in the Test Case Repository and the string will be written to this new file.



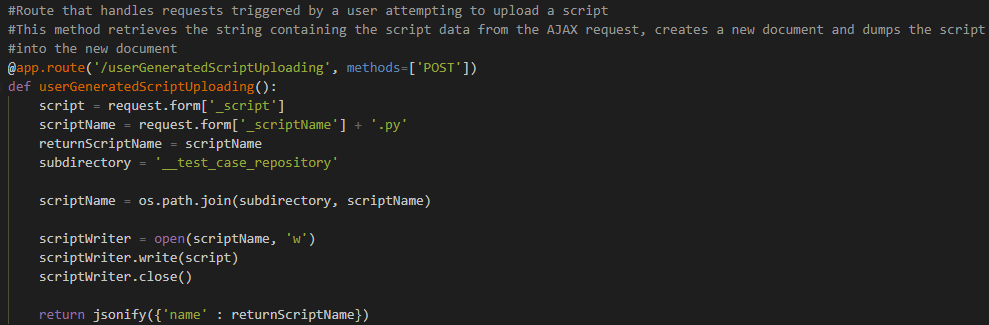
--this is the editPanelView element. All of its content is added/removed dynamically. When the page first loads, the editPanelView is empty.



--The JavaScript method that handles the local script upload functionality. The .done method adds the name of the newly uploaded script to the right panel.

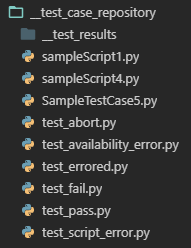


--the route in main.py that creates the new file into which the contents of the string are dumped. The name of the script is returned to the workstation just as a way to verify that the script upload process was successful.

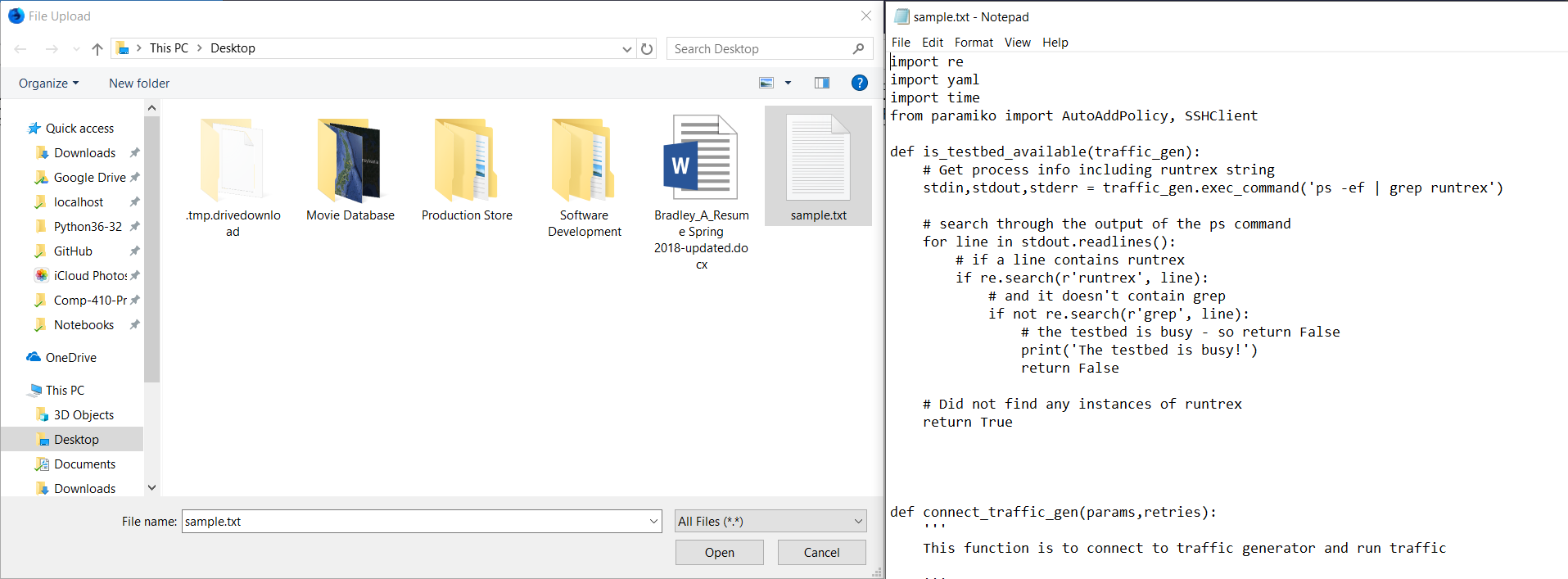


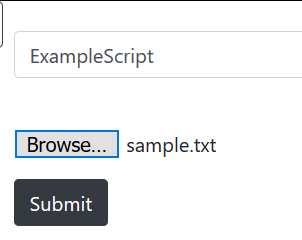
Test:

--test case repository pre-upload of example test

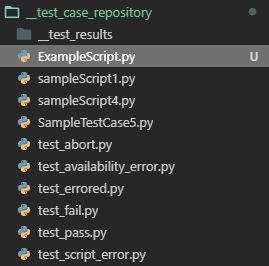


--the local file for upload

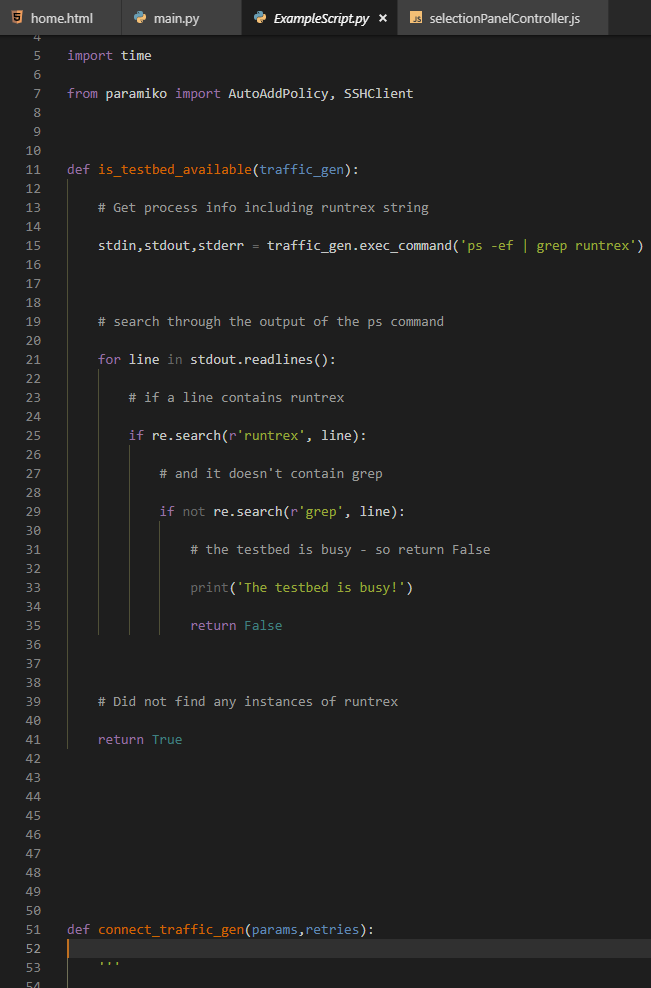




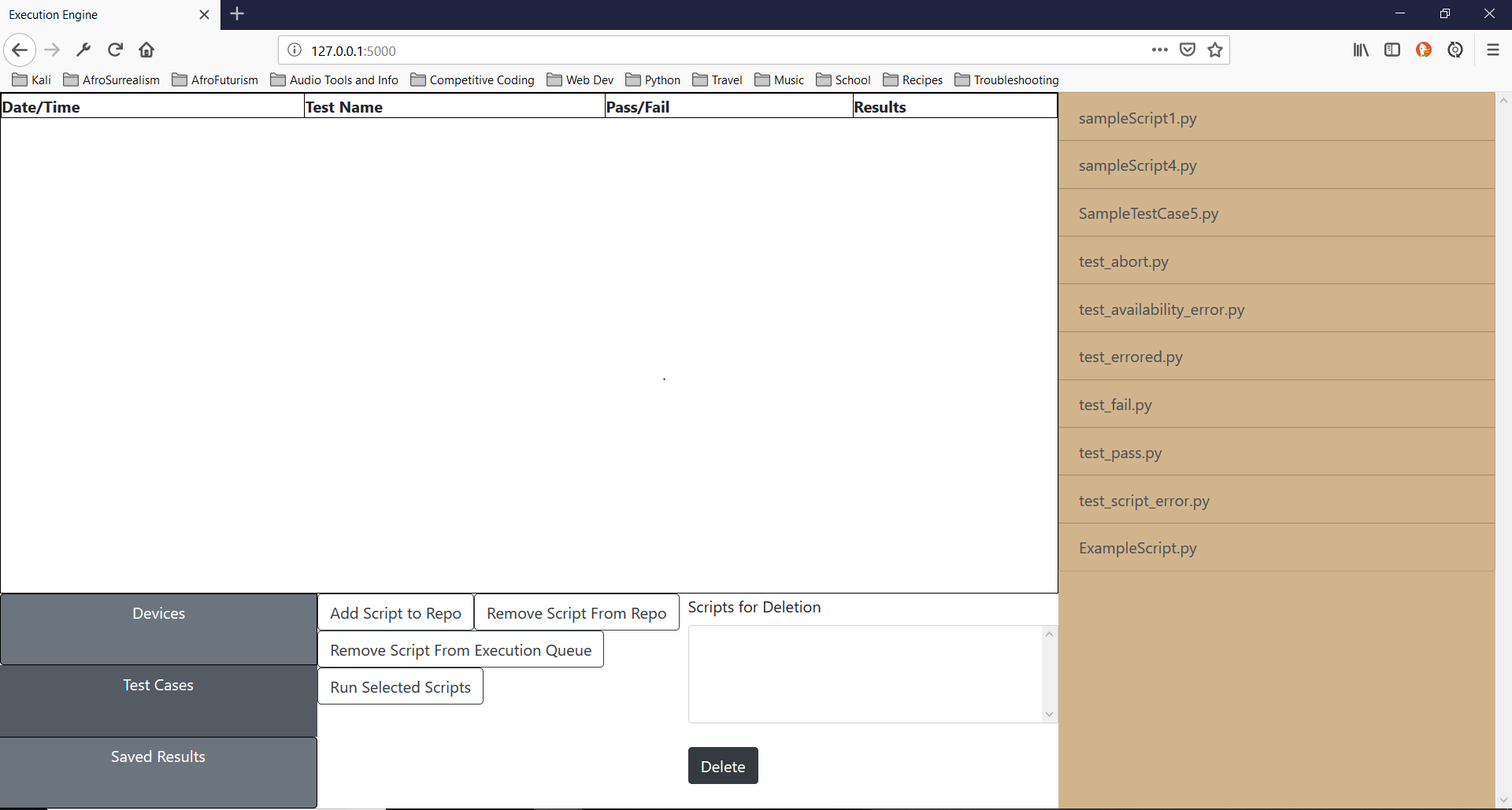
--test case repository post-upload



--the uploaded script, now in a document titled ExampleScript.py on the server



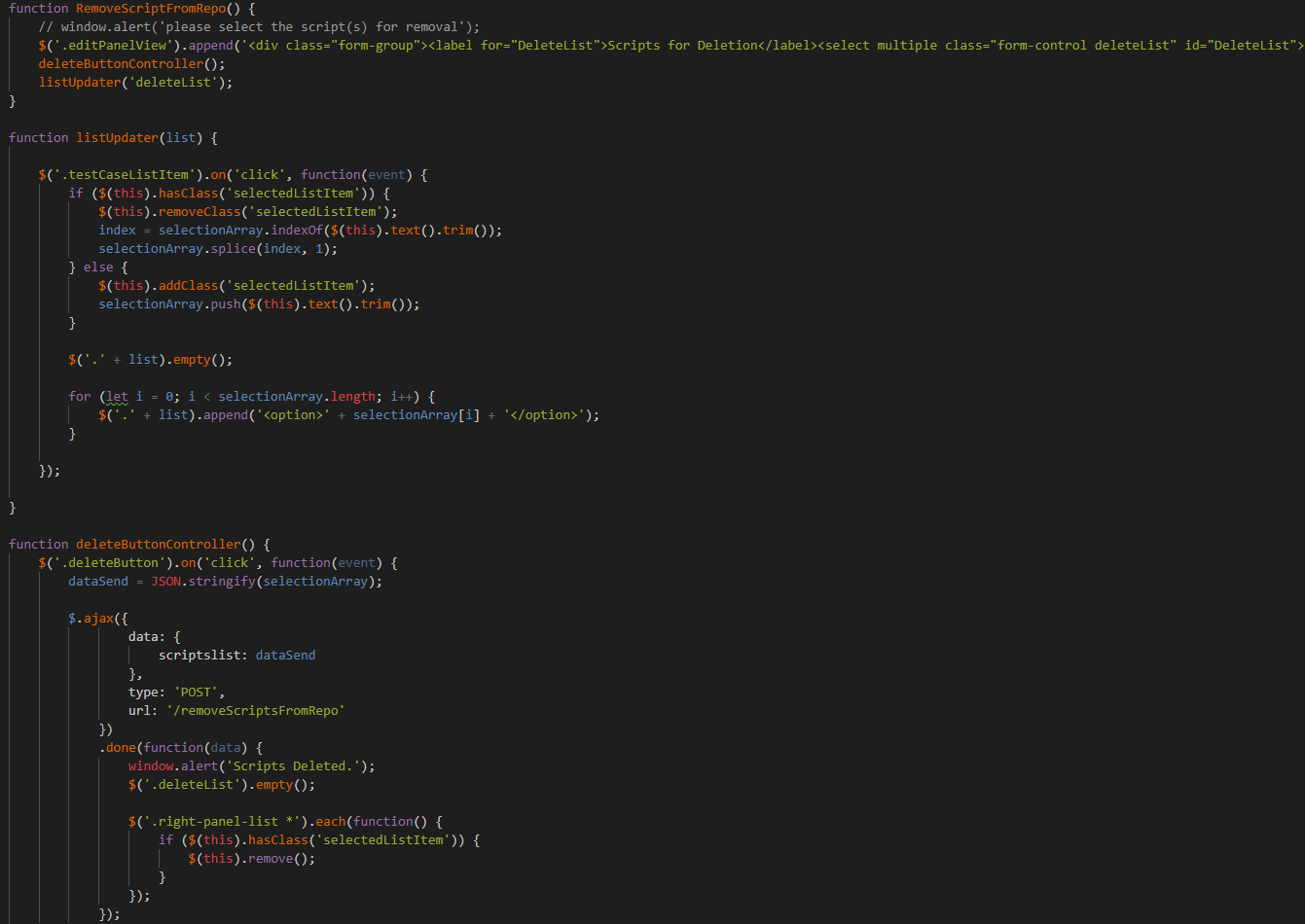
State 5: The Remove Script From Repo button has been clicked. The editPanelView has been cleared and repopulated with a BootStrap provided list. The scripts listed in the right panel can now be selected and upon selection, will also be added to the list in the editPanelView. When the delete button is clicked, an array of the names of the selected scripts is sent to the server for script removal.



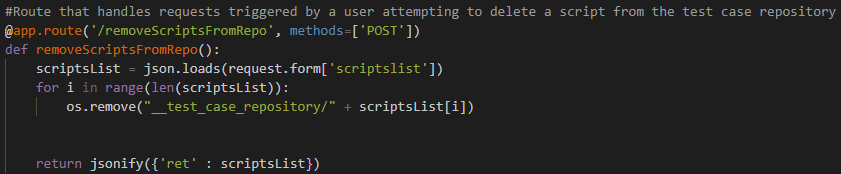
--Three JavaScript methods that handle the functionality of the Delete From Repo editViewPanel. RemoveScriptFromRepo() attaches two controllers.

The first controller attached is the deleteButtonController() and it provides the delete button with the functionality that sends the list of the names of the selected scripts, via an AJAX request, to the server.

The second controller attached is the listUpdater() which provides the right panel list items with the ability to be selected, changes the background color of the item to show that it has been selected/deselected (by attaching/removing the selectedListItem class which has an associated CSS rule). It also adds/removes selected scripts from a selectionArray. If the selected list item has the selectedListItem class, the class is removed, the index of the script name within the selectionArray is found and the array is spliced to remove the script from the list. If the selected list item does not have the selectedListItem class, the name of the script (the text value of the list item element) is pushed into the array and the selectedListItem class is added to said element.

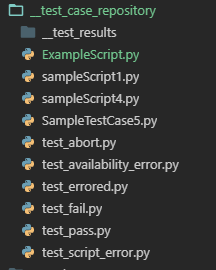


--the route in main.py that handles script deletion

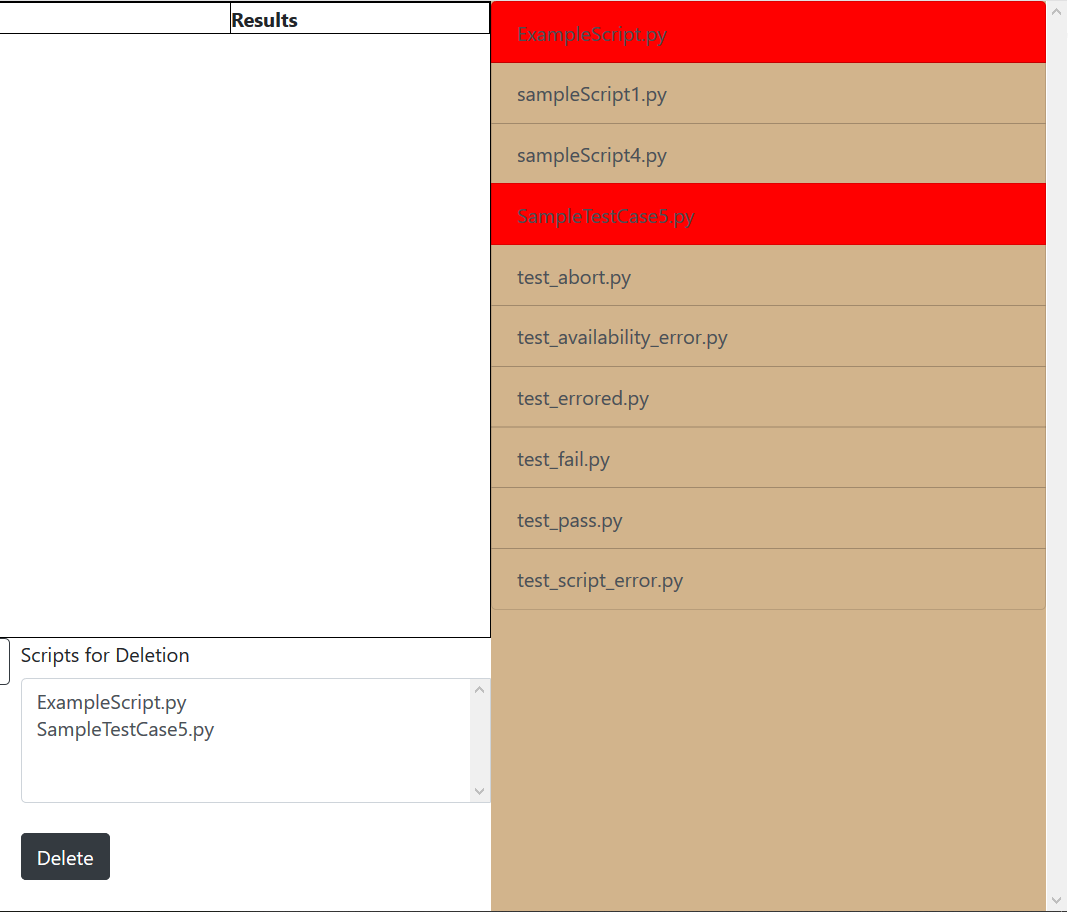


Test:

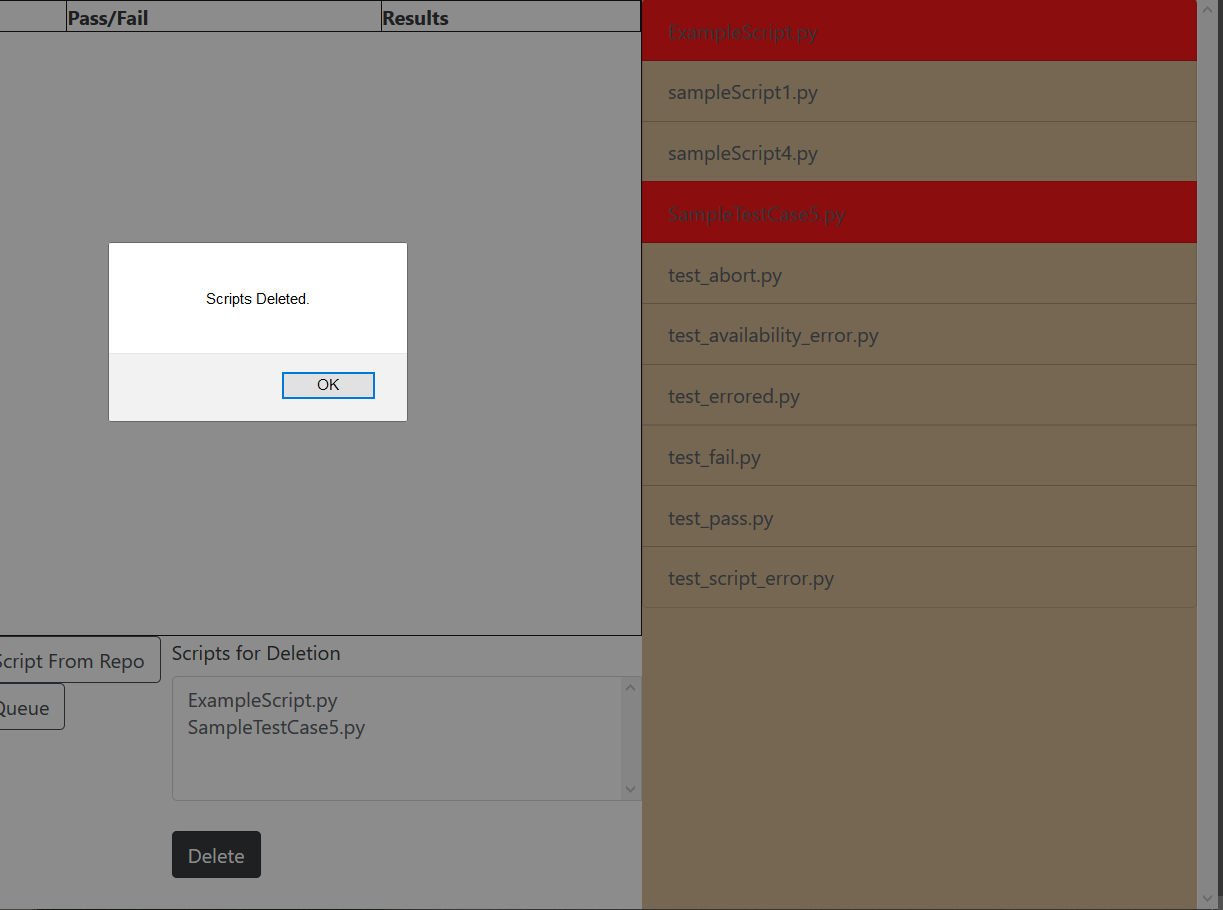
--test case repository pre-script deletion



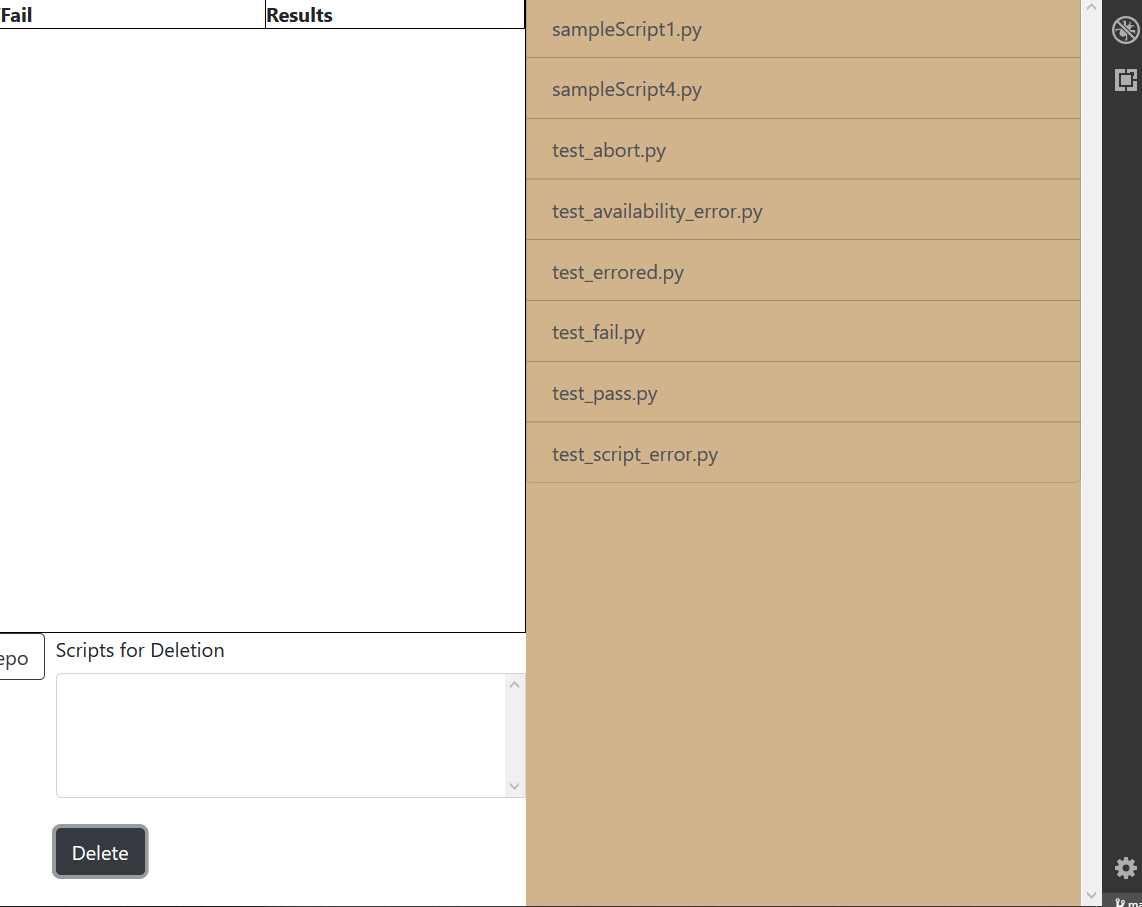
--editPanelView and rightPanel when scripts are selected



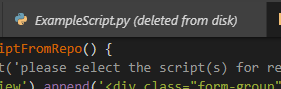
--A message (provided by the .done method of the AJAX request) lets the user know that the script deletion was successful



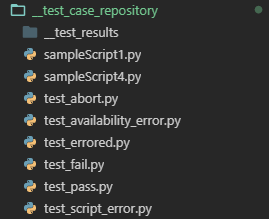
--editPanelView and rightPanel post deletion (no refresh needed)



--the ExampleScript.py happened to be open when this test was run. VS Code is indicating that the file has been “deleted from disk”

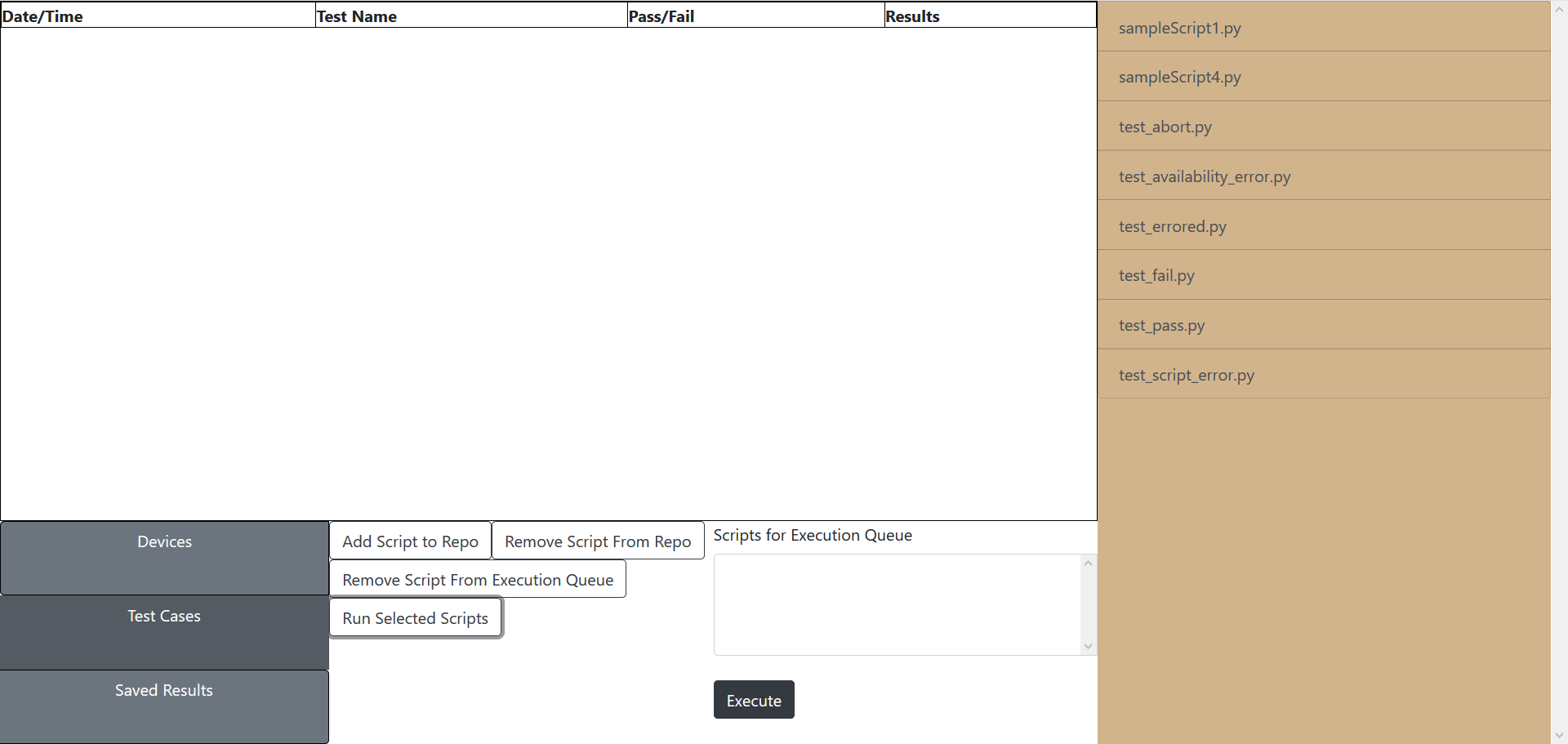


--test case repository post-deletion



State 6: The Run Selected Scripts Button has been clicked. This functions much like the previous state, the only difference is that the scripts are set to be run rather than deleted. The selectionArray of script names is forwarded to the server and, upon arrival, are dumped into an execution queue (a Queue object of the multiprocessing module). There is an executioner that is looping infinitely on a separate thread, this action begins as soon as the server starts. The executioner is constantly checking the execution queue to see if there are scripts to be run.

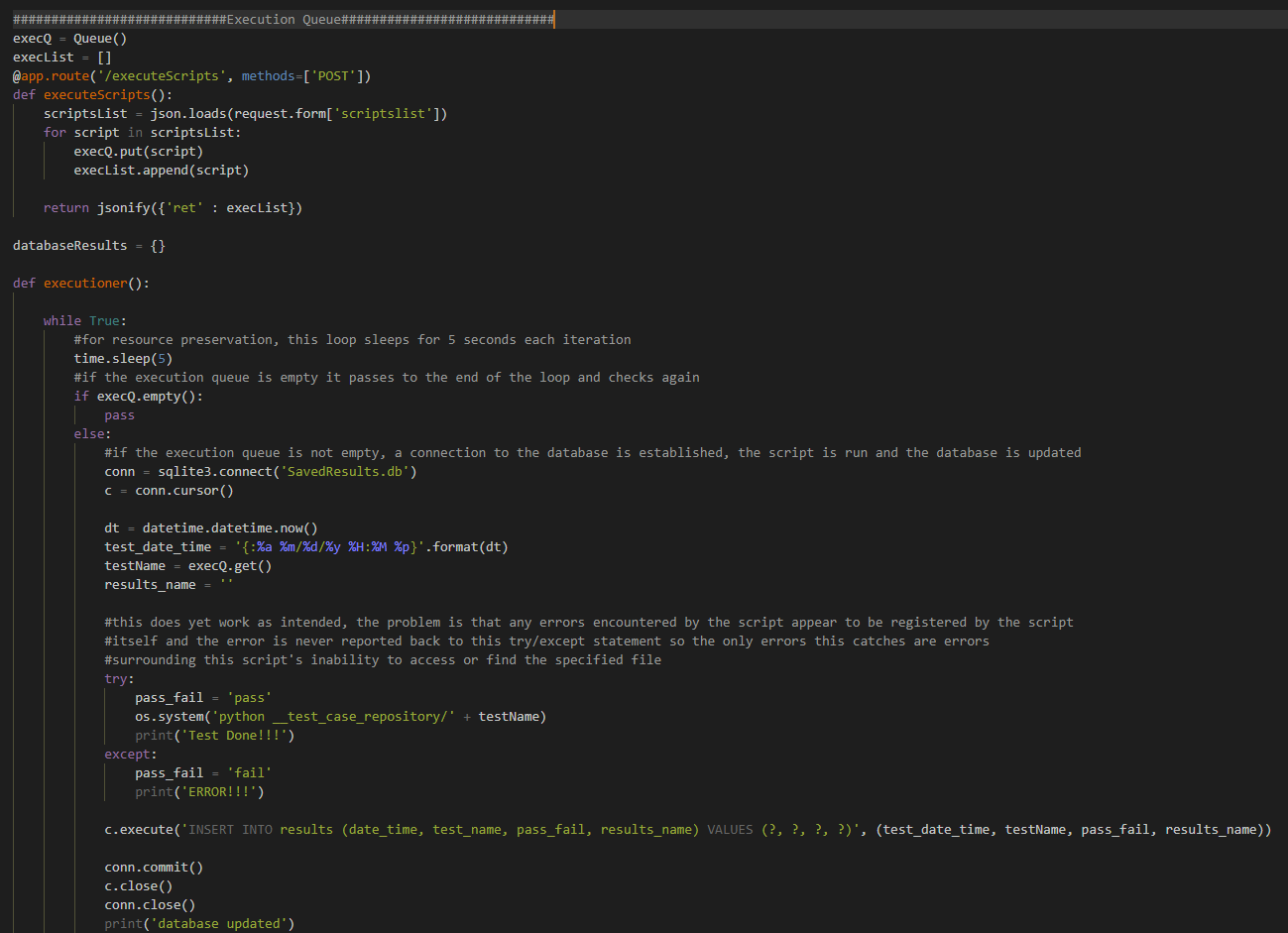
In the event that there are scripts to be run, several things occur. First, a connection is established to the results database (SavedResults.db), the current date/time is captured and formatted appropriately, and the first test case is removed from the queue and stored in a testName variable (this occurs so that the test case name is still available for the database update). The test is run within the context of a try/except statement but this does not work as expected at the moment. Errors in script execution don’t appear to be reported back to the script that called it. At the moment, the try/except statement only works to handle issues when the script set to run is either not found or not accessible (ie. errors local to main.py). After the test is complete, the SavedResults.db is updated and the connection to the database is closed.



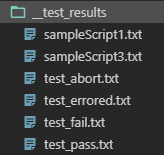
--the JavaScript method that handles execution requests



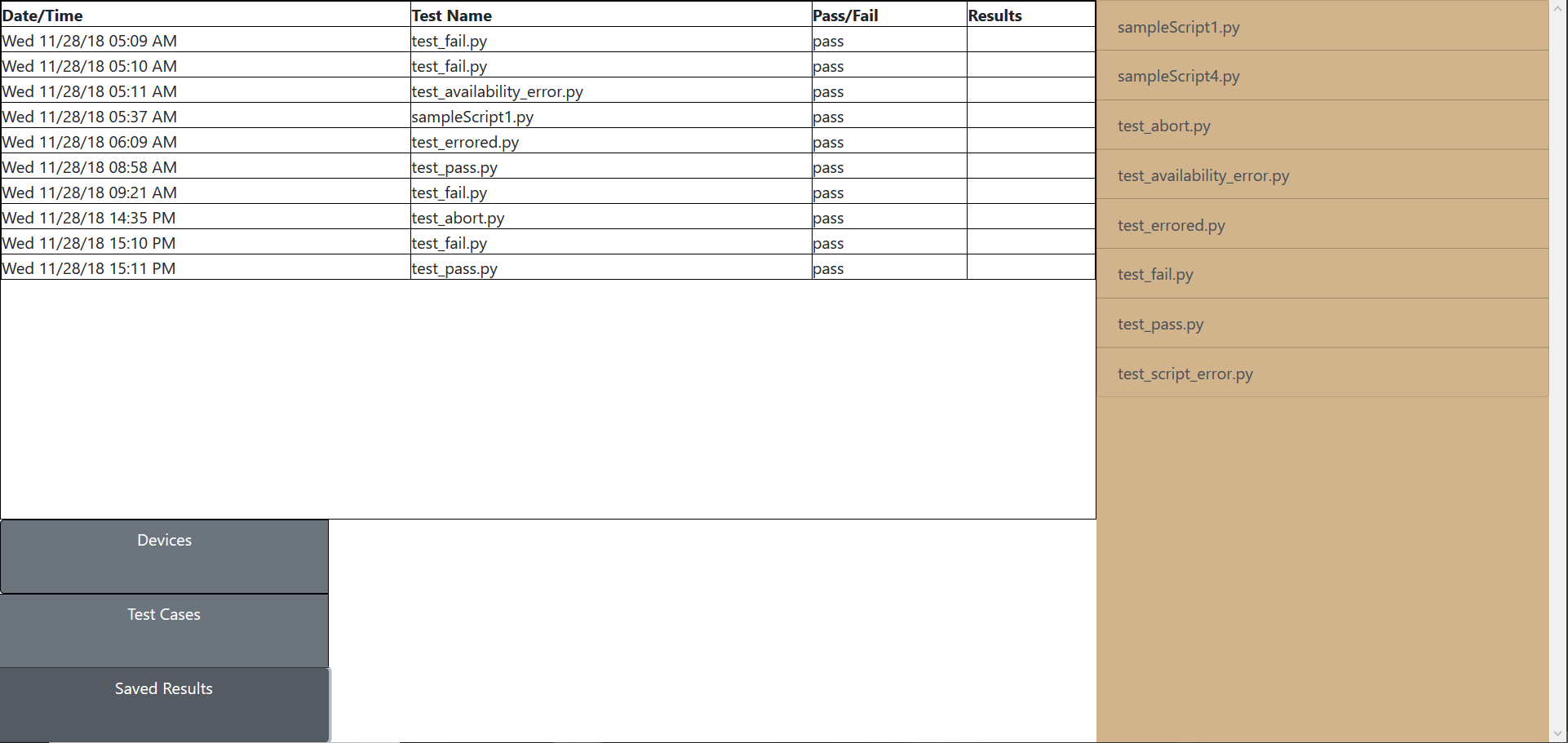
--both the method to handle execution requests and the executioner method, both located in main.py



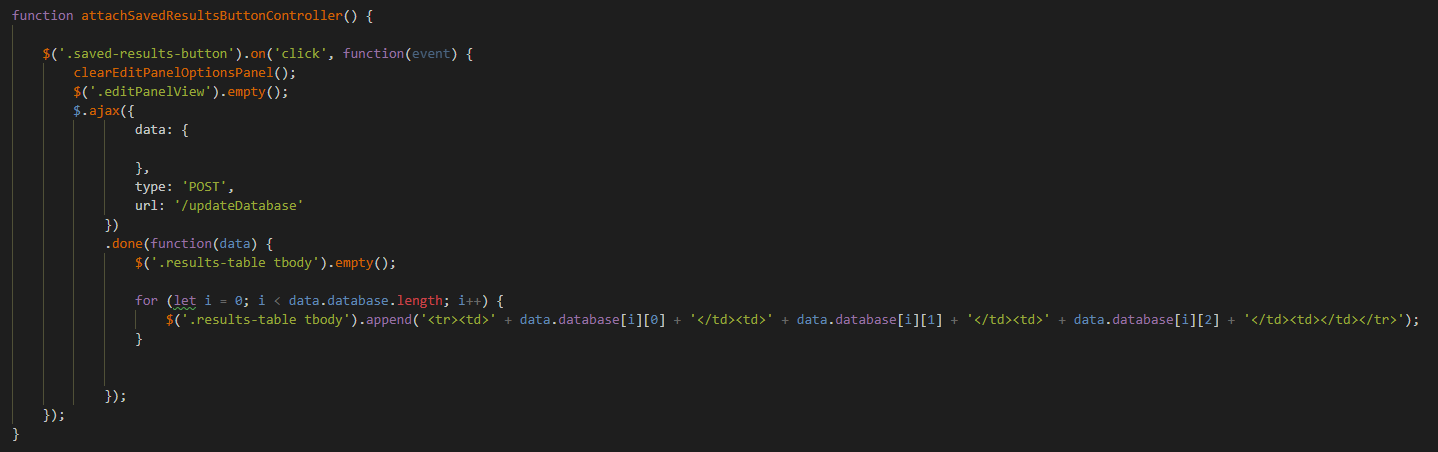
--the test cases have been slightly altered so that the actual results of the test are written to file to be accessed from the local workstation at a later date. The results stored in the SavedResults.db database are very generic and ultimately just capture whether or not the test run was successful (keep in mind, the try/except statement doesn’t currently work as intended so even if the test script ran into a problem, it will still register as a successful run simply because main.py was able to locate and run the script), the date/time of the test, and the name of the test.



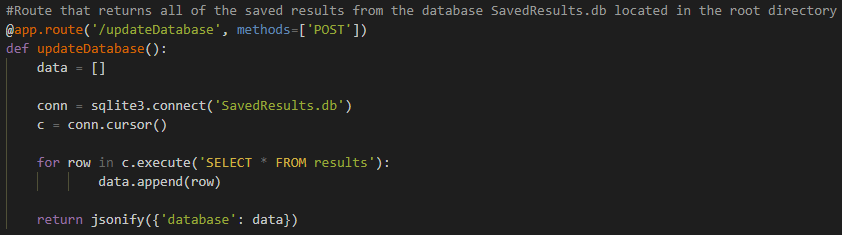
State 7: The Saved Results button is clicked. The editPanel and the editPanelView are cleared (not ideal) and the visible database of saved results is updated. Essentially, an AJAX request is sent to the server but the request itself sends nothing to the server. A connection to the SavedResults.db is established and the database is iterated over and pushed to a Python list. The list is returned to the local workstation where it is iterated over



--the JavaScript method that handles the functionality of the Saved Results button. The .done method empties the visible database (assuming there are row elements to be removed), iterates over the list of saved results and places the contents into a new row, each data point in the correct column.



--the route in main.py that handles the Saved Results request



--If this were a continuing project, the next steps would be:

1. complete all button functionality
2. refactor JavaScript file, separate into individual scripts
3. encrypt all communication between local workstation and server
4. add a login page and create users – also figure out how to allow users to store their own results and sort the results database by their own user ID
5. the fourth column in the saved results table would contain links that would allow the user to retrieve the actual test results