Jacob Mueller

jacobam3

Robbie Krokos

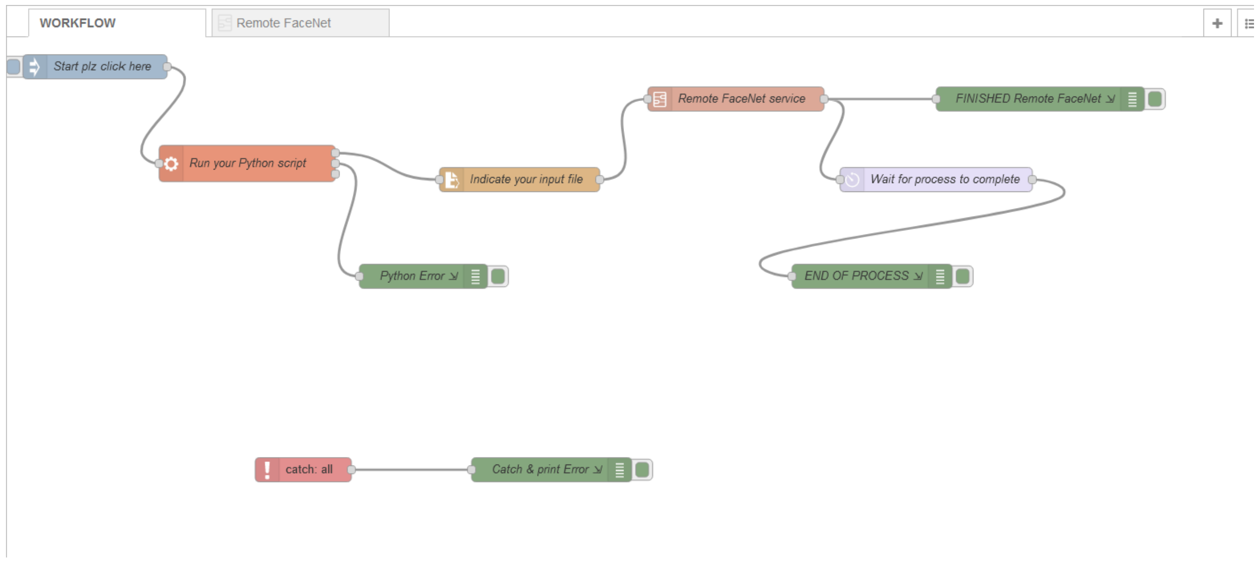
rkroko2

4/30/2021

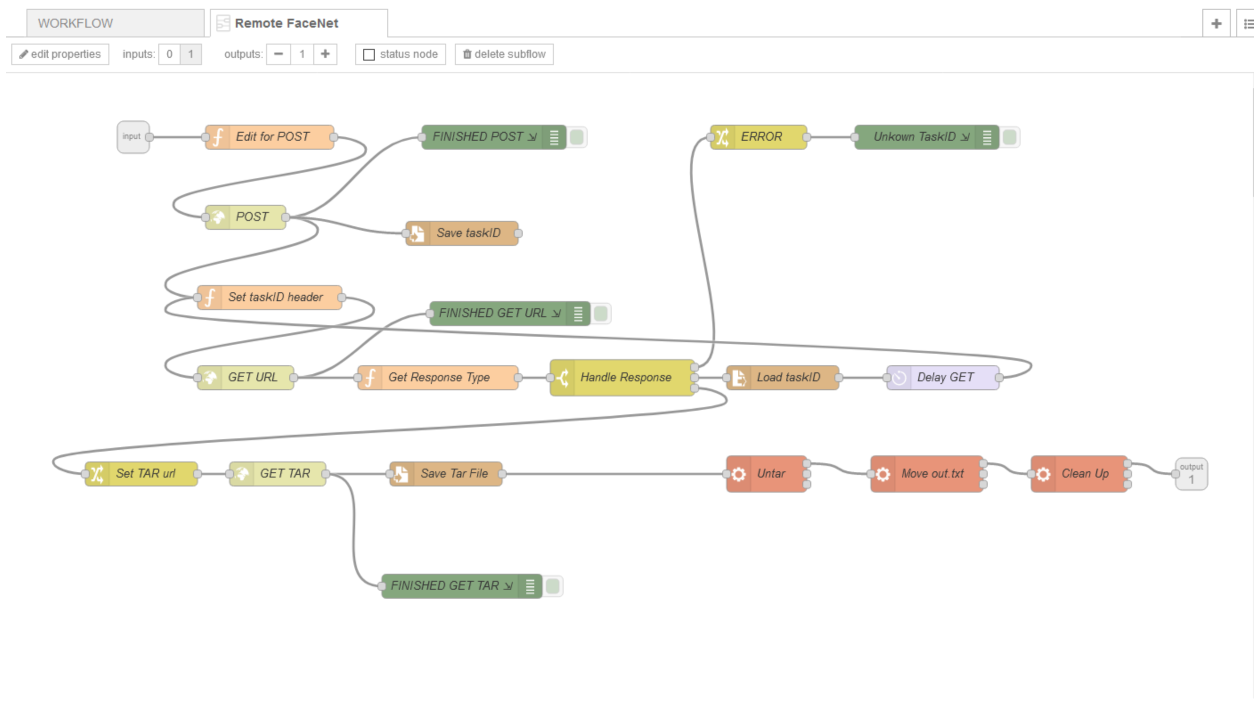
Section IL1

ECE 498 ICC Lab 3 Report

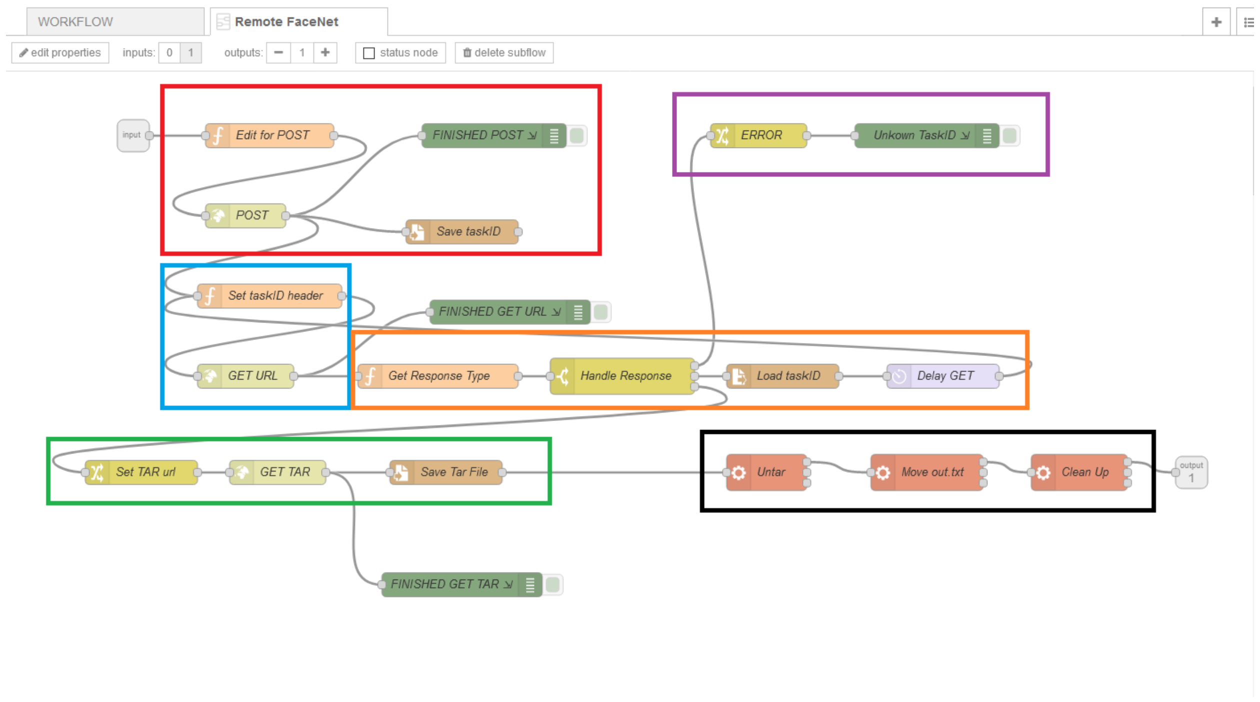
WORKFLOW



Remote FaceNet



Remote FaceNet (Grouped)



The group of nodes in the red box dealt with the POST request. The function node edited the msg passed in to meet the requirements necessary for a successful POST request. It added the Content-type header and many fields to the payload including; file, device_type, & task_type. The request node used the token we generated on the Raspberry Pi earlier to access the proxy server and receive the taskID. The debug node printed the taskID to the debug console. Finally, the file node saved the taskID to a file so that we could use it to query the server again if necessary.

The group of nodes in the blue box dealt with querying the server using a GET request for the URL. The function node placed the taskID into the header in the form of, taskID: msg.payload. Thus, the GET request could function as intended and would query the server for the URL while also using the token generated earlier.

The group of nodes in the orange box dealt with status detection and flow direction. The function node would check what was returned by the GET request node in the previous section and modify the payload accordingly. If the payload was "UNKNOWN", it would reset the payload to say "ERROR." If the payload was "SUBMITTED", it would reset the payload to say, "WAIT" and similarly if the payload said "RUNNING" or "PENDING" it would also reset it to say, "WAIT." Finally, if the payload was none of the above, it would not modify it as it would be the URL and just returned the URL. The switch node would then check for "ERROR", "WAIT", or otherwise and reroute the flow accordingly. If it was "ERROR", the flow would be rerouted to the purple group. If it said "WAIT", the flow would be rerouted to load the taskID again. If it was the URL requested, it would continue down to the green group to download the tar file. The next file in node loads the taskID saved earlier from the file into msg.payload. And the final node waits 2 seconds as to not query the server too often.

The group of nodes in the purple box dealt with the error incase of an unknown taskID. If the flow was rerouted to this group of nodes, the change node would modify the payload to say, "ERROR: taskID is not valid!" This would then be printed to the debug console using the debug node.

The group of nodes in the green box dealt with downloading the tar file using a GET request. The change node modified the payload to be the msg.url returned from the GET request in the previous section. The GET request node retrieved the tar file and returned it as a binary buffer. This was then downloaded to the Raspberry Pi using the file node.

The final group of nodes in the black box dealt with executing functions on the Raspberry Pi. The first execute node moved to the correct directory and untarred the tar file. The next execute node moved into the new "build" directory and moved out.txt to /home/pi. The final execute node cleaned up the Pi by deleting the new "build" directory. So the out.txt file could now be accessed in the home directory and we printed our output to the terminal using, "grep 'netid' out.txt".

Challenges Encountered:

1. Communication Between Nodes & JavaScript/JSON:
   Neither of us has had much experience with working with JavaScript and modifying JSONs, and additionally neither of us has used Node-RED before. When first starting on the lab, we struggled with understanding how the nodes pass information among one another, and how to actually modify, retrieve, and use this information properly. Initially we focused on figuring out what was in the messages by using debug nodes to print things as well as looking at documentation online to piece things together. Additionally, we both researched some basic JavaScript syntax and how to modify/create JSONs. After a bit of time, things made more sense and we were able to construct the flow.

2. Python Script from Lab2:
   At the start of the project, we focused on modifying our Lab 2 python script to fit the requirements for Lab 3. We removed some unnecessary functions and the ML models, but we ran into some issues with capturing and saving the images properly as well as forgetting to remove a 'show image' function that caused our flow to break. When capturing and saving the images, we realized that the images had a purple shade to them and couldn't understand why. After some research, we realized the function we were using to save the image expected a different color format than what we were passing in; fixing this allowed for the inferences made to be much more accurate. Also, when running the flow for the first couple times, we noticed that it kept getting stuck on the exec node that ran the python script. After adding debug nodes and running the script on the raspberry pi manually, we realized that we left a line of code in that displayed an image. This blocks execution until the image window has been closed, and removing that code allowed for execution to flow through the model as expected.

3. Untarring the tar.gz files & retrieving its information (exec nodes):
   One other major challenge we encountered was using the exec nodes to untar the tar.gz file retrieved from the GET request. We had trouble figuring out how to get the exec nodes to the proper directory, and then properly untarring the file. We knew where the tar file would be located, but we did not know what working directory the exec node was executing in. However, we then realized that we could use the '&&' operator to perform multiple CLI commands in a single exec node. Another issue with this that we had to figure out was how to actually get to the output.txt file, since the directory that contains it has a non-static name. After some experimentation and testing, we realized that the '*' operator can be used in a 'cd' command to change the directory if there is only one directory present. Therefore, if we deleted the untarred ball after moving the output.txt each time, we would be able to 'cd' into '\input[Date & Time]' since it would be the only directory in the '\build' directory. Thus, we arrived on the solution to have one exec node to untar, one exec node to move output.txt, and one exec node to delete the tar ball (we realized that these could probably be all combined into one exec node with the '&&' CLI operator, but we decided for readability/understandability, it would be best to keep them separate). In order to arrive at this solution just described, we made the assumption that the

information in this tarball (asides from output.txt, which is moved outside of the untarred package) will not be needed later on and can therefore be deleted. However, if they are needed, the flow can simply be modified to move those files/directories to a specified location so that they are not deleted.

Asides from these challenges, we only ran into minor bugs/issues, which mainly stemmed from us not being used to Node-RED and its tools. After some time, we both had a solid understanding of how to use Node-RED and how to debug when things aren't working as expected, and so things went smoother than they had initially.

Conclusion

We learned a lot about Node-RED and its uses in IoT systems. Node-RED is a great tool when it comes to developing IoT software and it makes it very easy to create a flow with features that might be more difficult to implement in a more traditional manner. It was great to get hands-on experience with a GUI tool such as Node-RED and learn how powerful they can be. Also, in Lab 2 we learned a lot about how edge computing and development is done, so it was really interesting to work with a more Cloud computing focused project in Lab 3. Also, it was very cool to see our final product work as intended.