# Overview

The aim of the project was to create a user interface for the user that helps them extract, edit, or delete datasets. This was decided to be done by developing a dashboard that has two sections, the first section is the left side of the dashboard which is a user interface where the user can navigate through the categories to choose any section of the datasets and be able to manage the datasets according to the user role. The other section of the dashboard is the command line part, where there is a command line interface and the user can input the directory of the dataset along with the function like “GET, POST, or DELETE” to create a command that is used to be applied on the datasets. Once the user inputs a full command in the command line part of the dashboard, the dashboard creates an API method or request and displays the result for the user, so for example if the user types “/datasets/education” and selects the DELETE method, the dashboard will send an API request like “curl.exe -X DELETE <http://localhost:5000/datasets/health/RAKCSS-HEALTH02-2022.xlsx>”.

This method directly interacts with the server without the user having to load up the website or deal with the interface to navigate through the datasets or folders manually.

# Creating the dashboard

We used streamlit framework to create the GUI of the dashboard, we tried to make the user interface as user friendly as possible so that its easy for the user to navigate through, since we were using streamlit, we had to start a session so that the dashboard can connect to and run on, this was done by:

if "role" not in st.session\_state:

    st.session\_state.role = None

if "last\_action" not in st.session\_state:

    st.session\_state.last\_action = None

if not st.session\_state.role:

    st.title("🔐 Select Your Role")

    role = st.selectbox("Choose your access level:", list(ROLES.keys()))

    if st.button("Enter Dashboard"):

        st.session\_state.role = role

    st.stop()

after the dashboard connects to the streamlit session, it asks the user the input their role, and this can be selected from a drop-down menu, then the dashboard will take the user to the main screen directly and will give the user the role permissions accordingly.

# Creating the API

The API was created through flask on python. We made the code simple and sectioned, so there is a function part of the code for each method the user can request. So for example, the “upload file” method that can be done through the dashboard takes the base directory for the API and the file from the user then places this file in the directory, here is a demonstration of the upload file function:  
# Upload or update a file

@app.route('/datasets/<category>', methods=['POST'])

def upload\_file(category):

    if 'file' not in request.files:

        abort(400, description="No file part in the request.")

    file = request.files['file']

    if file.filename == '':

        abort(400, description="No selected file.")

    dir\_path = os.path.join(BASE\_DIR, category)

    os.makedirs(dir\_path, exist\_ok=True)

    file.save(os.path.join(dir\_path, file.filename))

    return jsonify({"message": f"File '{file.filename}' uploaded successfully."})

# integrating the API into the dashboard

in order to integrate the API into the dashboard we had to edit the dashboard’s CLI, so that instead of the board just taking a directory, it actually creates an API request. We also relied on the “requests” library of python to allow the dashboard to send an HTTP request when the user enters a directory and a method, so for example if the user selects GET and enters “/datasets/health/xyz.xlsx”, the dashboard sends an HTTP request to the Flask API using the requests library and would be coded by “requests.get()” function. An example from the code here shows the how the dashboard converts the dashboard function into an HTTP request:

elif action == "GET":

                    try:

                        response = requests.get(f"{API\_URL}/datasets/{cat}")