Building a new project

Setting up: \* assumes that backend has been built with no front-end html page

Note: All of the following is simply how the project is originally formatted and if anyone is building a new project then they can format it a completely new and better way if they so wish. The following is just one way to keep each project and web app modular and separate.

1. To set up for a new project a few things must be done. First off gather a list of all external dependencies (any libraries used and the such) used by backend programs. Then list all inputs and outputs of your project, meaning that if your project intakes 10 lists and 5 different numbers and outputs a graph then that should be well documented.

Note: for simplicity sake it would be good to write a single function that you could pass in all the data needed for your project and then return all necessary data.

1. A webpage front end must be built in order to display your project and intake date from any users. This requires a functioning HTML page that has a CSS stylesheet. The HTML page doesn’t need to have any forms put on it at this point, or graphs, just the barebones general idea.
2. Once the backend (python files) and front end (webpage) have been set up, they need to be connected. First thing first, the route (route refers to URL) must set up. In application.py create a decorator and function with the following format:

@app.route(‘/name’, methods=['POST', 'GET']) # Decorator

def Example\_Project\_page(): # Function name

return example\_project\_form() # Return the form function

The decorator should have url extension in it, for example the lacOp url extension is /lacop and the whole decorator looks like:

@app.route('/lacop', methods=['POST', 'GET'])

Next the function just needs to return a form function (which will be apparent why later) which looks like:

def Example\_Project\_page(): # Function name

return example\_project\_form() # Return the form function

1. Now that the URL path has been set up in the main application file, a project specific application file must be made, with a form function that can be called and returned by the functions described above.
   1. The project specific app file should always import the following:

from flask import Flask, render\_template, request, url\_for

* 1. Next you need to import your project specific files as well.

Example from fase: from fase import main\_fase

* 1. Once you have imported correct project files and flask functions then you need to write the form function. The form function will handle all form data from the webpage and project data and is the true connection between the back end scripts and the front end webpage. Currently the from function needs to return a flask function called render\_template which has the following format:

render\_template('webpage.html', data = data)

The html webpage is the first argument in the function, and any data you have will be passed in as additional arguments. The render\_template function can be passed in as many data arguments as wanted, but be wary of passing in dictionaries or lists because they require more work on the webpage side to unpack the data (making jinja macros if you wanted proper format).

* 1. A general pseudo code version of a form function looks like:

Def project\_form( ):

Check for user input

Collect data from user

Send and receive data to backend

.

.

.

return render\_template(‘foo.html’, data = someValues)

A example of a complete (but barebones) project\_app file. Lets call our new project foo.

from flask import Flask, render\_template, request, url\_for

from foo import foo\_do\_stuff

def foo\_form():

if request.method = ‘POST’:

data = foo\_do\_stuff(request.form.get(‘user data’))

return render\_template(‘foo.html’, data = data)

else:

return render\_template(‘foo.html’ , data = ‘ ’)

Once the project app file has been made it can now be imported by the main application and the form function can be called within the application page.