

# Lab 5

## Web Programming I

In this lab you will practice basic HTML, CSS, and Javascript using CodePen. We will create a simple web API using Azure App Service.

### Step 0.0: Get a CodePen account and create a new "pen"

If you do not have one already, go to <https://codepen.io/> and create a user account. Once your email is confirmed log in and explore the website.

### Step 0.1: Set up your initial environment for Azure/Python

- Create an Azure account with an active subscription. [Create an account for free.](#)
- Install [Python 3.6 or higher](#).
- Install the [Azure CLI](#) 2.0.80 or higher, with which you run commands in any shell to provision and configure Azure resources.

Open a terminal window and check your Python version is 3.6 or higher:

```
python --version
```

Check that your Azure CLI version is 2.0.80 or higher:

```
az --version
```

Then sign in to Azure through the CLI:

```
az login
```



## Step 1: Clone example

technin510au21

[Home](#)[Calendar](#)[Resources](#)[Project](#)[Labs](#)

Clone the sample repository using the following command and navigate into the sample folder. ([Install git](#) if you don't have git already.)

```
git clone https://github.com/wesleybeckner/myImageClassifier.git
```

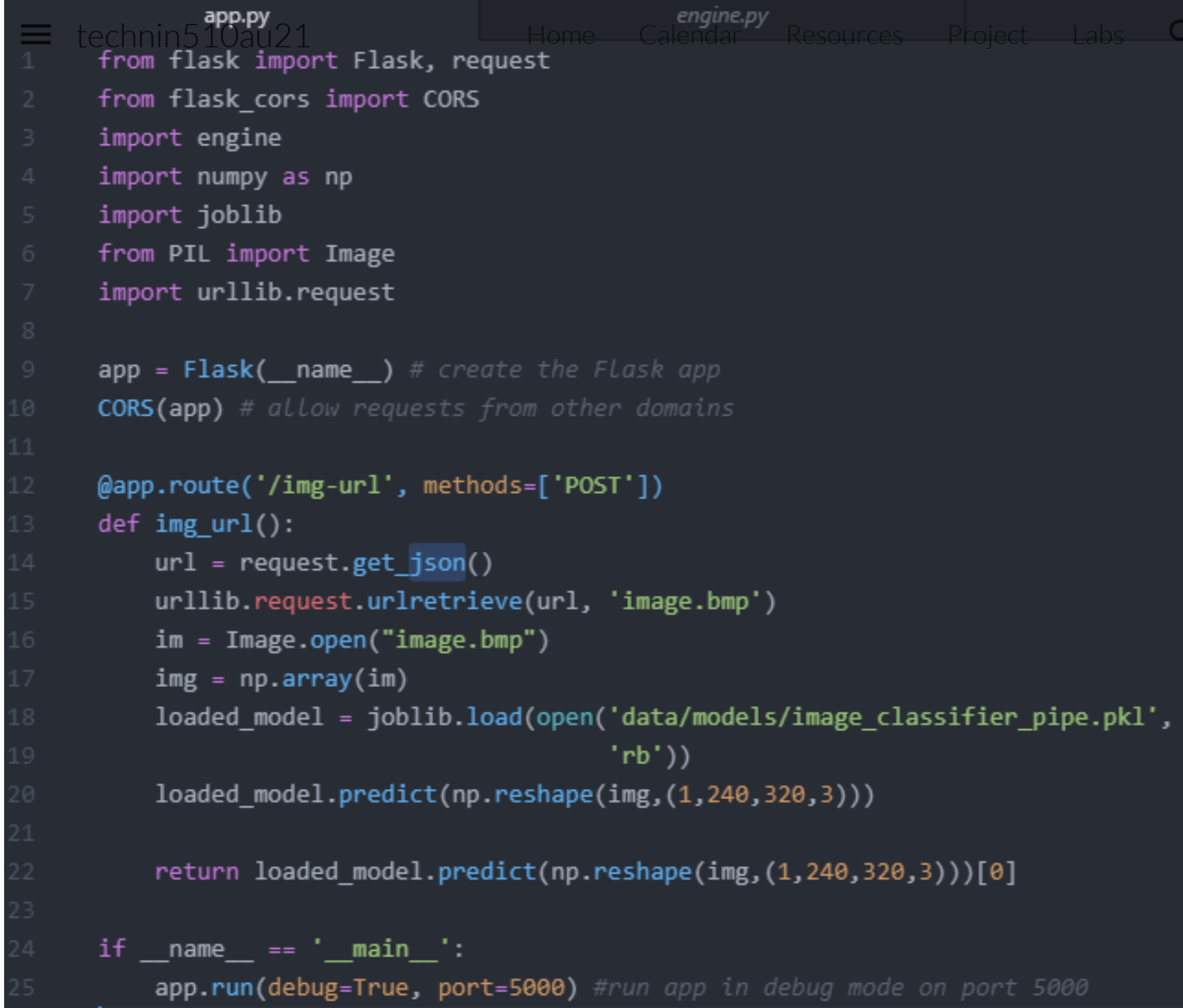
Then navigate to the folder:

```
cd myImageClassifier/myimageclassifier/
```

This is a good opportunity to get familiar with the command line if you haven't already. The example contains a trained classifier from lab4, stored as a pickle file under data/models/. It also contains the train and test data used from lab4. We will be using GitHub as a "database" for our image files for our application (typically we reserve GitHub for source code).

There are two main python files, `app.py` and `engine.py`. For python servers we almost always call our main application file `app.py`. This file contains our Flask application that will be used to create our python server. `engine.py` contains the classes and methods that describe our machine learning model. `a` imports this file so that it can read and understand the stored pickle model.





```
1  from flask import Flask, request
2  from flask_cors import CORS
3  import engine
4  import numpy as np
5  import joblib
6  from PIL import Image
7  import urllib.request
8
9  app = Flask(__name__) # create the Flask app
10 CORS(app) # allow requests from other domains
11
12 @app.route('/img-url', methods=['POST'])
13 def img_url():
14     url = request.get_json()
15     urllib.request.urlretrieve(url, 'image.bmp')
16     im = Image.open("image.bmp")
17     img = np.array(im)
18     loaded_model = joblib.load(open('data/models/image_classifier_pipe.pkl',
19                                     'rb'))
20     loaded_model.predict(np.reshape(img,(1,240,320,3)))
21
22     return loaded_model.predict(np.reshape(img,(1,240,320,3)))[0]
23
24 if __name__ == '__main__':
25     app.run(debug=True, port=5000) #run app in debug mode on port 5000
```

app.py contains a route that receives a post request containing an image url. This url is used to grab an image and run it through our trained model, image\_classifier\_pipe.pkl.

## Step 2: Run the sample

technin510au21

[Home](#)[Calendar](#)[Resources](#)[Project](#)[Labs](#)

- Make sure you're in the `myImageClassifier/myimageclassifier` folder.
- Create a virtual environment and install dependencies:

```
python -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

If you encounter "[Errno 2] No such file or directory: 'requirements.txt'.", make sure you're in the `myImageClassifier/myimageclassifier` folder.

Run the development server.

```
flask run
```

```
Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
Debug mode: off
Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

output from flask run.

The server assumes that the app's entry module is in `app.py`, as used in the example. (If you were to use a different module name, set the `FLASK_APP` environment variable to that name.)

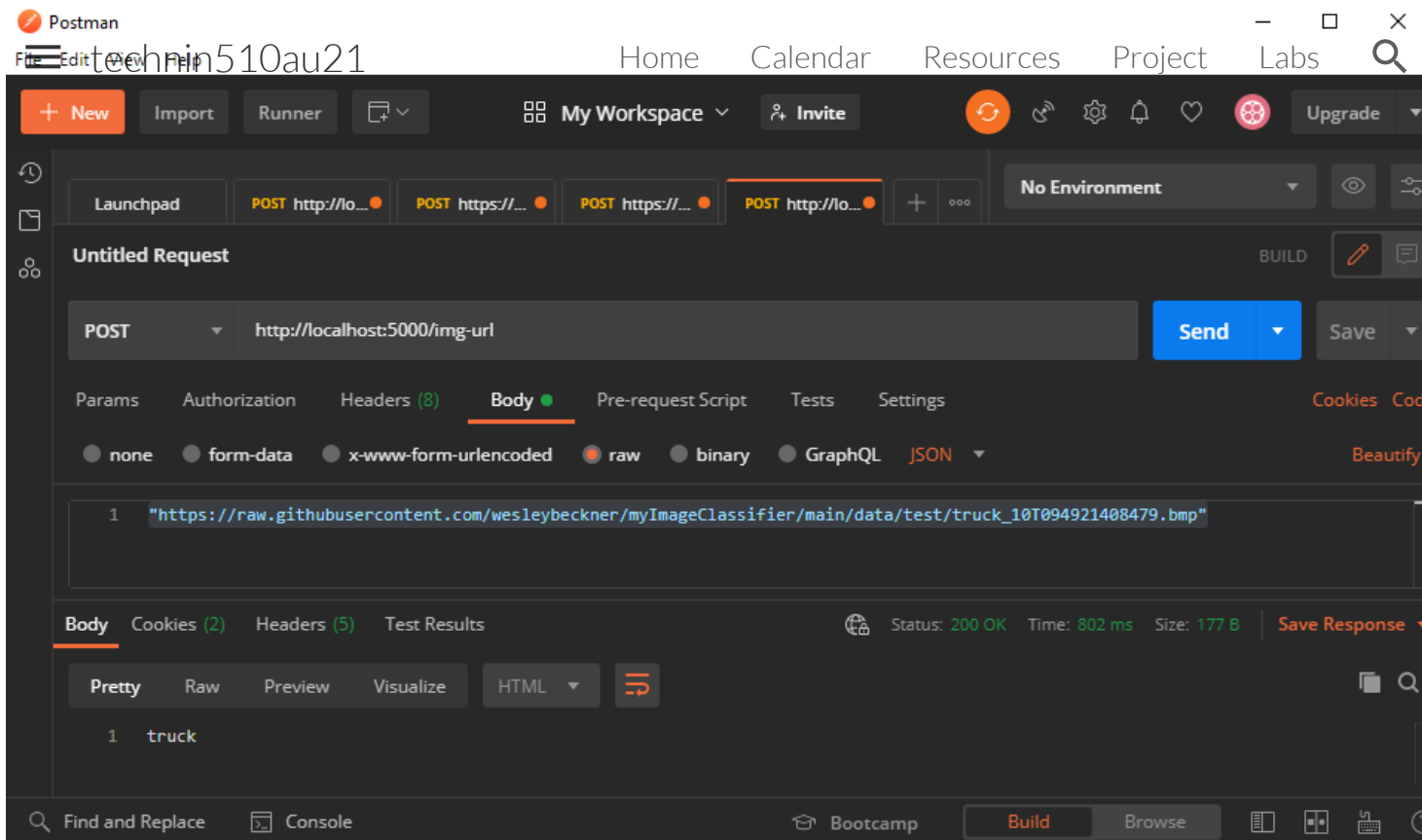
Install [postman](#) if you do not already have it.

Open postman. create a new request, set the type to POST. Add into the url address the local flask application followed by 'img-url': `http://localhost:5000/img-url`

Under body, select the 'raw' radio button and the input type to JSON. Paste the url of one of the images on the github website to send as part of the post request:

```
"https://raw.githubusercontent.com/wesleybeckner/myImageClassifier/main/myimageclassifier/data/test/truck_10T094921408479.bmp" Hit Send.
```





In postman, after you submit the post request to your local flask application with a valid url from the dataset, you should see a label returned, in this case "truck".

### Step 3: Deploy the sample

Deploy the code in your local folder (*myImageClassifier/myimageclassifier*) using the az webapp up command:

```
az webapp up --sku F1 --name <app-name>
```

- If the az command isn't recognized, be sure you have the Azure CLI installed as described in [Set up your initial environment](#).
- If the webapp command isn't recognized, because that your Azure CLI version is 2.0.80 or higher. If not, [install the latest version](#).
- Replace <app\_name> with a name that's unique across Azure.
- The --sku F1 argument creates the web app on the Free pricing tier.

It takes a few moments to start the app initially. After a minute, go back to Postman and replace the local url with the azure url. Make sure the request gets a response.

The Python sample code is running a Linux container in App Service using a built-in image.



```
Webapp 'myImageClassifier' already exists. The command will deploy contents to the existing app.
Creating zip with contents of dir /mnt/e/Dropbox/work/gix/510/myImageClassifier ...
Getting scm site credentials for zip deployment
Starting zip deployment. This operation can take a while to complete ...
Deployment endpoint responded with status code 202
You can launch the app at http://myimageclassifier.azurewebsites.net
{
  "URL": "http://myimageclassifier.azurewebsites.net",
  "appserviceplan": "wab665_asp_Linux_centralus_0",
  "location": "centralus",
  "name": "myImageClassifier",
  "os": "Linux",
  "resourcegroup": "wab665_rg_Linux_centralus",
  "runtime_version": "python|3.7",
  "runtime_version_detected": "-",
  "sku": "FREE",
  "src_path": "//mnt//e//Dropbox//work//gix//510//myImageClassifier"
}
```

You should get a response like the one shown here, except that it will be a new application and not one recognized.

## Step 4: Integrate with CodePen

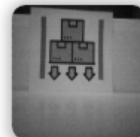
go to the [example on CodePen](#) and copy the HTML/JS/CSS into your own CodePen. Be sure to replicate the settings options as well to properly generate the application.

In this application, whenever the user clicks on a new image, a request is sent out to the python application on azure. The server takes the received image url and runs it through the machine learning model, it then returns the image label.

For the remainder of step 4 do the following:

- Change the url for the server to your own in the JS pen.
- Change the JS so that the sequencing (the ordering) of the images as they first appear on the page is random every time the page is booted





inspection

## Optional:

Replace engine.py with your own code. This should include all the classes used to define your model. You will also need to pickle your trained model so that it can be loaded by app.py.

- When you do this, you need to make sure the same class definitions that were used to create the model are imported when you unpickle the model. Otherwise you will get an attribute error.
- You will also need to include any relevant libraries in the requirements.txt file

Replace the CodePen application with a front end of your choosing. You can show model training results, display plots, use other images, etc. Be creative, have fun. Does this give you ideas for how to create your application architecture for the class project?

## Step 5:

Complete this lab by submitting public links to the CodePen pen on Canvas, by Nov 3 Tuesday, 11:59pm. We will test your code by opening the CodePen and interacting with the rendered page to make sure:

- The page is sending requests to your own azure server
- Your dynamically generated images are randomly ordered every time the page is loaded

We will inspect code as needed. See Canvas for a grading rubric.

