# *Web Development II (ITAS256)*

# *Assignment 2 – Flask or Bottle Server*

Date assigned: February 29, 2023

Date due: **March 18, 2023**

**Learning Objectives**

Upon successful completion of this assignment, the student will be able to:

* Create a functional web server using either the Bottle or Flask python library;
* Access and update data using routes;
* Use modules such as json and others in a web application.

To do:

**General idea of the assignment:**

Create a web server using either the bottle or flask library. The server will act as the server for a pizza delivery company (very rudimentary) and allow them to perform CRUD operations for pizza orders. I RECOMMEND STRONGLY READING THE ENTIRE ASSIGNMENT AND SPENDING SOME TIME ARCHITECTING IT. THIS CAN BE DONE WITH YOUR CLASSMATES TO MAKE SURE EVERYONE UNDERSTANDS THE ASSIGNMENT.

**Details:**

You are creating a web server using either the flask or bottle library (dealer’s choice) and other libraries that you might find useful. There is to be NO React, Angular or any other front-end framework allowed. Wax-on, wax-off.

1. You will need the following folder structure for your app. You will not be branching off an existing github repo this time, but MUST create your own. Create your own repo called username-256A02 and share it with me. Make sure you set up your README file for your repo.

A screenshot of a computer

Description automatically generated

There may be other folders under those folders. The folders are used as follows:

1. *Username-*256A02 – This is where your server files go and any other classes or imported files you write would go.
2. *data* – where the data file (pizzaorders.json) is kept and the initialization file (init.json) for the pizza parameters go. It also contains the users.json file for the list of valid users. More about these files below.
3. *Static –* whereany static files such as CSS, JavaScript or static HTML pages go
4. *templates or views* – This will be one name or the other (NOT BOTH) depending on the library you use. With bottle it is called views and with flask it is called templates. IT IS NEVER BOTH.

The initialization file is called init.json and contains the possible values for the type, crust and size of the pizzas that can be order. You MUST use this file to create these values. You CANNOT hard code the values in the file. The file is a simple JSON record that looks like this:

A white background with red and blue text

Description automatically generated

Note, I will be adding a few values to each list to make sure that you are reading from this file when creating your form.

The data file you will be using is a json file called pizzaorders.json and is located in the data folder (or will be when you create it). The format of each record is:  
{  
 “id”: an integer a number which is the “key-value” for access the order record,  
 “type”: one of the type values from the init file,  
 “crust”: one of crust values from the init file,   
 “size”: one of size values from the init file,  
 “quantity”: an integer,  
 “price\_per”: a floating point number (price per pizza),  
 “order\_date”: a string of the date the order was placed in the format yyyy/mm/dd  
}

The users.json file contains a list of users that can log into the system. It is an array of JSON records. The format of each record is:  
{  
 “email”: an email address,  
 “password”: a password (no formatting or minimal requirements unless you want to)  
 “role”: ‘s’ for staff or ‘c’ for customer   
}

Your assignment must be in the form of a web server application/package using templates where ever possible. You should use WTForms for all forms (yes it works with bottle too). You must format your pages; you can use local CSS or Tailwinds (if you want to use a different library, please check with me first).

You will create a web server that runs on port 8888. Following is the functionality of that web server.

The web server must function as follows:

1. A /login route that returns a login page that validates the user login. Check the login from a JSON file you create in the data folder. The users.json file should be a JSON array of users with the fields: email, password and role as described above.
   1. When called with GET it displays the login screen which contains an email field, a password field and a submit button in a form. The form should call the /login route with the POST method.
   2. When called with POST the /login route validates the form information against the users.json file.
   3. A successful login must set either two session variables or two temporary cookies called login with the value of the user email and role with the value of the role. An unsuccessful login returns to the /login route with the GET method.
   4. You must provide a ‘Create account’ link from the login page to create a new account. This will call the /create route.
2. Create a /create route that, when called with the GET method, displays a blank form to enter the new account information. This form must be created using a WTForm class that is fed to a template. It includes an email field, two password fields, a radio button for role (staff or customer) and a submit button. All fields must have appropriately defined labels (including the submit button). The form should call the /create route with the POST method.

1. When the /create route is called with the POST method, validate that the password fields both have the same value. You do NOT have to check that the user already has an account but can if you like. If the user information is valid, add the user to the users.json file and
2. Every route except the /create route MUST check that the user is logged on before displaying. If there is no user logged in the user must be redirected to the login page.
3. The header of every page must contain a link to logout of the system. The logout link calls the /logout route. The /logout route removes the login and role cookie or session variable and redirects the user to the login page.
4. The “main” page (route /) displays a sorted list of all the pizzas in the data file sorted (descending) by date ordered (that is, the most recent on top). The list should be nicely formatted and must use a template to display. All of the fields in the pizza order EXCEPT THE id FIELD must be displayed. As well, the subtotal cost (quantity \* price\_per), a 10% delivery charge and a total cost for each order. Even thought he id field is not displayed it still must be part of the record (you can use a hidden field or an attribute if you like).
5. The main page contains a link or button or something that allows the user to order a new pizza. This link goes to the /pizza route.
6. When the /pizza route is called with the “GET” method display the form to allow the user to enter a new pizza order (the form is described below). The form must be created using a WTForm class and a template. NOTE: below you are going to add functionality to the corresponding /pizza route function to pass in an existing pizza record. You can read ahead and do it together or do it separately now (I recommend the latter).
7. The form contains the following fields:
   1. A select drop down for the type, crust and size fields.
   2. An integer field for the quantity (between 1 and 10)
   3. A floating point field for the price per pizza
   4. A date field for the date of the order.   
      **(NOTE: THE ID IS NOT PART OF THE FORM)**
8. When the /pizza route is called with the POST method create a dictionary object of the form data and append the data to the pizzaorders.json file. You will need to generate the <id> value as the largest value in the file + 1. Be careful how you add the record because you will have to add the record to the JSON array and not just append the record at the end as it will not work properly as part of the file then. This is the same thing we did in labs 4 and 5. Once the record is added, redirect the user to the main page.
9. Back on the main page (/), you must provide a way for the user to select a specific order to view and modify it. When the user selects a specific order (this can be through a pencil or edit icon, clicking the order on the screen or however you want), call the /pizza route with the GET method. In this case pass the record (I would recommend using a dictionary) to the /pizza route function.

In this case, /pizza route is called with the GET this only difference is that the record is displayed and the form uses the PUT method by using JavaScript and, likely, the fetch method.

1. When the /pizza route is called with the PUT method, update the corresponding record in the pizzaorders.json file and redirect the user to the main page (/)

To update, read in the file and go through the records until you find the matching record (check the id value) and update that record before writing it to the file. This implies that, although it is not displayed, the id needs to be part of the record. That will also be the case for the POST, but the id will be blank or 0 or something and then get changed to the proper value as above.

1. Back on the main page (/), you must provide a way for the user to select a specific order to delete it. When the user selects a specific order (this can be through a garbage can icon, clicking the order on the screen or however you want), call the /confirm route with the GET method and pass the pizza record.
2. Add a /confirm route. When called with a GET method this page displays the pizza order sent to it as well as a form with a label saying Confirm and a button for Yes and No. If the user says No, simply call the main page (/) without doing anything.
3. If the user confirms the deletion, call the /pizza route so that the pizza will be deleted. This is, again, likely to be an AJAX fetch call. Update the /pizza route to do the following when a DELETE method is received.
   1. Remove the record from the pizzaorders.json file. Do this by looping through the records in the file and copy them to a new dictionary list, skipping the one with the matching id. Then write the new list to the file.
   2. After this redirect the user to the main page (/)
4. If you wish to split the pizza route into two routes, feel free to do so. It may make processing the commands easier. For example, /pizza and /pizzado or something like that. It is likely easier this way.
5. You will likely have to have CORS set up but may not. If you are getting 405 errors when POST or PUT or DELETE is used, that is likely a CORS problem.
6. Remember, any static files, including JavaScript, go in the static folder.
7. There are many ways to complete this assignment. Some are better than others. Try and use OOP principles and make sure that you call functions to do the work (for example, the /pizza route should not be a lot of code, but should call functions to do the work and make the code readable). Be pythonic and use design. I STRONGLY recommend sitting and architecting your solution before plunging headlong into the morass.

**To submit**

Create a githib repo with the name <<username>>-256A02 and share it with me. There is nothing to hand in to the portal for the assignment (Yes I know this is different, but it’s a larger project and probably something you will want to have in your personal portfolio).