# CS-1520 Fall 2023 Final Project

## 1 Introduction

For this CS-1520 final project, you will create an Flask server api that mimics an order system that can be found in a fast food store. Your system shall have 2 distinct parts: one that interacts with the customer, and a second that provides information of the order for someone in the kitchen (more details later on).

For the first part (the customer side), your application shall:

- Allow the user to select food items from an on-screen menu.
- Show an order summary in real-time (meaning this order summary is updated constantly as the customer makes a new selection).
- Allow the user to enter his/her name into the order.
- Provide a final order summary page.

For the second part (the kitchen side), your application shall:

- Provide a screen (page) that shows the current orders not yet processed.
- Provide a button (or a link) that deletes this order when it has been delivered to the customer.

This project does not need to have the Responsive Web Application capability since the customer interaction is always from a big monitor found in the establishment.

This project touches the following topics discussed in CS-1520 class:

- HTML
- CSS
- JavaScript
- DOM
- HTTP requests
- Python Classes and objects
- Flask routes
- Flask Templates
- Flask Models
- Pooling
- Database CRUD operations
- Jinja2 commands

## 2 Detailed description of the project parts

### 2.1 Customer side

In the customer side, your application may display 2 different pages to the customer:

- The main menu
- The final order summary

In the next subsections a detailed description of each page is presented:

#### 2.1.1 The main menu

When the customer arrives into the establishment, the monitor shall present the main menu as shown in Figure 1 in the next page.


**Figure 1: Main menu – customer side**

This main page has the following features that you must implement in your project:

a) A main title on top of the page.
b) Some words about the store itself.
c) The page is divided into two sections: the left and right sections.
d) The left section has the following features:
    i) A section title ("Main Menu")
    ii) A menu for each group of items sold in the store, in this specific project the "Burgers & Fries", "Drinks", "Coffees", and "Desserts". If the customer clicks on one of those options, it will make the page to jump to the corresponding section.
    iii) Each section, such as Burgers & Fries, will have 4 items. Each item has a picture, an item description and an input text field allowing the user to enter how many units of that given item they want to buy.
    iv) As the user changes the order, the order summary shall be updated.
e) The right section has the following features:
    i) A title (Order Summary)
    ii) A textarea that will list all the items selected by the customer, which includes the quantity of the items, the item description, and the total value for that given item (3 hot fudge sundaes will cost 3*$2.50 = $7.50, as shown in the Figure 1.
    iii) A label and its respective text input where the customer can write his/her name.
    iv) A submit order button. By clicking this button, your application shall take the customer to another page (another route: order_summary) as discussed in the next section.

### 2.1.2 The Order Summary Page

An order summary page shall be displayed by your application when the customer clicks on the Submit Order button shown in Figure 1.

An example of order summary is shown in Figure 2 below. This page has the following features that you must implement in your project:

- It shall be in the route "order_summary".
- It shall contain the title Order Summary.
- It shall handle following data that comes from the main page:
    - Order Number (a counter that increases by one for every new order)
    - The customer's name.
    - An unordered list of items chosen by the customer.

Your application shall show this page for about 10 seconds and then redirect the browser to the main page again, for the next customer.

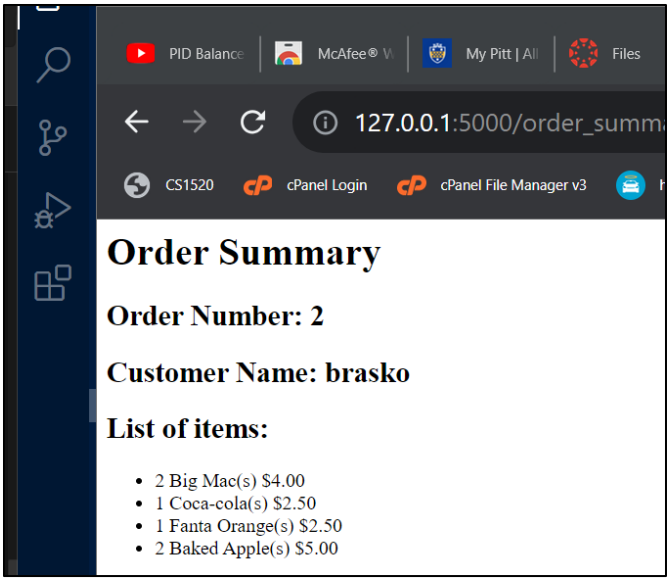


**Figure 2: The Order Summary Page**

## 2.2 Kitchen Side

Your application shall have a route named "kitchen". This route is used only by the staff in the kitchen, where a monitor is placed showing the current orders. Figure 3 shows how your application shall show this page.

The main features of the kitchen page are:

- A main title
- A list of orders with the following features:
    - Each order shall contain:
        - The order number (in my initial project I forgot to update the order number, but please make it right in your program).
        - The customer name.
        - An unordered list of items that the customer has selected.
        - A delete button or link (if link is used, use CSS to make it look like a button, like I did in mine project and shown in Figure 3). When the order is delivered, the kitchen staff will press this button to delete it from the page.
- Your application shall automatically refresh this screen every 15 seconds, so if a customer enters a new order, this order will show up in the kitchen page shortly after. The kitchen staff do not need to refresh the screen him/herself to get the new orders.

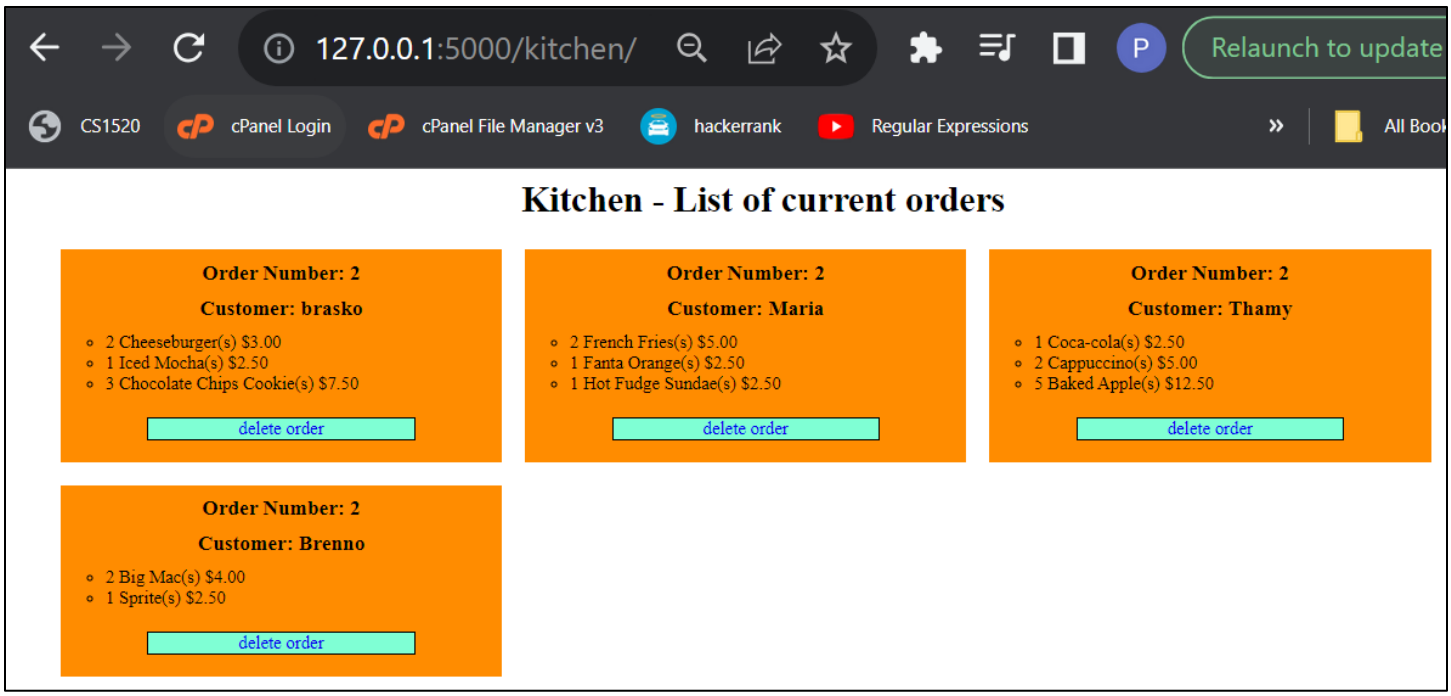


**Figure 3: The Kitchen Page**

# 3 General Suggestions on how to create your project

As I have mentioned in class over and over again, in big projects like this one, try to implement and test one feature at a time. Do not try to create all the pages, all the functionality, formatting, etc. in one single shot and then try to run/debug your application! It will be very difficult to figure out what is wrong by analyzing dozens of files with lots of lines of code. Implement one feature and then test it. If something is wrong, you know that it is something in that small block of code. After fixing all the issues in this single implementation, then go for the next implementation and repeat the run/debug again.

Below I am providing a kind of Step-by-step instructions on how you could work around in this project. You can following it or not, it is up to you.

As I have mentioned in class, do not use chatgpt to solve this project. It is easy for me to spot if you have used it or not and you will get a zero if you do so.

## 3.1 Create a virtual environment

Any project should have a virtual environment, so, create one for your project. Activate your virtual environment. Finally, install flask and flask_sqlalchemy libraries into your project.

## 3.2 Your program is a Flask application

Remember your program is a Flask application, so go ahead and create a Flask application with a single route that prints Hello World in the browser. Test your progress.

## 3.3 Save the images into your project

All the images that you will need for this project have been provided to you already (available in the course Canvas). Please copy the images in the “static/images” directory in your project.

## 3.4 Create your main page

To create your main page html file (such as index.html), remember that you need to create it in the "templates" directory in your project. Place a "Hello World 2" in this main page just for debugging purposes for the time being.

Update the route "/" of your main_page.py application to render the index.html page.

Test your application and see if you get Hello World 2 in your browser.

### 3.4.1 Add components into the index.html page

Start adding the required components to your index.html page as described in the section below.

#### *3.4.1.1 The header and Store information*

- Add a header for the main title.
- Add a paragraph for some store information.
- Create a CCS file in the "static/css" directory (such as index.css) that will format the header and paragraph as shown in Figure 1.
    - Page background image
    - Title centered and white
    - Paragraph centered and white.
- Link this CSS file into your index.html file.
- Test/debug your application.

#### *3.4.1.2 The left area of the main page*

Your main page is divided into 2 areas: left and right. To create such areas you can use DIVs. In this section we will discuss the left div only.

In the index.css file, set the left div to 60% of the total width of a window and the right div to 35%. Just for debugging your application, you can set their heights to let's say 800 px and background color blue, so you can see those divs when displaying this page (the height and background color should be removed from your final submittal, this is just for debugging your code if you need to).

Test your code and see if you can see those two divs.

##### 3.4.1.2.1 The main menu in the left area

- Create a div for the main menu.
- Add an unordered list.
    - Each list item in this unordered list shall have its contents surrounded by an anchor tag. This anchor tag shall have a href pointing to someplace in this page (e.g., href="#drinks").
    - Each list item shall have an image of the item (img) and its description (h3).
    - Update index.css file such that:
        - 4 list items shall fit in a single line, with some margin between each list item.
        - No bullet is shown in the list items.
        - No link underline is shown for each link.
        - You have done similar formatting in your YouTube lab. Check how you did in there.
- Test your code and see if you get the main menu shown in Figure 1.

#### 3.4.1.2.2 The Burgers and Fries menu

- Create a div for the burgers and fries submenu.
- Create the title “Burgers, Fries, and Co.” as shown in Figure 1.
- Create an unordered list:
    - Each list item shall have an image of the item (img) and its description (h3).
    - Each list item shall have a text field so that the customer can add numbers in it.
    - Update index.css file such that:
        - 4 list items shall fit in a single line, with some margin between each list item.
        - No bullet is shown in the list items.
        - You have done similar formatting in your YouTube lab. Check how you did in there.

#### 3.4.1.2.3 Drinks, Coffees, and Dessert Menus

Repeat what you have done in Section 3.5.4 when implementing the for Drinks, Coffees, and Desserts submenus. Test your code for each implementation.

### *3.4.1.3 The right area of the main page*

Create a form tag (this form can be used instead of a div tag for grouping things to come) and do the following:

- Set its “action” attribute to the order summary path (you can hard-code the path for now, but later I would advice you to use the url_for later on.
- Set its method to the right HTTP request type
- Create an ID so you can format it later with CSS
- Add the following HTML tags inside this form:
    - A heading for the “Order Summary”.
    - A textarea.
    - A label.
    - A submit button to submit the order.
- Update the index.css file to reproduce the design shown in Figure 1.

Test your code and see if it is working as expected in terms of visual layout. At this point, since there is no JavaScript associated with this page, it is ok to not have dynamic behavior yet. See next section for that.

## 3.4.2 Implement functionality to the main page

To implement the desire functionality in the main page, such as if the customer chooses an item in the menu, the order summary textarea is updated accordingly, you need to develop some JavaScript code. You can use the following steps as guidance to implement the necessary functionality.

a) Create a JavaScript file in the static/js directory in your project.
b) Use window.addEventListener to wait for the browser to load the page components before adding your JavaScript functions (we have done this in multiple examples in class and labs).
c) Retrieve the DOM elements (getElementById) for all text inputs, textarea, customer name, etc.
d) Add event listeners to the necessary DOM elements. All the text input event listeners can call the same function, let’s say updateOrderSummary().
e) In the updateOrderSummary function, go over all the text inputs in the main page html and update the text area accordingly. Use something like orderSummaryElement.value += whatever to concatenate items in the textarea.
f) Link this JavaScript file to your HTML file.

Test your progress. Besides the “submit order”, all the other functionality on this page should be working at this point.

## 3.5 Order Summary Page

### 3.5.1 The basic functionality

Your application shall be able to display an order summary page (shown in Figure 2) from the following URL: /order_summary, therefor:

- You must create a new route in your main_page.py flask application.
- The HTTP method for this route shall be POST, since it will be receiving data to populate this page (order number, customer name, and list of items). Hint: the list of items can be a single string containing the orders separated by commas or invisible characters such as “\r\n” (new line in Windows OS), such as "2 Baked Apple (s) $5.00 \r\n 1 Chocolate Chips Cookie(s) $2.50". Then you can split this string later to get individual items to be placed in list item bullets.
- Read the data that is coming from the index.html form and populate this order_summary page accordingly (you need to pass this data via arguments in your render_template function, and use {{ }} in your order_summary.html page to replace these Jinja2 place holders with the actual data as discussed in class and labs. Hint: you will need to use a Jinja2 FOR loop to create list items here.
- Update your index.html form to point to this route.
- No special CSS formatting needs to be done for this page.

Run/test your application and see if you get the order_summary page to display the correct items selected by the customer from the main page.

- Last thing to do in your summary_page: after being displayed for about 10 seconds, your application should redirect the browser to display the main menu for the next customer in line. This is easily done by creating a JavaScript for this page containing the following code (I am giving this one for you guys, you are welcome…)

```
var timeout = 5000;

window.setTimeout(sendBackToMainScreen, timeout);

function sendBackToMainScreen() {
    window.location = "http://127.0.0.1:5000";
}
```

### 3.5.2 The database functionality

In the summary_order route in your flask application, it not only redirect the customer to that page but also saves the current order in a database. The entries in this database will be later used by the kitchen staff to retrieve the current orders.

#### *3.5.2.1 Creating a model for the database*

- In your project, create a new python file, such as order_model.py
- Based on what has been discussed in class (such as the TodoModel.py), create a python model for an Order. This model shall have the following columns:
    - Id.
    - customer_name.
    - orders (just a single string containing all the items ordered by the customer, as discussed above).
    - order_number.
    - date_created.

- This model shall have one function, named toJSON. This function transforms the model fields (id, customer_name, etc.) into a JSON object. This transformation is important when passing the order information that comes from the database (in "Order" format) to the kitchen page (in JSON format). This function is also given to you, as shown below:

```
def toJSON(self):
    return {"id":self.id,
        "customer_name": self.customer_name,
        "orders" : self.orders,
        "order_number" : self.order_number,
        "date_created" : self.date_created}
```

- Update your main app python program to:
    - Import and create the database, in a similar way as in the Todo hands-on project discussed in class.
    - The order_summary route:
        - Create a new order object, based on the data from the form tag (index.html page).
        - Add the new order into the database.

Run/test your application. If you have not installed the extension SQLite Viewer (as discussed in class), please do it so. After running your application, the database binary file should be automatically created and, by using the SQLite Viewer extension, you can look inside this file and see if your order has successfully been created.

## 3.6 The kitchen page

The kitchen page is a page that will be displayed in the kitchen area, from which the kitchen staff will ready and prepare the orders as they pop up in the monitor screen. The kitchen page shall resemble the one shown in Figure 3. Its functionality will involve the creation of HTML, CSS, and JavaScript files, along with changes in the main python/flask application. This section discusses details for all these components.

### 3.6.1 The kitchen route

In your python/flask application:

- Create a new route /kitchen/.
- In its associated function, retrieve all the orders found in the database. You will get a list of Order objects, not a JSON objects.
- Do a FOR loop and convert each Order object into a JSON object and save those JSON objects into a new array. Use the toJSON() function that we created in the Order model already.
- Render the kitchen html file and pass the JSON object array as an input.

### 3.6.2 The Kitchen.html page

- Create a kitchen.html page in the templates directory.
- Create a title page using a h1 header (as shown in Figure 3).
- Create an unordered list.
    - Inside the unordered list, create a Jinja2 FOR loop and loop over the array of JSON objects sent from the router (See Section 3.6.1) and create the order summary as shown in Figure 3. Hint: there will be a second Jinja2 FOR loop for the list of items chosen by the customer.
- Create an anchor link for each list item. Its href attribute should be set to a new route: delete/order.id. By clicking on this link, the order will be removed from the database (order has been delivered to the customer).

### 3.6.3 The kitchen CSS file

- Create a new CSS file in the static/css directory.
- Add formatting rules for each order. Try to get your application to reproduce the layout/formatting found in Figure 3 as close as possible.

### 3.6.4 The kitchen JavaScript file

- As mentioned in the introductory sections of this document, this page should automatically refresh itself every 15 seconds or so. The kitchen staff interaction with this page is with the delete order button/link only, nothing else.
- To create an automatic page refresh you can simply add the following code (given to you ready to use again):

```
var timeout = 5000;
window.setTimeout(poller, timeout);

function poller() {
    window.location = "http://127.0.0.1:5000/kitchen";
    window.setTimeout(poller, timeout);
}
```

### 3.6.5 The delete route

- Create a route /delete/<int:id> and its associated function delete(id).
- Retrieve a reference of the order to be deleted from the database (see Todo hands-on for similar code).
- Delete item from the database.
- Redirect to the kitchen page again, so the kitchen staff can see the incoming orders once again.