# Group Report

## I. Introduction [Written by Georgios]

Welcome to the introduction of our web application. Oaxaca is a restaurant group that has been slowly growing its presence in London and the UK. Due to the increasing competition in the market, the Chief Operations Officer (COO) fundamentally believes that to out-compete, Oaxaca will need to outgrow its competition by opening more restaurants at a faster rate than they have been doing so. However, the issue lies in hiring and training people fast enough to fill the staffing needs for the restaurant.

Our objective is to help the COO of Oaxaca to identify new technologies in order to help them mitigate the issue by reducing the staffing and/or training needs of their restaurants with technology. Throughout our research, we identified that waiting tables is one particular area of the business that requires a lot of staffing and training, meaning that our main goal is to develop a web application that makes this process easier for Oaxaca.

The scope of the project is to assist Oaxaca with its staffing and training needs by implementing software which is user-friendly, robust and accurate. Our web application is written in HTML, CSS, JavaScript, Python and SQL alongside the use of frameworks and templates such as Bootstrap, Flask and Jinja. It provides a fast, intuitive UI that someone doesn't need to be trained to use.

## II. Design, Planning, and Coordination [Written by Aryaman]

As instructed, we used scrum methodology for our project, integrating the design process from the outset. Our development team comprised of eight developers, although one was absent throughout the entire process. We collaborated closely with the product owner to prioritise the product backlog and determine the requirements for implementation. We divided the project into five sprints, each lasting around two weeks, and used Trello to manage our workflow, assigning tasks to developers for each sprint.

To initiate the project, we agreed to use Visual Studio Code as our main integrated development environment (IDE) and used Flask as our main framework, despite some initial download issues, which we resolved as a team. We also employed HeidiSQL which is a free and open-source administration tool for MySQL and SQLite to host our database. During our first sprint, our main objective was to set up VSC and go through tutorials and get used to using VSC as most people used Eclipse for the previous project and needed some practice to get used to VSC.

Planning played a critical role in our scrum development, with a designated scrum master for each sprint conducting scrum meetings both online and in person. Tasks were assigned based on priority, with consideration given to each team member's skills and ability to complete the task. Pair programming or task takeover was employed as needed to ensure completion. During scrum meetings, the scrum master would inquire about the team's progress and any difficulties they faced. If a team member had completed their assigned tasks, they would be given additional work to complete.

We held a sprint planning meeting on Mondays when the sprint started then held a scrum meeting every other day throughout the sprint. Initially, we planned on having a scrum meeting every day but it was not possible as everyone had different time schedules and other commitments, so we settled on 3 meetings each week, Mondays, Wednesdays and Fridays. Before the deadline, we planned to merge all working code into the main branch to avoid conflicts and ensure we had enough work to present to the product owners. We also made sure that our git repository was clean at all times to avoid any merge conflicts or Github-related issues. Team members documented and structured the code well to make it easy for others to understand and modify it.

During the observed scrum meetings, we had to carry out the usual scrum meeting which was observed by our product owners so we could receive feedback on what can be improved and ask any queries we had about the project. At the end of each sprint, we had demo sessions where we demonstrated our work to the product owners, receiving feedback and suggestions for further improvement. Coordination was key, as we allocated enough time for meetings to discuss how we could enhance the product's current status and implement the product owners' suggestions. We used Microsoft Teams and WhatsApp for communication, with Teams serving as our primary source for online scrum meetings, and WhatsApp for personal queries or issues. Overall, we made a great effort in designing, planning and coordinating this project with each other to get the best overall possible result.

## III. Coding and Testing [Written by Georgios]

The coding process for this web application was relatively simple. We organised a meeting at the beginning of the project in order to evaluate which backend languages people were most comfortable with and the languages that would offer us the best results. From the meeting it was understood that the team had little to no experience in PHP, therefore we decided to go with Python, using the Flask framework in order to make the website. Furthermore, we decided that it would be more optimal to have a solid front end first whilst other members of the team were working on the backend or sharpening their skills with Flask by following a tutorial. Normally, we would have one or two members working on the front end, one or two members working on the database and the rest of the team would be implementing new features in the backend. We started off with the customer menu which gave us a solid foundation to work with since it was the kitchen and waiter pages that needed access to a functional customer menu. Each team member had a similar role from sprint to sprint. Typically, the people that would work on the front end would keep working on the front end by fixing styling issues or implementing additional designs. The backend team would also keep working on the backend each sprint and that made it easier to coordinate our tasks and not have people feel like they are confused with their assignments.

The tools and technologies used for this web application describe an application written in Flask. The front end consisted of HTML, CSS, JavaScript and the CSS framework Bootstrap which was significant for our website due to the fact that it allowed us to make it responsive. Since most of the customers of a restaurant order on their phones, it made sense to use Bootstrap to make the website responsive. The backend was written in Flask, which is a microweb framework written in Python. The backend also included the Jinja web template engine. For the database, we used SQLAlchemy and this allowed us to keep the whole of the backend in Python.

Since we had people working on different aspects of the website, we implemented system testing in order to test the software. This meant that the people that were not working on a feature that others were implementing, would test it by inserting different variables or by checking for missing components. This allowed us to be on top of the functionality of the software due to the fact that people were aware of what was not working and they were willing to improve upon it. This line of testing continued until the very end of the project with the goal of not having any implementations of features that were dysfunctional.

# IV. High-level Description of Product [Written by Ellie & Callum]

## Overview of main components of the product)

### User interface components)

- Home page
    - The home page is where any user is taken upon loading the website.
    - The user is greeted with an input box for them to input the table they are seated at, and only accepts whole numbers greater than or equal to 1.
    - Below this input box is an order now button that takes them to the ordering page, if no table number is inputted, they are met with a message informing them to input a table number.
    - The nav bar at the top allows the user to travel to the following pages:
        - Menu
        - Help
        - About us
        - Staff login
    - The nav bar is visible on all pages, but the links change depending on the page you are on.
- Menu page
    - The menu page displays our menu with the following:
        - Image of each item
        - Name of each item
        - Calories of each item

        - Price of each item
        - Description of each item
        - Allergens of each item
        - Order on main page button (not working)
- Help page
    - Displays some FAQ’s customers may have.
- About us
    - Displays a short story talking about the restaurant.
- Ordering page
    - Displays the same info as the menu page.
    - Each item has an input box allowing the customer to input the quantity of an item they would like, defaults to 1.
    - Each item has an add-to order button that adds the item to their order in whatever quantity is specified.
    - Current order displays all items added to the customer's order alongside the quantity of each item and the total price of the order.
        - Within this the customer is able to remove items from their order, they can either remove a singular instance of an item or all of them, e.g. they ordered 13 beef tacos but then realised they only wanted 12 they could remove 1 taco, or if they wanted to remove all 13 they could.
    - Checkout button, upon being pressed a pop-up appears prompting the customer to input their card details, with validation making sure a valid card number, expiry and CVC are inputted. Below input for card details is a pay button that after being pressed a pop-up that says payment success appears informing the customer their order has been sent to the staff. The payment info pop-up has a close button for if the user changes their mind.
    - Need help button, upon being pressed the waiters are informed that the table number that has pressed the button needs help.
- Staff login
    - Two input boxes, one for username and one for password.
    - If the user gets either username or password wrong, a message is displayed informing them their login credentials were incorrect.
    - Upon successful login the staff member is taken to the relevant page for their staff type.
- Waiter page
    - Orders to review table:
        - Lists all unconfirmed orders from customers with the following columns:
            - Order ID
            - Table ID
            - Time of order
            - Total price of order
            - Actions
        - Orders are sorted from oldest to newest.
        - Actions are the following buttons:

            - Confirm marks the order as confirmed, which is sent to the kitchen. Once an order is confirmed it is no longer visible to the waiter staff until it has been cooked.
            - Cancel cancels the order, removing it from the waiter page.
    - Orders to deliver table
        - Lists all cooked orders with the following columns:
            - Order ID
            - Table ID
            - Time of order
            - Total price
            - Actions
        - Orders are sorted from oldest to newest.
        - Actions are the following buttons:
            - Delivered, marks order as delivered removing it from the waiter page.
            - Back to kitchen marks the order as not ready and is sent to the kitchen.
    - Tables that need help table
        - Table displaying all table numbers that have their help status set to true.
        - Button to mark a table as helped, after this button has been pressed the table number is removed from the table.
- Modify menu
    - Page for waiters to be able to modify the menu as needed.
    - Lists each item from the menu table in the database in a clear table with the following columns:
        - Name of item
        - Description
        - Price
        - Calories
        - Allergies
        - Item type
        - Item availability
        - Freeze item
    - There is a button in each row of the freeze item button that, when clicked, will update the database to show that item is temporarily unavailable if previously available and vice versa.
    - To modify any other column in the database the staff member can click the modify item button at the top of the page and a pop up window will appear.
    - Once the column name, item name and new value input boxes have been filled, the submit button will then send this information and update the database accordingly.
- Kitchen page:
    - Lists all confirmed orders from oldest to newest, each order is in its own table one after another. Each table contains all of the items from each order.

    - Each table has a row at the top showing the order ID and time of order.
    - Underneath are the following columns:
        - Name of item
        - Quantity of item
        - Status
        - Actions
    - The actions column is either a ‘complete’ or ‘uncomplete’ button depending on the status of the item.
    - Upon marking the last item of an order as complete a pop-up message stating that the order has been completed and sent to the waiters is displayed.
- Admin page:
    - Staff login information table:
    - Lists all staff login information with the following columns:
            - Staff ID
            - Staff role
            - Staff username
            - Staff password
            - Delete
        - Delete column contains a delete button that allows an admin to delete a staff member from the login, after being deleted that login will no longer be usable.
    - New staff form
        - Drop-down box with staff types (Admin, Waiter, Kitchen)
        - Input box for username
        - Input box for password
        - Add button, upon being pressed staff is added and able to login with corresponding credentials
    - Payment history table
        - Lists payment details of all orders with the following columns:
            - Payment ID
            - Card number
            - Table ID
            - Total price
            - Time stamp (date and time)

## Back end code)

See app.html for pydoc.

## Server-side components)

The only server-side component of the application is a MySQL database. The database has five tables: items, menu, orders, payment_history, and staff_login, which store information about items in each order, the menu, customer orders, customer payment history, and staff logins, respectively.

The "items" table stores information about items in an order. Each row in the table represents an item in an order and has columns for the item ID, order ID, quantity, price, and whether the item is ready, i.e. if it has been cooked. The "order_id" column is a foreign key that references the "order_id" column in the "orders" table, and the "item_id" column is a foreign key that references the "item_id" column in the "menu" table.

The "menu" table stores information about menu items. Each row in the table represents a menu item and has columns for the item ID, name, price, calories, allergies, item type, availability, and description. The "item_id" column is the primary key for this table. The table is populated with a txt file with all the relevant information.

The "orders" table stores information about orders. Each row in the table represents an order and has columns for the order ID, table ID, total price, whether the order is cooked, delivered, confirmed, and the time of order. The "order_id" column is the primary key for this table.

The "payment_history" table stores information about payment history. Each row in the table represents a payment transaction and has columns for the payment ID, card number, table ID, total price, and timestamp. The "id" column is the primary key for this table.

The "staff_login" table stores information about staff logins. Each row in the table represents a staff member and has columns for the staff ID, staff role, username, and password. The "staff_id" column is the primary key for this table.

The "tables" table stores information about tables in the restaurant. Each row in the table represents a table and has columns for the table ID and whether the table needs help. The "table_id" column is the primary key for this table.

## Libraries and frameworks)

Flask: Flask is a micro web framework for building web applications in Python. It was used to handle the routing and request handling of the application.

Flask_SQLAlchemy: Flask_SQLAlchemy is an extension of Flask that adds support for SQLAlchemy, a popular SQL toolkit and Object-Relational Mapping (ORM) library. It was used to interact with the MySQL database and perform CRUD operations on the database.

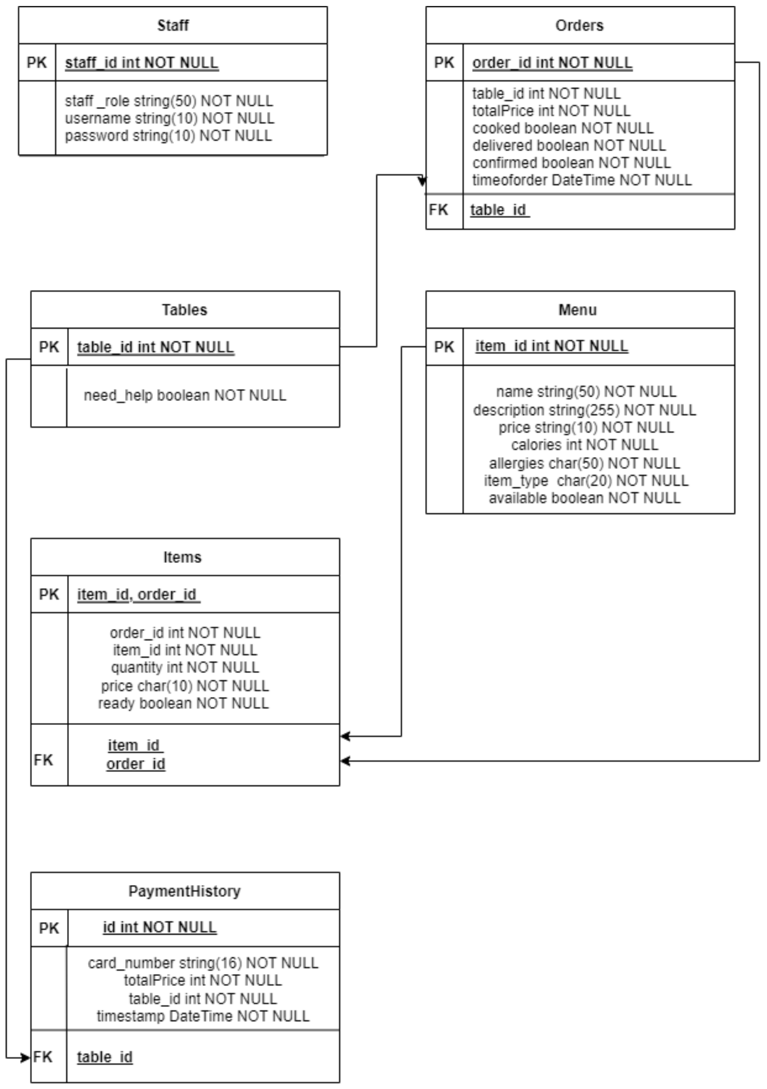
Database: The 'Database' module is a custom module developed for this project. Its purpose is to establish a connection to the MySQL database and allow us to interact with the database.

Jinja: Jinja is a popular templating engine for Python. It is used in Flask to render HTML templates and dynamically generate content based on data retrieved from the database or other sources passed in. It was used for if statements and for loops on the pages.

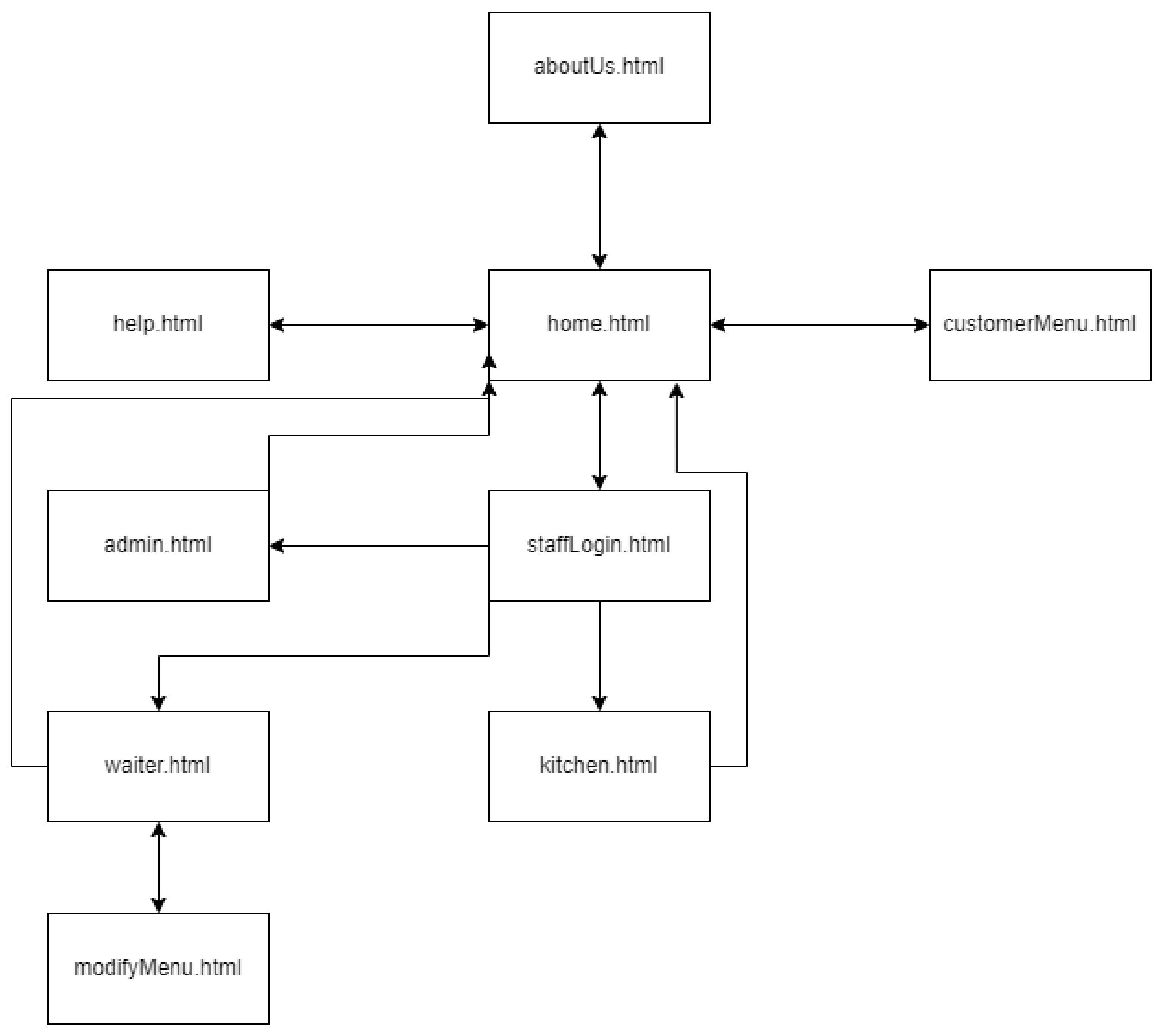
Bootstrap: Bootstrap is a free, open-source front-end development framework for the creation of websites and web apps. It was used to allow our application to be responsive to the user’s device size as customers are likely to order on different devices as they will use their own device, likely a mobile phone.

## Diagrams)

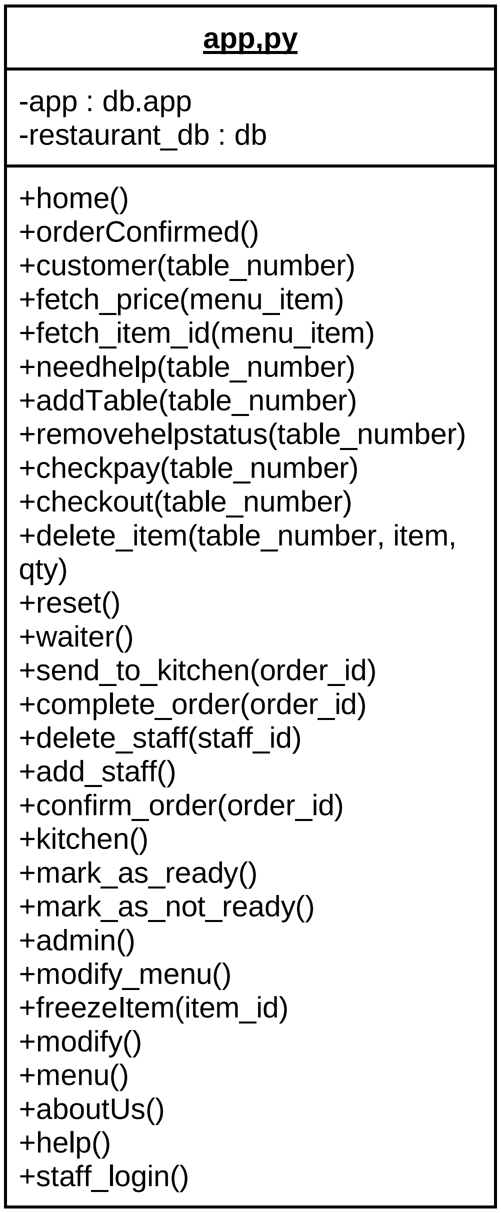
The diagram below shows the relationships between the tables in the database. The primary and foreign keys in each table (if applicable) are shown as well as the data types for each of the columns.

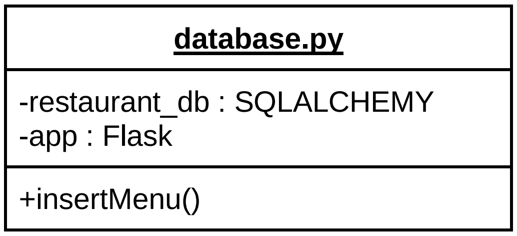


The diagram below shows the flow of the website through the HTML pages. This may be via the navigation bar at the top of each page or by clicking on certain buttons on individual pages.



The two tables below show the methods and global variables for the app.py and database.py files:


app.py
-app : db.app
-restaurant_db : db
+home()
+orderConfirmed()
+customer(table_number)
+fetch_price(menu_item)
+fetch_item_id(menu_item)
+needhelp(table_number)
+addTable(table_number)
+removehelpstatus(table_number)
+checkpay(table_number)
+checkout(table_number)
+delete_item(table_number, item, qty)
+reset()
+waiter()
+send_to_kitchen(order_id)
+complete_order(order_id)
+delete_staff(staff_id)
+add_staff()
+confirm_order(order_id)
+kitchen()
+mark_as_ready()
+mark_as_not_ready()
+admin()
+modify_menu()
+freezeItem(item_id)
+modify()
+menu()
+aboutUs()
+help()
+staff_login()


database.py
-restaurant_db : SQLALCHEMY
-app : Flask
+insertMenu()

## V. Description of Packages [Written by Aimee]

In programming, a package is a collection or directory of related modules that is used to create a hierarchical structure for organising code. This can be installed via package managers such as ‘pip’ and can then be imported in the source code.

Flask is a framework that allows us to build our web application quickly using python. The package provides tools, libraries as well as other features and is installed using ‘pip’. Below is a list of the main components that flask provides:

- Routing: The path selection mechanism enables requests specified by URLs and HTTP to be mapped to the code that handles them. This makes it easy to create web applications with multiple pages. For example, in the case of our ‘add_staff()’ function, a route is defined with ‘@app.route’ decorator to handle a ‘POST’ request for adding a new staff member to the database.
- Rendering template: The templating feature allows developers to create HTML files(such as kitchen.html and waiter.html) with dynamic contents. This builds web pages that display data from the database and respond to user inputs.
- Request: The request module handles incoming requests and generates responses. As the user submits data through a ‘POST’ request, the ‘request’ object retrieves the data and stores it in the ‘request.form’ object as key-value pairs. In addition, ‘Request.method’ is used to indicate what actions should the function handle, where ‘GET’ is used to request data from the server and ‘POST’ is used to send data to the server to update the resource.
- Session: The session module in flask stores data that is specific to the user including requests and interactions between the client and server when the client logs on the web application.
- Jsonify: The Jsonify function converts a python dictionary into JSON format, where JSON format is used in web applications to transmit data between the client and server.
- ‘Url_for’: The ‘url_for’ function is used to generate view functions that are registered in the flask application, where it takes the endpoint’s name as its first argument and returns a URL that corresponds to the endpoint using the @app.route decorator.

SQLalchemy is another package that is implemented in our web application. It provides tools and libraries that allow interactions and cooperation with the database. The package can also be installed using pip via the terminal. Some of the useful components in SQLalchemy include:

- Object Relational Mapper (ORM): ORM is a tool that handles the translation between python classes and tables on the relational database as well as converting function calls to SQL statements.
- Core SQL construction system: The core is the foundation of SQLalchemy and provides essential features such as reading, updating, creating, and deleting tables in addition to executing queries.
- Relationally-oriented system: The relationally-oriented system is used to implement joins, subqueries and correlation.

The Pydoc package is also imported to generate an interface for users to access and view the documentation of our python code which provides an overview of the program. Please find a generated pydoc named 'app.html' in the git repo and the submitted zip file.
The main components of the pydoc package are as follows:

- Pydoc module: The Pydoc module provides a 'help' function which indicates help documentation for a given function, keyword or class.
- Pydoc-GUI module: A graphical user interface is generated by this module which displays the documentation.
- Tools module: This module extracts documentation from the python source code and generates HTML files.

Datetime package is implemented in our program to work with dates, times and time intervals. One of the main components used is the datetime module which contains attributes such as year, month, and day, which allows the program to carry out operations on dates and times including obtaining the current time.

The Database package is used to connect to the MySQL database, allowing it to be able to interact with each other. Below is a list of the components that the package contains:

- MySQL connector: The MySQL connector establishes a connection with the MySQL server for client programs.
- Database Management System: This main component of the database package controls the overall functionality of the database as it provides interfaces for the application to interact with the database.
- Data Manipulation Language: This component provides tools for inserting and updating the database, enabling users to manipulate the data that is stored in the database.

## VI. List of User Stories [Written by Ellie & Callum]

Customer:
- As a customer I want to be able to see an electronic menu so I can select my dishes.
    - On our web application we have two types of menus. The first displays only the items, relevant item information and images but the customer cannot order from here. This menu is just to view what items our restaurant has. The second menu contains the same as the first but also has the functionality to add items to an order.
- As a customer I want to be able to have complete information on ingredients and calories for each dish so I can both avoid foods I am allergic to and follow my diet.
    - In both menus each item is displayed along with a full description as well as allergies and calories. Our web application does not allow the user to filter items by allergies. This is a feature that we ran out of time to implement.
- As a customer I want to be able to order food so I can eat.
    - The customer menu page allows a customer to order items from the menu and submit their order after payment.

- As a customer I would like to have an intuitive way of submitting an order so that the process is straightforward.
    - The customer menu page allows for a customer to order quantities of a singular menu item at once, such as 6 beef tacos instead of having to press the button 6 times. They can also delete individual items or all of a singular added item.
    - A customer can simply submit their order by clicking on the checkout button under their order summary, an alert box will then display to take payment for their order. If the card credentials are valid then an alert will display to confirm the order.
- As a customer I want to be able to call the waiter at any time so I can get help with my order.
    - The waiter can be called at all times once the customer has entered their table number. This is done by pressing on the ‘Need Help’ button under the order summary table.
- As a customer I want to see pictures of the food so that I can see what the food looks like.
    - On both menu pages a picture of the item is displayed above the item information. The pictures are large and show clearly what each item looks like when served.
- As a customer I want to be able to filter my choices when ordering so that I can view the food I'm interested in.
    - The customer can only filter the menu display between all items and gluten-free items only.
- As a customer, I want to be able to track the progress of my order, so that I know how much longer I have to wait.
    - Due to the way in which we implemented the flow between web pages this feature was not implemented as it would need a complete redesign of the implementation. This feature was added as part of the additional user stories halfway through and we did not have enough time to change our implementation without running a huge risk of the bulk of our functionality not working.
    - As a note we spoke to our PO about this and they said that this was fine as they had instructed us early on to not have a user registration system and without users being registered tracking orders was not feasible.
- As a customer I want to be able to pay from the electronic interface I have at the table so that I can leave at my own convenience.
    - Once the order has been finished, the customer is automatically asked to pay for their order and then once they have finished with their meal they can leave at their own convenience. The only drawback of this is that if an order is cancelled or not confirmed by the waiter, a refund would need to be given to the customer manually.

Waiter:

- As a waiter, I want to be assigned to a certain table, so that I won't interfere with other waiters.
    - Due to the setup of our database from the start this feature was not added as it meant changing a large number of functions to satisfy the conditions of which table is allocated to which staff member.
- As a waiter, I want to view the orders that are currently being prepared and see when they are ready to be delivered.
    - Once the waiter has logged in they are able to see a table of orders which clearly shows which orders are ready to be delivered. The waiter will need to refresh the page

    - regularly to update this table for when the kitchen staff have submitted an order to be delivered.
- As a waiter, I want to cancel the customer's order, so that the kitchen staff knows when people change their mind.
    - Once a customer has completed their order it will not be displayed to the kitchen staff until it has been confirmed by a waiter, the waiter has the option to either accept or decline the order.
    - Once the order has reached the kitchen it is no longer possible to cancel the order, this was done to reduce the clutter on the waiter page as we believed it made the most sense for them to only be able to view orders not in the kitchen i.e. those that need to be confirmed/cancelled and those that need to be delivered after being cooked.
- As a waiter I want to be notified when the client needs help so I can assist them.
    - On the top right corner of the waiter page a table is shown that displays any clients that have requested help. Once the waiter has assisted that client they can click the button to remove them from the table and the page will refresh to show this.
    - This table is persistent, always visible on the waiter page no matter how far they scroll.
- As a waiter, I want to be able to mark the order as delivered, so that the order progress is tracked correctly.
    - A button is shown next to each order that has been prepared by the kitchen that will allow the waiter to confirm when they have delivered the order. The page will then refresh and that order will no longer appear in the table.
- As a waiter, I want to be able to change the status of an order so that the customer is kept informed.
    - As we did not implement a tracking system in our application this feature was not implemented in a way that keeps the customer informed of the order status.
    - The waiter is able to change the status of an order from unconfirmed to confirmed, or cancel the order
    - The waiter is able to mark a cooked order as delivered or send it back to the kitchen marking it as uncooked.
- As a waiter, I want to be notified when a table is ready to order, so that I can confirm the order.
    - Once the customer has submitted an order it comes up at the first table on the waiter page along with two buttons, one to confirm the order and send it to the kitchen staff to be prepared and one to cancel the order if there is a problem.
- As a waiter, I want to see the orders by the time they are placed, so that I can prioritize them.
    - Each order is stored in the database along with a timestamp of when the order was submitted. When displaying the orders to be delivered, the orders that were placed first are shown at the top, in ascending order of time.
- As a waiter, I would like to be able to change the menu, so I can show customers only currently available dishes.
    - From the waiter page they can use the nav bar at the top to go to a separate page that displays each menu item with all its information. A ‘freeze’ button can be clicked for each item that will temporarily remove it from the item by updating its availability status in the database. The waiter can modify part of an item’s information by clicking on the part they wish to edit after entering the new value in

    - the textbox at the top of this page. Each time an item is frozen or changed the page is refreshed to show the updated version.
- As a waiter, I want to be notified when the kitchen is ready with a dish so that I can deliver it to the table.
    - A notification is not displayed but upon refresh of the waiter page the new orders to be delivered are clearly shown in the table.

Kitchen:
- As kitchen staff I want to be informed when a customer order is confirmed so I can start preparing it.
    - When a new order has been confirmed by the waiter it will show as a new table at the bottom of the page, as the older orders take priority at the top. Regular refreshes of the page are needed to show these new orders.
- As kitchen staff I want to know the times at which the orders were made so I can tell if I am on track.
    - Each order is displayed as a separate table along with the timestamps of when they were submitted by the customer. This is clearly shown at the top of the table so the kitchen staff can see if they are on track.
- As kitchen staff I want to be able to notify waiters once an order is ready so that they can deliver it.
    - Once every item in an order has been marked as complete by the kitchen staff, the order is removed from the kitchen staff page and will be updated in the database as ready to be delivered. This order will then be displayed on the waiter page to be delivered.
- As kitchen staff I want to be able to view items to be cooked grouped by their order.
    - Each order is displayed in its own table on the kitchen staff page and shows each item in the order. The orders are shown clearly and separated by the order information being in different colours so the kitchen staff can see when one order ends and a new one starts.

Admin:
- As admin staff I want to be able to view login details for each staff member.
    - When the admin member logs in, a table shows all the logins for the staff members, including their ID, role, username and password.
- As admin staff I want to be able to delete a login.
    - Next to each login there is a ‘delete’ button that, when clicked, will delete that login from the database. The admin page will then be refreshed automatically to show that this login no longer exists.
- As admin staff I want to be able to add a new login.
    - On the top right corner of the admin page there is a section that allows a new login to be added to the DB. The admin staff selects the job role from a drop-down menu and then enters the new username and password into the individual input boxes. Once the ‘add’ button is clicked, the new login will be added to the database and the page will automatically refresh to show the login in the table.

## VII. Final Software Demonstration [Done by Callum]

https://youtu.be/6_zVeUvecx0

## VIII. Conclusion [Written by Georgios]

In conclusion, the main objective of this project was to help the COO of Oaxaca to identify new technologies in order to help them mitigate the issue of increasing competition by reducing the staffing and/or training needs of their restaurants with technology. It can be said with confidence that the main objective has been successfully completed and the outcome of this project is a new fast, intuitive UI that someone doesn't need to be trained to use.

Throughout this project, the team had strengths and weaknesses. Our strengths were that we are all experienced programmers with HTML, CSS and Python so implementing features and styling the website was not a task which required a lot of learning. Furthermore, we made sure that we had a meeting every 2 working days to make sure that everyone is caught up on what other people are doing and to discuss any issues that anyone could be facing. The main weakness of the team was miscommunication between members that worked on the same feature, or similar features since they would re-write and delete each other's code without realising it. Also, most members of the team were inexperienced with using git; especially merging branches so quite a few merge conflicts had to be resolved throughout the development of the software. These issues lead to some user stories not being completed due to a shortage of time towards the end of the project.

The project itself was a success but there is room for improvement both in styling matters, functionality matters and some extra additional features. For example, the payment system could be improved by implementing Stripe so that money is taken out of the customer's bank account. Another feature that could be implemented would be an ordering tracking system for the customer so that they can track their order.

Despite the issues that the team faced when developing the web application, everything tied itself perfectly in the end and there is a fully functional software ready with a fast intuitive UI for Oaxaca to use.