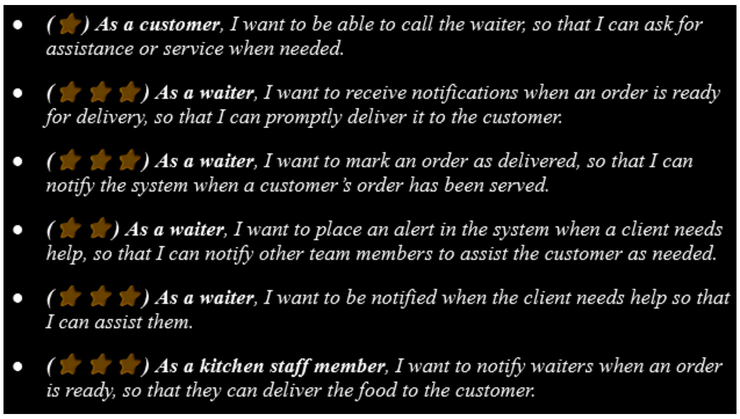
Individual Documentation

Royal Holloway University of London
24-25 CS2810/CS2815: Team Project

Q1.

Choose a snippet of code that you wrote and feel proud of. Explain how the code works and how it contributed to the success of the project.

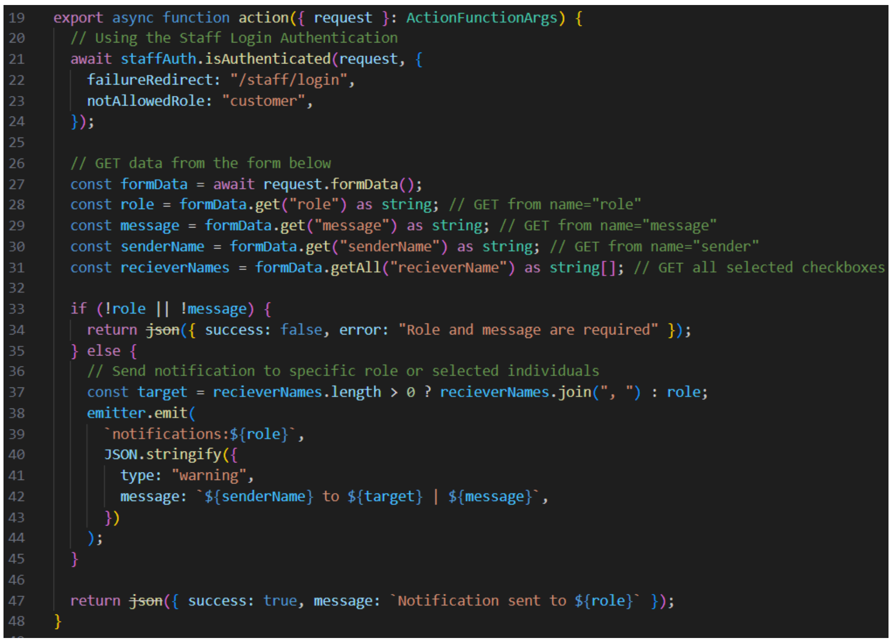
Across 6 user stories, one of the difficulties I've faced was in regards to being able to properly send alerts or notifications. To solve this issue, I've constructed a custom notification sender which is a nested web page where users can select the role, name of the employee, and write a message and automatically send a alert that is formatted in a way in which it pulls the sender's name from user authentication, recipient's name retrieved from the database, and the custom message filled in the form.



- *(⭐) **As a customer**, I want to be able to call the waiter, so that I can ask for assistance or service when needed.*
- *(⭐⭐⭐) **As a waiter**, I want to receive notifications when an order is ready for delivery, so that I can promptly deliver it to the customer.*
- *(⭐⭐⭐) **As a waiter**, I want to mark an order as delivered, so that I can notify the system when a customer's order has been served.*
- *(⭐⭐) **As a waiter**, I want to place an alert in the system when a client needs help, so that I can notify other team members to assist the customer as needed.*
- *(⭐⭐⭐) **As a waiter**, I want to be notified when the client needs help so that I can assist them.*
- *(⭐⭐⭐) **As a kitchen staff member**, I want to notify waiters when an order is ready, so that they can deliver the food to the customer.*

(All 6 User Stories that can be positively affected by this code.)

In the code, the 'action' function acts as a listener, constantly checking if the user is authenticated and if the login form is completed. Upon opening the webpage, the user is prompted to log in. Their role is then verified, automatically restricting customer access. Once authenticated, they can filter employees by role, select recipients, and type a message.



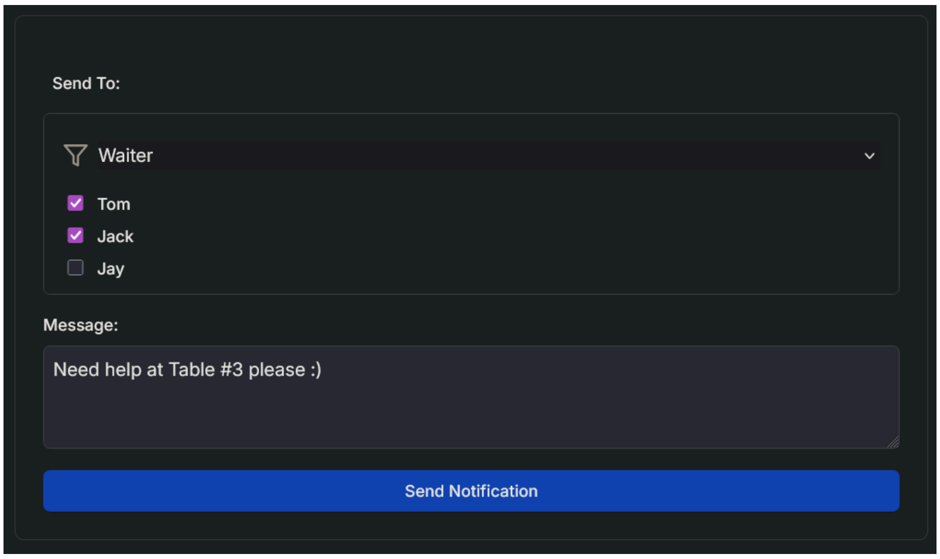
```
export async function action({ request }: ActionFunctionArgs) {
  // Using the Staff Login Authentication
  await staffAuth.isAuthenticated(request, {
    failureRedirect: "/staff/login",
    notAllowedRole: "customer",
  });

  // GET data from the form below
  const formData = await request.formData();
  const role = formData.get("role") as string; // GET from name="role"
  const message = formData.get("message") as string; // GET from name="message"
  const senderName = formData.get("senderName") as string; // GET from name="sender"
  const recieverNames = formData.getAll("recieverName") as string[]; // GET all selected checkboxes

  if (!role || !message) {
    return json({ success: false, error: "Role and message are required" });
  } else {
    // Send notification to specific role or selected individuals
    const target = recieverNames.length > 0 ? recieverNames.join(", ") : role;
    emitter.emit(
      `notifications:${role}`,
      JSON.stringify({
        type: "warning",
        message: `${senderName} to ${target} | ${message}`,
      })
    );
  }

  return json({ success: true, message: `Notification sent to ${role}` });
}
```

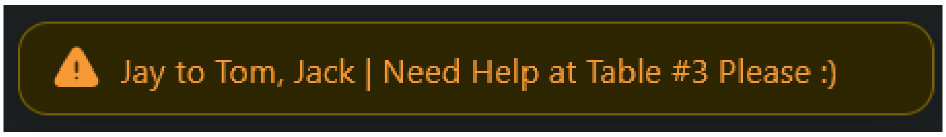
(Code to write and send real time notifications)



Need help at Table #3 please :)

(Notification UI with role-based dropdown, auto-displays employees. Super cool.)

Upon submission, the recipient names and messages are stored in constants, and the real time alert/notifications are queued to be displayed on the website. If specific recipients are selected, it mentions them directly; otherwise, it's mentioned to an entire role group. The notification type is set to *warning*, which is the name of a prefab design for the notification.



Jay to Tom, Jack | Need Help at Table #3 Please :)

(Notification with the "warning" style prefab. Displays in real time on top left of all pages using SSE.)

In its entirety, this code simplifies the process of sending alerts and notifications as it can all be done within this page as any employee has the ability to communicate with any of their co-workers in a speedy fashion. All other notification-related user stories may use this function to send alerts/notifications. As nothing is hard-coded, the function can also be automated to send specific messages with a click of a couple button. Following the principles of the Agile Manifesto, I ensured simplicity.

Q2.

Consider the scenario below, based on a fictional restaurant system.

The fields in each table are listed below:

- **client**: name, date of birth, address, phone number, email address, and amount spent in the restaurant since registering the details;
- **order_status**: ID, name, and description;
- **order**: order ID, table number, time placed, time of last change, notes, and order_status ID.

The user stories for the system are as follows:

- **Story 1:** A client should be able to register with the application, providing a name, date of birth, address, phone number, and email address.
- **Story 2:** A manager user should be able to see the total spent by all clients since they registered their details, for each of a set of age ranges.
- **Story 3:** A waiter user should be able to see all orders, each showing the order ID, table number, time placed, time of last change, notes, and the corresponding order_status name.

Using Function Point Analysis (FPA), update the table with all the **EI**s, **EO**s, **EQ**s, **ILF**s and **EIF**s in the system. There may be 0, 1 or more of each. Using the FPA reference sheet, update the table by classifying each row as **low (L)**, **average (A)**, or **high (H)** complexity and assign a **Function Point value** to each. Include calculations/workings you used to determine this information, below the table. Finally, estimate the total **Unadjusted FP Count (UFC)**

		Complexity	**FPA**
EI *User-provided data items. Ex. Data entry by user*	*Entering Client Details* *(name, birth date, address, phone number, email address)*	**(A) Average**	4
EO *Generated data items; requires transformation of stored data* *Ex. Reports, messages*	*Display Client's total expenditure*	**(L) Low**	4
	Categorize Client's age range from date of birth	**(L) Low**	4
EQ *Interactive inputs requiring response, derived from stored data (no transformation)*	*Display Client Details* *(Age range)*	**(L) Low**	3
	Display Order Details *(order ID, table number, time placed, time of last change, notes, order status name)*	**(A) Average**	4
ILF *Logical master files* *Ex. Database tables, flat files, user preference*	*Client Details* *(name, date of birth, address, phone number, email address, time spent)*	**(L) Low**	7
	Order Details *(order ID, table number, time placed, time of last change, notes, order status ID)*	**(L) Low**	7
	Order Status Details *(order status ID, name, description)*	**(L) Low**	7
EIF *Machine readable interfaces to other systems. i.e. ILF but from external applications*	-	-	-
			UFC = 40

Complexity Workings:

DETs = 5, FTRs = 1 Therefore **(A) Average**
DETs = 1, FTRs = 1 Therefore **(L) Low**
DETs = 1, FTRs = 1 Therefore **(L) Low**
DETs = 1, FTRs = 1 Therefore **(L) Low**

DETs = 7, FTRs = 2 Therefore **(A) Average**
DETs = 6, FTRs = 1 Therefore **(L) Low**
DETs = 7, FTRs = 1 Therefore **(L) Low**
DETs = 3, FTRs = 1 Therefore **(L) Low**

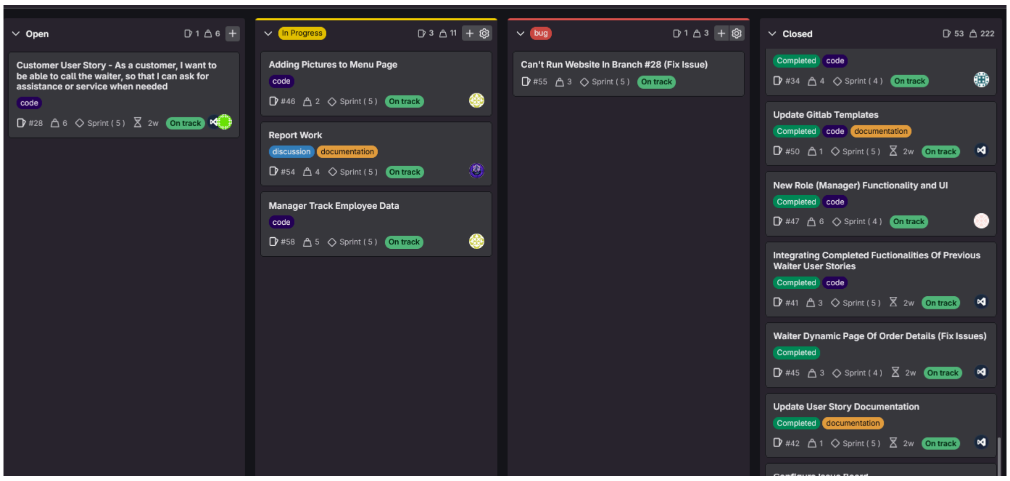
Q3.

Describe what you learned about developing software as part of a group using Scrum. Justify your points using specific examples from your experiences in your team. Also relate your explanations to relevant professional issues discussed in the course, including the codes of practice/conduct.

One of the biggest issues when working in any large-scale team is communication. With multiple developers working on different features, keeping everyone aligned is challenging. Miscommunication can lead to duplicated work, unresolved blockers, and delays. By implementing proper SCRUM practices, I was able schedule SCRUM meetings, ensuring that every team member had a clear understanding of ongoing tasks. These meetings provided a structured way for the team to discuss obstacles, report bugs, and share solutions—whether through discussions or helpful resources like documentation and tutorial videos. As a result, we accelerated our progress and completed most of the user stories within the first few sprints.

Furthermore, I introduced standardized Git issues and bug templates. This ensured proper documentation in every issue, keeping track of tasks, deliverables, weights, and more. Everything was neatly organized on the issue board with clear labels, allowing anyone to quickly see who was working on what and what needed to be completed. This approach ensures Section 3: 'Duty Of Relevant Authority' within the BCS Code of Conduct is fulfilled as each member's duties are documented alongside metrics to track progress in adherence to the BCS Code of Practice

Additionally, adhering to the principles of the Agile Manifesto, The team and I ensured that each member had the freedom to self-organize and choose their preferred issues to work on, staying away from micro-management, opting for a more 'Laissez-faire' approach whilst prioritizing the most important user stories for our product owner. In doing so, we were able to deliver software each sprint.


Open 1 6

Customer User Story - As a customer, I want to be able to call the waiter, so that I can ask for assistance or service when needed
code
#28 6 Sprint (5) 2w On track

In Progress 3 11

Adding Pictures to Menu Page
code
#46 2 Sprint (5) On track

Report Work
discussion documentation
#54 4 Sprint (5) On track

Manager Track Employee Data
code
#58 5 Sprint (5) On track

bug 1 3

Can't Run Website In Branch #28 (Fix Issue)
#55 3 Sprint (5) On track

Closed 53 222

Completed code
#34 4 Sprint (4) On track

Update Gitlab Templates
Completed code documentation
#50 1 Sprint (5) 2w On track

New Role (Manager) Functionality and UI
Completed code
#47 6 Sprint (4) On track

Integrating Completed Fuctionalities Of Previous Waiter User Stories
Completed code
#41 3 Sprint (5) 2w On track

Waiter Dynamic Page Of Order Details (Fix Issues)
Completed
#45 3 Sprint (4) 2w On track

Update User Story Documentation
Completed documentation
#42 1 Sprint (5) 2w On track

(Organized GitLab issues using GitLab Issue Boards)

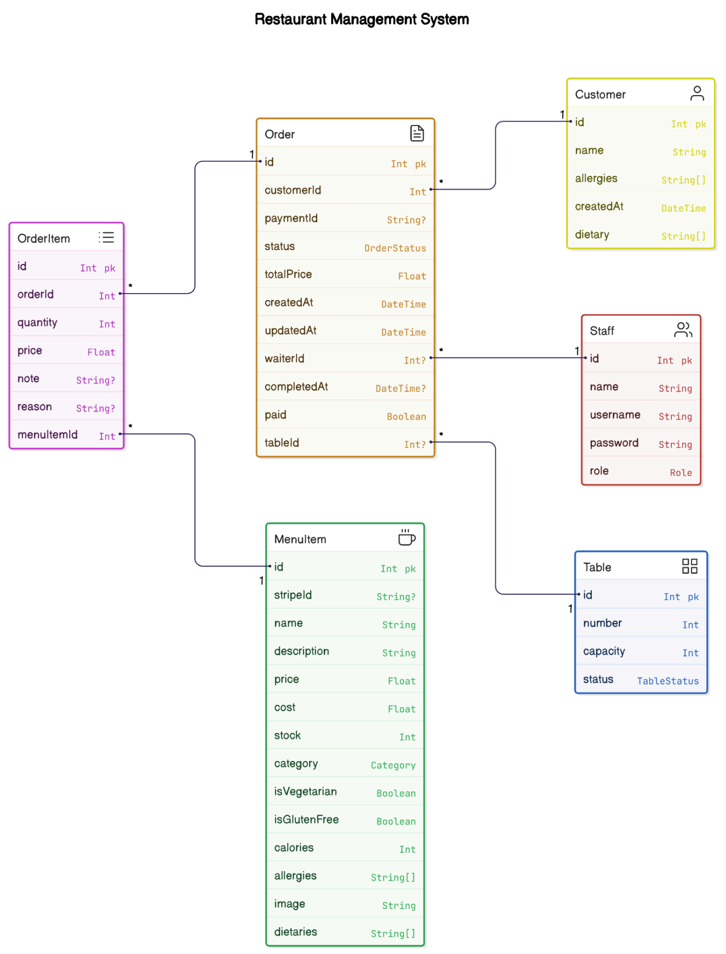
Q4.

Select 1 topic from: Architectural Model, Refactoring, Robust Code, Software Security.
Describe the relevance of your chosen topic to the design and code produced in your team project. Justify your points using specific examples from your project. [Maximum: 250 words]

Chosen: Architectural Model

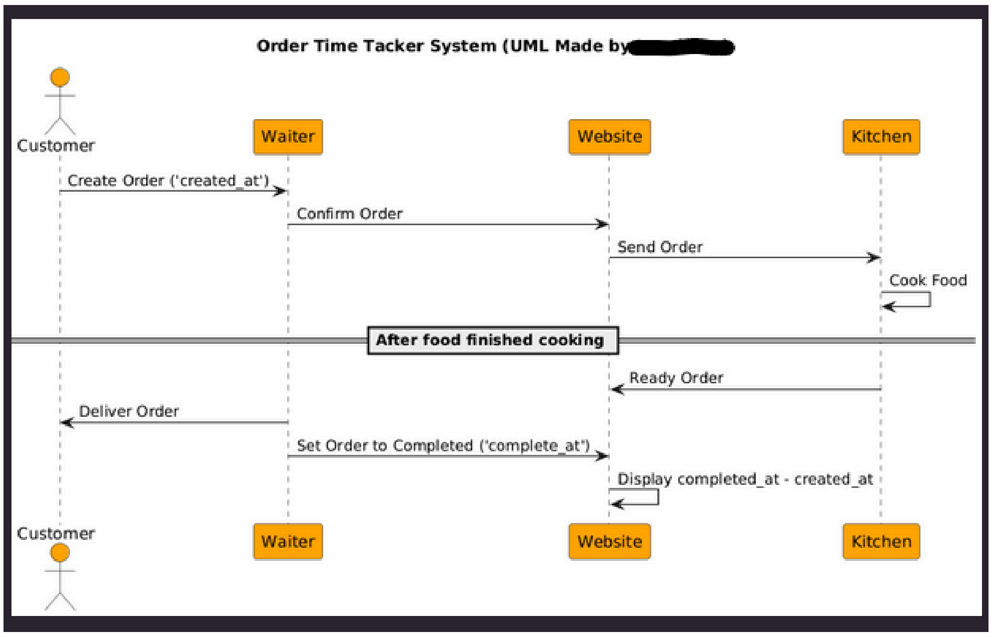
Due to the nature of Agile development, the architectural models were comprised of multiple architectural views, typically in the form of diagrams and text within various GitLab issues. The majority of the code developed focuses primarily on interactions with the database. The image below shows the final **logical view**, illustrating how the entities within the database are related to one another.

Diagrams were only made if it was deemed useful and cost-effective strategies were used in order to make each diagram, most of the programs used being free to use. This adheres to the ideal architectural model standards described in the lecture slides



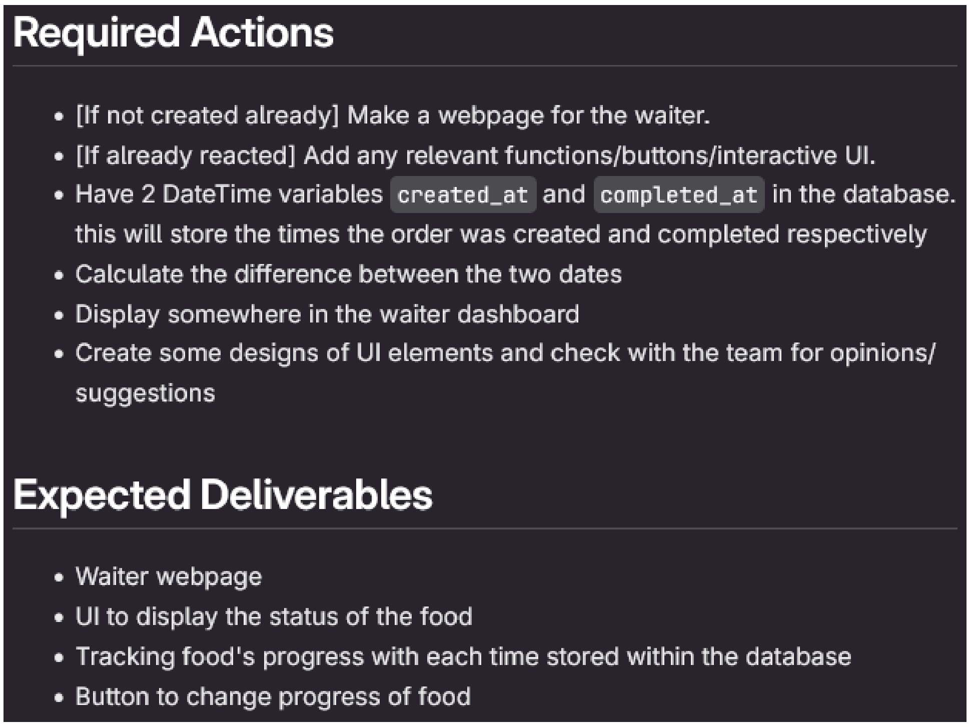
(Diagram of the design of database and each entities attributes and relations)

A UML sequence diagram using PlantUML is used to represent the **process view,** showing the interactions between the users and various parts of the website. In this example, the user story focused on enabling waiters to track the order time for each order.


Order Time Tacker System (UML Made by [redacted]
Customer
Waiter
Website
Kitchen
Create Order ('created_at')
Confirm Order
Send Order
Cook Food
After food finished cooking
Ready Order
Deliver Order
Set Order to Completed ('complete_at')
Display completed_at - created_at
Customer
Waiter
Website
Kitchen

(Initial UML sequence diagram to plan for order tracking time)

Appropriate information regarding a detailed breakdown, outlining each required action needed to be undergone during development and their corresponding expected outcomes are all listed upon each GitLab issue entry in entirety following the team's GitLab issue template. This can be considered as the **development view**.



Required Actions

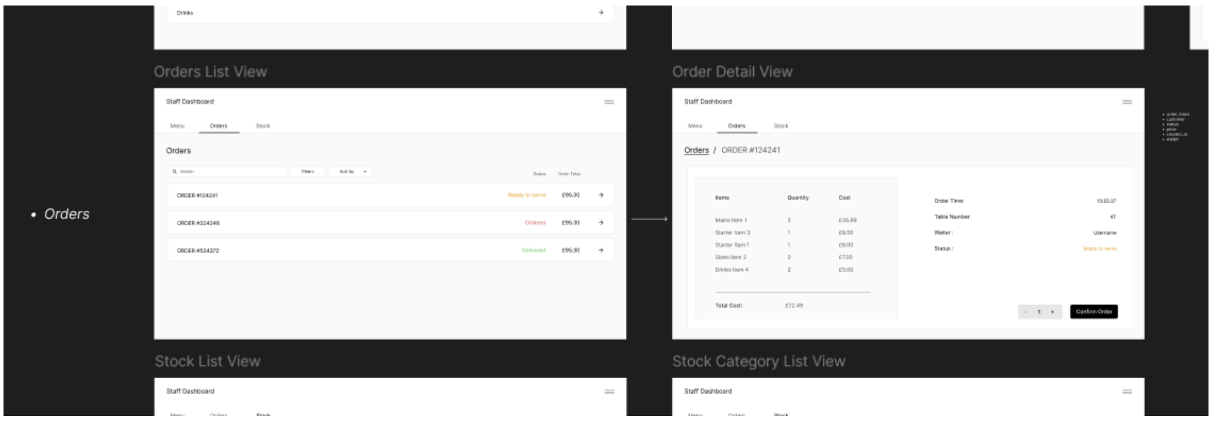
- [If not created already] Make a webpage for the waiter.
- [If already reacted] Add any relevant functions/buttons/interactive UI.
- Have 2 DateTime variables `created_at` and `completed_at` in the database. this will store the times the order was created and completed respectively
- Calculate the difference between the two dates
- Display somewhere in the waiter dashboard
- Create some designs of UI elements and check with the team for opinions/ suggestions

Expected Deliverables

- Waiter webpage
- UI to display the status of the food
- Tracking food's progress with each time stored within the database
- Button to change progress of food

(GitLab issue required actions and expected deliverables information section)

Finally, in order to help all team members visualize the project and implement the front-end design as intended, a Figma board was created. This board served as a **physical view** and a reference for the website's layout, showing how different page components appear and interact.



Orders

	Status	Order Total
ORDER #124241	Ready to serve	£95.30
ORDER #224246	Ordered	£95.30
ORDER #524372	Delivered	£95.30

Orders / ORDER #124241

Items	Quantity	Cost
Mains Item 1	2	£45.99
Starter Item 3	1	£8.50
Starter Item 1	1	£6.00
Sides Item 2	2	£7.00
Drinks Item 4	2	£5.00
Total Cost:	£72.49	

Order Time:	13:25:37
Table Number:	47
Waiter :	Username
Status :	Ready to serve

(Orders section of the)