**PRCS251A – Computer Science Integrating Project**

Craig Banyard (10488708),

Daniel Scott (10501358),

Joseph Kellaway (10503639),

Mujaheed Abubakar (10519512),

Zakaria Robinson (10500227)

University of Plymouth

06/05/2017

# Contents

|  |  |
| --- | --- |
| Introduction | 2 |
| Project Management | 2 |
| Requirements Analysis and Design | 8 |
| Database Design | 24 |
| Human-Computer Interaction | 34 |
| Security | 39 |
| Software Engineering | 40 |
| Evaluation and Reflection | 43 |

# Introduction

This report details the planning, design, development, and project management of the Computer Science integrating project.

The integrating project consists of a desktop application developed in Java that is used to populate and manage the content stored on a database for sale at a pizza fast food chain. This application is also required to create and manage staff accounts for use with the following two applications; another desktop application that displays live orders from when they are placed through to when they are ready for delivery, with the ability to progress individual orders during this process, and an application that is developed to run on Android mobile devices to display orders that are ready for delivery, guide the user to the destination, and progress the order to the point of being delivered to (or refused by) the customer. The final component of the project is a website that has been developed using HTML, CSS, JavaScript/JQuery and Java, for members of the public to register an account, browse current menu items and place orders, which are then displayed on the kitchen desktop application.

# Project Management

The following table details the members of this project, any roles assigned during the initial planning stages of the project, and contact details shared amongst the group for use during implementation. The table does not show how actual production of the project was carried out.

|  |  |  |
| --- | --- | --- |
| **Name** | **Roles** | **Contact** |
| Mujaheed Abubakar | Desktop developer  Mobile developer | 07440258680  [mujaheed.abubakar@students.plymouth.ac.uk](mailto:mujaheed.abubakar@students.plymouth.ac.uk) (preferred)  skype: mujaheed.abubakar |
| Craig Banyard | Lead database developer  Web developer  Mobile developer | 07584575250  [craig.banyard@students.plymouth.ac.uk](mailto:craig.banyard@students.plymouth.ac.uk)  skype: craig\_b03 (preferred) |
| Joseph Kellaway | Lead web developer  Mobile developer  Database developer | 07472691769  [joseph.kellaway@students.plymouth.ac.uk](mailto:joseph.kellaway@students.plymouth.ac.uk) (preferred)  skype: j.kellaway88 |
| Zakaria Robinson | Project manager  HCI lead  Mobile developer | 077414459903  [zakaria.robinson@students.plymouth.ac.uk](mailto:zakaria.robinson@students.plymouth.ac.uk)  skype: Zaperon0 (preferred) |
| Daniel Scott | Lead software developer  API developer | 07594875693  [daniel.r.scott@students.plymouth.ac.uk](mailto:daniel.r.scott@students.plymouth.ac.uk)  skype: danzo111994 (preferred) |

The following table details the tools that were chosen to be used during the production of the project.

|  |  |
| --- | --- |
| **Tool** | **Use** |
| Android Studio | Production of mobile applications for delivery staff. |
| Bitbucket | Online repository and source control.  https://bitbucket.org/PRCS251\_TeamA/ |
| Lucidchart | Production of ERDs and UML diagrams. |
| Microsoft Visual Studio | Production of API. |
| Notepad++ / Atom | Production of website for customers. |
| NetBeans IDE | Production of desktop application for chef and manager. |
| SQL Developer | Production of database. |
| Trello | Project/sprint management.  https://trello.com/b/MAurDWgT/prcs251a-sprint-board |
| SQL Map | To test against SQL Injection and database attack |

During implementation, plans deviated slightly. Netbeans was used for the website production instead of Notepad++ or Atom, and there was not sufficient time to test protection from SQL attacks using SQL Map. The team did however prevent the use of characters that are required to start SQL injection (“ ‘ ;) and also prevented cut, copy and paste in the textbox inputs. A tool overlooked by the initial plan was Photoshop. Custom buttons were a necessity for certain elements of the project, such as the website, and Photoshop was the tool of choice to create them.

## Project Closure Meeting

The project closure meeting commenced on Monday 8th May 2017. Mujaheed Abubakar, Craig Banyard, Joseph Kellaway and Daniel Scott were in attendance. The following was discussed and agreed upon within the meeting:

### Details of work carried out

Throughout the project the team were regularly separated into sub-teams that worked on applications that could be implemented simultaneously.

The website was entirely written by Joseph Kellaway and Craig Banyard, who agree that credit should be split evenly. If one was working on implementing a feature, the other was either aiding them to research possible solutions to any issues encountered, or working on a separate feature. The art for the website was created by Craig Banyard, with Joseph Kellaway aiding in the form of researching conventions created by competitors.

Security for the project was primarily required for the website component, since it is there that the system would most likely be subjected to an attack. As there was already a team working on the website it was decided that they should continue working together on what was deemed an extension of this. Research and initial implementation is therefore credited to Joseph Kellaway and Craig Banyard.

Both manager and chef desktop applications were primarily created by Daniel Scott, with Mujaheed Abubakar assisting on the manager application.

Development of the mobile application has been carried out primarily by Daniel Scott, with the assistance of both Craig Banyard and Joseph Kellaway.

The library started as a data model that was started by Zakaria Robinson. This data model has been updated by Daniel, Joseph and Craig as they have been implementing website and desktop applications. The data model was originally converted into a library by Daniel when he sought to use code used for API requests as previously implemented by Joseph and Craig for the website. These controllers were separated out into their own package and created in such a way that one exists per object class in the data model. Once they had confirmed it was functional, the security package was moved into the library by Joseph and Craig to allow the desktop and mobile applications to perform the same algorithms as the website.

The API endpoints were originally opened as needed by Daniel Scott as he was working remotely, and the computers in Babbage Open Access didn't have the required updates installed to deploy it. As the project progressed Joseph Kellaway also opened up and modified endpoints as required to include additional features for the website such as "custom pizza".

Database implementation was carried out by Craig Banyard. Database design (in the form of drawing entity relationship diagrams and normalisation) was carried out by Joseph Kellaway and Zakaria Robinson.

### Review of project

Overall, the project has fallen short of initial expectations due to the lack of functionality in relation to the initial designs. If this project was created for an actual customer, both the functional and aesthetic aspects of the completed product would be appreciated, but there would be a level of disappointment due to the failure to complete the project in accordance with the initial plan.

The project contains the ability to add menu items to the database through the manager application, add them to orders placed by new and existing customer accounts on the website, and pay for those orders using either cash on delivery or PayPal, which is integrated within the website. These orders are displayed on the desktop application and can be advanced through the process to "Ready for Delivery", which is then in turn displayed on the mobile application for the delivery driver. The mobile application can accept orders for delivery and request a route to the customer’s address, before confirming that the delivery is complete or notifying the system that the customer has refused delivery.

The manager application provides the ability to add new and edit existing members of staff so that additional chefs and delivery drivers can be added to the system when necessary. This system also allows the manager to void orders at their discretion.

Whilst this summary of the system shows that the base functionality of the assignment has been implemented, the group's collective management of the project has been inadequate in some areas due to various difficulties faced.

### Difficulties faced through the project

The difficulties that have arisen during the project can almost entirely be summarised as a lack of two things: communication and time.

A minor amount of work was lost at the beginning of the project due to lack of communication when using source control. In hindsight this was insignificant, and provided valuable lessons in how to use source control effectively, but emphasised that there was a lack of communication amongst the team at that point in time.

Having inadequate time to implement all planned aspects of the project was caused by two things; underestimating the time required for a task, and team members devoting too finite an amount of their time to the project.

Primary examples of the group underestimating the time required for specific tasks are the implementation of the website, and the implementation of the application. Unfamiliarity with CSS and Android were the respective causes for these underestimations. This in turn led to future tasks that had been allocated a sufficient amount of time needing to be rushed in order to complete them on time, or discarded.

Two of the team allocated approximately half as much time as suggested to the project[[1]](#footnote-1), which added extra pressure on the remaining three members of the team. Lack of commitment was discussed at a team meeting, but no extra effort was applied. One member of the team has failed to contribute to the project or communicate with the rest of the group for the final 6 weeks.

### Lessons learnt

The difficulties previously mentioned have taught the team to confidently and assertively communicate requirements amongst each other. Whilst a lack of participation from members of the team is frustrating, complaining about this and initiating arguments distracts the remaining members and prevents them from working, so it was not in the interest of the group to do so.

Additionally, the team has learnt that it is better to overestimate time requirements for aspects of the project where there is a lack of experience. This was the first project that the team has used scrum cards for, and this inexperience meant that they were naïve when allotting time for certain tasks.

Overall, splitting the team into sub-teams was an ineffective method of work distribution, especially when factoring in failed scrum estimates. If carried out again, improvements would be made to the planning of the project, such as distributing a single sprint task to each person at any one time, so that team members would be more interchangeable between project applications. This would prevent sections of the project from falling behind, and would also increase team familiarity with the entirety of the project, improving the overall efficiency.

## Risk Analysis Review

|  |  |  |  |
| --- | --- | --- | --- |
| **Risk** | **Probability** | **Impact** | **Course of Action** |
| Loss of small portions of project. | Medium | Low | Use repository to ensure up to date version accessible. |
| Loss of entire project. | Low | High | Keep numerous backups. |
| Equipment loss/damage. | Medium | Low | Use open access equipment. |
| Fall behind sprints/work takes longer than anticipated. | Medium | Medium | Assess the situation and use time more efficiently to maximise productivity. |
| Team member stops contributing for any reason. | Low | Low | Accept risk until risk occurs and redistribute work accordingly. |
| Team lacks specific knowledge and/or skills to complete segment of assignment. | High | High | Accept that the assignment is part of a learning process and that we must develop and improve upon our current skill sets within the team. |
| User’s requirements change after they are gathered. | Medium | Medium | Keep logs of feedback from user with dates feedback was given. If the user changes their mind, then ask them to explain why they have changed their mind and re-adjust accordingly. Should reduce probability because the user may remember why they wanted the original requirements, will remove undesirable features to reduce impact on final product. |

* The scope was set too high to accomplish everything within the time period allowed. This was partly caused by team members not applying themselves fully to the project. This was also in part caused by setting extremely high initial standards and targets.
  + In hindsight; reduce the “minimal” scope of the project and include a section of “optional” features for after this minimal scope is fulfilled.
  + Team members failing to commit themselves to the project is something beyond the control of the collective group.
* Some small segments of the code were lost at the start of the project due to inexperience with version control software. Once team members had gained experience using version control, these issues dissipated.
  + In hindsight; separating applications out into separate branches would reduce the amount of merge conflicts and thus reduce loss of work.
* At no point was the project lost, though keeping backups would still be advised.
  + In hindsight; whilst keeping backups has proven unnecessary during this project, it has taken minimal time or effort to do so, and therefore isn’t an excessive technique to use.
* During the project the team experienced technical difficulties with a laptop that was being used to implement part of the project. Whilst the laptop was being repaired the team used the computer labs on campus to maintain work output.
  + In hindsight; this was a very reasonable risk to make an allowance for and whilst the course of action may seem like a trivial alternative to put into place, the team knew exactly what to expect when the risk did occur.
* As the project progressed, the team improved on assessing the pace required to accurately distribute work for sprints. However, individual tasks within a sprint often took a different amount of time than anticipated, despite the sprint balancing out.
  + In hindsight; allowing a set amount of time on top of each task within a sprint to fix any bugs would be effective in managing the time with optimal efficiency. The main reason for tasks taking longer than expected was due to the necessity of testing multiple features to ensure that the applications are all working as anticipated when adding a new feature.
* During the project two of the team contributed considerably less hours per week than the guidelines suggest are required (as detailed in the project diary[[2]](#footnote-2)). The other three team members redistributed work on a weekly basis, and increased the number of hours committed to the project to accommodate for this issue.
  + In hindsight; this is a risk that the group as a whole were unable to prevent, due to each member only being accountable for their own actions. At the beginning of the project all members appeared fully committed, but the attitudes of some changed as time elapsed. Individuals can only focus on their own input and hope that the rest of the group does the same.
* As predicted within the risk assessment, the team did encounter tasks where they lacked specific knowledge to implement certain requirements of the project specifications (the UML diagrams required had very limited teaching time allocated to them, and the Computer Science course lacks any security-based teaching within any of the modules). The team were able to do some research into these areas, and seek clarification on whether a source was accurate where necessary.
  + In hindsight; this risk is an inevitability in any project that seeks to compete within any industry in the world. Planning research as early as possible would make it easier to distribute research tasks amongst the team and ensure that there is an equal workload throughout.
* The project encountered changes to the user requirements, which was also accounted for within the risk assessment. Fortunately for the progress of the assignment, end user requirements were clarified before any implementation was made on the feature discussed. The client also requested a particular function was not included within the mobile application because “it simply wouldn’t work for a business operating in the real world.”
  + In hindsight, this risk is an inevitability in any project. The group did well to write down the exact requirements of the end user and the dates that they supplied them. It would also be prudent to get the end user to sign the minutes of the meeting before concluding so that they are unable to deny making the requests.

## Project Plan and Implementation Comparison

Planning for any project is essential, but ensuring that plans are fluid and can be adjusted in an agile way is even more important. Throughout the implementation of the project plans were altered, in particular the roles assigned to members of the team.

Development of the website became a larger task than initially anticipated. Implementing a minimum viable product was swiftly achieved, but the team focused on getting a system that was using the database as soon as possible, leaving it open to scrutiny during the usability demonstrations. This highlighted the importance of spending time on the aesthetics of the website – developing a website that isn’t intuitive for a customer to use is going to alienate that customer and potentially cost the business money. This meant that more time would be required to implement the website, and in turn team members were unable to work on other applications within the project for as long as required. As a result, the optional features discussed in the initial planning stages such as “half and half” custom pizzas weren’t implemented.

Overall, the implementation appears to be a minimum viable product version of the plan – all of the features implemented perform well and have had known bugs associated with them fixed as and when they have been discovered, but there are several features that were out of scope for the time period allowed for the project.

This lack of desired progress was no doubt partially caused by some of the team’s failure to contribute, but the limited use of project management software and an absence of group management can also be deemed causal. Self-motivation was key in the performance of the three group members that contributed the most to this project, as they would actively seek out functionalities that were yet to be implemented and begin to work on them. Writing detailed commit messages on the source control aided with this, but Trello was an underutilised technology throughout, and was often an afterthought to keep others up to date on what has been finished.

# Requirements Analysis and Design

## Functional and non-functional requirements

### Functional requirements

Manager application

* Ability to add, edit and delete menu items.
* Ability to add, edit and delete discounts.
* Ability to add, edit and delete new staff members.
* Ability to sort orders by status, view specific details of orders and void orders if necessary.
* Ability to view sales information – popular items, popular days, popular times of day etc.

Chef application

* Ability to view and advance orders from “Placed” to “Ready for Delivery”.

Mobile application

* Ability to view and accept orders that are “Ready for Delivery”.
* Ability to advance accepted deliveries to either “Delivered” or “Failed Delivery”.
* Option for directions from driver's current location to delivery address.
* Notifies the nearest available driver of deliveries – *requirement removed by end user Mark Dixon.*

Website

* Ability to register a new customer account.
* Ability to edit account details for an existing account.
* Ability to browse menu items and deals.
* Ability to add menu items to an order.
* Ability to review order items before purchase.
* Ability to place desired order.
* Ability to pay for order.
* Ability to view order status of placed order.
* Ability to view previous orders placed on the account.
* Manager login to manage menus.

### Non-functional requirements

Manager application

* Ability to keep unauthorised users out.
* Application is easy to use.
* Must be prompt when responding to user actions.

Chef application

* Application displays up-to-date list of orders without any action required of the user.
* Application is easy to use.
* Must be prompt when responding to user actions.

Mobile application

* Ability to keep unauthorised users out.
* Application is easy to use - especially button sizes.
* Must be prompt when responding to user actions.

Website

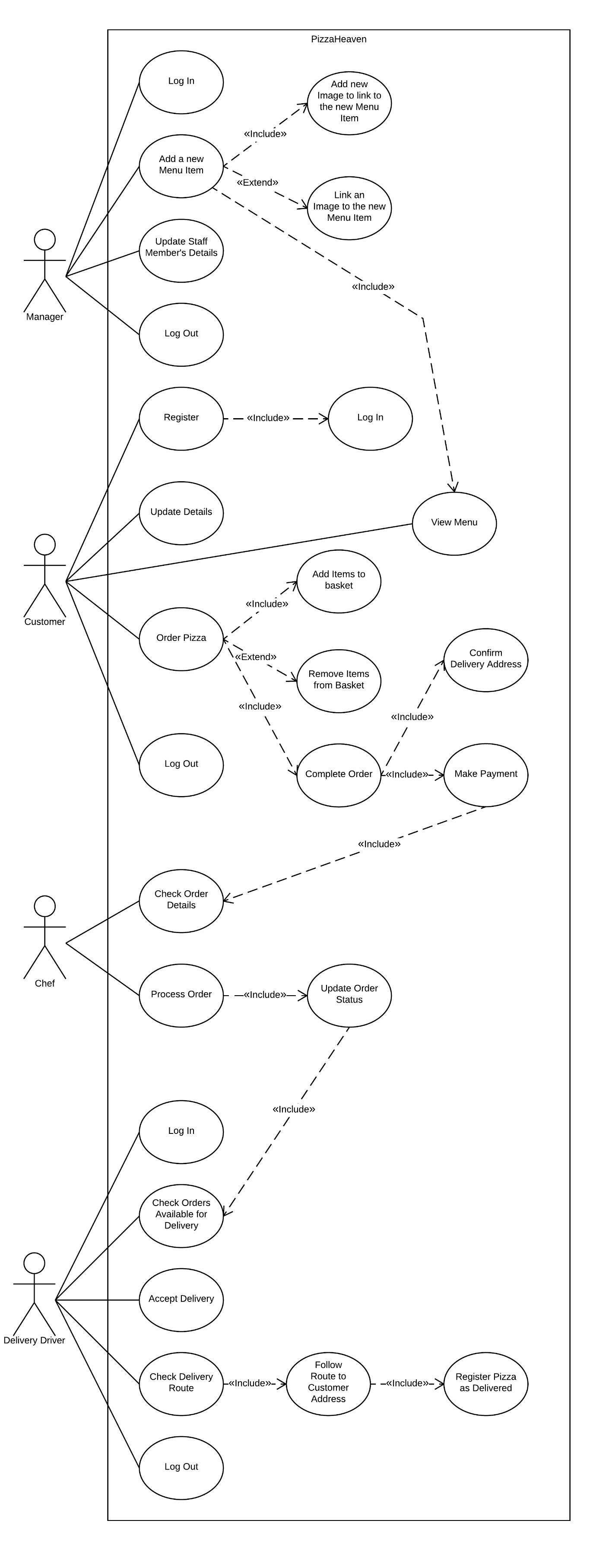
* Ability to secure sensitive information after the user is finished with the site.
* Display up-to-date list of menu items and discounts from database.
* Website is easy to use.
* Must be prompt when responding to user actions.
* Must ensure user inputs enough information before they are allowed to make an order.

## Analysis and Design

### Use Case Model

The next page shows the flow of data in our use cases from start to finish in a single use case diagram. It also shows a complete single iteration of an order within the system. The manager application can post menu items that appear on the next refresh of the website menu system for the customer to order. Confirmed customer orders are posted to the database which are in turn loaded by the kitchen application. When progressed to the “Ready for Delivery” state, these orders are removed from the view of the kitchen application and added to the delivery application to notify drivers that they are ready for delivery. When accepted by a delivery driver then the order is removed from all other delivery driver’s view. An option of directions to the customer’s address from the driver’s current location is given to the delivery driver.

There is also a second view for the kitchen application that only displays orders that are “Ready for Delivery”. This is view intended for the front of the store to show staff exactly what orders are available for collection by a delivery driver at any given time, without access to the mobile application.



### User Stories and Acceptance Tests

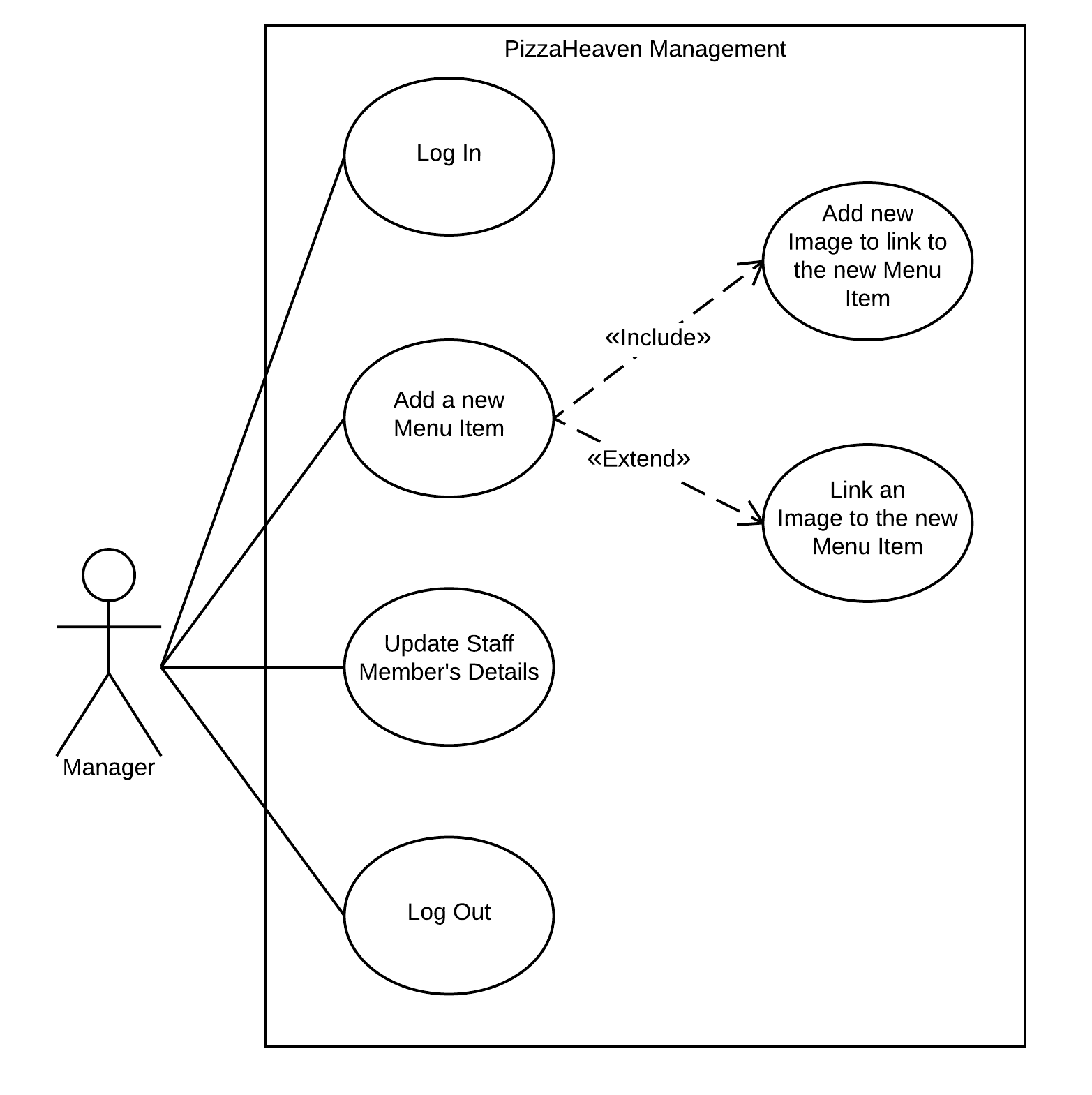
* As a manager, I want to add menu items so that the business can keep up with competitors.
  1. Able to add menu items from manager application.
* As a manager, I want to edit menu items so that prices can be raised to match inflation.
  1. Able to edit menu items from manager application.
* As a manager, I want to delete menu items so that unpopular products can be removed.
  1. Able to delete menu items from manager application.
* As a manager, I want to add discounts to encourage customers to purchase products.
  1. Able to add new discounts from manager application.
* As a manager, I want to edit discounts so that the company can be reactive to real world situations.
  1. Able to edit discounts from manager application.
* As a manager, I want to delete discounts so that they aren't permanent.
  1. Able to remove discounts from manager application.
* As a manager, I want to add new staff members so that new delivery staff accounts can be set up.
  1. Able to add new staff from manager application.
* As a manager, I want to edit staff accounts so that details can be kept up to date.
  1. Able to edit staff from manager application.
* As a manager, I want to delete staff members from the system so that I can close accounts of former members of staff.
  1. Able to delete staff from manager application.
* As a manager, I want to view orders with a particular status so that I can see details about unsuccessful orders.
  1. Able to view a list of orders from manager application.
  2. Able to query orders by order status with drop-down menu.
* As a manager, I want to pull up order details so that I can see specific details about them.
  1. Able to view order details from manager application.
* As a manager, I want to void orders so that I can give a refund.
  1. Able to void order from order details screen whilst on manager application.
* As a chef, I want to view orders that are relevant to me so that I can prepare them.
  1. Able to view all orders on chef application.
  2. Orders on chef application filtered to show "Placed" and "Prep" statuses only.
* As a chef, I want to advance orders as I work on them so that the delivery driver knows when to collect them.
  1. Able to advance order that is "Placed" to "Prep" when chef starts working on it.
  2. Able to advance order that is "Prep" to "Ready for Delivery" when chef finishes working on it.
* As a delivery driver, I want to view orders that are relevant to me so that I can accept and deliver them.
  1. Able to view all orders on delivery application.
  2. Orders on delivery application filtered to show "Ready for Delivery".
* As a delivery driver, I want to accept orders for delivery so that I can take them to the customer.
  1. Able to claim deliveries by clicking "Claim" button.
* As a delivery driver, I want to advance orders that I have completed so that I can let my manager know I have finished that delivery.
  1. Able to click "Delivered" button if the order is given to the customer as expected.
  2. Able to click "Failed" button if the order cannot be delivered to the customer as expected.
* As a delivery driver, I want to be provided directions to a customer's address so that I can deliver to unfamiliar areas.
  1. Able to click "Launch Map" to launch Google Map application.
  2. Feeds current location to Google Map application.
  3. Feeds destination to Google Map application.
* As a customer, I want to register a new account so that I can make orders.
  1. Account registration page exists with all fields required by the database.
  2. Page sends details to servlet to construct customer objects that are passed to the database via the API.
  3. Page has input validation to ensure that customer inputs account details that are within expected sizes and formats.
* As a customer, I want to edit my account details so that I don't need a new account when I change address.
  1. Account edit page exists with all fields required by database.
  2. Compares stored password with input password to ensure authorisation.
  3. Page sends details to servlet to construct customer objects that are passed to the database via the API.
  4. Page has input validation to ensure that customer inputs account details that are within expected sizes and formats.
* As a customer, I want to browse menu items so that I can find the food I want to order.
  1. Menu pages pull database contents to dynamically and meaningfully display menu information within it.
* As a customer, I want to browse discounts so that I can get the best value for money.
  1. Discount page pulls database contents to dynamically and meaningfully display discount information within it.
* As a customer, I want to add menu items to my order so that I can purchase them.
  1. "Add to basket" button associated with each menu button that creates a list of items that the customer wants to order.
  2. "Quantity" select input that allows customer to change quantity added in a single click.
* As a customer, I want to review my order before placing it so that I don't forget anything or order something that I don't want.
  1. Basket page exists to display all items that the customer has selected to order.
  2. "Remove from basket" button to take an item off of the order entirely.
  3. "Plus" and "minus" buttons to adjust basket quantities.
* As a customer, I want to place my order so that I can receive my food.
  1. Ability to click "place order" button so that customer order can be sent to the database and notify the store of the order being placed.
* As a customer, I want to be able to pay for my order online so that I don't have to pay cash when it arrives.
  1. "Place order" replaced with "cash on delivery" option, payment method set to "cash on delivery" if this option is selected.
  2. "PayPal" button integrated within website, feeds back to same order servlet as cash on delivery button but sets payment method to "PayPal".
* As a customer, I want to view the status of my order so that I can know when it is about to be delivered.
  1. After order has been placed, the browser redirects to a status page that refreshes every 30 seconds and re-pulls information in database. Database updated by store staff as order is received/cooked/sent for delivery etc.
* As a manager, I want to keep unauthorised users out of the management application to protect the business from attack.
  1. Username and password set up for each manager account.
  2. When the application is opened a correct username and password pair is required.
* As a manager, I want the application to be easy to use so that I don't get confused.
  1. Implement improvements to graphical user interfaces based on feedback from HCI expert meeting, end-user meetings and mock and live usability demonstrations.
* As a manager, I want the system to be prompt when responding to user actions so that I don't accidentally do something that I don't want to.
  1. Code is efficient so response times aren't noticeable to users.
* As a chef, I want the application to display an up-to-date list of orders without having to act myself so that I can focus on preparing food.
  1. Application re-pulls orders from API every 1 second and redraws graphical user interface.
* As a chef, I want the system to be prompt responding to my actions so that I can continue focusing on my food preparation.
  1. Code is efficient so response times aren't noticeable to users.
* As a delivery driver, I want to keep other users off my account so that I don't get in trouble for their mistakes.
  1. Username and password set up for each delivery driver account.
  2. When the application is opened a correct username and password pair is required.
* As a delivery driver, I want the application to be easy to use so that I don't waste time that I could be delivering pizzas.
  1. Implement improvements to graphical user interfaces based on feedback from HCI expert meeting, end-user meetings and mock and live usability demonstrations.
* As a delivery driver, I want the application to be prompt when responding to my actions so that I don't waste time waiting whilst I could be delivering pizzas.
  1. Code is efficient so response times aren't noticeable to users.
* As a customer, I want my personal data to be safe and secure so that I am not the victim of fraud.
  1. Information is encrypted before being sent to the customer table and only shown in plaintext:
     1. When necessary (for deliveries/for the customer themselves).
     2. When a valid username and password combination are input.
* As a customer, I want the website to be up to date so that I see the correct prices and get the latest discounts.
  1. Each time the menu page is loaded, the servlet responsible for pulling menu information from the database makes a new connection and pulls fresh information.
* As a customer, I want the website to be easy to use so that I can order my food as quickly as I desire.
  1. Implement improvements to graphical user interfaces based on feedback from HCI expert meeting, end-user meetings and mock and live usability demonstrations.
* As a customer, I want the website to be responsive to my actions so that I get the correct order.
  1. Code is efficient so response times aren't noticeable to users.
* As a customer, I want the system to have the full information before charging me for my order so that I actually receive my food.
  1. Set mandatory fields in database for minimum amount of information required by the system to get the customer's order to them.
  2. Force website to fill mandatory fields before passing to servlet that is responsible for sending it to the API.

### Product Backlog

The user requirements that are additional to the brief of the assignment (and therefore summarised in the functional and non-functional requirements list), or required clarification from the end user are as follows:

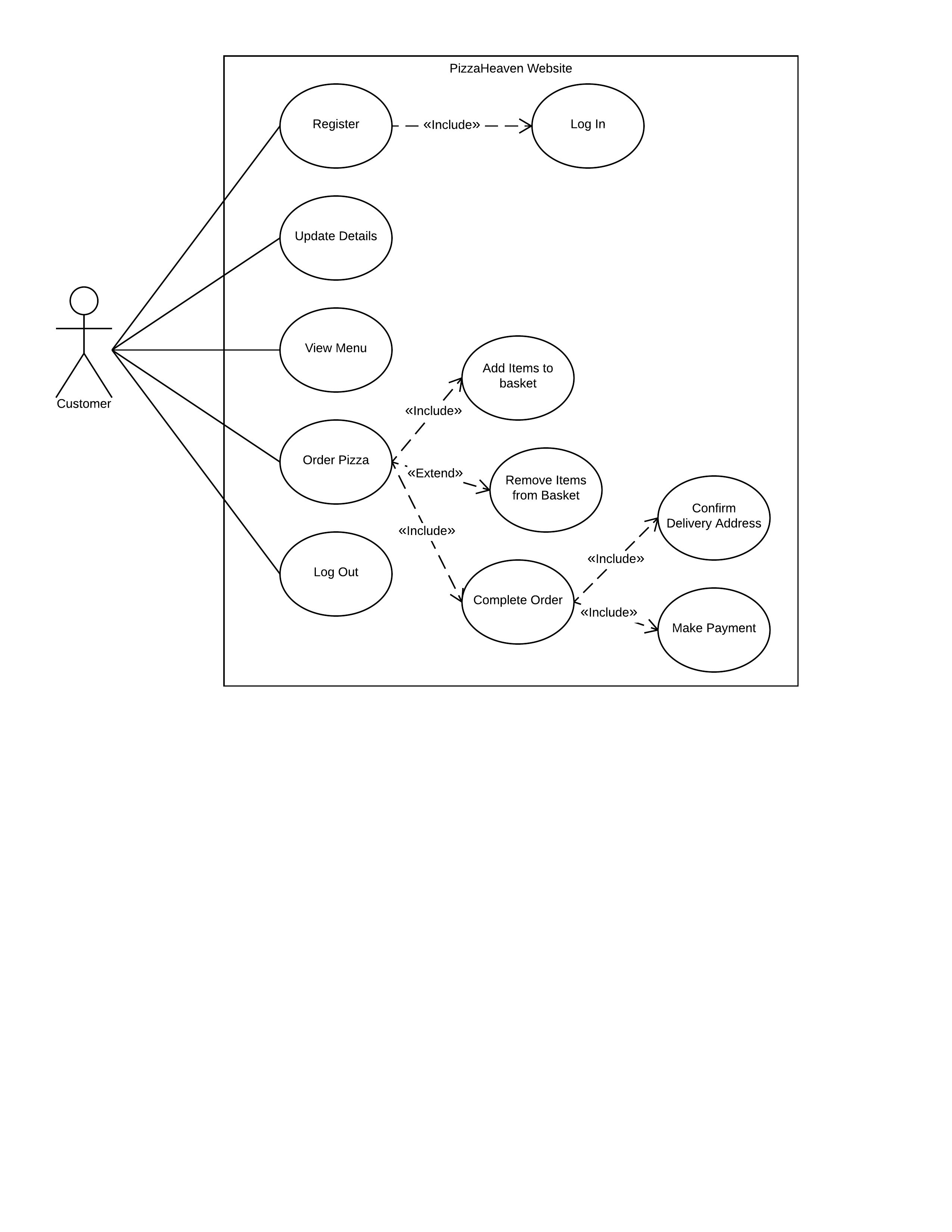
* No form of stock control is required in the application.
* The application is for a single store only.
* Sides and drinks are required as well as pizzas.
* No specific colour scheme/branding is required.
* Implement both percentage based discounts that are based on money spent, and allow the manager to create meal deals that give discounted product deals.
* Website should allow cash on delivery with no maximum value. The manager will be intelligent enough to deal with large quantities as the orders are made.
* The store offers a 45 minute money-back guarantee.
* The store does not have a reward scheme and there is no need to implement one within the system as it isn't something that falls within current plans.
* I would like to see popular items on the sales statistics screen.
* If the customer receives an incorrect order or the items arrive in an unsatisfactory condition, then the customer can have a fresh version of the same order or their money back.
* Specific drivers should not be offered deliveries by the system as the brief suggests – the customers collect deliveries from the store and accept them as they do, then carry out the delivery. If something happens to the driver before they return to the store for a delivery that has already been accepted then issues will arise so it is better to only have drivers manually accept them.
* We offer three different sizes of pizzas. Small, medium and large. We may change the size throughout the course of the system's life so that will be input in the description of the item.

### Use Cases



The manager logs in and adds a new menu item to the graphical user interface. They then add an image name to the menu item to link it to an image stored within the database with the same name before posting the menu item itself to the database. If there is no suitable image stored within the database then the manager can select a desired image, configure it for the system and post it to the database before posting the menu item itself.

The manager can then switch to the staff tab and update any details stored about a staff member. The manager then logs out of the system to ensure that unauthorised users have no access to the system.



The customer’s web browser opens the Pizza Heaven website homepage and the customer decides to register since they haven’t registered for an account. They navigate to the registration page and input their personal information, guided by prompts if any invalid information has been entered. Upon submitting their registration, they are logged into their account.

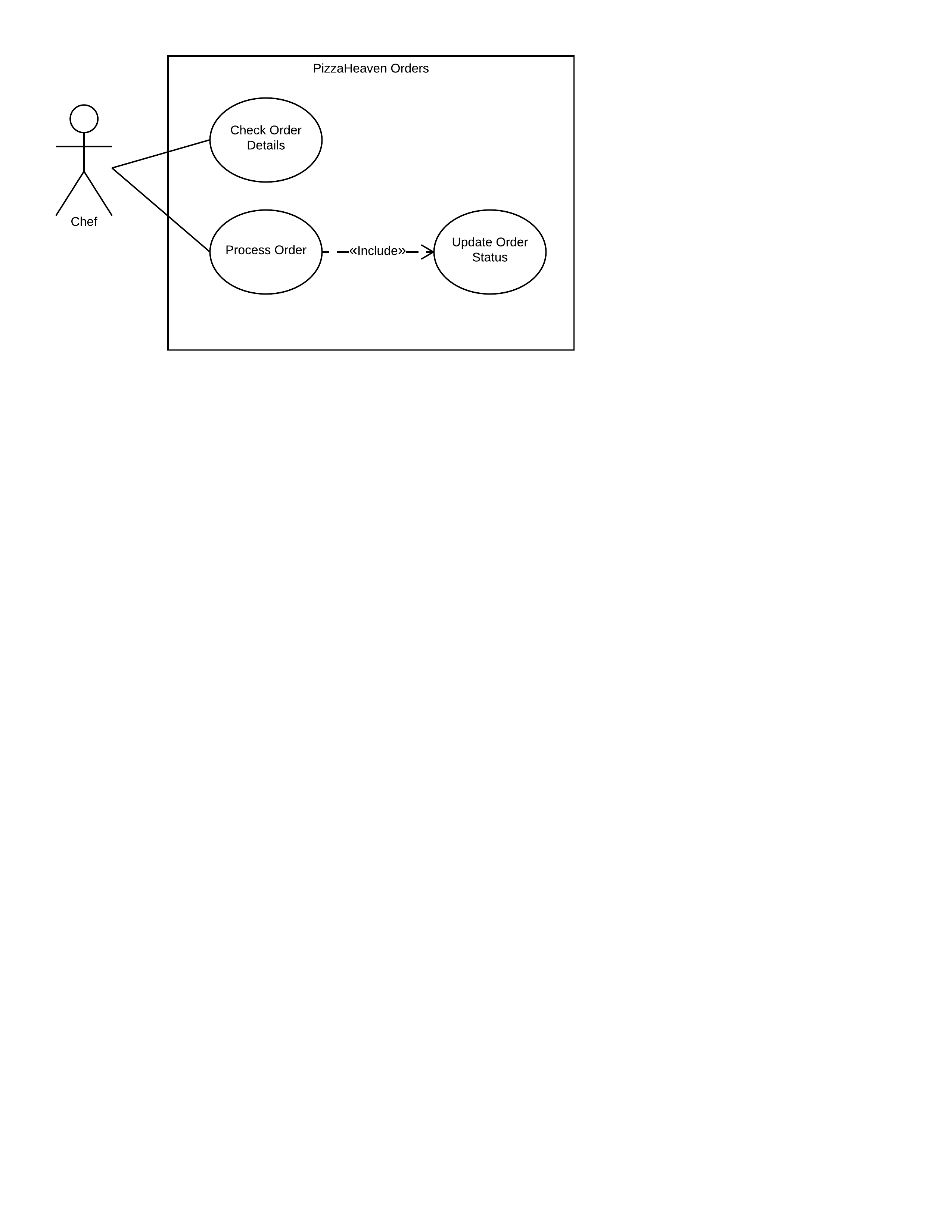
The customer then realises that they have input the incorrect address and so they navigate to the account page and update their details.

The customer begins to browse the menu and adds items to the basket at their discretion. When they have exhausted the menu they proceed to navigate to the basket screen and check their order, increasing and decreasing quantities as required, and removing items from the basket completely if they are no longer desired.

Content that their order is correct, the customer clicks “Proceed to Delivery” to check the delivery address and select a payment option.

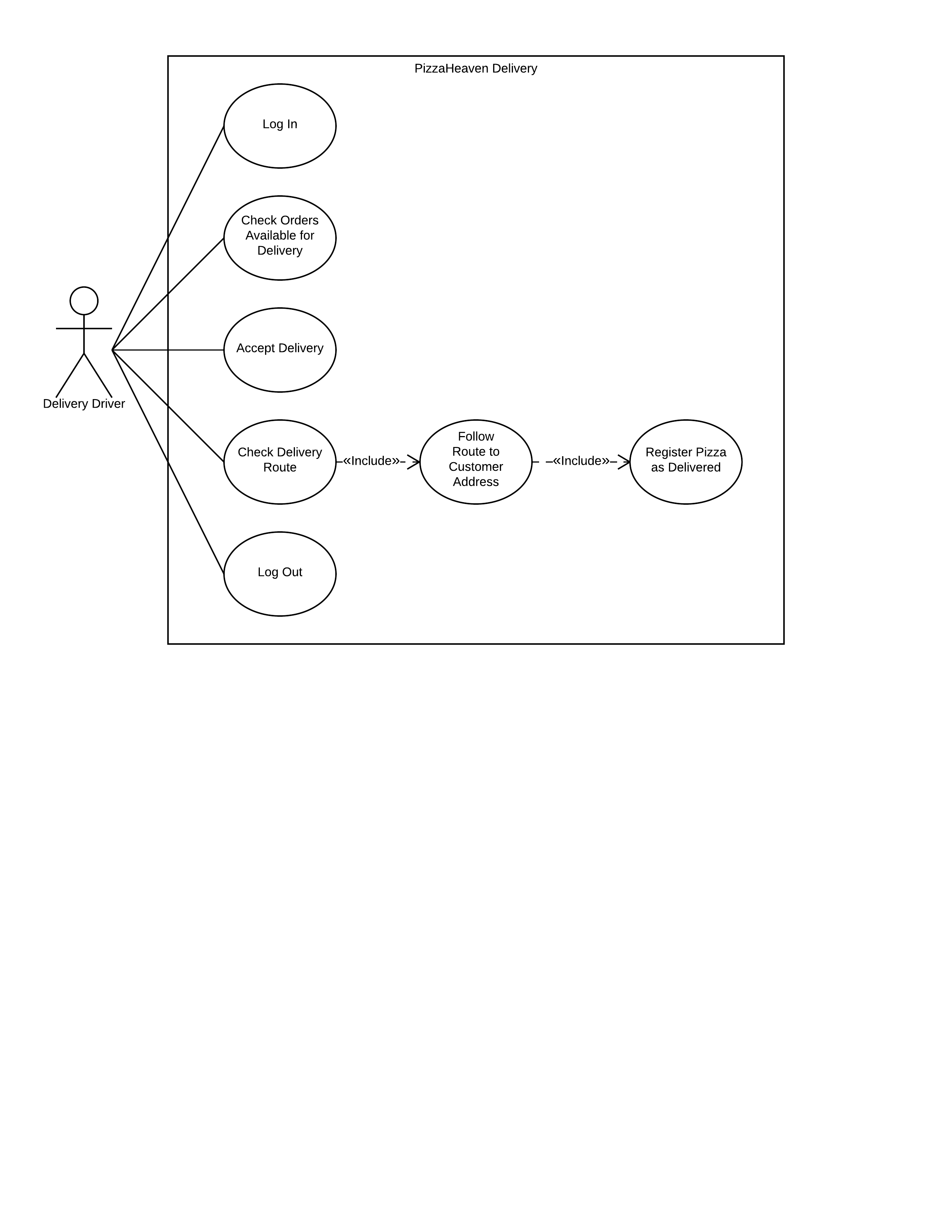
Regardless of payment option (assuming that they fill out the information that PayPal needs correctly) they end up on the order status page. It is assumed that if they cancel their PayPal payment then they have forgotten an item on their order and so they get redirected back to the basket, which still remembers the order.

The order status page shows details about every active order associated with the customer’s account. An active order is any order that has a status that isn’t “Delivered” or “Void”.



Upon opening the kitchen desktop application the view displays all orders that have a status “Placed” or “Prep”. The chef selects an order and clicks “Progress Order” which updates the status from “Placed” to “Prep” or from “Prep” to “Ready for Delivery” dependent on the current status. Orders that advance to “Ready for Delivery” are no longer a concern of the chef and so they are removed from the kitchen view.

There is a second view attached to the kitchen desktop application that shows orders that have the status “Ready for Delivery” or “Out for Delivery”. This view is intended to advise shop staff of what orders need to be sent out for delivery without access to the mobile application.



The delivery driver logs in to the application and checks through the list of orders that have the “Ready for Delivery” status and any orders that are “Out for Delivery” that are assigned to the account that is currently logged in.

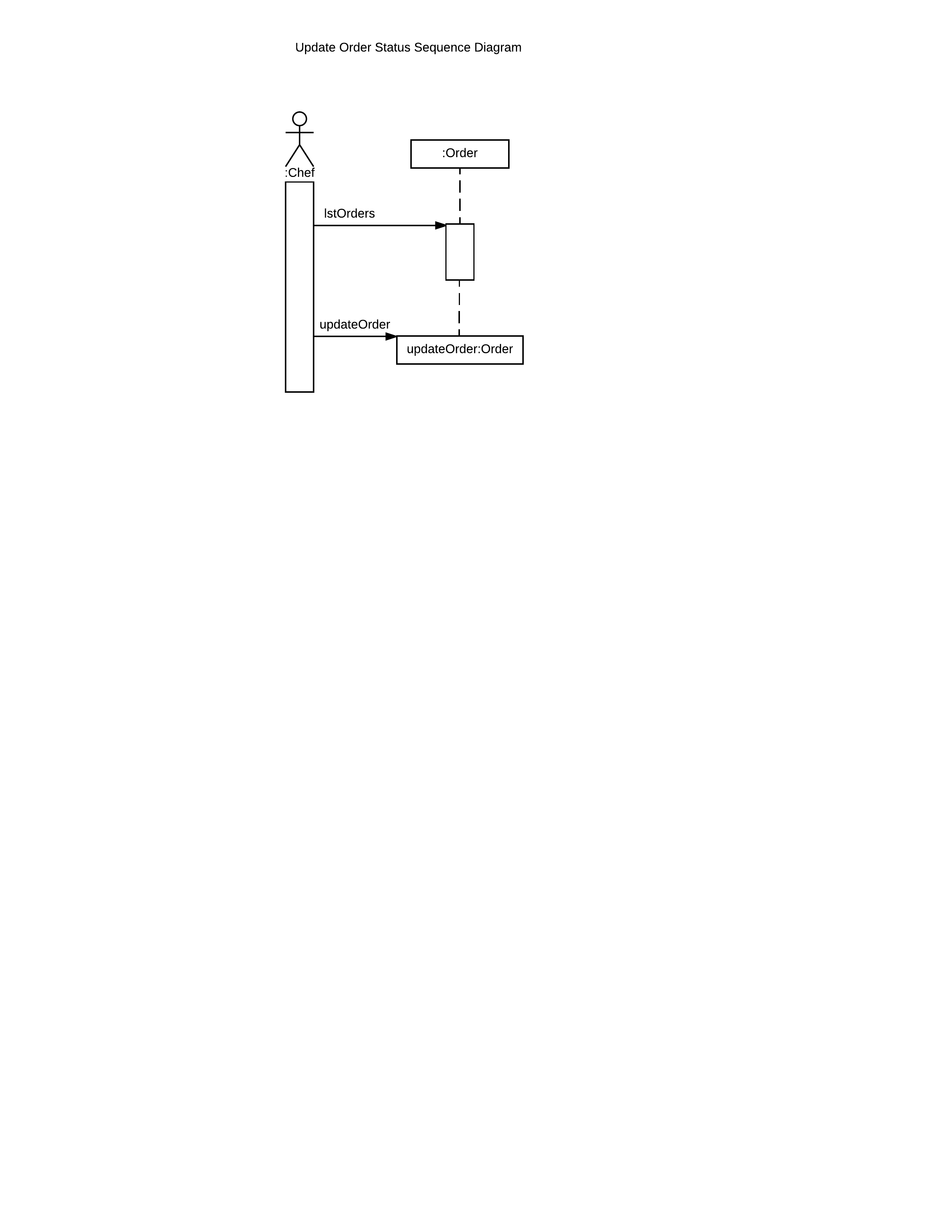
When the user has found an acceptable delivery or deliveries they click the “Claim Delivery” button and the application assigns the order to the logged in account and sets the order status to “Out for Delivery” (which removes it from any other mobile application user’s screen).

If the driver wishes to be given directions to the customer’s address then they can click the “Launch Map” button which launches Google Map application which then gives the user the fastest route between the mobile device’s current location and the customer’s address.

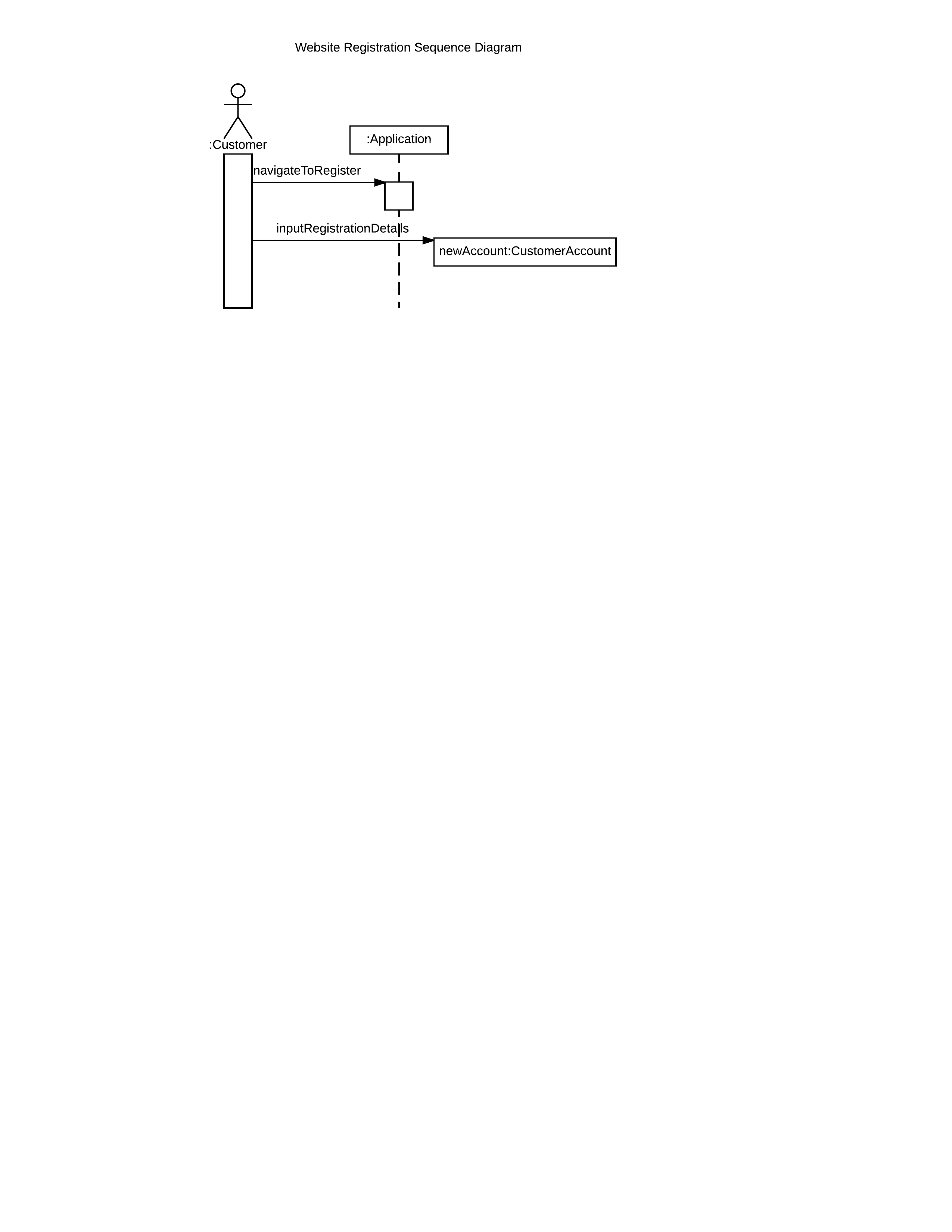
### Sequence Diagrams



The manager logs into the system with the correct login details then clicks on the “Menu Item” tab. The system lists the menu items and the manager clicks “Add” for whichever menu item to be added. Once the menu item details have been added, the manager selects “Add New Image” to add an image to the system before linking it to the new menu item using the image name.

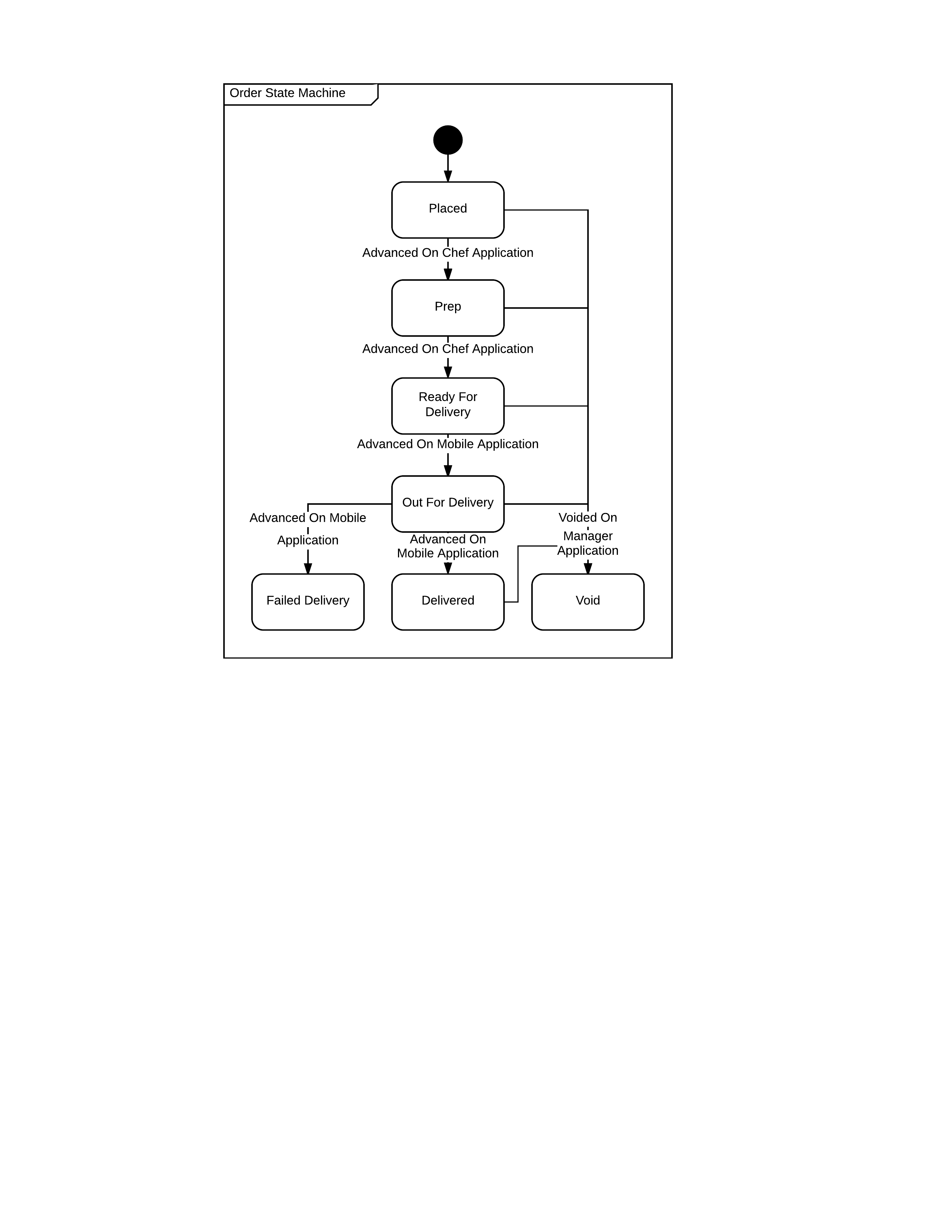


Once the chef application is started it lists the orders relevant to the chef view. As the chef works on preparing orders they press the update button to progress it to the next order status.

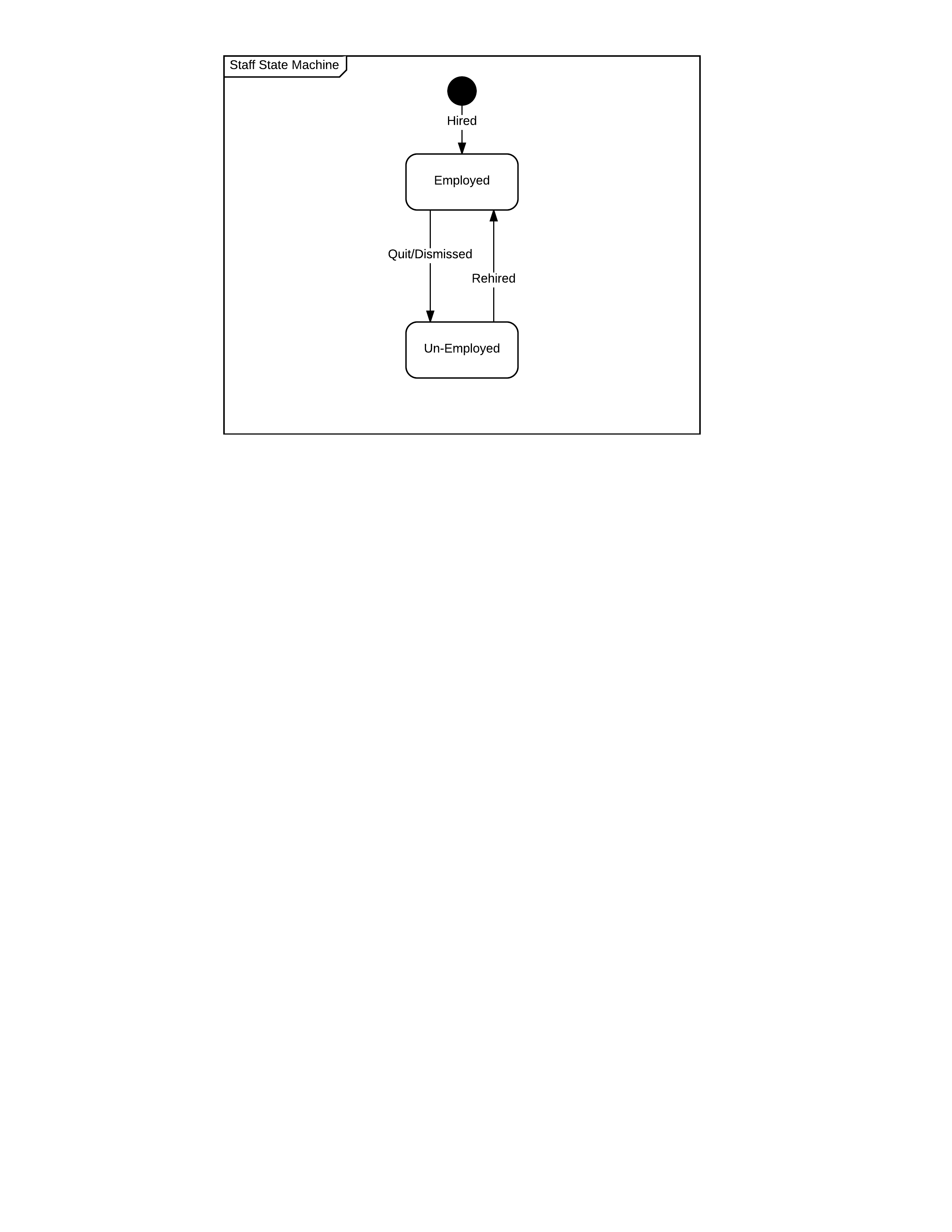


Once the customer reaches the website, they navigate to the registration page, input valid details and a new customer account is created and stored in the customer table.

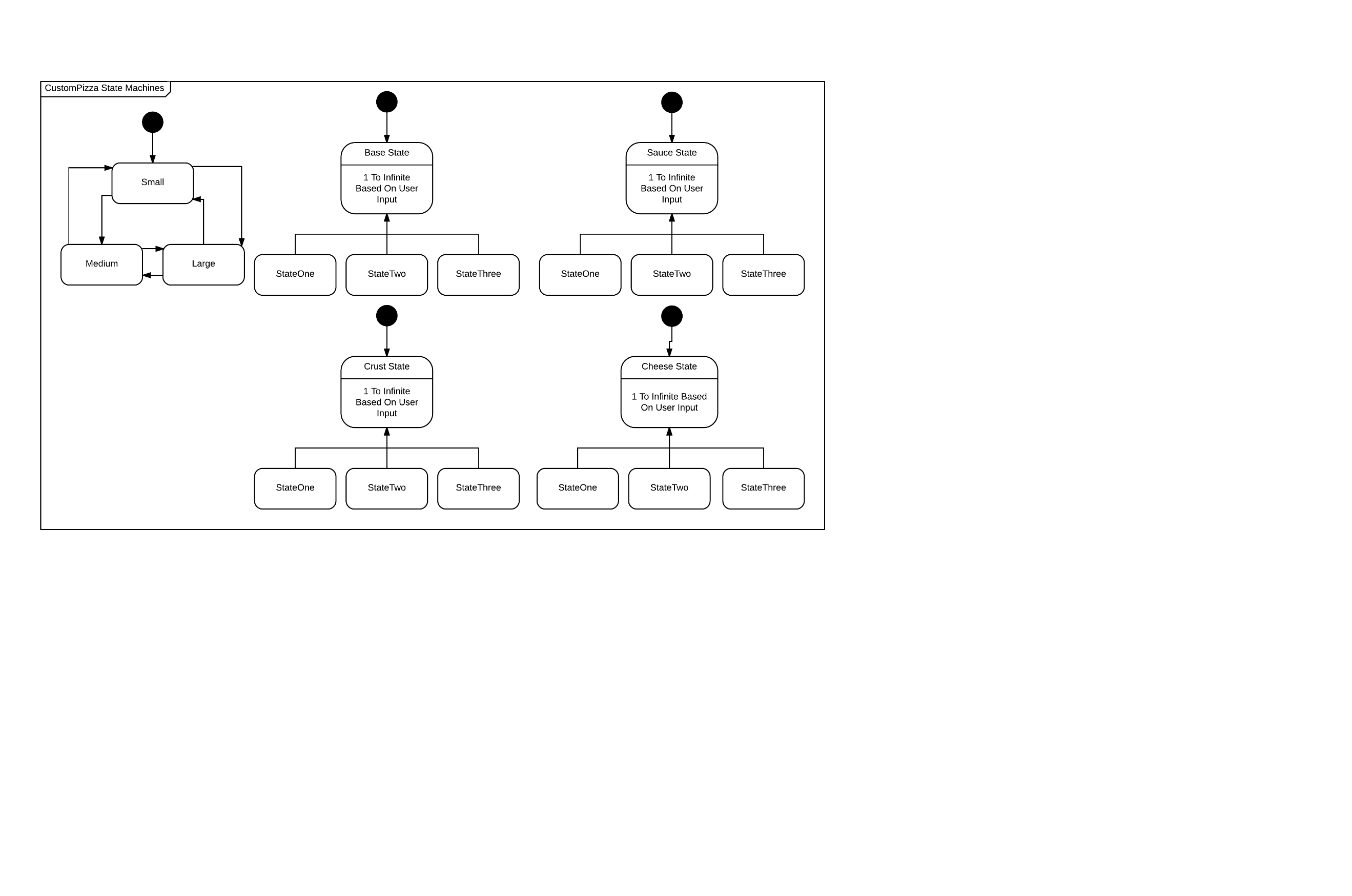
### State Machine Diagrams



An order progresses through various states: initially it is "Placed" by the customer, then it is advanced to "Prep" by the chef, followed by "Ready for Delivery" when the chef has finished cooking and gathering the contents of the order. The delivery driver claims it and takes it "Out for Delivery" and finally declares that it is "Delivered" to the customer. If the customer isn't there to accept it, the driver notifies the system it has "Failed Delivery”. At any point during this process the manager can declare the order "Void".

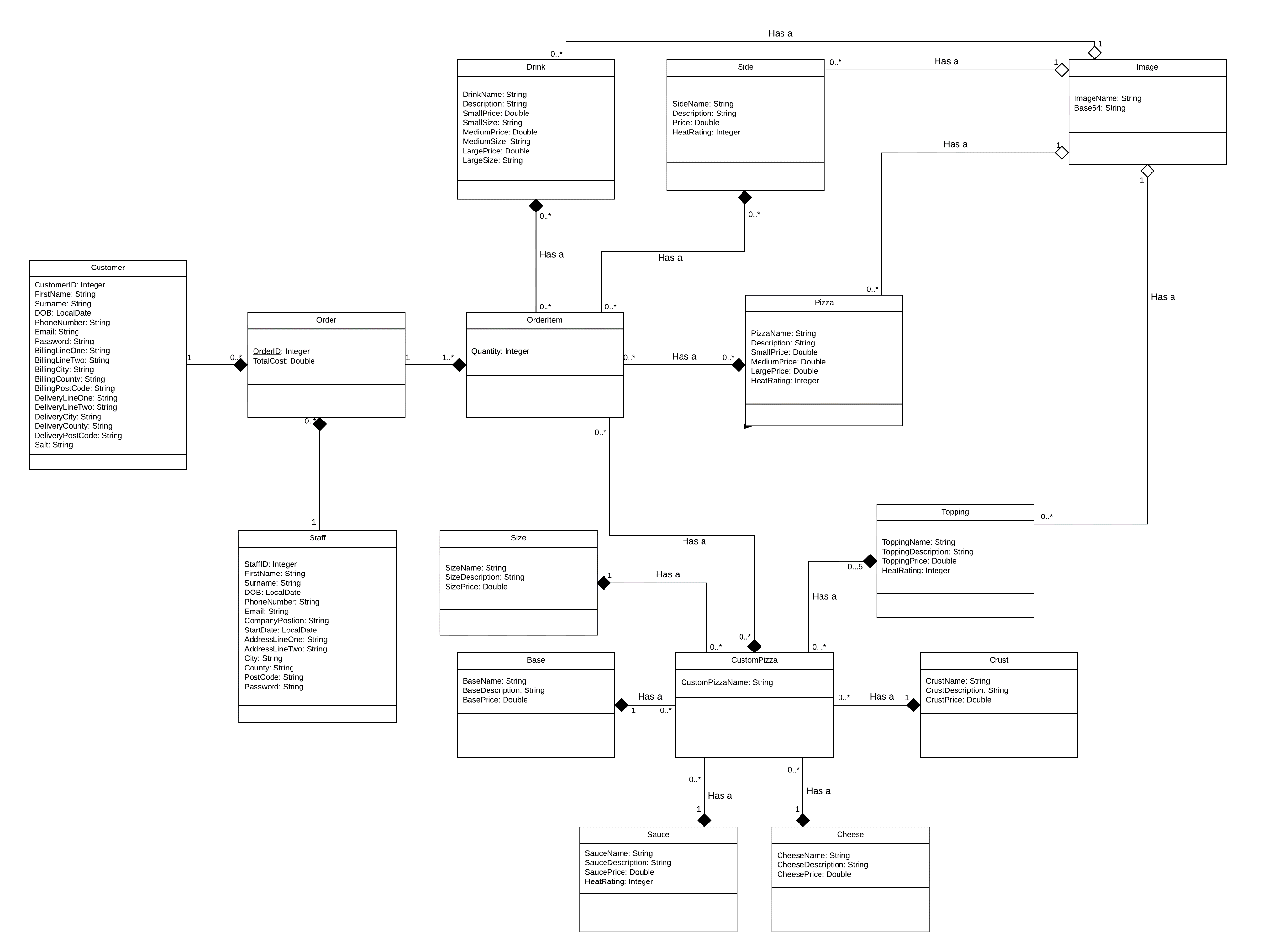


A member of the public can be hired and become "Employed" as a member of staff. This member of staff can quit or be dismissed to be "Un-Employed". If they are rehired, they revert to "Employed".



The lookup tables within the custom pizza table are effectively states. When designing the custom pizza, the customer selects a radio button to initialise a component as a particular state. The size has the state of small, medium or large. The base state dictates what type of base the custom pizza has. The sauce state dictates what type of sauce the custom pizza has. The crust state dictates what type of crust the custom pizza has. The cheese state dictates the type of cheese the custom pizza has.

### Class Diagram



## Evaluation of design changes

There have been several changes between the design and implementation of the project. These can be categorised into four main reasons for occurring; requests from the client, requests from the usability demonstrations and HCI expert, issues between theory and practice, and finally, time issues.

Naturally when working for any client (particularly if they have a limited understanding of software development), their requirements can and will change. For an extensive list of what changes have been made based on the client's requirements, see the product backlog. The change that made the most impact on the project was their request to not have the nearest delivery driver notified by the mobile application when an order is ready for delivery. This removed a section of the project entirely and as such resulted in its removal from the initial design. The next largest change caused by the client would be their request to not include a reward scheme. Our initial designs included an, in hindsight, out of scope reward points system, based on the experience and research that large pizza takeaway chains do normally offer a reward scheme to earn a "free" order. A final notable change between the planning, design and implementation that was caused by the end user would be their insistence on the software only needing to be designed for a single store. Inclusion of multiple stores would have required either the customer to select the closest store to them, or the website being able to determine which store to send the order to, based on the one that closest to the customer's address.

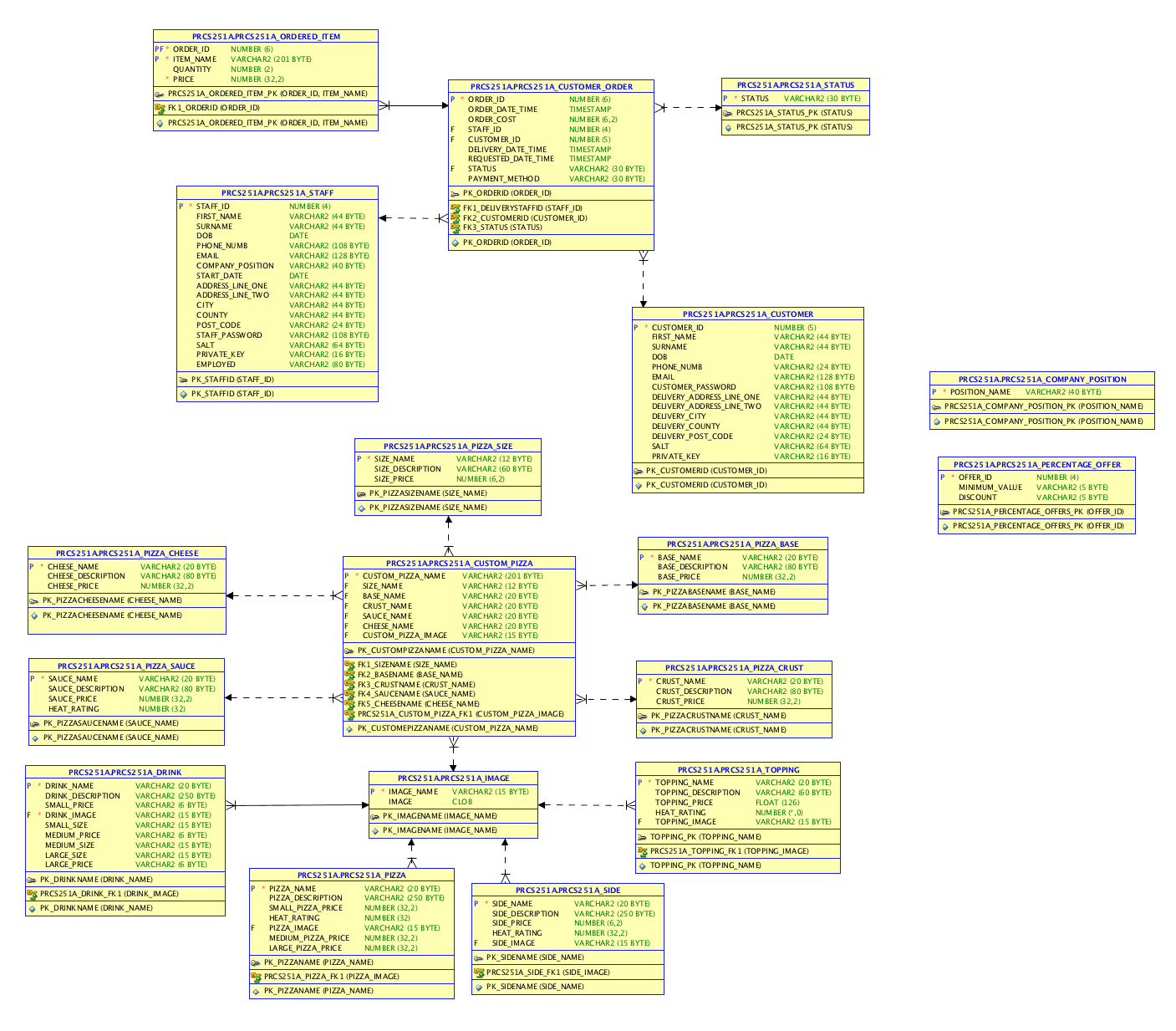
Most of the changes made to the system following the feedback from usability demonstrations and the HCI expert were minor interface changes that had very little impact to the entire design of the assignment, however, the HCI expert did insist that one of the features of the menu system be removed completely. Original plans contained the option to take a menu pizza and click a "customise" button, which would pre-populate a custom pizza with that pizza's size, base, crust, sauce, cheese and topping options and then allow the customer to edit the chosen menu pizza. When planned, it was meant to represent an ease-of-use feature that allows the customer to remove a disliked topping without selecting the 5 other pizza attributes first. When proposed to the HCI expert they gave the feedback that it would not be a useful feature, and it should be removed entirely. This meant that the pizza table in the database didn't need to be made up of the various tables that custom pizza is made up of, and in turn that the manager wouldn't be subjected to the same limitations as the customer when creating menu items e.g. mixing sauces or adding more toppings than the customer is allowed.

An overlooked design flaw within the database changed the way that the database worked, but not the way that the product itself functions. "Order Item" always contains the name of a menu item and a customer ID as a compound key, but menu items are split into different tables because a pizza will have different attributes stored against it than a drink. This made enforcing referential integrity impossible, because when a pizza was added as an order item, the database would error when it couldn't find the pizza in the drinks table. The website only displays menu items in the table, and the order is compared to the database before being posted to ensure that the menu item exists, so whilst integrity isn't enforced by the database, it is still enforced.

The design changes emphasise the need to develop using agile project management principles. The changes often made the task easier, but it is still important for the team to meet and discuss tasks regularly to ensure that every member is aware of any issues and everyone is working with the same knowledge of the system.

# Database Design

## Database Diagram



## Database Table Explanations

### Company Position

This is a lookup table for the role of staff members within the business. It is set up to enforce the roles of staff at the company to follow a convention. "Delivery Drivers" will always maintain the same position rather than some be given a slightly differently worded/spelt position.

CREATE TABLE "PRCS251A"."PRCS251A\_COMPANY\_POSITION"

( "POSITION\_NAME" VARCHAR2(40 BYTE) NOT NULL ENABLE,

CONSTRAINT "PRCS251A\_COMPANY\_POSITION\_PK" PRIMARY KEY ("POSITION\_NAME");

### Custom Pizza

The Custom Pizza table holds variants of custom pizzas created by customers. This table was created because originally the item name of menu items would act as a foreign key within the Ordered Item table. Currently it is somewhat excess to requirements because we were unable to implement the name field as a foreign key within Ordered Item table, as the database quite rightly expected the name to exist in all 4 menu item tables. For example, it would look for a pizza in the pizzas table, but also the sides, drinks and custom pizzas tables too. Each of the other fields act as a lookup to the relevant tables, allowing simply the name to be stored against each custom pizza, but it connects the entry to more information about the attribute.

CREATE TABLE "PRCS251A"."PRCS251A\_CUSTOM\_PIZZA"

( "CUSTOM\_PIZZA\_NAME" VARCHAR2(201 BYTE),

"SIZE\_NAME" VARCHAR2(12 BYTE),

"BASE\_NAME" VARCHAR2(20 BYTE),

"CRUST\_NAME" VARCHAR2(20 BYTE),

"SAUCE\_NAME" VARCHAR2(20 BYTE),

"CHEESE\_NAME" VARCHAR2(20 BYTE),

"CUSTOM\_PIZZA\_IMAGE" VARCHAR2(15 BYTE),

CONSTRAINT "PK\_CUSTOMPIZZANAME" PRIMARY KEY ("CUSTOM\_PIZZA\_NAME")

CONSTRAINT "FK5\_CHEESENAME" FOREIGN KEY ("CHEESE\_NAME")

CONSTRAINT "FK1\_SIZENAME" FOREIGN KEY ("SIZE\_NAME")

CONSTRAINT "FK2\_BASENAME" FOREIGN KEY ("BASE\_NAME")

CONSTRAINT "FK3\_CRUSTNAME" FOREIGN KEY ("CRUST\_NAME")

CONSTRAINT "FK4\_SAUCENAME" FOREIGN KEY ("SAUCE\_NAME")

CONSTRAINT "PRCS251A\_CUSTOM\_PIZZA\_FK1" FOREIGN KEY ("CUSTOM\_PIZZA\_IMAGE");

### Customer

This table holds customer account details. For a customer to log into the website they need an entry in this table. This entry is posted by the API when the customer inputs a valid entry into the registration page on the website (input validation exists on the page to ensure that expected values are entered). The size of fields are based on the size of strings after being encrypted by the security package.

Below the table create statement is the trigger create statement responsible for using the sequence that generates a unique Customer ID for each entry. Below this is the sequence create statement itself.

CREATE TABLE "PRCS251A"."PRCS251A\_CUSTOMER"

( "CUSTOMER\_ID" NUMBER(5,0),

"FIRST\_NAME" VARCHAR2(44 BYTE),

"SURNAME" VARCHAR2(44 BYTE),

"DOB" DATE,

"PHONE\_NUMB" VARCHAR2(24 BYTE),

"EMAIL" VARCHAR2(128 BYTE),

"CUSTOMER\_PASSWORD" VARCHAR2(108 BYTE),

"DELIVERY\_ADDRESS\_LINE\_ONE" VARCHAR2(44 BYTE),

"DELIVERY\_ADDRESS\_LINE\_TWO" VARCHAR2(44 BYTE),

"DELIVERY\_CITY" VARCHAR2(44 BYTE),

"DELIVERY\_COUNTY" VARCHAR2(44 BYTE),

"DELIVERY\_POST\_CODE" VARCHAR2(24 BYTE),

"SALT" VARCHAR2(64 BYTE),

"PRIVATE\_KEY" VARCHAR2(16 BYTE),

CONSTRAINT "PK\_CUSTOMERID" PRIMARY KEY ("CUSTOMER\_ID");

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCS251A"."CUSTOMER\_TRIGGER"

BEFORE INSERT ON PRCS251A\_CUSTOMER

FOR EACH ROW

BEGIN

<<COLUMN\_SEQUENCES>>

BEGIN

IF INSERTING THEN

SELECT CUSTOMER\_SEQUENCE.NEXTVAL INTO :NEW.CUSTOMER\_ID FROM SYS.DUAL;

END IF;

END COLUMN\_SEQUENCES;

END;

CREATE SEQUENCE "PRCS251A"."CUSTOMER\_SEQUENCE"

MINVALUE 1

MAXVALUE 99999

INCREMENT BY 1

START WITH 1;

### Customer Order

This is a table holding all orders placed on the website and all details required for a single order that has been placed. The order has a unique identifier, a timestamp to show when it was placed, an order cost rather than a calculated value to ensure that menu prices can change without affecting historical records, the staff ID of the delivery driver to apply accountability if the order goes missing, a requested timestamp to allow the customer to potentially specify when they want their order delivered, as well as a delivery timestamp to state when the order was actually delivered. The status of the order is also stored to easily display which orders need to be dealt with at any given time, along with a description of the payment method to allow delivery staff to see whether to expect a cash payment upon delivery or not.

Below the table create statement is the trigger create statement responsible for using the sequence that generates a unique Order ID for each entry. Below this is the sequence create statement itself.

CREATE TABLE "PRCS251A"."PRCS251A\_CUSTOMER\_ORDER"

( "ORDER\_ID" NUMBER(6,0),

"ORDER\_DATE\_TIME" TIMESTAMP (6),

"ORDER\_COST" NUMBER(6,2),

"STAFF\_ID" NUMBER(4,0),

"CUSTOMER\_ID" NUMBER(5,0),

"DELIVERY\_DATE\_TIME" TIMESTAMP (6),

"REQUESTED\_DATE\_TIME" TIMESTAMP (6),

"STATUS" VARCHAR2(30 BYTE),

"PAYMENT\_METHOD" VARCHAR2(30 BYTE),

CONSTRAINT "PK\_ORDERID" PRIMARY KEY ("ORDER\_ID")

CONSTRAINT "FK2\_CUSTOMERID" FOREIGN KEY ("CUSTOMER\_ID")

CONSTRAINT "FK1\_DELIVERYSTAFFID" FOREIGN KEY ("STAFF\_ID")

CONSTRAINT "FK3\_STATUS" FOREIGN KEY ("STATUS")

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCS251A"."ORDER\_TRIGGER"

BEFORE INSERT ON PRCS251A\_CUSTOMER\_ORDER

FOR EACH ROW

BEGIN

<<COLUMN\_SEQUENCES>>

BEGIN

IF INSERTING THEN

SELECT ORDER\_SEQUENCE .NEXTVAL INTO :NEW.ORDER\_ID FROM SYS.DUAL;

END IF;

END COLUMN\_SEQUENCES;

END;

CREATE SEQUENCE "PRCS251A"."ORDER\_SEQUENCE"

MINVALUE 1

MAXVALUE 999999

INCREMENT BY 1

START WITH 1;

### Drink

This is a table holding drinks to display on the menu. The name acts a primary key and was intended to be a foreign key as previously mentioned with custom pizza. The description is a generic entry about the product i.e. "Tropical flavour carbonated drink." With the image being a link to an image stored within the image table. Up to three prices can be stored for each drink, to be associated with three different sizes, but unlike pizza where the size will be controlled by the store, independent drinks companies can and do produce bespoke quantities a field to describe each size drink was included.

CREATE TABLE "PRCS251A"."PRCS251A\_DRINK"

( "DRINK\_NAME" VARCHAR2(20 BYTE),

"DRINK\_DESCRIPTION" VARCHAR2(250 BYTE),

"SMALL\_PRICE" VARCHAR2(6 BYTE),

"DRINK\_IMAGE" VARCHAR2(15 BYTE) NOT NULL ENABLE,

"SMALL\_SIZE" VARCHAR2(15 BYTE),

"MEDIUM\_PRICE" VARCHAR2(6 BYTE),

"MEDIUM\_SIZE" VARCHAR2(15 BYTE),

"LARGE\_SIZE" VARCHAR2(15 BYTE),

"LARGE\_PRICE" VARCHAR2(6 BYTE),

CONSTRAINT "PK\_DRINKNAME" PRIMARY KEY ("DRINK\_NAME")

CONSTRAINT "PRCS251A\_DRINK\_FK1" FOREIGN KEY ("DRINK\_IMAGE");

### Image

Table holding images for the menu items. The name of an image acts a primary key within the images table and a foreign key in various other tables. The CLOB related to an image name is a base64 representation of images stored in the database for use within applications. The image is converted back into an image by the applications.

CREATE TABLE "PRCS251A"."PRCS251A\_IMAGE"

( "IMAGE\_NAME" VARCHAR2(15 BYTE),

"IMAGE" CLOB,

CONSTRAINT "PK\_IMAGENAME" PRIMARY KEY ("IMAGE\_NAME");

### Ordered Item

The Ordered Item table holds items belonging to each customer order. Identified by a compound key of the order ID and the item name (which was originally designed as a foreign key but issues explained within the custom pizza description prevented this). The quantity ordered and price paid per item (to prevent updating prices from corrupting historical data) are stored against the order item entry.

CREATE TABLE "PRCS251A"."PRCS251A\_ORDERED\_ITEM"

( "ORDER\_ID" NUMBER(6,0) NOT NULL ENABLE,

"ITEM\_NAME" VARCHAR2(201 BYTE) NOT NULL ENABLE,

"QUANTITY" NUMBER(2,0),

"PRICE" NUMBER(32,2) NOT NULL ENABLE,

CONSTRAINT "PRCS251A\_ORDERED\_ITEM\_PK" PRIMARY KEY ("ORDER\_ID", "ITEM\_NAME")

CONSTRAINT "FK1\_ORDERID" FOREIGN KEY ("ORDER\_ID");

### Percentage Offer

This as a table holding percentage-based offers. Offer ID acts as primary key (although minimum value could equally take this role to reduce data redundancy). Minimum value serves as the point that the discount is taken off the order total before storing it into the order table.

Below the table create statement is the trigger create statement responsible for using the sequence that generates a unique Offer ID for each entry. Below this is the sequence create statement itself.

CREATE TABLE "PRCS251A"."PRCS251A\_PERCENTAGE\_OFFER"

( "OFFER\_ID" NUMBER(4,0) NOT NULL ENABLE,

"MINIMUM\_VALUE" VARCHAR2(5 BYTE),

"DISCOUNT" VARCHAR2(5 BYTE),

CONSTRAINT "PRCS251A\_PERCENTAGE\_OFFERS\_PK" PRIMARY KEY ("OFFER\_ID");

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCS251A"."OFFER\_TRIGGER"

BEFORE INSERT ON PRCS251A\_PERCENTAGE\_OFFER

FOR EACH ROW

BEGIN

<<COLUMN\_SEQUENCES>>

BEGIN

IF INSERTING THEN

SELECT OFFER\_SEQUENCE.NEXTVAL INTO :NEW.OFFER\_ID FROM SYS.DUAL;

END IF;

END COLUMN\_SEQUENCES;

END;

CREATE SEQUENCE "PRCS251A"."OFFER\_SEQUENCE"

MINVALUE 1

MAXVALUE 9999

INCREMENT BY 1

START WITH 1;

### Pizza

This is a table holding pizzas to display on the menu. The name acts a primary key and was intended to be a foreign key as previously mentioned. It holds a description of the pizza to be used on the website, small, medium and large prices as requested by the client, a heat rating which is a value from 0 – 3 describing how spicy the pizza is, and as with all tables, the image attribute links the entry to an entry in the images table.

CREATE TABLE "PRCS251A"."PRCS251A\_PIZZA"

( "PIZZA\_NAME" VARCHAR2(20 BYTE),

"PIZZA\_DESCRIPTION" VARCHAR2(250 BYTE),

"SMALL\_PIZZA\_PRICE" NUMBER(32,2),

"HEAT\_RATING" NUMBER(32,0),

"PIZZA\_IMAGE" VARCHAR2(15 BYTE),

"MEDIUM\_PIZZA\_PRICE" NUMBER(32,2),

"LARGE\_PIZZA\_PRICE" NUMBER(32,2),

CONSTRAINT "PK\_PIZZANAME" PRIMARY KEY ("PIZZA\_NAME")

CONSTRAINT "PRCS251A\_PIZZA\_FK1" FOREIGN KEY ("PIZZA\_IMAGE");

### Pizza Base

A table holding base options for custom pizzas. The name acts as a unique identifier, the description is used on the website as customer information and the price is used to calculate the total cost of a custom pizza when it is added to the order item table.

CREATE TABLE "PRCS251A"."PRCS251A\_PIZZA\_BASE"

( "BASE\_NAME" VARCHAR2(20 BYTE),

"BASE\_DESCRIPTION" VARCHAR2(80 BYTE),

"BASE\_PRICE" NUMBER(32,2),

CONSTRAINT "PK\_PIZZABASENAME" PRIMARY KEY ("BASE\_NAME");

### Pizza Cheese

This is a table holding cheese options for custom pizzas. The name acts as a unique identifier, the description is used on the website as customer information and the price is used to calculate the total cost of a custom pizza when it is added to the order item table.

CREATE TABLE "PRCS251A"."PRCS251A\_PIZZA\_CHEESE"

( "CHEESE\_NAME" VARCHAR2(20 BYTE),

"CHEESE\_DESCRIPTION" VARCHAR2(80 BYTE),

"CHEESE\_PRICE" NUMBER(32,2),

CONSTRAINT "PK\_PIZZACHEESENAME" PRIMARY KEY ("CHEESE\_NAME");

### Pizza Crust

A table holding crust options for custom pizzas. The name acts as a unique identifier, the description is used on the website as customer information and the price is used to calculate the total cost of a custom pizza when it is added to the order item table.

CREATE TABLE "PRCS251A"."PRCS251A\_PIZZA\_CRUST"

( "CRUST\_NAME" VARCHAR2(20 BYTE),

"CRUST\_DESCRIPTION" VARCHAR2(80 BYTE),

"CRUST\_PRICE" NUMBER(32,2),

CONSTRAINT "PK\_PIZZACRUSTNAME" PRIMARY KEY ("CRUST\_NAME");

### Pizza Sauce

This as a table holding sauce options for custom pizzas. The name acts as a unique identifier, the description is used on the website as customer information and the price is used to calculate the total cost of a custom pizza when it is added to the order item table and the heat rating is an value from 0 – 3 that describes how spicy the sauce is.

CREATE TABLE "PRCS251A"."PRCS251A\_PIZZA\_SAUCE"

( "SAUCE\_NAME" VARCHAR2(20 BYTE),

"SAUCE\_DESCRIPTION" VARCHAR2(80 BYTE),

"SAUCE\_PRICE" NUMBER(32,2),

"HEAT\_RATING" NUMBER(32,0),

CONSTRAINT "PK\_PIZZASAUCENAME" PRIMARY KEY ("SAUCE\_NAME");

### Pizza Size

This is a table holding size options for custom pizzas. The name acts as a unique identifier, the description is used on the website as customer information and the price is used to calculate the total cost of a custom pizza when it is added to the order item table.

CREATE TABLE "PRCS251A"."PRCS251A\_PIZZA\_SIZE"

( "SIZE\_NAME" VARCHAR2(12 BYTE),

"SIZE\_DESCRIPTION" VARCHAR2(60 BYTE),

"SIZE\_PRICE" NUMBER(6,2),

CONSTRAINT "PK\_PIZZASIZENAME" PRIMARY KEY ("SIZE\_NAME");

### Side

This table holds sides to display on the menu. The name acts a primary key and was intended to be a foreign key as previously mentioned with custom pizza. It holds a description of the side to be used on the website, a single price, a heat rating is an value from 0 – 3 that describes how hot the side is, and as with all tables, the image attribute links the entry to an entry in the images table.

CREATE TABLE "PRCS251A"."PRCS251A\_SIDE"

( "SIDE\_NAME" VARCHAR2(20 BYTE),

"SIDE\_DESCRIPTION" VARCHAR2(250 BYTE),

"SIDE\_PRICE" NUMBER(6,2),

"HEAT\_RATING" NUMBER(32,2),

"SIDE\_IMAGE" VARCHAR2(15 BYTE),

CONSTRAINT "PK\_SIDENAME" PRIMARY KEY ("SIDE\_NAME")

CONSTRAINT "PRCS251A\_SIDE\_FK1" FOREIGN KEY ("SIDE\_IMAGE");

### Staff

This tabled holds the account details of staff. For a staff member to log into either the desktop application or android application they need an entry in this table. This entry is posted by the API when the staff inputs a valid entry into the desktop application's staff tab (input validation exists on the page to ensure that expected values are entered). The size of fields are based on the size of strings after being encrypted by the security package.

Below the table create statement is the trigger create statement responsible for using the sequence that generates a unique Staff ID for each entry. Below this is the sequence create statement itself.

CREATE TABLE "PRCS251A"."PRCS251A\_STAFF"

( "STAFF\_ID" NUMBER(4,0),

"FIRST\_NAME" VARCHAR2(44 BYTE),

"SURNAME" VARCHAR2(44 BYTE),

"DOB" DATE,

"PHONE\_NUMB" VARCHAR2(108 BYTE),

"EMAIL" VARCHAR2(128 BYTE),

"COMPANY\_POSITION" VARCHAR2(40 BYTE),

"START\_DATE" DATE,

"ADDRESS\_LINE\_ONE" VARCHAR2(44 BYTE),

"ADDRESS\_LINE\_TWO" VARCHAR2(44 BYTE),

"CITY" VARCHAR2(44 BYTE),

"COUNTY" VARCHAR2(44 BYTE),

"POST\_CODE" VARCHAR2(24 BYTE),

"STAFF\_PASSWORD" VARCHAR2(108 BYTE),

"SALT" VARCHAR2(64 BYTE),

"PRIVATE\_KEY" VARCHAR2(16 BYTE),

"EMPLOYED" VARCHAR2(80 BYTE),

CONSTRAINT "PK\_STAFFID" PRIMARY KEY ("STAFF\_ID");

CREATE OR REPLACE EDITIONABLE TRIGGER "PRCS251A"."STAFF\_TRIGGER"

BEFORE INSERT ON PRCS251A\_STAFF

FOR EACH ROW

BEGIN

<<COLUMN\_SEQUENCES>>

BEGIN

IF INSERTING THEN

SELECT STAFF\_SEQUENCE.NEXTVAL INTO :NEW.STAFF\_ID FROM SYS.DUAL;

END IF;

END COLUMN\_SEQUENCES;

END;

CREATE SEQUENCE "PRCS251A"."CUSTOMER\_SEQUENCE"

MINVALUE 1

MAXVALUE 99999

INCREMENT BY 1

START WITH 1;

### Status

This is a lookup table for the current state of an order. This table is set up to enforce the order status and follow a convention so there is no ambiguity in our system.

CREATE TABLE "PRCS251A"."PRCS251A\_STATUS"

( "STATUS" VARCHAR2(30 BYTE) NOT NULL ENABLE,

CONSTRAINT "PRCS251A\_STATUS\_PK" PRIMARY KEY ("STATUS");

### Topping

This is a table holding topping options for custom pizzas. This table is set up to hold details of each topping offered on a custom pizza. The image is a name that relates to an image in the Image table. The heat rating is a value from 0 – 3 that describes how hot each topping is.

CREATE TABLE "PRCS251A"."PRCS251A\_TOPPING"

( "TOPPING\_NAME" VARCHAR2(20 BYTE) NOT NULL ENABLE,

"TOPPING\_DESCRIPTION" VARCHAR2(60 BYTE),

"TOPPING\_PRICE" FLOAT(126),

"HEAT\_RATING" NUMBER(\*,0),

"TOPPING\_IMAGE" VARCHAR2(15 BYTE),

CONSTRAINT "TOPPING\_PK" PRIMARY KEY ("TOPPING\_NAME")

CONSTRAINT "PRCS251A\_TOPPING\_FK1" FOREIGN KEY ("TOPPING\_IMAGE");

Please note that there had to be a change to the way drinks prices were stored, from Number to Varchar2 as there was an issue with rounding and checking prices when building the basket.

Also note that tables start with “PRCS251A” to make the names harder to guess in the event of someone trying to perform SQL injection on the system.

## Normalisation

Please find a supplementary Excel file within the submission that contains the normalisation[[3]](#footnote-3) for the project.

## Database Schema and Class Diagram

There are several differences between the class diagram and the database schema, caused by changes between the design and implementation of the project.

Firstly, the OrderItem class is clearly linked to the menu item classes within the class diagram, something that does not occur within the database schema because of an error thrown by Oracle if referential integrity is enforced.

Secondly, tables that were not originally included in the planned class diagram, such as Company Position and Percentage Discount, have been added to the database. The position serves as a lookup table for an attribute in the staff table. This lookup relationship had to be removed due the encryption performed on the staff object. The highest applicable discount is applied to the price stored in the Order table before it is posted. The discount isn't stored against anything other than the total cost of an order, which cannot remain a calculated price due to historic data potentially being ruined by future discount changes.

The final major difference is the changing of field types. Prices were planned to be stored as doubles, but this created rounding errors when comparing client-side prices with database prices to ensure that the customer hadn't tried to modify prices to reduce the price of their order. Where possible, name comparisons were used, but in the case of pizzas and drinks this was unable to occur due to there being multiple prices per product.

# Human-Computer Interaction

## Design Process

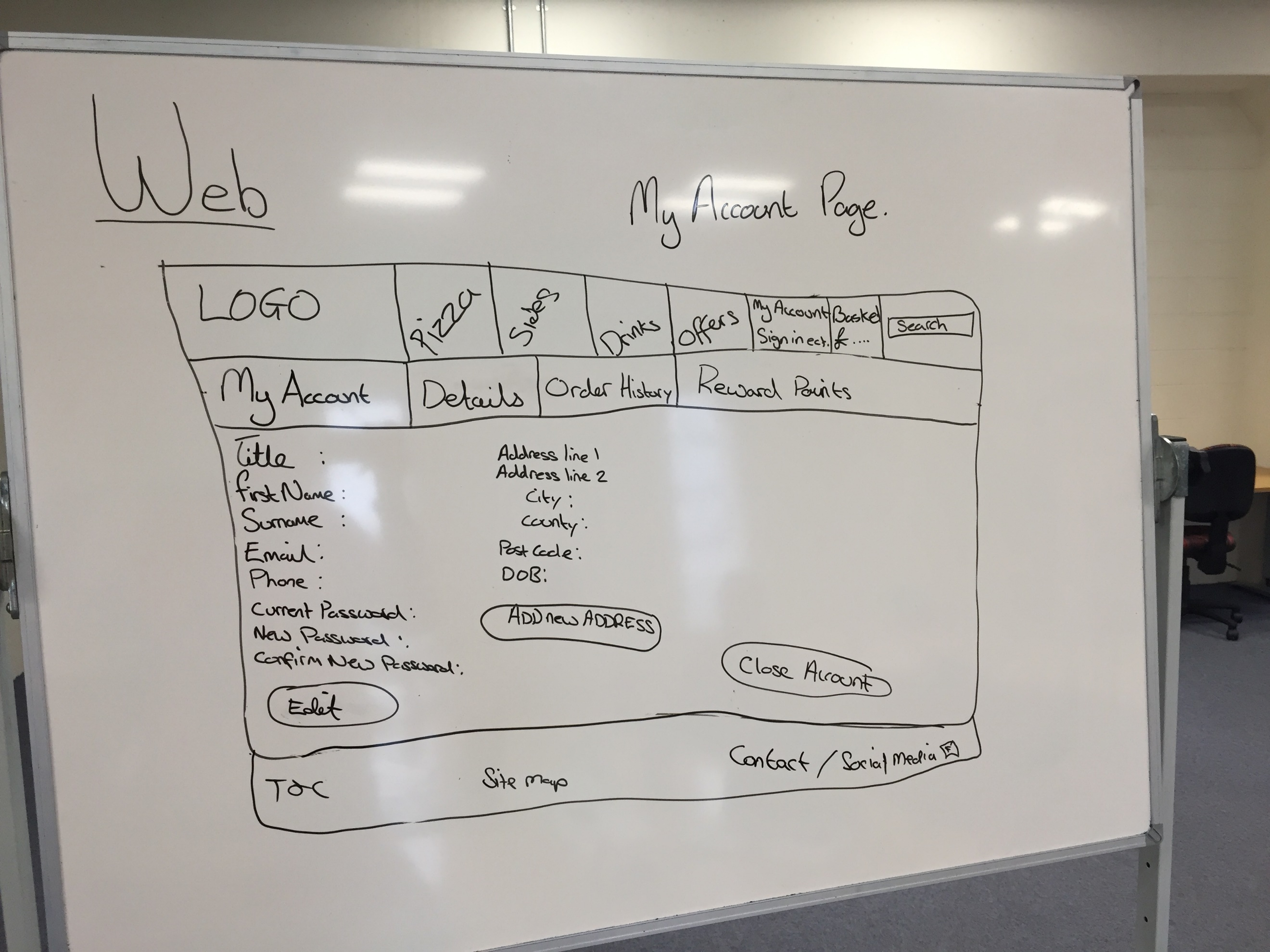
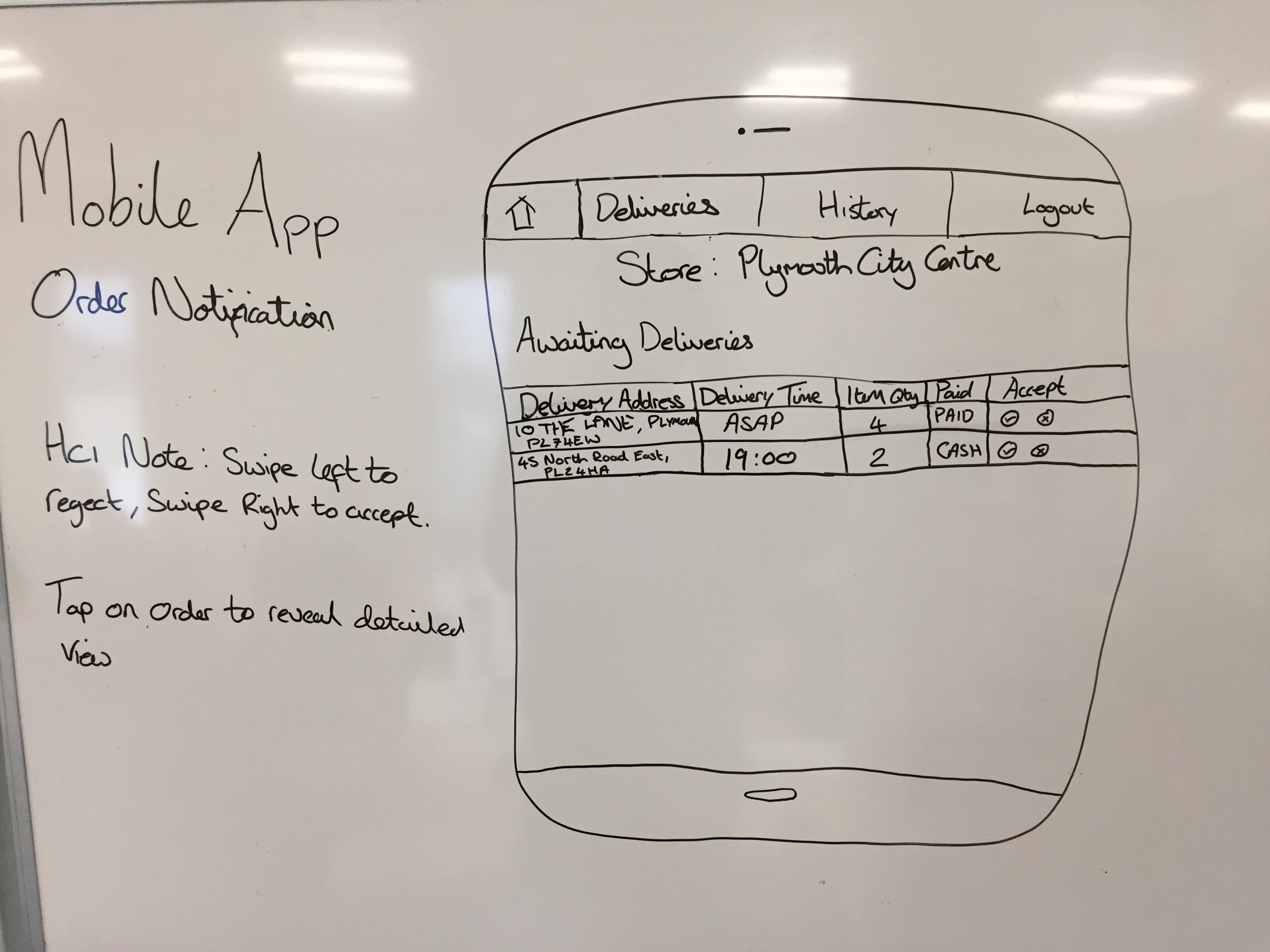
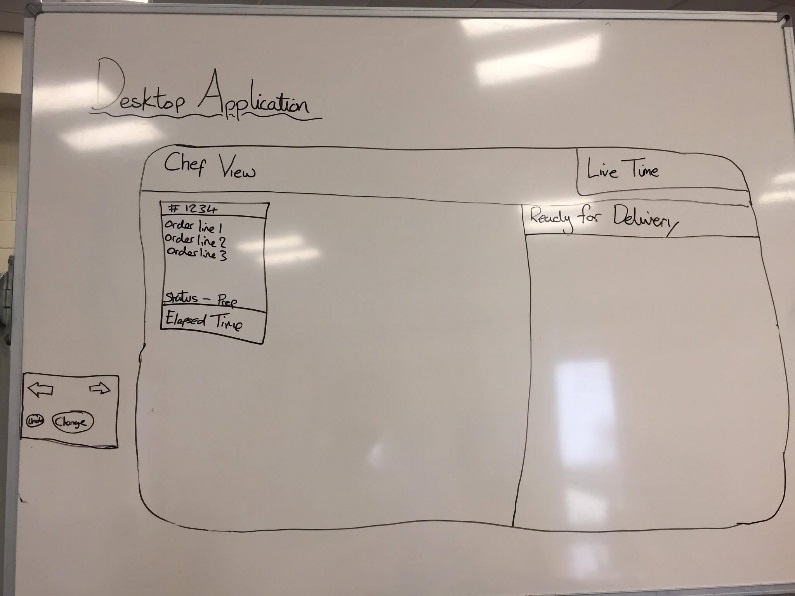
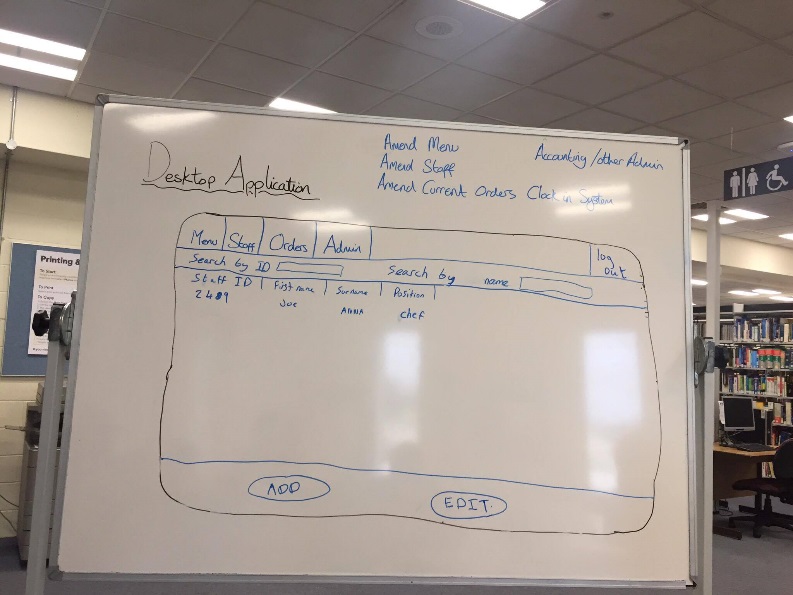


Fig 1. Initial sketches of the interfaces created during the design process.

## Critical Evaluation of HCI

The feedback received during the Mock and Live usability sessions allowed us to implement changes that have improved the usability of the products.

A personalised message was added to the banner when a customer has successfully logged in to the website, due to multiple testers requiring a visual confirmation that this had occurred. The feedback from these testing sessions also highlighted that users expected a toast or confirmation of items being added to the basket on the basket icon within the menu bar when they added items to their basket. This was implemented so that each time the user adds an item, the number on the basket increases and a message appears informing them that the item has been added. Another implemented feature that was suggested by a multitude of users during the usability sessions is a postcode address lookup tool, which has been implemented on the website to reduce the amount of input needed by the user to create an account, therefore enhancing the usability of the site.

Changes implemented to the Desktop that have arisen from these sessions include switching to a tab-based menu that allows the user to be confident of where they are within the application. They have also resulted in an error message occurring when an attempt is made to log in to the desktop application with invalid login details.

The main weakness identified in the web application is the lack of information regarding the user input when there are specific validation rules regarding what is input into certain fields.

Feedback is given retrospectively, but informative text would aid the user in successfully completing the form during the first attempt.

The main weakness of the mobile application is that orders claimed by the delivery driver are kept within the same view as the orders that are still available to be claimed by other drivers.

## Mock usability feedback

|  |  |  |
| --- | --- | --- |
| Minor | Essential | Desirable |
| Website: Tab ordering. | Rework registration page in general (downwards flow, not across then down). | Third banner along order process to show progress. |
| Website: Stop enter from submitting form on registration page. | # already in basket. |  |
| Website: Headers for data groups (personal, login, address) on registration page. | Rework basket logo to include quantity in basket. |  |
| Website: Right-align labels. | User tips for account information requirements. |  |
| Website: Separate basket into pizzas, sides, drinks. | Explicit “Thank you for ordering message” OrderID etc. |  |
| Website: “Back” button along order process. | Allow removal from basket. |  |
| Website: Label for quantity selector on sides & drinks. | Website: Welcome page on registering. |  |
| Website: Larger and bolder subtotal on basket page. | Desktop: Heat rating change from a slider to a drop down from a lookup table. |  |
| Website: Move order button to below subtotal value on basket page. |  |  |
| Website: Add asterisk to required fields. |  |  |
| Desktop: Price as 2 separate input fields for £ and p. |  |  |
| Desktop: Date formatted more meaningfully and to change the input to 3 different input fields for YYYY MM and DD |  |  |
| Desktop: Prompt before logging out. |  |  |

User feedback from Mock Usability sessions for the desktop suggested that there were not enough user tasks, and the lack of detail hindered the amount of feedback that could be collected at this stage. This was altered for the Live Usability sessions, allowing for a more informed and a greater quantity of feedback to be acquired.

## Live Usability Feedback

|  |  |  |
| --- | --- | --- |
| Minor | Essential | Desirable |
| Website: Stop register button from hiding. | Website: Input validation. | Website: API lag (Shirley?) |
| Website: Order Number & estimated delivery time on status page. Plus make status stand out more. | Website: Welcome page on registering. | Website: Replace placeholder banner with actual offers. |
| Website: Customised account label. | Website: Reduce menu banner size. | Website: Vegetarian flag on pizzas/toppings |
| Website: Cursor change on information button. | Website: Sort menu by A-Z (default), price, calories. | Website: Guest user. |
| Website: Longer toast when adding to basket. | Website: Larger buttons | Website: Customise pizza. (Explicitly told by Liz in our paper demo to not do customise) |
| Website: Padding to separate customer details. | Website: Volume on drinks needs to be clearer. – add price tiers to database | Website: “Continue shopping” button on basket page. |
| Website: Padding below banner. | Desktop: The scroll bars when used were slow, there also wasn’t much indication that the form did extend beyond the page. | Website: Quantity on pizza. |
| Website: Basket subtitles. | Desktop: top buttons were using rather than tabbed pane. Not clear and our forms lacking titles it was difficult to tell where in the program you were and what State the program was in. | Website: Offer position should be on left of menu rather than right. |
| Website: Larger quantity box. |  | Website: Separate out calorific information from description. |
| Website: Populate billing fields when use delivery address checked.[[4]](#footnote-4) |  | Website: Google Chrome’s autofill overlaps next field on registration. |
| Website: Input boxes turn red when selected. Indicates invalid input. |  | Desktop: Ordered item changes (voids and replaces) need more significant indicators, such as colour changes to the text and the new item appearing in the list. |
| Website: Stars not on all required fields. |  | Desktop: Once a change has been made to an ordered item a colour change or other indicator would be useful. |
| Website: Break up order status screen, add subtitles so it mimics basket. |  |  |
| Website: Feedback on unmatched/incorrect username/password |  |  |
| Desktop:Very few people used the sliders for prices, most just tabbed through the form and typed a number |  |  |
| Desktop: There were issues with the birthdate date picker if it was used, in that is was slow to navigate and started from the current date. |  |  |
| Desktop: Input validation needed when entering details to stop users entering details that can't be stored. |  |  |
| Desktop: When editing a menu item A currency indicator was missing (£ and P) |  |  |
| Desktop: Fields with max length need to be clear to the user when inputting. |  |  |
| Desktop: No error message on log in when the fields were empty |  |  |

## Expert user feedback

|  |  |  |
| --- | --- | --- |
| Minor | Essential | Desirable |
| Website: Register & Login buttons should be separated from the menu on the bar at the top. | Website: Add to basket RIGHT of customisation options. | Website: Customise” is a bad word (as is customising the menu item):   * Remove customise. * Add proper create/make/choose/select-your-own pizza instead. |
| Website: House name/number & street are better than address line1 address line2 on front end. | Website: Update # on basket on menu bar at top of page. | Delivery App: List of active orders for (expert) users who take multiple. |
| Website: Box sizes should be indicative of the amount of data to enter (registration, addresses). | Website: Feedback “toast” required when adding to basket. | Delivery App: Actual time for delivery not ASAP. |
| Website: Email should be on twice. | Website: Quantity for side dishes, drinks & deserts. Basket on right. | Delivery App: Reorder table based on time/distance/quantity etc. |
| Website: Info about password length & character requirements. | Website: Description overlay mustn’t cover pizza size / add to basket option. |  |
| Website: Feedback on acceptable password as they go. | Website: Move delete to be with quantity on basket page (“Is that delete?”) |  |
| Website: Finishing with “register” button is confusing/vague | Website: Step “tabs” (Order, delivery, payment etc.) |  |
|  | Website: “Proceed” button on pages is too vague “complete order” etc. would be much better. |  |
|  | Delivery app: when selecting items to deliver : Tried clicking all 3 ticks at once – caused confusion. |  |
|  | Delivery App: Group “my” orders together (colour system?) |  |
|  | Delivery App: Remove declined deliveries (and ones accepted by other drivers?) from list. |  |
|  | Delivery App: Distance/size important, not economic route (e.g. PL6 yes, PL9 no). |  |
|  | Delivery App: Unable to deliver option. |  |
|  | Delivery App: Actions done most need to be LARGER (registering a pizza has been delivered). |  |
|  | Delivery App: Delivery & order details wrong way around. |  |

# Security

Throughout the project many elements have been added to enhance the security. Customer passwords are required to have at least 8 characters, contain at least one upper and lower case character, a numeric character, and a special character. This password is then stretched using a salt that is unique to each user. The salt is generated using a SHA1PRNG instance of SecureRandom number. SHA1PRNG was chosen to generate the salt as this is not required to be unbreakable, since the combined original password and salt is then key stretched and hashed. To implement the key stretching SHA256 was used with 1000 iterations of the stretching algorithm. The key stretched password is then hashed using a site-wide salt of “PizzaHeaven” and SHA256. These processes have been implemented to prevent dictionary and brute force attacks on the system.

After the password has been strengthened in this way, the entirety of the customer data is encrypted in transit and rest. Encryption is used to ensure that if the database is compromised or the data is intercepted in transit, it is unable to be used. AES256 was chosen as the encryption algorithm to be implemented. Advanced Encryption Standard (AES) is widely considered to be a suitable level of encryption for storing sensitive data, which is reinforced by the fact that “in June 2003, the U.S. government announced that AES could be used to protect classified information” (Yi, Paulet and Bertino, 2014). A randomly generated 16 character private key is stored against each customer account and a 16 character public key of “PizzaHeavenCDJMZ” is also used.

The staff details are also dealt with in this way, with the only exception being that there are no set requirements to the length or characters used when creating a password.

When invalid login details are entered on the website, the message returned is “Incorrect login details”. This message is displayed regardless of which login detail(s) have been input incorrectly, preventing would-be attackers from confirming whether an email address is registered with the site.

As a user signs in, a session is created storing the customer’s details. This session is also used to pass order information back to the API via servlets. The session is invalidated when a user signs out, or if the session has been left idle for 30 minutes.

To minimise load times whilst a customer is adding items to their basket, sessionStorage has been used to store these items on the client’s machine. To prevent the tampering of order details such as menu prices, these sessionStorage variables are compared to the prices stored within the database. If discrepancies are found, then the item is currently flagged as being tampered with so that store staff are able to deal with the customer in a manner deemed appropriate by the manager.

To help prevent SQL Injection attacks, prefix’s have been added to the table names. This stops potential hackers from guessing the name of tables, for instance if the table was just named “Customer”. In addition to this JQuery has been implemented to deny the use of the characters that would be used to initiate SQL Injection (“ ‘ ;). Aware that attackers could copy and paste these characters into text fields there is also JQuery to unbind cut copy and paste.

During the initial planning stage discussion around how the system would handle different payment methods concluded that cash and card were the most important payment methods to implement. Implementing a cash on delivery payment method causes no security issues, especially when considering the end user’s stance of “shop staff are intelligent enough to handle anomalous purchases” (such as high cash values). When implementing a system to handle accepting credit card details, various checks are required to ensure that the details are both correct and secure from attackers. PayPal offers a platform that handles both of these issues so that the system doesn’t have to, and so the website has a redirect implemented on it which allows the customer to pay online securely through this platform, with minimal effort and maximum confidence in the system.

HTTPS was another security measure that was researched. Financial investment in the upgrade from HTTP to HTTPS is advised if the program was to go live, but it has not been implemented for the website due to the fact that the website will not be taking live orders from real customers in the near future. The web server used (Eeyore) is also behind the corporate firewall and so an alternative hosting platform would be required before HTTPS were to be implemented.

# Software Engineering

When implementing the applications, the target has been to adhere to common object-oriented guidelines, and where possible use established software patterns. The main focus of the project has been to ensure that every aspect of the code follows the Model-View-Controller design pattern.

One of the first implementations within the project was a data model containing classes to populate all of the tables within the database. In short, anything that has a table exists as a class within the data model. Secondly, any class that exists within the data model is required to be sent to the database for storage, so each class had a controller set up for it to handle this. Each controller largely works very similarly so an interface was set up to create a base set of methods required by each controller. Equally, requests to the API would all follow a very similar format, meaning that an API controller was set up to be extended by the class controllers to call a generic method, and feed in the minimal class-specific data that changed between two different classes. Finally, views have been populated with the GET requests made, and then the view only handles formatting the information and displaying imagery relevant to that situation. This implementation method maximises the ease of expansion of the software. For instance, initial plans included an order history page on the website, but other functions were deemed more important and it wasn’t possible to complete within the allowed time. With the MVC method it would be reasonably simple to expand on current functionality to provide this since there is already code in place to display orders that aren’t yet “Delivered” or “Void”, it would simply be a case of querying data with a different status to the page currently implemented.

In accordance with the GRASP principles, attempts have been made to implement commonly recognised design patterns where possible. Whilst the factory design pattern hasn’t been used, it is indirectly implemented in the sense that the sessionStorage “customer” variable (which contains a Customer object) must not be null for the OrderServlet to post a new Order to the database. The Order object can be created, but the post itself isn’t permitted, to ensure that Orders cannot be placed without a Customer. A particularly good example of high-cohesion within the project is the security package of the library. Each class has a very specific role, but it is important to call each and every part required, in order to ensure that data is stored in such a way that the decryption side of the package is able to decrypt it correctly so that it can be compared at a later date. The primary example of indirection within the project is in the data model. A class has been created for each table within the database, allowing each entry to be an instantiation of each class. The controllers library contains controller classes that act as information experts within the project. They do this by determining how data within the system is fetched and stored via the API. Throughout the assignment there has been an emphasis on minimising coupling by separating tasks out into classes, and dividing those classes up into packages that reflect their general purpose, grouping them with classes responsible for similar activities. This has meant that working on one application has a minimal impact on the other applications within the project, which was beneficial for the way in which the work was distributed. Polymorphism is a concept central to the project’s implementation. As mentioned within the earlier description about the MVC design pattern, all controllers are implementations of an interface and extensions of a primary API controller that handles the complex GET/POST/PUT/DELETE. The specific message of the API request is all that the majority of controllers have to handle, because they use generic methods held in the extended class. Inspired by server-side web development, a session is created within the manager application when the user logs in. This session is used to access class and API controllers when they are required. There is a generic GET method within the session class that is passed a String to identify which type of controller is required. This session class is an implementation of the singleton design pattern.

In addition to the above design patterns being implemented, the assignment contains a library which has been regularly updated with any functions that are required across multiple applications, or could potentially be required across multiple applications at a future date. This includes things previously mentioned such as the data model and controllers, but has also been expanded to include classes responsible for implementation of security measures, global variables such as API endpoint addresses and a date formatter and general helpers to ensure that currency values contain two digits for pence and a date formatter to convert an entered String into a Date. The library in general is aimed at maximising code-reuse, and minimising the time taken to write any functionality that has been implemented elsewhere within the project.

Several pieces of third party software have been used within the implementation of the project to introduce functionality that would not be possible to implement from scratch within the timescale of the assignment. In some places code has been modified to ensure that the library interacts with views implemented from scratch, but most of the time the import works as a “black box”, where an input can be fed in and the required output gets fed out.

The first library imported is an example of exactly that. The Jackson Project[[5]](#footnote-5) contains a JSON parser that allows the developer to apply tags within a model class that link the key descriptions in a String of JSON to variable declarations within the class and populate an array of instantiations of that class. This was initially used to allow implementation to begin across the project, but there wasn’t sufficient time to replace it and implement all of the other features that have been implemented, and so it is still in use.

There are two pieces of third party software imported specifically for the website for two very differing reasons. During the live usability demonstrations feedback was given that suggested typing out full addresses can be tedious when registering an account on a website. Codeberry[[6]](#footnote-6) supply a (limited daily use) free API endpoint for their postcode address lookup tool, and they even supply the code for users to implement it on their website. Having already implemented the address textboxes with various CSS and JavaScript/JQuery effects targeting specific and already implemented where each textbox would be stored in the database, it made more sense to take a copy of their code and modify it to fit the existing implementation (credited within the .js file).

PayPal was the second API imported specifically for the website. Storing a billing address and taking credit card details for store staff to bill the customer through a website creates various security concerns and extra work for the staff working at the business. By integrating PayPal into the website, it allows the application to reduce the amount of data stored (in the form of the billing address), and remove at least two security concerns; transferring and storing credit card information in a way that is secure in the event of a data leak, and ensuring that staff can be trusted with the information.

Google Maps is another API used within the implementation of the project. When the “Launch Map” button is pressed within the application it launches the Google Map application which then gives the user the fastest route between the mobile device’s current location and the customer’s address.

# Evaluation and Reflection

As a whole the project can be described as a success when factoring in time and labour constraints. The final product may not fulfil every criteria highlighted in the brief, but the brief fails to consider any issues arising. There are many aspects of the project that work effectively and as planned, such as the system being able to complete an order cycle; and any discovered bug that could be prevented in a reasonable amount of time has been covered.

The main issue left within the software development of the project is that an order posts in two separate sets of API connections. The first connection submits a new order in full, the second set of connections submit the order items individually. If a disconnection occurs before a post is complete, it will result in an incorrect order being posted. If the disconnection occurs between the order post and the first item, then an order will appear on the kitchen application without any items on it, which will then be obvious to the staff that they must contact the customer and query the order (the client has already said that staff are competent enough to deal with cash orders of any size). However, if the disconnection occurs after the first order item is posted then only the customer and manager will be able to see the issue. The customer will see items missing from the order status webpage and the manager will see too few order items for the price charged on the order itself.

Project management was an issue throughout the assignment. If the project were to be repeated, more focus would be placed on managing the group as an entirety, rather than relying on all members to be self-managing. Those in the group that were capable of managing themselves delivered implementation, and those that were reliant on guidance from a project manager did not.

Security that was implemented within the project was done so to the best of the team’s knowledge, based on the independent research and guidance that was given throughout. A major feature that should have been implemented but wasn’t due to time constraints was two-step verification. This would create a barrier for customers that want to spend money on the website, but it would offer them greater security if their password were to be compromised. It would be the decision of the client as to whether the positive outcomes would outweigh the negative. The main areas of the project that it would have most use in would be with the manager application; to prevent an authorised user from gaining access to the menus etc., and also to add a second layer of protection for the delivery application. Distributing a company phone with the delivery application and deploying a two-step verification application to the delivery driver's company phone would mean that if it were to be lost or stolen, members of the public would be denied access to the application and the customers details would remain protected.

HCI feedback for the application has been largely positive and this has been further improved through adopting several of the suggestions made by the HCI expert and users in the usability demonstrations who have used our product. It is therefore appropriate to conclude that the HCI of the product is at an adequate level.

The database has been modified as the implementation of the project has been expanded with additional functionality. In its current state, the database has issues, but data redundancy has been limited by using a minimal number of ID fields as primary keys by focusing on finding unique ways of identifying data without them. Several small improvements could be made to this, but it is in a reasonably efficient state.

The planning of the assignment took place in two stages, maximising efficient time use. An initial plan was drawn up based on the project brief before meeting with the client, and changes made to the plan were made based on conversations had with the client. Changes to the implementation were made as issues were encountered but the initial plans were mainly followed.

To conclude, despite several limitations during the process of the project, it was largely successful. A high-quality and functional portfolio of applications were created that fall in accordance with the brief guidelines, and any issues that did occur have been utilised to serve as experience for future projects.

1. <https://docs.google.com/spreadsheets/d/1-C2OT_bir7ZkaHblNqUjfOWIt6iJCKti9xsK_dPSckE/edit#gid=0> [↑](#footnote-ref-1)
2. <https://docs.google.com/spreadsheets/d/1-C2OT_bir7ZkaHblNqUjfOWIt6iJCKti9xsK_dPSckE/edit?usp=sharing> [↑](#footnote-ref-2)
3. <https://docs.google.com/spreadsheets/d/1sKriFVe4JmzAcTfIBb8DtDC6wBBtfDsC4c5s7gNNpzE/edit?usp=sharing> [↑](#footnote-ref-3)
4. This feedback was implemented but the system was later changed to no longer store billing addresses and it was therefore no longer required. [↑](#footnote-ref-4)
5. <https://github.com/FasterXML/jackson> [↑](#footnote-ref-5)
6. <https://getaddress.io/> [↑](#footnote-ref-6)