CO4015 MComp Computer Science Project

Final Report

An Android Application of a Canteen Ordering System

Joshua D. Brookes

School of Informatics, University of Leicester

Submitted: May 2021

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).
Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Joshua Brookes

Signed:

Date: 6th May 2021

Table of Contents

1 Abstract 3
    1.1 Glossary of Terms 3
2 Introduction 3
3 Survey of Information Sources 4
    3.1 Personal Data 4
    3.2 Hosting Costs 5
    3.3 Allergens 5
4 Requirements 6
    4.1 Administrator Application 6
        4.1.1 Optional Requirements 6
    4.2 Android Application 6
        4.2.1 Optional Requirements 7
5 Design and Implementation 7
    5.1 Design 7
        5.1.1 Database 7
        5.1.2 Admin Application 9
        5.1.3 Android Application 10
    5.2 Implementation 10
        5.2.1 Database 10
        5.2.2 Admin Application 11
        5.2.3 Android Application 12
6 Results and Discussion 13
7 Critical Appraisal 16
    7.1 Summary and Critical Analysis 16
    7.2 Sustainability, Commercial and Economic Discussion 17
    7.3 Personal Development 18
8 Conclusion 18
9 Bibliography and Citations 19
Appendix 1: ApiRetrieval Class Snippet 20

1 Abstract

Through the Coronavirus pandemic, small restaurant businesses as well as large have faced difficulties and closures due to lockdowns, and when they are able to open, they are constrained to using ordering services to be able to accept business, many of which charge large fees and commission in order to use their services. Educational institutions in many cases through the pandemic and its associated lockdowns have needed to house students on campus while not being able to accept them on campus for teaching. These institutions are unable to utilise the same services as normal restaurants due to their more private access. This dissertation attempts to complete a challenge to give these institutions a service to utilise that is as flexible and versatile as possible while remaining cost effective for the institution to use.

1.1 Glossary of Terms

Campus – An institution or business being hosted by the system.
Location – A distinct place managed by a campus from which users can place and collect orders.
Deal – A promotion similar in nature to a supermarket ‘Meal Deal’, in which a user can buy a specific set of items at a special price.
Offer – A promotion in which a single item is reduced in price from usual for a limited time.
Advert – A promotion in which some aspect of a campus is displayed more prominently.
Promotion – A collective term for a deal, offer or advert.

2 Introduction

This dissertation presents a prototype of a system for ordering food from educational institutions that can be used safely during a pandemic . This consists of an Android app that allows users to order products in advance, and allows them to be able to collect the order safely and on demand. This is paired with an administration side program that can recreate the brick-and-mortar location’s product selection easily on the system, manage orders and serve customers. The end product is targeted towards universities and other educational institutions, and is especially designed for campuses with multiple outlets for patrons to visit.

The proposed benefit of the app from the patron’s point of view is the ability to plan meals in advance, as the app will provide menus for every location up to the end of the next full week that users can order from. This will be enabled by having the administration side of the system set menus for each day at each location. To make this as user-friendly as possible, the administrator will be able to reuse previous menus from that location, and in the case of creating a new menu, be able to toggle the availability of products from the list that would then be hidden to the patron ordering from that menu. In addition, it will be possible for the user to filter out allergens they may have, and add this to their order to ensure their order is safely prepared; as such it will also be possible for the administrator to set the allergens and other dietary necessities for every item. As a result, it will be possible for patrons to filter items in a menu by dietary restrictions and cost, as well as simply searching by an item’s name.

The main objectives of this project are as follows:

- Develop a database that can store the information for locations, users and orders.
- Develop a desktop application that can manage the administration of the ordering system.
- Develop an Android app from which users can order items in advance for collection on demand.

3 Survey of Information Sources

In this section, I will investigate the validity of creating a password-free environment when personal data is not used. I will outline what laws are currently in place for providing nutritional and allergen information, and the steps required to collect it for display in the system. I will review current systems used for ordering food, namely Just Eat and Deliveroo, including running costs for a restaurant to use those services and what those services offer to the business and to the end user, and how allergens are declared by both parties. I will then discuss how feasible it will be to handle allergens, the steps that would need to be taken to do so, and what information will be included in the final system. It should be stated outright that the information that follows will apply specifically to the United Kingdom when relating to any regulations.

3.1 Personal Data

Firstly, when developing an application of this type that would have the potential to need personal data to function correctly. However, as investigated by Polykalas and Prezerakos, “17 per cent [of free apps] could be sharing such info with third parties without disclosing so.” [3] This paper hypothesises that there is a high probability that the business model behind some free apps is centred around selling users’ personal data to third parties. This could either be without it being disclosed, or it could be disclosed but hidden in the privacy policy of the app, or displayed via a pop-up to the user but ignored “almost instinctively to complete the installation.” [3] From this, it can be argued from an ethical perspective that the app produced here should aim to require as few permissions as possible, and collect minimal amounts of personal data. This should be done not only to reduce the legal liability of deployment in a public setting, but also to protect the privacy of the app’s prospective users.

In order to gain an understanding of what amount of personal data is required, as well as a potential business model to develop this application to suit, it follows that current systems should be reviewed. In this instance, I will observe Just Eat and Deliveroo. Just Eat’s privacy policy, in accordance with General Data Protection Regulations, lists clearly what data is stored and at what point. This collected data includes, but is not limited to: name, address, telephone number, email address, password. This data is collected voluntarily, but some of this is required to use the service. This data is relatively standard and necessary, however, the automatically collected information starts to match the hypothesis from Polykalas and Prezerakos. The most questionable collected data includes collecting “information about you from the content you create and the notifications or messages you post or send as well as what you search for, look at and engage with.” [5] The reasoning behind collecting this data is likely one or both of either suggesting more suited restaurants to the user or, as suggested by Polykalas and Prezerakos, sold to a third party. It should be noted that they do specify what they use some of the data for, but not all of it.

Deliveroo by comparison does collect the same voluntary information as Just Eat, such as name and address, but their privacy policy makes no mention of collecting information that is not directly relevant to the use of the service like Just Eat does, although it does mention gathering data from a social media account but only if it is directly linked to the user’s Deliveroo account. Such a clarification was not made in Just Eat’s privacy policy. This could in part be due to Deliveroo’s alternative business model that is closer to a ‘Freemium’ service, whereby the app and the service itself is free to use with the exception of the delivery fee and service charge paid to Deliveroo when placing an order. However the premium tier is a paid subscription to the service that waives the delivery charges for placing an order above a certain threshold, in addition to other benefits. As a consequence of this ‘Freemium’ structure, following the findings of Polykalas and Prezerakos, which states that “free mobile apps request access to personal data in a higher extent compared to the relevant requests by paid apps,” [3] Deliveroo’s business model and therefore app structure is not as likely to sell a user’s personal data to a third party.

3.2 Hosting Costs

From the restaurant's perspective, the financial gain for Just Eat is somewhat more direct. Their restaurant sign up webpage states that it costs £295 excluding VAT to join, plus a 14% commission excluding VAT on all orders placed, except for cash orders. [4] Therefore, the first month for a new business could cost them, based on £10,000 of orders in that timespan, £2,034, which equates to 20% of a restaurant's monthly takings. Considering the running costs of a restaurant, this reduction in revenue is unsustainable, particularly for a small or new business.

On the other hand, Deliveroo does not openly publish their commission rates, but one article does claim that delivery sites including Deliveroo "charge up to 30 per cent per transaction." [6][7] Again, such charges are prohibitively expensive, although Deliveroo does provide a lower commission fee of 5 percent for "orders where the restaurant provides its own delivery drivers." [7] This motive behind this measure is linked to the effect of hospitality caused by the Coronavirus pandemic. However, the saving provided by this reduction in commission may not adequately compensate for the labour costs of the delivery staff that earn this incentive, especially for a small business.

3.3 Allergens

Next, the UK government provides regulations on food packaging and what nutritional information must be displayed, and this is defined in the Technical Guidance on Nutrition Labelling [1]. In this guidance, it states that "back of pack" nutrition labelling must declare energy in kilojoules (kJ) and kilocalories (kcal), as well as the amount of "fat, saturates, carbohydrate, sugars, protein and salt" in the product.

However, this is not mandatory for "food offered for sale to the final consumer […] without pre-packaging" or "foods prepacked for direct sale" [1], but options are given that reduce the amount of information displayed; this can be as little as only the energy values, if one wishes to include any information in the first place. As such, for a university to provide any amount of information as described above, they would likely need to send samples of all of their products to a private company for nutritional testing. This added expense is naturally not one a university's catering department would wish to have if at all possible. As a result, it wouldn't make sense for this system to require caterers to provide nutritional information when they don't legally have to, and in many cases don't anyway. It should still be that the administrator is able to enter the energy value of the products, but this should not be required for any items.

On the other hand, allergens must be declared on all prepacked and non-prepacked food and drink, as explained by the Food Standards Agency [2]. These 14 allergens are: celery, cereals containing gluten, crustaceans, eggs, fish, lupin, milk, molluscs, mustard, nuts, peanuts, sesame seeds, soya and sulphur dioxide [2]. As this information is already gathered by university catering departments, this information should be included in the system. However, it could also be said that a patron's allergens should also be included when placing an order, to ensure that their order is prepared safely. Similarly, there are other dietary restrictions that are not included in the allergen list that could be described as self-imposed, such as vegetarianism and veganism, that should also be included in the system as an allergen. This is predominantly because other than the consequences of ingestion, these self-imposed allergens and the 14 medically imposed allergens are the same, especially from a food preparation perspective.

Finally I will compare and evaluate the current system for declaring allergens from both the user and restaurant perspective in both Just Eat and Deliveroo. Many restaurants create and possibly publish (depending on size and access to their own website) a directory of every item they sell and the ingredients and allergens in those items. Just Eat makes use of these where available, albeit indirectly by linking to the page where this directory is kept in the middle of a disclaimer that passes responsibility to the user rather than Just Eat or the restaurant themselves. However, this applies only to larger restaurants. Smaller businesses simply provide the number for the restaurant and ask that the user calls the restaurant before ordering (even though they could also use this

number to order and skip the middle man). It could be argued that this disclaimer and the method to disclose allergens is in such a way that it allows for Just Eat to fulfil only the bare minimum required to recuse them of any liability. Deliveroo's method is even less accessible, their method defaults to either calling the restaurant in all circumstances, but the same directory is still linked, but in a text box that can't be copied.

4 Requirements

4.1 Administrator Application

- Locations can be setup with a calendar view which can have each day selected.
- Each day selected opens a menu with further options.
- If a menu has not been set with a month until the date in question and the location is set to open, a small warning is displayed on that date on the calendar view.
- A day's menu contains options to create a menu.
- A location can be set to be open or closed on a selected day.
- A menu consists of categories of items, and a section for deals which is always at the top of the menu.
- Each item can be set on each day to be available or not.
- Unavailable items will be hidden to the customers ordering.
- Each item can have the following attributes set: name, price, allergens, calories.
- The allergens that can be selected are the 14 main allergens as defined by the Food Information Legislation.
- Upcoming remote orders can be summarised to allow for stock control.
- Upcoming remote orders can be viewed by day for the upcoming week or by the week itself.
- Each location's orders can be viewed to allow its compilation by staff.
- Available payment methods can be enabled or disabled.

4.1.1 Optional Requirements

- A day's menu contains an option to load a previously used menu.
- An item can have a photo set to demonstrate it.
- Each item can be selected from the prospective menu to be promoted or have a reduced price set (special offer).
- An advert can be set so that an item or deal can be highlighted to customers on their home page.
- An advert consists of a picture advert that is placed in the top banner of the customer's app.
- A special offer can be set that reduces an item's price between two set dates.
- A deal can be set by grouping together items of a similar type into multiple groups, connected by either an 'and' connection or an 'or' connection.
- A deal can have its price set with mark-ups for certain items in groups.
- A deal can be set to run between two dates or set indefinitely.
- Each location can reject orders as necessary, which alerts the customer.

4.2 Android Application

- New locations can be added by searching for the campus by name and selecting it from the list of results.
- Clicking a location starts an order for a selected day.
- Categories on a menu can be expanded to view that category's items.
- Orders can be started for the remainder of the current week, plus the next week.

- Items in a menu can be filtered by the 14 main allergens as defined by the Food Information Legislation, and their price.
- Items in a menu can be filtered by dietary restrictions.
- Orders can be paid for upon checking in at the location.
- Orders can be updated or deleted at any time by the customer until they check in at the location.
- An order should list any allergies a customer has.

4.2.1 Optional Requirements

- The top of the main screen should show a rolling view of adverts.
- Clicking an advert takes the user to the promoted item in the menu.
- Orders are collected by scanning a QR code upon entry to the pickup location.
- Meals can be recommended to the customer when they select the recommend button on the toolbar at the bottom of the screen.
- Recommendations will be based on similar items that the user has purchased recently.
- Items in a menu can be searched for by name or filtered by cost.

5 Design and Implementation

5.1 Design

5.1.1 Database

There will be three main aspects to this system: the administration program, the Android app, and the database that links the two together. This section will outline this database, its structure, how concurrency issues are to be dealt with, how information will flow between the database and the two subsystems. Finally, this section will show how deals will be stored in the database. The

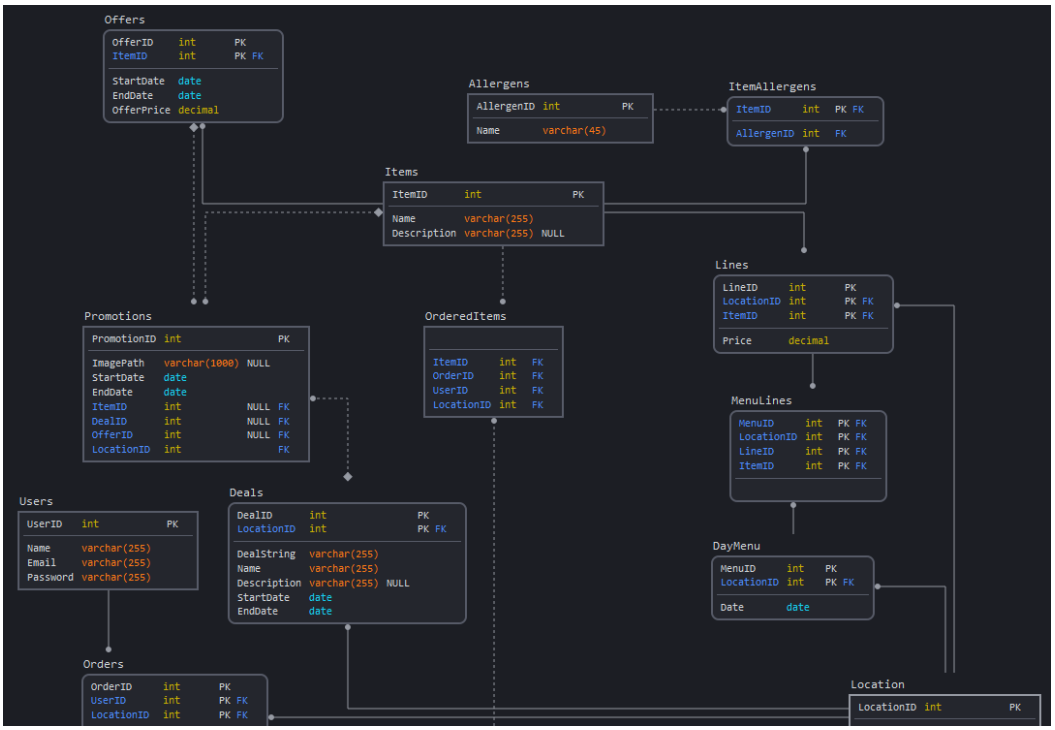

Figure 1: Planned structure of the database

diagram in Figure 1 shows the planned structure of the database system. Relationships between tables are identifying if the line between them is solid, and by contrast the relationship is non-identifying if the line is dashed. If the end of a line is circular, this represents ‘many’, while a diamond represents ‘zero or more’, and no shape on the end of a line means ‘one’. For example, the relationship between the tables ‘Offers’ and ‘Promotions’ is one offer to many adverts, but not every advert has to have a deal as an attribute; the ‘DealID’ attribute can remain null.

The system will allow for menus to vary between days and locations, using the ‘DayMenu’ table. This covers a typical occurrence in campus eating establishments that makes it slightly different to standard restaurants in that the menu often changes on a daily basis. To improve user friendliness, it will also be possible to search through past menus for reuse, saving the administrator of the system a significant amount of time by not having to manually recreate menus for every day. A similar case is present for items and lines. Items can be searched for across all locations for reuse, then used to create a line, which makes that item specific to a location with a certain price. An item can naturally be present in multiple lines as it may appear in multiple locations.

With a database on this size, it is important to consider concurrency issues. There are likely two common ways where concurrency could occur, one where an administrator and a patron make changes to the database simultaneously (in the patron’s case by placing an order in all likelihood), and one where two or more patrons place orders simultaneously. The former is likely to be less of an issue because of the design of the database, the only tables where the patron can make edits are the Orders, Users and OrderedItems tables, all of whom being tables the administrator would never edit (although the Orders table could theoretically be edited by the administrator in an emergency). However, the latter offers more of a problem. Multiple users making an order could cause a clash in the Orders table if not dealt with appropriately. However, modern database management systems are able to deal with concurrent insert queries, for example MySQL will perform the concurrent queries in sequence without data clashes provided the “concurrent_insert” setting is set to 1 (Auto) or 2 (Always) [9].

Finally, there are three extra tables included in the database system that represent temporary changes to a location’s pricing structure or visibility. The first of these tables is the Promotions table; this gives the administrator the ability to promote with an advert an item, a deal or a special offer for a set time period by displaying an image as a banner at the top of a patron’s app’s home page. Pressing on this image would then redirect the patron to the promoted item or deal. Special offers are stored in the Offers table; special offers allow the administrator to reduce the price of an item for a set period of time. On the patron’s app, both the usual price and the reduced price would be displayed, as well as the end date of the special offer.

The final table stores deals in its namesake table in the database. Deals give the administrator the ability to group items together into groups, and link that group with others using combination commands (“and” and “or”), and set that combination to an overall price. This structure is most commonly seen in “meal deals” wherein you would typically be able to buy a sandwich, a packet of crisps and a drink for example for a certain price. The structure that such a deal could be defined is stored in the “DealString” attribute, and is outlined below:

Syntax:

- Use ItemID rather than the Item’s name (to ensure the deal string is parsable)
- ( ) – Statement Group
- { } – Item Group (commas used to separate items in the item group)
- [ ] – Item Surcharge
- * – And
- / – Or
- = End of statement (this is followed by the standard deal price before any item surcharges)
- Whitespace is ignored

Example:

({3, 6, 17 [0.3], 27} * {14, 2, 28}) / ({71, 66, 112 [0.5]} * {229, 236}) = 4.5
In the example, firstly selecting items with the ID 17 or 112 incur a 30p and 50p surcharge respectively. The patron can then select an item from either the first and second groups, or from the third and fourth groups. Finally, the final price of the deal is £4.50, excluding any surcharge from premium items (items 17 or 112 in this example).

5.1.2 Admin Application

This section will cover the initial design of the admin application. The main purpose of the admin application is to give the managers of the location full control over their menu and the running of their location, rather than lengthening the process by sending a menu to the hosts of the ordering service. In order to achieve this level of control while remaining user-friendly and easy to use, several features need to be considered.

Firstly, the admin will be able to create categories for a ‘master menu’, which in turn will contain every item that the location could serve at some point. The idea of this is to allow the admin to be able to easily change items on a menu, similar in nature to a specials board for example, without having to delete the item and recreate it later. Each item will be able to have with it a description of it, a standard price set, and its allergens declared easily. This is a departure from the current systems as this method allows the allergens of each item to be clearly displayed next to each item. Categories and items will be movable, so the order they are shown can be adjusted to suit specific needs. However, these controls are only for creating the ‘master menu’ for a location. In order to have the user see the menu in the Android app, they will need to set the specific items for the menu in each day. This will consist of selecting each item from each category that will be on sale on that day.

Next, in order to handle allergens, the 14 main allergens plus the dietary requirements of vegetarianism and veganism as discussed in the Survey of Information Sources will be entered directly in to the database to be read as necessary from the database as a reference. For the admin application however, as previously mentioned, allergens will be easily declarable for every item individually, so they are easily accessible immediately to the user. Its secondary purpose will then be to allow the user ordering from a location to filter to remove items that have a certain allergen.

Then, there needs to be the functionality of handling special promotions that a location might offer from time to time. These promotions can be characterised into three types: adverts, offers and deals. An advert will contain a link to a specific aspect of a campus, be that a location, an item or another promotion. Naturally with any advert, the most prominent way to show this link is through an image. Both the image and the link are then stored in the database. An offer will reduce the price of a selected item, options will be provided to reduce by a percentage or to a specific price for a time period set by the admin. Deals will be constructed to fit the format outlined in the previous section, with their syntax broken down into their component parts that can be selected individually and the relevant components’ properties adjusted to fit the bespoke requirements of the location’s deal. Deals will be able to handle selecting one item from a group, selecting an item from multiple groups or from certain groups only, as well as surcharges for certain items. Once the deal is constructed and a save of the deal is attempted, the system will validate the deal to ensure the syntax is valid, and will provide error messages to inform the admin of any necessary corrections. The deal can, like offers, have an identifying name set, a price for the promotion, and a time span.

Finally, in order to handle orders and provide an outlook that can be used for stock control and preparation, the admin application will contain a screen that displays upcoming orders (as well as past orders for the purposes of record keeping). Each order for a day can be selected individually and its details will be displayed, including the items in the order, any allergens, the order cost and its current status (whether it was collected or if the user has checked in ready to collect the order). This screen will also provide functionality to mark the order as collected, and as

such this screen would be the main screen open during service to keep track of orders during the course of the day.

5.1.3 Android Application

This section will cover the initial design of the android application. Firstly, in a departure from the method of current systems, the android app will allow users to declare their allergens ahead of time, and link it to their device's account. The user will be shown a list of the 14 main allergens, as well as vegetarianism and veganism, and their selections will be saved automatically to the database, with a link to their device's account. The purpose of this is that this list of allergens will be sent with an order to the location, who can then adjust their cooking environment to suit those allergens and protect the user. In addition, this allergen declaration will also be used to filter the items on the menu, removing those items which the user would not be able to consume because they contain one or more of the declared allergens.

Then, in order to view deals that admins have created for a location, a distinct format will need to be adopted that is separate to the one designed to select individual items. The deals will be kept in a separate category at the top of the categories list for a location, in order to make them as prominent as possible. The deals inside will be displayed as an item; beyond this is where the divergence will be. When a deal is selected, a list of the groups in the deal will show, in the order they're listed in the deal string created by the admin. When the user selects a group, they will be shown a list of the items in that group, of which the user shall select one. When an item has been selected from all necessary groups, the user will press the submit button and the selections are validated, and any surcharges added to the usual deal price. The items will then be added to the basket and the basket total is increased.

Finally, to place the order with the location, there needs to be a basket screen that displays the state of the pending order, as well as allowing the user to place the order. The basket screen will specifically consist of a list of the items currently selected, an option to set the name the order will be collected under, and the total cost of the order.

5.2 Implementation

This section will cover the implementation of the overall system, including the decisions taken during development.

5.2.1 Database

The database was constructed and hosted in Microsoft Azure. Azure was chosen because of its easy access for all formats as it is a cloud platform. It also allowed for the creation of web apps, as this was required for secure and stable connection to the database from Android apps. The structure of the database remained largely unchanged from the initial design. However, some aspects were altered to improve its capabilities. Firstly, a table for campuses was added, as the use of the database changed from being for a single campus, with its structure theoretically copied to different servers for each new campus that would use the service, to a single centralised database that would contain each campus' data. This would then allow for the centralisation of user data as and when a user switched between campuses to place orders.

Also, the structure of recording the opening times of locations was adjusted to be more flexible. Previously, it was designed to have a record for every individual day's opening hours. This has been changed to have a separate table for usual opening hours for each day of the week, and any specific, individual changes to those usual hours is added to a day's menu record in the DayMenu table. To ensure generalisation and no confusion over the time when stored with a date in the database, the times are saved as an integer from 0 to 48, each number representing half an hour, which can then be decoded in the admin or Android app easily.

Furthermore, the storage of allergens has been simplified to make it more efficient, encoding it into an integer in a similar way to encoding a character into ASCII or Unicode. The 16 allergens are converted into a 16-bit binary number, whose integer value is then stored in a single field in the database, rather than recording each allergen for each item individually in a separate table. The encoding and decoding is then performed on the client side.

Finally, users are stored in the database with only an ID string generated with UUID. This is stored in conjunction with the allergen number in the same way as previously stated. The purpose of this is so the user no longer needs to enter any passwords or any personal data whatsoever, with the exception of a name for collecting an individual order, which does not need to be the user's actual name in the first place. The user's ID is stored in the user's device's files, and is never seen by the user. As such, any order placed by a user can only ever be collected by the user, as they can only check in from their device. In the event of data loss on the part of the user, and they can't use their device to check in, they can simply re-order from another device, as payment is only taken at the point of collection by the staff at the location; the system defaults to pay on collection, in order to accommodate any payment system currently in place by an institution as seamlessly as possible.

The main addition to the database system is the logic apps that were created to form a bridge between the database and the Android app. These consist of a URL generated by Azure to use to call the logic app, with request data within a JSON file in the request body. The data in this JSON is then used in an SQL query to retrieve the information. The results of the query are then output in the response body. By doing this, the database is more protected from SQL injections, as every request is called only from the app and the user's selections, rather than any data entry, with the exception of the collection name.

5.2.2 Admin Application

The admin application was developed using Java Swing with the Netbeans IDE. Netbeans was used primarily because of the visual IDE that allows developers to drag and drop components easier and presents a code-free environment for the majority of front end development. Only minor changes were made to the initial design of the features that were implemented. Firstly, the location selection screen reads a list of the locations managed by a campus from the database. The campus the database reads from is hardcoded into the admin application, and at deployment this would vary to match the name of the client, so the client would be guaranteed to only receive data relating to their own locations. On the side of the screen are a list of options, including to add and delete a location, view its orders and deals, and adjust its calendar.

The calendar is where the location's menu can be created and customised, as well as opening times for the location for each day of the week and for specific days. The location's check in number is also shown here. This is used in place of a QR code as a 6-digit number is arguably more accessible for users than a QR code. Opening times can be set using two sliders for opening and closing time, plus a button to toggle the location being open or closed in the first place. The only differences between setting opening times for days of the week and a specific day are a drop-down box to select a day of the week being absent naturally for the specific day settings, and a specific day's opening times can only be set when that date is selected in the calendar.

To create a menu, the admin does not need to select a specific day at this point; when they click on the 'Create Menu' button, they are shown a screen that handles the organisation of categories in a menu. Every category can have its name edited and its position in the list changed so an admin can order the list of categories as they see fit. They can then edit a category which takes them to the item screen, where they can create each item in the selected category. To begin with this has the same options as with the category screen, it can be created and deleted, have its name changed and its order changed. From then is where the divergence lies; each item can have its properties edited, including a description, price and allergens. Once all the categories and items within them are created, the admin can then customise a specific day's menu to only include the items they wish to sell on that day. This is done by selecting a date in the calendar, then selecting the 'Set Menu' option, which is a single screen in which the admin can select a category and view

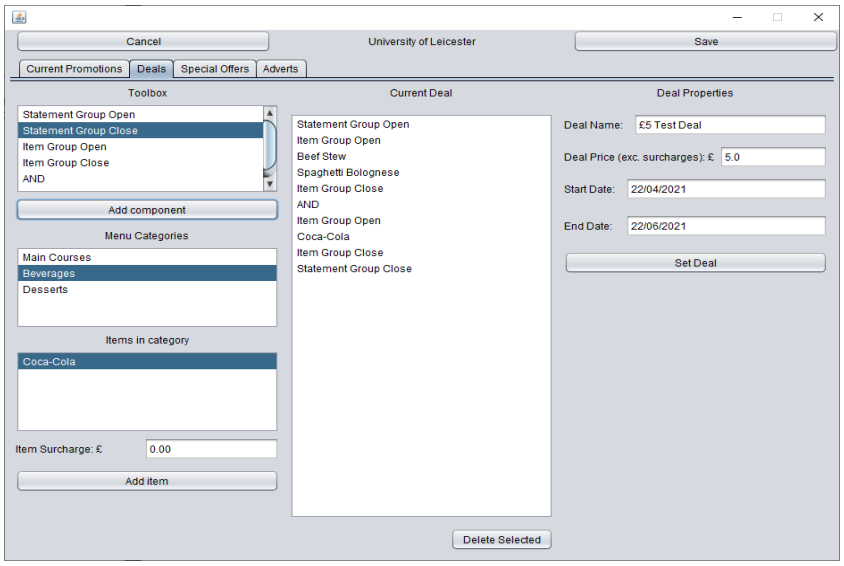

Figure 1: Deal Creator Screen

the items in that category. Then they can select each item to move it back and forth from the menu. There is also an add all button that adds every item in the selected category to the menu. After the menus are created, an admin can then create and handle promotions starting from the location selection menu. The promotion screen initially contains a list of the promotions currently created, as well as options to create new deals and special offers. Both of these can be edited or deleted later on. A special offer can be created in the same way as it was designed; the item to be reduced is selected, then its reduced price is set either manually or by setting a percentage reduction. Then the time period of the offer is set, and it is saved to the database. A deal is created in much the same way as an offer, the only difference is in what the deal contains in terms of the items. Figure 1 shows the layout of the deal creator. The toolbox contains the different syntactical components. Selecting one of those will place it at the end of the list in the current deal. Any component can be removed by pressing the delete selected button at the bottom of the screen. An item can be selected by selecting a category first then selecting the item from the list of items in the selected category. An item surcharge can also be applied at this stage.

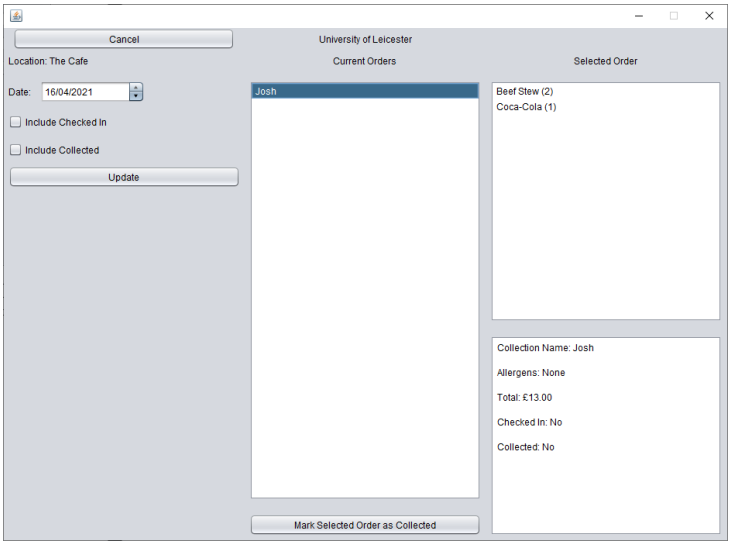

Figure 2: Order Viewer Screen

Finally, Figure 2 shows the screen that displays all of the orders for a location. The default for that screen shows the orders placed but not collected or checked in, so admins can see in advance what orders are still to come that day. The date shown can be changed, including to dates in the past so admins can keep a record, as well as toggling viewing checked in and collected orders. A selected order will show the items and quantity of them in the order, the collection name, allergens and price. Admins can also use this screen to set orders as having been collected.

5.2.3 Android Application

The Android application was developed using Java with the Android Studio IDE. Firstly, the home screen for the Android app shows a list of all of the campuses hosted on the system, from which the user can select and change at will rather than being forced into a single campus as may be the case with some other systems targeted for specific universities. This screen also contains options to access the user's placed orders and their allergen settings. The allergen settings screen contains a list of switches, one for each of the 14 main allergens plus vegan and vegetarian. The

settings confirmed in this screen are saved to a file on the user's phone, and sent to the database as an update to the user's settings the next time they place an order.

Once the user places an order, they can check in to the location of the order to collect it in the 'Your Orders' screen, accessible from the campus selection screen. Orders will only be shown if they have scheduled an order for the current day. When a user selects an order to check in for, a pop up menu will appear that will prompt the user to enter a 6 digit code which will be visible at the location somewhere clearly visible. If this number is correct, the database will be updated to say that the order has been checked in for collection. The staff at the location would then use this prompt to finish preparing the order, request payment from the user in person, then hand over the order.

However, in order to create an order in the first place, the user must first select a campus and then a location to order from. Once this is done, the user is then shown a list of the categories in that location's menu, with an extra category containing deals shown at the top followed by the admin created categories in the order the admin set. At the top of this screen, the user can change the date being shown to the date they wish to order for, from the current date to the end of the next week. If the date is changed while the basket has items in, the basket will be reset. When a category is selected, the items available on the selected day within the selected category are listed. Each item lists its name, description, price (or offer price is one exists on that date) and its allergens abbreviated (the key to those abbreviations is shown on the allergen settings screen). Pressing on an item will add one of that item to the basket. The view basket button at the bottom of the screen will then increment to show the new value of the items in the basket and the quantity of items in it.

Deals are selected in a similar screen to items. However when the deal is selected, the user is directed to a separate screen consisting of two lists: the list at the top displays the groups in the deal, and the list at the bottom displays the items in a selected group, just as in a normal item selection screen, only without the price of the item. Once an item is selected from each necessary group, the user can press the submit button and the options are checked to make sure they are valid for the deal. If they are valid, the items are added to the basket and the user returns to the deal selection screen to continue their order. If the options are not valid, the user is notified of the issue with a relevant message.

Once the user is ready to place the order, they can press the view basket button, where they are taken to a screen to finalise their order. A list of the items in the basket is shown; the user can press on a item to delete it, but warned by a pop up message first to make sure the user doesn't delete the item erroneously. Below that is a box where the user can enter the name under which they wish to collect the order. The details of the location, date of collection, payment method (always pay on collection) and order price are listed here also. Provided there are items in the basket and a name has been input, upon pressing the place order button the order is sent to the database to be recorded, and the user is returned to the home screen.

6 Results and Discussion

In this section, I will discuss the results of testing of key sections of the final system and analyse the source code, specifically for calls that the Android app makes to the database via Azure logic apps, as well as the functions used in both the admin and Android apps to convert to and from the deal string language. Azure logic apps allow for functions to be created on the server side and allow more secure access to a database in this instance. For this system, 18 such apps were created to allow data to

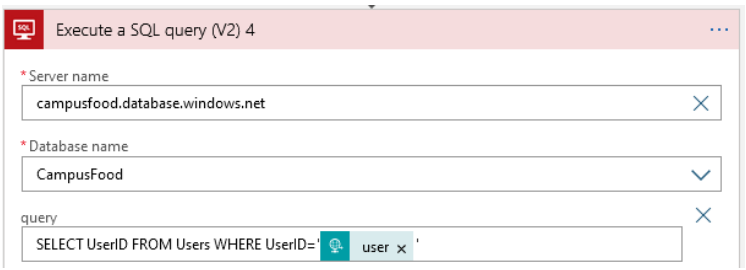

Figure 3: First SQL query searching for an existing user

be displayed and manipulated on the Android app. In this section I will cover two of them, one for placing an order, and another for checking in for collecting an order. The SetOrder function created in Azure is designed to place an order with a location, and is called from the Android app. The

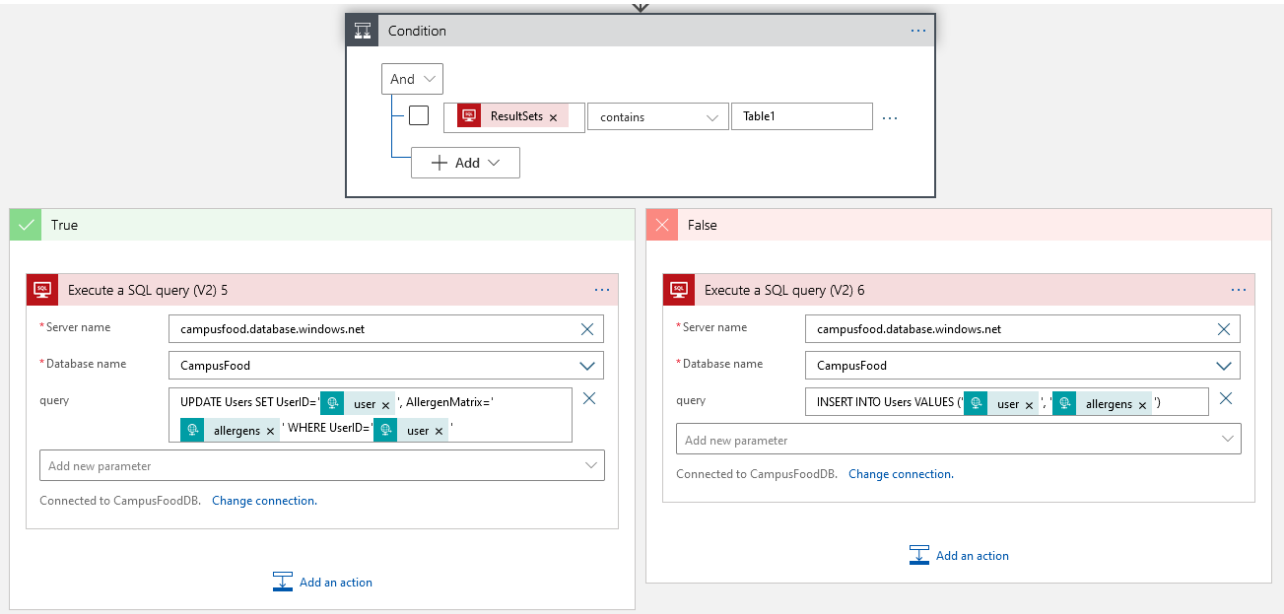

Figure 4: Condition block checking for existing user

request data is first collected from the request body, which uses a JSON format, and contains the properties of an order to be placed. Next, a check is performed with the database as shown in Figure 3 to see if the user who placed the order has been recorded on the database. The output from that SQL query is checked in a condition block as shown in Figure 4. If the user exists in the database, the data for that user would be returned in a table called Table1 by default. If no user exists in the table, the response would be only a pair of braces. Therefore it follows that a user would have been found if the output of the query is anything other than a pair of curly braces. So if no user was found, it would then be placed into the database with an INSERT query. Otherwise an UPDATE query is used to make sure the allergen settings for that user are completely up to date.

At this point the order can be added to the database. Figure 5 shows a query to collect the order's location's ID from the database for the porpose of simplifying the queries later on. The output of the query then needs to be parsed based on the JSON schema partially shown in Figure 5 to retrieve the correct data to be used in further SQL queries. The SQL query is then made to insert the order to the database. This is surrounded by a for each loop; this is required by Azure as it needs to loop through each result found by the location SQL query, despite the chance of more than one result being the same as the chance of having two identical UUID strings in a given dataset, which is around 1 in 2^{122}. As such, this for each loop can remain with the confidence that it will never loop more than once.

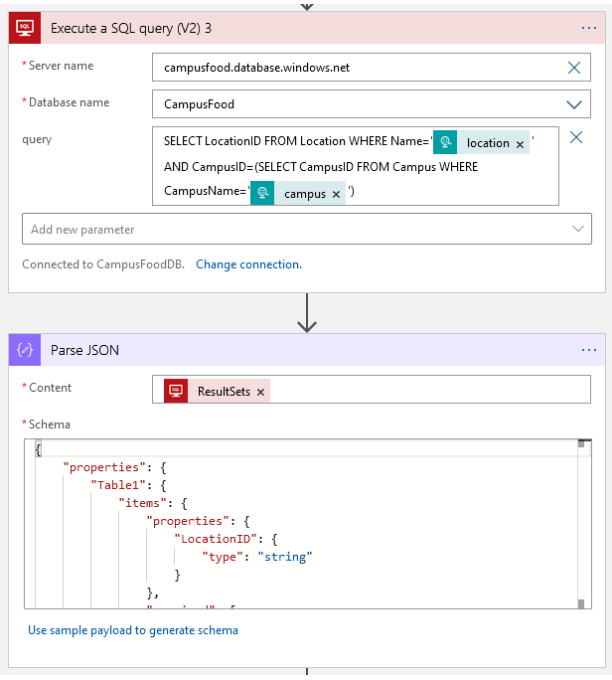

Figure 5: Second SQL query to collect location ID

Following that, another query is run to save each of the items in the order to the OrderItems table, which in this case requires a nested for each loop, the outer loop for the list of items, and the inner loop for the location ID search as before with the order insertion query. After all of these queries are performed, the logic app will send back a response code of 200 with the message “Order placed successfully.” In terms of performance, the logic app will typically return responses in less than 2 seconds, with a fastest response time of 1.27 seconds.

CheckIn function created in Azure performs checks on the verification number and marks the order as checked in. This can only be called from the Android app. The request body for this function contains the order ID, location name and campus name, and user ID, as well as the verification number that the user entered. The first SQL query, as shown in Figure 6, collects the verification number from the database, to be checked against the number input by the user in the Android app. The campus name and location name are both needed to ensure the correct location is used to cover off the distinct possibility of indentically named locations on different campuses.

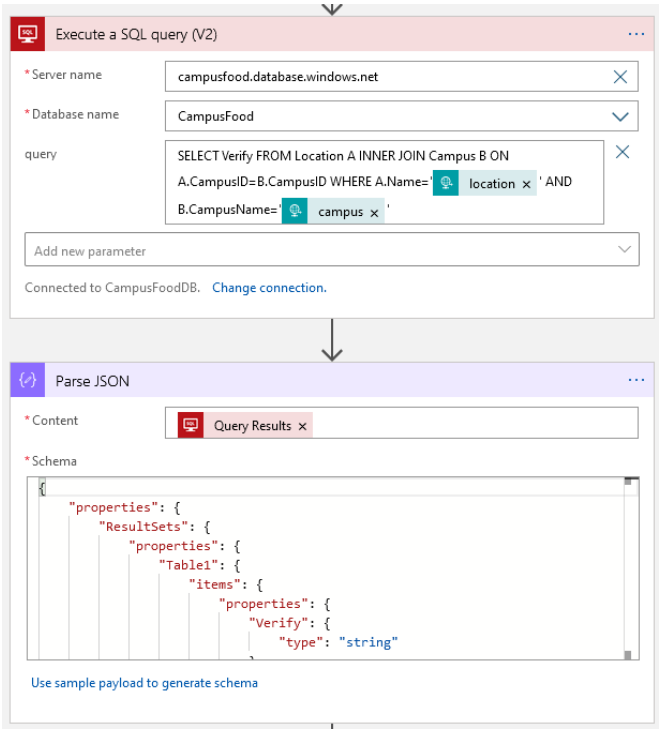

Figure 6: SQL query to collect check in number for location

As with the location ID collection for the order placing function, a JSON parser is used here as well to extract the information from the output of the SQL query, which is in a JSON format. As such, the extracted value can then be compared against the value that the user entered. This is shown through the condition block in Figure 7. If the numbers match, the order is updated in the database to state that the user has checked in. Then, irrelevant of whether the verification was correct or not, another SQL SELECT query is performed to get the check in status of the same order. The purpose of

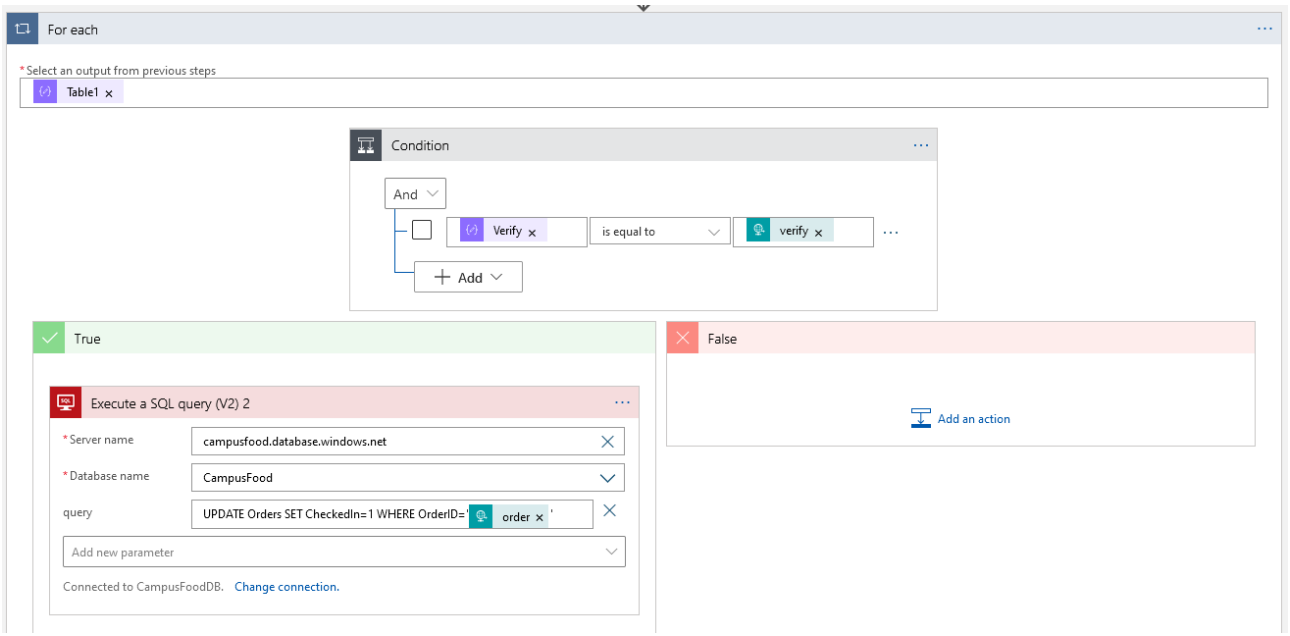

Figure 7: Condition block to check verification number

this is to send the result of that in the response body, and the Android app will then use that result of

either true or false to display the correct message: either a message to say the check in was successful, or a message to say the verification number was wrong.

In order to connect to either of these Azure logic apps, as well as any of the other 16, the Android app must first prepare the request body. This is done in a single class, called ApiRetrieval; a portion of this class is shown in Appendix 1. An asynchronous task is used because by default, any attempt at an internet connection on the main thread throws a NetworkOnMainThreadException, so an asynchronous task is the only solution to this. The overridable method doInBackground contains the parameter urls which can be of a variable length, as such it is possible to use the function for any call made to the API in Azure.

Appendix 1 shows the call to place an order on the database. The arguments in this case would be, in order, the name of the call to be made ("setorder") and the order contents, which were pre-generated in the class the method is being called from. This was done in order to keep the ApiRetrieval class as generalised and therefore neat as possible. Due to the pre-processing of sorts, the connection setup becomes relatively straightforward. The url is set to the constant that holds the SetOrder logic app call URL, and the request body is set to the pre-processed JSON text. The connection is then opened and the request method is set to POST and the request properties set to make sure the request is acknowledged to be in English, using UTF-8 and that the request body is in JSON. The request is sent to the URL and a response returned. The response code and body are returned and the response body is returned from the method. By using an asynchronous task, it prevents the app from hanging while the relevant data is obtained from the database, and the format described allows the class to be as flexible as possible, and forms the centre of the app, as all of the data is gathered through this class.

7 Critical Appraisal

In this section, I will summarise the project and evaluate it against the original aims and objectives, then I will discuss the project's relevance from a social, sustainability, commercial and economic context, and then from a personal development standpoint.

7.1 Summary and Critical Analysis

Firstly, there were a couple of minor changes to certain aspects of the admin application to what was initially planned; the original plan was to use a warning icon in one corner of a date's box to alert an admin if a menu hadn't been set for a day in the near future on which a location would be open. As development continued, it became apparent that the look of the screen would become more simple and minimalistic to incorporate the warning into the number on each box. The result was a functional look that also matched the style of the indicator shown when a date is selected.

On the Android side, the most notable change was allergen filtering. The initial plan was to have the menu filtered to hide items that a user was unable to consume because of the allergens they declared. However, this became impractical as it was felt that transparency was important, and it was better to allow the user to select that item if they wished, predominantly if they are ordering the item for someone else.

Menu customisation was successfully implemented, and provides significant flexibility to the admin, as they can create items in advance and be able to reuse them in other menus at a later date without having to recreate it. On the other hand, being able to reuse these customised menus was not implemented, and admins in the current system will need to complete the process individually for each day, however this time loss is mitigated somewhat by the addition of the function to add all items from a category to the menu. That being said, the structure of the system currently would support the implementation of menu reuse; the menu currently selected could be saved as a JSON file that would include a list of all the items the location had at the time and which of those items were enabled for that menu. This would be saved on the admin's file system, and be loaded later. Any items on the JSON that no longer existed would be ignored, as well as any items that existed that didn't when the menu was saved.

Furthermore, order management from an admin perspective was entirely successful, they are able to view orders in advance as well as in the past in any state. They can mark those orders as having been collected, and they can view the details of the order. On the Android app however, order editing was not implemented due to the structure of the app, although given the fact that payment is not taken until the order is collected, the user could easily delete the order and redo it correctly, but naturally this is by no means a perfect solution. The ability to edit orders is not something that is usually possible for current systems like Deliveroo or Just Eat without contacting the actual location to inform them of the change. However, given the structure of this system, it could be possible, by starting a new basket containing the items selected in the current order, then after the changes are made, the system would delete the current order and replace it with the changed order.

Finally, cost and name filtering was a planned feature of the system, however the structure of the final system did not make it possible to support this, as items are collected from the database only when a category is selected, and as such a search within the items currently stored in the app wouldn't be especially useful compared to its initial purpose, which was to search within all items at all locations in a campus. That being said, it could be possible to implement with some adjustment to the structure, namely creating a new screen that would be separate to the usual category and item selection screen. The search would include a wildcard on either side of whatever the user enters, and the query would search within the MenuLines table in the database for any items with that name included in any menu set for that day at any location in the selected campus. As a result, the same process could be applied to a cost filter.

7.2 Sustainability, Commercial and Economic Discussion

Firstly, while the system in its current state is targeted toward educational institutions, it would require minimal adaptation to make it viable for use by any other restaurant or eatery. This could be achieved by making use of the location section as a format for limiting which menu is used at a certain part of a day, for example a lunch, dinner or a la carte menu, where you could only order from one menu because of restrictions put in place by the restaurant, or because of timings as would be the difference between lunch and dinner menus. This would be of great benefit to such restaurants compared with current services, due to increased flexibility and control over how they utilise the options offered by this system. Such control is also offered by the ability to set the menus themselves and customise it without interference. This makes the system more sustainable compared to Deliveroo and Just Eat because of the increased freedom.

This system is also arguably more commercially viable due to the low running costs that Azure permits. For the purposes of this dissertation, a 2GB database is used for free from Azure, which considering the small size of the data in the database, many locations could be stored on a free basis in the first place. Naturally however, a 2GB database would not be sufficient with a publicly deployed system, and as such a larger database would be required, though this can still be achieved for a sufficiently low price. As a result, hosting a client on the system could arguably only require a relatively small monthly payment from them to cover running costs, plus a small amount extra to make the system profitable for myself as the developer. Commission would by no means be required to make a profit. As such this speaks to the viability of the system from a commercial point of view, and to the benefits a client would have from an economic point of view.

7.3 Personal Development

During development of this system, my project management skills improved dramatically. I was able to easier judge what tasks needed to be completed and in what order in order to facilitate development in the most efficient and successful manner. The general process I followed was to run through a use of the app from start to finish and develop each section of that process in that order, so links to the previous and next section were developed at the same time, which also meant testing was made easier. This project also compelled me to develop technical skills which I had previously been apprehensive or reluctant to learn. This includes cloud technology, Android app development and more advanced SQL queries. These skills have boosted my confidence and in turn improved my confidence in being able to gain a fulfilling career.

8 Conclusion

The primary goal of this project was to develop a prototype for a system for ordering food from an educational institution safely during a pandemic, and this was certainly achieved. However, more could have been done, for example the promotions was not developed as much as it could have been and there is more in the potential of that feature, but as a prototype, it was successful.

The first objective was to develop a database for storing locations, users and orders, and given the challenges faced by the Android app, this ended up being exceeded with the addition of an cloud API that could interface between the database and the Android app securely. The storage of users was also adjusted and reduced somewhat in order to fit the adjusting storage needs as the concept of passwords and identification was removed entirely save for the basic identification required to collect an order.

The next objective was to create a desktop application that could manage the administration of a campus. This too was achieved, as with only a few exceptions such as the reuse of menus and ability to create adverts, the full implementation of this application was completed.

Finally, the last objective was to develop an Android application to order food in advance for on demand collection. While this could be considered successful, as the main functions of the app have been implemented, there remain other features of the app that were not implemented, though again these features are not critical to being able to order food in advance. Also, the GUI of the app was not as developed as it could have been. Overall however, the system developed achieves the intended objectives.

9 Bibliography and Citations

[1] Department of Health, “Technical guidance on nutrition labelling” [Online], September 2016. Available: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/595961/Nutrition_Technical_Guidance.pdf

[2] Food Standards Agency. (2020, April 2). *Allergen guidance for food businesses* [Online] Available: https://www.food.gov.uk/business-guidance/allergen-guidance-for-food-businesses

[3] S. Polykalas, G. Prezerakos. (2018, Dec 10). *When the mobile app is free, the product is your personal data.* [Online] Available: https://www.emerald.com/proxy/resource?binaryId=urn:emeraldgroup.com:asset:id:binary:DPRG_21_2.pdf&docId=urn:emeraldgroup.com:asset:id:article:10_1108_DPRG-11-2018-0068&title=DPRG_21_2.pdf

[4] Just Eat. (2021). *Just Eat Takeaway & Restaurant Sign Up.* [Online] Available: https://restaurants.just-eat.co.uk/

[5] Just Eat. (2021, January, 4). *Just Eat – Privacy Policy – Consumer* [Online] Available: https://www.just-eat.co.uk/info/privacy-policy

[6] Harris J. (2020, November, 17) *Pubs and restaurants criticise Deliveroo and other takeaway apps over commission fees* [Online] Available: https://inews.co.uk/news/business/pubs-restaurants-deliveroo-takeaway-delivery-apps-commission-fees-762462

[7] Marston J. (2020, May, 7) *Deliveroo Lowers Restaurant Commission Fees to 5 Percent, but There’s a Catch* [Online] Available: https://thespoon.tech/deliveroo-lowers-restaurant-commission-fees-to-5-percent-but-theres-a-catch/

[8] Deliveroo (2021). *Deliveroo Plus* [Online] Available: https://deliveroo.co.uk/plus

[9] MySQL. (Unknown). *MySQL 8.0 Reference Manual 8.11.3 Concurrent Inserts* [Online] Available: https://dev.mysql.com/doc/refman/8.0/en/concurrent-inserts.html

Appendix 1: ApiRetrieval Class Snippet

```
public class ApiRetrieval extends AsyncTask<String, Void, String> implements Serializa-
ble {

        public static final String SETORDER = "https://prod-31.uksouth.logic.az-
ure.com:443/workflows/44cce86546e543b4b884537b345fec95/triggers/manual/paths/in-
voke?api-version=2016-10-01&sp=%2Ftriggers%2Fman-
ual%2Frun&sv=1.0&sig=wMFKGpD_PQZcGF1SrbSMRSE7jEIbHK8daVOa1kuxXvI";

        @Override
        protected String doInBackground(String... urls) {
                String request = "";
                String output = "";
                URL url;

                switch (urls[0]) {
                        […]
                        case "setorder":
                        url = new URL(SETORDER);
                        request += urls[1];
                        break;
                        […]
                }
                HttpsURLConnection con = (HttpsURLConnection) url.openConnection();

                con.setRequestMethod("POST");
                con.setRequestProperty("USER-AGENT", "Mozilla/5.0");
                con.setRequestProperty("ACCEPT-LANGUAGE", "en-GB,en;0.5");
                con.setRequestProperty("Content-Type", "application/json; utf-8");
                con.setRequestProperty("Accept", "application/json");

                con.setDoOutput(true);
                DataOutputStream dStream = new DataOut-
putStream(con.getOutputStream());

                dStream.writeBytes(request);
                dStream.flush();
                dStream.close();

                int responseCode = con.getResponseCode();

                BufferedReader br = new BufferedReader(new InputStreamReader(con.get-
InputStream()));
                String line = "";
                StringBuilder responseOutput = new StringBuilder();

                while ((line = br.readLine()) != null) {
                        responseOutput.append(line);
                }
                br.close();

                output += System.getProperty("line.separator") + responseOut-
put.toString();

                return output;
        }
}
```