The University Of Manchester

School of Computer Science

Bsc (Hons) Computer Science

Third Year Project Report

Restaurant Application System

Author: Nathan Leigh

Supervisor: Uli Sattler

April 29th, 2014

Abstract

The goal of this project is to create a software application system for a restaurant. The aims are that the application can be used for the ordering, management and the kitchen and that it should improve restaurants. The software produced went beyond the initial requirements and would be suitable in a real world environment with a few adaptations.
The main achievements is a customisable menu, stock deductor and a Nutrition calculator.
The report is split into 3 main parts, the planning and design, the implementation and the results and evaluation. This report was written by Nathan Leigh as part of his third year project in Bsc (Hons) Computer Science. The project supervisor was Uli Sattler. It was published on April 29th 2014.

Pages: 79
Word Count: 13985

Acknowledgements

I would like to thank my supervisor, Uli Sattler, for her support, encouragement and guidance throughout the development of this project.

Contents

1

Introduction

This chapter introduces the project and describes what the project hopes to accomplish and the reasons why. The chapter also gives an overview of the different chapters in the report.

1.1. What is the project

The goal of the project was to develop a restaurant application.
The application was to have a computerised system to automate and digitalise certain aspects of a restaurant such as the ordering process or how the ingredient stock levels are accounted.

1.2. What the project hopes to achieve

The project's aim is to be able to bring in new technology that can improve restaurants for customers, waiters and managers and owners. The project hopes to cut out inefficiencies in restaurants, automate and speed up certain processes and to aid the ordering options. The ultimate aim is to improve a restaurants business by providing an application with the capabilities to cut costs and that provides a faster and better customer service.

Some of the ways that the application can improve a restaurant are as follows;

- Wireless communication between the customer and the kitchen across the restaurant, will reduce the need for waiters to spend as much time taking orders, saving time and possibly the amount of staff needed, potentially cutting labour costs. Waiters can spend more time giving better service.

- Tablets can allow the customer to easily order and customise items, they can provide better customer satisfaction by allowing the customer to feel more involved and in control with their order, it will also prevent order taking errors from waiters.

- Another customer advantage with the ordering being electronic is it can calculate the running total of orders which is useful if on a budget, it can help customers with splitting the bill which can be a problem in restaurants with large groups or no calculator

- A big advantage is service, a customer doesn’t have to wait around for a waiter to begin

ordering, or wait around for a waiter to pay at the end, this can minimise customer waiting times and improves the customer service.
- The kitchen can view orders much more clearly on a well designed screen compared to bits of paper that could be difficult to view and read what is written.

- Having the restaurant running on one whole system can be beneficial for organisation and make things easier to review such as receipts, weekly sales, food stocks etc. This can be used by the manager plan ahead appropriately and revise business strategies. The system can store and backup data more efficiently compared to using a paper alternative.

- The manager can also customise the menu, add items, update the price, add discounts and offers and other features easily without printing all new menus, allows the manager to adapt and and allow the restaurant to be more flexible and reduce some expenditure.

- The ordering application can provide health benefits, it can include details about the meals such as the nutritional information or dietary warnings, this can be used by customers so they can make the best informed choice, improving customer satisfaction.

There are some disadvantages with using tablets in a restaurant such as;

- User accessibility, if customers struggle to use the application they will be put off using it, people with disabilities such as poor eyesight or dexterity may have difficulty using the device. In these cases human interaction with a waiter might be better for a customer.

- Some customers prefer being waited on and like to ask questions to the waiter before ordering so needs to still allow a waiter to order to provide this customer service.

- There are also maintenance issues of the tablets, they can become damaged or break, the cost of buying and installing devices onto tables and powering them can be costly.

The restaurant business is very competitive and lucrative, by creating new and novel ways to improve restaurants to make them stand apart would be by incorporating new technology such as the application.

1.3 Summary of Chapters

The report consists of 7 Chapters. Chapter 2 provides some background information such as existing products, available technologies and software development methodologies. Chapter 3 goes into detail about the design process. Chapter 4 explains how parts of the design were implemented and the results of the implementation are presented in Chapter 5.
Chapter 6 provides an analysis of the testing and evaluation of the application and the last

chapter, chapter 7, is a reflection of the project and application.

2 Background and literature survey

This chapter describes the background information, it explains what similar products are currently around and what technologies, development tools and techniques are available to be used to develop the application.

2.1. Existing Products

More and more restaurants are starting to add electronic devices to restaurants. Most are bespoke to specific restaurants or can be customised or adapted to be used in many. Some devices are only used by waiters while some are used by the customers also, here are some current examples of devices being used in restaurants and bars.

2.1.1. Elacarte

The Elacarte device[2.1.1. a] is a custom built tablet called the presto tablet running on Android. It represents 4 years of research by MIT engineers and restaurant experts. It uses an intuitive touch screen which is designed to withstand bumps and spills in a restaurant environment. The tablet permits dynamic availability and pricing, supports happy hour, daily specials, and out-of-stock items. The tablet allows the diner to order food, play games and allows a pay-at-table solution. They've found that the tablet cuts wait times and customers often spend more on appetisers and desserts and leave bigger tips. AppleBee's a popular restaurant in America are installing 100,000 of the Elacarte tablet for its 1860 restaurants[2.1.1. b].

2.1.2. Ziosk

The Ziosk[2.1.2. a] is a custom built 7inch tablet which offers many similar features as the Elacarte but offers a few more such as an optional built in printer, a camera and a multicolour LED to indicate order, pay and call server to waiters. It also allows guests to enrol in e-club programs, restaurants see a 300% enrolment increase with this feature. Enrolment allows guest to check in, earn and redeem awards in real time at the table, promoting customer loyalty. It has social features such as being able to share the experience on social networks. Ziosk also allows guests to give feedback. Over 20% of guests opt-in to give feedback using the device without incentive where traditional printed receipt methods yield less than 1%. Ziosk provides daily, weekly and monthly reports on sales, server performance and guest satisfaction. Currently

serves 8 million customers a months, hopes by mid 2014 that it can serve 30 million.

The research results show that there is a place for electronic devices in restaurants, it is a fairly new up and coming market for restaurants, and these examples show that there are benefits and improvements in using tablets in restaurants, the applications vary in different ways due to the software of the application or hardware of the device. There is an opportunity to get into this market and compete against these existing products if the application has advantages compared to the others.

2.2. Hardware and Software research

The first step in creating the application was to decide what software and hardware to use. The hardware requirements decided upon were that the device needed to be portable and use a touch interface, so a 7 inch tablet seemed like a good fit to this criteria.
The next step was to decide the software environment to use and information gathered from researching was that most popular mobile application software for tablets are Android, Apple and Windows operating systems in that order.

Global Smartphone Operating System Marketshare %	Q4 '12	2012	Q4 '13	2013
Android	70.3%	68.8%	78.4%	78.9%
Apple iOS	22.0%	19.4%	17.6%	15.5%
Microsoft	2.7%	2.7%	3.2%	3.6%
Others	5.0%	9.1%	0.7%	2.0%
Total	100.0%	100.0%	100.0%	100.0%

Figure 2.2.a Global Smartphone Share[2.2.a]

By researching them all in detail and weighing up the positive and negatives of each operating system in the end the **Android Platform** looked like the best platform to create and develop the application on.

Figure 2.2.b Android Logo[2.2.b]

The main reasons for choosing Android is that it best matched the standards that the restaurant application required, Android devices are generally cheaper to buy and maintain than the other devices, the GUI is familiar to a large user base and it is has many guides and tutorials available to aid with development. Android looked like it could get the best results for the application

because it offered a lot of advantages compared to the other mobile operating systems.

Apple is restrictive in ways, it requires a MAC OS X operating system to develop its applications on, whereas Android can be developed on multiple operating systems. Android has more freedom with developing options compared to Apple, it is more open. IOS applications are described as being sandboxed with very limited possibilities to communicate with each other. Android applications can publish data, which other applications can consume. iOS is restrictive in many ways. You need a certificate from Apple to even install an application on your own device whereas Android you only need a certificate when it comes to publishing the application to the market[2.2.c].

Some other reasons against using apple are the hardware costs, they tend to be much more expensive compared to other devices[2.2.d],
The maintenance costs can be more expensive as well Apple also make fixing its electronics extremely difficult[2.2.e].

Windows seems to be emerging in the tablet market with the new windows 8 metro interface. This seemed like a suitable option because a lot of people have familiarity with windows. The main reason for not choosing to develop on this operating system was the window mobile user base is not as popular compared to Apple or Android and there seemed to less development help and information available due its late arrival into the industry compared to Android [2.2.f].

2.2.1 Android

Android is an operating system engineered by Google, it is based on the Linux kernel and it is designed primarily for touch screen mobile devices such as smartphones and tablet computers. Android was unveiled in 2007, the latest version of Android as of writing is 4.4, named Kitkat. Android is designed to be powerful, fast and responsive which is needed with touch interfaces. The operating system is constantly being updated to having the latest mobile technologies, Android 4.4 devices that support NFC will include Tap & Pay for easy payments using HCE. It offers services for Wifi Communication, Security, Printing, Voice Search and much more [2.2.1.a].

Millions of mobile devices, in more than 190 countries around the world are powered by Android. It has the largest installed base of any mobile platform and it is growing fast[2.2.1.b]. It is the most popular platform for developers, used by 71% of the mobile developer population[2.2.1.c].

Figure 2.2.1.a Android Powered Devices[2.2.1.d]

2.2.2 Java and XML

Android applications are primarily written in the programming language Java, screens layouts are designed using XML. Java has many benefits such as it is well known and taught and it runs in a Virtual Machine so there is no need to recompile it for every phone out there[2.2.2.a].
XML is a very popular and familiar language used to model screen layouts and there are a number of IDE tools that natively support it.

2.2.3 Design Tools

The Android Developer Tools (ADT) plugin for Eclipse provides a professional-grade development environment for building Android applications. It has a full Java IDE with advanced features, a graphical UI builder for designing the XML screens, on-device developer options & services, the ability to develop on hardware and virtual devices, powerful debugging & testing tools and native development support[2.2.3.a].

Android provides many design tools and building blocks to assist in creating a good GUI. There are tools to get user input, navigate screens, display data in scrollable lists, alert users and many more in the API. There are tools to make sure that transitions are fast and clear by using animations. You can also change the style and colours using these tools, this can add a polished look to the application, giving the application a higher quality feel. These tools can be added by coding them in XML or by using the Graphical Layout View which generates the XML code automatically[2.2.3.b].

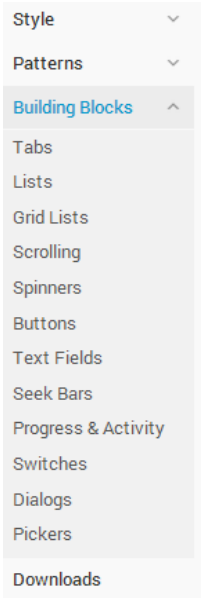


Figure 2.2.3.b
UI Building Blocks[2.2.3.c]

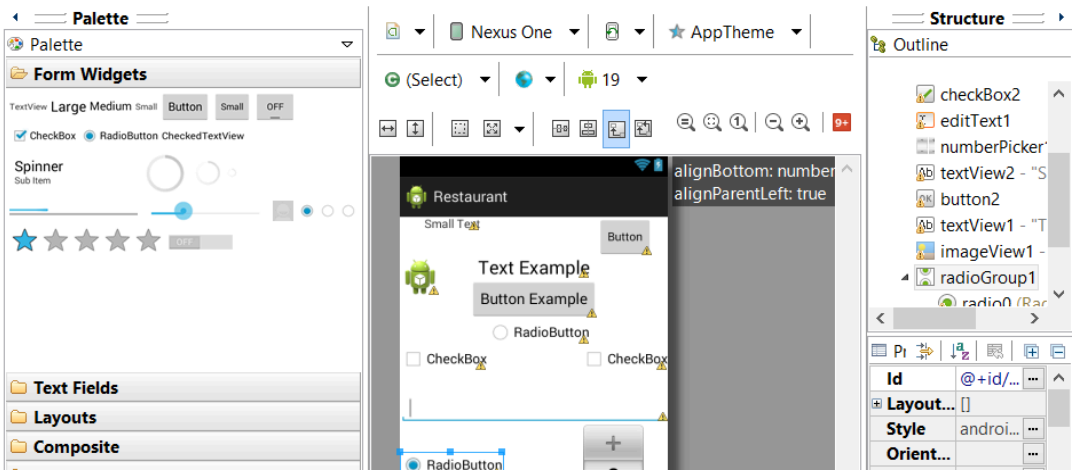


Figure 2.2.3.b Graphical UI Builder

2.2.4 SQLite

Android and Apple iOS devices both use SQLite for database management[2.2.4.a].
SQLite is a free to use software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is designed to be small, fast and reliable. It is believed to be the most widely deployed SQL database, it has bindings for a large number of programming languages and its a very popular database engine choice on memory constrained gadgets[2.2.4.b].

SQLite mains features[2.2.4.c] is that it is;

- Self-contained

A complete database is stored in a single cross-platform disk file, It has a small code footprint and uses a low amount of memory consumption, and it supports terabyte-sized databases and gigabyte-sized strings.

- Transactional

Transactions are ACID even if interrupted by system crashes or power failures. SQLite is very carefully tested prior to every release and has a reputation for being very reliable[2.2.4.d].

- Zero Configuration

SQLite does not need to be "installed" or “setup” before it is used. There is no server process that needs to be started, stopped, or configured. It requires very minimal support from external libraries or from the operating system. It uses a simple and easy to use API.

- Serverless

The main advantage to being serverless is that there is no separate server process to install, setup, configure, initialize, manage, and troubleshoot. Most SQL database engines are client/server based. Of those that are serverless, SQLite is unique in that it allows multiple applications to access the same database at the same time.

Figure 2.2.4.a Sqllte logo[2.2.4.e]

2.2.5. Eclipse Debugging Tools

The Eclipse IDE has many tools and functions to assist with catching bugs, fixing bugs and optimizing code. The Android SDK includes a virtual mobile device emulator that runs on a computer. The emulator lets you prototype, develop and test Android applications without using a physical device. The emulator allows you to create many configurations to test many Android platforms and hardware permutations. The application could also be tested on a real device, testing on a real device gives a more truthful feedback than the emulator because you can test accessibility, performance and functionality more accurately under real conditions.

When an error occurs the Eclipse IDE alerts the user with error messages to help find out where and why the error occurred. Eclipse allows the use of System Logs which can be used to follow program execution flow. This helps with understanding the program at runtime to see what methods are being called, what values variables are. Logs assist to check if everything is working as supposed to or in finding where an error occured. Eclipse has a specialised Android debugging tool called the Dalvik Debug Monitor Server(DDMS)[2.2.5.a]. It has many features, it can be used to set breakpoints and view variables during execution. It also has options to view the memory allocation, heap, files and the currently running threads.

2.3. Agile Practice

An important design choice was to choose a development method to follow which provided the best help and support with designing, building and testing the application.The Agile development practice was chosen to be used for this project because of lot of its practices and principles provided better development advantages compared to other practices[2.3.a].
Agiles main selling point is its iterative and incremental approach which allows you to be flexible and adaptive in responding to change, these features seemed suited for the project at hand.
The principles helped greatly in designing and developing the application[2.3.b].
Some of the agile practices used and found very helpful in developing the project were Paper Prototypes, User Stories, Learning Spikes and following the Incremental and Iterative approach.

2.3.1 Project Plan

The Agile structure recommended evaluating progress frequently, this process used with a project plan, helped to see how on schedule you were and to adjust accordingly.
A project plan can be used to organise and assist with the time management, it can account for unexpected problems such as technical issues, exams and other events which may slow down development. A project plan is a useful way to make sure what features and functions were prioritised before the project deadline.

2.3.2 Paper Prototypes

The Paper Prototyping practice[2.3.2.a] was used because it allowed quick experiments and comparisons between different designs and screen layout ideas, these could be evaluated very quickly. The ability to easily draw and scribble examples on paper provides a fast and efficient visual representation of how the screens should appear and interact. It offers freedom to play around and express creativity compared to making the changes in code, which could take a long amount of time. Paper Prototypes assist with designing a GUI to be as user-friendly as possible, it permits the designer to just concentrate on their vision and innovate without concerning about any of the implementation aspects.

Figure 2.3.2.a Device Template [2.3.2.b]

Another part of the project that Paper Prototypes can be useful is in the designing of the database system. By using Paper Prototypes drafts and UML models of the database can easily be created and edited. Models and table attributes can be drawn to show how they interact. Over time the models can evolve by iteratively adding elements gradually over time.
This design evolution process improves and defines the models, achieving a better designed database system. The requirements of the project were likely to change a lot so the quick ability to redesign screen layouts and the database would be extremely advantageous.

There are lots of design principles to learn and Paper Prototyping helped to check if the application was following them to achieve the best visual screen or database design possible.

2.3.3 User Stories

User Stories[2.3.3.a] provide a way to specify what the users of the application should be capable of doing. They help in describing and evaluating the requirements more clearly, they also offer the ability to create a virtual restaurant environment and perform all the user interactions that are possible. An example of user stories;

- **Waiter** – a waiter can open a table and begin taking orders for the table.
- **Kitchen Staff** - a kitchen worker can view and navigate a list of the most recent orders.
- **Manager** – a manager can edit the prices of menu items.

User Stories can be written that react with one another. This was useful for getting a sense of all the interactions capable within the application. User stories were very helpful in designing the functionality of the application and testing to see if it had achieved certain goals and

requirements.

2.3.4 Learning Spikes

Learning Spikes[2.3.4.a] is a practice used to gain familiarity and knowledge of new technology quickly. The application was being developed in the Android environment. The author had never used this environment before and didn't possess any skills in Android Application Development. To overcome this lack of understanding the Agile practice Learning Spikes was applied.
It advises to focuses on learning new useful technology efficiently by researching, training, studying and practising. The more that was learnt about Android development the better and faster the application would be developed.

The Android Environment is constantly being updated with new features to use, it has vast libraries in the SDK that can be used to improve the project. Android also provides many helpful application samples that can be use to learn from. The Android website provides an Android training course for beginners which was good way to apply the Agile Learning Spike practice.

It was important to find a good balance between spending time learning new things and actually developing, its was also essential to focus on learning skills which were of use to the project. The project progression was slow at the start, coding often produced errors which took up a lot of time fixing. As time went on this happened less often as more familiarity and experience was gained. This practice was necessary for the project and very beneficial in discovering new design and implementation techniques.

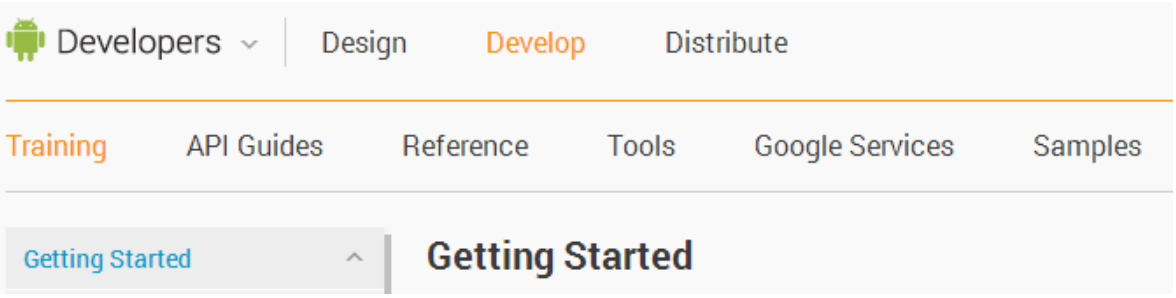


Figure 2.3.4.a Screenshot of Android Developer website [2.3.4.b]

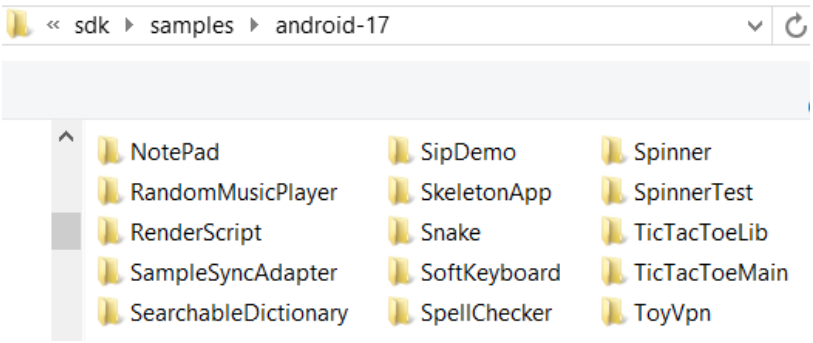


Figure 2.3.4.b Screenshot of sample library

2.3.5 Iterative and Incremental Approach

This practice[2.3.5.a] was very useful for this project, especially the incremental characteristic. Development progress was slow at the beginning, only basic things could be built at first, but as time went on and development skills improved, more and more could be added, so what started off as something simple could turn into something complex. It was also important not to try to take on too much at once and get discouraged, which with using a new API and environment, the author wanted to avoid.

The iteration aspect also proved useful, for the 1st iteration a prototype was created, it was a good basis to get practice with Android development and some of its tools. The prototype was small and simple and it was able to evolve over time, it was adapted and fleshed out slowly with more features. This cyclic approach allowed opportunities to have something of real value to show to the project tutor and receive feedback for improvements and advice, it also helped to manage how close development was with the project plan schedule. This practice proved very useful in helping to manage time and in building a complex application and database system.

2.4 Chapter conclusion

The best solution found to tackle the project and reach the objectives that were set out was to use the Android operating system. It has all the features that are required to make the restaurant application. It is popular and familiar to lots of people. Android tools provide the ability to design good user interfaces using XML and Java. Data can be communicated and managed well using SQLite. Android hardware tends to be the cheapest to buy and maintain out of Apple and Windows. The Agile Practice contains principles which will aid the projects development.

3 Design

This chapter describes the approaches and decisions used in designing the GUI, database and the behaviour of the application.

3.1. Users, Project Plan and Requirements

As mentioned earlier in the user stories section 2.3.3, planning what users were going to use the application and what they are able, and not able to do, was an important design step. Originally the users were a waiting staff, kitchen staff and the manager of the restaurant, but during development an extra user was added, the customer.

Customer User – A person who visits the restaurant and orders items.

Waiter User – A person who works in the restaurant and provides service.

Kitchen User – A person who works in the kitchen, views and prepares the incoming orders.

Manager User – A person who controls the menu and manages stock levels.

A project plan was created to to try and organise and structure the projects development. The project plan was used as a guide to how time should be spent to finish the project before the deadline. Here is an example of the project plan for February and March.

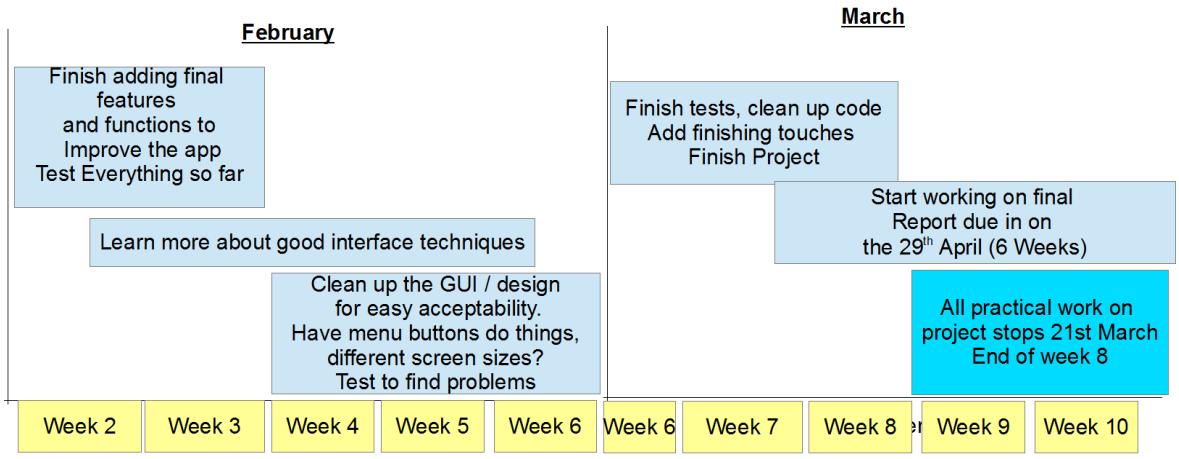


Figure 3.1. Project Plan Extract

Requirements gathering and deciding what functional and nonfunctional attributes were required by the application was necessary to begin the design process. As the project went on the requirements changed, these were a few of the original functional requirements.
These requirements listed provide a set of objectives that the final application hoped to achieve.

Customer Functional Requirements
Customer can begin using the tablet, which starts a table session.
Customer must be able to add items from the menu to their current order.
Customer must be able to access information about each item, such as suitable for vegetarians, contains nuts, lactose and calorie information if available.
If the menu item has options, a customer can easily choose one and the current item price can change to reflect the option chosen.
If the menu item has addons, a customer can easily add them to the item and the current item price can change to reflect the addons added.
Customers can not order menu items to the order that are out of stock.
Customer is able to add order comments to the order.
Customer must be able to confirm current order is complete.
If a customer does not wish to use a tablet then a waiter can take orders.
Customer can send orders throughout the duration of the table session.
Customer can ask to pay the bill which sends an alert to waiters.

When the bill is sorted the table session will end.
Customer can give feedback on the order using the tablet.

Kitchen Functional Requirements
When a current order is complete its gets sent to the kitchen to be viewed.
Each order has an ID, time issued and table number.
All current orders can be easily navigated and viewed by the kitchen.
Orders will be prioritised on the list, the most recent orders will appear at the bottom.
Kitchen staff can use the application to notify waiter when an order is complete.

Waiter Functional Requirements
Waiter can open the table for the session.
If a table is already opened, a customer on the table who wishes to use a waiter to add orders to that current open table session can do so.
Waiter must be able to select which table is ordering.
Waiter can add menu items to the current order, choosing the options and addons that the customer asks for, and add order comments to this order.
Waiter can send the completed order to the kitchen.

Manager Functional Requirements
A manager must be able to adjust the stock levels of ingredients.
A manager can set up a warning feature for when stocks of certain items are low
When a current order is complete the constituent ingredients that make up each item in the order is deducted from the stock levels.
A manager can view the order history of the restaurant, it can be viewed by day,week,month,year and can be sorted to show most and least popular orders.

The manager can preload the database with ingredient data .
The manager can adjust the number of tables in the restaurant.
A manager can create a menu for any type of restaurant.
A manager can create different menu sections, and add menu items to the sections.
A manager must be able to create, edit, duplicate and delete items from the menu.
A manager can add discounts and offers to menu items and orders.

Menu Functional Requirements
A menu item can have a description, including a picture/s of the item.
A menu item can consist of raw ingredients and complex ingredients.
Complex ingredients consist of other raw ingredients.
Complex ingredients can consist of other complex ingredients.
Ingredients can have details such as unit, nutritional information and dietary warnings.
A menu item can be given options, such as size, how its cooked and any other type of options.
A menu item can have addons/extras, which can consist of a ingredients.
Different options can have different ingredient amounts.
All menu items must have one default option and price.
Different options can have a different price.
The addon ingredient amounts and price can be affected by the options, e.g. a small pizza with an addon of mushrooms could cost less and use less ingredients compared to the addon of mushrooms on a large pizza.

Customer, Kitchen and Waiter Non-functional Requirements

Must be very accessible and intuitive to use, provide help and more information options available to users.
Must be efficient to use, ordering should be simple and fast using the tablet, touch interface must have a very fast response time.
Must be robust, if faults occur must be able to recover.

Manager Non-functional Requirements
Must have disaster recovery such as backups to maintain the databases if faults occur.
Must be reliable, stable and of good quality.
Must be secure, restrict access to manager functions if not the manager.
Creating menu items must be straightforward and not complicated or cumbersome.

3.2. Graphical User Interface Design

A big challenge in the project was designing the GUI. The GUI was being designed for a 7inch tablet screen, so the design and screen layouts had to be planned for a screen of that size. The GUI also had to to be designed for the different screen orientations possible, the screen can dynamically change or be set to be displayed vertically or horizontally. The screen layouts and interactions all had to be planned, designed and tested using paper prototypes before being implemented using XML. The GUI needed to have a good user experience and look great. The application interactions needed to be simple, intuitive and as easy to use as possible. This required applying lots of creative design techniques and clever ideas and vision to achieve the best possible GUI design for the application.

3.2.1. Design Principles

The Android Team created a list of design principles to follow which are trailered to improve the Android User Experience. The key point of the principles is that the Android GUI should be designed to keep the users' best interest in mind[3.2.1.a].

The design principles advise to only show the user things what they need when they need it, to make every pixel count and to allow the user to know where they are in the application at all times. The principles discourage making the user feel overwhelmed with too much choice on the screen. Another of the principles is that simple tasks should never require complex procedures, it instructs designers to break any complex tasks into simple steps. The list states that pictures

are faster than words because users will absorb the information more quickly.
As well as principles to follow Android advices to follow certain design patterns[3.2.1.b] which promotes a universal design throughout the application. Using a universal design improves the consistency and predictability of the application thus making the end-to-end experience feel seamless and cohesive. The use of design patterns is encouraged wherever applicable so users don't face a learning curve.

Interactions can happen in a variety of ways, the user can use gestures – touch, long touch, swipe or drag, double touch, pinch and can even use voice commands to interact, these should be designed in ways that purposefully engage and retain users. Users enjoy touch feedback and encouragement on actions. Thought should be put in to make the user interface design powerful, robust, enjoyable, effortless and user friendly.

The Android design principles and patterns helped in designing a good user experience and GUI which was visually compelling and where the users felt firmly in control when using the application.

3.2.2. Accessibility

Accessibility was an important consideration as part of design process, especially with one of the users, the customer, being a random person. They could have issues or disabilities which hinder using the application such as unfamiliarity with using devices, visual impairment, color deficiency, hearing loss or limited dexterity. There is the screen size, brightness, colours and contrast settings which have to be considered with accessibility as well. To try and prevent accessibility issues the application followed the Android accessibility guidelines[3.2.2.a].

The Android design principle "I should always know where I am" is key for accessibility concerns.

> “All users benefit from a strong sense of information hierarchy and an architecture that makes sense. Most users benefit from visual and haptic feedback during their navigation (such as labels, colours, icons, touch feedback). Low vision users benefit from explicit verbal descriptions and large visuals with high contrast.” [3.2.2.b]

Android lists ways to design an application to be as universally accessible as possible, one of them is to make navigation intuitive. The application navigation needs to be planned so that screens can be easily reachable from others and that the navigation from screen to screen is be consistent and behaves in a predictable fashion. Android provides navigation buttons and action bars to navigate a screen hierarchy by going back to screens previously visited or to go up a level. A rule in screen navigation is to keep the hierarchies shallow by using horizontal navigation and shortcuts.

An accessibility solution for users with dexterity disabilities is to always use recommended touch target sizes. All elements should be designed to be at least 48dp high and wide and spacing between each UI element is 8dp. This helps to guarantee that targets will never be smaller than the minimum recommended target size of 7mm, regardless of what screen they are displayed on. A good compromise needs to be struck between overall information density and target ability of UI elements.

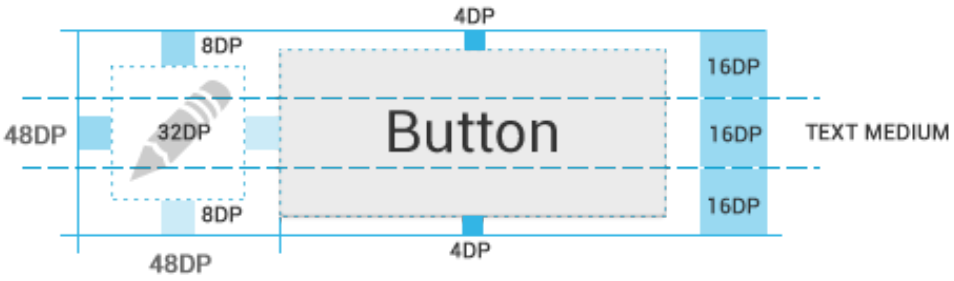


Figure 3.2.2.a Image of element spacing recommendations [3.2.2.c]
Density-independent Pixels (dp) - an abstract unit that is based on the physical density of the screen.

Another way to make applications accessible is to label visual UI elements meaningfully. Android advices to keep text concise, simple, and friendly. Text should just describe only what the user needs to know and be easily scannable and not contain redundant information. The text should be displayed using a well designed typography so that the font and size are easily readable. It is important to make sure text is punctuated and formatted correctly. Text should be designed so
that the tone is casual and conversational and avoids slang and certain words[3.2.2.d].

There are many other design properties you can add to an application to improve the accessibility, by following these accessibility design guidelines the application will become more usable to wider range of people with varying abilities.

3.2.3. Navigation Design

The navigation hierarchy for the screens was designed to minimise the number of touches to access data, while keeping the interface intuitive and consistent. Below are screen maps displaying the navigation between screens.

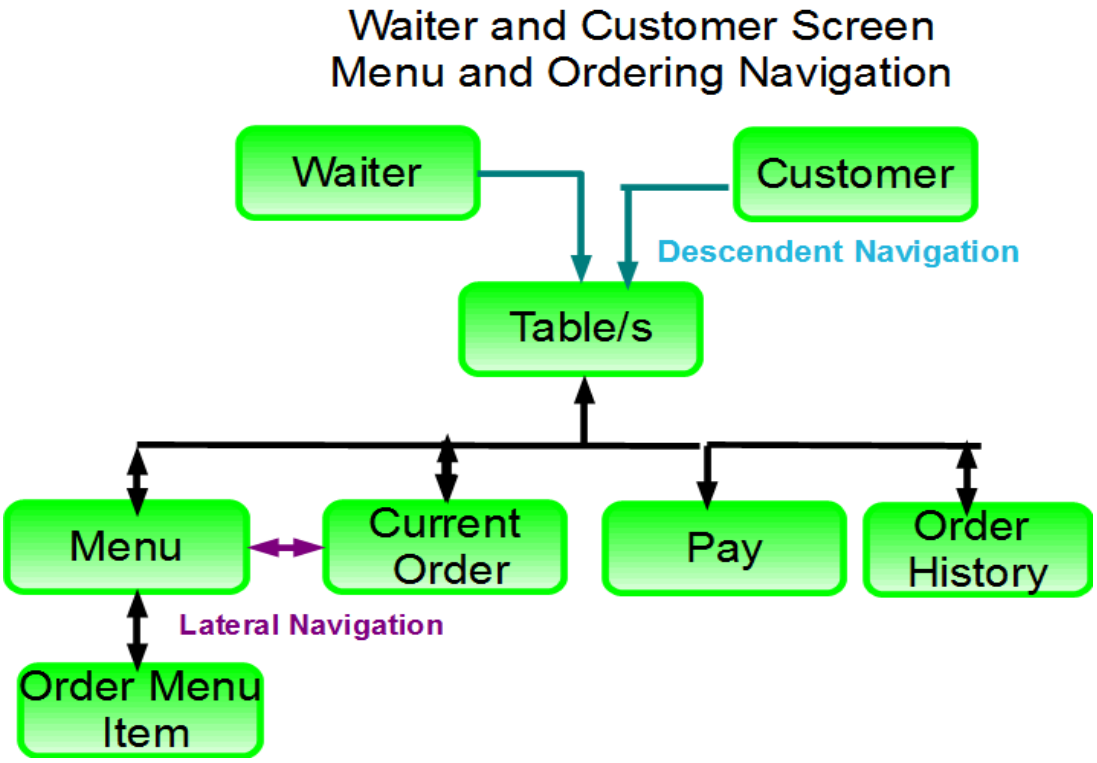


Figure 3.2.3.a Waiter and Customer Screen Navigation Model

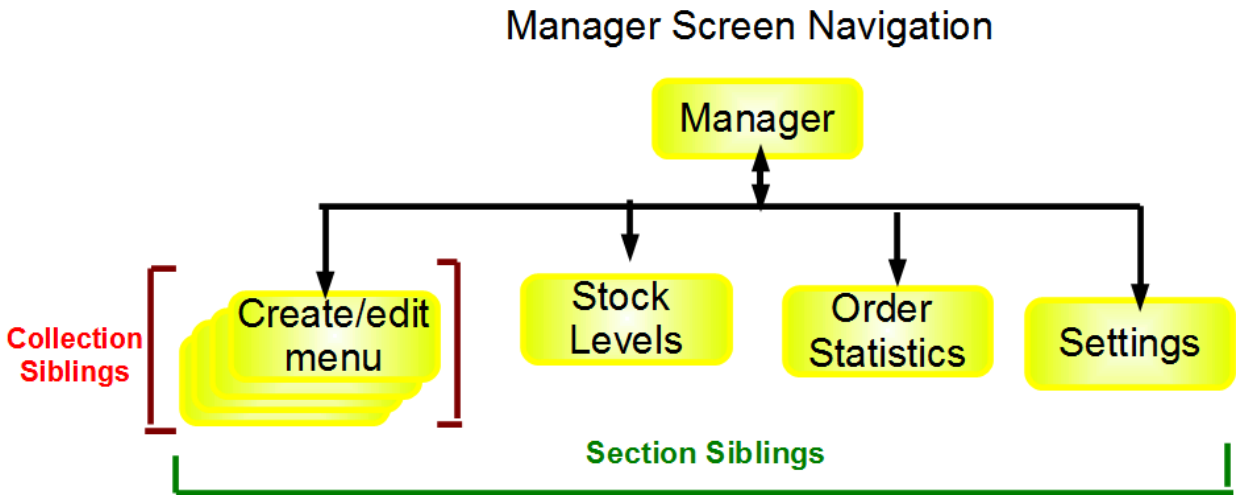


Figure 3.2.3.b Manager Screen Navigation Model

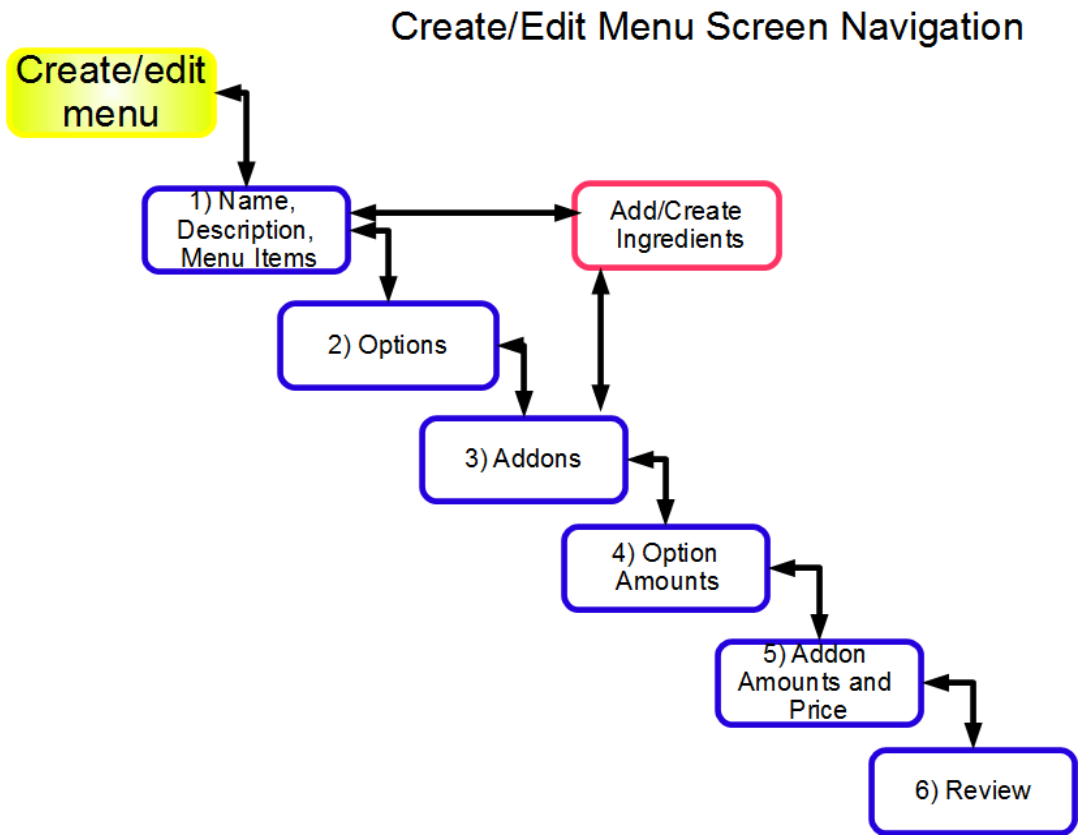


Figure 3.2.3.c Create/Edit Menu Item Screen Navigation Model

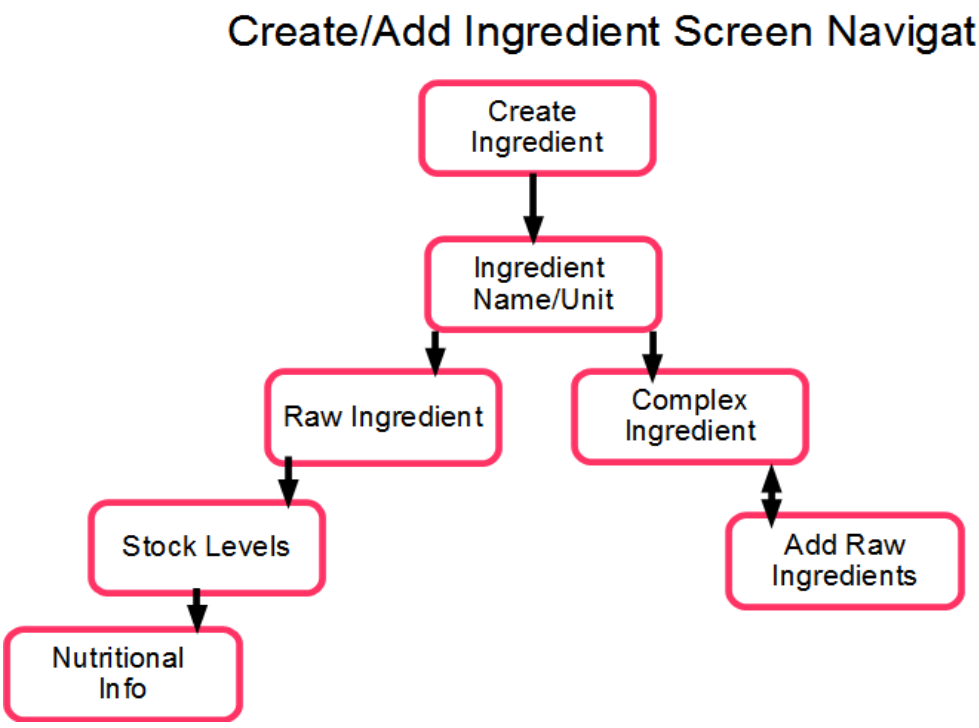


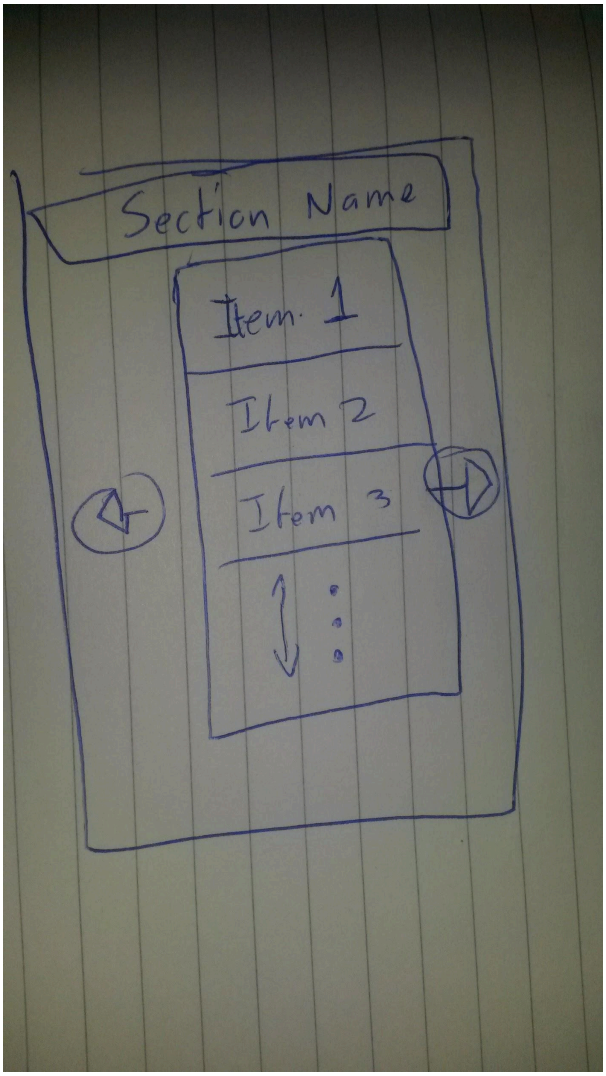
Figure 3.2.3.d Create Ingredient Screen Navigation Model

3.3. Screen Design

The following section describe how the application screens were designed . All of the screen appearance design was first done using paper prototypes.

3.3.1. Menu GUI Design

The menu was designed so that it could contain lists of menu items, these menu items could be split into different sections, this allowed similar menu items to be grouped together for easier navigation. The arrows in this paper prototype were buttons which would navigate to different sections.



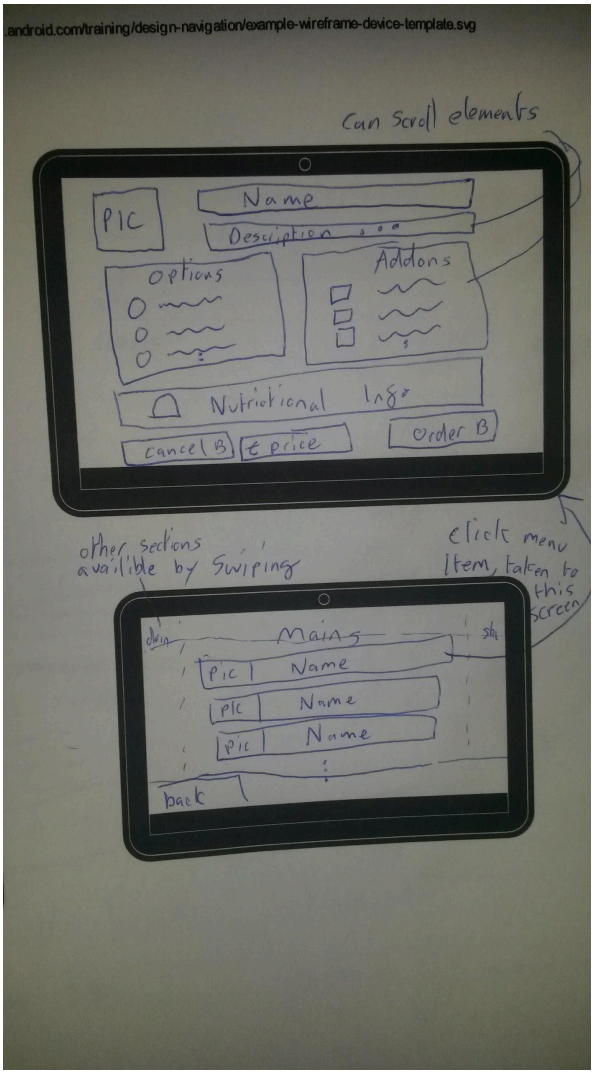


Figure 3.3.1.a Menu Sections Paper Prototype

Figure 3.3.1.b Menu Item and Sections Paper Prototype

3.3.2. Menu Items Design

The application requirements was that any type of menu item could be able to be created so the menu items had to be designed so that they were very customisable. To achieve this customisability a structure was designed for menu items;

- Menu Items can have a list of options to choose from that are mutually exclusive.
- Menu Items can have a list of addons which can be added to the menu item, addons are potentially inclusive.

The options choice allowed customisation in various ways, for example menu items can have size, cooking, type option choices and variations. A menu item **must** have at least one(default option), this design choice was to make sure the menu item had a price. The addons also allowed more customisation by adding extra components to meal items.

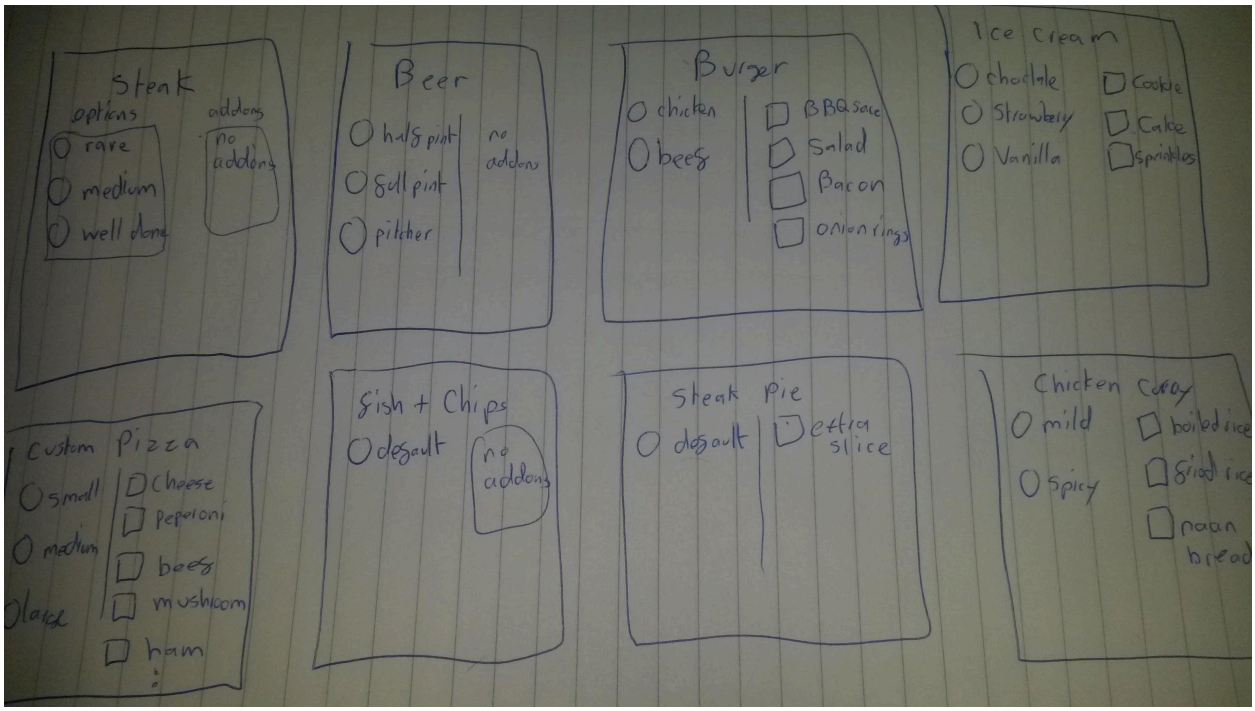


Figure 3.3.2.a Menu Item customisation possibilities

A design solution for the price was to allow menu item options and addons to be given a price
The price of menu items and the addons were designed to be dependent on the option.
This allows more price control and customisation because different options can affect the price of the menu item and the addons. Addon prices can be dependent on each option.
e.g. extra cheese on a small pizza = 50p, extra cheese on a large pizza = 80p.
Because there must be one default option this made sure that an item can be given a price.

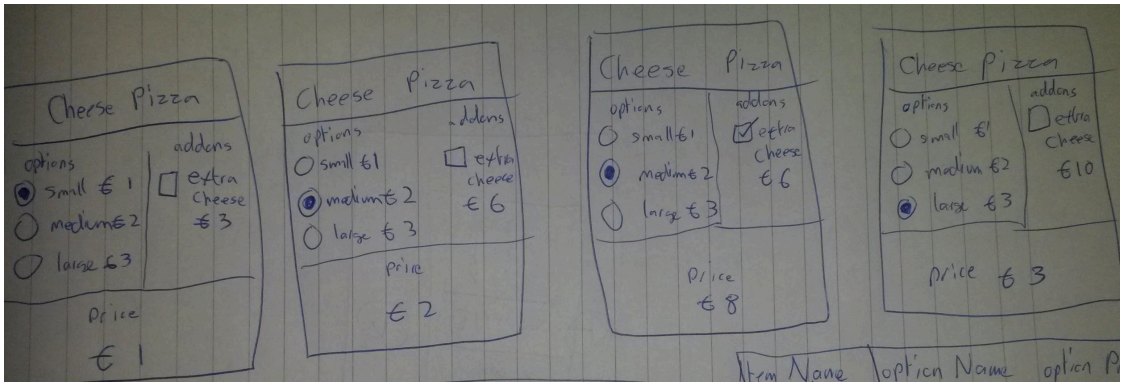


Figure 3.3.2.b Paper Prototype showing price change when addon or option changes

3.3.3 Ingredients Design

Menu Items were designed so that they could consist of Ingredients, this was so that ingredient stock levels could be affected by orders. Menu items do not have to consist of ingredients, the possibility of a manager not wishing to use the stock level feature was planned for as well. Ingredients are either grams(g) or (ml), this choice was to keep the units of measurements as simple and standard as possible.

Menu items that do consist of ingredients can be given set amounts, these can be customised depending on the options of the menu item. Addons of a menu item consist of an ingredient. The amount of the ingredient that makes up the addon can be customised depending on the menu item option/s. This ingredients design allows ingredients and ingredient amounts of a menu item to change depending on the option selected and the addons selected.

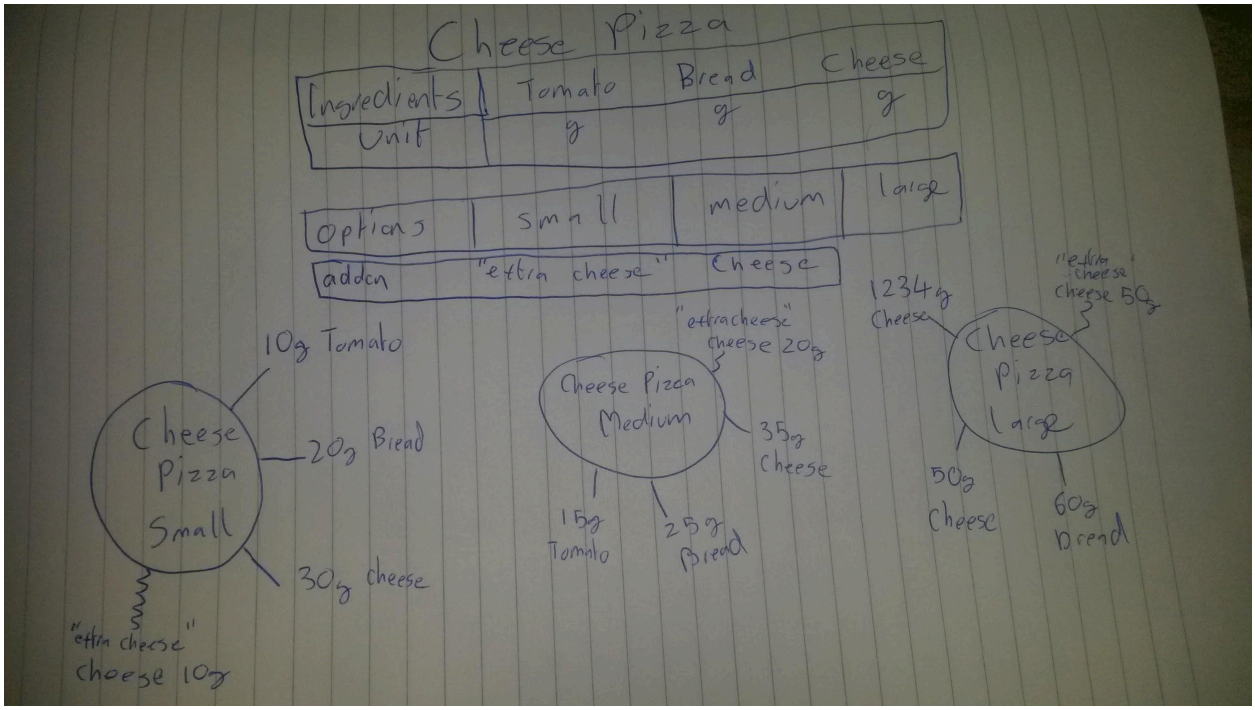


Figure 3.3.3.a Prototype of the options containing different ingredient amounts for the menu item or addons

3.3.4. Nutritional and Dietary Information

Ingredients were designed to have nutritional and dietary details stored about them so that a menu item can display or update this information automatically and dynamically.

The nutritional information followed the structure of the traffic light system used by the Food Standards Agency[3.3.4.a]. Ingredients can store the Calorie, Fat, Saturates, Salt and Sugar amounts for a certain amount of the ingredient. Ingredients could contain dietary warnings details such as if it is suitable for Vegetarians, contains Gluten, Lactose or Nuts.

This ingredient design feature gave menu items the ability to dynamically calculate the nutritional information and any dietary warnings for the menu item. Ingredient amounts can change depending on the options or addons selected.

Menu item are designed so that they will calculate the new nutritional or dietary information when a different customisation(option or addon) is selected.

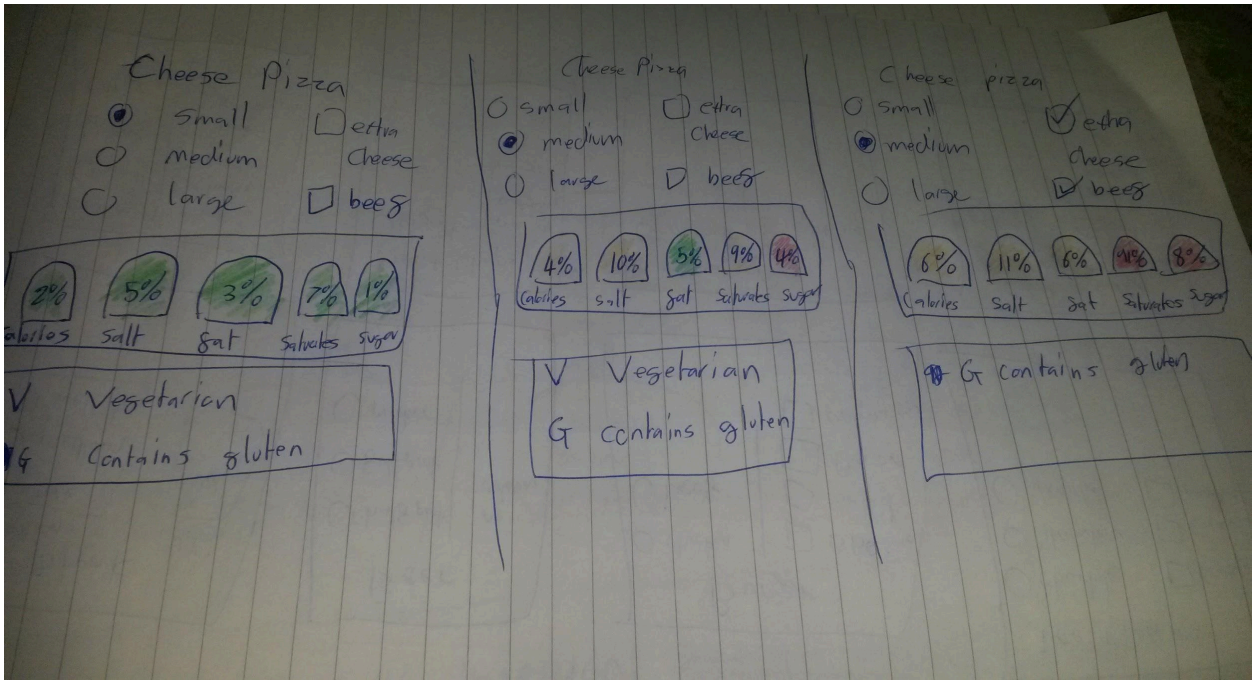


Figure 3.3.4.b. Paper design of nutrition values and warnings changing when option/addon change

3.3.5. Raw and Complex Ingredients Design

In the original design plans, items just consisted of simple ingredients, during development an issue with this was found. When entering ingredients and their amounts for menu items with lots of option and addons, it could be a very long and repetitive process. Also addons were restricted to one ingredient which limited customisability.

A solution to improve the efficiency of inputting the ingredients and their amounts and to allow addons to consist of multiple ingredients was designed.

The solution was to allow ingredients to consist of of other ingredients.

- An ingredient would be raw if it did not consist of other ingredients.
- An ingredient would be complex if it was made up of other ingredients.

Ingredients can be raw ingredients such as carrots, tomatoes, chicken breast etc.
Ingredients can also be complex I.e. they consist of raw ingredients.
Complex ingredients were designed to be able to consist of other complex ingredients.

e.g. Pepperoni Pizza → Cheese and Tomato Pizza Base, Pepperoni.
Cheese and Tomato Pizza Base → Cheese, Tomato Sauce, Dough.
Tomato sauce → Tomato, Garlic, Onions.
Dough → Flour, Yeast.

'→' means consists of the ingredients

The Pepperoni Pizza is split into 2 ingredients, where it actually consists of 7 raw ingredients; pepperoni, cheese, tomato, garlic, onions, flour and yeast. This solution allowed pre-set ingredients to be created and used in other menu items, this was able to save time inputting menu item ingredient data and to allow addons to consist of multiple ingredients.

Once the Pizza Base complex ingredient was created it could be used for all the other pizza bases without having to enter every single raw ingredient.

e.g. Pepperoni Pizza → Cheese and Tomato Pizza Base, Pepperoni.
Margarita Pizza → Cheese and Tomato Pizza Base.
Chicken and Sweetcorn Pizza → Cheese and Tomato Pizza Base, Chicken, Sweetcorn.

3.3.6. Menu Item Creation Design

One of the requirements was that menu items can be created and updated. The solution designed for this was to use a wizard. Each step of the wizard would allow details to be added about the menu item. The steps of the wizard are;

1. Menu Item Details and Ingredients
2. Menu Item Options Names and Price
3. Menu Item Addons Name and Ingredients
4. Menu Item Options Ingredient Amounts
5. Menu Item Addons Ingredients Amounts and Price for each Option
6. Review and Confirm

The wizard screens can be navigated by pressing next or back.
It could be used for creation, where new values are added, or for updating, where the old values are loaded into the wizard and can be changed.

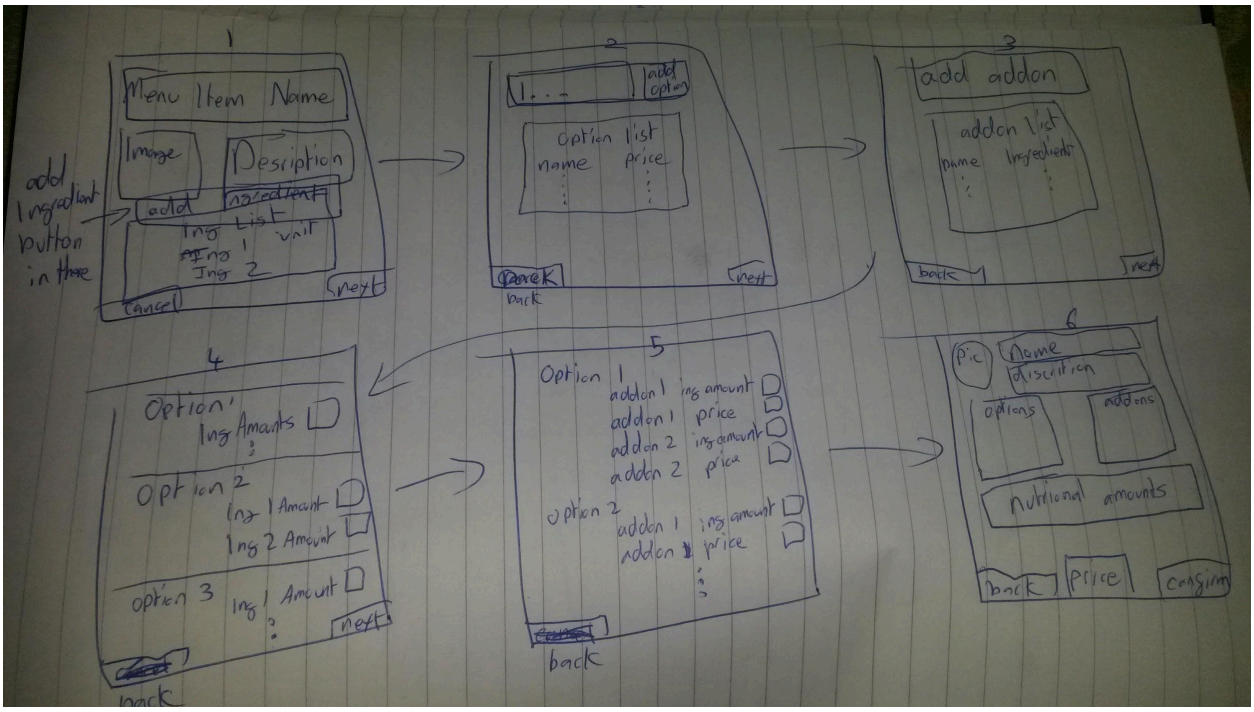


Figure 3.3.6.a Paper Prototype Image of the wizard screens and navigation

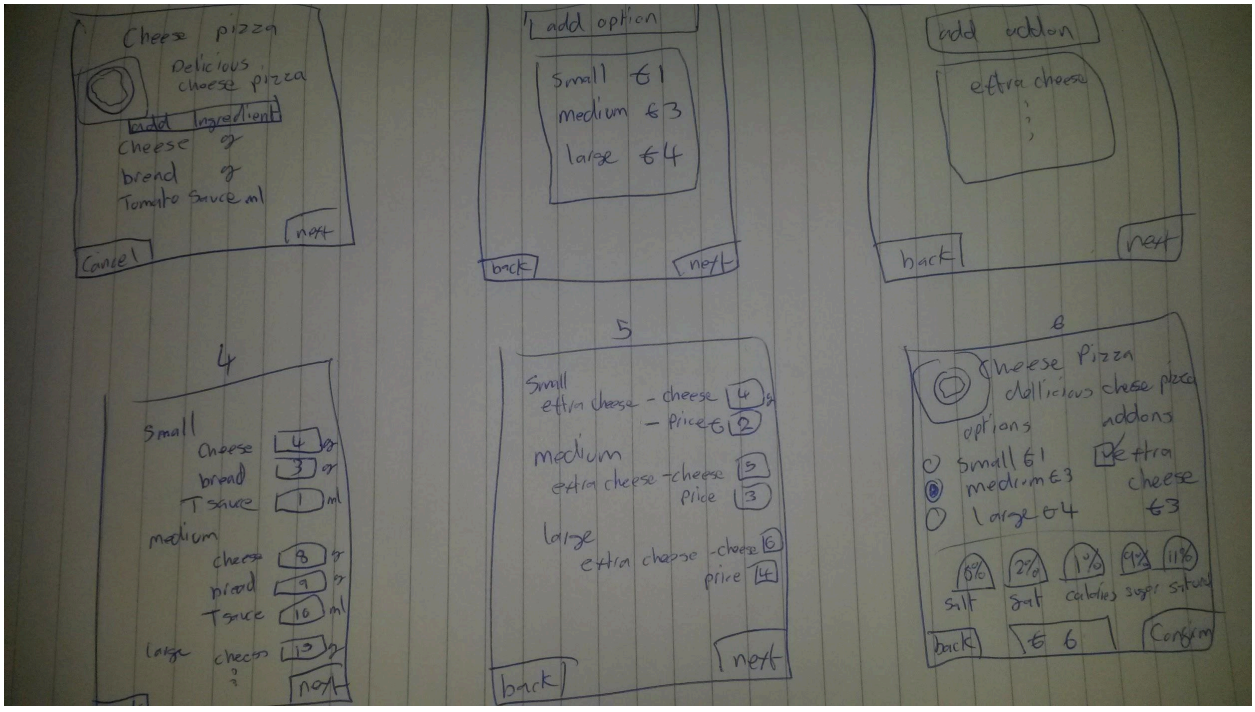


Figure 3.3.6.b Paper Prototype Image of the wizard screens and navigation for a Cheese Pizza

3.3.7. Ordering Design

A solution to allow both waiter and customers to order on a table was to design a feature called table sessions. Tables have 2 states, open or closed. When a table is open a both the waiter and customer can order menu items under this session.

During a table session all menu items that get ordered get added to a current order list and displayed on the current orders screen. This screen is used as a quick check to see if everything is OK with the user and to add order comments before confirming the order. Once confirmed this order gets sent to the kitchen. Other orders for this session can continue to be placed as long as this session is open, each new set of orders is called a group of orders.

The order history of the session can be viewed and the total price of all the orders for this session is shown. The different groups are split and each group price is shown, This helps when it comes to splitting the bill because a “group” of the orders could be for each individual. When the entire order is paid then the table session is ended and the table state returns to closed.

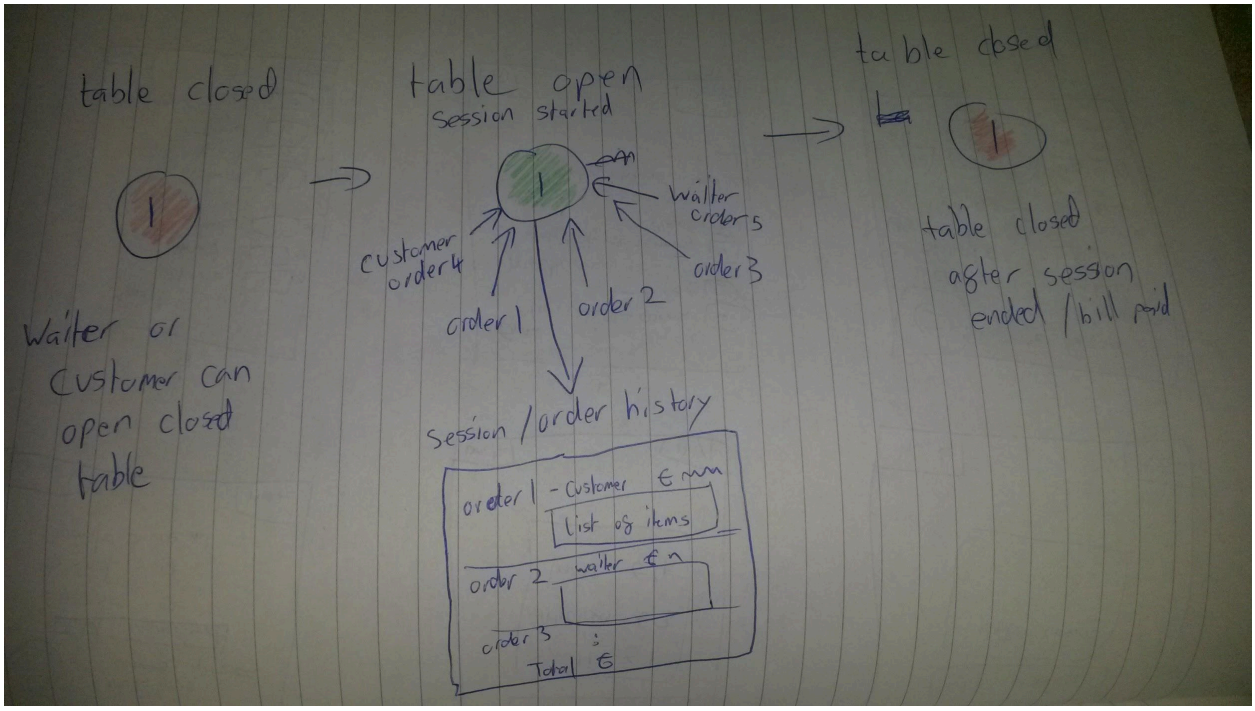


Figure 3.3.7.a Image of table design prototype, each order can consist of many menu items

3.3.8. Kitchen Design

The Kitchen GUI is designed so that the screen shows information about recent orders in a clear readable layout that can be easily navigated by scrolling to show orders that do not fit in the visible screen space.

The kitchen needs to display information about the ordered menu item, its name, option and addons(if any). It needs to list the time of the order, so priority can be given to later orders and the table number so the when the order is complete it can be taken to the correct table.

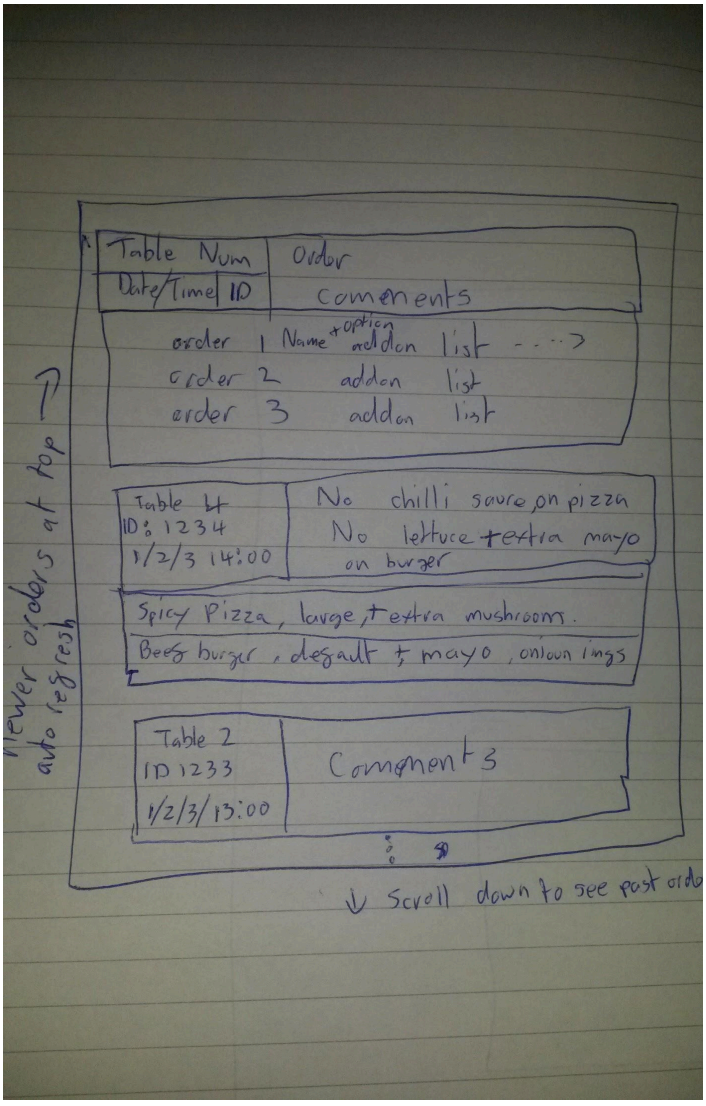


Figure 3.3.7.b Image of kitchen design prototype

3.4 Database Design

The database system needed to provide solutions for when information needed to be created, stored, sent, read, updated, duplicated and deleted for objects such as the menu, ingredients, orders and other components of the application. This required a lot of design to tackle this problem effectively and efficiently. Simple tables were created at first to get a feel and understanding of how the information will be stored and interacted with, and then they were transformed into more complex tables and interaction models.

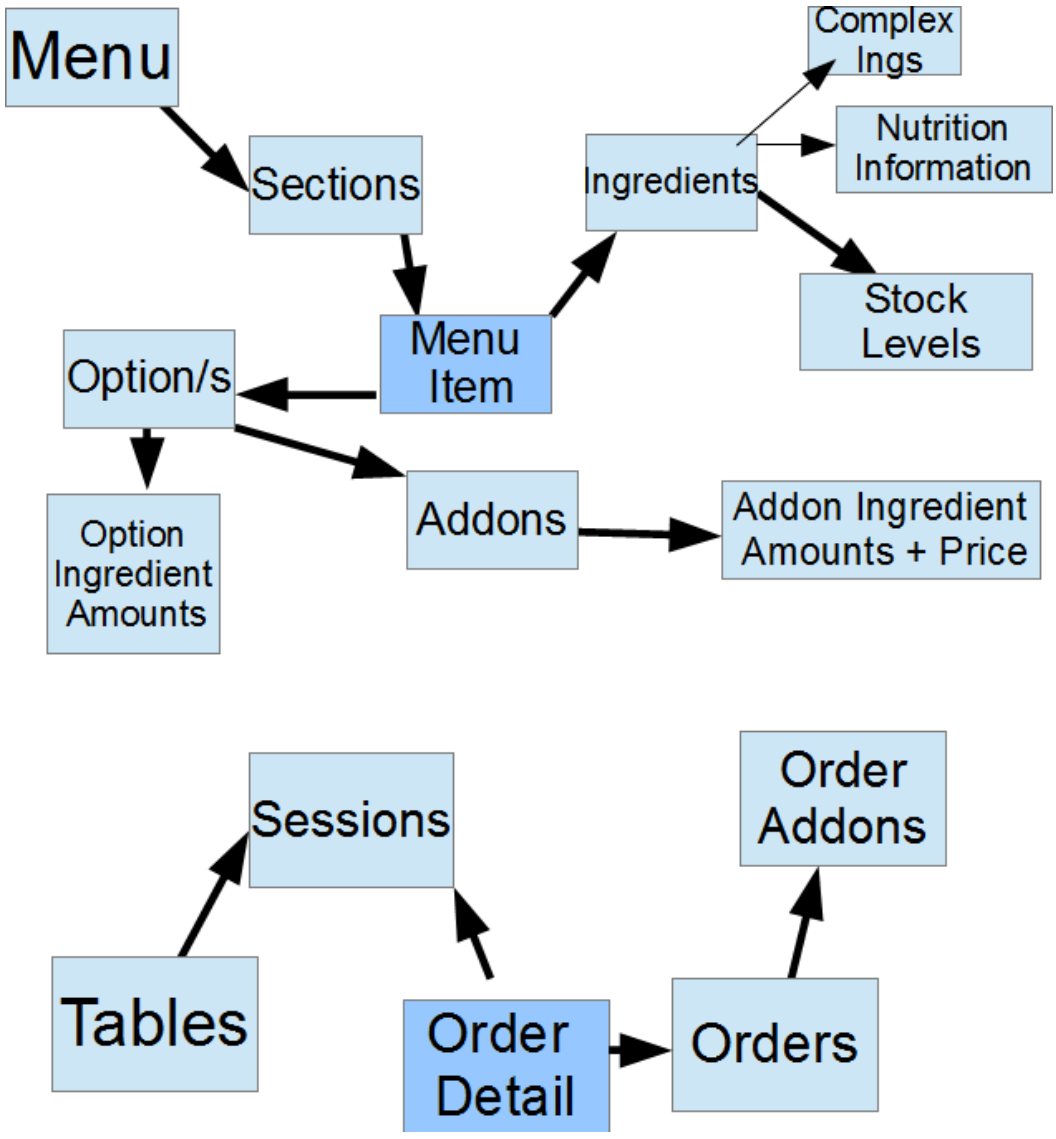


Figure 3.4.a Model of DB tables and interactions for Menu Items and Orders

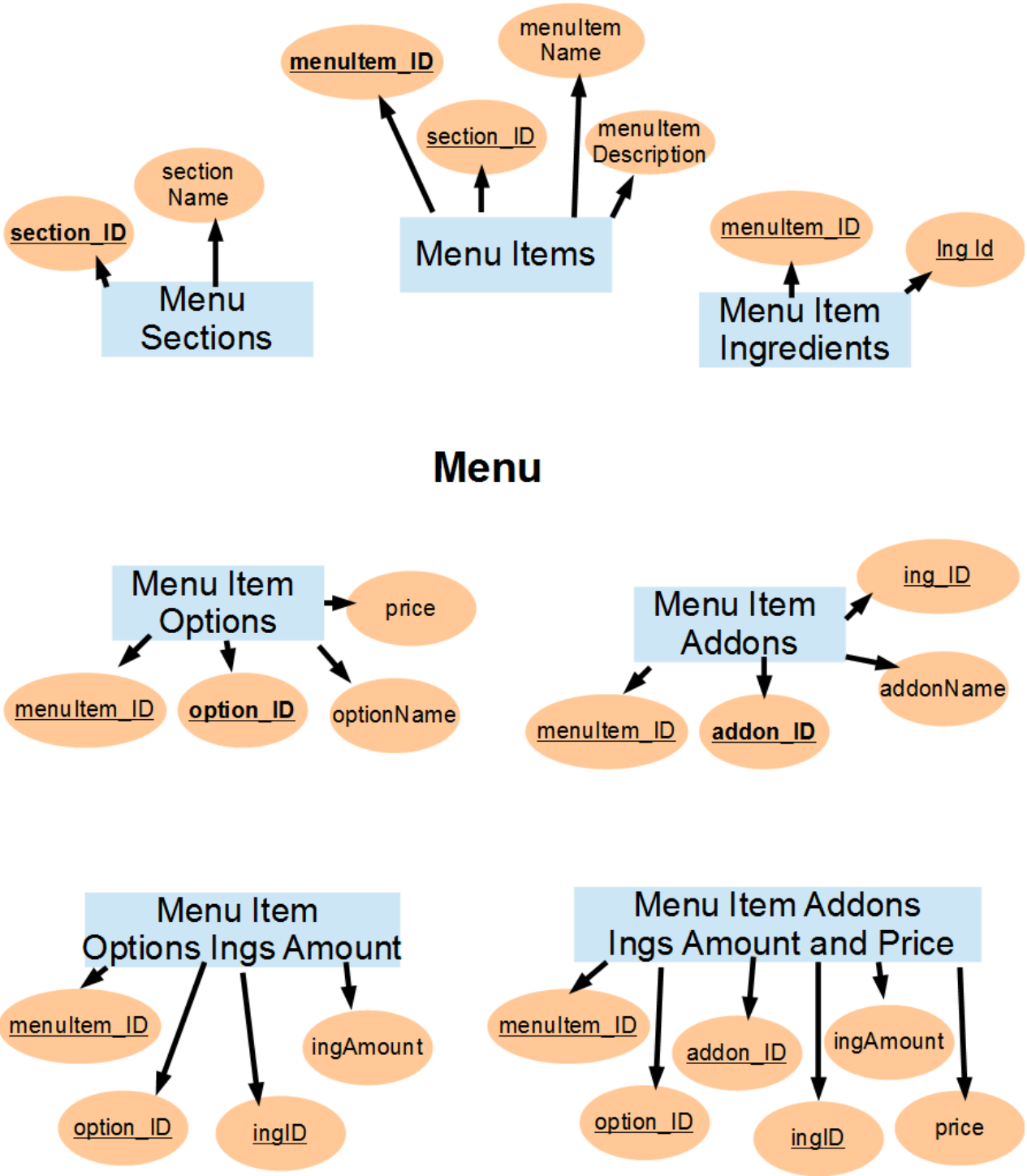


Figure 3.4.b Model of DB tables and their attributes for the menu and menu items

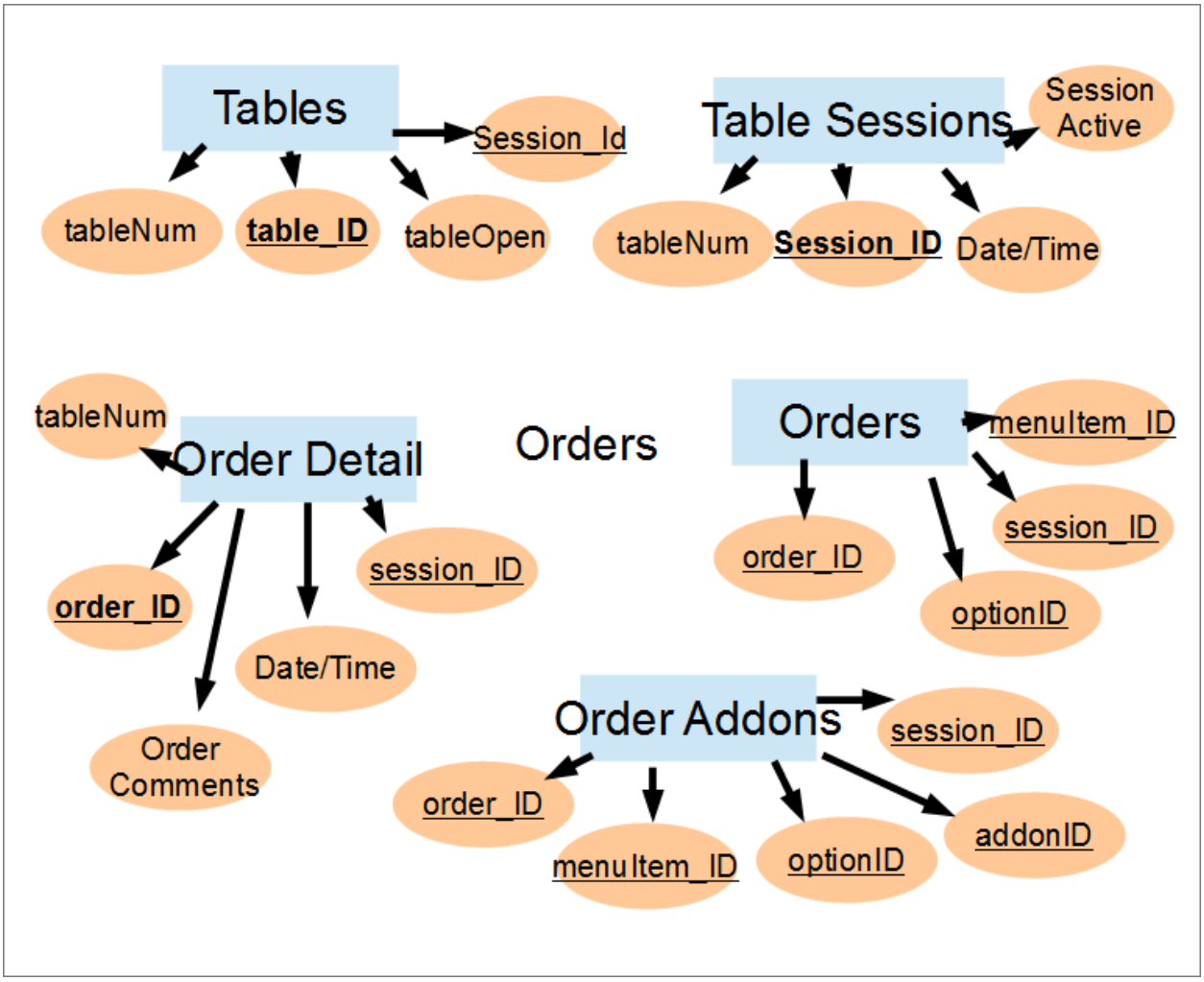


Figure 3.4.c Model of DB tables and attributes for orders

3.5. Chapter conclusion

The project required careful planning. The requirements needed consideration, the GUI design a lot of thought, and to have all the implementation and testing to be complete by the deadline needed a well structured development methodology to follow.

4 Implementation

This chapter describe how parts of the design and the application requirements were implemented.

Practices that assisted in optimizing Android performance[4.a] were followed such as write efficient code, don't do work that you don't need to do and don't allocate memory if you can avoid it. There were other tips followed like use static final for constants, avoid internal getters/setters, use enhanced for loop syntax and to avoid using floating point values.

4.1. Menu Implementation

One of the android widget tools which helped to achieve a good mechanism to view and navigate the different screen sections was a tool called a screen slider. It allows the users to easily navigate by using a sliding gesture to view the different section lists of menu items.
The appropriate menu items for each section are loaded from the database using SQLite and displayed in the list using a Cursor adapter in Java. The list of menu items are set in a scrollview layout in the XML, which allows the menu items to be scrolled if they all don't fit on the visible screen display.

4.2. Menu Item wizard implementation

A wizard format was created for the manager to enter information about the menu item using an Activity. An Activity is an application component that provides a screen with which users can interact in order to do something.The activity created and and handled the lifecycle of all the different screen fragments of the wizard.

By dividing an activity into fragments, you are able to modify the activity's appearance at runtime and preserve the changes in a back stack that's managed by the activity, such as when an activity is paused (partially obscured) or resumed during state changes[4.2.a].
There were buttons placed on the fragments using XML. Java listener methods were used to call certain methods when the corresponding button was pressed.
It was good practice to implement support libraries so the application could run and be compatible on different devices which have an older API level[4.2.b].

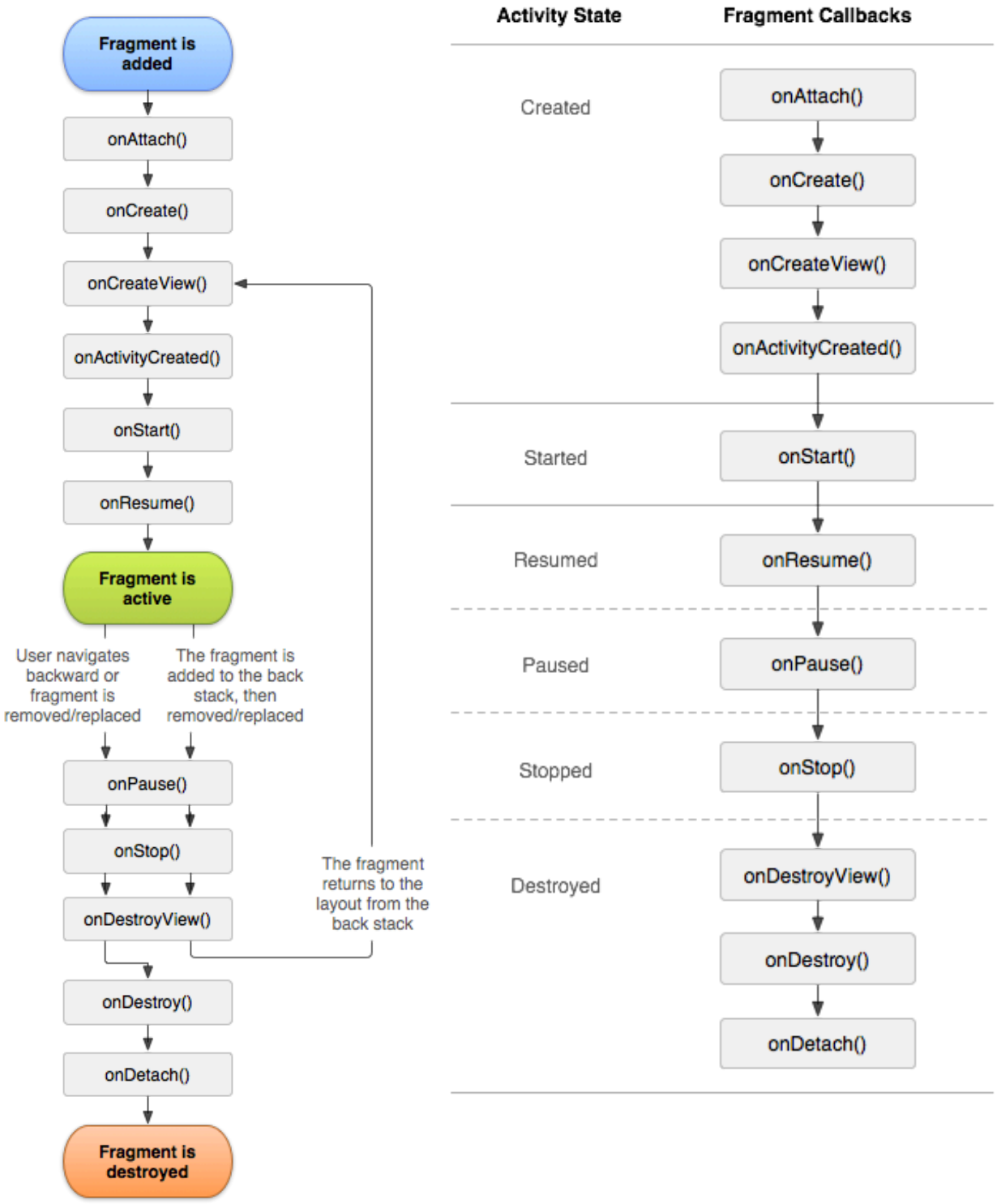


Figure 4.2.a Diagram of Android Activity Lifecycle [4.2.a]

4.3. Menu Item Name, Details and Ingredients

On the 1st step of the wizard the user must enter a name for the item, this was collected using a widget in the XML called an EditText box, this allows a user to enter text on the device. A warning message is made to appear if the user tries to continue without entering a name, this message uses an Android feature called Toasts. Toasts display a short custom message to the user.
The user can also enter a description of the item by clicking the current description text, this opens a component called a Dialog box which temporarily takes up most of the screen so that the user can enter more text easily.

The user can also add ingredients to the menu item. This can be accomplished by searching and clicking on an ingredient name to add it or by creating an ingredient. They can also remove ingredients easily by clicking a delete(X) icon, this uses an Image button listener. This information gets saved to a temp DB table to be used later on in the wizard or for filling in screen data when the user returns to the fragment.

Temp tables were created for most parts of the wizard for storing temporary data. When the user is in the wizard the temp tables contain all the data to be loaded in on each screen. This allow an update implementation which copies the real data back to the temp tables. This can then be changed and manipulated until the user confirms the update which then overwrites the real data with the new values from the temp table. The temp values are discarded when the wizard is exited.

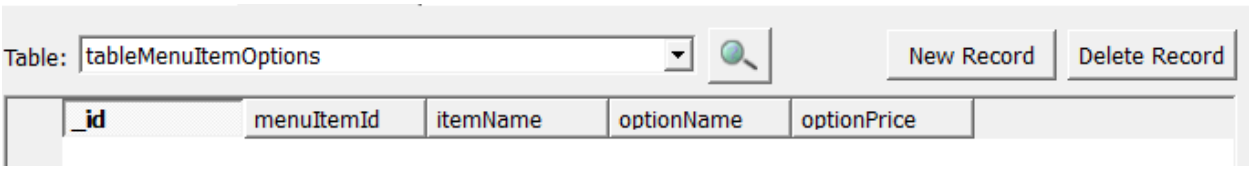


Figure 4.3.a Permanent/Real Table Example

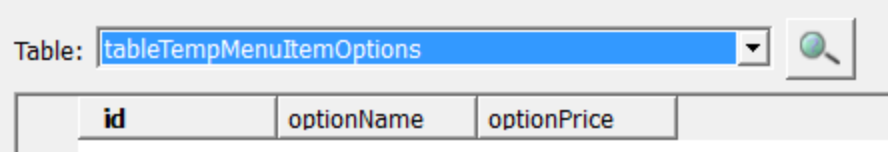


Figure 4.3.b Temporary Table Example

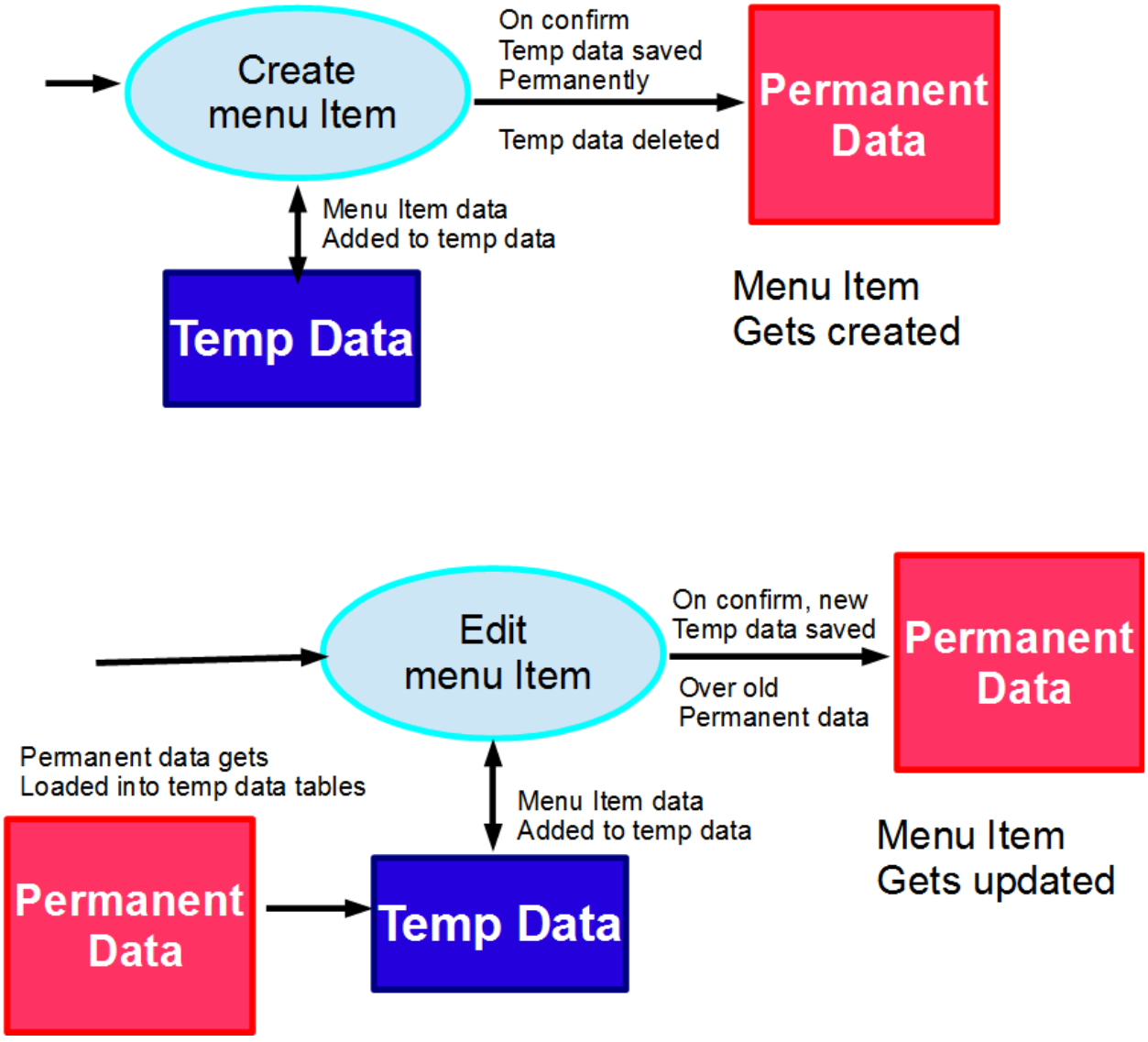


Figure 4.3.c Model of how menu item data from the the database is created and updated

4.4. Options and Addons

The 2nd wizard step is where the application gets data about the option names and price, a tool called a TextWatcher listens for input changes in an EditText Box, methods were written to automatically format the input into currency, showing a fixed symbol and decimal point place.
A good feature of the currency methods is that they are able to display the correct currency formatting based on the devices current country and locale settings.
This allows the application to be used in many countries such as Japan which have 0 decimal places or Tunisia which uses 3 digits in their currency[4.4.a].
The 3rd step gets the addon name and ingredient.

The next screens are where the user can enter information about the ingredient amounts for the different options and addon ingredient amounts and prices.
An Expandable List XML component was used by creating a custom Expandable List Adapter in the Java code.
An expandable list is a list where each element in the list (called the parents) can contain a list(called the children).
In the options ingredients amount screen, the parents were the menu Item option names and the children of each parent were the menu item ingredients with an EditText box to enter the ingredient amounts.

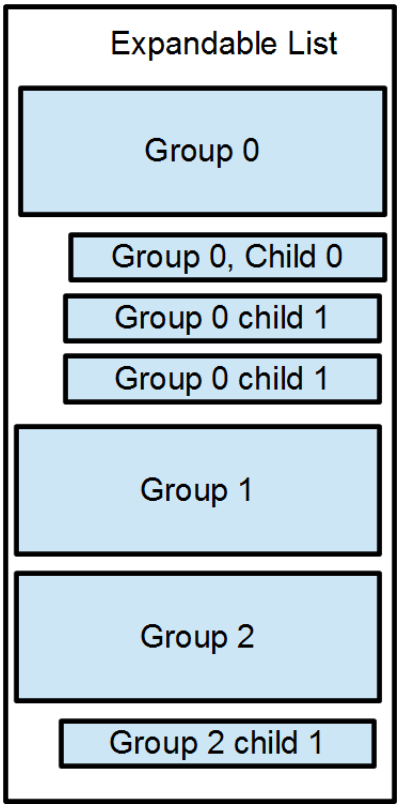


Figure 4.4.a Model of Expandable List

An optimizing feature of Android is called recycling views, if a list has many elements, say a 1000, the list would have to create them all, even though maybe only 10 could be displayed at a time, this would be very inefficient, this is where views get recycled.

If you enter a value and scroll so that text is no longer visible, when you scroll back up the value you entered can change because the list does not automatically save information.
A solution was implemented using view holders and double arrays to load and store the ingredient values to and from the list dynamically.

4.5. Review/Menu Screen

The review and menu screens are very similar except that data for the review screen is loaded from the temp tables and data for the menu screen is loaded from the permanent tables.
The options use radio group buttons and the addon buttons use check buttons implemented in XML and Java.

The input changes affect the addon prices, total price and nutrition values so everytime a input button is changed all these values have to be recalculated and updated.
Methods were implemented to listen for input changes so screen could dynamically update itself to reflect any input changes. The calculation methods could have been parallelised and calculated using threads if a performance boost was needed but it wasn't deemed necessary.

4.6. Ingredients implementation

The ingredient implementation was quite complex so it is split into different parts.

4.6.1. Ingredient search

A search box was implemented which showed a list of ingredients that begin with what the user queried. It used a Fast Text Search table (FTS4 extension module) on the SQLite Ingredients table. A special virtual table was created with a built-in full-text index which permitted efficient search queries on the database. On a dataset of 517430 Documents, the query on a FTS4 table returned in approximately 0.03 seconds, versus 22.5 for querying an ordinary table [4.6.1.a].

Triggers had to be implemented to keep consistency between the real ingredients table and the virtual Ingredients Table. When a row is changed it will apply the same function on the virtual table. Using this FTS4 Table implementation ingredients were able to be searched much quicker, optimizing performance and increasing efficiency.

```
private static String TRIGGER_AFTER_INSERT = "CREATE TRIGGER t2_ai "
        + " AFTER INSERT " + " ON " + DATABASE_TABLE_INGREDIENTS
        + " BEGIN " + " INSERT INTO " + DATABASE_FTS_VIRTUAL_TABLE_ING
        + "(docid, " + KEY_INGNAME + ", " + KEY_INGUNIT + ", "
        + KEY_INGCOMPLEX + ")"
        + " VALUES(new.rowid, new." + KEY_INGNAME + ", new." + KEY_INGUNIT
        + ", new." + KEY_INGCOMPLEX
        + ");" + " END;";
```

Figure 4.6.1.a Screenshot of one of the virtual Ingredient table trigger methods

4.6.2. Raw and complex Ingredient creation

To implement the ability to create raw and complex ingredients a wizard format was created where the user enters information about the ingredient at each step.

To create a raw ingredient it follows these steps. each step opens a new Dialog box.

1) Enter an ingredient Name

2) Choose the unit(g or ml)

3) Enter stock level information

4) Enter nutritional and dietary information

After confirmation the data is collected and the ingredient is stored in the Ingredients Table and copied to the FTS Ingredient table via a trigger.

For a complex ingredient the steps are different. Step 1 & 2 are the same but step 3 takes you to another screen with tools to create the complex ingredient. This wizard allows the user to add raw ingredients that already exist or create raw ingredients which get added to the complex ingredient when they are created.

One big implementation issue was how to specify how much of the raw ingredients, make up the complex ingredient. For example if an order is placed for x amount of a complex ingredient, how much of each raw ingredient needs to be deducted from the stock levels?
The solution implemented was to use a ratio function to calculate the amount of raw ingredients to deduct from the complex amount.

The user specifies what amounts of the raw ingredients make up the complex amount.
The values are calculated to find the ratio amounts of each raw ing which makes up 1g/ml of the complex ingredient and is stored in the database.
Then whenether x amount of the complex ingredient is ordered you multiply by the raw ing ratio from the database, to find the raw ing amount to deduct (Figure 4.5.2.a).

A big design decision made was not allowing complex ingredients to be created when making a complex ingredients. The reason for this is because it became apparent that the application might run into recursion problems, which could ultimately slow the system down or run out of memory in extreme cases. I also thought if ingredient creation goes down many levels it would get complicated and confusing for the user. By only allowing one level its keeps things relatively simple, and accessible when creating complex ingredients.

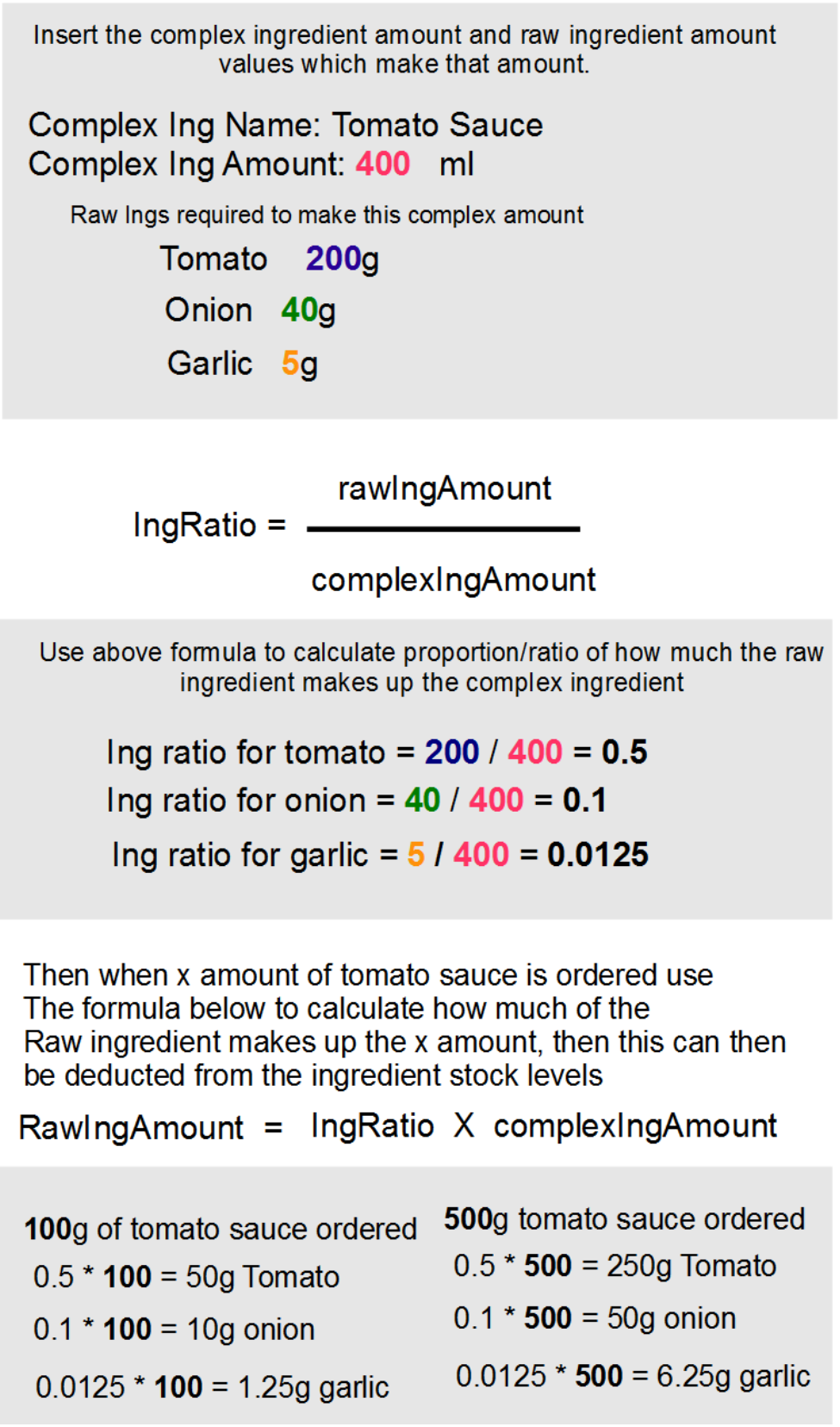


Figure 4.5.2.a Explanation of how the raw ingredient amounts are deducted from the complex ingredient amount.

4.5.3. Nutrition and Dietary calculations

A function to calculate nutritional information from ingredient amounts was implemented. Some common ingredients can be preloaded into the database so the user doesn't have to insert all common ingredient information manually. A lot of information about ingredients can be found using this nutritional website[4.5.3.a]. To display this information in a meaningful way the food standards agency traffic light system was used. A class was created which used the daily guideline amounts table[4.5.3.b] below to display the nutrition amounts visually. The nutrition calculation works similarly to how the ingredient amounts are calculated(Figure 4.5.3.b). The calculation implements recursion when complex ingredients consist of complex ingredients.

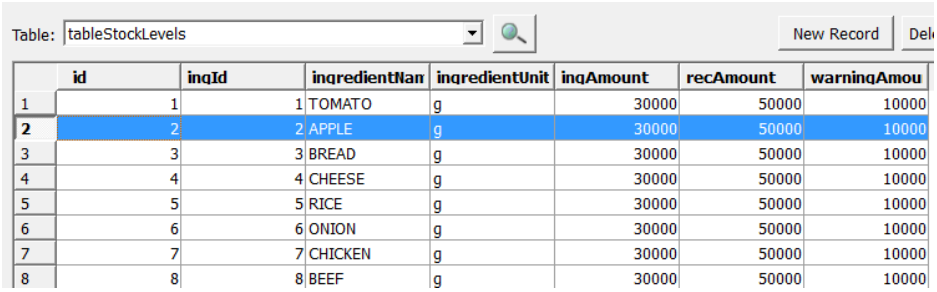
For food (per 100g):

	Green (low)	Amber (medium)	Red (high)
Fat	≤ 3.0g	> 3.0 to ≤ 20.0g	> 20.0g
Saturated	≤ 1.5g	> 1.5 to ≤ 5.0g	> 5.0g
Sugars	≤ 5.0g	> 5.0 to ≤ 12.5g	> 12.5g
Salt	≤ 0.30g	> 0.30 to ≤ 1.50g	> 1.50g

4.5.4. Stock Levels

The Stock Level screen uses a List to display the current stock level, recommended and warning amounts and displays a coloured notification circle. These values are read from the ingredient stock levels table. It uses the warning and recommended amounts to set its colour according to the stock level amount, it is green if above the recommended, red if below the warning amount and yellow if in-between the 2 values. This allows the manager to see what stocks are low much more efficiently.

Figure 4.5.3.a. Screenshot of Stock Levels table and elements



	id	ingId	ingredientNam	ingredientUnit	ingAmount	recAmount	warningAmou
1	1	1	TOMATO	g	30000	50000	10000
2	2	2	APPLE	g	30000	50000	10000
3	3	3	BREAD	g	30000	50000	10000
4	4	4	CHEESE	g	30000	50000	10000
5	5	5	RICE	g	30000	50000	10000
6	6	6	ONION	g	30000	50000	10000
7	7	7	CHICKEN	g	30000	50000	10000
8	8	8	BEEF	g	30000	50000	10000

Insert the nutritional information for a raw ingredient amount

Raw Ing Name: Tomato
Raw Ing Amount: 100 g

Insert Nutritional amounts for x amount of the raw ingredient

Calories 18
Sugar 2.6g
Salt 0.005g
Saturates 0g
Fat 0.2g

$$nutritionRatio = \frac{nutritionAmount}{rawIngAmount}$$

Use above formula to calculate proportion/ratio of how much the raw ingredient nutritional info makes up 1g of the raw ingredient

Nutrition ratio for calories = **18** / **100** = **0.18**
Nutrition ratio for sugar = **2.6** / **100** = **0.026**
Nutrition ratio for salt = **0.005** / **100** = **0.00005**
Nutrition ratio for saturates = **0** / **100** = **0**
Nutrition ratio for fat = **0.2** / **100** =**0.002**

Then when x amount of tomato is in the selection use
The formula below to calculate how much of the
Nutritional amounts makes up the x amount, then this can
then be totalled to compare with the guideline daily allowance

NutritionAmount = nutritionRatio X rawIngAmount

e.g. 300g of tomato in a order

0.18 * 3**00** = 54 Calories
0.026 * 3**00** = 7.8g sugar
0.00005 * 3**00** = 0.015g salt
0 * 3**00** = 0 g saturates
0.002 * 3**00** = 0.6g fat

Figure 4.5.3.b. Explanation of how the nutritional amounts are calculated for a raw ingredient.

4.6. Ordering implementation

A big feature implemented was the ability for customers and waiters to order for the same table simultaneously. A solution to achieve this was to create table sessions in the database. A table starts of in a closed state and can be opened, when a table is opened it is given a unique session_Id. A customer can open the table they are on and start a session, a waiter can do the same for any table in the restaurant.

For waiters, an XML layout called a GridView was used to display the tables, it was implemented in Java using a custom grid view adapter, it was made to display and control the tables and their current states, red meant closed and green meant open. If the table state was closed and the table was clicked then the table would change to open. This creates the session_Id as mentioned just above, next a function runs which invalidates all the GridViews then refreshes the screen so new updated state of the table is reflected on the screen.

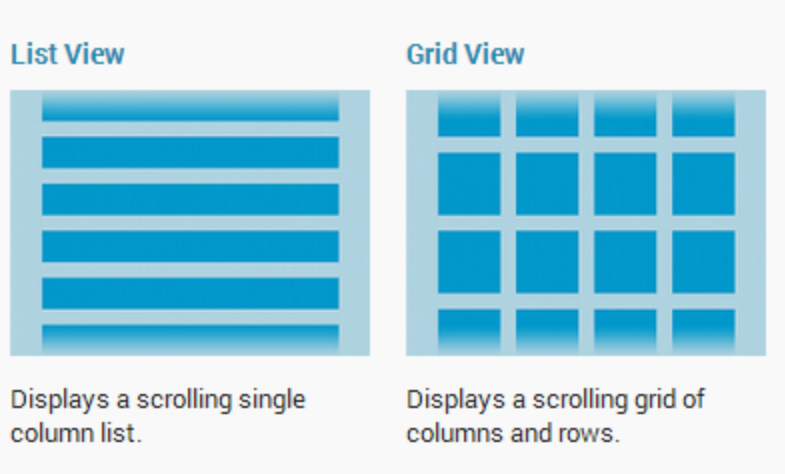


Figure 4.6.a Image of a List View and Grid View Layout [4.6.a]

The customer/waiter can then begin viewing the menu and add menu items to their order under this session. Clicking order on a menu item adds the order to the current order list by saving it to the temporary orders database table The user can continue adding menu items to their current order. The user can view their current order, it displays a list of the the menu items which have been added to the current order with their options, price, addons and the total cost amount of all the current orders. The addons are displayed in a horizontal scrolling TextView so that the unpredictability of the length of list does not affect the visibility or list layout dimensions.

When the order button is pressed on the current order screen this procedure is followed;

1. The stock levels of the ingredients of the orders gets deducted

2. A method transfers everything in the temp order table to the 'real' order tables. Information about the order such as table num, time/date and order comments are added

3. The orders get sent to the kitchen to be displayed.

4. A Toast message appears informing the order was successful

5. The User gets returned to the menu screen.

During a session, multiple orders can be placed. They each get given a unique order groupID. The customer/waiter can view the session order history, this screen displays all the orders and details split into groups using the groupID. The total price for all the combined session group orders are calculated using a method to sum up each order price in the session.
There is also a pay button, pressing it closes the current session and the table will go back to being closed. The table can then be opened again and new session be created.

To view the order statistics a function which writes the database to the SD Card was coded. The device has to allow permission to do this for security reasons. This had to be declared in the projects Manifesto. The ability to save the DB allows the use for backups and for closer inspection. The DB can then be viewed using a SQL View program, this can be used to view information or statistics about the orders and other tables.

4.7. Kitchen Implementation

The kitchen display screen shows a list of orders that have been sent to it. It uses a custom expandable list adapter where the parent list items gets information from the orderDetails table such as the order time, table num and order comments for that order group. The child views of the parents contain the menu item order details. The list is ordered by time and is set to auto refresh every 30 seconds so new orders of less priority appear at the bottom of the list.

4.8 Chapter Conclusion

The technical knowledge of all the languages and techniques of XML, Java, SQLite and the Android Environment was necessary in implementing the design. Most of the prioritised

requirements were able to be implemented successfully.

5 Results

This chapter presents the results of the design and implementation steps. It uses screenshots of the device running on an emulator with annotations to describe features and components..

5.1. Tables

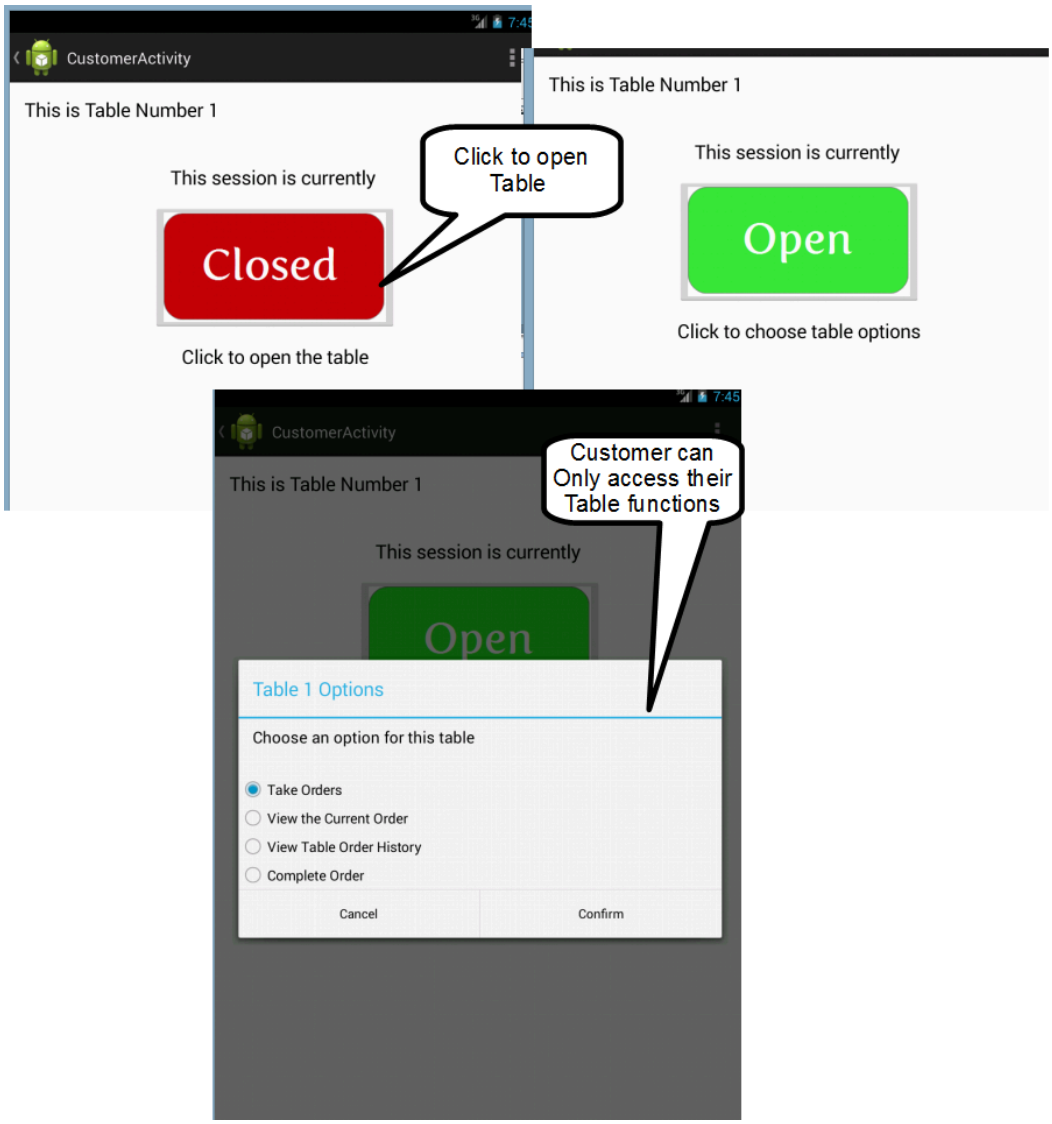


Figure 5.1.a Screenshot of Customer at table features

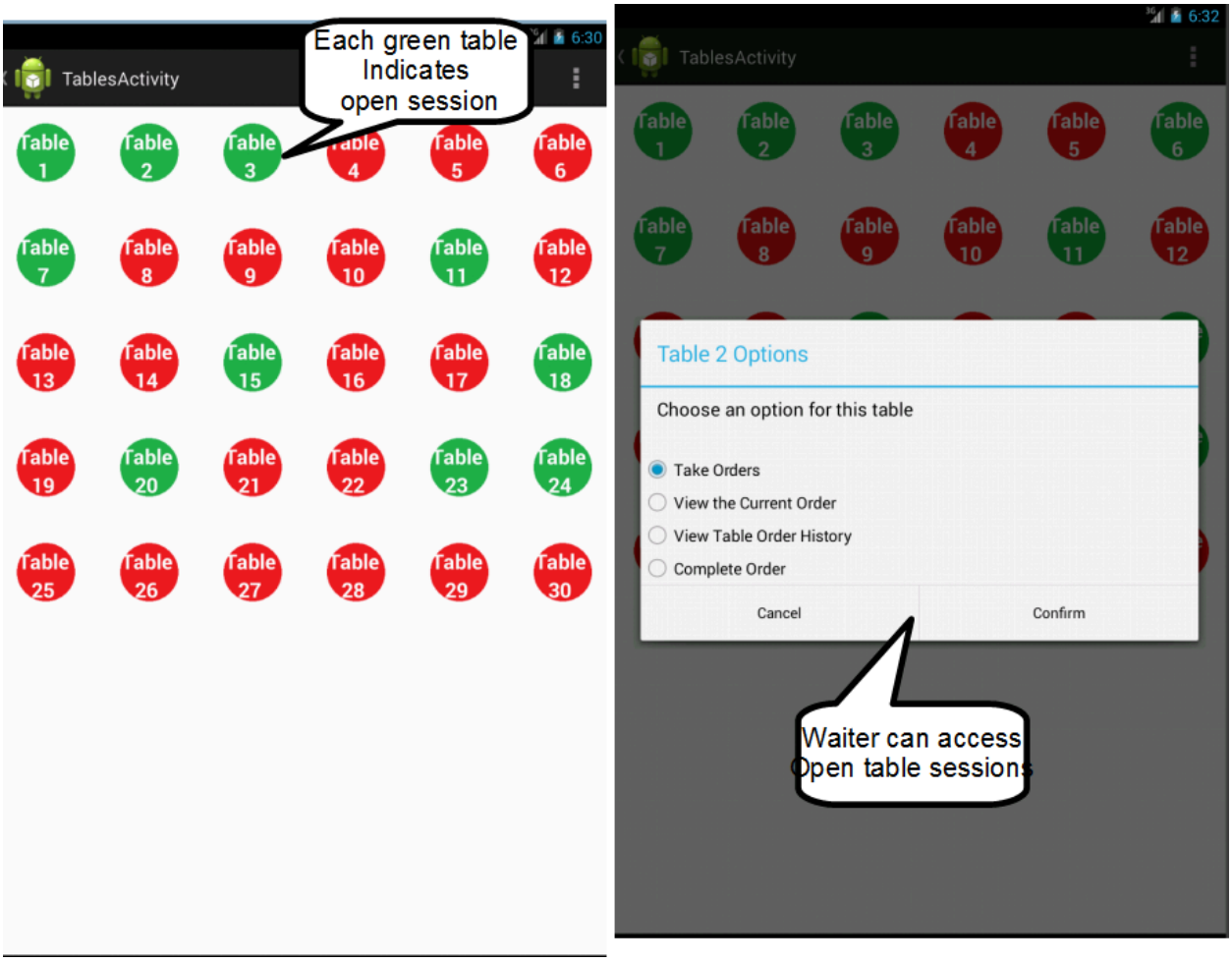


Figure 5.1.b Screenshot of Waiter at table features

5.2. Menu

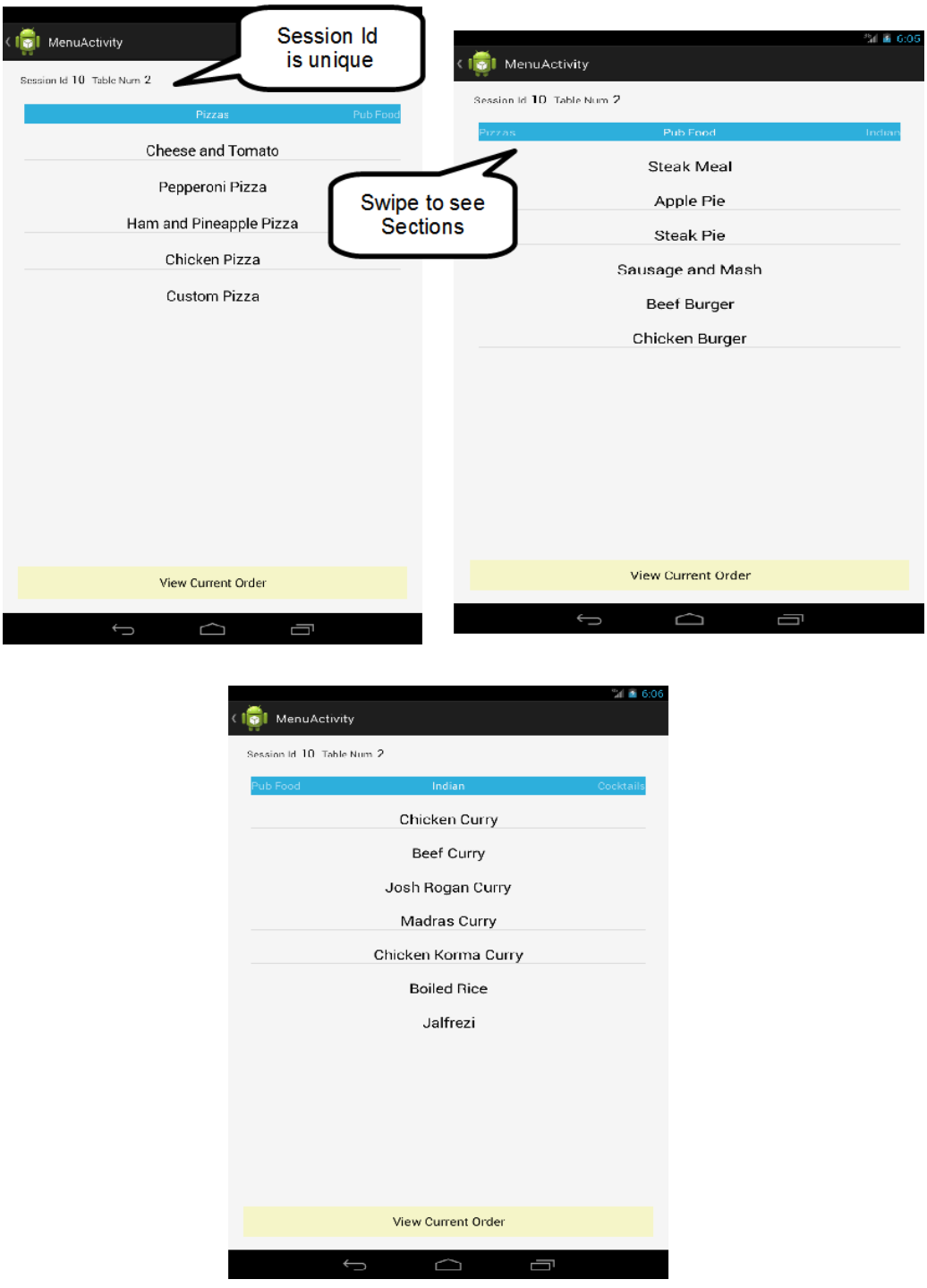


Figure 5.2.a Screenshot of Menu Sections

5.3. Menu Items

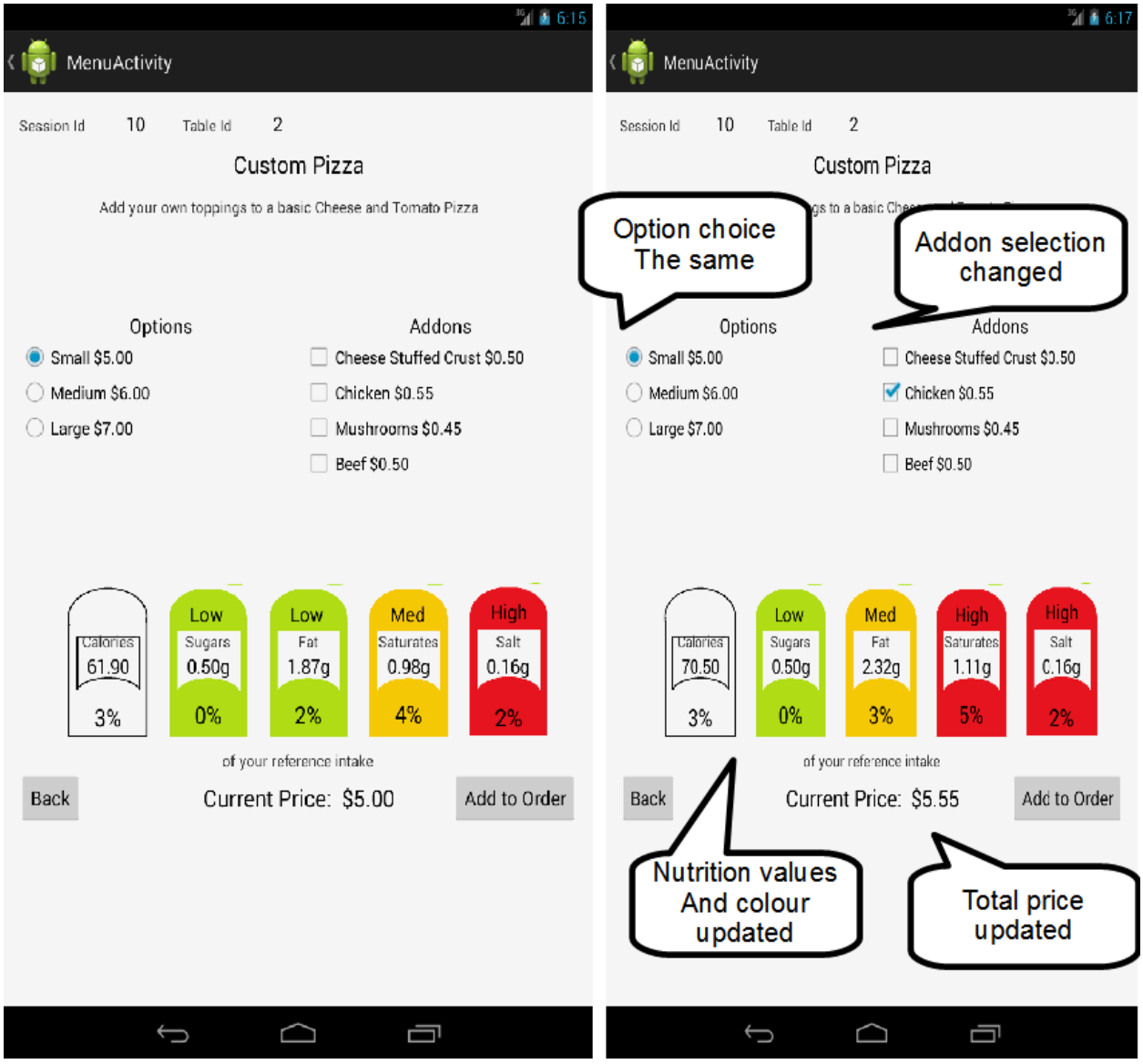


Figure 5.3.a Screenshot of Menu Item Customization

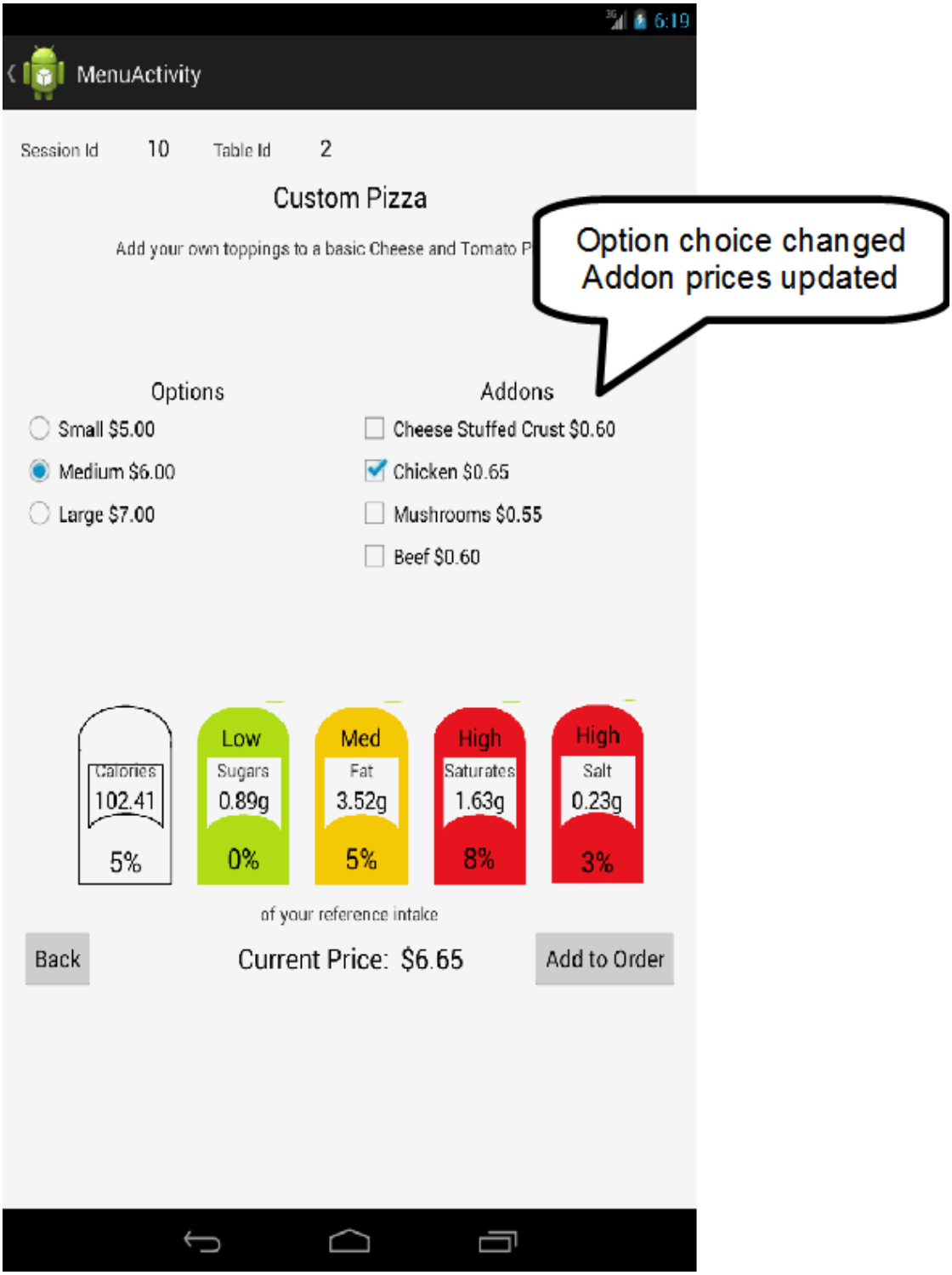


Figure 5.3.b Screenshot of Menu Item when option changed


MenuActivity
Session Id 10 Table Id 2
Steak Meal
Tasty Steak meal served with chips and peas
Menu item Can have no addons
Options
Addons
Rare $5.00
Medium $6.00
Well Done $7.00
Calories 130.45 6%
Low Sugars 0.72g 0%
Med Fat 8.06g 11%
High Saturates 3.61g 18%
Med Salt 0.03g 0%
of your reference intake
Back
Current Price: $5.00
Add to Order
MenuActivity
Session Id 10 Table Id 2
Cheese and Tomato
Margheritta
Options
Addons
Small $5.00
Medium $6.00
Large $7.00
Cheese Stuffed Crust $5.00
Calories 106.33 5%
Low Sugars 1.05g 1%
Med Fat 2.15g 3%
High Saturates 1.05g 5%
High Salt 0.24g 3%
of your reference intake
Back
Current Price: $5.00
Add to Order
Emulator locale is In in the US hence $ symbol
MenuActivity
Session Id 10 Table Id 2
Beef Burger
Beef Burger Meal, served with Chips
Options
Addons
Default $5.00
Cheese $0.50
Extra Chicken burger $1.00
Extra Beef burger $0.45
Cheese Stuffed Crust $0.50
Menu Items Must have 1 Default option
Calories 59.70 2%
Low Sugars 0.43g 0%
Low Fat 1.02g 1%
Med Saturates 0.38g 1%
Med Salt 0.08g 1%
of your reference intake
Back
Current Price: $5.00
Add to Order

Figure 5.3.c Screenshot of Menu Items examples

5.4. Orders

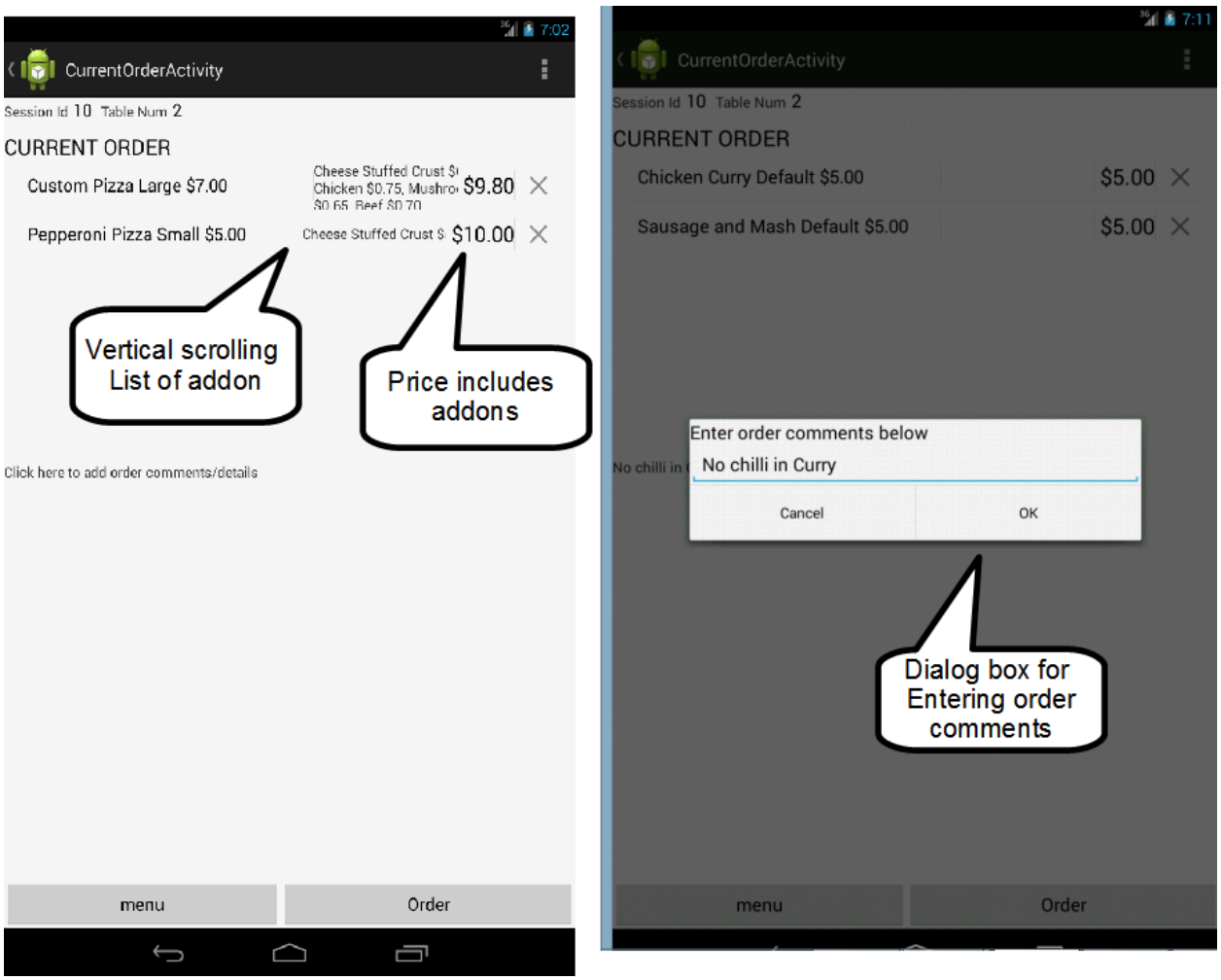


Figure 5.4.a Screenshot of Current Orders Screen

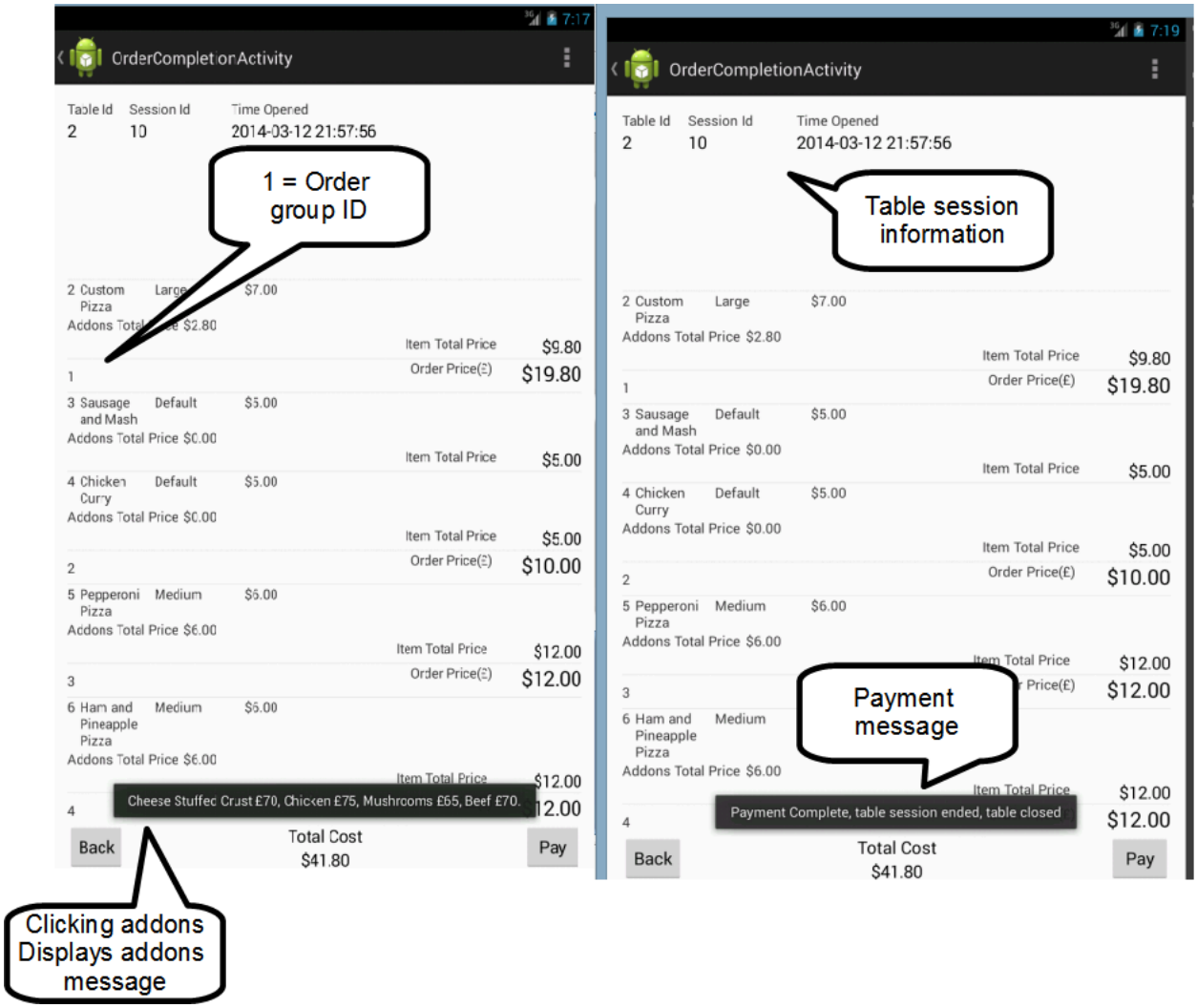


Figure 5.4.b Screenshot of Session Order History Screen

5.5. Ingredients

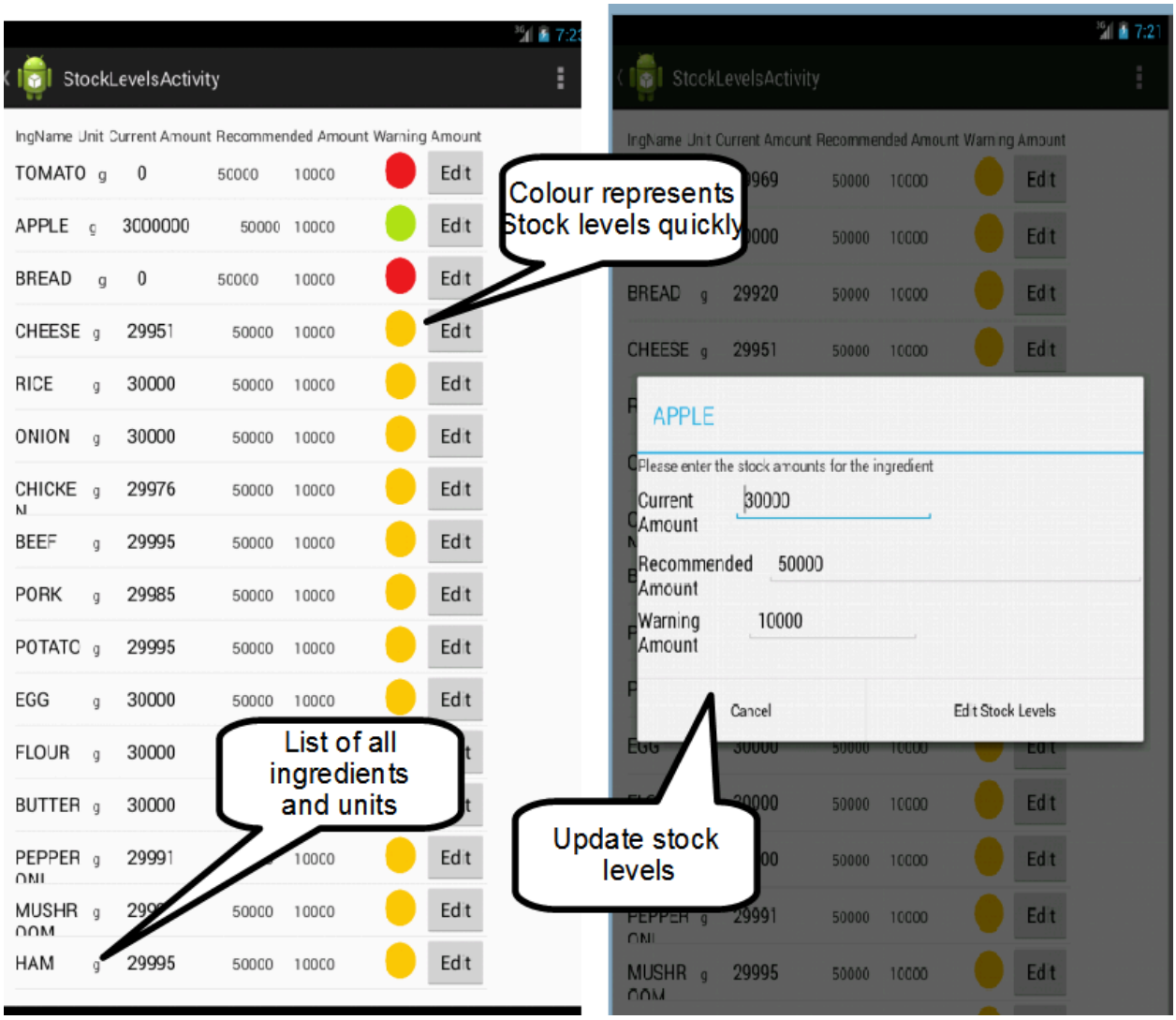


Figure 5.5.a Screenshot of Stock Levels Screen

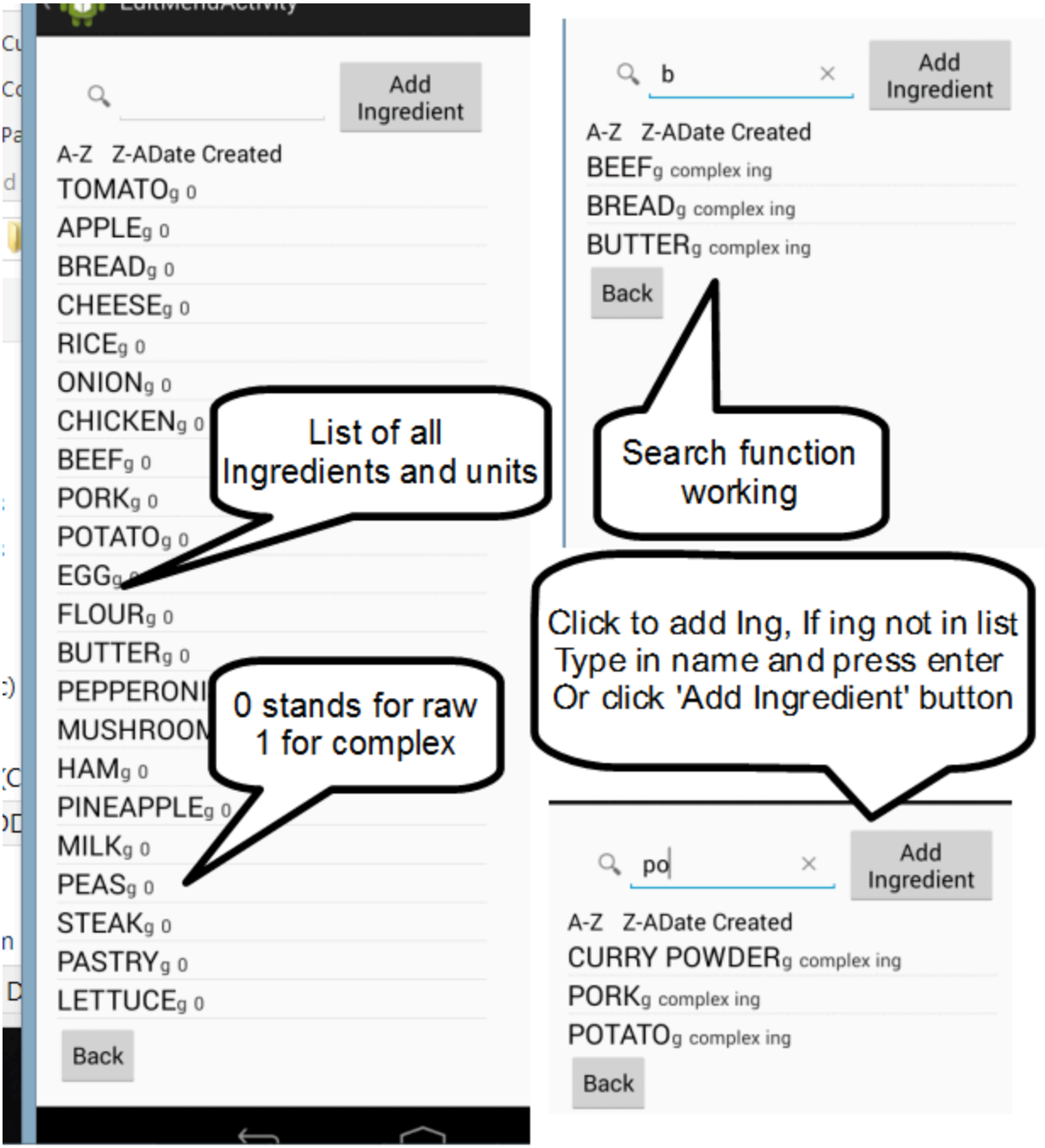


Figure 5.5.b Screenshot of Add/Create Ingredient screen

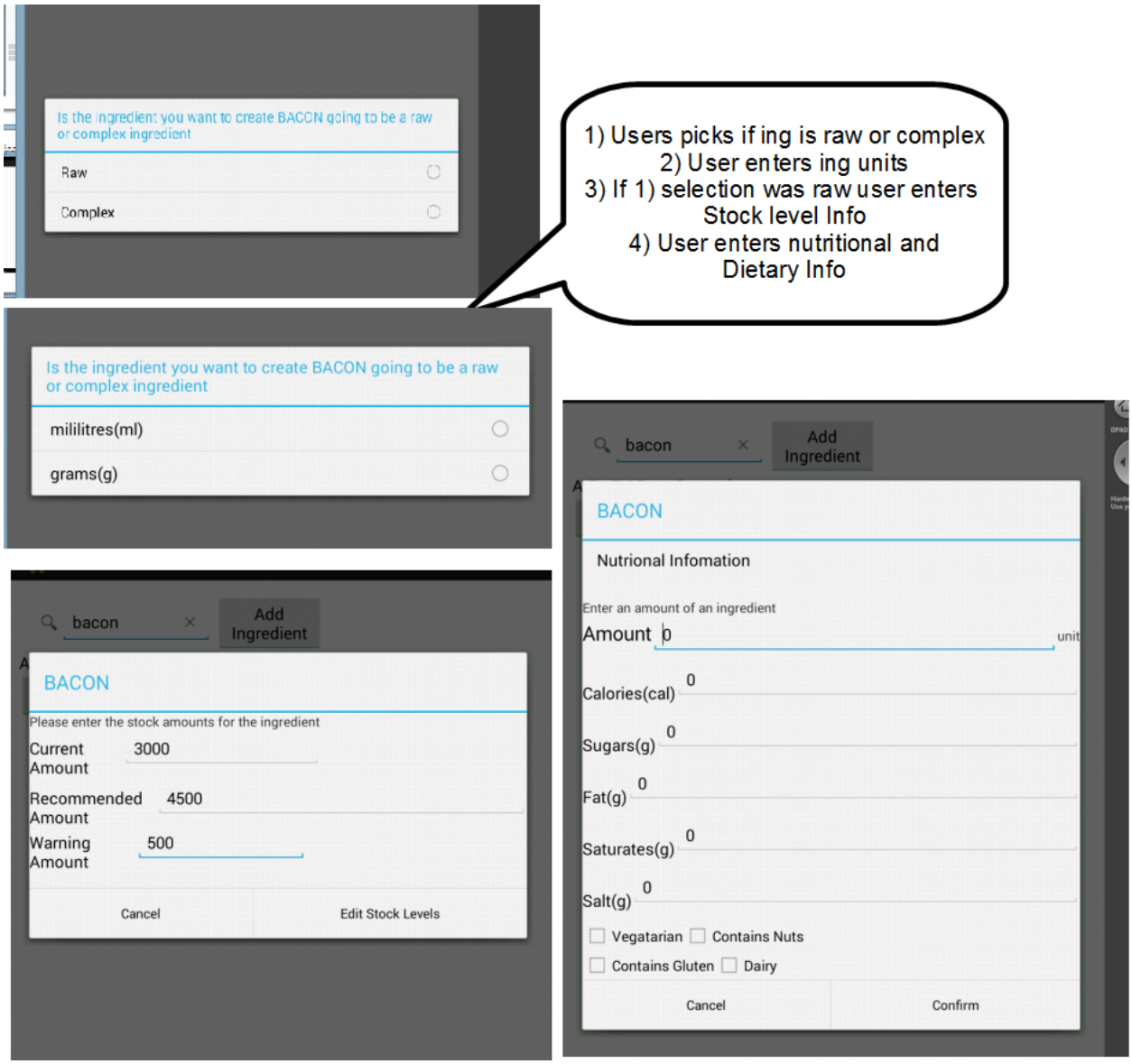


Figure 5.5.c Screenshot of Create Raw Ingredient Steps

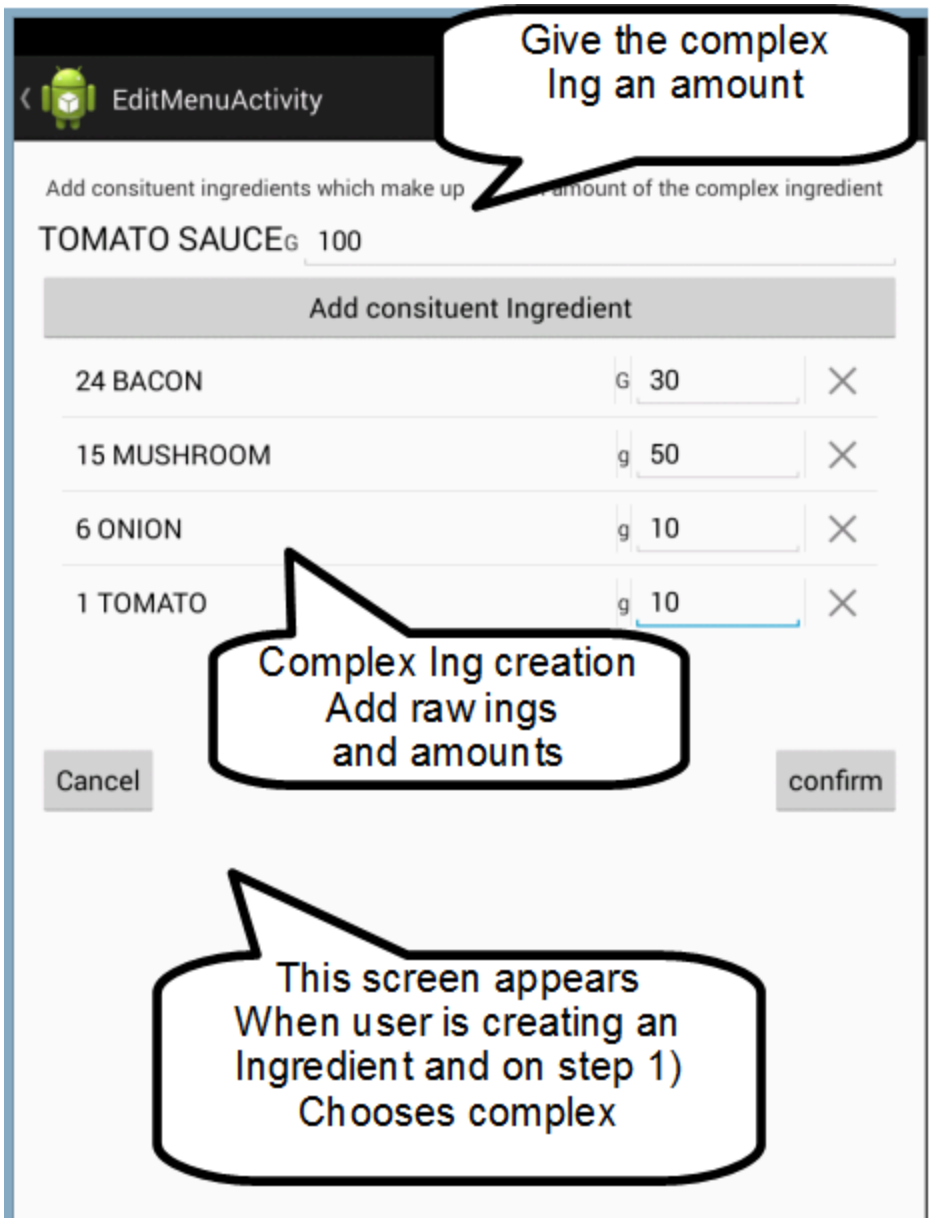


Figure 5.5.d Screenshot of Create complex Ingredient

5.6. Menu Item Creation/Update Wizard

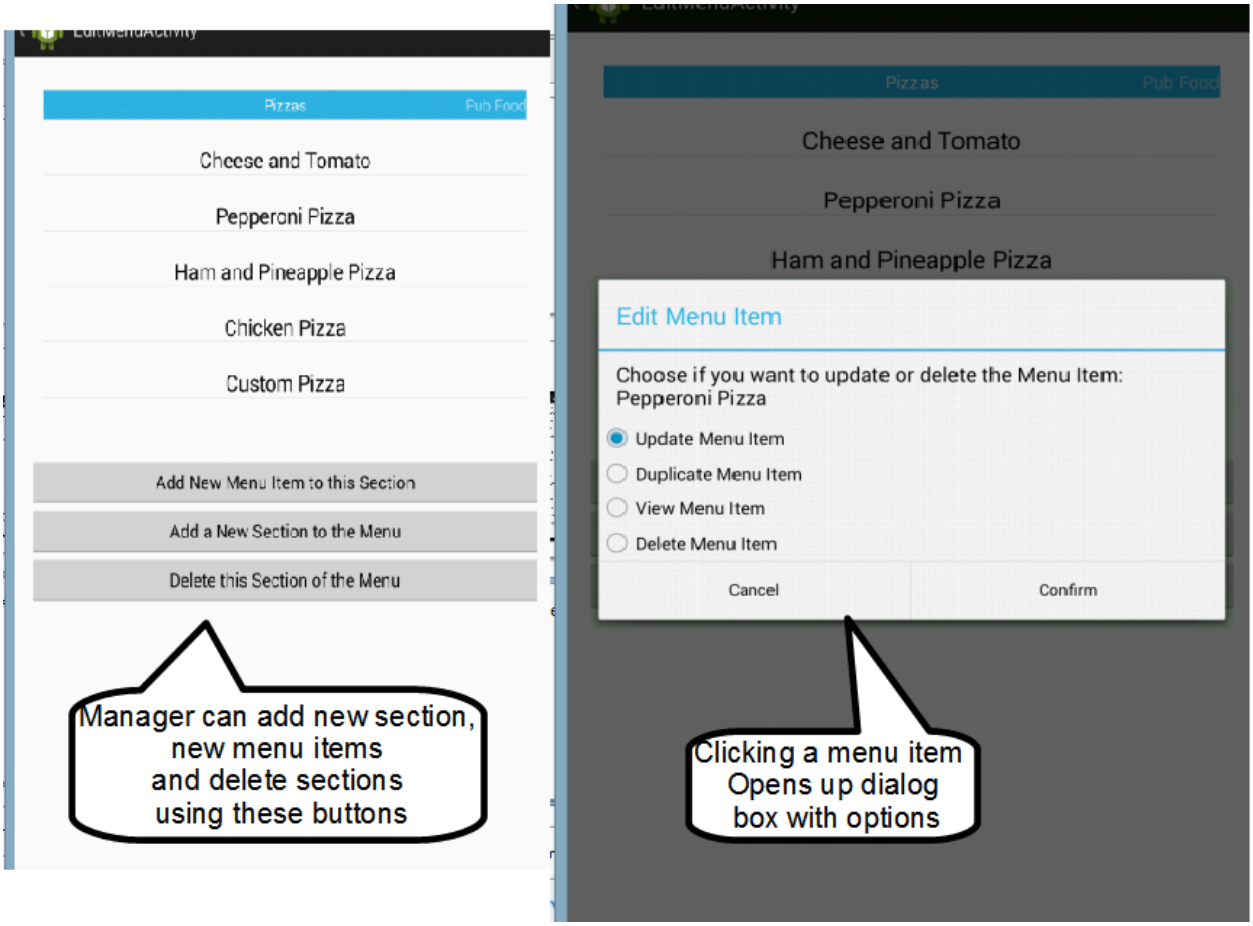


Figure 5.6.a Screenshot of Manager Menu and Menu Item tools

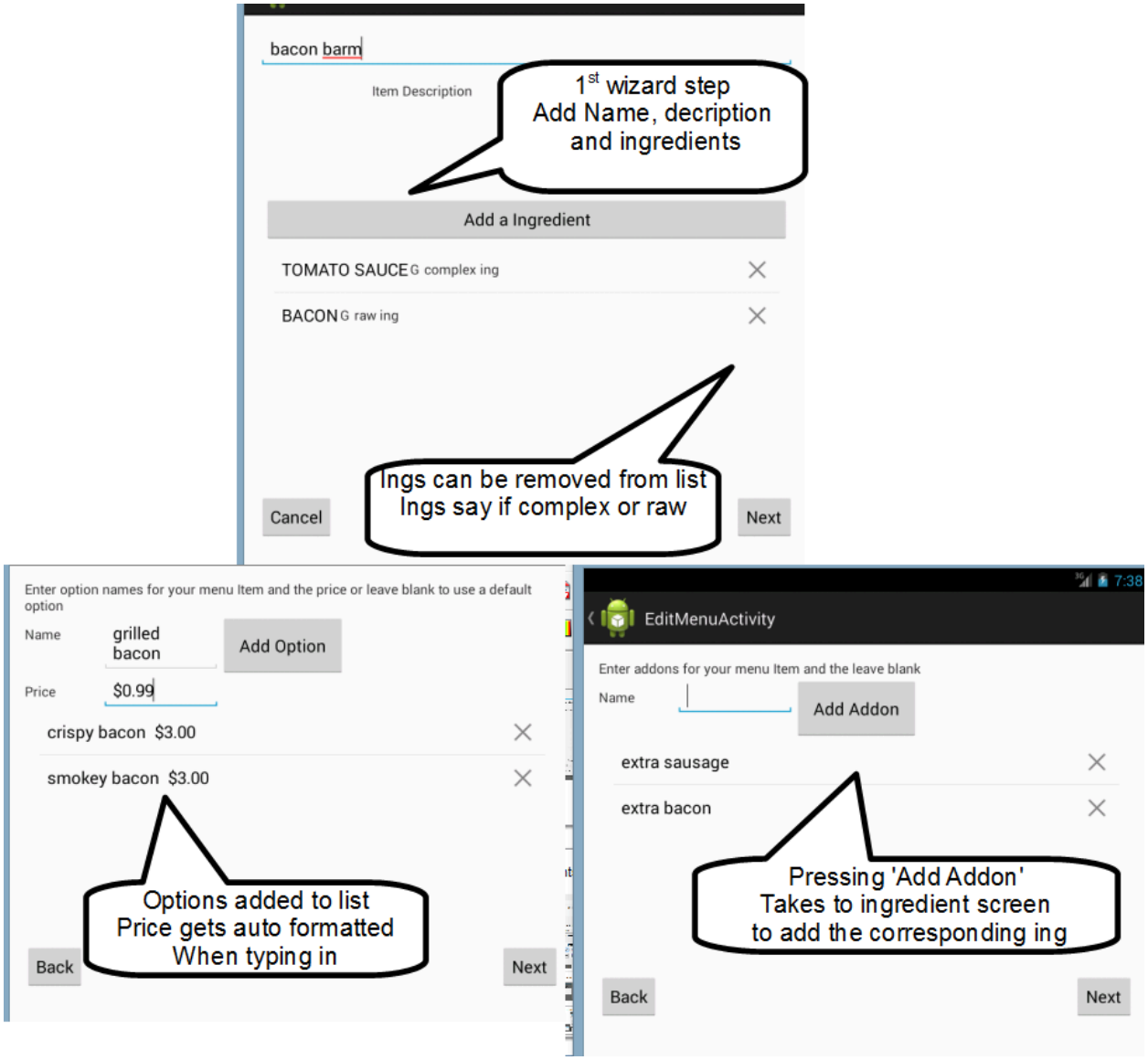


Figure 5.6.b Screenshot of Wizard steps 1,2 and 3

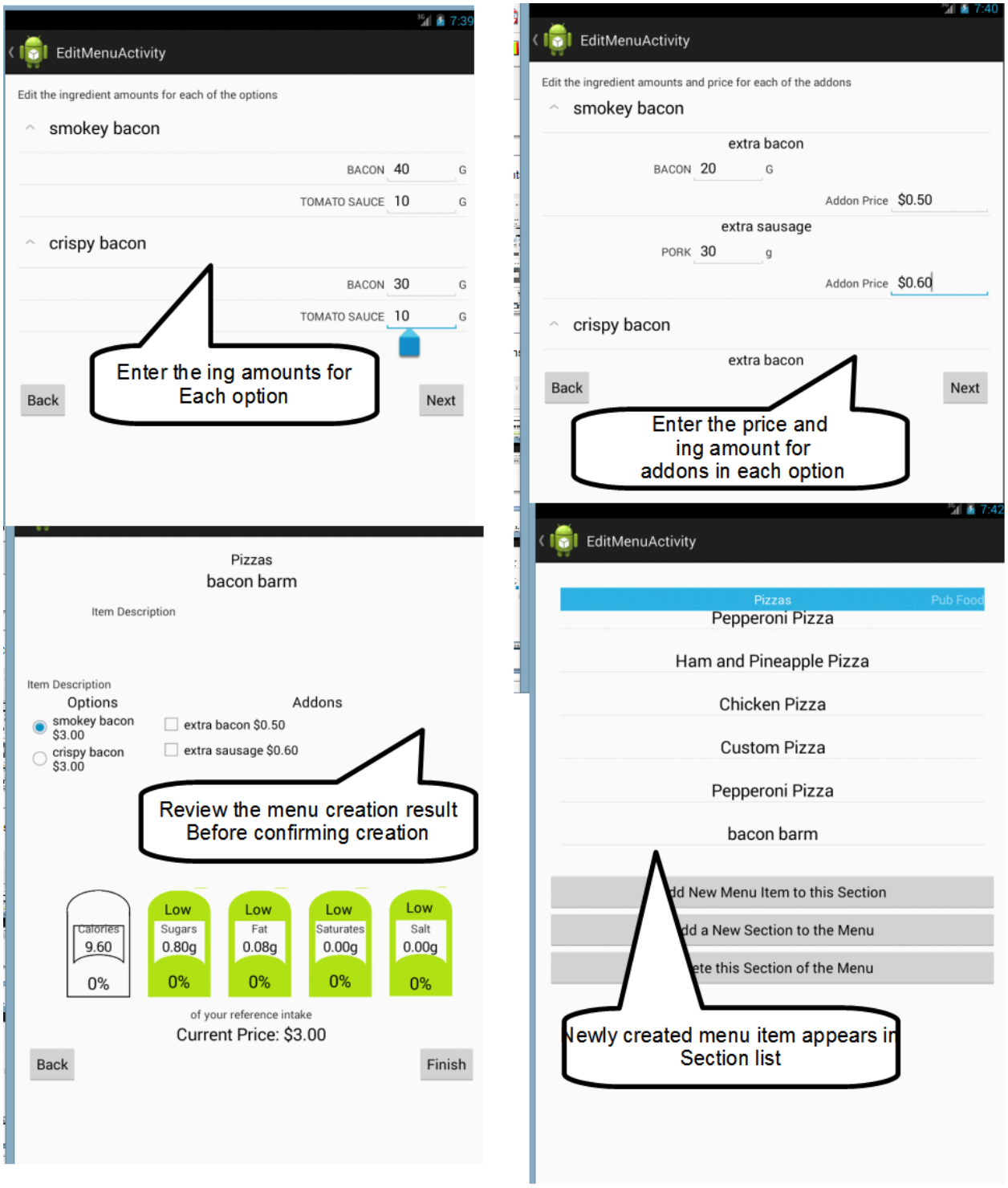


Figure 5.6.b Screenshot of Wizard steps 4,5 and 6 and the newly created menu Item

6 Testing and Evaluation

This chapter give details of how the system designed and implemented was tested and evaluated.

6.1. Testing

Testing was used to confirm if everything was working correctly, it was critical because it gave confidence that the application functions as supposed to. All software and hardware can have bugs and exhaustive testing helps to remove most of them. To prevent as many bugs and program failures as possible, steps were taken during and after implementation. Code was tested thoroughly under many conditions and variables to try and spot or eliminate the likelihood of bugs.

6.1.1. Testing the GUI Design

For the screen design, the testing approach used was to look at the emulator or real device screen and compare it with the designed paper prototypes to check if the layout components are displayed as expected. By comparing the screen view to the XML design view and paper prototypes it was possible to quickly recognise and get fast feedback if components were the correct size/dimensions and situated in the correct positions.

While manual testing has its drawbacks such as repetition and human error, for the GUI testing it was sufficient. The approach was to test each component of the GUI by hand such as the Buttons, EditTexts, Check boxes and Scroll Lists and assess if there any errors or faults occur. If there was an issue it then an attempt to discover why the bug/error occurred would take place and it would tried to be fixed. Doing this helped to learn and improve and prevent the likelihood of making the same mistakes again in the future. In the image below (Figure 6.1.1.a), the DDMS has a tool called the UI View Hierarchy viewer which was used to inspect and analyze the layout of screen elements and component properties.

The GUI could be tested automatically using Android's custom testing framework, it is based on the popular JUnit framework[6.1.1.a]. Unit tests can be written for cases to verify specific behaviour or layouts in the application and to check for consistency across different Android devices.

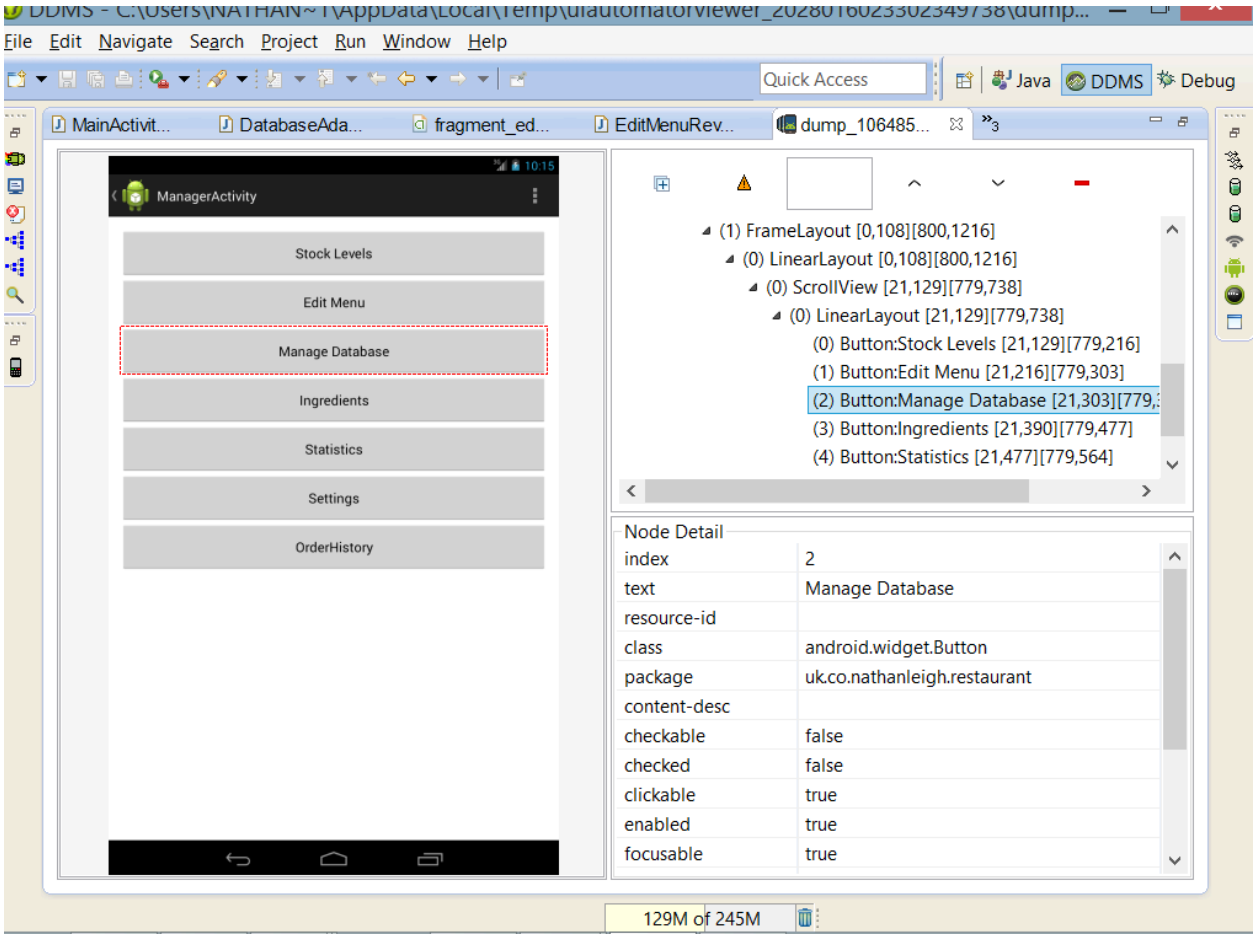


Figure 6.1.1.a Screenshot of Eclipse DDMS UI View Hierarchy viewer

Test cases were created to test parts of the layout but the author seemed that setting up many test classes, frameworks and fixtures for all the 50+ XML classes seemed less efficient than just testing manually. The test cases were used to verify and validate UI behaviour. This type of UI testing was called white-box testing. Here is an example of a test case used in the application which checked to see if a new screen is launched when a button is clicked, in this case the Stock Levels screen. The test cases provided feedback on whether the screen was launched correctly.

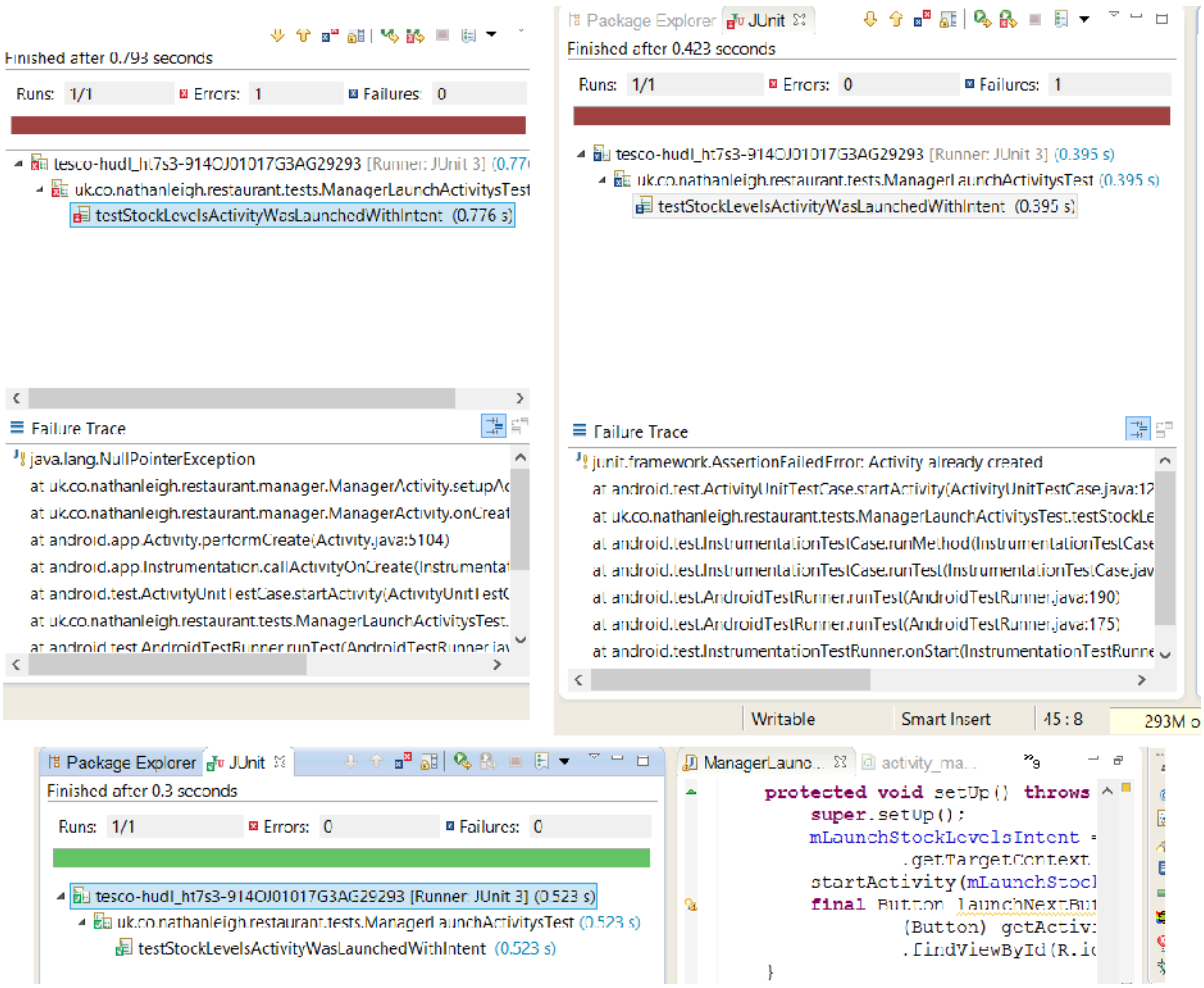


Figure 6.1.1.a Screenshot of Test cases used to test if a certain activity was launched successfully

6.1.2. Testing accessibility

The accessibility testing should have the following, high level goals[6.1.2.a]:

- Setup and use the application without sighted assistance
- All task workflows in the application can be easily navigated using directional controls and provide clear and appropriate feedback

The Application must follow and meet certain testing requirements[6.1.2.b] in order to ensure a minimum level of accessibility. The application should implement features to allow talkback from the screen to help navigation and exploration, the components should be certain sizes and the users should be able to explore by touch and audio feedback must always have a secondary feedback mechanism to support users who are deaf or hard of hearing among many other requirements.

Unfortunately the application didn't meet all the requirements, the TalkBack feature was not fully implemented and the Explore by Touch tools were not tested due to time constraints. Implementing accessibility wasn't as prioritised compared with implementing functionality, looking back this was probably not the best development strategy to use.

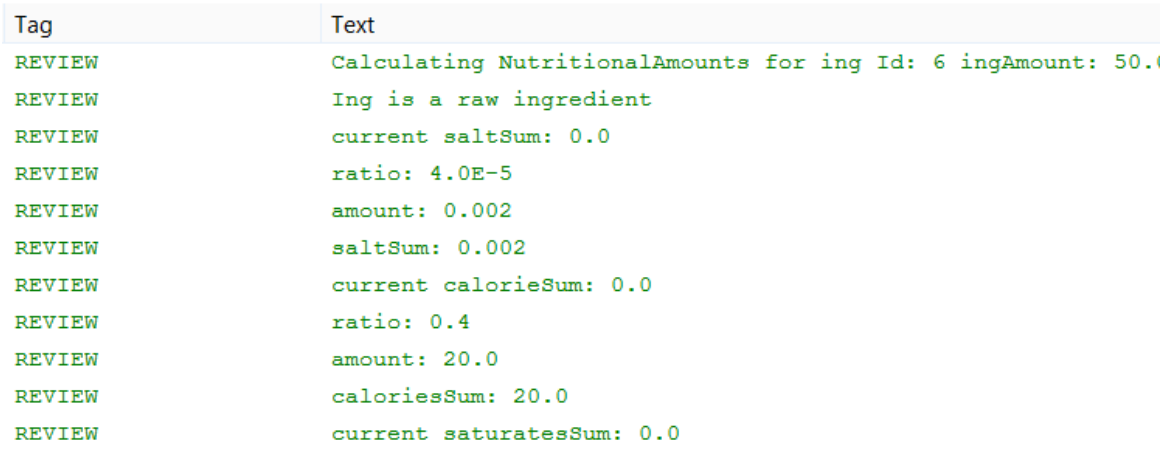
6.1.3. Functionality tests

The functionality of the application required testing, features such as the database, nutrition and stock level calculations needed to be checked to see if they behaved correctly. The functional requirements and stories required testing to see if they were successfully working.

To test the database, the implemented CRUD methods were tested manually for each table, if there was a problem with these methods there would normally be an SQL Error such as no such column exists. The harder errors to spot with the database was when the data being inserted was incorrect or in the wrong format.

This could happen with user input, to prevent this whenever there was user input the user is restricted in certain ways. Best user input practices were followed such as if digits only need to be entered then a digit only keyboard will appear or if input text can not be blank then a message notifying the user about this is made to appear.This helped prevent most erroneous input but there was a rather large security flaw with user input. The database is susceptible to SQL injection, a user could enter SQL code which could alter the database in theory. Fixing this security flaw was not a priority.

To test whether the nutrition and stock deduct calculations were working as expected, tests were made on paper and excel of the expected values, these could then be compared with the values in the application for correctness. System logs were placed to track variables and values and the database was inspected to check elements. The ingredient values were from a trusted source and the implementation methods were designed and coded carefully, the results of the functions passed with confidence.



Tag	Text
REVIEW	Calculating NutritionalAmounts for ing Id: 6 ingAmount: 50.(
REVIEW	Ing is a raw ingredient
REVIEW	current saltSum: 0.0
REVIEW	ratio: 4.0E-5
REVIEW	amount: 0.002
REVIEW	saltSum: 0.002
REVIEW	current calorieSum: 0.0
REVIEW	ratio: 0.4
REVIEW	amount: 20.0
REVIEW	caloriesSum: 20.0
REVIEW	current saturatesSum: 0.0

Figure 6.1.3.a Image of Log Code when calculating Nutrition amount Sums

6.2. Evaluation

To measure the value of the application it was tested by the author, this provided information about how well the application runs, its ease of use and its ability to be used in the real world.

6.2.1. Non-Functionality Performance

The applications responsiveness and run speed was tested manually. All components of the application was tested on a Tesco Hudl Device running Android version 4.2.2.
The results were impressive, the layout displayed correctly, run time was fast, screen components and navigation was smooth, quick and responsive. The database is able to be backed up to an SD card for redundancy. There are some security issues due to time constraints, currently any user of the application can access any part of it.

6.2.2. Usability in a Restaurant

To evaluate the application it was tested to see how well it could replicate a restaurant menu, the applications performance could then be evaluated for its use in a real restaurant environment.

Nando's Menu [6.2.2.a]
The application has ability to divide everything into sections like the Nando's menu has done. The extra's component can be replicated as Addons. The options for menu items can represent the different sauces used. This evaluation found that the menu can be replicated on the application. It offers advantages that it can calculate nutrition which the paper menu does not offer. The layout design of the menu on the device would not be as attractive as the paper menu though.

Wetherspoons Menu [6.2.2.b]
The menu items and sides could all be sucesfully inputted into the application. There wasn't a way to have the meal + drink deals implemented without having multiple options, also the options list could get very long if there are lots of drink options available. The disadvantage of the application is that it can not implement the offers and deals without having many options.

McDonalds [6.2.2.c]
Each individual menu Item and their options can be inputted.
E.g. Fries - Options(small, medium, large). Coke - Options(small, medium, large), Addons (ice).
The application struggles to replicate when menu items get grouped. The problem occurs if there are 2 items that have options.
E.g. Custom Burger meal = beef burger + fries(option) + drink(option),
The amount of options can become exponential.

Overall I believe the application is able to replicate most Menu's individual menu items very well, unless menu items with multiple options get grouped together in a meal.
If more time was available these issues would have been fixed. I think the automatic Nutrition calculator is very useful and informative for menu items that paper menus do not have.

6.2.3. Usability in Other Environments

The framework of the application could be adapted for more possibilities other than restaurants. Shops could use the application to extend their market audience to phones/tablets.
The application could be extended to have a account/payment function and then items could be ordered online using the application. Retail items could be represented on the application such as games, books, TV's, grocery, guitars and many more. The stock level feature would be useful in this instance as well.

The nutritional calculation feature of the application could adapted to be used by anyone, maybe home cooking people want to control their daily nutrition, they could use the application as if they were the manager and input their meals and find instant feedback. Parents or gym enthusiasts may find it very useful.
The customisability aspect of the application makes this versatility possible.

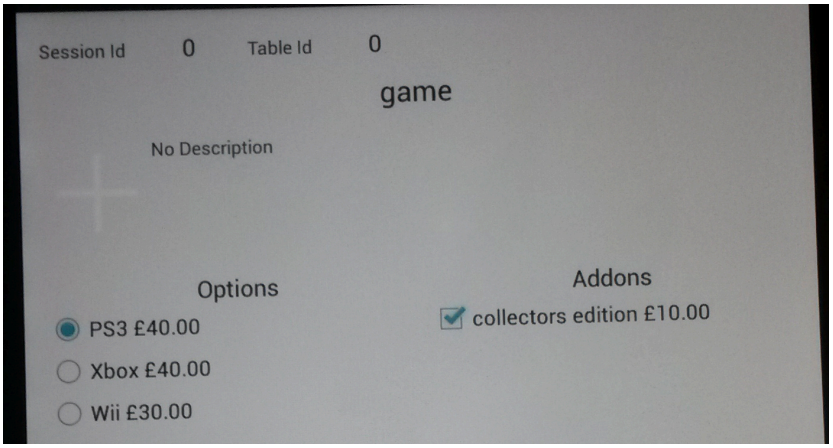


Figure 6.2.3.a Image of application used for ordering a game

6.3. Chapter Conclusion

The application has been tested mostly manually, the application features appear to work correctly and successfully. The evaluation provided feedback on the application, it found that It is able to replicate menu items to a reasonable degree but there is room for improvement. The evaluation also found the application may not just be restricted to restaurant environments.

7 Conclusions

This chapter explains the results of the project and what was achieved. It reflects on what was learnt and what could have been done differently.

7.1. What I have achieved

The application successfully achieved many of the original objectives and with a little more work could be adapted to be used in restaurants, bars, produce and other environments. The nutrition calculator is a very useful feature which offers improvements over using a paper menu for the customer. The stock levels automation saves time and can be used to easily recognise when stock is low.

The order history allows managers to assess what is popular or unpopular on certain days or time of the year. I feel the application achieved good usability and accessibility for most users yet more could have been done for disabled users. I believe users would enjoy using the application in real life and the advantages it offers.

7.2. What I have learnt

During this project I have learnt and improved many skills. My logical problem solving improved, my ability in finding design and programming solutions got better and my Java proficiency is of a higher quality. I learnt a great deal about designing good GUI's and user experiences. My ability using XML and SQL languages also increased.

My database design and programming skills were refined by practicing them throughout the project as well. A useful new technology I got experience in was the Android mobile platform. I learnt software developing skills, more familiarity with Agile development and and time management experience which will be very helpful in the future.

7.3. What I would have liked to have done differently

I spent plenty of time writing code which then had to be changed during development. This was a result of a lack of Android programming expertise and the projects requirements and design constantly changing, thankfully, the agile approach allowed me to adapt quickly to these issues.

I tended to mostly design solutions for problems I faced in the present, not looking at the big picture or into the future as much as I should have, I didn't spend enough time fully fleshing out the database design. This caused problems in implementation. I think this came from lack of practise in complex database design so hopefully in the future I can learn from this experience.

Features I would have liked to include was to implement the wireless communication between devices over wifi, this was due to a lack of available hardware. I also would have liked the application to support offers on menu items such as 2 for 1 or different items are discounted when bought together. Being able to have groups of options would have been a useful feature, this was discovered from the evaluation performed in section 6.2.2.

Another feature I would like to include is user accounts, where users can log in to the device for any restaurant chain. This could then be used to track data about the user and reward loyalty points, email offers, get customer statistics/info, offer recommendations and to recieve feedback. A login system could provide security and allow the system to follow a customer's session orders. This could be used to post the overall nutrition/cost for their meal and even notify if they are over the driving alcohol limit, which could save lives.

There are many other features I would have liked to include but was unable to due to hardware and time constraints.

7.4. Closing remarks

In reflection this project has been challenging, interesting and rewarding. Being able to learn new skills, technologies and development techniques will be very useful in other real world future projects.

Thankyou for reading this report.

References

[2.1.1. a] http://elacarte.com/
[2.1.1. b] http://www.businessweek.com/articles/2013-12-03/applebees-is-now-serving-tablets
[2.1.2.a] http://ziosk.com/

[2.2 a] http://blogs.strategyanalytics.com/WSS/post/2014/01/29/Android-Captured-79-Share-of-Global-Smartphone-Shipments-in-2013.aspx
[2.2.b] http://developer.android.com/images/brand/Android_Robot_200.png
[2.2.c] http://www.passion4teq.com/articles/ios-android-development-comparison-2/
[2.2.d] http://www.statista.com/chart/1903/average-selling-price-of-android-and-ios-smartphones/
[2.2.e] http://www.tuaw.com/2014/03/31/ifixit-ceo-kyle-wiens-apple-is-doing-all-they-can-to-put-third/
[2.2.f] http://www.trustedreviews.com/news/Windows-Tablet-OS-Not-Arriving-Till-Late-2012
[2.2.1.a] http://developer.android.com/about/versions/android-4.4.html
[2.2.1.b] http://developer.android.com/about/index.html
[2.2.1.c] http://www.developereconomics.com/reports/q3-2013/
[2.2.1.d] http://developer.android.com/design/index.html
[2.2.2.a] http://www.zdnet.com/blog/btl/trial-android-chief-on-why-java-was-picked-for-android/75179
[2.2.3.a] http://developer.android.com/tools/index.html
[2.2.3.b] http://developer.android.com/design/building-blocks/index.html
[2.2.3.c] http://developer.android.com/design/building-blocks/index.html
[2.2.4.a] https://sqlite.org/famous.html
[2.2.4.b] http://sqlite.org/mostdeployed.html
[2.2.4.c] https://sqlite.org/about.html
[2.2.4.d] https://sqlite.org/testing.html
[2.2.4.e] https://sqlite.org/
[2.2.5.a] http://developer.android.com/tools/debugging/ddms.html

[2.3.a] http://agilemanifesto.org/
[2.3.b] http://agilemanifesto.org/principles.html
[2.3.2.a] http://www.agile-ux.com/tag/paper-prototyping/
[2.3.2.b] http://developer.android.com/training/design-navigation/example-wireframe-device-template.svg
[2.3.3.a] http://www.mountaingoatsoftware.com/agile/user-stories
[2.3.4.a] http://scaledagileframework.com/spikes/
[2.3.4.b] http://developer.android.com/training/basics/firstapp/index.html
[2.3.5.a] http://guide.agilealliance.org/guide/incremental.html

[3.2.1.a] http://developer.android.com/design/get-started/principles.html
[3.2.1.a] https://developer.android.com/design/patterns/index.html
[3.2.1.b] http://developer.android.com/training/best-ux.html
[3.2.2.a] http://developer.android.com/guide/topics/ui/accessibility/index.html
[3.2.2.b] http://developer.android.com/design/patterns/accessibility.html
[3.2.2.c] http://developer.android.com/design/style/metrics-grids.html
[3.2.2.d] http://developer.android.com/design/style/writing.html

[3.3.4.a] https://www.gov.uk/government/publications/front-of-pack-nutrition-labelling-guidance

[4.a] http://developer.android.com/training/articles/perf-tips.html
[4.2.a] http://developer.android.com/guide/components/fragments.html
[4.2.b] http://developer.android.com/tools/support-library/index.html
[4.2.a] http://developer.android.com/guide/components/fragments.html

[4.4.a] http://www.currency-iso.org/en/home/tables/table-a1.htm

[4.5.3.a] http://nutritiondata.self.com/facts/vegetables-and-vegetable-products/2383/2
[4.5.3.b] http://multimedia.food.gov.uk/multimedia/pdfs/frontofpackguidance2.pdf

[4.6.a] http://developer.android.com/guide/topics/ui/declaring-layout.html#AdapterViews
[4.6.1.a] https://www.sqlite.org/fts3.html

[6.1.1.a] http://developer.android.com/tools/testing/testing_ui.html
[6.1.2.a] http://developer.android.com/tools/testing/testing_accessibility.html
[6.1.2.b] http://developer.android.com/guide/topics/ui/accessibility/checklist.html

[6.2.2.a] http://nandos.co.uk/sites/default/files/Restaurant-Menu.pdf
[6.2.2.b] http://www.zomato.com/london/wetherspoons-j-d-wetherspoon-finchley-road/menu
[6.2.2.c] http://www.mcdonalds.co.uk/ukhome/Food.html

Appendix

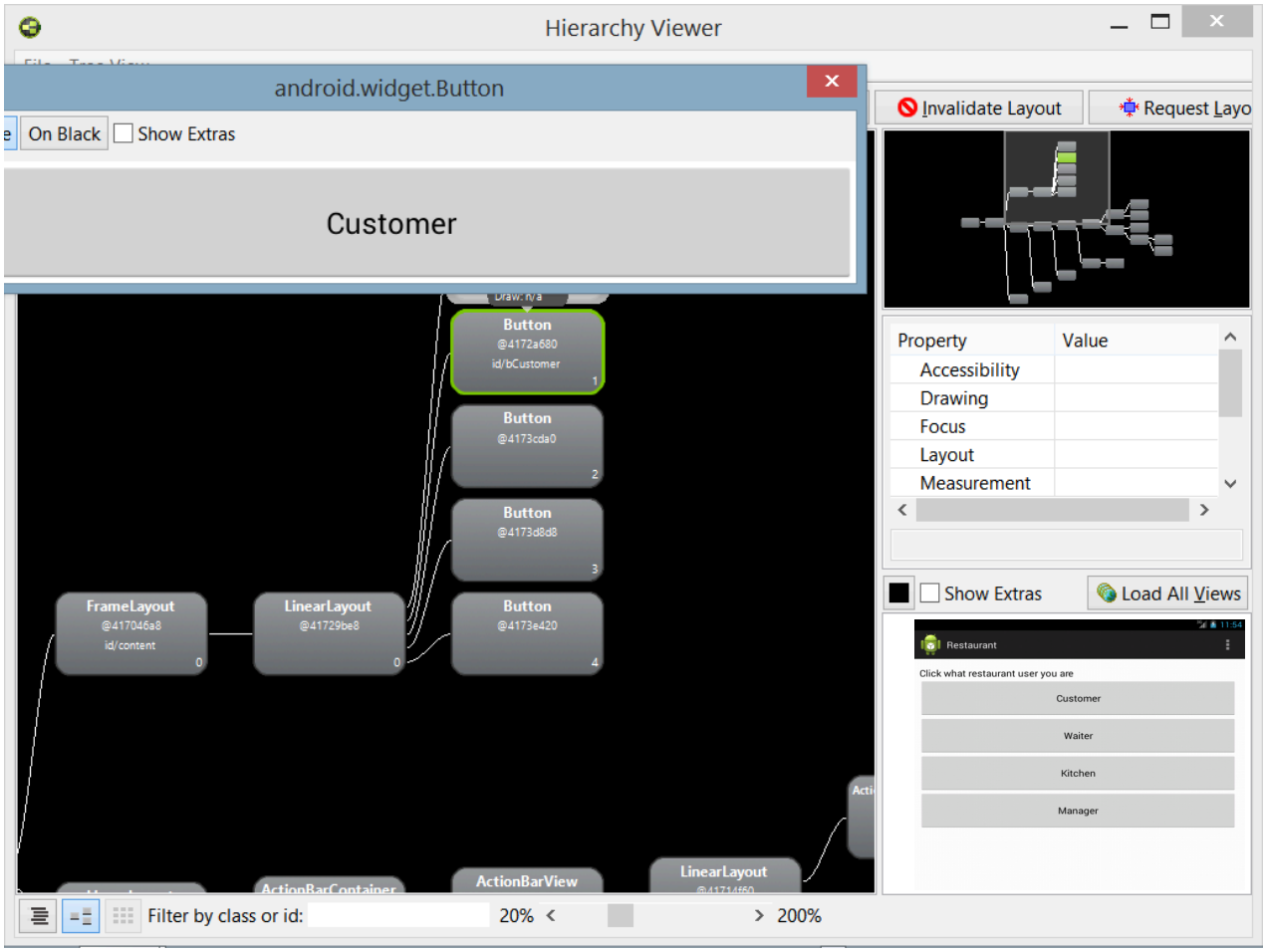


Image of Hierarchy viewer for GUI components

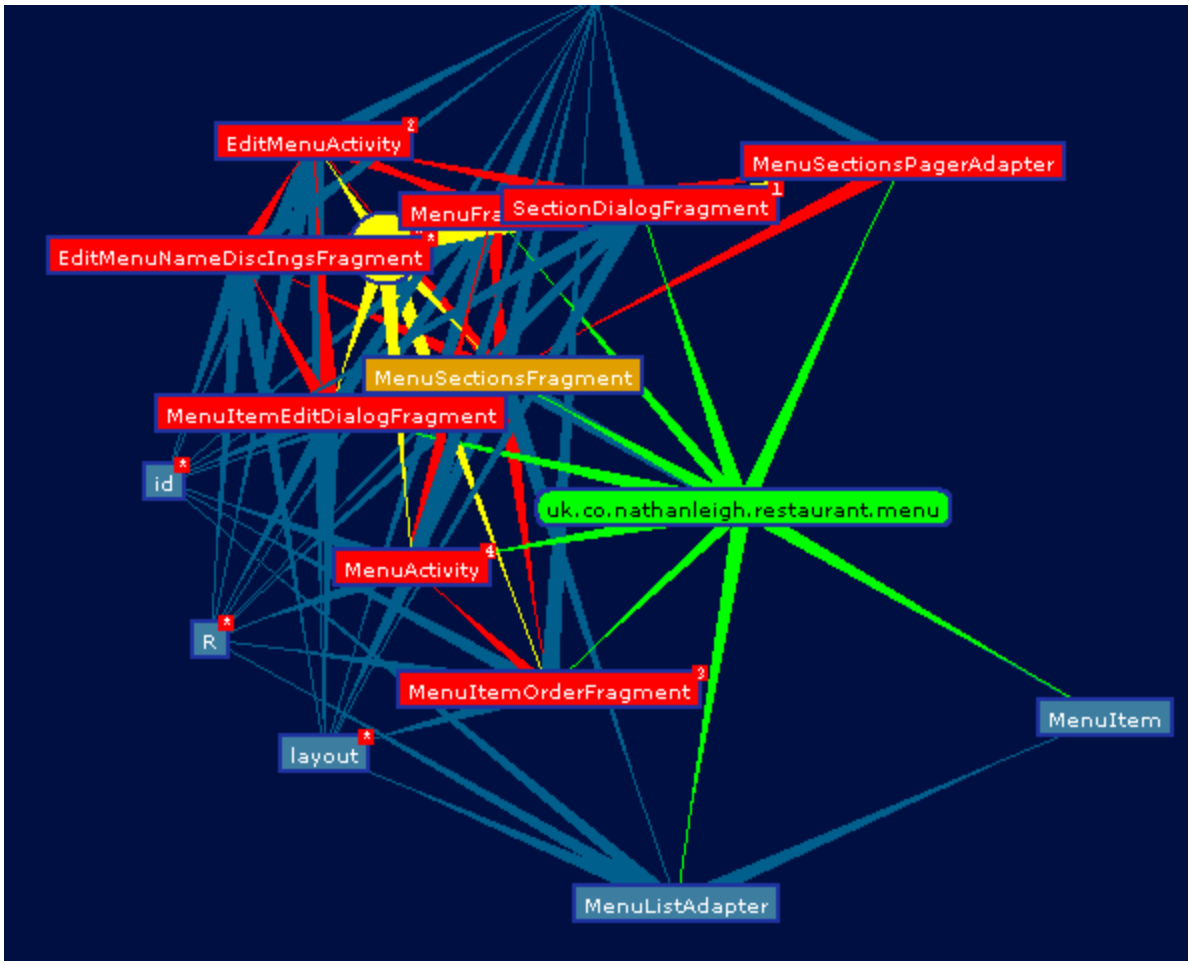
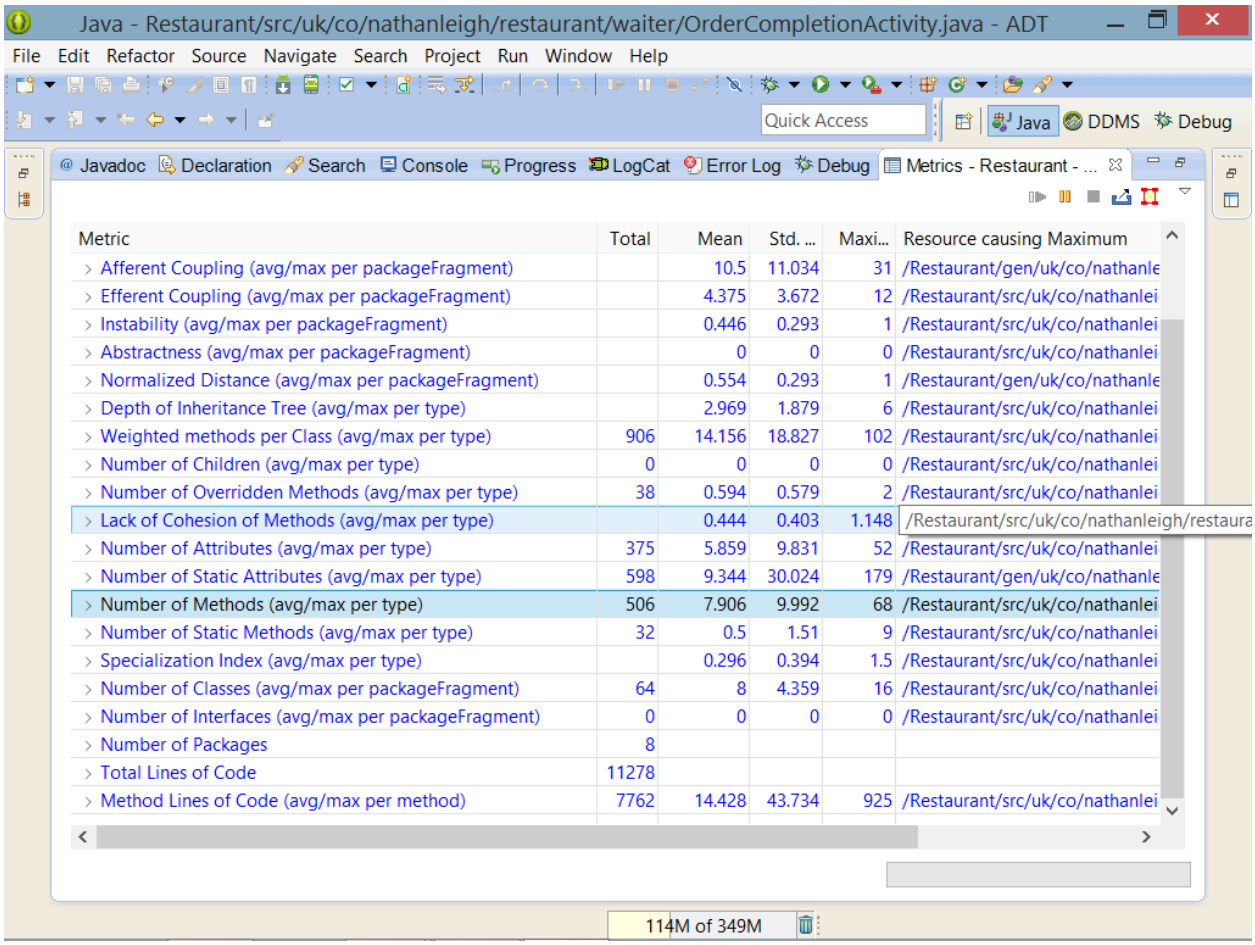


Image of class dependencies



Screenshot of project metrics

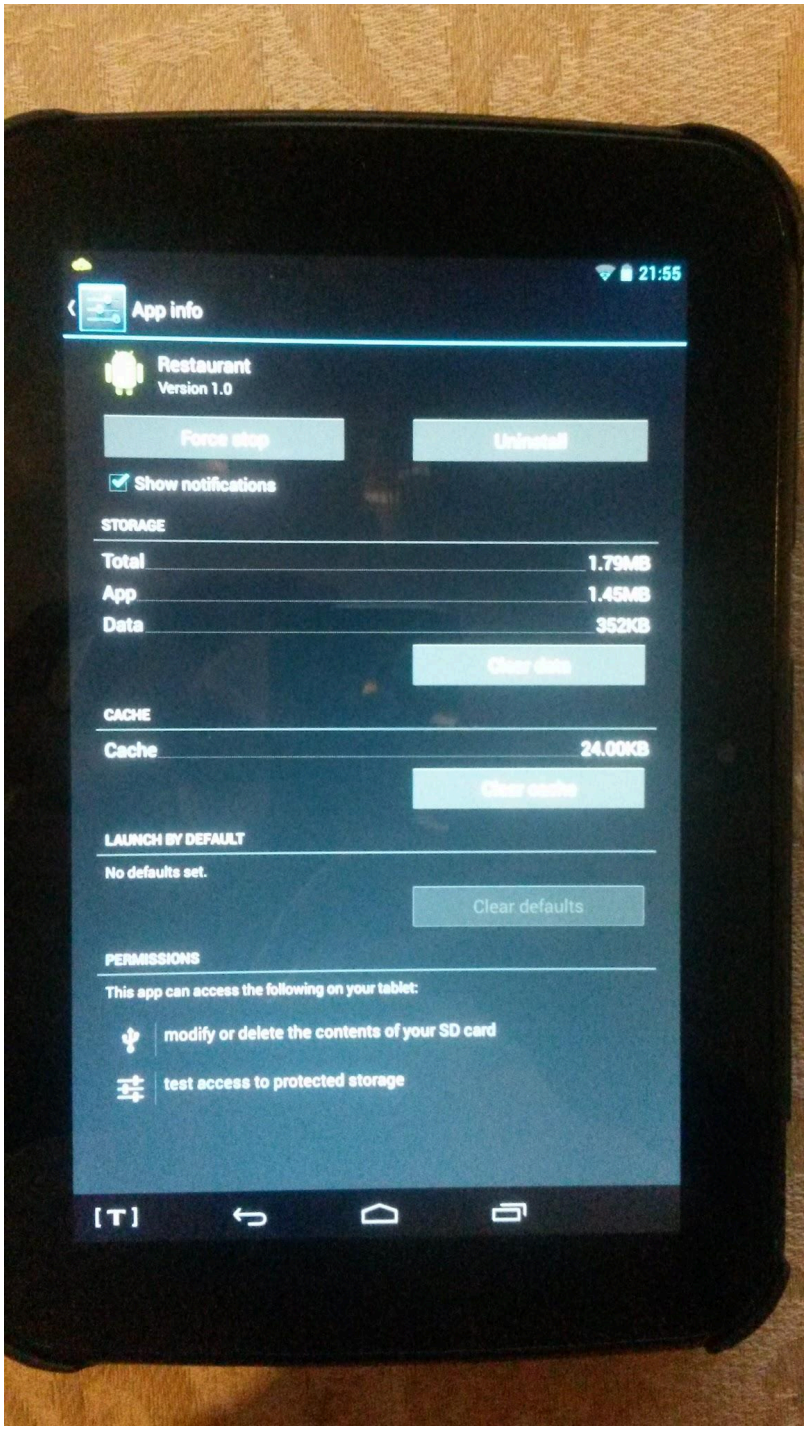


Photo of the application information on a real device