
UNSW
AUSTRALIA

**COMP[3/9]900 – Computer Science & Information Technology Project**

**Project 6-WAITER-T1, 2020**

# Project Report

**Submitted by**

## Team – WAITLESS


| | |
|---|---|
| Kavitha Narayanan | z5190588@student.unsw.edu.au |
| Matthew Chen | z5075025@student.unsw.edu.au |
| Dankoon Yoo (Scrum Master) | z5116090@student.unsw.edu.au |
| Daniel He | z5167773@student.unsw.edu.au |
| Kai Xiang Yong | z5175681@student.unsw.edu.au |

# Contents

# Introduction

In a highly competitive age of customer service and business efficiency, technologies are transforming the hospitality experience in restaurants.

Traditional restaurants employ many wait staff to receive and carry the orders from the customers to the kitchen. This leads to the problem of long wait times during busy periods. Additionally, miscommunication between the customer and the wait staff can lead to wrong orders or improper queuing of orders between customers. Furthermore, management has the problem of adjusting the workforce according to the demand and day, leading to increased costs or increased risk of having too many or too little staff working on the day.

Technologies are widely used in the food industry nowadays as part of an effort to enhance customer dining experience and make service more efficient. However, some implementations of such software applications available on the market currently do not eliminate the inefficiencies of the traditional restaurant service. Many require customers to have their own device in order to work and some only help the customers in ordering stopping short and not enhancing the workflow for restaurant managers, kitchen and wait staff.

The Waitless System attempts to solve the problems of both the traditional restaurant setting as well as the partial software applications currently used by providing a complete end-to-end application; from setting up the menu by the managers, to the moment the customer receives their order.

In the Waitless System, the restaurant provides its own tablet device for its customers. As a result, customers are not expected to use their own device in order to place an order or enjoy any technology-enhanced services. The digital menu is easily and immediately updated on the fly through a separate management-only interface, and changes can be reflected without delay in the customer application at no additional cost unlike traditionally printed menus.

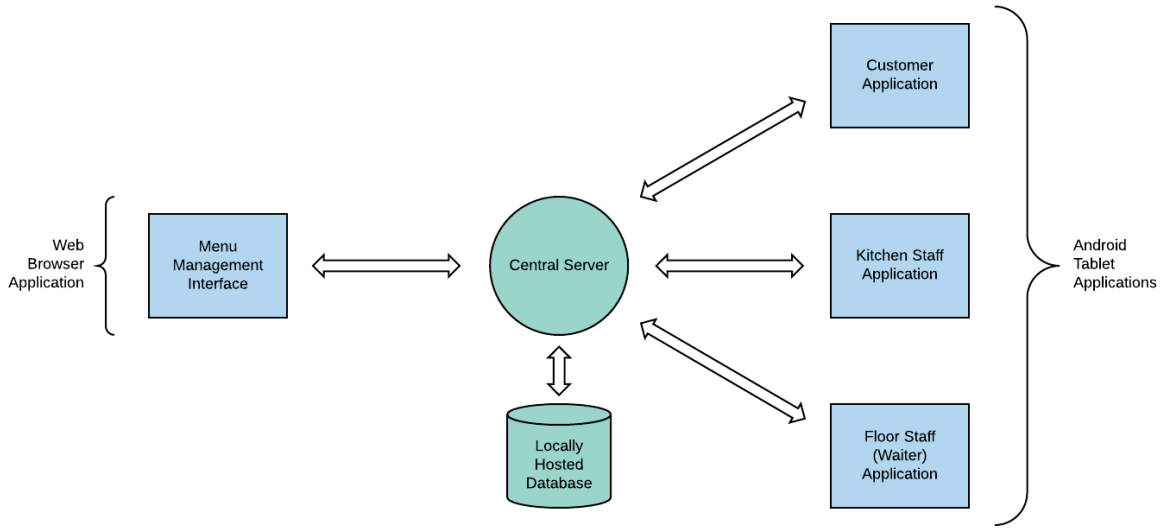
Additionally, we acknowledge that restaurant staff are equally as important as customers. Waitless provides interfaces for wait-staff and kitchen staff, to assist them in keeping track of order progressions with a transparent and easy-to-use interface to help enhance and refine their work processes which also leads to increased productivity and worker efficiency.

To surmise, Waitless is a software-driven system that is faster with low latency and more convenient for both restaurant staff and customers, highly scalable to meet demand and cheaper to operate than traditional restaurant systems that are entirely human dependent.

# Architecture and Design

## System Design

A restaurant setting has many different actors, each with different functional requirements. As a result, the Waitless system incorporates a distinct and specific application for each of the four types of end users. Each of the user applications communicate solely with a single central server.




The **Server** acts as a central hub - user applications use HTTP requests to make API calls to the central server in order to push or pull data. This ensures there is a single repository of data, and all applications will be able to access data updated in real time. Without the need to directly communicate between end user applications, the system is scalable without a complex communication network.

The **Database** is hosted locally on the same machine as the central Server, and the database can only be accessed via this server. This allows multiple instances of the user applications to access identical data, and any changes to the data are not specific to any single application instance. For more information about the design of the database, see the *Database Design* section.

The **Menu Management Interface** is a web browser application to be run on the same machine as the Server. A web browser is a familiar terminal for many users and ensures users will be comfortable performing the many functions of this interface.

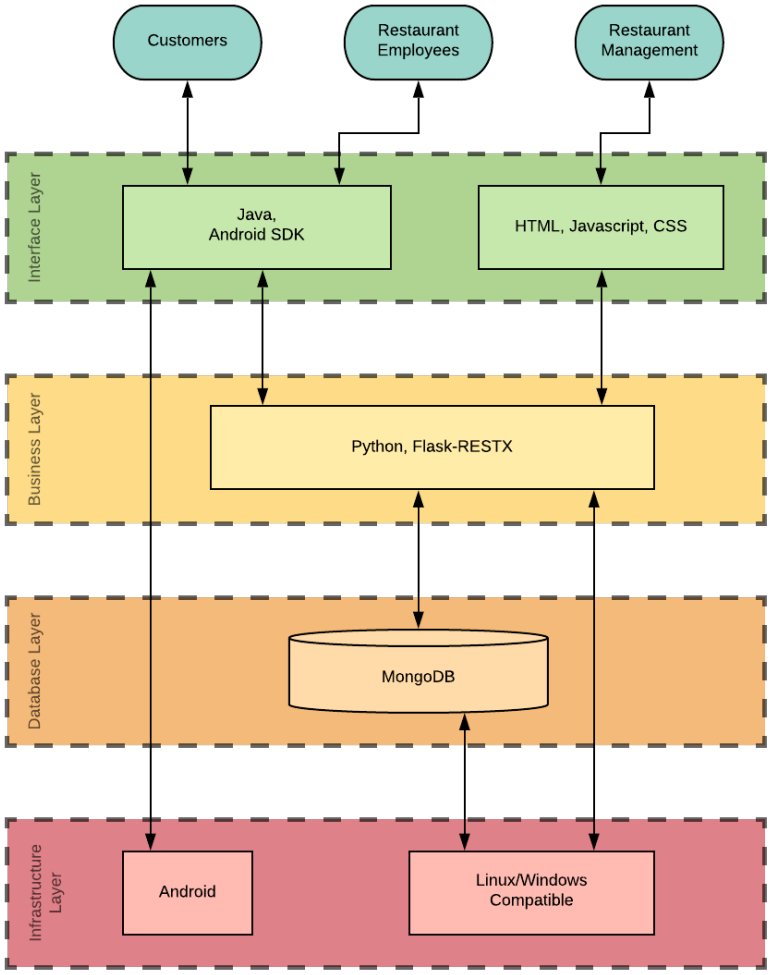
The **Customer Application**, **Kitchen Staff Application** and **Wait Staff Application** are designed to be Android tablet applications. Tablet type devices are ubiquitous in today's technological landscape and should be familiar to most end users. Android tablets have the additional benefit of a diverse range of manufacturers, with many more cost-effective models being perfectly suitable for the Waitless system. This consideration ensures the initial investment in the Waitless system is not excessive, and the cost to scale up the system to increase table covers is affordable. In the long term, the one-off cost of hardware investment will be significantly lower than the ongoing cost of human labour (waiters) in a traditional restaurant setting.

The Customer Application is designed to be scalable, with each single instance of the application servicing each table in the restaurant.

The Employee Applications (Kitchen and Wait Staff) are designed to be singular hubs for each type of employee. There is no need to invest in individual machines for each employee. Instead, traditional skills of teamwork and negotiation are still necessary amongst each group of employees.

The Waitless system largely replaces the need for communication between customers, waiters and the kitchen. This software-driven system ensures faster, more accurate and more convenient communication between customers and restaurant staff, while being highly scalable and cheaper to operate than a traditional restaurant system.

## Software Architecture

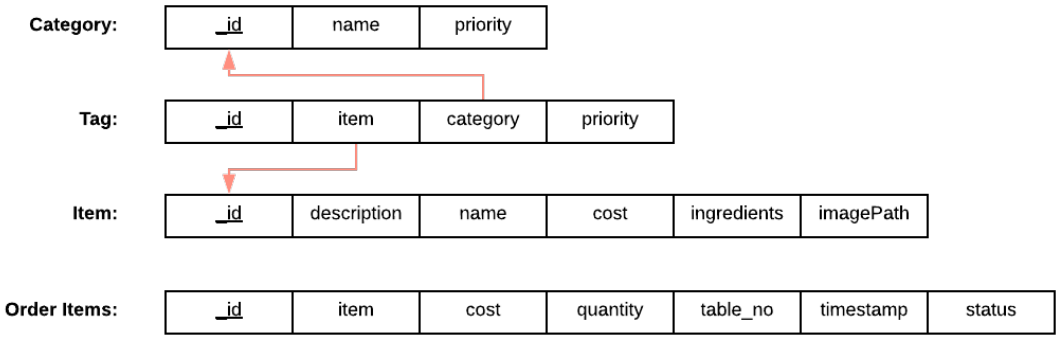



The interfaces of the Waitless system are built with Java and Android SDK for the Android applications, and HTML, JavaScript and CSS for the browser-based application.

The Business Layer of our Server is built with Python and Flask-RESTX to create API endpoints which can be called by the Interface Layer.

The Database Layer will only communicate with the Business Layer, and is utilizing MongoDB, which is free and easy to integrate with our Python Business Layer.

## Database Design




Although MongoDB is not a relational database, the above relational schema best represents the design of Waitless' database.

The Category, Tag, and Item collections are used to store data related to the restaurant menu.

**Category** is a simple collection of menu category names (name), and what order they appear in the menu (priority). Each entry also has a unique ID (_id).

**Item** is a collection of all menu items. Each entry has an ID (_id), a short description (description), an item name (name), a price in decimals (cost), a list of ingredients (ingredients), and a path to an image of the item (imagePath). Item images should be stored on the local machine that is running the server and hosting the database. The Menu Management Interface should also be run on the same machine to ensure correct and valid image paths are provided.

**Tag** is a collection that represents the relationship between Items and Categories. Each entry has an ID (_id), a reference to an item's ID (item), a reference to a category's ID (category) and what order the specified item should appear within the specified category (priority). This implementation of the relationship between items and categories allows for a many-to-many relationship, such that categories can contain many individual items, and items can be "tagged with" a multitude of appropriate categories.

**Order Items** is a collection that represents orders placed by customers. Each entry has a unique ID (_id), the name of the item ordered (item), the decimal price of the item (cost), the quantity of the item ordered (quantity), the table that placed the order (table_no), when the order was placed (timestamp), and the current progress of the order (status). The status of the order can have the following values: "Pending", "In Progress", "Ready", "Delivered", and "Paid". The timestamp of the order should contain an ISO date format string.

The Order Items collection was designed with consideration for potential extension of the Waitless system's functionality to include the generation of sales data and statistics. All past orders can be stored in the database, and the database can be simply queried to generate data for a given time range.

Notably, the Order Items collection entries do not contain references to Item entries. This ensures Order Items entries are accurate snapshots of sales at the time of ordering. Future changes to a particular Item's cost or potentially the removal of an Item will not affect the accuracy of past orders. As a result, it is necessary to store the item name and cost in each entry.

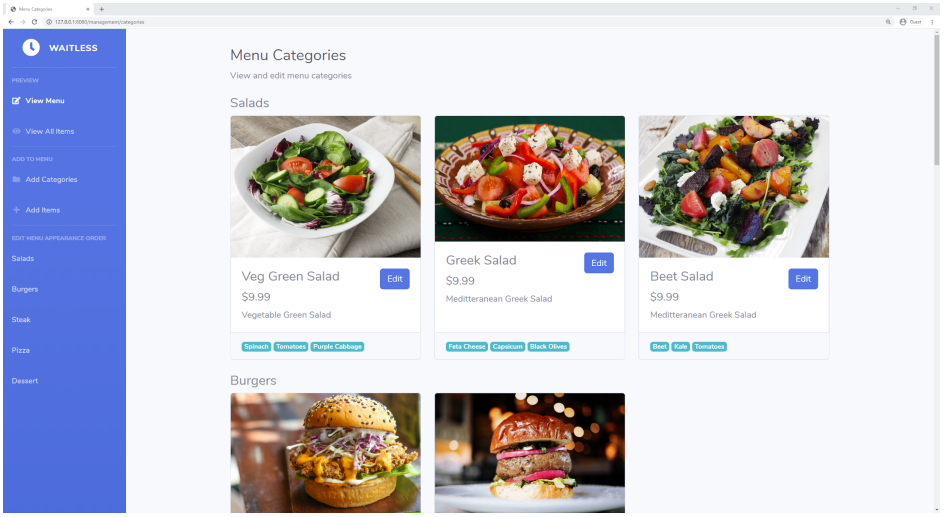
The database is designed to only store significant data that needs to persist between running instances of the server. Data for tables requesting assistance from waiters is not stored in the database. A simple queue structure housed in the server itself was the most efficient implementation for this function.

# System Functionality

The following functionalities cover the EPICS C1, C2 (Customer), E1 (Kitchen Staff), E2, E3 (Wait Staff) and M1 (Restaurant Manager)

## Restaurant Menu Management

The main purpose of the website is for management to edit the menu. This includes adding, editing, and removing offerings, and adding and changing the appearance of categories.

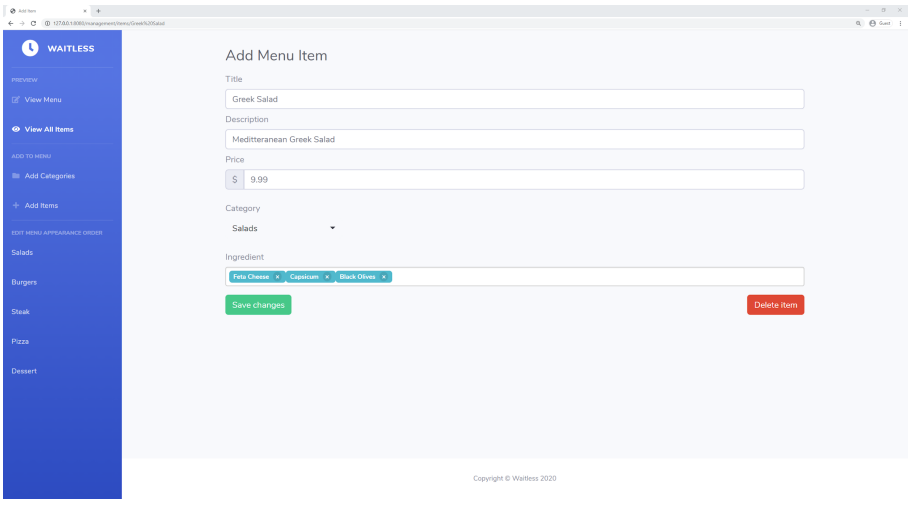



Unlike the flow in the original proposal, the tasks are listed on the left hand side similar to a navigation bar to provide more familiarity.

The first tab allows the user to view the menu as customers would. Dishes are displayed in cards divided by category and show the name, description, price, ingredients, and picture. Dishes that are under multiple categories would appear multiple times.

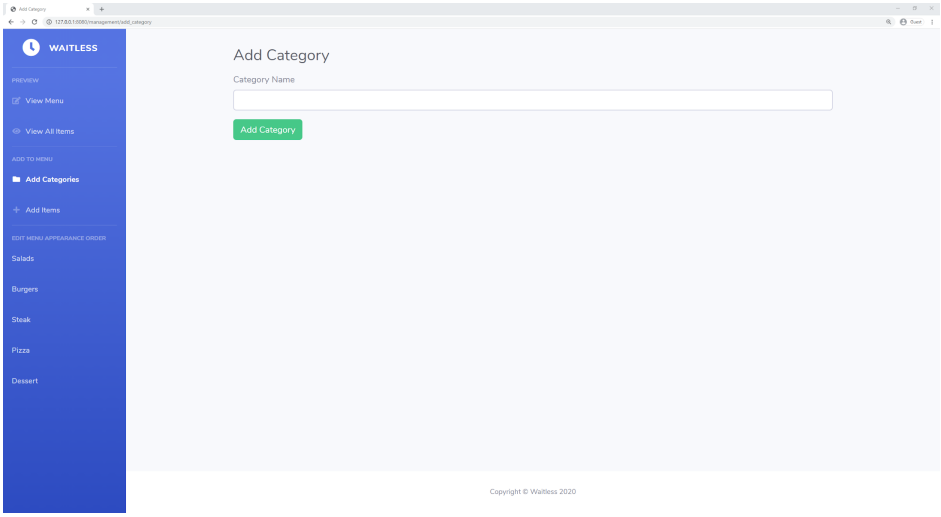
'View All Items' provides an alternative view of the menu, where all items are displayed without category headers.

The user can edit an item through the edit button, which leads to a page where the name, description, price, ingredients, and categories which it is under can be changed.



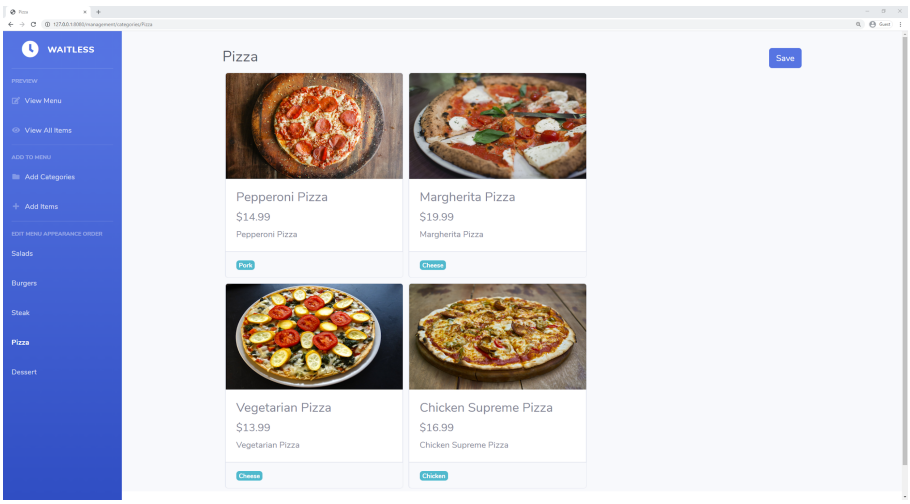
Here the user can also opt to delete the item entirely. Saving the changes will update the menu and items and return the webpage to the ‘View All Items’ tab.

‘Add Categories’ provides a simple way to create a new category.




The ‘Add Item’ tab allows the addition of new dishes. The interface is almost identical to editing an item but has the additional option of selecting a picture.

The user can rearrange the order of categories that would be displayed on the customer app. They are listed under ‘edit menu appearance order’ on the navigation bar and can be dragged and dropped into the desired order. The order of items within each category can be changed. Tapping on one of the categories leads to a page where the dishes can also be dragged and dropped into the desired order. Changes in the order of categories are auto saved while changing item order requires the save button.

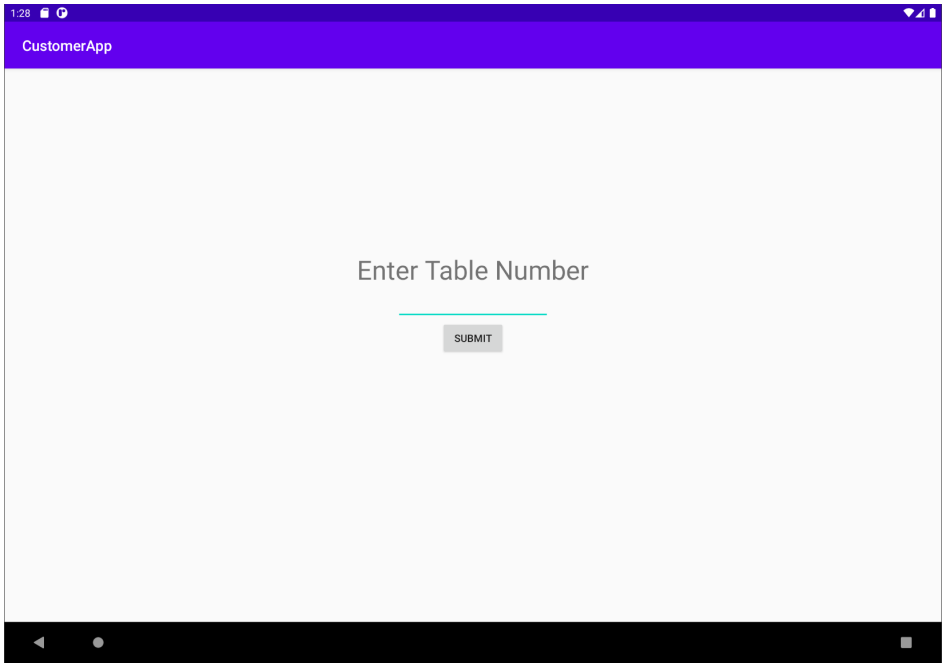



Overall this provides a simple intuitive way for management to control how customers view the menu.

## Customer Application

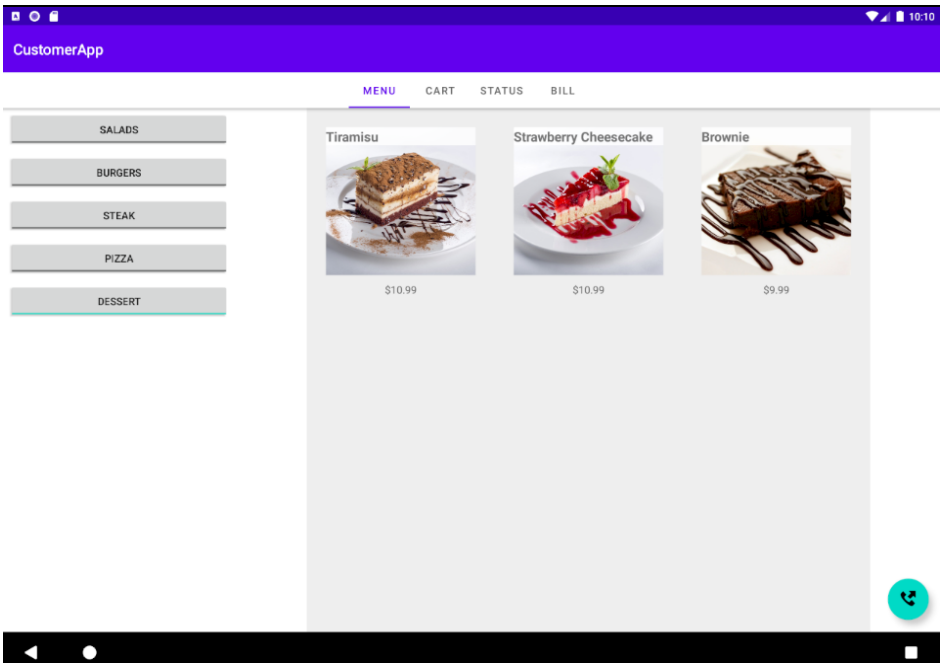
The customer application acts as a terminal for a particular table and it is the main interface that restaurant customers will interact with. Its main functions are to allow the user to view the menu, order items from the menu, view order progress, view the bill, and request assistance.

**Setup Page**

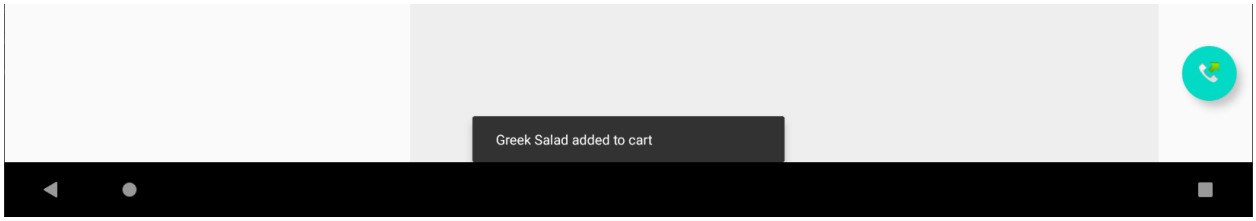



On start-up of the customer app, it will require a unique table number. This is entered by staff members for every table during the opening of the restaurant. Allowing the staff to manually enter the number is beneficial as it can intuitively match the physical arrangement of the tables.

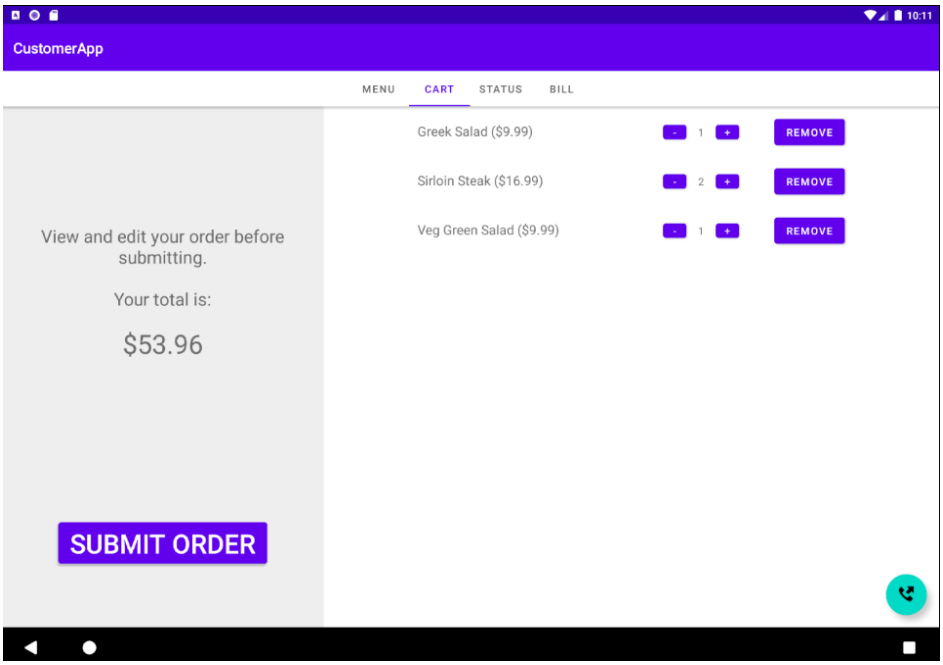
**Menu Page**




The home page displays the menu. There is a list of categories on the left-hand side. Selecting a category will display all the dishes in that category on the right-hand side. The order of the categories and the order of items within each category are all predefined. Each item card shows the dish's name, a picture, and its price. In the original proposal, selecting a card would open a page which would show details and ingredients of the dish. To avoid disorienting the user and cluttering the screen, this page was omitted entirely and selecting a card will simply add one of the item to the cart with a confirmation message.

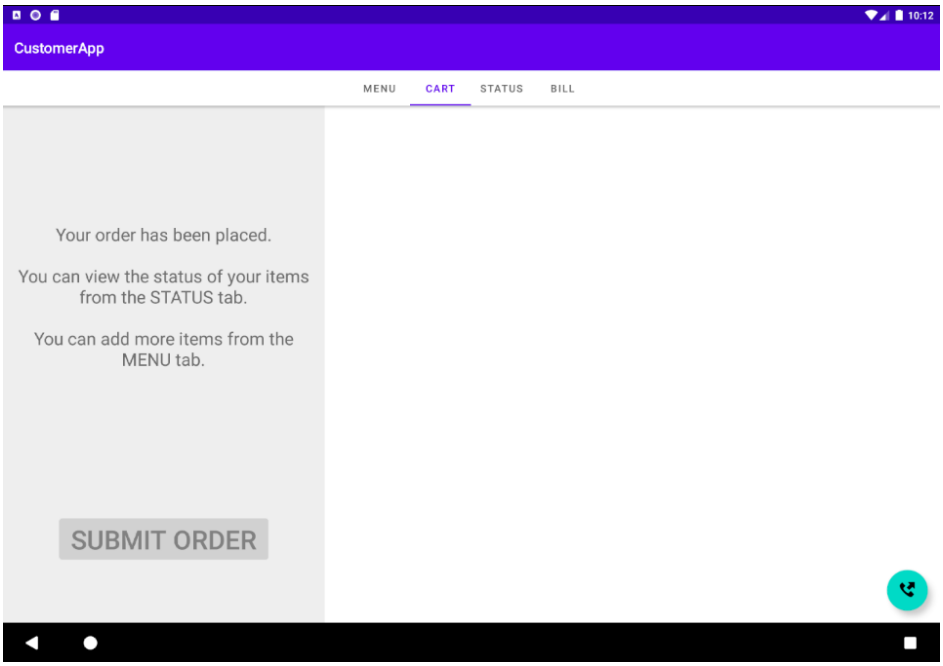


**Cart Page**




The cart page allows the customer to view and edit their selected items before ordering. This is similar to a cart or shopping basket used in websites so the user should be familiar with the layout. It allows the quantity of each item to be changed and shows the total price of the cart. Tapping the remove button or setting a quantity to zero will remove an item from the cart.

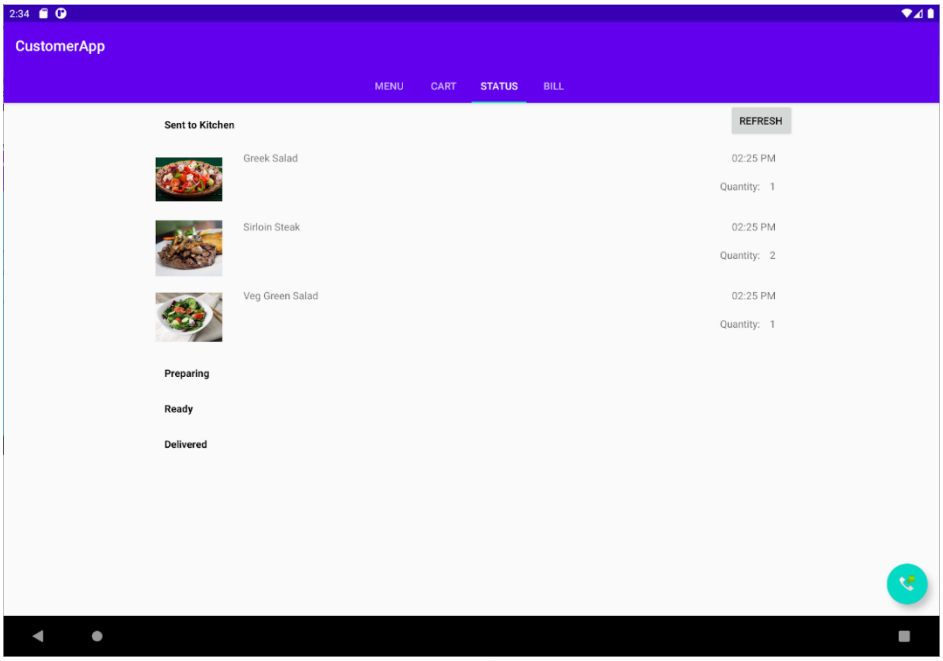
Submitting the order will send the items to the server, which will then pass them through to the kitchen as well as add the items to the table's bill.




The cart will be cleared, and information is displayed on the screen. Unlike the original proposal the user is not immediately redirected to the status page; this allows them more control over the flow. In

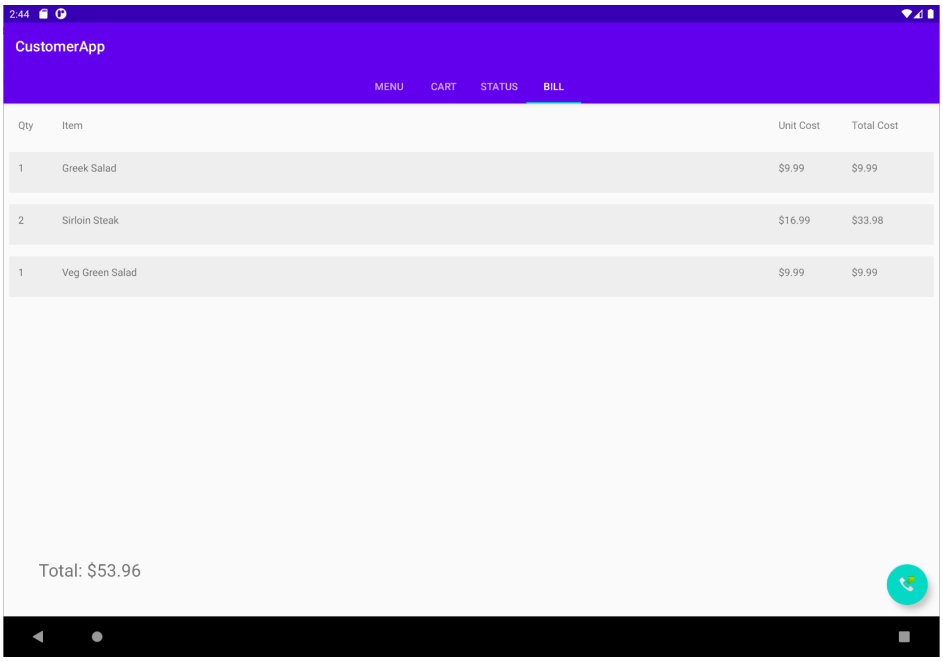
general, throughout the app the tabs (pages) will not change without the user manually selecting the tab or swiping to the left/right. Compared with the proposal, this behaviour is more predictable and produces less disorientation.

**Status Page**




Here the user can view the progress of their orders. Ordered items will appear under one of the categories ‘Sent to Kitchen’, ‘Preparing’, ‘Ready’, and ‘Delivered’ to reflect its current status. Updates are requested from the server periodically, however a refresh button accommodates impatient users. This is an addition after the proposal and gets an update on demand.

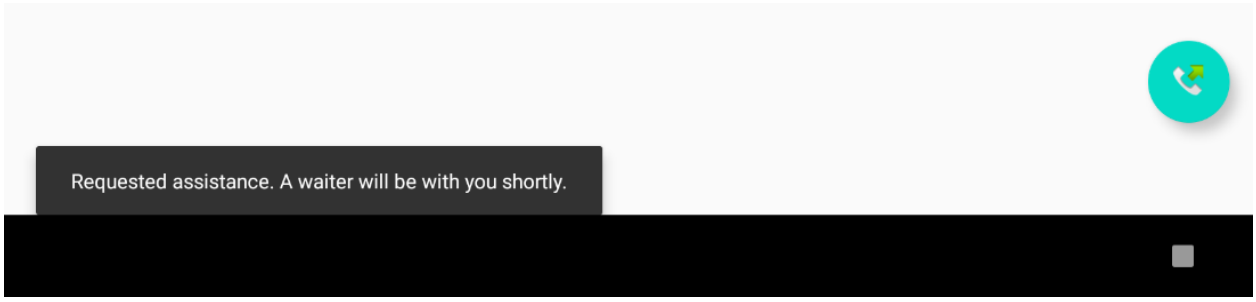
**Bill Page**



Switching to the bill tab will retrieve the current bill from the server for that table. It shows all the ordered items, quantities, and costs.

**Assistance Button**

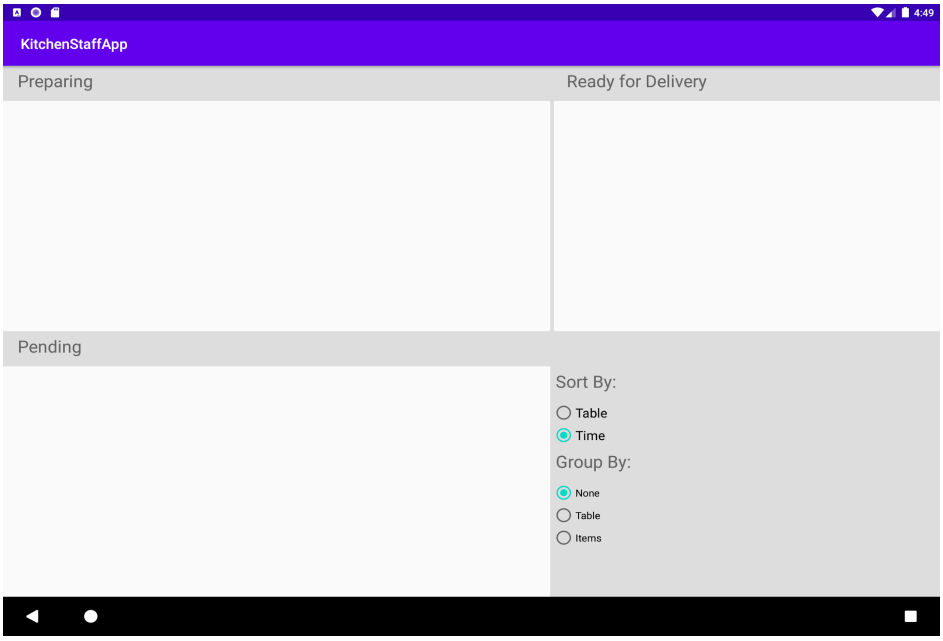
The assistance button is visible on every page and allows the user to request assistance from wait staff for any reason. Selecting it will display a pop-up message either confirming a request was sent or that a pending request was already sent.




## Kitchen Staff Application

The main functions of the Kitchen Staff Application are to display orders placed by customers so cooks can prepare order items, and to allow cooks to update the status of these items to notify customers and waiters.

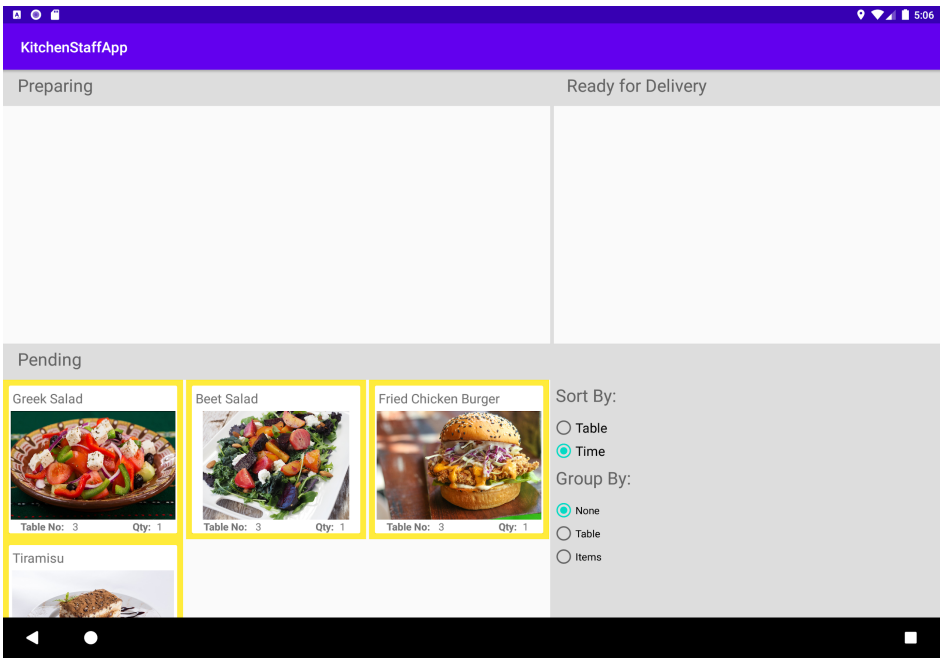
The Kitchen Staff Application has a simple, single-page user interface.




Most of the screen is dedicated to displaying orders placed by customers. These are divided into three sections: “Pending” to the bottom left, “Preparing” to the top left, and “Ready for Delivery” to the top right.

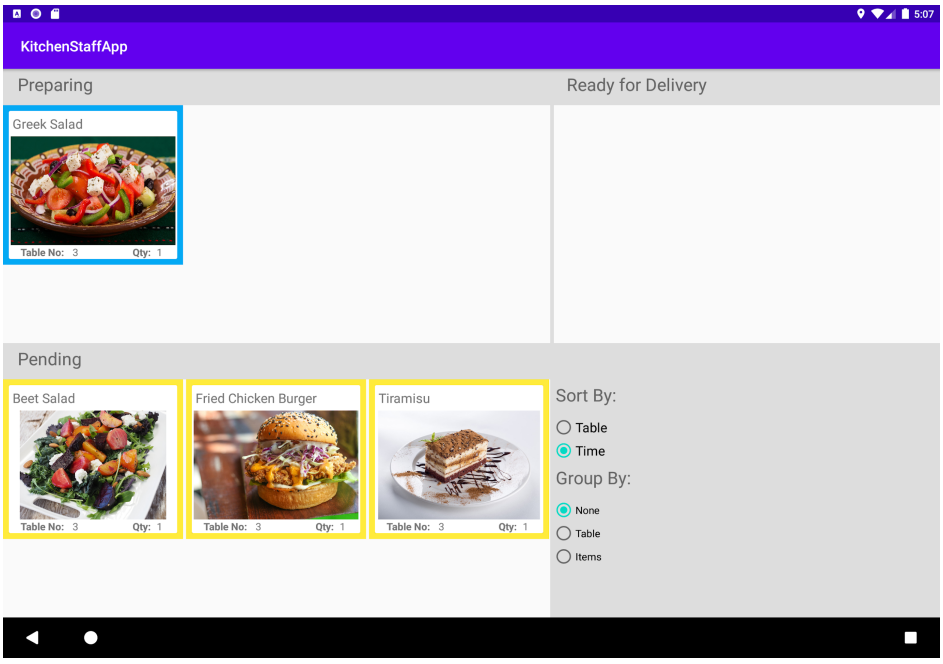
To the bottom right, there are a few options to group and sort order items within each section. By default, orders are sorted by the time they are placed, so orders can be completed and delivered in a timely manner. However, if kitchen staff choose to, orders can be sorted by table. Orders can also be grouped by table to allow kitchen staff to attempt to complete items for a particular table at the same time.

Orders can also be grouped by items, allowing the kitchen staff the flexibility to efficiently prepare the same items in bulk.

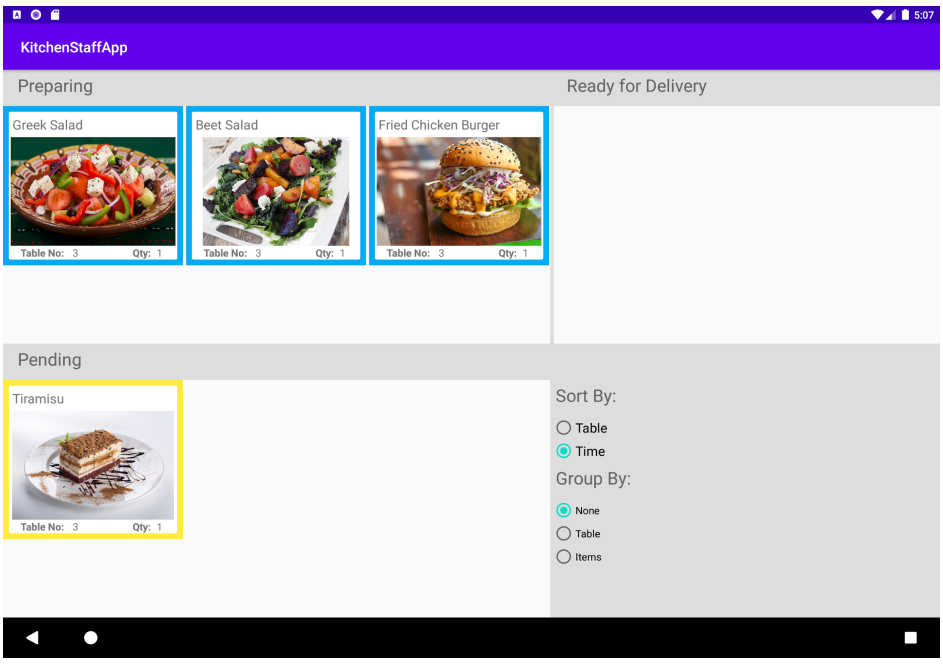



New orders placed by Customers will automatically appear in the “Pending” section. This section is colour-coded yellow for quick identification.

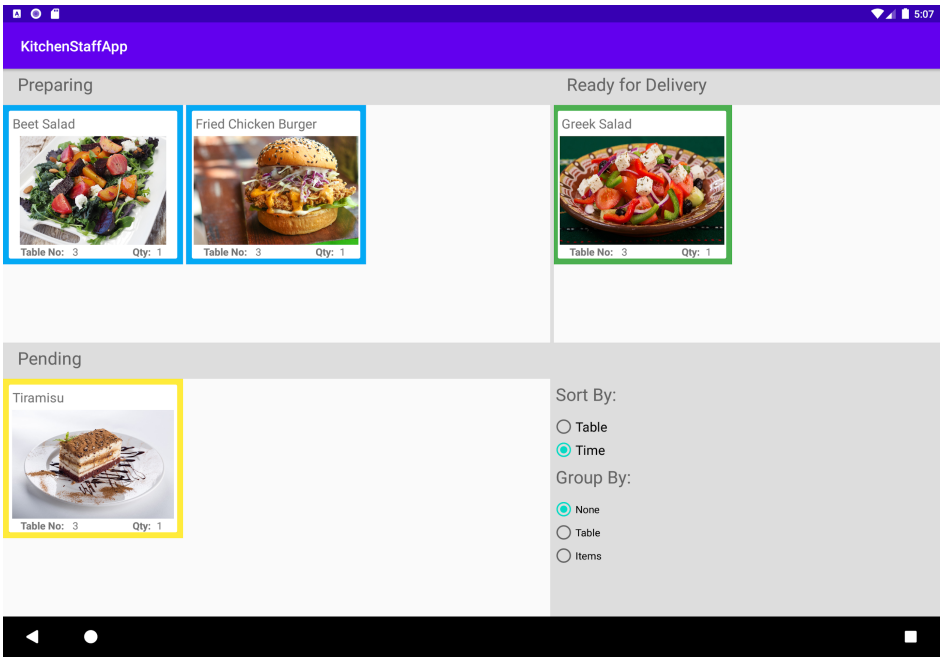
Kitchen staff can tap on a particular item from the “Pending” section to advance its status to “Preparing”.




Items in the “Preparing” status indicate a cook is currently preparing that item. Items in this section are colour-coded blue.




There may be multiple items being prepared by different cooks at the same time. Kitchen staff will still require the basic skills of effective communication and teamwork to keep track of themselves and others.




Once a cook has completed an item, they can tap on that item from the “Preparing” section to advance it to the “Ready for Delivery” section. Items in this section are colour-coded green, and are ready for waiters to deliver to the correct table.

These colour-coded order statuses are also displayed in real-time in the waiter’s app, so waiters will be aware when orders are completed.

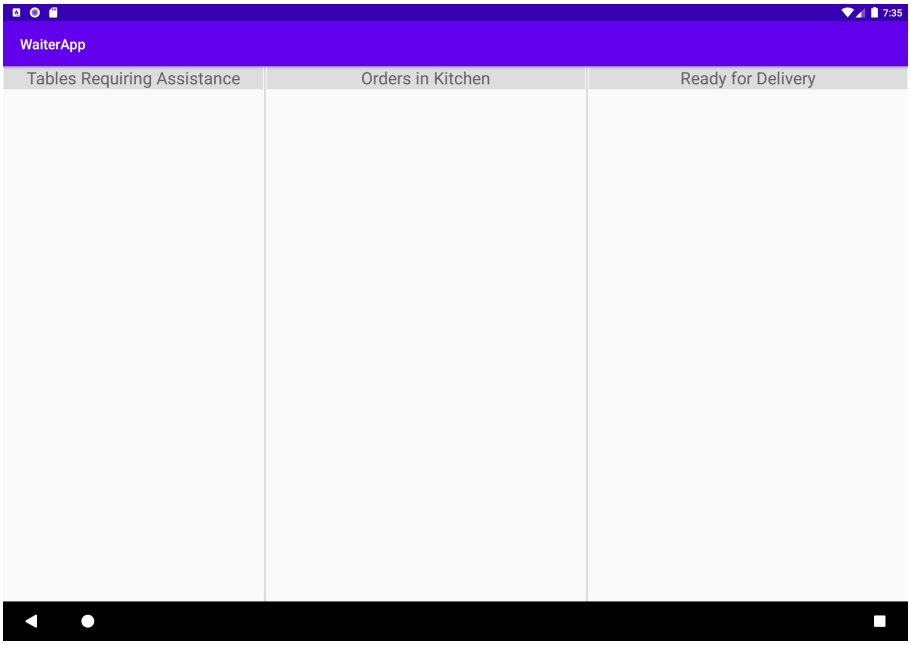
## Wait Staff Application

There are two main functions of the Wait Staff Application.

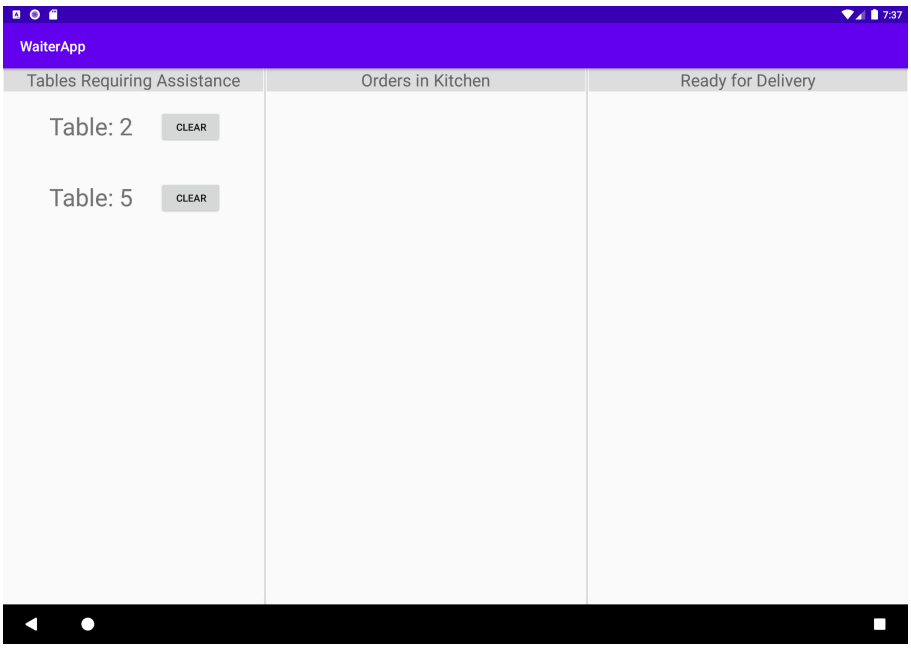
Firstly, the app notifies waiters of tables requiring human assistance. Although the Waitless system eliminates many drawbacks of a traditional restaurant system, the flexibility and familiarity of a human waiter may be preferable in some situations.

Secondly, waiters must be notified when items are completed by the kitchen and ready to be delivered to customers.

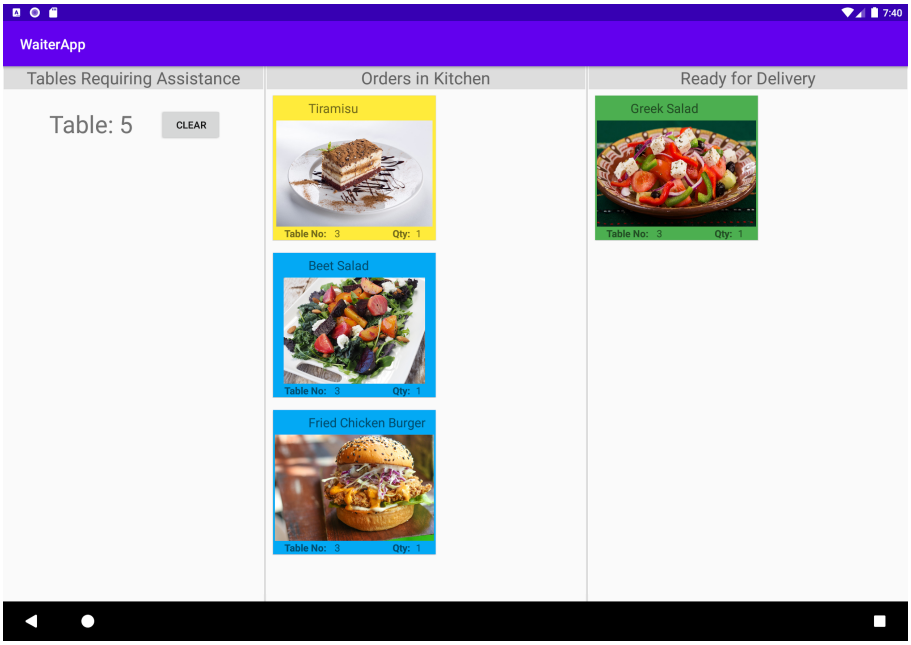
Like the Kitchen Staff Application, the Wait Staff Application has a simple, single-page user interface.




The screen is divided into three sections: “Tables Requiring Assistance”, “Orders in Kitchen”, and “Ready for Delivery”.



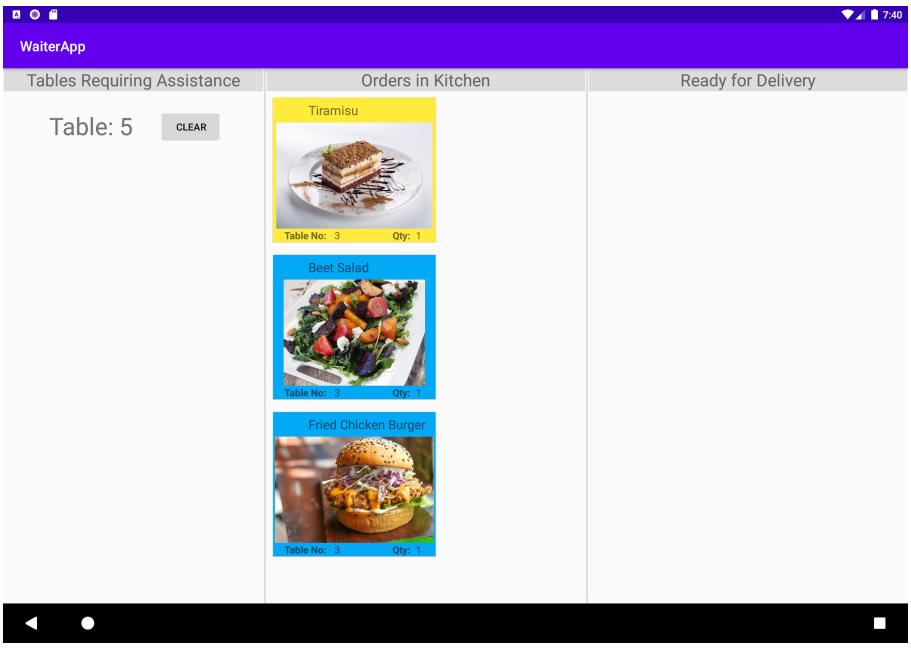

When a customer pages for assistance from waiters, an entry will be added to the “Tables Requiring Assistance” section. This list acts as a queue, with the earliest request at the top of the list. Waiters can tap the “Clear” button to remove the entry from the list. This signals that the request for assistance has been resolved.




The remaining two sections display the statuses of orders placed by customers.

In the “Orders in Kitchen” section, items from both “Pending” and “Preparing” statuses will appear here. Waiters are unable to tap on the items to change their status, as this responsibility lies with the Kitchen Staff Application. Items will still retain their colour-codes (yellow for “Pending” and blue for “Preparing”).

The “Ready for Delivery” section displays items completed by the kitchen staff. These items retain their green colour-coding. Wait staff can tap on items in this section to update their status to “Delivered”, and the item will be removed from the “Ready for Delivery” section in both the Kitchen Staff Application and the Wait Staff Application.




This app is designed as a central hub for restaurant wait staff.

A restaurant will likely employ more than one human waiter, so it is up to the wait staff team to communicate and coordinate the allocation of tasks.

# Implementation challenges (Reflection)

## Technical Challenges

### Lack of knowledge/skill with technologies

For this project, the complexity and scale which we were aiming for required the use of a number of technologies and frameworks the team was not familiar with. The technologies used all had varying levels of difficulty. Because most members of the team had little-to-no exposure to technologies outside of python or c, the team invested countless days into learning new technologies.

One of the more difficult of the technologies to learn included Android. We were using the Android SDK Android Studio IDE to develop apps facing the customer, wait staff and kitchen staff. The customer app in particular featured many tabs, layouts and nested layouts as necessary for meeting our requirements. One of the main challenges was in trying to have an action in an inner class trigger a change in a separate set of elements on a page or an external class on another page. We called these issues intra-fragment communication and inter-fragment communication respectively. The solutions involved countless hours of research into the topic to figure out a solution from multiple members of the team. The intra-fragment communication was first figured out for use in displaying menu items for categories and a similar solution was applied to inter-fragment communication required for adding menu items to an order list.

## Management Challenges

### Task Delegation

Task delegation for such a complex and interconnected set of applications posed challenges that would otherwise not be present in smaller or simpler endeavours. The tasks to delegate could not simply be copied over from user stories or epic schedules in the timeline. This is because of the strong cross dependency of each user story to the next. For example, the app would need to be made and the API calls must work before a menu can be implemented and a menu must be implemented before orders could be implemented. Due to this, the team worked with less-than-ideal efficiency when the rest of the team were waiting on 1 or 2 team members to finish their parts so that another member could start working.

Such a challenge was heavily discussed in a team retrospective which was a reflective meeting to discuss successes, shortfalls and how the team could best overcome some of the shortfalls. During the retrospective, the team unanimously agreed that the previous approach to task delegation was not working to our advantage and came up with new methods to resolve the identified issues.

The solution involved assigning members an entire subsection of epics such as all the backend processes and API's involved in an Epic regardless of user story, or all the frontend of the Epic. Each member of the team was assigned a bigger task that they could work on independently for a longer period of time. For example, instead of everyone waiting around for others to work on epics 'x', 'y' in week 1 and repeating that for epics 'm', 'n' in week 2 and again in week 3 with epic 'j'; the team would work on x, y, m, n, j simultaneously and work in parallel to complete all by week 3. To make sure no one member was struggling with a particularly hard epic or had sudden or other commitments that prevented progress, more regular status checks happened in the form of half-weekly standup meetings. The purpose of the meetings was not to criticize or flaunt but to ensure everyone had visibility of each other's progress and to make regular readjustment of workloads. This regular readjustment of workloads or tasks was crucial to ensuring maximum productivity and output as it could actively redistribute workloads to make the most efficient use of time and manpower. The changes increased our productivity by an exponential margin, and we believe these changes were

crucial to our success in the completion of the project to leave room to spare, something that didn't seem likely before the changes.

### Communication

For virtual conferencing, the team relied on blackboard ultra, google hangouts or Facebook messenger video call feature. Virtual conferencing was primarily used for half-weekly stand ups/progress checks and or group planning/coding sessions. The non-regular calls were planned in advance on Facebook messenger or during a call.

Some communication challenges involved member commitments to other university courses, work, extra-curricular activities and family; which occasionally made it difficult to schedule calls or chat online.

### Tools

A number of tools were used to help manage this project. However, Jira was considered to be quite incompatible with some of our work processes and therefore did not play a big role in our project management. This is because of some of Jira's shortcomings. Jira's inflexibility with sprints and our new project management strategies that stemmed as a result of the retrospective mentioned above simply did not work with Jira's implementation. This was because it was difficult to have Jira accurately track progress beyond user stories, an approach we abandoned during the retrospective. That along with longer periods of parallel work and more regular standups including a regular readjustment of delegation also made Jira a poor indicator and unsuitable as a management tool since it only added to work and did not simplify it.

## Timeline Challenges

### Failed assumptions of workload and ambitious timeline

The initial proposal included a timeline that was very ambitious and indicative of our failure to properly identify the workload required for each epic. For example, epic c1 which turned out to be the most challenging and complex epic was given 1 week alongside another epic e1 in our timeline. Such gross miscalculation of workload made it difficult to meet the overly ambitious deadlines set by us in the timeline. The ambitiousness in the timeline is also visible in the fact that the timeline expected the huge project to be completed in week 8 with more than 2 weeks to spare. The timeline targets were reconsidered during the retrospective and more realistic targets were set along with the new strategies to meet them. This was key in finishing in the week prior to the final demo giving us 1 week to spare after completion to polish code quality, work on the report and improve other aspects of the project submission.

## Other Challenges

### COVID19

The novel coronavirus outbreak (COVID-19), declared a worldwide pandemic by the world health organisation, has become the biggest disruption to daily life in modern history since World War 2. New lockdowns and restrictions to travel, university and work has forced a full transition to all-online work. Other impacts of COVID 19 could be felt in the change to lifestyles, the scarcity of essential items in shops, shutdown of grocery home delivery services and travel restrictions which only made it more difficult for everyone to continue working as normal. The team transitioned successfully to this challenge despite the complete shutdown of all in person meetings, by increasing the regularity of online check-ups and communication. The team was also able to work to the best of their abilities despite the changing conditions around them that could and were affecting livelihoods.

# User Documentation / Manual

This User Documentation / Manual will give step by step instructions on how to download / setup and run the Waitless Application.

## Python Installation

Python can be downloaded from the following location;

https://www.python.org/downloads/

This user manual assumes that any version of Python >= 3.6.5 is installed and available in the System.

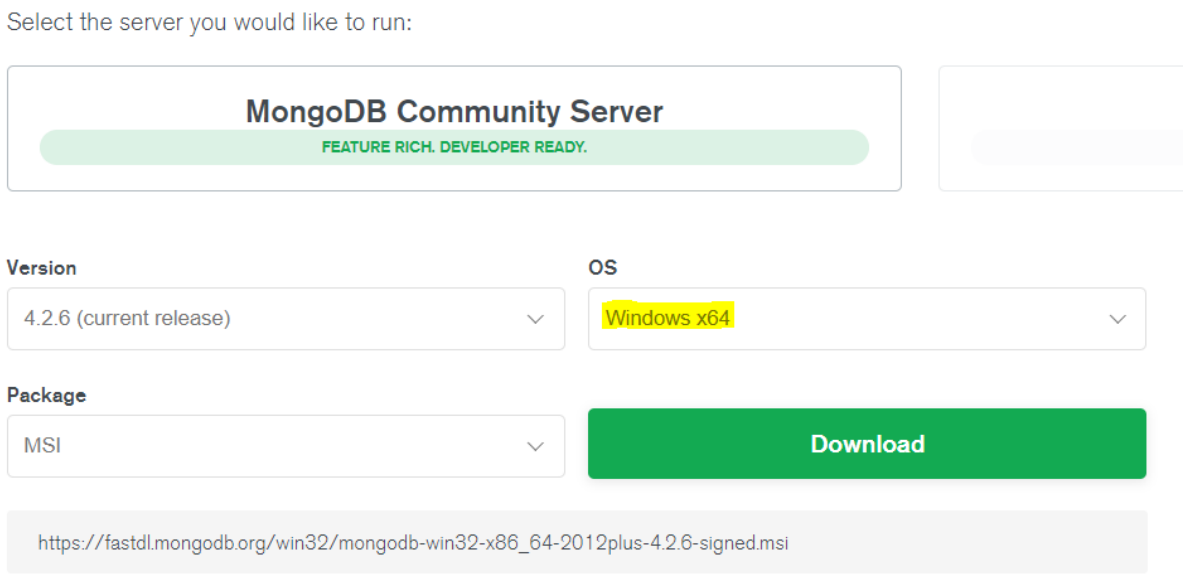
## MongoDB Community Server

### Download and Install

Download and Install MongoDB Community Server latest version (4.2.6 as of 28th April, 2020) from the following location

https://www.mongodb.com/download-center/community

Select the appropriate OS from the dropdown.




### Running MongoDB Server

The Server 'bin' folder will be automatically set in 'PATH' during installation. If not, open 'Command Prompt' (in Windows Operating System) and navigate to the folder where MongoDB community server is installed.

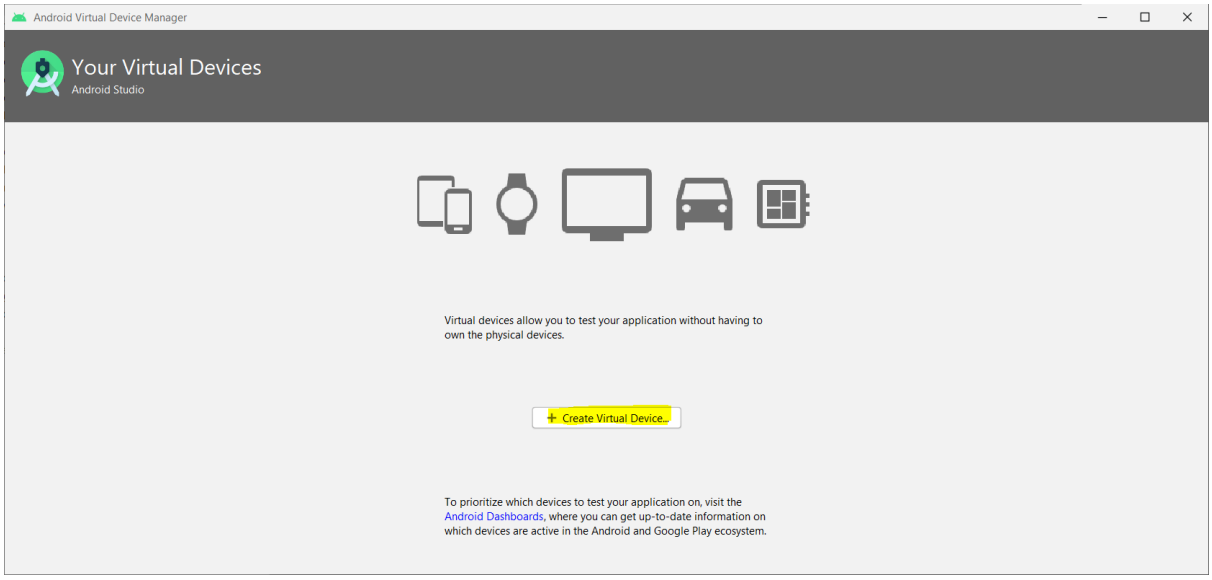
```
c:\Program Files\MongoDB\Server\4.2\bin>
```

Execute 'mongod.exe' which will start the MongoDB Server.

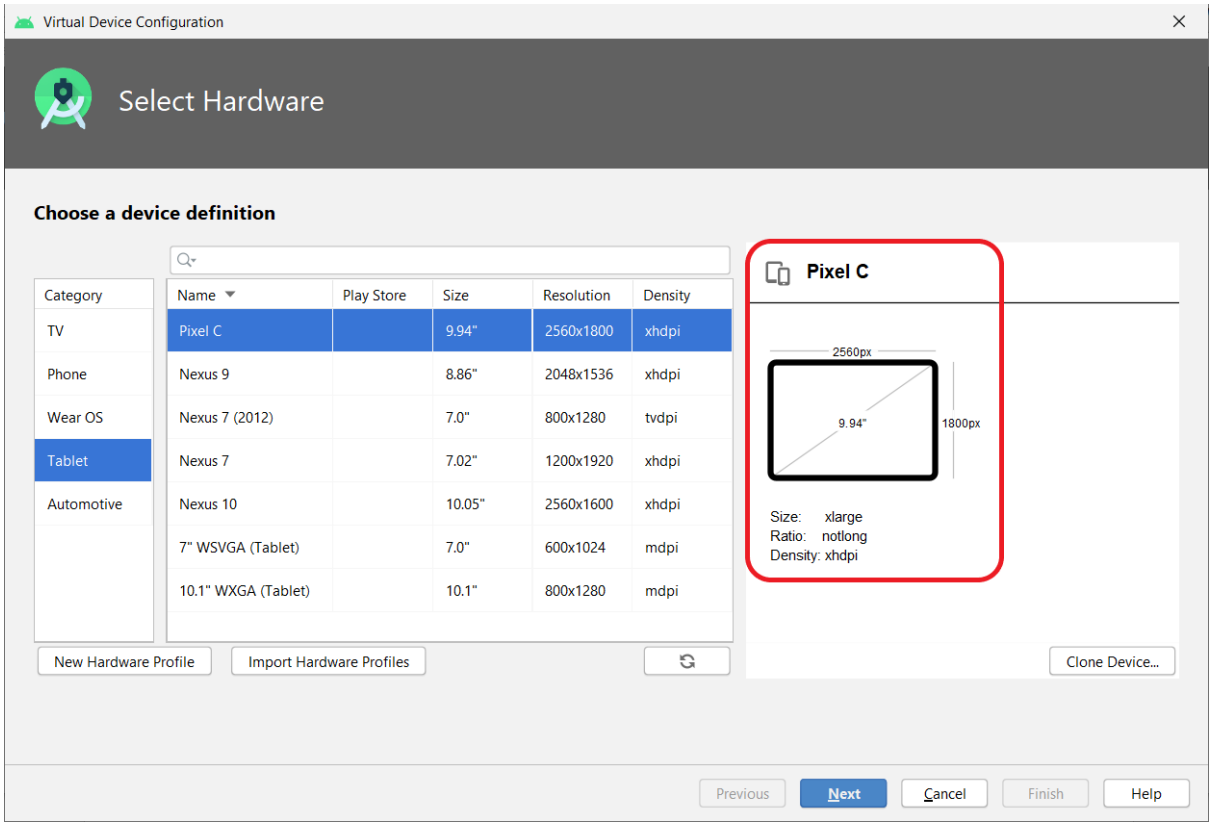
```
c:\Program Files\MongoDB\Server\4.2\bin>mongod.exe
```

The server will be running on localhost (127.0.0.1) and listening for incoming connections on port 27017.

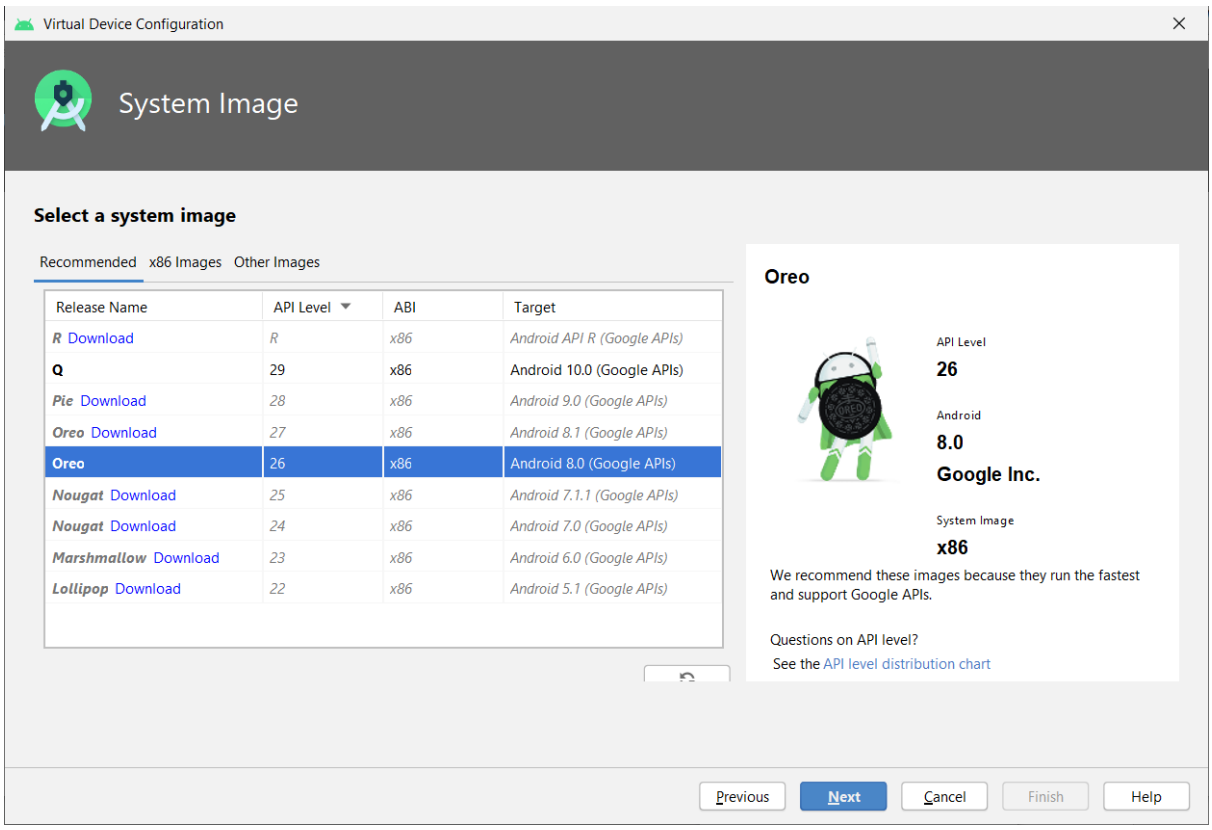
```
2020-04-28T12:35:32.794+1000 I  NETWORK  [listener] Listening on 127.0.0.1
2020-04-28T12:35:32.794+1000 I  NETWORK  [listener] waiting for connections on port 27017
```



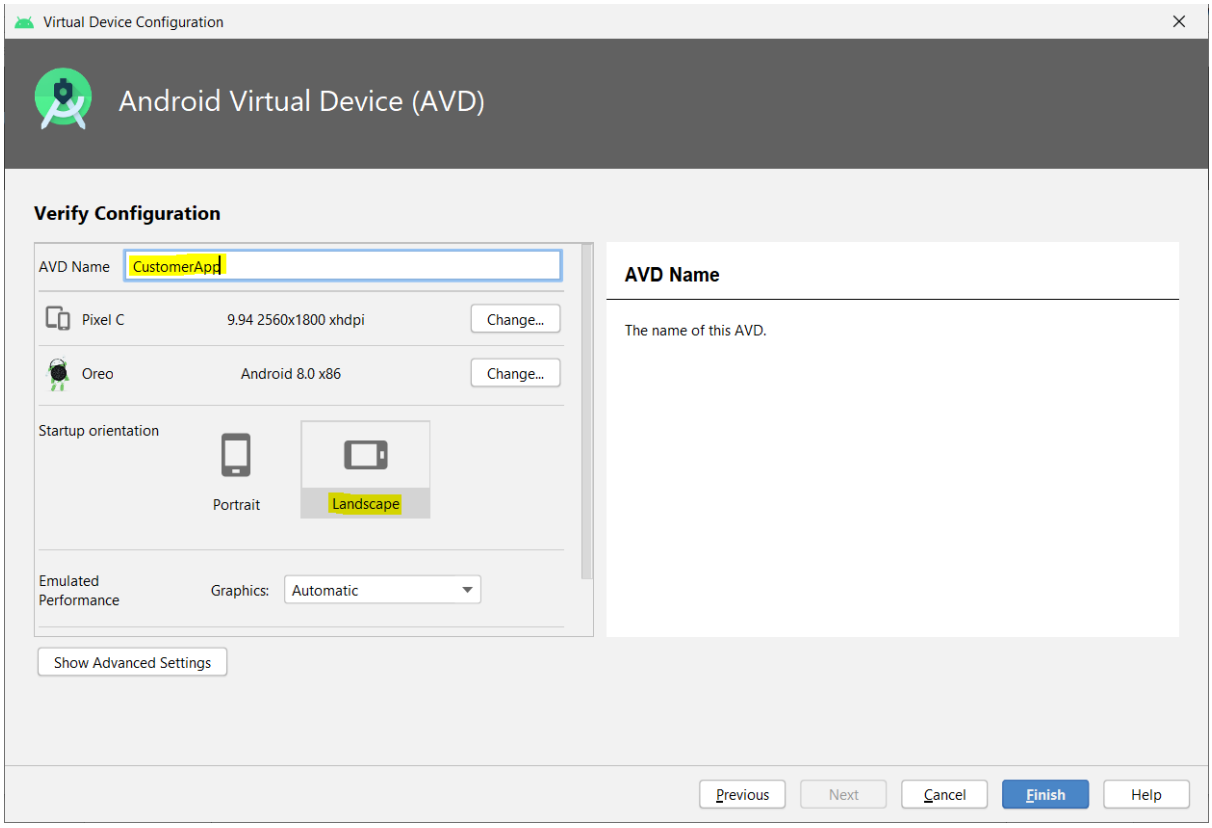

The Android Apps used in this Project are developed for a 'Pixel C' type of Tablet. It has the following configuration.



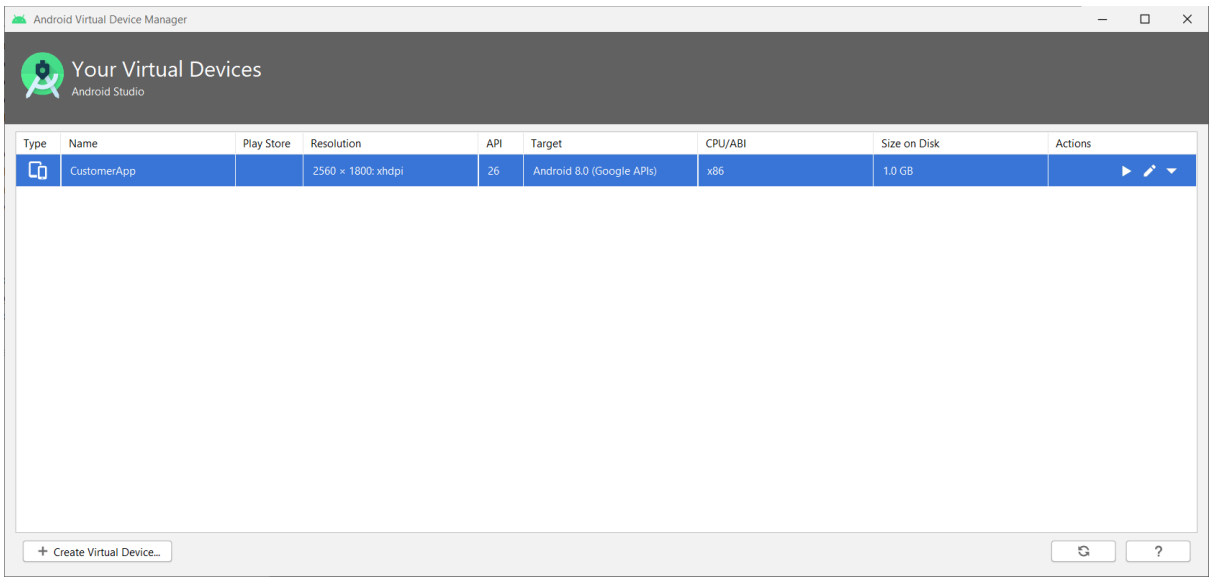

Click 'Next' and select 'Oreo' API Level 26.




Provide ‘CustomerApp’ as the AVD Name and set ‘Startup orientation’ as ‘Landscape’ and click ‘Finish’.

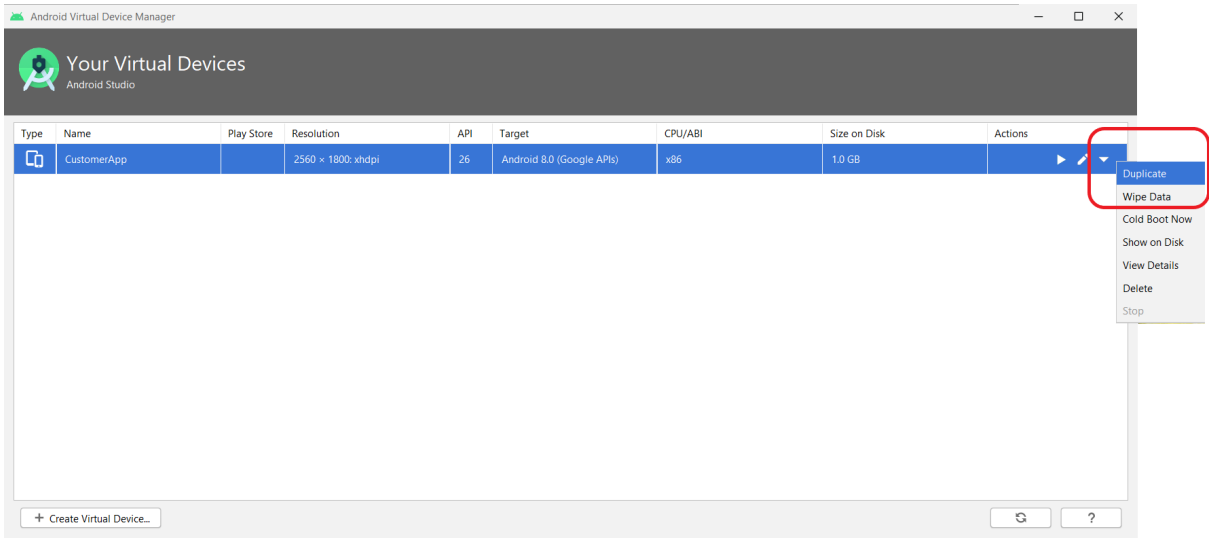


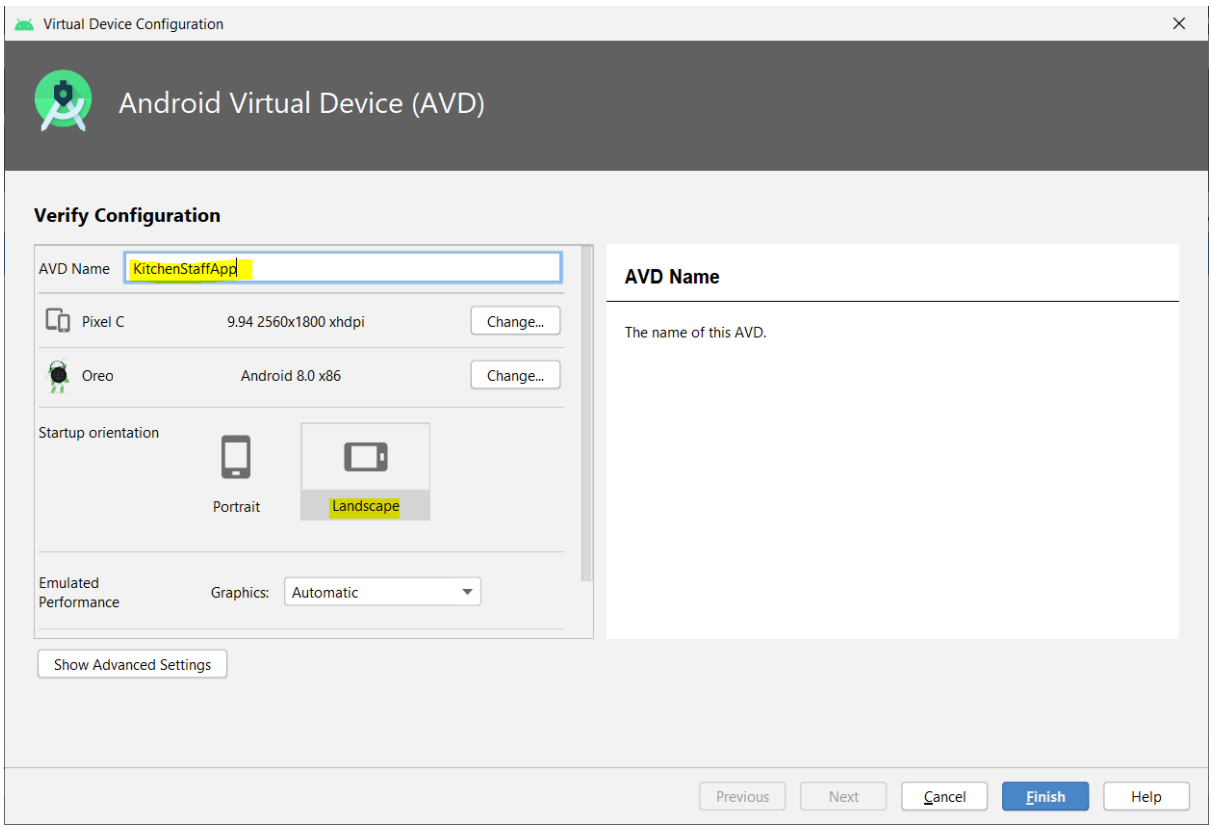

This will create the ‘CustomerApp’ AVD.

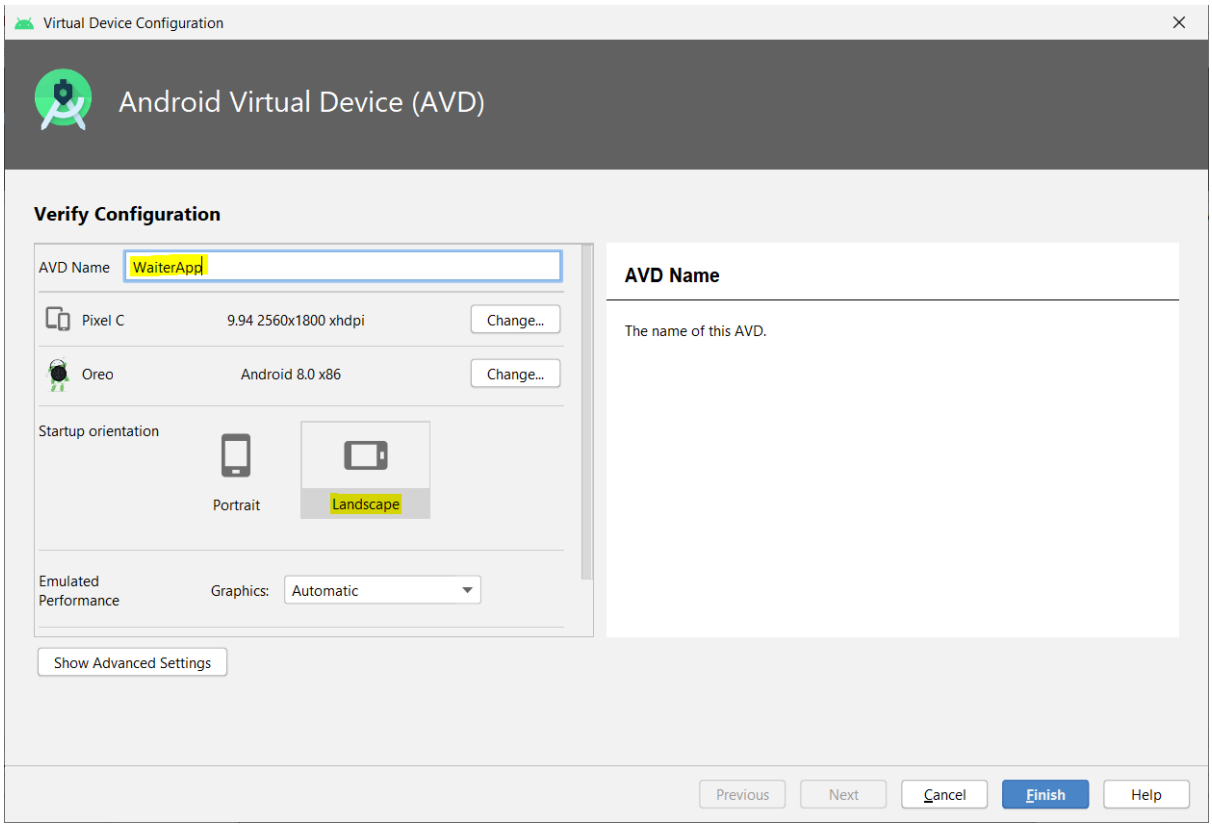


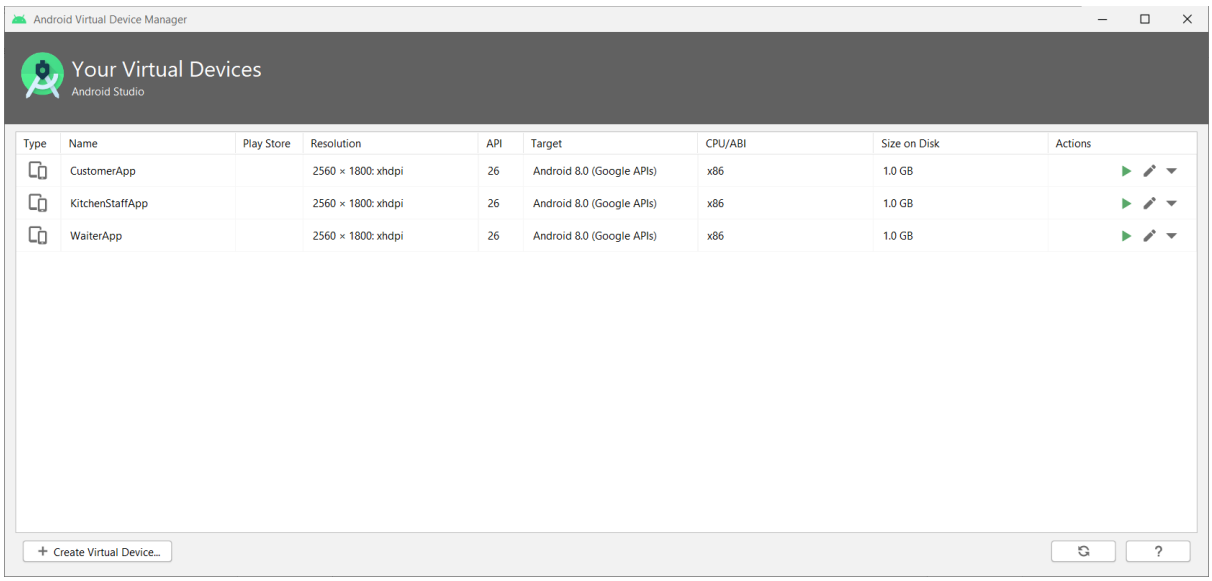

## Creating Duplicate AVDs

Now select ‘Duplicate’ to create Android Virtual Devices for ‘KitchenStaffApp’ and ‘WaiterApp’.



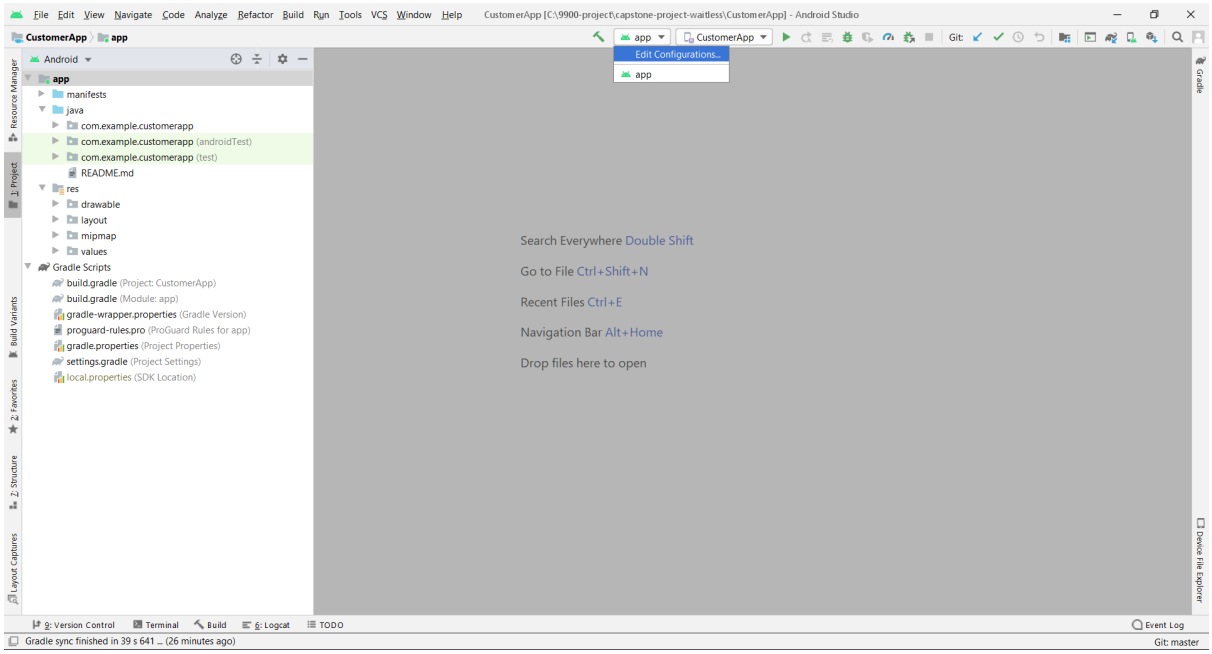

Virtual Device Configuration
Android Virtual Device (AVD)
Verify Configuration
AVD Name
KitchenStaffApp
Pixel C
9.94 2560x1800 xhdpi
Change...
Oreo
Android 8.0 x86
Change...
Startup orientation
Portrait
Landscape
Emulated Performance
Graphics:
Automatic
Show Advanced Settings
AVD Name
The name of this AVD.
Previous
Next
Cancel
Finish
Help


Virtual Device Configuration
Android Virtual Device (AVD)
Verify Configuration
AVD Name
WaiterApp
Pixel C
9.94 2560x1800 xhdpi
Change...
Oreo
Android 8.0 x86
Change...
Startup orientation
Portrait
Landscape
Emulated Performance
Graphics:
Automatic
Show Advanced Settings
AVD Name
The name of this AVD.
Previous
Next
Cancel
Finish
Help

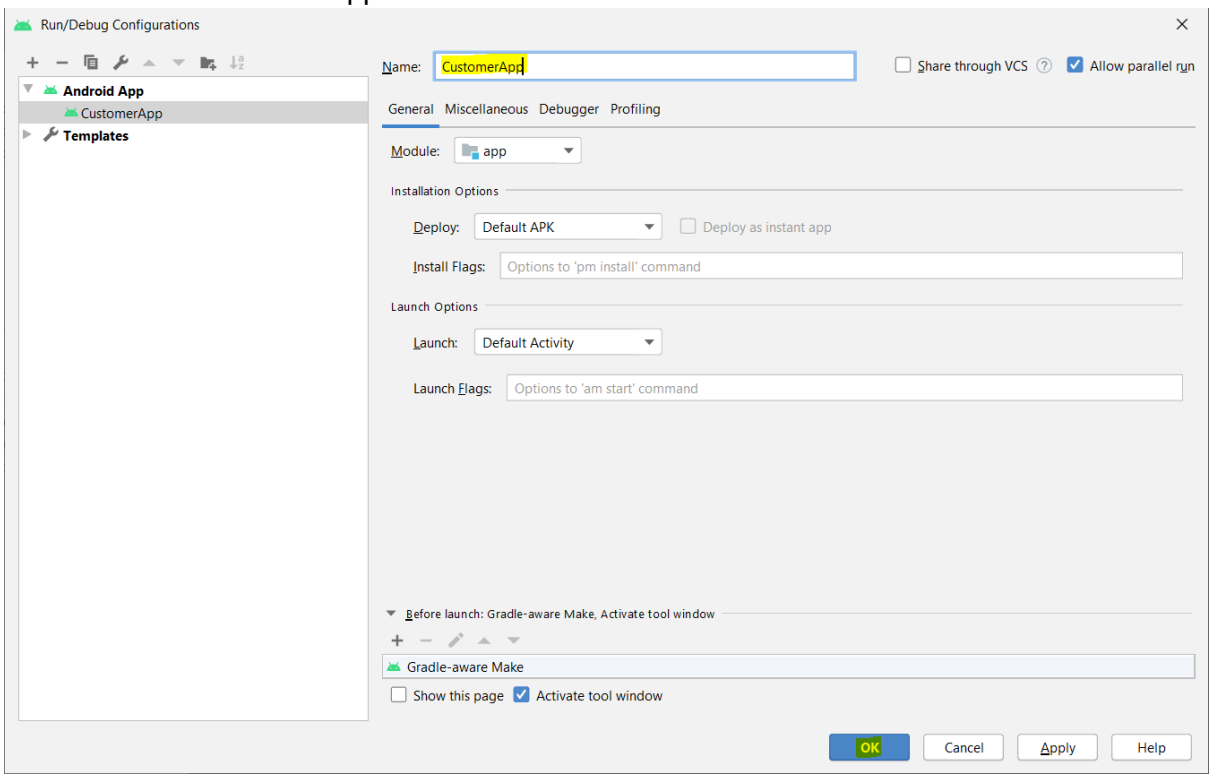



## Running CustomerApp Android App

Click on the dropdown next to 'Hammer' icon and select 'Edit Configurations…'



Give name as ‘CustomerApp’ and click ‘OK’.




Now the CustomerApp can be run in the ‘Pixel C’ android Tab emulator (nicknamed as ‘CustomerApp’) by clicking on the green ‘Play’ button.

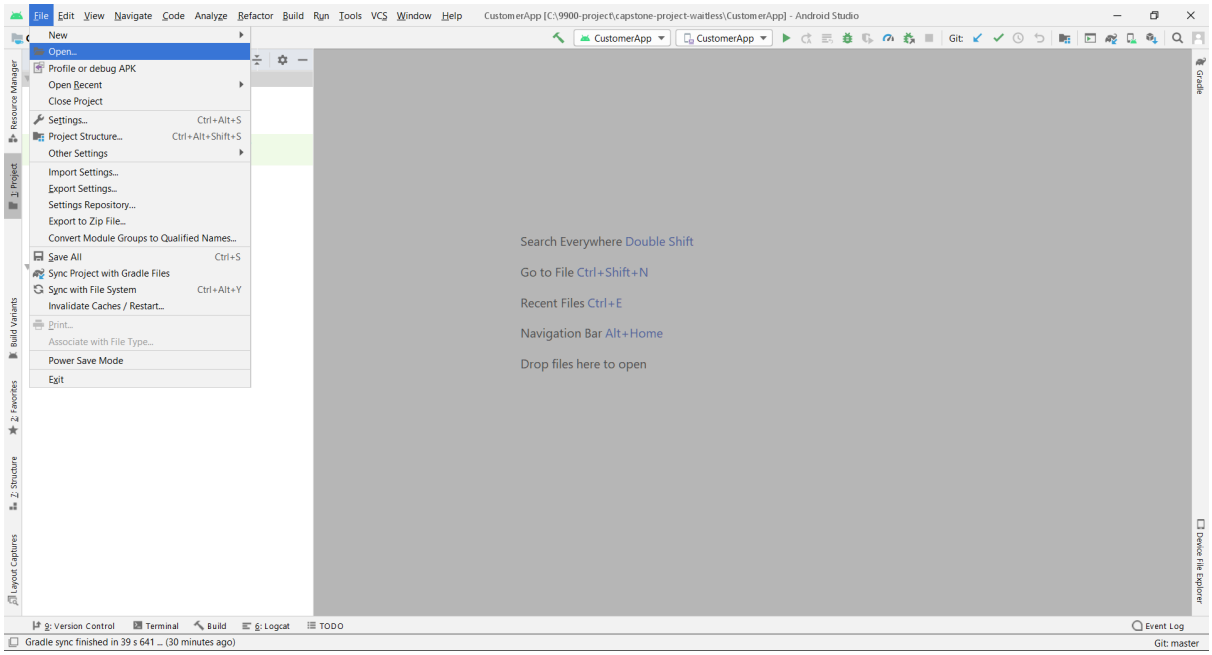

This will execute the ‘Gradle’ build and start the Emuator with the ‘CustomerApp’.

The details on how to use the application will be available in the ‘Functionalities and Implementation Challenges’ section.

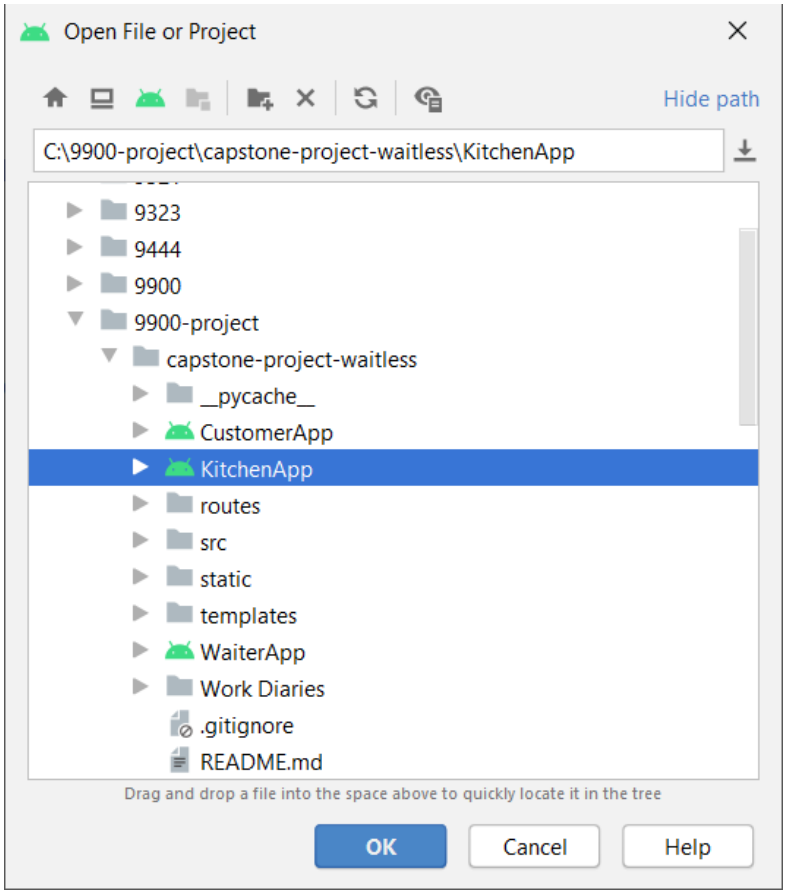
### Running KitchenStaff App and Waiter Apps

Similarly, to execute the Kitchen App and Waiter Apps the following steps are to be done;

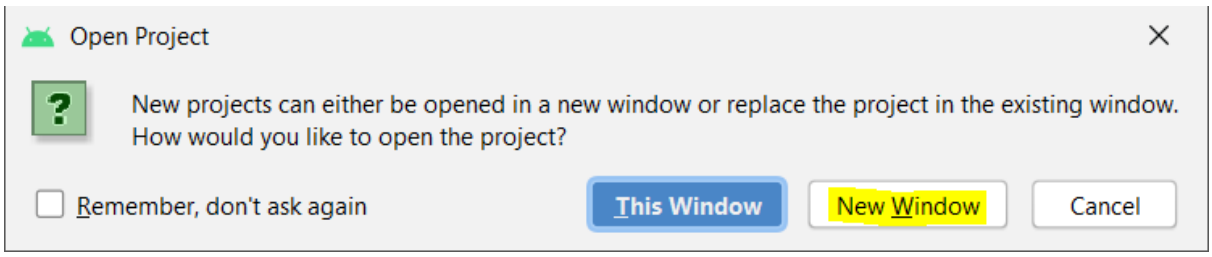
Click ‘Open’ from ‘File’ menu;




Select ‘KitchenApp’ and click ‘OK’

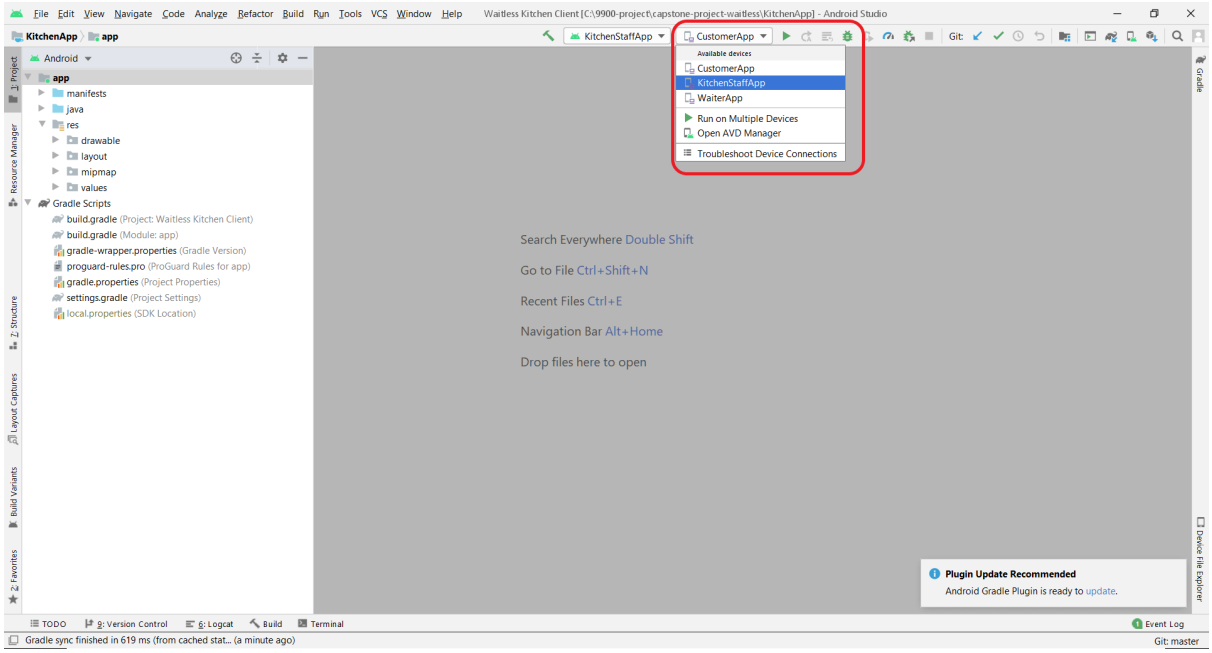



Select ‘New Window’.

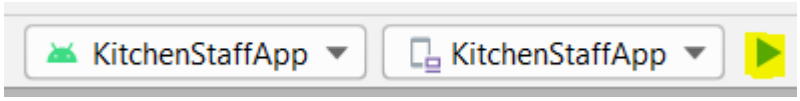



Click on the dropdown next to 'Hammer' icon and select 'Edit Configurations…' and give name as 'KitchenStaffApp' (like how it was done for CustomerApp).

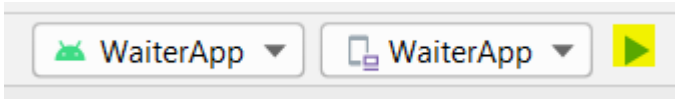
In the AVD dropdown, select 'KitchenStaffApp'.




Now, the KitchenStaff App can be executed on the Pixel C device emulator as shown below;




Following the exact similar steps will enable the WaiterApp to be run on the WaiterApp AVD.




The details on how to use the KitchenStaff and Watier Apps will be available in the 'Functionalities and Implementation Challenges' section.

This completes the Install / Setup and Running of all the different applications in this Project.