**BINDURA UNIVERSITY OF SCIENCE EDUCATION**

**FACULTY OF SCIENCE AND ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE**

****

**COMPUTER SCIENCE LEVEL 2.2 GROUP PROJECT**

**SYSTEM DOCUMENTATION**

**TOPIC:**

**MOBILE APPLICATION SYSTEM FOR BINDURA UNIVERSITY DINING**

System files: <https://github.com/carldevzw/four_meals_android.git>

**GROUP MEMBERS AND ROLES**

|  |  |  |
| --- | --- | --- |
| Melisa T. Kapimbi | B200772B | Team Leader, Testing, Research |
| Daniel C. Mubvumbi | B201086B | Development, UI/UX, Testing |
| Yolanda Mashlengwe | B201830B | Research, System Documentation |
| Sithabile Chikowero | B202223B | Research, System Documentation |

**SUBMITTED TO:**

**MR C. ZANO.**

**Table of Contents**

[**CHAPTER ONE: PROBLEM IDENTIFICATION** 6](#_Toc106274825)

[**1.1** **Introduction** 6](#_Toc106274826)

[**1.2** **The current system** 6](#_Toc106274827)

[**1.3** **Problem Statement** 6](#_Toc106274828)

[**1.4** **Proposed System** 7](#_Toc106274829)

[**1.5** **System Objectives** 7](#_Toc106274830)

[**1.6** **Limitations and Challenges** 7](#_Toc106274831)

[**1.7** **Scope of the system** 7](#_Toc106274832)

[**CHAPTER TWO: SYSTEM REQUIREMENTS** 9](#_Toc106274833)

[**2.1** **Introduction** 10](#_Toc106274835)

[**2.2** **Requirements Engineering** 10](#_Toc106274836)

[**2.2.1.1** **Elicitation** 10](#_Toc106274837)

[**2.2.1.2** **Functional Requirements** 10](#_Toc106274838)

[**2.2.1.3** **Non-Functional Requirements** 10](#_Toc106274839)

[**2.2.1.4** **User Software Requirements** 10](#_Toc106274840)

[**2.3** **System Requirements** 10](#_Toc106274841)

[**2.3.1** **Inputs** 10](#_Toc106274842)

[**2.3.2** **Processes** 10](#_Toc106274843)

[**2.3.3** **Outputs** 10](#_Toc106274844)

[**CHAPTER THREE: SYSTEM DESIGN** 9](#_Toc106274845)

[**3.1** **Introduction** 9](#_Toc106274847)

[**3.2** **User Interface Design Snapshots** 9](#_Toc106274848)

[**3.3** **Process Diagrams** 12](#_Toc106274849)

[**3.3.1** **Process Flowchart** 12](#_Toc106274850)

[**3.3.2** **Use Case Diagram** 13](#_Toc106274851)

[**3.3.3** **Database Design** 14](#_Toc106274852)

[**3.3.4** **Key Capabilities** 14](#_Toc106274853)

[**CHAPTER FOUR: CODING AND TESTING** 16](#_Toc106274854)

[**4.1** **Introduction** 16](#_Toc106274856)

[**4.2** **System Code** 16](#_Toc106274857)

[**4.3** **Testing** 19](#_Toc106274858)

[**4.3.1** **Unit Testing** 19](#_Toc106274859)

[**4.3.2** **Integration Testing** 20](#_Toc106274860)

[**4.3.3** **User Validation** 20](#_Toc106274861)

[**CHAPTER FIVE: CLOSURE AND ANALYSIS** 21](#_Toc106274862)

[**5.1** **Introduction** 21](#_Toc106274864)

[**5.2** **Challenges** 21](#_Toc106274865)

[**5.3** **Recommendations** 21](#_Toc106274866)

[**5.4** **Conclusion** 21](#_Toc106274867)

# **CHAPTER ONE: PROBLEM IDENTIFICATION**

## **Introduction**

The advent of technology brought with it a much conspicuous array of possibilities that would have not yet been explored. Mobile technology solved what was considered human’s most difficult tasks which include hardships in communication and information access amongst many other problems. It goes without saying how humans are dependent on mobile technology nowadays, in particular, how mobile systems have become an essential part of our daily lives. It is on this very premise that we embark on this journey in an attempt to take full advantage of the tools that are at our disposal and further address the challenges that students and university staff face in terms of service delivery in the school dining hall.

## **The current system**

Being a group of students who have required service of the school’s dining hall has made it relatively unchallenging for us to understand the intricacies of the system, from the student’s perspective at least. However, we still carried out a series of investigations that would help us further understand the system from both the student’s and the administration’s perspectives. We surveyed both the university residences and interviewed several administration staff who directly operate the dining halls and discovered that the current status quo at the residence is centred around a much inefficient system.

We discovered that:

1. At mealtime, students head to the dining hall where they queue up, pay or confirm payment, and their food is served.
2. Data capturing and handling is done manually, a tedious task that involves tonnes of physical pages and effort.
3. Students have no prior knowledge of the day’s meals.
4. The dining staff has no idea how many students they will be serving for the day and hence no idea how much food to prepare.
5. Meals are served on a first-come-first-serve basis.

## **Problem Statement**

Students are not at liberty to choose what it is that they want to eat before mealtime. The time and effort used by the staff to authenticate students’ details are a lot and make them inefficient at their jobs as evidenced by the average time it takes for students to pay for their food and get served.

The dining staff have no idea how much food to prepare per day or per meal, cases were reported where more food than required was prepared and a lot would go wasted.

The use of paperwork (manual) to keep the records has no place in the modern-day world where technology and databases have advanced so much. This took a lot of effort that could otherwise be dedicated to more deserving tasks.

## **Proposed System**

A Mobile Application System aimed at improving efficiency and bringing convenience to all the parties involved, that is, the dining administration and the student body. The application is going to assist and automate the day-to-day routine of dining services at the school, the services include how the food will be served, and what type of foods to be served.

## **System Objectives**

1. To allow the students to make, view, and make changes to their orders before submitting
2. To provide an interface that allows students to see the day’s menu.
3. To provide an interface that shows students’ order details to the dining staff.
4. To deliver a system that allows the management to modify the menu prices and add new meals.

## **Limitations and Challenges**

The mobile application system will improve efficiency and also bring convenience to all the operations at the university dining Hall. However, the system also has limitations, these include:

1. The application will be native, that is, can be used by android users only.
2. Requirement of internet access to make orders and get updates on meal status.
3. Inability to reverse order when it has been confirmed by the student.
4. Not having a way of tracking if all orders have been collected by the students who made them.

## **Scope of the system**

Although the project covers the boundary of developing an android application to be used by students and staff from the Bindura University of Science Education, we shall develop only a prototype as a proof of concept for the Faculty of Science Education Complex Dining Hall. This is on the basis that if the system works for one dining hall, it can easily work for all dining halls at the university because the problems we intend to address herein are universal to all the dining halls.

# **CHAPTER TWO: SYSTEM REQUIREMENTS**

## 



## **Introduction**

In this chapter, we shall fully elaborate on the set of all requirements that are to be imposed on the design and verification of the application. Both the functional and non-functional requirements will be briefed herein.

## **Requirements Engineering**

### **Elicitation**

In order to get data and facts to be able to come up with the scope management plan, the following techniques were applied.

1. **Questi**o**nnaires**: We distributed questionnaires to a number of students residing at the school hostels to gather precise information on service provision in the school’s dining halls.
2. **Interviews**: We carried out a number of in-person interviews with the dining staff at all levels, from management all the way to the cooks.
3. **Observation:** Being a group of students made it relatively easy for us to be able to get first-hand experience of the former system and gather an understanding of what exactly is required.

We then came up with the following lists of requirements for the system.

### **Functional Requirements**

1. Register a new user, that is, the student.
2. Allow login to registered users.
3. Show the meals available for the day to the student.
4. Provide the users with the type of food served and its price tag.
5. Allow students to make orders.
6. It should allow DH admins to log in as admins and update meals.
7. Give feedback to the users when the order is ready.
8. It should give the DH admins the ability to see the number of orders per meal.

### **Non-Functional Requirements**

1. Intuitive, attractive, and functional user interface design.
2. Secure user information architecture.
3. Efficiency and throughput.
4. Fast.
5. Functional.

### **User Software Requirements**

The final application will run on mobile devices with the **Android Operating System** API version 21 or better.

## **System Requirements**

### **Inputs**

The system shall require user input through forms and touch gestures. Below is a generic list of some of the input to be collected.

1. User details (names, password).
2. Administrator inputs. (Images, meal details)

### **Processes**

1. Username and password validation.
2. Capturing user data.
3. Updating user and meal data in the database

### **Outputs**

1. View of the popular meals and meal suggestions.
2. View of the available menu.
3. View of the selected orders and a confirmation button to confirm orders.
4. View of the number of orders made per meal.
5. View of orders and their ready status.

# **CHAPTER THREE: SYSTEM DESIGN**



## **Introduction**

The systems design phase focuses on the development of the objectives of the proposed system and it outlines how the proposed system is going to be developed, configured, and deployed.

## **User Interface Design Snapshots**

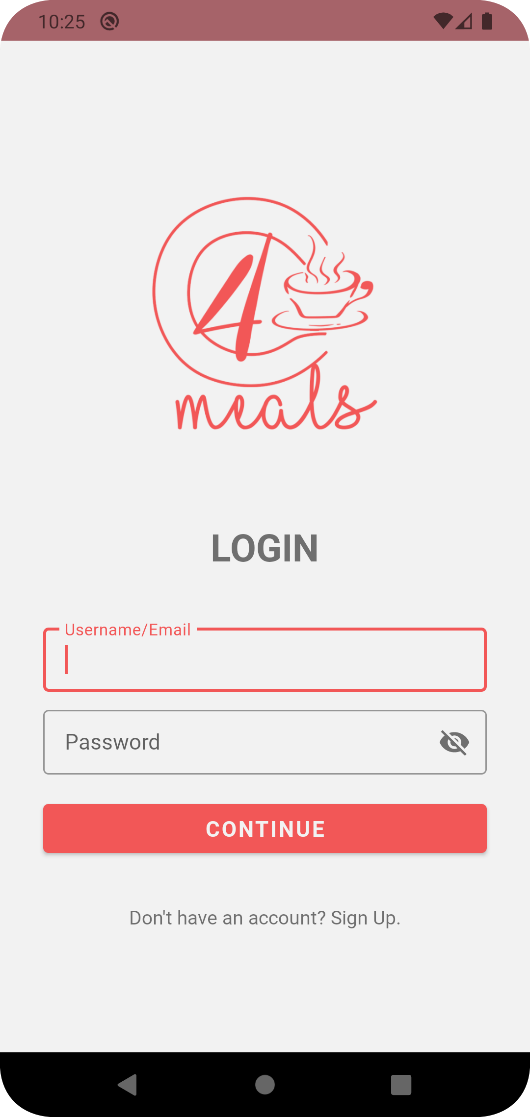
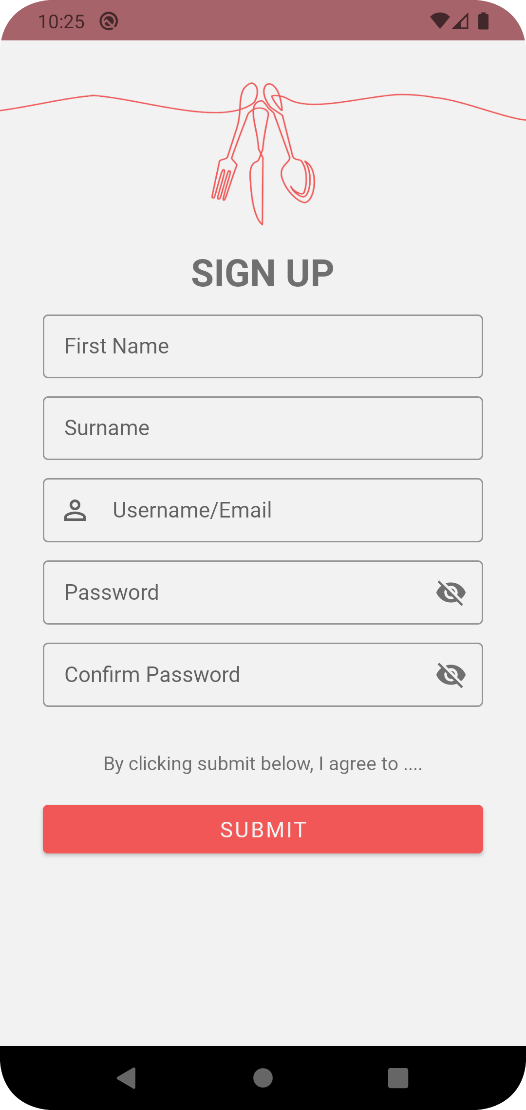


Figure 2 Signup

Figure 1 Login Design

If it is the first time using the application the user is required to Sign up as shown in Figure 2. Figure 1 is where the user will log in if they are signed up.

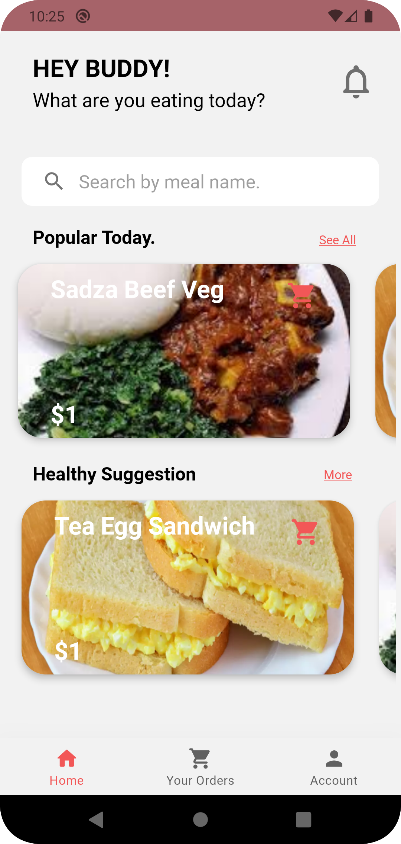
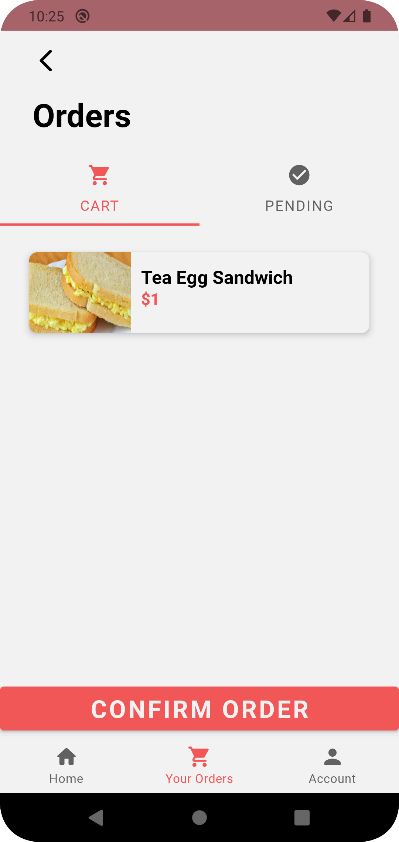
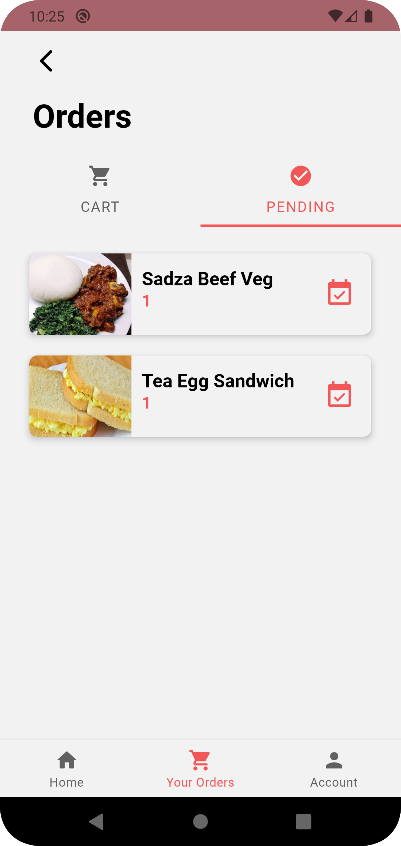
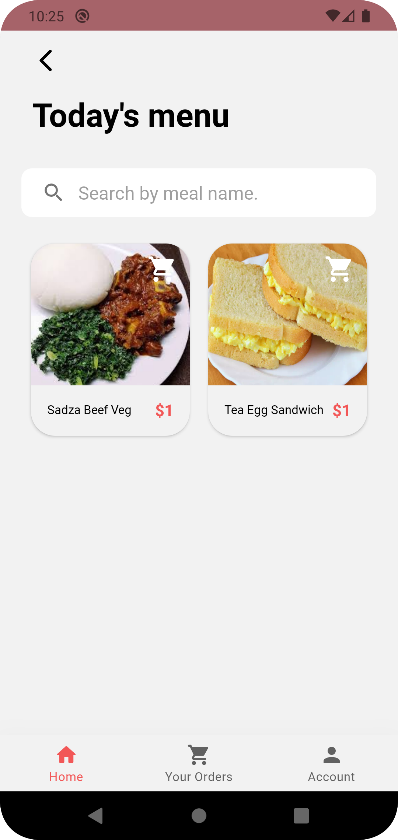
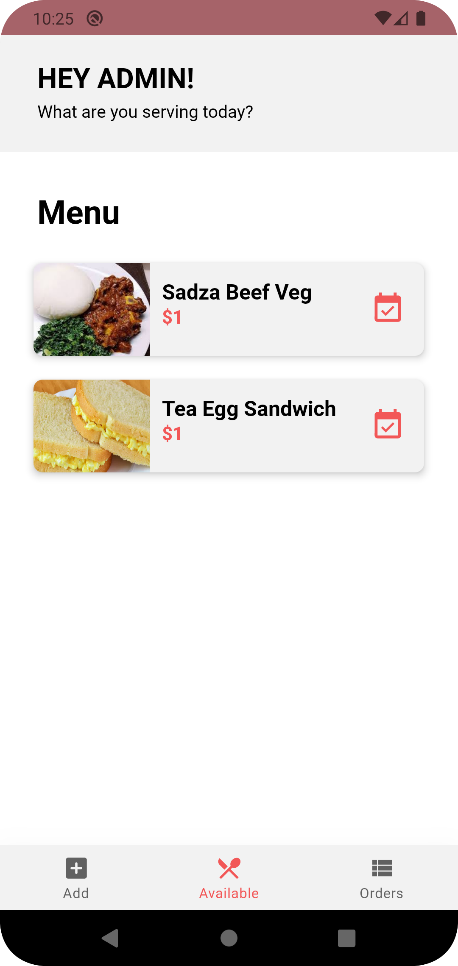
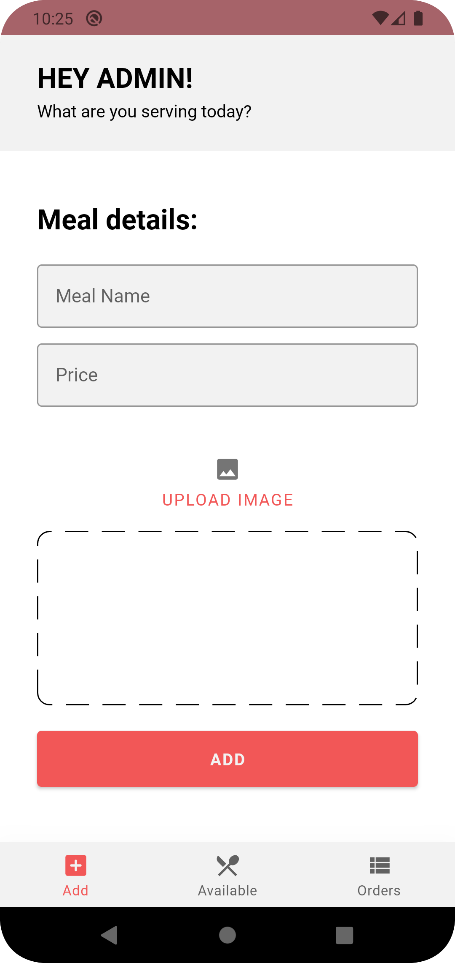
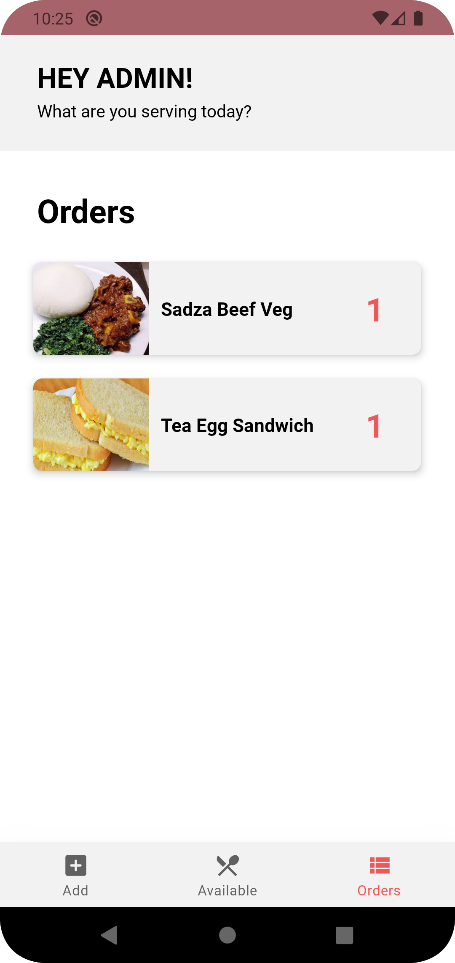
 ****

Figure 8 Confirmed Meals

Figure 6 Dashboard

Figure 7 Meals List

Figure 9 Cart

Figure 4 Available Meals

Figure 5 Orders

Figure 3 Add Meals

## **Process Diagrams**

### **Process Flowchart**

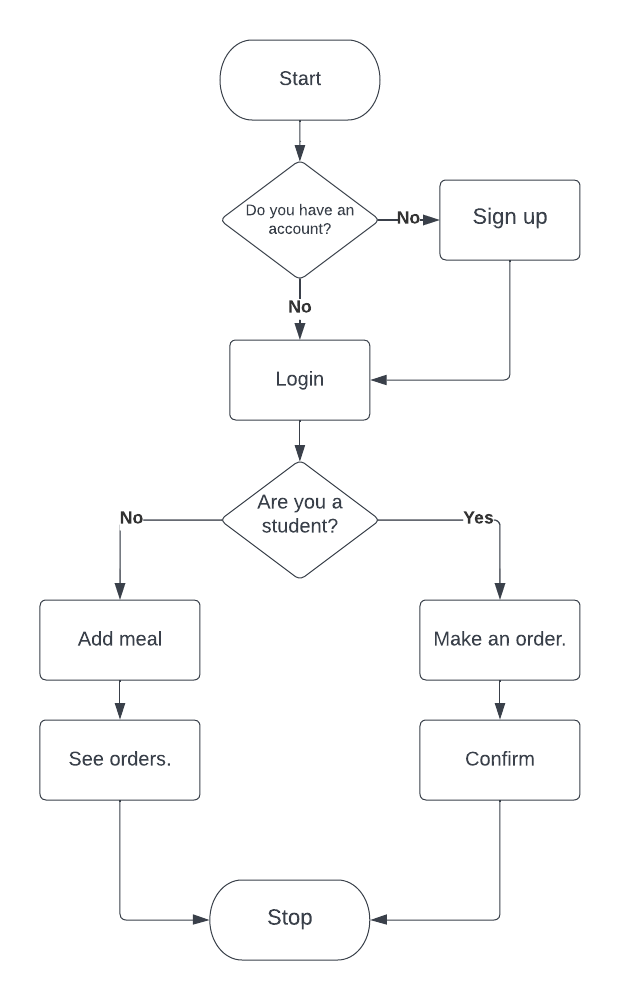


Figure 10 Flowchart showing the flow of events within the app

### **Use Case Diagram**

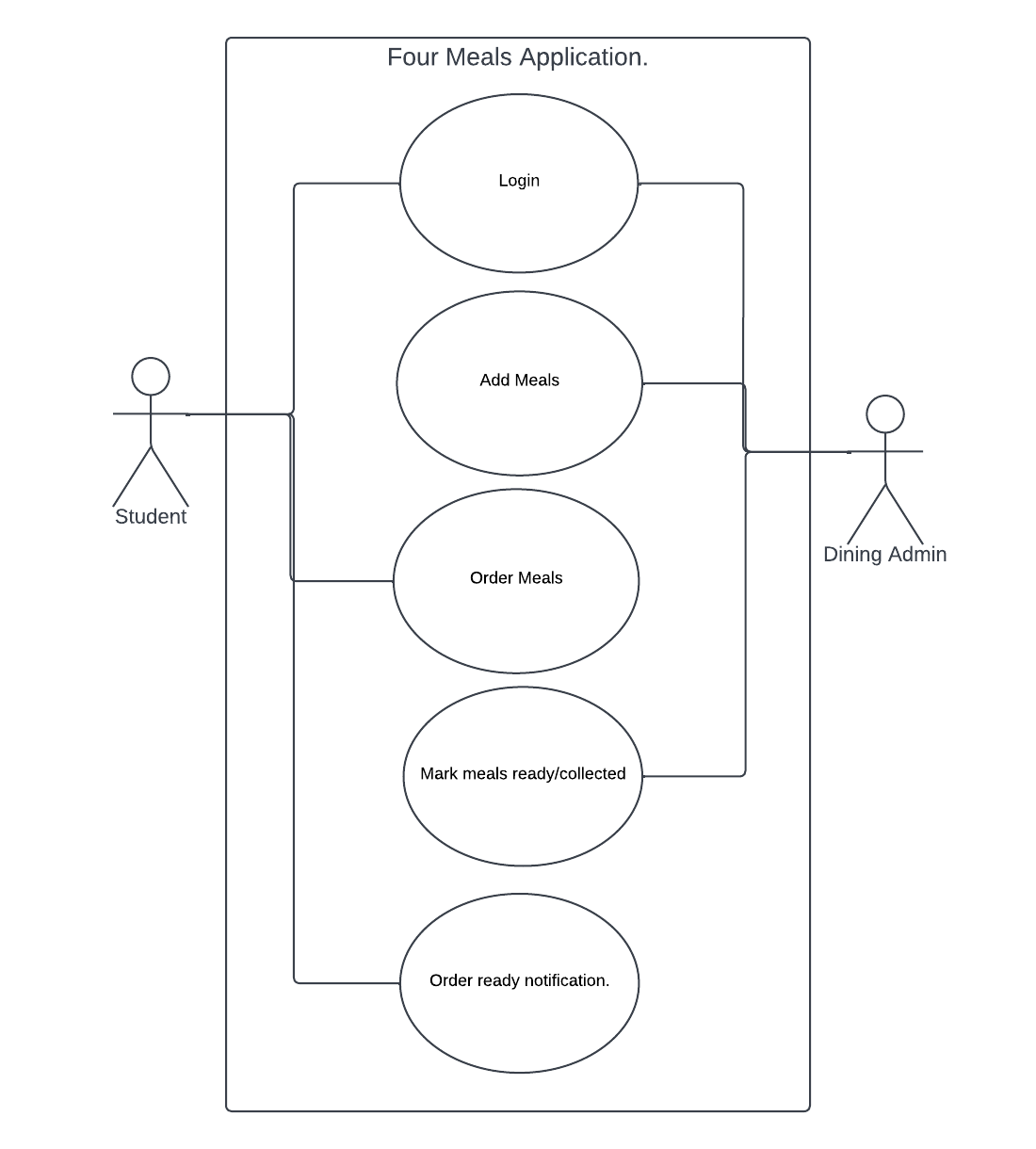


Figure 11 Use Case showing the various activities and their

interaction with the user.

### **Database Design**

The application will use the Firebase Cloud Firestore Database, a cloud database system, whose dynamics make it easy for the end-users to access it via the internet. This means that the user can expect high availability with maximal SLA compliance.

Following Cloud Firestore's NoSQL data model, data is stored in documents that contain fields mapping to values. These documents are stored in collections, which are containers for your documents that you can use to organize your data and build queries. Documents support many different data types, from simple strings and numbers to complex, nested objects. You can also create subcollections within documents and build hierarchical data structures that scale as your database grows. The Cloud Firestore data model supports whatever data structure works best for your app.

### **Key Capabilities**

1. **Flexibility.**

The Cloud Firestore data model supports flexible, hierarchical data structures. Store your data in documents, organized into collections. Documents can contain complex nested objects in addition to subcollections.

1. **Expressive querying.**

In Cloud Firestore, you can use queries to retrieve individual, specific documents or to retrieve all the documents in a collection that match your query parameters. Your queries can include multiple, chained filters and combine filtering and sorting. They're also indexed by default, so query performance is proportional to the size of your result set, not your data set.

1. **Offline support**

Cloud Firestore caches data that your app is actively using, so the app can write, read, listen to, and query data even if the device is offline. When the device comes back online, Cloud Firestore synchronizes any local changes back to Cloud Firestore.

Below is a snapshot of the database with the relevant collections containing the various documents.

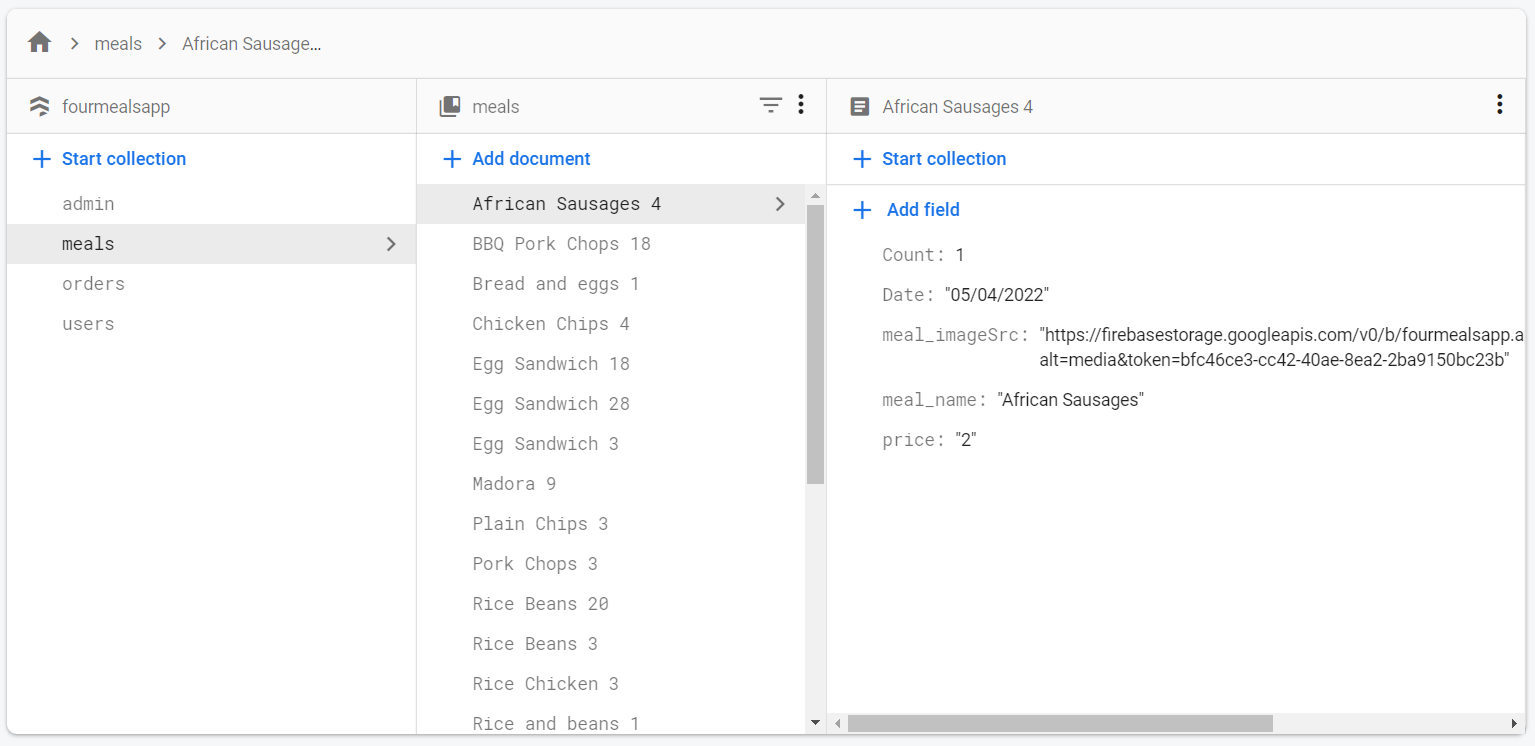
****

Figure 12 Snapshot of the app's database structure.

­

# **CHAPTER FOUR: CODING AND TESTING**



## **Introduction**

We developed the application using the JAVA programming language and the Android Studio IDE. Below are the main code snippets with a brief explanation of the functionalities. Most of the application code will be delivered through the system files where further analysis can be done.

## **System Code**

public class Meal\_Model {  
  
 private String meal\_name;  
 private String price;  
 private String meal\_imageSrc;  
 private String documentID;  
 private int Count;  
 boolean Ready;  
  
 public Meal\_Model(String meal\_name, String price, String meal\_imageSrc, int Count, boolean Ready) {  
 this.meal\_name = meal\_name;  
 this.price = price;  
 this.meal\_imageSrc = meal\_imageSrc;  
 this.Count= Count;  
 this.Ready= Ready;  
 }  
 public Meal\_Model(){  
  
 }  
 public boolean isReady() {  
 return Ready;  
 }  
 public String getDocumentID() {  
 return documentID;  
 }  
 public void setDocumentID(String documentID) {  
 this.documentID = documentID;  
 }  
 public String getMeal\_name() {  
 return meal\_name;  
 }  
 public String getPrice() {  
 return price;  
 }  
 public int getCount() {  
 return Count;  
 }  
 public String getMeal\_imageSrc() {  
 return meal\_imageSrc;  
 }  
  
}

The above code snippet is a custom class representing a meal. Each meal has various attributes like the meal name, price, image source, and because a meal is basically a database document, we also have a document identification number that is unique to each meal. We also have two sets of constructors, an empty one and another one that accepts input for all the meal details. Inside the class, we have some getters and setters for some of the attributes of the class.

The Meal\_Model class is the basic building block and the most significant class driving the whole application. From fetching the meals from the database to feeding the user interface, it is this class the application relies on.

public class Meal\_Orders\_Adapter extends RecyclerView.Adapter<OrdersViewholder> {  
  
 private static final String *TAG* = "Meal\_Orders\_Adapter";  
 private Context context;  
 private ArrayList<Meal\_Model> Meal\_ModelArrayList;  
 FirebaseFirestore db;  
  
 public Meal\_Orders\_Adapter(Context context, ArrayList<Meal\_Model> meal\_ModelArrayList) {  
 this.context = context;  
 Meal\_ModelArrayList = meal\_ModelArrayList;  
 }  
  
 public void deleteFromCart(int position){  
 Meal\_Model model = Meal\_ModelArrayList.get(position);  
 db= FirebaseFirestore.*getInstance*();  
 FirebaseUser firebaseUser= FirebaseAuth.*getInstance*().getCurrentUser();  
 String userId= firebaseUser.getUid();  
   
 CollectionReference collectionReference= db.collection("users").document(userId).collection("orders");  
 DocumentReference documentReference= db.collection("users").document(userId).collection("orders").document(model.getDocumentID());  
  
 documentReference.delete()  
 .addOnCompleteListener(new OnCompleteListener<Void>() {  
 @Override  
 public void onComplete(@NonNull Task<Void> task) {  
 if(task.isSuccessful()){  
 Log.*d*(*TAG*, "onComplete: Order removed from database.");  
 }  
 }  
 });  
 }  
 @NonNull  
 @Override  
 public OrdersViewholder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
 View view = LayoutInflater.*from*(parent.getContext()).inflate(R.layout.*order\_item*, parent, false);  
 return new OrdersViewholder(view);  
 }  
 @Override  
 public void onBindViewHolder(@NonNull OrdersViewholder holder, int position) {  
  
 Meal\_Model model = Meal\_ModelArrayList.get(position);  
 holder.orderMealNameTV.setText(model.getMeal\_name());  
 holder.orderMealPriceTV.setText("$" + model.getPrice());  
 Glide.*with*(context)  
 .load(model.getMeal\_imageSrc())  
 .placeholder(R.drawable.*ic\_baseline\_no\_food\_24*)  
 .centerCrop()  
 .into(holder.orderMealIV);  
 }  
 @Override  
 public int getItemCount() {  
 return Meal\_ModelArrayList.size();  
 }  
}

Above is an adapter class. Adapters are abundant in the whole code. They play a very crucial role in the functionality of the application. All adapters in the application have the same basic structure as the one above. Adapters’ purpose is to populate the system user interface with information that is dynamically loaded from the database. The adapter class overrides many functions of the Recycler View Adapter class which it inherits from. Needless to say, it is a child of a parent class as explained. The View holder class which is instantiated in the onBindViewHolder method is the container for the application’s user interface item that is to be populated with this particular adapter.

public void listMeals(){  
 Date date = new Date();  
 DateFormat dateFormat = new SimpleDateFormat("dd/MM/yyyy");  
  
 // Choose time zone in which you want to interpret your Date  
 Calendar cal = Calendar.*getInstance*(TimeZone.*getTimeZone*("Central Africa"));  
 cal.setTime(date);  
 String todaysDate = dateFormat.format(date);  
  
 mealModelArrayList = new ArrayList<>();  
  
 mealRV= findViewById(R.id.*RVMealItems*);  
 mealRV.setHasFixedSize(true);  
  
 GridLayoutManager layoutManager=new GridLayoutManager(this,2);  
  
 mealRV.setLayoutManager(layoutManager);  
  
 db= FirebaseFirestore.*getInstance*();  
  
 db.collection("meals")  
 .whereEqualTo("Date", todaysDate)  
 .orderBy("meal\_name", Query.Direction.*ASCENDING*)  
 .addSnapshotListener(new EventListener<QuerySnapshot>() {  
 @Override  
 public void onEvent(@Nullable QuerySnapshot value, @Nullable FirebaseFirestoreException error) {  
  
 if(error != null){  
 Log.*d*(*TAG*, "Snapshot may have returned null.");  
 }else {  
 assert value != null;  
 for(DocumentChange dc: value.getDocumentChanges()){  
 if(dc.getType()== DocumentChange.Type.*ADDED*){  
 mealModelArrayList.add(dc.getDocument().toObject(Meal\_Model.class));  
 }  
 Meal\_Adapter mealAdapter = new Meal\_Adapter(MainActivity.this,mealModelArrayList);  
 mealRV.setAdapter(mealAdapter);  
 Log.*i*(*TAG*, "Snapshot Updated.");  
 mealAdapter.notifyDataSetChanged();  
  
 }  
 }  
 }  
 });  
}

The function above is an extract of the main list of meals that the user interacts with. The method fetches meals from the database and feeds them to the adapters in form of the meal model class as described earlier. Since there are probably the rarest cases where only one meal will be served, the meals have to be loaded into a list of some sort, in this case, an array list, which the team chose after careful consideration of other list types. The array list accepts elements of type Meal\_Model (explained above) and plugs them into the adapter which then populates the user interface. The application retrieves meals from the database with the help of some classes from the firebase dependency. These bring with them some very powerful querying abilities such as selecting meals posted on a certain date only. This feature is actually used throughout the whole application for users to be able to see only the meals for that particular date.

## **Testing**

### **Unit Testing**

As described above, the application architecture consists of a few base classes that drive most of the application functionality. It is these classes that we focused on during our unit test as we tested the various objects at different application levels and modules that are created from these base custom classes.

During the testing of these base modules, we fed different allowable input to the modules and we monitored the output. All the modules were working properly as expected from uploading data to the database and then retrieving the data into all our user interfaces. Needless to say, the unit test was indeed a successful one.

### **Integration Testing**

Our team carried out further testing with all the components combined. Again, it was a successful integration test as we observed that all the modules were responding as expected to the actions performed in another activity if they have to. For example, student interfaces were updated in almost an instance every time a new meal was added to the database from the admin interface and admins were able to see in an instance a list of updated meal orders that were made in the student interfaces.

### **User Validation**

The application was then tested with some real users. We installed the application on a number of students’ smartphones where we ran a virtual dining hall that simulated the school’s dining hall. Our development team would act as the admins, uploading meals for every meal time and students would then make preferred orders and wait for the orders to be ready. All was going well, all features working well, except we had a few suggested features that could further improve our application system. These will be talked about in Chapter Five.

# **CHAPTER FIVE: CLOSURE AND ANALYSIS**



## **Introduction**

The application’s development was a journey the team takes much pride in. Herein, we shall brief our experiences and any further ideas to improve the application.

## **Challenges**

1. Time constraints was a difficulty to deal with. It proved almost impossible to stay on schedule. We discovered that this was mostly due to a lack of experience on the development team’s end.
2. Technical challenges in software and hardware also proved challenging. Several times, we had to depend on our previous iterations of the system, thanks to version control. The whole team was depending on one computer which was hardly capable to keep up with the resource-thirsty Android Studio IDE.

## **Recommendations**

To run the system more efficiently and without any delays or interruptions, it is recommended that one use a consistent and uninterruptible internet supply. It is known that for any meaningful computer-based system to be integrated into any organization, proper training and orientation have to be given both to the staff and the students. The staff should also be highlighted the need and advantages of the system and how it will equally assist them in their various field of work. They should also be informed of the cost of maintaining this new system so that they will handle it with all carefulness.

To further improve the application, we recommend some machine learning integration to help monitor the dining menu and help improve the student’s diet. More so, as this is just a prototype, the application should further be developed to cater to all campuses. It would be amazing during the maintenance to then introduce some sort of payment gateway in the application. Every order will then be accompanied by payment. Furthermore, the dining administration should be given the ability to place a particular order to a specific individual to know who ordered what, when, and did they successfully collect the order.

## **Conclusion**

The development of the Four Meals Application (Buse Dining) assistant application had objectives and this was accomplished hence the requirements were met.Conclusively, this research project was a success. The development of the Buse Dining assistant application involved many phases. The first phase started with a detailed study of the problems and prospects of how the current system at the Buse Dining Hall. These problems, information needs, and activities were documented and later used as the basis for system design, which immediately followed the design phase. The design phase was concerned primarily with the specification of the system elements in a manner that best met the organization’s needs. During this phase, strict adherence was made to proven software engineering principles and practices. To implement this design, an application was then developed using android studio. It is hoped that the effective implementation of this software product would eliminate many problems discovered during systems investigation.

Link to the system files: <https://github.com/carldevzw/four_meals_android.git>