CHAPTER 1

INTRODUCTION

1.1 Introduction to Kotlin

JetBrains developed IntelliJ IDEA, which is the basis for Android Studio. In 2011, the company introduced the Kotlin language. Kotlin is a statically-typed, modern programming language that runs on a Java Virtual Machine (JVM) by compiling Kotlin code into Java byte-code. It can also be compiled to JavaScript source code and to native executables. Kotlin is flexible. Kotlin is object-oriented language, and a "better language" than Java, but still be fully interoperable with Java code. Though Kotlin was production ready, the language wasn't stable. When important changes in the language happened, developers had to change their codebase. Five years later, Kotlin released 1.0. At Google I/O 2017, Google announced that Android will support Kotlin as a first-class programming language from now on. For this to happen, the 3.0 release of Android Studio (AS) integrated Kotlin support out of the box! The following three minor releases of AS continued to improve the Kotlin support and the tools available. Kotlin support for JavaScript (i.e., classic back-end) is considered stable in Kotlin 1.3 by its developers, while Kotlin/JS (IR-based) in version 1.4, is considered alpha. Kotlin/Native Runtime (for e.g., Apple support) is considered beta.

1.2 Features of Kotlin

- Statically typed
- Data Classes
- Concise and Safe
- Interoperable with Java
- Functional and Object-Oriented Capabilities
- Less Compilation time
- User-Friendly

1.3 How the idea got into existence

The idea for developing a food order mobile application stemmed from the growing demand for convenient and efficient food delivery services. As people's lifestyles became busier and more fast-paced, the need for a simple solution to order food from local restaurants became evident.

1.Identifying a Need: The creators or developers of the application recognized that people often face challenges when ordering food. These challenges may include difficulty finding restaurants, long wait times on the phone, or limited menu options. They identified a need for a more streamlined and user-friendly way to order food..

2.Market Research: Extensive market research was conducted to analyze the current landscape of food delivery services and mobile applications. This research aimed to understand consumer preferences, competitor offerings, and potential opportunities for improvement..

3.Development and Testing: With the wireframes and prototypes in place, the development process began. The team worked on building the backend infrastructure, frontend interfaces, and integrating necessary APIs (e.g., payment gateways, geolocation services). Throughout development, regular testing was conducted to identify and fix any bugs or usability issues..

4.Ongoing Maintenance and Updates: After the launch, the development team continued to monitor the application's performance, address user feedback, and release updates to enhance functionality, fix bugs, and adapt to evolving user needs.

It's important to note that the actual process may vary depending on the specific circumstances, team structure, resources, and goals of the project. The steps outlined above provide a general framework for how the idea for a food order mobile application may have come to fruition as a mini project.

CHAPTER 2

SYSTEM REQUIREMENT SPECIFICATIONS

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems The first stable build was released in December 2014. Android Studio provides many excellent features that enhance productivity when building Android apps, such as a blended environment where one can develop for all Android devices, apply Changes to push code and resource changes to the running app without restarting the app, a flexible Gradlebased build system, a fast and feature-rich emulator, GitHub and Code template integration to assist you to develop common app features and import sample code, extensive testing tools and frameworks, C++ and NDK support, and many more.

2.1 Hardware Requirements

- Processor - Intel® Core i5 CPU M 370
- Processor Speed - 500 MHz or above
- RAM – Minimum of 8GB.
- Monitor resolution - A color monitor with a minimum resolution of 1000*700

2.2 Software Requirements

- Java Development Kit (JDK)
- Android Studio
- Kotlin access
- Firebase connection
- Android Emulator or a Physical device

CHAPTER 3

ABOUT THE PROJECT

3.1 Overview

The Food Order is a digital solution which involves creating a convenient and user-friendly platform for users to browse and order food from various restaurants. This aims to provide a seamless and efficient experience for customers who want to order food using their smartphones. The application also requires integration with a backend system to handle various functionalities. This includes creating a database to store restaurant details, menu items, and user information. Additionally, integrating a payment gateway allows users to make secure transactions directly from the app.

3.2 Functionalities provided by FOOD ORDER are as follows:

1. **User Registration and Login:** Allow users to create accounts and log in using email, social media accounts, or phone numbers. This feature enables personalized experiences, order history, and saved preferences.

2. **Restaurant Listings:** Display a list of restaurants or food outlets along with their menus, ratings, reviews, and other relevant information. Users can browse and select their preferred restaurant.

3. **Menu Exploration:** Provide an interactive menu with categories, item descriptions, prices, and images. Users can explore the available food items, filter by cuisine type or dietary preferences, and add items to their cart. Notifications: The app sends notifications to warden regarding the student check-in and check-out informations, providing them with important information and ensuring they are aware of the required procedures.

4. **Reviews and Ratings:** Allow users to rate and review restaurants and food items. Display average ratings and reviews to help users make informed decisions. Encourage feedback to improve the overall user experience.

5. **Order History:** Provide users with a history of their past orders for reference and easy reordering. Include details such as order date, items ordered, restaurant name, and order status

6. **Customer Support:** Offer customer support channels such as chat, email, or phone to assist users with their queries, concerns, or complaints.

7. **Menu Exploration:** Provide an interactive menu with categories, item descriptions, prices, and images. Users can explore the available food items, filter by cuisine type or dietary preferences, and add items to their cart.Notifications: The app sends notifications to warden regarding the student check-in and check-out informations, providing them with important information and ensuring they are aware of the required procedures.

8. **Communication and Support:** The app may include a communication feature that allows students to contact the hostel staff or administration for any inquiries, requests, or assistance during their stay.

9. **Discounts and Offers:** Display special deals, discounts, or promotional offers to attract users and encourage repeat orders.

10. **Reporting:** The development of the food order mobile application proved successful in achieving its objectives of providing a user-friendly interface, seamless functionality, and efficient order management. The application offers a convenient platform for users to browse menus, place food orders, track deliveries, and share feedback. With further enhancements and continuous support, the application has the potential to become a reliable and popular choice for food ordering.

3.3 Tech stack

1. **Android Development:**

- Programming Language: Kotlin
- Integrated Development Environment (IDE): Android Studio
- Jetpack: Utilize various Jetpack components such as ViewModel, LiveData, and Navigation for efficient development.

2. **Firebase Integration:**

- Firebase Authentication: Utilized Firebase Authentication for user sign-in with google , login, and OTP verification.
- Firestore database: Store user data, such as food information and details.
- Firebase Cloud Functions: Implement server-side logic using Firebase Cloud Functions to handle Android additional functionalities or data manipulation.

3. **UI/UX:**

- XML: Design user interfaces using XML layouts in Android Studio.
- Material Design: Implement Material Design guidelines to ensure a consistent and visually appealing UI.

4. **Networking and API Integration:**

- Retrofit: Communicate with backend APIs, if required, using Retrofit library for network requests.
- Firebase Cloud Functions: Connect to custom Firebase Cloud Functions for additional backend functionality.

5. **Storage**:

- Firebase Cloud Storage: Store and retrieve files such as user information or any other relevant data.

6. **Dependency Management:**

- Gradle: Utilized Gradle build system for managing dependencies and building the Android app.

7. **Version Control:**

- Git: Git for version control and collaboration.

CHAPTER 4

SYSTEM DESIGN

4.1 Benefits of the application

1. **Convenience:** Food order mobile applications provide a convenient way for users to order food anytime and anywhere. Users can browse menus, select dishes, customize orders, and make payments through the app, eliminating the need for phone calls or physical visits to restaurants.

2. **Enhanced User Experience:** Mobile applications offer a more interactive and engaging user experience compared to traditional methods. Users can explore visually appealing menus, access detailed dish descriptions, view food images, and read reviews or ratings from other customers. This improves the overall ordering process and increases customer satisfaction.

3. **Increased Efficiency:** With a mobile application, the food ordering process becomes more streamlined. Users can quickly search for restaurants, filter options based on preferences, and place orders with just a few taps. This efficiency benefits both users and restaurants, reducing the time and effort required for manual order processing.

4. **Increased Efficiency:** With a mobile application, the food ordering process becomes more streamlined. Users can quickly search for restaurants, filter options based on preferences, and place orders with just a few taps. This efficiency benefits both users and restaurants, reducing the time and effort required for manual order processing.

5. **Data Analysis and Insights:** By collecting user data and order history, a food order mobile application can provide valuable insights to restaurants. This data can be used to analyze customer preferences, identify popular dishes, optimize menus, and make informed business decisions. It also allows for targeted marketing campaigns and personalized promotions.

6. **Increased Revenue:** A well-designed food order mobile application can attract a larger customer base and generate additional revenue streams. The convenience and ease of use

offered by the app can encourage more frequent orders, larger order sizes, and increased customer loyalty, ultimately leading to business growth.

7. **Personalization:** A food order mobile application allows for personalized experiences. Users can create profiles, save favorite dishes or restaurants, and receive recommendations based on their preferences or past orders. This customization enhances user engagement and loyalty.

8. **Increased Efficiency:** With a mobile application, the food ordering process becomes more streamlined. Users can quickly search for restaurants, filter options based on preferences, and place orders with just a few taps. This efficiency benefits both users and restaurants, reducing the time and effort required for manual order processing.

9. **Personalization:** A food order mobile application allows for personalized experiences. Users can create profiles, save favorite dishes or restaurants, and receive recommendations based on their preferences or past orders. This customization enhances user engagement and loyalty.

10. **Enhanced User Experience:** Mobile applications offer a more interactive and engaging user experience compared to traditional methods. Users can explore visually appealing menus, access detailed dish descriptions, view food images, and read reviews or ratings from other customers. This improves the overall ordering process and increases customer satisfaction.

11. **Increased Efficiency:** With a mobile application, the food ordering process becomes more streamlined. Users can quickly search for restaurants, filter options based on preferences, and place orders with just a few taps. This efficiency benefits both users and restaurants, reducing the time and effort required for manual order processing.

12. **Personalization:** A food order mobile application allows for personalized experiences. Users can create profiles, save favorite dishes or restaurants, and receive recommendations based on their preferences or past orders. This customization enhances user engagement and loyalty.

4.2 List of kotlin components used for application

1. **Activities and Fragments:**

   - Use Kotlin classes to implement activities and fragments, representing the different screens and UI components of the app.

2. **Intents:**

   - Utilize Kotlin's intent mechanism to navigate between activities and pass data between different screens.

3. **ViewModels and LiveData:**

   - Use Kotlin's ViewModel and LiveData components from the Android Architecture Components to manage UI-related data and ensure data updates are reflected in real time.

4. **Coroutines:**

   - Utilize Kotlin Coroutines to handle asynchronous tasks, such as network requests or database operations, in a more concise and efficient manner.

5. **Firebase Authentication:**

   - Integrate Firebase Authentication in Kotlin to handle user registration, login, and OTP verification.

6. **Firebase Realtime Database or Firestore:**

   - Use Kotlin code to interact with Firebase Realtime Database or Firestore for storing and retrieving data related to user information, check-in/check-out records, and notifications.

7. **Push Notifications:**

   - Implement Firebase Cloud Messaging in Kotlin to send push notifications to users, delivering check-in/check-out reminders, updates, or important information.

8. **Material Design Components:**

   - Utilize Kotlin code to implement various Material Design components provided by the Material Components for Android library, ensuring a consistent and visually appealing UI.

9. **Navigation Components:**

   - Use Kotlin code to implement the Navigation Component from the Android Jetpack library, facilitating navigation between different screens and handling back stack management.

10. **Retrofit or OkHttp:**

    - Integrate Retrofit or OkHttp in Kotlin to handle network requests and communicate with backend APIs, if required for additional functionalities.

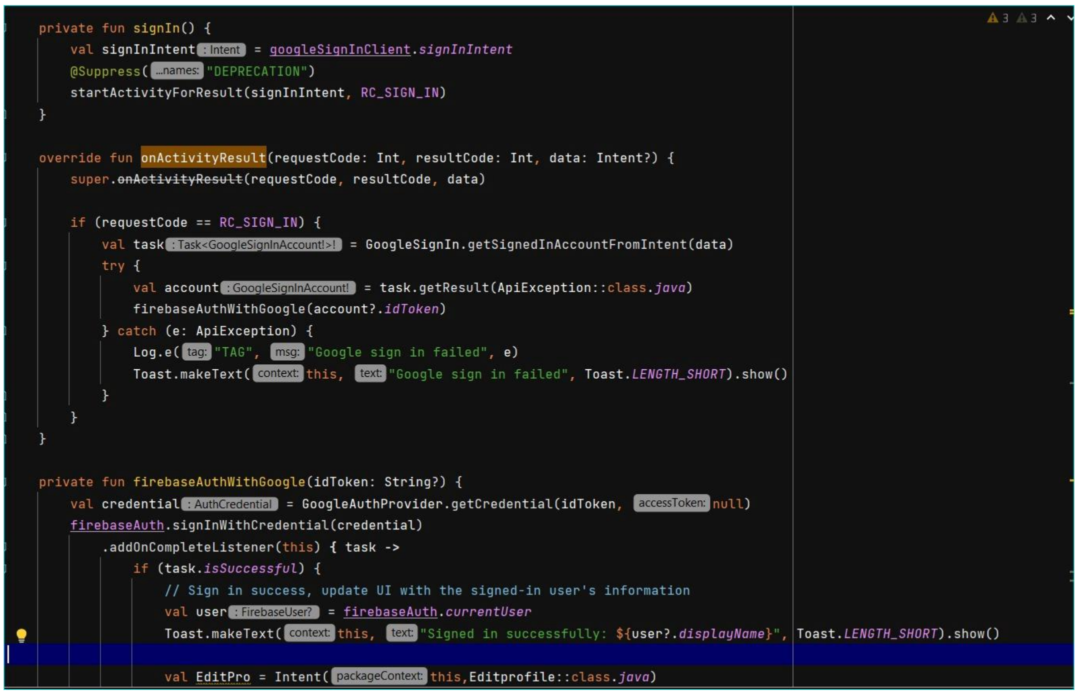
11. **Dependency Injection:**

    - Implement dependency injection frameworks like Dagger or Koin in Kotlin to manage and inject dependencies into various components of the app.

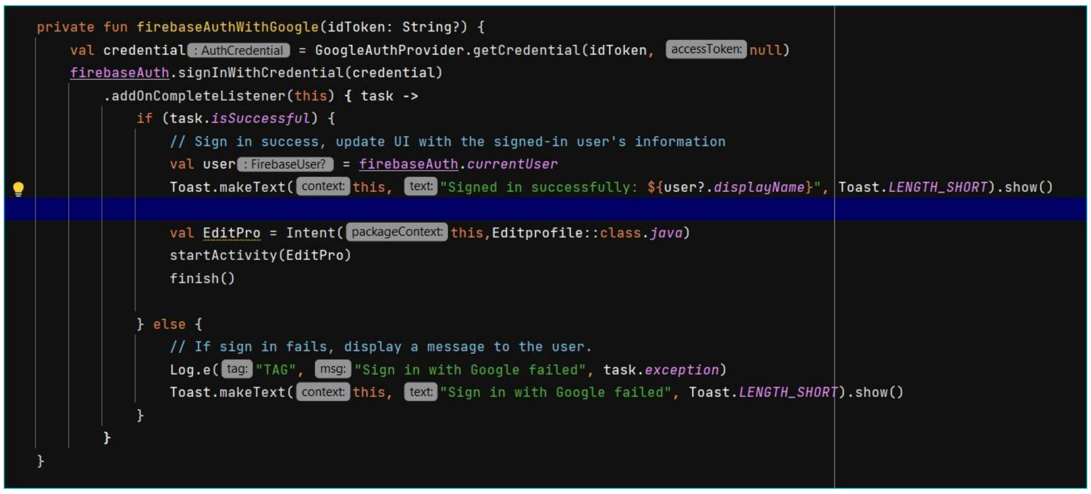
CHAPTER 5

IMPLEMENTATION AND INTRODUCTION TO TESTING

Login.kt

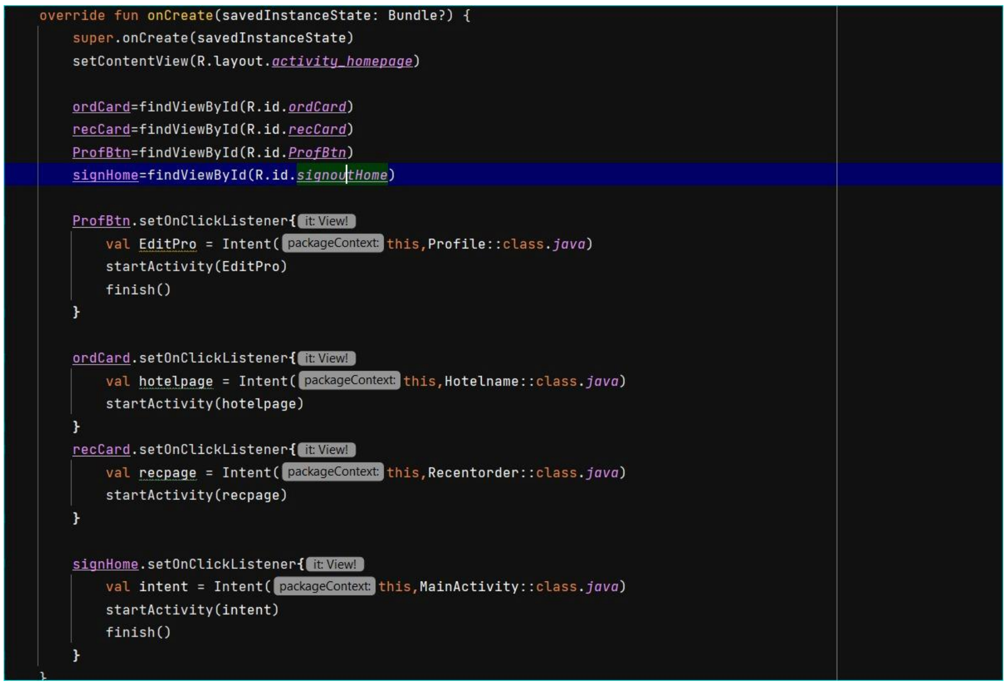


```
private fun signIn() {
    val signInIntent : Intent = googleSignInClient.signInIntent
    @Suppress( ...names: "DEPRECATION")
    startActivityForResult(signInIntent, RC_SIGN_IN)
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.~~onActivityResult~~(requestCode, resultCode, data)

    if (requestCode == RC_SIGN_IN) {
        val task : Task<GoogleSignInAccount!>! = GoogleSignIn.getSignedInAccountFromIntent(data)
        try {
            val account : GoogleSignInAccount! = task.getResult(ApiException::class.java)
            firebaseAuthWithGoogle(account?.idToken)
        } catch (e: ApiException) {
            Log.e( tag: "TAG",  msg: "Google sign in failed", e)
            Toast.makeText( context: this,  text: "Google sign in failed", Toast.LENGTH_SHORT).show()
        }
    }
}

private fun firebaseAuthWithGoogle(idToken: String?) {
    val credential : AuthCredential = GoogleAuthProvider.getCredential(idToken,  accessToken: null)
    firebaseAuth.signInWithCredential(credential)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                // Sign in success, update UI with the signed-in user's information
                val user : FirebaseUser? = firebaseAuth.currentUser
                Toast.makeText( context: this,  text: "Signed in successfully: ${user?.displayName}", Toast.LENGTH_SHORT).show()

                val EditPro = Intent( packageContext: this,Editprofile::class.java)
```



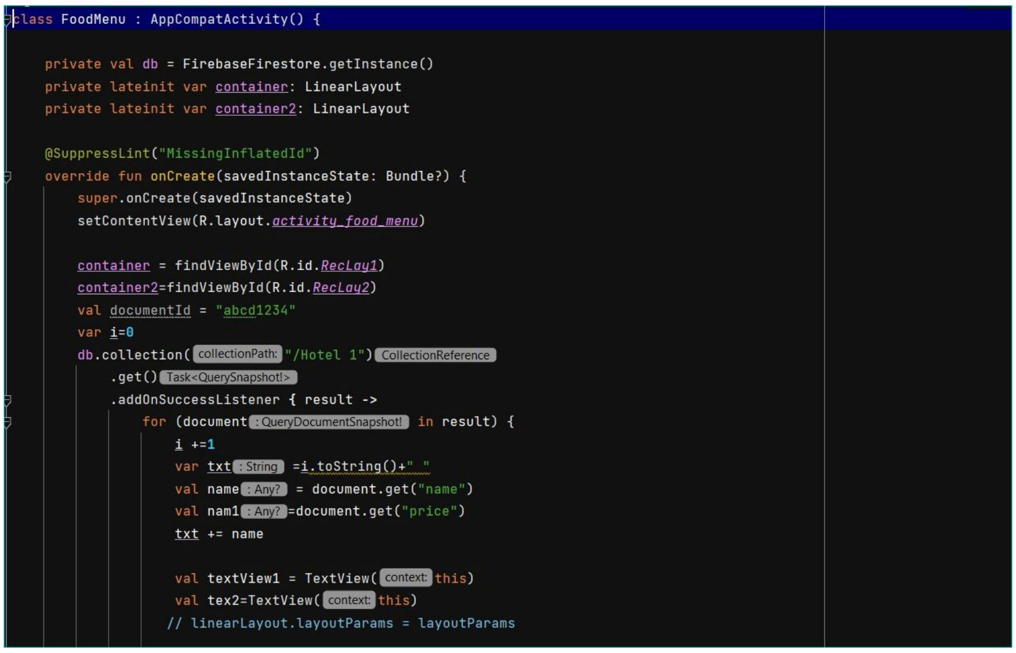
```
private fun firebaseAuthWithGoogle(idToken: String?) {
    val credential : AuthCredential = GoogleAuthProvider.getCredential(idToken,  accessToken: null)
    firebaseAuth.signInWithCredential(credential)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                // Sign in success, update UI with the signed-in user's information
                val user : FirebaseUser? = firebaseAuth.currentUser
                Toast.makeText( context: this,  text: "Signed in successfully: ${user?.displayName}", Toast.LENGTH_SHORT).show()

                val EditPro = Intent( packageContext: this,Editprofile::class.java)
                startActivity(EditPro)
                finish()

            } else {
                // If sign in fails, display a message to the user.
                Log.e( tag: "TAG",  msg: "Sign in with Google failed", task.exception)
                Toast.makeText( context: this,  text: "Sign in with Google failed", Toast.LENGTH_SHORT).show()
            }
        }
}
```

Hpmepage.kt

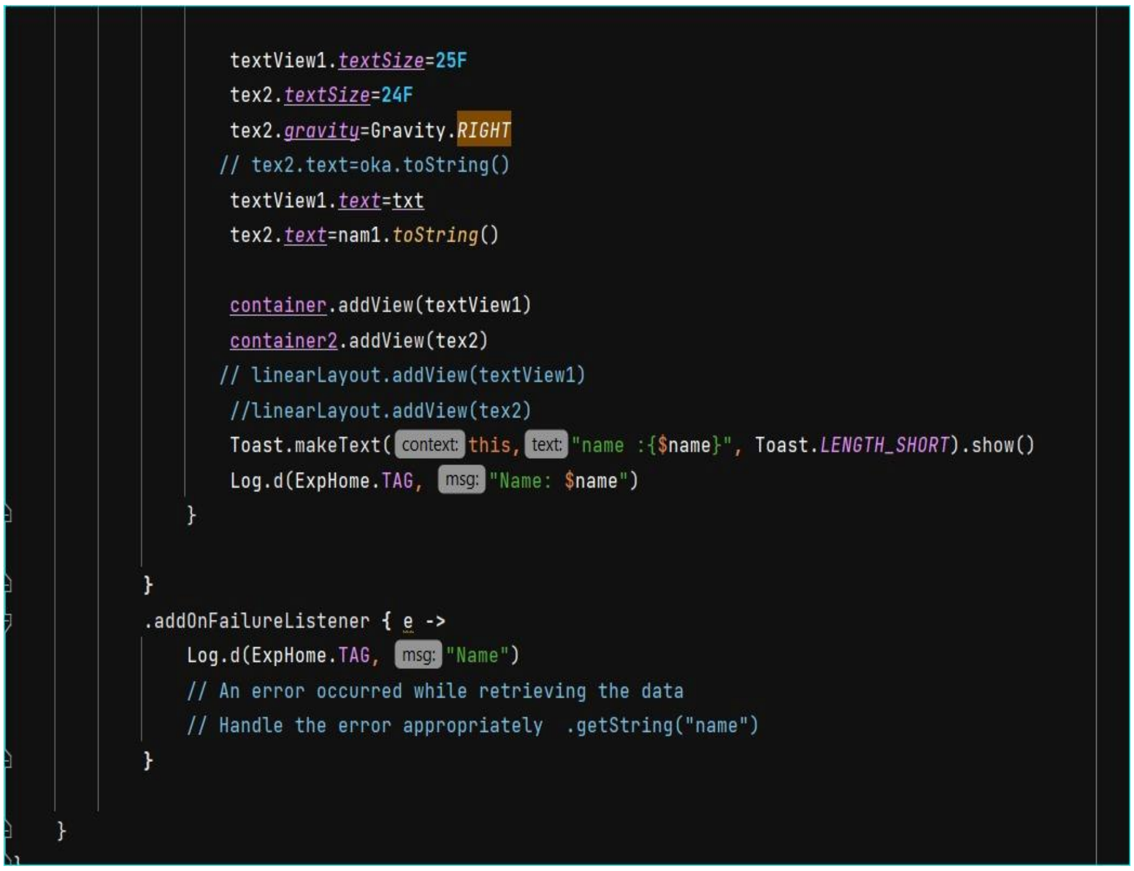


```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_homepage)

    ordCard=findViewById(R.id.ordCard)
    recCard=findViewById(R.id.recCard)
    ProfBtn=findViewById(R.id.ProfBtn)
    signHome=findViewById(R.id.signoutHome)

    ProfBtn.setOnClickListener{ it: View!
        val EditPro = Intent( packageContext: this,Profile::class.java)
        startActivity(EditPro)
        finish()
    }

    ordCard.setOnClickListener{ it: View!
        val hotelpage = Intent( packageContext: this,Hotelname::class.java)
        startActivity(hotelpage)
    }
    recCard.setOnClickListener{ it: View!
        val recpage = Intent( packageContext: this,Recentorder::class.java)
        startActivity(recpage)
    }

    signHome.setOnClickListener{ it: View!
        val intent = Intent( packageContext: this,MainActivity::class.java)
        startActivity(intent)
        finish()
    }
}
```

FoodMenu.kt



```
class FoodMenu : AppCompatActivity() {

    private val db = FirebaseFirestore.getInstance()
    private lateinit var container: LinearLayout
    private lateinit var container2: LinearLayout

    @SuppressLint("MissingInflatedId")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_food_menu)

        container = findViewById(R.id.RecLay1)
        container2=findViewById(R.id.RecLay2)
        val documentId = "abcd1234"
        var i=0
        db.collection( collectionPath: "/Hotel 1") CollectionReference
            .get() Task<QuerySnapshot!>
            .addOnSuccessListener { result ->
                for (document : QueryDocumentSnapshot! in result) {
                    i +=1
                    var txt : String =i.toString()+" "
                    val name : Any? = document.get("name")
                    val nam1 : Any? =document.get("price")
                    txt += name

                    val textView1 = TextView( context: this)
                    val tex2=TextView( context: this)
                   // linearLayout.layoutParams = layoutParams
```



```
                textView1.textSize=25F
                tex2.textSize=24F
                tex2.gravity=Gravity.RIGHT
               // tex2.text=oka.toString()
                textView1.text=txt
                tex2.text=nam1.toString()

                container.addView(textView1)
                container2.addView(tex2)
               // linearLayout.addView(textView1)
                //linearLayout.addView(tex2)
                Toast.makeText( context: this, text: "name :{$name}", Toast.LENGTH_SHORT).show()
                Log.d(ExpHome.TAG,  msg: "Name: $name")
            }

        }
        .addOnFailureListener { e ->
            Log.d(ExpHome.TAG,  msg: "Name")
            // An error occurred while retrieving the data
            // Handle the error appropriately  .getString("name")
        }

    }
```

TESTING:

Verification and validation are a generic name given to food processes, which ensures that the software conforms to its specifications and meets the demands of users.

- Validation: Are we building the right product? Validation involves checking that the program has implanted meets the requirement of the users.
- Verification: Verification involves checking that the program confirms to its specification.

Results:

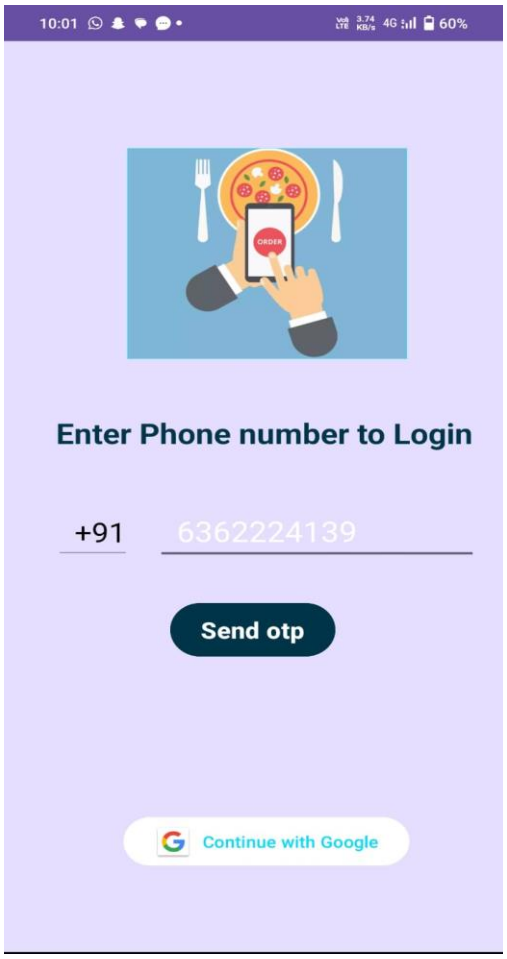
Several errors were detected and rectified and the whole project is working as it should have to work with proper output and high efficiency.

Test case ID	**Test case**	**Steps to execute the test case**	**Expected result**	**Actual result**	**Pass/ Fail**
1	Sign in with Google	Click the button ‘continue with google’	Update profile	Update profile	Pass
2	View Hotel Name	Click the button ‘Order Food’	Hotel Name	Hotel Name	Pass
3	View Recent Orders	Click the button ’Recent Order’	Recent Orders	Recent Orders	Pass
4	View Menu List	Click the button ‘Hotel 1’	Menu List	Menu List	Pass
5	View Sign out page	Click the button ‘sign out’ in Homa page	Sign out page	Sign out page	Pass
6	Sign out	Click the button ‘sign out’ in sign out page	Login page	Login Page	pass

Table 5.1 Testing for the “FOOD ORDER” Project

CHAPTER 6

SNAPSHOTS



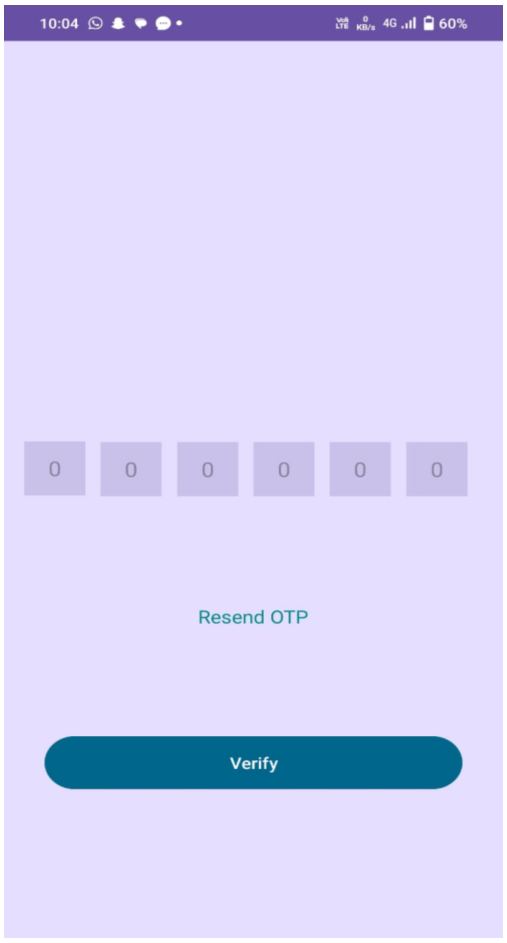


Figure 6.1 Login page

Figure 6.2 OTP verify

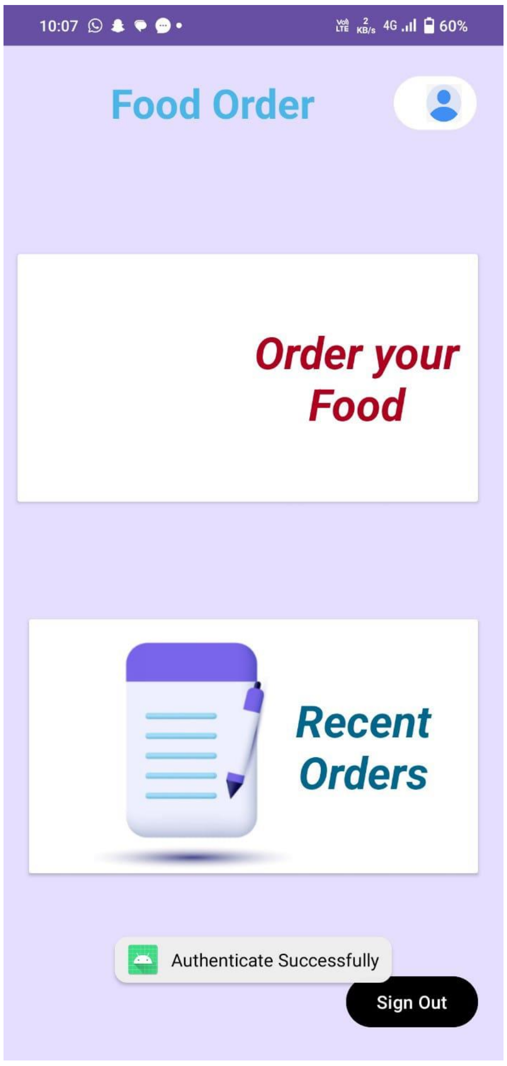


Figure 6.3 Homepage

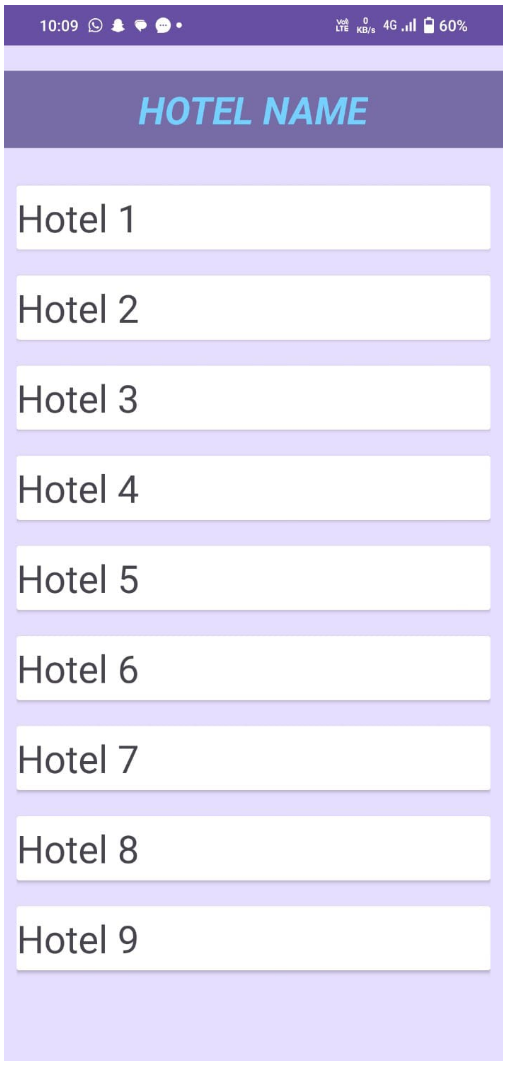
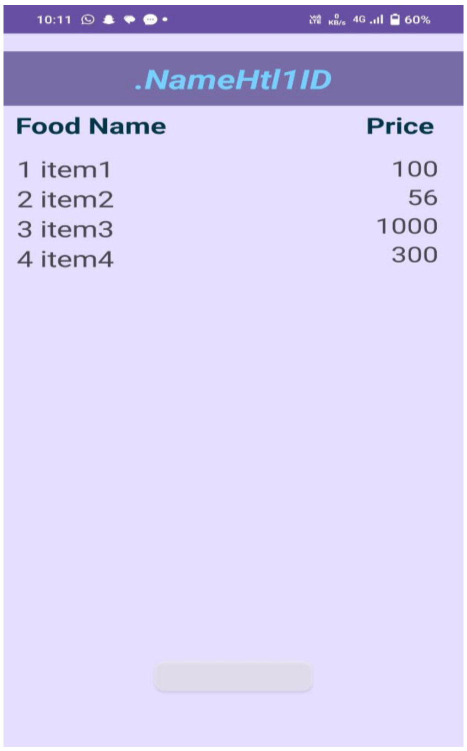


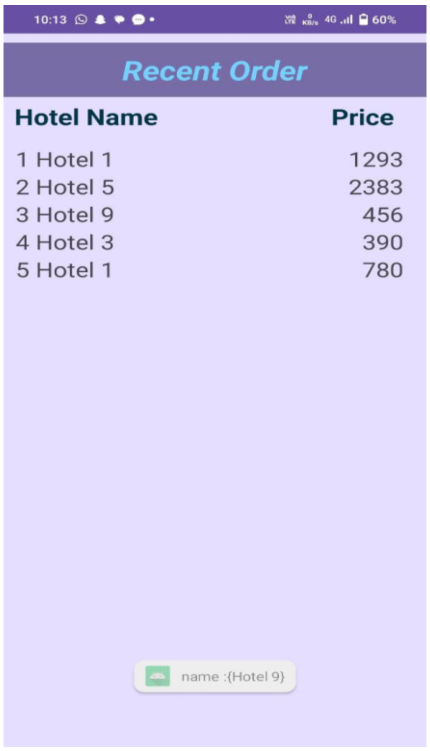
Figure 6.4 Hotel List



.NameHtl1ID

Food Name	Price
1 item1	100
2 item2	56
3 item3	1000
4 item4	300

Figure 6.5 Menu List



Recent Order

Hotel Name	Price
1 Hotel 1	1293
2 Hotel 5	2383
3 Hotel 9	456
4 Hotel 3	390
5 Hotel 1	780

name :{Hotel 9}

Figure 6.6 Recent Order from hotel

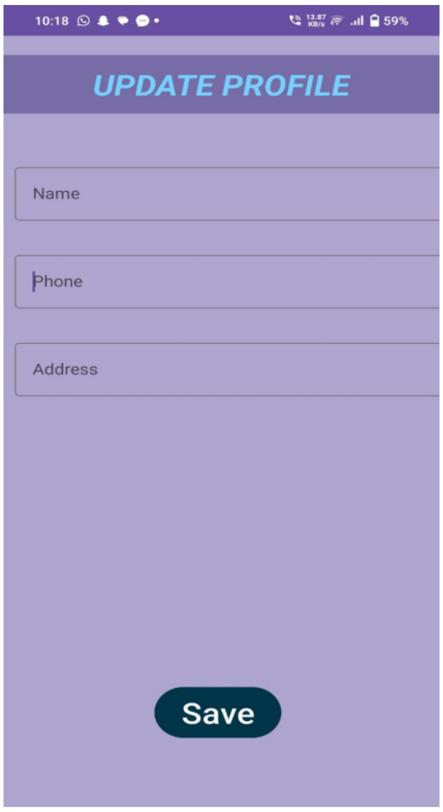
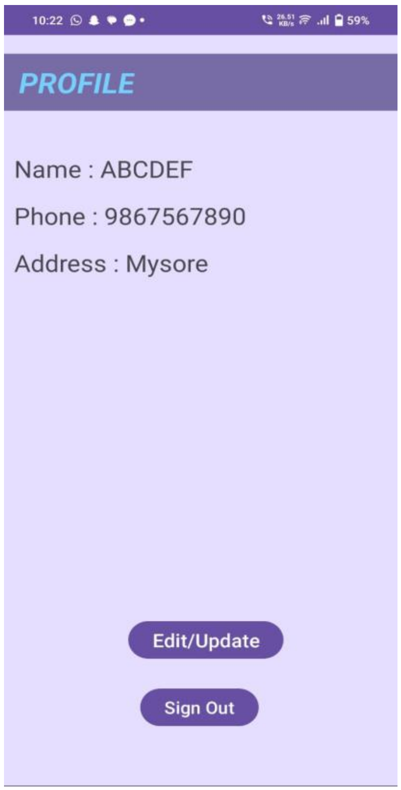


Figure 6.7 Profile Update



PROFILE

Name : ABCDEF

Phone : 9867567890

Address : Mysore

Figure 6.8 Profile Page

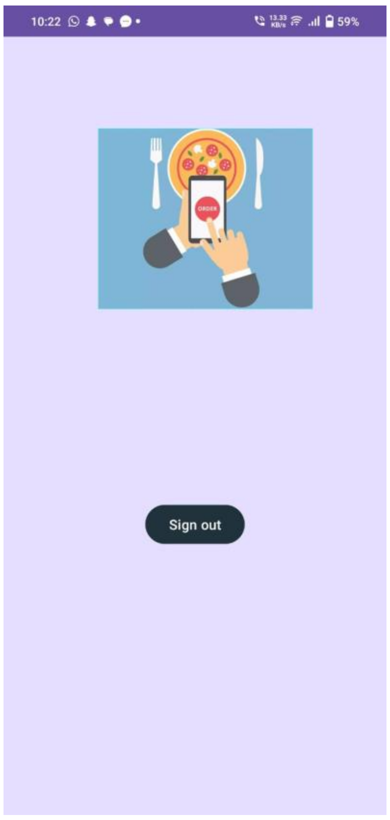


Figure 6.9 Sign Out page

CHAPTER 7

CONCLUSION AND FUTURE ENHANCEMENT

In conclusion, the development of a mobile application for food ordering in hotels is an essential and beneficial mini project. Such an application streamlines the food ordering process, making it more convenient for both hotel guests and staff. With this application, guests can easily browse the menu, place their orders, and make payments directly from their mobile devices. This eliminates the need for traditional paper menus, phone calls, and physical interactions, enhancing efficiency and reducing errors.

The implementation of a food order feature provides numerous benefits for both parties involved. For customers, it offers convenience, as they can easily access and explore a wide range of food options through the mobile app. They can place orders at any time and from anywhere, eliminating the need to wait in long queues or make multiple phone calls. Additionally, customers can customize their orders according to their preferences, making the experience more personalized.

For hostel administration, the app revolutionizes their operations and simplifies administrative tasks. By automating processes, such as recording check-ins and check-outs, generating reports, and managing communication, the app saves time, minimizes errors, and improves overall data management. Hostel staff can focus on providing better support and addressing student needs, rather than being burdened with paperwork and manual record-keeping.

Moreover, the food order feature opens up new opportunities for upselling and cross-selling. Through the mobile app, hotels can showcase their special offers, promotions, and recommendations to customers, encouraging them to try new items or upgrade their orders. This feature serves as a marketing tool, allowing hotels to effectively promote their menu items and increase sales.

FUTURE ENHANCEMENT:

1. In the future, there are several potential enhancements that can be incorporated into a hotel mobile application for food ordering. These enhancements aim to improve the overall user experience, streamline the ordering process, and provide additional convenience to hotel guests,

2. One possible enhancement is the integration of artificial intelligence (AI) technologies, such as natural language processing and machine learning. By implementing AI, the application can better understand and interpret user preferences and dietary restrictions. It can provide personalized recommendations based on previous orders or user profiles, ensuring that guests receive tailored suggestions that match their tastes and requirements.

3. Another enhancement could involve the implementation of augmented reality (AR) features. With AR, users can visualize the dishes they are ordering in a virtual environment, allowing them to get a realistic preview of the food before making a decision. This feature can help guests make more informed choices and reduce the chances of disappointment when their orders arrive.

4. Furthermore, incorporating real-time tracking capabilities into the application can provide users with updates on the status of their orders. This feature can include notifications at each stage of the process, from order confirmation to preparation, cooking, and delivery. Guests can easily track their food's progress and estimated time of arrival, enhancing transparency and reducing uncertainty.

5. To enhance convenience, integrating mobile payment options and digital wallets within the application can streamline the checkout process. Guests can securely make payments within the app, eliminating the need for physical cash or credit cards. Additionally, the integration of loyalty programs and discounts can incentivize guests to use the application repeatedly, fostering customer loyalty.

6. Lastly, incorporating social sharing features into the app can allow users to share their dining experiences with friends and family on social media platforms. This can help hotels promote their food offerings and generate positive word-of-mouth marketing.

7. Social Features: Incorporate social features within the app, such as a community forum or chat functionality, where students can connect, interact, and share information with each other. This can foster a sense of community and provide a platform for students to engage with each other.

8. Multi-Language Support: Extend language support to cater to international students or students from diverse backgrounds. This can enhance accessibility and ensure that the app can be used by a wider range of students.

9. Overall, future enhancements in a hotel mobile application for food ordering may involve AI-driven personalization, AR visualization, real-time tracking, mobile payment integration, loyalty programs, and social sharing features. By implementing these enhancements, hotels can provide a seamless and delightful food ordering experience for their guests, enhancing customer satisfaction and loyalty.

10. Data Analytics and Insights: Enhance the analytics and reporting capabilities of the app to provide deeper insights into student behavior, trends, and preferences. This can help hostel administration make data-driven decisions, optimize operations, and enhance the overall student experience.

BIBLIOGRAPHY

Websites

www.wikipedia.com

https://firebase.google.com/

www.Youtube.com

www.chatgpt.com

www.Github.com