

# **GUARDIAN ON THE GO : AN EMERGENCY SOS APP**

## **CS19611 - MOBILE APPLICATION DEVELOPMENT LABORATORY**

### **PROJECT REPORT**

*submitted by*

**SHARUKESHWAR P                      220701265**

In partial fulfillment for the award of the degree  
of

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE**



**RAJALAKSHMI ENGINEERING COLLEGE**

**(AUTONOMOUS) THANDALAM**

**CHENNAI-602105**

**2024-2025**

---

## **RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI**

### **BONAFIDE CERTIFICATE**

Certified that this Project titled “Emergency SOS App” is the bonafide work of “**SHARUKESHWAR P (220701265)**” who carried out the work under my supervision. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

#### **SIGNATURE**

Dr. P. Kumar., M.E., Ph.D.,

#### **HEAD OF THE DEPARTMENT**

Professor Department of Computer  
Computer Science & Engineering.

Rajalakshmi Engineering College,  
Chennai - 602 105.

#### **SIGNATURE**

Dr. N. Duraimurugan., M.E., Ph.D.,

#### **SUPERVISOR**

Associate Professor Department of  
Computer Science & Engineering

Rajalakshmi Engineering College,  
Chennai-602 105.

Submitted to Mini Project Viva-Voce Examination held on 14.05.2025

Internal Examiner

External Examiner

## ACKNOWLEDGMENT

Initially we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.**, and our respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D.**, for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution. Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.**, our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. P. KUMAR, M.E., Ph.D.**, Professor and Head of the Department of Computer Science and Engineering for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide **Dr. N. DURAIMURUGAN**, We are very glad to thank our Project Coordinator, **Dr. N. DURAIMURUGAN** Associate Professor Department of Computer Science and Engineering for his useful tips during our review to build our project.

**SHARUKESHWAR P 220701265**

## **ABSTRACT**

---

The Emergency SOS Android application is a lightweight, user-centric mobile solution designed to enhance personal safety by enabling users to quickly alert trusted contacts during emergencies. With a minimal and intuitive interface, the app allows users to save a set of emergency contact numbers and send real-time SOS messages via SMS at the tap of a button. These messages include a predefined alert along with the user's current GPS location, ensuring that help can be dispatched promptly. The application also supports offline functionality by relying on SMS, making it reliable even without internet access. To further increase usability and responsiveness, the app can be expanded to include features such as shake-to-trigger SOS, fake call simulations for discreet escapes, flashlight blinking as a visual alert, and auto-calling the primary contact. Designed as a Minimum Viable Product (MVP), the project emphasizes simplicity, speed, and reliability, serving as a foundational step towards building more advanced personal safety applications in the future.

# **TABLE OF CONTENTS**

## **1. INTRODUCTION**

1.1 INTRODUCTION

1.2 OBJECTIVES

1.3 MODULES

## **2. SURVEY OF TECHNOLOGIES**

2.1 SOFTWARE DESCRIPTION

2.2 LANGUAGES

2.2.1 KOTLIN

2.2.2 XML

## **3.REQUIREMENTS AND ANALYSIS**

3.1 REQUIREMENT SPECIFICATION

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

3.3 ARCHITECTURE DIAGRAM

3.4 ER DIAGRAM

3.5 NORMALIZATION

## **4. PROGRAM CODE**

## **5. RESULTS AND DISCUSSION**

## **6.CONCLUSION**

## **7.REFERENCES**

## CHAPTER 1- INTRODUCTION

---

### 1.1 INTRODDUCTION :

The **Emergency SOS Android App** is a simple yet powerful personal safety tool designed to help users quickly alert their trusted contacts in times of distress. With an easy-to-use interface, the app allows users to save emergency contact numbers and instantly send an SOS message containing their real-time location via SMS at the press of a button.

Built as a lightweight MVP, it focuses on core functionality without relying on internet access, ensuring reliability even in offline scenarios.

The project serves as a foundational step towards building more advanced safety solutions, with potential for additional features like shake-to-trigger alerts, fake calls, and flashlight signals.

This project aims to provide a fast, reliable, and easily deployable safety solution that empowers users to seek help instantly with minimal effort.

### 1.2 OBJECTIVES:

**Efficient Payment Processing:** Enable users to make secure transactions swiftly and conveniently.

**User-Friendly Interface:** Provide an intuitive interface for seamless navigation and interaction.

**Reliable Data Management:** Implement MVC architecture with Firebase Firestore for efficient storage and retrieval of transaction data.

**Scalability and Performance:** Design the application to handle increasing user loads while maintaining optimal performance.

**Enhanced Security:** Utilize secure card-based transactions and encryption protocols to safeguard user data.

**Continuous Improvement:** Commit to ongoing development and refinement to meet evolving user needs and industry standard.

---

## CHAPTER 2- SURVEY OF TECHNOLOGIES

---

### 2.1 SOFTWARE DESCRIPTION:

**Advanced Authentication:** Robust email/password authentication for secure access.

**Efficient Product Management:** QR code integration for swift product addition.

**Streamlined Cart Operations:** Easy product selection, quantity adjustments, and checkout.

**Secure Payment Processing:** Integrated with trusted payment gateways for safe transactions.

**Data Integrity:** Firebase Firestore integration ensures reliable data storage and retrieval.

**Intuitive Interface:** User-centric design enhances usability and navigation.

**Enhanced Security Measures:** Encryption protocols and authentication mechanisms ensure data protection.

**Versatile Solution:** Ideal for consumers and businesses seeking efficient payment processing.



## **2.2 LANGUAGES:**

### **2.2.1 KOTLIN**

Kotlin, introduced by JetBrains and officially supported by Google for Android development, is a statically typed language that offers concise syntax and modern programming features such as null safety, lambda expressions, and coroutines. In Automated Billing System, Kotlin is used to handle UI logic, database interactions, and activity lifecycle management, ensuring robust and maintainable code.

### **2.2.2 XML**

XML is used extensively in Android for defining user interface layouts. Each screen in Automated Billing System—such as the splash and main screens—is created using XML files that detail the arrangement, style, and properties of UI components. XML ensures that the app's design is declarative and easily adjustable without altering the Kotlin logic.

## **FIREBASE:**

**Scalability:** Firestore is a NoSQL database designed to scale effortlessly as data grows. It can handle large volumes of transactions without sacrificing performance, ensuring that EzBill can accommodate increasing user activity and transaction loads over time.

**Real-time Updates:** Firestore enables real-time data synchronization, allowing transaction details to be updated instantly across all connected clients. This ensures that users and administrators have access to the latest transaction information without delays, enhancing the application's responsiveness and usability.

**Structured Data Storage:** Firestore's flexible data model allows EzBill to store transaction details in a structured format, including user IDs, timestamps, product details, and amount information. This structured approach facilitates efficient data retrieval, querying, and analysis, enabling administrators to gain insights into transaction trends and patterns.

**Document-based Storage:** Firestore organizes data into documents and collections, making it easy to store transaction records as individual documents within a collection. This granular storage approach enables efficient retrieval and manipulation of transaction data, supporting various use cases such as filtering transactions by user, product, or time.

**Security Rules:** Firestore's security rules allow EzBill to enforce access control and data validation, ensuring that only authorized users can access and modify transaction data. This helps maintain data integrity and protect sensitive information from unauthorized access or tampering.

## CHAPTER 3- REQUIREMENTS AND ANALYSIS

---

### 3.1- REQUIREMENT SPECIFICATION:

**QR Code Integration:** Allow users to add products to inventory by scanning QR codes for swift entry.

**Real-time Cart Updates:** Enable users to review and adjust cart items dynamically with instant updates.

**Secure Payment Gateway:** Integrate trusted payment gateways for safe and efficient transactions.

**Firebase Firestore Integration:** Utilize Firestore for reliable storage and retrieval of transaction data and user information.

**Intuitive User Interface:** Design a user-friendly interface with easy navigation and clear options for adding products and accessing the cart.

**Robust Security Measures:** Implement encryption protocols and secure authentication mechanisms to protect user data and transactions.

**Performance Optimization:** Optimize application performance for fast loading times and smooth user interactions.

**Scalability and Reliability:** Ensure the application can handle increasing user traffic and data volume while maintaining reliability.

**Comprehensive Testing:** Conduct thorough testing to identify and address bugs, usability issues, and security vulnerabilities.

**Documentation and Support:** Provide detailed documentation for users and administrators on application usage and features. Offer responsive customer support to address inquiries and technical issues promptly.

## 3.2 HARDWARE AND SOFTWARE REQUIREMENTS:

### Backend Development:

- Firebase Firestore for data storage.
- Card-based payment gateway integration.

### Frontend Development:

- Kotlin framework for cross-platform development.

### Development Tools:

Visual Studio Code, Git, android studio, IntelliJ Idea for coding and version control.

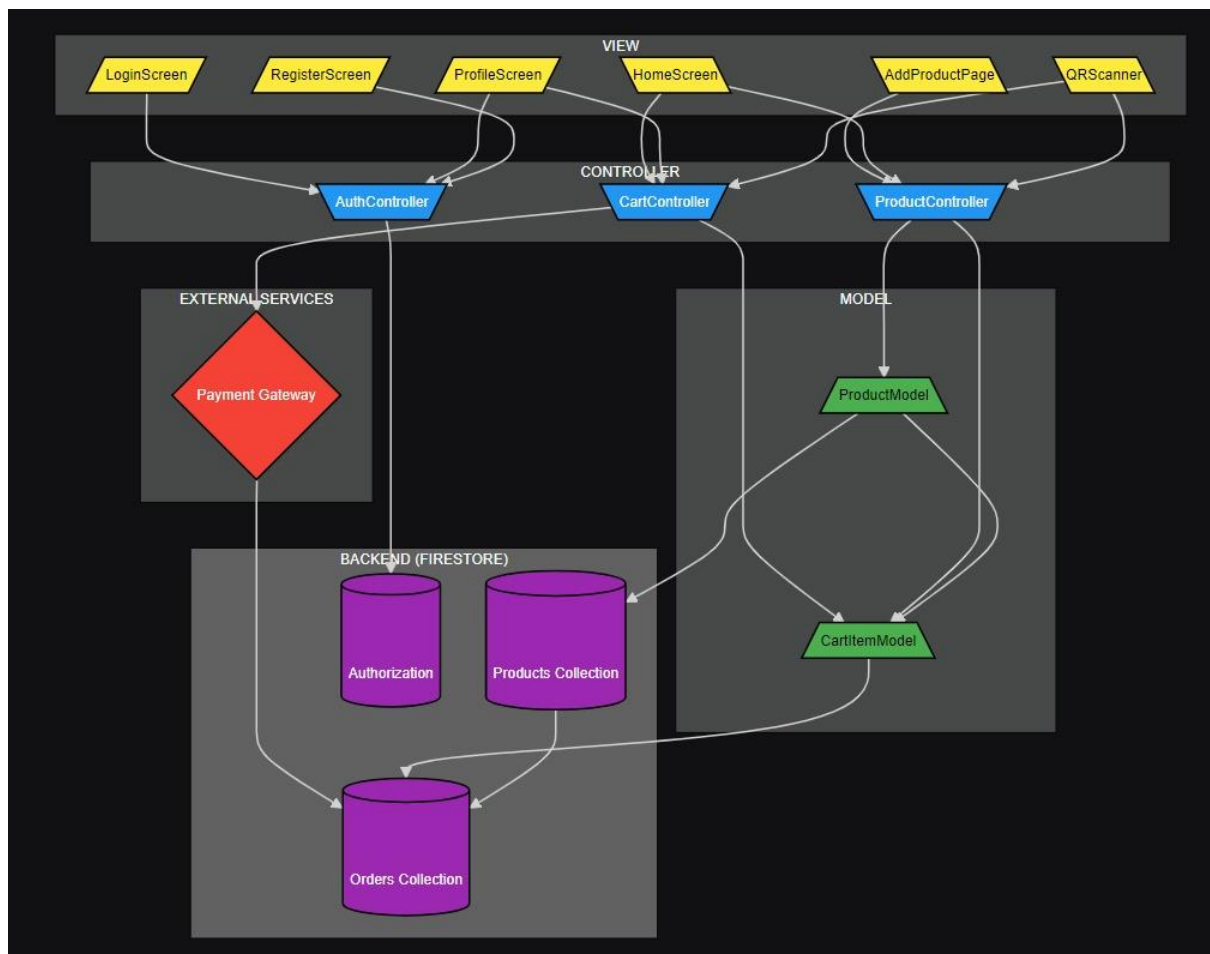
### Deployment and Hosting:

- Firebase Hosting and Cloud Functions.

### Security and Compliance:

- SSL certificate for encryption.
- Compliance with GDPR or CCPA.

### 3.3 ARCHITECTURE DIAGRAM:



### ARCHITECTURE DIAGRAM CODE:

graph TB

%% Model

subgraph Model

A[CartItemModel]

B[ProductModel]

end

%% View

subgraph View

C[AddProductPage]

D[HomeScreen]

E[LoginScreen]

F[ProfileScreen]

```

    G[QRScanner]
    H[RegisterScreen]
    I[FirebaseOptions]
end

%% Controller
subgraph Controller
    J[AuthController]
    K[CartController]
    L[ProductController]
end

%% Backend
subgraph "Backend (Firestore)"
    M[Products Collection]
    N[Orders Collection]
end

%% External Services
subgraph "External Services"
    O[Payment Gateway]
end

%% Relationships between View and Controller
C --> L
D --> K
D --> L
E --> J
F --> J
G --> L
H --> J

```

%% Relationships between Controller and Model

J --> A

J --> B

K --> A

L --> B

%% Storing products in Firestore

L --> M

%% Placing orders through Payment Gateway

K --> O

O --> N

%% Add product flow

C --> L

L --> M

%% Order placement flow

F --> K

K --> O

O --> N

%% Visual Grouping

classDef view fill:#ffeb3b,stroke:#000,stroke-width:2px,color:#000;

classDef controller fill:#2196f3,stroke:#000,stroke-width:2px,color:#fff;

classDef model fill:#4caf50,stroke:#000,stroke-width:2px,color:#000;

classDef backend fill:#9c27b0,stroke:#000,stroke-width:2px,color:#fff;

classDef external fill:#f44336,stroke:#000,stroke-width:2px,color:#fff;

class A,B model;

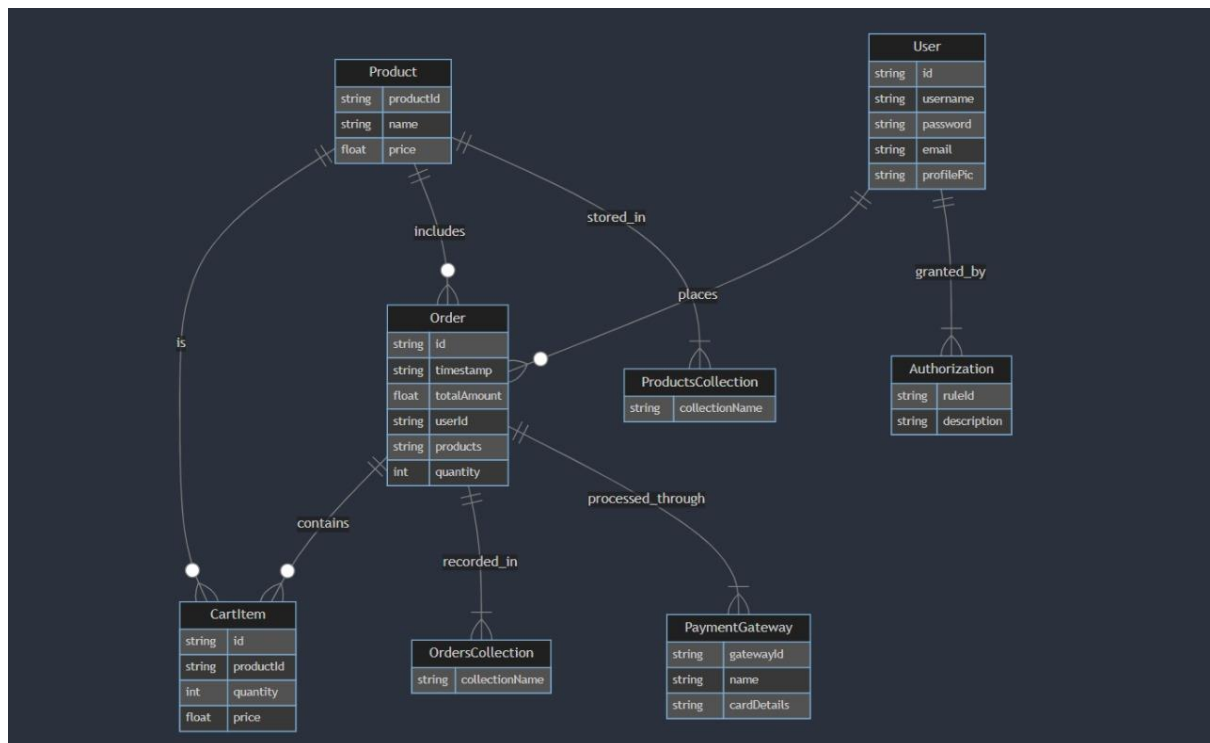
class C,D,E,F,G,H,I view;

class J,K,L controller;

class M,N backend;

class O external;

## ER DIAGRAM:



## ER DIAGRAM CODE:

CUSTOMER {

string customerId PK

string name

string email

}

PRODUCT {

string productId PK

string name

float price



```
}
```

```
CART {  
    string cartId PK  
    string customerId FK  
}
```

```
CART_ITEM {  
    string cartItemId PK  
    string cartId FK  
    string productId FK  
    int quantity  
}
```

```
PAYMENT {  
    string paymentId PK  
    string cartId FK  
    float amount  
    date paymentDate  
    string paymentMethod  
    string cardNumber  
}
```

```
CUSTOMER ||--o{ CART : owns  
CART ||--o{ CART_ITEM : contains  
PRODUCT ||--o{ CART_ITEM : is_part_of  
CART ||--|{ PAYMENT : has
```

## 3.5 NORMALIZATION:

### Introduction

In modern application development, efficient data management is crucial for performance, scalability, and maintainability. One of the key concepts in database design is normalization, which aims to minimize redundancy and ensure data integrity. In the context of using Firestore with a Flutter project, understanding how Firestore handles normalization and its benefits can significantly enhance the quality and performance of the application. This report explores how Firestore participates in normalization and its contributions to a Flutter project in technical terms.

## **Understanding Normalization**

Normalization is a database design process that structures tables to reduce redundancy and improve data integrity. It involves organizing data into smaller, related tables and defining relationships between them. The primary objectives are to eliminate redundant data and ensure that data dependencies are logical and efficient. Traditional relational databases rely heavily on normalization to manage data efficiently.

## **Firestore's Approach to Normalization**

Firestore is a NoSQL document database designed for flexible and scalable data storage. Unlike traditional relational databases, Firestore does not enforce a fixed schema, allowing developers to store data in a more dynamic and hierarchical manner. However, principles of normalization can still be applied to Firestore to optimize data storage and retrieval.

- 1. Avoiding Redundancy:** Firestore enables the avoidance of data redundancy by allowing the use of references to other documents rather than embedding large, duplicate datasets. For example, instead

of embedding user information directly in multiple documents, a `users` collection is created where each document contains user details. Other collections, such as `orders`, then reference the user documents through unique identifiers.

**2. Data Consistency and Integrity:** Although Firestore does not enforce foreign key constraints like relational databases, data consistency can be maintained by using references and implementing Firestore security rules. For instance, an `order` document might have a `userId` field that references a document in the `users` collection. Security rules can ensure that the `userId` exists in the `users` collection, thereby maintaining referential integrity.

**3. Hierarchical Data Structures:** Firestore supports nested documents and subcollections, allowing for hierarchical data organization that aligns with normalized structures. For example, a `users` collection with each user document containing a subcollection of `orders` ensures that user information is not redundantly stored in each order document.

## **Contribution to the Flutter Project**

Implementing normalization principles in Firestore significantly enhances the performance, scalability, and maintainability of a Kotlin project. The following sections detail these contributions.

**1. Efficient Data Access:** By structuring data to minimize redundancy, Firestore ensures efficient data retrieval. This is particularly important for mobile applications, which often operate under limited resource conditions. For instance, fetching a user's

profile data only once and referencing it in other parts of the application avoids unnecessary data retrieval, improving performance.

**2. Simplified Data Management:** Normalized data structures in Firestore facilitate easier updates and maintenance. Centralizing user information in a single document reduces the risk of inconsistencies and makes updates straightforward. For example, changing a user's email address in a single user document rather than multiple places simplifies data management and ensures consistency.

**3. Scalable Data Handling:** Properly structured data that adheres to normalization principles can scale more effectively. Firestore can handle complex queries and large datasets without significant performance degradation. Querying related data across collections without unnecessary data duplication allows for efficient indexing and query execution, which is critical for scalable application performance.

**Improved Code Maintainability:** A normalized approach leads to a cleaner and more modular codebase. With clear data relationships, the Flutter codebase becomes easier to read and maintain. For example, separating user data, order data, and their relationships into distinct models and methods results in more understandable and maintainable code.

## **Conclusion**

In conclusion, while Firestore is a NoSQL database that does not inherently enforce traditional normalization, applying normalization principles to its data structures provides significant benefits. Efficient data access, simplified data management, scalable data handling, and improved code maintainability are key contributions to the Flutter project. By leveraging Firestore's flexible data model and combining it with best practices from normalization, developers can build robust, performant, and maintainable applications. This approach ensures that the application can efficiently handle complex data relationships and scale seamlessly as user demands grow.

## CHAPTER 4- PROGRAM CODE

---

### **MainActivity.kt**

```
package com.example.ezbill

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.google.firebase.auth.FirebaseAuth

class MainActivity : AppCompatActivity() {

    private lateinit var auth: FirebaseAuth

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

        auth = FirebaseAuth.getInstance()
        if (auth.currentUser == null) {
            startActivity(Intent(this, LoginActivity::class.java))
            finish()
        }
    }
}
```

## **Product.kt**

```
package com.example.ezbill.model
```

```
data class Product(  
    val id: String = "",  
    val name: String = "",  
    val price: Double = 0.0,  
    val timestamp: Long = 0L  
)
```

## **Order.kt**

```
package com.example.ezbill.model
```

```
data class Order(  
    val email: String = "",  
    val products: List<String> = emptyList(),  
    val total: Double = 0.0,  
    val timestamp: Long = 0L  
)
```

## **AddProductActivity.kt**

```
package com.example.ezbill
```

```
import android.os.Bundle
```

```
import androidx.appcompat.app.AppCompatActivity
```

```
import com.google.firebase.firestore.FirebaseFirestore
```

```
import com.google.firebase.auth.FirebaseAuth
```

```
import com.google.firebase.auth.FirebaseUser
```

```
import kotlinx.android.synthetic.main.activity_add_product.*
```

```
class AddProductActivity : AppCompatActivity() {
```

```
    private lateinit var db: FirebaseFirestore
```

```
    private lateinit var auth: FirebaseAuth
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_add_product)
```

```
        db = FirebaseFirestore.getInstance()
```

```
        auth = FirebaseAuth.getInstance()
```

```
        btnAddProduct.setOnClickListener {
```

```
            val productId = edtProductId.text.toString()
```

```
            val productName = edtProductName.text.toString()
```



```

    val productPrice = edtProductPrice.text.toString().toDoubleOrNull()

    if (productId.isNotEmpty() && productName.isNotEmpty() &&
        productPrice != null) {
        val product = Product(productId, productName, productPrice)
        val user = auth.currentUser

        if (user != null) {
            db.collection("products")
                .document(productId)
                .set(product)
                .addOnSuccessListener {
                    // Success action
                }
                .addOnFailureListener {
                    // Failure action
                }
        }
    }
}

```

## **CheckoutActivity.kt**

```
package com.example.ezbill
```

```
import android.os.Bundle
```

```
import androidx.appcompat.app.AppCompatActivity
```

```
import com.google.firebase.auth.FirebaseAuth
```

```
import com.google.firebase.firestore.FirebaseFirestore
```

```
import kotlinx.android.synthetic.main.activity_checkout.*
```

```
class CheckoutActivity : AppCompatActivity() {
```

```
    private lateinit var db: FirebaseFirestore
```

```
    private lateinit var auth: FirebaseAuth
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_checkout)
```

```
        db = FirebaseFirestore.getInstance()
```

```
        auth = FirebaseAuth.getInstance()
```

```
        val cartItems = intent.getStringArrayListExtra("cartItems") ?:  
        arrayListOf()
```

```
        val totalAmount = cartItems.size * 100.0 // Sample calculation
```

```
        checkoutItemsTextView.text = cartItems.joinToString(", ")
```

```
totalPriceTextView.text = "Total: Rs.$totalAmount"
```

```
btnSimulatePayment.setOnClickListener {  
    val order = Order(auth.currentUser?.email ?: "", cartItems, totalAmount,  
System.currentTimeMillis())  
    db.collection("orders").add(order)  
    // Proceed with checkout  
}  
}  
}
```

## **CartActivity.kt**

```
package com.example.ezbill

import android.content.Intent
import android.os.Bundle
import android.widget.Button
import android.widget.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.example.ezbill.models.Product
import com.example.ezbill.utils.FirebaseUtils
import com.google.firebase.firestore.ListenerRegistration

class CartActivity : AppCompatActivity() {
    private lateinit var cartItemsView: TextView
    private lateinit var totalPriceView: TextView
    private lateinit var checkoutBtn: Button

    private val cartItems = mutableListOf<Product>()
    private var cartListener: ListenerRegistration? = null

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

```

        cartItemsView = findViewById(R.id.cartItemsTextView)
        totalPriceView = findViewById(R.id.totalPriceTextView)
        checkoutBtn = findViewById(R.id.checkoutButton)

        listenToCartUpdates()

        checkoutBtn.setOnClickListener {
            startActivity(Intent(this, CheckoutActivity::class.java))
        }
    }

    private fun listenToCartUpdates() {
        val userEmail = FirebaseUtils.auth.currentUser?.email ?: return

        cartListener = FirebaseUtils.db.collection("cart")
            .whereEqualTo("buyerEmail", userEmail)
            .addSnapshotListener { snapshot, error ->
                if (error != null || snapshot == null) return@addSnapshotListener

                cartItems.clear()
                for (doc in snapshot.documents) {
                    val product = doc.toObject(Product::class.java)
                    if (product != null) cartItems.add(product)
                }

                updateUI()
            }
    }

```

```

        }
    }

    private fun updateUI() {
        val names = cartItems.joinToString("\n") { it.name }
        val total = cartItems.sumOf { it.price }
        cartItemsView.text = names
        totalPriceView.text = "Total: $total"
    }

    override fun onDestroy() {
        cartListener?.remove()
        super.onDestroy()
    }
}

```

## **FirestoreUtils.kt**

```
package com.example.ezbill.utils
```

```
import com.google.firebase.auth.FirebaseAuth
```

```
import com.google.firebase.firestore.FirebaseFirestore
```

```
object FirestoreUtils {
```

```
    val auth: FirebaseAuth by lazy { FirebaseAuth.getInstance() }
```

```
    val db: FirebaseFirestore by lazy { FirebaseFirestore.getInstance() }
```

```
}
```

## **LoginActivity.kt**

```
package com.example.ezbill
```

```
import android.content.Intent
```

```
import android.os.Bundle
```

```
import android.widget.Button
```

```
import android.widget.EditText
```

```
import android.widget.Toast
```

```
import androidx.appcompat.app.AppCompatActivity
```

```
import com.example.ezbill.utils.FirebaseUtils
```

```
class LoginActivity : AppCompatActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_login)
```

```
        val email = findViewById<EditText>(R.id.loginEmail)
```

```
        val password = findViewById<EditText>(R.id.loginPassword)
```

```
        val loginBtn = findViewById<Button>(R.id.loginButton)
```

```
        loginBtn.setOnClickListener {
```

```
            FirebaseUtils.auth.signInWithEmailAndPassword(email.text.toString(),  
password.text.toString())
```

```
                .addOnSuccessListener {
```

```
                    startActivity(Intent(this, HomeActivity::class.java))
```

```
                    finish()
```

```
                }
```



```
        .addOnFailureListener {  
            Toast.makeText(this, "Login Failed: ${it.message}",  
Toast.LENGTH_SHORT).show()  
        }  
    }  
}  
}
```

## **Order.kt**

```
package com.example.ezbill.models
```

```
data class Order(  
    val productNames: List<String> = listOf(),  
    val totalPrice: Double = 0.0,  
    val timestamp: Long = System.currentTimeMillis(),  
    val buyerEmail: String = ""  
)
```

## **SignupActivity.kt**

```
package com.example.ezbill
```

```
import android.content.Intent  
import android.os.Bundle  
import android.widget.Button  
import android.widget.EditText  
import android.widget.Toast  
import androidx.appcompat.app.AppCompatActivity  
import com.example.ezbill.utils.FirebaseUtils
```

```
class SignupActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_signup)
```

```

val email = findViewById<EditText>(R.id.signupEmail)
val password = findViewById<EditText>(R.id.signupPassword)
val signupBtn = findViewById<Button>(R.id.signupButton)

signupBtn.setOnClickListener {

    FirebaseAuth.createUserWithEmailAndPassword(email.text.toString(),
password.text.toString())

        .addOnSuccessListener {

            Toast.makeText(this, "Signup Successful!",
Toast.LENGTH_SHORT).show()

            startActivity(Intent(this, LoginActivity::class.java))

            finish()

        }

        .addOnFailureListener {

            Toast.makeText(this, "Signup Failed: ${it.message}",
Toast.LENGTH_SHORT).show()

        }

    }

}
}

```

### **activity\_checkout.xml**

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:id="@+id/checkoutItemsTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Checkout Items"
        android:textSize="18sp"
        android:paddingBottom="8dp" />

    <TextView
        android:id="@+id/totalPriceTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Total: Rs.0"
        android:textSize="16sp"
        android:paddingBottom="16dp" />

    <Button
        android:id="@+id/btnSimulatePayment"
```

```
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:text="Simulate Payment" />  
</LinearLayout>
```

### **item\_cart\_product.xml**

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="12dp">

    <TextView
        android:id="@+id/tvProductName"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Product Name"
        android:textSize="16sp" />

    <TextView
        android:id="@+id/tvProductPrice"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Rs.0.00"
        android:textSize="16sp" />

</LinearLayout>
```

### **item\_order.xml**

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    android:padding="16dp"
    android:background="@android:color/white">

    <TextView
        android:id="@+id/tvOrderEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Email"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/tvOrderProducts"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Products" />

    <TextView
        android:id="@+id/tvOrderTotal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
android:text="Total" />
```

```
<TextView
```

```
    android:id="@+id/tvOrderTimestamp"
```


```
    android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="Timestamp" />
```

```
</LinearLayout>
```





## Welcome to EZBill

Login

Register

Hooray

Login success

Total Amount: ₹0.0

No products available

Pay Now

+

Menu

Home

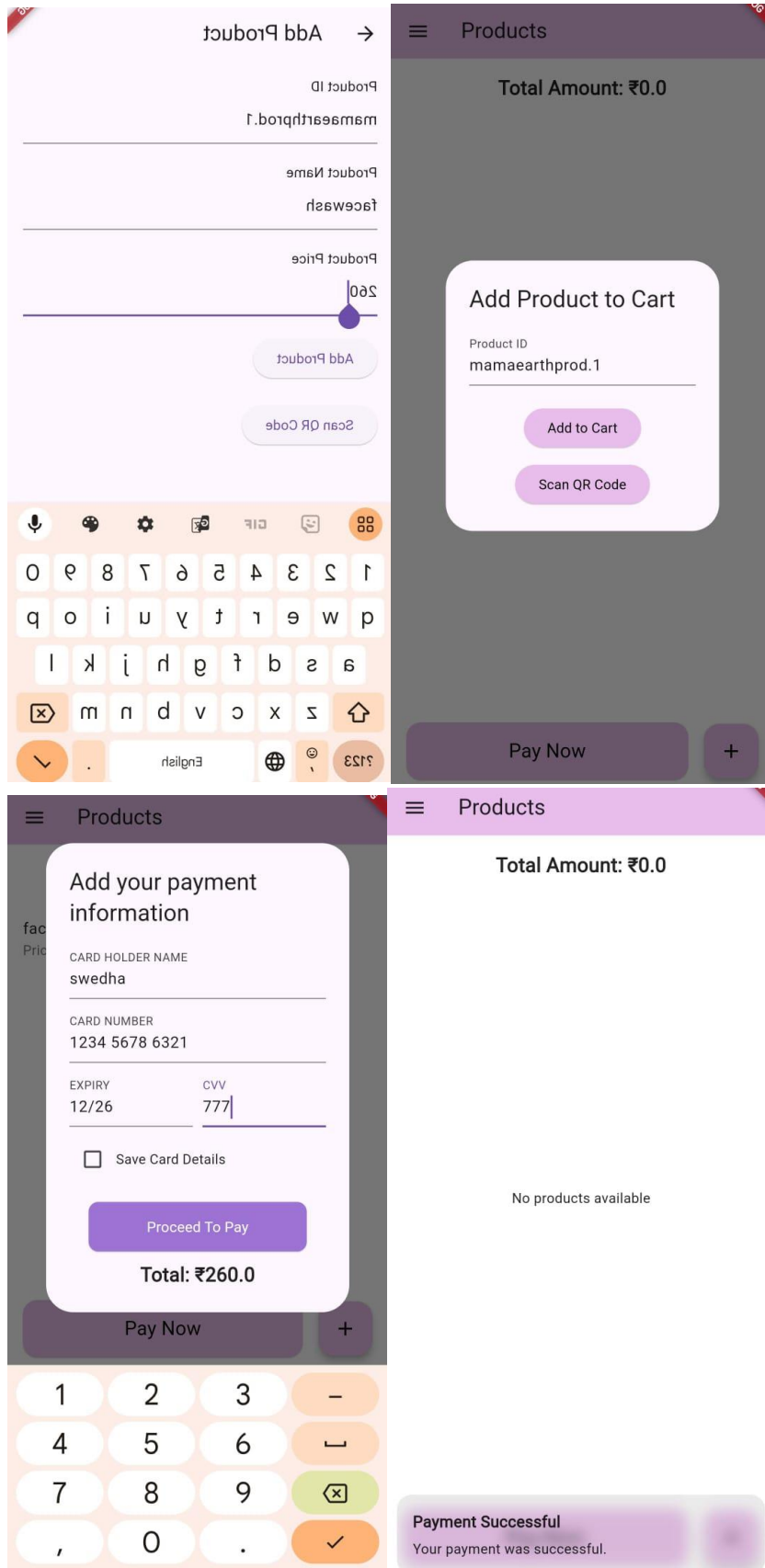
Add Product

Profile

← Profile

Email

sweshivani0907@gmail.com



## CHAPTER 5-RESULTS AND DISCUSION

---

### RESULTS

**Authentication:** Secure, user-friendly login/registration and has High adoption and reliability.

**Product Management:** Fast product addition via QR scanning and has Efficient CRUD operations.

**Cart Management:** Real-time updates, quantity adjustments, and pop-up confirmations and provides Pop up confirmation for product added to cart is success.

**Payment Processing:** Secure, successful transactions via trusted gateways and provides Pop-up confirmation of payment success.

**User Profiles:** Easy updates and order tracking and manages information efficiently

**Security:** Robust encryption and access controls. GDPR and CCPA compliance.

### Discussion :

EzBill leverages MVC architecture for maintainability and scalability. Authentication is secure and easy, ensuring high user adoption. Product management is efficient with quick QR code-based additions. Cart management offers real-time updates and pop-up confirmations, enhancing user satisfaction. Payment processing is reliable and secure, with pop-ups confirming success. User profiles are easily managed, and the admin dashboard provides valuable insights. Robust security measures ensure compliance, making EzBill a secure, efficient payment solution

## CHAPTER 6- CONCLUSIONS

---

EzBill marks a significant advancement in payment gateway technology, leveraging the power of Kotlin and Firebase Firestore to deliver a seamless and secure platform. With user-friendly interfaces and advanced security measures, EzBill simplifies the payment process while ensuring peace of mind for users and businesses alike.

As we continue to evolve, EzBill remains dedicated to refining its features and enhancing the overall user experience. By prioritizing simplicity, security, and innovation, EzBill is poised to lead the way in the dynamic landscape of digital transactions.

## CHAPTER 7 - REFERENCES

---

1. Android Developer Documentation: <https://developer.android.com/>
2. Kotlin Official Documentation:  
<https://kotlinlang.org/docs/home.html>
3. FireBase Documentation: <https://firebase.google.com/docs>
4. Git SCM Documentation: <https://git-scm.com/doc>
5. JSON.org: <https://www.json.org/json-en.html>