CAR ACCESSORIES SHOP MANAGEMENT SYSTEM

A project report submitted to
Bishop Heber College (Autonomous), Tiruchirappalli
affiliated to Bharathidasan University, Tiruchirappalli – 620024
in partial fulfillment of the requirements for the award of the degree of

Bachelor of Vocation in Information Technology

By

S. MOHANRAJ

(REGISTER NO: 225915135)

Under the guidance of
Mrs. P. USHA,MCA .,M.Phil., SET., NET.,

Department of Information Technology

Bishop Heber College(Autonomous)

(Nationally Re-accredited with 'A++' Grade by NAAC at CGPA of 3.69 out of 4)

(Recognized by UGC as "College of Excellence")

Tiruchirappalli 620 017

NOVEMBER 2024

Department of Information Technology
Bishop Heber College(Autonomous)
Tiruchirappalli-620 017,Tamilnadu,India
Phone No:0431-277 0136

CERTIFICATE

This Viva-Voce examination for the candidate **S.MOHANRAJ** **(Reg No:225915135)** was held on_____

Signature of HOD

Examiners:

1.

2.

Mrs. P.USHA, MCA .,M. Phil.,SET.,NET..
Assistant Professor,
Department of Information Technology,
Bishop Heber College (Autonomous),
Tiruchirapalli-620 017

Date:

CERTIFICATE

This is to certify that the project work entitled **"CAR ACCESSORIES SHOP MANAGEMENT SYSTEM"** is a bonafide work done under my supervision by **S.MOHANRAJ (Reg No:225915135)** and submitted to Bishop Heber College (Autonomous), Tiruchirappalli – 620 017 in partial fulfillment of the requirements for the award of the degree of Bachelor of Vocation in Information Technology during the odd semester of the academic year (2024-2025).

Signature of the Guide

DECLARATION

I hereby declare that the work presented in this project work report is the original work done by me under the guidance of **Mrs. P.USHA, MCA., M.Phil., SET., NET.,** Assistant Professor of Information Technology, Bishop Heber College (Autonomous),Tiruchirapalli-620 017 and has not been included in any other project work submitted for any other degree.

Name of the Candidate :S.MOHANRAJ

Register Number :225915135

Semester :FIFTH

Academic Year :2024 – 2025

Course Code :U21ITPJ5

Signature of the Candidate

ACKNOWLEDGEMENTS

First of foremost, I thank **THE ALMIGHTY GOD** for granting abundant grace. Good health and knowledge to do this Project.

I express my sincere gratitude to **Dr.J.PRINCY MERLIN, M.Sc., SET., B.Ed., M.Phil., Ph.D., PGDCI.,** Principal., Bishop Heber College (Autonomous), Tiruchirappalli for providing the opportunity to pursue this programme.

I am highly indebted to thanks **Dr. J. JOHN RAYBIN JOSE, M.sc.,MCA, M.Phill, PGDCA ,Ph.D, SET.** Associate professor and Head , Department of Information Technology , Bishop Heber College (Autonomous), Trichy for providing all the support and facilities to do this project work.

I wish to place on record my gratitude to **Mrs. P.USHA, MCA., M.Phil., SET., NET.,** Assistant Professor in Department of Information Technology, Bishop Heber College (Autonomous), Tiruchirappalli for grant me permission to pursue carry on with this project and guiding me to complete the project successfully.

I thank all the Staff members of the Department of Information Technology for their contribution to complete this project work successfully.

I record my deep sense of gratitude to my beloved parents and my friends for their encouragement and moral support extended during the period of my project.

S.MOHANRAJ

ABSTRACT

The Car Accessories Shop Management System is a web-based solution designed to streamline the operations of a car accessories retail business. Built using HTML, CSS, JavaScript for the front-end, and PHP with MySQL for the back-end, this system offers an efficient way to manage product inventory, customer orders, and sales transactions.

The platform allows shop administrators to manage product details, stock levels, and pricing dynamically through an easy-to-use dashboard. Customers can browse through a wide range of car accessories, view detailed product information, and make purchases by adding items to their cart. The system supports customer registration and login, enabling users to track their order history and manage their profiles.

A robust shopping cart system allows users to select products, specify quantities, and proceed with secure checkout, with the order being stored in the database for further processing. The admin side of the application also features a reporting system, which helps in viewing sales data and generating business insights, making it easier to track daily, weekly, and monthly sales.

The system is designed to be user-friendly, scalable, and secure, providing both customers and administrators with a smooth and efficient shopping experience. Additionally, it is flexible enough to accommodate future enhancements, such as adding new features or integrating third-party services.

CONTENTS

1.INTRODUCTION

The Car Accessories Shop Management System is an application designed for maintaining and managing various aspects of a car accessories store. This project covers the basic functionality of managing inventory, sales, and customer orders within a car accessories shop environment. Car accessory shops play a critical role in the automotive industry by providing a wide range of parts and accessories to enhance the performance, comfort, and aesthetics of vehicles. These shops also offer services such as product installations, maintenance, and customization for vehicles.

The goal of this project is to develop a system that solves the operational needs of a car accessories store, ensuring efficient management of products and customer demands. The system should provide various ways to perform tasks such as inventory tracking, order management, sales processing, and customer service. Additionally, it should enhance the store's workspace with extra functionalities that go beyond the scope of a traditional management system, such as customer feedback tracking and product recommendations.

There are different types of car accessory stores, including those specializing in performance upgrades, aesthetic modifications, and essential maintenance parts. Some stores may focus on performance accessories, offering parts to improve speed, handling, and overall vehicle dynamics. Others may focus on aesthetic enhancements such as custom wheels, lighting, or body kits. There are also general car accessory shops that provide a broad range of products, including basic items like seat covers, air fresheners, and car care products. This system will cater to the specific needs of these different types of car accessory shops, ensuring smooth operations and improved customer satisfaction.

2.SYSTEM STUDY

System analysis is a process of gathering the facts concerning the system, breaking them into elements, and understanding the relationships between these elements. It provides a framework for visualizing the organizational and environmental factors that operate on a system. The quality of work performed by a machine is usually uniform, neat, and more reliable when compared to performing the same operations manually.

2.1.Project Description

The scope of the project is to store and access the database consisting of customer details, product inventories, and transaction records. This data can be shared with the concerned departments, such as sales, inventory management, and customer service, to streamline operations. The Car Accessories Shop Management System is an application that manages information on customer orders, available products, and transactions, typically for small to medium-sized car accessory shops.

2.1.1.Existing System

The existing system operates manually. It involves a lot of complexity and requires significant human effort, including paperwork and manual tracking of sales, inventory, and customer data. All data must be recorded on physical ledgers, making it a tedious and risky process. As the number of transactions increases, so does the volume of data, making data management more challenging. To retrieve any information, an extensive manual search through papers is required, making it time-consuming and inefficient.

2.1.2.Proposed System.

The proposed system is designed to provide the best services to both the car accessory shop and its customers by being user-friendly and reducing the time it takes to complete tasks compared to the current manual system. The new system is highly computerized, ensuring that data related to customer orders, product inventories, and sales are securely stored and managed with high accuracy. This reduces errors caused by human mistakes or machine malfunctions. The system also validates data as it is entered, ensuring data integrity. When necessary, appropriate messages are displayed to prevent user confusion. The data entry screen offers features for viewing records and modifying various types of data as needed.

Advantages:

1. It satisfies the user's requirements.
2. The system generates various types of reports and information for different purposes (sales, inventory, etc.).
3. It is easy for both users and operators to understand.
4. It is simple to operate.
5. The system is expandable, allowing for future upgrades or the addition of new features.

2.1.3 Modules and Description

Banking Application is the add record and delete record to all details.

- **Register Account:**

  This module is used to register an account, by entering customer ID, Name, Account type, and finally your Initial Deposit

- **Profile :**

  This module is used to user can easily view our data.

- **Deposit :**

  To deposit an amount from the account.

- **Withdraw:**

  To withdraw an amount from the account.

- **Transfer Money:**

  To transfer amount from one account to another. by entering the Sender's account number and receiver's account number.

- **Recent Transaction:**

  This module is show the recent transactions(History).

- **Pay Bills:**

  This Module is use to the pay bills.

- **View customer Details :**

  This module is use to view all customer data.

- **View Transaction Details :**

  This module is use to view all transaction details.

2.2. Requirement Specification

Requirements specification involves frequent communication with system users to determine specific feature expectations, resolve conflicts or ambiguities, and ensure that the system meets the needs of its users. The goal is to ensure the system or product conforms to the client's needs rather than forcing users to adapt to predefined requirements. Requirements analysis is a team effort that demands a combination of hardware, software, and human factors engineering, along with strong communication and interpersonal skills.

2.2.1. Hardware Requirement

The hardware specification of the computer system required for developing the **Car Accessories Shop Management System** is as follows:

- **Processor**: Intel Core i3 (or higher)
- **Hard Disk**: 500 GB (or more)
- **RAM**: 8 GB (or more)
- **Keyboard**: Standard Keyboard
- **Mouse**: Standard Mouse Pad

2.2.2. Software Requirement

A **Software Requirement Specification (SRS)** is a detailed description of the system's behavior. It includes the use cases that describe all user interactions with the system. The software tools used for the **Car Accessories Shop Management System** are as follows:

- **Operating System**: Windows 10
- **Software Applications**: WAMP
- **Front End**: PHP, HTML, CSS, JavaScript
- **Back End**: MySQL Database

Operating System

The **Operating System** manages the communication between hardware and software, allowing the system to function. This project will use **Windows 10** , which is widely supported and ideal for running the required software stack.

Development Tools

- **PHP**: PHP (Hypertext Preprocessor) is a widely-used, open-source scripting language that is especially suited for web development and can be embedded in HTML. PHP is server-side and allows the creation of dynamic content and interaction with the database.
- **HTML/CSS**: HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are used for structuring and designing the web pages of the system. HTML creates the structure, while CSS is responsible for the layout, design, and responsiveness.
- **JavaScript**: JavaScript is a client-side scripting language used to add dynamic elements, interactivity, and enhanced functionality to the web pages. It enables features such as form validation, dynamic content updates, and user interaction with the interface without reloading the page.

Database:

- **MySQL**: MySQL is an open-source relational database management system that will be used to manage all data related to customers, products, orders, and transactions. MySQL is known for its reliability, ease of use, and support for high transaction loads.

3.SYSTEM DESIGN

System design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. System design could be seen as the application of system theory to product development.

3.1. Logical Design

Logical design is an abstract concept in computer programming by which programmers arrange data in a series of logical relationships known as attributes or entities. An entity refers to a chunk of information, whereas an attribute defines the unique properties of an entity.

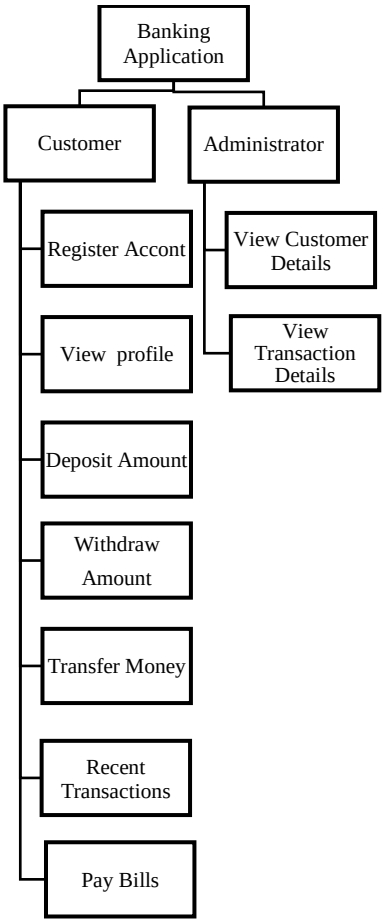

Fig.3.1 Banking Application

3.2 Database Design

Database design is the process of producing a detailed data model of the database. This data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a data definition language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity. A good database design is important in ensuring consistent data, elimination of data redundancy, efficient execution of queries and high-performance application.

Users Table :

Field	Type
Full name	String
Dob	Date
Address	String
Phone Number	Integer
Email Address	String
Occupation	String
Pan Card	String
Aadhar	Integer
Balance	Double
Account Number	Integer

RecentTransactions Table :

Field	Type
Sender	String
Recipient	String
Transaction Id	String
Amount	Double

4.SYSTEM DEVELOPMENT

The Software Development Life Cycle(SDLC), also referred to as the application development life-cycle, is a term used in system engineering, information system and software engineering to describe a process for planing, creating testing and deploying an information system. The system developments life-cycle concept applies to a range of hardware and software configurations, as a system can be composed of hardware only, software only, or a combination of both.

4.1 Program Code

Here I displayed the coding of my program:

LoginActivity.java:

```
package com.example.bankms;
public class LoginActivity extends AppCompatActivity {
    private EditText usernameEditText, passwordEditText;
    private Button loginButton;
     private TextView signupTextView;
     private FirebaseAuth firebaseAuth;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_login);
usernameEditText = findViewById(R.id.usernameEditText);
passwordEditText = findViewById(R.id.passwordEditText);
loginButton = findViewById(R.id.loginButton);
signupTextView = findViewById(R.id.signupTextView);
firebaseAuth = FirebaseAuth.getInstance();
loginButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
String username = usernameEditText.getText().toString().trim();
String password = passwordEditText.getText().toString().trim();
if (!username.isEmpty() && !password.isEmpty()) {
signIn(username, password);
} else {
Toast.makeText(LoginActivity.this, "Please enter email and password",
Toast.LENGTH_SHORT).show();}} });
signupTextView.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Intent intent = new Intent(LoginActivity.this, SignupActivity.class);
startActivity(intent);}}); }
private void signIn(String email, String password) {
firebaseAuth.signInWithEmailAndPassword(email, password)
.addOnCompleteListener(this, task -> {
if (task.isSuccessful()) {
FirebaseUser user = firebaseAuth.getCurrentUser();
```

```
Toast.makeText(LoginActivity.this, "Login successful.", Toast.LENGTH_SHORT).show();
if (user != null && user.getEmail() != null && user.getEmail().equals("admin@gmail.com")) {
Intent intent = new Intent(LoginActivity.this, Admin.class);
startActivity(intent);
} else {
Intent intent = new Intent(LoginActivity.this, DashboardActivity.class);
startActivity(intent);}
} else {
Toast.makeText(LoginActivity.this, "Authentication failed.", Toast.LENGTH_SHORT).show();} });}}
```

SignupActivity.java:

```
package com.example.bankms;

public class SignupActivity extends AppCompatActivity {

    private EditText emailEditText, passwordEditText;

     private Button signupButton;

      private FirebaseAuth firebaseAuth;

 @Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

  setContentView(R.layout.activity_signup);

 emailEditText = findViewById(R.id.usernameEditText);

passwordEditText = findViewById(R.id.passwordEditText);

 signupButton = findViewById(R.id.signupButton);

 firebaseAuth = FirebaseAuth.getInstance();

 signupButton.setOnClickListener(new View.OnClickListener() {

 @Override

public void onClick(View v) {

String email = emailEditText.getText().toString().trim();

String password = passwordEditText.getText().toString().trim();

 if (!email.isEmpty() && !password.isEmpty()) {

createAccount(email, password);

 } else {

Toast.makeText(SignupActivity.this,"Pleaseenteremailandpassword",Toast.LENGTH_SHORT).show(
);}} }); }

private void createAccount(String email, String password) {

 firebaseAuth.createUserWithEmailAndPassword(email, password)

.addOnCompleteListener(this, new OnCompleteListener() {

@Override
```

```
 public void onComplete(@NonNull Task task) {
 if (task.isSuccessful()) {
Toast.makeText(SignupActivity.this, "Sign up successful.", Toast.LENGTH_SHORT).show();
finish();
} else {
Toast.makeText(SignupActivity.this,"Signupfailed."+task.getException().getMessage(),
Toast.LENGTH_SHORT).show();}}});}}
```

Dashboard Activity:

```
public class DashboardActivity extends AppCompatActivity {
private DatabaseReference databaseReference;
private FirebaseUser user;
Private Button addAccountButton ,profile ,deposite ,withdraw ,transfermoney , recentTransaction
,payBill ,logoutButton;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_dashboard);
addAccountButton = findViewById(R.id.addaccount);
profile=findViewById(R.id.profile);
deposite=findViewById(R.id.deposite);
withdraw=findViewById(R.id.withdraw);
transfermoney=findViewById(R.id.transferMoneyButton);
recentTransaction=findViewById(R.id.recentTransactionsButton);
payBill=findViewById(R.id.payBillsButton);
logoutButton = findViewById(R.id.logout);
databaseReference = FirebaseDatabase.getInstance().getReference("users");
user = FirebaseAuth.getInstance().getCurrentUser();
checkIfEmailExists();
addAccountButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
startActivity(new Intent(DashboardActivity.this,CreateAccountActivity.class));}});
profile.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
startActivity(new Intent(DashboardActivity.this, Profile.class));
}  });
deposite.setOnClickListener(new View.OnClickListener() {
```

```
 @Override
public void onClick(View v) {
 startActivity(new Intent(DashboardActivity.this, Deposite.class));
 } });
withdraw.setOnClickListener(new View.OnClickListener() {
@Override
 public void onClick(View v) {
startActivity(new Intent(DashboardActivity.this, Withdraw.class));
}});
transfermoney.setOnClickListener(new View.OnClickListener() {
@Override
 public void onClick(View v) {
 startActivity(new Intent(DashboardActivity.this, TransferMoney.class));} });
recentTransaction.setOnClickListener(new View.OnClickListener() {@Override
public void onClick(View v) {
 startActivity(newIntent(DashboardActivity.this, TransactionActivity.class ));}});
payBill.setOnClickListener(new View.OnClickListener() {
@Overridepublic void onClick(View v) {
startActivity(new Intent(DashboardActivity.this, PayBills.class));
}});logoutButton.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View view) {
 logoutUser();} }); }
 private void logoutUser() {
FirebaseAuth.getInstance().signOut();
 Intent intent = new Intent(DashboardActivity.this, LoginActivity.class);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK | Intent.FLAG_ACTIVITY_CLEAR_TASK);
startActivity(intent);
finish(); }
private void checkIfEmailExists() {
if (user != null) {
 String userEmail = user.getEmail();
 Queryquery = databaseReference.orderByChild("emailAddress").equalTo(userEmail);
query.addListenerForSingleValueEvent(new ValueEventListener() {
@Override
 public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
if (dataSnapshot.exists()) {
addAccountButton.setVisibility(View.GONE);
} else {
addAccountButton.setVisibility(View.VISIBLE); }}});}}}
```

RegisterAccountActivity:

```
package com.example.bankms;
public class CreateAccountActivity extends AppCompatActivity {
   private EditText fullNameEditText, dobEditText, addressEditText, phoneNumberEditText, emailAddressEditText,
         occupationEditText, pancardEditText, aadharEditText, balanceEditText;
   private Button createAccountButton;
   private DatabaseReference databaseReference;
   private FirebaseAuth firebaseAuth;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_create_account);
      databaseReference = FirebaseDatabase.getInstance().getReference("users");
      fullNameEditText = findViewById(R.id.etFullName);
      dobEditText = findViewById(R.id.etDateOfBirth);
      addressEditText = findViewById(R.id.etAddress);
      phoneNumberEditText = findViewById(R.id.etPhoneNumber);
      emailAddressEditText = findViewById(R.id.etEmailAddress);
      occupationEditText = findViewById(R.id.etOccupation);
      pancardEditText = findViewById(R.id.etPancard);
      aadharEditText = findViewById(R.id.etAadharNo);
      balanceEditText = findViewById(R.id.etInitialAmount);
      createAccountButton = findViewById(R.id.btnCreateAccount);
      createAccountButton.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View view) {
            createAccount();}});}
   private void createAccount() {
      String fullName = fullNameEditText.getText().toString().trim();
      String dob = dobEditText.getText().toString().trim();
      String address = addressEditText.getText().toString().trim();
      String phoneNumber = phoneNumberEditText.getText().toString().trim();
      String emailAddress = emailAddressEditText.getText().toString().trim();
      String occupation = occupationEditText.getText().toString().trim();
      String pancard = pancardEditText.getText().toString().trim();
      String aadhar = aadharEditText.getText().toString().trim();
      String balance = balanceEditText.getText().toString().trim();
      if (fullName.isEmpty() || dob.isEmpty() || address.isEmpty() || phoneNumber.isEmpty() ||
```

```
            emailAddress.isEmpty() || occupation.isEmpty() || pancard.isEmpty() || aadhar.isEmpty() ||
balance.isEmpty()) {
        Toast.makeText(this, "Please fill in all the details", Toast.LENGTH_SHORT).show();
        return;}
      String userID = FirebaseAuth.getInstance().getCurrentUser().getUid();
      generateAccountNumber(new OnAccountNumberGeneratedListener() {
        @Override
        public void onAccountNumberGenerated(String accountNumber) {
          if (accountNumber != null) {
            saveUserDetails(userID, fullName, dob, address, phoneNumber, emailAddress,
                occupation, pancard, aadhar, balance, accountNumber);
            Toast.makeText(CreateAccountActivity.this, "Account created successfully",
Toast.LENGTH_SHORT).show();
          } else {
            Toast.makeText(CreateAccountActivity.this, "Error generating account number",
Toast.LENGTH_SHORT).show();}}});}
   private void generateAccountNumber(OnAccountNumberGeneratedListener listener) {
      DatabaseReference countersRef = databaseReference.child("counters");
      DatabaseReference accountNumberCounterRef = countersRef.child("accountNumberCounter");
      accountNumberCounterRef.runTransaction(new Transaction.Handler() {
        @NonNull
        @Override
        public Transaction.Result doTransaction(@NonNull MutableData mutableData) {
          Long currentAccountNumber = mutableData.getValue(Long.class);
          if (currentAccountNumber == null) {
            currentAccountNumber = 240300L;  // Initialize with a default value
          }
          mutableData.setValue(currentAccountNumber + 1);
          return Transaction.success(mutableData);
        }@Override
        public void onComplete(@NonNull DatabaseError databaseError, boolean b, @NonNull
DataSnapshot dataSnapshot) {
          if (databaseError == null) {
            Long newAccountNumber = dataSnapshot.getValue(Long.class);
listener.onAccountNumberGenerated(String.valueOf(newAccountNumber));
          } else {
            listener.onAccountNumberGenerated(null);}}});}
   interface OnAccountNumberGeneratedListener {
      void onAccountNumberGenerated(String accountNumber);}
private void saveUserDetails(String userID, String fullName, String dob, String address,
```

```
String phoneNumber, String emailAddress, String occupation,
String pancard, String aadhar, String balance, String accountNumber) {
        DatabaseReference userRef = databaseReference.child(userID);
        Map<String, Object> userDetails = new HashMap<>();
        userDetails.put("fullName", fullName);
        userDetails.put("dob", dob);
        userDetails.put("address", address);
        userDetails.put("phoneNumber", phoneNumber);
        userDetails.put("emailAddress", emailAddress);
        userDetails.put("occupation", occupation);
        userDetails.put("pancard", pancard);
        userDetails.put("aadhar", aadhar);
        userDetails.put("balance", balance);
        userDetails.put("accountNumber", accountNumber);
        userRef.setValue(userDetails);}
    interface OnIDGeneratedListener {
        void onIDsGenerated(String userID, String accountNumber);}}
```

ProfileActivity:

```
package com.example.bankms;
public class Profile extends AppCompatActivity {
    private TextView fullNameTextView, emailTextView, balanceTextView,acc,occ,phno,add;
    private FirebaseAuth firebaseAuth;
    private FirebaseUser user;
    private DatabaseReference databaseReference;
    @SuppressLint("MissingInflatedId")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_profile);
        firebaseAuth = FirebaseAuth.getInstance();
        databaseReference = FirebaseDatabase.getInstance().getReference("users");
        user = firebaseAuth.getCurrentUser();
        fullNameTextView = findViewById(R.id.tvFullName);
        emailTextView = findViewById(R.id.tvEmail);
        balanceTextView = findViewById(R.id.tvBalance);
        acc=findViewById(R.id.tvano);
```

```
        add=findViewById(R.id.tvadd);
        occ=findViewById(R.id.tvoccu);
        phno=findViewById(R.id.tvphno);
        if (user != null) {
            loadUserProfile();}}
    private void loadUserProfile() {
        String userId = user.getUid();
        databaseReference.child(userId).addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    String fullName = dataSnapshot.child("fullName").getValue(String.class);
                    String accountno1=dataSnapshot.child("accountNumber").getValue(String.class);
                    String phno1=dataSnapshot.child("phoneNumber").getValue(String.class);
                    String oc1=dataSnapshot.child("occupation").getValue(String.class);
                    String email = dataSnapshot.child("emailAddress").getValue(String.class);
                    String add1=dataSnapshot.child("address").getValue(String.class);
                    String balance = dataSnapshot.child("balance").getValue(String.class);
                    fullNameTextView.setText("Name :"+fullName);
                    emailTextView.setText("Email :"+email);
                    balanceTextView.setText("Balance :"+balance);
                    phno.setText("PhoneNumber :"+phno1);
                    occ.setText("Occupation :"+oc1);
                    add.setText("Address :"+add1);
                    acc.setText("Account Number :"+accountno1)
                });}}
```

WithdrawActivity:

```
package com.example.bankms;
public class Withdraw extends AppCompatActivity {
    private EditText amountEditText;
    private TextView balanceTextView;
    private Button withdrawButton;
    private FirebaseAuth firebaseAuth;
```

```
    private DatabaseReference userRef;
    private DatabaseReference transactionsRef;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_withdraw);
        firebaseAuth = FirebaseAuth.getInstance();
        FirebaseUser currentUser = firebaseAuth.getCurrentUser();
        if (currentUser != null) {
            String uid = currentUser.getUid();
            userRef = FirebaseDatabase.getInstance().getReference().child("users").child(uid);
            transactionsRef = FirebaseDatabase.getInstance().getReference().child("transactions");
            updateBalance();
            amountEditText = findViewById(R.id.etWithdrawAmount);
            balanceTextView = findViewById(R.id.tvWithdrawBalance);
            withdrawButton = findViewById(R.id.btnWithdraw);
            withdrawButton.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    withdrawAmount();}});}}
    private void updateBalance() {
        userRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    String balance = dataSnapshot.child("balance").getValue(String.class);
if (balance != null) {
balanceTextView.setText("Current Balance: ₹" + balance)} else {
balanceTextView.setText("Balance not available");}} else {
balanceTextView.setText("User not found");}}
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {} }); }
    private void withdrawAmount() {
        String amountStr = amountEditText.getText().toString().trim();
```

```
        if (!amountStr.isEmpty()) {
            double amount = Double.parseDouble(amountStr);
            userRef.addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                    if (dataSnapshot.exists()) {
                        String balance1 = dataSnapshot.child("balance").getValue(String.class);
                        double currentBalance = Double.parseDouble(balance1);
                        if (amount <= currentBalance) {
                            double newBalance = currentBalance - amount;
                            String stringValue = String.valueOf(newBalance);
                            userRef.child("balance").setValue(stringValue);
                            saveTransaction(amount);
                            Toast.makeText(Withdraw.this, "Withdrawal successful",
Toast.LENGTH_SHORT).show();
                            updateBalance();
                        } else {
                            Toast.makeText(Withdraw.this, "Insufficient balance",
Toast.LENGTH_SHORT).show();}}}
                @Override
                public void onCancelled(@NonNull DatabaseError databaseError) {}});
        } else {
            Toast.makeText(this, "Please enter a valid amount", Toast.LENGTH_SHORT).show();}}
    private void saveTransaction(double amount) {
        String transactionId = transactionsRef.push().getKey();
        String uid = firebaseAuth.getCurrentUser().getUid();
        Transaction1 transaction = new Transaction1(transactionId, uid, "Withdraw", amount);
        transactionsRef.child(transactionId).setValue(transaction);}}
```

TransferMoneyActivity:

```
package com.example.bankms;
public class TransferMoney extends AppCompatActivity {
    private EditText recipientIdentifierEditText, transferAmountEditText;
    private Button transferButton;
    private DatabaseReference databaseReference;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_transfer_money);
        databaseReference = FirebaseDatabase.getInstance().getReference("users");
        recipientIdentifierEditText = findViewById(R.id.etRecipientIdentifier);
        transferAmountEditText = findViewById(R.id.etTransferAmount);
        transferButton = findViewById(R.id.btnTransfer);
        transferButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {transferMoney();}});}
    private void transferMoney() {
        String recipientIdentifier = recipientIdentifierEditText.getText().toString().trim();
        String transferAmountStr = transferAmountEditText.getText().toString().trim();
        if (recipientIdentifier.isEmpty() || transferAmountStr.isEmpty()) {
            Toast.makeText(this, "Please enter both recipient identifier and transfer amount",
Toast.LENGTH_SHORT).show();
            return;
        }
        double transferAmount = Double.parseDouble(transferAmountStr);
        transferMoneyToRecipient(recipientIdentifier, transferAmount);
    }
    private void transferMoneyToRecipient(String recipientIdentifier, double transferAmount) {
        FirebaseUser currentUser = FirebaseAuth.getInstance().getCurrentUser();
        if (currentUser != null) {
            String senderUID = currentUser.getUid();
            checkBalance(senderUID, transferAmount, new OnBalanceCheckListener() {
                @Override
                public void onBalanceCheck(boolean isSufficient, double currentBalance) {
                    if (isSufficient) {
                        updateBalance(senderUID, currentBalance - transferAmount);
                        findRecipient(recipientIdentifier, new OnRecipientUIDListener() {
                            @Override
                            public void onRecipientUID(String recipientUID) {
                                if (recipientUID != null) {
                                    checkRecipientBalance(recipientUID, new OnRecipientBalanceListener() {
                                        @Override
                                        public void onRecipientBalance(double recipientCurrentBalance) {
                                            updateBalance(recipientUID, recipientCurrentBalance + transferAmount);
                                            Toast.makeText(TransferMoney.this,
```

```
                                "Money transferred successfully", Toast.LENGTH_SHORT).show();
 finish();}});} else {
                        Toast.makeText(TransferMoney.this, "Recipient not found",
Toast.LENGTH_SHORT).show();
}}});} else {
                Toast.makeText(TransferMoney.this, "Insufficient balance",
Toast.LENGTH_SHORT).show();}}});}}
   private void findRecipient(String recipientIdentifier, OnRecipientUIDListener listener) {
      DatabaseReference usersRef = FirebaseDatabase.getInstance().getReference("users");
      Query query;
      if (recipientIdentifier.matches("\\d{10}")) {
         query = usersRef.orderByChild("phoneNumber").equalTo(recipientIdentifier);
      } else {
         query = usersRef.orderByChild("accountNumber").equalTo(recipientIdentifier);}
      query.addListenerForSingleValueEvent(new ValueEventListener() {
         @Override
         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.exists()) {
               String recipientUID = dataSnapshot.getChildren().iterator().next().getKey();
               listener.onRecipientUID(recipientUID);
            } else {
               listener.onRecipientUID(null);}}
         @Override
         public void onCancelled(@NonNull DatabaseError databaseError) {
            listener.onRecipientUID(null);}});}
   private void checkRecipientBalance(String recipientUID, OnRecipientBalanceListener listener) {
      DatabaseReference recipientRef =
FirebaseDatabase.getInstance().getReference("users").child(recipientUID);
      recipientRef.addListenerForSingleValueEvent(new ValueEventListener() {
         @Override
         public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
            if (dataSnapshot.exists()) { String
recurrentbalance=dataSnapshot.child("balance").getValue(String.class);
               double recipientCurrentBalance =Double.parseDouble(recurrentbalance);
               listener.onRecipientBalance(recipientCurrentBalance);
            } else {
               listener.onRecipientBalance(0);}}
         @Override
         public void onCancelled(@NonNull DatabaseError databaseError) {
            listener.onRecipientBalance(0);}});}
```

```
    interface OnRecipientUIDListener {
        void onRecipientUID(String recipientUID);}
    interface OnRecipientBalanceListener {
        void onRecipientBalance(double recipientCurrentBalance);}
    interface OnBalanceCheckListener {
        void onBalanceCheck(boolean isSufficient, double currentBalance);}
    private void checkBalance(String senderUID, double transferAmount, OnBalanceCheckListener
listener) {
        DatabaseReference senderRef =
FirebaseDatabase.getInstance().getReference("users").child(senderUID);
        senderRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    String currentbalance1=dataSnapshot.child("balance").getValue(String.class);
                    double currentBalance =Double.parseDouble(currentbalance1);
                    boolean isSufficient = currentBalance >= transferAmount;
                    listener.onBalanceCheck(isSufficient, currentBalance);
                } else {
                    listener.onBalanceCheck(false, 0);}}
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
                listener.onBalanceCheck(false, 0);}});}
    private void updateBalance(String uid, double newBalance) {
        DatabaseReference userRef = FirebaseDatabase.getInstance().getReference("users").child(uid);
        String newbalance1=String.valueOf(newBalance);
        userRef.child("balance").setValue(newbalance1);}}
```

RecentTransactionActivity:

```
package com.example.bankms;
public class TransactionActivity extends AppCompatActivity {
    private RecyclerView recyclerView;
    private TransactionAdapter adapter;
    private List<Transaction1> transactionList;
    private DatabaseReference transactionsRef;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_transaction);
        recyclerView = findViewById(R.id.transactionRecyclerView);
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
        transactionsRef = FirebaseDatabase.getInstance().getReference("transactions");
        transactionList = new ArrayList<>();
        adapter = new TransactionAdapter(this, transactionList);
        recyclerView.setAdapter(adapter);
        loadTransactions()}
    private void loadTransactions() {
        transactionsRef.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                transactionList.clear();
                for (DataSnapshot snapshot : dataSnapshot.getChildren()) {
                    Transaction1 transaction = snapshot.getValue(Transaction1.class);
                    transactionList.add(transaction)}
                adapter.notifyDataSetChanged();}
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
                Toast.makeText(TransactionActivity.this, "Failed to load transactions.",
Toast.LENGTH_SHORT).show();}});}}
```

PayBillsActivity:

```
package com.example.bankms;
public class PayBills extends AppCompatActivity {
    private Spinner billTypeSpinner;
    private EditText amountEditText;
    private Button payButton;
    private FirebaseAuth firebaseAuth;
    private DatabaseReference userRef;
    private DatabaseReference transactionsRef;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pay_bills);
        firebaseAuth = FirebaseAuth.getInstance();
        FirebaseUser currentUser = firebaseAuth.getCurrentUser();
        billTypeSpinner = findViewById(R.id.billTypeSpinner);
        amountEditText = findViewById(R.id.amountEditText);
        payButton = findViewById(R.id.payButton);
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
                this, R.array.bill_types, android.R.layout.simple_spinner_item);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        billTypeSpinner.setAdapter(adapter);
        if (currentUser != null) {
            String uid = currentUser.getUid();
            userRef = FirebaseDatabase.getInstance().getReference().child("users").child(uid);
            transactionsRef = FirebaseDatabase.getInstance().getReference().child("transactions");
updateBalance();
            payButton.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    withdrawAmount();}});}}
    private void updateBalance() {
        userRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                if (dataSnapshot.exists()) {
                    String balance = dataSnapshot.child("balance").getValue(String.class);} }
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {}})}
    private void withdrawAmount() {
        String amountStr = amountEditText.getText().toString().trim();
        if (!amountStr.isEmpty()) {
            double amount = Double.parseDouble(amountStr);
            userRef.addListenerForSingleValueEvent(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
```

```
                if (dataSnapshot.exists()) {
                    String balance1 = dataSnapshot.child("balance").getValue(String.class);
                    double currentBalance = Double.parseDouble(balance1);
                    if (amount <= currentBalance) {
                        double newBalance = currentBalance - amount;
                        String stringValue = String.valueOf(newBalance);
                        userRef.child("balance").setValue(stringValue);
                        saveTransaction(amount);
                        Toast.makeText(PayBills.this, "Bill payed successful",
Toast.LENGTH_SHORT).show();
                        updateBalance();
                    } else {
                        Toast.makeText(PayBills.this, "Insufficient balance",
Toast.LENGTH_SHORT).show();}}}
            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
            })} else {
          Toast.makeText(this, "Please enter a valid amount", Toast.LENGTH_SHORT).show();}}
    private void saveTransaction(double amount) {
        String transactionId = transactionsRef.push().getKey();
        String uid = firebaseAuth.getCurrentUser().getUid();
        Transaction1 transaction = new Transaction1(transactionId, uid, "Bill Payed", amount);
        transactionsRef.child(transactionId).setValue(transaction);}}
```

5.SYSTEM TESTING

System testing is the process of evaluation and software item to detect differences between given input and expected output.Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words, software testing is a verification and validation process.

5.1. Unit Testing:

Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. may require developing test driver modules or test harnesses.

Fig:5.1 Login Page

5.2. Integration Testing:

Testing of integrated modules to verify combined functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

Fig:5.2 Register Account Page

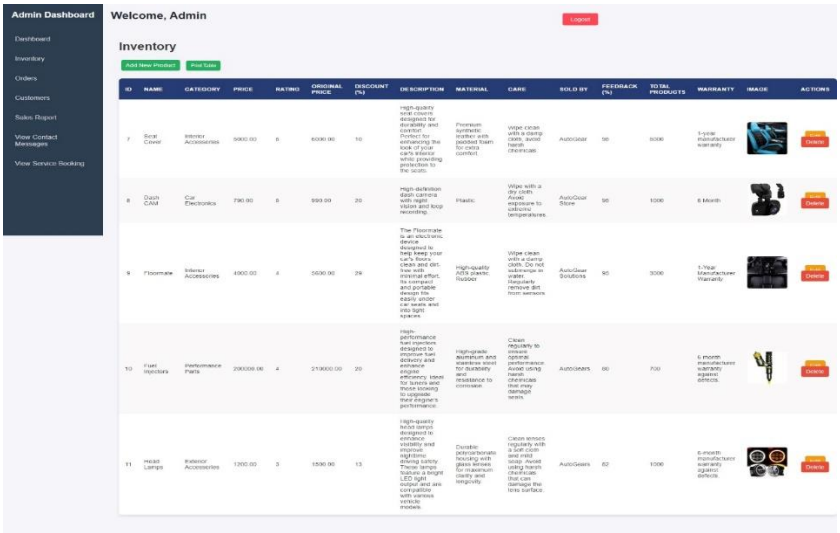

Fig:5.3 View User Details Page

5.3. Validation Testing:

The process of validating a software product, or determining if it meets high level criteria, involves determining whether it is up to par. It is the procedure used to verify that the product we are producing is the proper one. The actual and anticipated product are being validated.

Testing in motion is validation.

Fig:5.4 Validation Testing on Login

1. SYSTEM IMPLEMENTATION

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus, it can be considering the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

6.1 SOFTWARE DEMONSTRATION

6.1.1 Login Page

A user authentication interface allowing access to an Android app by entering credentials such as email and password.

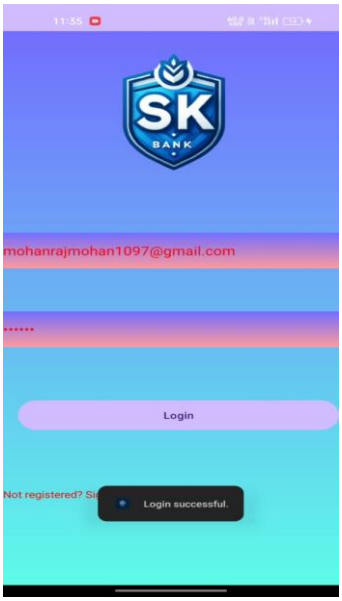

Fig:6.1 Login Page

6.1.2 Signup Page

An interface within the Android app enabling users to input email and password to signup.

Fig:6.2 SignUp Page

6.1.3 Dashboard

The main screen of the Android app, serving as the entry point for users.It typically showcases essential features, content, or navigation options for easy access.

Fig:6.3 Dashboard

6.1.4 RegisterAccount

A page within the Android app where users can input information to register and create a new account.It typically collects personal details and credentials necessary for account creation..

Fig:6.4 Register Account

6.1.5 Profile

The page displays a user's personal information, such as their name, accountno., bio, and contact details. It serves as a hub for users to manage their settings, preferences, and interactions within an application.

Fig:6.5 Profile

6.1.6 DepositAmount

The deposit amount page allows users to input and confirm the amount of money they want to deposit into their account.

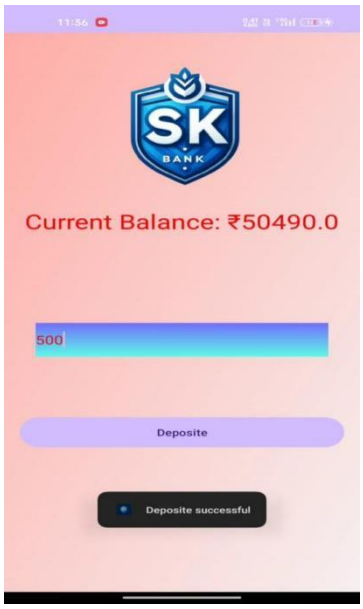

Fig:6.6 Deposite

6.1.7 WithdrawAmount

The withdraw amount page allows users to input and confirm the amount of money they want to withdraw into their account.

Fig:6.7 Withdraw

6.1.8 TransferMoney

The transfer money using account number page facilitates users in sending funds to another account by inputting the recipient's account number or phonr number and specifying the transfer amount.

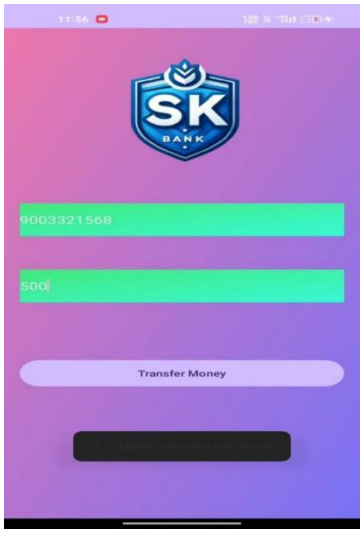

Fig:6.8 Transfer Money

6.1.9 RecentTransactions

The recent transaction page displays your latest banking activities, providing a clear overview of your financial transactions. It allows you to track your spending, monitor deposits, and stay informed about your account's activity.

Fig:6.9 Recent Transactions

6.1.10 PayBills

The pay bills page streamlines bill payment, enabling quick and efficient settlement of obligations. It offers a convenient platform to schedule payments, manage recurring expenses, and maintain financial organization.

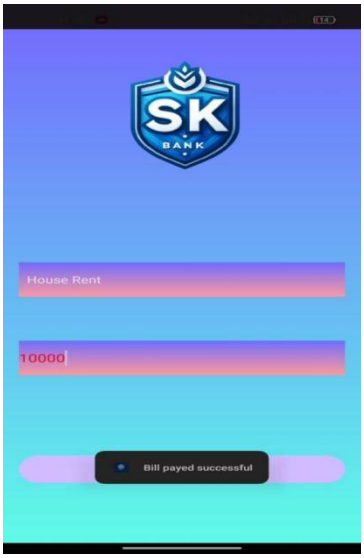

Fig:6.10 Pay Bills

6.1.11 ViewCustomerDetails

The view customer details page offers comprehensive insights into client information, facilitating personalized service." "It provides a consolidated view of account status, transaction history, and contact details for efficient customer management.

Fig:6.11 View Customer Details

6.1.12 View TransactionDetails

The view transaction details page offers a detailed breakdown of individual transactions, including , amount, and sender-id. It provides a comprehensive overview of account activity, empowering admin to track expenses and reconcile their finances.

7. CONCLUSION

In **"Banking Application,"** the platform offers a user-friendly interface, efficient management of banking services, and convenient features for both consumers and administrators. Administrators can view customer details and transaction details. It allows customers to easily register, view their profiles, make deposits and withdrawals, transfer money, view transaction details, and make bill payments. The platform is compatible across different Android versions and is committed to continuous improvement to enhance user productivity and satisfaction. Future development aims to evolve into an advanced tool for smooth financial transactions and efficient banking service administration.

BIBLIOGRAPHY

BOOK REFERENCE:

1. Barry A. Burd, “Android Application Development”,2020.

2. Lauren Darcey, “Introduction to Android Application”,2013.

WEBSITE REFRENCES:

1. https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-1-get-started/lesson-1-build-your-first-app/1-0-c-introduction-to-android/1-0-c-introduction-to-android.html.

2. https://developer.android.com/training/basics/firstapp.

3. https://github.com/AppIntro/AppIntro.