CHAPTER-1

INTRODUCTION

Mobile Application Development is the process of creating software applications that run on a mobiledevice, and a typical mobile application utilizes a network connection to work with remote computing resources. Hence, the mobile development process involves creating installable software bundles, implementing backend services such as data access with an API, and testing the application on target devices. There are four major development approaches when building mobile applications.

- Native Mobile Applications
- Cross- Platform Native Mobile Applications
- Hybrid Mobile Applications
- Progressive Web Applications

Each of these approaches for developing mobile applications has its own set of advantages and disadvantages, when choosing the right development approach for their projects, developers consider the desired user experience, the computing resources and native features required by the app the development budget, time target, and resources available to maintain the app.

1.1 Overview about the Topic

Mobile banking applications are software applications designed to allow users to access and manage their bank accounts using their mobile devices. They provide a range of services, including account balance inquiries, fund transfers, bill payments, loan applications, account statements, and more. The purpose of this project is to provide an overview of mobile banking applications, their benefits, functionalities, and their impact on the banking industry. Mobile banking applications have revolutionized the way people manage their finances, offering convenience, accessibility, and security to users.

1.2 Problem Statement

Mobile banking applications provide convenient access to banking services anytime, anywhere. Without them, people may have to rely on physical bank branches, which may be limited in number or have restricted operating hours. This could result in long waiting times and inconvenience for individuals who need to carry out banking transactions. Without mobile banking applications, individuals may have to visit the bank in person to complete transactions such as fund transfers, bill payments, or account inquiries.

1.3 Objective

Objective for Android Mobile Banking Application :

- Develop a robust and intuitive mobile banking application for the Android platform.
- Provide users with a range of banking functionalities, including account management, fund transfers, bill payments, and loan applications.
- Ensure the security of user information and transactions through the implementation of advanced security measures.
- Enhance the user experience by incorporating intuitive design, easy navigation, and personalized features.
- Comply with industry standards and regulatory requirements for mobile banking applications.

CHAPTER-2

SYSTEM REQUIREMENT SPECIFICATION

A software requirements specification (SRS) is a description of a software system to developed. It laysout functional and non-functional requirements, and may include a set of use cases that describe user interactions that the software must provide. This section introduces the requirement specification document for the Student Management System. It provides the purpose and scope of the system. Any definition and references are listed in this section as well as an overview of the remaining requirementsspecification document. It describes the design decisions, architectural design and the detailed design needed to implement the system. It provides the visibility in the design and provides information neededfor software supports. It'll help you with your daily student management routines and deliver you fromyour paperwork. In order to make this possible, the system greatly decreases the amount of paperwork.Student Management System is developing for general purpose and used to replace old paper work system. Student Management System is built in order to efficiently provide student information to teachers and school administration. This increases in efficiency of result making, provide results. It provides a mechanism to edit the student information form which makes the system flexible.

2.1 Hardware Requirements

- Android Device with Screen resolution of at least 480 x 800 hdpi or above.
- Device should be connected with the network. It may be LAN or WAN.

2.1.1 Functional Requirements

This will allow access only to authorized users with specific roles. Depending upon the user's role, he/she will be able to access only specific modules of the system.

- User registration process with account creation.
- View account balances, transaction history, and statements.
- Card transaction history and details.
- Set card spending limits or transaction alerts.
- Transfer funds between user's own accounts.

2.1.2 Non - Functional Requirements

Non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviour. They are contrasted with functional requirementsthat define specific behaviour or functions. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture, because they are usually architecturally significant requirements.

User friendly

User friendly means product is accessible and understandable for every user. All things are visible anduser doesn't need to know the detailed description about each feature. He / She just access everythingin just a single moment. All the features are well manner arrange and easily visible.

Accessibility

Accessibility means product should be easily accessible anywhere any place any time, it means our website is full responsive all the time and back end works properly to give easy access to user and administration. Also keep in mind our mini project is accessible in any time of machine, either it is PCs, laptops or Mobile phones.

Efficiency

Efficiency means our product is consumed minimum resources to given the equal load which leads toquick access and it also increase accessibility and make website user friendly. It is better to say give minimum input and takes out maximum output. This leads to our mini project more efficient.

Security

The term security refers to many things. The security of data lose, the security from internal breaches, thesecurity from hacking and un-authorized users etc. Security also refers to save the cost and time to complete when our product is under development process.

Quality

Quality refers that our mini project is fully functional developed under the software quality assurance (SQA) criteria. Quality made our product according to standard, which is acceptable by all over the world.

Reliability

Reliability means there is less chance of error occur in our product. As discussed above if the qualityof our website is good then there are less chance of error to occur in our product and this thing made our website is effective for all users and administration.

2.2 Software Requirements

- Android OS
- XML for designing front-end
- PLATEFORM: Android Language
- INTEGRATED DEVELOPMENT ENVIRONMENT (IDE): Android Studio

CHAPTER-3

SYSTEM DESIGN

When designing and implementing an Android project, it is essential to follow a structured approach to ensure a well-organized and efficient development process. Here is a general outline of the steps involved:

1. Requirements Gathering: Gather and analyze the requirements for your Android project. Understand the project's purpose, target audience, functionality, and any specific features or constraints.

2. System Architecture: Design the overall system architecture of your Android project. Determine the components, modules, and their interactions. Consider using design patterns and best practices to ensure scalability, maintainability, and reusability.

3. User Interface Design: Create wireframes or prototypes to design the user interface (UI) of your Android app. Consider user experience (UX) principles, material design guidelines, and accessibility requirements. Define the app's navigation structure, layouts, and interactive elements.

4. Backend Services: If your Android project requires server-side functionality or data storage, design and implement the necessary backend services. This may involve developing APIs, databases, or integrating with existing services.

5. Data Management: Determine how data will be stored, accessed, and managed within your Android app. This may involve using local storage (such as SQLite), remote databases, or cloud-based services. Implement data models and define data access methods.

6. Coding: Begin coding your Android project, following the system architecture and design specifications. Use a suitable Integrated Development Environment (IDE), such as Android Studio, and apply coding best practices. Break down the functionality into manageable tasks and implement them incrementally.

7. User Interface Implementation: Translate your UI designs into XML layout files and implement the necessary UI components using Android XML syntax. Apply appropriate styling, themes, and animations to enhance the visual appeal and usability of your app.

8. Functionality Implementation: Implement the core functionality of your Android project, including user interactions, data processing, and any additional features required. Write code to handle user inputs, perform calculations, make API calls, and manage app state.

9. Testing: Conduct thorough testing of your Android project to ensure it functions as intended. Perform unit tests, integration tests, and user acceptance testing.

10. Optimization and Performance: Optimize your Android app's performance by identifying and addressing any bottlenecks, memory leaks, or resource inefficiencies. Optimize network requests, database queries, and UI rendering for a smooth user experience.

11. Deployment: Prepare your Android project for deployment by generating a signed APK (Android Package) file. Create necessary app store accounts (such as Google Play Store) and follow the guidelines for submission. Test the deployment package on various devices to ensure compatibility.

12. Maintenance and Updates: After deployment, monitor user feedback and address any reported issues or bugs promptly. Plan for future updates, feature enhancements, and bug fixes to keep your Android app up to date.

3.1 Use Case Diagram

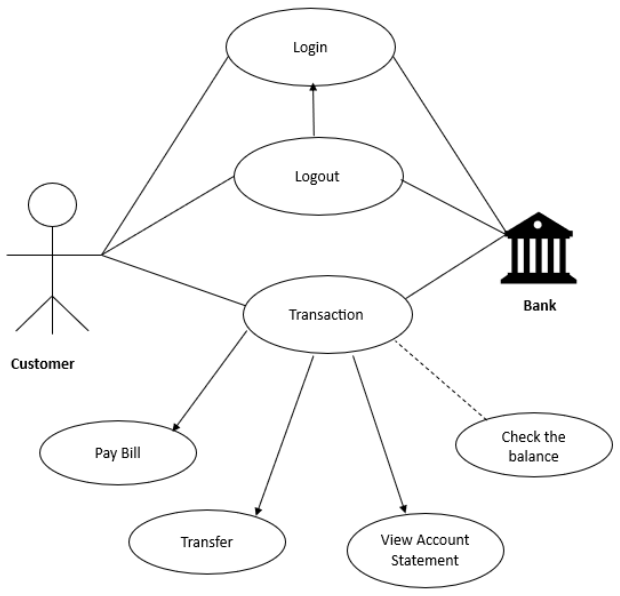


Figure 3.1: Use case Diagram

In this use case diagram, we have three main actors: Customer and Bank. The Customer interacts with the mobile banking application and can perform various tasks. The Customer actor represents the functionalities related to the user's bank account, such as checking the balance and viewing the statement. The Bank actor represents actions related to performing transactions, such as transferring funds.

The main use cases in the diagram are:

- Check Balance: The User can check the balance of their bank account.
- View Statement: The User can view the account statement, which provides details of transactions.
- Transfer Funds: The User can transfer funds from their account to another account.
- Manage Payees: The User can manage a list of payees, which includes adding, editing, or deleting payee details.

3.2 Flow Chart Diagram

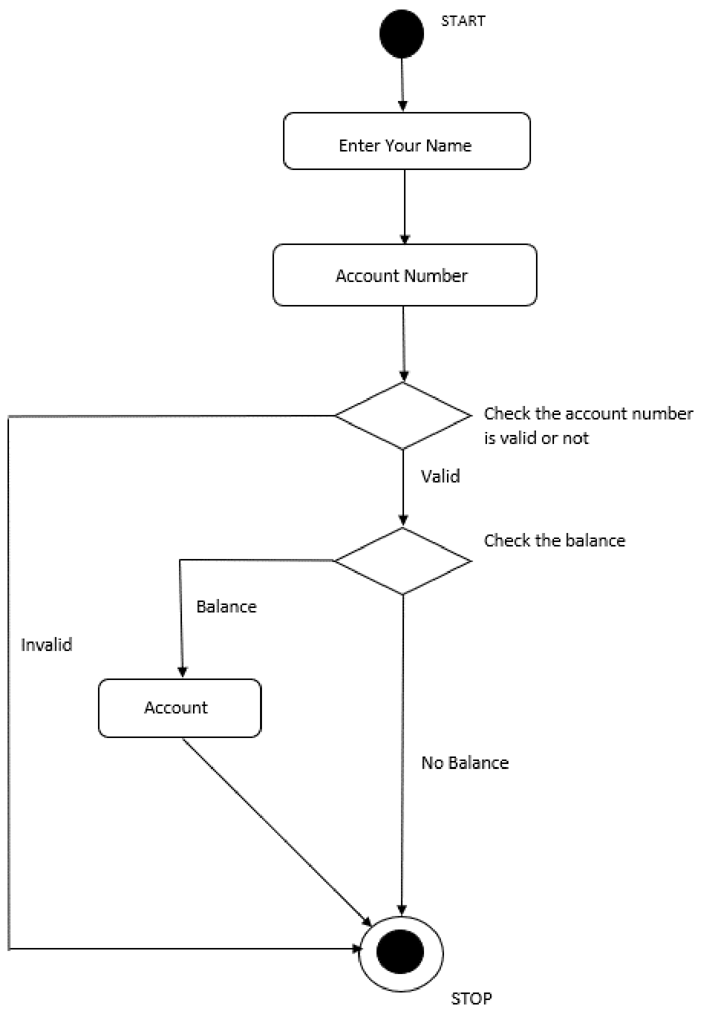


Figure 3.2: Flow Chart Diagram

Flow Chart diagram is a graphical representation of the flow of data within a system. It provides a visual depiction of how data moves through different components or processes within the system. The individuals interacting with the Android mobile banking application. The users enters the account number and password . The application verifies whether the account number is valid or not. If it is valid it further checks for the balance or else it terminates.

CHAPTER 4

IMPLEMENTATION

System implementation is the process of defining how the information system should be built by ensuring that the information system is operational and should meet certain quality standards. Implementation is done using a higher-level language such as: the code can be written fastly, it is compact and also easier to debug and understand.

Pseudocode

Java Files

MainActivity.java

```
package com.bank.izbank.MainScreen;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import android.content.Intent;
import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;
import com.bank.izbank.MainScreen.FinanceScreen.CryptoModel;
import com.bank.izbank.MainScreen.FinanceScreen.FinanceFragment;
import com.bank.izbank.R;
import com.bank.izbank.Sign.SignIn;
import com.bank.izbank.service.ICryptoAPI;
import com.google.android.material.bottomnavigation.BottomNavigationView;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.parse.LogOutCallback;
import com.parse.ParseException;
import com.parse.ParseUser;
import java.util.ArrayList;
import java.util.List;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
public class MainScreenActivity extends AppCompatActivity {
    private BottomNavigationView bottomNavigationView;
    Fragment fragment1 = new AccountFragment();
    final Fragment fragment2 = new CreditFragment();
    final Fragment fragment4 = new BillFragment();
    final Fragment fragment5 = new SettingFragment();
    private Fragment tempFragment=fragment1;
    final FragmentManager fm = getSupportFragmentManager();
    private ArrayList<CryptoModel> cryptoModels;
    private final String BASE_URL = "https://api.nomics.com/v1/";
    private Retrofit retrofit;
    private ImageView downloadingImageView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_screen);
        //Retrofit & JSON
        Gson gson=new GsonBuilder().setLenient().create();

        retrofit=new Retrofit.Builder().
                baseUrl(BASE_URL).
                addConverterFactory(GsonConverterFactory.create(gson)).
                build();
        loadData();
        //Retrofit end

        bottomNavigationView = findViewById(R.id.bottom_navigation);
fm.beginTransaction().add(R.id.fragment_container,fragment5,"5").hide(fragment5).commit();
 fm.beginTransaction().add(R.id.fragment_container,fragment4,"4").hide(fragment4).commit();
 fm.beginTransaction().add(R.id.fragment_container,fragment2,"2").hide(fragment2).commit();
        fm.beginTransaction().add(R.id.fragment_container,fragment1,"1").commit();
        bottomNavigationView.setOnNavigationItemSelectedListener(new
BottomNavigationView.OnNavigationItemSelectedListener() {
            @Override
            public boolean onNavigationItemSelected(@NonNull MenuItem item) {
                switch (item.getItemId()){
                    case R.id.menu1:
                        fragment1 = new AccountFragment();
fm.beginTransaction().add(R.id.fragment_container,fragment1,"1").commit();
fm.beginTransaction().hide(tempFragment).show(fragment1).commit();
                        tempFragment = fragment1;
                        break;
                    case R.id.menu2:

                        fm.beginTransaction().hide(tempFragment).show(fragment2).commit();
                        tempFragment = fragment2;


                        break;
                    case R.id.menu3:
getSupportFragmentManager().beginTransaction().hide(tempFragment).
add(R.id.fragment_container,tempFragment=new
FinanceFragment(cryptoModels)).show(tempFragment).commit();
                        break;
                    case R.id.menu4:
fm.beginTransaction().hide(tempFragment).show(fragment4).commit();
                        tempFragment = fragment4;
                        break;
                    case R.id.menu5:
fm.beginTransaction().hide(tempFragment).show(fragment5).commit();
                        tempFragment = fragment5;
                        break;
                }
                return true;
            }
        });
    }

    private void loadData(){
        ICryptoAPI cryptoAPI=retrofit.create(ICryptoAPI.class);
        Call<List<CryptoModel>> call=cryptoAPI.getData();
        call.enqueue(new Callback<List<CryptoModel>>() {
            @Override
            public void onResponse(Call<List<CryptoModel>> call,
Response<List<CryptoModel>> response) {
                if(response.isSuccessful()){
                    List<CryptoModel> responseList=response.body();
                    cryptoModels=new ArrayList<>(responseList);
                }
            }

            @Override
            public void onFailure(Call<List<CryptoModel>> call, Throwable t) {
                Toast.makeText(getApplicationContext(),  t.getLocalizedMessage(),
Toast.LENGTH_LONG).show();
            }
        });
```

Sign-in

```
package com.bank.izbank.Sign;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
import com.bank.izbank.Bill.Bill;
import com.bank.izbank.Bill.Date;
import com.bank.izbank.Credit.Credit;
import com.bank.izbank.Job.Job;
import com.bank.izbank.MainScreen.AdminPanelActivity;
import com.bank.izbank.R;
import com.bank.izbank.UserInfo.Address;
import com.bank.izbank.UserInfo.Admin;
import com.bank.izbank.UserInfo.BankAccount;
import com.bank.izbank.UserInfo.CreditCard;
import com.bank.izbank.UserInfo.History;
import com.bank.izbank.UserInfo.User;
import com.bank.izbank.UserInfo.UserContext;
import com.bank.izbank.UserInfo.UserTypeState;
import com.bank.izbank.splashScreen.splashScreen;
import com.parse.FindCallback;
import com.parse.GetDataCallback;
import com.parse.LogInCallback;
import com.parse.ParseException;
import com.parse.ParseFile;
import com.parse.ParseObject;
import com.parse.ParseQuery;
import com.parse.ParseUser;
import java.util.ArrayList;
import java.util.List;
import java.util.Stack;

public class SignIn extends AppCompatActivity {
    private EditText userName,userPass;
    public static User mainUser;
    public static ArrayList<User> allUsers;
    private String billType;
    private String billAmount;
    private String billDate;
    private ArrayList<Bill> bills;
    private ArrayList<BankAccount> bankAccounts;
    private Stack<History> history;
    private ArrayList<CreditCard> creditCards;
    private String bankCash,bankAccountNo;
    private String cardNo, cardLimit;
    private Intent intent ;
    private ArrayList<Credit> credits;
    private String creditAmount;
    private String creditInstallment;
    private String creditInterestRate;
    private String creditPayAmount;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_sign_in);//load screen
            userName=findViewById(R.id.edittext_id_number_sign_in);
            userPass=findViewById(R.id.edittext_user_password_sign_in);

    }
    public void signUp(View view){
        Intent signUp=new Intent(SignIn.this, SignUpActivity.class);
        startActivity(signUp);

    }

    public void getUserBills(){
        ParseQuery<ParseObject> queryBill=ParseQuery.getQuery("Bill");
        queryBill.whereEqualTo("username",SignIn.mainUser.getId());
        queryBill.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> objects, ParseException e) {
                if(e!=null){
                    e.printStackTrace();
                 }else{
                    bills = new ArrayList<>();
                    if(objects.size()>0){
                        for(ParseObject object:objects){
```

```
                    billType=object.getString("type");
                    billAmount=object.getString("amount");
                    billDate=object.getString("date");

                    String [] date = billDate.split("/");

                    Date tempdate = new Date(date[0],date[1],date[2]);
                    Bill tempBill = new Bill(billType,Integer.parseInt(billAmount),tempdate);
                    bills.add(tempBill);
                  }
                }
                SignIn.mainUser.setUserbills(bills);
              }
            }
        });
    }

    public void getUserCredits(){
        ParseQuery<ParseObject> queryBill=ParseQuery.getQuery("Credit");
        queryBill.whereEqualTo("username",SignIn.mainUser.getId());
        queryBill.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> objects, ParseException e) {
                if(e!=null){
                    e.printStackTrace();
                }else{
                    credits = new ArrayList<>();
                    if(objects.size()>0){
                        for(ParseObject object:objects){

                            creditAmount = object.getString("amount");
                            creditInstallment = object.getString("installment");
                            creditInterestRate = object.getString("interestRate");
                            creditPayAmount = object.getString("payAmount");
                    Credit tempCredit = new
Credit(Integer.parseInt(creditAmount),Integer.parseInt(creditInstallment),
                          Integer.parseInt(creditInterestRate),Integer.parseInt(creditPayAmount));
                    credits.add(tempCredit);
                }
            }
            SignIn.mainUser.setCredits(credits);
        }
    }
   });
}

public void getCreditCards(User user){
    ParseQuery<ParseObject> queryBankAccount=ParseQuery.getQuery("CreditCard");
    queryBankAccount.whereEqualTo("userId", user.getId());
    queryBankAccount.findInBackground(new FindCallback<ParseObject>() {
        @Override
        public void done(List<ParseObject> objects, ParseException e) {
            if(e!=null){
                e.printStackTrace();
            }else{
                creditCards = new ArrayList<>();
                if(objects.size()>0){
                    for(ParseObject object:objects){


                        cardNo=object.getString("creditCardNo");
                        cardLimit=object.getString("limit");


                        creditCards.add(new CreditCard(cardNo,Integer.parseInt(cardLimit)));
                    }
                }
                user.setCreditcards(creditCards);
            }
        }
        });
    }

    public void getBankAccounts(User user){
        ParseQuery<ParseObject> queryBankAccount=ParseQuery.getQuery("BankAccount");
        queryBankAccount.whereEqualTo("userId", user.getId());
        queryBankAccount.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> objects, ParseException e) {
                if(e!=null){
                    e.printStackTrace();
                }else{
                    bankAccounts = new ArrayList<>();
                    if(objects.size()>0){
                        for(ParseObject object:objects){

                            bankAccountNo=object.getString("accountNo");
                            bankCash=object.getString("cash");
                            bankAccounts.add(new
BankAccount(bankAccountNo,Integer.parseInt(bankCash)));

                        }
                    }
                    user.setBankAccounts(bankAccounts);
                }
            }
        });
    }
    public void getHistory(){
        ParseQuery<ParseObject> queryBankAccount=ParseQuery.getQuery("History");
        if (!mainUser.getId().equals("9999")){
            queryBankAccount.whereEqualTo("userId", SignIn.mainUser.getId());
        }
        else{
            getAllUsers();
        }
        queryBankAccount.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> objects, ParseException e) {
                if(e!=null){
                    e.printStackTrace();
                }else{
                    history = new Stack<>();
                    if(objects.size()>0){
                        for(ParseObject object:objects){


                            String id = object.getString("userId");
                            String process = object.getString("process");
                            java.util.Date date = object.getDate("date");
                            history.push(new History(id,process,date));
                        }
                    }
                    SignIn.mainUser.setHistory(history);
                }
            }
        });
    }
    public void getAllUsers(){
        ParseQuery<ParseObject> queryBankAccount=ParseQuery.getQuery("UserInfo");
        queryBankAccount.findInBackground(new FindCallback<ParseObject>() {
            @Override
            public void done(List<ParseObject> objects, ParseException e) {
                if(e!=null){
                    e.printStackTrace();
                }else{
                    allUsers = new ArrayList<>();
                    if(objects.size()>0){
                        for(ParseObject object:objects){
```

```
                    String name=object.getString("userRealName");
                    String phone=object.getString("phone");
                    String userId=object.getString("username");
                    String address_string= object.getString("address");
                    String[] str = address_string.split(" ");
                    Address address = new
Address(str[0],str[1],Integer.parseInt(str[2]),Integer.parseInt(str[3]),Integer.parseInt(str[4]),
str[5],str[6],str[7]);
                    String jobName = object.getString("job");
                    String maxCreditAmount = object.getString("maxCreditAmount");
                    String interestRate = object.getString("interestRate");
                    String maxCreditInstallment = object.getString("maxCreditInstallment");
                    Job tempJob = new
Job(jobName,maxCreditAmount,maxCreditInstallment,interestRate);
                    User tempUser = new User(name,userId, phone,address,tempJob);
                    getBankAccounts(tempUser);
                    getCreditCards(tempUser);
                    ParseFile parseFile=(ParseFile)object.get("images");
                    if( parseFile!=null){
                        parseFile.getDataInBackground(new GetDataCallback() {
                            @Override
                            public void done(byte[] data, ParseException e) {
                                if(data!=null && e==null){
                                    Bitmap downloadedImage=
BitmapFactory.decodeByteArray(data,0,data.length);
                                    tempUser.setPhoto(downloadedImage);


                                }
                            }
                        });
                    }
                    allUsers.add(tempUser);
                }
            }
        }
```

```
            }
        });
    }
    public void signIn(View view){
        //loading screen
        ParseUser.logInInBackground(userName.getText().toString(), userPass.getText().toString(),
new LogInCallback() {
            @Override
            public void done(ParseUser user, ParseException e) {
                if(e !=null){
Toast.makeText(getApplicationContext(),e.getLocalizedMessage(),Toast.LENGTH_LONG).show();
}else{
                    ParseQuery<ParseObject> query=ParseQuery.getQuery("UserInfo");
query.whereEqualTo("username",userName.getText().toString());
                    query.findInBackground(new FindCallback<ParseObject>() {
                        @Override
                        public void done(List<ParseObject> objects, ParseException e) {
                            if(e!=null){
                                e.printStackTrace();
                            }else{
                                if(objects.size()>0){
                                    for(ParseObject object:objects){
                                        String name=object.getString("userRealName");
                                        String phone=object.getString("phone");
                                        String userId=object.getString("username");
                                        String address_string= object.getString("address");
                                        String[] str = address_string.split(" ");
                                        Address address = new
Address(str[0],str[1],Integer.parseInt(str[2]),Integer.parseInt(str[3]),Integer.parseInt(str[4]),str[
5],str[6],str[7]);
                                        String jobName = object.getString("job");
                                        String maxCreditAmount = object.getString("maxCreditAmount");
                                        String maxCreditInstallment = object.getString("maxCreditInstallment");
                                        String interestRate = object.getString("interestRate");
                                        Job tempJob = new
Job(jobName,maxCreditAmount,maxCreditInstallment,interestRate);

                                        mainUser = new User(name,userId, phone,address,tempJob);
                                        ParseFile parseFile=(ParseFile)object.get("images");
                                       if( parseFile!=null){
                                           parseFile.getDataInBackground(new GetDataCallback() {
                                               @Override
                                               public void done(byte[] data, ParseException e) {
                                                   if(data!=null && e==null){
                                                       Bitmap downloadedImage=
BitmapFactory.decodeByteArray(data,0,data.length);
                                                       mainUser.setPhoto(downloadedImage);
                                                   }
                                               }
                                           });
                                       }
                                        Toast.makeText(getApplicationContext(),"Welcome
"+name,Toast.LENGTH_LONG).show();
                                        getBankAccounts(mainUser);
                                        getCreditCards(mainUser);
                                        getHistory();
                                        getUserBills();
                                        getUserCredits();
                                    }
                                }
                            }
                        }
                    });
                    intent = new Intent(SignIn.this, splashScreen.class);
                    startActivity(intent);
                }
            }
        });
    }
}
```

CHAPTER-5

RESULTS

5.1:Sign-In Page

In the figure 5.1 the user can signup through the application by enrolling the name,id,password and Phonenumber. The detailed Address such as street name, House number,State and Country must be filled in order to Sign-in .After Sign-up the the application is rendered to the signin page where the user must enter his id-number and password tin order to Sign in to his account.

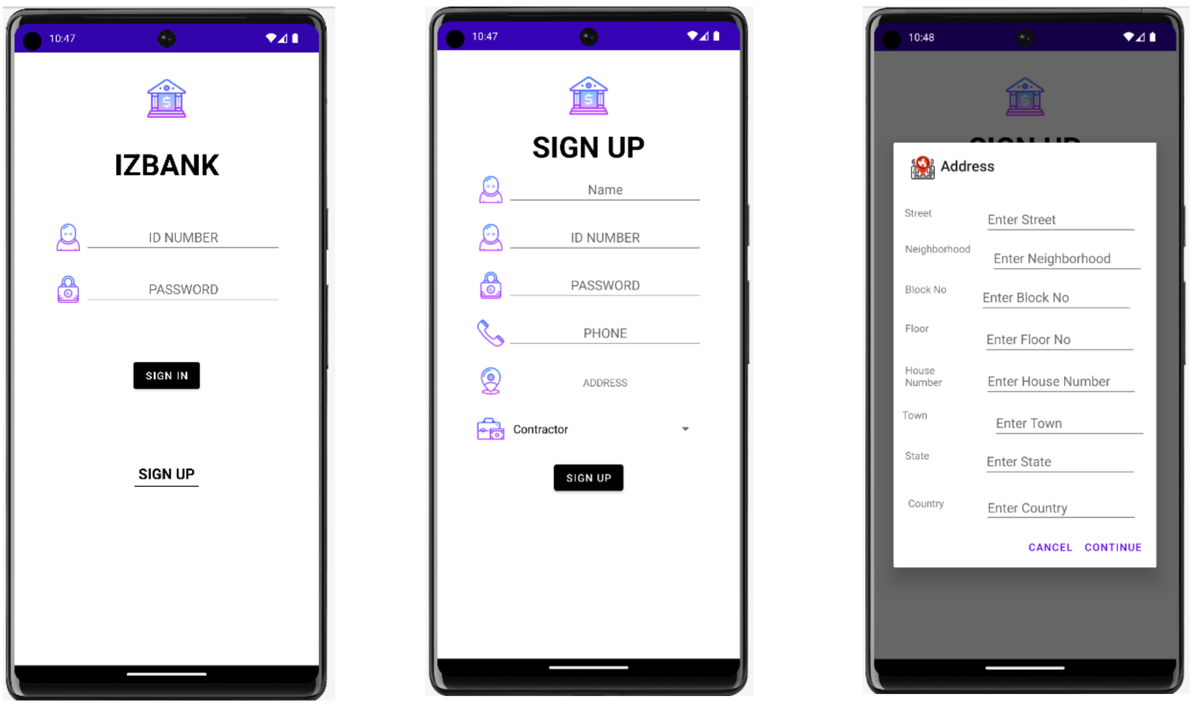


Figure 5.1:Sign-In Page

5.2: Home Page

In the figure 5.2 the user can observe the home Page where the individual can get the information about the date, bank balance of individual account and the credit details. The user can send the money to his different account. The money deduced from the account and the money credited to the account is also displayed .

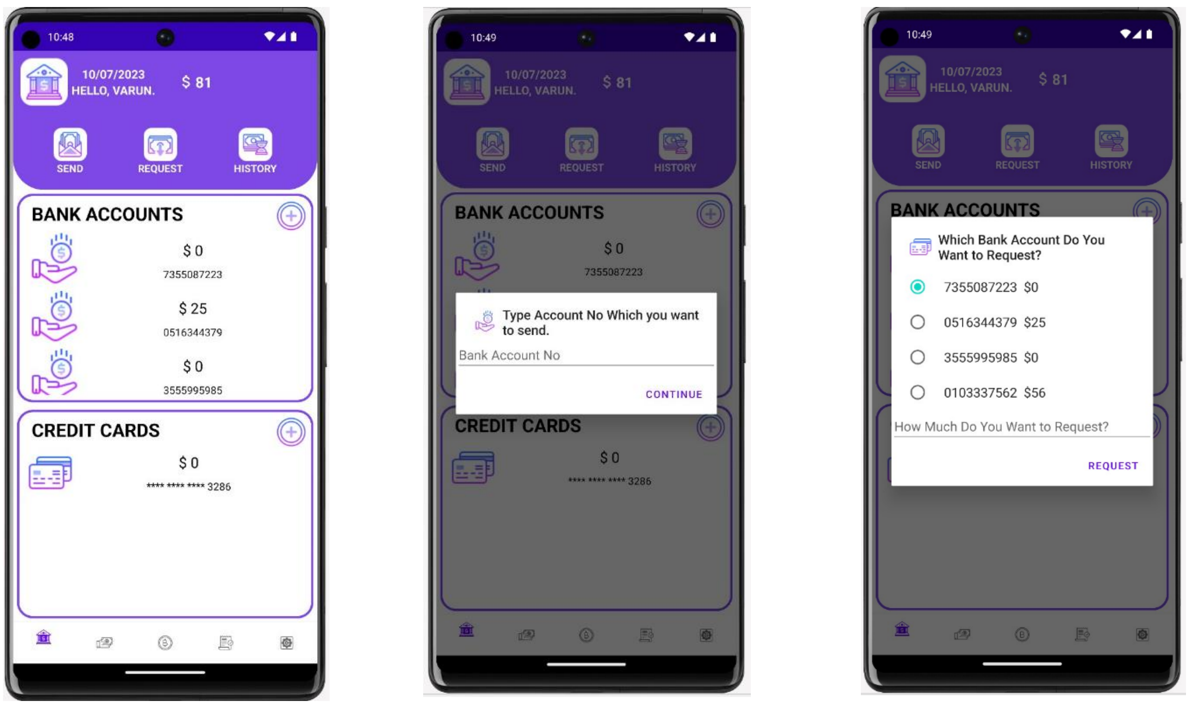
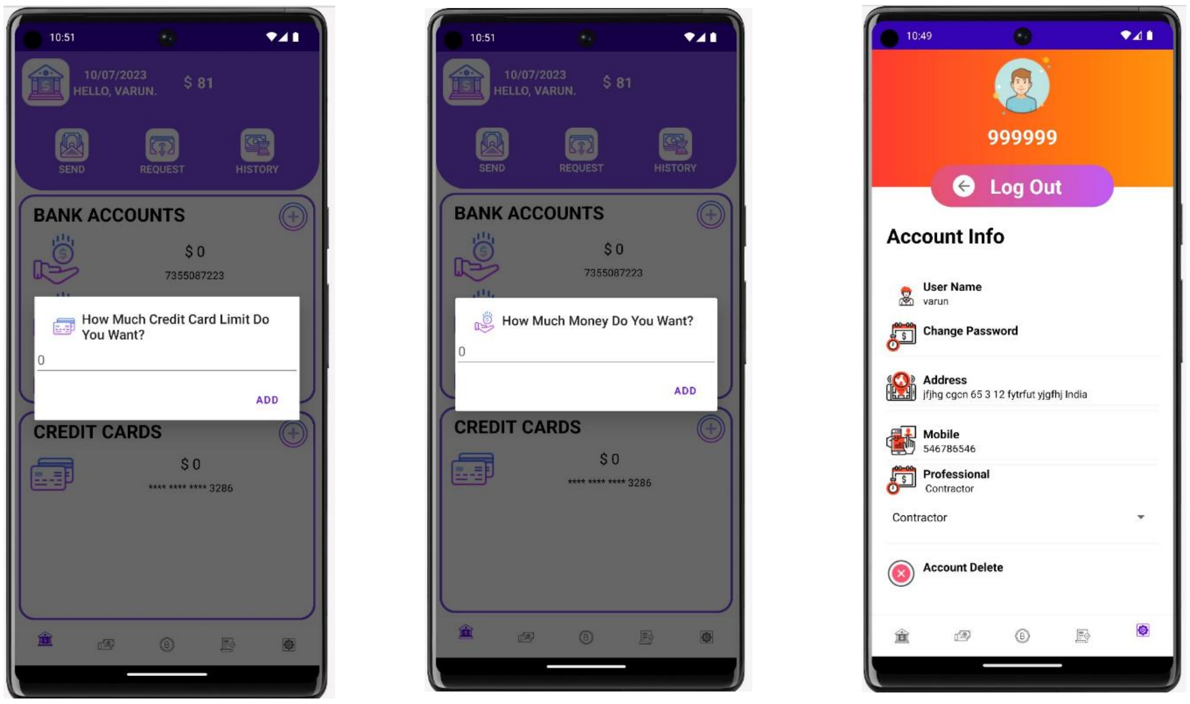


Figure 5.2: Home Page

5.3: Account info Page

In the figure 5.3 the individual can observe their account information and in case if they want to make any modification such as changing of password is also entertained . The individual can also set their credit limit .


10:51
10/07/2023
HELLO, VARUN.
$ 81
SEND REQUEST HISTORY
BANK ACCOUNTS
$ 0
7355087223
How Much Credit Card Limit Do You Want?
0
ADD
CREDIT CARDS
$ 0
**** **** **** 3286

10:51
10/07/2023
HELLO, VARUN.
$ 81
SEND REQUEST HISTORY
BANK ACCOUNTS
$ 0
7355087223
How Much Money Do You Want?
0
ADD
CREDIT CARDS
$ 0
**** **** **** 3286

10:49
999999
Log Out
Account Info
User Name
varun
Change Password
Address
jfjhg cgcn 65 3 12 fytrfut yjgfhj India
Mobile
546786546
Professional
Contractor
Contractor
Account Delete

Figure 5.3: Account info Page

CONCLUSION

Mobile banking Application has become an integral part of the banking industry, providing customers with convenience, accessibility, and enhanced financial management capabilities. They have transformed the way people interact with their banks and have contributed to the digital transformation of the financial sector. As technology continues to advance, mobile banking applications are expected to evolve further, offering even more innovative features to meet the changing needs of customers. mobile banking applications offer unparalleled convenience. With a few taps on a smartphone, users can access their accounts, check balances, make payments, transfer funds, and perform various financial transactions from anywhere at any time. This level of accessibility eliminates the need for physical visits to a bank branch and saves valuable time and effort.

REFERENCES

[1] Erik Hellman, “Android Programming – Pushing the Limits”, 1 st Edition, Wiley India Pvt Ltd,2014. ISBN-13: 978-8126547197

[2] Dawn Griffiths and David Griffiths, “Head First Android Development”, 1 st Edition, O‟ReillySPD Publishers, 2015. ISBN-13: 978-9352131341

[3] Bill Phillips, Chris Stewart and Kristin Marsicano, “Android Programming: The Big Nerd RanchGuide”, 3 rd Edition, Big Nerd Ranch Guides, 2017. ISBN-13: 978-0134706054

Websites

1. https://www.geeksforgeeks.org/android-tutorial/
2. https://www.w3schools.blog/android-tutorial
3. https://www.tutorialspoint.com/sql/