

# **MULTIBANKING SYSTEM**

Submitted in partial fulfillment of requirement for the award of the Degree  
Bachelor of Computer Science

In the faculty of Computer Science of Bharathiar University, Coimbatore

Submitted by

**R.DHANUSHYA**

**(Reg.No.2022K0122)**

Under the guidance of

**Dr.P.KOKILA MSc., M.Phil., Ph.D**

Guest lecturer , Department of Computer Science



Department of Computer Science

**L.R.G GOVERNMENT ARTS COLLEGE FOR  
WOMEN**

**(Affiliated To Bharathiar University)**

**TIRUPUR-4**

**APRIL-2023**

*CERTIFICATE*

## **CERTIFICATE**

This is to certify that the project work entitled “ **MULTIBANKING SYSTEM**” Submitted to Bharathiar University in partial fulfilled of the requirement for the award of the Degree of Bachelor of computer science is a record of the original work done by **Ms.R.DHANUSHYA (Reg.No.2022K0122)** Under my supervisor and that project work has not formed the basis for the any Degree /Diploma /Association /Fellowship or similar title to any candidate of any university.

**Internal Guide**

**Dr.P.KOKILA MSc.,M.Phil.,Ph.D**

**Head of the Department**

**Dr.R.PARIMALA MSc.,M.Phil.,Ph.D**

Viva-voce examination is held on \_\_\_\_\_ L.R.G Government Arts College for Women, Tirupur-641604.

**Internal Examiner**

**External Examiner**

# *DECLARATION*

## **DECLARATION**

I hereby declare that the project work submitted to the **Department of the Computer Science, L.R.G. Government Arts College for Women, Tirupur** , affiliated to Bharathiar University, Coimbatore in the partial fulfillment of the required for the award of Bachelor of Computer Science is an original work done by me during the sixth semester.

**Place:**

**Date:**

**Signature of the Candidate**

**(R.DHANUSHYA)**

**(Reg.No:2022K0122)**

# *ACKNOWLEDGEMENT*

# ACKNOWLEDGEMENT

As first and foremost I would like to external my thankfulness to the almighty for blessing the work of my hands, I am grateful to my parents for continued encouragement that they had given to me.

I would like to external my profound gratitude and sincere thanks to **Dr.M.R.YEZHILI MA.,M.Phil.,Ph.D** Principal, L.R.G Government Arts College For Women , for the encouragement rendered to me during this project.

It's my privilege to thank **DR.R.PARIMALA MSc.,M.Phil.,Ph.D** Incharge Head of the department of computer science for her valuable guidance and support throughout the project development work.

I express my deep sense of gratitude and sincere thanks to my guide **Dr.P.KOKILA MSc.,M.Phil.,Ph.D Guest** Lecturer, Department of computer science, for providing all sorts of facilities to complete my project work successfully.

I also extend my sincere thanks to all the other faculty member of the department and technical assistance for their co-operation and valuable guidance.

I thank our college library for providing me with many informative books that help me to enrich my knowledge to bring out the project successfully.

# *SYNOPSIS*



## **SYNOPSIS**

The purpose of this Multi Banking System Project is to build a system which will integrate many bank account to the single software-based system. It will provide an easy and fast way to access all the bank through a single interface. By using this system a person who has different accounts in the respective bank or other banks can log in the multi-banking system and can access to their bank accounts. So this system will provide a structure which will link all the bank of a person into one place. Multi banking bank applications also offer enhanced security features such as two-factor authentication, encryption, and biometric verification, which helps to protect users' financial information from unauthorized access and fraud. Moreover, these applications provide a wealth of data and insights that can help users make informed decisions about their finances. Customers can analyze their spending patterns, track their expenses, and identify areas where they can reduce their spending, ultimately leading to better financial planning and management.

# *CONTENTS*

# CONTENTS

<b>CONTENTS</b>	<b>PAGE NO</b>
<b>1. INTRODUCTION</b>	<b>1</b>
1.1 OVERVIEW OF THE PROJECT	1
1.2 SYSTEM SPECIFICATION	2
1.2.1 HARDWARE REQUIREMENTS	2
1.2.2 SOFTWARE SPECIFICATIONS	2
1.2.3 SOFTWARE DISCRIPTION	3
<b>2. SYSTEM STUDY</b>	<b>8</b>
2.1 EXISTING SYSTEM	8
2.1.1 DRAWBACKS	8
2.2 PROPOSED SYSTEM	8
2.2.1 FEATURES	8
<b>3. SYSTEM DESIGN AND DEVELOPMENT</b>	<b>9</b>
3.1 FILE DESIGN	9
3.2 INPUT DESIGN	10
3.3 OUTPUT DESIGN	11
3.4 DATABASE DESIGN	12
3.5 SYSTEM DEVELOPMENT	13
3.5.1 DESCRIPTION OF MODULES	13
<b>4. TESTING AND IMPLEMENTATION</b>	<b>15</b>
<b>5. CONCLUSION</b>	<b>24</b>
<b>FUTURE ENHANCEMENT</b>	<b>25</b>
<b>BIBLIOGRAPHY</b>	<b>25</b>
<b>APPENDICES</b>	<b>26</b>
A. DATA FLOW DIAGRAM	26
B. TABLE STRUCTURE	28
C. SAMPLE CODING	31
D. SAMPLE INPUT AND OUTPUT	44

# *INTRODUCTION*

# 1. INTRODUCTION

## 1.1 OVERVIEW OF THE PROJECT

Nowadays the use of bank applications has increased and many banking operations are done online. In this project we proposed a **Multi Banking System** using web services (for transactions).

It is developed for those customers who are having multiple accounts in various banks. Multi-banking interface is a web-based application through which customer can access his/her multiple accounts with single user id and password. The customer does not have to remember all the user id and password for each account he/she has in a banking website.

The objective of this application is to allow the Customers of various Banks access their accounts and make transactions using this solution. They need not interact with various bank applications or web sites. The customer can login with single id and password.

In addition to this, the customer has certain privileges such as he/she can view his/her transaction and account details and also make fund transfers from one bank account to other. Also, user can create various bank account in a single application.

## **1.2 SYSTEM SPECIFICATION**

System Requirements Specification also known as Software Requirements Specification, is a document or set of documentation that describes the features and behavior of a software application

### **1.2.1 HARDWARE SPECIFICATION**

□ Processor	: Intel ® Core (TM)
□ Processor Speed	: 2.60. GHz
□ RAM	: 4 GB RAM
□ Hard Disk Drive	: 180 GB
□ Keyboard	: standard 102 keys

### **1.2.2 SOFTWARE SPECIFICATION**

□ Operating System	: Windows 10
□ Web Server	: Apache Tomcat 8.5
□ Front End	: PYTHON
□ Back End	: MySQL

## **1.2.3 SOFTWARE DESCRIPTION**

### **WINDOWS OS**

Windows is a graphical operating system developed by Microsoft. It allows users to view and store files, run the software, play games, watch videos, and provides a way to connect to the internet. It was released for both home computing and professional works.

Microsoft introduced the first version as 1.0

It was released for both home computing and professional functions of Windows on 10 November 1983. Later, it was released on many versions of Windows as well as the current version, Windows 10.

In 1993, the first business-oriented version of Windows was released, which is known as Windows NT 3.1. Then it introduced the next versions, Windows 3.5, 4/0, and Windows 2000. When the XP Windows was released by Microsoft in 2001, the company designed its various versions for a personal and business environment. It was designed based on standard x86 hardware, like Intel and AMD processor. Accordingly, it can run on different brands of hardware, such as HP, Dell, and Sony computers, including home-built PCs.

Play Video

#### **Editions of Windows**

Microsoft has produced several editions of Windows, starting with Windows XP. These versions have the same core operating system, but some versions included advance features with an additional cost. There are two most common editions of Windows:

- Windows Home
- Windows Professional

Windows Home is basic edition of Windows. It offers all the fundamental functions of Windows, such as browsing the web, connecting to the Internet, playing video games, using office software, watching videos. Furthermore, it is less expensive and comes pre-installed with many new computers.

# INTRODUCTION TO FRONT END

## PYTHON

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems. This versatility, along with its beginner-friendliness, has made it one of the most-used programming languages today. A survey conducted by industry analyst firm RedMonk found that it was the second-most popular programming language among developers in 2021.

Python is commonly used for developing websites and software, task automation, data analysis, and data visualization. Since it's relatively easy to learn, Python has been adopted by many non-programmers such as accountants and scientists, for a variety of everyday tasks, like organizing finances.

"Writing programs is a very creative and rewarding activity," says University of Michigan and Courser instructor Charles R Severance in his book Python for Everybody. "You can write programs for many reasons, ranging from making your living to solving a difficult data analysis problem to having fun to helping someone else solve a problem."

# INTRODUCTION TO BACKEND

## MY-SQL 4.0

MySQL is a full-featured relational database management system. It is very stable and has proven itself over time. MySQL has been in production for over 10 years. MySQL is a multithreaded server. *Multithreaded* means that every time someone establishes a connection with the server, the server program creates a thread or process to handle that client's requests. This makes for an extremely fast server. In effect, every client who connects to a MySQL server gets his or her own thread.

Yet another feature of MySQL is its portability—it has been ported to almost every platform. This means that you don't have to change your main platform to take advantage of MySQL. And if you do want to switch, there is probably a MySQL port for your new platform.



MySQL also has many different application programming interfaces (APIs). They include APIs for Perl, TCL, Python, C/C++, Java (JDBC), and ODBC. So no matter what your company's expertise is, MySQL has a way for you to access it.

MySQL is also very cheap. For an unlicensed, full version of MySQL, the cost is nothing. To license your copy will currently cost you \$200. This is an incredible deal, considering what you are getting for your money.

Database systems that provide half the features that MySQL has can cost tens of thousands of dollars. MySQL can do what they do better and for less.

MySQL is a relational database. It uses tables and columns to hold data that can be related by keys. It is well suited for this role. It is also very well suited for various architectures. It can be used in a strictly client/server architecture or as a standalone database.

## **The MySQL Supported Types**

MySQL has various data types that support different functions. A data type is the type of data a column will store. There can be many different data types inside a table, but each column will store its own specific type of information. You can think of a data type as a kind of definition for a column.

A column defined as an integer column will only hold numeric information, whereas a column defined as a CHAR (10) will hold up to 10 alphanumeric characters. These definitions are the key to a quick and efficient database. There are basically three groups of data formats. The first is obviously numeric. Numeric data is data that is a positive or negative number such as 4 or -50.

Numeric data can also be in hexadecimal format (2ee250cc), scientific notation ( $2 \times 10^{23}$ ), or a decimal. The second type is character or string format. This format can consist of letters and numbers, whole words, addresses, phone numbers, and generally anything you have to put quotations around.

It consists of everything that doesn't quite fit into either of the other two categories. Some, like dates and times, could be alphanumeric but are stored like numbers. As well as data types, MySQL also provides column modifiers. These modifiers further help define a column's attributes.

They are AUTO\_INCREMENT, UNSIGNED, PRIMARY KEY, NULL, NOT NULL, and BINARY. A more detailed discussion of column modifiers takes place following the coverage of the basic data types.

MySQL runs on many platforms, and binaries are available for most of them. Binaries are the result of compiling the source code. This is by far the easiest way of acquiring MySQL. The alternative is downloading the source code for your platform and then compiling it.

MySQL has many utilities to import as well as export data. It shares some of the common options, but this utility does a little more. It takes the entire database and dumps it into a single text file.

This file contains all the SQL commands needed to recreate your database. It takes the schema and converts it to the proper DDL syntax (**CREATE** statements), and it takes all the data and creates **INSERT** statements out of them. This utility reverse engineers your database.

*SYSTEM STUDY*

## **2. SYSTEM STUDY**

### **2.1 EXISTING SYSTEM**

Currently we are having lot of banks in the market and any person can do transactions of any individual bank either manually or in online. But no one can do all banks transactions in a single portal or in single bank.

#### **2.1.1 DRAWBACKS**

- Customer need to use multi software for different account
- Waste of time for just create an bank account
- Single workers can able to handle the customer account

### **2.2 PROPOSEDSYSTEM**

The Multi Banking System Interface is targeted to the future banking solution for the users who is having multiple bank accounts in multiple banks. This interface integrates all existing banks and provides business solutions for both retail and corporate.

#### **2.2.1 FEATURES**

- By using this portal any client who maintain accounts in various banks can directly log on to Multi Banking System Interface and make any kind of transactions.
- Can access different bank account details in an single application
- All the action has taken care by single application

*SYSTEM DESIGN &  
DEVELOPMENT*

### **3. SYSTEM DESIGN AND DEVELOPMENT**

#### **3.1 FILE DESIGN**

The selection of the file system design approach is done according to the needs of the developers what are the needed requirements and specifications for the new design. It allowed us to identify where our proposal fitted in with relation to current and past file system development. Our experience with file system development is limited so the research served to identify the different techniques that can be used. The variety of file systems encountered show what an active area of research file system development is. The file systems may be from one of the two fundamental categories. In one category, the file system is developed in user space and runs as a user process. Another file system may be developed in the kernel space and runs as a privileged process. Another one is the mixed approach in which we can take the advantages of both aforesaid approaches. Each development option has its own pros and cons. In this article, these design approaches are discussed.

A file system is the data structure designed to support the abstraction of the data blocks as an archive and collection of files. This data structure is unique because it is stored on secondary storage (usually the disk), which is a very slow device.

The file system structure is the most basic level of organization in an operating system. Almost all of the ways an operating system interacts with its users, applications, and security model are dependent upon the way it organizes files on storage devices.

File Design Information systems in business are file and database oriented. Data are accumulated into files that are processed or maintained by the system. The systems analyst is responsible for designing files, determining their contents and selecting a method for organizing the data.

The most important purpose of a file system is to manage user data. This includes storing, retrieving and updating data. Some file systems accept data for storage as a stream of bytes which are collected and stored in a manner efficient for the media.

## 3.2 INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:’

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

### FEATURES

- Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.
- It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.
- When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user
- will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

### **3.3 OUTPUT DESIGN**

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed, and consider the necessary output controls and prototype report layouts.

#### **External Outputs**

Manufacturers create and design external outputs for printers. External outputs enable the system to leave the trigger actions on the part of their recipients or confirm actions to their recipients.

Some of the external outputs are designed as turnaround outputs, which are implemented as a form and re-enter the system as an input.

#### **Internal outputs**

Internal outputs are present inside the system, and used by end-users and managers. They support the management in decision making and reporting.

#### **Output Integrity Controls**

Output integrity controls include routing codes to identify the receiving system, and verification messages to confirm successful receipt of messages that are handled by network protocol.

Printed or screen-format reports should include a date/time for report printing and the data. Multipage reports contain report title or description, and pagination. Pre-printed forms usually include a version number and effective date.



### 3.4 DATABASE DESIGN

Today's businesses depend on their databases to provide information essential for day-to-day operations, especially in case of electronic commerce businesses who has a definite advantage with up-to-date database access. Good design forms the foundation of any database, and experienced hands are required in the automation process to design for optimum and stable performance.

Software Solutions have been constantly working on these platforms and have attained a level of expertise. We apply proven methodologies to design, develop, integrate and implement database systems to attain its optimum level of performance and maximize security to meet the client's business model.

#### **Business needs addressed:**

- Determine the basic objects about which the information is stored
- Determine the relationships between these groups of information and the objects
- Effectively manage data and create intelligent information
- Remote database administration or on site administrative support
- Database creation, management, and maintenance
- Information retrieval efficiency, remove data redundancy and ensure data security

The most important consideration in designing the database is how the information will be used. The main objective of designing a database is Data Integration, Data Integrity and Data Independence.

#### **Data Integration**

In a database, information from several files is coordinated, accessed and operated upon as though it is in a single file. Logically, the information is centralized, physically; the data may be located on different devices, connected through data communication facilities.

#### **Data Integrity**

Data integrity means storing all data in one place only and how each application accesses it. This approach results in more consistent information, one update being

sufficient to achieve a new record status for all applications. This leads to less data redundancy that is data items need not be duplicated.

### **Data Independence**

Data independence is the insulation of application programs from changing aspects of physical data organization. This objective seeks to allow changes in the content and organization of physical data without reprogramming of application and allow modifications to application programs without reorganizing the physical data.

## **3.5 SYSTEM DEVELOPMENT**

Systems development is the process of defining, designing, testing, and implementing a new software application or program. It could include the internal development of customized systems, the creation of database systems, or the acquisition of third party developed software.

Systems development life cycle phases include planning, system analysis, system design, development, implementation, integration and testing, and operations and maintenance.

### **3.5.1. DESCRIPTION OF MODULES**

There are six modules in the project,

- **Bank Registration**
- **Manager Registration**
- **Customer Registration**
- **Approve and Reject Request**
- **Create Bank Account**
- **Customer detail report**



## **Bank Registration**

Admin first login into the banking system and create linked bank details, once the admin creates the bank it will show in the all the requested places. Admin provide all the information about the bank

## **Manager Registration**

Admin also have access to create manager under respective banks. This is a big process to add the manager details. Once the manager allots to the bank, he gave access to managing the customer details.

## **Customer Registration**

A customer manually selects the bank and gives request to the respective bank manager; manager decide to check the customer details and give approve request. Once the customer got access the account will be created.

## **Approve and Reject request**

This module will be handling by the bank manager, once manager get customer account request, just verified and do the action for approve or reject.

## **Create Bank Account**

By default, it will create after the manager approving customer request. Once the account has created the customer can able to login and check the account details.

## **Customer details Report**

Manager can view the all the customer details as well in a single window. Once the customer has approved status the customer has shown in this window.

*TESTING &  
IMPLIMENTATION*

## **4. TESTING AND IMPLEMENTATION**

### **TESTING METHODOLOGIES**

System testing is state of implementation, which is aimed at ensuring that the system works accurately and efficiently as expect before live operation commences. It certifies that the whole set of programs hang together.

System testing requires a test plan that consists of several key activities and step for run program, string, system and user acceptance testing. The implementation of newly designed package is important in adopting a successful new system

Testing is the important stage in software development. the system test in implementation stage in software development process. The system testing implementation should be confirmation that all is correct and an opportunity to show the users that the system works as expected. It accounts the largest percentage of technical effort in the software development process.

Testing phase in the development cycle validates the code against the functional specification testing is vital to achievement of the system goals. The objective of the testing is to discover errors to fulfill this objective a series of test step unit, integration. Validation and system tests were planned and executed the test steps are:

### **SYSTEM TESTING**

Testing is an integral part of any system development life cycle. Insufficient and untested applications may tend to crash and the result is loss of economic and manpower investment besides user's dissatisfaction and downfall of reputation. Software testing can be looked upon as one among many processes, an organization performs, and that provides the lost opportunity to correct any flaws in the developed system. Software testing includes selecting test data that have more probability of giving errors.

The first step in system testing is to develop a plan that tests all aspects of the system. Completeness, correctness, reliability and maintainability of the software are to be tested for the best quality assurance that the system meets the specification and requirements for its intended use and performance. System testing is the most useful practical process of executing a program with the implicit intention of finding errors that make the program fails. System testing is done in three phases.

- Unit Testing
- Integration Testing
- Validation Testing

## **UNIT TESTING**

Unit testing focuses verification effort on the smallest unit of software the module. Using the detailed design and the process specification testing is done to registration by the user with in the boundary of the Login module. The login form receives the username and password details and validates the value with the database. If valid, the home page is displayed.

## **INTEGRATION TESTING**

Integration Testing is the process of this activity can be considered as testing the design and hence module interaction. The primary objective of integration testing is to discover or in the interfaces between the components. Login form and registration form are integrated and tested together. If the user is newly registered, the received details will be stored in the registration table. While logging in, the application will check for valid user name and password in the registration table and if valid the user is prompted for submitting complaints.

Data can be lost across an interface, one module can have adverse effect on another sub function when combined it may not produce the desired major functions. Integration testing is a systematic testing for constructing test to uncover errors associated within an interface.

The objectives taken from unit tested modules and a program structure is built for integrated testing. All the modules are combined and the test is made.

A correction made in this testing is difficult because the vast expenses of the entire program complicated the isolation of causes. In this integration testing step, all the errors are corrected for next testing process.

## **VALIDATION TESTING**

Validation are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it fulfill it in purpose the actual result from the expected result for the complaint process. Select the complaint category of the complaint by user. The input given to various forms fields are validated effectively. Each module is tested independently. It is tested that the complaint module fields receive the correct input for the necessary details such as complaint category, complaint id, reference name, complaint description, and email for further process.s

After the completion of the integrated testing, software is completely assembled as a package; interfacing error has been uncovered and corrected and a final series of software test validation begins.

Validation testing can be defined in many ways but a simple definition is that validation succeeds when the software function in a manner that can be reasonably expected by the customer. After validation test has been conducted, one of two possible conditions exists.

## **OUTPUT TESTING**

The next process of validation testing, is output testing of the proposed system, since no system could be successful if it does not produce the required output in the specified format. Asking the user about the format required, list the output to be generated or displayed by the system under considerations.

Output testing is a different test whose primary purpose is to fully exercise the computer based system although each test has a different purpose all the work should verify that all system elements have been properly integrated and perform allocated functions.

The output format on the screen is found to be corrected as the format was designed in the system design phase according to the user needs for the hard copy also; the output testing has not resulted in any correction in the system.



## **SYSTEM IMPLEMENTATION**

When the initial design was done for the system, the client was consulted for the acceptance of the design so that further proceedings of the system development can be carried on. After the development of the system a demonstration was given to them about the working of the system. The aim of the system illustration was to identify any malfunction of the system.

After the management of the system was approved the system implemented in the concern, initially the system was run parallel with existing manual system. The system has been tested with live data and has proved to be error free and user friendly.

Implementation is the process of converting a new or revised system design into an operational one when the initial design was done by the system; a demonstration was given to the end user about the working system.

This process is used to verify and identify any logical mess working of the system by feeding various combinations of test data. After the approval of the system by both end user and management the system was implemented.

System implementation is made up of many activities. The six major activities are as follows.

## **CODING**

Coding is the process of whereby the physical design specifications created by the analysis team turned into working computer code by the programming team. A design code may be a tool which helps ensure that the aspiration for quality and quantity for customers and their requirements, particularly for large scale projects, sought by the Multibanking Design pattern are documented tried and tested solutions for recurring problems in a given context. So basically you have a problem context and the proposed solution for the same.

## **INSTALLATION**

Installation is the process during which the current system is replaced by the new system. This includes conversion of existing data, software, and documentation and work procedures to those consistent with the new system.

## **DOCUMENTATION**

Documentation is descriptive information that describes the use and operation of the system. The user guide is provided to the end user as the student and administrator. The documentation part contains the details as follows,

customer requirement and bank details administration has been made online. Any customer can request to the respective manager through online and also use of documentation, they can view the purpose of each purpose, The manager could verify the authentication of the users, users request and need to take approve/reject process, thus the documentation is made of full view of project thus it gives the guideline to study the project and how to execute also.

## **USER TRAINING AND SUPPORT**

The software is installed at the deployment environment, the developer will give training to the end user of the bank admin officer in that software. The goal of an end user training program is to produce a motivated user who has the skills needed to apply what has been to apply what has been learned to perform the job related task. The following are the instruction which is specified the handling and un-handling events in the application,

- The authenticated user of admin and office workers only login in the application with authorized username and password.
- Don't make user waste their time to come straight to the bank or make a phone call.
- It can easily track through online by the user.
- Very user friendliness software

## **IMPLEMENTATION PROCEDURES**

Implementation includes all the activities that take place to convert the old system to the new one. Proper implementation is essential to provide a reliable system to meet the organization requirements. Implementation is the stage in the project where the theoretical design is turned into a working system. The most crucial stage is achieving a successful new system & giving the user confidence in that the new system will work efficiently & effectively in the implementation state.

## **IMPLEMENTATION PROCEDURES**

### **PILOT RUNNING**

Processing the current data by only one user at a time called the pilot running process. When one user is accessing the data at one system, the system is set to be engaged and connected in network. This process is useful only in system where more than one user is restricted.

### **PARALLEL RUNNING:**

Processing the current data by more than one user at a time simultaneously is said to be parallel running process. This same system can be viewed and accessed by more than one user at the time. Hence the implementation method used in the system is a pilot type of implementation.

Implementation is the stage in the project where the theoretical design is turned into a working system. The most crucial stage is achieving a successful new system & giving the user confidence in that the new system will work efficiently & effectively in the implementation state.

The stage consists of,

- Testing the developed program with sample data.
- Detection's and correction of error.
- Creating whether the system meets user requirements.
- Making necessary changes as desired by the user.
- Training user personnel.

## **USER TRAINING**

User Training is designed to prepare the user for testing &consenting the system. .

- User Manual.
- Help Screens.
- Training Demonstration.

## **USER MANUAL**

The summary of important functions about the system and software can be provided as a document to the user.

## **HELP SCREENS**

This features now available in every software package, especially when it is used with a menu. The user selects the “Help” option from the menu. The system accesses the necessary description or information for user reference.

## **TRAINING DEMONSTRATION:**

Another User Training element is a Training Demonstration. Live demonstrations with personal contact are extremely effective for Training Users.

## **SYSTEM MAINTENANCE**

Maintenance is actually the implementation of the review plan. As important as it is, many programmers and analysts are to perform or identify themselves with the maintenance effort. There are psychological, personality and professional reasons for this. Analysts and programmers spend far more time maintaining programs than they do writing them. Maintenance accounts for 50-80 percent of total system development

Maintenance is expensive. One way to reduce the maintenance costs are through maintenance management and software modification audits.

- Maintenance is not as rewarding as exciting as developing systems. It is perceived as requiring neither skill not experience.
- Users are not fully cognizant of the maintenance problem or its high cost.
- Few tools and techniques are available for maintenance.
- A good test plan is lacking.
- Standards, procedures, and guidelines are poorly defined and enforced.
- Programs are often maintained without care for structure and documentation.
- There are minimal standards for maintenance.
- Programmers expect that they will not be in their current commitment by time their programs go into the maintenance cycle.

### **Corrective Maintenance**

It means repairing, processing or performance failure or making changes because of previously uncovered problems or false assumptions. Task performed to identify, isolate, and rectify a fault so that the failed equipment, machine, or system can be restored to an operational condition within the tolerances or limits established for in-service operations.

Corrective maintenance can be subdivided into "immediate corrective maintenance" (in which work starts immediately after a failure) and "deferred corrective maintenance" (in which work is delayed in conformance to a given set of maintenance rules).

### **Perfective Maintenance**

It means changes made to a system to add new features or to improve performance. Preventive maintenance is predetermined work performed to a schedule with the aim of preventing the wear and tear or sudden failure of equipment components. process or control equipment failure can have adverse results in both human and economic terms. In addition to down time and the costs involved to repair and/or replace equipment parts or components, there is the risk of injury to operators, and of acute exposures to chemical and/or physical agents.

Time-based or run-based Periodically inspecting, servicing, cleaning, or replacing parts to prevent sudden failure .On-line monitoring of equipment in order to use important/expensive parts to the limit of their serviceable life. Preventive maintenance involves changes made to a system to reduce the chance of future system failure.

An example of preventive maintenance might be to increase the number of records that a system can process far beyond what is currently needed or to generalize how a system sends report information to a printer so that so that the system can adapt to changes in printer technology.

### **Preventive Maintenance**

Changes made to a system to avoid possible future problems Perfective maintenance involves making enhancements to improve processing performance, interface usability, or to add desired, but not necessarily required, system features. The objective of perfective maintenance is to improve response time, system efficiency, reliability, or maintainability.

During system operation, changes in user activity or data pattern can cause a decline in efficiency, and perfective maintenance might be needed to restore performance. Usually, the perfective maintenance work is initiated by the IT department, while the corrective and adaptive maintenance work is normally requested by users.

*CONCLUSION*

## **5. CONCLUSION**

Multibanking bank applications are becoming increasingly popular among customers as they provide a convenient and secure way to manage multiple bank accounts from a single platform. These applications offer several benefits to users, including enhanced convenience, improved financial management, and increased security.

With a multibanking bank application, customers can view and manage all their bank accounts from a single platform, making it easier to track their finances and manage their money efficiently. The application allows users to check their account balances, transaction history, and make transfers or payments, all from one place.

Multibanking bank applications also offer enhanced security features such as two-factor authentication, encryption, and biometric verification, which helps to protect users' financial information from unauthorized access and fraud.

Moreover, these applications provide a wealth of data and insights that can help users make informed decisions about their finances. Customers can analyze their spending patterns, track their expenses, and identify areas where they can reduce their spending, ultimately leading to better financial planning and management.

In conclusion, multibanking bank applications offer a convenient, secure, and efficient way to manage multiple bank accounts from a single platform. As technology continues to evolve, these applications are expected to become even more sophisticated, offering more advanced features and benefits to users.



*FUTURE ENHANCEMENT  
& BIBLIOGRAPHY*

## **FUTURE ENCANCEMENT**

The scope of the project includes that what all future enhancements can be done in this system to make it more feasible to us:-

- Databases for different products range and storage can be provided.
- Multilingual support can be provided so that it can be understandable by the person of any language.
- More graphics can be added to make it more user-friendly and understandable.
- Manage & backup versions of documents online.

## **BIBLIOGRAPHY**

### **Textual Reference**

- Summerfield, Mark. Programming in Python 3: A Complete Introduction to the Python Language. Addison-Wesley Professional, 2009.
- Lutz, Mark. Learning Python, 5th Edition. O'Reilly Media, 2013.
- Hill, Christian, and Harris, Chad. Learning Scientific Programming with Python. Cambridge University Press, 2016.
- McKinney, Wes. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. O'Reilly Media, 2017.
- Reitz, Kenneth, and Schlusser, Tanya. Flask Web Development: Developing Web Applications with Python. O'Reilly Media, 2018.

## Website

- W3Schools. "Python Tutorial." W3Schools, <https://www.w3schools.com/python/>
- Tutorialspoint. "Python Tutorial" tutorialspoint, <https://www.tutorialspoint.com/python/index.htm>
- Geeksforgeeks. "Learn Python Programming From scratch." geeksforgeeks, <https://www.geeksforgeeks.org/python-programming-language/learn-python-tutorial/>
- Javatpoint. "Learn Python Tutorial" javatpoint, <https://www.javatpoint.com/python-tutorial>
- Academia.edu. "Multi Banking System", [http://www.academia.edu/13143648/multi\\_Banking\\_System](http://www.academia.edu/13143648/multi_Banking_System)
- Github. "Multi-Banking-System", <http://github.com/mohitagarwal124/Multi-Banking-System>

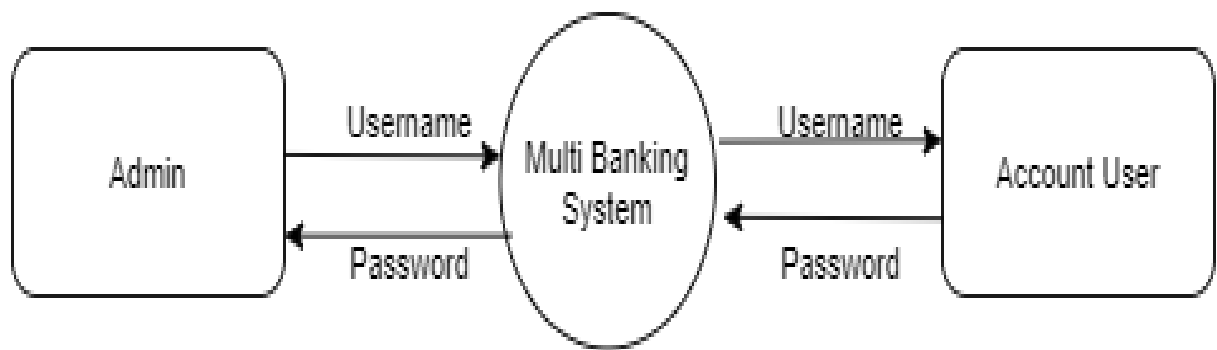
# *APPENDICES*

# APPENDICES

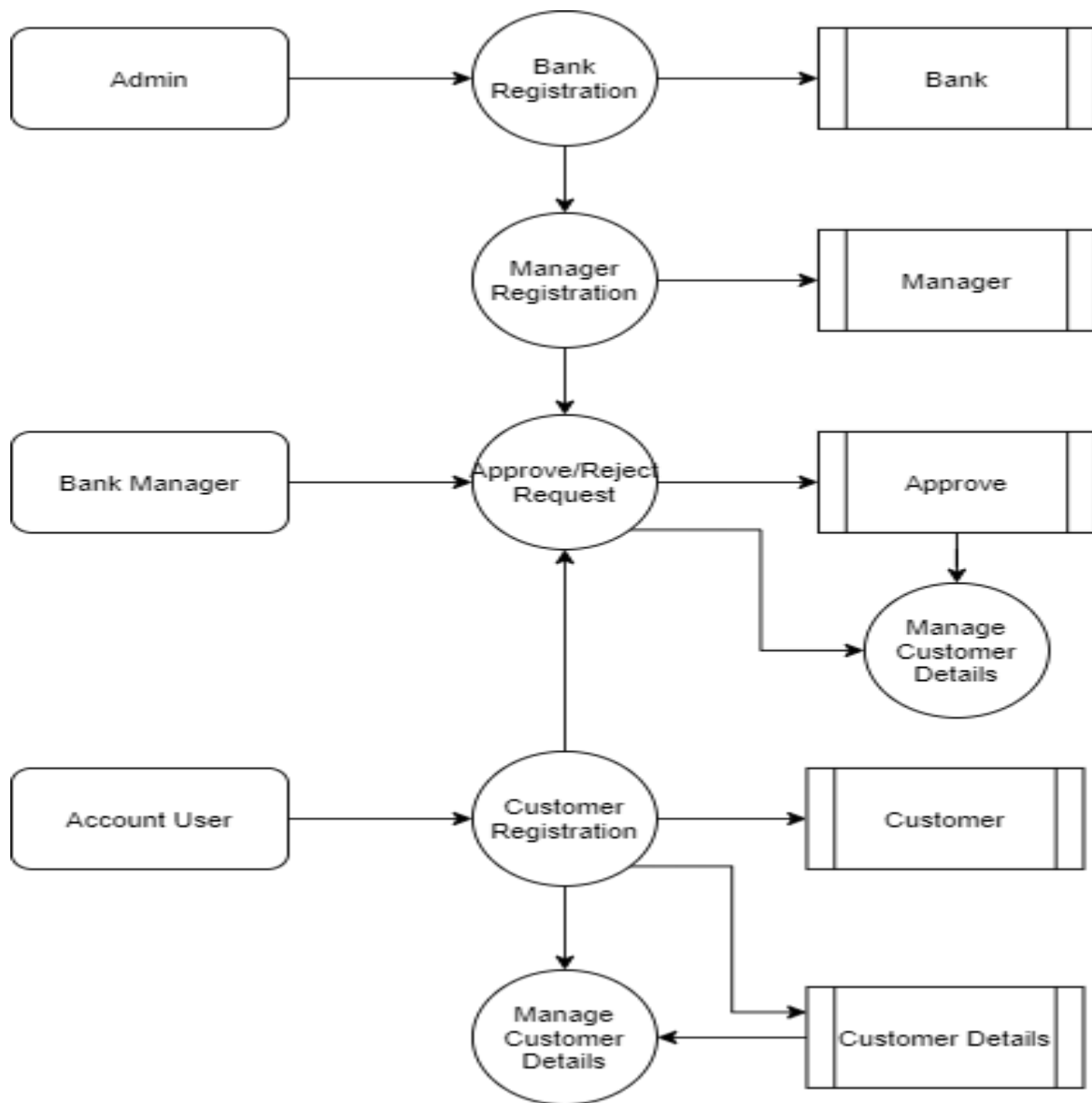
## A. DATA FLOW DIAGRAM

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or system. The DFD also provides information about the outputs and inputs of each entity and process itself. A data-flow diagram is a part of structured-analysis modeling tools.

### LEVEL 0:



## LEVEL 1:



## B. TABLE STRUCTURE

The table needed for each module was designed and the specification of each and every column was given based on the records and details collected during record specification of the system study.

**TABLE NAME: ADMIN**

FIELD	DATA TYPE	SIZE	CONSTRAINT
Admin id	INT	10	Primary key
Username	Varchar	20	Not null
Password	Varchar	20	Not null

**TABLE NAME: BANK**

FIELD	DATATYPE	SIZE	CONSTRAINT
Bank id	Int	10	Primary key
Bank Name	Varchar	30	Not Null
Branch Name	Varchar	30	Not Null
Address	Varchar	50	Not Null
Landline	Int	15	Not Null
IFSC Code	Varchar	10	Not Null

**TABLE NAME: MANAGER**

FIELD	DATA TYPE	SIZE	CONSTRAINT
Manager id	Int	10	Primary key
Bank id	Int	10	Foreign key
Branch Name	Varchar	30	Not Null
Mobile	Int	10	Not Null
Landline	Int	10	Not Null

**TABLE NAME: CUSTOMER**

FIELD	DATA TYPE	SIZE	CONSTRAINT
Customer id	Int	10	Primary key
First Name	Varchar	20	Not Null
Last Name	Varchar	20	Not Null
Mobile	Int	10	Not Null
Aadhar	Int	15	Not Null
License	Varchar	20	Not Null
Age	Int	2	Not Null
Gender	Varchar	7	Not Null
Address 1	Varchar	30	Not Null
Address 2	Varchar	30	Not Null
City	Varchar	20	Not Null
State	Varchar	20	Not Null



Pincode	Int	6	Not Null
---------	-----	---	----------

**TABLE NAME: TRANSACTION**

FIELD	DATATYPE	SIZE	CONSTRAINT
Customer id	Int	10	Primary key
First Name	Varchar	25	Not Null
Last name	Varchar	25	Not Null
Mobile	Int	10	Not Null
Aadhar	Int	15	Not Null
License	Varchar	20	Not Null
Age	Int	2	Not Null
Gender	Varchar	7	Not Null
Address 1	Varchar	30	Not Null
Address 2	Varchar	30	Not Null
City	Varchar	20	Not Null
State	Varchar	20	Not Null

**TABLE NAME: APPROVE**

FIELD	DATA TYPE	SIZE	CONSTRAINT
Approve id	Int	10	Primary key
Bank id	Int	10	Foreign key
Customer id	Int	10	Foreign key
Approve status	Varchar	7	Not null

## C. SAMPLE CODING

```
#!/usr/bin/env python
# package: com.example.demo.controller
importpython.util.List

importorg.springframework.beans.factory.annotation.Autowired

importorg.springframework.http.ResponseEntity

importorg.springframework.web.bind.annotation.GetMapping

importorg.springframework.web.bind.annotation.PathVariable

importorg.springframework.web.bind.annotation.PostMapping

importorg.springframework.web.bind.annotation.RequestMapping

importorg.springframework.web.bind.annotation.RestController

importcom.example.demo.dao.ApiDao

importcom.example.demo.response.GetBankResponse

importcom.example.demo.response.GetCitizenResponse

importcom.example.demo.response.GetComplaintResponse

importcom.example.demo.response.GetCustomerBank

importcom.example.demo.service.ApiService
```

```
@RequestMapping(value="/api")
classApiController(object):
    """ generated source for class ApiController """
    service = ApiService()
    dao = ApiDao()
```

```
@GetMapping("/login/{username}/{password}")
def login(self, username, password):
    """ generated source for method login """
    return self.service.login(username, password)
```

```
@GetMapping("/manager_login/{username}/{password}")
def manager_login(self, username, password):
    """ generated source for method manager_login """
    return self.dao.manager_login(username, password)
```

```
@PostMapping("/add_bank/{bankname}/{branch}/{address}/{ifsc}/{landline}")
def bank_register(self, bankname, branch, address, ifsc, landline):
    """ generated source for method bank_register """
    self.dao.bank_register(bankname, branch, address, ifsc, landline)
    return "Bank Saved Successfully"
```

```
@PostMapping("/add_manager/{bankid}/{mobile}/{landline}/{username}/{password}")
def add_manager(self, bankid, mobile, landline, username, password):
    """ generated source for method add_manager """
    self.dao.add_manager(bankid, mobile, landline, username, password)
    return "Manager Added Successfully"
```

```
@GetMapping("/get_bank")
def get_bank(self):
    """ generated source for method get_bank """
    return ResponseEntity.ok().body(self.service.get_bank())
```

```
@PostMapping("/add_customer/{fname}/{lname}/{mobile}/{aadhar}/{idproof}/{age}/{gen
```

```

der}/{address1}/{address2}/{city}/{state}/{pincode}/{bankId}")
defadd_customer(self, fname, lname, mobile, aadhar, idproof, age, gender, address1,
address2, city, state, pincode, bankId):
    """ generated source for method add_customer """
    self.dao.add_customer(fname, lname, mobile, aadhar, idproof, age, gender, address1,
address2, city, state, pincode, bankId)
    return "Customer Added Sucessfully"

```

```

@PostMapping("/add_bank_account/{customer_id}/{bank_id}/{account type}")
defadd_bank_account(self, customer_id, bank_id, account_type):
    """ generated source for method add_bank_account """
    self.dao.add_customer(customer_id, bank_id, account_type)
    return "Bank Request send Sucessfully"

```

```

@GetMapping("/get_customer_bank/{customer_id}")
defget_customer_bank(self, customer_id):
    """ generated source for method get_customer_bank """
    return ResponseEntity.ok().body(self.service.get_customer_bank(customer_id))

```

```

@GetMapping("/get_customer/{bank_id}")
defget_customer(self, bank_id):
    """ generated source for method get_customer """
    returnself.dao.get_customer(bank_id)

```

```

@GetMapping("/approve/{id}")
def approve(self, id):
    """ generated source for method approve """
    self.dao.approve(id)
    return "Sucessfully Approved"

```

```

#!/usr/bin/env python
# package: com.example.demo.dao
importpython.util.List

importorg.hibernate.Session

```

```
importorg.hibernate.SessionFactory
```

```
importorg.hibernate.query.NativeQuery
```

```
importorg.springframework.beans.factory.annotation.Autowired
```

```
importorg.springframework.stereotype.Repository
```

```
importpythonx.transaction.Transactional
```

```
classApiDao(object):
```

```
    """ generated source for class ApiDao """
```

```
sf = SessionFactory()
```

```
defbank_register(self, bankname, branch, address, ifsccode, landline):
```

```
    """ generated source for method bank_register """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "INSERT INTO `bank` (`id`, `bankname`, `branch`, `address`, `ifsccode`, `landline`) " + "  
VALUES (NULL, '" + bankname + "', '" + branch + "', '" + address + "', '" + ifsccode + "', '" + landline  
+ "');" 
```

```
session.createSQLQuery(sql).executeUpdate()
```

```
defget_bank(self):
```

```
    """ generated source for method get_bank """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "Select * from bank"
```

```
nq = session.createNativeQuery(sql)
```

```
returnnq.list_()
```

```
defadd_manager(self, bankid, mobile, landline, username, password):
```

```
    """ generated source for method add_manager """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "INSERT INTO `manager` (`id`, `bank_id`, `mobile`, `landline`, `username`, `password`) " + "  
VALUES (NULL, " + bankid + ", '" + mobile + "', '" + landline + "', '" + username + "', '" + password  
+ "');" 
```

```
session.createSQLQuery(sql).executeUpdate()
```

```

def login(self, username, password):
    """ generated source for method login """
    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()
    sql = "select * from admin where username='" + username + "' and password='" + password + "'"
    nq = session.createNativeQuery(sql)
    if nq.list_().size() != 0:
        return "admin"
    else:
        if nq1.list_().size() != 0:
            return "manager-home.html?bankId=" + a.get(0)[1]
        else:
            if nq2.list_().size() != 0:
                return "customer-home.html?id=" + a.get(0)[0]
                # return "customer";
            else:
                return "Invalid"

```

```

def manager_login(self, username, password):
    """ generated source for method manager_login """
    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()
    sql = "select * from manager where username='" + username + "' and password='" + password + "'"
    nq = session.createNativeQuery(sql)
    if nq.list_().size() != 0:
        return True
    else:
        return False

```

@overloaded

```

def add_customer(self, fname, lname, mobile, aadhar, idproof, age, gender, address1, address2, city,
state, pincode, bankId):
    """ generated source for method add_customer """
    # TODO Auto-generated method stub
    session = self.sf.getCurrentSession()
    sql = "INSERT INTO `customer` (`id`,`bankId`,`fname`,`lname`,`mobile`,`aadhar`,`idproof`,`age`,`gender`,`address1`,`address2`,`city`,`state`,`pincode`) " + "VALUES (NULL, '" + bankId + "', '" +
fname + "', '" + lname + "', '" + mobile + "', '" + aadhar + "', '" + idproof + "', '" + age + "', '" + gender + "', '" + address1 + "', '" + address2 + "', '" + city + "', '" + state + "', '" + pincode + "');"

```

```
session.createSQLQuery(sql).executeUpdate()
```

```
@add_customer.register(object, int, str, str)
```

```
def add_customer_0(self, customer_id, bank_id, account_type):
```

```
    """ generated source for method add_customer_0 """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "INSERT INTO `account` (`id`, `customer_id`, `bank_id`, `account_type`) VALUES " +
```

```
"(NULL, '" + customer_id + "', '" + bank_id + "', '" + account_type + "');" 
```

```
session.createSQLQuery(sql).executeUpdate()
```

```
def get_customer_bank(self):
```

```
    """ generated source for method get_customer_bank """
```

```
session = self.sf.getCurrentSession()
```

```
sql = "Select * from account where id="
```

```
nq = session.createNativeQuery(sql)
```

```
return nq.list_()
```

```
def get_customer(self, bank_id):
```

```
    """ generated source for method get_customer """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "Select * from customer where status!='Approved' AND bankId=" + bank_id
```

```
nq = session.createNativeQuery(sql)
```

```
return nq.list_()
```

```
def approve(self, id):
```

```
    """ generated source for method approve """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "UPDATE `customer` SET `status` = 'Approved' WHERE `customer`.`id` = " + id
```

```
session.createSQLQuery(sql).executeUpdate()
```

```
#!/usr/bin/env python
```

```
# package: com.example.demo.configuration
```

```
import python.util.Properties
```

```
import pythonx.sql.DataSource
```

```
importorg.springframework.beans.factory.annotation.Value
```

```
importorg.springframework.context.annotation.Bean
```

```
importorg.springframework.context.annotation.Configuration
```

```
importorg.springframework.jdbc.datasource.DriverManagerDataSource
```

```
import org.springframework.orm.hibernate5.HibernateTransactionManager
```

```
import org.springframework.orm.hibernate5.LocalSessionFactoryBean
```

```
import org.springframework.transaction.annotation.EnableTransactionManagement
```

```
@Value("${db.driver}")
```

```
@Value("${db.password}")
```

```
@Value("${db.url}")
```

```
@Value("${db.username}")
```

```
@Value("${hibernate.dialect}")
```

```
@Value("${hibernate.show_sql}")
```

```
@Value("${entitymanager.packagesToScan}")
```

```
classHibernateConfiguration(object):
```

```
    """ generated source for class HibernateConfiguration """
```

```
    DB_DRIVER = str()
```

```
    DB_PASSWORD = str()
```

```
    DB_URL = str()
```

```
    DB_USERNAME = str()
```

```
    HIBERNATE_DIALECT = str()
```

```
    HIBERNATE_SHOW_SQL = str()
```

```
#    @Value("${hibernate.hbm2ddl.auto}")
```

```
    HIBERNATE_HBM2DDL_AUTO = str()
```

```
    ENTITYMANAGER_PACKAGES_TO_SCAN = str()
```

```
defsessionFactory(self):
```

```
    """ generated source for method sessionFactory """
```

```
    sessionFactory = LocalSessionFactoryBean()
```

```
    sessionFactory.setDataSource(dataSource())
```

```
    sessionFactory.setPackagesToScan(self.ENTITYMANAGER_PACKAGES_TO_SCAN)
```

```
    hibernateProperties = Properties()
```



```

hibernateProperties.put("hibernate.dialect", self.HIBERNATE_DIALECT)
hibernateProperties.put("hibernate.show_sql", self.HIBERNATE_SHOW_SQL)
#         hibernateProperties.put("hibernate.hbm2ddl.auto", HIBERNATE_HBM2DDL_AUTO);
sessionFactory.setHibernateProperties(hibernateProperties)
return sessionFactory

```

```

def dataSource(self):
    """ generated source for method dataSource """
    dataSource = DriverManagerDataSource()
    dataSource.setDriverClassName(self.DB_DRIVER)
    dataSource.setUrl(self.DB_URL)
    dataSource.setUsername(self.DB_USERNAME)
    dataSource.setPassword(self.DB_PASSWORD)
    return dataSource

```

```

def transactionManager(self):
    """ generated source for method transactionManager """
    txManager = HibernateTransactionManager()
    txManager.setSessionFactory(self.sessionFactory().getObject())
    return txManager

```

```

@PostMapping("/add_customer/{fname}/{lname}/{mobile}/{aadhar}/{idproof}/{age}/{gender}/{address1}/{address2}/{city}/{state}/{pincode}/{bankId}")
def add_customer(self, fname, lname, mobile, aadhar, idproof, age, gender, address1, address2, city, state, pincode, bankId):
    """ generated source for method add_customer """
    self.dao.add_customer(fname, lname, mobile, aadhar, idproof, age, gender, address1, address2, city, state, pincode, bankId)
    return "Customer Added Successfully"

```

```

@PostMapping("/add_bank_account/{customer_id}/{bank_id}/{account_type}")
def add_bank_account(self, customer_id, bank_id, account_type):
    """ generated source for method add_bank_account """
    self.dao.add_customer(customer_id, bank_id, account_type)
    return "Bank Request send Successfully"

```

```

@GetMapping("/get_customer_bank/{customer_id}")
def get_customer_bank(self, customer_id):
    """ generated source for method get_customer_bank """

```

```
return ResponseEntity.ok().body(self.service.get_customer_bank(customer_id))
```

```
@GetMapping("/get_customer/{bank_id}")
```

```
def get_customer(self, bank_id):
```

```
    """ generated source for method get_customer """
```

```
return self.dao.get_customer(bank_id)
```

```
@GetMapping("/approve/{id}")
```

```
def approve(self, id):
```

```
    """ generated source for method approve """
```

```
self.dao.approve(id)
```

```
return "Sucessfully Approved"
```

```
#!/usr/bin/env python
```

```
# package: com.example.demo.dao
```

```
import python.util.List
```

```
import org.hibernate.Session
```

```
import org.hibernate.SessionFactory
```

```
import org.hibernate.query.NativeQuery
```

```
import org.springframework.beans.factory.annotation.Autowired
```

```
import org.springframework.stereotype.Repository
```

```
import python.transaction.Transactional
```

```
class ApiDao(object):
```

```
    """ generated source for class ApiDao """
```

```
sf = SessionFactory()
```

```
def bank_register(self, bankname, branch, address, ifsc_code, landline):
```

```
    """ generated source for method bank_register """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "INSERT INTO `bank` (`id`, `bankname`, `branch`, `address`, `ifsc_code`, `landline`) + "  
VALUES (NULL, " + bankname + ", " + branch + ", " + address + ", " + ifsc_code + ", " + landline  
+ ");"
```

```
session.createSQLQuery(sql).executeUpdate()
```

```
def get_bank(self):
```

```
    """ generated source for method get_bank """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "Select * from bank"
```

```
nq = session.createNativeQuery(sql)
```

```
return nq.list_()
```

```
def add_manager(self, bankid, mobile, landline, username, password):
```

```
    """ generated source for method add_manager """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "INSERT INTO `manager` (`id`, `bank_id`, `mobile`, `landline`, `username`, `password`) " +  
"VALUES (NULL, " + bankid + ", " + mobile + ", " + landline + ", " + username + ", " + password  
+ ");"
```

```
session.createSQLQuery(sql).executeUpdate()
```

```
def login(self, username, password):
```

```
    """ generated source for method login """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "select * from admin where username='" + username + "' and password='" + password + "'"
```

```
nq = session.createNativeQuery(sql)
```

```
if nq.list_().size() != 0:
```

```
    return "admin"
```

```
else:
```

```
    if nq1.list_().size() != 0:
```

```
        return "manager-home.html?bankId=" + a.get(0)[1]
```

```
    else:
```

```
        if nq2.list_().size() != 0:
```

```
            return "customer-home.html?id=" + a.get(0)[0]
```

```
                # return "customer";
```

```
    else:
```

```
        return "Invalid"
```

```
def manager_login(self, username, password):
```

```
    """ generated source for method manager_login """
```

```
# TODO Auto-generated method stub
```

```

session = self.sf.getCurrentSession()
sql = "select * from manager where username='" + username + "' and password='" + password + "'"
nq = session.createNativeQuery(sql)
if nq.list_().size() != 0:
    return True
else:
    return False

# package: com.example.demo.dao
import python.util.List

import org.hibernate.Session

import org.hibernate.SessionFactory

import org.hibernate.query.NativeQuery

import org.springframework.beans.factory.annotation.Autowired

import org.springframework.stereotype.Repository

import python.transaction.Transactional

class ApiDao(object):
    """ generated source for class ApiDao """
    sf = SessionFactory()

    def bank_register(self, bankname, branch, address, ifsccode, landline):
        """ generated source for method bank_register """
        # TODO Auto-generated method stub
        session = self.sf.getCurrentSession()
        sql = "INSERT INTO `bank` (`id`, `bankname`, `branch`, `address`, `ifsccode`, `landline`)" + "
VALUES (NULL, '" + bankname + "', '" + branch + "', '" + address + "', '" + ifsccode + "', '" + landline
+ "');"
        session.createSQLQuery(sql).executeUpdate()

    def get_bank(self):
        """ generated source for method get_bank """
        # TODO Auto-generated method stub
        session = self.sf.getCurrentSession()

```

```
sql = "Select * from bank"
```

```
nq = session.createNativeQuery(sql)
```

```
return nq.list_()
```

```
def add_manager(self, bankid, mobile, landline, username, password):
```

```
    """ generated source for method add_manager """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "INSERT INTO `manager` (`id`, `bank_id`, `mobile`, `landline`, `username`, `password`) " +  
"VALUES (NULL, " + bankid + ", " + mobile + ", " + landline + ", " + username + ", " + password  
+ ");"
```

```
session.createSQLQuery(sql).executeUpdate()
```

```
def login(self, username, password):
```

```
    """ generated source for method login """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "select * from admin where username='" + username + "' and password='" + password + "'"
```

```
nq = session.createNativeQuery(sql)
```

```
if nq.list_().size() != 0:
```

```
    return "admin"
```

```
else:
```

```
    if nq1.list_().size() != 0:
```

```
        return "manager-home.html?bankId=" + a.get(0)[1]
```

```
    else:
```

```
        if nq2.list_().size() != 0:
```

```
            return "customer-home.html?id=" + a.get(0)[0]
```

```
                # return "customer";
```

```
    else:
```

```
        return "Invalid"
```

```
def manager_login(self, username, password):
```

```
    """ generated source for method manager_login """
```

```
# TODO Auto-generated method stub
```

```
session = self.sf.getCurrentSession()
```

```
sql = "select * from manager where username='" + username + "' and password='" + password + "'"
```

```
nq = session.createNativeQuery(sql)
```

```
if nq.list_().size() != 0:
```

```
    return True
```

```
else:
```

return False

@overloaded

defadd\_customer(self, fname, lname, mobile, aadhar, idproof, age, gender, address1, address2, city, state, pincode, bankId):

""" generated source for method add\_customer """

# TODO Auto-generated method stub

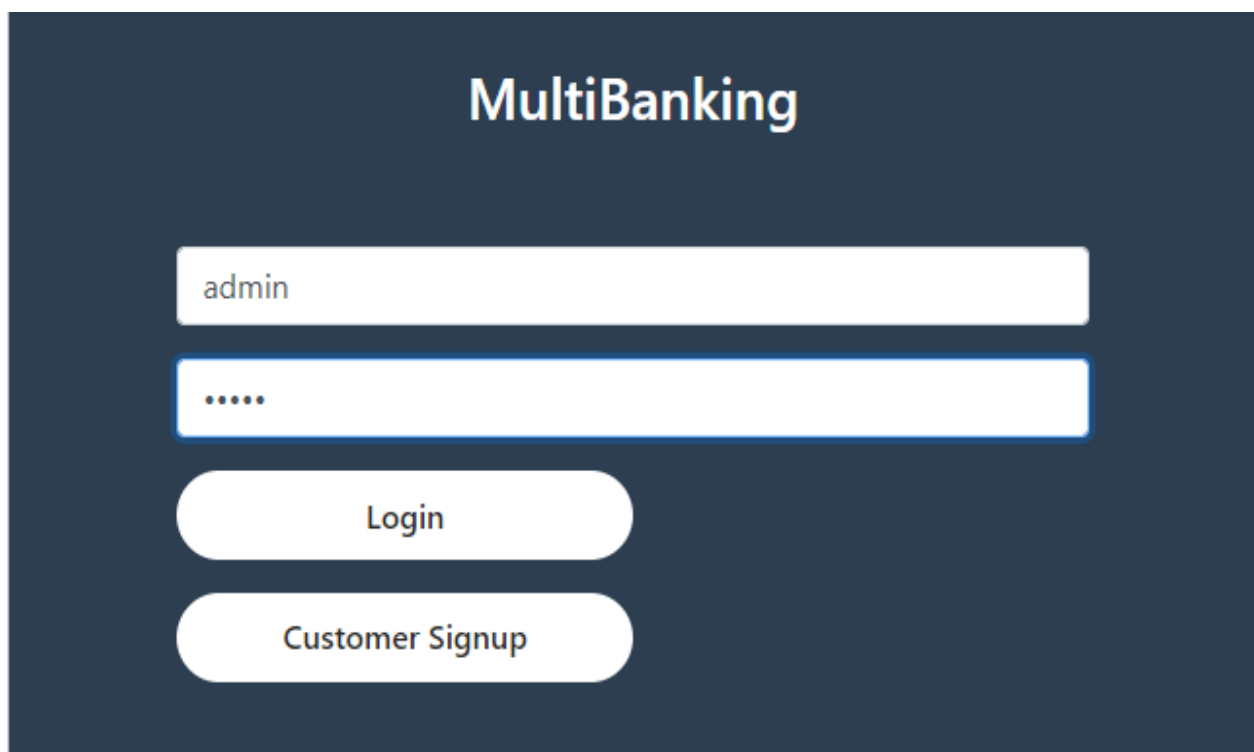
session = self.sf.getCurrentSession()

sql = "INSERT INTO `customer` (`id`,`bankId`,`fname`,`lname`,`mobile`,`aadhar`,`idproof`,`age`,`gender`,`address1`,`address2`,`city`,`state`,`pincode`) " + "VALUES (NULL,'" + bankId + "','" + fname + "','" + lname + "','" + mobile + "','" + aadhar + "','" + idproof + "','" + age + "','" + "" + gender + "','" + address1 + "','" + address2 + "','" + city + "','" + state + "','" + pincode + ');"

session.createSQLQuery(sql).executeUpdate()

## D. SAMPLE INPUT & OUTPUT DESIGN

### ADMIN LOGIN



The image shows a login interface for 'MultiBanking'. It features a dark blue background with the title 'MultiBanking' in white. Below the title are two white input fields: the first contains the text 'admin' and the second contains six dots, indicating a password field. Below these fields are two white, rounded rectangular buttons. The top button is labeled 'Login' and the bottom button is labeled 'Customer Signup'.

MultiBanking

admin

.....

Login

Customer Signup

# BANK REGISTRATION

Bank Registration	Manager Registration	Logout
-------------------	----------------------	--------

Bank Name

ICICI bank

Branch Name

palladam

Address

anna nagar

IFSC Code

11456787

Landline

9876543210

Create Bank



# MANAGER REGISTRATION

Bank Registration	Manager Registration	Logout
-------------------	----------------------	--------

Select Bank

ICICI bank-8345762109

Mobile

764539754

Landline

1234567890

Username

vinoth

Password

\*\*\*|

Create Manager

# CUSTOMER REGISTRATION

Customer Registration

First Name

dhanu

Last Name

dhanu

Mobile

8610200586

Select Bank

canara bank-1234567890

Aadhar

11112222

Gender

female

Address 1

anna nagar

Address 2

kumaran colony

City

tirupur

State

tamilnadu

Pincode

641666

Create Bank Account

## MANAGER LOGIN

### MultiBanking

Login

Customer Signup

# APPROVE LIST

Approve List

Logout

#	First Name	Last Name	Mobile	Aadhar	pincode	Approve
1	dhanu	dhanu	8610200586	11112222	641666	<div>Approve</div>

## CUSTOMER LOGIN

### MultiBanking

Login

Customer Signup

# TRANSACTION

Transaction

Logout

From Account

Select Account

To Account

IFSC code

Amount

Payment