

A PROJECT REPORT ON

VISHNU LEGACY BANK

Summer Training Project

Of

BACHELOR OF COMPUTER APPLICATIONS (BCA)

To

**GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY,
DELHI**

Submitted By

Kirti Shukla(50191102021)

UNDER THE SUPERVISION OF

Mr. Sanjay Upadhyay

(Trainer)



SESSION: 2021-2024

**SRI GURU TEGH BAHADUR INSTITUTE OF
MANAGEMENT AND INFORMATION TECHNOLOGY**

ADJ. GURUDWARA NANAK PIAO, G.T. KARNAL ROAD, DELHI-110033
(AFFILIATED TO GURU GOBIND SINGH INDRAPRASTHA
UNIVERSITY, DWARKA, DELHI)

DECLARATION

I, Kirti Shukla , Enrolled As A Trainee, Hereby Declare That The Summer Training Project Titled "Vishnu Legacy Bank" Is The Outcome Of My Efforts And Is A Genuine Representation Of My Work During This Training Period. I Have Been Under The Guidance And Mentorship Of Mr. Sanjay Upadhya.

All External Sources And References Utilized In This Project Have Been Appropriately Acknowledged And Cited. I Take Full Responsibility For The Content And Originality Of This Project, Confirming That It Is My Independent Work And Does Not Infringe Upon Any Copyrights Or Proprietary Information Of Any Organization.

I Also Affirm That This Project, In Its Entirety, Has Not Been Previously Submitted For Any Degree Or Diploma At This Institute, Or At Any Other Institution Or University.

Kirti Shukla(50191102021)

ACKNOWLEDGEMENT

First and foremost, we would like to express our heartfelt appreciation to our mentor, Mr. Sanjay Upadhyaya, who served as a constant source of inspiration throughout this training project. His unwavering encouragement, innovative thinking, and unhesitating support have been instrumental in our journey. Mr. Upadhyaya's extensive knowledge, vast experience, and professional expertise have paved the way for the successful execution of this project. We are truly grateful for his exceptional guidance and supervision, and we consider ourselves fortunate to have had such an outstanding mentor in our training.

This endeavor would not have been possible without the collective contributions of our colleagues and fellow trainees. We all stood by each other, providing motivation and support that proved essential for our collective success.

We would like to extend our thanks to Tech Access for granting us the opportunity to work on this project and gain valuable experience. Without their support, this project would not have been possible.

Kirti Shukla(50191102021)

ABSTRACT

The "Vishnu Legacy Bank" project is a comprehensive endeavor designed to enhance and modernize the banking experience by introducing a range of innovative features and an advanced expense predictor. This project aims to revolutionize traditional banking practices and provide customers with more efficient, user-friendly, and intelligent banking solutions.

The key objective of this project is to bring together various banking operations and services, encompassing traditional banking, digital transactions, and financial planning tools, into a unified platform. By doing so, it aims to simplify and enhance the banking experience for customers.

The introduction of these new features, including the expense predictor, will significantly improve financial planning and management for customers. The expense predictor will utilize data analytics to provide accurate and real-time insights into spending patterns, helping customers make informed decisions and achieve their financial goals.

The "Vishnu Legacy Bank" project offers numerous advantages. It will increase operational efficiency by automating routine banking tasks, reducing processing times, and enhancing customer satisfaction. Real-time access to financial data and predictive tools will empower customers to make better financial decisions, ultimately improving their financial well-being. Additionally, it will foster stronger customer engagement, enabling seamless communication and improved banking services.

The successful implementation of the "Vishnu Legacy Bank" project will not only transform the banking experience but also contribute to the growth and success of the bank. It equips both customers and bank employees with valuable tools and resources, allowing them to focus on more strategic financial planning and wealth management. Customers will enjoy a more convenient and secure banking experience, resulting in improved financial outcomes and increased satisfaction.

In summary, the "Vishnu Legacy Bank" project represents a groundbreaking step in the evolution of banking services, providing customers with an intelligent and user-centric approach to banking while delivering enhanced financial planning tools for a more prosperous future.

TABLE OF CONTENTS

SNO.	CONTENT	PAGE NO.
1.	LIST OF TABLES	7
2.	LIST OF FIGURES	7
3.	LIST OF SYMBOLS	7
4.	CHAPTER 1: INTRODUCTION	8
5.	1.1 INTRODUCTION TO THE PROJECT	8
6.	1.2 FEATURES OF THE PROJECT	8
7.	1.3 OBJECTIVE AND SCOPE OF THE PROJECT	8
8.	1.4 NEED OF THE PROJECT	8
9.	1.5 LIMITATIONS OF THE PROJECT	8
10.	CHAPTER 2: REQUIREMENT AND ANALYSIS	9
11.	2.1 REQUIREMENTS AND ANALYSIS	9
12.	2.2 MODEL USED FOR THE PROJECT	9
13.	2.3 FUNCTIONAL REQUIREMENTS	9
14.	2.4 SOFTWARE REQUIREMENTS	9
15.	2.5 HARDWARE REQUIREMENTS	10
16.	2.6 USE-CASE DIAGRAM	10
17.	CHAPTER 3: SOFTWARE DESIGN	11
18.	3.1 INTRODUCTION	11
19.	3.2 FLOW CHART	11
20.	3.3 DATASET DESCRIPTION	12
21.	CHAPTER 4: DATABASE DESIGN	14
22.	4.1 INTRODUCTION	14
23.	4.3 DATABASE FIELD SPECIFICATION(TABLES)	15-16

25.	CHAPTER 5: TESTING	29
26.	5.1 INTRODUCTION	29
26.	5.2 TESTING METHODS	29-30
27.	5.3 TEST CASES	30
28.	CHAPTER 6: ROLES AND RESPONSIBILITY	31
29.	6.1 PROJECT ROLES AND RESPONSIBILITY	31
30.	CHAPTER 7: CONCLUSION AND FUTURE ENHANCEMENTS	32
31.	7.1 CONCLUSION	32
32.	7.2 FUTURE ENHANCEMENTS	32
33.	CHAPTER 8: APPENDICES (SCREENSHOTS AND CODING)	33
34.	REFERENCES	33-70




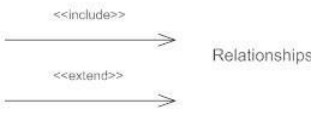
LIST OF TABLES

SNO.	NAME OF THE TABLES	PAGE NO.
1.	Registration (Client Detail)	22
2.	Card_Info	23

LIST OF FIGURES

SNO.	TITLES	PAGE NO.
1.	Figure No. 1: Waterfall Model	9
2.	Figure No 2: Use Case Diagram	11
3.	Dataset Description	13
4.	Flow chart	14-15

LIST OF SYMBOLS

SYMBOL	NAME	FUNCTION
 System	System	Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.
 Use case	Use Case	Draw use cases using ovals. Label the ovals with verbs that represent the system's functions.
 Actor	Actors	Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.
 Relationships	Relationships	Illustrate relationships between an actor and a use case with a simple line. A "uses" relationship indicates that one use case is needed by another to perform a task. An "extends" relationship indicates alternative options under a certain use case.

Chapter 1: INTRODUCTION

1.1 INTRODUCTION

The Vishnu Legacy Bank redefines the modern banking experience by offering a wide array of innovative features, with a strong focus on security and user-centricity. This project encompasses features such as multi-level user authentication, account registration, deletion, deposits, withdrawals, fund transfers, expense prediction, account modification, and the addition of card types. Every step of this banking journey is fortified by stringent security measures, requiring both a unique User_Name and Password at each point.

1.2 FEATURES OF THE PROJECT

Key features of the project include:

- **Multi-level User Authentication:** At each step of a banking operation, the project demands a unique User_Name and Password combination for enhanced security.
- **Registration:** New customers can create accounts with the bank, providing them with a unique account ID and access to various banking services, with secure authentication at every point.
- **Account Deletion:** Customers have the ability to delete their accounts when needed, reinforced by rigorous authentication measures.
- **Deposits:** Customers can make deposits into their accounts, with multi-level security to safeguard their transactions.
- **Withdrawals:** The project ensures secure withdrawals, requiring user authentication for each transaction.
- **Fund Transfers:** Customers can transfer funds between accounts with robust authentication protocols.
- **Expense Prediction:** An advanced expense predictor employs data analytics to offer real-time insights into spending patterns, facilitating informed financial decisions for customers.
- **Account Modification:** Customers can make secure modifications to their accounts, ensuring flexibility and user control.
- **Card Type Addition:** The project allows customers to add different card types to their accounts, expanding their banking options.
- **Advanced Error Handling:** The system is equipped with advanced error-handling features to provide a secure and seamless banking experience.
- **Rule based chatbot :**clear queries

1.3 OBJECTIVE AND SCOPE

The primary objectives and scope of the Vishnu Legacy Bank project encompass:

1. **Security-First Banking:** The project prioritizes the security of banking operations by implementing multi-level user authentication at every step.
2. **User Empowerment:** The project empowers customers with control over their finances, ensuring their transactions are secure and trustworthy.
3. **Financial Planning:** The advanced expense predictor assists customers in making informed financial decisions, improving their financial well-being.

1.4 NEED OF THE PROJECT

The need for the Vishnu Legacy Bank project arises from:

- **Security Assurance:** Customers demand secure and trustworthy banking services, and this project caters to that need by implementing stringent security measures at each step.
- **User-Centric Banking:** Modern banking requires customer control and convenience, which this project offers.
- **Data-Driven Financial Management:** Customers need data-driven tools to make informed financial decisions, and the expense predictor meets this demand.

1.5 LIMITATIONS OF THE PROJECT

Despite its advanced features, the project has certain limitations, including:

- **Platform Limitations:** The project is currently available as a desktop application and is not accessible on mobile phones or web platforms.

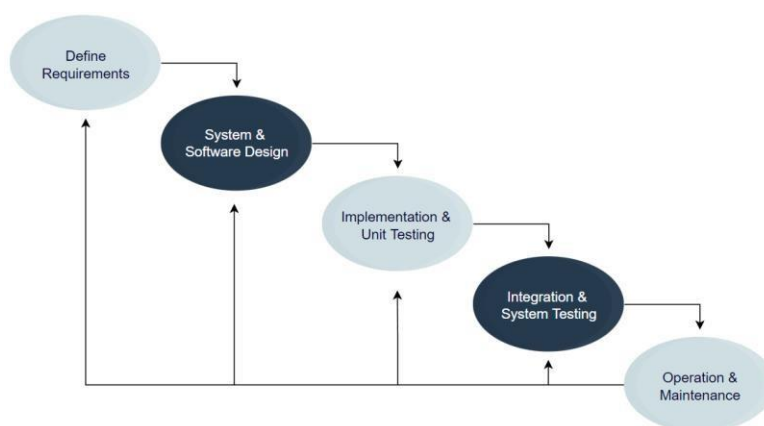
- Implementation: Expenses related to software, hardware, and training may be associated with project implementation.
- Requirements: Laptop or Computer System is Required
- Implementation Duration: Implementing the project may take time, potentially impacting regular banking operations.
- Inter-departmental Efficiency: While promoting efficiency, interconnectivity can present challenges, necessitating careful planning and optimization within the

CHAPTER 2: REQUIREMENT AND ANALYSIS

2.1 SOFTWARE REQUIREMENT SPECIFICATION

The Software Requirement Specification (SRS) is a comprehensive document that defines the expected performance and functionality of the Vishnu Legacy Bank system. This document serves as a foundation for all software engineering activities and is created once all the requirements have been elicited and analyzed. The SRS is a formal report that represents the software system, enabling customers to review its alignment with their needs. It includes user requirements and detailed specifications.

2.2 MODEL USED (Waterfall Model)



The development model utilized for the Vishnu Legacy Bank project is the Waterfall Model. This model encompasses the following sequential phases:

- **Requirement Gathering and Analysis:** This phase captures all potential system requirements and documents them in a requirement specification document.
- **System Design:** It involves studying the requirement specifications from the first phase and preparing the system design, specifying hardware and system requirements, and defining the overall system architecture.
- **Implementation:** The system is developed in small units based on the system design, which are integrated into the next phase. Each unit undergoes unit testing to verify its functionality.
- **Integration and Testing:** All units developed in the implementation phase are integrated into a complete system after thorough testing of each unit. The entire system is tested for any defects or failures.
- **Deployment of System:** After functional and non-functional testing, the product is deployed in the customer environment or released into the market.
- **Maintenance:** Ongoing maintenance is performed to address issues arising in the client environment. Patches are released to fix these issues and to enhance the product with newer versions.

2.3 FUNCTIONAL REQUIREMENTS

The functional requirements for the Vishnu Legacy Bank system are as follows:

- Users must possess a valid User Name and Password to log in or sign up to create their individual profiles.
- Admin can log in using a unique User Id and Password.
- Users can access and view their personal details, marks, and attendance records.
- Client can add, delete, and modify student details, marks, and attendance records for future reference.
- Security has the authority to add, delete, or update faculty and user/student information.
- user-friendly it has chatbot for queries

2.4 SOFTWARE REQUIREMENTS

The software requirements for the Vishnu Legacy Bank project are as follows

- **Operating System:** Compatible with Windows and other relevant systems.

- Front-End: Utilizes Tkinter for the user interface, pandas for data manipulation, linear regression for predictive analysis, and numpy for numerical operations.
- Back-End: Relies on SQL Server version 19.01 for secure data management.
- Usage: Designed as a Desktop Application for easy access and efficient use.
- Language Used: Developed using the Python programming language.
- Tools: Developed and integrated using Visual Studio Code for streamlined development and maintenance.

2.5 HARDWARE REQUIREMENTS

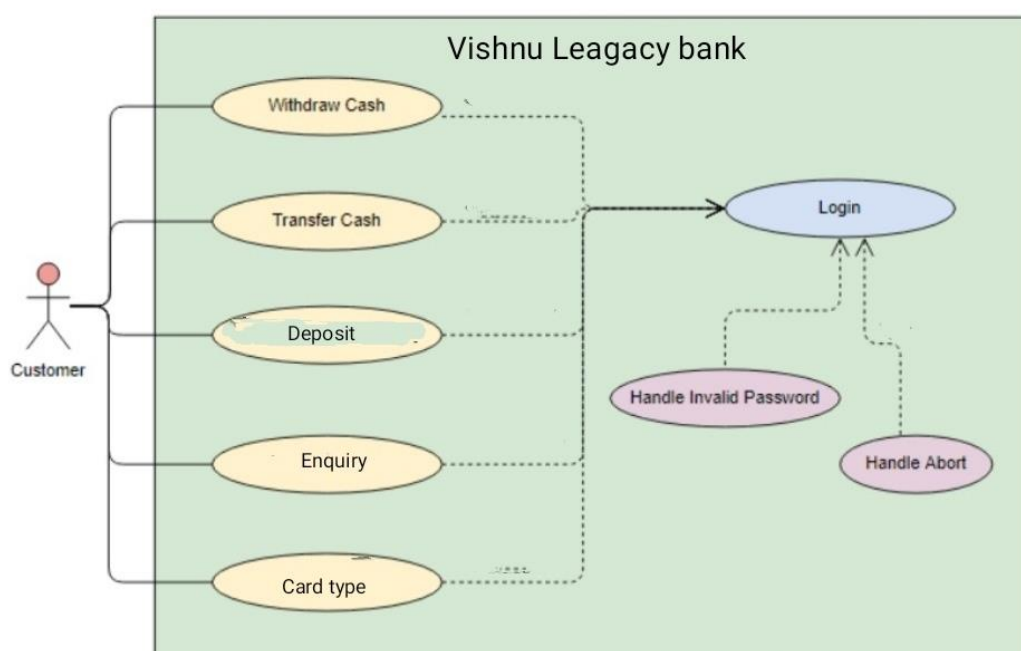
The hardware requirements for the Vishnu Legacy Bank project are as follows:

Processor Speed: 2.5 GHz and above

Hard Disk: 2 GB to 30 GB

RAM: 4GB

2.6 USE CASE DIAGRAM



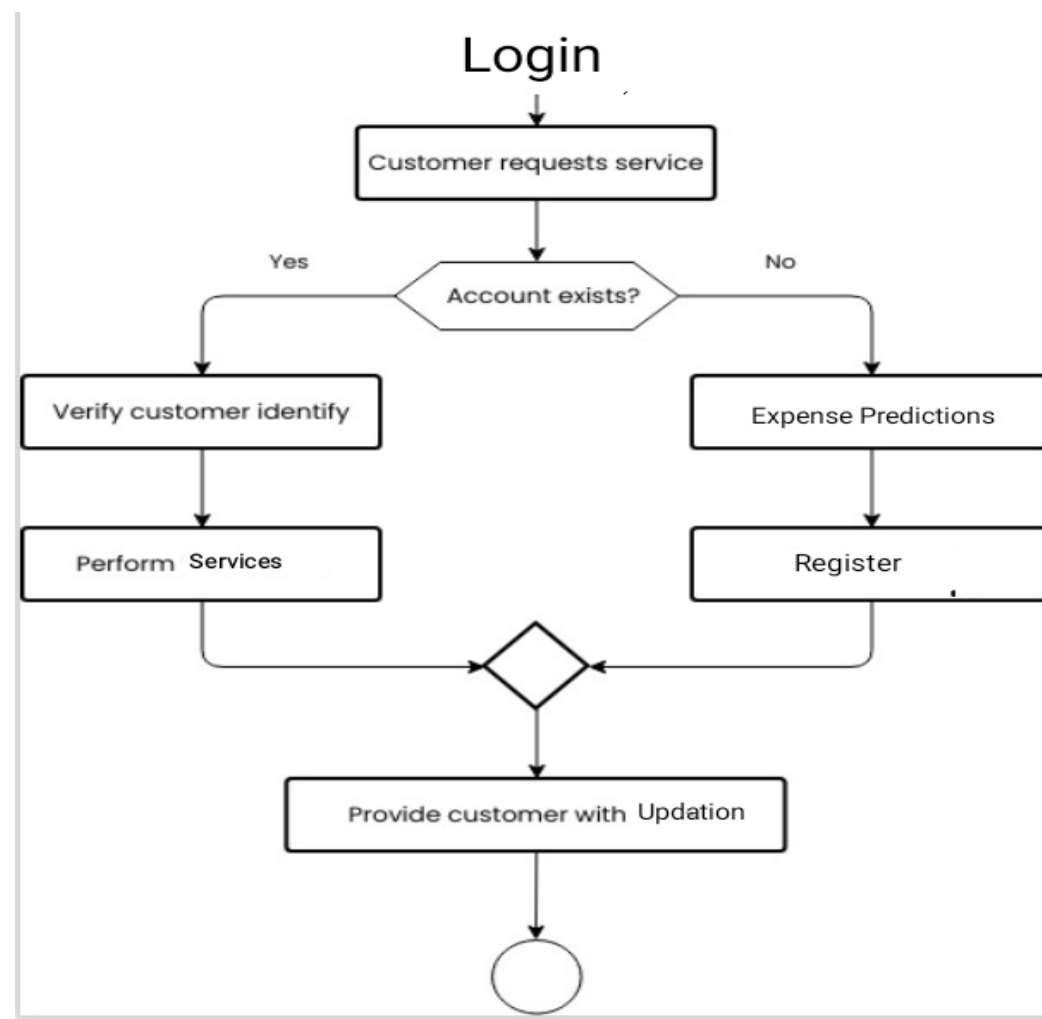
CHAPTER 3: SOFTWARE DESIGN

3.1 INTRODUCTION

Software design is the process of translating user requirements into a suitable form that facilitates software coding and implementation. It involves representing the client's requirements, as outlined in the SRS (Software Requirement Specification) document, in a format that is easily implementable using a programming language. This chapter is dedicated to the design of the Vishnu Legacy Bank system, incorporating flowcharts to visually illustrate specific sections of the software system.

3.2 FLOWCHARTS

Flowcharts are a visual representation of how information flows within a process or system. They serve to improve our understanding of process and system operations, enabling us to identify potential issues, enhance operational efficiency, and develop more effective processes. The Vishnu Legacy Bank project utilizes flowcharts to illustrate the flow of data within the three core services: Login, Register, and Expense Predictor. These flowcharts provide a visual representation of how data is processed and transferred within the system, aiding in a more comprehensive understanding of the software's operational flow.



3.3 DATASET DESCRIPTION

The Expense Predictor service utilizes a dataset that includes variables such as monthly income, age, gender, and [add other relevant features]. This dataset is essential for providing users with data-driven insights into their spending patterns and assisting them in making informed financial decisions. It serves as a valuable resource for improving financial management and making recommendations to the users, enhancing their overall banking experience.

the dataset

Transactio	Age	Items	Monthly Ir	Transactio	Record	Gender	City Tier	Total Spend		
TXN001	42	10	7313	627.6681	5	Female	Tier 1	4198.385		
TXN002	24	8	17747	126.9046	3	Female	Tier 2	4134.977		
TXN003	47	11	22845	873.4697	2	Male	Tier 2	5166.614		
TXN004	50	11	18552	380.2194	7	Female	Tier 1	7784.448		
TXN005	60	2	14439	403.3742	2	Female	Tier 2	3254.16		
TXN006	49	6	6282	48.97427	2	Male	Tier 2	2375.036		
TXN007	21	14	7086	961.2038	8	Male	Tier 1	7494.475		
TXN008	58	9	8881	962.2537	10	Male	Tier 3	10782.94		
TXN009	20	6	5635	858.3281	5	Male	Tier 1	3854.277		
TXN010	48	12	20861	43.03674	4	Female	Tier 2	5346.14		
TXN011	37	8	2556	947.8464	1	Male	Tier 3	1712.074		
TXN012	49	8	27947	89.64335	6	Male	Tier 2	8047.046		
TXN013	48	12	23445	773.636	9	Male	Tier 3	11334.58		
TXN014	34	10	14297	245.8942	2	Male	Tier 2	3791.364		
TXN015	39	1	3405	120.9168	1	Male	Tier 1	1674.999		
TXN016	29	13	28714	546.5523	6	Male	Tier 2	8446.608		
TXN017	37	13	14069	131.2064	8	Female	Tier 1	8338.439		
TXN018	58	14	28428	102.0524	9	Female	Tier 3	11933.06		
TXN019	58	15	14982	69.32488	10	Male	Tier 2	11898.85		
TXN020	58	13	27864	350.7857	8	Female	Tier 1	10502.4		
TXN021	30	5	16960	588.3179	10	Male	Tier 2	11640.43		
TXN022	51	14	3041	361.6467	8	Male	Tier 3	6929.271		
TXN023	34	13	5037	579.5193	6	Female	Tier 2	4659.601		
TXN024	44	4	27403	755.191	2	Female	Tier 3	5186.995		
TXN025	30	11	23570	484.6853	6	Male	Tier 1	7573.79		
TXN026	29	5	8954	87.43645	1	Female	Tier 3	2113.269		
TXN027	28	15	12015	997.7114	8	Male	Tier 1	8316.444		

CHAPTER 4: DATABASE DESIGN

4.1 INTRODUCTION

Database design, in the context of the Vishnu Legacy Bank, is a set of essential tasks and processes aimed at improving the creation, development, implementation, and maintenance of the bank's data management system. A well-designed database reduces maintenance costs, enhances data consistency, and optimizes disk storage space usage. Therefore, creating a database for the Vishnu Legacy Bank requires careful consideration and planning.

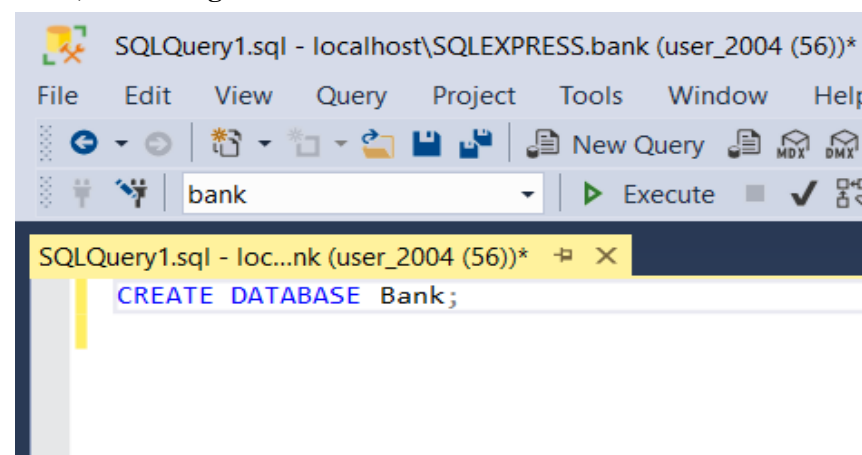
The primary objectives of database design for the Vishnu Legacy Bank are to establish both physical and logical design models for the proposed database system. The logical model primarily focuses on data requirements, ensuring that the stored data remains independent of specific physical conditions. On the other hand, the physical database design model translates the logical design into a database system that is optimized for hardware resources and software systems like the Database Management System (DBMS).

Database design is critically important for several reasons:

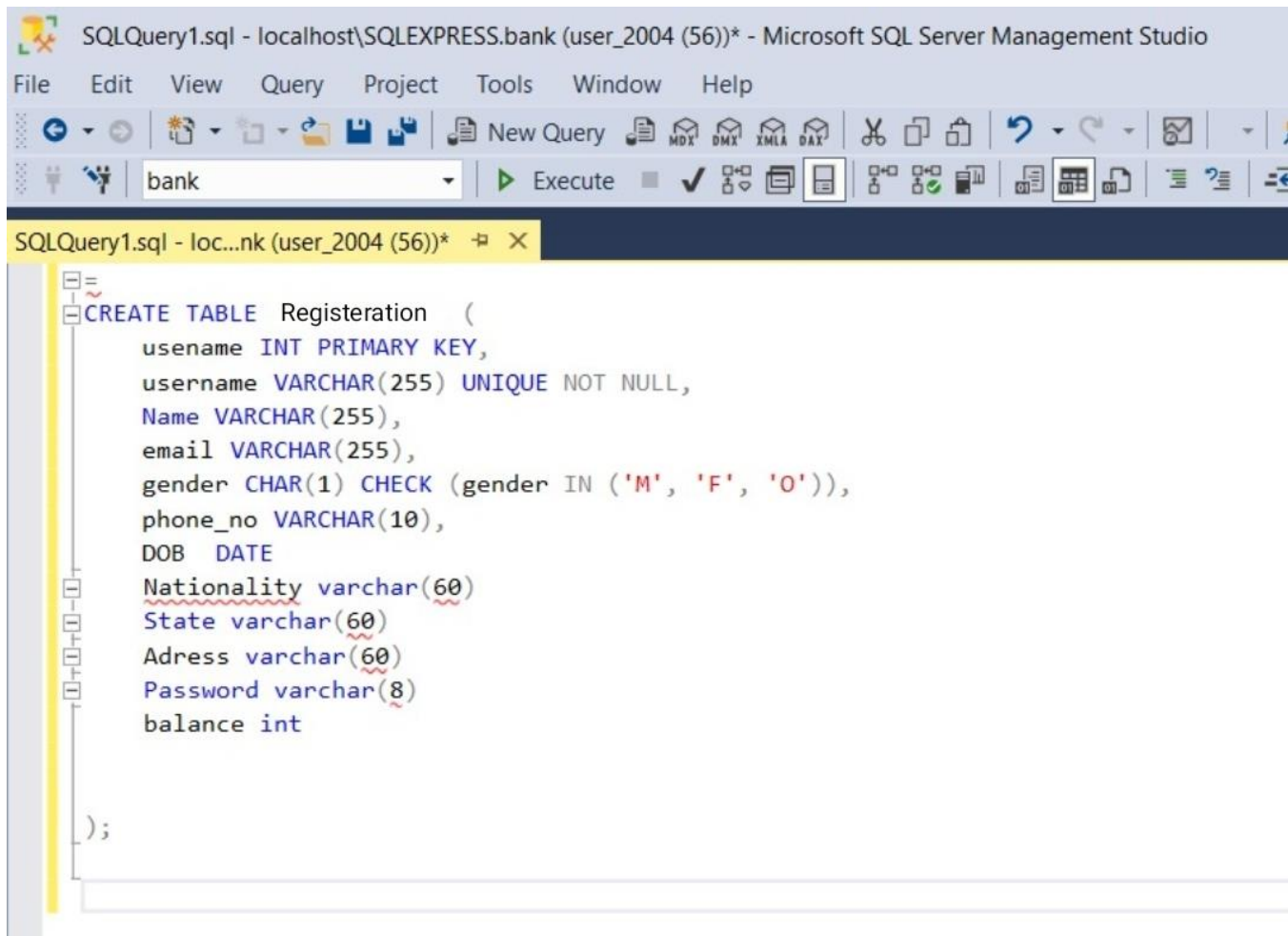
- Database designs serve as blueprints for how data will be stored within the system, directly impacting the overall performance of the bank's applications.
- The design principles laid out for the database provide a clear understanding of how the bank's applications behave and how user requests are processed.
- Proper database design ensures that all user requirements are met effectively, enhancing the quality of service provided by the Vishnu Legacy Bank.
- Efficient database design reduces the processing time of applications, resulting in quicker and more responsive services for bank customers.

Here's the database design query :

1) Creating database



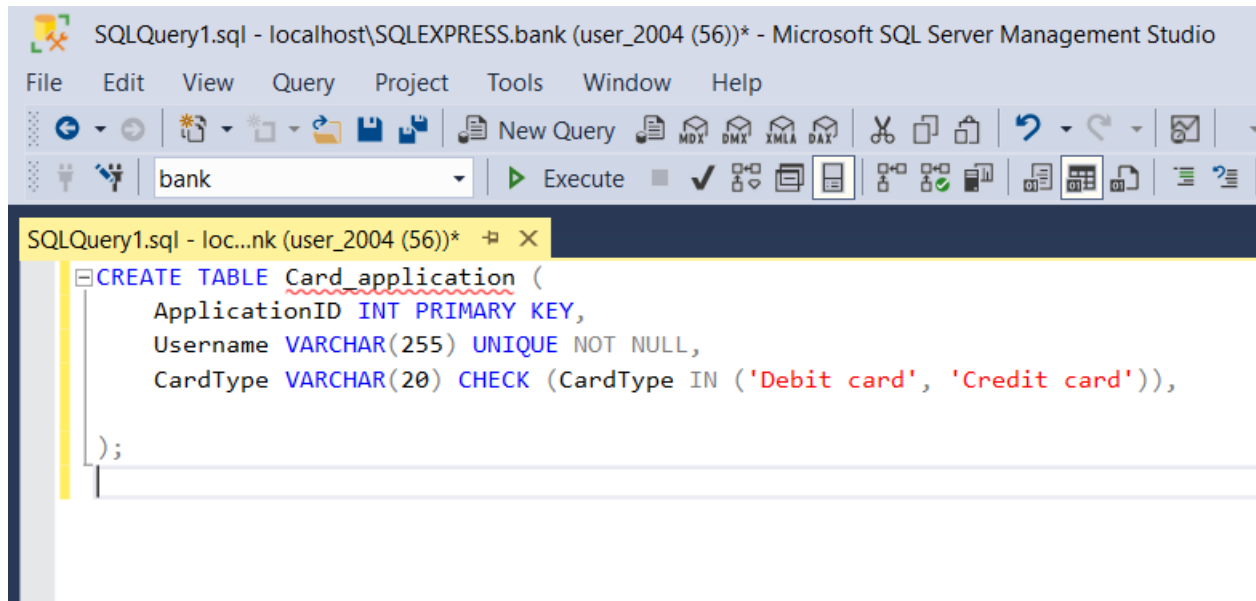
2)Creating Registration table consist of user_information :



The screenshot displays the Microsoft SQL Server Management Studio interface. The title bar indicates the file is 'SQLQuery1.sql' located at 'localhost\SQLEXPRESS.bank (user_2004 (56))*'. The menu bar includes File, Edit, View, Query, Project, Tools, Window, and Help. The toolbar contains various icons for file operations, query execution, and data management. The 'Query' tab is active, showing a SQL query to create a table named 'Registration'. The query is as follows:

```
CREATE TABLE Registration (  
    username INT PRIMARY KEY,  
    username VARCHAR(255) UNIQUE NOT NULL,  
    Name VARCHAR(255),  
    email VARCHAR(255),  
    gender CHAR(1) CHECK (gender IN ('M', 'F', 'O')),  
    phone_no VARCHAR(10),  
    DOB DATE  
    Nationality varchar(60)  
    State varchar(60)  
    Adress varchar(60)  
    Password varchar(8)  
    balance int  
);
```


3)Creating Card_Application Table Consist Of Card Information



CHAPTER 5: TESTING

5.1 INTRODUCTION

Testing is an essential process in the development of the Vishnu Legacy Bank system. It is a systematic way of identifying errors or issues to ensure the system functions correctly and meets user requirements. Testing was conducted at various stages of development, using a combination of artificial and live data to verify the functionality and performance of the system.

5.2 METHODS EMPLOYED FOR TESTING

The following testing methods were employed during the development of the Vishnu Legacy Bank system:

5.2.1 Unit Testing

Unit testing involved the testing of individual modules to ensure they met their functional specifications. It was performed to identify and rectify syntax and logical errors. Test data was prepared during the development of technical specifications, and the coding process was carried out after verifying the output against this test data.

5.2.2 Integration Testing

Integration testing was conducted to validate the entire system when all individual modules were integrated. This phase aimed to confirm that the system operated according to the specified requirements, and any interface errors were corrected. Test data was used to evaluate the system's ability to detect errors.

5.2.3 Functional Testing

Functional testing was performed for each module or sub-module to validate whether the functionality of the system aligned with the original user requirements. Test schedules were created, including test data preparation, test case writing, conformance testing, and bug listing for non-conformities.

5.2.4 System Testing

System testing was carried out once all the modules were fully integrated. This phase assessed how different modules interacted with each other and whether the system delivered the expected functionality.

5.2.5 Acceptance Testing

Acceptance testing verified whether the entire system worked as intended, meeting the acceptance criteria set by stakeholders.

5.2.6 Performance Testing

Performance testing evaluated how the software performed under various workloads, including load testing to assess real-life load conditions.

5.2.7 Regression Testing

Regression testing was conducted to ensure that new features did not break or degrade existing functionality. Sanity testing was

used for surface-level verification when a full regression test was not feasible.

5.2.8 Stress Testing

Stress testing measured the system's resilience by determining how much strain it could handle before failure. This type of testing focused on non-functional aspects.

5.2.9 Usability Testing

Usability testing evaluated the ease with which customers use desktop application with ease

5.3 TEST CASES (TC)

Test cases were created to systematically assess the functionality of the Vishnu Legacy Bank system. These test cases covered various scenarios and were executed independently for some components. Here are a few sample test cases:

NO.	TEST CASE TITLE	DESCRIPTION	EXPECTED OUTCOME	RESULT
1	Successful User Verification	Login with correct username and Password	Successful login, redirected to home page	Passed
2	Unsuccessful User Verification	Login with incorrect password	Login failure,forget password no redirection	Passed
3	User Registration	Register a new user login ID	Successful registration, redirected to login	Passed
4	Forget Password Feature	Test the "Forgot Password" feature	Receive OTP, reset password successfully	Passed
5	Blank Fields on Login	Submit with blank email and password fields	Unsuccessful login, no redirection	Negative
6	register to Create New user	user creates new registration	Successful creation of new user	Positive
7	check for expense predictor	Age ,income ,gender defines expense	Successful checked correct prediction	Positive
8	User services for client	Service updation as user intractsservices	Successful modification of user	Negative

6.1 ROLE AND RESPONSIBILITY

As the sole developer of the Vishnu Legacy Bank application, I hold a multifaceted role with various responsibilities:

1. Project Manager

Define project goals and scope. Plan project timelines and milestones. Oversee all aspects of development and testing.

2. System Architect

Design the system's architecture and structure. Make technical decisions. Ensure the application's scalability and maintainability.

3. UI/UX Designer

Create user interface designs that are user friendly and visually appealing. Ensure the responsiveness and accessibility of the application. Implement the designed interface into the application.

4. Frontend Developer

Develop the user interfaces based on the UI/UX designs. Ensure the website's responsiveness across different devices. Implement the frontend components of the application.

5. Backend Developer

Develop the core logic and functionality of the Vishnu Legacy Bank system. Implement database management and server side processing. Ensure data security and authentication mechanisms.

6. Quality Assurance (QA) Tester

Plan and execute testing activities to identify and rectify defects. Create and maintain test cases and test data. Ensure the application's reliability and performance through thorough testing.

7. Database Administrator (DBA)

Manage the database structure, schema design, and optimization. Implement data backup and recovery strategies. Ensure data integrity and security.

8. Business Analyst

Define and analyze user requirements to align project objectives. Create functional and technical documentation. Act as the bridge between stakeholders and the development process.

9. Documentation Specialist

Create and maintain project documentation, including user manuals and technical documentation. Organize and manage project related documents.

10. Problem Solving

Collaboratively address and resolve any issues or challenges that arise during development. Identify and implement solutions to keep the project on track.

7.1 CONCLUSION

The Vishnu Legacy Bank project represents a significant leap in modernizing and streamlining banking operations. It harnesses the power of Information Technology (IT) to enhance operational efficiency, reduce costs, and provide an improved banking experience for customers. While the implementation of this system involves initial investments, it promises long-term cost savings and substantial benefits for both the bank and its clientele.

The implementation of an ERP system like the Vishnu Legacy Bank marks a major milestone in the digital transformation of banking services. By integrating various banking functions into a unified and efficient system, it paves the way for seamless communication and collaboration among different departments, resulting in enhanced productivity and customer satisfaction.

ERP systems are a driving force in the ever-evolving IT landscape, and their role is pivotal. The success of ERP systems is evident in their multi-billion-dollar industry. The Vishnu Legacy Bank project, by adopting ERP principles, joins this industry to deliver efficient and integrated banking solutions.

By unifying all organizational information, the Vishnu Legacy Bank project empowers the bank with a comprehensive information infrastructure. It benefits the institution in multiple ways, including improved performance, enhanced access to accurate and timely data, streamlined workflows, reduced reliance on paper, knowledge sharing, tight control, and process automation by seamlessly coordinating and integrating information across all departments.

This project's success lies in its ability to create an interactive web and, in the future, a mobile application. The Data Model and Process Model demonstrate the construction of a robust database with various tables, showcasing efficient data access and processing capabilities.

7.2 FUTURE ENHANCEMENTS

The Vishnu Legacy Bank project is poised for continuous growth and improvement. As part of our commitment to serving our customers better and staying at the forefront of technology, we envision several future enhancements:

1. **Web and Mobile Application** : We aim to evolve into a full-fledged web and mobile application to provide our customers with the flexibility to access their banking services seamlessly through a web browser or mobile app.
2. **Enhanced Security** : In the future, we plan to implement advanced security features, including biometric authentication such as face recognition, to fortify the safety of our customers' financial data.
3. **Expanded Services** : We intend to diversify our range of banking services to cater to the evolving needs of our customers. This may include investment tools, loan management, and additional financial products.
4. **Improved User Experience** : Continuous user interface and user experience (UI/UX) enhancements to ensure that interacting with our banking services remains intuitive and efficient.
5. **Customer-Centric Features** : We will focus on introducing features that put the customer at the center of our services, making banking with Vishnu Legacy Bank a seamless and satisfying experience.
6. **Integration with Emerging Technologies** : As new technologies emerge, we will explore their integration to offer cutting-edge banking solutions, such as blockchain-based transactions and digital currency support.

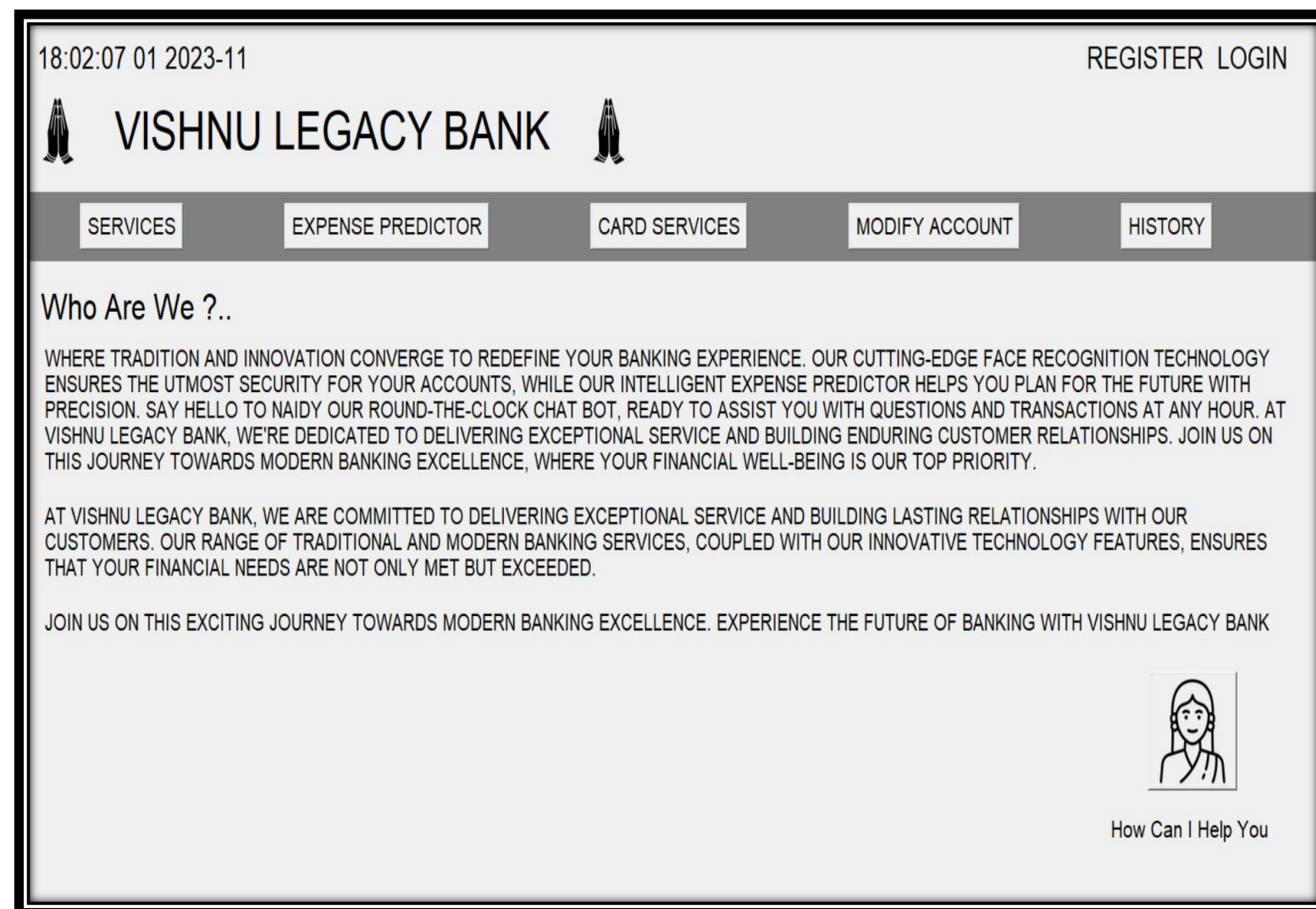
The Vishnu Legacy Bank project is committed to staying at the forefront of technological advancements in the banking sector. We look forward to a future where our customers enjoy even more convenience, security, and efficiency in their banking experiences.

CHAPTER 8: APPENDICES

(CODING AND SCREENSHOTS)

8.1 SCREENSHOTS OF THE WEBSITE:

1)HOME PAGE (Main)



2)REGISTRATION PAGE:

CREATE YOUR ACCOUNT

NAME

DATE OF BIRTH

PHONE NO.

EMAIL ID

GENDER

NATION

STATE

ADDRESS

Username

PASSWORD

LOGIN

REGISTER

3)Login page :

LOGIN

USERNAME

PASSWORD

LOGIN

FORGET PASSWORD

4) Chatbot :



Namaste How Can I Help You

You: hello
 ChatBot: Namaste!
 You: how can i change password
 ChatBot: I'm sorry, I don't understand that.
 You: what is the eligibility for insurance
 ChatBot: I'm sorry, I don't understand that.
 You: what is the eligibility for insurance
 ChatBot: Insurance eligibility depends on various factors such as your age, health, and the type of insurance you are interested in. It's best to check the insurance page
 You: how to reset password
 ChatBot: To reset your password, visit our website and click on the 'Forgot Password' link on the login page. Follow the instructions to reset your password.
 You: how to check profile
 ChatBot: You can check your profile by logging into your online account or register yourself first to check profile if forgotten password choose forget password option then reset it

Send

5) Expense predictor:

Enter Your Details to Predict Total Spend:

AGE

18

MONTHLY INCOME

12000

GENDER

☒ Male ☐ Female

Predict

TOTAL SPEND

5315

6)Services:

SERVICES

DEPOSIT

WITHDRAW

TRANSFER

CALCULATOR

ENQUIRY

6.1)Deposit:

DEPOSIT

Username:

Password:

Amount:

Deposit

Deposited Successfully.....kk123,Amount 2000

6.2)Withdraw:

WITHDRAW

Username:

Password:

Amount:

Withdraw

Withdrawn 1000 successfully for kk123

6.3)Transfer :

TRANSFER WITHIN BANK

Sender Username:

kk123

Sender Password:

Receiver Username:

sk123

Amount:

4000

Transfer

Transferred 4000 to sk123 successfully

6.4)Calculator:

15+6

7	8	9	/
4	5	6	*
1	2	3	-
0	.	+	=
	C		=

6.6)Inquiry:

INQUIRE DATA.....

Username:

Retrieve

Name:

Phone:

Email:

Gender:

Nationality:

State:

Address:

Balance:

Card:

7)Card services :

Apply for Card Services

Username:

Card Type:

Submit

8)Modify Account :

MODIFY YOUR ACCOUNT

Username:

Password:

Modify

New Value:

Modify

Account information updated successfully!

9)Forget password:

FIND YOUR USERNAME AND PASSWORD

Email:

Contact:

Retrieve

Username and password found.

Username:

Password:

8.2FEW CODE SNIPS FROM THE WEBSITE

1)Mainfile.py

```

from tkinter import Tk, Label, Button, Frame, Text, Entry, Toplevel, END, messagebox
from datetime import datetime
from tkinter import *
from PIL import Image, ImageTk
import random
import allfunctions as al
import pyodbc

win = Tk()
win.title("home")
entry = None
chatbox = None
original_image = Image.open("chatbot.png")
resized_image = original_image.resize((100, 100), Image.ANTIALIAS)
chatbot_image = ImageTk.PhotoImage(resized_image)

rules = {
    "hello": ["Namaste!", "Hello!", "Hey!"],
    "namestey":["Namestey","Hello","hey"],
    "ram ram":["JAI SHREE RAM "],
    "how are you": ["I'm doing well, thanks!", "I'm just a bot, so I don't have feelings, but I'm here to help!"],
    "what's your name": ["I'm a chatbot.", "I don't have a name, but you can call me ChatBot."],
    "bye": ["Goodbye!", "See you later!", "Have a great day!"],
    "how old are you": ["I don't have an age. I'm just a computer program.", "I exist in the digital realm, so I don't age."],
    "what is your purpose": ["My purpose is to assist you with information and answer your questions to the best of my knowledge."],
    "what is the eligibility for insurance": ["Insurance eligibility depends on various factors such as your age, health, and the type of insurance you are interested in. It's best to check the insurance page"],
    "how can i apply for card services": ["You can apply for card services by login in and choosing card services and then filling out an online application form on our website. Make sure to provide all the necessary documents."],
    "how to convert normal account to fixed account": ["To convert a normal account to a fixed account login or register yourself then check for other services and select for conversion all the best!!."],
    "how many available schemes": ["We offer a variety of schemes, including savings accounts, fixed deposits, and insurance plans, and other schemes for that Please visit our website"],
    "how to check profile": ["You can check your profile by logging into your online account or register yourself first to check profile if forgotten password choose forget password option then reset it "],
    "how to reset password": ["To reset your password, visit our website and click on the 'Forgot Password' link on the login page. Follow the instructions to reset your password."],
    "who is jesika": ["jesika is a good girl"]
}

def connect():
    try:

```

```
server = 'localhost\SQLEXPRESS'
database = 'bank'
```

```
connection_string = f'DRIVER={{SQL Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;'
```

```
connection = pyodbc.connect(connection_string)
```

```
print(f"Connected to SQL Server: {server}, Database: {database}")
```

```
return connection
```

```
except pyodbc.Error as e:
```

```
    print(f"Error connecting to the database: {e}")
```

```
    return None
```

```
def insert_data(name, phone, email, gender, nationality, state, address, password, username, dob,balance):
```

```
    connection=connect()
```

```
    if connection:
```

```
        try:
```

```
            cursor = connection.cursor()
```

```
            sql_insert = """
```

```
            INSERT INTO registration(Name, Phone, Email, Gender, Nationality, State, Address, Username, Password,
DOB,balance)
```

```
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

```
            """
```

```
            values = (name, phone, email, gender, nationality, state, address, username, password, dob,balance)
```

```
            cursor.execute(sql_insert, values)
```

```
            connection.commit()
```

```
            cursor.close()
```

```
        print("Data inserted successfully")
```

```
        messagebox.showinfo("welcome ", "vishnu legacy bank : " + username)
```

```
    except pyodbc.Error as e:
```

```
        messagebox.showinfo("DATABASE SERVER ERROR ",f"Error inserting data: {e}")
```

```
    finally:
```

```
        connection.close()
```

```
def register():
```

```
    register_window = Tk()
```

```
    register_window.title("REGISTER")
```

```
    name = StringVar()
```

```

DOB = StringVar()
phone_no = StringVar()
emailid = StringVar()
gender = StringVar()
nationality = StringVar()
state = StringVar()
address = StringVar()
password = StringVar()
username = StringVar()

fields = [
    ("NAME"), ("DATE OF BIRTH"), ("PHONE NO."),
    ("EMAIL ID"), ("GENDER"), ("NATION"),
    ("STATE"), ("ADDRESS"), ("Username"), ("PASSWORD")
]

for i, (field_name) in enumerate(fields, start=1):
    label = Label(register_window, text=field_name,
                  bg="SystemButtonFace", font=("Arial", 16))
    label.grid(row=i, padx=400, sticky="w", pady=20)

name_entry = Entry(register_window, textvariable=name, width=30)
name_entry.grid(row=1, padx=650, sticky="w")
phone_entry = Entry(register_window, textvariable=phone_no, width=30)
phone_entry.grid(row=3, padx=650, sticky="w")
email_entry = Entry(register_window, textvariable=emailid, width=30)
email_entry.grid(row=4, padx=650, sticky="w")
gender_entry = Entry(register_window, textvariable=gender, width=30)
gender_entry.grid(row=5, padx=650, sticky="w")
nation_entry = Entry(register_window, textvariable=nationality, width=30)
nation_entry.grid(row=6, padx=650, sticky="w")
state_entry = Entry(register_window, textvariable=state, width=30)
state_entry.grid(row=7, padx=650, sticky="w")
add_entry = Entry(register_window, textvariable=address, width=30)
add_entry.grid(row=8, padx=650, sticky="w")
pass_entry = Entry(register_window, textvariable=password, width=30)
pass_entry.grid(row=10, padx=650, sticky="w")
user_entry=Entry(register_window, textvariable=username, width=30)
user_entry.grid(row=9, padx=650, sticky="w")

heading_lb = Label(register_window, text="CREATE YOUR ACCOUNT",
                  bg="SystemButtonFace", font=("Arial", 25))
heading_lb.grid(row=0, padx=400, sticky="w")

```

```
dob_label = Label(register_window, text="DATE OF BIRTH",bg="SystemButtonFace", font=("Arial", 16))
dob_label.grid(row=2, padx=400, sticky="w", pady=20)
```

```
dob_entry = Entry(register_window, textvariable=DOB, width=30)
dob_entry.grid(row=2, padx=650, sticky="w")
```

```
balance=0
```

```
register_btn = Button(register_window, text="REGISTER", bg="grey", fg="white", font=("bold", 20), width=10,
command=lambda:insert_data(name_entry.get(),phone_entry.get(),email_entry.get(),gender_entry.get(),nation_entry.get(),state_
entry.get(),add_entry.get(),pass_entry.get(),user_entry.get(),dob_entry.get(),balance))
register_btn.grid(row=10, sticky="e")
```

```
login_btn = Button(register_window, text="LOGIN", bg="grey",fg="white", font=("bold", 20), command=al.login_user)
login_btn.grid(row=10, padx=55, sticky="w")
```

```
register_window.mainloop()
```

```
def modify_account_window(win):
```

```
def connect_to_database():
```

```
try:
```

```
connection = pyodbc.connect(
    "Driver={SQL Server};"
    "Server=localhost\\SQLEXPRESS;"
    "Database=bank;"
    "Trusted_Connection=yes;"
)
```

```
return connection
```

```
except pyodbc.Error as e:
```

```
print(f"Database connection error: {e}")
```

```
return None
```

```
def update_user_info(username, field_var, new_value,password):
```

```
connection = connect_to_database()
```

```
if not connection:
```

```
return
```

```
cursor = connection.cursor()
```

```
try:
```

```
cursor.execute("SELECT username FROM registration WHERE username = ? AND password=?", (username,password))
existing_username = cursor.fetchone()
```

```
if existing_username:
```

31

```
cursor.execute(f"UPDATE registration SET {field_var} = ? WHERE username = ?", (new_value, username))
```



```

        connection.commit()
        status_label.config(text="Account information updated successfully!", fg="green")
    else:
        status_label.config(text="Username not found in the database.", fg="red")
except pyodbc.Error as e:
    print(f"Error updating registration info: {e}")
    status_label.config(text="Error updating account information.", fg="red")
finally:
    connection.close()

def modify_account():
    username = username_var.get()
    field = field_var.get()
    new_value = entry_var.get()
    password=password_var.get()

    if not username or not field or not new_value:
        status_label.config(text="Please fill in all fields.", fg="red")
    return

    update_user_info(username, field, new_value,password)

modify = Toplevel(win)
modify.title("Modify Account Information")
head_lb = Label(modify, text="MODIFY YOUR ACCOUNT", font=("Helvetica", 25), bg="SystemButtonFace")
head_lb.grid(row=0, sticky="w", padx=40, pady=10)
frame = Frame(modify, padx=20, pady=20)
frame.grid()
username_label = Label(frame, text="Username:", font=("Helvetica", 16), bg="SystemButtonFace")
username_label.grid(row=0, sticky="w", padx=60)
password_label = Label(frame, text="Password:", font=("Helvetica", 16), bg="SystemButtonFace")
password_label.grid(row=1, sticky="w", padx=60)

global username_var
global password_var
username_var = StringVar()
username_entry = Entry(frame, textvariable=username_var)
username_entry.grid(row=0, sticky="w", padx=250)
password_var = StringVar()
password_entry = Entry(frame, textvariable=password_var)
password_entry.grid(row=1, sticky="w", padx=250)

field_label = Label(frame, text="Modify", font=("bold", 16), bg="SystemButtonFace")
field_label.grid(row=2, sticky="w", padx=60)

```

```

fields = ["name", "phone", "email", "gender", "nationality", "state", "address", "username"]
global field_var
field_var = StringVar()
field_dropdown = OptionMenu(frame, field_var, *fields)
field_dropdown.grid(row=2, sticky="w", padx=250)

entry_label = Label(frame, text="New Value:", font=("Helvetica", 16), bg="SystemButtonFace")
entry_label.grid(row=4, sticky="w", padx=60)

global entry_var
entry_var = StringVar()
entry = Entry(frame, textvariable=entry_var)
entry.grid(row=4, sticky="w", padx=250)
modify_button = Button(frame, text="Modify", command=modify_account, font=("Helvetica", 16, "bold"), bg="grey",
fg="white")
modify_button.grid(row=6, sticky="w", padx=60)

global status_label
status_label = Label(frame, text="", fg="green", font=("Helvetica", 16), bg="SystemButtonFace")
status_label.grid(row=7, columnspan=2)

modify.mainloop()

def navigate(label):
    if label == "SERVICES":
        al.services()
        print("Services function called")
    elif label == "EXPENSE PREDICTOR":
        al.expense_predictor()
        print("Expense Predictor function called")
    elif label == "CARD SERVICES":
        al.card_application_page()
        print("card application window opened")
    elif label == "MODIFY ACCOUNT":
        modify_account_window(win)
        print("Modify Account function called")
    elif label == "HISTORY":
        print("History function called")

def generate_response(user_input):
    for key in rules:
        if key in user_input:
            return random.choice(rules[key])
    return "I'm sorry, I don't understand that."

```

```

def send_message():
    global entry, chatbox
    user_input = entry.get().lower()
    if user_input == "exit":
        chatbox.config(state=NORMAL)
        chatbox.insert(END, "You: " + user_input + "\n")
        chatbox.insert(END, "ChatBot: Goodbye!\n")
        chatbox.config(state=DISABLED)
        entry.config(state=DISABLED)
    else:
        response = generate_response(user_input)
        chatbox.config(state=NORMAL)
        chatbox.insert(END, "You: " + user_input + "\n")
        chatbox.insert(END, "ChatBot: " + response + "\n")
        chatbox.config(state=DISABLED)
        entry.delete(0, END)

def open_chatbot_window():
    global entry, chatbox
    chatbot_window = Toplevel(win)
    chatbot_window.title("ChatBot")
    chatbot_window.geometry("500x600")

    chatbot_label = Label(chatbot_window, image=chatbot_image)
    chatbot_label.pack()

    greeting_label = Label(
        chatbot_window, text="Namaste How Can I Help You ", fg="black", font=("Arial", 12))
    greeting_label.pack()

    chatbox = Text(chatbot_window, state=DISABLED)
    chatbox.pack()

    entry = Entry(chatbot_window)
    entry.pack()

    send_button = Button(chatbot_window, text="Send", command=send_message)
    send_button.pack()

def update_time():
    current_time = datetime.now().strftime("%H:%M:%S %d %Y-%m")
    time_label.config(text=current_time)
    win.after(1000, update_time)

time_label = Label(win, font=("Helvetica", 20), bg="SystemButtonFace")

```

```
time_label.grid(row=0, pady=6, padx=6, sticky="w")
```

```
update_time()
```

```
register_btn_m = Button(win, text="REGISTER", font=("Helvetica", 20), bg="SystemButtonFace", bd=0, command=register)
```

```
register_btn_m.grid(row=0, pady=6, padx=130, sticky="e")
```

```
login_btn = Button(win, text="LOGIN", font=("Helvetica", 20), bg="SystemButtonFace", bd=0, command=al.login_user)
```

```
login_btn.grid(row=0, pady=6, padx=30, sticky="e")
```

```
left_image = Image.open("namaste.png")
```

```
left_photo = ImageTk.PhotoImage(left_image)
```

```
left_label = Label(win, image=left_photo, bg="SystemButtonFace")
```

```
left_label.grid(row=3, sticky="w", padx=655)
```

```
head_lb = Label(win, text="VISHNU LEGACY BANK", font=(
```

```
    "Helvetica", 35), bg="SystemButtonFace", bd=0)
```

```
head_lb.grid(row=3, sticky="w", padx=100)
```

```
right_image = Image.open("namaste.png")
```

```
right_photo = ImageTk.PhotoImage(right_image)
```

```
right_label = Label(win, image=right_photo, bg="SystemButtonFace")
```

```
right_label.grid(row=3, sticky="w")
```

```
nav_frame = Frame(win, bg="gray")
```

```
nav_frame.grid(row=4, pady=20, sticky="we")
```

```
win.columnconfigure(0, weight=1)
```

```
nav_links = ["Services", "Expense Predictor",
```

```
            "card services", "Modify Account", "history"]
```

```
nav_links = [link.upper() for link in nav_links]
```

```
for i, link in enumerate(nav_links):
```

```
    link_button = Button(nav_frame, text=link, command=lambda l=link: navigate(l), font=("Helvetica", 16))
```

```
    link_button.grid(row=4, column=i, padx=60, pady=10)
```

```
intro_lb = Label(win, text="Who Are We ?..", font=("Helvetica", 24))
```

```
intro_lb.grid(row=5, sticky="w", padx=10)
```

para_text = """"where tradition and innovation converge to redefine your banking experience. Our cutting-edge face recognition technology ensures the utmost security for your accounts, while our intelligent expense predictor helps you plan for the future with precision. Say hello to Naidy our round-the-clock chat bot, ready to assist you with questions and transactions at any hour. At Vishnu Legacy Bank, we're dedicated to delivering exceptional service and building enduring customer relationships. Join us on this journey towards modern banking excellence, where your financial well-being is our top priority.

At Vishnu Legacy Bank, we are committed to delivering exceptional service and building lasting relationships with our customers. Our range of traditional and modern banking services, coupled with our innovative technology features, ensures that your financial needs are not only met but exceeded.

Join us on this exciting journey towards modern banking excellence. Experience the future of banking with Vishnu Legacy Bank

```

para_text = para_text.upper()

para_lines = para_text.split("\n")
for i, line in enumerate(para_lines):
    para_line_lb = Label(win, text=line, font=(
        16), wraplength=1500, justify="left", anchor="w")
    para_line_lb.grid(row=7+i, sticky="nsew", padx=15, pady=10)

original_image = Image.open("chatbot.png")
resized_image = original_image.resize((100, 100), Image.ANTIALIAS)
chatbot_button = Button(win, image=chatbot_image, command=open_chatbot_window)
chatbot_button.grid(row=10, sticky="e", padx=100, pady=20)
chatbot_lb = Label(win, text="How Can I Help You",
    bg="SystemButtonFace", fg="black", font=("Arial", 16))
chatbot_lb.grid(row=11, sticky="e", padx=60)

win.mainloop()

```

2)allfunction.py:

```

from tkinter import Tk, Label, Button, Frame, Text, Entry, Toplevel, END, messagebox
from datetime import datetime
from tkinter import *
from PIL import Image, ImageTk
import random
import pyodbc
from tkinter import ttk
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import pandas as pd

def connect_to_database(server='localhost\SQLEXPRESS', database='bank'):
    try:
        # Define the connection string
        connection_string = f'DRIVER={{SQL Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;'
        # Connect to the database
        connection = pyodbc.connect(connection_string)

        print(f"Connected to SQL Server: {server}, Database: {database}")

        return connection

    except pyodbc.Error as e:
        print(f"Error connecting to the database: {e}")

```

return None

def insert_data(name, phone, email, gender, nationality, state, address, password, username, dob,balance):

 connection=connect_to_database()

 if connection:

 try:

 cursor = connection.cursor()

 sql_insert = """

 INSERT INTO registration(Name, Phone, Email, Gender, Nationality, State, Address, Username, Password, DOB,balance)

 VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)

 """

 values = (name, phone, email, gender, nationality, state, address, username, password, dob,balance)

 cursor.execute(sql_insert, values)

 connection.commit()

 cursor.close()

 print("Data inserted successfully")

 messagebox.showinfo("welcome ", "vishnu legacy bank : " + username)

 except pyodbc.Error as e:

 print(f"Error inserting data: {e}")

 finally:

 connection.close()

def fetch_user_data(username):

 try:

 connection=connect_to_database()

 cursor = connection.cursor()

 cursor.execute("SELECT Name, DOB, Phone, Email FROM registration WHERE username = ?", (username))

 user_data = cursor.fetchone()

 if user_data:

 user_info = {

 'Name': user_data[0],

 'DOB': user_data[1],

 'Phone': user_data[2],

 'Email': user_data[3]

 }

 return user_info

```

except pyodbc.Error as e:
    print(f"pyodbc error: {e}")
finally:
    connection.close()
return {
    'Name': 'User Not Found',
    'DOB': "",
    'Phone': "",
    'Email': "",
}

profile = Tk()
profile.title("PROFILE PAGE ")
def update_time():
    current_time = datetime.now().strftime("%H:%M:%S %d %Y-%m")
    time_label.config(text=current_time)
    profile.after(1000, update_time)
time_label = Label(profile, font=("Helvetica", 20), bg="SystemButtonFace")
time_label.grid(row=0, pady=6, padx=6, sticky="w")
update_time()

head_lb = Label(profile, text="VISHNU LEGACY BANK", font=(
    "Helvetica", 35), bg="SystemButtonFace", bd=0)
head_lb.grid(row=3, sticky="w", padx=100)

nav_frame = Frame(profile, bg="gray")
nav_frame.grid(row=4, pady=20, sticky="we")
profile.columnconfigure(0, weight=1)
nav_links = ["Services", "Expense Predictor", "card services", "Modify Account", "history"]
nav_links = [link.upper() for link in nav_links]
for i, link in enumerate(nav_links):
    link_button = Button(nav_frame, text=link, command=lambda l=link: navigate(l), font=("Helvetica", 16))
    link_button.grid(row=4, column=i, padx=60, pady=10)

login_btn = Button(profile, text="LOGOFF", font=(
    "Helvetica", 20), bg="SystemButtonFace", bd=0)
login_btn.grid(row=0, pady=6, padx=6, sticky="ne")
user_data = fetch_user_data(username)
user_name_label = Label(profile, text=f"Name: {user_data['Name']}", font=("Arial", 20))
user_name_label.grid(row=5, padx=100, sticky="w", pady=50)

dob_label = Label(profile, text=f"Date of Birth: {user_data['DOB']}", font=("Arial", 20))
dob_label.grid(row=6, padx=100, sticky="w", pady=10)

```

```
contact_no_label = Label(profile, text=f"Contact No.: {user_data['Phone']}", font=("Arial", 20))
contact_no_label.grid(row=7, padx=100, sticky="w", pady=20)
```

```
email_label = Label(profile, text=f"Email: {user_data['Email']}", font=("Arial", 20))
email_label.grid(row=8, padx=100, sticky="w", pady=30)
profile.mainloop()
```

```
def clear_placeholder(entry_widget, placeholder_text):
```

```
    if entry_widget.get() == placeholder_text:
        entry_widget.delete(0, "end")
```

```
def set_entry_placeholder(entry_widget, placeholder_text):
```

```
    entry_widget.insert(0, placeholder_text)
    entry_widget.bind("<FocusIn>", lambda event: clear_placeholder(entry_widget, placeholder_text))
```

```
def register():
```

```
    register_window = Tk()
    register_window.title("REGISTER")
    name = StringVar()
    DOB = StringVar()
    phone_no = StringVar()
    emailid = StringVar()
    gender = StringVar()
    nationality = StringVar()
    state = StringVar()
    address = StringVar()
    password = StringVar()
    username = StringVar()
```

```
    fields = [
        ("NAME", name), ("DATE OF BIRTH", DOB), ("PHONE NO.", phone_no),
        ("EMAIL ID", emailid), ("GENDER", gender), ("NATION", nationality),
        ("STATE", state), ("ADDRESS", address), ("Username", username), ("PASSWORD", password)
    ]
```

```
    for i, (field_name, field_variable) in enumerate(fields, start=1):
```

```
        label = Label(register_window, text=field_name, bg="SystemButtonFace", font=("Arial", 16))
        label.grid(row=i, padx=400, sticky="w", pady=20)
```

```
        entry = Entry(register_window, textvariable=field_variable, width=30)
        entry.grid(row=i, padx=650, sticky="w")
```

```
    heading_lb = Label(register_window, text="CREATE YOUR ACCOUNT", bg="SystemButtonFace", font=("Arial", 25))
    heading_lb.grid(row=0, padx=400, sticky="w")
```

```
    dob_label = Label(register_window, text="DATE OF BIRTH", bg="SystemButtonFace", font=("Arial", 16))
```



```
dob_label.grid(row=2, padx=400, sticky="w", pady=20)
```

```
dob_entry = Entry(register_window, textvariable=DOB, width=30)
```

```
dob_entry.grid(row=2, padx=650, sticky="w")
```

```
set_entry_placeholder(dob_entry, "YYYY-MM-DD")
```

```
balance=0
```

```
register_btn = Button(register_window, text="REGISTER", bg="grey", fg="white", font=("bold", 20), width=10,
command=lambda: insert_data(name.get(), phone_no.get(), emailid.get(), gender.get(), nationality.get(), state.get(),
address.get(), password.get(), username.get(), DOB.get(),balance))
```

```
register_btn.grid(row=10, sticky="e")
```

```
login_btn = Button(register_window, text="LOGIN", bg="grey", fg="white", font=("bold", 20), command=login_user)
```

```
login_btn.grid(row=10, padx=55, sticky="w")
```

```
register_window.mainloop()
```

```
def calculator():
```

```
def button_click(number):
```

```
    current = entry.get()
```

```
    entry.delete(0, END)
```

```
    entry.insert(0, current + str(number))
```

```
def clear():
```

```
    entry.delete(0, END)
```

```
def calculate():
```

```
    try:
```

```
        expression = entry.get()
```

```
        result = eval(expression)
```

```
        entry.delete(0, END)
```

```
        entry.insert(0, result)
```

```
    except Exception as e:
```

```
        entry.delete(0, END)
```

```
        entry.insert(0, "Error")
```

```
cal_win = Tk()
```

```
cal_win.title("Calculator")
```

```
entry = Entry(cal_win, width=20)
```

```
entry.grid(row=0, column=0, columnspan=4)
```

```

buttons = [
    ('7', 1, 0), ('8', 1, 1), ('9', 1, 2), ('/', 1, 3),
    ('4', 2, 0), ('5', 2, 1), ('6', 2, 2), ('*', 2, 3),
    ('1', 3, 0), ('2', 3, 1), ('3', 3, 2), ('-', 3, 3),
    ('0', 4, 0), ('.', 4, 1), ('+', 4, 2), ('=', 4, 3)
]

```

```

for (text, row, col) in buttons:

```

```

    button = Button(cal_win, text=text, padx=20, pady=20, command=lambda t=text: button_click(t))
    button.grid(row=row, column=col)

```

```

clear_button = Button(cal_win, text='C', padx=20, pady=20, command=clear)
clear_button.grid(row=5, column=0, columnspan=3)

```

```

equal_button = Button(cal_win, text='=', padx=20, pady=20, command=calculate)
equal_button.grid(row=5, column=3)

```

```

cal_win.mainloop()

```

```

def deposit_page():

```

```

def authenticate_and_deposit():

```

```

    username = username_entry.get()
    password = password_entry.get()
    amount = amount_entry.get()

```

```

    try:

```

```

        server = 'localhost\SQLEXPRESS'
        database = 'bank'

```

```

        connection_string = f'DRIVER={{SQL
Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;'

```

```

        connection = pyodbc.connect(connection_string)
        cursor = connection.cursor()

```

```

        cursor.execute("SELECT username, password FROM registration WHERE username=? AND password=?",
(username, password))

```

```

        user_data = cursor.fetchone()

```

```

    if user_data:

```

```

        if amount is not None and amount != "" and float(amount)>0:

```

```

            cursor.execute("UPDATE registration SET balance=balance+? WHERE username=?", (amount, username))

```

```

            connection.commit()

```

```

            connection.close()

```

```

            lb=Label(message_label,text=f'Deposited Successfully.....{username},Amount
{amount}',font=('bold',16),fg='green')

```

```

        lb.grid(row=5)
    else:
        connection.close()
        messagebox.showinfo("AMOUNT ERROR.....",'Amount Cannot be NULL')
    else:
        connection.close()
        messagebox.showinfo("INVAILD.....", f"WRONG INPUTS : {username},{password}")
except Exception as e:
    connection.rollback()
    messagebox.showinfo("Database error: " + str(e))

depo_win = Tk()
depo_win.title("Deposit Page")

head_lb=Label(depo_win,text="DEPOSIT",font=("bold",25))
head_lb.grid(row=0,sticky="w",padx=0)
username_label = Label(depo_win, text="Username:",font=("bold",16))
username_label.grid(row=1,padx=16,sticky="w")
username_entry = Entry(depo_win)
username_entry.grid(row=1,sticky="w",padx=200)

password_label = Label(depo_win, text="Password:",font=("bold",16))
password_label.grid(row=2,padx=16,sticky="w")
password_entry = Entry(depo_win, show="*")
password_entry.grid(row=2,sticky="w",padx=200)

amount_label = Label(depo_win, text="Amount:",font=("bold",16))
amount_label.grid(row=3,padx=16,sticky="w")
amount_entry = Entry(depo_win)
amount_entry.grid(row=3,sticky="w",padx=200)

find_button = Button(depo_win, text="Deposit", bg="grey", fg="white",font=("bold",16) ,
command=authenticate_and_deposit)
find_button.grid(row=4,padx=116,sticky="w",pady=10)

message_label = Label(depo_win, text="", fg="green")
message_label.grid()

depo_win.mainloop()

def withdraw_page():
    def authenticate_and_withdraw():
        # Get data from the entry widgets
        username = username_entry.get()
        password = password_entry.get()

```

```
amount = amount_entry.get()
```

```
try:
```

```
    server = 'localhost\SQLEXPRESS'
```

```
    database = 'bank'
```

```
    connection_string = f'DRIVER={{SQL  
Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;'
```

```
    connection = pyodbc.connect(connection_string)
```

```
    cursor = connection.cursor()
```

```
    cursor.execute("SELECT username, password, balance FROM registration WHERE username=? AND password=?",  
(username, password))
```

```
    user_data = cursor.fetchone()
```

```
if user_data:
```

```
    current_balance = user_data[2]
```

```
    if current_balance >= float(amount) and float(amount)>0:
```

```
        new_balance = current_balance - float(amount)
```

```
        cursor.execute("UPDATE registration SET balance=? WHERE username=?", (new_balance, username))
```

```
        connection.commit()
```

```
        connection.close()
```

```
        message_label.config(text=f"Withdrawn {amount} successfully for {username}")
```

```
    else:
```

```
        message_label.config(text="Insufficient balance or wrong inputs ")
```

```
else:
```

```
    message_label.config(text="Invalid username or password")
```

```
except Exception as e:
```

```
    message_label.config(text="Database error: " + str(e))
```

```
withdraw_win = Tk()
```

```
withdraw_win.title("Withdraw Page")
```

```
head_lb = Label(withdraw_win, text="WITHDRAW", font=("bold", 25))
```

```
head_lb.grid(row=0, sticky="w", padx=0)
```

```
username_label = Label(withdraw_win, text="Username:", font=("bold", 16))
```

```
username_label.grid(row=1, padx=16, sticky="w")
```

```
username_entry = Entry(withdraw_win)
```

```
username_entry.grid(row=1, sticky="w", padx=200)
```

```
password_label = Label(withdraw_win, text="Password:", font=("bold", 16))
```

```
password_label.grid(row=2, padx=16, sticky="w")
```

```
password_entry = Entry(withdraw_win, show="*")
```

```
password_entry.grid(row=2, sticky="w", padx=200)
```

```
amount_label = Label(withdraw_win, text="Amount:", font=("bold", 16))
```

```
amount_label.grid(row=3, padx=16, sticky="w")
```

```
amount_entry = Entry(withdraw_win)
```

```
amount_entry.grid(row=3, sticky="w", padx=200)
```

```
withdraw_button = Button(withdraw_win, text="Withdraw", bg="grey", fg="white", font=("bold", 16),
command=authenticate_and_withdraw)
```

```
withdraw_button.grid(row=4, padx=116, sticky="w", pady=10)
```

```
message_label = Label(withdraw_win, text="", fg="green")
```

```
message_label.grid()
```

```
withdraw_win.mainloop()
```

```
def transfer_page():
```

```
def authenticate_and_transfer():
```

```
sender_username = sender_username_entry.get()
```

```
sender_password = sender_password_entry.get()
```

```
receiver_username = receiver_username_entry.get()
```

```
amount = amount_entry.get()
```

```
try:
```

```
server = 'localhost\SQLEXPRESS'
```

```
database = 'bank'
```

```
connection_string = f'DRIVER={{SQL  
Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;'
```

```
connection = pyodbc.connect(connection_string)
```

```
cursor = connection.cursor()
```

```
# Authenticate sender
```

```
cursor.execute("SELECT username, password, balance FROM registration WHERE username=? AND password=?",  
(sender_username, sender_password))
```

```
sender_data = cursor.fetchone()
```

```
# Authenticate receiver
```

```
cursor.execute("SELECT username FROM registration WHERE username=?", (receiver_username,))
```

```
receiver_data = cursor.fetchone()
```

```
if sender_data and receiver_data:
```

```
sender_balance = sender_data[2]
```

```
if sender_balance >= float(amount) and float(amount) > 0:
```

```
# Update sender's balance
```

```

new_sender_balance = sender_balance - float(amount)
cursor.execute("UPDATE registration SET balance=? WHERE username=?", (new_sender_balance,
sender_username))

```

```

# Update receiver's balance

```

```

cursor.execute("UPDATE registration SET balance=balance+? WHERE username=?", (amount,
receiver_username))

```

```

connection.commit()

```

```

connection.close()

```

```

message_label.config(text=f"Transferred {amount} to {receiver_username} successfully")

```

```

else:

```

```

connection.close()

```

```

message_label.config(text="Insufficient balance or WRONG inputs")

```

```

else:

```

```

connection.close()

```

```

message_label.config(text="Invalid sender or receiver username")

```

```

except Exception as e:

```

```

message_label.config(text="Database error: " + str(e))

```

```

transfer_win = Tk()

```

```

transfer_win.title("Transfer Page")

```

```

head_lb = Label(transfer_win, text="TRANSFER WITHIN BANK", font=("bold", 16))

```

```

head_lb.grid(row=0, column=1, padx=20, pady=10)

```

```

sender_username_label = Label(transfer_win, text="Sender Username:", font=("bold", 12))

```

```

sender_username_label.grid(row=1, column=0, padx=20, pady=10, sticky="w")

```

```

sender_username_entry = Entry(transfer_win)

```

```

sender_username_entry.grid(row=1, column=1, padx=20, pady=10)

```

```

sender_password_label = Label(transfer_win, text="Sender Password:", font=("bold", 12))

```

```

sender_password_label.grid(row=2, column=0, padx=20, pady=10, sticky="w")

```

```

sender_password_entry = Entry(transfer_win, show="*")

```

```

sender_password_entry.grid(row=2, column=1, padx=20, pady=10)

```

```

receiver_username_label = Label(transfer_win, text="Receiver Username:", font=("bold", 12))

```

```

receiver_username_label.grid(row=3, column=0, padx=20, pady=10, sticky="w")

```

```

receiver_username_entry = Entry(transfer_win)

```

```

receiver_username_entry.grid(row=3, column=1, padx=20, pady=10)

```

```

amount_label = Label(transfer_win, text="Amount:", font=("bold", 12))

```

```

amount_label.grid(row=4, column=0, padx=20, pady=10, sticky="w")

```

```

amount_entry = Entry(transfer_win)

```

```

amount_entry.grid(row=4, column=1, padx=20, pady=10) 45

```

```

transfer_button = Button(transfer_win, text="Transfer", bg="grey", fg="white", font=("bold", 12),
command=authenticate_and_transfer)
transfer_button.grid(row=5, column=1, padx=20, pady=10)

```

```

message_label = Label(transfer_win, text="", fg="green")
message_label.grid(row=6, column=1, padx=20, pady=10)

```

```

transfer_win.mainloop()

```

```

def card_application_page():

```

```

    def submit_application():

```

```

        username = username_entry.get()

```

```

        card_type = card_type_combo.get()

```

```

        try:

```

```

            server = 'localhost\SQLEXPRESS'

```

```

            database = 'bank'

```

```

            connection_string = f'DRIVER={{SQL
Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;'

```

```

            connection = pyodbc.connect(connection_string)

```

```

            cursor = connection.cursor()

```

```

            cursor.execute("SELECT username FROM card_application WHERE username=?", (username,))

```

```

            existing_application = cursor.fetchone()

```

```

            if existing_application:

```

```

                cursor.execute("UPDATE card_application SET card_type=? WHERE username=?", (card_type, username))

```

```

                connection.commit()

```

```

                connection.close()

```

```

                messagebox.showinfo("Application Status", "Card type updated successfully.")

```

```

            else:

```

```

                cursor.execute("INSERT INTO card_application (username, card_type) VALUES (?, ?)", (username, card_type))

```

```

                connection.commit()

```

```

                connection.close()

```

```

                messagebox.showinfo("Application Status", "Card application submitted successfully.")

```

```

        except Exception as e:

```

```

            messagebox.showerror("Database Error", str(e))

```

```

card_app_win = Tk()

```

```

card_app_win.title("Card Application")

```

```

head_label = Label(card_app_win, text="Apply for Card Services", font=("bold", 16))

```

```

head_label.grid(row=0, column=1, padx=20, pady=10, columnspan=2)

```

```

username_label = Label(card_app_win, text="Username:", font=("bold", 12))
username_label.grid(row=1, column=0, padx=20, pady=10, sticky="w")
username_entry = Entry(card_app_win)
username_entry.grid(row=1, column=1, padx=20, pady=10, columnspan=2)

```

```

card_type_label = Label(card_app_win, text="Card Type:", font=("bold", 12))
card_type_label.grid(row=2, column=0, padx=20, pady=10, sticky="w")
card_type_combo = ttk.Combobox(card_app_win, values=["Debit Card", "Credit Card"])
card_type_combo.grid(row=2, column=1, padx=20, pady=10, columnspan=2)
card_type_combo.set("Debit Card")

```

```

submit_button = Button(card_app_win, text="Submit", bg="grey", fg="white", font=("bold", 12),
command=submit_application)
submit_button.grid(row=3, column=1, padx=20, pady=10)

```

```

card_app_win.mainloop()

```

```

def modify_account_window():

```

```

    def connect_to_database():

```

```

        try:

```

```

            connection = pyodbc.connect(
                "Driver={SQL Server};"
                "Server=localhost\\SQLEXPRESS;"
                "Database=bank;"
                "Trusted_Connection=yes;"
            )

```

```

            return connection

```

```

        except pyodbc.Error as e:

```

```

            print(f"Database connection error: {e}")
            return None

```

```

def update_user_info(username, field_var, new_value,password):

```

```

    connection = connect_to_database()

```

```

    if not connection:

```

```

        return

```

```

    cursor = connection.cursor()

```

```

    try:

```

```

        cursor.execute("SELECT username FROM registration WHERE username = ? AND password=?", (username,password))
        existing_username = cursor.fetchone()

```

```

    if existing_username:

```

```

        cursor.execute(f"UPDATE registration SET {field_var} = ? WHERE username = ?", (new_value, username))
        connection.commit()

```



```

        status_label.config(text="Account information updated successfully!", fg="green")
    else:
        status_label.config(text="Username not found in the database.", fg="red")
except pyodbc.Error as e:
    print(f"Error updating registration info: {e}")
    status_label.config(text="Error updating account information.", fg="red")
finally:
    connection.close()

def modify_account():
    username = username_var.get()
    field = field_var.get()
    new_value = entry_var.get()
    password=password_var.get()

    if not username or not field or not new_value:
        status_label.config(text="Please fill in all fields.", fg="red")
    return

    update_user_info(username, field, new_value,password)

modify = Tk()
modify.title("Modify Account Information")
head_lb = Label(modify, text="MODIFY YOUR ACCOUNT", font=("Helvetica", 25), bg="SystemButtonFace")
head_lb.grid(row=0, sticky="w", padx=40, pady=10)
frame = Frame(modify, padx=20, pady=20)
frame.grid()
username_label = Label(frame, text="Username:", font=("Helvetica", 16), bg="SystemButtonFace")
username_label.grid(row=0, sticky="w", padx=60)
password_label = Label(frame, text="Password:", font=("Helvetica", 16), bg="SystemButtonFace")
password_label.grid(row=1, sticky="w", padx=60)

global username_var
global password_var
username_var = StringVar()
username_entry = Entry(frame, textvariable=username_var)
username_entry.grid(row=0, sticky="w", padx=250)
password_var = StringVar()
password_entry = Entry(frame, textvariable=password_var)
password_entry.grid(row=1, sticky="w", padx=250)

field_label = Label(frame, text="Modify", font=("bold", 16), bg="SystemButtonFace")
field_label.grid(row=2, sticky="w", padx=60)

fields = ["name", "phone", "email", "gender", "nationality", "state", "address", "username"]

```

```

global field_var
field_var = StringVar()
field_dropdown = OptionMenu(frame, field_var, *fields)
field_dropdown.grid(row=2, sticky="w", padx=250)

entry_label = Label(frame, text="New Value:", font=("Helvetica", 16), bg="SystemButtonFace")
entry_label.grid(row=4, sticky="w", padx=60)

global entry_var
entry_var = StringVar()
entry = Entry(frame, textvariable=entry_var)
entry.grid(row=4, sticky="w", padx=250)
modify_button = Button(frame, text="Modify", command=modify_account, font=("Helvetica", 16, "bold"), bg="grey",
fg="white")
modify_button.grid(row=6, sticky="w", padx=60)

global status_label
status_label = Label(frame, text="", fg="green", font=("Helvetica", 16), bg="SystemButtonFace")
status_label.grid(row=7, columnspan=2)

modify.mainloop()

def forgot_password():
    global username,password

def get_password():
    entered_email = email_entry.get()
    entered_contact = contact_entry.get()

    if not entered_email or not entered_contact:
        result_label.config(text="Please enter both email and contact.", fg="red")
        return

    connection = connect_to_database()
    if not connection:
        result_label.config(text="Database connection error.", fg="red")
        return

    cursor = connection.cursor()
    try:
        cursor.execute("SELECT username, password FROM registration WHERE email = ? AND phone = ?", (entered_email,
entered_contact))
        user_data = cursor.fetchone()

        if user_data:

```

```

username.set(user_data[0])
password.set(user_data[1])
result_label.config(text="Username and password found.", fg="green")
username_entry.config(state='normal')
username_entry.delete(0, 'end')
username_entry.insert(0, username.get())
username_entry.config(state='readonly')
password_entry.config(state='normal')
password_entry.delete(0, 'end')
password_entry.insert(0, password.get())
password_entry.config(state='readonly')

```

else:

```
result_label.config(text="No matching user found.", fg="red")
```

except pyodbc.Error as e:

```
print(f"Database error: {e}")
```

```
result_label.config(text="Error retrieving data.", fg="red")
```

finally:

```
connection.close()
```

```
forgot_password_window = Tk()
```

```
forgot_password_window.title("Forgot Password")
```

```
head=Label(forgot_password_window,text="FIND YOUR USERNAME AND
PASSWORD",font=("bold",25),bg="SystemButtonFace").grid(row=0,padx=60,sticky="w")
```

```
email_label = Label(forgot_password_window, text="Email:",font=("bold",16))
```

```
email_label.grid(row=1,padx=60,sticky="w")
```

```
email_entry = Entry(forgot_password_window)
```

```
email_entry.grid(row=1,padx=200,sticky="w")
```

```
contact_label = Label(forgot_password_window, text="Contact:",font=("bold",16))
```

```
contact_label.grid(row=2,padx=60,sticky="w")
```

```
contact_entry = Entry(forgot_password_window)
```

```
contact_entry.grid(row=2,padx=200,sticky="w")
```

```
retrieve_button = Button(forgot_password_window,
text="Retrieve",command=get_password,fg="white",bg="grey",font=("bold",16))
```

```
retrieve_button.grid(row=3,padx=60,sticky="w")
```

```
result_label = Label(forgot_password_window, text="", fg="black",font=("bold",16))
```

```
result_label.grid(row=4,padx=60,sticky="w")
```

```
username = StringVar()
```

```
password = StringVar()
```

```

username_label = Label(forgot_password_window, text="Username:",font=("bold",16))
username_label.grid(row=5,padx=60,sticky="w")

username_entry = Entry(forgot_password_window, textvariable=username, state='readonly')
username_entry.grid(row=5,padx=200,sticky="w")

password_label = Label(forgot_password_window, text="Password:",font=("bold",16))
password_label.grid(row=6,padx=60,sticky="w")

password_entry = Entry(forgot_password_window, textvariable=password, state='readonly')
password_entry.grid(row=6,padx=200,sticky="w")
forgot_password_window.mainloop()

```

```
def inquire_user_details():
```

```

    global name,phone,email,gender,nationality,state,address,balance,card
    name = StringVar()
    phone = StringVar()
    email = StringVar()
    gender = StringVar()
    nationality = StringVar()
    state = StringVar()
    address = StringVar()
    balance = StringVar()
    card=StringVar()

```

```
def get_user_details():
```

```

    entered_username = username_entry.get()
    print(f"Entered username: {entered_username}")

```

```

    if not entered_username:
        messagebox.showerror('MISSING ','ENTER USERNAME')
        return

```

```

    connection = connect_to_database()
    if not connection:
        messagebox.showerror('database error',"DATABASE ERROR")
        return

```

```

    cursor = connection.cursor()
    try:
        cursor.execute("SELECT * FROM registration WHERE username = ?", (entered_username,))
        user_data = cursor.fetchone()

```

```

if user_data:
    name.set(user_data[0])
    phone.set(user_data[1])
    email.set(user_data[2])
    gender.set(user_data[3])
    nationality.set(user_data[4])
    state.set(user_data[5])
    address.set(user_data[6])
    balance.set(user_data[10])
    cursor.execute("SELECT card_type FROM card_application WHERE username = ?", (entered_username,))
    card_data = cursor.fetchone()
    if card_data:
        card.set(card_data[0])
    else:
        card.set("NO data found")
        messagebox.showinfo('FOUND', f"USER FOUND: {entered_username}")
messagebox.showinfo('FOUND', f"USER FOUND: {entered_username}")
name_entry.config(state='normal')
name_entry.delete(0, 'end')
name_entry.insert(0, name.get())
name_entry.config(state='readonly')
phone_entry.config(state='normal')
phone_entry.delete(0, 'end')
phone_entry.insert(0, phone.get())
phone_entry.config(state='readonly')
email_entry.config(state='normal')
email_entry.delete(0, 'end')
email_entry.insert(0, email.get())
email_entry.config(state='readonly')
gender_entry.config(state='normal')
gender_entry.delete(0, 'end')
gender_entry.insert(0, gender.get())
gender_entry.config(state='readonly')
nationality_entry.config(state='normal')
nationality_entry.delete(0, 'end')
nationality_entry.insert(0, nationality.get())
nationality_entry.config(state='readonly')
state_entry.config(state='normal')
state_entry.delete(0, 'end')
state_entry.insert(0, state.get())
state_entry.config(state='readonly')
address_entry.config(state='normal')
address_entry.delete(0, 'end')
address_entry.insert(0, address.get())

```

```

address_entry.config(state='readonly')
balance_entry.config(state='normal')
balance_entry.delete(0, 'end')
balance_entry.insert(0, balance.get())
balance_entry.config(state='readonly')
card_entry.config(state='normal')
card_entry.delete(0, 'end')
card_entry.insert(0, card.get())
card_entry.config(state='readonly')
else:
    messagebox.showinfo('NOT FOUND...',f"NO MATCHING INFO FOUND FOR : {entered_username}")
except pyodbc.Error as e:
    print(f"Database error: {e}")
    messagebox.showinfo('DATABASE ERROR',"DATABASE ERROR")
finally:
    connection.close()

inquire_details_window = Tk()
inquire_details_window.title("Inquire User Details")
head_lb=Label(inquire_details_window,text="INQUIRE DATA.....",font=("bold",25))
head_lb.grid(row=0,padx=60,sticky="w")
username_label = Label(inquire_details_window, text="Username:",font=("Ariel",16))
username_label.grid(row=1,padx=60,sticky="w")

username_entry = Entry(inquire_details_window)
username_entry.grid(row=1,padx=200,sticky="w")

retrieve_button = Button(inquire_details_window, text="Retrieve",
command=get_user_details,font=("Ariel",16,'bold'),bg="grey",fg="white")
retrieve_button.grid(row=2,padx=100,sticky="w")

name_label = Label(inquire_details_window, text="Name:",font=("Ariel",16))
name_label.grid(row=3,padx=60,sticky="w")
name_entry = Entry(inquire_details_window, textvariable=name, state='readonly')
name_entry.grid(row=3,padx=200,sticky="w")
phone_label = Label(inquire_details_window, text="Phone:",font=("Ariel",16))
phone_label.grid(row=4,padx=60,sticky="w")
phone_entry = Entry(inquire_details_window, textvariable=phone, state='readonly')
phone_entry.grid(row=4,padx=200,sticky="w")
email_label = Label(inquire_details_window, text="Email:",font=("Ariel",16))
email_label.grid(row=5,padx=60,sticky="w")
email_entry = Entry(inquire_details_window, textvariable=email, state='readonly')
email_entry.grid(row=5,padx=200,sticky="w")
gender_label = Label(inquire_details_window, text="Gender:",font=("Ariel",16))

```

```

gender_label.grid(row=6,padx=60,sticky="w")
gender_entry = Entry(inquire_details_window, textvariable=gender, state='readonly')
gender_entry.grid(row=6,padx=200,sticky="w")
nationality_label = Label(inquire_details_window, text="Nationality:",font=("Ariel",16))
nationality_label.grid(row=7,padx=60,sticky="w")
nationality_entry = Entry(inquire_details_window, textvariable=nationality, state='readonly')
nationality_entry.grid(row=7,padx=200,sticky="w")
state_label = Label(inquire_details_window, text="State:",font=("Ariel",16))
state_label.grid(row=8,padx=60,sticky="w")
state_entry = Entry(inquire_details_window, textvariable=state, state='readonly')
state_entry.grid(row=8,padx=200,sticky="w")
address_label = Label(inquire_details_window, text="Address:",font=("Ariel",16))
address_label.grid(row=9,padx=60,sticky="w")
address_entry = Entry(inquire_details_window, textvariable=address, state='readonly')
address_entry.grid(row=9,padx=200,sticky="w")
balance_label = Label(inquire_details_window, text="Balance:",font=("Ariel",16))
balance_label.grid(row=11,padx=60,sticky="w")
balance_entry = Entry(inquire_details_window, textvariable=balance, state='readonly')
balance_entry.grid(row=11,padx=200,sticky="w")
card_label = Label(inquire_details_window, text="Card:",font=("Ariel",16))
card_label.grid(row=12,padx=60,sticky="w")
card_entry = Entry(inquire_details_window, textvariable=card, state='readonly')
card_entry.grid(row=12,padx=200,sticky="w")

```

```

inquire_details_window.mainloop()

```

```

def expense_predictor():

```

```

    def linear_regression():

```

```

        age_val = int(age_entry.get())

```

```

        income_val = int(income_entry.get())

```

```

        gender_val = gender_var.get()

```

```

        dataset = pd.read_csv('Ecom Expense.csv')

```

```

        dataset = pd.get_dummies(dataset, columns=['Gender'], drop_first=True)

```

```

        X = dataset[['Age ', 'Monthly Income', 'Gender_Male']]

```

```

        y = dataset['Total Spend']

```

```

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

```

```

        model = LinearRegression()

```

```

        model.fit(X_train, y_train)

```

```

        gender_mapping = {'Male': 1, 'Female': 0}

```

```

        gender_display = 'Male' if gender_val == 1 else 'Female'

```

```

predicted_total_spent = model.predict([[age_val, income_val, gender_val]])
spent.set(f"{int(predicted_total_spent[0])}")
spend_entry.config(state='normal')
spend_entry.delete(0, 'end')
spend_entry.insert(0, spent.get())
spend_entry.config(state='readonly')

expense_window = Tk()
expense_window.title("Expense Predictor")
spent = StringVar()
age_var = StringVar()
income_var = StringVar()
gender_var = IntVar()

label = Label(expense_window, text="Enter Your Details to Predict Total Spend:", font=("Arial", 16))
label.grid(row=0, sticky="w", padx=10)

age_lb = Label(expense_window, text="AGE", font=("Arial", 16))
age_lb.grid(row=1, sticky="w", padx=10, pady=6)
age_entry = Entry(expense_window, textvariable=age_var)
age_entry.grid(row=1, sticky="w", padx=250)

income_lb = Label(expense_window, text="MONTHLY INCOME", font=("Arial", 16))
income_lb.grid(row=2, sticky="w", padx=10, pady=6)
income_entry = Entry(expense_window, textvariable=income_var)
income_entry.grid(row=2, sticky="w", padx=250)

gender_lb = Label(expense_window, text="GENDER", font=("Arial", 16))
gender_lb.grid(row=3, sticky="w", padx=10, pady=6)
male_rb = Radiobutton(expense_window, text="Male", variable=gender_var, value=1)
male_rb.grid(row=3, sticky="w", padx=250)
female_rb = Radiobutton(expense_window, text="Female", variable=gender_var, value=0)
female_rb.grid(row=3, sticky="w", padx=300)

predict_button = Button(expense_window, text="Predict", bg="grey", fg="white", font=("Arial", 16),
command=linear_regression)
predict_button.grid(row=4, sticky="w", padx=50, ipadx=2, ipady=2)

spent_lb = Label(expense_window, text="TOTAL SPEND", font=("Arial", 16))
spent_lb.grid(row=5, sticky="w", padx=10)
spend_entry = Entry(expense_window, textvariable=spent)
spend_entry.grid(row=5, sticky="w", padx=250)

expense_window.mainloop()

```



```

def services():
    services=Tk()
    services.title("SERVICES")
    heading=Label(services,text="SERVICES",font=("Arial",25),bg="SystemButtonFace").grid(row=0,padx=10,sticky="w")
    deposit_btn=Button(services,text="DEPOSIT",font=("Arial",16),bd=1,fg="white",bg="grey",command=deposit_page).grid(row=1,padx=10,sticky="w",pady=10)
    withdraw_btn=Button(services,text="WITHDRAW",font=("Arial",16),bd=1,fg="white",bg="grey",command=withdraw_page).grid(row=1,padx=200,sticky="w",pady=10)
    transfer_btn=Button(services,text="TRANSFER",font=("Arial",16),bd=1,fg="white",bg="grey",command=transfer_page).grid(row=1,padx=450,sticky="w",pady=10)
    calculator_btn=Button(services,text="CALCULATOR",font=("Arial",16),bd=1,fg="white",bg="grey",command=calculator).grid(row=2,padx=10,sticky="w")
    enquiry_btn=Button(services,text="ENQUIRY",font=("Arial",16),bd=1,fg="white",bg="grey",command=inquire_user_details).grid(row=2,padx=200,sticky="w",pady=10)
    services.mainloop()

```

```

def navigate(label):
    if label == "SERVICES":
        services()
        print("Services function called")
    elif label == "EXPENSE PREDICTOR":
        expense_predictor()
        print("Expense Predictor function called")
    elif label == "CARD SERVICES":
        card_application_page()
        print("card application window opened")
    elif label == "MODIFY ACCOUNT":
        modify_account_window()
        print("Modify Account function called")
    elif label == "HISTORY":
        print("History function called")

```

```

def profile(username):
    profile = Tk()
    profile.title("PROFILE PAGE ")
    def update_time():
        current_time = datetime.now().strftime("%H:%M:%S %d %Y-%m")
        time_label.config(text=current_time)
        profile.after(1000, update_time)

    time_label = Label(profile, font=("Helvetica", 20), bg="SystemButtonFace")
    time_label.grid(row=0, pady=6, padx=6, sticky="w")
    update_time()

```

```

head_lb = Label(profile, text="VISHNU LEGACY BANK", font=(
    "Helvetica", 35), bg="SystemButtonFace", bd=0)
head_lb.grid(row=3, sticky="w", padx=100)

nav_frame = Frame(profile, bg="gray")
nav_frame.grid(row=4, pady=20, sticky="we")
profile.columnconfigure(0, weight=1)
nav_links = ["Services", "Expense Predictor", "card services", "Modify Account", "history"]
nav_links = [link.upper() for link in nav_links]
for i, link in enumerate(nav_links):
    link_button = Button(nav_frame, text=link, command=lambda l=link: navigate(l), font=("Helvetica", 16))
    link_button.grid(row=4, column=i, padx=60, pady=10)

login_btn = Button(profile, text="LOGOFF", font=(
    "Helvetica", 20), bg="SystemButtonFace", bd=0)
login_btn.grid(row=0, pady=6, padx=6, sticky="ne")
user_data = fetch_user_data(username)
user_name_label = Label(profile, text=f"Name: {user_data['Name']}", font=("Arial", 20))
user_name_label.grid(row=5, padx=100, sticky="w",pady=50)

dob_label = Label(profile, text=f"Date of Birth: {user_data['DOB']}", font=("Arial", 20))
dob_label.grid(row=6, padx=100, sticky="w",pady=10)

contact_no_label = Label(profile, text=f"Contact No.: {user_data['Phone']}", font=("Arial", 20))
contact_no_label.grid(row=7, padx=100, sticky="w",pady=20)

email_label = Label(profile, text=f"Email: {user_data['Email']}", font=("Arial", 20))
email_label.grid(row=8, padx=100, sticky="w",pady=30)
profile.mainloop()

def login_data(username, password):
    connection = connect_to_database()
    if connection:
        try:
            cursor = connection.cursor()
            query = "SELECT * FROM registration WHERE username = ? AND password = ?"
            cursor.execute(query, (username, password))
            registration = cursor.fetchone()

            if registration:
                profile(username)
            else:
                messagebox.showinfo(
                    "Login Failed", "Login failed for username: " + username)
        except pyodbc.Error as e:

```

```

        print(f"Error executing SQL query: {e}")
    finally:
        connection.close()

def login_user():
    login_window = Tk()
    login_window.title("LOGIN")

    def login_button_click():
        entered_username = username_entry.get()
        entered_password = password_entry.get()
        print("Entered Username:", entered_username)
        print("Entered Password:", entered_password)

        if not entered_username or not entered_password:
            print("Please enter both username and password")
            return

        if login_data(entered_username, entered_password):
            print("Login successful")

    heading_lb = Label(login_window, text="LOGIN", bg="SystemButtonFace", font=("Arial", 25))
    heading_lb.grid(row=0, padx=470, sticky="w")

    username_label = Label(login_window, text="USERNAME", bg="SystemButtonFace", font=("Arial", 16))
    username_label.grid(row=1, padx=400, sticky="w", pady=20)
    username_entry = Entry(login_window, width=30)
    username_entry.grid(row=1, padx=650, sticky="w")

    password_label = Label(login_window, text="PASSWORD", bg="SystemButtonFace", font=("Arial", 16))
    password_label.grid(row=2, padx=400, sticky="w", pady=20)
    password_entry = Entry(login_window, show="*", width=30) # Use show="*" to hide password
    password_entry.grid(row=2, padx=650, sticky="w")

    login_btn = Button(login_window, text="LOGIN", bg="grey", fg="white", font=("bold", 20), width=10,
command=login_button_click)
    login_btn.grid(row=4, sticky="w", padx=470, pady=30)

    forget_btn = Button(login_window, text="FORGET PASSWORD", bg="SystemButtonFace", bd=0, font=("Arial",
16,command=forgot_password)
    forget_btn.grid(row=4, padx=800, sticky="w")
    login_window.mainloop()

```

REFERENCES

Python

<https://docs.python.org/3/>

Tkinter (GUI library for Python)

<https://docs.python.org/3/library/tkinter.html>

Sql Server

<https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation>

Machine Learning with Python

<https://docs.microsoft.com/en-us/sql/sql-server/sql-server-technical-documentation>

Windows Application Development

<https://docs.microsoft.com/en-us/windows/desktop/>

OPEN AI CHATGPT:

<https://chat.openai.com/>