**Introduction**

Test Buddy is a web-based platform designed to simplify and enhance the learning experience for developers and testers by providing a unified space to explore, request, and write scripts for various components—all within the same webpage. Unlike traditional approaches that require visiting multiple websites for different components, Test Buddy centralizes resources, saving time and improving efficiency.

This project aims to offer an interactive and seamless experience by integrating component documentation, code examples, and scripting tools in a single interface. By reducing the need for external searches, Test Buddy enhances productivity, streamlines the learning process, and makes component-based development more accessible.

* 1. **ABSTRACT**

Test Buddy is an innovative web-based platform designed to streamline automation testing education and practice by providing a centralized collection of interactive UI components. This platform eliminates the need for students and educators to navigate multiple websites to find diverse testing elements, offering a consolidated, efficient, and interactive environment for learning and experimentation.

* **Simulated Authentication Components:** While Test Buddy itself does not require user login for component access, it provides examples of login forms and authentication-related UI elements. These components simulate the authentication process, allowing students to practice testing login fields, error handling for invalid credentials, and password input security, mirroring the security protocols used in web applications.
* **Simulated Data Input and Processing:** Test Buddy includes UI components that simulate data input and processing, similar to a cash withdrawal scenario. For example, it features input fields for numerical values, buttons to trigger actions, and display areas to show results or error messages. These components allow students to practice testing data validation, input constraints (e.g., maximum values), and the display of dynamic information, which are crucial aspects of testing web applications that handle transactions or data processing.
* **Dynamic Information Display:** Test Buddy provides components that demonstrate how to display information dynamically, similar to a balance inquiry. Examples include displaying text based on user actions, updating content after a button click, or showing success/error messages. These components allow students to practice testing the retrieval and display of information, verifying data accuracy, and ensuring that the UI reflects the correct state of the application..
* **Password/PIN Change Simulation:** Test Buddy includes components that simulate password/PIN change functionality. This includes input fields for old and new passwords, confirmation fields, and error/success message displays. Students can use these components to practice testing password validation rules (e.g., complexity requirements), error handling for incorrect input, and the secure handling of sensitive data input, which are essential for testing user account management features
* **UI Feedback and Error Handling Simulation:** Test Buddy emphasizes the importance of UI feedback and error handling by including components that simulate these features.

This includes:

* + **Alerts and Messages:** Components to display success, error, and informational messages.
  + **Button States:** Examples of how to enable/disable buttons based on user actions or input validation.
  + **Dynamic Content:** Components that change their appearance or behaviour based on user interactions, demonstrating how to test dynamic UI updates.
* **Educational and Practical Benefits:** Test Buddy is structured to educate students on essential automation testing concepts by providing a hands-on environment to interact with various UI components. By offering a centralized platform with diverse elements, Test Buddy helps students:
* **Understand UI Component Interaction:** Practice testing how users interact with different UI elements (e.g., buttons, forms, tables).
* **Apply Automation Testing Techniques:** Learn how to use automation tools and frameworks to interact with and validate UI components.
* **Develop Testing Strategies:** Practice designing test cases to cover different scenarios, including positive and negative testing.
* **Improve Debugging Skills:** Learn how to identify and debug issues in web applications by testing different UI components and their behaviour.
  1. **EXISTING SYSTEM AND LIMITATIONS OF THE EXISTING SYSTEM**

The traditional approach to learning automation testing often involves navigating to various websites to find different UI components for practice. While many websites offer individual components, this approach has several limitations for students and educators:

* **Fragmented Learning:** Students must spend significant time searching for websites that contain the specific UI components they need to practice testing. This can be time-consuming and inefficient, hindering the learning process.
* **Inconsistent Environments:** Different websites may implement UI components in different ways, making it difficult for students to learn consistent testing techniques. Variations in HTML structure, CSS styling, and JavaScript behaviour can lead to confusion and make it harder to generalize testing skills.
* **Lack of Centralized Practice:** There is often no single platform where students can find a comprehensive collection of diverse UI components in a consistent and controlled environment. This makes it challenging for educators to provide a standardized learning experience and for students to practice a wide range of testing scenarios in one place.
  1. **Need for the Proposed System**

The proposed Test Buddy platform addresses the limitations of fragmented learning and inconsistent environments by providing a centralized, interactive environment for automation testing practice. This platform serves as an educational tool and provides insights into testing methodologies and UI component behaviour. The primary motivations for this project include:

* **Enhanced Learning Experience:** Test Buddy offers students practical exposure to key automation testing concepts by providing a variety of interactive UI components in a single location. By working with these components, students can enhance their understanding of UI testing, automation frameworks, and test case design.
* **Focused Practice on Testable Elements:** Test Buddy focuses specifically on providing a range of testable UI components, allowing students to concentrate on learning and applying automation testing techniques without the distractions of complex application logic or navigation.
* **Centralized and Consistent Environment:** Test Buddy provides a consistent and controlled environment for practicing automation testing. This allows educators to provide standardized instruction and enables students to develop consistent testing skills across various UI components.
  1. **Scope of the System**

The scope of this project covers essential UI components for automation testing practice and lays a foundation for future expansion:

* **Core UI Component Provision:** The system provides a collection of essential UI components, such as input fields, buttons, checkboxes, radio buttons, dropdown menus, tables, and forms. This provides a comprehensive set of elements for practicing common automation testing tasks.
* **Interactive Component Examples:** The system includes examples of how UI components can be used to simulate interactive scenarios, such as form submission, data validation, and dynamic content updates. This allows students to practice testing common web application interactions.
* **Clear and Consistent Presentation:** The system presents UI components in a clear and consistent manner, with labels, instructions, and examples. This ensures that students can easily understand how to interact with the components and how to test them effectively.
* **Extensible Component Library:** The system is designed to be extensible, allowing for the addition of more UI components and examples in the future. This scalability ensures that the platform can grow to include a more comprehensive set of elements for automation testing practice.
  1. **Brief Description of Technology Used**

This section outlines the technological components used to build the Test Buddy platform, focusing on the technologies relevant to the user experience and component interaction.

**1.6.1 Operating Systems Used**

Test Buddy is designed to be a web-based platform, making it inherently cross-platform compatible. Users can access the platform through any modern web browser on various operating systems, including Windows, macOS, Linux, Android, and iOS. This ensures accessibility and ease of use for students and educators regardless of their preferred operating system. The development of Test Buddy focuses on web technologies (HTML, CSS, JavaScript) to ensure broad compatibility and ease of deployment.

**1.6.2 Data Interaction Simulation**

While Test Buddy's primary focus is on providing front-end UI components for testing, it demonstrates concepts related to data handling and interaction. For example, it utilizes JavaScript to simulate dynamic content updates and interactions that would typically involve a backend database. This approach offers several advantages for educational purposes:

* **Simulated Data Interaction:** JavaScript is used to simulate data interaction, such as validating user input, updating UI elements, and displaying messages. This allows students to practice testing these interactions without the need for a complex backend setup, simplifying the learning process.
* **Emphasis on Front-End Security:** Test Buddy allows students to focus on testing front-end security aspects, such as input validation, preventing cross-site scripting (XSS) vulnerabilities, and ensuring secure handling of user input within the browser. This is crucial for web application security.
* **Scalability and Extensibility:** Test Buddy's modular design and use of web technologies make it scalable and extensible. New UI components and interactive examples can be easily added to the platform, allowing it to grow and adapt to evolving automation testing needs.

Test Buddy leverages web technologies (HTML, CSS, JavaScript) to create a user-friendly and interactive platform that supports effective automation testing practice and education

**Proposed System**

The proposed Test Buddy platform is a web-based, interactive platform designed to provide a centralized collection of UI components for automation testing practice. By leveraging web technologies (HTML, CSS, JavaScript) to create interactive components and simulate user interactions, this platform allows students and educators to access a variety of testable elements in a single location. This project aims to make learning and teaching automation testing more efficient, engaging, and practical.

**2.1 Study of Similar Systems**

In the current landscape, learning automation testing often involves navigating numerous websites to find various UI components for practice. This fragmented approach leads to inconsistencies in environments and a lack of centralized learning resources. Many existing platforms lack comprehensive component libraries, detailed code examples, and clear documentation, limiting their effectiveness for educational purposes.

In contrast, the proposed Test Buddy platform incorporates:

* **Centralized Component Library:** A wide range of interactive UI components in a single location.
* **Comprehensive Code Examples:** Detailed code examples in multiple languages (Python Selenium, JavaScript Selenium, HTML, JavaScript) for each component.
* **Clear Documentation:** User-friendly documentation explaining each component and its relevance to automation testing.
* **Modular Extensibility:** A design that allows for easy addition of new components and features.

By comparing similar systems, we identified the need for a user-centered, comprehensive, and well-documented platform that surpasses the limitations of existing resources

**2.2 Feasibility Study**

The feasibility of the Test Buddy project is assessed across technical, economic, and operational perspectives.

* **Technical Feasibility:** The project utilizes standard web technologies (HTML, CSS, JavaScript) and common automation testing tools (Selenium). These technologies are widely available and well-documented, ensuring long-term viability.
* **Economic Feasibility:** As an educational project, the development cost is primarily time investment. All necessary software and tools are freely available or open-source.
* **Operational Feasibility:** The platform's web-based nature ensures accessibility across various operating systems and devices. Its intuitive interface and clear documentation make it easy to use for students and educators.

**2.3 Objectives of the Proposed System**

The main objectives of the Test Buddy platform are as follows:

* **Educational Tool:** Provide a centralized, interactive platform for learning and practicing automation testing.
* **Comprehensive Component Coverage:** Offer a wide range of UI components commonly used in web applications.
* **Clear Code Examples and Documentation:** Provide detailed code examples and user-friendly documentation for each component.
* **Extensibility:** Design the platform to be easily expandable with new components and features.

**2.4 Functional and Non-Functional Requirements**

**Functional Requirements:**

* **Collection of Interactive UI Components:** 
  + **This includes a diverse range of UI elements such as:** 
    - **Buttons:** Single-click, double-click, and right-click buttons to test click events.
    - **Input Fields:** Text fields, password fields, and other input types to practice data entry and validation.
    - **Dropdowns:** Single-select and multi-select dropdown menus to test option selection.
    - **Checkboxes and Radio Buttons:** To test selection and state verification**.**
    - **Date Pickers:** To simulate and test date selection.
    - **Drag and Drop and Slider:** To simulate user interface drag and drop and slider functionality.
    - **Web Tables:** To test data extraction and manipulation from tables.
* **Code Examples in Multiple Languages:** 
  + **Provide comprehensive code examples for each UI component in:** 
    - **Python Selenium:** Demonstrating how to automate interactions using Python and Selenium.
    - **JavaScript Selenium:** Showing how to automate interactions using JavaScript and Selenium.
    - **HTML:** Displaying the underlying HTML structure of each component**.**
    - **JavaScript**: Showing the underlying JavaScript functionality of each component.
* **Clear Component Documentation:** 
  + **Detailed documentation for each UI component, including:** 
    - **Description:** Explaining the purpose and functionality of the component.
    - **Usage Instructions**: Providing step-by-step instructions on how to interact with the component.
    - **Code Snippets:** Displaying relevant code examples.
    - **Explanation of how the code works.**
* **Interactive Features Implementation:** 
  + **Implement interactive features to simulate real-world user interactions:** 
    - **Button Clicks:** Handling single, double, and right-click events.
    - **Form Submissions:** Simulating form submissions and data validation.
    - **Drag-and-Drop:** Allowing users to drag and drop elements.
    - **Slider:** Allowing users to test slider functionality.
    - **Hover:** Allowing users to test mouse hover functionality.
    - **Switching Tabs and Windows:** Allowing users to test the automation of new tabs and windows.
* **Iframe Testing:** 
  + **Iframe:** 
    - Provide examples of nested and multiple iframes to practice switching between frames and interacting with elements within them.
    - Provide code examples of how to switch to the iframes.
* **File Upload and Download Testing:** 
  + **File Upload/Download:** 
    - Implement features to simulate file uploads and downloads, allowing users to practice testing these functionalities.
    - Provide code examples of how to automate file uploading and downloading.
* **Excel File Testing:** 
  + **Excel File Testing:** 
    - Include a module that simulates interactions with Excel files, enabling users to practice reading and writing data.
    - Provide code examples of how to automate excel file interactions.

**Non-Functional Requirements:**

* **Usability:** Maintain a clear and intuitive interface.
* **Accessibility:** Ensure the platform is accessible to users with disabilities.
* **Maintainability:** Design the platform for easy maintenance and updates.
* **Responsiveness:** Ensure the platform works well on different screen sizes and devices.
* **Performance:** Minimize loading times and ensure smooth interactions.

**2.5 Users of the System**

**The system primarily targets the following users:**

* Students learning automation testing.
* Educators teaching automation testing.
* Automation testers looking for practice and examples.
* Developers needing to test UI components.

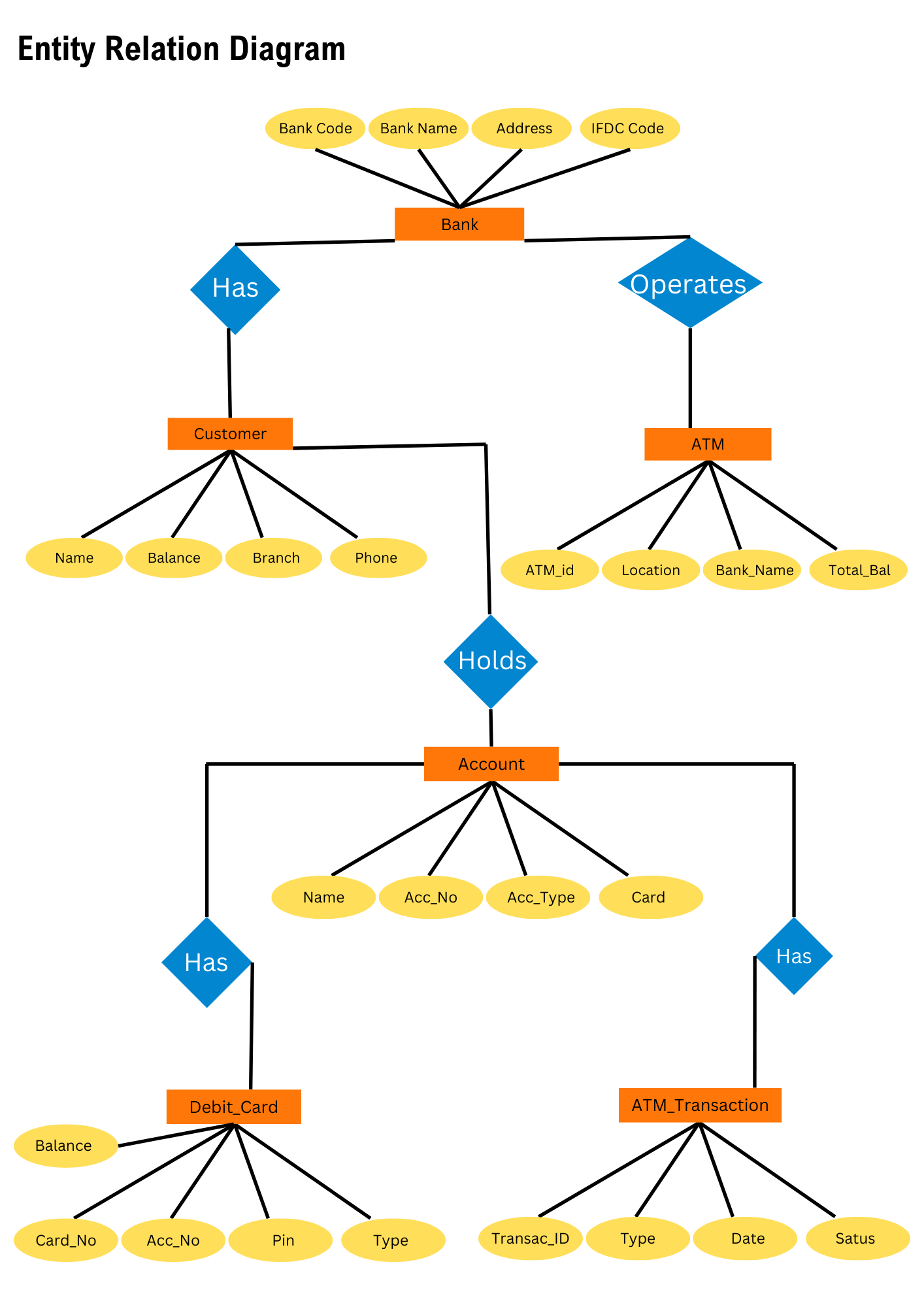
**2.6 Module Specification**

The Test Buddy platform is structured into modules, each focusing on a specific type of UI component or testing scenario.

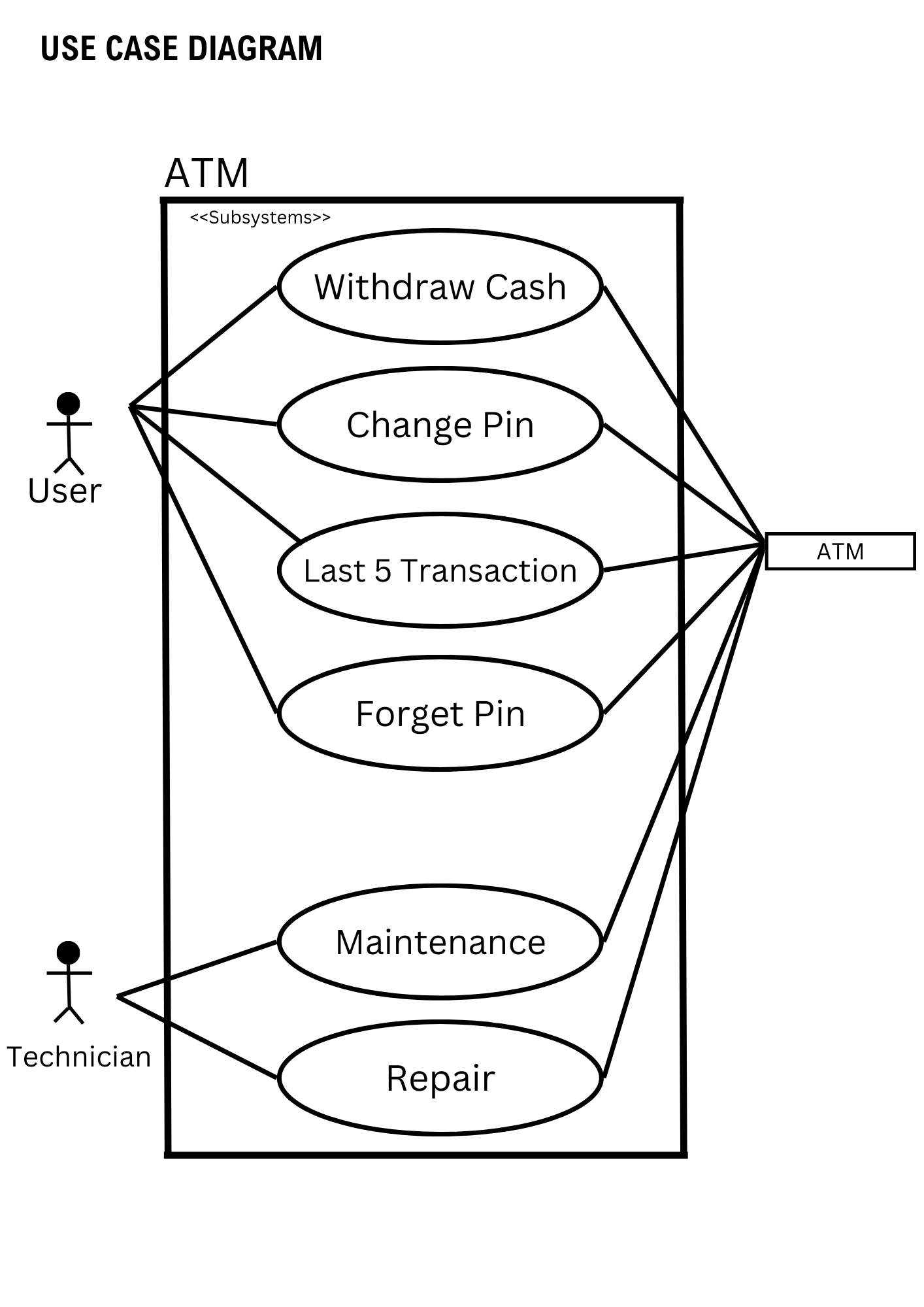
1. **Basic Interaction Module:**
   * Includes buttons (single, double, right-click), input fields (text, password), and form submissions.
   * Provides code examples for basic interactions.
2. **Selection and Navigation Module:**
   * Includes dropdown menus (single and multi-select), radio buttons, and checkboxes.
   * Includes hover and switching tabs/windows.
   * Provides code examples for selecting options and navigating.
3. **Advanced UI Module:**
   * Includes date pickers, drag-and-drop elements, and sliders.
   * Includes file upload and download.
   * Includes excel file interactions.
   * Includes Web Tables.
   * Includes scrolling.
   * Includes iframes (nested and multiple).
   * Provides code examples for advanced UI interactions.
4. **Documentation and Examples Module:**
   * Provides clear documentation for each component.
   * Includes code examples in multiple languages.
   * Includes Html code examples.
5. **JavaScript Events Module:**
   * Includes code examples of on mouse over, and on key press events.

**System and Analysis Design**

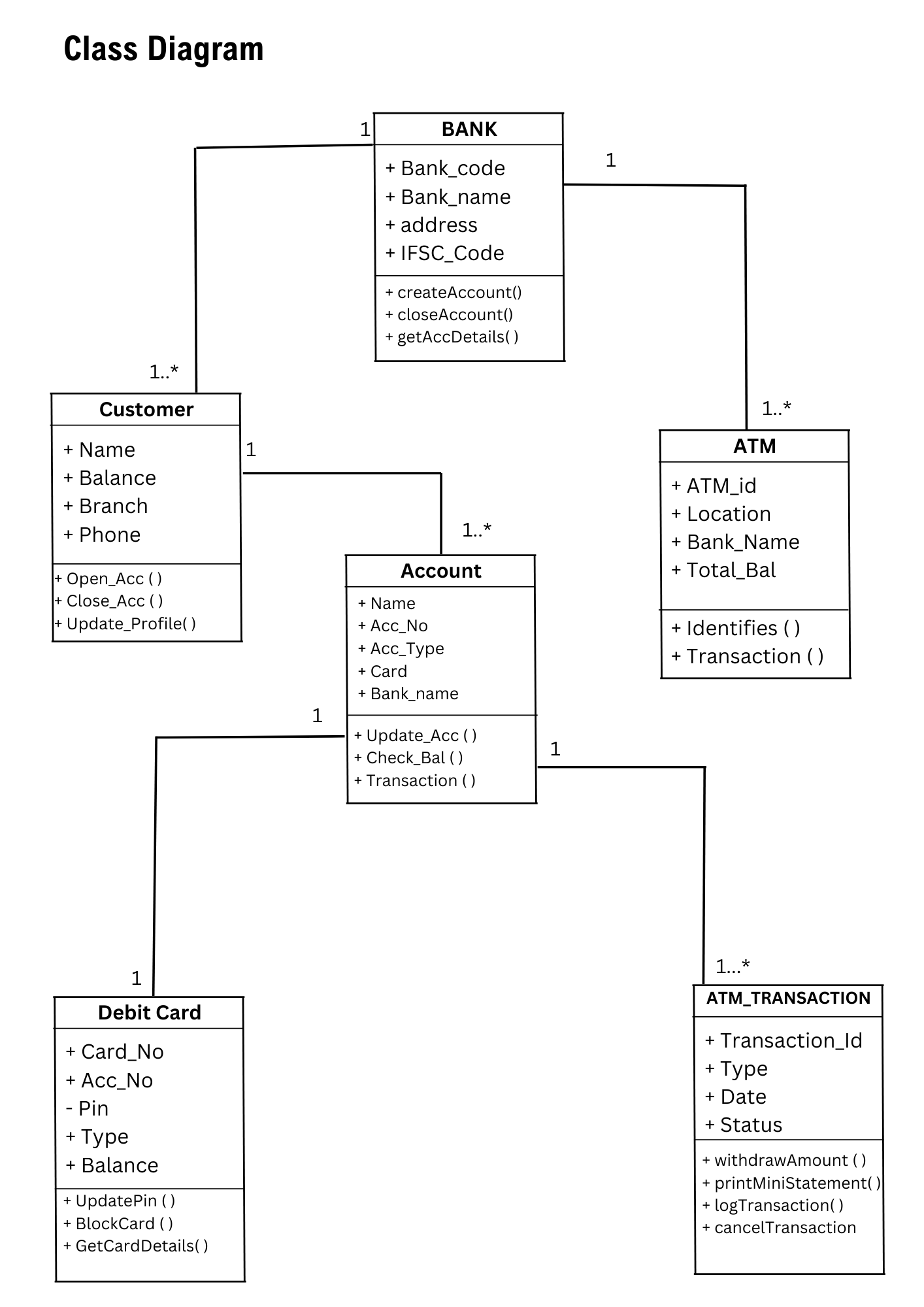
**3.1Entity Relation Diagram**

****

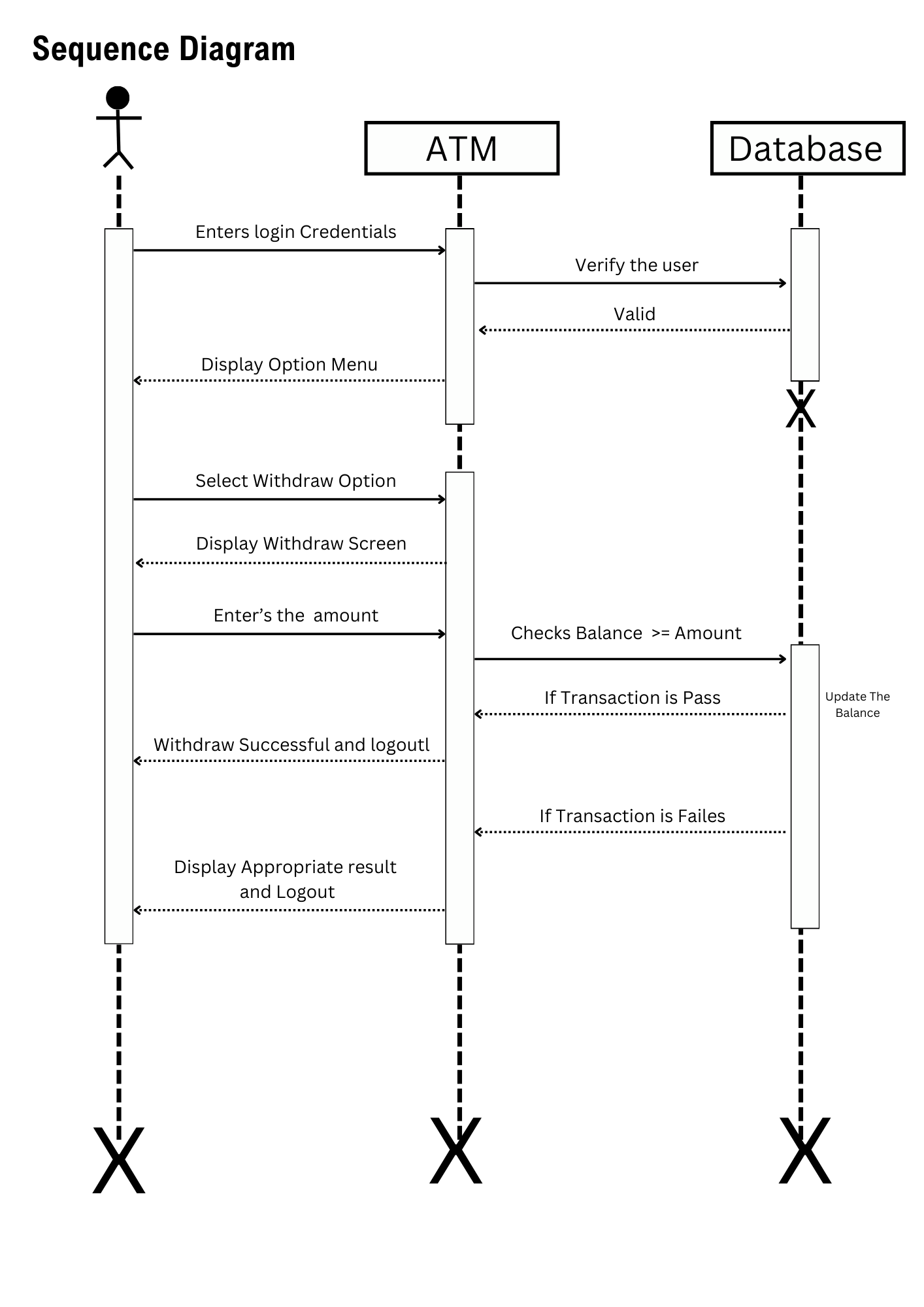
**3.2 Use Case Diagram**

****

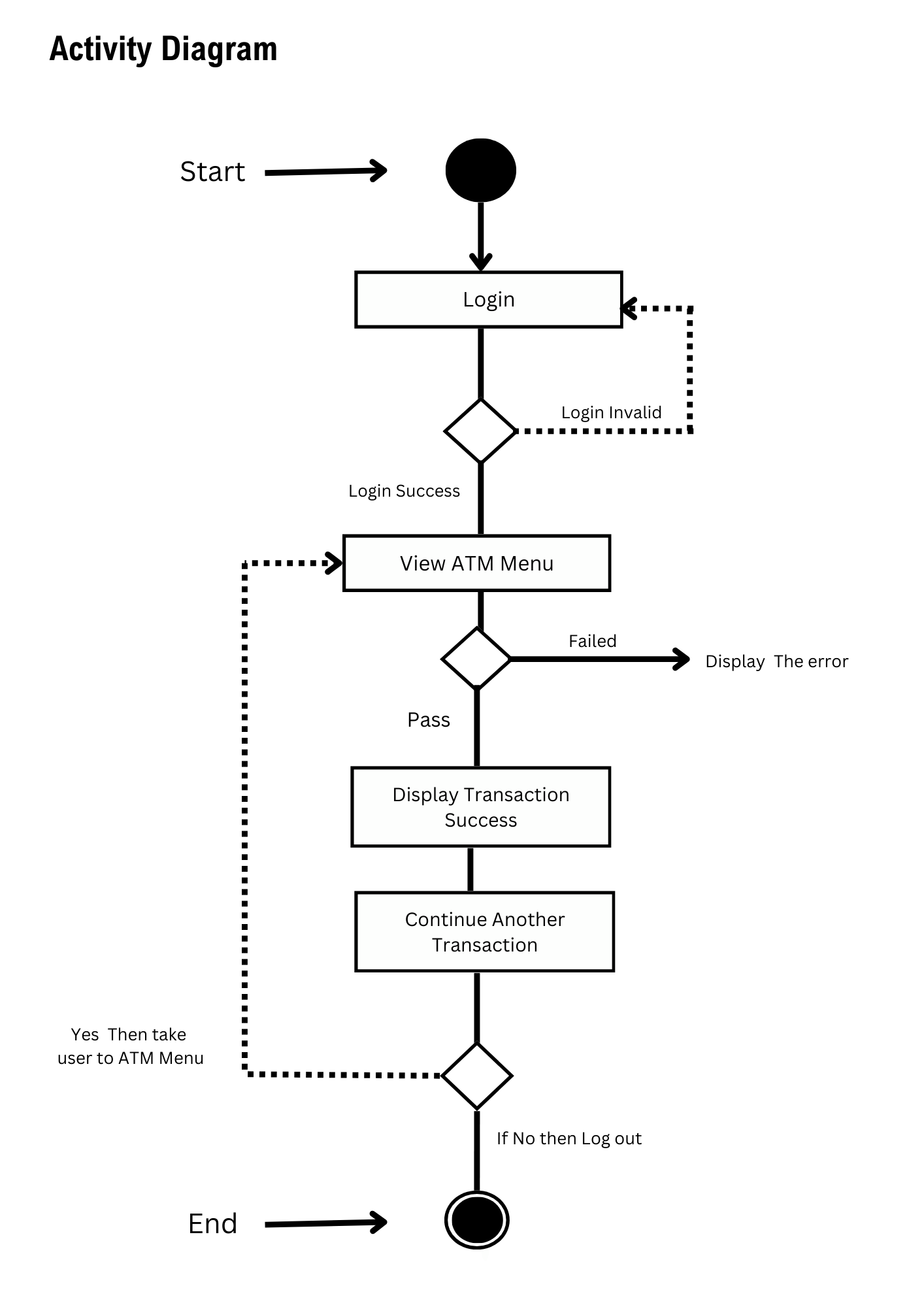
**3.3 Class Diagram**

****

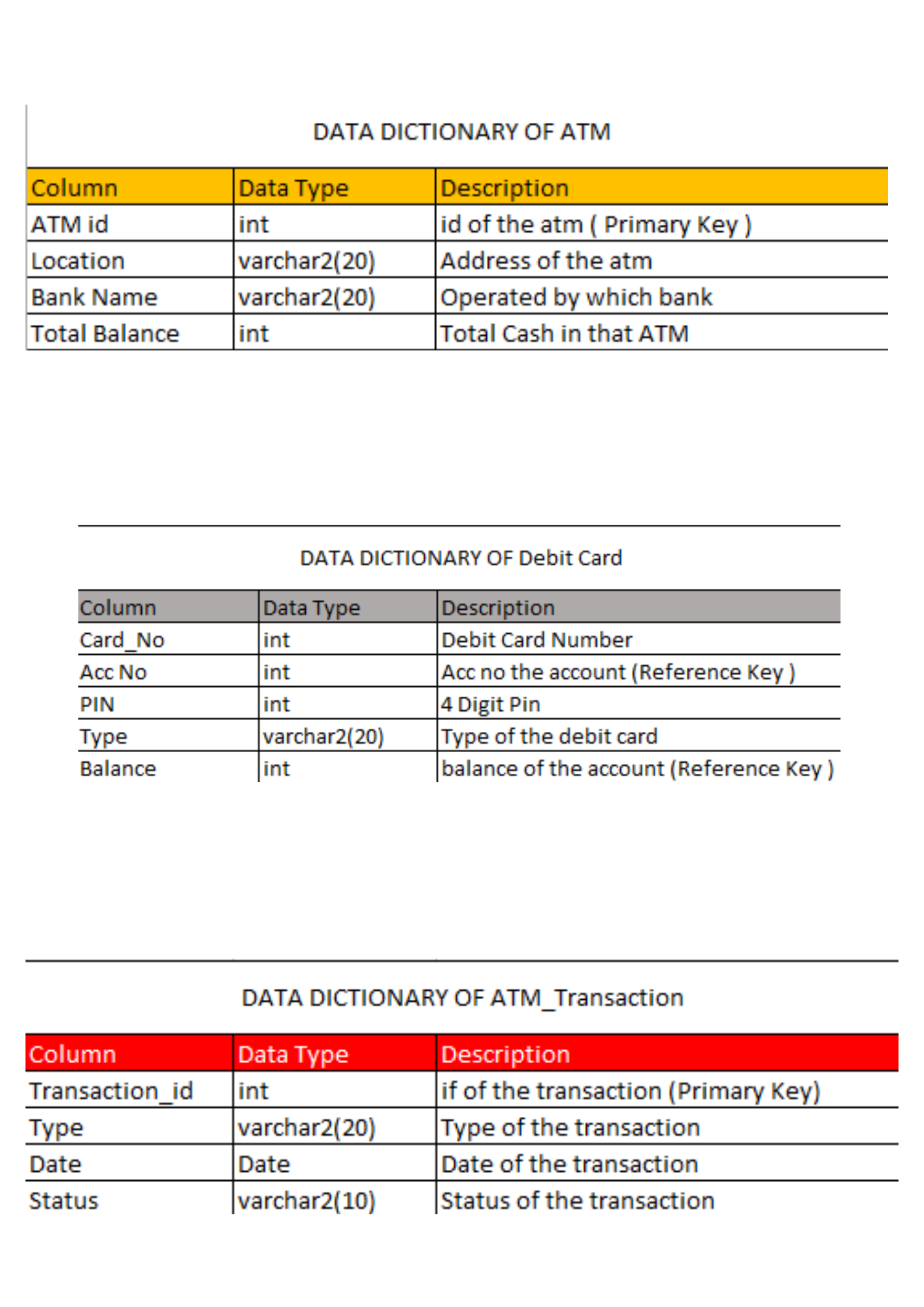
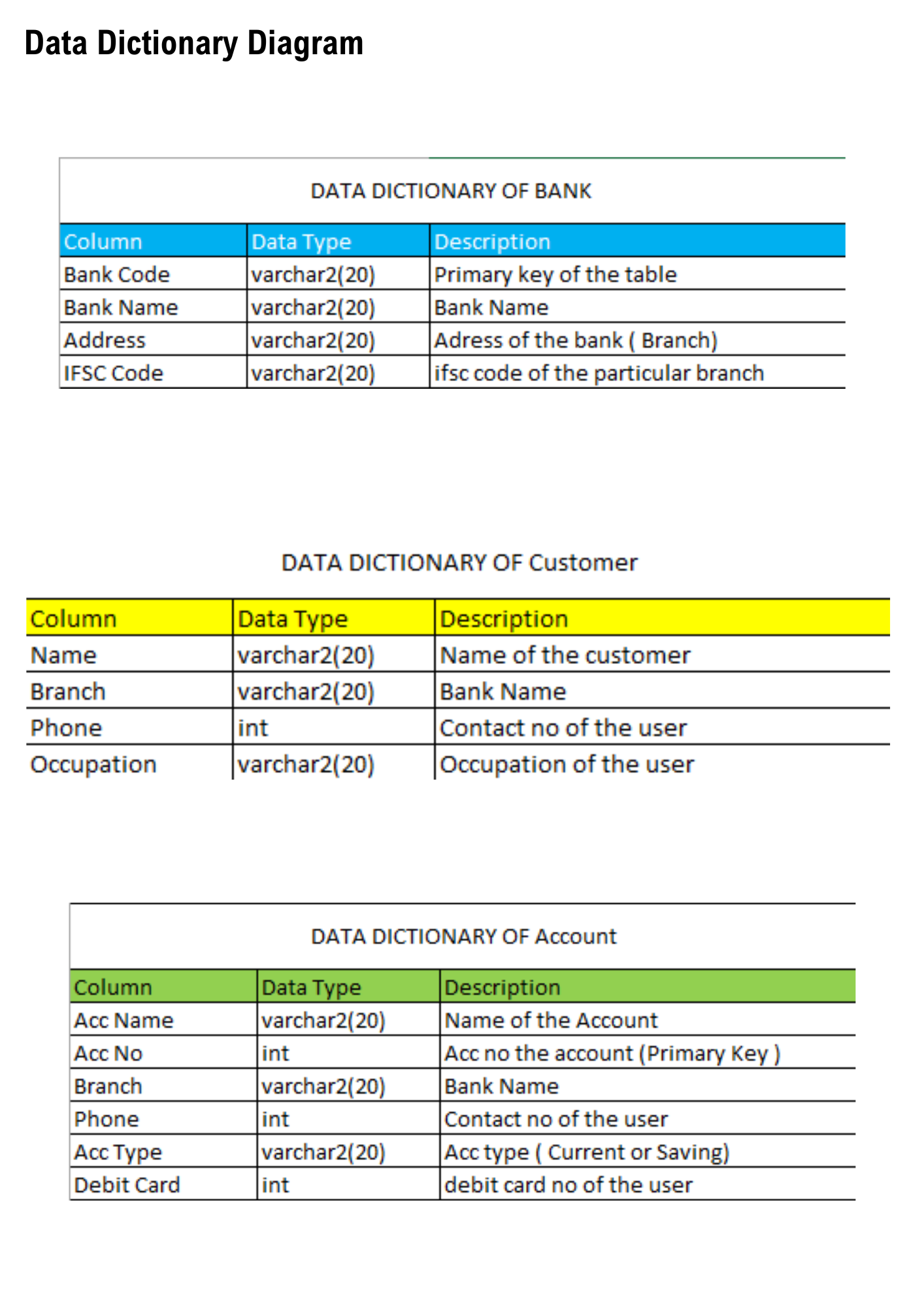
**3.4 Sequence Diagram**

****

**3.5 Activity Diagram**

****

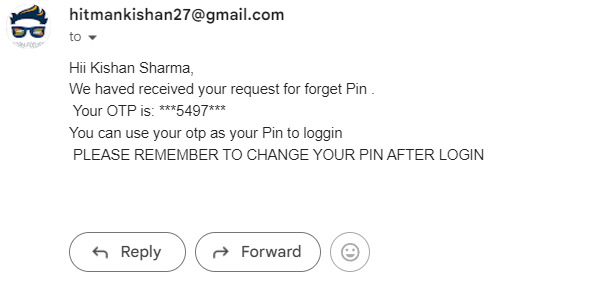
**3.6 Table Specification [ Data Dictionary ]**

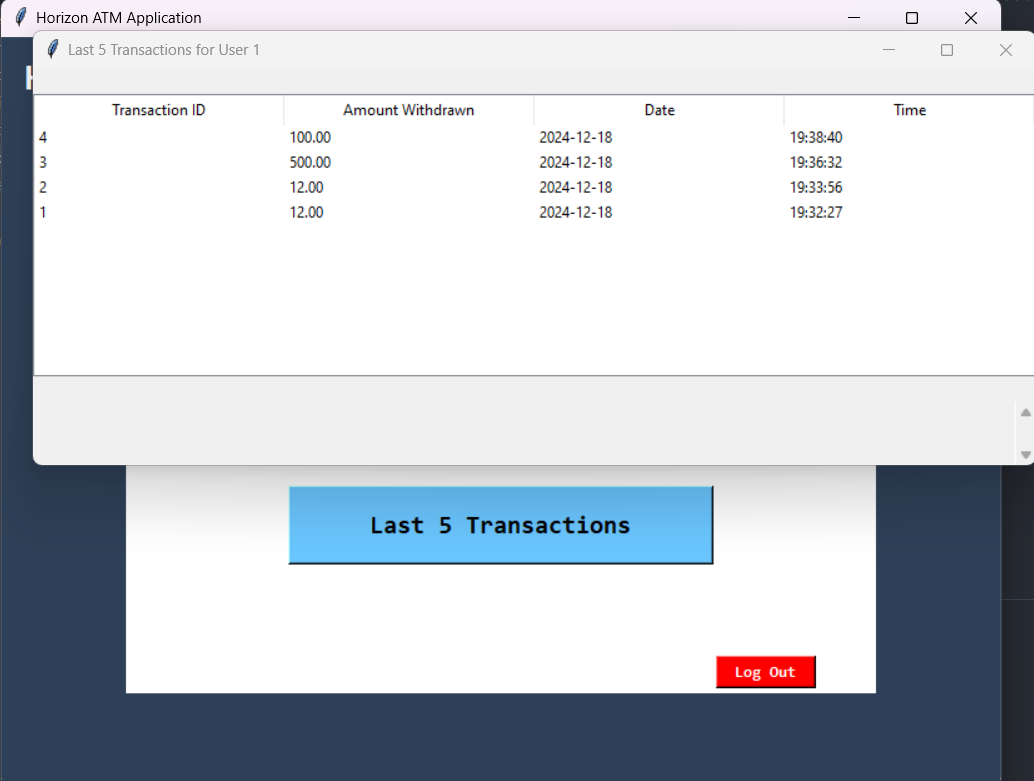
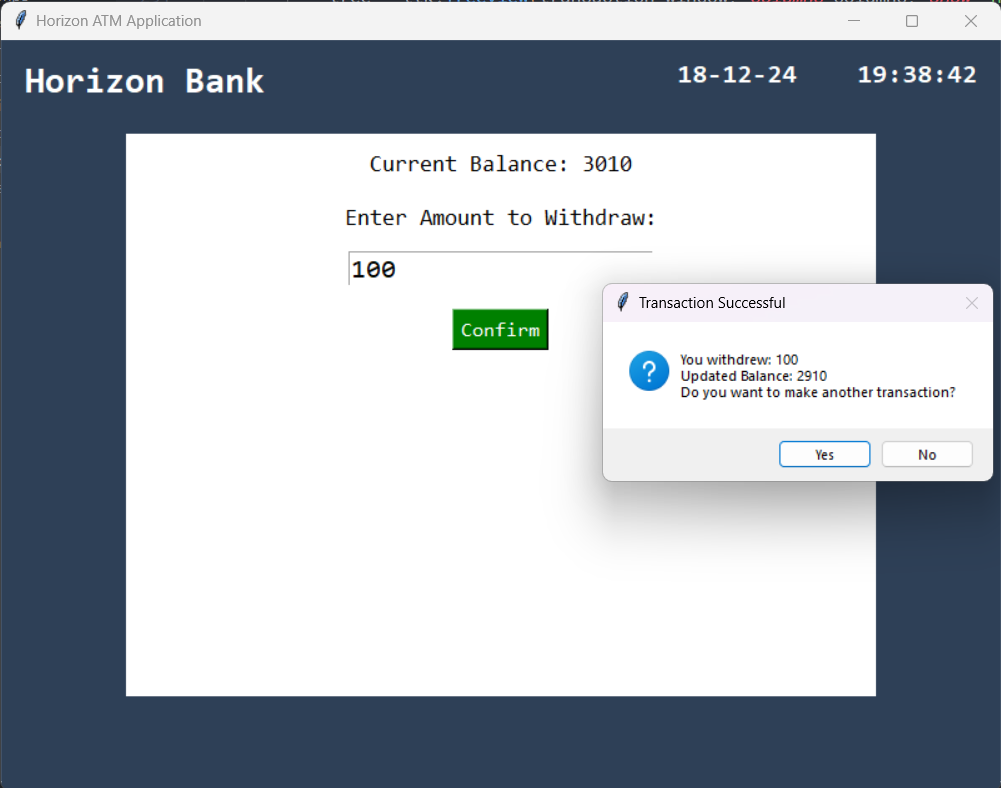
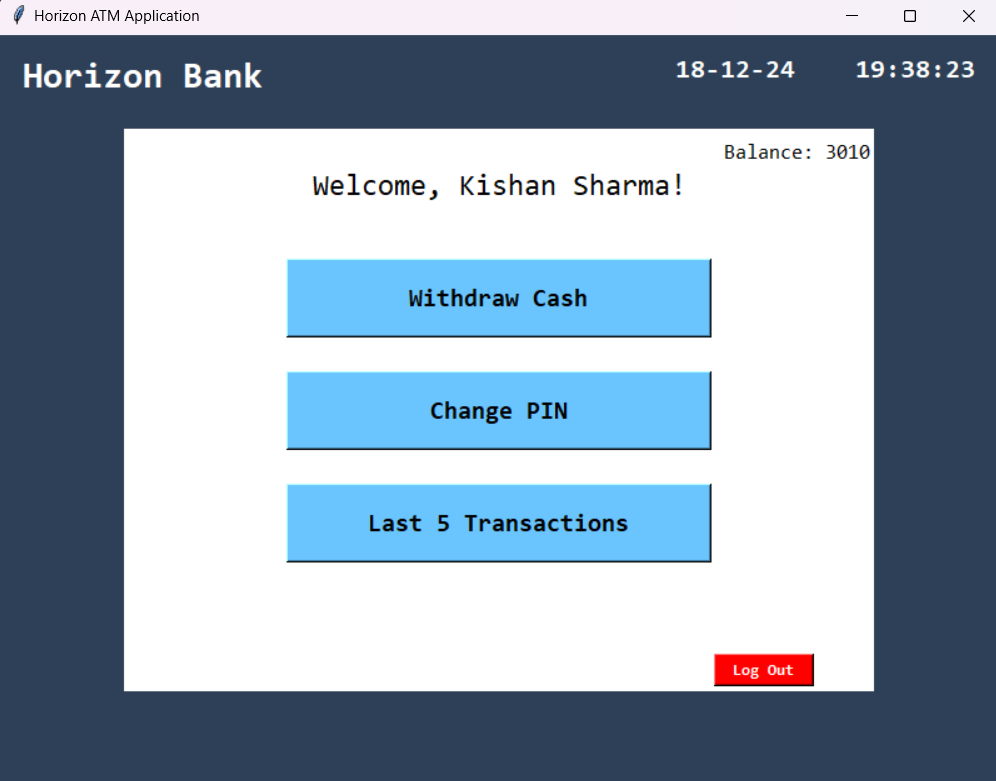
****

**3.7 User Interface Design**

****

****

****

****

**Drawbacks And Limitation**

While the ATM simulation project successfully demonstrates essential banking functionalities and provides a learning platform, it is not without its challenges and limitations. These drawbacks highlight the scope for refinement and provide direction for future improvements.

1. **Single-User Functionality**:  
   The current design supports only one user session at a time. In real-world applications, multiple users can access ATM services simultaneously, and this project does not address that scalability.
2. **Simplistic Error Handling**:  
   Error handling is limited to basic validations, such as PIN mismatch or insufficient balance. More robust systems include comprehensive logs, fail-safe mechanisms, and detailed feedback for troubleshooting.
3. **Lack of Advanced Security Protocols**:  
   Security features are limited to PIN-based authentication. Modern ATMs implement advanced cryptographic protocols, hardware security modules (HSM), and biometric authentication systems, which are not included in this project.
4. **Absence of Real-Time Synchronization**:  
   This system uses a local MySQL database for storing user data. Unlike actual ATMs, which synchronize data with centralized banking systems in real-time, the simulation operates in isolation.
5. **Graphical User Interface (GUI) Limitations**:
   * The GUI is designed for basic interactions, lacking sophisticated features such as touchscreen support or voice assistance for visually impaired users.
   * The absence of localization options, such as multi-language support, restricts its accessibility to a wider audience.
6. **Limited Feature Set**:  
   Essential functionalities like inter-account transfers, deposit options, or loan application management are not present. While the system serves its educational purpose, it falls short of replicating a full-fledged ATM experience.
7. **No Integration with External APIs**:  
   The system does not utilize APIs for real-world functionalities such as validating credentials with banking servers or sending transaction notifications via email or SMS.
8. **Dependency on Predefined Data**:  
   The system relies on pre-loaded data for demonstration purposes. Real-world ATMs allow users to dynamically create and manage accounts, which is not feasible in this project.

**Proposed Enhancements**

To address the aforementioned limitations and extend the capabilities of the project, the following enhancements are proposed:

1. **Multi-User Functionality**:  
   Implement session management techniques to allow multiple users to interact with the system concurrently. This would simulate real-world banking environments more effectively.
2. **Advanced Error Handling**:
   * Introduce a comprehensive logging mechanism to track user activities and system errors.
   * Add more user-friendly error messages and step-by-step guidance for resolving issues during transactions.
3. **Improved Security Measures**:
   * Incorporate encryption algorithms like AES or RSA for secure data storage and communication.
   * Add two-factor authentication (2FA) using OTPs sent via SMS or email.
   * Implement biometric verification, such as fingerprint or facial recognition, to enhance security.
4. **Real-Time Synchronization**:  
   Develop an API-based architecture to synchronize the local database with a central banking server, simulating real-world data management practices.
5. **Enhanced GUI Features**:
   * Utilize advanced UI frameworks like PyQt or Tkinter extensions for a more interactive and visually appealing interface.
   * Add features like voice assistance, high-contrast themes for accessibility, and touchscreen compatibility.
6. **Expanded Feature Set**:
   * Include additional functionalities like fund transfers, recurring payments, bill payments, and account statement generation.
   * Introduce customizable transaction limits and overdraft protection settings.
7. **API Integration**:
   * Integrate third-party APIs for sending transaction alerts, generating dynamic QR codes for payments, and validating user credentials against centralized databases.
   * Implement APIs for integrating with external banking services for a more holistic experience.
8. **Dynamic Account Management**:  
   Allow users to create and manage accounts directly through the interface, with features like password recovery, profile updates, and account deletion.

**Conclusion**

This project has successfully created a foundational framework for simulating ATM operations in a virtual environment. By leveraging Python for logic implementation, MySQL for data management, and Tkinter for GUI design, the application effectively bridges the gap between theoretical learning and practical application.

The ATM simulation serves as an educational tool, enabling users to explore software development concepts, understand database interactions, and appreciate the importance of security in financial applications. The modular design of the application ensures that it can be expanded and adapted to meet future requirements.

While the current iteration of the project demonstrates significant potential, incorporating the proposed enhancements will elevate it from an educational simulation to a robust, real-world application model. This project not only empowers learners with technical skills but also inspires further innovation in software development for the banking sector.

**Bibliography**

* Python Official Documentation: <https://docs.python.org/3/>
* MySQL Official Documentation: <https://dev.mysql.com/doc/>
* Tkinter GUI Programming: <https://tkdocs.com/>
* Stack Overflow Community Forum: <https://stackoverflow.com/>
* Programming Tutorials by GeeksforGeeks: <https://www.geeksforgeeks.org/>
* API Integration Concepts: <https://developer.mozilla.org/>

**Sample Code**

1. **Code of SMTP Module**

import tkinter as tk

from tkinter import messagebox

import mysql.connector # Make sure to install mysql-connector-python

from mysql.connector import Error

import smtplib

import random # Import random for generating OTP

def create\_connection():

"""Establish a database connection and return the connection and cursor."""

try:

connection = mysql.connector.connect(

host='localhost',

database='bank',

user='root',

password='kishan'

)

if connection.is\_connected():

cursor = connection.cursor()

return connection, cursor

except Error as e:

print(f"Error connecting to MySQL: {e}")

return None, None

def update(sent\_otp, entered\_otp, user\_id, parent\_frame):

if str(sent\_otp) == entered\_otp:

connection, cursor = create\_connection()

if connection and cursor:

try:

# Update the PIN in the database

cursor.execute("UPDATE Account SET pin = %s WHERE user\_id = %s", (entered\_otp, user\_id))

connection.commit() # Commit the changes to the database

messagebox.showinfo("Success", "OTP verified successfully. You can now proceed to log in with your new PIN.")

except Error as e:

messagebox.showerror("Error", f"Database error: {e}")

finally:

cursor.close()

connection.close()

else:

messagebox.showerror("Error", "Invalid OTP. Please try again.")

def create\_forgot\_pin\_frame(parent\_frame):

# Clear the parent frame (inside\_frame) before adding the forgot pin elements

for widget in parent\_frame.winfo\_children():

widget.destroy()

# Forgot PIN Frame

forgot\_pin\_frame = tk.Frame(parent\_frame, bg="white", padx=20, pady=20)

forgot\_pin\_frame.pack(fill="both", expand=True)

# User ID Entry

user\_id\_label = tk.Label(forgot\_pin\_frame, text="Enter User ID:", font=("Consolas", 14), bg="white")

user\_id\_label.pack(pady=(10, 5))

user\_id\_entry = tk.Entry(forgot\_pin\_frame, font=("Consolas", 14), width=30)

user\_id\_entry.pack(pady=(0, 20))

# Variable to store the generated OTP

generated\_otp = None

# Send OTP Button

def send\_otp():

nonlocal generated\_otp # Use the nonlocal variable to access it in nested function

# Get the User ID from the entry

user\_id = user\_id\_entry.get()

# Disable the Send OTP button

send\_otp\_button.config(state="disabled")

connection, cursor = create\_connection()

if connection and cursor:

try:

# Fetch the email associated with the User ID

cursor.execute("SELECT email FROM Account WHERE user\_id = %s", (user\_id,))

result = cursor.fetchone()

cursor.execute("SELECT name FROM Account WHERE user\_id = %s", (user\_id,))

result1 = cursor.fetchone()

if result:

# Generate a random 4-digit OTP

generated\_otp = random.randint(1000, 9999)

# Prepare email

email = result[0]

sender = "hitmankishan27@gmail.com"

subject = "Forget PIN"

message = f"Hii {result1[0]},\nWe haved received your request for forget Pin . \n Your OTP is: \*\*\*{generated\_otp}\*\*\* \nYou can use your otp as your Pin to loggin \n PLEASE REMEMBER TO CHANGE YOUR PIN AFTER LOGIN"

text = f"Subject: {subject}\n\n{message}"

# Send email

server = smtplib.SMTP("smtp.gmail.com", 587)

server.starttls()

server.login(sender, "ivroidpszqvszfev")

server.sendmail(sender, email, text)

server.quit() # Close the SMTP server connection

# Show alert message and directly create OTP entry

messagebox.showinfo("OTP Sent", f"An OTP has been sent to the registered Email-Id for User ID: {user\_id}")

# OTP Entry and Submit Button

otp\_label = tk.Label(forgot\_pin\_frame, text="Enter OTP:", font=("Consolas", 14), bg="white")

otp\_label.pack(pady=(10, 5))

otp\_entry = tk.Entry(forgot\_pin\_frame, font=("Consolas", 14), width=30)

otp\_entry.pack(pady=(0, 20))

# Submit button with a call to the update function

submit\_button = tk.Button(forgot\_pin\_frame, text="Submit", font=("Consolas", 12), bg="green", fg="white",

command=lambda: update(generated\_otp, otp\_entry.get(), user\_id\_entry.get(), parent\_frame))

submit\_button.pack(pady=(10, 5))

else:

messagebox.showerror("Error", "User ID not found.")

except Error as e:

messagebox.showerror("Error", f"Database error: {e}")

finally:

cursor.close()

connection.close()

else:

messagebox.showerror("Error", "Failed to connect to the database.")

# Create and place the Send OTP button

send\_otp\_button = tk.Button(forgot\_pin\_frame, text="Send OTP", font=("Consolas", 12), bg="blue", fg="white", command=send\_otp)

send\_otp\_button.pack(pady=(10, 5))

return forgot\_pin\_frame

1. **Login Verfication**

def check\_credentials(user\_id, pin):

connection, cursor = create\_connection()

if connection and cursor:

cursor.execute("SELECT name, balance FROM Account WHERE user\_id = %s AND pin = %s", (user\_id, pin))

result = cursor.fetchone()

if result:

username, balance = result

clear\_frame(inside\_frame)

# Display balance and welcome message

balance\_label = tk.Label(inside\_frame, text=f"Balance: {balance}", bg="white", fg="black", font=("Consolas", 13))

balance\_label.place(relx=1, rely=0.01, anchor="ne")

welcome\_label = tk.Label(inside\_frame, text=f"Welcome, {username}!", bg="white", fg="black", font=("Consolas", 18))

welcome\_label.place(relx=0.5, rely=0.1, anchor="center")

# Function buttons

create\_function\_buttons(user\_id, balance, pin)

else:

invalid()

cursor.close()

connection.close()