

**RAJALAKSHMI ENGINEERING COLLEGE**

**(AUTONOMOUS)**

THANDALAM

In partial fulfillment for the award of the degree of

**BACHELOR OF TECHNOLOGY IN**

# ARTIFICAL INTELLIGENCE AND MACHINE LEARNING

**A MINI PROJECT**

**REPORT ON**

**GYM MANAGEMENT SYSTEM:**

**Submitted by**

**TANISH P.S 231501165**

**TARUN A.D 231501163**

**VISHRUDHA 231501187**

**BONAFIDE**

**CERTIFICATE**

Certified that this project report “**GYM MANAGEMENT** ” is the bonafide work of **“TANISH. P.S (231501170), TARUN.A.D (231501171)”**

who carried out the project work under my supervision.

**Submitted for the Practical Examination held on** -----------------------------------------------

**SIGNATURE SIGNATURE**

**INTERNAL EXAMINER EXTERNAL EXAMINER**

# ABSTRACT

## Problem Statement

The Gym Management System aims to streamline the management of gym memberships, member information, and user authentication. It addresses challenges like inefficient record-keeping, cumbersome member registration, and lack of secure access control. The system will allow administrators to easily register members, manage their data, and securely authenticate users. It will display member details in an organized format and provide data security. This solution will enhance efficiency, accuracy, and user experience in managing gym operations. The goal is to simplify daily operations and improve overall management of gym resources.

## Introduction

The **Gym Management System** is a software solution designed to help gym owners and administrators manage their gym operations efficiently. In an era where digital solutions are essential for smooth operations, this system provides an automated way to handle key administrative tasks such as member registration, user authentication, and data management. By centralizing and organizing member information, the system reduces the need for manual record-keeping, thereby minimizing errors and saving time.

This system not only allows gym administrators to add, update, and view member details but also ensures secure access through a login system for authorized users. With a user-friendly interface, the Gym Management System is designed to streamline day-to-day activities, improve operational efficiency, and provide a seamless experience for both staff and members. By leveraging technology, the system enables gyms to focus more on providing excellent service to their members while managing their internal processes effectively.

## Novelty

The \*\*Gym Management System\*\* introduces a novel approach to managing gym operations by providing a centralized, secure, and automated solution. It simplifies member registration and data management, allowing gym administrators to update and track member details with ease. The system includes a secure user authentication process, ensuring only authorized personnel can access sensitive information. Real-time access to member data and attendance improves operational efficiency. The system is scalable and customizable, making it suitable for gyms of all sizes. By automating routine tasks, it reduces administrative workload and minimizes errors. This innovative approach enhances overall gym management and member experience.

## Key Features

1.Member Registration and Management: Easily add, update, and manage member information, including personal details, contact information, and membership status.

2.User Authentication: Secure login functionality for gym administrators and staff, ensuring that only authorized users can access and manage sensitive data.

3.Real-Time Data Access: Display and update member data in real-time using an organized table, enabling quick retrieval and modifications of member information.

4.Member Attendance Tracking: Keep track of member attendance and participation, helping gym staff monitor gym usage and member activity.

5.Secure Database: Store all member and user information in a secure database, ensuring privacy and data protection.

## Impact

The **Gym Management System** significantly improves operational efficiency by automating member registration, data management, and attendance tracking. It reduces administrative workload and minimizes errors, allowing staff to focus on providing better services. The secure user authentication ensures that sensitive information is protected. By offering real-time access to member data, it enhances decision-making and resource management. Overall, the system streamlines gym operations, improving both staff productivity and member experience.

## Potential Future Applications

### Potential Future Applications of the Gym Management System:

1. Mobile Integration: Developing a mobile app for members to track their workouts, memberships, and make payments, offering more convenience and engagement.

2. Advanced Analytics: Implementing data analytics to track member progress, gym attendance patterns, and financial reports to help gym owners make data-driven decisions.

3. Online Booking System: Enabling members to book classes, personal training sessions, or equipment reservations through the system, enhancing user experience.

4. Automated Billing and Payment: Integrating an automated payment system for subscriptions, renewals, and invoicing, simplifying financial management for gym owners.

5. Integration with Fitness Devices: Connecting the system with wearable fitness devices or gym equipment to track member activity and provide personalized workout recommendations.

### SURVEY OF TECHNOLOGY

#### 2.1 SOFTWARE DESCRIPTION

##### Visual Studio Code

Visual Studio Code (VS Code) is a widely-used, open-source code editor developed by Microsoft that caters to a diverse range of development needs. Known for its user-friendly interface, VS Code combines the simplicity of a text editor with robust developer tools, making it an excellent choice for programming in various languages and frameworks. With features that enhance productivity and streamline workflows, VS Code has become a favoured choice among developers.

**Key Features and Benefits**

1. **IntelliSense and Autocompletion**: VS Code’s IntelliSense provides intelligent code suggestions, helping developers reduce errors and code faster by offering syntax suggestions, function names, and variable autocompletion.
2. **Integrated Debugging**: With built-in debugging tools, VS Code allows developers to set breakpoints, step through code, and inspect variables, making debugging simpler and more efficient without needing external tools.
3. **Customization and Extensions**: VS Code’s extensive marketplace offers extensions for different programming languages, frameworks, and tools, enabling developers to tailor the editor to their project requirements.
4. **Git Integration**: VS Code supports version control directly within the editor through Git integration, allowing developers to commit, push, pull, and manage code versions seamlessly.

#### 2.2 LANGUAGES USED

**FRONT END / USER INTERFACE**

**2.2.4** **Python (Tkinter):**

The user interface for the **Gym Management System** is developed using Python's Tkinter library, which allows for the creation of interactive and graphical user interfaces (GUIs). Tkinter provides a simple way to create windows, buttons, forms, and display data in a user-friendly format.

#### 2.3 Database

##### 2.3.1 SQLite

1. **Lightweight:** SQLite is a serverless, self-contained database, making it ideal for smaller-scale applications or projects like this one.
2. **Ease of Use:** It is easy to set up and integrate directly into Python applications without requiring complex configuration or external server dependencies.
3. **Relational Database:** SQLite supports SQL queries and provides relational data storage, which is essential for organizing member and user data.
4. **File-Based:** SQLite stores the database in a single file, making it simple to manage and deploy.

**How It Works:**

* The **SQLite** database is used to store essential information like gym members' personal details (name, contact information, etc.), user login credentials (username, password), and other data related to the gym management.
* The database is accessed and manipulated via Python’s sqlite3 module, which allows the system to perform tasks like adding new members, authenticating users, and fetching member details.

In case of larger or more complex gym systems, databases like **MySQL** or **PostgreSQL** could be considered, but for a small-to-medium-scale gym management system, **SQLite** is a sufficient and efficient choice.

### 3.1 Requirement Specification

#### The project requirements are divided into functional and non-functional requirements to ensure clarity and completeness in addressing both the operational and System.

### 3.1. Functional Requirements

**1. User Authentication & Authorization**

* **Description**: Implement a secure login system for both administrators and staff. This includes password protection and role-based access control to ensure that only authorized users have access to specific functionalities.
* **Features**:
  + Login and registration functionality.
  + Password encryption and secure storage.
  + Role-based access control to differentiate between admin and staff permissions.

**2. Member Management**

* **Description**: Manage member records efficiently by adding, updating, deleting, and searching for member details, including their personal information and membership status.
* **Features**:
  + Add new member records with details such as name, date of birth, phone number, and email.
  + Update existing member information.
  + Delete member records when needed.
  + Search and retrieve member records based on various criteria.

**3. Class Scheduling & Booking**

* **Description**: Facilitate the creation and management of gym classes. Allow members to view schedules and book available slots for classes or training sessions.
* **Features**:
  + Admins can create, update, and delete class schedules.
  + Members can view class schedules and book slots.
  + Notification system for upcoming classes and booking confirmations.

**4. Subscription & Payment Tracking**

* **Description**: Support various membership plans with automatic billing and the ability to track payments and renewals.
* **Features**:
  + Define different membership plans and pricing.
  + Track member subscriptions and payment statuses.
  + Generate invoices and payment reminders.

**5. Real-Time Data & Reporting**

* **Description**: Display real-time data related to member activities, attendance tracking, and generate reports on gym performance, memberships, and revenue.
* **Features**:
  + Real-time dashboard displaying key metrics.
  + Attendance tracking for classes and gym visits.
  + Customizable reports on memberships and financial performance.

**3.1.2 Non-Functional Requirements**

**Performance**

* **Requirement**: The system must handle multiple simultaneous users without significant slowdowns, ensuring fast response times, especially during peak usage times.
* **Goal**: Maintain an average response time of less than 2 seconds for all operations.

**Scalability**

* **Requirement**: The system should be able to scale efficiently to accommodate a growing number of members, classes, and data as the gym expands.
* **Goal**: Support up to 10,000 active users and a similar number of class bookings per day without performance degradation.

**Security**

* **Requirement**: Sensitive member and user data, such as personal information and passwords, must be encrypted and stored securely, adhering to best practices for data protection and privacy regulations (e.g., GDPR).
* **Goal**: Implement AES-256 encryption for data storage and secure HTTPS for data transmission.

**Usability**

* **Requirement**: The user interface should be intuitive, easy to navigate, and responsive across devices to ensure a seamless experience for both gym staff and members.
* **Goal**: Achieve a user satisfaction rating of 90% or higher based on usability tests and feedback.

**Reliability and Availability**

* **Requirement**: The system must be highly reliable, with minimal downtime, and must ensure that data is consistently available for real-time operations, including member registrations, class bookings, and payments.
* **Goal**: Ensure 99.9% uptime and implement automated backups and failover mechanisms.

**3.2 Hardware and Software Requirements**

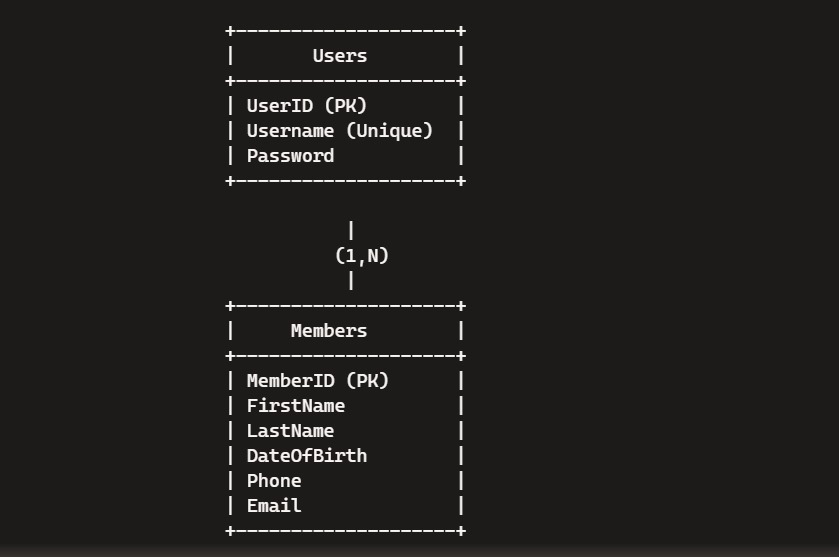
**3.2.1 Hardware Requirements**

1. **For Local System Deployment**:
   * **Processor**: Intel i5 or equivalent, 2.5 GHz or higher.
   * **RAM**: 8 GB or more.
   * **Storage**: Minimum 256 GB SSD or equivalent.
   * **GPU**: Recommended for advanced machine learning tasks (e.g., NVIDIA GTX 1050 or higher).
2. **For Cloud Deployment (Optional)**:
   * **Virtual Machine**: 4 vCPUs.
   * **RAM**: 16 GB.
   * **Storage**: 100 GB SSD storage.
   * **GPU instance**: If neural networks are used.

**3.2.2 Software Requirements**

1. **Programming and Libraries**:
   * **Python**: 3.8 or higher.
   * **Libraries**:
     + Data Processing: NumPy, Pandas.
     + Visualization: Matplotlib, Seaborn.
     + Machine Learning: scikit-learn, TensorFlow/PyTorch (if required).
2. **Development Environment**:
   * **Jupyter Notebook** or **IDE** like PyCharm/VS Code.
3. **Database**:
   * **SQLite** or **PostgreSQL**.
4. **Operating System**:
   * **Windows 10/11**, **macOS**, or **Linux (Ubuntu 20.04 or later)**.
5. **Optional Tools**:
   * Additional tools for data analysis, debugging, and project management as needed.

**ER DIAGRAM:**



Program code:

import tkinter as tk

from tkinter import messagebox, ttk

import sqlite3

# Database connection

def get\_db\_connection():

    conn = sqlite3.connect('gym\_management.db')

    return conn

# Create the database and the necessary tables if they don't exist

def create\_table():

    conn = get\_db\_connection()

    cursor = conn.cursor()

    cursor.execute('''

        CREATE TABLE IF NOT EXISTS Members (

            MemberID INTEGER PRIMARY KEY AUTOINCREMENT,

            FirstName TEXT NOT NULL,

            LastName TEXT NOT NULL,

            DateOfBirth TEXT NOT NULL,

            Phone TEXT NOT NULL,

            Email TEXT NOT NULL

        )

    ''')

    cursor.execute('''

        CREATE TABLE IF NOT EXISTS Users (

            UserID INTEGER PRIMARY KEY AUTOINCREMENT,

            Username TEXT NOT NULL UNIQUE,

            Password TEXT NOT NULL

        )

    ''')

    conn.commit()

    cursor.close()

    conn.close()

# Function to register a new user

def register\_user():

    username = register\_username\_entry.get()

    password = register\_password\_entry.get()

    conn = get\_db\_connection()

    cursor = conn.cursor()

    try:

        cursor.execute('INSERT INTO Users (Username, Password) VALUES (?, ?)', (username, password))

        conn.commit()

        messagebox.showinfo("Success", "User  registered successfully!")

        register\_window.destroy()  # Close registration window

    except sqlite3.IntegrityError:

        messagebox.showerror("Error", "Username already exists!")

    except Exception as e:

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

    finally:

        cursor.close()

        conn.close()

# Function to validate login

def show\_gym\_management():

    create\_gym\_management\_window()  # Call the function that creates the gym management window

def validate\_login():

    username = username\_entry.get()

    password = password\_entry.get()

    conn = get\_db\_connection()

    cursor = conn.cursor()

    cursor.execute('SELECT \* FROM Users WHERE Username = ? AND Password = ?', (username, password))

    user = cursor.fetchone()

    cursor.close()

    conn.close()

    if user:

        show\_gym\_management()

    else:

        messagebox.showerror("Login Failed", "Invalid username or password")

# Create the login window

def create\_login\_window():

    global login\_window, username\_entry, password\_entry

    login\_window = tk.Tk()

    login\_window.title("Login")

    tk.Label(login\_window, text="Username").grid(row=0, column=0, padx=10, pady=10)

    username\_entry = tk.Entry(login\_window)

    username\_entry.grid(row=0, column=1, padx=10, pady=10)

    tk.Label(login\_window, text="Password").grid(row=1, column=0, padx=10, pady=10)

    password\_entry = tk.Entry(login\_window, show='\*')

    password\_entry.grid(row=1, column=1, padx=10, pady=10)

    login\_button = tk.Button(login\_window, text="Login", command=validate\_login)

    login\_button.grid(row=2, columnspan=2, pady=10)

    register\_button = tk.Button(login\_window, text="Register", command=create\_register\_window)

    register\_button.grid(row=3, columnspan=2, pady=10)

    login\_window.mainloop()

# Create the registration window

def create\_register\_window():

    global register\_window, register\_username\_entry, register\_password\_entry

    register\_window = tk.Toplevel(login\_window)

    register\_window.title("Register")

    tk.Label(register\_window, text="Username").grid(row=0, column=0, padx=10, pady=10)

    register\_username\_entry = tk.Entry(register\_window)

    register\_username\_entry.grid(row=0, column=1, padx=10, pady=10)

    tk.Label(register\_window, text="Password").grid(row=1, column=0, padx=10, pady=10)

    register\_password\_entry = tk.Entry(register\_window, show='\*')

    register\_password\_entry.grid(row=1, column=1, padx=10, pady=10)

    register\_button = tk.Button(register\_window, text="Register", command=register\_user)

    register\_button.grid(row=2, columnspan=2, pady=10)

# Create the gym management window

def create\_gym\_management\_window():

    global gym\_window, first\_name\_entry, last\_name\_entry, dob\_entry, phone\_entry, email\_entry, tree

    gym\_window = tk.Toplevel()

    gym\_window.title("Gym Management System")

    tk.Label(gym\_window, text="First Name").grid(row=0, column=0)

    first\_name\_entry = tk.Entry(gym\_window)

    first\_name\_entry.grid(row=0, column=1)

    tk.Label(gym\_window, text="Last Name").grid(row=1, column=0)

    last\_name\_entry = tk.Entry(gym\_window)

    last\_name\_entry.grid(row=1, column=1)

    tk.Label(gym\_window, text="Date of Birth (YYYY-MM-DD)").grid(row=2, column=0)

    dob\_entry = tk.Entry(gym\_window)

    dob\_entry.grid(row=2, column=1)

    tk.Label(gym\_window, text="Phone").grid(row=3, column=0)

    phone\_entry = tk.Entry(gym\_window)

    phone\_entry.grid(row=3, column=1)

    tk.Label(gym\_window, text="Email").grid(row=4, column=0)

    email\_entry = tk.Entry(gym\_window)

    email\_entry.grid(row=4, column=1)

    # Add Member button

    add\_member\_button = tk.Button(gym\_window, text="Add Member", command=add\_member)

    add\_member\_button.grid(row=5, columnspan=2, pady=10)

    # Treeview for displaying members

    tree = ttk.Treeview(gym\_window, columns=("MemberID", "FirstName", "LastName", "DateOfBirth", "Phone", "Email"), show="headings")

    tree.heading("MemberID", text="Member ID")

    tree.heading("FirstName", text="First Name")

    tree.heading("LastName", text="Last Name")

    tree.heading("DateOfBirth", text="Date of Birth")

    tree.heading("Phone", text="Phone")

    tree.heading("Email", text="Email")

    tree.grid(row=6, columnspan=2)

    # Create the database and table

    create\_table()

    # Load existing members when the gym window starts

    load\_members()

# Function to add a member to the database

def add\_member():

    first\_name = first\_name\_entry.get()

    last\_name = last\_name\_entry.get()

    dob = dob\_entry.get()

    phone = phone\_entry.get()

    email = email\_entry.get()

    conn = get\_db\_connection()

    cursor = conn.cursor()

    try:

        cursor.execute('INSERT INTO Members (FirstName, LastName, DateOfBirth, Phone, Email) VALUES (?, ?, ?, ?, ?)',

                       (first\_name, last\_name, dob, phone, email))

        conn.commit()

        messagebox.showinfo("Success", "Member added successfully!")

        clear\_entries()

        load\_members()

    except Exception as e:

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

    finally:

        cursor.close()

        conn.close()

# Function to load members from the database

def load\_members():

    conn = get\_db\_connection()

    cursor = conn.cursor()

    cursor.execute('SELECT \* FROM Members')

    members = cursor.fetchall()

    cursor.close()

    conn.close()

    # Clear the current list

    for row in tree.get\_children():

        tree.delete(row)

    # Insert new members into the tree view

    for member in members:

        tree.insert("", "end", values=(member[0], member[1], member[2], member[3], member[4], member[5]))

# Function to clear input fields

def clear\_entries():

    first\_name\_entry.delete(0, tk.END)

    last\_name\_entry.delete(0, tk.END)

    dob\_entry.delete(0, tk.END)

    phone\_entry.delete(0, tk.END)

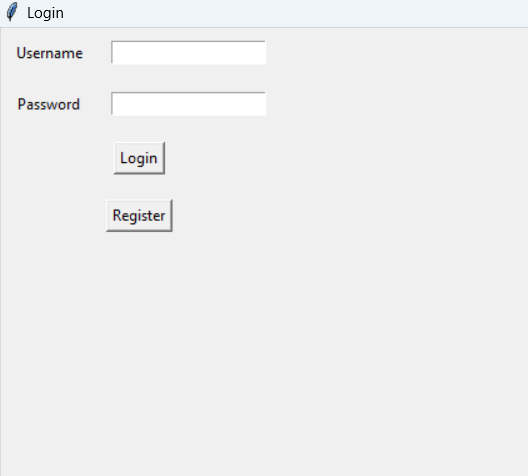
    email\_entry.delete(0, tk.END)

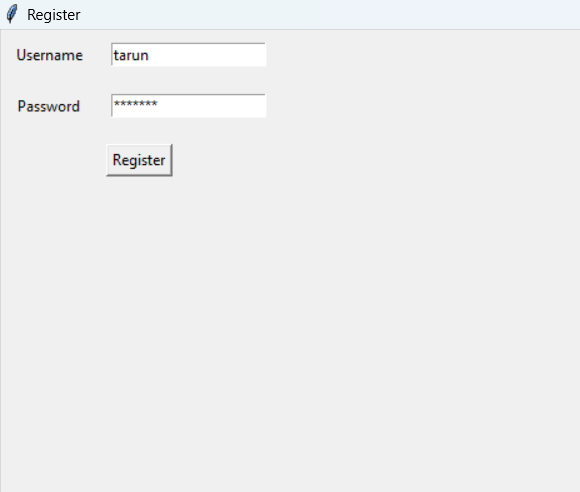
# Start the application by creating the database and the login window

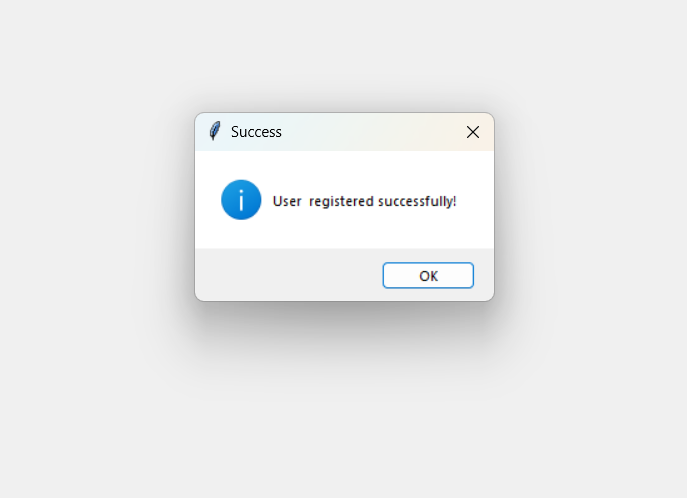
create\_table()

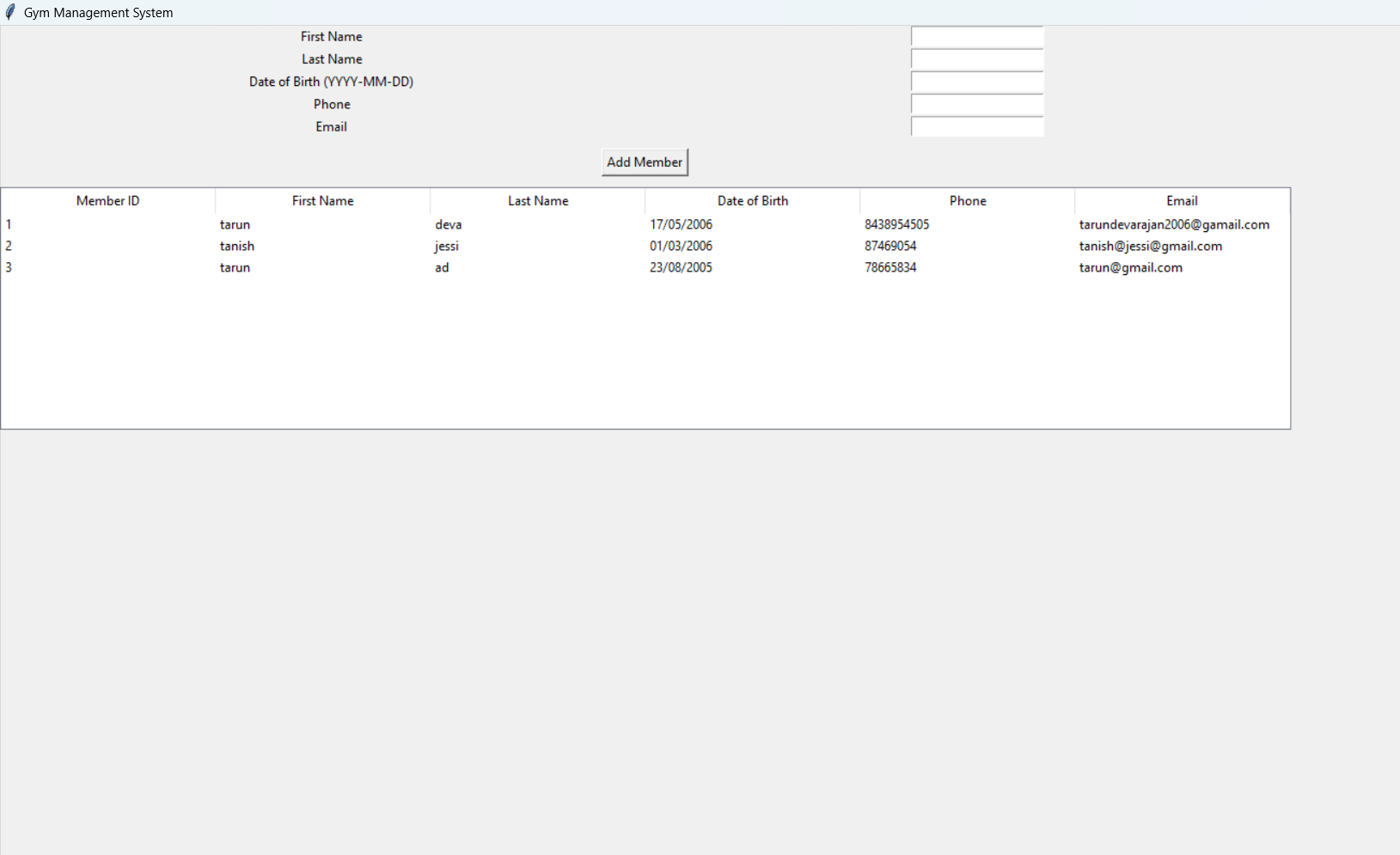
create\_login\_window()

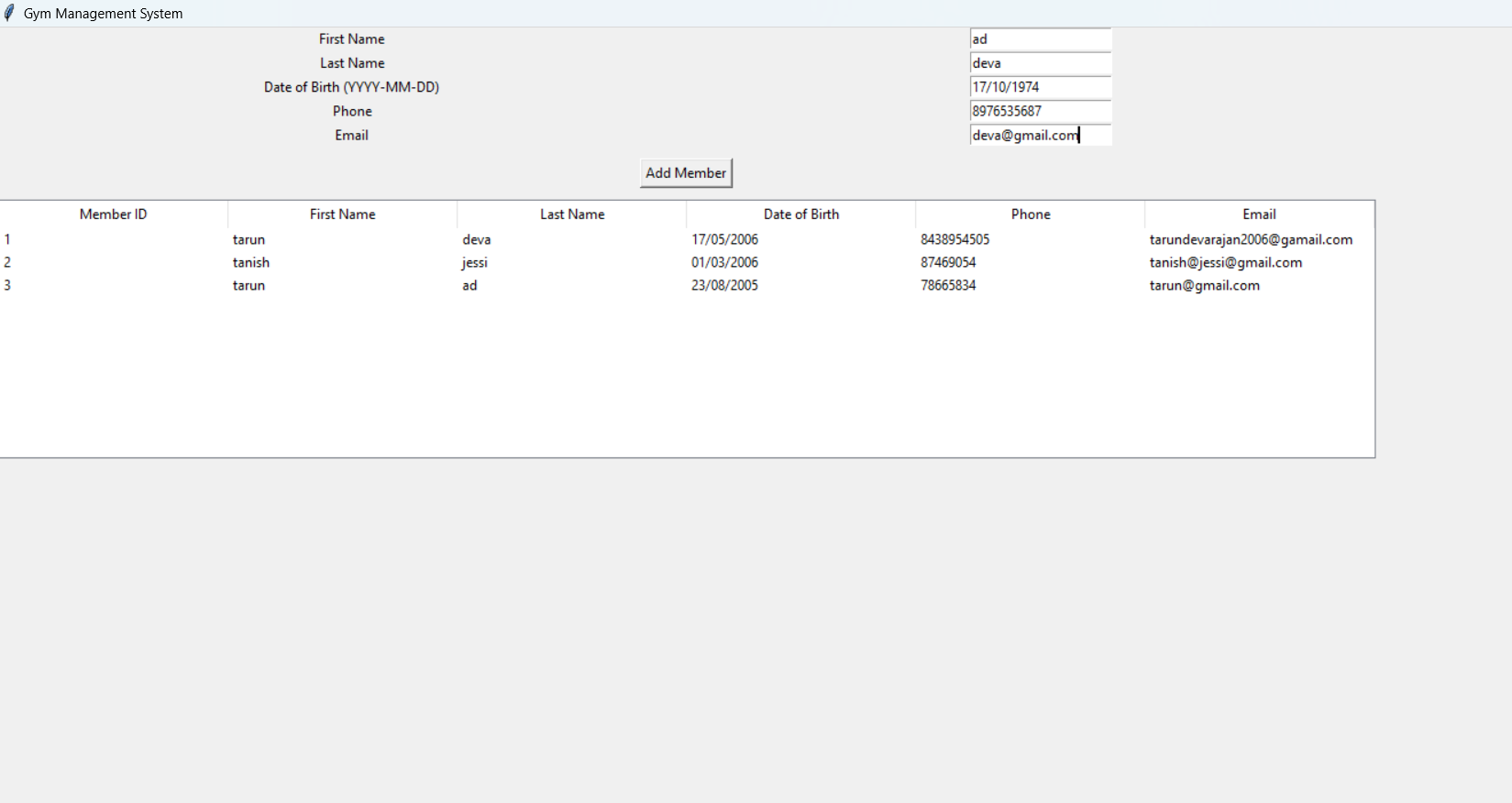
output for this code:

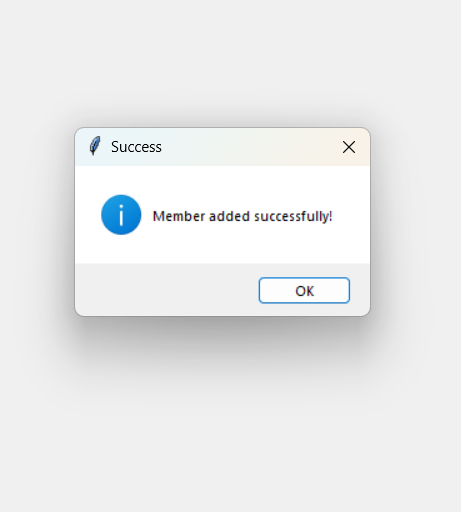


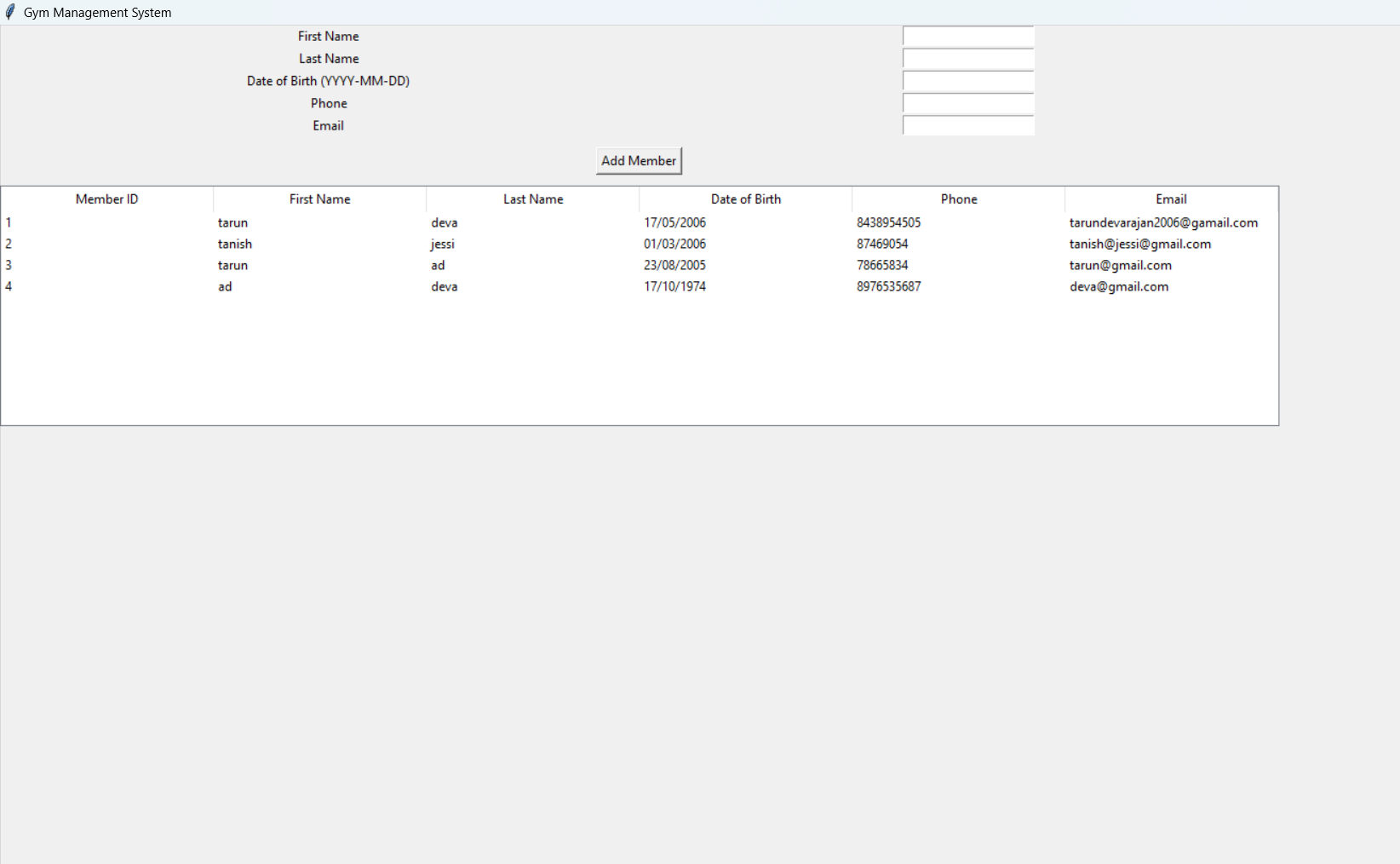












**Result Observation**

**1. User Management Results:**

* **The system successfully implements login and registration functionalities, storing user credentials securely in a SQLite database.**
* **Users can be uniquely identified, and access to the main gym management system is restricted to valid users only.**

**2. Member Management Results:**

* **Members can be added to the database with details such as First Name, Last Name, Date of Birth, Phone, and Email.**
* **The Treeview widget displays members’ information in a tabular format, allowing for easy visualization of the stored data.**

**3. Database Integration:**

* **A SQLite database was used to store both user and member data. This ensures the data remains persistent across sessions.**
* **The database schema includes two main tables:**
  + **Users: For managing system access.**
  + **Members: For storing gym member details.**

**4. Error Handling:**

* **The system gracefully handles duplicate usernames during registration, notifying the user of errors.**
* **Error messages are displayed for failed login attempts or database insertion issues.**

**Limits**

1. **Scalability:**
   * **The current SQLite database may not handle large-scale operations or multiple concurrent users efficiently.**
2. **Feature Limitations:**
   * **Lacks member update, delete, and search functionalities for enhanced user management.**
   * **No password encryption is implemented, leaving user credentials vulnerable.**
3. **UI Complexity:**
   * **User Interface is basic, with room for improvement in usability and design.**
4. **Real-Time Operations:**
   * **The system lacks real-time notifications or advanced analytics for gym members' activities.**

**Future Improvements**

1. **Database Migration:**
   * **Upgrade from SQLite to a relational database like MySQL or PostgreSQL for better performance and scalability.**
2. **Enhanced Security:**
   * **Implement password hashing using libraries like bcrypt for secure storage of user credentials.**
   * **Add role-based access control (e.g., admin and staff roles).**
3. **Advanced Member Management:**
   * **Introduce edit and delete functionalities for member records.**
   * **Add search and filter options to locate specific members.**
4. **Analytics Dashboard:**
   * **Provide visualizations for member statistics such as age distribution, member count trends, etc.**
5. **UI Enhancements:**
   * **Design a more user-friendly and intuitive interface with modern frameworks like Tkinter's ttk or integrate with a web-based frontend.**
6. **Real-Time Features:**
   * **Incorporate notifications for membership renewals or other alerts.**
   * **Add functionality for tracking gym equipment usage or attendance.**

**Testing Approaches**

1. **Functional Testing:**
   * **Verify individual functionalities such as login, registration, and adding members.**
2. **Integration Testing:**
   * **Test seamless interaction between the database and the Tkinter application.**
3. **Performance Testing:**
   * **Measure system responsiveness with large datasets.**
4. **Usability Testing:**
   * **Assess user-friendliness of the interface for end-users.**
5. **Scalability Testing:**
   * **Simulate scenarios with multiple users accessing the system simultaneously.**
6. **Stress Testing:**
   * **Push the system to handle a high volume of member records to identify potential bottlenecks.**

**References**

1. **Official Python Documentation for Tkinter:** [**https://docs.python.org/3/library/tkinter.html**](https://docs.python.org/3/library/tkinter.html)
2. **SQLite Documentation: https://sqlite.org/docs.html**
3. **Effective UI Design Guidelines for Tkinter: https://realpython.com/python-gui-tkinter/**

**Appendices**

**Appendix A: Database Schema**

* **Users Table:**
  + **UserID: Auto-increment primary key.**
  + **Username: Unique identifier for each user.**
  + **Password: Stores user passwords (to be hashed in future versions).**
* **Members Table:**
  + **MemberID: Auto-increment primary key.**
  + **FirstName: First name of the member.**
  + **LastName: Last name of the member.**
  + **DateOfBirth: Date of birth in YYYY-MM-DD format.**
  + **Phone: Contact number.**
  + **Email: Email address.**

**Appendix B: Key Features**

* **Secure login and registration system.**
* **Real-time display of member data using Treeview.**

**Appendix C: Acronyms**

* **GUI: Graphical User Interface.**
* **DBMS: Database Management System.**
* **UAT: User Acceptance Testing.**