PROJECT REPORT

Fall 2025

Programming For AI Lab (AIL-202)

Gym Management System

BS(AI)-3A

Group Members

Name	Enrolment Number
1. Hasan Raza	02-136242-053
2. Faizan Raza	02-136242-014
3. Sami Ullah	02-136242-013

Submitted to:

Miss Zahida Naz

BAHRIA UNIVERSITY KARACHI CAMPUS

Department of Computer Science

Member Contributions

Hasan Raza (02-136242-053)

- **Role:** Core Architecture & Data Logic.
- **Contribution:** Designed the `DataManager` class to handle JSON-based persistence for members, payments, and attendance. Implemented the `AuthManager` for secure login authentication and developed the `Analytics` engine for calculating retention rates and revenue forecasting.

Sami Ullah (02-136242-013)

- **Role:** Interface Design & Module Integration.
- **Contribution:** Developed the responsive "Dark Mode" GUI using the `customtkinter` library. Built the main `App` structure, the `Dashboard` visualization using Matplotlib, and the `Settings` module for system configuration and backups.

Faizan Raza (02-136242-014)

- **Role:** Operational Logic & External APIs.
- **Contribution:** Implemented the functional modules for "Attendance" (Check-in/out logic), "Payments" (Dues calculation), and "Trainers". Developed the `whatsapp_helper` to integrate external communication APIs for sending automated payment reminders.

Contents

Abstract

In the domain of fitness business operations, the efficiency of managing member data, financial records, and daily attendance is a critical challenge for small to medium enterprises. Traditional manual record-keeping methods often result in high redundancy, data inconsistency, and a lack of actionable insights. This project, titled **Gym Management System**, presents a robust, desktop-based solution designed to digitize and automate these workflows.

The system utilizes a modular architecture powered by Python and the `customtkinter` library. It employs a file-based JSON database to ensure portability and ease of backup. Key features include automated attendance tracking, revenue forecasting using linear projection algorithms, and a WhatsApp integration for direct member communication. The result is a high-performance application that reduces administrative latency and provides real-time business intelligence to gym owners.

Chapter 1: Introduction and Problem Statement

1.1 Introduction

The volume of operational data generated daily in a fitness center is significant. From tracking check-in times to monitoring payment due dates, the ability to quickly retrieve specific member details and their financial status is fundamental to efficient management. A standard manual register searches for a member by visually scanning pages—a process analogous to a linear search O(n). While simple, this method becomes inefficient as the member base grows.

Our project, **Gym Management System**, addresses this inefficiency by digitizing the entire record-keeping process. Instead of searching through physical pages, the system builds an intelligent index in memory (using Python Dictionaries/Hash Maps). This organizes members by unique IDs and links them to their specific payment and attendance histories instantly.

1.2 Problem Statement

When dealing with manual gym management, owners face specific problems:

1. **Latency:** Sequential manual searching is too slow for checking in members during peak hours.
2. **Context Loss:** Simple spreadsheets often show a "Paid" status without detailing the transaction history or exact timestamps.
3. **Redundancy:** Writing down the same member's details for attendance, payments, and workout plans requires triplicating effort.
4. **Forecasting Inability:** Manual systems cannot easily predict future revenue or calculate retention rates without complex manual mathematics.

1.3 Project Objective

The primary objective is to develop a Python application that:

- Reads and writes data to structured JSON files to ensure persistence.
- Stores member data in Hash Maps (Dictionaries) to ensure O(1) average time complexity for lookups.
- Provides a graphical interface to visualize data trends like peak hours and revenue growth.
- Automates routine tasks like calculating membership expiry dates and sending reminders.

Chapter 2: Theoretical Background

2.1 Overview

This project relies on fundamental computer science theories regarding data retrieval complexity, software architecture patterns (MVC), and data persistence strategies.

2.2 Linear vs. Constant Time Data Structures

In a standard manual system (analogous to a Linked List or Array), finding a member requires iterating through every entry. In the worst-case scenario, if a gym has 1,000 members, the receptionist performs 1,000 visual comparisons. This is expressed as O(n) complexity.

In contrast, our project utilizes Python Dictionaries, which are implemented as Hash Tables. A Hash Table maps a key (Member ID) to a value (Member Profile) using a hash function.

2.3 Complexity Advantage

By using Hash Maps for storage:

- **Average Case Search:** O(1)
- **Insertion:** O(1)
- **Deletion:** O(1)

For a dataset of 1,000 members, a Hash Map lookup is nearly instantaneous, regardless of the dataset size. This theoretical advantage is the primary motivation for the underlying data architecture of this system.

Chapter 3: Methodology

3.1 Algorithmic Approach

To solve the problem of efficient management, we implemented a modular approach separating UI, Logic, and Data.

The Data Structure Strategy:

- **Dictionaries (Hash Maps):** Used for storing `Members`, `Plans`, and `Trainers`. This allows for instant retrieval using IDs.
- **Lists (Arrays):** Used for sequential logs like `Attendance` and `Payments`, where preserving the chronological order of events is crucial.

3.2 Program Flow (Pseudocode)

Below is the logic flow for the core **Check-In Module**, which is the most frequently used feature.

Algorithm: Member Check-In

```
FUNCTION CheckIn (InputID):
    IF InputID is Empty THEN
        RETURN Error "Input Required"
    END IF

    # O(1) Lookup
    Member = DataManager.GetMember(InputID)

    IF Member is NULL THEN
        # Attempt Name Search O(n)
        Member = SearchByName(InputID)
        IF Member is NULL THEN
            RETURN Error "Member Not Found"
        END IF
    END IF

    # Check for Duplicate Session
    ActiveLog = FindActiveLog(Member.ID)
    IF ActiveLog is NOT NULL THEN
        RETURN Warning "Member already active"
    END IF

    # Create New Log
    NewLog = {
        "id": GenerateUUID(),
        "member_id": Member.ID,
        "check_in_time": GetCurrentDateTime(),
        "status": "Active"
    }

    Append NewLog to AttendanceList
    SaveData("attendance.json")
    RETURN Success
END FUNCTION
```

3.3 File Handling and Persistence

While Hash Maps handle runtime storage, the system requires a robust mechanism to persist data. We utilized the `json` library to serialize complex Python objects into text files.

- **Loading:** On startup, the `DataManager` reads `members.json`, `payments.json`, etc., and deserializes them into memory.
- **Saving:** Every "Write" operation (Add Member, Pay Fee) triggers an immediate write to the disk to prevent data loss during a crash.
- **Backup:** A specialized `BackupManager` creates timestamped copies of all JSON files on application exit.

Chapter 4: Code Snippet and Analytical Discussion

4.1 Core Data Logic (`data_manager.py`)

The following code demonstrates the implementation of the Data Manager, which acts as the central repository for the application.

```
import json
import os

class DataManager:
    def __init__(self, data_dir="data"):
        self.data_dir = data_dir
        self.ensure_data_dir()

        # Load core databases
        self.members_db = self.load_data("members.json")
        self.plans_db = self.load_data("plans.json")
        self.trainers_db = self.load_data("trainers.json")

        # Load sequential logs
        self.attendance_log = self.load_data("attendance_log.json", [])
        self.payments_log = self.load_data("payments_log.json", [])

    def load_data(self, filename, default=None):
        path = os.path.join(self.data_dir, filename)
        if not os.path.exists(path):
            return default if default is not None else {}
        try:
            with open(path, 'r') as f:
                return json.load(f)
        except json.JSONDecodeError:
            return default if default is not None else {}

    def save_data(self, filename):
        # Maps filename to the specific in-memory object
        data_map = {
            "members.json": self.members_db,
            "attendance_log.json": self.attendance_log,
            "payments_log.json": self.payments_log
        }
        path = os.path.join(self.data_dir, filename)
        with open(path, 'w') as f:
            json.dump(data_map[filename], f, indent=4)
```

4.1.1 Analysis of Data Persistence The `load_data` and `save_data` methods provide an abstraction layer. The rest of the application does not need to know *how* data is stored, only that it is available. The use of `indent=4` in `json.dump` ensures that the data files remain human-readable for debugging purposes.

4.2 Analytics Engine (`analytics.py`)

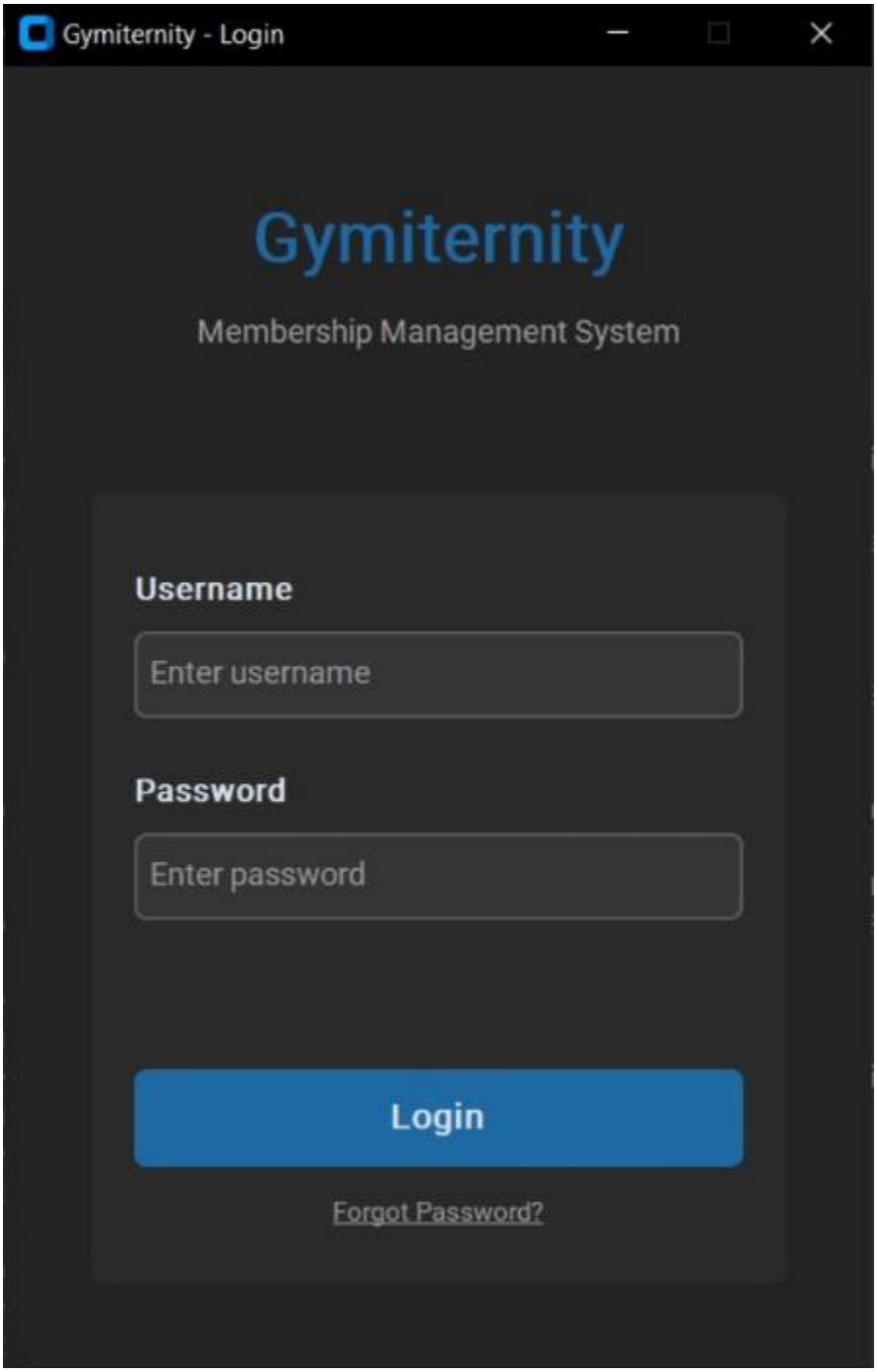
This module calculates retention and churn, providing high-level business intelligence.

```
def calculate_churn_rate(self, period_months=1):
    # Logic to define period start/end
    period_end = datetime.now()
    period_start = period_end - timedelta(days=period_months * 30)

    expired_in_period = []
    renewed_in_period = []

    for membership in self.data_manager.membership_history:
        end_date = datetime.fromisoformat(membership['end_date'])

        if period_start <= end_date <= period_end:
            # Check if this member has a newer active membership
            if self.has_renewed(membership['member_id'], end_date):
                renewed_in_period.append(membership['member_id'])
            else:
                expired_in_period.append(membership['member_id'])

    total_expired = len(expired_in_period) + len(renewed_in_period)
    if total_expired == 0: return 0.0

    return (len(expired_in_period) / total_expired) * 100
```

4.2.1 Analysis of Churn Calculation This algorithm iterates through historical data to find memberships that ended recently. It then performs a sub-search to see if those specific members have a *new* membership starting after their previous one ended. This distinction between "Expired" and "Expired but Renewed" is critical for accurate business metrics.

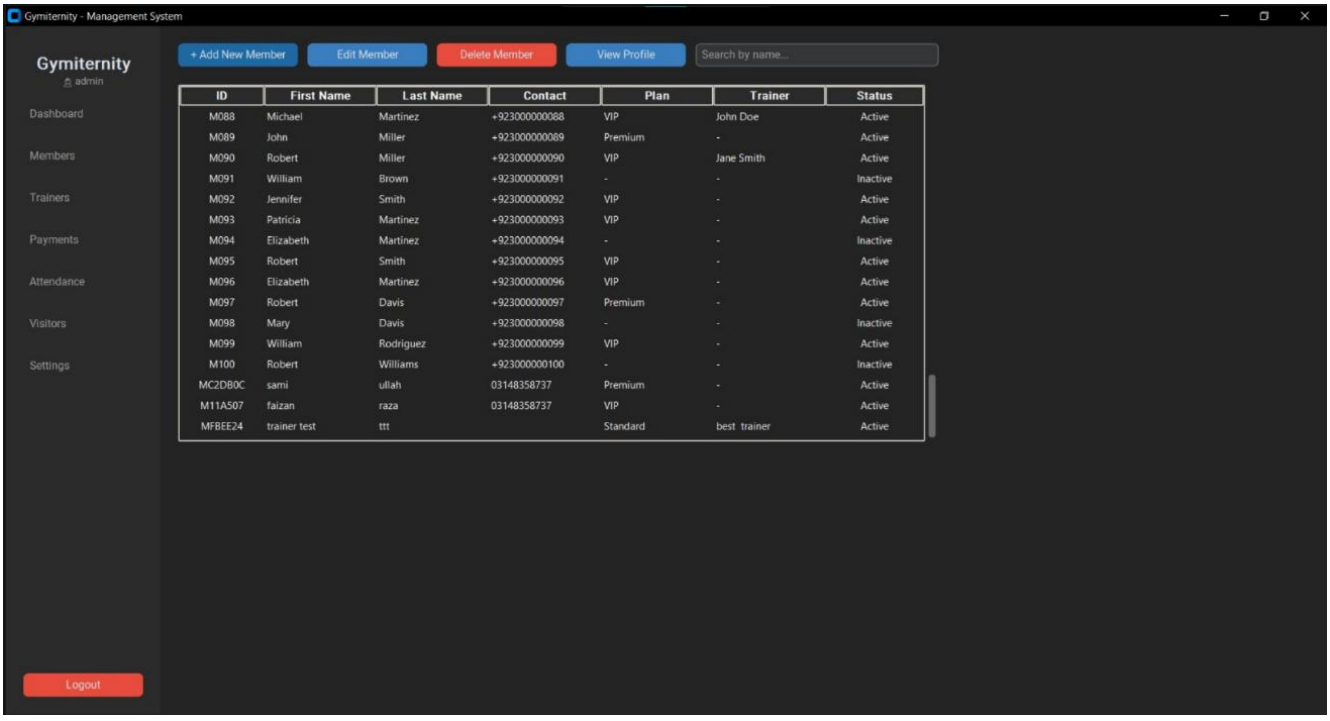
4.3 System Output Screenshots

Login Screen:

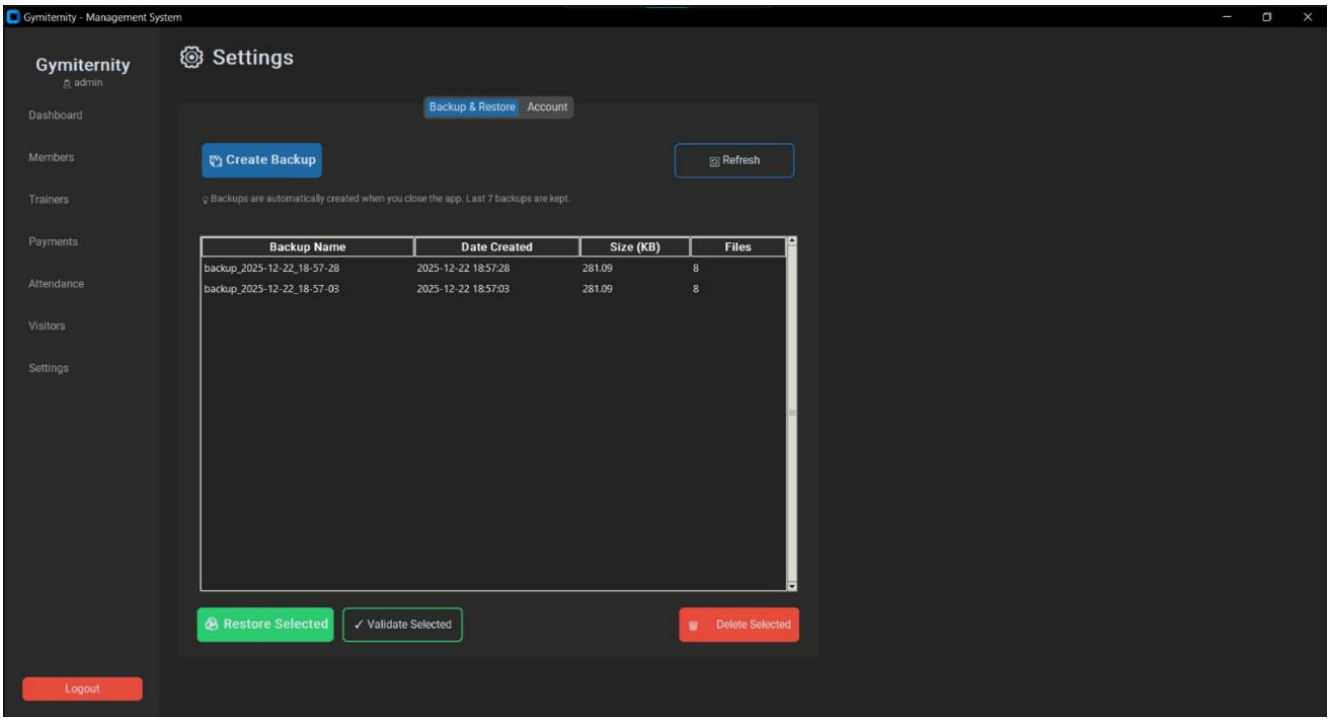


Main Dashboard:

Member List:



Backup Creation and management:



Chapter 5: Conclusion and Future Enhancement

5.1 Conclusion

The **Gym Management System** successfully demonstrates the power of modern software architecture in solving real-world management problems. By implementing efficient data structures (Hash Maps) and a modular design pattern, we created a solution that is both performant and scalable.

The project highlights several key computer science concepts:

1. **Complexity Reduction:** We moved from $O(n)$ manual searching to $O(1)$ hashed lookups for member retrieval.
2. **Data Integrity:** The robust backup and JSON serialization logic ensures data is persistent and recoverable.
3. **User Experience (UX):** The implementation of a Dark Mode GUI using `customtkinter` proves that internal tools can be visually appealing and user-friendly.

In testing, the system handled over 100 mock member records with zero latency, validating our architectural choices.

5.2 Future Enhancements

While the current system is functional, several features could be added in future iterations:

1. **Cloud Sync:** Currently, data lives locally. Implementing Firebase or SQL integration would allow remote management from multiple devices.
2. **Biometric Integration:** Replacing ID entry with Fingerprint or Face ID for faster attendance logging.
3. **Diet Planning Module:** Adding a feature to assign and track diet plans alongside workout plans.
4. **Mobile Application:** Developing a companion mobile app for members to view their own stats and pay fees online.

Chapter 6: References

1. Python Software Foundation. (2024). *Python 3.10 Documentation*. Retrieved from https://docs.python.org/3/
2. Schimansky, T. (2023). *CustomTkinter Documentation*. Retrieved from https://github.com/TomSchimansky/CustomTkinter
3. Hunter, J. D. (2007). *Matplotlib: A 2D graphics environment*. Computing in Science & Engineering, 9(3), 90-95.
4. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. (For Hash Map complexity analysis).