ASSIGNMENT COVERSHEET

UTS: ENGINEERING & INFORMATION TECHNOLOGY

SUBJECT NUMBER & NAME 31927 Application Development with .NET	**NAME OF STUDENT(s)** (PRINT CLEARLY) *Tran Ron* *Tran - Khoi Nguyen* *SURNAME* *FIRST NAME*	**STUDENT ID(s)** 24704862 25114442

STUDENT EMAIL PhamAnhVu.Tran@student.uts.edu.au KhoiNguyen.Tran@student.uts.edu.au	**STUDENT CONTACT NUMBER** 0406942463 0410961218

NAME OF TUTOR Zohair Gandhi	**TUTORIAL GROUP** Tutorial 6	**DUE DATE** 28/10/2024

ASSESSMENT ITEM NUMBER & TITLE
Assignment 2 Group Project

☐ I acknowledge that if AI or another nonrecoverable source was used to generate materials for background research and self-study in producing this assignment, I have checked and verified the accuracy and integrity of the information used.
☐ I confirm that I have read, understood and followed the guidelines for assignment submission and presentation on page 2 of this cover sheet.
☐ I confirm that I have read, understood and followed the advice in the Subject Outline about assessment requirements.
☐ I understand that if this assignment is submitted after the due date it may incur a penalty for lateness unless I have previously had an extension of time approved and have attached the written confirmation of this extension.

Declaration of originality: The work contained in this assignment, other than that specifically attributed to another source, is that of the author(s) and has not been previously submitted for assessment. I have rewritten any material provided by AI or other nonrecoverable sources and where appropriate acknowledged their contribution. I understand that, should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with University policy and rules. In the statement below, I have indicated the extent to which I have collaborated with others, whom I have named.

No content generated by AI technologies or other sources has been presented as my own work and I have rewritten any text provided by AI or other sources in my own words.

Statement of collaboration:
Signature of student(s) Ron Tran Khoi Nguyen-Tran **Date** 26/10/2024

ASSIGNMENT RECEIPT

To be completed by the student if a receipt is required

SUBJECT NUMBER & NAME	**NAME OF TUTOR**
SIGNATURE OF TUTOR	**RECEIVED DATE**

STYLE GUIDE for ASSIGNMENT SUBMISSION

Before submitting an assignment, you should refer to the policies and guidelines set out in the following:

- FEIT Student Guide
- UTS Library - referencing
- HELPS - English and academic literacy support
- UTS GSU - coursework assessment policy and procedures

Unless your Subject Coordinator has indicated otherwise in the Subject Outline, you must follow the instructions below for submission of assignments in the Faculty of Engineering and Information Technology.

Writing style

It is usually best to write your initial draft in the default settings of your software without formatting. Use the following guides in your writing.

Purpose and audience: use the correct genre and language style expected for the particular task.

Language: use 'plain English' for all technical writing. More information about this language style can be found at www.plainenglish.co.uk/free-guides.html.

Use spelling and grammar software tools to check your writing. Edit your document.

Standards: always use:

- Australian spelling standards (Macquarie Dictionary)
- SI (International System of Units) units of measurement
- ISO (International Organisation for Standardisation) for writing dates and times for international documents. For example **yyyy-mm-dd** or **hh-mm-ss**. However, for most applications it is more helpful to present the date in full as **26 August 2016**.

Graphics and tables should:

- be numbered
- have an appropriate heading and/or caption
- be fully labelled
- be correctly referenced.

Presentation

Unless otherwise instructed, all assignment submissions should be **word processed** using spell-check and grammar-check software. Work should be well **edited** before submission. Use the following default settings:

Page setup: set margins at no less than 20mm all around.
Paper: print on A4 bond, double-spaced and preferably double-sided, left justified.

Font: use the software default style to provide consistency. The recommended style includes:

- 10-12 pt font
- consistent formatting with a limited number of fonts
- lines no more than 60 characters (use wider margins or columns if you need to make lines shorter)

Header should include:

- your name and student number
- the title of the paper or task.

Footer should include the page number and current date.

Cover sheet and statement of originality: all work submitted for assessment must be the original work of the student(s) submitting the work. A standard faculty cover sheet (see over) must be attached to the front of the submission. Any collaboration between the submitting student and others must be declared on the cover sheet.

Referencing

All sources of information used in the preparation of your submission must be acknowledged using the APA system of referencing. This includes all print, video, electronic sources.

Phrases, sentences or paragraphs taken verbatim from a source must be in quotation marks and the source(s) cited using both **in-text** referencing and a **reference list**.

Plagiarism is the failure to acknowledge sources of information. You should be fully aware of the meaning of plagiarism and its consequences both to your marks, position at the university and criminal liability. The plagiarism in your assignment submissions can be assessed both in hard copy and in soft copy through software such as Turnitin.

The UTS Library and UTS HELPS (web links above) provide extensive information for students on referencing correctly to support you in avoiding plagiarism.

31927 - Application Development with .NET
Assignment 2 Report
Gym Membership System

Ron Tran - 24704862
Khoi Nguyen - Tran - 25114442

Table of Contents

Project Idea

The goal of this project is to provide a complete gym membership management system that will improve member satisfaction and expedite operations. The system will provide an all-in-one solution for members and staff by handling membership processing, session scheduling, progress tracking, rewards management, and administrative duties. The system provides new users with an easy-to-use registration interface that lets them join the gym and choose the membership option that best suits their needs. Through an intuitive interface, current members can quickly change their membership levels, schedule personal training sessions, and monitor their fitness progress. The session booking system is a crucial component that minimises scheduling conflicts and improves convenience by enabling members to monitor trainer availability and choose time slots that work for them. Members can create personal goals, visualise their trip, and report their fitness statistics with the progress tracking tool, which keeps them accountable and inspired. Furthermore, by enabling members to accrue points through actions like attending sessions and referring friends, a points-based rewards system encourages participation and may be exchanged for benefits like products or discounts. An administrative interface helps gym employees manage trainer schedules, member profiles, and session reservations, which decreases manual labour and boosts productivity. In order to assist data-driven decisions to enhance services and maximise gym operations, the system also produces reports and analytics on important parameters. The overall goal of this project is to develop a strong, integrated system that improves the gym experience for patrons while making management duties easier for employees.

Motivation

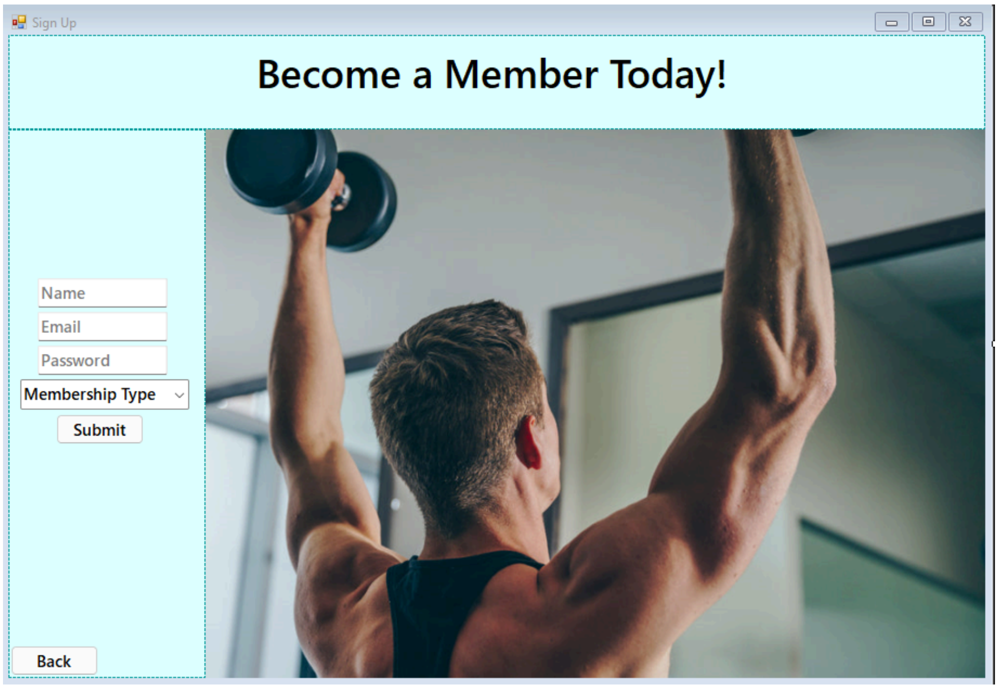
Many gyms suffer from outdated, manual procedures for handling memberships, scheduling sessions, and monitoring member progress, despite the increased desire for effective and customised gym experiences. The need to modernise these procedures is what spurred this project, which offers a system that automates repetitive work, lessens the administrative load on gym employees, and improves the general experience for members. The solution facilitates a more seamless and adaptable experience for members by simplifying membership administration and session scheduling, which helps them better manage their timetables and fitness objectives. Incorporating progress tracking also gives members motivating tools that promote regular involvement and goal achievement. An admin page streamlines essential tasks for gym employees, giving them effective tools to handle member profiles, trainer schedules, and session reservations. In general, the goal of this project is to close the gap between service and demand by developing a modernised gym management system that benefits employees and members alike by promoting a more stimulating, effective, and engaging atmosphere.

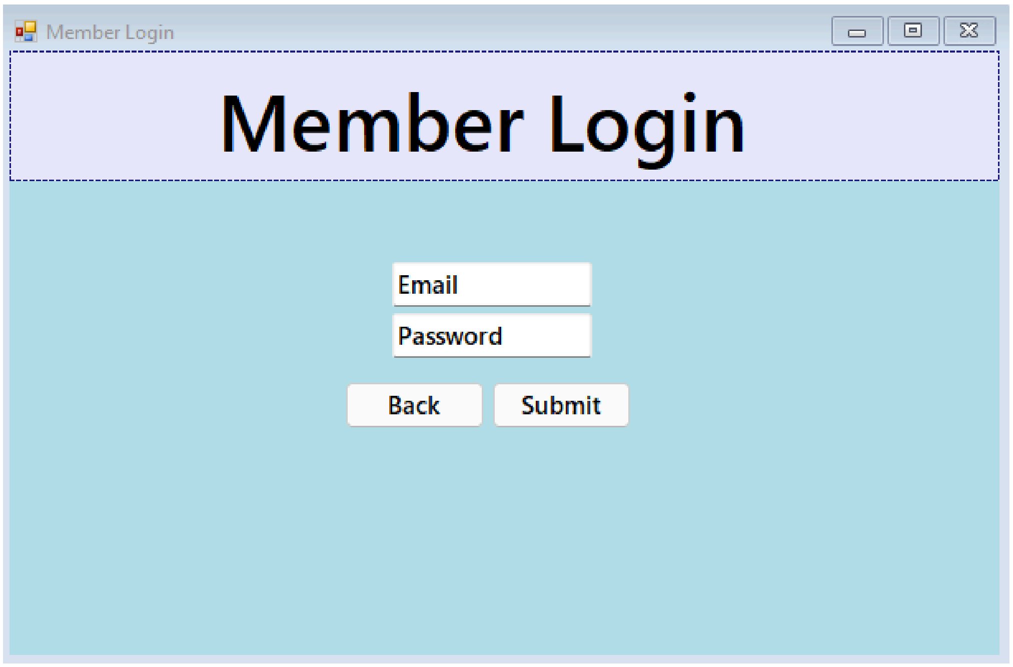
Key Features

Membership Sign-Up and Log In

The registration screen allows new users to quickly create an account by filling in a few essential details. This information is used to personalise each user’s membership experience and ensure smooth access to the gym's services, collecting their name, email, password, and membership choice.

For returning users, the login screen offers a straightforward way to access their accounts. By entering their registered email and password, members can securely log in and access their personalised gym profiles. This feature ensures that only authorised members can access account-specific features, preserving user privacy and security.

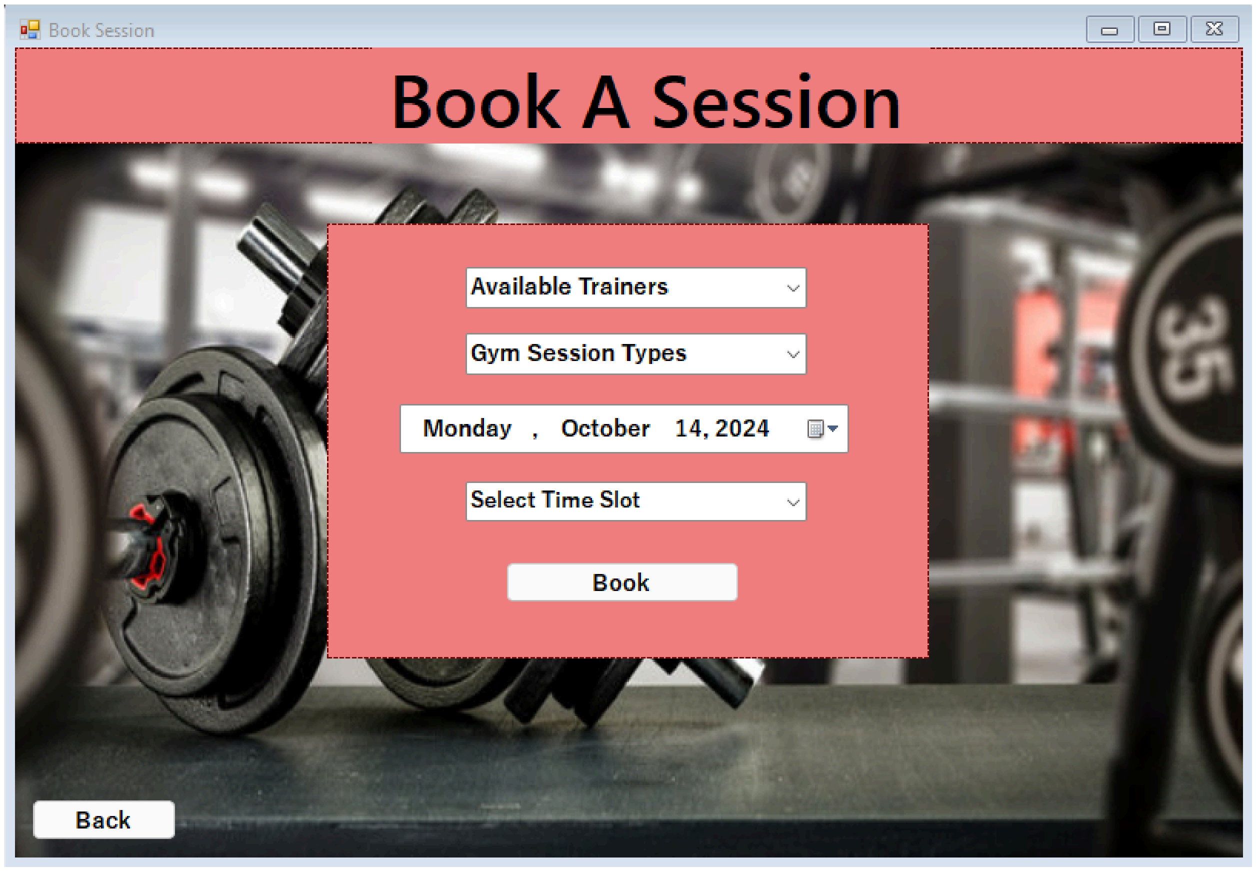




Session Booking System
The session booking system provides a convenient way for gym members to schedule appointments with personal trainers directly through the platform. This feature simplifies the booking process, allowing members to manage their training sessions without needing in-person or phone scheduling.

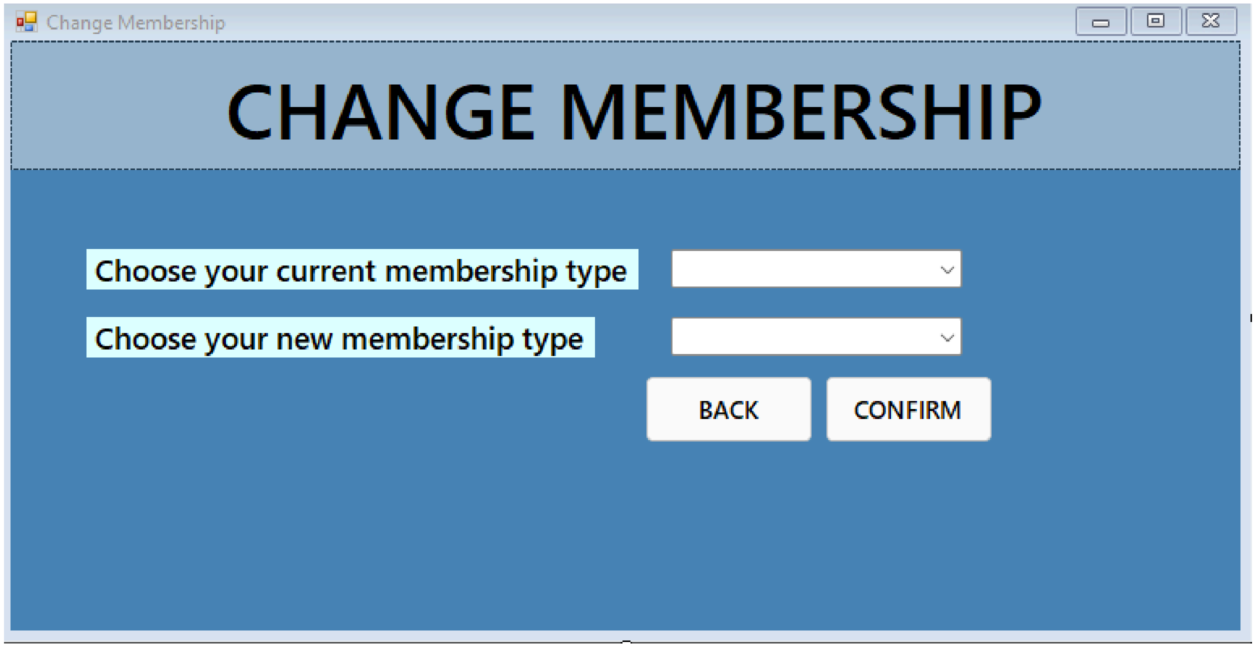
- Allows members to book sessions with personal trainers directly through the system.

- Enabling members to choose time slots that align with their schedules.



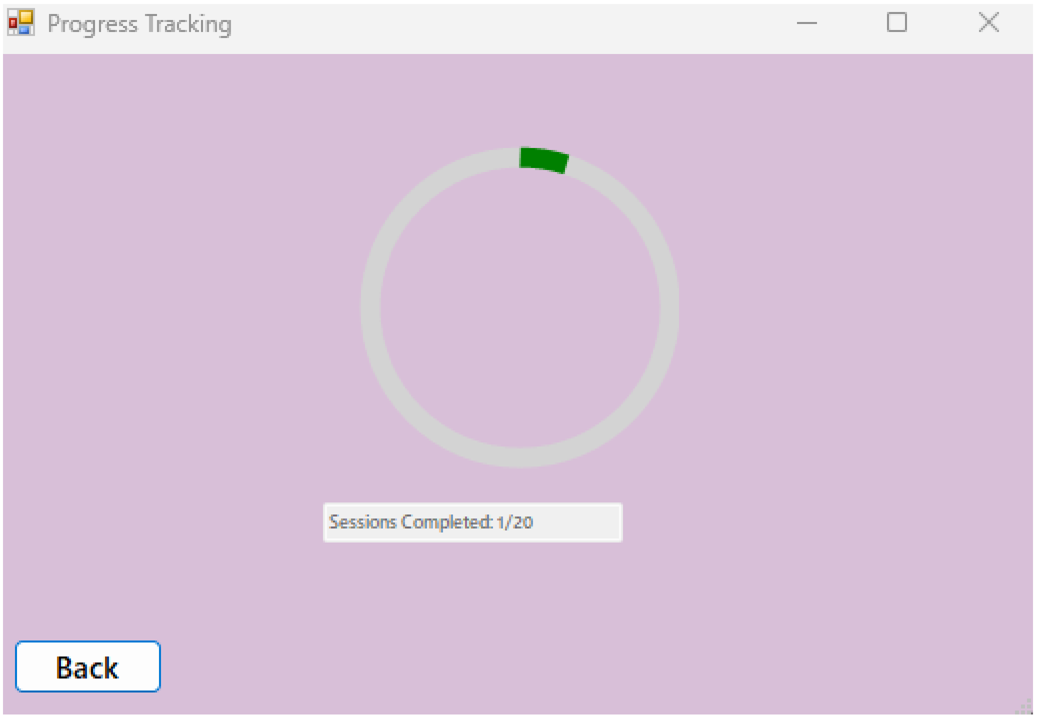
Membership Management

- Enables members to modify their membership types, allowing easy upgrades or downgrades.


Change Membership
CHANGE MEMBERSHIP
Choose your current membership type
Choose your new membership type
BACK
CONFIRM

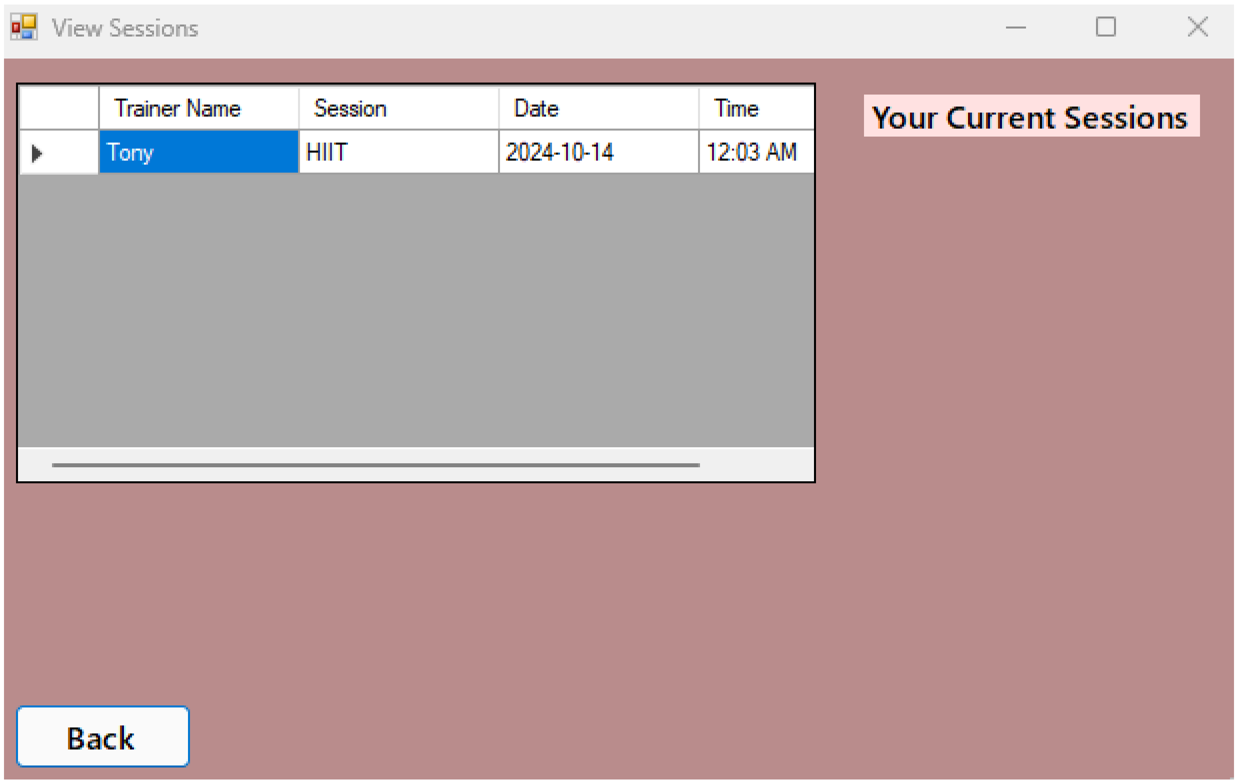
Progress Tracking

- Members can track their progress on the sessions they have attended, noting milestones and areas for improvement.
- A user-friendly dashboard lets members view their progress, motivating them to stay consistent with their fitness routines.



View booked sessions

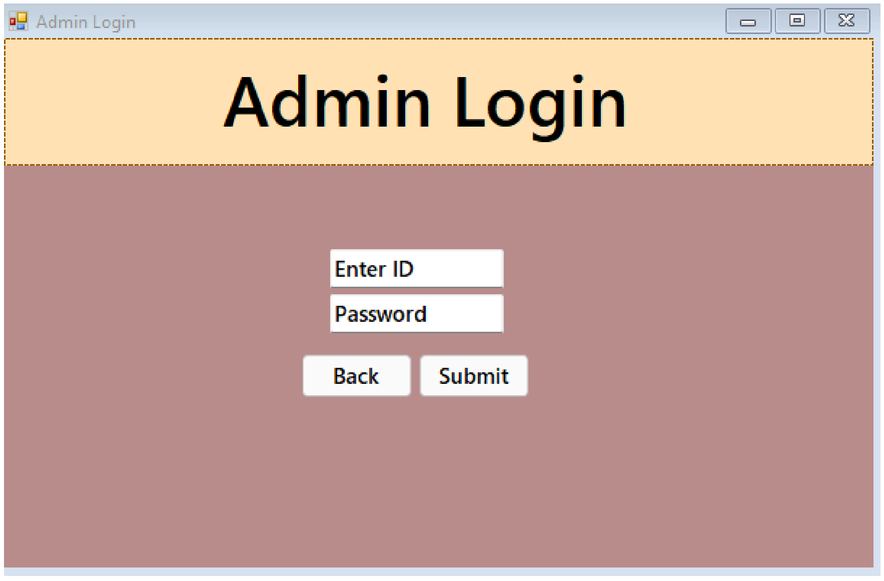
- Members can view their current booked sessions to keep up with the schedule of each session and know who they are training with and what they will be training on for that session.

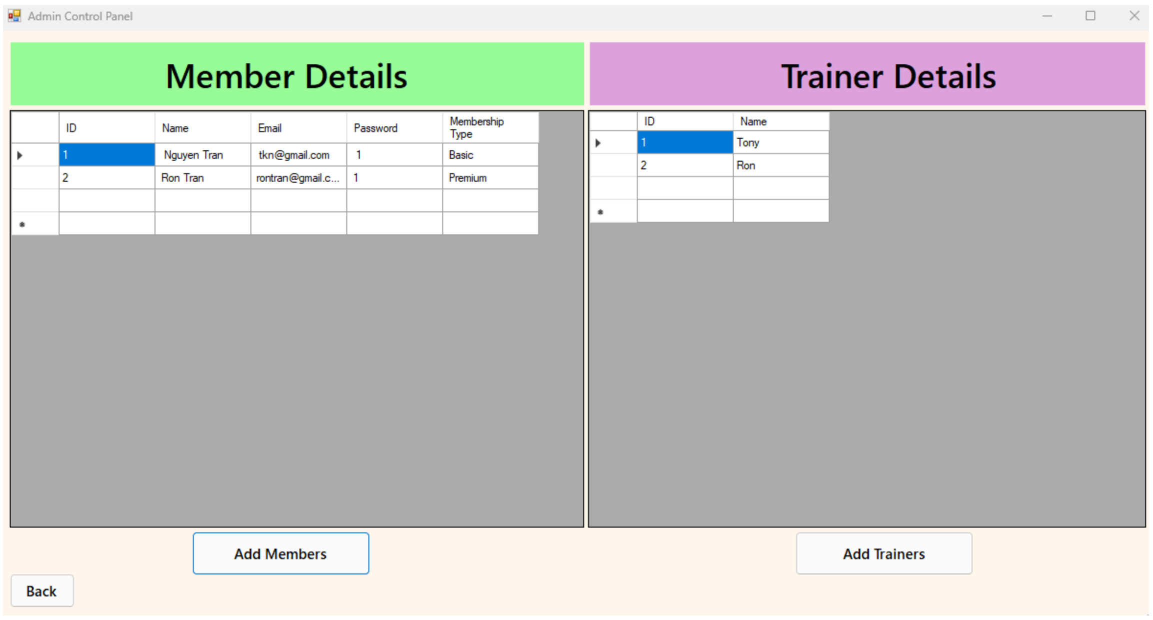


	Trainer Name	Session	Date	Time
▶	Tony	HIIT	2024-10-14	12:03 AM

Admin Page

- The admin page gives gym staff centralised control to manage a database of members and trainers.
- Staff can add new members and trainers, update their information, and monitor membership activity.
- Staff can access reports and analytics on member engagement and trainer utilisation, helping the gym make data-driven decisions.

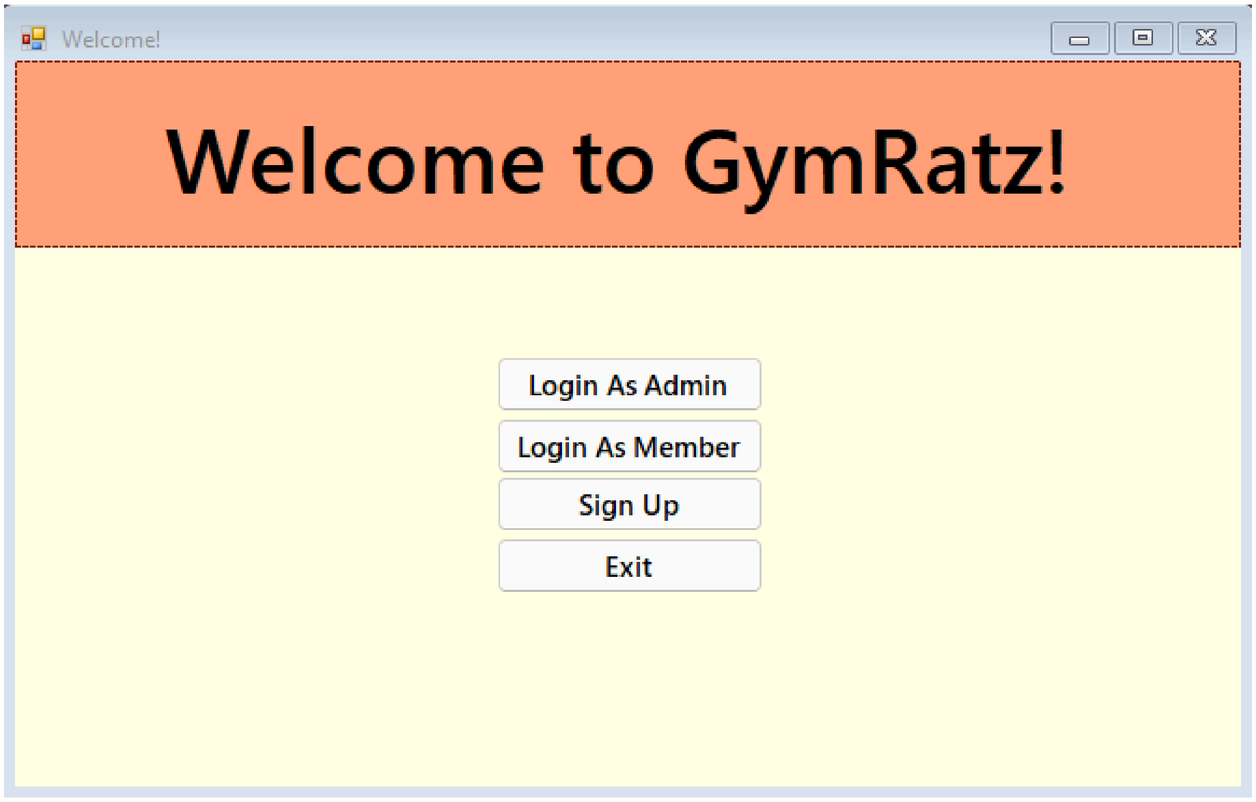




	ID	Name	Email	Password	Membership Type
▸	1	Nguyen Tran	tkn@gmail.com	1	Basic
	2	Ron Tran	rontran@gmail.c...	1	Premium
*					

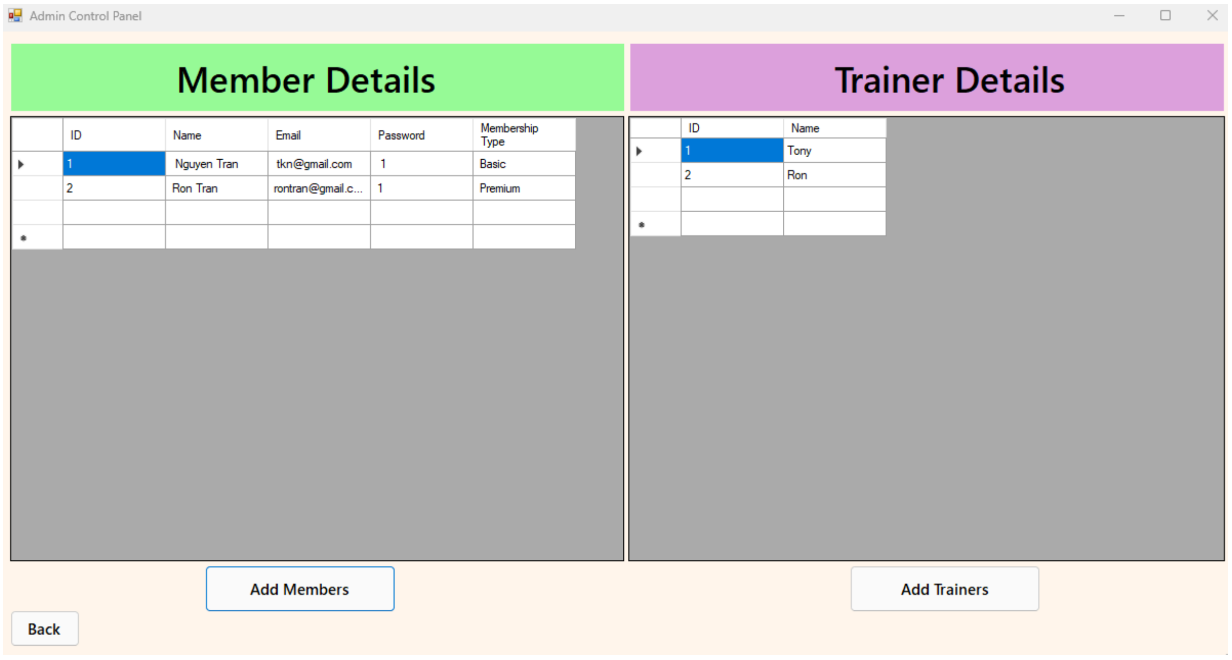
	ID	Name
▸	1	Tony
	2	Ron
*		

Usage Instructions



As you first open the app, there are four options for you to either log in as an admin or member, sign up to become a member or exit the app entirely.

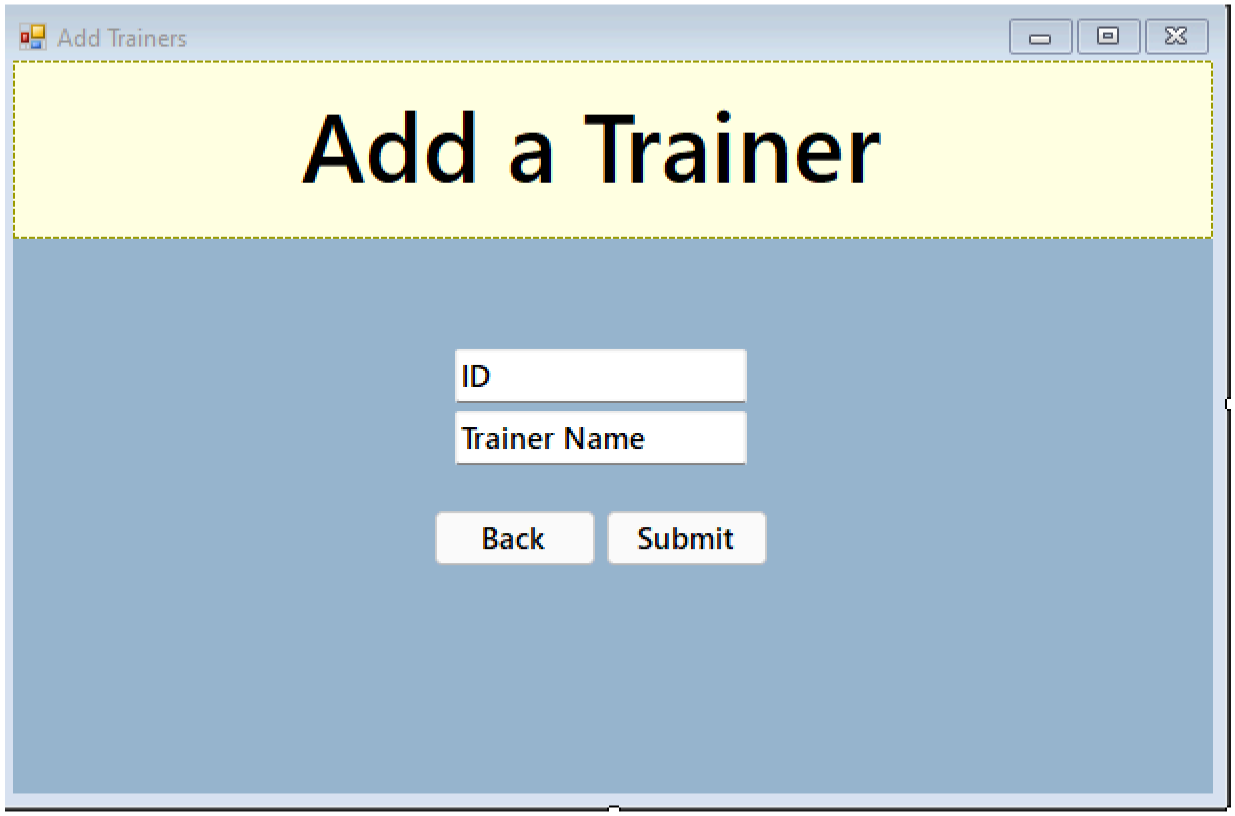
For Admin/Staff:



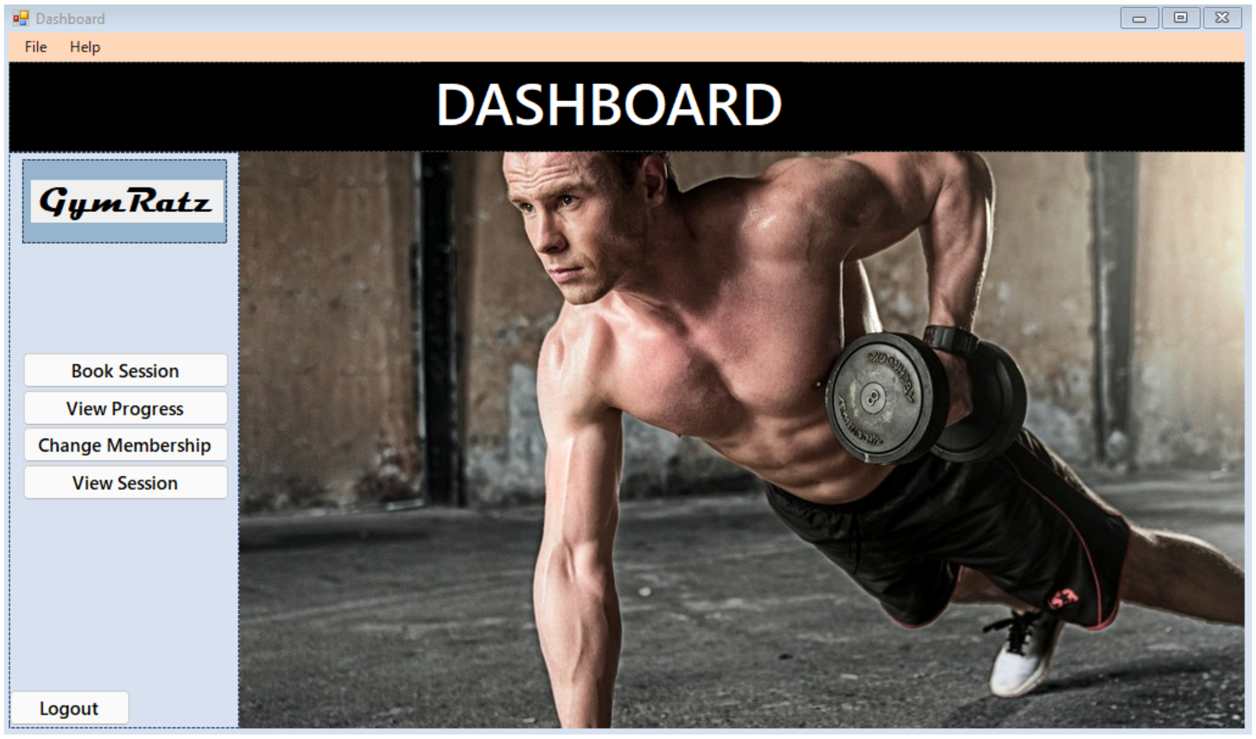
ID	Name	Email	Password	Membership Type
1	Nguyen Tran	tkn@gmail.com	1	Basic
2	Ron Tran	rontran@gmail.c...	1	Premium

ID	Name
1	Tony
2	Ron

Staff can use this dashboard to view data on member and trainer details. Clicking on "Add Member" will lead to the sign-up page for members to fill in new member details. Clicking on "Add Trainers" will lead to a form to fill in a trainer's name and ID.



For Members:

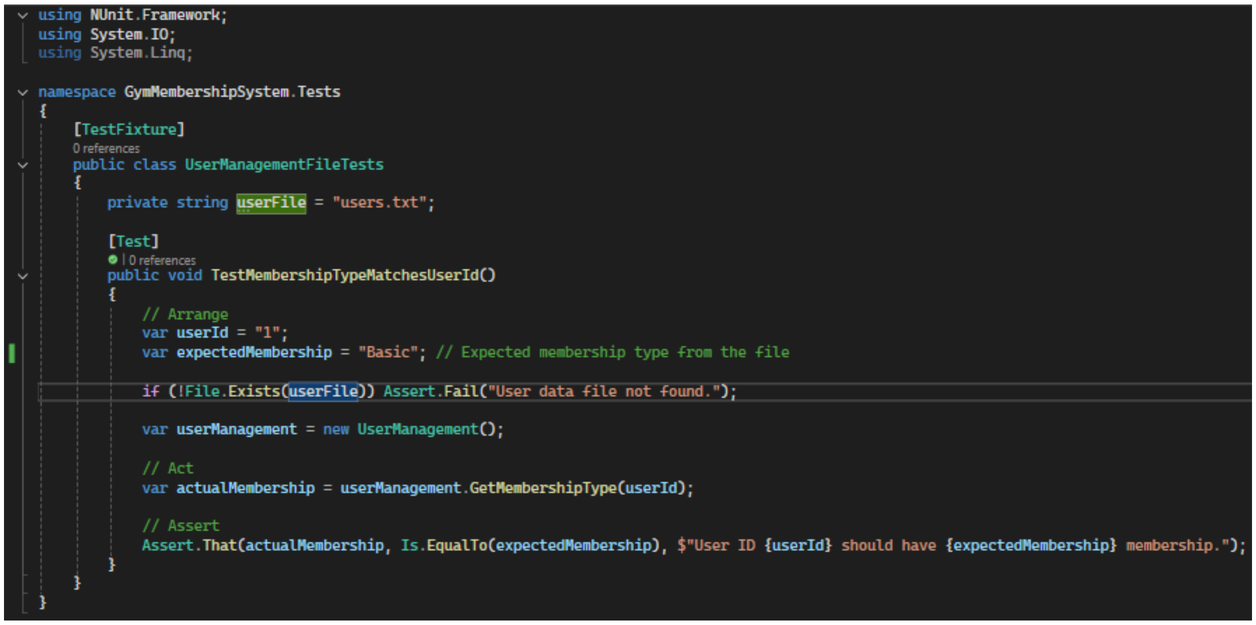


Once the member has logged in to the main dashboard, they can book a session, view progress, change membership, view the session they have signed up for, as mentioned in key features, or log out.

Unit Testing:

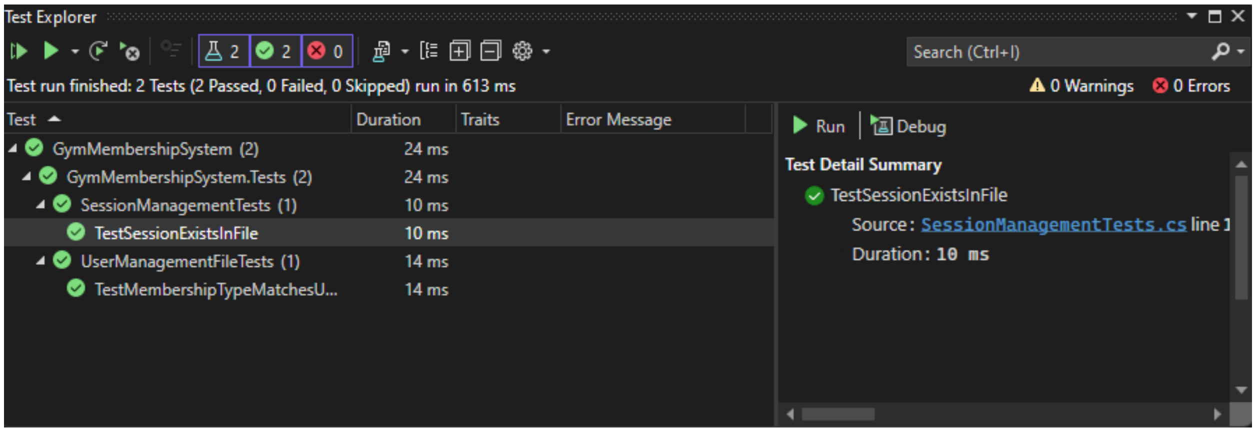
User Management Tests:

This method tests if the membership type retrieved for a specific user ID matches the expected membership type defined in a data file.



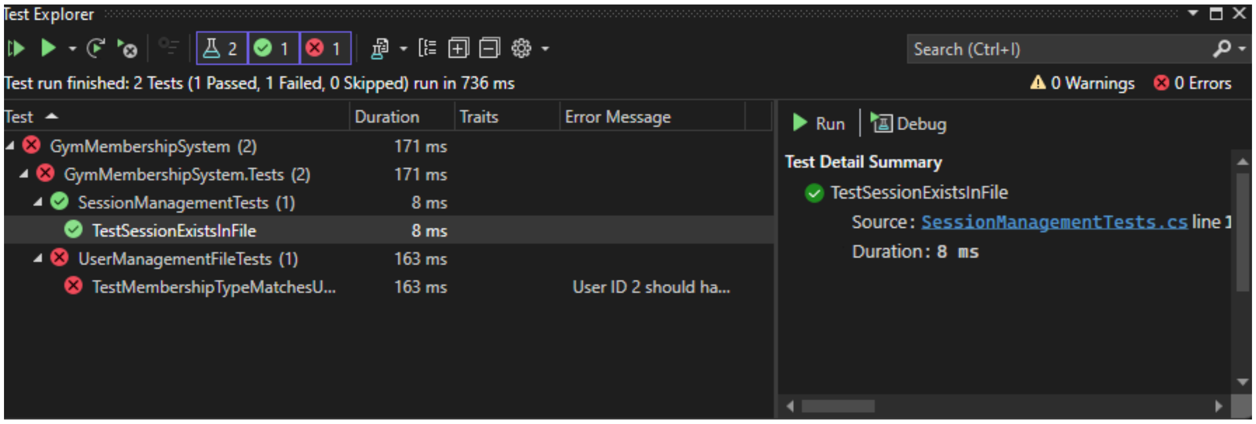
```
using NUnit.Framework;
using System.IO;
using System.Linq;

namespace GymMembershipSystem.Tests
{
    [TestFixture]
    0 references
    public class UserManagementFileTests
    {
        private string userFile = "users.txt";

        [Test]
        ● | 0 references
        public void TestMembershipTypeMatchesUserId()
        {
            // Arrange
            var userId = "1";
            var expectedMembership = "Basic"; // Expected membership type from the file

            if (!File.Exists(userFile)) Assert.Fail("User data file not found.");

            var userManagement = new UserManagement();

            // Act
            var actualMembership = userManagement.GetMembershipType(userId);

            // Assert
            Assert.That(actualMembership, Is.EqualTo(expectedMembership), $"User ID {userId} should have {expectedMembership} membership.");
        }
    }
}
```

1. **Purpose**:
   The test verifies that the system correctly retrieves the membership type for a user based on their ID by checking against an expected value. This is crucial to ensure users are assigned the correct membership level.
2. **Arrange**:
   - A sample `userId` (set to `"1"`) and an `expectedMembership` value (set to `"Basic"`) are defined. These represent the ID of the user being tested and the expected membership type associated with that user in the data file.
   - The `userFile` variable points to the data file (`"users.txt"`), which is expected to contain user data.
3. **Act**:
   - The test checks if the `userFile` exists. If it doesn't, the test fails with the message `"User data file not found."`
   - If the file exists, it instantiates an object of `UserManagement` (a class responsible for managing user data) and retrieves the actual membership type for `userId` by calling `GetMembershipType(userId)`.
4. **Assert**:
   - It compares the retrieved `actualMembership` with `expectedMembership`. If they match, the test passes, confirming the system correctly identifies the user's membership type. If they don't match, it fails, with a message indicating the expected membership for that user ID.



```
Test Explorer
Test run finished: 2 Tests (2 Passed, 0 Failed, 0 Skipped) run in 613 ms        0 Warnings   0 Errors
Test                                    Duration  Traits  Error Message
GymMembershipSystem (2)                   24 ms
  GymMembershipSystem.Tests (2)           24 ms
    SessionManagementTests (1)            10 ms
      TestSessionExistsInFile             10 ms
    UserManagementFileTests (1)           14 ms
      TestMembershipTypeMatchesU...       14 ms

Test Detail Summary
  TestSessionExistsInFile
    Source: SessionManagementTests.cs line 1
    Duration: 10 ms
```

If the data matches the test case, the test case will pass.

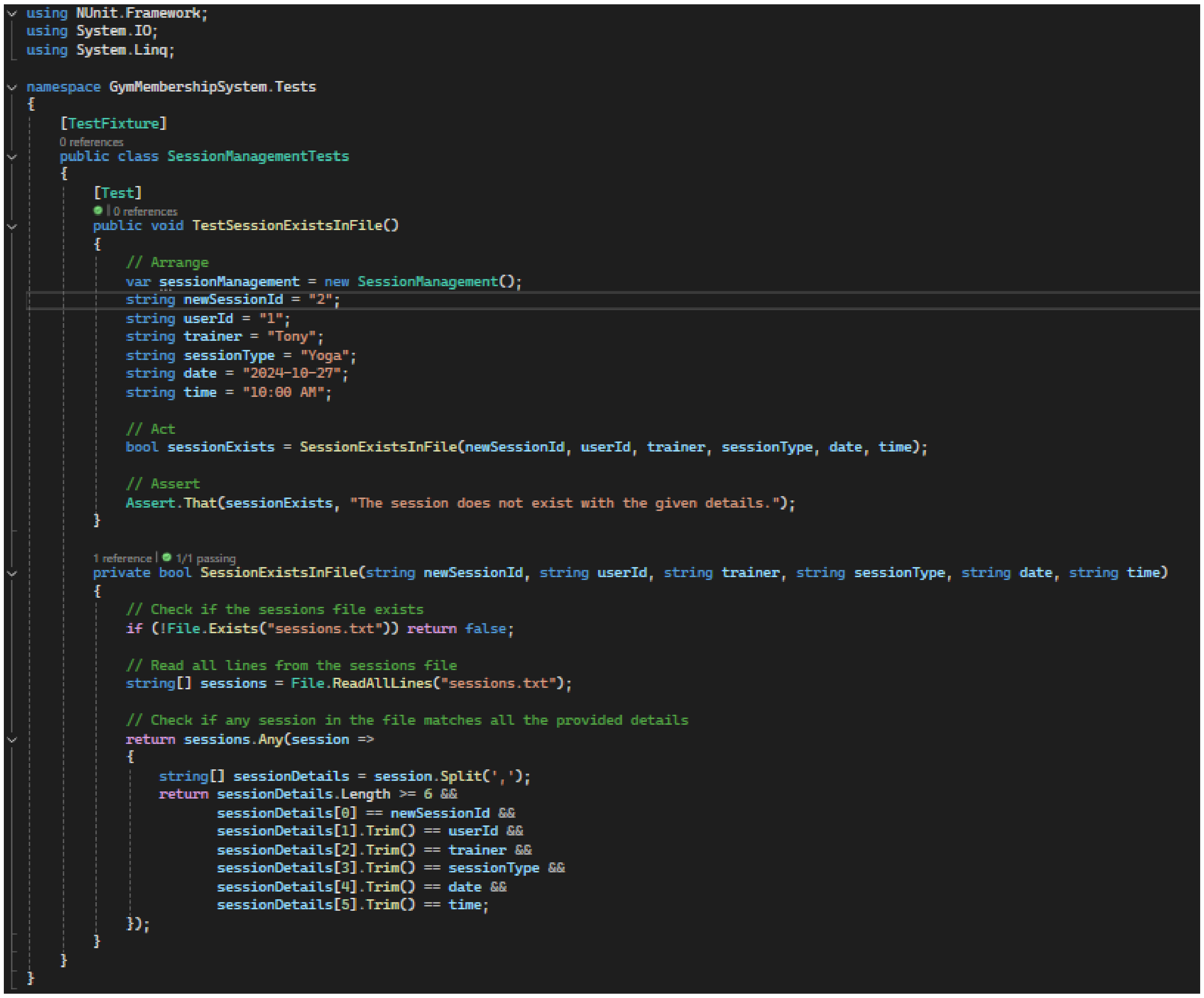


```
Test Explorer
Test run finished: 2 Tests (1 Passed, 1 Failed, 0 Skipped) run in 736 ms        0 Warnings   0 Errors
Test                                    Duration  Traits  Error Message
GymMembershipSystem (2)                  171 ms
  GymMembershipSystem.Tests (2)          171 ms
    SessionManagementTests (1)             8 ms
      TestSessionExistsInFile              8 ms
    UserManagementFileTests (1)          163 ms
      TestMembershipTypeMatchesU...      163 ms           User ID 2 should ha...

Test Detail Summary
  TestSessionExistsInFile
    Source: SessionManagementTests.cs line 1
    Duration: 8 ms
```

If the userid or expectedmembership is changed, the test will fail.

Session Management Tests:

This test method checks whether a gym session with specific details is correctly recorded in a data file.



```
using NUnit.Framework;
using System.IO;
using System.Linq;

namespace GymMembershipSystem.Tests
{
    [TestFixture]
    0 references
    public class SessionManagementTests
    {
        [Test]
        ● | 0 references
        public void TestSessionExistsInFile()
        {
            // Arrange
            var sessionManagement = new SessionManagement();
            string newSessionId = "2";
            string userId = "1";
            string trainer = "Tony";
            string sessionType = "Yoga";
            string date = "2024-10-27";
            string time = "10:00 AM";

            // Act
            bool sessionExists = SessionExistsInFile(newSessionId, userId, trainer, sessionType, date, time);

            // Assert
            Assert.That(sessionExists, "The session does not exist with the given details.");
        }

        1 reference | ● 1/1 passing
        private bool SessionExistsInFile(string newSessionId, string userId, string trainer, string sessionType, string date, string time)
        {
            // Check if the sessions file exists
            if (!File.Exists("sessions.txt")) return false;

            // Read all lines from the sessions file
            string[] sessions = File.ReadAllLines("sessions.txt");

            // Check if any session in the file matches all the provided details
            return sessions.Any(session =>
            {
                string[] sessionDetails = session.Split(',');
                return sessionDetails.Length >= 6 &&
                       sessionDetails[0] == newSessionId &&
                       sessionDetails[1].Trim() == userId &&
                       sessionDetails[2].Trim() == trainer &&
                       sessionDetails[3].Trim() == sessionType &&
                       sessionDetails[4].Trim() == date &&
                       sessionDetails[5].Trim() == time;
            });
        }
    }
}
```

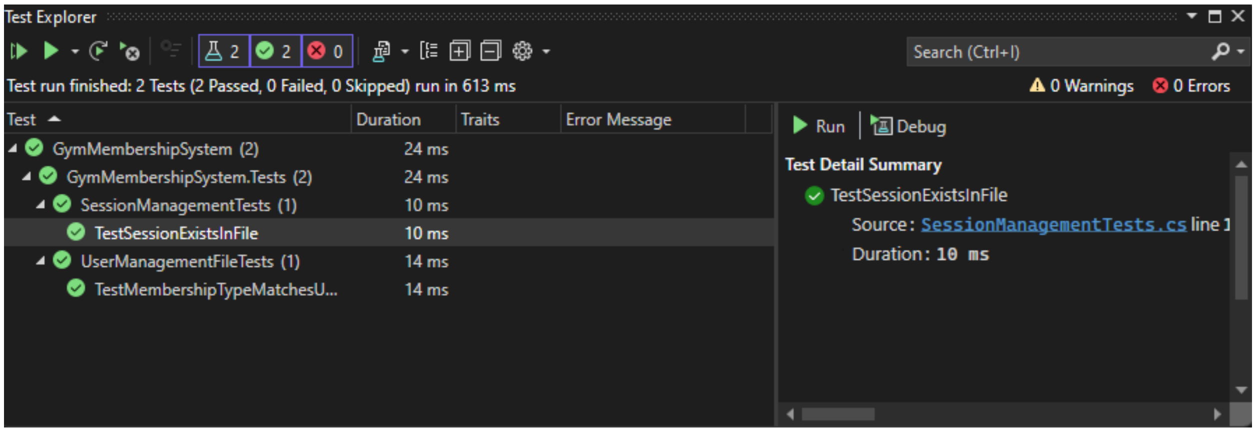
1. **Purpose**:
   The test verifies that the system can find a particular session based on multiple attributes such as session ID, user ID, trainer, session type, date, and time. This helps ensure that session records are accurate and can be reliably queried from the file.
2. **Arrange**:
   - A new instance of `SessionManagement` is created.
   - Sample data representing a session is defined as:
     - `newSessionId`: "2"
     - `userId`: "1"
     - `trainer`: "Tony"
     - `sessionType`: "Yoga"
     - `date`: "2024-10-27"
     - `time`: "10:00 AM"

3. **Act**:
    - The method `SessionExistsInFile` is called with the specified session details. This method checks if a session matching all these attributes exists in the `sessions.txt` file.
    - The result, `sessionExists`, is a boolean indicating whether the session was found.
4. **Assert**:
    - The test asserts that `sessionExists` is `true`. If the session does not exist in the file with the given details, the test will fail with the message "The session does not exist with the given details."

Method: `SessionExistsInFile`

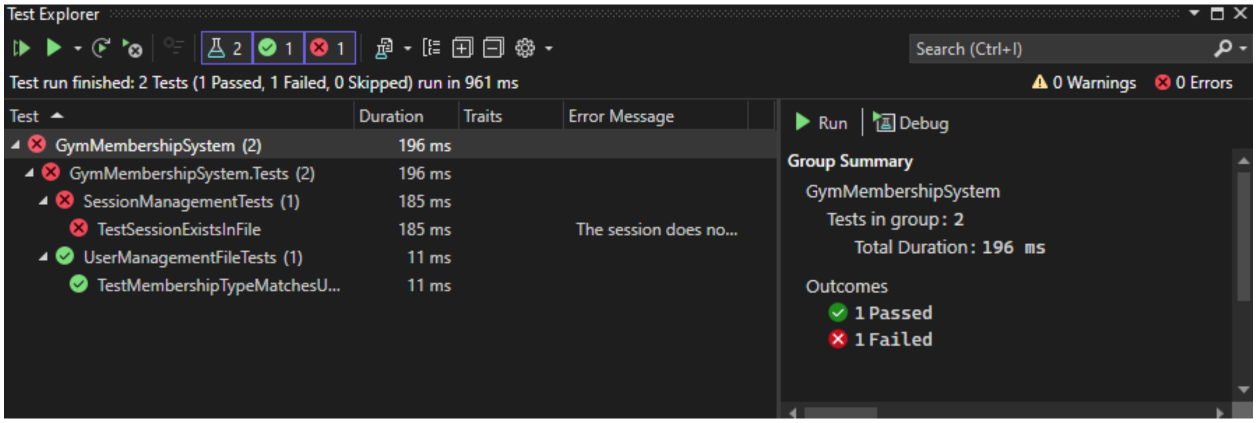
This private helper method performs the actual check within the `sessions.txt` file:

- It first checks if `sessions.txt` exists. If not, it returns `false`.
- It reads all lines from `sessions.txt` and iterates through each line, splitting the line into session details.
- It then compares each element (session ID, user ID, trainer, etc.) to the provided values to determine if a matching session exists.



```
Test run finished: 2 Tests (2 Passed, 0 Failed, 0 Skipped) run in 613 ms          0 Warnings   0 Errors

Test                                      Duration  Traits  Error Message
GymMembershipSystem (2)                   24 ms
  GymMembershipSystem.Tests (2)           24 ms
    SessionManagementTests (1)            10 ms
      TestSessionExistsInFile             10 ms
    UserManagementFileTests (1)           14 ms
      TestMembershipTypeMatchesU...       14 ms

Test Detail Summary
  TestSessionExistsInFile
    Source: SessionManagementTests.cs line 1
    Duration: 10 ms
```

If the data matches the test case, the test case will pass.



```
Test run finished: 2 Tests (1 Passed, 1 Failed, 0 Skipped) run in 961 ms          0 Warnings   0 Errors

Test                                     Duration   Traits   Error Message
GymMembershipSystem (2)                  196 ms
  GymMembershipSystem.Tests (2)          196 ms
    SessionManagementTests (1)           185 ms
      TestSessionExistsInFile            185 ms              The session does no...
    UserManagementFileTests (1)           11 ms
      TestMembershipTypeMatchesU...       11 ms

Group Summary
  GymMembershipSystem
    Tests in group: 2
      Total Duration: 196 ms

  Outcomes
    1 Passed
    1 Failed
```

If any of the data does not match up with the test case, the test case will fail.

Member Contributions

Tasks	Contributor
Main Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Add Trainers Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Admin Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Book Session Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Change Membership Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Dashboard Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Login Admin Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Login Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Progress Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Sign Up Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
View Session Form (Code Quality, Interface Design, and Functionality)	Ron Tran & Khoi Nguyen - Tran
Report	Ron Tran
Unit Testing	Khoi Nguyen - Tran
Project Idea	Ron Tran & Khoi Nguyen - Tran