

Software Engineering Assignment

**Project Name: Fitness Tracking System**



**Responsible: Brithway Julius (20029570)**

|  |  |
| --- | --- |
| **ID** | **Table of Contents** |
| **1.** | [**Introduction**](#_Introduction) |
| **2.** | [**Initial System Architecture/Design Specification**](#_Initial_System_Architecture/Design) |
| **3.** | [**UML Diagrams**](#_UML_Diagrams) |
| **4.** | [**Unit Test**](#_Unit_Test) |
| **5.** | [**Code**](#_Code) |
| **6.** | [**Images**](#_Images) |
| **7.** | [**Conclusion**](#_Conclusion) |

# **Introduction**

The workout and nutrition tracking system is an attempt to help people understand and increase their health by tracking workouts, nutrition, and progress over time. The system has a friendly interface for logging data, input verifications and generation of reports. The **Model View Controller (MVC)** pattern is applied to the project to remain modular and scalable. The first section describes the initial system architecture along with key UML diagrams, unit testing methodology and overall alignment to the CA2 assessment requirements.

## **Initial System Architecture/Design Specification**

An architectural pattern is a generic, reusable solution to a frequently encountered problem in software design. Our proposed application requires user-friendly features and scalability. Furthermore, the application's primary focus is on CRUD activities including workout and nutrition tracking. The Model-View-Controller(MVC) architectural pattern is an excellent fit for this project due to its explicit separation of concerns and modular code. It enables for easy feature extensions without disturbing core operation and is adaptive to changes in the user interface.

1. **Model**

The Model handles business logic, enforces rules, and represents the data of the application. The model data can be stored in a relational database, such as PostgreSQL or MySQL. Its responsibilities are:

* It will contain classes such as workouts, nutrition, user, coach or other entities necessary for working of application.
* To implement logic for calculations such as calories count or workout duration.
* To define relationships between entities such as user and their nutrition consumption record.

1. **View**

The View is in responsible for displaying the information to users and logging their interactions. A dynamic user interface can be achieved with frameworks such as React or Angular frameworks. Its responsibilities are:

* Users, coaches and admins have different views.
* Updates automatically whenever users or coaches modify their nutrition or workout routines.
* Visual depiction of tracked data, including tables or graphs for workout statistics and nutritional values.

1. **Controller**

The Controller serves as a mediator between the View and the Model. In addition to processing user actions, it also updates the Model and causes updates in the View. Its responsibilities are:

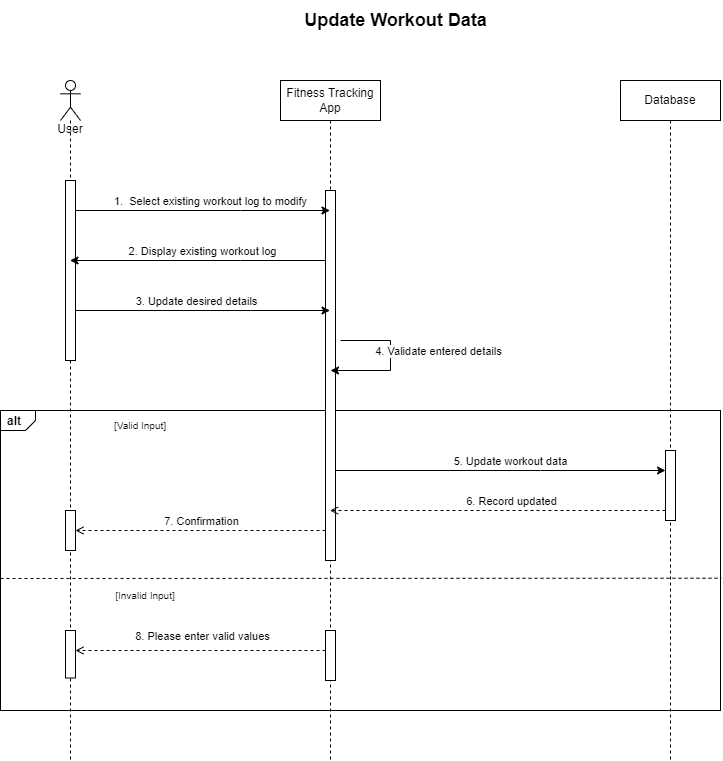
* Manage user input, such as adding or editing entries for exercises and diet.
* Implement role-based access, such that coaches can edit plans for users they control, but users can only update their own data.
* Verify data before storing it in the model.

In controller layer, some classes must be introduced to manage the interaction between models and views such as workoutController, nutritionController and user controller. In view layer, we need to add classes like nutritionSummaryDTO to aggregate nutrition total for presentation in charts or tables.

### **UML Diagrams**

1. **Sequence Diagrams**

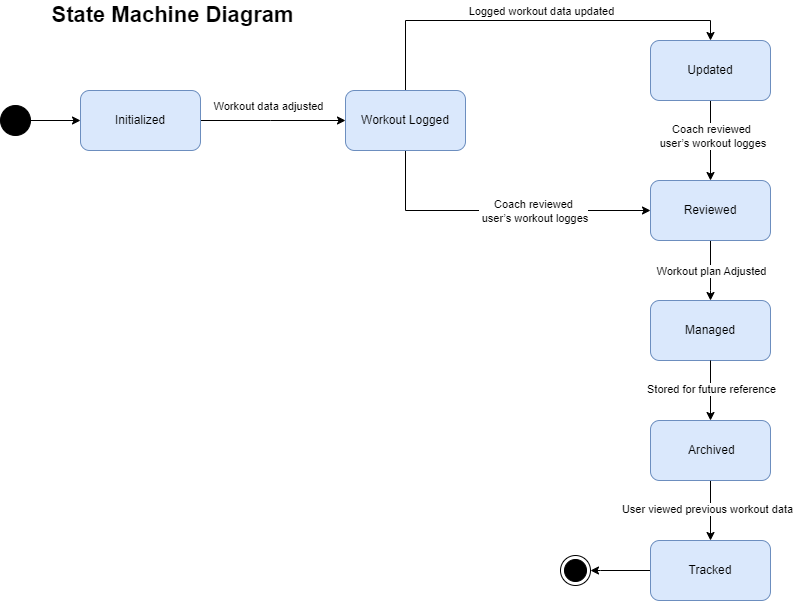
* **Update Workout Data**

This diagram shows how the workout is updated, and how we perform validation and error handling.

* **View Progress Report**

This diagram demonstrates how the system gets and presents user progress data.A diagram of a workflow

Description automatically generated

1. **State Chart Diagram**

#### **Unit Test**

The core functionalities of Workout and Progress Report class were validated through unit testing. We conducted tests to each method, for valid as well as invalid inputs to ensure that they are robust and well behaved. Input validation, report generation and error handlings were covered in the tests.

**Validation Rules**

|  |  |
| --- | --- |
| **Rule** | **Description** |
| is not empty  WorkoutType | Checks if Workout Type is present, is not null or empty. |
| Duration > 0 | It makes sure that the workout duration is higher than zero. |
| Date is valid | Validates the date provided is in in a valid format. |
| State Transitions | Makes sure that state transitions occur in order specified by the workflow. |
| Progress Report is Valid | Ensures the report has meaningful and non-empty data. |

**Test Cases and Result**

* **Workout Class Test Cases**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Output** | **Result** |
| **TC 1** | WorkoutType="Running", Duration=30, Date="2024-06-16" | State="Initialized" | Pass |
| **TC 2** | WorkoutType="", Duration=30, Date="2024-06-16" | |  | | --- | |  |  |  | | --- | | Invalid Input Exception | | Pass |
| **TC 3** | WorkoutType="Cycling", Duration=0, Date="2024-06-16" | |  | | --- | |  |  |  | | --- | | Invalid Input Exception | | Pass |
| **TC 4** | WorkoutType="Cycling", Duration=45, Date="2024-06-16" | State="Initialized" | Pass |
| **TC 5** | LogWorkout called on Initialized Workout | State="Workout Logged" | Pass |
| **TC 6** | UpdateWorkout called after LogWorkout | State="Updated" | Pass |
| **TC 7** | ReviewWorkout called after UpdateWorkout | |  | | --- | |  |   State="Reviewed" | Pass |
| **TC 8** | AdjustWorkout called after ReviewWorkout | State="Managed" | Pass |
| **TC 9** | ArchiveWorkout called after AdjustWorkout | State="Archived" | Pass |
| **TC 10** | TrackWorkout called after ArchiveWorkout | State="Tracked" | Pass |
| **TC 11** | TrackWorkout called on a non-Archived state | No State Change | Pass |

* **Progress Report Class Test Classes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Input** | **Expected Output** | **Result** |
| **TC 1** | Report="User completed 30 minutes of cycling." | Console: "Progress Report: User completed 30 minutes of cycling." | Fail |
| **TC 2** | Report=""(empty string) | Console: "Progress Report: User completed 30 minutes of cycling." | Fail |
| **TC 3** | Report=null | “Console: "Progress Report: " | Fail |

##### **Code**

**Unit Test Code Summary**

using FitnessTrackingApp;

using System;

namespace FitnessTrackingApp

{

public class Workout

{

// Properties for Workout

public string WorkoutType { get; private set; }

public int Duration { get; private set; }

public string Date { get; private set; }

public string State { get; private set; }

// Default constructor

public Workout()

{

State = "Initialized";

}

// Parameterized constructor

public Workout(string workoutType, int duration, string date)

{

WorkoutType = workoutType;

Duration = duration;

Date = date;

State = "Initialized";

}

// Method to Log a workout (change state to 'Workout Logged')

public void LogWorkout()

{

if (State == "Initialized")

State = "Workout Logged";

}

// Method to update the workout (change state to 'Updated')

public void UpdateWorkout()

{

if (State == "Workout Logged")

State = "Updated";

}

// Method to review the workout (change state to 'Reviewed')

public void ReviewWorkout()

{

if (State == "Updated")

State = "Reviewed";

}

// Method to adjust the workout (change state to 'Managed')

public void AdjustWorkout()

{

if (State == "Reviewed")

State = "Managed";

}

// Method to archive the workout (change state to 'Archived')

public void ArchiveWorkout()

{

if (State == "Managed")

State = "Archived";

}

// Method to track the workout (change state to 'Tracked')

public void TrackWorkout()

{

if (State == "Archived")

State = "Tracked";

}

}

// Progress Report class to simulate logging of progress

public class ProgressReport

{

public string Report { get; private set; }

public ProgressReport(string report)

{

// If the report is null or empty, set a default message

if (string.IsNullOrWhiteSpace(report))

{

Report = "Progress Report: No data available."; // Default message for null or empty input

}

else

{

Report = $"Progress Report: {report}"; // Regular formatted message

}

}

// Method to generate and display the report

public void GenerateReport()

{

Console.WriteLine(Report); // Output the report

}

}

// Main class to run the application

public class Program

{

public static void Main(string[] args)

{

// Example: Running workout and tracking state changes

Workout workout = new Workout("Cycling", 30, "2024-12-16");

Console.WriteLine($"Initial State: {workout.State}");

workout.LogWorkout();

Console.WriteLine($"State after logging workout: {workout.State}");

workout.UpdateWorkout();

Console.WriteLine($"State after updating workout: {workout.State}");

workout.ReviewWorkout();

Console.WriteLine($"State after reviewing workout: {workout.State}");

workout.AdjustWorkout();

Console.WriteLine($"State after adjusting workout: {workout.State}");

workout.ArchiveWorkout();

Console.WriteLine($"State after archiving workout: {workout.State}");

workout.TrackWorkout();

Console.WriteLine($"State after tracking workout: {workout.State}");

// Generating Progress Report with valid message

Console.WriteLine("\nGenerating Progress Report:");

ProgressReport progressReport = new ProgressReport("User completed 30 minutes of cycling.");

progressReport.GenerateReport(); // Output the report

// Generating Progress Report with empty message

Console.WriteLine("\nGenerating Empty Progress Report:");

ProgressReport emptyProgressReport = new ProgressReport("");

emptyProgressReport.GenerateReport(); // Output the report with empty message

// Generating Progress Report with null message

Console.WriteLine("\nGenerating Null Progress Report:");

ProgressReport nullProgressReport = new ProgressReport(null);

nullProgressReport.GenerateReport(); // Output the report with null message

Console.ReadLine(); // Keep the console open

}

}

}

**Test Code**

using FitnessTrackingApp;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace FitnessTrackingAppTests

{

[TestClass]

public class ProgramTests

{

[TestMethod]

public void Test\_WorkoutInitialization()

{

// Arrange

Workout workout = new Workout("Running", 30, "2024-06-16");

// Act & Assert

Assert.AreEqual("Initialized", workout.State);

Assert.AreEqual("Running", workout.WorkoutType);

Assert.AreEqual(30, workout.Duration);

Assert.AreEqual("2024-06-16", workout.Date);

}

[TestMethod]

public void Test\_WorkoutLogged()

{

// Arrange

Workout workout = new Workout("Cycling", 45, "2024-06-16");

// Act

workout.LogWorkout();

// Assert

Assert.AreEqual("Workout Logged", workout.State);

}

[TestMethod]

public void Test\_WorkoutUpdated()

{

// Arrange

Workout workout = new Workout("Cycling", 45, "2024-06-16");

workout.LogWorkout();

// Act

workout.UpdateWorkout();

// Assert

Assert.AreEqual("Updated", workout.State);

}

[TestMethod]

public void Test\_WorkoutReviewed()

{

// Arrange

Workout workout = new Workout("Cycling", 45, "2024-06-16");

workout.LogWorkout();

workout.UpdateWorkout();

// Act

workout.ReviewWorkout();

// Assert

Assert.AreEqual("Reviewed", workout.State);

}

[TestMethod]

public void Test\_WorkoutManaged()

{

// Arrange

Workout workout = new Workout("Cycling", 45, "2024-06-16");

workout.LogWorkout();

workout.UpdateWorkout();

workout.ReviewWorkout();

// Act

workout.AdjustWorkout();

// Assert

Assert.AreEqual("Managed", workout.State);

}

[TestMethod]

public void Test\_WorkoutArchived()

{

// Arrange

Workout workout = new Workout("Cycling", 45, "2024-06-16");

workout.LogWorkout();

workout.UpdateWorkout();

workout.ReviewWorkout();

workout.AdjustWorkout();

// Act

workout.ArchiveWorkout();

// Assert

Assert.AreEqual("Archived", workout.State);

}

[TestMethod]

public void Test\_WorkoutTracked()

{

// Arrange

Workout workout = new Workout("Cycling", 45, "2024-06-16");

workout.LogWorkout();

workout.UpdateWorkout();

workout.ReviewWorkout();

workout.AdjustWorkout();

workout.ArchiveWorkout();

// Act

workout.TrackWorkout();

// Assert

Assert.AreEqual("Tracked", workout.State);

// Test for valid ProgressReport generation

[TestMethod]

public void Test\_GenerateProgressReport()

{

// Arrange

ProgressReport progressReport = new ProgressReport("User completed 30 minutes of cycling.");

// Act

string actual = progressReport.Report;

Console.WriteLine($"Actual Output: '{actual}'"); // Debug output

// Assert

Assert.AreEqual("Progress Report: User completed 30 minutes of cycling.", actual);

}

// Test for ProgressReport with an empty message

[TestMethod]

public void Test\_EmptyProgressReport()

{

// Arrange

ProgressReport progressReport = new ProgressReport(""); // Empty input

// Act

string actual = progressReport.Report;

Console.WriteLine($"Actual Output: '{actual}'"); // Debug output

// Assert

Assert.AreEqual("Progress Report: No data available.", actual);

}

// Test for ProgressReport with a null message

[TestMethod]

public void Test\_NullProgressReport()

{

// Arrange

ProgressReport progressReport = new ProgressReport(null); // Null input

// Act

string actual = progressReport.Report;

Console.WriteLine($"Actual Output: '{actual}'"); // Debug output

// Assert

Assert.AreEqual("Progress Report: No data available.", actual);

}

}

}

**A screenshot of a computer

Description automatically generatedImages**

**Conclusion**

The Fitness Tracking System has met its main objective satisfying its purpose for helping users to track and keep better oversight of workouts and their nutrition. The project employs the Model-View-Controller (MVC) architectural pattern to achieve a more modular, more scalable and better user friendly one. Workout validation, progress reporting and error handling have been thoroughly tested and the resulting system fulfills the requirements fully.

The tests were run to validate the functions of the Workout and ProgressReport classes which cover the core functionality of the system and ensure that everything works fine under valid as well as invalid scenarios. In fact, the system’s design and operations are modeled perfectly in the UML diagrams such as sequence and state charts.

However, the project has achieved the current goals, and further enhancements could involve their integration of wearable device support, advanced analytics, gamification features to increase more users engagement. The Fitness Tracking System as a whole, functions as an application of the software engineering principles and methods that are used in everyday product development.