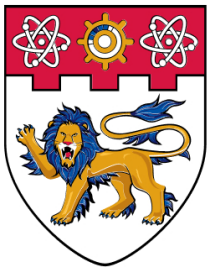


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2006: Software Engineering

Lab#3 Deliverables

Lab Group - Z51 Team: 4 – Binary Brains

Project name: *Sport Spot - where the player meets place*

Team Members:

Sanskkriti Jain	**U2123542L**
Sisodia Anushka	**U2123704F**
Sun Ming Zhong	**U2110180J**
Tan Jue Lin	**U2111250D**

1.0 Project Introduction

1.1 Mission Statement:

Sport Spot aims to be the most reliable source for recommending public sports facilities in Singapore for people of all age groups. Our website serves as a wellness resource for Singaporeans, offering a variety of opportunities for residents, communities, and societies to take part in and spread the joy of active living.

1.2 Scope:

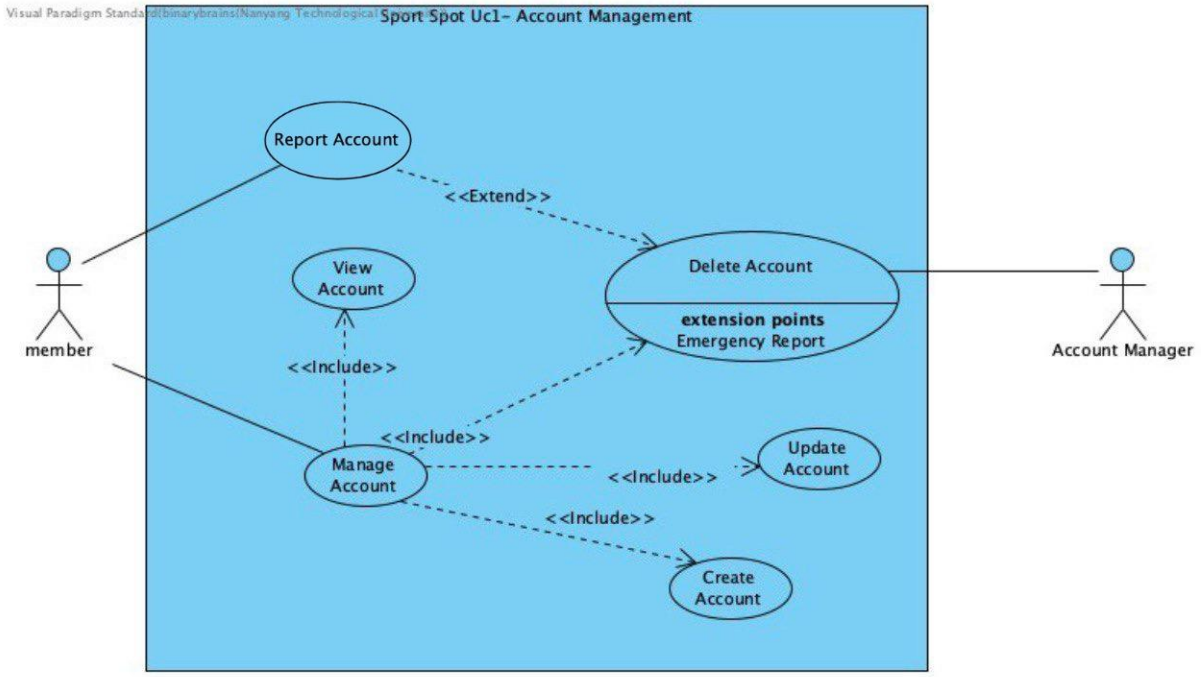
Our web application will allow users to find the nearest sports facility they are willing to go to.

1.3 Target Audience:

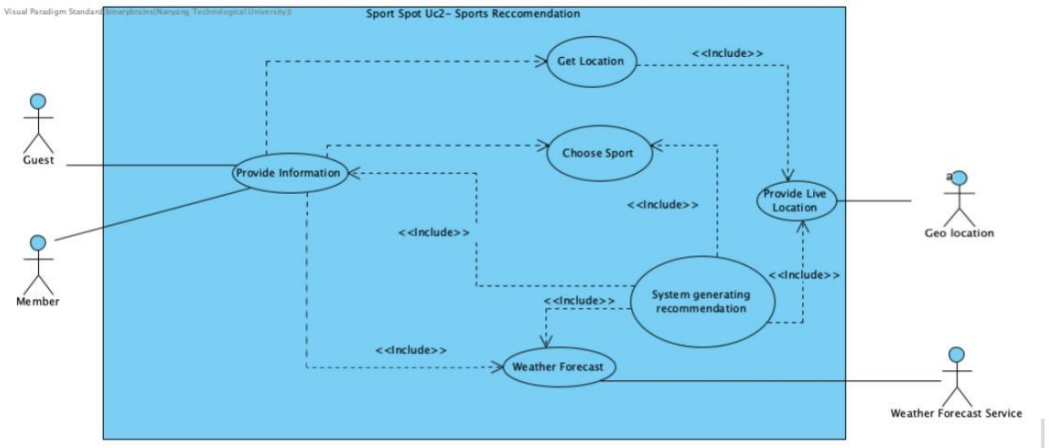
We want our web application to reach and be utilized by people of all age groups.

2.0 Use Case Diagrams -

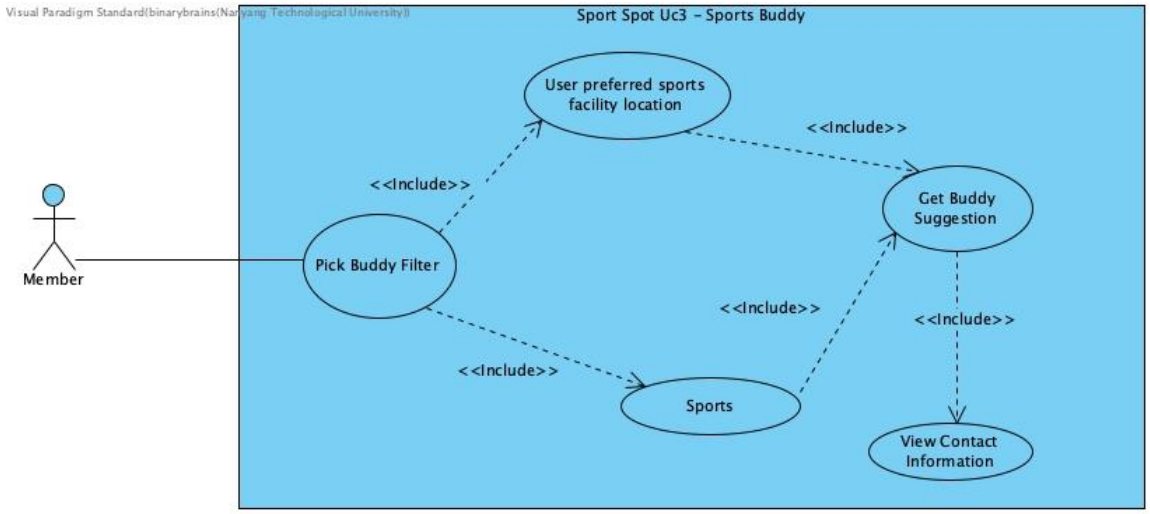
2.1 Use Case 1: Account Information



2.2 Use Case 2: Sports Recommendation



2.3 Use Case 3: Sports Buddy



2.2 Use Case Description

Use Case ID:	UCI_1		
Use Case name:	Create Account		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	12.02.2023

Actor:	Member
Description:	Create Account enables new users to register for an account in the database management system. Users must enter their Name, Email, TelegramID, and Password in order to establish an account.
Preconditions:	1. Username must not be taken by someone else. 2. Passwords must consist of at least 1 special character, letters in mixed cases, and at least 8 characters should be present 3. Email must not have been registered previously.
Postconditions:	1. The system sends a confirmation link to the user's email account. 2. The system displays the main home page.
Priority:	
Frequency of Use:	**1**
Flow of Events:	1. Members enter the required details which are : Name, Email and Password, TelegramID, ContactInfo Access Permission, Gender. 2. The system validates the information. 3. The system creates a new account for the user. 4. System updates account information in the database. 5. The system directs the member to the main home page.

Alternative Flows:	a. If the member account already exists: Email taken. 1. System displays “member account already exists” 2. The system prompts users to create an account or log in. 3. If you create an account, then continue from main flow step 1. 4. If log in, then the system directs to log in page to continue from Login b. Username is already taken 1. The system displays “Username already taken” 2. The system prompts members to input a different username. 3. Continue from main flow step 1. c. Passwords do not meet requirements. 1. The system displays the “Please input a password of at least 1 special character, mixed case and at least 8 characters long” message. 2. Continue from main flow step 1. d. The member wants to sign in as a guest instead of creating an account.

Use Case ID:	UCI_2		
Use Case name:	Update Account		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	

Actor:	Member
Description:	Update Account allows members to change all the data fields in the system database.
Preconditions:	1. Passwords must adhere to requirements: at least 1 special character, letters in mixed cases, and at least 8 characters.
Postconditions:	1. The system sends a confirmation link to the member's email account if there are any changes. 2. The system displays the main landing page.
Priority:	
Frequency of Use:	**1 - 2 times per login**
Flow of Events:	1. Member inputs old email address 2. Member selects the fields that they want to change 3. Member keys in new data to be updated 4. The system validates the information 5. System updates account information in the database 6. The system directs the member to the main landing page
Alternative Flows:	a. Passwords do not meet requirements. 1. The system displays the “Please input a password of at least 1 special character, mixed case and at least 8 characters long” message. 2. Continue from the main flow step 1.

Use Case ID:	UCI_3		
Use Case name:	Delete Account		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	12.02.2023

Actor:	Member, Account Manager
Description:	Delete Account allows members to delete their own account from the system database.
Preconditions:	1. A valid account must exist in the database. 2. The member account is already logged in.
Postconditions:	
Priority:	
Frequency of Use:	**1**
Flow of Events:	1. The member goes to the account management page 2. The member keys in its email and password. 3. The system validates that the member has an account and the credentials are correct. 4. The system deletes member accounts from the database.
Alternative Flows:	a. The User details are inaccurate 1. The system displays the “User with this email does not exist” message. 2. Continue from the main flow step 1.

Use Case ID:	UCI_4		
Use Case name:	View Account		
Created By:	Binary Brains	Last Updated By:	12.02.2023
Date Created:	29.01.2023	Date Last Updated:	

Actor:	Member
Description:	View Account allows members to view their account from the system database
Preconditions:	1. A valid account must exist in the database
Postconditions:	
Priority:	
Frequency of Use:	**1**
Flow of Events:	1. The member goes to account settings 2. The member can view their account
Alternative Flows:	

Use Case ID:	UCI_5		
Use Case name:	Choose Sports		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	12.02.2023

Actor:	Member, Guest
Description:	Choose sports allows users to pick their sports preference from a list of sports given.
Preconditions:	
Postconditions:	1. The system displays all facility's locations according to the user's chosen sports preferences.

Priority:	
Frequency of Use:	**1 - 5 times per login**
Flow of Events:	1. The user must choose from the list of sports available 2. The user will then have to choose which filter they want to further refine the options. 3. The user chooses the location filter or the weather filter or both 4. The system will show sport facility recommendations to the user on the basis of the chosen sport.
Alternative Flows:	A. The user chooses to filter by location and sports 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses sports and location option 3. The user chooses a sport from the drop down menu 4. System generates the facilities 10 km from the users current location 5. System displays facilities areas that have the sports the user want to play and are in 10km range from the users current location B. The user chooses to filter by weather and sports 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses sports and weather option 3. The user chooses a sport from the drop down menu 4. System generates the weather in that area 5. System displays facilities areas that it isn't raining at and they have the sports the user want to play C. The user chooses to filter by weather and sports and location 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses all the 3 options 3. The user chooses a sport from the drop down menu 4. System generates the weather in that area and filters the location by distance 5. System displays facilities in areas that it isn't raining at and they have the sports the user want to play and those that are 10 km in range from their current location.

Use Case ID:	UCI_6		
Use Case name:	Provide live location		
Created By:	Binary Brains	Last Updated By:	
Date Created:	5.02.2023	Date Last Updated:	

Actor:	Member, Guest, Geolocation
Description:	In provide live location, the system uses the users current location (longitude, latitude) generated by the Geolocation API
Preconditions:	
Postconditions:	1. The system generates the user's current location.
Priority:	
Frequency of Use:	**1 - 5 times per login**
Flow of Events:	1. The user chooses the filter by location option. 2. The Geolocation API generates the user's live location.
Alternative Flows:	

Use Case ID:	UCI_7		
Use Case name:	Get Location		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	12.02.2023

Actor:	Member, Guest
Description:	Get location allows registered members and guests to provide their live location to the system for the system to make recommendations.
Preconditions:	
Postconditions:	1. The recommended sports facilities are displayed
Priority:	

Frequency of Use:	**1 - 5 times per login**
Flow of Events:	1. The system allows users to choose whether the user wants to filter according to weather or distance from the detected live location or both 2. The user chooses location option 3. System generates the user's live location 4. The system displays facilities up to 10 km from the user's current location 5. The system generates all the facility locations in that area
Alternative Flows:	A. The user chooses to filter by weather and location 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses location and weather option 3. System generates the user's live location and the weather at the location 4. System displays facilities up to 10 km from the user's current location B. The user chooses to filter by location and sports 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses sports and location option 3. The user chooses a sport from the drop down menu 4. System generates the facilities 10 km from the users current location 5. System displays facilities areas that have the sports the user want to play and are in 10km range from the users current location C. The user chooses to filter by weather and sports and location 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses all the 3 options 3. The user chooses a sport from the drop down menu 4. System generates the weather in that area and filters the location by distance

	5. System displays facilities in areas that it isn't raining at and they have the sports the user want to play and those that are 10 km in range from their current location.

Use Case ID:	UCI_8		
Use Case name:	Weather Forecast		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	12.02.2023

Actor:	Member, Guest, Weather Forecast Service
Description:	Weather allows the user to get suitable facilities based on current weather conditions
Preconditions:	1. The user has chosen the sports facility they want to go to. 2. The user must input their location. 3. The user inputs time to get a weather forecast.
Postconditions:	1. System displays the recommended facilities according to weathers suitability
Priority:	
Frequency of Use:	**1 - 5 times per login**
Flow of Events:	1. The system asks users to choose the sports they want to do. 2. Once the user chooses the sports, the system asks the user whether to filter by weather or distance 3. The user chooses weather 4. The user inputs time for the system to provide a weather forecast. 1. The system generated the nearest sports facility for users within a 10km range. .
Alternative Flows:	A. The user chooses to filter by weather and location 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses location and weather option

	3. System generates the user's live location and the weather at the location 4. System displays facilities up to 10 km from the user's current location B. The user chooses to filter by weather and sports 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses sports and weather option 3. The user chooses a sport from the drop down menu 4. System generates the weather in that area 5. System displays facilities areas that it isn't raining at and they have the sports the user want to play C. The user chooses to filter by weather and sports and location 1. The system allows users to choose whether the user wants to filter according to weather or distance from a live location or sport. 2. The user chooses all the 3 options 3. The user chooses a sport from the drop down menu 4. System generates the weather in that area and filters the location by distance 5. System displays facilities areas that it isn't raining at and they have the sports the user want to play and those that are 10 km in range from their current location.

Use Case ID:	UCI_9		
Use Case name:	System generating recommendation		
Created By:	Binary Brains	Last Updated By:	
Date Created:	05.02.2023	Date Last Updated:	

Actor:	Member, Guest
Description:	The system generating recommendation is the recommendation feature which suggests the nearest sports facility to users according to the filters they have chosen.

Preconditions:	
Postconditions:	1. The system displays the recommended sports facility to the user.
Priority:	
Frequency of Use:	**1 - 5 times per login**
Flow of Events:	1. The system asks users to choose which filter they want to generate facility recommendations. 2. User picks only the sports filter. 3. The user chooses the sport from the drop down menu 2. The system generates the location that offers the sports they want to play
Alternative Flows:	

Use Case ID:	UCI_10		
Use Case name:	Pick Buddy Filter		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	12.02.2023

Actor:	Member
Description:	Pick buddy filter allows users to choose whether to filter buddy lists by their preferred sports locations or by the sports they play.
Preconditions:	1. The user must give permission to the system to share their profile details.
Postconditions:	1. The system shows the information about users according to the chosen filter.
Priority:	
Frequency of Use:	**1 - 5 times per login**

Flow of Events:	1) The user goes to the buddy recommendation menu. 2) The user can choose whether they want to filter by sports facility location or sports.
Alternative Flows:	A. The user chooses not to filter by any of the options 1. The user goes to the buddy recommendation menu 2. The user can choose whether they want to filter by sports facility location or sports 3. The system shows the entire list of users on the system who have allowed their information to be shown

Use Case ID:	UCI_11		
Use Case name:	User Preferred Sports Facility Location		
Created By:	Binary Brains	Last Updated By:	
Date Created:	05.02.2023	Date Last Updated:	

Actor:	Member
Description:	User Preferred Sports Facility Location is a list of member's top 3 sports facilities which matches their sports buddy.
Preconditions:	1. The member must give permission to the system to share their profile details. 2. This feature is only available to users who have created an account. 3. The member must have picked the filter to list sports buddies.
Postconditions:	1. The system shows a list of all the locations. 2. The system allows users to select their preferred location. 3. The system generates buddy recommendation based on the chosen location by the user.
Priority:	
Frequency of Use:	**1 - 5 times per login**
Flow of Events:	1. The member goes to the buddy recommendation menu. 2. The member must choose a filter to recommend sports buddy on the basis of sports facility location.

	3. The member must select the location from the drop down menu. 4. Members may choose a buddy based on the location and contact them.
Alternative Flows:	

Use Case ID:	UCI_12		
Use Case name:	Get Buddy suggestion		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	12.02.2023

Actor:	Member
Description:	Users can connect with others who utilize the same sports facility as them using the buddy recommendation feature.
Preconditions:	1. The user must give permission to the system to share their activity details.
Postconditions:	
Priority:	
Frequency of Use:	**1 - 5 times per login**
Flow of Events:	1. The user goes to the buddy recommendation menu 2. The user can choose whether they want to filter by sports facility location or sports 3. The user chooses not to filter by either option 4. The user can view information of all users 5. The user can contact them using Telegram directly
Alternative Flows:	A. The user chooses sports as a filter. 1. The user goes to the buddy recommendation menu. 2. The user can choose whether they want to filter by sports facility location or sports. 3. The user chooses to filter by sports

	4. The user can view information (like Telegram ID, Sports they play) of users who play the same sport as the user. B. The user chooses sports facility location as a filter 1. The user goes to the buddy recommendation menu. 2. The user can choose whether they want to filter by sports facility location or sports. 3. The user chooses to filter by sports facility location. 4. The user chooses a sport from the drop down menu that they want to find a buddy for 5. The user can view information (like Telegram ID, sports they play, and chosen facility location) of users who have also chosen that facility location.

Use Case ID:	UCI_13		
Use Case name:	Sports		
Created By:	Binary Brains	Last Updated By:	
Date Created:	29.01.2023	Date Last Updated:	12.02.2023

Actor:	Member
Description:	Sports allows users to view profiles of users who play the same sport as the user.
Preconditions:	1. This feature is only available to users who have created an account. 2. The user must give permission to the system to share their profile details.
Postconditions:	1. The system shows a list of all the sports. 2. The system allows users to select their preferred sports. 3. The system generates buddy recommendation based on the chosen sports by the user.
Priority:	
Frequency of Use:	**1 - 5 times per login**

Flow of Events:	1) The member goes to the buddy recommendation menu 2) The member chooses to filter buddy by sports 3) The member clicks on the sports from the list. 4) The system displays the filtered buddy profiles which matches the sports member.
Alternative Flows:	A. The user chooses not to filter by any of the options 1. The user goes to the buddy recommendation menu 2. The user can choose whether they want to filter by sports facility location or sports 3. The system shows the entire list of users on the system who have allowed their information to be shown B. The user chooses to filter by sports facilities location 1. The user goes to the buddy recommendation menu 2. The user can choose to filter by sports facility location 3. The system shows the filtered list of users on the system who have at least one common chosen sports

Use Case ID:	UCI_14		
Use Case name:	View Contact		
Created By:	Binary Brains	Last Updated By:	
Date Created:	05.02.2023	Date Last Updated:	

Actor:	Member
Description:	View contact is the contact information of the member through which their sports buddy can contact them.
Preconditions:	1. The member must give permission to the system to share their profile details.
Postconditions:	1. If all the preconditions are met, the system must show the contact information of the sports buddy to the member.
Priority:	
Frequency of Use:	**1 - 5 times per login**

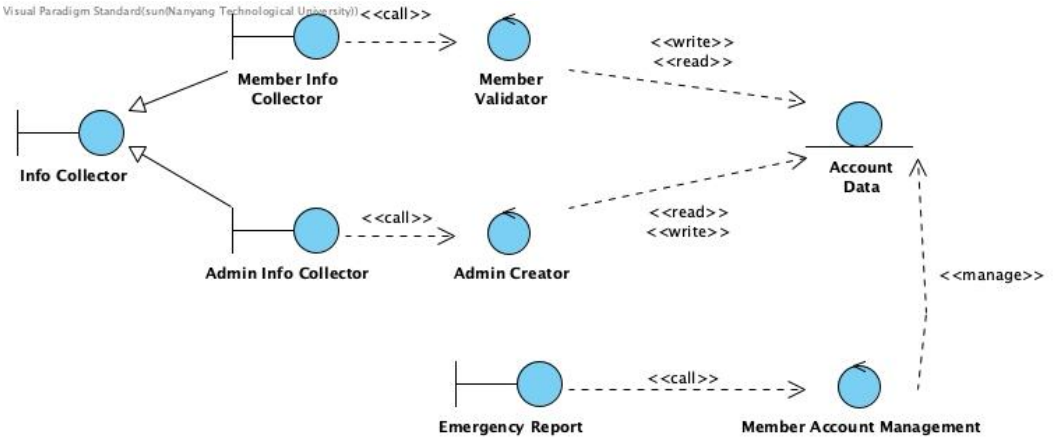
Flow of Events:	1. The member goes to the buddy recommendation menu 2. The member chooses to filter buddy 3. The member selects the buddy they want to match with. 4. Once the buddy accepts the request, the member can view the buddy's contact information.
Alternative Flows:	

Use Case ID:	UCI_15		
Use Case name:	Report Account		
Created By:	Binary Brains	Last Updated By:	
Date Created:	12.02.2023	Date Last Updated:	

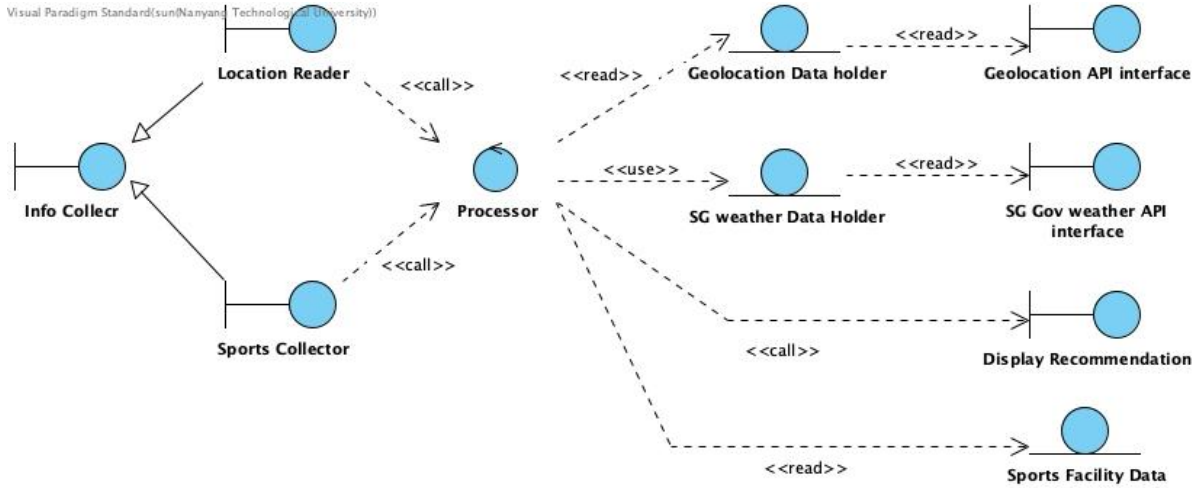
Actor:	Account Manager, User
Description:	Report Account allows a user to report any incidents and the individuals involved.
Preconditions:	1. The reported member must be an actual registered user
Postconditions:	1. The account manager investigates and deletes the reported member's account.
Priority:	
Frequency of Use:	**1 - 5 times per login**
Flow of Events:	1. The member reports a member's account. 2. The system verifies if the reported member is existing in the database. 3. The account manager externally contacts the reporting member to get details. 4. The reported member's account is deleted if the details are accurate.
Alternative Flows:	

3.0 Key boundary classes and control classes

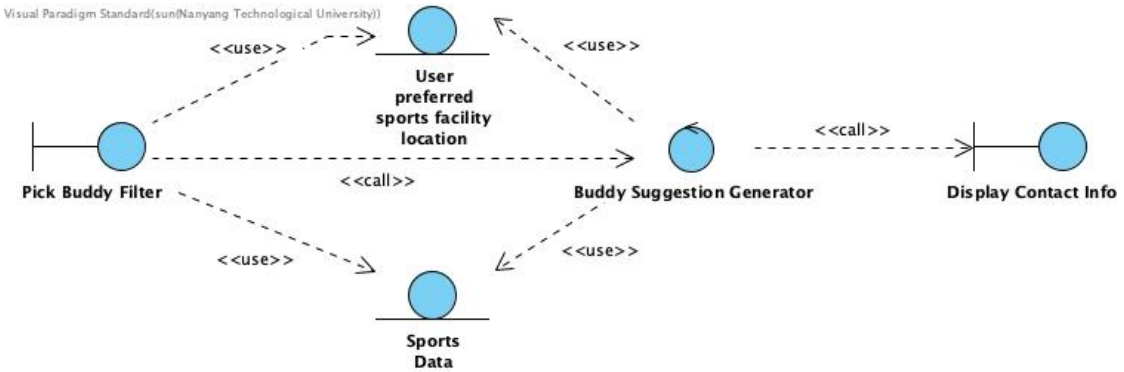
3.1 Class diagram for Account Management



3.2 Class diagram for the Sports Facility recommendation

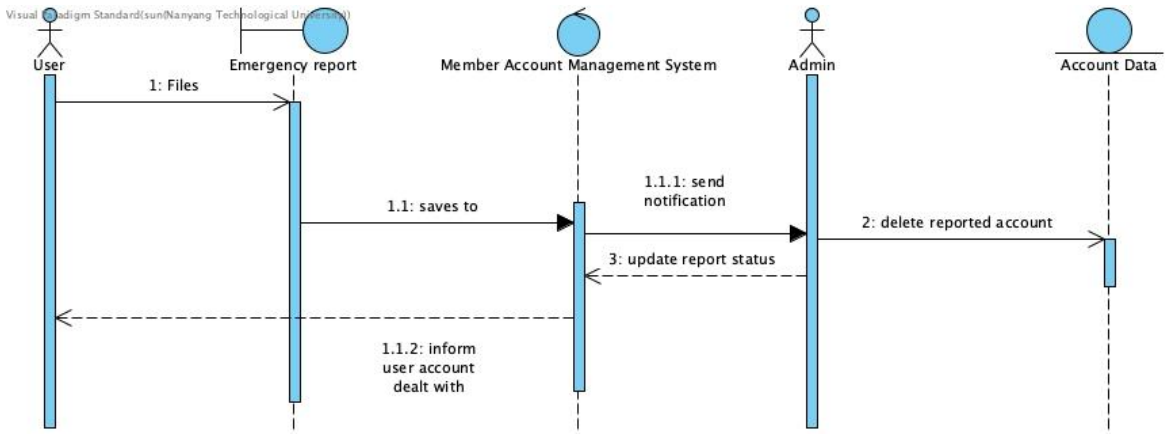


3.3 Class diagram for the sports buddy recommender

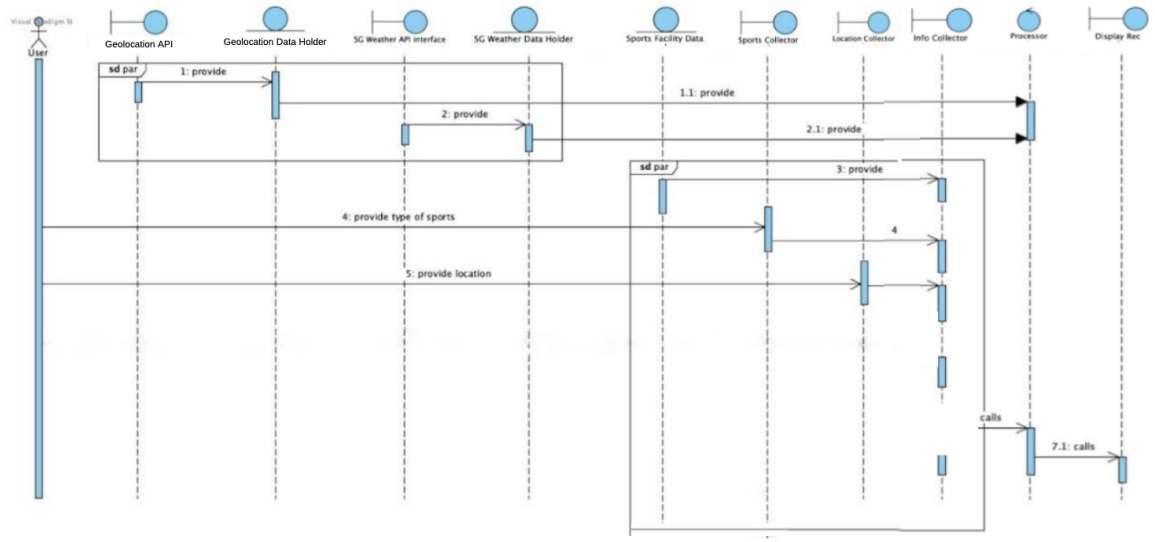


4.0 Sequence diagrams

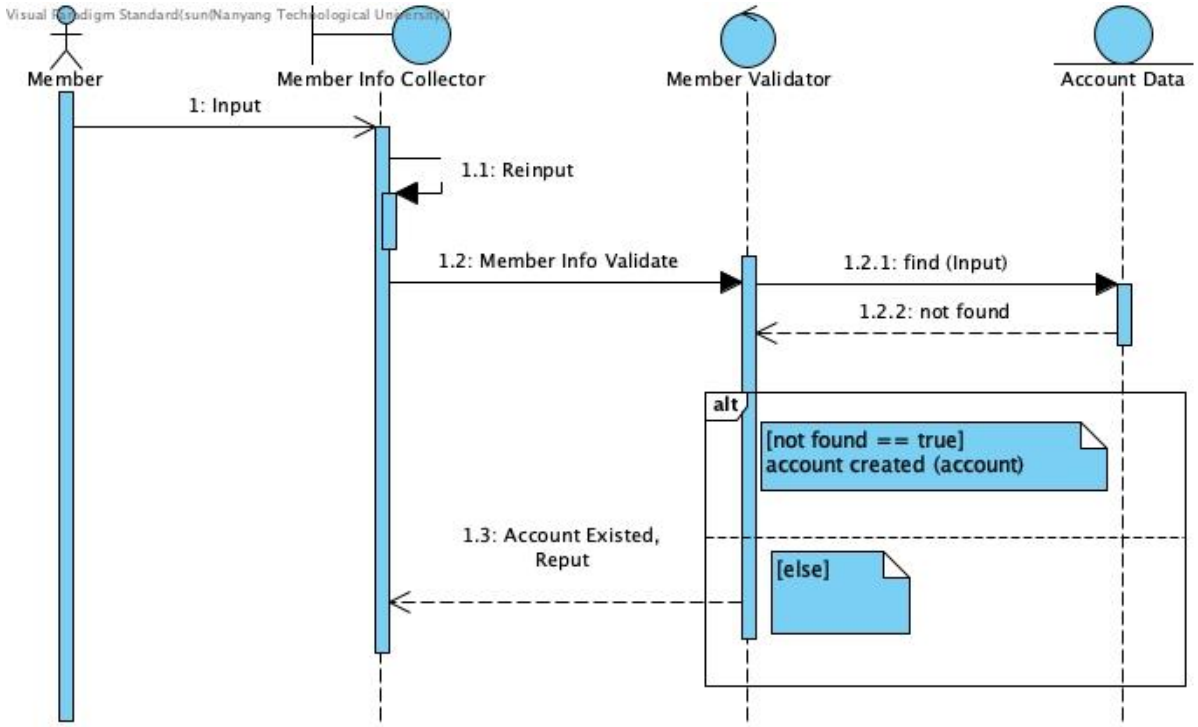
4.1 Sequence Diagram of Emergency Report



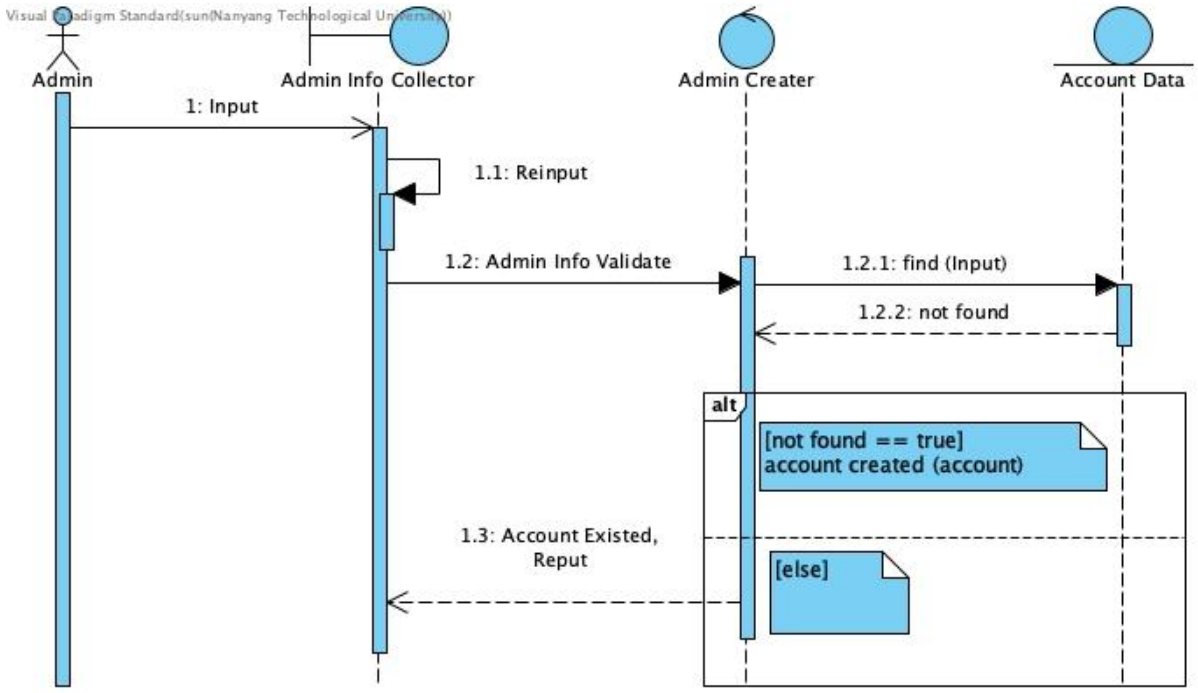
4.2 Sequence Diagram of Sports Recommendation



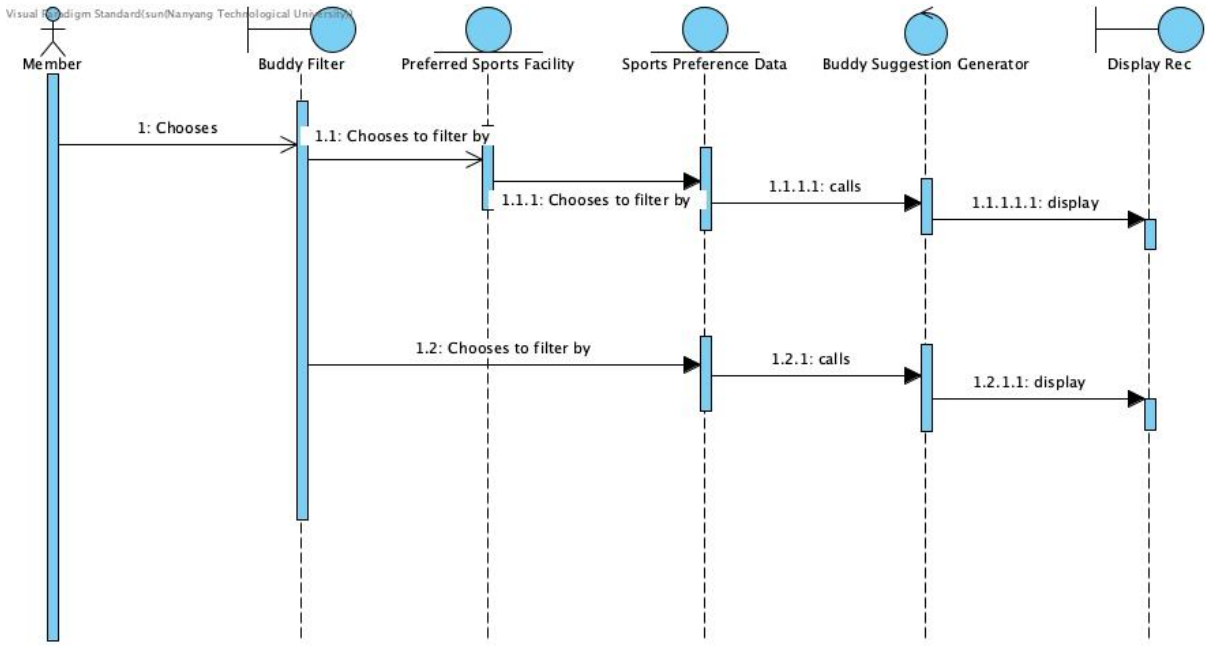
4.3 Sequence Diagram of Member Account Creation



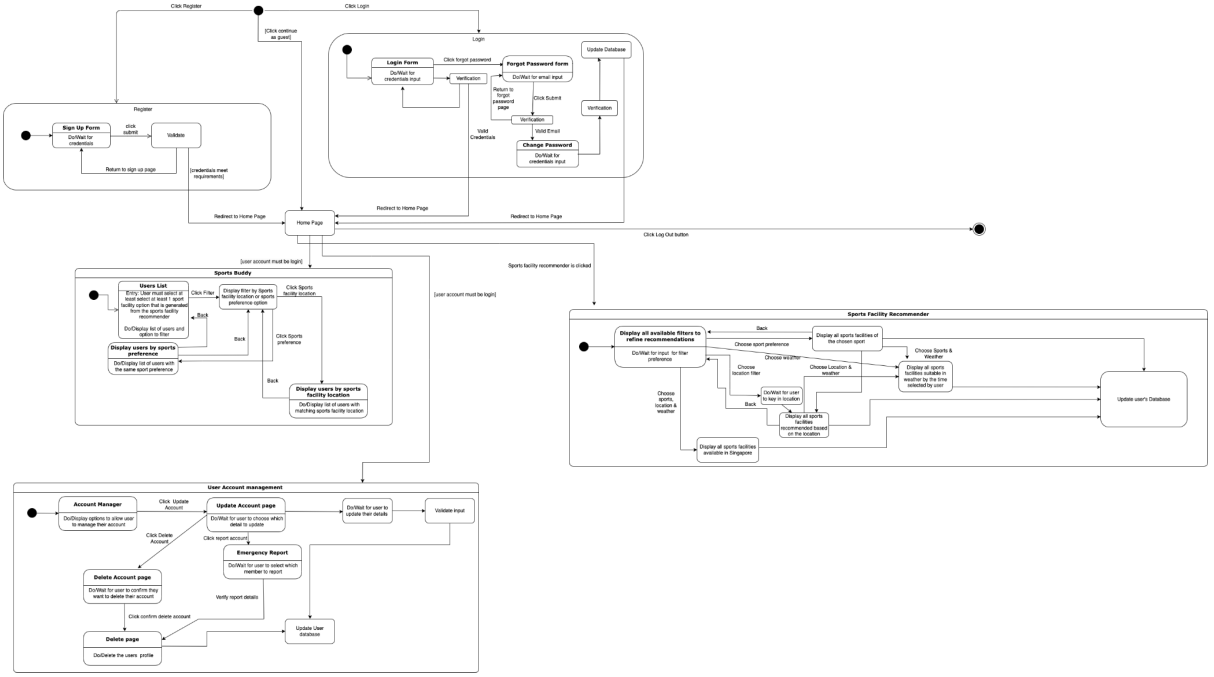
4.4 Sequence Diagram of Admin Account Creation



4.5 Sequence Diagram of Sports Buddy allocation



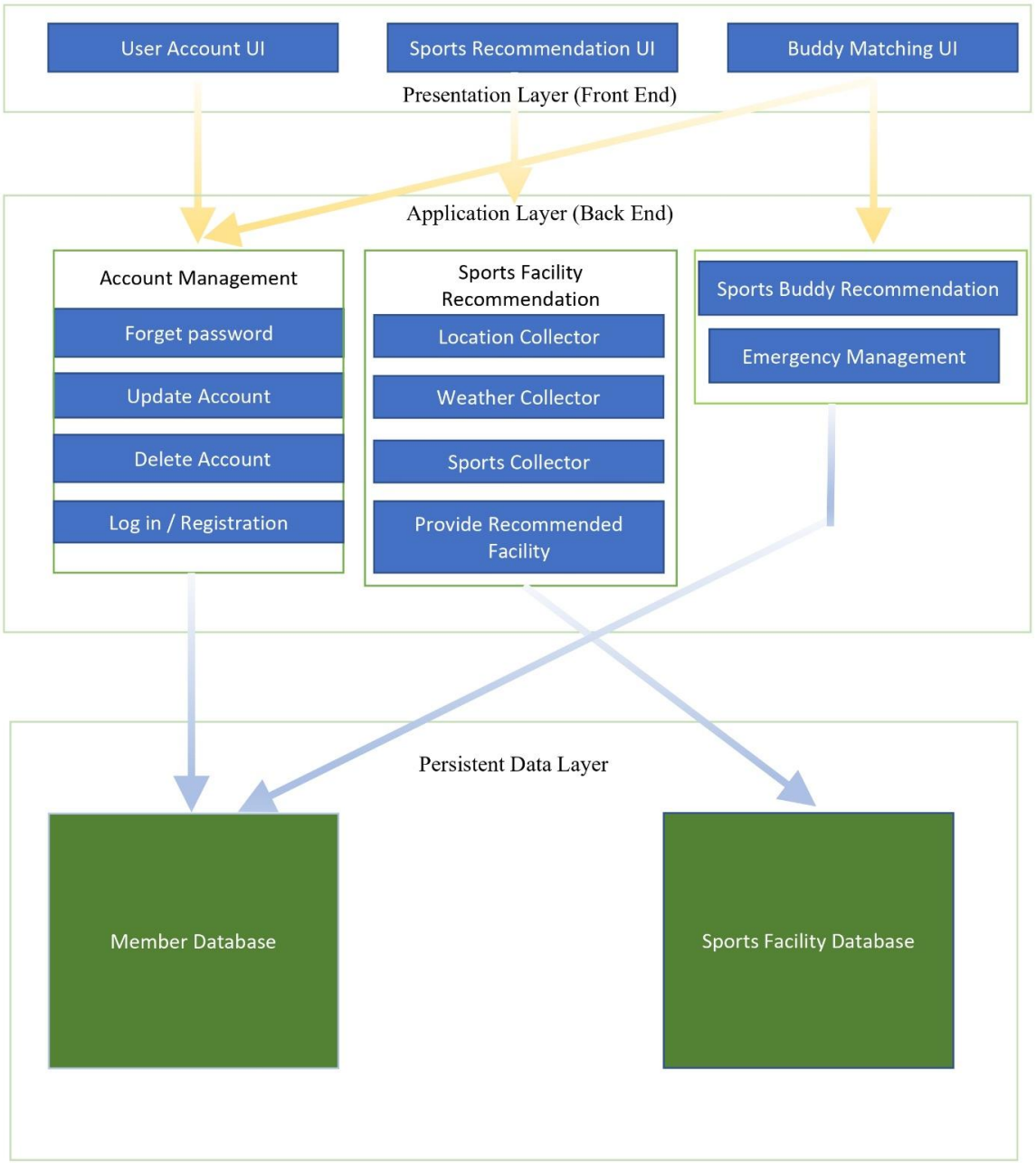
5.0 Dialog map



6.0 Software System Architecture Design

As our program isn't an intensive interactive event-driven system, we select 3 layered architecture rather than MVC.

3 layered Architecture



7.0 Skeleton Application Code

1. Mainpage.html

```
<!DOCTYPE html>
<html>
<head>
  <title>SPORT SPOT | HOME PAGE</title>
  <link rel="stylesheet" type="text/css" href="mainpage.css">

</head>

<script>
    // This is an HTML code for a sports website's homepage.
    //It includes a header section with a logo, navigation menu, and a call-to-action button section.
    //The navigation menu has several links, including a dropdown menu for account management options.
    //The website also provides options for users to register or log in to their accounts.
</script>

<body>
  <header>
    <div class="main">
    <div class="logo">
        <img src="logo.png" alt="Logo" >
        margin-top: 30px;
        margin-right:60px ;
      </div>
     <ul>
     <li class="active"><a href="#"><i class="fa fa-home"></i>Home</a></li>
     <li><a href="#">Recommemender</a></li>
     <li><a href="sportsbuddy.html">Sports Buddy</a></li>
     <li><a href="#"><i class="fa fa-caret-down"></i> Account Management <i class="fa
fa-caret-down"></i></a>
       <div class="sub-menu">
         <ul>
          <li><a href="#">Update Account</a></li>
          <li><a href="#">Report Account</a></li>
          <li><a href="#">Delete Account</a></li>
        </ul>
       </div>
     <li><a href="#">Contact Us</a></li>
     Already registered? <a href= "login.html" > Login </a>
       or <a href="register.html">Register Now</a>
     </li>
     </ul>

     </div>
     <div class="title">
       <h1>SPORT SPOT</h1>
         <h2> WHERE PLAYER MEETS THE PLACE</h2>
     </div>
```

```
      <div class="button">
        <a href="#" class="btn">RECOMMENDER</a>
        <a href="sportsbuddy.html" class="btn">FIND A BUDDY</a>
        <a href="#" class="btn">MANAGE ACCOUNT</a>
      </div>
            </header>
</body>
</html>
```

2. register.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registration Form</title>
  <link rel="stylesheet" href="styleregister.css">
</head>
  <script>
    /* Registration form */
  </script>

  <div class="registration-box">
    <h2>Registration</h2>
    <label>Name</label> <input id="Namebox" type="text">  /* Name of the user */
    <label>Gender</label> /* Gender of the user */
    <select id="Genbox">
      <option value ="Male">Male</option>
      <option value ="Female">Female</option>
      <option value ="Others">Others</option>
    </select>
    <label>Email</label> <input id="Emailbox" type="text"> /* User Email */
    <label>Password</label> <input id="Passwordbox" type="password"> /* User Password */
    <label>Confirm Password</label> <input id="CPasswordbox" type="password"> /* Confirm User password
*/
    <label>TelegramID</label> <input id="TelegramIDbox" type="text"> /* User Telegram */
    <label>Contact</label> /* Permission to use contact */
    <select id="Contactbox">
      <option value ="Yes">Yes</option>
      <option value ="No">No</option>
    </select>

    <hr>

    <button id="Registerbtn">REGISTER</button>
  </div>
```

```
<script type="module">
    // Import the functions you need from the SDKs you need
    import { initializeApp } from "https://www.gstatic.com/firebasejs/9.17.2/firebase-app.js";
    import { getAnalytics } from "https://www.gstatic.com/firebasejs/9.17.2/firebase-analytics.js";
    // TODO: Add SDKs for Firebase products that you want to use
    // https://firebase.google.com/docs/web/setup#available-libraries

    // Your web app's Firebase configuration
    // For Firebase JS SDK v7.20.0 and later, measurementId is optional
    const firebaseConfig = {
            apiKey: "AIzaSyAbyiewHyshcRadjps4bYrs1IhoDcyJPgM",
            authDomain: "sc2006-website.firebaseapp.com",
            databaseURL: "https://sc2006-website-default-rtdb.asia-southeast1.firebasedatabase.app",
            projectId: "sc2006-website",
            storageBucket: "sc2006-website.appspot.com",
            messagingSenderId: "140548370620",
            appId: "1:140548370620:web:c2f845e99edb812c4e60b9",
            measurementId: "G-5GWV6T127Q"
        };

    // Initialize Firebase
    const app = initializeApp(firebaseConfig);
    const analytics = getAnalytics(app);

    import {getFirestore, doc, getDoc,setDoc,collection, addDoc,updateDoc,deleteDoc,deleteField}
    from "https://www.gstatic.com/firebasejs/9.17.2/firebase-firestore.js"

    const db = getFirestore();

    let Namebox = document.getElementById('Namebox');
    let Genbox= document.getElementById('Genbox');
    let Emailbox = document.getElementById('Emailbox');
    let Passwordbox = document.getElementById('Passwordbox');
    let CPasswordbox = document.getElementById('CPasswordbox');
    let TelegramIDBox = document.getElementById('TelegramIDbox');
    let ContactBox = document.getElementById('Contactbox');

    let Registerbtn = document.getElementById("Registerbtn");

    // Define an asynchronous function called AddDocument_AutoID
    async function AddDocument_AutoID() {
        // Use validation functions to check if form inputs are valid
        if (validateName() && validateEmail() && validatePassword() && validateConfirmPassword() &&
validateTelegramID() && validateContact()&&validateGender()) {
        // Get the email value from the email input field and remove any leading/trailing white space
        const email = Emailbox.value.trim();
        // Create a reference to the document in the "Userlist" collection with the specified email
        const userRef = doc(db, "Userlist", email);
        // Get the document with the specified email from the "Userlist" collection
        const userDoc = await getDoc(userRef);
```

```
      // If the document already exists, display an alert message
      if (userDoc.exists()) {
        alert("User with this email is already registered.");
      } else {
        try {
        // If the document doesn't exist, create a new document in the "Userlist" collection with the specified email and set its field values
        const docRef = await setDoc(userRef, {
          full_name: Namebox.value,
          Gender: Genbox.value,
          Email: email,
          Password: Passwordbox.value,
          Confirm_Password: CPasswordbox.value,
          TelegramID: TelegramIDBox.value,
          Contact: ContactBox.value
        });
        // Display a success message and redirect the user to the mainpage.html page
        alert("Successful Registration!! Press OK to continue");
        window.location.href = "/mainpage.html";
        } catch (error) {
        // If there was an error creating the new document, display an error message
        alert("Unsuccessful Registration!, error: " + error);
        }
      }
      }
    }

    Registerbtn.addEventListener("click", AddDocument_AutoID);

      // Validates the user's name
    // Input: None
    // Output: Boolean indicating whether the name is valid or not
    function validateName(){
      let name = Namebox.value.trim();

      // Check if the name field is empty
      if(name === ""){
        alert("Please enter your name.");
        return false;
      }
      // Check if the name contains only letters
      else if(!/^[a-zA-Z]+$/.test(name)){
        alert("Please enter a valid name.");
        return false;
      }
      return true;
    }

    // Validates the user's email address
    // Input: None
    // Output: Boolean indicating whether the email address is valid or not
```

```
function validateEmail(){
  let email = Emailbox.value.trim();

  // Check if the email field is empty
  if(email === ""){
    alert("Please enter your email address.");
    return false;
  }
  // Check if the email address has a valid format
  else if(!/\S+@\S+\.\S+/.test(email)){
    alert("Please enter a valid email address.");
    return false;
  }
  return true;
}

// Validates the user's password
// Input: None
// Output: Boolean indicating whether the password is valid or not
function validatePassword(){
  let password = Passwordbox.value.trim();

  // Check if the password field is empty
  if(password === ""){
    alert("Please enter your password.");
    return false;
  }
  // Check if the password has at least 6 characters
  else if(password.length < 6){
    alert("Please enter a password with at least 6 characters.");
    return false;
  }
  // Check if the password has at least one special character
  else if(!/[!@#$%^&*(),.?":{}|<>]/.test(password)){
    alert("Please include at least one special character in your password.");
    return false;
  }
  // Check if the password has at least one uppercase letter
  else if(!/[A-Z]/.test(password)){
    alert("Please include at least one uppercase letter in your password.");
    return false;
  }
  // Check if the password has at least one lowercase letter
  else if(!/[a-z]/.test(password)){
    alert("Please include at least one lowercase letter in your password.");
    return false;
  }
  return true;
}
```

```
function validateConfirmPassword(){
    // Get the password and confirm password values
    let password = Passwordbox.value.trim();
    let confirmPassword = CPasswordbox.value.trim();
    // If confirm password is empty, show an alert and return false
    if(confirmPassword === ""){
        alert("Please confirm your password.");
        return false;
    }
    // If confirm password does not match password, show an alert and return false
    else if(confirmPassword !== password){
        alert("Passwords do not match. Please try again.");
        return false;
    }
    // Otherwise, return true
    return true;
}

function validateTelegramID(){
    // Get the Telegram ID value
    let telegramID = TelegramIDBox.value.trim();
    // If Telegram ID is empty, show an alert and return false
    if(telegramID === ""){
        alert("Please enter your Telegram ID.");
        return false;
    }
    // Otherwise, return true
    return true;
}

function validateContact(){
    // Get the contact value
    let contact = ContactBox.value.trim();
    // If contact is empty, show an alert and return false
    if(contact === ""){
        alert("Please select an option for contact.");
        return false;
    }
    // Otherwise, return true
    return true;
}

function validateGender(){
    // Get the gender value
    let gender = Genbox.value.trim();
    // If gender is empty, show an alert and return false
    if(gender === ""){
        alert("Please select an option for gender.");
        return false;
    }
    // Otherwise, return true
```

```
        return true;
}
    </script>
  </body>
</html>
```

3. login.html

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login Form</title>
    <link rel="stylesheet" href="stylelogin.css">
</head>

<body>
    <label>Email</label> <input id="Emailbox" type="text">
    <label>Password</label> <input id="Passwordbox" type="password">

    <hr>
    <button id ="Loginbtn">LOGIN</button>

    <script type="module">
        // Import the functions you need from the SDKs you need
        import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-app.js";
        import { getAnalytics } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-analytics.js";
        // TODO: Add SDKs for Firebase products that you want to use
        // https://firebase.google.com/docs/web/setup#available-libraries

        // Your web app's Firebase configuration
        // For Firebase JS SDK v7.20.0 and later, measurementId is optional
        const firebaseConfig = {
            apiKey: "AIzaSyAbyiewHyshcRadjps4bYrs1IhoDcyJPgM",
            authDomain: "sc2006-website.firebaseapp.com",
            projectId: "sc2006-website",
            storageBucket: "sc2006-website.appspot.com",
            messagingSenderId: "140548370620",
            appId: "1:140548370620:web:c2f845e99edb812c4e60b9",
            measurementId: "G-5GWV6T127Q"
        };

        // Initialize Firebase
        const app = initializeApp(firebaseConfig);
```

```
        const analytics = getAnalytics(app);

        // Firebase Firestore database instance
        const db = getFirestore();
        // get elements from DOM
        let Emailbox = document.getElementById('Emailbox');
        let Passwordbox = document.getElementById('Passwordbox');
        let Loginbtn = document.getElementById("Loginbtn");
        // function to authenticate user
        async function Authenticate() {
            const email = Emailbox.value.trim();
            const password = Passwordbox.value.trim();

            // get user document from Firestore
            const userRef = doc(db, "Userlist", Email);
            const userDoc = await getDoc(userRef);
            if (userDoc.exists()) {
                const data = userDoc.data();
                // check if password is valid
                if (data.Password === password ||
data.Password===Updatedpassword) {
                    alert("Login successful!");
                    window.location.href = "/mainpage.html";
                } else {
                    alert("Invalid password!");
                }
            } else {
                alert("User with this email does not exist.");
            }
        }
        // add event listener to login button
        Loginbtn.addEventListener("click", Authenticate);
    </script>
</body>
</html>
```

4. updateaccount.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Update User Details</title>
    <link rel="stylesheet" href="styleupdate.css">
```

```
</head>

<body>
 <label>Old Email</label> <input id="Emailbox" type="text">
 <label><input type="checkbox" id="updateEmail">New Email</label> <input
id="NewEmailbox" type="text">
 <label><input type="checkbox" id="updateName">New Name</label> <input
id="NewNamebox" type="text">
 <label><input type="checkbox" id="updateGender">New Gender</label> <input
id="NewGenderbox" type="text">
 <label><input type="checkbox" id="updateTelegramID">New Telegram ID</label>
<input id="NewTelegramIDbox" type="text">
 <label><input type="checkbox" id="updatePassword">New Password</label>
<input id="NewPasswordbox" type="text">
 <label><input type="checkbox" id="updateContactInfo">New Contact
Info</label> <input id="NewContactInfobox" type="text">

   <hr>
   <button id ="Updatebtn">UPDATE</button>

   <script type="module">
       // Import the functions you need from the SDKs you need
       import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-app.js";
       import { getAnalytics } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-analytics.js";
       // TODO: Add SDKs for Firebase products that you want to use
       // https://firebase.google.com/docs/web/setup#available-libraries

       // Your web app's Firebase configuration
       // For Firebase JS SDK v7.20.0 and later, measurementId is optional
       const firebaseConfig = {
           apiKey: "AIzaSyAbyiewHyshcRadjps4bYrs1IhoDcyJPgM",
           authDomain: "sc2006-website.firebaseapp.com",
           databaseURL:
"https://sc2006-website-default-rtdb.asia-southeast1.firebasedatabase.app",
           projectId: "sc2006-website",
           storageBucket: "sc2006-website.appspot.com",
           messagingSenderId: "140548370620",
           appId: "1:140548370620:web:26bea8bb411408da4e60b9",
           measurementId: "G-NCGC3EKYZ2"
         };

       // Initialize Firebase
       const app = initializeApp(firebaseConfig);
```

```
       const analytics = getAnalytics(app);

         import {getFirestore, doc, getDoc,setDoc,collection,
addDoc,updateDoc,deleteDoc,deleteField}
         from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-firestore.js"

         // Initialize Firestore database
         const db = getFirestore();

         // Get input elements from the DOM
         let Emailbox = document.getElementById('Emailbox');
         let NewPasswordbox = document.getElementById('NewPasswordbox');

         // Get update button element from the DOM
         let Updatebtn = document.getElementById("Updatebtn");

         // Function to update user document in Firestore
         async function UpdateDocument() {
             // Validate email and password inputs
             if (validateEmail() && validatePassword()) {
                 const email = Emailbox.value.trim();
                 const userRef = doc(db, "Userlist", email);
                 const userDoc = await getDoc(userRef);
                 if (userDoc.exists()) {
                     try {
                         // Update user document with new password
                         await updateDoc(userRef, {
                             Updatedpassword: NewPasswordbox.value,
                         });
                         alert("User details updated successfully.");
                     } catch (error) {
                         alert("Failed to update user details, error: " +
error);
                     }
                 } else {
                     alert("User with this email does not exist.");
                 }
             }
         }

         // Attach event listener to update button
         Updatebtn.addEventListener("click", UpdateDocument);

         // Function to validate email input
```

```
        function validateEmail(){
            let email = Emailbox.value.trim();
            if(email === ""){
                alert("Please enter the email address of the user to be
updated.");
                return false;
            }
            else if(!/\S+@\S+\.\S+/.test(email)){
                alert("Please enter a valid email address.");
                return false;
            }
            return true;
        }

        // Function to validate new password input
        function validatePassword(){
            let Updatedpassword = NewPasswordbox.value.trim();
            if(Updatedpassword === ""){
                alert("Please enter the new password for the user.");
                return false;
            }
            return true;
        }


   </script>
</body>

</html>
```

5. deleteaccount.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Delete User Account</title>
   <link rel="stylesheet" href="styledelete.css">
</head>
<body>
   <label>Email</label> <input id="Emailbox" type="text">
   <label>Password</label> <input id="Passwordbox" type="text">
   <hr>
   <button id ="Deletebtn">DELETE ACCOUNT</button>
```

```
   <script type="module">
     // Import the functions you need from the SDKs you need
     import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-app.js";
     import { getAnalytics } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-analytics.js";
     import { getFirestore, doc, getDoc, deleteDoc } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-firestore.js";
      // Your web app's Firebase configuration
     const firebaseConfig = {
         // TODO: Add your Firebase project configuration object here
     };

     // Initialize Firebase
     const app = initializeApp(firebaseConfig);
     const analytics = getAnalytics(app);
     const db = getFirestore();
      let Emailbox = document.getElementById('Emailbox');
     let Passwordbox = document.getElementById('Passwordbox');
     let Deletebtn = document.getElementById("Deletebtn");
      async function DeleteDocument() {
         // Validate email and password inputs
         if (validateEmail() && validatePassword()) {
             const email = Emailbox.value.trim();
             const password = Passwordbox.value.trim();
             const userRef = doc(db, "Userlist", email);
             const userDoc = await getDoc(userRef);
             if (userDoc.exists()) {
                 const userData = userDoc.data();
                 console.log("Password entered:", password);
                 console.log("Password stored:", userData.password);
                 // Compare input password with stored password
                 if (password === userData.Password) {
                     try {
                         await deleteDoc(userRef);
                         alert("User account deleted successfully.");
                     } catch (error) {
                         alert("Failed to delete user account, error: " +
error);
                     }
                 } else {
                     alert("Incorrect password.");
                 }
             } else {
```

```
                alert("User with this email does not exist.");
            }
        }
    }
     Deletebtn.addEventListener("click", DeleteDocument);
     function validateEmail(){
        let email = Emailbox.value.trim();
        if(email === ""){
            alert("Please enter the email address of the user to be
deleted.");
            return false;
        }
        else if(!/\S+@\S+\.\S+/.test(email)){
            alert("Please enter a valid email address.");
            return false;
        }
        return true;
    }
     function validatePassword() {
        let password = Passwordbox.value.trim();
        if (password === "") {
            alert("Please enter the password for the user account.");
            return false;
        } else {
            return true;
        }
    }
 </script>
</body>
</html>
```

6. forgotpassword.html

```
<!DOCTYPE html>
<html lang="en">

<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Update User Details</title>
   <link rel="stylesheet" href="styleupdate.css">
</head>

<body>
   <label>Your Email</label> <input id="Emailbox" type="text">
   <label>New Password</label> <input id="NewPasswordbox" type="text">
```

```
    <div class="signup_link">
        <a href= "login.html" > Login </a>
        or <a href="register.html">Register</a>
        or <a href="mainpage.html">Continue as guest</a>
      </div>

    <hr>
    <button id ="Updatebtn">UPDATE</button>

    <script type="module">
        // Import the functions you need from the SDKs you need
        import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-app.js";
        import { getAnalytics } from
"https://www.gstatic.com/firebasejs/9.17.2/firebase-analytics.js";
        // TODO: Add SDKs for Firebase products that you want to use
        // https://firebase.google.com/docs/web/setup#available-libraries

        // Your web app's Firebase configuration
        // For Firebase JS SDK v7.20.0 and later, measurementId is optional
        const firebaseConfig = {
            apiKey: "AIzaSyAbyiewHyshcRadjps4bYrs1IhoDcyJPgM",
            authDomain: "sc2006-website.firebaseapp.com",
            databaseURL:
"https://sc2006-website-default-rtdb.asia-southeast1.firebasedatabase.app",
            projectId: "sc2006-website",
            storageBucket: "sc2006-website.appspot.com",
            messagingSenderId: "140548370620",
            appId: "1:140548370620:web:26bea8bb411408da4e60b9",
            measurementId: "G-NCGC3EKYZ2"
          };

        // Initialize Firebase
const app = initializeApp(firebaseConfig);
const analytics = getAnalytics(app);

// Import Firestore modules and initialize Firestore
import {getFirestore, doc, getDoc,setDoc,collection,
addDoc,updateDoc,deleteDoc,deleteField}
from "https://www.gstatic.com/firebasejs/9.17.2/firebase-firestore.js"
const db = getFirestore();

// Get HTML elements for email and password fields, as well as the update
button
let Emailbox = document.getElementById('Emailbox');
```

```
let NewPasswordbox = document.getElementById('NewPasswordbox');
let Updatebtn = document.getElementById("Updatebtn");

// Update user details when the update button is clicked
async function UpdateDocument() {
   // Check if email and password are valid before attempting to update
   if (validateEmail() && validatePassword()) {
       const email = Emailbox.value.trim();
       const userRef = doc(db, "Userlist", email);
       const userDoc = await getDoc(userRef);
       if (userDoc.exists()) {
           // Update user details if user exists
           try {
               await updateDoc(userRef, {
                   Updatedpassword: NewPasswordbox.value,
               });
               alert("User details updated successfully.");
           } catch (error) {
               alert("Failed to update user details, error: " + error);
           }
       } else {
           alert("User with this email does not exist.");
       }
   }
}

// Add event listener to the update button
Updatebtn.addEventListener("click", UpdateDocument);

// Validate email address
function validateEmail(){
   let email = Emailbox.value.trim();
   if(email === ""){
       alert("Please enter the email address of the user to be updated.");
       return false;
   }
   else if(!/\S+@\S+\.\S+/.test(email)){
       alert("Please enter a valid email address.");
       return false;
   }
   return true;
}

// Validate password
function validatePassword(){
```

```
    let Updatedpassword = NewPasswordbox.value.trim();
    if(Updatedpassword === ""){
        alert("Please enter the new Telegram ID for the user.");
        return false;
    }
    return true;
}

    </script>
</body>
</html>
```

7. sportsbuddy.html

```
<!DOCTYPE html>
<html lang = "en">

    <head>
        <meta charset="UTF-8">
        <meta http-equiv="X-UA-Compatible" content="IE=edge">
        <meta name ="viewport" content=""width=device-width,
initial-scale="1.0">
        <title> Sports Buddy Finder</title>
        <link rel="stylesheet"

href="https://cdn.jsdelivr.net/npm/bootstrap@4.4.1/dist/css/bootstrap.min.cs
s"

integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23
Q9Ifjh"
            crossorigin="anonymous"/>
            <style type='text/css'>
                body {
                    font-family: 'Gill Sans';
                    padding: 10px;
                    margin: 10px;
                }
                table.recipe {
                    width: 100%;
                    margin-bottom: 5px;
                }
                caption {
                    text-align: left;
                    margin-bottom: 5px;
                    text-transform: lowercase;
                    font-size: 160%;
```

```
                    padding: 5px;
                    letter-spacing: 10px;
                    font-weight: bold;
                }
                </style>

    </head>

    <body>
        <div class = "container mt-3">

            <div class="input-group mb-3 mt-3">
                <input id ="SearchBar"type="text" class="form-control"
placeholder="Search a User" aria-label="Recipient's username"
aria-describedby="Sports Buddy Finder">
                <div class="input-group-append">
                    <select class="input-group mb-3" id = "CategorySelected">
                          <option selected value = "1" > By Name</option>
                          <option value="2">By Gender</option>
                    </select>
                </div>
              </div>

            <table class ="table table-dark">
                <caption style =" caption-side:top"> Sports Buddy Finder
</caption>
                <thead>
                    <th>StdNo</th>
                    <th>Full Name</th>
                    <th>Email</th>
                    <th>Gender</th>
                    <th>TelegramID</th>
                </thead>
                <tbody id="tbody1"></tbody>
            </table>
        </div>

        <!-- Import Firebase and initialize it with your app's config -->
        <script type="module">

            // Import Firebase modules
            import { initializeApp } from
'https://www.gstatic.com/firebasejs/9.1.1/firebase-app.js';
```

```
            import { getFirestore, collection, doc, getDoc, getDocs, query,
where, onSnapshot } from
'https://www.gstatic.com/firebasejs/9.1.1/firebase-firestore.js';

            // Initialize some variables for the table and student number
            var stdNo = 0;
            var tbody = document.getElementById('tbody1');

            // Function to add a single row of data to the table
            function AddItemsToTable(full_name, Email, Gender, TelegramID) {
                let trow = document.createElement('tr');
                let td1 = document.createElement('td');
                let td2 = document.createElement('td');
                let td3 = document.createElement('td');
                let td4 = document.createElement('td');
                let td5 = document.createElement('td');

                // Increment the student number for each row added
                td1.innerHTML = ++stdNo;
                td2.innerHTML = full_name;
                td3.innerHTML = Email;
                td4.innerHTML = Gender;
                td5.innerHTML = TelegramID;

                // Add each cell to the row, and the row to the table body
                trow.appendChild(td1);
                trow.appendChild(td2);
                trow.appendChild(td3);
                trow.appendChild(td4);
                trow.appendChild(td5);

                tbody.appendChild(trow);
            }

            // Function to clear the table and add all data from an array of
users
            function AddallitemstoTable(users) {
                stdNo = 0;
                tbody.innerHTML = '';
                users.forEach((element) => {
                    AddItemsToTable(
                        element.full_name,
                        element.Email,
                        element.Gender,
                        element.TelegramID
```

```
                );
            });
        }

        // Your web app's Firebase configuration
        const firebaseConfig = {
            apiKey: "<your-api-key>",
            authDomain: "<your-auth-domain>",
            databaseURL: "<your-database-url>",
            projectId: "<your-project-id>",
            storageBucket: "<your-storage-bucket>",
            messagingSenderId: "<your-sender-id>",
            appId: "<your-app-id>",
            measurementId: "<your-measurement-id>"
          };

        // Initialize Firebase with your app's configuration
        const app = initializeApp(firebaseConfig);
        const db = getFirestore(app);

        // Function to get all data from Firebase in realtime
        function GetAllDataRealtime() {
            const userRef = collection(db, 'Userlist');
            const q = query(userRef);

            // Listen for changes to the data in Firebase and update the
table accordingly
            onSnapshot(q, (snapshot) => {
                var users = [];

                snapshot.forEach((doc) => {
                    users.push(doc.data());
                });

                AddallitemstoTable(users);
            });
        }

        // Call the function to get all data and display it in the table
when
```

8. filter.html (Sports recommender filter page HTML)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Weather App</title>
</head>
<body>



    <h1>SportSpot Website</h1>
    <!-- Sport filter -->
    <p>Sport:</p>
    <label for="sport-filter"></label>
        <select width=300 style="width: 350px"
        size="8" id="sport-filter" name="sportFilter" multiple
class="sports-filter">
        <option value="">All Sports</option>
        <option value="Running">Running</option>
        <option value="Basketball">Basketball</option>
        <option value="Swimming">Swimming</option>
        <option value="Yoga">Yoga</option>

    </select>
    <p>Hold down the Ctrl (windows) /
        Command (Mac) button to select multiple sports options.</p>



    <!-- Location filter -->
    <label for="location-filter">Filter by Area:</label>
    <input type="checkbox" id="location-filter" />

    <!-- Weather filter -->
    <label for="weather-filter">Filter by Weather:</label>
    <input type="checkbox" id="weather-filter" />

    <button id="filterBtn">Filter</button>
```

```
    <ul id="gymsList"></ul> <!--CHECK AGAIN-->

    <p>The result is: <span id="result"></span></p>


   <!-- <script type="module" src="apiScript.js"></script> -->
    <script type="module" src="tempApiScript.js"></script>



</body>
</html>
```

9. filter.js (Sports recommender filter page Javascript)

```
// Import the functions you need from the SDKs you need
import { initializeApp } from
"https://www.gstatic.com/firebasejs/9.17.1/firebase-app.js";
import { getFirestore } from
"https://www.gstatic.com/firebasejs/9.17.1/firebase-firestore.js"
import { collection, setDoc, doc, getDocs, addDoc, Timestamp,
updateDoc,deleteField} from
"https://www.gstatic.com/firebasejs/9.17.1/firebase-firestore.js"
import { query, orderBy, limit, where, onSnapshot } from
"https://www.gstatic.com/firebasejs/9.17.1/firebase-firestore.js"

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional
const firebaseConfig = {
    apiKey: "AIzaSyAbyiewHyshcRadjps4bYrs1IhoDcyJPgM",
    authDomain: "sc2006-website.firebaseapp.com",
    databaseURL:
"https://sc2006-website-default-rtdb.asia-southeast1.firebasedatabase.a
pp",
    projectId: "sc2006-website",
    storageBucket: "sc2006-website.appspot.com",
    messagingSenderId: "140548370620",
    appId: "1:140548370620:web:26bea8bb411408da4e60b9",
    measurementId: "G-NCGC3EKYZ2"
};
```

```
// Initialize Firebasec(database)
const app = initializeApp(firebaseConfig);
const db = getFirestore(app);

//To store references to three HTML elements in a web page with
specific IDs.
const sportFilter = document.getElementById("sport-filter");
const locationFilter = document.getElementById("location-filter");
const weatherFilter = document.getElementById("weather-filter");

let sum; //To be used to capture the filtering options selected by user
let queryref = collection(db,"gym_collection");
let q3; //To store query results from Firestore

//Array to store the filtered results
let sportFilArr = [];
let weaFilArray = [];
let filArray = [];
let flag;

//Event listener for filter button and reset the filter results as
empty at the start
filterBtn.addEventListener("click", () => {
    sum=0;
    flag=0;
    sportFilArr = [];
    weaFilArray = [];
    locFilArr = [];
    filArray = [];

    getFilteredGyms(); //Function to start filtering process

});

// Store the current filter results
let currentFilters = {
    sports: [],
    location: false,
    weather: false,
};
```

```
async function getFilteredGyms() {

    // Update the current filters based on the user input
    currentFilters.sports =
Array.from(sportFilter.selectedOptions).map((option) => option.value);
    currentFilters.location = locationFilter.checked;
    currentFilters.weather = weatherFilter.checked;

    //To be used to capture the filtering options selected by user
    if(currentFilters.sports.length > 0){
        sum = sum + 1;
    }

    if(currentFilters.weather) {
        sum = sum + 2
    }

    if(currentFilters.location){
        sum = sum + 4
    }

    //Switch statement cases to cover all possible filtering options
selected by user
    switch(sum) {
        case 1: //To filter sports facilities by Type of Sports
            sFilter(currentFilters.sports);
            break;

        case 2: //To filter sports facilities by Weather
            wFilter();
            break;

        case 3: //To filter sports facilities by Type of Sports and
Weather
            sFilter(currentFilters.sports);
            wFilter();
            setTimeout(function(){ //Find common sport facilities from
the results of the 2 filtering
                swFilter(sportFilArr,weaFilArray);
            }, 7000);
            break;
```

```
        case 4: //To filter sports facilities by Live Location of user
            Lfilter();
            break;

        case 5: //To filter sports facilities by Type of Sports and
Live Location of user
            sFilter(currentFilters.sports);
            Lfilter();
            setTimeout(function(){ //Find common sport facilities from
the results of the 2 filtering
                swFilter(sportFilArr,locFilArr);
            }, 500);
            break;

        case 6: //To filter sports facilities by Weather and Live
Location of user
            wFilter();
            Lfilter();
            setTimeout(function(){ //Find common sport facilities from
the results of the 2 filtering
                swFilter(weaFilArray,locFilArr);
            }, 7000);
            break;

        case 7: //To filter sports facilities by Type of Sports,
Weather and Live Location of user
            sFilter(currentFilters.sports);
            Lfilter();
            setTimeout(function(){ //Find common sport facilities from
the results of the 2 filtering
                swFilter(sportFilArr,locFilArr);
            }, 500);
            wFilter();
             //Going into second round of filtering using the results
from above
            setTimeout(function(){ //Find common sport facilities from
the results of the 3 filtering
                flag++;
                swFilter(filArray,weaFilArray);
            }, 7000);
```

```
            break;

        default:
          // code block

      }

    sportFilArr = [];

}

//Function to filter sports facilities by Type of Sports
function sFilter(sportsArray){

    //Creates a query object q3 that references a collection of
documents in Firestore using the queryref variable.
    //The query function is used to apply a filter to the query by
specifying that the sportsType field must contain any of the values in
the sportsArray
    q3 = query(queryref, where('sportsType', 'array-contains-any',
sportsArray));

        //The getDocs function is called on the q3 query object to
retrieve the matching documents as a query snapshot.
        getDocs(q3)
        .then((querySnapshot) => {

            //The code then iterates over each document in the query
snapshot using the forEach function and
            //extracts the data for each document using the data
method.
            //In this case, the code retrieves the name field from each
document and adds it to an array called sportFilArr.
            querySnapshot.forEach((doc) => {
            const data = doc.data();
            const fName = doc.data().name;
            sportFilArr.push(fName);
        });
        })

        //A setTimeout function is used to delay updating the HTML
element with the ID of "result" by 500 milliseconds.
```

```
        //This is because the query and document retrieval process is
asynchronous and takes some time to complete.
        setTimeout(function(){
            const resultElement = document.getElementById('result');
            resultElement.textContent = sportFilArr;
        }, 500);
}


//The following code retrieves weather forecast data from an API,
processes the data, and stores the relevant data in a Firestore
collection.
function getWeather(){

    //Use the fetch function to make a request to the Data.gov.sg API
and
    //Retrieve the weather forecast data for a specific date.
    //The API returns a promise that resolves to a response object.
    const weather =
fetch('https://api.data.gov.sg/v1/environment/2-hour-weather-forecast?d
ate=2023-03-20') //2023-01-10
    .then(response => response.json()) //Use the response.json() method
to extract the JSON data from the response object. This also returns a
promise that resolves to the JSON data.
    .then(data => {
        return data.items[0]; //Extract the relevant data from the JSON
data using dot notation and return the first item in the items array.
    });

    const saveWeather = async() => { //Declare an asynchronous function
saveWeather that waits for the weather promise to resolve
        //and then extracts the forecast data for each of the 47 areas
in the JSON data.
        const data = (await weather);
        for (let i = 0; i<47; i++) {
            const area = data.forecasts[i].area;
            const forecast = data.forecasts[i].forecast;

            //For each area in the forecast data, create a Firestore
document reference using the doc method and the relevant collection
name, document name, and area name.
```

```
            //This stores each Area name and its corresponding weather
type in the database document
            const docOne = doc(db, "Weather_forecasts", area + "
document"); //db, collectionName, documentName
            setDoc(docOne, {
                Area: area,
                Forecast: forecast }, { merge: true });
            }
    };
    saveWeather(); //Call the saveWeather function to store the
forecast data in Firestore.

}

//This code uses the Firestore database service to filter out areas
that are not experiencing rain, based on the weather forecast data
stored in the Firestore collection.
function wFilter(){

    //Call the getWeather function to retrieve the weather forecast
data and store it in the Firestore collection
    getWeather();
    //Declare a Firestore database reference to the Weather_forecasts
collection using the collection method.
    const Weather_forecastRef = collection(db,"Weather_forecasts")
    //Create a Firestore query that filters the documents based on the
Forecast field value, which contains the weather forecast data for each
area.
    const q = query(Weather_forecastRef, where('Forecast', 'in',
['Partly Cloudy (Day)', 'Partly Cloudy (Night)', 'Cloudy', 'Partly
Cloudy', 'Fair & Warm', 'Fair (Night)', 'Fair (Day)']));

    //Create an empty array areas to store the names of areas that are
not experiencing rain
    let areas = []; //all areas option where its is not raining

    //Use the getDocs method to retrieve the documents that match the
Firestore query
    getDocs(q)
    //Iterate over the documents using the querySnapshot.forEach method
and extract the area name from each document.
        .then((querySnapshot) => {
```

```
        querySnapshot.forEach((doc) => {
            const data = doc.data();
            const area = data.Area;
            areas.push(area);//Push each area name into the areas
array.
        });

        //Call the displayAreas function with the areas array as an
argument
        setTimeout(() => {
            displayAreas(areas);
        }, 5000);

        })

        .catch((error) => {
        console.log("Error getting documents: ", error);
        });


        //After getting all areas that are not raining,
        //the function finds out the sports facilities located at those
areas that are not raining
        //which will be the result of the filtering
        function displayAreas(areas) {

            let count;
            if(areas.length%9==0){
                count= (areas.length/9);
            }
            else{
                count= ((areas.length-areas.length%9)/9)+1;
            }


            for (let i = 0; i<count*9; i+=9) {
                let k=i+9;
                const slicedArray = areas.slice(i,i+9); //for all areas
where its is not raining, can only displayed up to 8 locations if users
choose all areas that are not raining
```

```
                const Gym_collectionRef =
collection(db,"gym_collection")

                if(slicedArray.length == 0){
                    console.log("All areas are raining today!")
                } else{
                    const q2 = query(Gym_collectionRef, where('area',
'in', slicedArray));

                    getDocs(q2)
                    .then((querySnapshot) => {
                        querySnapshot.forEach((doc) => {
                        const data = doc.data();
                        const fName = data.name;
                        weaFilArray.push(fName);
                        });
                    })
                    .catch((error) => {
                    console.log("Error getting documents: ", error);
                    });
                }
            }

            setTimeout(function(){
                log();

            }, 500);
        }

}

//To display the results of the sports facilities filtered by weather
function log(){
    weaFilArray.forEach((name) => {
        const resultElement = document.getElementById('result');
        resultElement.textContent = weaFilArray;
    });
}


// Declare global variables to be used in location filter
let dataLong;
```

```
let dataLat;
let userLat;
let userLong;
var locFilArr = []

//This code uses the Firestore database service to filter out sports
facilities that are within 10km of the user's live location

function Lfilter(){

    // Define the getPosition function to get user's geolocation
    function getPosition() {
        return new Promise(function(resolve, reject) {
          if (navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(function(position)
{
              userLat = position.coords.latitude; //Store user's
latitude
              userLong = position.coords.longitude; //Store user's
longitude
              resolve();
            }, function(error) {
              reject(error);
            });
          } else {
            reject("Geolocation is not supported by this browser.");
          }
        });
      }

      //Call the getPosition function to get user's geolocation
      getPosition().then(async function() {

        // Get the sport facilities data from gym_collection Firestore
        const querySnapshot = await
getDocs(collection(db,"gym_collection"));

        //Loop through the all the sport facilities data from
gym_collection Firestore
        querySnapshot.forEach(doc=>{
            dataLong = doc.data().longtitude; //Store longitude
            dataLat = doc.data().latitude; //Store latitude
```

```
            const toRadians = (angle) => angle * (Math.PI / 180);

            // Convert latitudes and longitudes to radians
            const lat1Rad = toRadians(userLat);
            const lon1Rad = toRadians(userLong);
            const lat2Rad = toRadians(dataLat);
            const lon2Rad = toRadians(dataLong);

            // Calculate the differences between the latitudes and
longitudes of database data and user data
            const dLat = lat2Rad - lat1Rad;
            const dLon = lon2Rad - lon1Rad;

            // Apply the Haversine formula
            const a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                        Math.cos(lat1Rad) * Math.cos(lat2Rad) *
                        Math.sin(dLon / 2) * Math.sin(dLon / 2);
            const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

            // Earth's mean radius in kilometers (approximately 6,371
km)
            const R = 6371;

            // Calculate and store the sports facilities data if it's
within 10km of the users live location
            const distance = R * c;

            if(distance<10){
                locFilArr.push(doc.data().name);
            }

            //Store and Display results of filtered sports facilities
on the website
            const resultElement = document.getElementById('result');
            resultElement.textContent = locFilArr;
    });
        }).catch(function(error) {
            console.error(error);
        });

}
```

```
function swFilter(arrayOne, arrayTwo){

    filArray=[]; //"filArray" that will hold the filtered values

    //Use a nested loop to iterate through the elements of both arrays
and compare their values,
    //to find common sport facilities
    //If an element of arrayOne matches an element of arrayTwo, push
the matching element to the "filArray"
    //This will be the result from 2 or more filtering options
    for(let j = 0; j<arrayOne.length;j++ ){
        for(let i = 0;i<arrayTwo.length;i++){
            if(arrayOne[j]===arrayTwo[i]){
                filArray.push(arrayOne[j]);
                if (sum==7){
                    if(flag==1){ //When all 3 filtering options are
selected, then when we are at the second round of filtering, which is
the last round, then we allow the printing of the final results
                        const resultElement =
document.getElementById('result');
                        resultElement.textContent = filArray;
                    }
                } else{ //To display results of the filtered sports
facilities when 2 filtering options are selected
                    const resultElement =
document.getElementById('result');
                    resultElement.textContent = filArray;
                }

            }
        }
    }

}
```