WORKOUT TRACKER

Iteration 3 – Client Side Documentation

Team 11

Mahek Parmar 202052379

Tejus Revi 202061206

Brief update from Iteration 2

- As required, we have progressed using the MVC framework, mostly working on the View aspect during in this iteration.
- We have made extensive use of HTML, CSS and JQuery to develop the Client Side making the front end/ user's view as attractive and convenient as possible.
- When developing the front end, we have also used the BootStrap library.
- We renamed our application to *Workout Tracker* from the previous Workout Manager, we felt this new name suited the application better.

Brief summary of the overall structure of the application.

The application consists of 3 models and their respective controllers, which were extensively analyzed in the previous iteration. Their structure remains the same. Below is a brief recap:

The 3 models are:

1. User
2. Exercise
3. WorkoutProgram – stores Exercise along with their respective reps and sets. It can be set public (viewed by all) or private (only viewable by the creator).

The 3 controllers are:

1. UserController – allows users to interact with the User model
2. ExerciseController – allows users to interact with the ExerciseController model
3. WorkoutProgramController – allows users to interact with the WorkoutProgram model

Detailed analysis on the structure, routes and functions involving the above can be found here.

Functionalities implemented (ctnd from Iteration 2)

- The user can create their accounts, either using their Google authentication, or by manually entering the username, email and password.
- The user can update/change their usernames and passwords, whenever they wish to.
- The user can delete their account if they want to
- Every user account has user-related information such as their age, height, weight, gender and target weight. The user may view their details and may also update these accordingly whenever they wish to.
- The user can view exercises – they can either search for single specific exercises, or they may search for a range of different exercises providing some exercise details such as equipment, target muscle and body part.
- The user can create their own custom made workout program, and give it a name.
- They can set the workout program to either public (can be viewed by others) or private (can only be viewed by user themselves)
- After creating a workout program, the user may delete it if they want.
- The user can update the workout program details if they want to.
- The user can add exercise to the workout program, specifying how many repetitions and sets per exercise added to their workout program.

- The user can also remove/delete exercises from their workout programs.
- The user can view all the workout programs created by them
- The user can view workout programs created by others which are public

Functionalities not implemented.

- Developing a multi-lingual user interface and accounting for color pallets turned out to me more difficult and trickier than we expected it to be. It was something we were really looking forward to implementing, but eventually decided not to implement it due to time pressures.
- As briefly mentioned in the previous iteration, we opted out of the functionality involving exporting workout plans. The application conveniently serves that purpose hence we didn't feel the need for exporting workout programs to the user's local devices.

Added Functionalities

- The User can see their Body Mass Index values based on their height and weight. Their BMI is bound to change with changes in their height and/or weight.
- We also provided a minor progress status, showing how far behind/ahead the user is from their target weight.

Views and elements

Our application provides 6 different views to our users. These include:

1. Log-in page
2. Home
3. Body Metrics
4. Account
5. About
6. Workout Program

Views 2-5 are accessed through the dashboard view. We will now provide detailed analysis of how the views were implemented and what they provide for the user.

1. Log-in Page

Our system management application is user-oriented hence every user must have an account to use our application. This view allows the user to achieve one of our key desired functionalities – *"The user can create their accounts, either using their Google authentication, or by manually entering the username, email and password."*

The user can provide their email and password to log-in to the app. Providing invalid/incorrect credentials will prompt an error message informing the user that about " Incorrect credentials". On providing the correct credentials, the user is taken to their home page.

If a user does not have an account, they may create one by clicking on the "Not a member? Register Today" button. This takes the user to the registration page where the user is asked to provide the necessary credentials in a form. To register the user is asked to provide:

1. Username
2. Email
3. Password (asked twice to ensure the user provides their desired password)

The user is notified on successful registration and is asked to log-in using their credentials.

In todays world, a lot of web based applications and sign-ups happen via Google accounts. This is indeed a very practical feature, and hence we are proud that we successfully implemented this feature. Hence, user's may also sign-in using their Google accounts. If a user opts to do so, they are redirected to the Google sign up page.

The following components interact together to achieve this view

1) HTML template: index.html

2) JS template: main.js

3) CSS template: in-line CSS and style.css

Log-in

This is what the initial log-in page looks like:

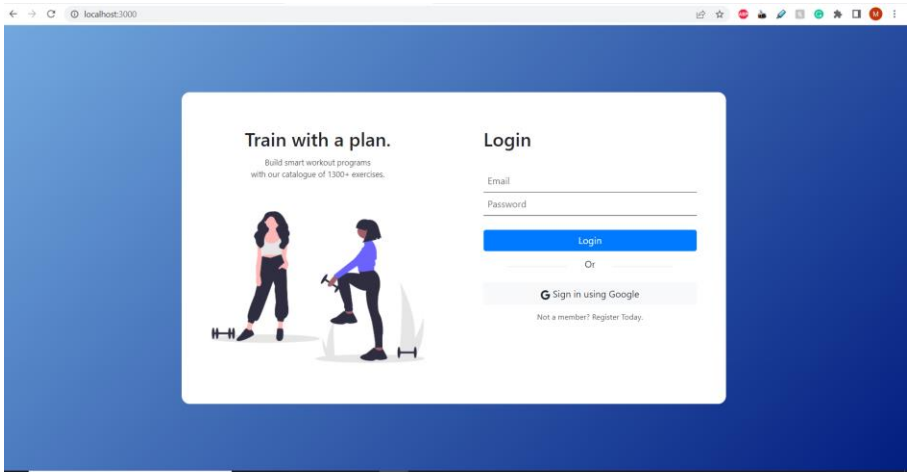


Figure 1.1 – User's view of the Login page

Below is a small HTML snippet of how the username and password forms were implemented

```
<form>
  <input
    class="text-input"
    type="text"
    id="login-email-input"
    placeholder="Email"
  />
  <br />
  <input
    class="text-input"
    type="password"
    id="login-password-input"
    placeholder="Password"
  />
  <br />
  <br />
  <button
    id="submit-login"
    class="btn btn-primary full-width-button"
  >
```

Code snippet 1.1 – HTML implementation of log-in form

A click event is added to the log in button, so when the user clicks the login button, this executes a function to verify the credentials and if correct, log the user in and take them to their home page. We use our authentication middleware PassportJS for this – an ajax GET request is then made to facilitate this. There is also some front-end validation such that none of the fields need to be empty. The JavaScript aspect for this functionality is shown below: (update img)

```
$("#submit-login").click(function (event) {
  event.preventDefault();

  // logout user, if already logged in
  $.ajax({
    url: "/logout",
    type: "GET",
    contentType: "application/json",
  });
  let u = {};

  if (
    //if the credentials are null/empty
    $("#login-email-input").val() == "" ||
    $("#login-password-input").val() == ""
  ) {
    $("#alert-danger").text("Please fill in all required information");
    $("#alert-danger").show();
  } else {
    //validate the credentails
    u.email = $("#login-email-input").val();
    u.password = $("#login-password-input").val();

    $.ajax({
      url: "/auth/local",
      type: "POST",
      data: JSON.stringify(u),
      contentType: "application/json",
      success: function (response) {
        console.log(response);
        if (response._id != undefined) {
          window.location.replace("/dashboard");
        } else {
          $("#alert-danger").text("Incorrect credentials");
          $("#alert-danger").show();
        }
      },
      error: function (xhr, status, error) {
        var errorMessage = xhr.status + ": " + xhr.statusText;
        alert("Error - " + errorMessage);
      },
    });
  }
});
```

Code Snippet 1.2 – click event for "Login" button (main.js - 34)

Registration

The "Not a member? Register today" button listens to click events, when clicked it invokes a function which hides the login form (Figure 1.1) and shows the registration form (Figure 1.2), where the user can provide their registration credentials. That view appears as below:

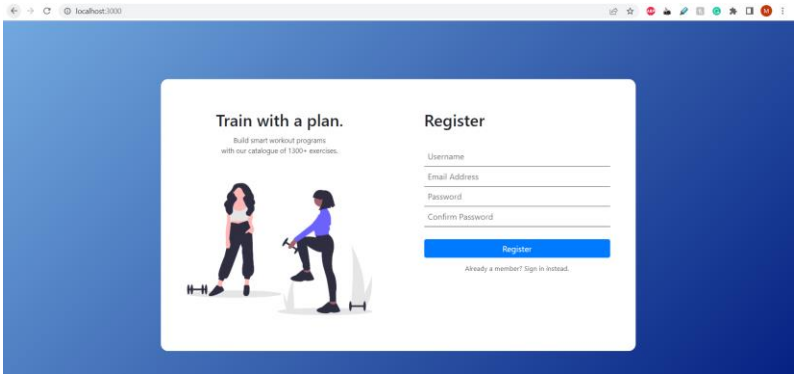


Figure 1.2

The JS code snippet for registering this click event is as below:

```
$("#login-register-toggle-button").click(function (event) {
  event.preventDefault();
  $("#register-form").show();
  $("#login-form").hide();

  $("#alert-danger").hide();
  $("#alert-success").hide();
});
```

Code Snippet 1.3 – Click event for "Register" button (main.js – 5)

Below is the HTML snippet of how the above form was implemented.

```
<h2>Register</h2>
<br />
<form>
  <input
    class="text-input"
    type="text"
    id="register-username-input"
    placeholder="Username"
  />
  <br />
  <input
    class="text-input"
    type="text"
    id="register-email-input"
    placeholder="Email Address"
  />
  <br />
  <input
    class="text-input"
    type="password"
    id="register-password-input"
    placeholder="Password"
  />
  <br />
  <input
    class="text-input"
    type="password"
    id="confirm-password"
    placeholder="Confirm Password"
  />
  <br />
  <br />
  <button
    id="submit-register"
    class="btn btn-primary full-width-button"
  >
    Register
  </button>
</form>
```

Code Snippet 1.4 - HTML implementation of the registration form

Similarly, a click event has been added to the "Register" button (on Figure 1.2), which when clicked invokes a function that validates the provided credentials and if valid, registers the user. If the credentials are correct we add the user to our *user* database. This is facilitated via an ajax POST request. Again, there is some friend-end validation such that none of the fields can be empty and both the passport inputs must match. It also validates the user email to ensure a correctly formatted email is provided.

Note: If the email address already exists, registration will not be successful (the email address serves as the PK to the *user* collection)

The JavaScript code snippet for this can be found in main.js lines 84-144(not attached due to its length).

Google sign-in

Opting to sign in using Google account takes the user to the respective Google sign-in page:

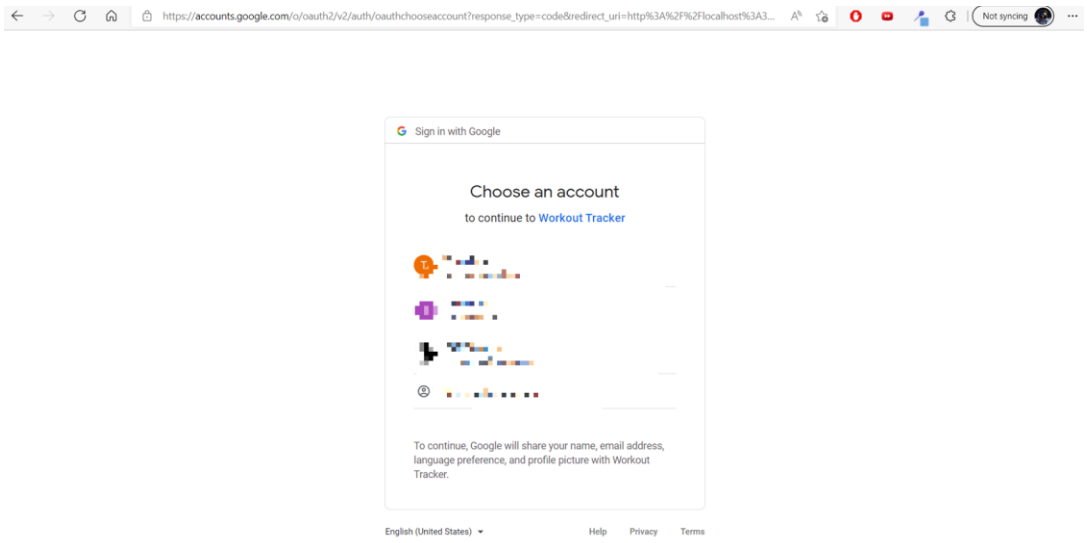


Figure 1.3 – Google sign-in

Again, a click event is added to the "Sign in using Google" button which redirects the user to the Google sign-in page when the button is clicked. The JavaScript snippet for this is attached hereby:

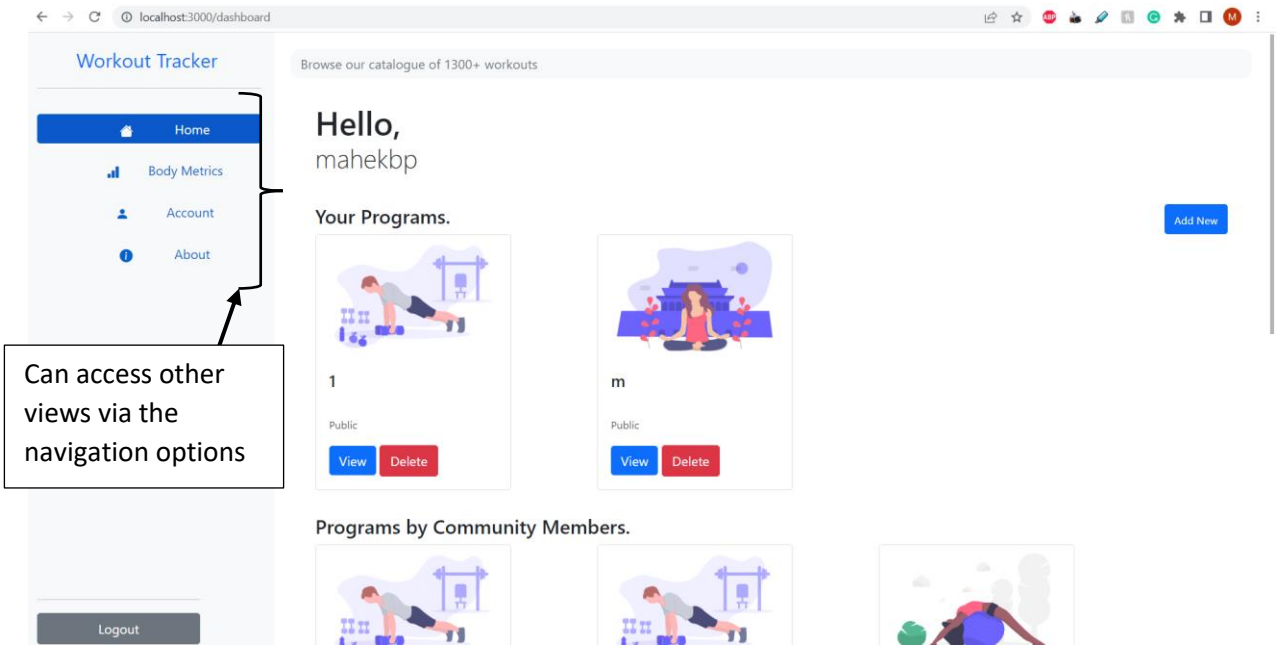
```
$("#google-button").click(function (event) {
  event.preventDefault();
  window.location.replace("/auth/google");
});
```

Code snippet 1.5 – Click event for Google Sign in button (main.js – 149)

The dashboard

The dashboard provides a general view of the whole application. It is from here that the user can access and view all the other views provided by our application. By default, the dashboard shows the home page, but there are navigation buttons on the left side which allow the user to navigate through the different views and aspects of the application.

General view of the dashboard.



The HTML components used to build this dashboard are found on the dashboard.html file.

The Home, Body Metrics, Account, About and WorkoutProgram views can all be accessed via this dashboard. Hence rather than analyzing the dashboard as a whole, we will analyze the specific views that can be accessed via the dashboard in the following pages.

2. Home page/view.

The home page is the main view for the user once they log in. It shows a brief greeting based on the user's username. On this view, the user can also use the search bar at the top to search for an exercise of their choice.

Just beneath this, the user can view all the workout programs created by them. The user can scroll down and also view the "Program by Community Members" which includes all the public workout programs created by other users. The user can view (but not edit) these workout programs as they are public.

This view provides the following functionalities for the user:

- *The user can view exercises – they can either search for single specific exercises, or they may search for a range of different exercises providing some exercise details such as equipment, target muscle and body part.*
- *The user can create their own custom made workout program, and give it a name.*
- *They can set the workout program to either public (can be viewed by others) or private (can only be viewed by user themselves).*
- *After creating a workout program, the user may delete it if they want.*
- *The user can view all the workout programs created by them.*
- *The user can view workout programs created by others which are public.*

The following components interact together to achieve this view;

1) HTML template : dashboard.html

2) JS template : dashboard.js, home.js

3) CSS template : in-line CSS and style.css

General View

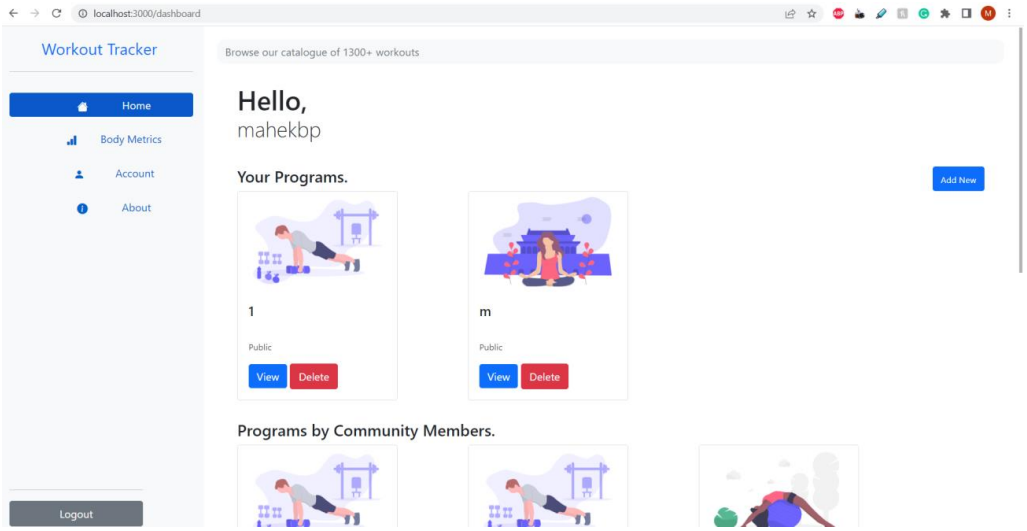


Figure 2.1 – General view of home page

HTML Aspect

The Home view/page has mainly been built using flexbox, divs, search input types, buttons and headers. Further details may be found on dashboard.html, lines 97-299

Making the page ready

We use the .isReady() of JQuery to make the homepage ready whenever the user elects to view it. When the user clicks the home page, or when the user logs in, we first make an ajax GET request that gets the currently logged in user's information. We then display a small greeting based on the user's username.

We then call the helper function getWorkoutProgramsByUser() which retrieves and displays all the workout programs made by the currently logged in user. These workout programs will be displayed under "Your Programs". Each of them contains an a-tag which appears to the user as a "View" button. Clicking this takes the user to another page where they can view and modify this workout program. Since these workout programs are made by the user, a "Delete" button is also added that deletes the workout program when clicked.

We then make another ajax GET request that retrieves all the public workout programs. We then add each one of them under the "Programs by Community Members" section. Since these are public, they are view-only so they only have a "View" button and no "Delete" button.

This process will be done whenever we enter the home page/view. The JavaScript implementation for this is as below:

```
$(document).ready(function () {
  $.ajax({
    url: "/user",
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      $("#greeting-username").text(response.username);
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });

  getWorkoutProgramsByUser();

  $.ajax({
    url: "/workoutprogram",
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      console.log(response.length);
      response.forEach((element) => {
        $("#public-programs").append(
          `<div class="card" style="width: 15rem;">\
              <div style="height: 150px; background-image: linear-gradient(135deg, #72a9df, #021E80)"></div>\
              <div class="card-body">\
                <h5 class="card-title program-name">${element.nameOfProgram}</h5>\
                <a href="/workout-program/${element._id}" target="_blank" class="btn btn-primary">View</a>\
              </div>\
            </div>`
        );
      });
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
});
```

Code Snippet 2.1 - Making the home page ready(home.js – 71)

The helper function getWorkoutProgramsByUser() makes an ajax GET request retrieving all the workout programs made by the user. If the user has none, a text message is displayed, else the workout programs belonging to the user are displayed in a card fashion. Since here we are referring to workout programs created by the user, we have both a "View" a-tag and a delete button. The delete button stores the workout program id as its value. It listens for click events and when clicked it calls the deleteProgram(objButton) function which will delete the workout program. The JavaScript implementation for this is as below:

```
function getWorkoutProgramsByUser() {
  $.ajax({
    url: "/user/workoutprograms",
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      $("#programs-by-user").empty();
      if (response.length == 0) {
        $("#programs-by-user").text("You do not have any programs created");
      } else {
        response.forEach((element) => {
          $("#programs-by-user").append(
            `<div class="card" style="width: 15rem; overflow: hidden">\
              <div style="height: 150px; background-image: linear-gradient(135deg, #72a9df, #021E80)"></div>\
              <div class="card-body">\
                <h5 class="card-title program-name">${
                  element.nameOfProgram
                }</h5>\
                <p class="card-text program-link" style="font-size: small; opacity: 0.9;">${
                  element.isPublic ? "Public" : "Private"
                }</p>\
                <a href="/workout-program/${
                  element._id
                }" target="_blank" class="btn btn-primary">View</a>\
                <button class="btn btn-danger" value=${
                  element._id
                } onclick="deleteProgram(this)">Delete</button>\
              </div>\
            </div>`
          );
        });
      }
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
}
```

Code Snippet 2.2 - Gets all the workout programs by the user(home.js – 25)

```
function deleteProgram(objButton) {
  $.ajax({
    url: "/workoutProgram/" + objButton.value,
    type: "DELETE",
    contentType: "application/json",
    success: function (response) {
      getWorkoutProgramsByUser();
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
}
```

Code Snippet 2.3 – deleteProgram() function which deletes a workout program (home.js - 5)

Searching exercises (keyword search)

The user can search for an exercise of their choice by typing on the exercise search input bar.

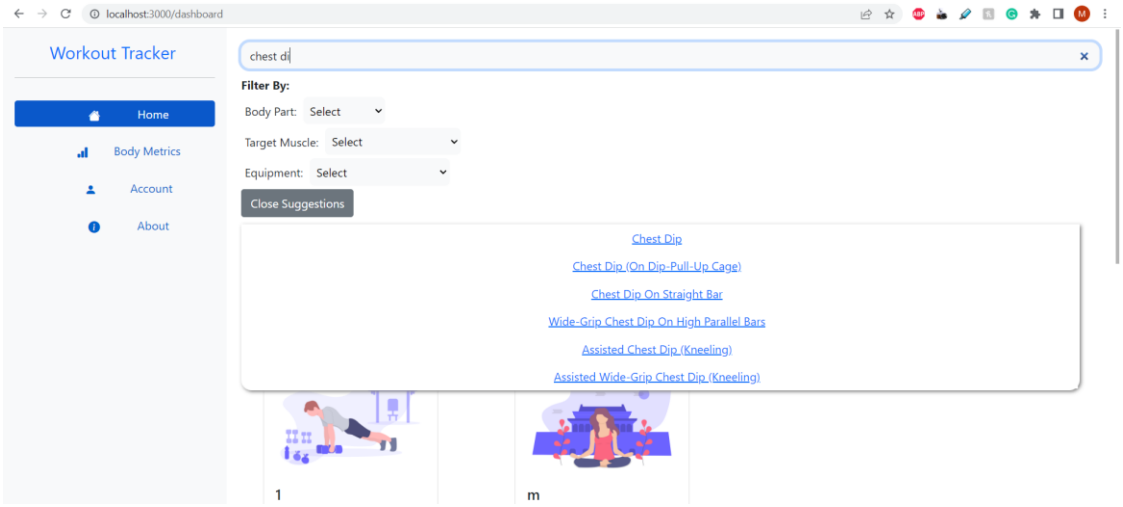


Figure 2.2 – Searching exercise's on the search bar

This search input bar along with the suggestion panel were implemented in HTML as follows:

```
<div id="render-area">
  <div id="home" style="display: inline">
    <div id="search-bar-container">
      <div class="input-group rounded">
        <input
          type="search"
          id="search-bar"
          class="form-control rounded"
          placeholder="Browse our catalogue of 1300+ workouts"
          aria-label="Search"
          aria-describedby="search-addon"
          style="
            border: none;
            background-color: #f8f9fa;
            border-radius: 12px !important;
          "
        />
      </div>
    </div>
    <div
      id="search-bar-suggestion-container"
      style="
        display: flex;
        flex-direction: column;
        position: absolute;
        background-color: white;
        min-width: 75vw;
        z-index: 1;
        max-height: 100%;
        overflow-y: scroll;
        box-shadow: 2px 2px 5px grey;
        border-bottom-right-radius: 14px;
        border-bottom-left-radius: 14px;
      "
    ></div>
```

Code Snippet 2.4 – HTML implementation of the search bar

When the user types input on the search bar, an event handler is attached to this and thus a function is invoked that generates suggestions when more than 1 character is inputted – it displays the matching exercises. The JavaScript implementation for this is as below:

```
$("#search-bar").on("keyup", function () {
  $("#filters").show();
  searchBarInput = $(this).val().toLowerCase();
  if (searchBarInput.length > 1) {
    filterExercises();
  } else {
    $("#search-bar-suggestion-container").empty();
  }
});
```

Code Snippet 2.5 - Searching for exercises on the Home page search bar(dashboard.js – 120)

This in turn calls the helper function filterExercises(). This function builds a list of exercises whose name matches the input provided by the user. JavaScript implementation for this is as below:

```
function filterExercises() {
  let filteredList = exercises.filter((element) => {
    let nameMatch = element.name.includes(searchBarInput);

    let bodyPartMatch = true;
    let equipmentMatch = true;
    let targetMuscleMatch = true;

    if (bodyPartInput != 'none'){
      bodyPartMatch = bodyPartInput == element.bodyPart;
    }
    if (equipmentInput != 'none'){
      equipmentMatch = equipmentInput == element.equipment;
    }
    if (targetMuscleInput != 'none'){
      targetMuscleMatch = targetMuscleInput == element.target;
    }
    return nameMatch && bodyPartMatch && equipmentMatch && targetMuscleMatch;
  });
  updateSuggestions(filteredList);
}
```

Code Snippet 2.6 – Building a list of exercises whose name match the input searched by the user (dashboard.js – 217)

As can be seen, it makes use of the updateSuggestions(filteredList) helper function. This function appends all the matching exercises in the filteredList to the suggestion bar. Exercises are appended in the form of buttons that display the exercise name and store the exerciseID as their values.

```
function updateSuggestions(list) {
  $("#search-bar-suggestion-container").empty();
  list.forEach((element) => {
    $("#search-bar-suggestion-container").append(`
    <button type="button" class="btn btn-link exercise-suggestion" value="${element.id}" onclick="handleSuggestionSelect(this)">${element.name}</button>
    `);
  });
}
```

Code Snippet 2.7 - Updating exercise search suggestions(dashboard.js – 92)

Although in the suggestion bar, the exercises appear as text to the user, they are implemented as buttons(as can be seen above). Every exercise button on the suggestion panel listens for click events. When clicked, it displays a modal showing details pertaining to the exercise. The user's view of this is shown below:

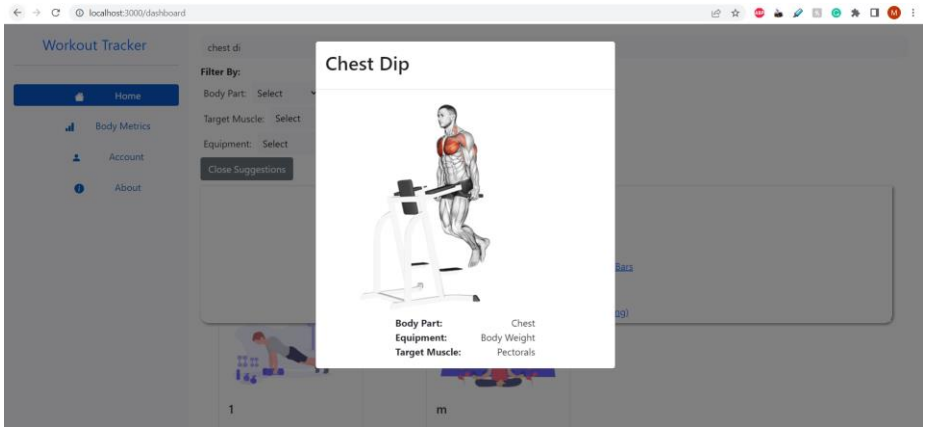


Figure 2.3 – Modal on clicking an exercise

Showing the modals when an exercise is clicked is facilitated by the function handleSuggestionSelect(). This function eventually invokes the updateModal(exerciseID) function which deals with displaying the modal. The exerciseID of the exercise clicked is obtained by getting the value of the button which was clicked.

```
function handleSuggestionSelect(objButton) {
  $('#exerciseInfoModal').modal('toggle')
  updateModal(objButton.value)
}

function updateModal(exerciseID){
  $.ajax({
    url: "/exercise/" + exerciseID,
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      $('#exercise-name').text(response.name);
      $('#exercise-image').attr("src",response.gifUrl);
      $('#body-part').text(response.bodyPart);
      $('#equipment').text(response.equipment);
      $('#target-muscle').text(response.target);
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
}
```

Code Snippet 2.8 - Helper functions that help display the Exercise modals on Home page (dashboard.js – 81,56)

The updateModal function uses the exerciseID and makes an ajax GET request that gets the Exercise object given the exerciseID from our *exercise* collection. If the request was successful, it displays the exercise details on the modal.

The HTML implementation for the modal is pretty simple, it is mainly built using span tags, images and headers, it is as follows:

```
<div
  class="modal fade"
  id="exerciseInfoModal"
  tabindex="-1"
  aria-labelledby="exampleModalLabel"
  aria-hidden="true"
>
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h2 id="exercise-name"></h2>
      </div>
      <div class="modal-body">
        <img id="exercise-image" />
        <div style="margin: auto">
          <div class="exercise-info">
            <span> <strong>Body Part:</strong></span>
            <span id="body-part"></span>
          </div>
          <div class="exercise-info">
            <span><strong>Equipment:</strong></span>
            <span id="equipment"></span>
          </div>
          <div class="exercise-info">
            <span><strong>Target Muscle:</strong></span>
            <span id="target-muscle"></span>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Code Snippet 2.9 – HTML implementation of the Exercise modal

Filtered search of Exercises

One of the main functionalities of our application is allowing users to search for exercises based on their choice of body part, target muscle and equipment. We provide this functionality on the Home page/view.

The user's view for this would be as follows:

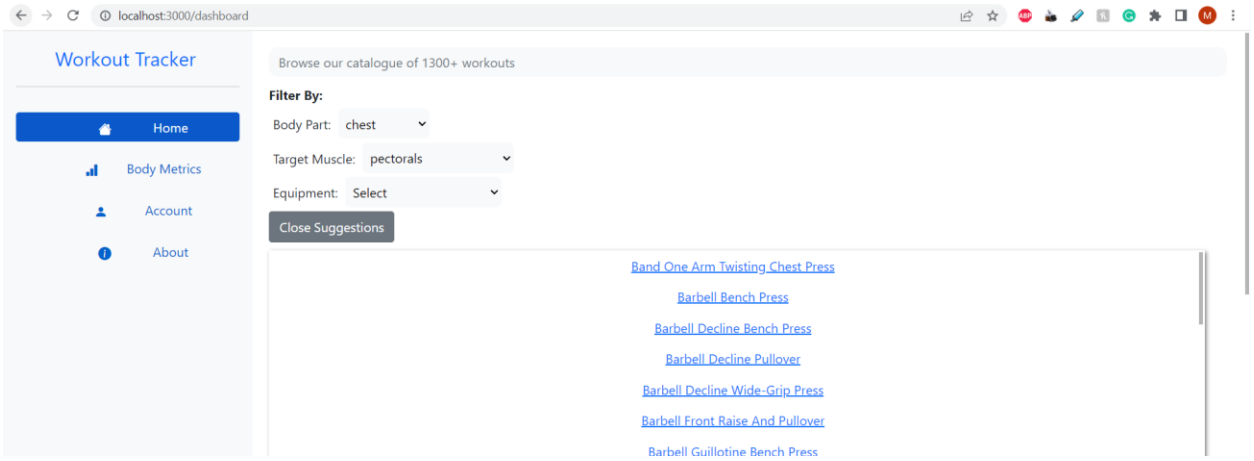


Figure 2.3 Filtered search of Exercises

This was implemented in HTML using select boxes. The HTML snippet for this can be found below.

```
</div>
<div id="filters" style="display: flex !important; flex-direction: row; justify-content: space-evenly; padding: 10px 0px; display: none;">
  <div>
    Filter By:
  </div>
  <div class="filter-container">
    Body Part:
    <select id="body-part-select" class="filter" aria-label="Default select example">
      <option value="none" selected>Select</option>
    </select>
  </div>

  <div class="filter-container">
    Target Muscle:
    <select id="target-muscle-select" class="filter" aria-label="Default select example">
      <option value="none" selected>Select</option>
    </select>
  </div>

  <div class="filter-container">
    Equipment:
    <select id="equipment-select" class="filter" aria-label="Default select example">
      <option value="none" selected>Select</option>
    </select>
  </div>

  <button id="close-suggestions" class="btn btn-secondary">
    Close Suggestions
  </button>
```

Code Snippet 2.9 -HTML implementation of the select boxes

The select boxes are filled in with options representing all the possible body part, target muscle and equipment options we have available. This is done whenever the dashboard is loaded(using .ready() of JQuery), the JavaScript code snippet for this is as below:

```
$(document).ready(function () {
  $.ajax({
    //a GET request to get all the exercises from the exercise collection
    url: "/exercise",
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      exercises = response;
      //get all the possible types of body part, target muscles and equipment available
      exercises.forEach((exercise) => {
        if (!listOfBodyParts.includes(exercise.bodyPart)) {
          listOfBodyParts.push(exercise.bodyPart);
        }
        if (!listOfTargetMuscles.includes(exercise.target)) {
          listOfTargetMuscles.push(exercise.target);
        }
        if (!listOfEquipments.includes(exercise.equipment)) {
          listOfEquipments.push(exercise.equipment);
        }
      });
      //add them as options to the respective select boxes
      listOfBodyParts.forEach((bodyPart) => {
        $("#body-part-select").append(`<option value="${bodyPart}">${bodyPart}</option>`);
      });
      listOfEquipments.forEach((equipment) => {
        $("#equipment-select").append(`<option value="${equipment}">${equipment}</option>`);
      });
      listOfTargetMuscles.forEach((targetMuscle) => {
        $("#target-muscle-select").append(`<option value="${targetMuscle}">${targetMuscle}</option>`);
      });
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
});
```

Code Snippet 2.10 – Function that adds all the possible body part, muscle and equipment options to the select boxes when DOM Is fully loaded (dashboard.js-15)

Whenever the dashboard is loaded, we first make an ajax GET request that gets all the exercises from our *exercise* collection. We then retrieve all the different possible types of body part, target muscle and equipment from all the exercises we have. We then add these as options to their respective select boxes.

As can be seen in the HTML implementation (Code Snippet 2.9), all the select boxes belong to the same class "filter". Whenever a different option is chosen on any of the select box, this updates the suggestions. We use the new selected options and then update the suggestions based on that. The JavaScript code snippet for this is as follows:

```
$(".filter").on("change", function () {
  if (this.id == "body-part-select") {
    bodyPartInput = this.value;
  } else if (this.id == "equipment-select") {
    equipmentInput = this.value;
  } else if (this.id == "target-muscle-select") {
    targetMuscleInput = this.value;
  }
  filterExercises()
});
```

Code Snippet 2.11 – Updating the suggested exercise list on selection of a new option when searching exercises with filters (dashboard.js - 105)

This makes use of the helper function called filterExercises() – this function filters all the exercises based on the options selected. Which ever exercise matches is added to the filteredList.

```
function filterExercises() {
  let filteredList = exercises.filter((element) => {
    let nameMatch = element.name.includes(searchBarInput);

    let bodyPartMatch = true;
    let equipmentMatch = true;
    let targetMuscleMatch = true;

    if (bodyPartInput != 'none'){
      bodyPartMatch = bodyPartInput == element.bodyPart;
    }
    if (equipmentInput != 'none'){
      equipmentMatch = equipmentInput == element.equipment;
    }
    if (targetMuscleInput != 'none'){
      targetMuscleMatch = targetMuscleInput == element.target;
    }
    return nameMatch && bodyPartMatch && equipmentMatch && targetMuscleMatch;
  });
  updateSuggestions(filteredList);
}
```

Code Snippet 2.12 – Filtering all the available exercises based on the user's selected options (dashboard.js - 217)

We then call updateSuggestions() passing the filtered list. This updates the suggested exercises for the user. For the user, every exercise on the suggestion box appears as text, but the implementation for it is as a HTML button whose value is the exerciseID. The JavaScript implementation for this is as follows:

```
function updateSuggestions(list) {
  $("#search-bar-suggestion-container").empty();
  list.forEach((element) => {
    $("#search-bar-suggestion-container").append(`
    <button type="button" class="btn btn-link exercise-suggestion" value="${element.id}" onclick="handleSuggestionSelect(this)">${element.name}</button>
    `);
  });
}
```

Code Snippet 2.13 - Updating exercise search suggestions(dashboard.js – 92)

Whenever the button is clicked, it calls the handleSuggestionSelect(this), this shows a modal displaying the exercises details (code Snippet 2.8)

Adding a new workout program

From this view, the user can also add a new workout program by simply clicking the "Add New" button. When clicked it displays a modal asking the user to provide the workout programs name and clicking on a check box whether the program is public or not. The user's view of this would be as follows:

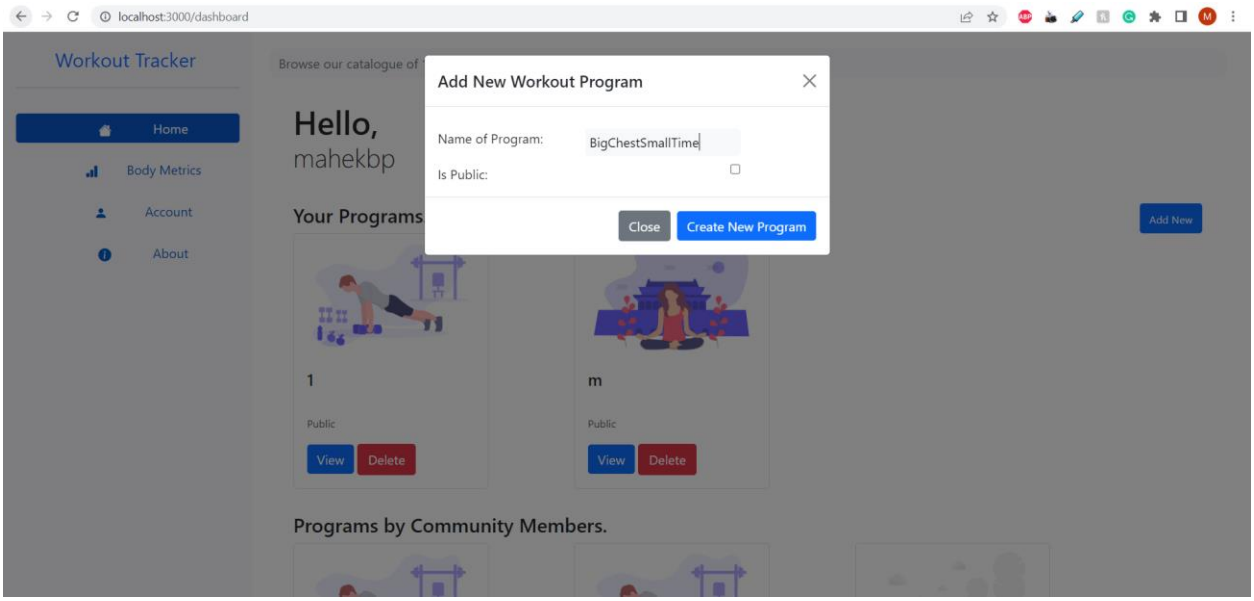


Figure 2.4 – Adding a new Workout Program

The HTML implementation for this modal is as follows:

```
<div
  class="modal fade"
  id="addNewWorkoutProgram"
  tabindex="-1"
  aria-labelledby="exampleModalLabel"
  aria-hidden="true"
>
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLabel">
          Add New Workout Program
        </h5>
        <button
          type="button"
          class="btn-close"
          data-bs-dismiss="modal"
          aria-label="Close"
        ></button>
      </div>
      <div class="modal-body">
        <div class="new-program-row">
          <label for="name-of-program">Name of Program: </label>
          <input
            id="name-of-program-input"
            class="new-program-input"
          />
        </div>
        <div class="new-program-row">
          <label for="is-public">Is Public: </label>
          <input
            id="is-public-input"
            class="new-program-input"
            type="checkbox"
            value="1"
          />
        </div>
      </div>
      <div class="modal-footer">
        <button
          type="button"
          class="btn btn-secondary"
          data-bs-dismiss="modal"
        >
          Close
        </button>
        <button
          id="add-new-program"
          type="button"
          class="btn btn-primary"
        >
          Create New Program
        </button>
      </div>
    </div>
  </div>
</div>
```

Code Snippet 2.14 – HTML implementation of Add new workout program modal

Once the user has provided the workout program details, they can click the "Create New Program" button which listens for click events. When clicked, it first validates the inputted information and if valid, it makes an ajax POST request to enter the workout program into our *workout-program* collection.

```
$("#add-new-program").click(function (event) {
  event.preventDefault();

  let workoutProgram = {};
  if ($("#name-of-program-input").val() != "") {
    workoutProgram.nameOfProgram = $("#name-of-program-input").val();
    if ($("#is-public-input").prop("checked")) {
      workoutProgram.isPublic = 1;
    } else {
      workoutProgram.isPublic = 0;
    }
    $.ajax({
      url: "/workoutProgram",
      type: "POST",
      data: JSON.stringify(workoutProgram),
      contentType: "application/json",
      success: function (response) {
        getWorkoutProgramsByUser();
        $("#addNewWorkoutProgram").modal("hide");
      },
      error: function (xhr, status, error) {
        var errorMessage = xhr.status + ": " + xhr.statusText;
        alert("Error - " + errorMessage);
      },
    });
  }
});
```

Code Snippet 2.15 – Click event for "Create New Program button (home.js – 119)

3. Workout Program View

As mentioned above, users can add and/or remove exercises of their choice to a workout program. They can then refer to this workout program whenever they want to.

The workout program view is the view once the user clicks the "View" button on a WorkoutProgram. This takes the user to a new page, where they can view the workout program i.e. the exercises (along with their frequencies) that make up the workout program. Due to workouts being public or private, there are actually 2 types of workout Program views – public and private view. The key difference being that the private workout programs are modifiable by user who created them while the public ones are view only.

It is on this view that the user can achieve the following functionalities:

- *They can set the workout program to either public (can be viewed by others) or private (can only be viewed by user themselves)*
- *The user can update the workout program details if they want to.*
- *The user can add exercise to the workout program, specifying how many repetitions and sets per exercise added to their workout program.*
- *The user can also remove/delete exercises from their workout programs.*

3.1) Workout Program view/page (Private workout programs)

The following components interact together to achieve this view;

1) HTML template : workout-program.html

2) JS template : workout-program.js (for private) else public-workout-program.js (for public)

3) CSS template : in-line CSS, style.css

When the user clicks "View" on a workout program they made, this is their view:

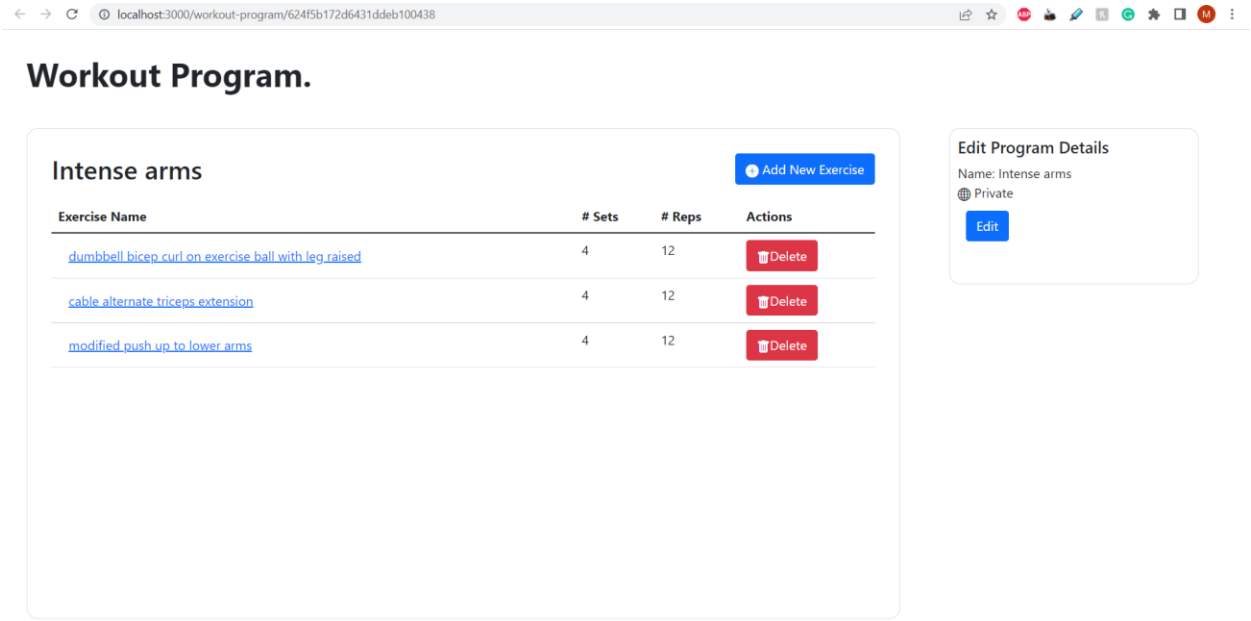


Figure 3.1 – User view of a private workout program

They can clearly see details pertaining to their workout program. They can see a table with the exercise names, reps and sets. On the right they also have an option to edit details of the workout program.

The full details of the HTML to build this can be found on the workout-program.html file.

NOTE:

As can be seen, the workout program view contains a table with exercise names and delete buttons. The JavaScript + HTML implementation of this table is crucial for the delete and view exercise functionality, and hence we feel it is important to mention. It is as follows:

```
function updateTable() {
  $("#exercise-table-body").empty();
  let programID =
    window.location.pathname.split("/")[window.location.pathname.split.length];
  $.ajax({
    url: "/workoutProgram/" + programID,
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      updateWorkoutProgramInfo(response);
      response.exercises.forEach((element) => {
        console.log(element);
        $("#exercise-table-body").append(
          `
          <tr>
          <td>
            <button type="button" class="btn btn-link" value=${element.exercise.id} onclick="handleExerciseClick(this)">
              ${element.exercise.name}
            </button>
          </td>
          <td>${element.numSets}</td>
          <td>${element.numReps}</td>
          <td>
            <button type="button" class="btn btn-danger" value="${element.exercise.id}" onclick="handleDelete(this)">
              <i class="bi bi-trash-fill"></i>Delete
            </button>
          </td>
        </tr>
          `
        );
      });
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
}
```

Code Snippet 3.1 – This function makes and updates the table on addition/deletion of exercises (workout-program.js – 102) (includes the HTML components)

This function is called whenever we add and delete an exercise.

When called it retrieves the current WorkoutProgramID from the endpoint. Using this, it then makes an ajax GET request that gets the current workout program from the server. If the request is successful, it adds every exercise in the workout program to the table – every table row contains two buttons – one representing the exercise name (when clicked, shows the exercise modal), and the other allowing for deletion of the exercise (when clicked, deletes the exercise from the workout program). **Both the buttons store the exerciseID of the exercise they represent as their values.**

Adding an exercise.

When the user clicks the "Add New Exercise" button, it shows a modal asking the user to search and select an exercise and then add it to the program specifying its reps and sets.

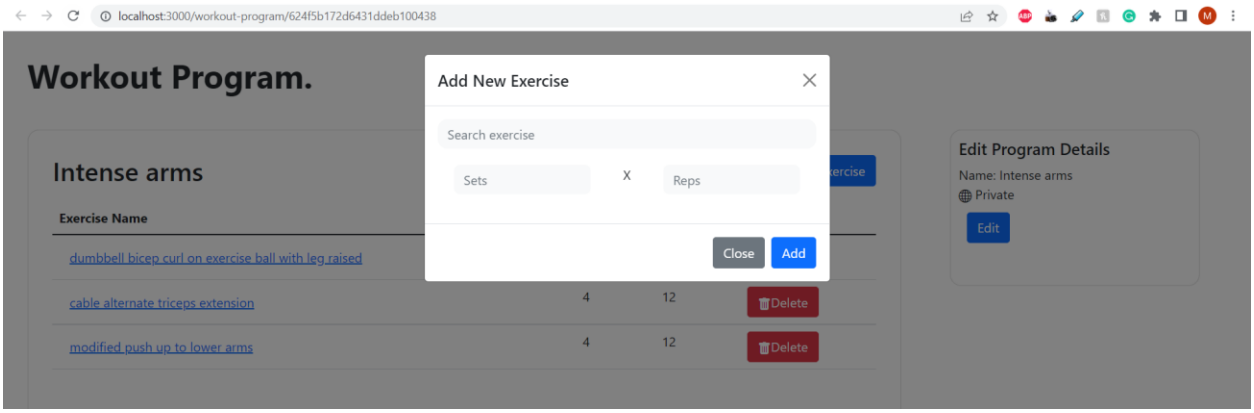


Figure 3.2 – User's view of adding an exercise to a workout program

The user can then search an exercise of their choice by typing its name on the "Search exercise" input panel. As soon as the user types 2 characters, we filter all the available exercises for the user's input and then add all the matching exercises to a list. This then calls a helper method updateSuggestions(list) which deals showing with data on the suggestion panel. The JavaScript code for this is as below:

```
$("#add-exercise-search").on("keyup", function () {
  var value = $(this).val().toLowerCase();
  if (value.length > 1) {
    let filteredList = exercises.filter((element) => {
      return element.name.includes(value);
    });

    updateSuggestions(filteredList);
  } else {
    $("#suggestion-container").empty();
  }
});
```

Code Snippet 3.2 – The search bar listens for key up events (workout-program.js – 169)

The updateSuggestions(list) function adds every matching exercise to the suggestion panel. It adds them as HTML buttons which listen for click events, when clicked we display a modal showing the exercise's details. Below is the JavaScript implementation of this function:

```
function updateSuggestions(list) {
  $("#suggestion-container").empty();
  list.forEach((element) => {
    $("#suggestion-container").append(`
    <button type="button" class="btn btn-link exercise-suggestion" value="${element.id}" onclick="handleSuggestionSelect(this)">${element.name}</button>
    `);
  });
}
```

Code Snippet 3.3 – the updateSuggestion(list) function – includes HTML implementations (workout-program.js – 78)

When the user clicks an exercise of their choice from the suggestion list, this invokes handleSuggestionSelect(objButton) which assigns the exerciseId of the selected exercise to the global variable selectedID (this represents the ID of the selected exercise). The JavaScript implementation for this function is as below:

```
function handleSuggestionSelect(objButton) {
  $("#suggestion-container").empty();
  selectedID = objButton.value;
}
```

Code Snippet 3.4 – the handleSuggestionSelect(objButton) function (workout-program –8)

This way the user can search, select and view exercises of their choice.

When adding to the workout program, they also need to enter the number of reps and sets for the exercise. They simply need to provide a numerical value in the said input boxes.

When the user is satisfied with their input, they can click the Add button to add the exercise to their workout program. The "Add" button listens for click events. When clicked, it calls a function which first validates the inputs for the number of reps and sets – they must be numeric values. It then gathers all the exercise related information and obtains the ID of the workout program (from endpoint).

Finally, it makes an ajax PUT request sending the exercise related information to be added to the current workout program. Once this is done, it calls the updateTable() (code snippet 3.1) function that updates the workout program table, reflecting the newly added exercise. The JavaScript implementation for this is as below:

```
$("#add-exercise").click(function (event) {
  event.preventDefault();
  if (
    $("#sets-input").val() != "" &&
    $("#reps-input").val() != "" &&
    selectedID != null
  ) {
    let exerciseData = {};
    exerciseData.exerciseID = selectedID;
    exerciseData.numSets = $("#sets-input").val();
    exerciseData.numReps = $("#reps-input").val();
    let programID =
      window.location.pathname.split("/")[
        window.location.pathname.split.length
      ];
    $.ajax({
      url: "/workoutProgram/addExercise/" + programID,
      type: "PUT",
      data: JSON.stringify(exerciseData),
      contentType: "application/json",
      success: function (response) {
        updateTable();
      },
      error: function (xhr, status, error) {
        var errorMessage = xhr.status + ": " + xhr.statusText;
        alert("Error - " + errorMessage);
      },
    });
  } else {
    alert("Please fill in all details");
  }
});
```

Code Snippet 3.5 – Click event for "Add exercise" button (workout-program.js – 186)

Deleting an exercise

Since this view shows the workout programs made by the users, the user can also delete/remove exercises from their programs. They simply need to click the "Delete" button beside every exercise, this will successfully remove the exercise from the workout program.

The delete button also listens for click events, when clicked it calls the handleDelete(objButton) function that first gets the workout programs id (from endpoint), and then makes an ajax PUT request that removes the exercise from the said workout program.

As mentioned previously, the id of the exercise to remove is attained as the value of the Delete button. Every delete button's value represents the id of its corresponding exercise – this was done as it makes the code more compact and reusable.

The JavaScript code for this handling deletion is as follows:

```
function handleDelete(objButton) {
  $("#suggestion-container").empty();
  let programID =
    window.location.pathname.split("/")[window.location.pathname.split.length];
  $.ajax({
    url:
      "/workoutProgram/removeExercise/" +
      programID +
      "?exerciseID=" +
      objButton.value,
    type: "PUT",
    contentType: "application/json",
    success: function (response) {
      console.log(response);
      updateTable();
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
}
```

Code Snippet 3.6 – Click event for "Delete" exercise button (workout-program.js – 48)

Updating workout program details.

Since this view shows the workout program made by the user, they can update their workout program name and visibility from this view.

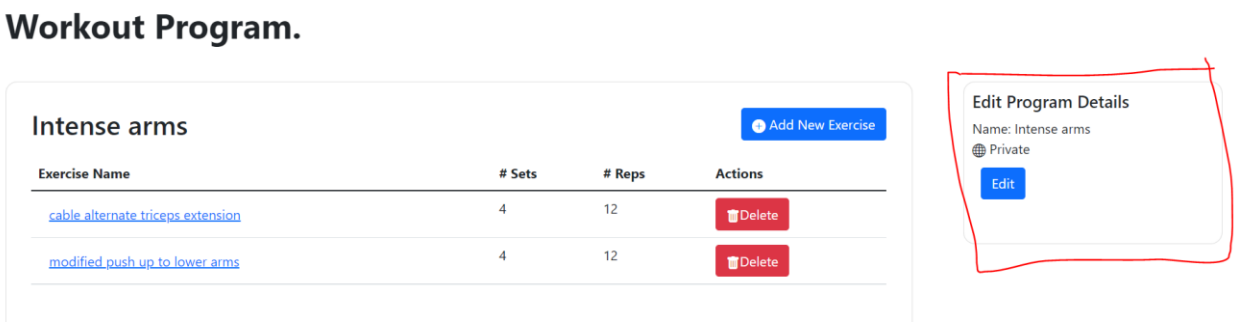


Figure 3.3 – User's view for editing their workout program's details

The "Edit" button listens to click events, when clicked it hides the view end of the workout program details and shows the edit panel. The JavaScript implementation for this is as follows:

```
$("#edit-program-info").click(function (event) {
  event.preventDefault();
  $("#view-side-panel").hide();
  $("#edit-side-panel").show();
});
```

Code Snippet 3.7 – Click event for the "Edit" button on the private workout program view (workout-program.js – 230)

The user can then go on and update the workout program name and visibility.

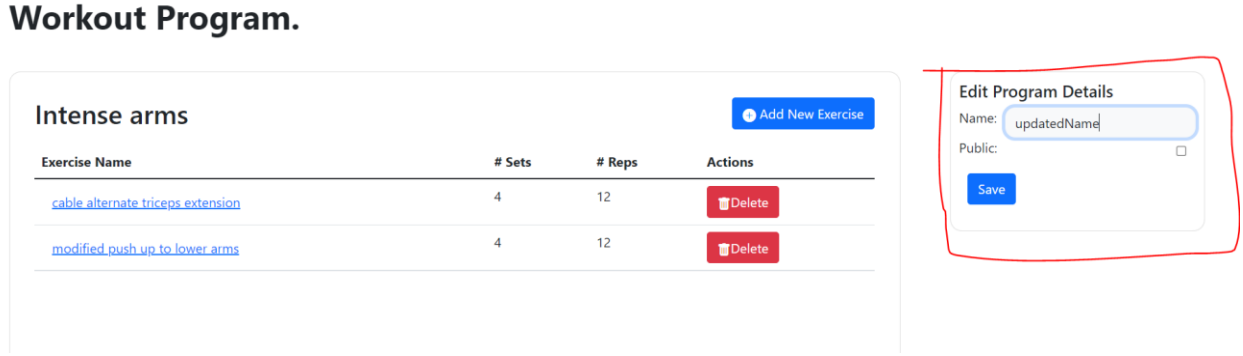


Figure 3.3 – User's view when editing their workout programs details

Once updated details are provided, the user can click the "Save" button, this button also listens for click events. When clicked, it invokes a function that first validates the updated details and gets the workout programs ID (from endpoint). If valid, it makes an ajax PUT request that updates details pertaining to the workout program. The JavaScript implementation of this is as follows:

```
$("#save-program-info").click(function (event) {
  event.preventDefault();
  let workoutProgram = {};
  if ($("#name-side-panel-input").val() != "") {
    workoutProgram.nameOfProgram = $("#name-side-panel-input").val();
    if ($("#is-public-side-panel-input").prop("checked")) {
      workoutProgram.isPublic = 1;
    } else {
      workoutProgram.isPublic = 0;
    }
    let programID =
    window.location.pathname.split("/")[window.location.pathname.split.length];
    $.ajax({
      url: "/workoutProgram/" + programID,
      type: "PUT",
      data: JSON.stringify(workoutProgram),
      contentType: "application/json",
      success: function (response) {
        console.log(response);
        updateTable();

        $("#view-side-panel").show();
        $("#edit-side-panel").hide();
      },
      error: function (xhr, status, error) {
        var errorMessage = xhr.status + ": " + xhr.statusText;
        alert("Error - " + errorMessage);
      },
    });
  } else {
  }
});
```

Code Snippet 3.8 – Click event for the "Save" button when editing workout program details (workout-program.js – 240)

Viewing the exercises.

As has been mentioned, our application caters for how exercises should be performed. A very simple yet effective approach for this was to use GIF images showing how the exercise should be performed correctly.

On the user's end, they need to click the exercise name from their table of exercises

This is how it would appear to the user:

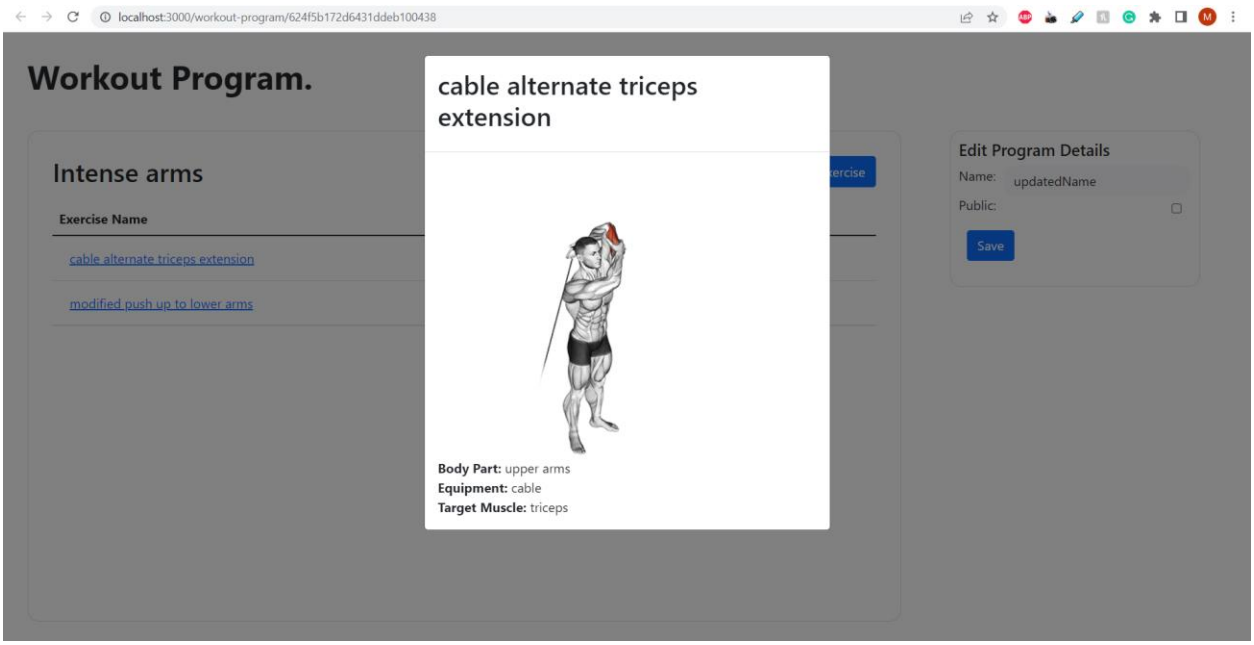


Figure 3.4 – User view when clicking an Exercise on the workout program

As mentioned previously, the exercise names are implemented as HTML buttons, and they hold the exercise id as their values. These buttons listen to click events. When clicked, this calls the handleExerciseClick(objButton) function. This then makes an ajax GET request to the server which retrieves all exercise related information, given the exerciseID. This information is then displayed to the user as seen above. The JavaScript implementation is as below:

```
function handleExerciseClick(objButton){
  $.ajax({
    url: "/exercise/" + objButton.value,
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      $('#exercise-name').text(response.name);
      $('#exercise-image').attr("src",response.gifUrl);
      $('#body-part').text(response.bodyPart);
      $('#equipment').text(response.equipment);
      $('#target-muscle').text(response.target);

      $('#exerciseInfoModal').modal('toggle')
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
}
```

Code Snippet 3.9 – Click event for Exercises on the workout program (workout-program.js – 18)

3.2) Workout Program view/page (Public workout programs)

This view comes in place when a user elects to view a public workout from their home page.

To do so, they click the "View" button on the workout program. When this button is clicked, this opens a new page displaying the public workout programs details.

Here, we make use of .ready() feature of JQuery, hence whenever the user elects to view a public workout, the following piece of JavaScript code will be implemented

```
function updateTable() {
  $("#exercise-table-body").empty();
  let programID =
    window.location.pathname.split("/")[window.location.pathname.split.length];
  $.ajax({
    url: "/workoutProgram/" + programID,
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      $("#name-of-program").text(response.nameOfProgram);
      response.exercises.forEach((element) => {
        $("#exercise-table-body").append(
          `
          <tr>
          <td>${element.exercise.name}</td>
          <td>${element.numSets}</td>
          <td>${element.numReps}</td>
        </tr>
          `
        );
      });
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
}

$(document).ready(function () {
  updateTable();
});
```

Code Snippet 3.10 – updates the table when this view is loaded (public-workout-program.js – 34)

We retrieve the ID of the workout program (from endpoint) and make an ajax GET request requesting details of the selected workout program. We then display every exercise in this public workout program along with their reps and sets to the user. Similar to the private workout program exercises, the user may click the exercise to view a modal further describing the exercise.

4. Body Metrics page/view

On the dashboard, in addition to the other views, we also have a Body Metric's view. This view allows us to achieve the following functionality:

Every user account has user-related information such as their age, height, weight, gender and target weight. The user may view their details and may also update these accordingly whenever they wish to.

Our application is fitness based and hence users would want to see their goals and progress. This Body Metric's page displays the user's secondary information such as their age, gender, height, weight and goal weight.

As user's workout, their weight and goal weight are likely to change. Hence, we allow the user to update all their secondary information. Changing the Gender was also included as we wanted to allow for equality.

In addition to that, based on their height and weight, we show the user their Body Mass Index (BMI) along with a small label that indicates the state of the user's BMI – whether it is normal, over, or under. This is indeed a very practical functionality that would be desired by the user. Additionally, we also show how far behind or ahead the user is from their target weight.

The following components interact together to achieve this view;

1) HTML template : dashboard.html

2) JS template : body-metrics.js

3) CSS template : in-line CSS and style.css

Below is how this view would look like to the user.

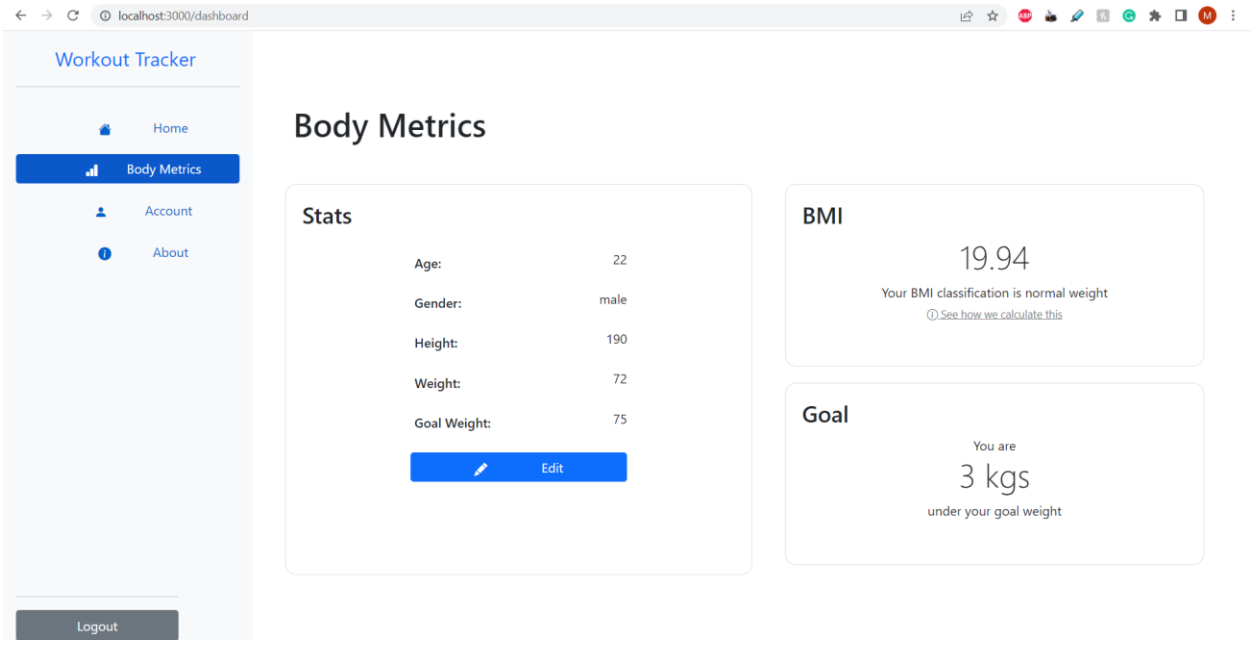


Figure 4.1 – User's view of Body Metrics page

The HTML aspect for this view is based on a flexbox and is built mainly using span tags and buttons. The details for this can be found from line 300-465 of dashboard.html.

```html
<div id="view-body-metrics">
  <span class="metric-row">
    <label for="age" class="metric-label">Age: </label>
    <span id="age"></span>
  </span>
  <span class="metric-row">
    <label for="gender" class="metric-label">Gender: </label>
    <span id="gender"></span>
  </span>
  <span class="metric-row">
    <label for="height" class="metric-label">Height: </label>
    <span id="height"></span>
  </span>
  <span class="metric-row">
    <label for="weight" class="metric-label">Weight: </label>
    <span id="weight"></span>
  </span>
  <span class="metric-row">
    <label for="goal-weight" class="metric-label"
      >Goal Weight:
    </label>
    <span id="goal-weight">80 kg</span>
  </span>
  <button
    id="edit-body-metrics-button"
    class="btn btn-primary metric-row"
    style="margin-top: 20px; justify-content: space-evenly"
  >
    <i class="bi bi-pencil-fill"></i>
    Edit
  </button>
</div>
```

Code Snippet 4.1 – HTML implementation of the Stats panel in Figure 4.1

If the user wishes to, they may change their stats by clicking on the "Edit" button. On the JavaScript end, this button listens for a click event. When clicked, it allows the user to edit their metrics/secondary information. We then make an ajax GET request which gets the currently logged in user and displays their current stats, we then allow the stats to be updated. The JavaScript implementation for this is as follows:

```javascript
$("#edit-body-metrics-button").click(function (event) {
  event.preventDefault();
  $("#body-metrics-alert-success").hide()
  $("#body-metrics-alert-danger").hide();
  $("#view-body-metrics").hide();
  $("#edit-body-metrics").show();

  $.ajax({
    url: "/user",
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      $("#age-input").val(response.personalInfo.age);
      $("#gender-input").val(response.personalInfo.gender).change();
      $("#height-input").val(response.personalInfo.height);
      $("#weight-input").val(response.personalInfo.weight);
      $("#goal-weight-input").val(response.personalInfo.goalWeight);

      updateBMI(response.personalInfo.height, response.personalInfo.weight)
      updateGoal(response.personalInfo.weight, response.personalInfo.goalWeight)
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
});
```

Code Snippet 4.1 – click event for "Edit" button on Body metrics page (body-metrics.js - 83)

Once the user has provided the update stats, they can click the "Save" button to save the changes. The Save button also listens for click events, hence when it is clicked, it invokes a JavaScript function that validates all the provides stats and if valid, updates them for the currently logged in user. Updating the stats for the user is done via an ajax PUT request that updates the stats/secondary information for the user.

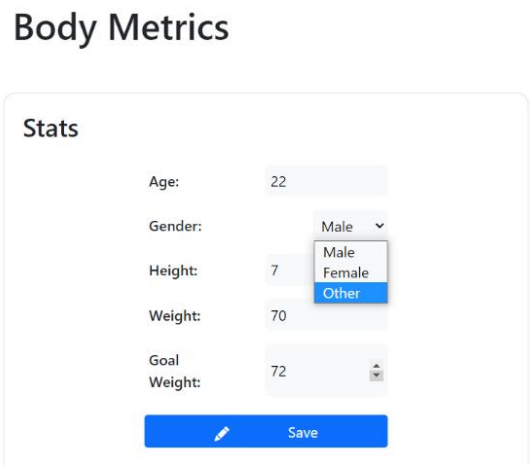


Figure 4.2 – user's view when updating the Body Metrics

The JavaScript implementation for the save button and saving the body metrics changes is as follows:

```
$("#save-body-metrics-button").click(function (event) {
  event.preventDefault();
  $("#body-metrics-alert-success").hide();
  $("#body-metrics-alert-danger").hide();
  $("#view-body-metrics").show();
  $("#edit-body-metrics").hide();

  let bodyMetrics = {};

  bodyMetrics.age = $("#age-input").val();
  bodyMetrics.gender = $("#gender-input").val();
  bodyMetrics.height = $("#height-input").val();
  bodyMetrics.weight = $("#weight-input").val();
  bodyMetrics.goalWeight = $("#goal-weight-input").val();

  console.log(bodyMetrics);
  $.ajax({
    url: "/user/personalInformation",
    type: "PUT",
    data: JSON.stringify(bodyMetrics),
    contentType: "application/json",
    success: function (response) {
      updateMetrics();
      $("#body-metrics-alert-success").text("Your information was updated");
      $("#body-metrics-alert-success").show();
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
});
```

Code Snippet 4.2 – click event for the "Save" button when editing Body Metrics (body-metrics.js – 119)

As can be identified from code snippet 4.2, we make use of 3 internal helper methods that help us update the metrics for the given user. These methods are;

1. updateMetrics() – gets the currently logged in user (using ajax GET request) and then updates the user's stats/secondary information for the front end view. This function calls the following two functions.

```
function updateMetrics() {
  $.ajax({
    url: "/user",
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      $("#age").text(response.personalInfo.age);
      $("#gender").text(response.personalInfo.gender).change();
      $("#height").text(response.personalInfo.height);
      $("#weight").text(response.personalInfo.weight);
      $("#goal-weight").text(response.personalInfo.goalWeight);
      updateBMI(response.personalInfo.height, response.personalInfo.weight)
      updateGoal(response.personalInfo.weight, response.personalInfo.goalWeight)
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
}

$(document).ready(function () {
  updateMetrics();
});
```

Code Snippet 4.3 – reflects the updated body metrics changes on the front end (body-metrics.js – 43)

2. updateBMI() – this function calculates and updates the BMI label for the user.

```
function updateBMI(height, weight) {
  let bmi = Math.round( (weight / (height/100 * height/100)) * 100) / 100;
  $("#bmi-number").text(bmi);
  if (bmi < 18.5){
    $("#bmi-text").text("Your BMI classification is underweight");
  }else if (bmi < 18.5 && bmi < 24.9){
    $("#bmi-text").text("Your BMI classification is normal weight");
  }else if (bmi >25.0){
    $("#bmi-text").text("Your BMI classification is overweight");
  }
}
```

Code Snippet 4.4 – updates the BMI for the user (body-metrics.js – 6)

3. updateGoal() – this function shows how far behind/ahead the user's weight is compared to their goal weight.

```
function updateGoal(weight, goalWeight) {
  let diff = goalWeight - weight;

  $("#goal-number").text(Math.abs(diff) + " kgs");

  if (diff < 0){
    $("#goal-text").text('over');
  }else {
    $("#goal-text").text('under');
  }
}
```

Code Snippet 4.5 – shows how far ahead/behind user's weight is compared to their target weight (body-metrics.js – 27)

Therefore, on saving, the stats are updated and so are the BMI and weight progress box.

5. Account page/view.

On the dashboard, we also have an Account view. As mentioned above, our application is user-based, and users may want to change their primary information (such as their username and password), hence the Accounts view caters for these needs. It helps us achieve the following 2 functionalities;

- *The user can update/change their usernames and passwords, whenever they wish to.*
- *The user can delete their account if they want to*

There are several reasons for users to change their usernames and passwords, security being the main one. As was explained in the previous iteration, we do not allow users to change their email address, as the email address serves as the primary key (PK) for our *user* database.

If a user no longer feels the need to use the application, we provide them the option to delete their account. This removes information related to the user from our databases.

The following components interact together to achieve this view;

1) HTML template : dashboard.html

2) JS template : account.js

3) CSS template : in-line CSS and style.css

General View.

This is how the Account page/view would appear for the user:

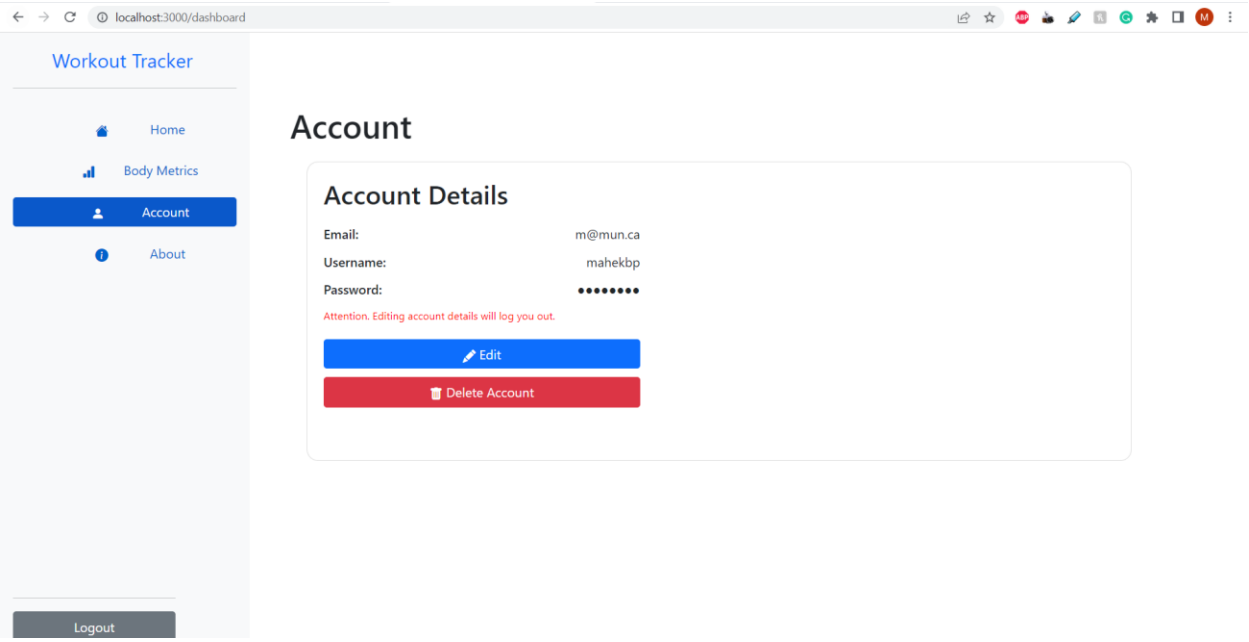


Figure 5.1 – User's view of the Account page

Below is a small HTML code snippet of how the Account details are presented. The complete HTML details on how this view was implemented can be found on dashboard.html, lines 466-620. It was mainly done using divs, labels, span tags and flexbox.

```
<div id="account" style="display: none">
  <div class="heading">
    <h1 style="margin-bottom: 0px">Account</h1>
    <div class="card metrics-card" style="height: 23rem; width: 90%">
      <h2>Account Details</h2>
      <div id="view-account">
        <div class="account-row">
          <label for="email" style="font-weight: 500">Email:</label>
          <span id="email"></span>
        </div>
        <div class="account-row">
          <label for="username" style="font-weight: 500">Username:</label>
          <span id="username"></span>
        </div>
        <div class="account-row">
          <label for="password" style="font-weight: 500">Password:</label>
          <span id="password">●●●●●●●●</span>
        </div>
        <div style="font-size: small; color: red; margin-top: 10px">
          Attention. Editing account details will log you out.
        </div>
```

Code Snippet 5.1– HTML implementation of the Accounts Details panel from Figure 5.1

As was mentioned in the previous iteration, updating the primary information (i.e. username and password) would log the user out. This has been mentioned in a small prompt for the user.

Updating/editing Account details.

The user may edit their primary information. The "Edit" button listens for click events. When clicked, it invokes a function that allows the user to edit their account details. It then makes an ajax GET request which retrieves the currently logged in user, and we show the users email address (unchangeable) and their username. The password is not shown for security reasons. The user can then go on to change their username and password.

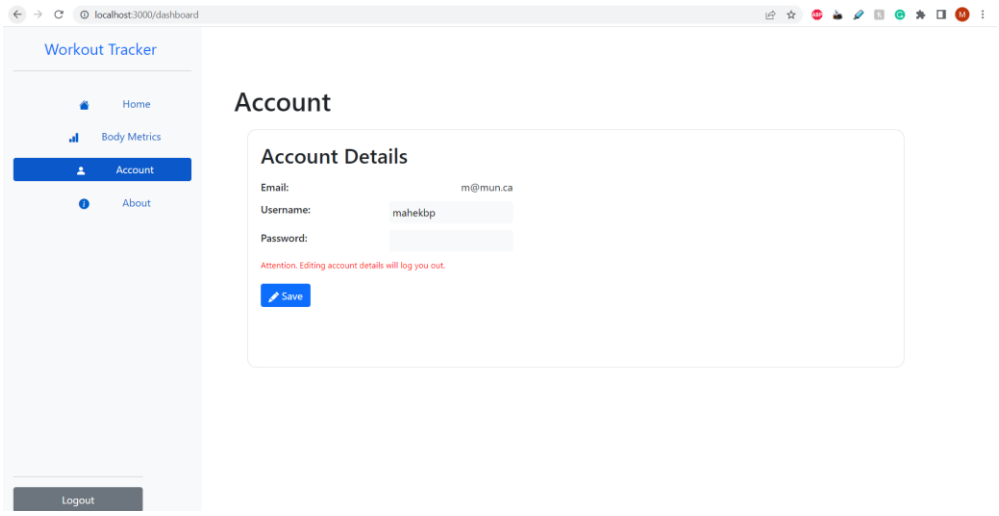


Figure 5.2 – User's view when editing their Account details

The JavaScript code to implement this functionality is as follows:

```
$("#edit-account-button").click(function (event) {
  event.preventDefault();
  $("#account-alert-success").hide();
  $("#account-alert-danger").hide();
  $("#view-account").hide();
  $("#edit-account").show();

  $.ajax({
    url: "/user",
    type: "GET",
    contentType: "application/json",
    success: function (response) {
      $("#username-input").val(response.username);
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
});
```

Code Snippet 5.2 – Click event for the "Edit" button on the Account page (account.js – 27)

Once the user has provided their updated details, they can click the "Save" button. Again, this button listens for click events, when clicked, it first validates the updated username and password. If valid it makes an ajax PUT request sending the updated username and password to the server. The username and password are then updated for the user, and the user is told that they will be logged out in 5 seconds.

```
$("#save-account-button").click(function (event) {
  event.preventDefault();
  $("#account-alert-success").hide();
  $("#account-alert-danger").hide();

  let updatedUser = {};
  if ($("#username-input").val() != "" && $("#password-input").val()) {
    updatedUser.username = $("#username-input").val();
    updatedUser.password = $("#password-input").val();

    $.ajax({
      url: "/user",
      type: "PUT",
      data: JSON.stringify(updatedUser),
      contentType: "application/json",
      success: function (response) {
        $("#account-alert-success").text(
          "Account updated. You will be logged out in 5 seconds"
        );
        $("#account-alert-success").show();
        $("#account-alert-danger").hide();
        setInterval(function () {
          window.location.replace("/");
        }, 3000);
      },
      error: function (xhr, status, error) {
        var errorMessage = xhr.status + ": " + xhr.statusText;
        alert("Error - " + errorMessage);
      },
    });
  } else {
    $("#account-alert-danger").text("Please enter a username and password.");
    $("#account-alert-danger").show();
  }
});
```

Code Snippet 5.3 – Click event for the "Save" button when editing Account details (account.js – 53)

Deleting user account.

This view also provides the user an opportunity to delete their account. To do so they can click the "Delete Account" button. Clicking the button brings up a modal asking them to confirm of their selection. This is how it would appear for the user.

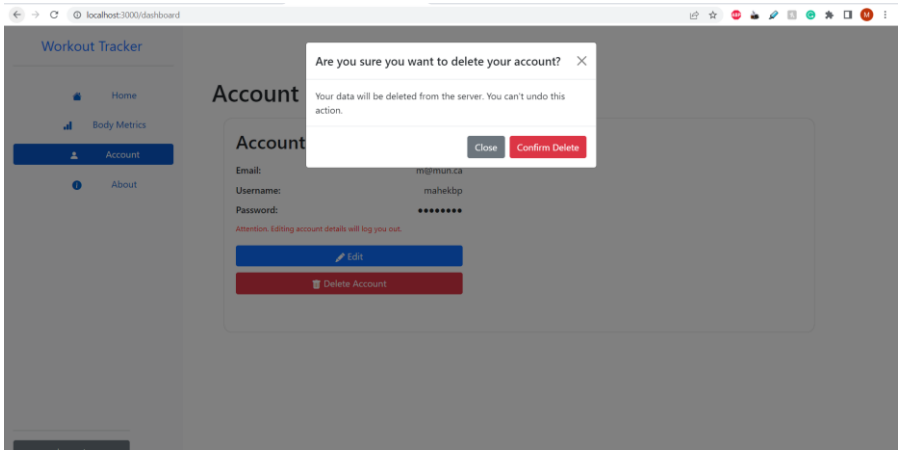


Figure 5.3 – User's view when deleting their account

If the user confirms the deletion by clicking the "Confirm Delete" button, their account will be deleted from the server/system. The "Confirm Delete" button listens for click events. When clicked, it makes an ajax DELETE request, which deletes/removes the user and their related data from the application's databases.

```
$("#confirm-account-delete").click(function (event) {
  $.ajax({
    url: "/user",
    type: "DELETE",
    contentType: "application/json",
    success: function (response) {
      console.log(response);
      window.location.replace("/");
    },
    error: function (xhr, status, error) {
      var errorMessage = xhr.status + ": " + xhr.statusText;
      alert("Error - " + errorMessage);
    },
  });
});
```

Code Snippet 5.4 – Click event for the "Delete Account" when deleting user Account (account.js – 96)

6. About page/view

The about view is a rather simple information related view. Here, we provide general information regarding the application. To the user, it appears as:

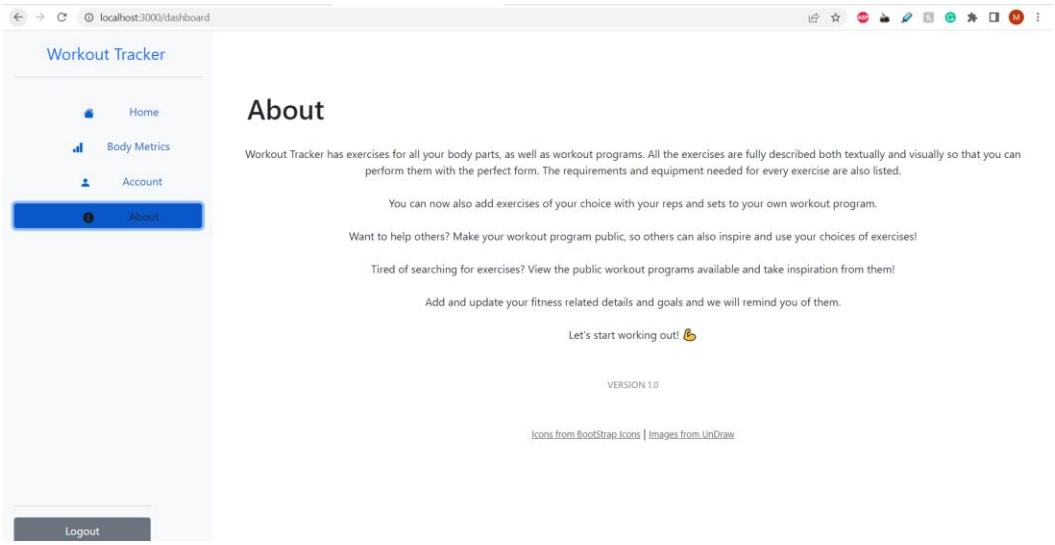


Figure 6.1 – User's view of the About page

The following components interact together to achieve this view;

1) HTML template : dashboard.html

2) JS template : dashboard.js

This is a rather simple, text-based view hence it was implemented mainly using HTML paragraphs and break points as follows:

```
<div id="about" style="display: none">
  <div class="heading">
    <h1 style="margin-bottom: 0px">About</h1>
  </div>
  <p style="text-align: center">
    Workout Tracker has exercises for all your body parts, as well as
    workout programs. All the exercises are fully described both textually
    and visually so that you can perform them with the perfect form. The
    requirements and equipment needed for every exercise are also listed.
    <br /><br />
    You can now also add exercises of your choice with your reps and sets
    to your own workout program.
    <br /><br />
    Want to help others? Make your workout program public, so others can
    also inspire and use your choices of exercises!
    <br /><br />
    Tired of searching for exercises? View the public workout programs
    available and take inspiration from them!
    <br /><br />
    Add and update your fitness related details and goals and we will
    remind you of them.
    <br /><br />
    Let's start working out! 💪
    <br /><br /><br />
    <span style="opacity: 0.8">Version 1.0 (04-22)</span>
  </p>
</div>
</div>
```

Code Snippet 6.1 – HTML implementation of the About Page contents

This view appears when the user clicks the "About" navigation button. This button listens to click events, when clicked, it calls a function that hides all the other pages/views and makes the about view/page visible. The JavaScript code for this implementation is as follows:

```
$("#about-nav").click(function (event) {
  event.preventDefault();
  $("#home").hide();
  $("#body-metrics").hide();
  $("#account").hide();
  $("#about").show();

  $(".link-dark").removeClass("active");
  $("#about-nav").addClass("active");
});
```

Code Snipper 6.2 – Click event for the "About" navigation button (dashboard.js – 178)

Design and Animations

We are using a combination of bootstrap styling, external stylesheet and in-line CSS to style our application. When a component is styled through multiple methods, following priority sequence is enforced.

In rare instances, we are using !important rule of CSS to bypass the priority sequence.

Bootstrap offers a clean, modern styling, which proved to be useful for styling most of our project components. When we had to bypass some of bootstrap's styling, or to add new custom styling, we used an external stylesheet (*stylesheet/style.css*) and in-line CSS.

A rule that we followed was that, if a styling rule only applies to a single component, then we use in-line CSS. When the styling rule should affect multiple components, we would declare it in an external stylesheet (*stylesheet/style.css*). This meant we would not have to provide id to every component that needs to be styled (such as div containers), hence simplifying our code.

```
.capitalize{
  text-transform: capitalize;
}
```

An example of CSS styling using class selector. The style would be applied to any component with the class name "capitalize".

```
<div style="font-size: small; color: red; margin-top: 10px">
  Attention. Editing account details will log you out.
</div>
```

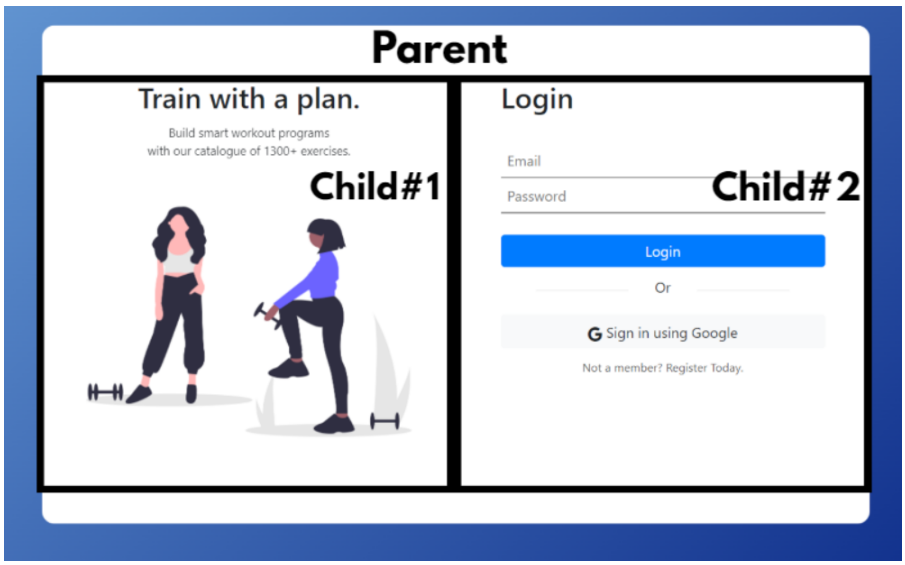
An example of a component with in-line styling. The styling rule will only apply to this component.

Positioning Elements

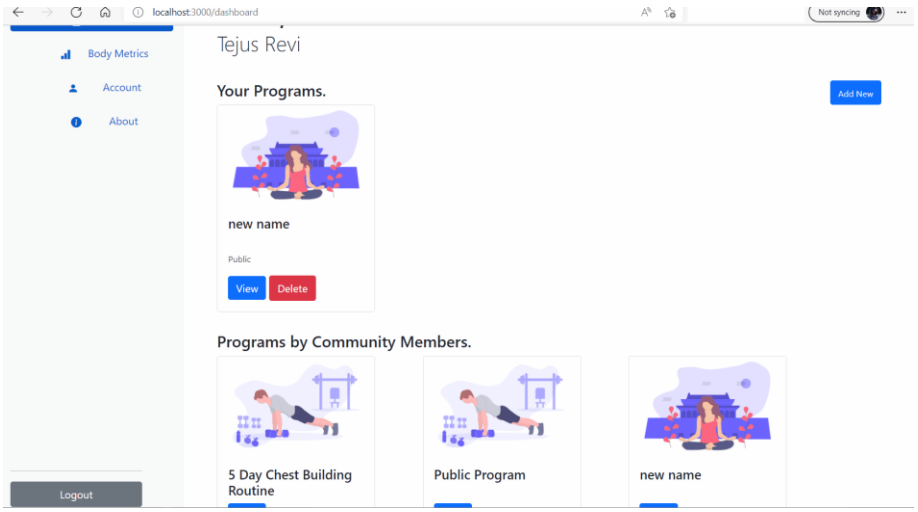
The application mostly relies on flex layout to position components. An example of how we used flex layout can be seen in the file index.js, where we positioned login/register form and a static div side by side.

```
<div
  id="form-card"
  class="horizontally-center vertically-center"
  style="
    width: 60vw;
    border-color: white;
    height: 70vh;
    border-radius: 0px;
    display: flex;
    flex-direction: row;
    padding: 0px;
    border-width: 0px;
    border-radius: 14px;
    overflow: hidden;
  "
>
```

Children of #form-card will be positioned on the same row



An element that we positioned absolutely in regard to the browser window is the logout button. We used a fixed position for the logout button, so that it will stay on screen as the user scrolls down.



GIF showing Logout button being stationary.

Scroll Bar

We are using CSS to overwrite the scrollbar design of major web browsers to match the modern aesthetics of our application. This is facilitated by using WebKit CSS extension and will only work in select browsers.

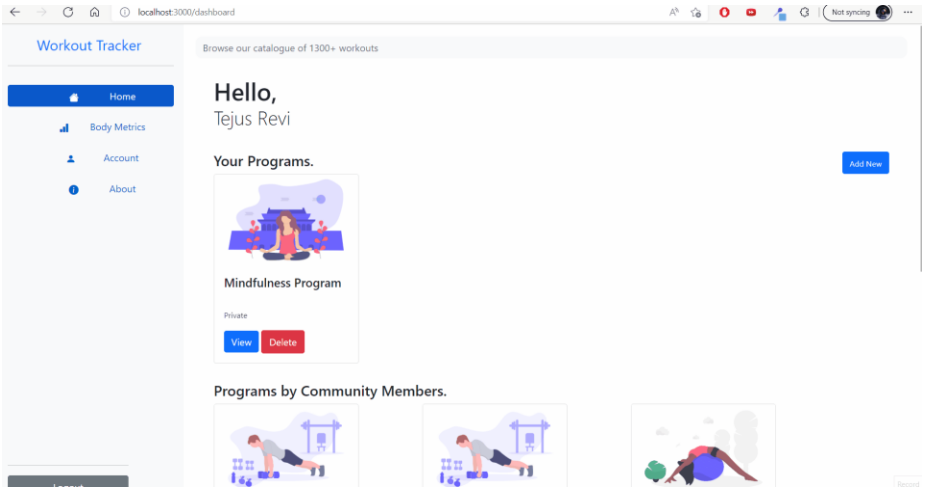
```
::-webkit-scrollbar {
  width: 10px;
}

::-webkit-scrollbar-track {
  box-shadow: inset 0 0 10px 10px rgb(255, 255, 255);
  border: solid 3px transparent;
}

::-webkit-scrollbar-thumb {
  box-shadow: inset 0 0 10px 10px #b5b5b5;
  border: solid 3px transparent;
}
```

CSS styling to replace scroll bar design

Animations

We made careful use of animations to gently introduce content to the user to create an intuitive experience. An example of a CSS based animation that we used can be seen in how workout

program cards are gently faded into view as the user refreshes their webpage. This is facilitated using @keyframes property in the external stylesheet.

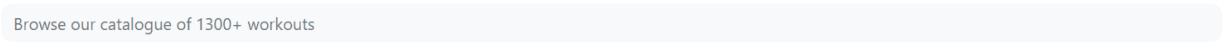


GIF showing workout program cards fading into view.

```
.workout-program-card{
  animation: fadeIn 0.5s;
  animation-delay: 1s;
  -webkit-animation: fadeIn 0.5s;
  -moz-animation: fadeIn 0.5s;
  -o-animation: fadeIn 0.5s;
  -ms-animation: fadeIn 0.5s;
}

@keyframes fadeIn {
  0% {opacity:0;}
  100% {opacity:1;}
}
```

The opacity of components with class name "workout-program-card" will go from 0% to 100% in 0.5 second.

Another example of an animation we used is how the search bar in Home view widens up to occupy available space.

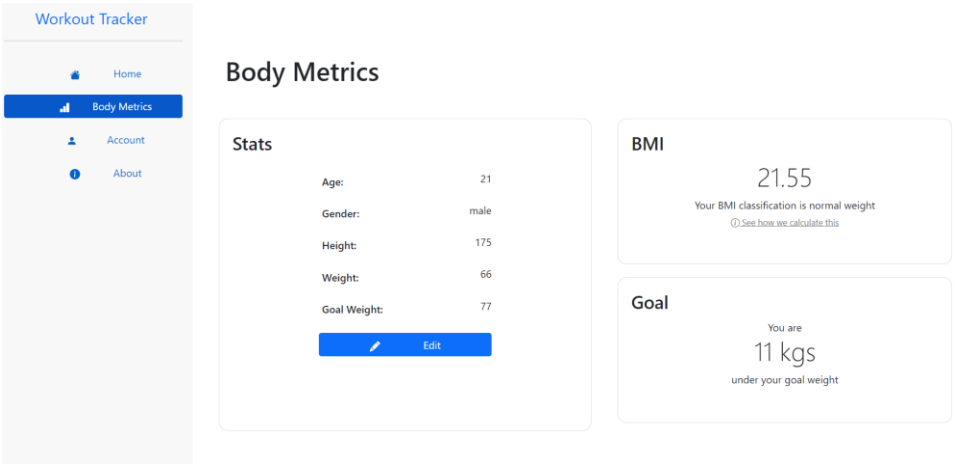


GIF showing search bar animation

```css
#search-bar{
  margin: auto;
  animation: fadeIn 0.5s;
  -webkit-animation: searchBarIntro 0.5s;
}

@keyframes searchBarIntro {
  0% {max-width: 0%;}
  100% {max-width: 100%;}
}
```

The maximum width of component with id "search-bar" will go from 0% to 100% in 0.5 second.

A final example of an animation that is being used in the application is how different views in dashboard page are faded into browser window. Unlike previous examples, this animation uses the .fadeIn() function within jQuery to fade in components.

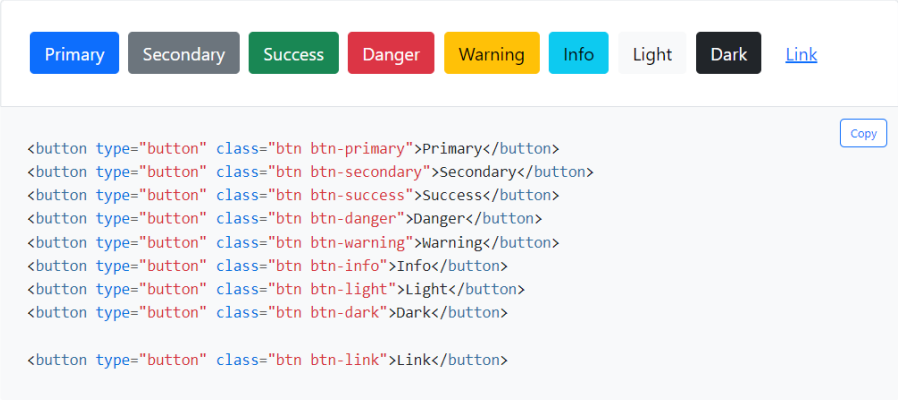


GIF showing different views in dashboard being faded in.

Bootstrap Components

Buttons

All HTML buttons used in our application follow Bootstrap's guide to accessible front end.

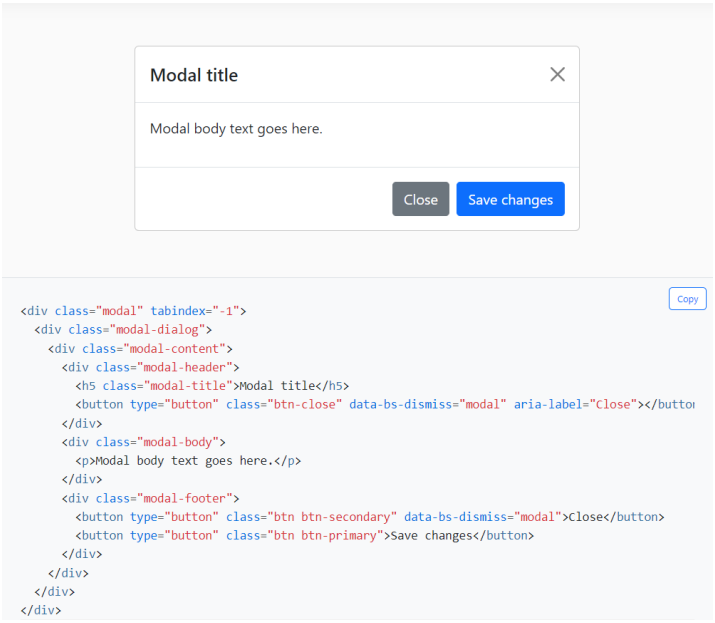


Bootstrap button convention from getbootstrap.com/docs

We used the class name “btn-primary” to indicate a positive action (eg. Login/ register, adding new workout program) and the class name “btn-danger” to indicate a negative action (eg. Deleting account, deleting workout program). We also used the class name “btn-link” to emulate the appearance of <a> tag, but the functionality required <button>.

Modals

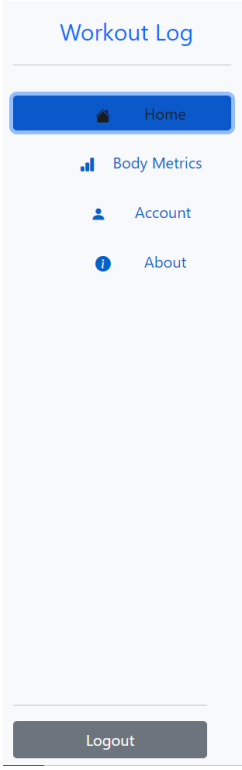
We used Bootstrap models to build pop-up windows to seek user input, display information and to request validation before performing critical tasks. Modals proved to be very useful to conserve screen real estate, while performing these side actions. Some of our modals, such as the modal displaying exercise information, are dynamically generated depending on the button clicked by the user.



Bootstrap sample modal from getbootstrap.com/docs

Navbar

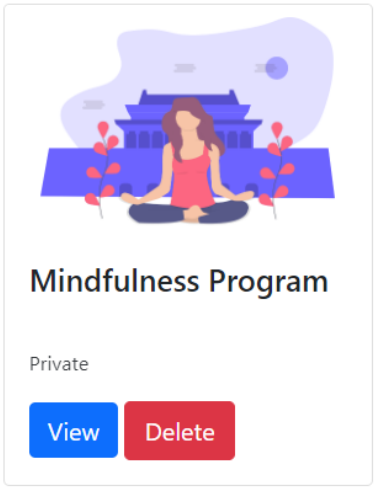
Our dashboard page uses a Bootstrap Navbar to toggle between Home, Body Metrics, Account and About views. By using a Navbar, we were able to condense all four views into a single webpage, and selectively render content depending on the view selected by the user.



Bootstrap Navbar

Card

Dashboard page uses cards to display workout program information. Bootstrap cards are flexible and extensible, enabling us to display workout program information to the user. Cards also include a thumbnail. In this version of the application, we are using random images as our thumbnail.



Bootstrap card

We would also like to take this opportunity to thank the professor, Dr. Amilcar Soares for making this course a fun and practical learning experience. The course widened our knowledge base and we both learnt a lot of new skills and we are thankful for your efforts making this course a successful one for us. So thank you and we wish you good health and luck for what lies ahead !