Stories For Infuse

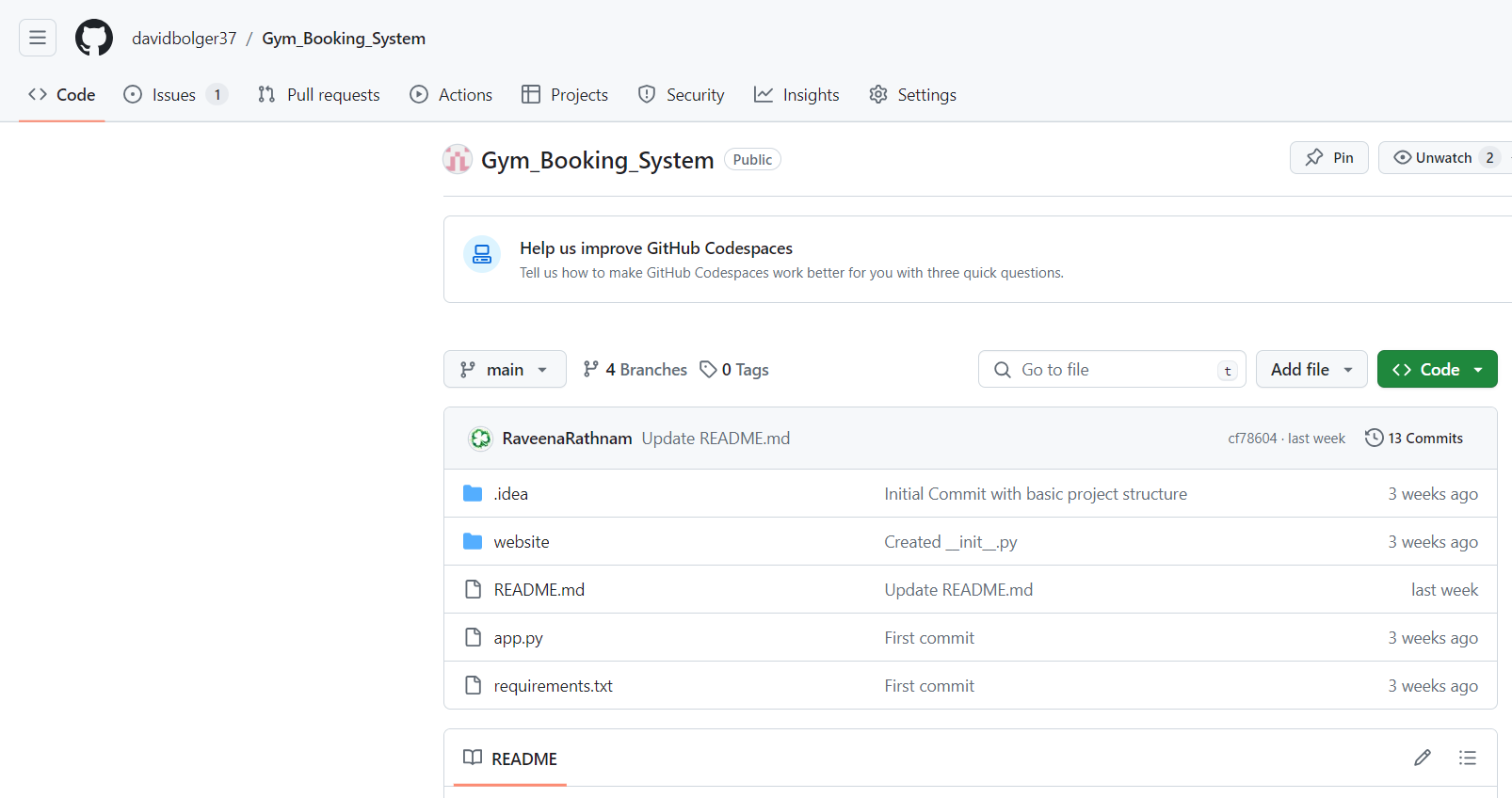
**As an** Intern at Ericsson

**I want** a fully functional website for a fictional gym called GETFIT

**So that** I can have demo the project to my product owner

**Acceptance Criteria:**

* Complete project stored on GitHub
* Code is peer reviewed and commented with beginners in mind
* Project is Demoed to Product Owner/ INFUSE Representative



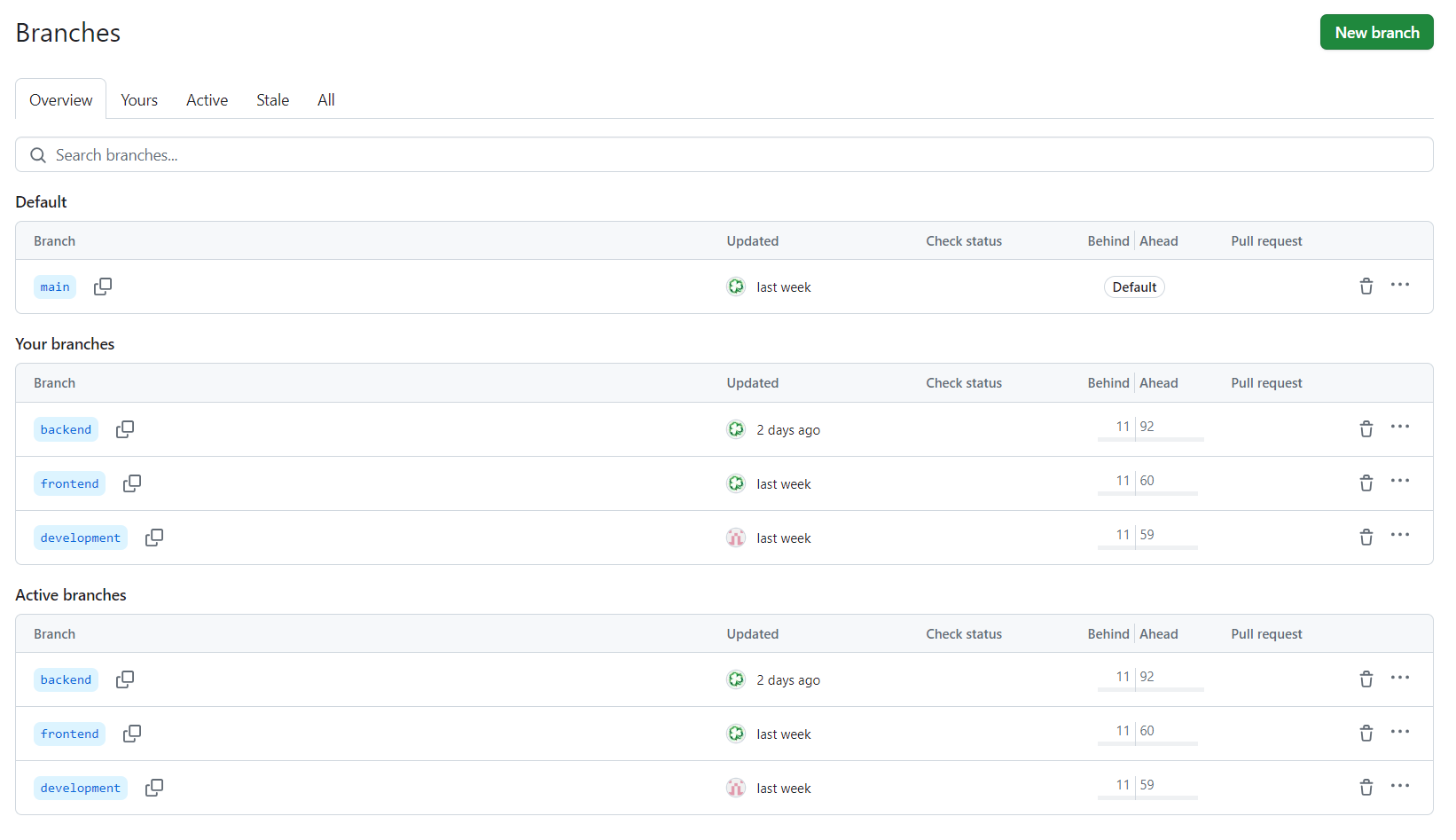
**As a** Developer

**I want** my project in a single location that it can be version controlled

**So that** I can easily download and run the project as well as look through the history of code added/deleted

**Acceptance Criteria:**

* A GitHub project is available containing all resources needed to run the project locally
* Branches are used per feature(frontend/backend)
* Proper commit messages are used to detail changes
* a readme.md file that explains the project, how to run it and what it was created for



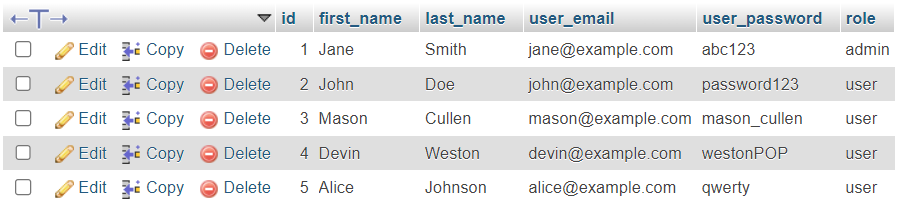
**As a** developer

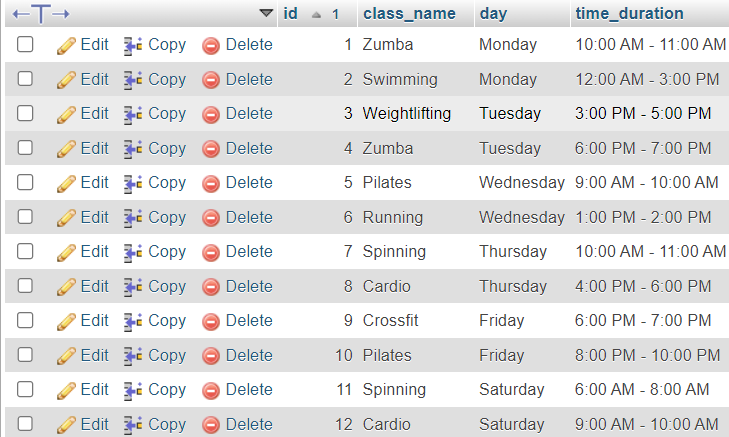
**I want** a well defined MySQL database

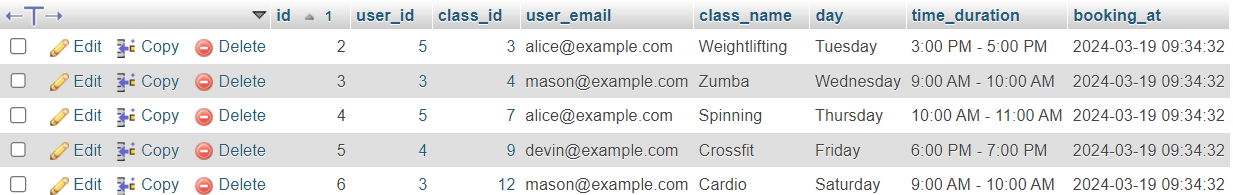
**So that** all my information is stored without duplication and can be manipulated through our CRUD commands

**Acceptance Criteria:**

* Tables created to represent:
  + Users (ID, First Name, Last Name, Email, Password, etc.)
  + Class (ID, (class name, day, class duration)
  + Booking (ID, users id, class id, users email, class name, day, class duration, time of booking)
* Tables normalized to 3rd Normal Form [Info Link](https://www.kevsrobots.com/learn/sqlite3/15_3nf.html#:~:text=The%20Third%20Normal%20Form%20(3NF,Second%20Normal%20Form%20(2NF).)
* Create sample Create, Read, Update & Delete commands uploaded to a file in the project
* Create MySQL script to create and populate tables with sample data in a file in the project
* Using XAMPP shell enter the script and confirm it worked using phpMyAdmin

****

****

****

**HTML/CSS**

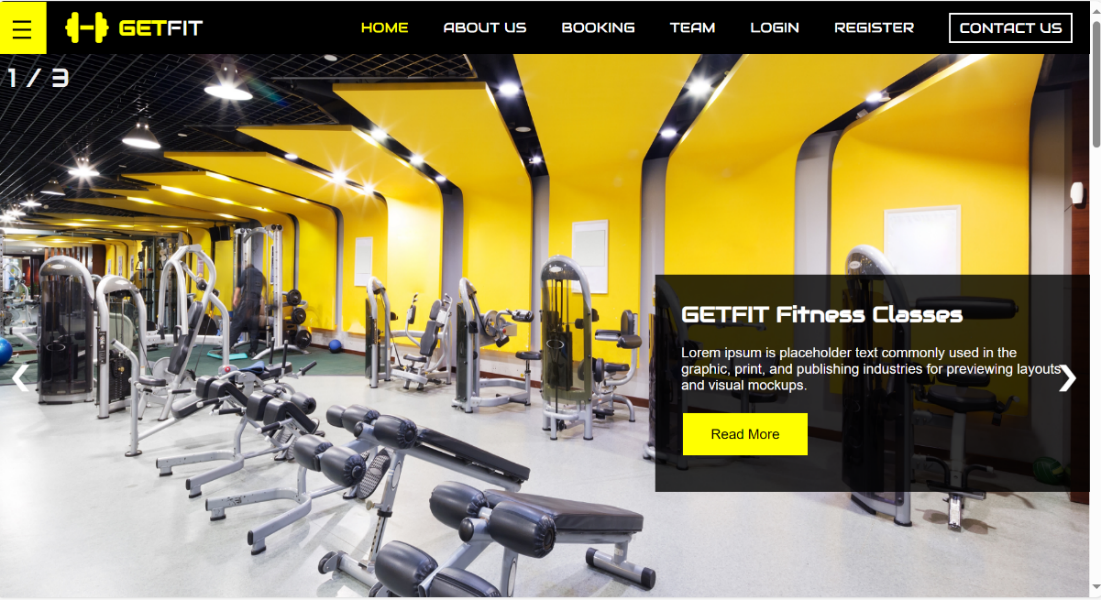
**As a developer**

**I want** to create a well-structured/appealing site using HTML and CSS

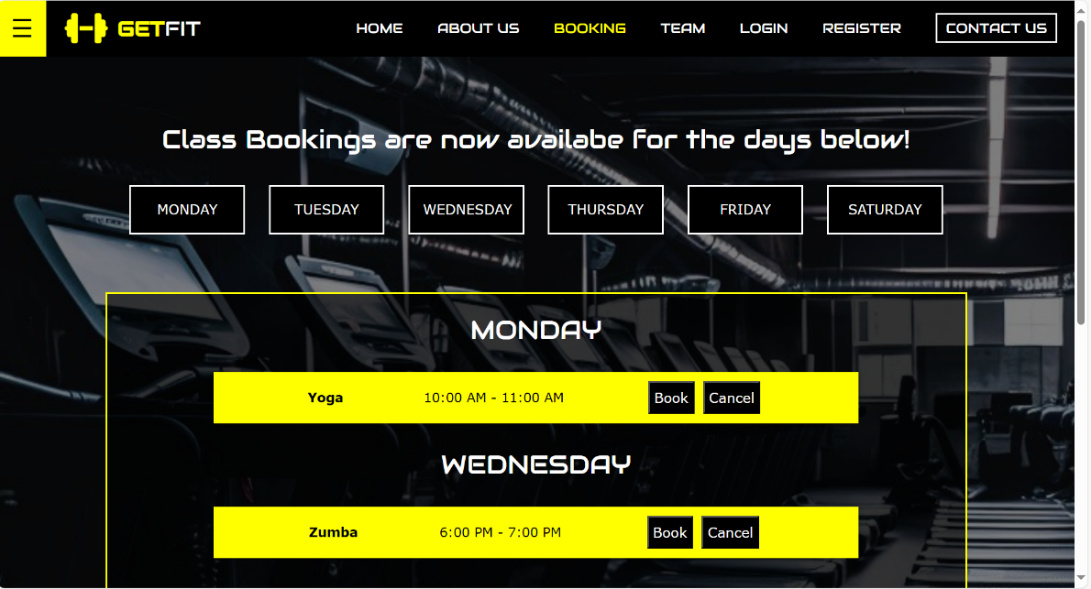
**So that** customers can navigate the site and book a class with ease

Acceptance Criteria:

* XAMPP is setup and running
* A working navbar which will take you to the other pages
* A booking page dedicated to adding to the database
* Buttons added to booking page to handle CRUD operations
* A responsive design that renders correctly depending on the device
* Consistent styling across the site's pages
* Header and footer included on each page
* Flash messages/popups to let the user know of a successful booking
* Pages for login and register
* The site responds accordingly to who is logged in e.g. (user or admin)
* All HTML elements are nested correctly
* CSS files are linked correctly to the corresponding HTML file
* Front End is demoed to the PO/Shareholders and approved



Index.html



Booking.html

**Login/Register**

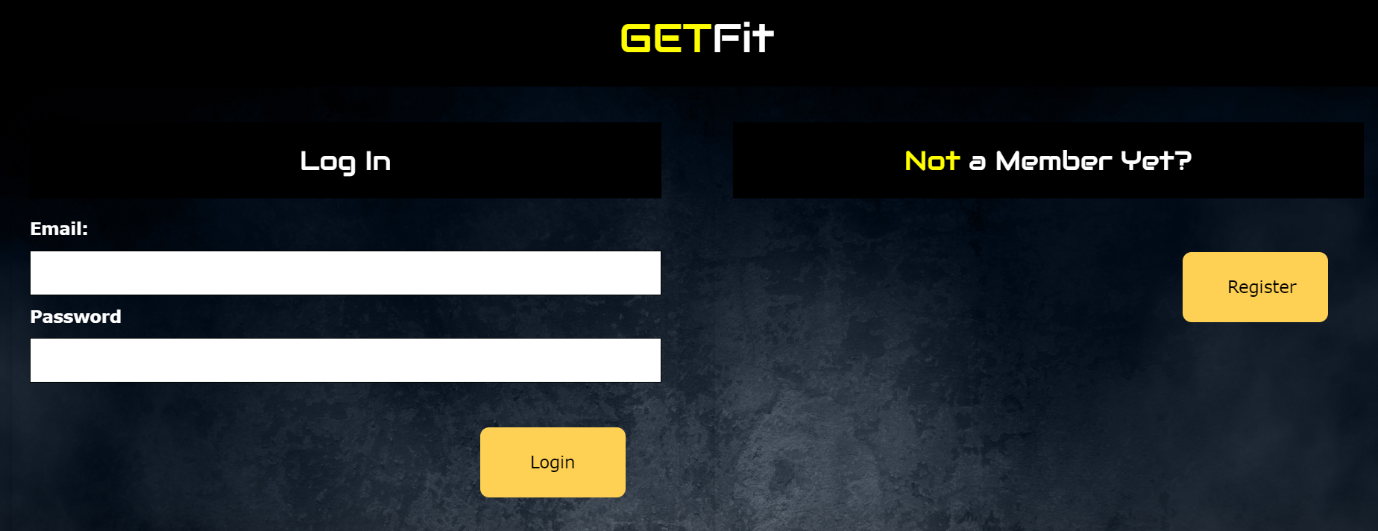
**As a** customer

**I want** to login to my account

**So that** I can cancel an existing booking

Acceptance Criteria:

* Login page is created with HTML/CSS.
* The link to the login page should be visible/accessible to the user.
* The login page should have a form with two clearly labelled input fields:
  + Email address and password
* The user’s password input is obscured.
* The page should have a submit button.
* The login credentials should be verified as existing in the database.
* The user should be prompted to re-enter details if invalid.
* The user has the option to logout.
* The option to create, edit and delete classes are only visible to the logged in admin.
* The create class page is only accessible to the admin.
* A logged-out user has only view access and cannot book/cancel a class.



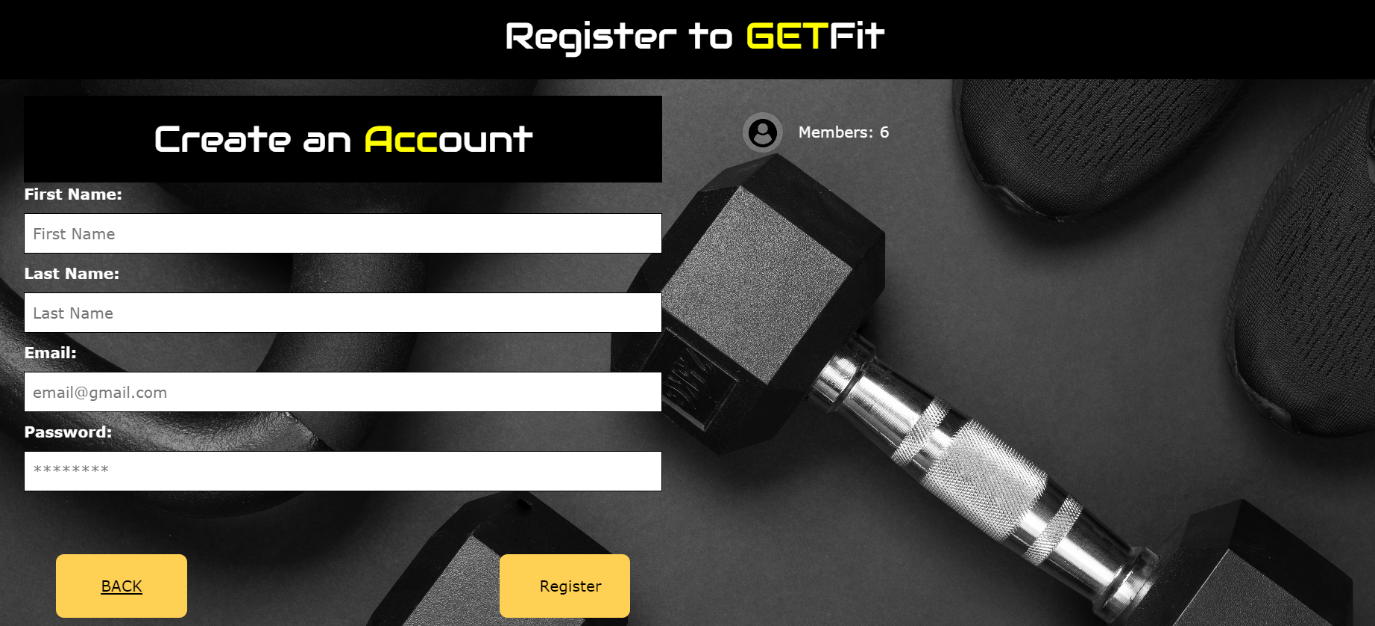
**As a** customer

**I want** to register a new account

**So that** I can book a yoga class

Acceptance Criteria:

* Register page is created with HTML/CSS.
* The link to the register page should be visible/accessible to the user.
* The register page should have a form with four clearly labelled input fields:
  + First name, last name, email address and password.
* The page should have a submit button.
* Invalid or empty fields should re-prompt the user to enter valid details on submission.
* If the account already exists, an appropriate message is displayed, and the user is redirected to the login page.
* Successful registration should redirect the user to the login page.



**CRUD User Stories**

**Create a Class:**

**As a** gym administrator,

**I want to** add a new class to the schedule/timetable,

**So that** gym members can book and attend the class.

**Acceptance Criteria:**

* A “Create Class” option/button is accessible to logged-in admins on the “Booking” page.
* Clicking “Create Class” brings up a form with input fields for class name, time slot, and day.
* The form includes a submit button labeled “Create Class”.
* The form includes a dropdown menu for day selection to ensure that a valid day is selected.
* Validation ensures all fields are filled out correctly, with time slots matching the required format (e.g., “11:00 AM – 12:00 PM”).
* On successful class creation, a confirmation message is displayed on the “Booking” page, and the class is visible in the schedule/timetable.
* Attempting to create a duplicate class, an error message is displayed on the “Create Class” page indicating that a class with same details already exists.

**Coding the Create View:**

So, the **Create View** will be:

@app.route('/create-class', methods=['GET', 'POST'])  
def create\_class():  
 if 'user\_id' not in session or session.get('user\_role') != 'admin':  
 # If the user is not logged in or not an admin, redirect them  
 flash('You do not have permission to access this page.', 'error')  
 return redirect(url\_for('login'))  
  
 if request.method == 'POST':  
 # Retrieve form data  
 class\_name = request.form.get('class\_name')  
 day = request.form.get('day')  
 time\_duration = request.form.get('time\_duration')  
  
 # Regex pattern for matching "11:00 AM - 12:00 PM" format  
 pattern = r'^(?:1[0-2]|0?[1-9]):[0-5][0-9] [AP]M - (?:1[0-2]|0?[1-9]):[0-5][0-9] [AP]M$'  
  
 # Server-side validation of the time slot format  
 if not re.match(pattern, time\_duration):  
 flash('Invalid time slot format. Please use the "11:00 AM - 12:00 PM" format.', 'error')  
 return render\_template('create\_class.html')  
  
 # Check if the class already exists to prevent duplicates  
 existing\_class = Class.query.filter\_by(class\_name=class\_name, day=day, time\_duration=time\_duration).first()  
  
 if existing\_class:  
 flash('Class already exists.', 'error')  
 return redirect(url\_for('create\_class'))  
  
 # Create a new Class instance if it doesn't exist  
 try:  
 new\_class = Class(class\_name=class\_name, day=day, time\_duration=time\_duration)  
 db.session.add(new\_class)  
 db.session.commit()  
 flash('Class created successfully!', 'success')  
 except IntegrityError:  
 db.session.rollback()  
 flash('An error occurred while creating the class.', 'error')  
  
 # Redirect to the class listing page (booking)  
 return redirect(url\_for('booking'))  
 # For a GET request, just render the class creation form  
 return render\_template('create\_class.html')

The **create\_class.html** will contain the HTML Form:

<form id="create\_form" method="post">  
 <label for="class\_name">Class Name:</label>  
 <input type="text" id="class\_name" name="class\_name" placeholder="Enter class name.."><br>  
  
 <label for="time\_duration">Time Slot:</label>  
 <input type="text" id="time\_duration" name="time\_duration" placeholder="Eg. 11:00 AM - 12:00 PM" required pattern="(?:1[0-2]|0?[1-9]):[0-5][0-9] [AP]M - (?:1[0-2]|0?[1-9]):[0-5][0-9] [AP]M">><br>  
  
 <label for="day">Day:</label>  
 <select id="day" name="day" required>  
 <option value="Monday">Monday</option>  
 <option value="Tuesday">Tuesday</option>  
 <option value="Wednesday">Wednesday</option>  
 <option value="Thursday">Thursday</option>  
 <option value="Friday">Friday</option>  
 <option value="Saturday">Saturday</option>  
 </select><br>  
  
 <input type="submit" value="Create Class">  
</form>

**Read/View Classes (For Admins):**

**As a** gym administrator,

**I want to** view the all the scheduled classes,

**So that** I can manage the gym schedule.

**Acceptance Criteria:**

* In the “Booking” page the admin can view the schedule and manage it.
* Admins can see a list of classes including class name, time slot, and day.
* Classes are ordered according to the days of the week.
* The admins should be able to retrieve the classes scheduled on the day of the week by clicking on the corresponding button with the day name on it.
* Each class entry includes administrative options like “Edit” and “Delete” to manage the class.

**Coding the Read View (For Admins):**

So, the **Read View** will be:

@app.route('/booking')  
def booking():  
 user\_role = None  
 if 'user\_id' in session:  
 user = User.query.get(session['user\_id'])  
 user\_role = user.role  
  
 # Define an ordering for the days of the week  
 day\_ordering = case(  
 value=Class.day,  
 whens={  
 'Monday': 1,  
 'Tuesday': 2,  
 'Wednesday': 3,  
 'Thursday': 4,  
 'Friday': 5,  
 'Saturday': 6,  
 },  
 )  
  
 # Fetch all classes and order them first by day, then by time\_duration  
 classes = Class.query.order\_by(day\_ordering, Class.time\_duration).all()  
  
 classes\_by\_day = {}  
 for class\_ in classes:  
 if class\_.day not in classes\_by\_day:  
 classes\_by\_day[class\_.day] = []  
 classes\_by\_day[class\_.day].append(class\_)  
  
 return render\_template('booking.html', user\_role=user\_role, classes\_by\_day=classes\_by\_day)

The **booking.html** file will display the list of Classes:

{% for day, classes in classes\_by\_day.items() %}  
<div id="{{ day.lower() }}">  
 <h2 style="color: white; text-align: center; text-transform: uppercase; font-family:Audiowide, sans-serif;">{{ day }}</h2>  
 <div class="w3-container" style="width: 80%; margin: 0 auto; ">  
 {% for class in classes %}  
 <div class="w3-panel w3-card "  
 style="display:flex; justify-content:space-between; align-items:center; background-color:yellow;height:60px;">  
 <b>{{ class.class\_name }}</b>  
 <p style="font-weight:300;text-align: left;"><b>{{ class.time\_duration }} </b></p>

{% endfor %}

</div>

</div>

{% endfor %}

**Administrative options** for Admins:

{% if user\_role == 'admin' %}  
<button class="card\_button" onclick="window.location.href='{{ url\_for('update\_class', class\_id=class.id) }}'">  
 Edit  
</button>  
  
<button class="card\_button" onclick="document.getElementById('id01-{{ class.id }}').style.display='block'">  
 Delete  
</button>

{% endif %}

**Read/View Classes (For Members):**

**As a** gym member,

**I want to** view the schedule of all available classes,

**So that** I can plan and book the classes that I wish to attend.

**Acceptance Criteria:**

* A “Book” option/button is accessible to logged-in members on the “Booking” page.
* The schedule displays a list of all classes with details (class name, time and day).
* The schedule should be sorted out according to the days of the week.
* The “Booking” page includes buttons with days of the week on them, to allow users to retrieve the classes scheduled on the day of the week.
* A “Cancel” option/button is accessible to logged-in members on the booking page to cancel a booking for a class that has already been made by the member.

**Coding the Read View (For Members):**

So, the **members' Read View** will be the same as the administrators'; the only thing that will differ is that members can access the book or cancel options exclusively, not the administrative ones.

**Booking/Cancellation options** for Members:

{% if user\_role == 'user' %}  
<button class="card\_button" onclick="document.getElementById('id03-{{ class.id }}').style.display='block'">  
 Book  
</button>  
<button class="card\_button" onclick="document.getElementById('id02-{{ class.id }}').style.display='block'">  
 Cancel  
</button>

{% endif %}

**Update a Class:**

**As a** gym administrator,

**I want to** update the details of an already existing class in the schedule/timetable,

**So that** I can keep the class schedule accurate and up to date for the gym members.

**Acceptance Criteria:**

* Next to each class listing, an “Edit” option/button is accessible to logged-in admins on the “Booking” page.
* Clicking “Edit” opens a form pre-filled with the class’s existing details (class name, time slot, and day).
* The admin can modify the class name, time slot, and day.
* The form includes a submit button labeled “Update Class”.
* The form includes a dropdown menu for day selection to ensure that a valid day is selected when modified.
* Validation ensures all modified fields are correctly filled out.
* A confirmation message is displayed on the “Booking” page when the class has been updated successfully.
* An error message is displayed on the “Update Class” page when the updated class has the details of an existing class.
* All existing bookings for the updated class are automatically cancelled.

**Coding the Update View:**

So, the **Update View** will be:

@app.route('/update-class/<int:class\_id>', methods=['GET', 'POST'])  
def update\_class(class\_id):  
 if 'user\_id' not in session or session.get('user\_role') != 'admin':  
 # If the user is not logged in or not an admin, redirect them  
 flash('You do not have permission to access this page.', 'error')  
 return redirect(url\_for('login'))  
  
 # Retrieve the class from the database  
 class\_to\_update = Class.query.get\_or\_404(class\_id)  
  
 if request.method == 'POST':  
 # Retrieve form data  
 class\_name = request.form.get('class\_name')  
 day = request.form.get('day')  
 time\_duration = request.form.get('time\_duration')  
  
 # Regex pattern for matching "11:00 AM - 12:00 PM" format  
 pattern = r'^(?:1[0-2]|0?[1-9]):[0-5][0-9] [AP]M - (?:1[0-2]|0?[1-9]):[0-5][0-9] [AP]M$'  
  
 # Server-side validation of the time slot format  
 if not re.match(pattern, time\_duration):  
 flash('Invalid time slot format. Please use the "11:00 AM - 12:00 PM" format.', 'error')  
 return render\_template('create\_class.html')  
  
 # Check if the class already exists to prevent duplicates  
 existing\_class = Class.query.filter\_by(class\_name=class\_name, day=day, time\_duration=time\_duration).first()  
  
 if existing\_class:  
 flash('A class with the same name, day, and time already exists.', 'error')  
 return redirect(url\_for('update\_class', class\_id=class\_id))  
  
 # Delete associated bookings for the class being updated  
 try:  
 bookings\_to\_delete = Booking.query.filter\_by(class\_id=class\_id).all()  
 for booking in bookings\_to\_delete:  
 db.session.delete(booking)  
 db.session.commit()  
 except Exception as e:  
 db.session.rollback()  
 flash('An error occurred while deleting associated bookings.', 'error')  
  
 # Update the Class instance  
 try:  
 class\_to\_update.class\_name = class\_name  
 class\_to\_update.day = day  
 class\_to\_update.time\_duration = time\_duration  
 db.session.commit()  
 flash(f'Class:{class\_name} updated successfully!', 'success')  
 except IntegrityError:  
 db.session.rollback()  
 flash('An error occurred while updating the class.', 'error')  
  
 # Redirect to the class listing page (booking)  
 return redirect(url\_for('booking'))  
  
 # For a GET request, just render the class update form  
 return render\_template('update\_class.html', class\_=class\_to\_update)

The **update\_class.html** will contain the HTML Form that is pre-filled with the class’s existing details:

<form id="update\_form" method="post">  
 <label for="class\_name">Class Name:</label>  
 <input type="text" id="class\_name" name="class\_name" value="{{class\_.class\_name}}" placeholder="Enter class name.."><br>  
  
 <label for="time\_duration">Time Slot:</label>  
 <input type="text" id="time\_duration" name="time\_duration" value="{{class\_.time\_duration}}"  
 placeholder="Eg. 11:00 AM - 12:00 PM" required pattern="(?:1[0-2]|0?[1-9]):[0-5][0-9] [AP]M - (?:1[0-2]|0?[1-9]):[0-5][0-9] [AP]M">><br>  
  
 <label for="day">Day:</label>  
 <select id="day" name="day" required>  
 <option value="Monday" {% if class\_.day=='Monday' %}selected{% endif %}>Monday</option>  
 <option value="Tuesday" {% if class\_.day=='Tuesday' %}selected{% endif %}>Tuesday</option>  
 <option value="Wednesday" {% if class\_.day=='Wednesday' %}selected{% endif %}>Wednesday</option>  
 <option value="Thursday" {% if class\_.day=='Thursday' %}selected{% endif %}>Thursday</option>  
 <option value="Friday" {% if class\_.day=='Friday' %}selected{% endif %}>Friday</option>  
 <option value="Saturday" {% if class\_.day=='Saturday' %}selected{% endif %}>Saturday</option>  
 </select><br>  
 <input type="submit" value="Update Class">  
</form>

**Delete a Class:**

**As a** gym administrator,

**I want to** delete a class in the schedule/timetable,

**So that** I can remove classes that are no longer offered.

**Acceptance Criteria:**

* Next to each class listing, a “Delete” option/button is accessible to logged-in admins on the “Booking” page.
* Clicking “Delete” prompts the admin for confirmation to avoid accidental deletions.
* Confirming the deletion removes the class from the schedule and displays a confirmation message on the “Booking” page.
* All existing bookings for the deleted class are automatically cancelled.

**Coding the Delete View:**

So, the **Delete View** will be:

@app.route('/delete-class/<int:class\_id>', methods=['POST'])  
def delete\_class(class\_id):  
 if 'user\_id' not in session or session.get('user\_role') != 'admin':  
 flash('You do not have permission to access this page.', 'error')  
 return redirect(url\_for('login'))  
  
 class\_to\_delete = Class.query.get(class\_id)  
 if class\_to\_delete:  
 db.session.delete(class\_to\_delete)  
 try:  
 db.session.commit()  
 flash('Class has been deleted successfully!', 'success')  
 except Exception as e:  
 db.session.rollback() # Roll back the changes if something goes wrong  
 flash('An error occurred while deleting the class.', 'error')  
 else:  
 flash('Class doesn\'t exist', 'error')  
  
 return redirect(url\_for('booking'))

The **delete modal** just re-confirms the deletion when the delete button is pressed on the booking page:

<button class="card\_button" onclick="document.getElementById('id01-{{ class.id }}').style.display='block'">  
 Delete  
</button>  
<div id="id01-{{ class.id }}" class="modal">  
 <span onclick="document.getElementById('id01-{{ class.id }}').style.display='none'"  
 class="close" title="Close Modal">&times;</span>  
 <form class="modal-content"  
 action="{{ url\_for('delete\_class', class\_id=class.id) }}"  
 method="post">  
 <div class="modal\_container">  
 <h1 style="font-family:Audiowide, sans-serif;font-size:35px;">DELETE CLASS</h1>  
 <p>Are you sure you want to delete this class?</p>  
  
 <div class="clearfix">  
 <button onclick="document.getElementById('id01-{{ class.id }}').style.display='none'"  
 type="button" class="cancelbtn">Cancel  
 </button>  
 <button type="submit" class="deletebtn">Delete</button>  
 </div>  
 </div>  
 </form>  
</div>

**JavaScript** for the modal to pop-up and close:

<script>  
 // Get the modal  
 var modal = document.getElementById('id01');  
// When the user clicks anywhere outside the modal, close it  
 window.onclick = function(event) {  
 if (event.target == modal) {  
 modal.style.display = "none";  
 }}  
</script>

If the admin presses **Delete,** then the Class is deleted. Or else he redirected to the booking page.