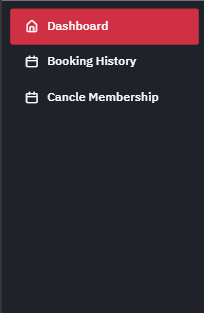
**Sidebar**

****

Code :-

View :-

Function: -  
This code creates a sidebar menu with links to different pages in a web application. It includes:

* **Dashboard**: Links to the dashboard page.
* **Booking History**: Links to the booking history page.
* **Cancel Membership**: Opens a modal to cancel the membership.

Each link has an icon next to the text for better visualization, and CSS classes are used for styling and effects.

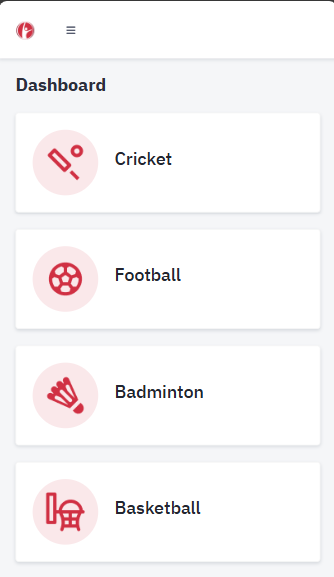
**Dashboard**

Code :-





View :-



Function: -

This code creates a section of the dashboard page with cards for different sports. Each card contains an image and a link to a listing page filtered by the respective sport. Here's a breakdown:

* **Container and Layout**: The container-fluid class holds the content, and the row class ensures the content is arranged in a row layout. Inside the row, there are four columns (col-sm-3), each containing a card for a different sport.
* **Card Structure**: Each card is created using the card class. Inside each card, there’s an image representing the sport and a heading with a link. The link directs to the listing page with a query parameter (?sport=<sport\_name>), which filters the listing based on the selected sport.
* **Icons and Text**: The img tags display images for the sports, such as cricket, football, badminton, and basketball. The <h4> tags wrap the links for each sport.

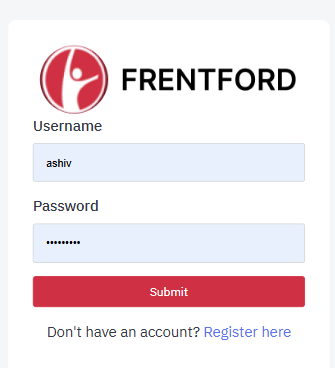
This layout allows users to click on a sport and view a filtered listing of relevant content.

**Login**

Code

Backend-code



View: -  
  


Function: -

This code handles user login in a Django web application.

* **POST Request Handling**: It first checks if the request method is POST, indicating the form was submitted.
* **Form Validation**: The LoginForm is validated. If the form is valid, it retrieves the username and password.
* **User Authentication**: It then tries to find the user in the UserData model by the provided username. If the user exists:
  + It checks if the account is active.
  + Verifies if the password is correct using check\_password().
  + If the credentials are valid, the user is logged in and redirected to the dashboard.
* **Error Handling**: If the username doesn’t exist or the password is incorrect, error messages are displayed.
* **Rendering the Form**: If the method is not POST, or after form validation fails, it renders the login form.

This code ensures that only active users with the correct credentials can log in.

**Register**

Code

View: -

No screen shot placed as it will be better to place from your side using the scrolling screen shot function to capture the whole form.

Function: -

This code handles user registration in web application.

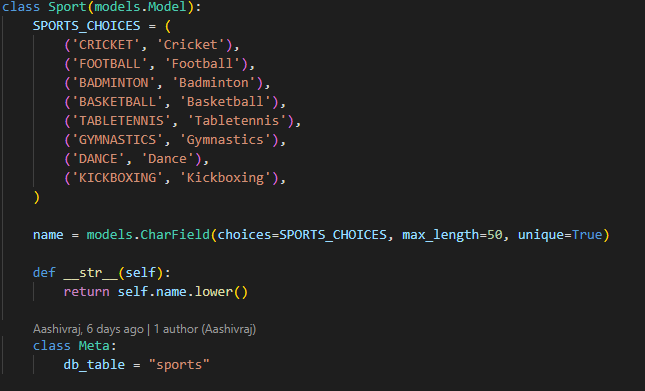
* **POST Request Handling**: If the request method is POST, the UserRegistrationForm is populated with the submitted data.
* **Form Validation**: If the form is valid, it processes the data:
  + It sets a default value for the join\_type field.
  + It checks if the two entered passwords match. If they do:
  + The password is hashed using set\_password() and the user is saved.
  + The user is then logged in and redirected to the login page with a success message.
  + If the passwords do not match, an error message is shown.
* **Form Errors**: If the form is invalid, error messages for each field are displayed.
* **Rendering the Form**: If the method is not POST, the registration form is rendered for the user to fill out.

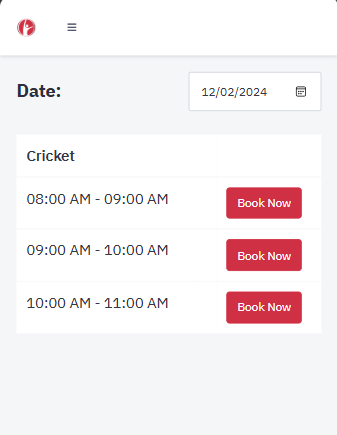
This code ensures proper user registration with password validation, error handling, and successful login after registration.

**Listing**

Code :-





View:-

Function: -

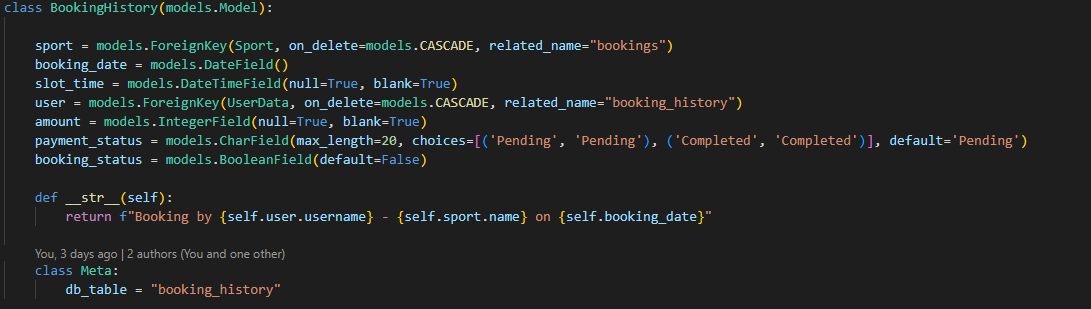
This code handles the listing page for sports for the web application.

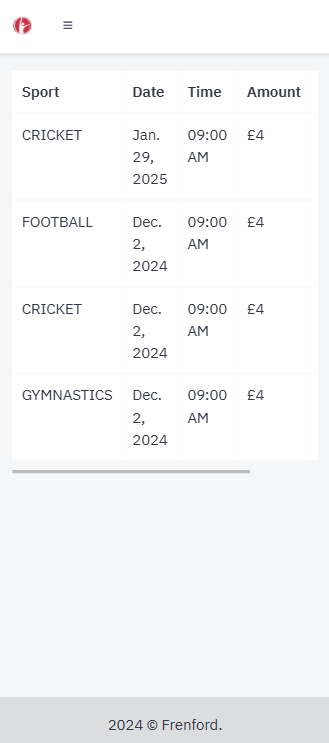
* **Login Required**: The @login\_required decorator ensures that the user must be logged in to access this view.
* **GET Request Handling**: If the request method is GET, it fetches the list of sport names from the Sport model and stores them in the sports variable.
* **Query Parameter**: It retrieves the selected sport from the query parameters (?sport=<sport\_name>), which is used to filter the sports displayed.
* **Rendering the Template**: The listing.html template is rendered with the list of sports and the selected sport passed as context.
* **Error Handling**: If the request method is not GET, it prints "wrong" and redirects the user to the login page.

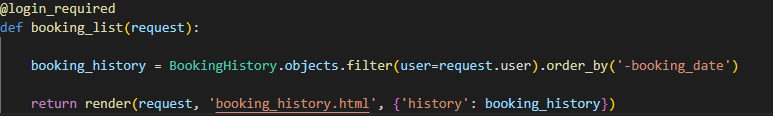
This code allows users to view and select sports from a list, with the option to filter based on the selected sport.

**Booking history**

Code:-



View:-



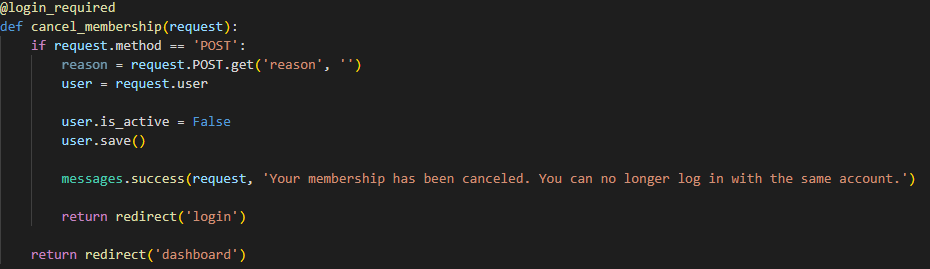
Function: -

This code displays a table showing booking history for the logged-in user.

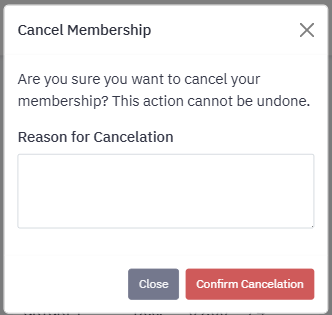
* **Table Structure**: It creates a responsive table with columns for **Sport**, **Date**, **Time**, **Amount**, and **Status**.
* **Dynamic Content**: The data is dynamically populated using a loop, where for each booking:
  + **Sport Name**: Displays the name of the sport.
  + **Booking Date**: Shows the date of the booking.
  + **Time**: Displays the time in a 12-hour format (H:i A).
  + **Amount**: Shows the booking amount, prefixed with the currency symbol (£).
  + **Payment Status**: Displays the status of the payment for the booking.

The data displayed is fetched from a database and passed into the template for rendering. This setup allows users to see details of their past bookings, such as the sport they booked, when the booking occurred, the amount paid, and the payment status.

**Cancel Membership**

Code: -

View:-



Function: -

This code provides functionality for cancelling a user's membership through a modal interface.

* **Modal Popup**: A modal is used to confirm the cancellation action. It displays a prompt asking if the user is sure they want to cancel their membership and provides a text area to enter a reason for cancellation.
* **Form Submission**: The user submits the form with the reason for cancellation. Upon submission, the account is deactivated, and a success message confirms the cancellation.
* **Action Confirmation**: After the cancellation, the user is redirected to the login page, as the account is no longer active, preventing further login attempts.

This approach ensures that users can easily cancel their membership with a clear confirmation step, and their account is deactivated immediately afterward.