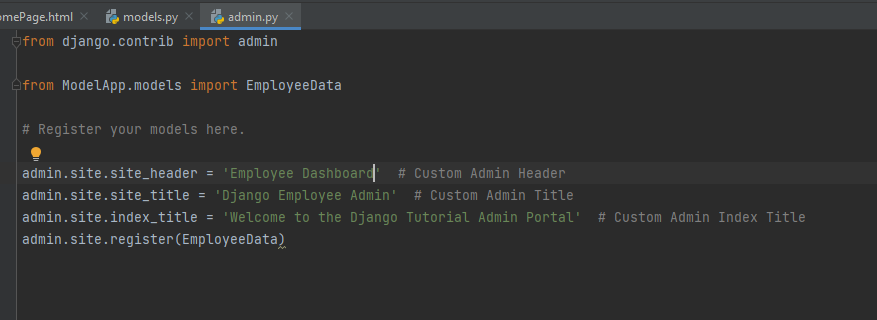
**Customizing Admin page**

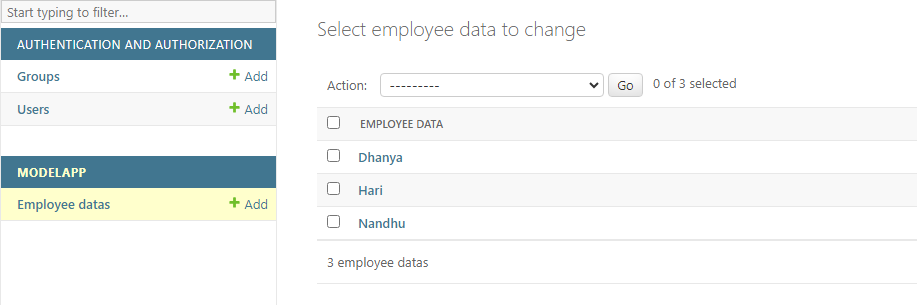
To customise the Django admin page, you can make changes to the admin.py file in your app or project



Replace the strings within the quotes with your desired text.

* site\_header: Sets the text that appears at the top of the admin panel.
* site\_title: Sets the text that appears on the browser tab or window title.
* index\_title: Sets the text that appears on the main admin page.

The customizations to the admin interface make it more specific and relevant to your application. After adding these lines to your admin.py file, you'll see the changes reflected when you access the Django admin interface for managing your EmployeeData model. The header, title, and index title will display the specified custom text.



the list\_display attribute allows you to specify the fields of your model that you want to display in the list view of the admin page. By default, Django displays the string representation of each object in the list view, but with list\_display, you can customize what fields are shown.

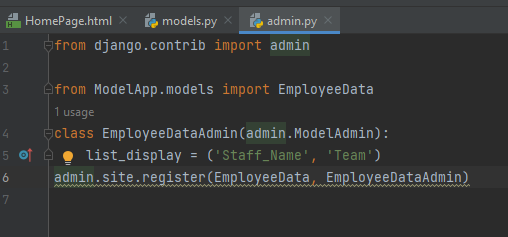
Custom Admin Class:

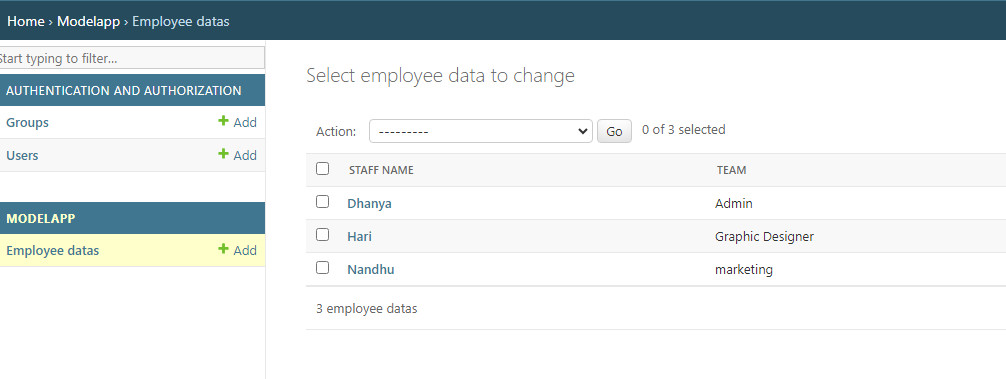
* This part defines a custom admin class named EmployeeDataAdmin. This class extends admin.ModelAdmin, which allows you to customize the behavior of the Django admin interface for the EmployeeData model.

list\_display Attribute:

* The list\_display attribute is used to specify which fields of the EmployeeData model should be displayed in the list view of the admin page for this model.
* In this case, it includes 'Staff\_Name' and 'Team', meaning that the list view will show the "Staff Name" and "Team" fields for each instance of EmployeeData.
* Registering the Model with the Custom Admin Class:
  + The admin.site.register function is used to register the EmployeeData model with the custom admin class (EmployeeDataAdmin).
  + This registration informs Django to use the specified admin class (EmployeeDataAdmin) for managing instances of the EmployeeData model in the admin interface.

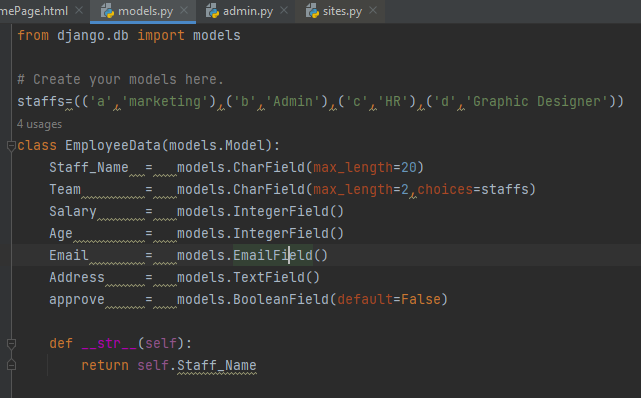
By using this custom admin class and specifying the list\_display attribute, you're customizing how the EmployeeData instances are displayed in the list view of the Django admin panel. The specified fields will be shown in a tabular format when you navigate to the admin page for managing EmployeeData instances.





Admin page

Custom Admin Actions



This is the definition of the EmployeeData model. It has fields such as Staff\_Name, Team with choices specified in staffs, Salary, Age, Email, Address, and approve which is a boolean field indicating approval status.

* Staff\_Name: CharField with a maximum length of 20 characters representing the name of the staff.
* Team: CharField with choices defined by the staffs tuple. It represents the team to which the employee belongs.
* Salary: IntegerField representing the salary of the employee.
* Age: IntegerField representing the age of the employee.
* Email: EmailField representing the email address of the employee.
* Address: TextField representing the address of the employee.
* approve: BooleanField with a default value of False. It represents whether the employee data is approved or not.

**\_\_str\_\_ Method:**

* This method provides a human-readable string representation of the EmployeeData instance. In this case, it returns the Staff\_Name, which is useful when displaying instances in the admin interface or other contexts where a string representation is needed.

This model defines the structure and characteristics of an employee's data, including their name, team, salary, age, email, address, and approval status. The staffs tuple ensures that the Team field has predefined choices.

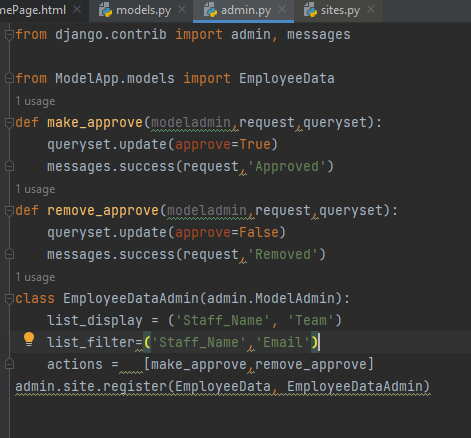
⇒ make\_approve Function:

* This function is a custom admin action for the EmployeeData model.
* It takes three parameters: modeladmin, request, and queryset.
* When triggered, it updates the approve field to True for all selected items in the queryset.
* It also displays a success message using the messages module, indicating that the items have been approved.

⇒

remove\_approve Function:

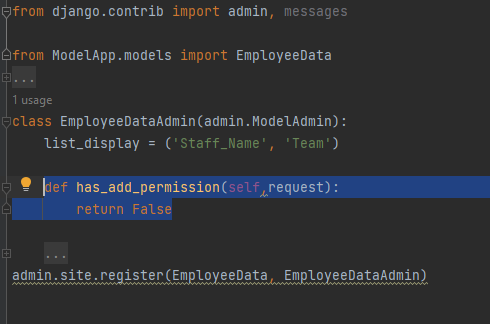
* Similar to make\_approve, this function is a custom admin action for the EmployeeData model.
* It takes three parameters: modeladmin, request, and queryset.
* When triggered, it updates the approve field to False for all selected items in the queryset.
* It displays a success message indicating that the approval has been removed.



⇒EmployeeDataAdmin Class:

* This class configures the appearance and behavior of the EmployeeData model in the Django admin interface.
* list\_display: Specifies the fields to be displayed in the list view, which are 'Staff\_Name' and 'Team'.
* list\_filter: Adds filters to the right sidebar of the admin page for 'Staff\_Name' and 'Email'.
* actions: Includes the custom actions make\_approve and remove\_approve for bulk actions on selected items.
* Registering Model with Admin Site:
  + This line registers the EmployeeData model with the admin site, using the custom admin configuration specified in the EmployeeDataAdmin class.

Overall, this code sets up a Django admin interface for the EmployeeData model with custom actions for bulk approval and removal of approval. The admin page will display the specified fields and filters for easy management of employee data.



The has\_add\_permission method is a Django admin method that you can override in your custom admin class to control whether a user has permission to add new instances of the associated model. In your case, the method is set to always return False, effectively preventing users from adding new instances through the admin interface.



The has\_delete\_permission method is another Django admin method that you can override in your custom admin class to control whether a user has permission to delete instances of the associated model. In this case, the method is set to always return False, preventing users from deleting instances through the admin interface.

