

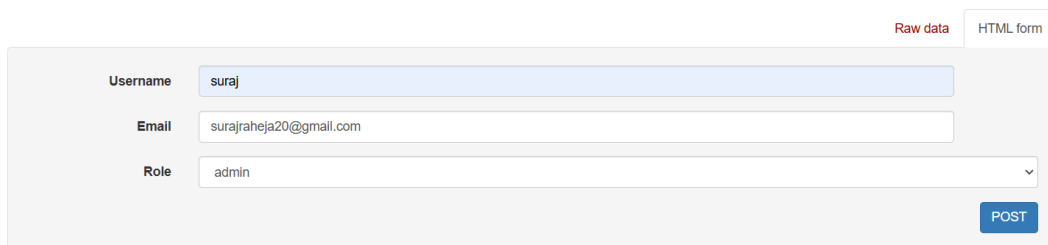
API DOCUMENTATION

User Management :

*Create User-

Implement an API endpoint that accepts user details (e.g., username, email, role) and stores them in the database.

Ensure proper validation and error handling during the user creation process.

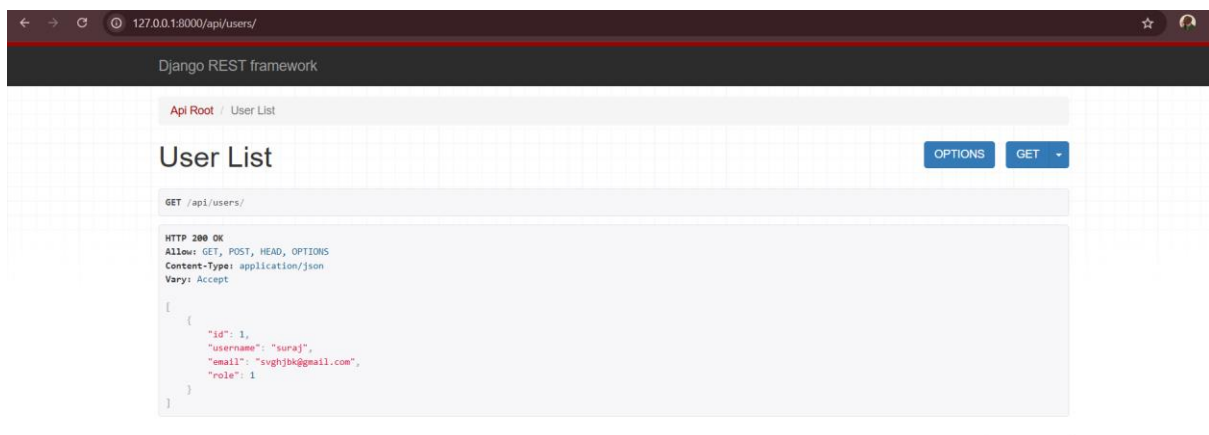


A screenshot of a web form for creating a user. The form has three input fields: 'Username' with the value 'suraj', 'Email' with the value 'surajraheja20@gmail.com', and 'Role' with a dropdown menu showing 'admin'. A blue 'POST' button is at the bottom right. Above the form, there are two tabs: 'Raw data' (highlighted in red) and 'HTML form'.

*Retrieve User Lists-

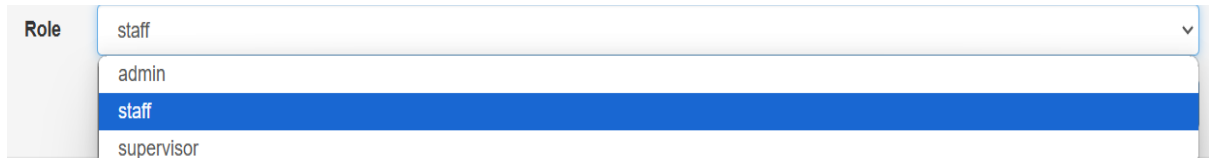
Create an API that fetches a list of all users from the database, with optional filters for id, username, roles or other attributes.

Paginate the results to optimize performance for large datasets.



*Assign predefined roles- (staff, supervisor, admin) to users:

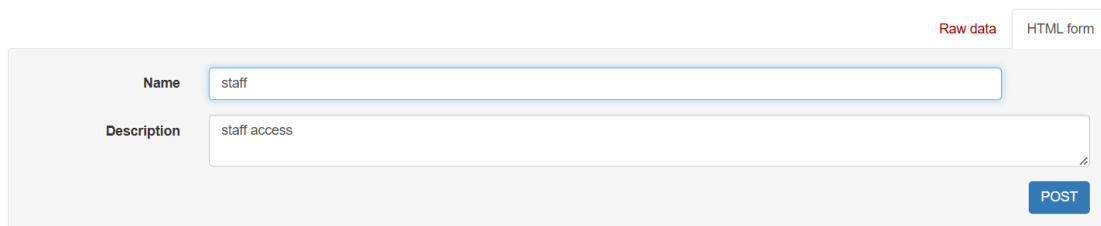
Develop a mechanism to assign predefined roles when creating or updating users. Ensure roles determine user permissions and access within the application (e.g., restricting admins to higher-level functionalities).



#Role Management :

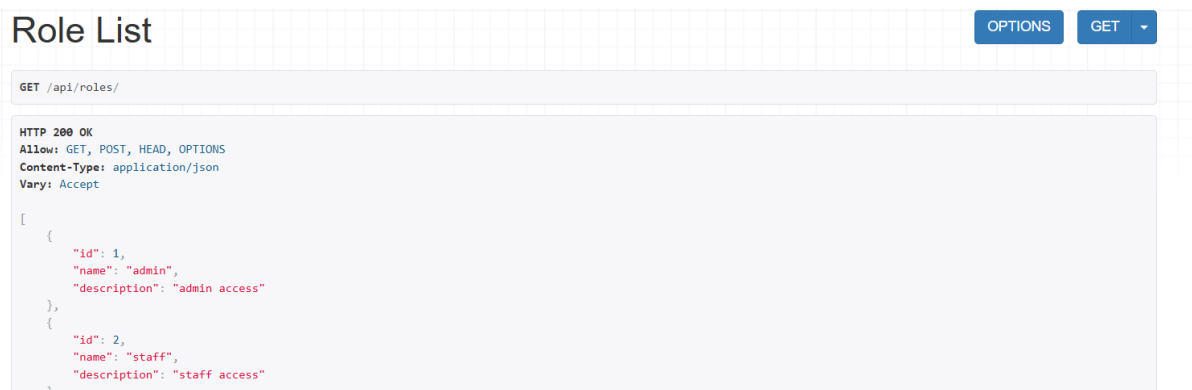
*Predefine the roles (Staff, Supervisor, Admin):

Define the roles in the system with specific access levels and responsibilities, ensuring that each role has predefined permissions. The staff role may have basic access, the supervisor role may manage users and settings, and the admin role will have full control over the system.



*Retrieve the list of roles:

Implement an API that returns a list of all available roles (e.g., staff, supervisor, admin) stored in the system. This can help to display role options for user management and permissions.



```
GET /api/roles/


HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
  {
    "id": 1,
    "name": "admin",
    "description": "admin access"
  },
  {
    "id": 2,
    "name": "staff",
    "description": "staff access"
  }
],
```

Permission Management :

*List all available permissions:

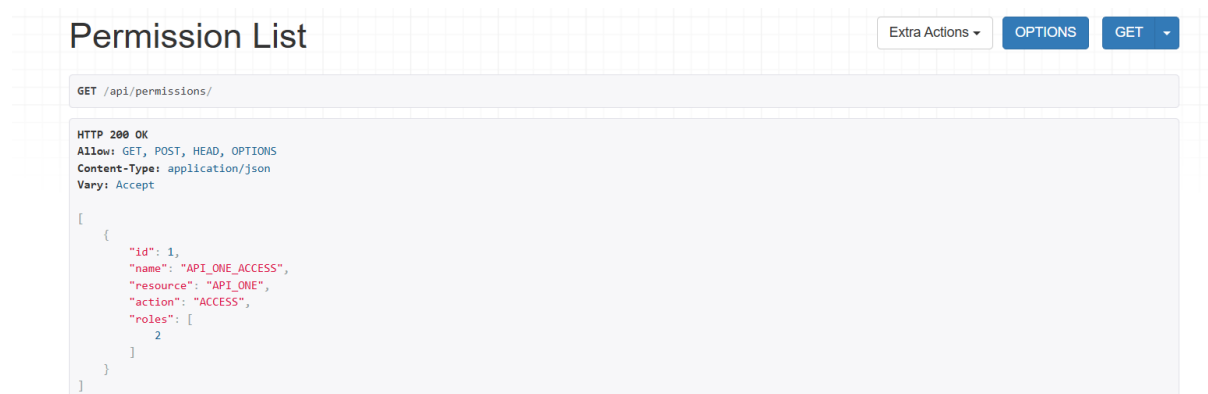
Implement an API that retrieves and lists all available permissions in the system, including descriptions and associated resources. This will allow admins to view and manage permission settings easily.



The screenshot shows a web form for managing permissions. At the top right, there are two tabs: 'Raw data' (highlighted in red) and 'HTML form'. The form has four main sections: 'Name' with a text input containing 'suraj'; 'Resource' with a dropdown menu showing 'API_ONE'; 'Action' with a dropdown menu showing 'ACCESS'; and 'Roles' with a text area containing a list: 'admin', 'staff', and 'supervisor'. A blue 'POST' button is located at the bottom right of the form.

*List permissions assigned to a specific role:

Create an API that returns a list of permissions currently assigned to a specific role, allowing administrators to easily view and modify the permissions associated with each role. This ensures that roles can be properly managed based on their access levels.



The screenshot shows a REST client interface. The title is 'Permission List'. On the right, there are buttons for 'Extra Actions', 'OPTIONS', and 'GET'. The main area displays the details of a GET request to the endpoint '/api/permissions/'. The response is an HTTP 200 OK with the following headers: 'Allow: GET, POST, HEAD, OPTIONS', 'Content-Type: application/json', and 'Vary: Accept'. The response body is a JSON array containing one object:

```
[ {  "id": 1,  "name": "API_ONE_ACCESS",  "resource": "API_ONE",  "action": "ACCESS",  "roles": [    2  ]  } ]
```

#Access Validation:

*Design APIs to validate whether a user can perform a specific action on a resource based on their assigned roles and permissions. Develop an API that checks if a user has the necessary role and permissions to perform actions (like reading, writing, or deleting) on specific resources. This API should verify user access rights before granting or denying access to sensitive data or functionalities.

Api Root / Permission List / Assign permissions

Assign permissions

Extra Actions ▾OPTIONS

GET /api/permissions/assign_permissions/

HTTP 405 Method Not Allowed
Allow: POST, OPTIONS
Content-Type: application/json
Vary: Accept

```
{  
  "detail": "Method \"GET\" not allowed."  
}
```

*Staff: Can access "API_ONE" and "API_TWO":

Design the system to ensure that staff users are only authorized to access specific APIs such as "API_ONE" and "API_TWO," with limited functionality based on their role.

* Supervisor: Can access "API_ONE," "API_TWO," and "API_THREE":

For supervisor users, grant access to additional APIs like "API_THREE" to enable them to perform higher-level functions while maintaining controlled access to the system.

*Admin: Can access all APIs:

Admin users should have unrestricted access to all APIs, enabling them to manage and configure the entire system without any limitation.

Raw dataHTML form

Name

suraj

Resource

API_ONE

Action

ACCESS

Roles

admin

staff

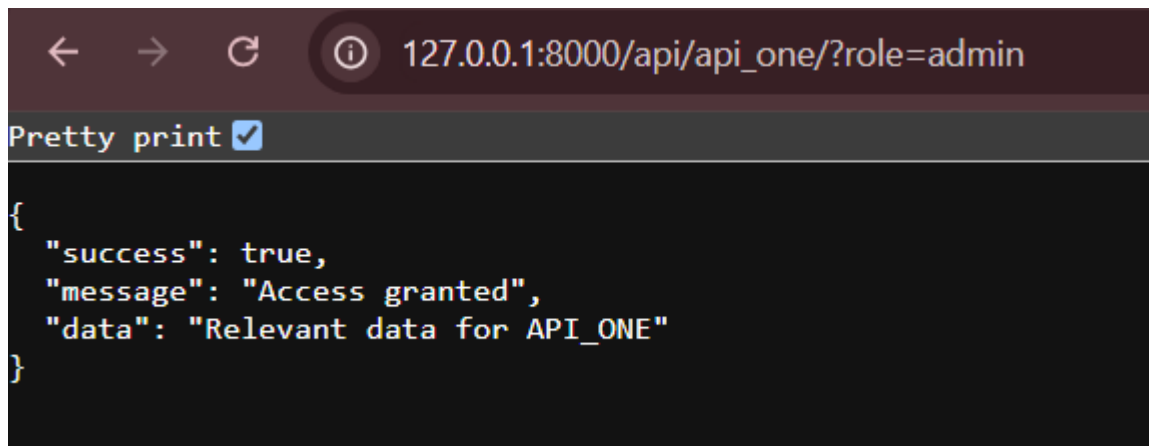
supervisor

POST

#General Response Format:

*Success Response:

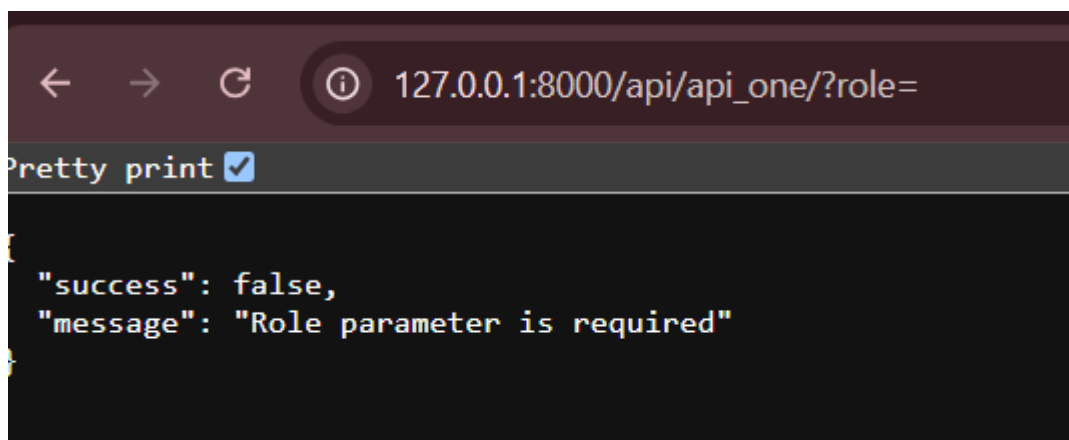
The success response should indicate that the operation was completed successfully, with "success": true, a meaningful "message" explaining the success, and any relevant data in the "data" field. This helps provide clear feedback on the outcome of the request and the data involved.

A screenshot of a web browser window. The address bar shows the URL "127.0.0.1:8000/api/api_one/?role=admin". Below the address bar, there is a "Pretty print" checkbox which is checked. The main content area displays a JSON response:

```
{  "success": true,  "message": "Access granted",  "data": "Relevant data for API_ONE"}
```

*Error Responses:

For error responses, include "success": false along with a detailed "message" that explains the nature of the error (e.g., invalid request, missing parameters, etc.). This helps the client understand what went wrong and how to address the issue.

A screenshot of a web browser window. The address bar shows the URL "127.0.0.1:8000/api/api_one/?role=". Below the address bar, there is a "Pretty print" checkbox which is checked. The main content area displays a JSON response:

```
{  "success": false,  "message": "Role parameter is required"}
```