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.



\*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.

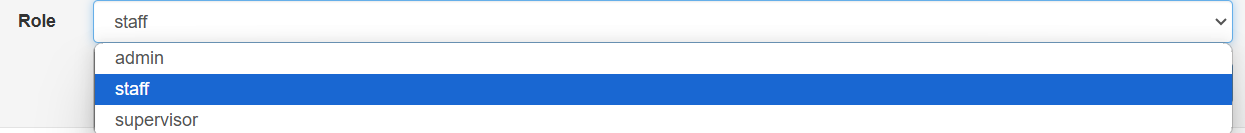
Paginate the results to optimize performance for large datasets.

A screenshot of a computer

Description automatically generated

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

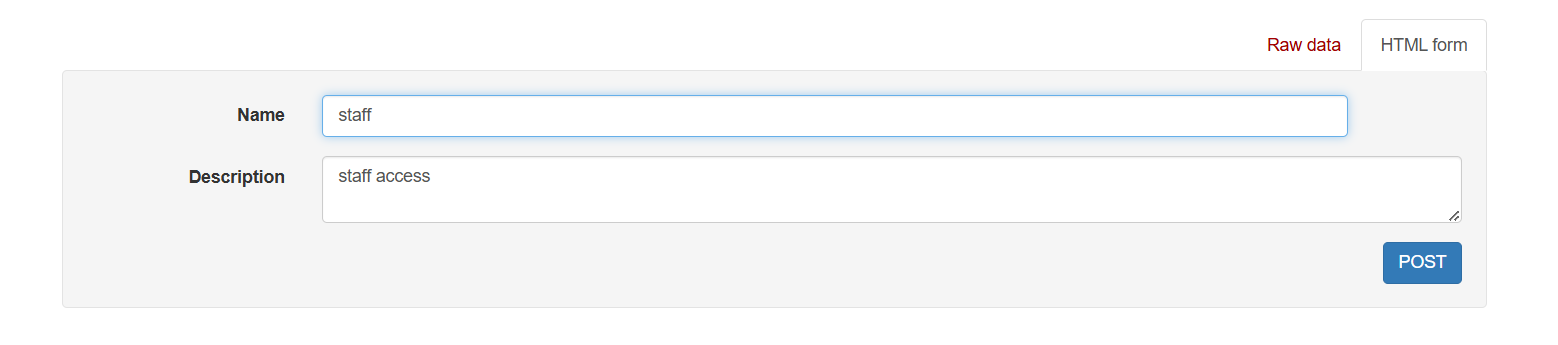
Develop a mechanism to assign predefined roles when creating or updating users. Ensure roles determine user permissions and access within the application (e.g., restricting admins to higher-level functionalities).



#Role Management :

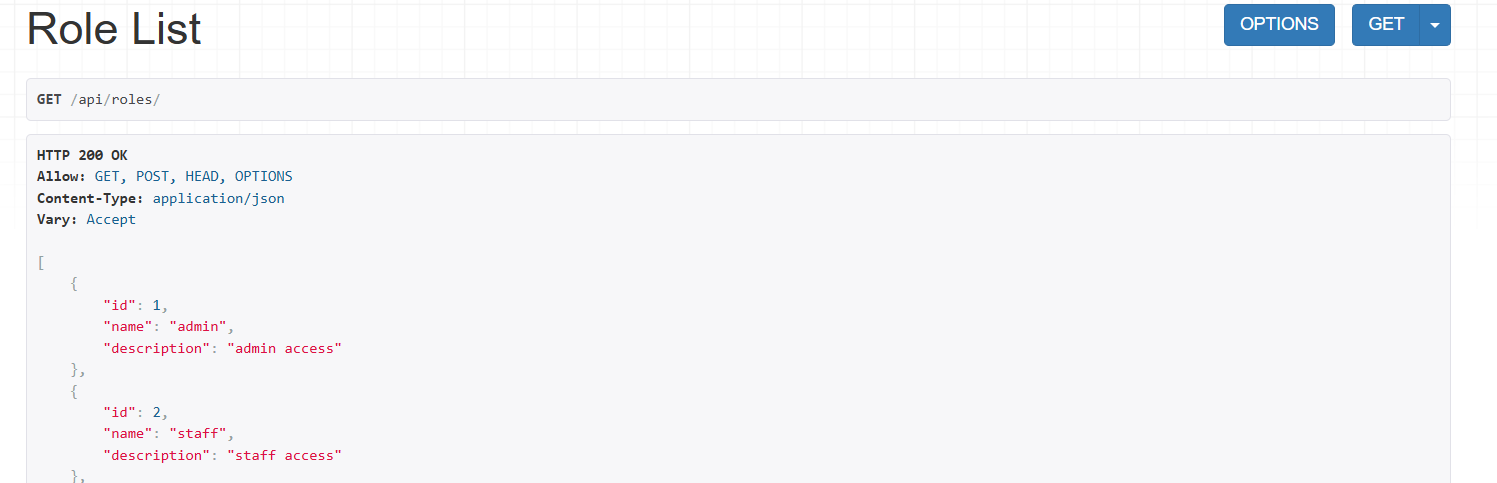
\*Predefine the roles (Staff, Supervisor, Admin):

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:

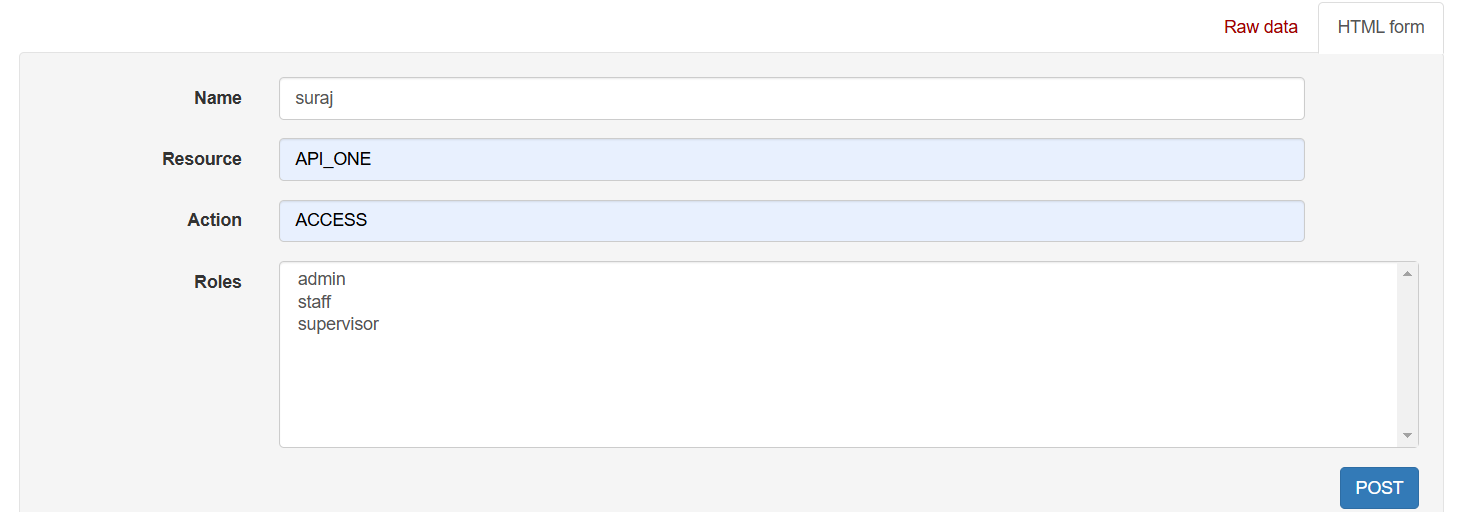
Implement an API that returns a list of all available roles (e.g., staff, supervisor, admin) stored in the system. This can help to display role options for user management and permissions.



# 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.



\*List permissions assigned to a specific role:

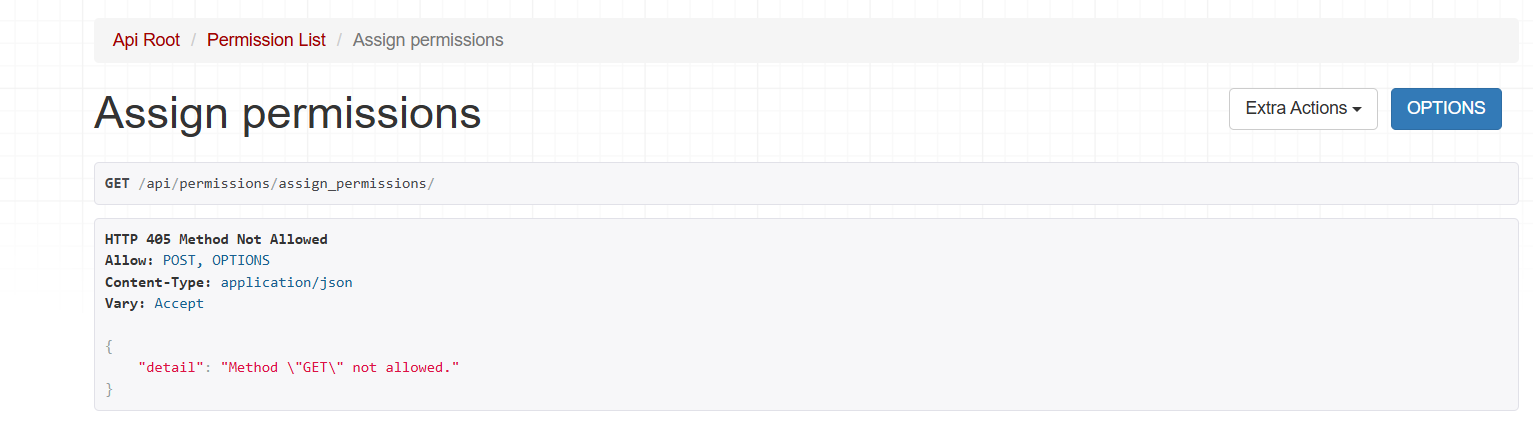
Create an API that returns a list of permissions currently assigned to a specific role, allowing administrators to easily view and modify the permissions associated with each role. This ensures that roles can be properly managed based on their access levels.

A white rectangular object with a grid

Description automatically generated

#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.



\*Staff: Can access "API\_ONE" and "API\_TWO":

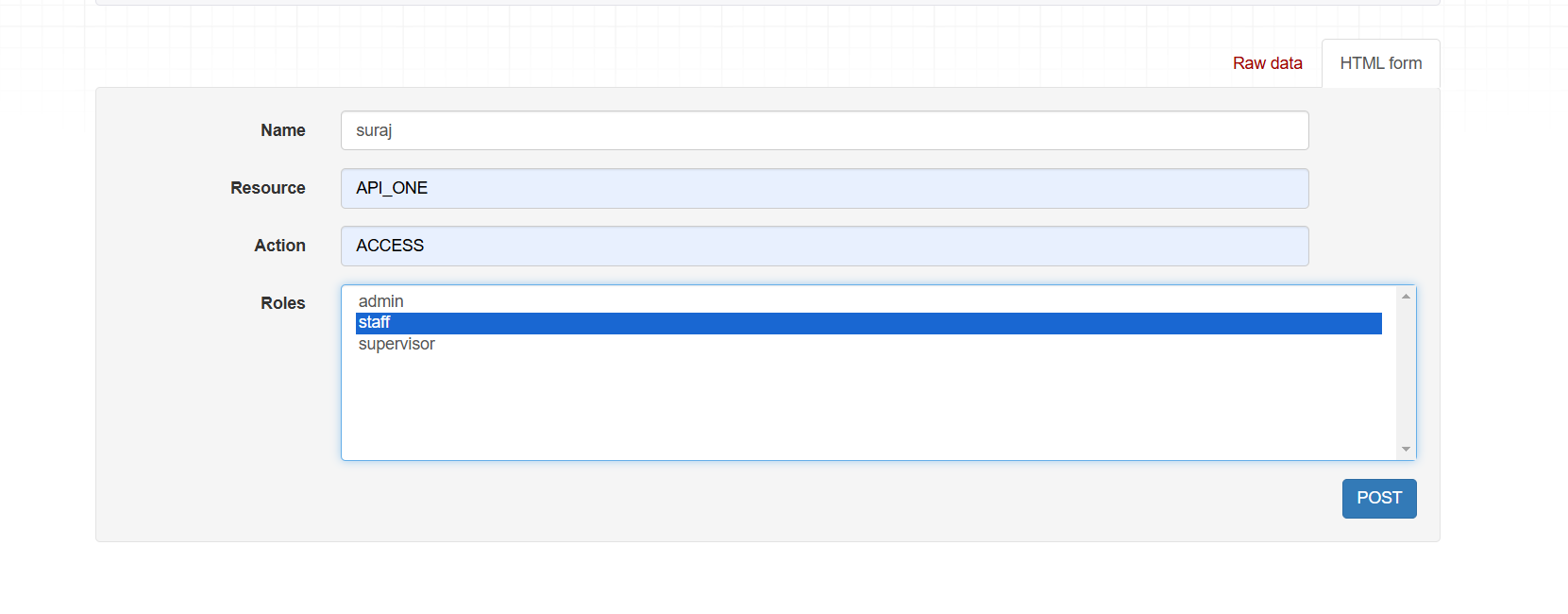
Design the system to ensure that staff users are only authorized to access specific APIs such as "API\_ONE" and "API\_TWO," with limited functionality based on their role.

\* 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:

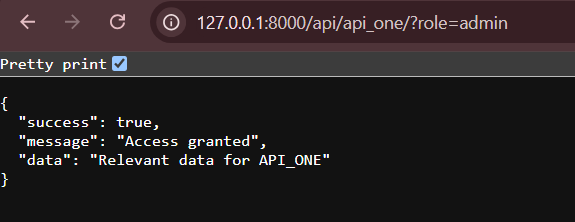
Admin users should have unrestricted access to all APIs, enabling them to manage and configure the entire system without any limitation.



#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.



\*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.

