**Detailed Design**

**For**

**Development of Tags**

Project Name: Tags for eNlight 360

Date: 10/12/2023 to 10/01/2024

Document Version: 01

Prepared by: Mahendra Sonawane

Table of Contents

[1. Introduction 4](#_Toc155361712)

[1.1. Purpose 4](#_Toc155361713)

[1.2. Scope 4](#_Toc155361714)

[2. Tag Attributes 4](#_Toc155361715)

[2.1. Tag Name 4](#_Toc155361716)

[2.2. Scope 4](#_Toc155361717)

[3. Assumptions, Constraints, and Dependencies 5](#_Toc155361718)

[3.1. Assumptions 5](#_Toc155361719)

[3.2. Constraints 5](#_Toc155361720)

[3.3. Dependencies 5](#_Toc155361721)

[4. Operations 5](#_Toc155361722)

[4.1. Assigning or Unassigning Tags to an Object: 5](#_Toc155361723)

[4.2. Assigning or Unassigning a Single Tag to Multiple Objects Simultaneously (Supported Only for VMs): 6](#_Toc155361724)

[4.3. Viewing a List of All Tags in the Inventory: 6](#_Toc155361725)

[4.4. Filtering the List of Tags by Tag Name, Tag Source, and Tag Scope: 6](#_Toc155361726)

[4.5. Viewing a List of Objects Assigned a Specific Tag: 7](#_Toc155361727)

[5. Tags With/Without Scope 7](#_Toc155361728)

[6. Database Relationship and ER Diagram: 7](#_Toc155361729)

[7. Tag Sequence Diagram 9](#_Toc155361730)

[8. Enhancement Overview: 11](#_Toc155361731)

[9. Tag Endpoints and Sample Requests and Responses 12](#_Toc155361732)

[9.1. Create Tags 12](#_Toc155361733)

[9.2. Get Tags 13](#_Toc155361734)

[9.2.1. List All Tags 13](#_Toc155361735)

[9.2.2. List Tags matching scope and tag values 14](#_Toc155361736)

[9.3. Delete Tags 15](#_Toc155361737)

[9.4. Assign Tag to an Object 15](#_Toc155361738)

[9.5. Unassign Tag to an Object 16](#_Toc155361739)

[9.6. Assign Tag to Multiple Object 16](#_Toc155361740)

[9.7. Assign Tag to Multiple Object 17](#_Toc155361741)

[9.8. List of Objects with a Specific Tag and scope 18](#_Toc155361742)

[9.9. List of Tags assigned to Object 20](#_Toc155361743)

[10. Error Handling 20](#_Toc155361744)

[10.1. HTTP Status Codes: 20](#_Toc155361745)

[10.2. Consistent Error Response Format: 21](#_Toc155361746)

[10.3. Logging 21](#_Toc155361747)

[10.4. Custom Exceptions: 21](#_Toc155361748)

[10.5. Additional Error Handling 21](#_Toc155361749)

[11. User Roles and Permissions 22](#_Toc155361750)

[11.1. Administration 22](#_Toc155361751)

[11.2. Standard User 22](#_Toc155361752)

[12. Conclusion 22](#_Toc155361753)

# Introduction

Tags in the eNlight360 system play a crucial role in organizing and managing objects efficiently. This documentation provides an in-depth guide on the usage and management of tags, covering attributes, operations, and best practices.

## Purpose

Tags in the eNlight360 system serve as invaluable tools for efficient organization and management of objects. This documentation is designed to provide an exhaustive guide on the usage and administration of tags, encompassing key attributes, operational procedures, and best practices. Understanding and implementing tags is crucial for enhancing searchability, facilitating troubleshooting and tracing activities, and performing various related tasks within the eNlight360 ecosystem.

## Scope

This documentation targets administrators, developers, and users involved in the eNlight360 system, aiming to empower them with comprehensive knowledge of tags. By utilizing tags, users can seamlessly label objects, thereby enabling quick retrieval, troubleshooting, and task execution. The scope of this documentation extends to cover the creation, assignment, and management of tags, with a particular focus on their attributes, such as tag name and scope.

# Tag Attributes

Tags within the eNlight360 system possess key attributes that define their characteristics. Understanding these attributes is fundamental to effectively working with tags.

## Tag Name

The Tag Name serves as the unique identifier for a tag. It is case-sensitive and must be distinct to avoid conflicts. When creating tags, it's essential to choose descriptive and meaningful names to enhance clarity and ease of use.

## Scope

The Scope attribute is optional and provides additional context to the tag name. Acting as a key, the scope allows users to categorize tags further. For instance, tagging objects based on their operating system (Windows, Linux, Mac) can involve scopes like OS. The scope contributes to a more organized and structured tagging system.

# Assumptions, Constraints, and Dependencies

## Assumptions

Database: The tagging system assumes the use of MySQL for database operations.

User Input: It is assumed that users will provide valid and required inputs when interacting with the tagging functionalities.

Concurrency: The system assumes a non-concurrent environment for tag management operations.

## Constraints

Tag Limit: Each object can have a maximum of 30 tags. This constraint should be considered during system design and usage.

Group Criteria Limit: The system supports up to 5 AND/OR group criteria for tags. Design considerations should account for this limitation.

## Dependencies

Database Connection: The system is dependent on a stable connection to the MySQL database for tag storage and retrieval.

# Operations

## Assigning or Unassigning Tags to an Object:

This involves assign or unassign tags to/from a specific object within the eNlight360 system. Tags are typically metadata or labels that help organize and categorize objects for better management.

**Procedure:**

1. Log in to eNlight360 through your web browser.
2. Navigate to the section or module that corresponds to the type of object you want to tag (e.g., Virtual Machines, Storage, etc.).
3. Locate the specific object to which you want to assign or unassign tags.
4. Enter the edit mode for the object. This is usually done by selecting the object and choosing an "Edit" option from the context menu.
5. Look for a section or tab related to tags within the object's settings.
6. Assign or unassign tags by selecting them from the available options or entering new tag names.
7. Save the changes to apply the tags to the object.

## Assigning or Unassigning a Single Tag to Multiple Objects Simultaneously (Supported Only for VMs):

Users can apply or remove a specific tag to/from multiple objects at the same time. However, this feature seems to be limited to Virtual Machines (VMs) in the eNlight360 environment. This bulk operation can save time and effort when managing tags for multiple VMs.

**Procedure:**

1. Log in to eNlight360.
2. Navigate to the module for Virtual Machines.
3. Select the option for bulk tag management or multi-object selection.
4. Choose the VMs to which you want to assign or unassign the tag.
5. Look for a bulk tag assignment or unassignment feature.
6. Specify the tag you want to assign or unassign.
7. Apply the changes to update the tags for the selected VMs.

## Viewing a List of All Tags in the Inventory:

Users can access a comprehensive list that displays all the tags currently available in the eNlight360 inventory. This provides an overview of the existing tags, making it easier to manage and organize them.

**Procedure:**

1. Log in to eNlight360.
2. Navigate to the Tags or Inventory Management section.
3. Look for an option to view the complete list of tags.
4. Explore additional details such as tag names, sources, and scopes if available.

## Filtering the List of Tags by Tag and Scope:

The system allows users to filter the list of tags based on specific criteria such as tag name, tag source, and tag scope. This filtering capability is useful for quickly locating and working with specific tags, especially in large inventories.

**Procedure:**

1. Log in to eNlight360.
2. Go to the Tags or Inventory Management section.
3. Use the available filter options to narrow down the list based on tag name, scope or both.
4. Apply the filters to display the desired subset of tags.

## Viewing a List of Objects Assigned a Specific Tag:

Users can retrieve a list of objects that have been assigned a particular tag. This functionality is beneficial for tracking and managing objects associated with specific tags. It helps users understand the relationships between tags and objects in the eNlight360 system.

**Procedure:**

1. Log in to eNlight360.
2. Navigate to the Tags or Inventory Management section.
3. Select the specific tag you are interested in.
4. View the list of objects associated with that tag.

# Tags With/Without Scope

eNlight360 tags can be created with or without scope, allowing flexibility in grouping workloads. When using tags with scope:

* Enables proper tag inventory management with both key and value.
* Allows inserting more tags/metadata than the 30-tag limit per object.
* Supports more than 5 AND/OR GROUP criteria indirectly.
* The best practice is to keep it simple when within the eNlight360 supported limit, using multiple individual tags with optional scope.

# Database Relationship and ER Diagram:

This database system manages the tagging of virtual machines (VMs) by users. It allows users to attach relevant tags to VMs and other Entities, enabling efficient searching, filtering, and organization within the eNlight environment.

**Table Descriptions:**

1. vms:

* vm\_id: Unique identifier for each virtual machine (primary key).
* vm\_name: Descriptive name of the virtual machine.
* Creation Date: Date and time the virtual machine were created.

1. tags:

* tag\_id: Unique identifier for each tag (primary key).
* tag: Textual representation of the tag (e.g., " prod", " hr\_app", " web").
* Scope: Defines the specific domain or context where the tag applies (e.g., enviornment, app\_name, app\_tier).

1. tags\_relation:

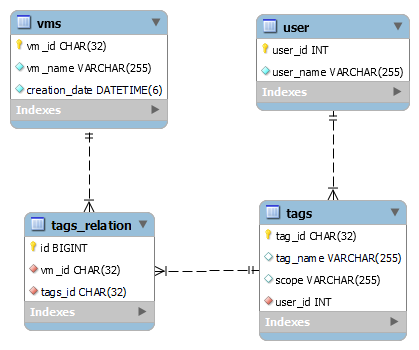
* id: Unique identifier for tags\_relation table (primary key).
* vm\_id: Foreign key referencing the ID of the virtual machine.
* tag\_id: Foreign key referencing the ID of the tag.

**Relationships:**

* Many-to-Many Relationship: Each virtual machine can have many associated tags, and each tag can be associated with many virtual machines. This relationship is implemented through the "tags\_relation" table.

Indexes:

* vm\_id index: Speeds up queries that search for virtual machines based on their tags.
* tag\_id index: Speeds up queries that search for tags associated with specific virtual machines.
* Additional Indexes: Depending on the expected query patterns, you might consider indexes on the "scope" or "name" attributes in the "tags" table.



# Tag Sequence Diagram

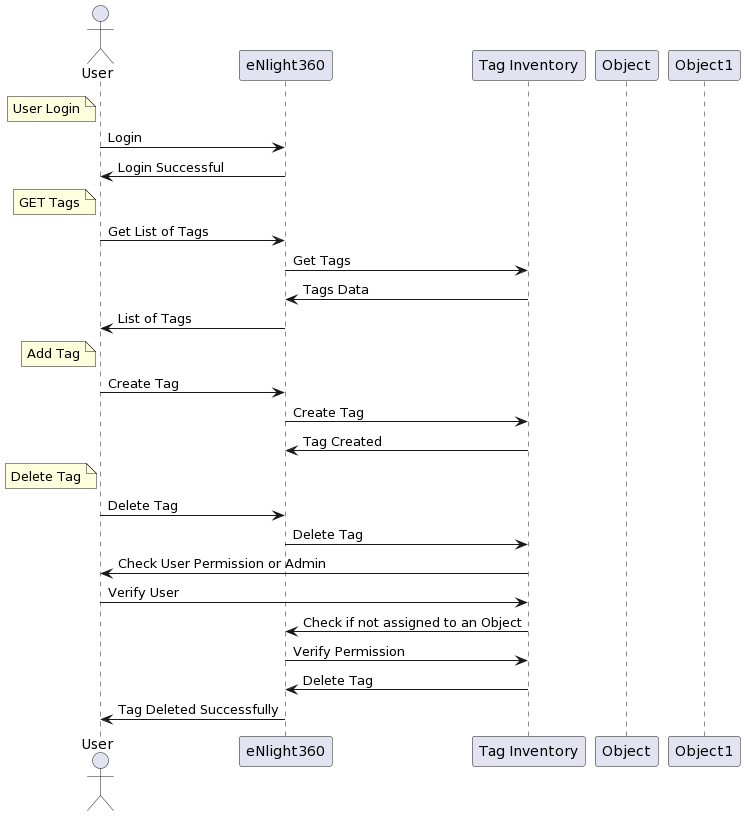
Below diagram illustrates the key interactions and state transitions involved in managing tags within the system.

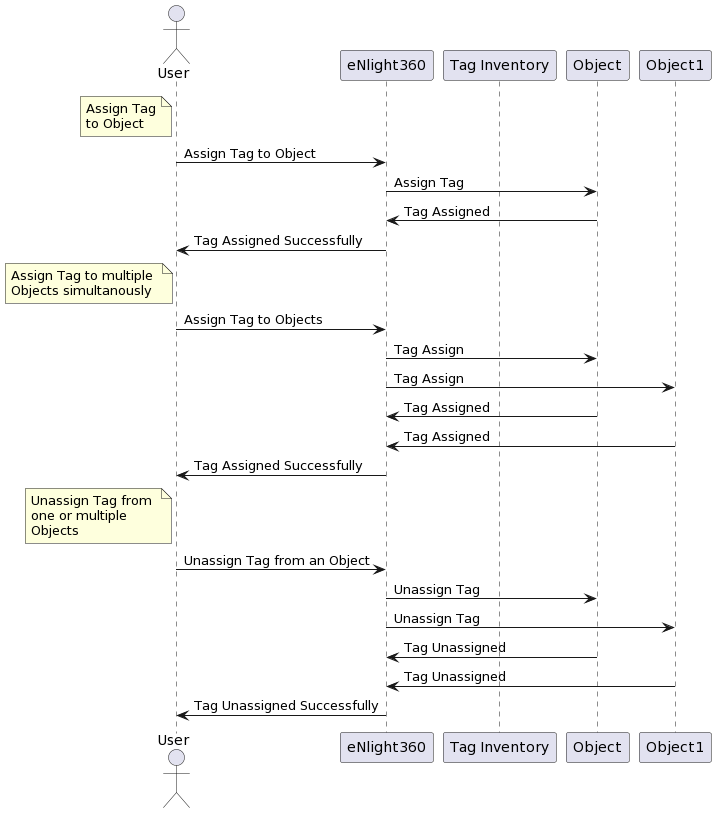
**Key Elements:**

* Sequence Diagram: Shows user interactions for creating, editing, assigning, and unassign tags.
* State Machine Diagram: Tracks the behavior of Tag Inventory Objects, including their states and transitions

**Key Functionalities:**

* Tag Creation and Editing: Users can add and modify tags.
* Tag Assignment and Unassignment: Users can link and unlink tags from relevant objects.
* Tag Inventory Management: The system tracks tags associated with each object.





# Enhancement Overview:

Key Features of the Enhancement:

1. **Universal Tag Assignment:**

With this enhancement, users can assign tags to a wide range of entities, including but not limited to Virtual Machines (VMs), Storage, Networks, and more. This flexibility ensures a comprehensive approach to entity classification and organization.

1. **Centralized Database Integration:**

All tag-related details are now seamlessly integrated into the existing eNlight360 database. This integration not only ensures data consistency but also allows for efficient cross-referencing and retrieval of information related to tagged entities.

1. **Improved User Interface:**

The user interface has been refined to accommodate the expanded tagging functionality. Users can easily navigate through the enhanced tagging features, accessing a unified and intuitive interface for assigning, modifying, and viewing tags across different entities.

1. **Bulk Tag Operations:**

Users can perform bulk tag operations, allowing for simultaneous assignment or removal of tags across multiple entities. This feature is particularly beneficial for scenarios where large-scale tagging operations are required, enhancing efficiency and reducing manual effort.

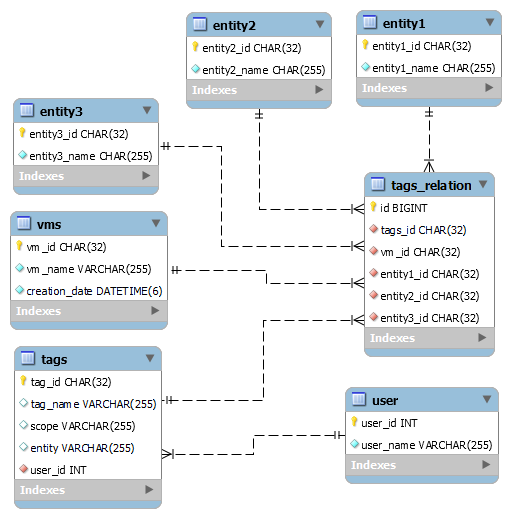
1. **Enhanced Search and Filtering:**

The enhancement includes advanced search and filtering options, empowering users to quickly locate tagged entities based on tag names, sources, scopes, or other relevant criteria. This contributes to a more efficient and organized entity management experience.

1. **Real-time Updates:**

Changes made to tags are reflected in real-time across the entire eNlight360 environment. This ensures that users have access to the latest information, promoting accuracy and consistency in entity tagging.

* Below Diagram shows ER diagram of enhanced Tags management system.



# Tag Endpoints and Sample Requests and Responses

## Create Tags

Request:

Method: POST

URI Path(s): /tags

Parameter: tag\_name, scope, entity, user\_id

Successful Response:

Response Code:200 OK

Example:

* Request

{

    "tag\_name": "mac",

    "scope": "os",

    "entity": VM

    "user\_id": "2"

}

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "Tag Added successfully",

    "data": ""

}

## Get Tags

### List All Tags

Request:

Method: GET

URI Path(s): /tags

Successful Response:

Response Code:200 OK

Example:

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "Tags get successfully",

    "data": [

        {

            "tag\_id": "28e15780-1073-4a47-aa82-4e8f755821be",

            "tag\_name": "linux",

            "scope": "os",

            "entity": "vm",

            "user\_id": 2

        },

        {

            "tag\_id": "40d3f407-4971-445f-9d0a-127176ddbc21",

            "tag\_name": "windows",

            "scope": "os",

            "user\_id ": 2

        }}

### List Tags matching scope and tag values

Request:

Method: GET

URI Path(s): / tags?tag\_name=" "

URI Path(s): / tags?scope=" "

Successful Response:

Response Code:200 OK

Example Response:

* For tags:

URL paths: /tags?tag\_name=windows

{

    "status": "success",

    "error\_code": 0,

    "message": "Tags get successfully",

    "data": [

        {

            "tag\_id": "40d3f407-4971-445f-9d0a-127176ddbc21",

            "tag\_name": "windows",

            "scope": "os",

            "user\_id\_id": 2

        }

    ]

}

* For Scope:

URL paths: /tags?scope=env

{

    "status": "success",

    "error\_code": 0,

    "message": "Tags get successfully",

    "data": [

        {

            "tag\_id": "ff6c8018-31f3-40fd-b0cd-2408c7fc31b5",

            "tag\_name": "dev",

            "scope": "env",

            "user\_id\_id": 2

        }

    ]

}

## Delete Tags

Request:

Method: DELETE

URI Path(s): / tags/<str:id>

Successful Response:

Response Code:200 OK

Example Response:

{

    "status": "success",

    "error\_code": 0,

    "message": "Tag deleted successfully."

}

## Assign Tag to an Object

Request:

Method: POST

URI Path(s): /assign\_unassign\_tags

Successful Response:

Response Code:200 OK

Example Response:

* Request

{

    "action": "assign",

    "tag\_name": "mac",

    "vm\_ids": [

        "5bfa8528be654ee08d8348cbee442892"

    ]

}

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "Tag Assigned to Objects successfully",

    "data": ""

}

## Unassign Tag to an Object

Request:

Method: POST

URI Path(s): /assign\_unassign\_tags

Successful Response:

Response Code:200 OK

Example Response:

* Request

{

    "action": "unassign",

    "tag\_name": "mac",

    "vm\_ids": [

        "5bfa8528be654ee08d8348cbee442892"

    ]

}

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "Tag Unassigned from Objects successfully",

    "data": ""

}

## Assign Tag to Multiple Object

Request:

Method: POST

URI Path(s): /assign\_unassign\_tags

Successful Response:

Response Code:200 OK

Example Response:

* Request

{

    "action": "assign",

    "tag\_name": "windows",

    "vm\_ids": [

        "2b70bf732a5e428eaeb8da03cadc2ea9",

        "c59b77b6ac9543f89207b2dcd057d18c",

        "f28fae63a1bd468082c0dff226dff173"

    ]

}

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "Tag Assigned to Objects successfully",

    "data": ""

}

## Unassign Tag to Multiple Object

Request:

Method: POST

URI Path(s): /assign\_unassign\_tags

Successful Response:

Response Code:200 OK

Example Response:

* Request

{

    "action": "unassign",

    "tag\_name": "mac",

    "vm\_ids": [

        "2b70bf732a5e428eaeb8da03cadc2ea9",

        "c59b77b6ac9543f89207b2dcd057d18c",

        "f28fae63a1bd468082c0dff226dff173"

    ]

}

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "Tag Unassigned from Objects successfully",

    "data": ""

}

## List of Objects with a Specific Tag or scope

Request:

Method: GET

URI Path(s): /vms?tag\_name=windows

Successful Response:

Response Code:200 OK

Example:

* 1. **For Tag**
* Request

{

    Tag\_name:windows

}

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "VMs retrieved successfully",

    "data": [

        {

            "vm\_id": "2b70bf73-2a5e-428e-aeb8-da03cadc2ea9",

            "vm\_name": "vm11",

            "creation\_date": "2024-01-01T08:54:50.813Z"

        },

        {

            "vm\_id": "c59b77b6-ac95-43f8-9207-b2dcd057d18c",

            "vm\_name": "vm10",

            "creation\_date": "2024-01-01T08:54:48.682Z"

        }

    ]

}

* 1. **For Scope**
* Request

{

    scope:os

}

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "VMs retrieved successfully",

    "data": [

        {

            "vm\_id": "12fccd49-b5ce-43e2-af96-232415e30e42",

            "vm\_name": "vm",

            "creation\_date": "2024-01-01T08:37:36.142Z"

        },

        {

            "vm\_id": "658a3c8d-2add-4d7d-b0f5-92c599119a08",

            "vm\_name": "vm1",

            "creation\_date": "2024-01-01T08:38:52.833Z"

        }

    ]

}

* 1. **For Tag and Scope**
* Request

{

    Tag\_name:windows

    scope:os

}

* Response

{

    "status": "success",

    "error\_code": 0,

    "message": "VMs retrieved successfully",

    "data": [

        {

            "vm\_id": "12fccd49-b5ce-43e2-af96-232415e30e42",

            "vm\_name": "vm",

            "creation\_date": "2024-01-01T08:37:36.142Z"

        }

    ]

}

## List of Tags assigned to Object

Request:

Method: GET

URI Path(s): /vms?vm\_id=10d146db88454c6585496e0b1ff0dc6f

Successful Response:

Response Code:200 OK

Example:

{

    "status": "success",

    "error\_code": 0,

    "message": "VMs retrieved successfully",

    "data": [

        {

            "vm\_id": "10d146db88454c6585496e0b1ff0dc6f",

            "vm\_name": "vm2",

            "tags": [

                {

                    "tag\_name": "xen",

                    "scope": "hypervisor"

                },

                {

                    "tag\_name": "test",

                    "scope": "env"

                }

            ]

        }

    ]

}

# Error Handling

## HTTP Status Codes:

Ensure that the HTTP status codes returned in your JSON responses accurately reflect the success or failure of the operation. For example, use status=200 for successful responses, status=201 for resource creation, and appropriate error codes (e.g., status=400, status=404, status=500) for error responses.

## Consistent Error Response Format:

Maintain a consistent format for error responses. For instance, include an error key in the response containing an object with code and message keys. This makes it easier for clients to handle errors consistently.

## Logging

Implement logging for errors to keep track of issues. Use Django's built-in logging framework to log errors, stack traces, and additional information that can help in debugging.

## Custom Exceptions:

Consider creating custom exception classes for more specific error handling. This can help in distinguishing different types of errors and providing more meaningful error messages.

Example of custom error handling given below:

try:

if form.is\_valid():

# ... (code)

else:

raise InvalidTagDataError("Invalid tag data: {0}".format(form.errors))

except InvalidTagDataError as e:

data = {'status': 'error', 'error\_code': 104, 'message': "error: {0} ".format(e)}

return JsonResponse(data)

## Additional Error Handling

try:

# ... (code)

except ValidationError as e:

data = {'status': 'error', 'error\_code': 103, 'message': "Validation error: {0} ".format(e)}

return JsonResponse(data)

except NameError as e:

data = {'status': 'error', 'error\_code': 103, 'message': "Name error: {0} ".format(e)}

return JsonResponse(data)

except IntegrityError:

data = {'status': 'error', 'error\_code': 102, 'message': \_("This Tag Already exists")}

return JsonResponse(data)

# User Roles and Permissions

## Administration

* Can perform all tag-related operations.
* Can create, delete any tags.
* Can assign and unassign tags from objects.
* Has full access to tag management functionalities.

## Standard User

* Can view tags and their details.
* Can create tags, and able to delete tags if same user has created the tag
* Can assign and unassign tags to/from objects that they have access to.
* Limited access compared to the administrator, focused on tag assignments and viewing.

# Conclusion

In conclusion, the tag management system provides a powerful mechanism for organizing and managing objects efficiently. By following the provided documentation, coding standards, and conducting comprehensive testing to developed tag management system.