Video CMS Backend Scope

Version: 1.1

Changes:

09-05-2023: added user registration API details.

Contents

[User Authentication and Profile 1](#_Toc134524710)

[Screen Management 2](#_Toc134524711)

[Video Management 2](#_Toc134524712)

[Collection Management 2](#_Toc134524713)

[Table Structure (User micro services) 3](#_Toc134524714)

[Cms\_users 3](#_Toc134524715)

[token 3](#_Toc134524716)

[Otp\_table 3](#_Toc134524717)

[User\_roles 3](#_Toc134524718)

[User\_permissions 3](#_Toc134524719)

[User\_activity\_logs 4](#_Toc134524720)

[APIs 4](#_Toc134524721)

[**Create user/Registration** 4](#_Toc134524722)

# User Authentication and Profile

1. Token
   1. Generate token
   2. Refresh token
   3. Delete token
2. OTP
   1. Send OTP (email and phone)
   2. Resend OTP (email and phone)
   3. Verify OTP
3. User Profile
   1. Create/Register user
   2. Update user
   3. Get user (sorting/searching/filter)
   4. Delete/Deactivate user
   5. Forgot password
   6. Update password
4. User Roles and Permissions
   1. Create roles
   2. Update roles
   3. Get roles
   4. Delete/reset roles
   5. Add permissions
   6. Update permissions
   7. Get permissions (sorting/searching/filter)
   8. Delete/reset permissions
   9. Manage permission for a role
   10. Manage role for a user

# Screen Management

1. Templates
2. Pages
3. Components / Widgets

# Video Management

1. Videos list (movies/short films/other fixed length videos)
2. TV series (videos having seasons and episodes)
3. Metadata
4. EPG
5. Live TV channels
6. Music

# Collection Management

1. Categories
2. Tags
3. Genre
4. Regions
5. Languages
6. Ads
7. DRM (optional)
8. Subscription Management
9. Recommendation Engine (backend)

# Table Structure (User micro services)

## Cms\_users

1. ID (primary key)
2. First\_name (mandatory)
3. Last\_name (optional)
4. Email (mandatory)
5. Emp\_id (optional)
6. Password (mandatory)
7. Role (mandatory)
8. Phone (mandatory)
9. Email\_verified (mandatory)
10. Phone\_verified (mandatory)
11. createdAt (mandatory)
12. updatedAt (optional)
13. is\_active (mandatory)

## token

1. ID (primary key)
2. Device\_id (optional)
3. Token (optional)
4. User\_id (mandatory)
5. Cms\_user (mandatory)

## Otp\_table

1. ID (primary key)
2. User\_id (mandatory)
3. Otp\_type (mandatory)
4. Phone\_code (mandatory)
5. Phone\_number (mandatory)
6. Email (mandatory)
7. Otp\_code (mandatory)
8. Expiry\_date (mandatory)
9. No\_of\_attempts (mandatory)
10. Is\_expired (mandatory)

## User\_roles

1. ID (primary key)
2. Role\_name (mandatory)
3. Role\_type (mandatory)
4. Permissions (mandatory)
5. Status (mandatory)

## User\_permissions

1. ID (primary key)
2. Permission\_name (mandatory)
3. Permission\_type (mandatory)
4. Collection (mandatory)
5. status (mandatory)

## User\_activity\_logs

1. ID (primary key)
2. User\_id (mandatory)
3. Activity (mandatory)
4. Collection (optional)
5. Datetime (mandatory)
6. Status (mandatory)

## APIs

### Create user/Registration

* 1. **Description:** APIs to register a user for CMS. User can do self-registration or a admin or super admin can add new user in CMS dashboard.
  2. **Workflow:**
     + 1. Open Registration page
       2. User fills the registration form
       3. Click on submit/register button
       4. Call registration API
       5. On success, backend will update the user details in the database
       6. If user already exists, the details won’t be updated
  3. **Endpoint:** /register
  4. **Method:** POST
  5. **Request:** 
     1. Request type:
        1. Body (JSON)
     2. Request parameters:
        1. Body:
           1. First Name
           2. Last Name
           3. Email
           4. Password (encoded format)
           5. Role
           6. Employee ID
           7. Phone
     3. Request JSON:

{

“first\_name”: String,

“last\_name”: String,

“email”: String,

“password”: String,

“role”: String,

“phone”:String,

“emp\_id”:String

}

* 1. **Response:**
     1. Success:

**{**

**"response": {**

**"code": 201,**

**"status": "success",**

**"alert": [**

**{**

**"message": "User created successfully",**

**"type": "created"**

**}**

**]**

**}**

**}**

* + 1. Failure:

**{**

**"response": {**

**"code": 409,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "email | phone no | emp\_id already exists.",**

**"type": "failure"**

**}**

**]**

**}**

**}**

### Login

1. **Description:** APIs to login a user for CMS.
2. **Workflow:**

i. Login

* + - 1. Open Login page
      2. User fills the login form
      3. Click on submit/login button
      4. Calls login API
      5. On success, backend will send an access token and a token type “bearer”, the token expires if the user is inactive for 5 minutes.
      6. The token will be deleted on log out.

1. **Endpoint:** /login
2. **Method:** POST
3. **Request:**

i.Request type:

* + - * 1. Body (JSON)

ii.Request parameters:

Body:

* + - * 1. Email
        2. Password (encoded format)

iii.Request JSON:

{

“email”: String,

“password”: String,

}

1. **Response:**
2. Success:

**{**

**"response": {**

**"code": 201,**

**"status": "success",**

**"alert": [**

**{**

**"message": "Login successful",**

**"type": "login"**

**}**

**],**

**"data": {**

**"access\_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhbmlydWRoaGVyYWR5MTdAZ21haWwuY29tIiwiZXhwIjoxNjg1NDQxMzM4fQ.8ooCJbawmDhPym0ikV4wlV92LhFYLy-EtaxRcIM8aTo",**

**"token\_type": "bearer"**

**},**

**"is\_data": 1**

**}**

**}**

ii. Failure:

**{**

**"response": {**

**"code": 404,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "Incorrect username or password.",**

**"type": "failure"**

**}**

**]**

**}**

**}**

**}**

### Send otp

1. **Description:** APIs to send otp.
2. **Workflow:**

i. Forgot password

* + - 1. Opens Forgot Password page
      2. Enter email in the form
      3. Click on submit button
      4. Calls send email otp API
      5. On success, backend will send an otp to the user’s email address .
      6. This otp will expire in 5 minutes.

ii. Verify user

1. Opens Verify user page
2. Enter email in the form
3. Click on submit button
4. Calls send email otp API
5. On success, backend will send an otp to the user’s email address .
6. This otp will expire in 5 minutes.
7. **Endpoint:** /sendotp
8. **Method:** GET
9. **Request:** 
   * 1. Request type:
        1. Query Parameter
     2. Request parameters:
        1. Email
10. **Response:**
11. Success

**{**

**"response": {**

**"code": 201,**

**"status": "success",**

**"alert": [**

**{**

**"message": "Otp sent successfully",**

**"type": "generated"**

**}**

**],**

**"data": {**

**"otp": 548780**

**},**

**"is\_data": 1**

**}**

**}**

ii. Failure:

**{**

**"response": {**

**"code": 404,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "Incorrect email.",**

**"type": "failure"**

**}**

**],**

**"is\_data": 0**

**}**

**}**

### **Verify User**

* **Description:** To verify the users email and phone number
* **Workflow:**
  + 1. The API receives a POST request with the user's OTP code.
    2. It calls the verifyuser function and passes the OTP code and the database as an argument.
    3. If no OTP record is found, it returns an error response indicating that the OTP code is invalid.
    4. If an OTP record is found, it retrieves the user associated with that OTP record from the CmsUsers table based on the user ID stored in the OTP record.
    5. If the user is found, it marks the user's email and phone as verified.
    6. If no user is found, it returns an error response indicating that the user is not found.
* **Endpoint:** /verify\_user
* **Method:** PUT
* **Request:** 
  + 1. Request type:
       1. Query Parameter
       2. Header(Access Token)
    2. Request parameters:
       1. OTP
* **Response:**

Success:

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "User successfully verified.",**

**"type": "Verify"**

**}**

**]**

**}**

**}**

Failure:

**{**

**"response": {**

**"code": 400,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "Invalid OTP code.",**

**"type": "failure"**

**}**

**]**

**}**

**}**

**}**

**{**

**"response": {**

**"code": 400,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "Your OTP has expired.",**

**"type": "failure"**

**}**

**]**

**}**

**}**

### **Forgot password**

* **Description:** To be able to update a new password
* **Workflow:**
  + 1. The API receives a POST request with the user OTP, new password, confirm password
    2. If no OTP record is found, it returns a response indicating that the OTP code is invalid.
    3. It checks if the OTP has expired by comparing the current UTC time with the OTP's expiry date
    4. It retrieves the user record associated with the OTP from the CmsUsers table based on the user ID stored in the OTP record.
    5. If a user record is found, it checks if the new password matches the confirm password. If they don't match, it returns a response indicating that the passwords do not match.
    6. It commits the changes to the database and returns a response indicating that the password has been successfully updated.
    7. If no user record is found, it returns a response indicating that the user was not found.
* **Endpoint:** /forgot\_password
* **Method:** PUT
* **Request:** 
  + 1. Request type:
       1. Query Parameter
    2. Request parameters:
       1. User\_otp
       2. New Password
       3. Confirm Password
* **Response:**

Success:

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "Password updated successfully.",**

**"type": "update"**

**}**

**]**

**}**

**}**

Failure:

**{**

**"response": {**

**"code": 400,**

**"status": "Failure",**

**"alert": [**

**{**

**"message": "Invalid OTP code.",**

**"type": "Failure"**

**}**

**]**

**}**

**}**

### **Get User by id**

* **Description:** To get user details by their user id
* **Workflow:**
  + 1. The API receives a GET request with a user\_id parameter.
    2. If a user is found with the specified user\_id, it returns the user object.
    3. If no user is found, it returns failure message.
* **Endpoint:** /user/{user\_id}
* **Method:** GET
* **Request:** 
  + 1. Request type:
       1. Query Parameter
    2. Request parameters:
       1. Body:
          1. User\_id
* **Response:**
  + 1. Success:

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "User fetched successfully ",**

**"type": "Fetch"**

**}**

**],**

**"data": {**

**"first\_name": "Anirudh",**

**"last\_name": null,**

**"email": "anirudhherady17@gmail.com",**

**"password": "$2b$12$16DN1ROa5uDrxgdgL6A4p.I2VHdKM/yyOKjT86j46JEAO9JKsUWp2",**

**"emp\_id": "5427",**

**"role": "super\_admin",**

**"phone": "6436289393",**

**"created\_at": "2023-05-30T14:17:29.920821",**

**"updated\_at": "2023-05-30T14:19:32.887768",**

**"email\_verified": true,**

**"phone\_verified": true,**

**"is\_active": false**

**},**

**"is\_data": 1**

**}**

**}**

* + 1. Failure:

**{**

**"response": {**

**"code": 404,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "User not found",**

**"type": "failure"**

**}**

**],**

**"is\_data": 0**

**}**

**}**

### **Get All Users**

* **Description:** To get all the user details
* **Workflow:**
  + 1. The API receives a GET request to fetch all users.
    2. If there are users found in the database, it returns the list of user objects.
    3. If there are no users found (an empty list), it returns False.
* **Endpoint:** /users
* **Method:** GET
* **Request:** 
  + 1. Request type:
       1. GET
    2. Request parameters:
       1. Body:
          1. None
* **Response:**

Success:

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "User fetched successfully ",**

**"type": "Fetch"**

**}**

**],**

**"data": [**

**{**

**"id": "9388b81cbf6a46bd9a1950d41d8fd32d",**

**"email": "anirudhherady17@gmail.com",**

**"password": "$2b$12$16DN1ROa5uDrxgdgL6A4p.I2VHdKM/yyOKjT86j46JEAO9JKsUWp2",**

**"role": "super\_admin",**

**"phone\_verified": true,**

**"created\_at": "2023-05-30T14:17:29.920821",**

**"updated\_at": "2023-05-30T14:19:32.887768",**

**"is\_active": false,**

**"first\_name": "Anirudh",**

**"last\_name": null,**

**"emp\_id": "5427",**

**"phone": "6436289393",**

**"email\_verified": true**

**},**

**{**

**"id": "f07576b3fbe142f1b72460f91c8cc774",**

**"email": "i6d10a5yn1gdtkbvuzld725x\_0qvihe2r497uhlitdjsj6frsvuli4b4giuplor6erdd59512w2ner69hswwhg8jrnad06@pi0xD8ECKHq0.com",**

**"password": "$2b$12$XiqdSXoBc6tDDoUliBnjTuL7XCca2EMp2vRrOXyM0y3YFFr8C1kRS",**

**"role": "super\_admin",**

**"phone\_verified": false,**

**"created\_at": "2023-05-31T10:59:27.458680",**

**"updated\_at": null,**

**"is\_active": false,**

**"first\_name": "string",**

**"last\_name": "string",**

**"emp\_id": "string",**

**"phone": "6463732182",**

**"email\_verified": false**

**},**

**{**

**"id": "cd96384996a44f598dccf948c41d0761",**

**"email": "johnny132gmail.com",**

**"password": "$2b$12$K.KC.Kg6ghOzAyf/7TQXUeQBzD5iV1T3RbLXqWrDNd03mHKgcBJT2",**

**"role": "editor",**

**"phone\_verified": true,**

**"created\_at": "2023-05-30T14:18:17.716251",**

**"updated\_at": "2023-05-31T11:43:02.490591",**

**"is\_active": false,**

**"first\_name": "johnny",**

**"last\_name": "string",**

**"emp\_id": "5437",**

**"phone": "8547382312",**

**"email\_verified": true**

**},**

**{**

**"id": "2b679aa2b52041a99269f8c713d80414",**

**"email": "i6d10a5yn1gdtkbvuzld725x\_0qvihe2r497uhlitdjsj6frsvuli4b4giuplor6erd@d52ner69hswwhg8jrnad06pi0xD8ECKHq0.com",**

**"password": "$2b$12$Ovjvp3LnrZAvri.phyKNlOz6jG8jRiTybK7DKJKh0seE4TsSMbj.K",**

**"role": "aDmin",**

**"phone\_verified": false,**

**"created\_at": "2023-05-31T11:50:26.227052",**

**"updated\_at": null,**

**"is\_active": false,**

**"first\_name": "string",**

**"last\_name": "string",**

**"emp\_id": "sting",**

**"phone": "6463932182",**

**"email\_verified": false**

**},**

**{**

**"id": "29a6f16a09b143c58c203c61168dd2a1",**

**"email": "efijkr@d52ner69hswwhg8jrnad06pi0xD8ECKHq0.com",**

**"password": "$2b$12$Q3PzUep3vnG1O/4BDoxCDuSs7hiTUeq2p5pr2UNlwm5EzLE.Han0q",**

**"role": "super\_admin",**

**"phone\_verified": false,**

**"created\_at": "2023-05-31T11:52:32.389078",**

**"updated\_at": null,**

**"is\_active": false,**

**"first\_name": "strng",**

**"last\_name": "strig",**

**"emp\_id": "stig",**

**"phone": "6463922182",**

**"email\_verified": false**

**}**

**],**

**"is\_data": 1**

**}**

**}**

Failure:

**{**

**"response": {**

**"code": 404,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "Users not found",**

**"type": "failure"**

**}**

**],**

**"is\_data": 0**

**}**

**}**

### Delete user

1. **Description:** API to delete user.
2. **Workflow:**
   * + 1. Client passes the user id for the user to be deleted
       2. Click on submit button
       3. Calls delete user API
       4. On success, backend will remove the details of the user from the database.
       5. On error, user id not found will be displayed.
3. **Endpoint:** /user/{user\_id}
4. **Method:** DEL
5. **Request:** 
   * 1. Request type:
        1. Query Parameter
        2. Header(Access Token)
     2. Request parameters:
        1. User ID
6. **Response:**

i.Success

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "user with ID dae12981db8d41609ced3d0cef180f18 deleted successfully.",**

**"type": "delete"**

**}**

**]**

**}**

**}**

ii. Failure:

**{**

**"response": {**

**"code": 404,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "User not found",**

**"type": "failure"**

**}**

**]**

**}**

**}**

### Update user

**a. Description:** APIs to update user details.

**b. Workflow:**

**i. Update single user**

1. User enters the id of the user to be updated and a parameter “bulk” as false so that only the details of one user has to be updated.

2. On click of the submit button, the update user api is called.

3. On success, the backend updates the given role in the database.

i**i. Update multiple users**

1. User enters a list of details of different users including user “id”, “role” & “is\_active” fields and a parameter “bulk” as true so that only multiple user details have to be updated.

2. On click of the submit button, the update user api is called.

3. On success, the backend updates the “role” and “is\_active” status for all the user ids given by the user, in the database.

**c. Endpoint:** /updateuser

**d. Method:** PUT

**e. Request:**

i.Request type:

1. Body (JSON)
2. Header(Access Token)

ii.Request parameters:

1. Body:
   * + - 1. Email
         2. Password (encoded format)

iii.Request JSON:

1. Update multiple users

{

“id”: String,

“role”: String

“is\_active”: Bool,

}

1. Update single user

{

"first\_name": String,

"last\_name": String,

"email": String,

"emp\_id": String,

"role": String

"phone": String  
}

1. **Response:**

**a. Update Single User**

i.Success

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "user id f07576b3fbe142f1b72460f91c8cc774 updated",**

**"type": "Update"**

**}**

**],**

**"data": {**

**"password": "$2b$12$XiqdSXoBc6tDDoUliBnjTuL7XCca2EMp2vRrOXyM0y3YFFr8C1kRS",**

**"email": "heisenberg62@gmail.com",**

**"updated\_at": "2023-05-31T14:54:53.230547",**

**"id": "f07576b3fbe142f1b72460f91c8cc774",**

**"phone\_verified": false,**

**"role": "editor",**

**"created\_at": "2023-05-31T10:59:27.458680",**

**"is\_active": true,**

**"emp\_id": "5426",**

**"first\_name": "Walter",**

**"last\_name": "White",**

**"phone": "6463932182",**

**"email\_verified": false**

**},**

**"is\_data": 1**

**}**

**}**

ii.Failure

**{**

**"response": {**

**"code": 404,**

**"status": "Failure",**

**"alert": [**

**{**

**"message": "User not found",**

**"type": "Failure"**

**}**

**],**

**"is\_data": 0**

**}**

**}**

**{**

**"response": {**

**"code": 404,**

**"status": "Failure",**

**"alert": [**

**{**

**"message": "Invalid role. Your role can be \"super\_admin\", \"admin\", \"editor\", \"author\", \"subscriber\" ",**

**"type": "Failure"**

**}**

**],**

**"is\_data": 0**

**}**

**}**

**b. Update Multiple User**

i.Success

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "multiple user ids updated",**

**"type": "Update"**

**}**

**],**

**"data": [**

**{**

**"id": "f07576b3fbe142f1b72460f91c8cc774",**

**"is\_active": true,**

**"role": "author"**

**},**

**{**

**"id": "cd96384996a44f598dccf948c41d0761",**

**"is\_active": true,**

**"role": "editor"**

**}**

**],**

**"is\_data": 1**

**}**

**}**

ii. Failure

**{**

**"response": {**

**"code": 404,**

**"status": "Failure",**

**"alert": [**

**{**

**"message": "Invalid role. Your role can be \"super\_admin\", \"admin\", \"editor\", \"author\", \"subscriber\" ",**

**"type": "Failure"**

**}**

**],**

**"is\_data": 1**

**}**

**}**

**{**

**"response": {**

**"code": 404,**

**"status": "Failure",**

**"alert": [**

**{**

**"message": "User not found",**

**"type": "Failure"**

**}**

**],**

**"is\_data": 1**

**}**

**}**

### Add role

**a. Description:** APIs to add a user role.

**b. Workflow:**

1. User fills a form where role name, permissions and status values are submitted.

2. On click of the submit button, the add role api is called.

3. On success, the backend will send the given role data by the user to the database.

**c. Endpoint:** /roles

**d. Method:** POST

**e. Request:**

1. Request type:

a. Body (JSON)

b.Header(Access Token)

1. Request parameters:

Body:

* + - * 1. Role Name
        2. Permissions
        3. Status

1. Request JSON:

{

“role\_name”: String,

“permissions”: List[String],

“status”: Bool

}



1. **Response:**
2. Success

**{**

**"response": {**

**"code": 201,**

**"status": "success",**

**"alert": [**

**{**

**"message": "New role added successfully",**

**"type": "created"**

**}**

**]**

**}**

**}**

ii. Failure:

**{**

**"response": {**

**"code": 501,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "Role already exists | internal server error",**

**"type": "failure"**

**}**

**]**

**}**

**}**

### Delete role

**a. Description:** APIs to delete a user role.

**b. Workflow:**

1. User enters the id of the user role to be deleted.

2. On click of the submit button, the deleted role api is called.

3. On success, the backend removes the given role from the database.

**c. Endpoint:** /roles

**d. Method:** POST

**e. Request:**

1. Request type:
   1. Parameter
   2. Header(Access Token)
2. Request parameters:
3. Role ID

1. **Response:**
2. Success

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "Role deleted successfully",**

**"type": "deleted"**

**}**

**]**

**}**

**}**

ii. Failure:

**{**

**"response": {**

**"code": 404,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "Role not found",**

**"type": "failure"**

**}**

**]**

**}**

**}**

### Get roles

**a. Description:** APIs to delete a user role.

**b. Workflow:**

1. User requests to get all roles.

2. On click of the submit button, the get roles api is called.

3. On success, the backend sends all roles from the database.

**c. Endpoint:** /roles

**d. Method:** GET

**e. Request:**

1. Request type:
   1. Header(Access Token)

1. **Response:**
2. Success

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "Roles fetched successfully ",**

**"type": "Fetch"**

**}**

**],**

**"users": [**

**{**

**"role\_name": "subscriber",**

**"id": "8e3a804e489c47faa750d7a44c9e0901",**

**"status": true,**

**"permissions": [**

**"add\_user",**

**"update\_user",**

**"delete\_user"**

**]**

**},**

**{**

**"role\_name": "editor",**

**"id": "a39fea8c43fe4cb081204aeaaa86b9d0",**

**"status": true,**

**"permissions": [**

**"update\_user",**

**"delete\_user",**

**"update\_role",**

**"delete\_role"**

**]**

**},**

**{**

**"role\_name": "admin",**

**"id": "74dd660b7a6a4572ac26e9de4234aa5e",**

**"status": true,**

**"permissions": [**

**"add\_user",**

**"delete\_user",**

**"update\_user",**

**"get\_user",**

**"get\_role",**

**"delete\_role",**

**"update\_role"**

**]**

**},**

**{**

**"role\_name": "super\_admin",**

**"id": "607064cc8a284561892c18b755c7c825",**

**"status": false,**

**"permissions": [**

**"add\_user",**

**"delete\_user",**

**"verify\_user",**

**"update\_user",**

**"get\_user",**

**"add\_role",**

**"get\_role",**

**"delete\_role",**

**"update\_role"**

**]**

**},**

**{**

**"role\_name": "author",**

**"id": "89ccd0ff27fb470fa0e82af04238d683",**

**"status": true,**

**"permissions": [**

**"add\_user",**

**"update\_user",**

**"get\_user"**

**]**

**}**

**],**

**"is\_data": 1**

**}**

**}**

ii.Failure

**{**

**"response": {**

**"code": 404,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "No roles found ",**

**"type": "failure"**

**}],**

**“is\_data”:0**

**}**

**}**

### Update role

**a. Description:** APIs to delete a user role.

**b. Workflow:**

1. User enters the id of the user role to be updated.

2. On click of the submit button, the update roles api is called.

3. On success, the backend updates the given role in the database.

**c. Endpoint:** /roles

**d. Method:** GET

**e. Request:**

1. Request type:
   1. Parameter
   2. Header(Access Token)
2. Request parameters:
3. Role ID
4. **Response:**
5. Success

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "Role with role id 607064cc8a284561892c18b755c7c825 updated successfully",**

**"type": "Update"**

**}**

**],**

**"data": {**

**"role\_name": "super\_admin",**

**"permissions": [**

**"add\_user",**

**"delete\_user",**

**"verify\_user",**

**"update\_user",**

**"get\_user",**

**"add\_role",**

**"get\_role",**

**"delete\_role",**

**"update\_role"**

**],**

**"status": true**

**},**

**"is\_data": 1**

**}**

**}**

ii.Failure

**{**

**"response": {**

**"code": 404,**

**"status": "Failure",**

**"alert": [**

**{**

**"message": "Role id not found.",**

**"type": "Failure"**

**}**

**],**

**"is\_data": 0**

**}**

**}**

### **Permission - POST**

* **Description:** To add permission in backend
* **Workflow:**
  + 1. The API receives a POST request with the permission data
    2. It calls the permission\_add function and passes the permission data and the database session as arguments.
    3. If a user permission with the same permission name is found, it returns False indicating that the permission already exists.
    4. It generates a unique ID for the user permission
* **Endpoint:** /permission
* **Method:** POST
* **Request:** 
  + 1. Request type:
       1. Body JSON
       2. Header(Access Token)
    2. Request parameters:
       1. body
          1. permission\_name
          2. permission\_type
          3. collection

list of permission

* + - * 1. status
    1. Request JSON:

{

"permission\_name": "string",

"permission\_type": "string",

"collection": ["string\_1","string\_2"],

"status": true

}

* **Response:**

Success:

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "new user permissions added.",**

**"type": "Created"**

**}**

**]**

**}**

**}**

### **Permission - GET**

* **Description:** To get all the permissions
* **Workflow:**
  + 1. The API receives a GET request to retrieve all user permissions.
    2. The API returns the list of user permissions retrieved from the database.
* **Endpoint:** /permission
* **Method:** GET
* **Request:** 
  + 1. Request type:
       1. Header(Access Token)
    2. Request parameters:
       - 1. NONE
* **Response:**

Success:

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "Permission fetched successfully ",**

**"type": "success"**

**}**

**],**

**"data": [**

**{**

**"id": "307a3386c7574d31a09f8fa8f503522a",**

**"permission\_name": "view\_user",**

**"permission\_type": "view",**

**"collection": [**

**"\_view"**

**],**

**"status": true**

**}**

**],**

**"is\_data": 1**

**}**

**}**

Failure:

### **Permission - UPDATE**

* **Description:** To update permission in backend
* **Workflow:**
  + 1. The API receives a PUT request to update a user permission based on the permission id
    2. If a user permission with the specified perm\_id exists, the function updates its attributes with the values from perm\_data
    3. The function adds the updated user permission to the database
* **Endpoint:** /permission
* **Method:** POST
* **Request:** 
  + 1. Request type:
       1. Query parameter
       2. Header(Access Token)
    2. Request parameters:
       - 1. Permission id

**Response:**

Success:

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "permission id 307a3386c7574d31a09f8fa8f503522a updated successfully",**

**"type": "Update"**

**}**

**],**

**"data": {**

**"permission\_name": "add",**

**"permission\_type": "add",**

**"collection": [**

**"\_add"**

**],**

**"status": true**

**},**

**"is\_data": 1**

**}**

**}**

Failure:

**{**

**"response": {**

**"code": 404,**

**"status": "Failure",**

**"alert": [**

**{**

**"message": "Permission id not found",**

**"type": "Failure"**

**}**

**],**

**"is\_data": 0**

**}**

**}**

### **Permission - Delete**

* **Description:** To delete permission in backend
* **Workflow:**
  + 1. The API receives a DELETE request to delete a user permission.
    2. If a user permission with the specified perm\_id exists, the function deletes it from the database
    3. if the user permission was successfully deleted, the function returns Success message.
    4. Otherwise, it returns Failure message.
* **Endpoint:** /permission
* **Method:** DELETE
* **Request:** 
  + 1. Request type:
       1. Query parameter
       2. Header(Access Token)
    2. Request parameters:
       - 1. Permission id

**Response:**

Success:

**{**

**"response": {**

**"code": 200,**

**"status": "success",**

**"alert": [**

**{**

**"message": "Permission deleted successfully",**

**"type": "deleted"**

**}**

**]**

**}**

**}**

Failure:

**{**

**"response": {**

**"code": 404,**

**"status": "failure",**

**"alert": [**

**{**

**"message": "Permission not found",**

**"type": "failure"**

**}**

**]**

**}**

**}**

**{**

**"response": {**

**"code": 404,**

**"status": "Failure",**

**"alert": [**

**{**

**"message": "Enter correct OTP.",**

**"type": "Failure"**

**}**

**]**

**}**

**}**