Authentication

Authentication :

Authentication is the process of verifying the identity of a user, device, or system before allowing access to an application or service.

***Login**
***Logout**

Login :

For All User, its generic login APIs (Admin, Sub-User)

For Admin :
POST - /api/v1/login

Request :


```
{
  "email": "shini90@gmail.com",
  "password": "Shini@123"
}
```


Response :


```
{
    "status": 200,
    "message": "Login successful",
    "data": {
        "id": 1,
        "email": "shini90@gmail.com",
        "isActive": true,
        "schemaName": "tenant_1_shini90_gmail_com",
        "accessExpiration": 1768297178234,
        "refreshExpiration": 1773394778234,
        "roles": [
            "ROLE_ADMIN"
        ],
        "token":
"eyJhbGciOiJIUzI1NiJ9.eyJpc19hY3RpdmUiOnRydWUsInNjaGVtYV9uYW1lIjoidGVuYW50XzFfc2hpbmk5MF9nbWFpbF9jb20iLCJyb2xlIjpbIlJPTEVfQURNSU4iXSwidXNlcl9pZCI6MSwidXNlcl9lbWFpbCI6InNoaW5pOTBAZ21haWwuY29tIiwiaWF0IjoxNzY4MjkzNTc4LCJleHAiOjE3NjgyOTcxNzh9.sbDt4AnQFGhXMiLhOyvcjBhNM4W7Jf1FzSx6OvVWDHg",
```

```
        "refreshToken":
"=eyJhbGciOiJIUzI1NiJ9.eyJpc19hY3RpdmUiOnRydWUsInNjaGVtYV9uYW1lIjoidGVuYW50XzFfc2hpbmk
5MF9nbWFpbF9jb20iLCJyb2xlIjpbIlJPTEVfQURNSU4iXSwidXNlcl9pZCI6MSwidXNlcl9lbWFpbCI6InNoa
W5pOTBAZ21haWwuY29tIiwiaWF0IjoxNzcyMDk4Nzc4LCJleHAiOjE3NzMzOTQ3Nzh9.uirJv_CBRDqAJ6plvb
iSXsHHsnqXmJ8aTi_gsBZNAgU",
        "branchId": null,
        "modules": null
    }
}
```


For User :

We have to pass one schema ID for accessing a particular company, Schema ID is unique for every company.

Request :


```
{
    "schema": "tenant_1_shini90_gmail_com",
    "email": "aarugaming2005gmail.com",
    "password": "Aaru@123"
}
```


Response :


```
{
    "status": 200,
    "message": "Login successful",
    "data": {
        "id": 1,
        "email": "aarugaming2005gmail.com",
        "isActive": true,
        "schemaName": "tenant_1_shini90_gmail_com",
        "accessExpiration": 1768297306827,
        "refreshExpiration": 1773394906827,
        "roles": [
            "ROLE_BRANCHADMIN",
            "ROLE_EMPLOYEE"
        ],
        "token":
"eyJhbGciOiJIUzI1NiJ9.eyJpc19hY3RpdmUiOnRydWUsInNjaGVtYV9uYW1lIjoidGVuYW50XzFfc2hpbmk5
MF9nbWFpbF9jb20iLCJyb2xlIjpbIlJPTEVfQlJBTkNIQURNSU4iLCJST0xFX0VNUExPWUVFIl0sInVzZXJfaW
QiOjEsInVzZXJfZW1haWwiOiJhYXJ1Z2FtaW5nMjAwNWdtYWlsLmNvbSIsImlhdCI6MTc2ODI5MzcwNiwiZXhw
IjoxNzY4Mjk3MzA2fQ.22oQXxw8Rn9zk4tDgYbsQpyphQ-CVnn3Pn8j3BENwbk",
```

```
        "refreshToken":
"=eyJhbGciOiJIUzI1NiJ9.eyJpc19hY3RpdmUiOnRydWUsInNjaGVtYV9uYW1lIjoidGVuYW50XzFfc2hpbmk
5MF9nbWFpbF9jb20iLCJyb2xlIjpbIlJPTEVfQlJBTkNIQURNSU4iLCJST0xFX0VNUExPWUVFIl0sInVzZXJfa
WQiOjEsInVzZXJfZW1haWwiOiJhYXJ1Z2FtaW5nMjAwNWdtYWlsLmNvbSIsImlhdCI6MTc3MjA5ODkwNiwiZXh
wIjoxNzczMzk0OTA2fQ.kiEcmwgdVCHuc4UFrSW5q8WRdwbKRu6F4uBdQiUWe4k",
        "branchId": 1,
        "modules": {
            "User Management": 1,
            "Branch Management": 2,
            "Product": 3,
            "Tax": 4,
            "Services Management": 5,
            "Leads": 6,
            "Follow Ups": 7,
            "Customers": 8,
            "Quotations": 9,
            "Sales Orders": 10,
            "AMC": 11,
            "Invoices": 12,
            "Payments": 13,
            "Receipts": 14,
            "Task": 15,
            "Live Tracking": 16,
            "Customer Support": 17,
            "Performance Management": 18,
            "Check In Check Out": 19,
            "Attendance": 20,
            "Salary": 21,
            "Leave": 22,
            "Profile": 23
        }
    }
}
```

Logout :

For all users common logout apis (admin, sub-user).

POST - /api/v1/root/logout

Request not required.

Response :

```
{
    "status": 200,
    "message": "Logout successful",
    "data": null
}
```

Roll Wise:-

1. **Company Admin :-**

   Admin authentication is the process of validating an admin user and creating a secure session for accessing admin features in the application. The admin logs in using their registered email and password. Once the credentials are verified, the system confirms that the account is active and assigns the admin role.

   After successful login, the frontend receives:

   - **Admin identity details** (email, ID, active status)
   - **Role information** to control admin-only screens and actions
   - **An access token** used to authorize future requests
   - **A refresh token** to maintain the session when the access token expires

- **Expiration times** to manage automatic logout or token renewal

The frontend should store the tokens securely and attach the access token when interacting with protected parts of the system. Based on the role and active status, the UI can enable or restrict admin-specific functionalities. This flow ensures secure access, session management, and role-based control without requiring repeated logins.

2. Branch Admin , Technician Manager , Sales Manager , Account Manager:-

User authentication is the process of validating a user and creating a secure session within a specific company. Since the system supports multiple companies, the user must provide a **schema ID**, which uniquely identifies the company they belong to, along with their registered email and password. This ensures the user is authenticated only under the selected company's environment.

Once the credentials and schema are verified, the system confirms that the user account is active and assigns the appropriate roles (such as Branch Admin or Employee).

After successful login, the frontend receives:

- User identity details (email, ID, active status)
- Company schema information to maintain company-specific access
- Role details to control feature-level access
- Branch ID for branch-specific operations
- Module permissions to enable or disable application features
- An access token used to authorize future requests
- A refresh token to maintain the session when the access token expires

- Expiration times to manage automatic logout or token renewal

The frontend should securely store the tokens and attach the access token when accessing protected parts of the application. Based on the assigned roles, branch, and module permissions, the UI should dynamically enable or restrict features. This flow ensures secure, company-specific access, proper role-based control, and smooth session management for users.

Admin

Company Admin

Postman : ADMIN

*Create
Update

Admin Creation :
POST - /api/v1/admins

-> From the landing page Admin can register with email, password and other information.
-> It is public anyone can register as a ADMIN

Get All Details of Current Admin For Profile :
GET - /api/v1/getDetails/admin
-> Get all information about the current login admin for profile or dashboard.
-> ADMIN only can access this API

Request not required.

Response :


```
{
    "status": 200,
    "message": "Admin Fetched Successfully!!",
    "data": {
        "id": 1,
        "name": "Palak Dave",  -  Set this on profile Name.
        "email": "aryanraval299@gmail.com",  - Store this, it can be useful.
        "contactNo": 1234567890,
        "createdAt": "2025-12-04T16:49:13.552376",
        "updatedAt": "2025-12-26T14:56:26.691115",
        "createdByRootUserId": 1,
        "lastUpdatedByRootUserId": 1,
        "documentUrl":
"https://seravion3960.s3.amazonaws.com/public/admin/profile/1cc58a3e-d3d2-402b-a292-86bf0ef5e322_WhatsAppImage2024-12-05at16.46.35_76d6a594.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T100111Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=2828ce4e176db2d9ae5a8d300514f292f2873f1b42ace6e11954f929308400e4",  - Set this on
Profile Image
```

```
        "isActive": true
    }
}
```

Update Admin Details :

PUT - /api/v1/update/admin

-> Admin only can update his profile, Update admin details only required field. Like Name, Contact Number, Image File.

Request :

```
{
    "name":"Palak Dave",
    "contactNo":1234567890,
    "files":[
        {
        "fileName":"WhatsApp Image 2024-12-05 at 16.46.35_76d6a594.jpg",
        "fileType":"image/jpeg",
        "fileData":"base64 String of file image”
       }
}
```

Company Admin :-

The Company Admin holds the highest level of access within the system and is responsible for overseeing and managing all company-level operations across multiple branches. This role focuses on strategic control, monitoring, and performance analysis rather than day-to-day field execution.

The Company Admin has full visibility into all dashboards and modules, enabling centralized decision-making and effective supervision of business activities across the organization.

Key Responsibilities & Access

- Full control over all company dashboards and system-wide data
- Ability to create and manage Branch Admins
- Complete control over branches, including:
    - Creating new branches
    - Editing branch details
    - Deleting existing branches

Service & CRM Oversight

- View access to all services offered by the company
- Full visibility into CRM modules, including:
    - Leads
    - Customers

Sales, Billing & Finance Monitoring

- Access to Sales modules, such as:
    - Quotations
    - Sales Orders
- View-only access to Billing & Finance, including:
    - Invoices
    - Payments
    - Receipts

Operations & Performance Tracking

- View all tasks across branches
- Monitor live tracking of operations
- Access customer support activities
- Analyze performance reports across all branches to evaluate productivity and efficiency

Access Limitation

- The Company Admin cannot add or manage technicians, ensuring a clear separation between administrative control and operational execution.

User Management

User Management

Postman : Enhanced ERP

User Creation :

ADMIN create all other users but recommend only BRANCHADMIN
BRANCHADMIN create all other user but recommend only OTHER Manager
TECHNICIANMANAGER create only TECHNICIAN, EMPLOYEE
SALESADMIN, HR, ACCOUNTMANAGER create only EMPLOYEE

For TECHNICIAN set Designation TECHNICIAN, for EMPLOYEE set EMPLOYEE, like that all Designation set via ROLE

This API is used to create a new user in the system with specific roles and organizational mapping. It supports the creation of different user types such as **Branch Admin, Manager, Account Manager, Technician Manager, Sales Admin,** and **Employee** based on the provided flags, designation, and roles.

The API assigns the user to a branch, maps reporting hierarchy, and generates a tenant schema automatically upon successful creation.

Access to this endpoint is typically restricted to authorized admin or management users.

The `reporterId` field is used to define the reporting relationship between users in the organization. It specifies the immediate superior (manager) to whom the newly created user reports.

POST - /api/v1/users

Request :

```
{
  "firstName": "Aryan",
  "lastName": "Raval",
  "email": "aarugaming2005gmail.com",
  "phoneNo": 7874591247,
```

```
  "password": "Aaru@123",
  "branchAdmin":true,  // for Branch Admin Creation
  "manager":false,         // for HR, ACCOUNTMANAGER, TECHNICIANMANAGER, SALESADMIN
Creation
  "branchId": 1,
  "reporterId":2,
  "designation" :"MANAGER",
  "roles": [
    {
      "roleName": "BRANCHADMIN"
    }
  ]
}
```

Response :


```
{
    "status": 201,
    "message": "User created successfully !!",
    "data": {
        "id": 9,
        "firstName": "Aryan",
        "lastName": "Raval",
        "email": "aarugaming2005gmail.com",
        "phoneNo": 7874591247,
        "schemaName": "tenant_1_aryanraval299_gmail_com",
        "createdAt": "2026-01-12",
        "lastModifiedAt": "2026-01-12",
        "roleNames": [
            "EMPLOYEE",
            "BRANCHADMIN"
        ],
        "branchId": 1,
        "reporterId": 2,
        "branchName": "GreenTechTech ServiceService Point (Koramangala)",
        "moduleName": null,
        "reportingTo": null,
        "fullName": "Aryan Raval",
        "profileUrl": null,
        "active": true
    }
}
```

Update User Profile :

Description :

This API is used to update the profile details of a sub-user (such as Employee, Branch Admin, or other assigned roles). It allows updating only the required fields including **first name, last name, phone number, and profile image**. Any fields not provided in the request will remain unchanged.

This endpoint requires authentication and updates the profile of the currently logged-in user.

Request

- Method: PUT
- Authentication: Required
- Request Body: Required (Partial update supported)

For all sub-user profile update

PUT - /api/v1/user/profile/update

Request :

```
{
    "firstName":"XYZ",
    "lastName":"ABC",
    "phoneNo":1234567890,
    "files":[
        {
```

```
            "fileName":"WhatsApp Image 2024-12-05 at 16.46.35_76d6a594.jpg",
            "fileType":"image/jpeg",
            "fileData":"base64 string”
        }
}
```


Response :


```
{
    "status": 200,
    "message": "User Updated Successfully!!",
    "data": {
        "id": 1,
        "firstName": "XYZ",
        "lastName": "ABC",
        "email": "aarugaming2005gmail.com",
        "phoneNo": 1234567890,
        "schemaName": "tenant_1_shini90_gmail_com",
        "createdAt": "2026-01-09",
        "lastModifiedAt": "2026-01-09",
        "roleNames": [
            "EMPLOYEE",
            "BRANCHADMIN"
        ],
        "branchId": 1,
        "reporterId": 2,
        "branchName": "Shrini West Bengaluru",
        "moduleName": null,
        "reportingTo": null,
        "fullName": "XYZ ABC",
        "profileUrl":
"https://seravion3960.s3.amazonaws.com/tenant_1_shini90_gmail_com/user/profile/fe161bcd-3bed-471e-9c64-a1277d165bfe_WhatsAppImage2024-12-05at16.46.35_76d6a594.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T102232Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=e42d6fccd52937660db5261d2657564bfb6f7df383871ad4b71312132126b6cc",
        "active": true
    }
}
```

Get Current User Details :

Description :

This API is used to **fetch the profile details of the currently logged-in user** based on the authenticated email. It returns complete user information including personal details, assigned roles, branch mapping, reporting hierarchy, and profile image.

No request body is required. The API identifies the user using the authentication token.

Request

- Method: GET
- Authentication: Required
- Request Body: Not required

GET - /api/v1/users/byEmail

Request not required.

Response :


```
{
    "status": 200,
    "message": "User Fetched Successfully!!",
    "data": {
        "id": 1,
        "firstName": "XYZ",
```

```
        "lastName": "ABC",
        "email": "aarugaming2005gmail.com",
        "phoneNo": 1234567890,
        "schemaName": "tenant_1_shini90_gmail_com",
        "createdAt": "2026-01-09",
        "lastModifiedAt": "2026-01-12",
        "roleNames": [
            "EMPLOYEE",
            "BRANCHADMIN"
        ],
        "branchId": 1,
        "reporterId": 2,
        "branchName": "Shrini West Bengaluru",
        "moduleName": null,
        "reportingTo": null,
        "fullName": "XYZ ABC",
        "profileUrl":
"https://seravion3960.s3.amazonaws.com/tenant_1_shini90_gmail_com/user/profile/fe161bcd-3bed-471e-9c64-a1277d165bfe_WhatsAppImage2024-12-05at16.46.35_76d6a594.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T102341Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=ca9c6b8de9d4e000a0be6b860ae8a5695a3106c30388a4045fa8aa418ef6f6c1",
        "active": true
    }
}
```

Get All Users For Admin : ACCESS ONLY ADMIN

Description :

This API is used to **fetch the complete list of users in the system**. It is accessible only to admin users and returns detailed information for each user, including personal details, assigned roles, branch mapping, reporting hierarchy, and account status.

No request parameters or request body are required.

Request

- Method: GET
- Authentication: Required
- Authorization: Admin Access Only
- Request Body: Not required

GET - /api/v1/users

Request Not Required

Response :


```
{
    "status": 200,
    "message": "List of users !!",
```

```
    "data": [
        {
            "id": 1,
            "firstName": "Raj",
            "lastName": "Patel",
            "email": "raj123@gmail.com",
            "phoneNo": 7874591247,
            "schemaName": "tenant_2_aryanraval299_gmail_com",
            "createdAt": "2026-01-12",
            "lastModifiedAt": "2026-01-12",
            "roleNames": [
                "EMPLOYEE",
                "BRANCHADMIN"
            ],
            "branchId": 1,
            "reporterId": 1,
            "branchName": "East Zone Service Center",
            "moduleName": null,
            "reportingTo": null,
            "fullName": "Raj Patel",
            "profileUrl": null,
            "active": true
        }
    ]
}
```

Get All Users Branch Wise For Manages : ACCESS ONLY ALL SUB USER

Description :

This API is used to **fetch the list of users belonging to the same branch as the currently logged-in user**. It is accessible to **managers and authorized sub-users** and returns branch-specific user details along with a total user count.

This endpoint helps managers view and manage users only within their assigned branch.

Request

- Method: GET
- Authentication: Required
- Authorization: Manager / Authorized Sub-User Access
- Request Body: Not required

GET - /api/v1/user/branchWise

Request Not Required.

Response :


```
{
    "status": 200,
    "message": "Fetched Users Successfully",
```

```
    "data": {
        "count": 32,
        "results": [
            {
                "id": 2,
                "firstName": "Deep",
                "lastName": "Ghevariya",
                "email": "deepghevariya900@gmail.com",
                "phoneNo": 7874591247,
                "schemaName": "tenant_1_shini90_gmail_com",
                "createdAt": "2026-01-09",
                "lastModifiedAt": "2026-01-09",
                "roleNames": [
                    "TECHNICIANMANAGER",
                    "EMPLOYEE"
                ],
                "branchId": 1,
                "reporterId": 1,
                "branchName": "Shrini West Bengaluru",
                "moduleName": null,
                "reportingTo": null,
                "fullName": "Deep Ghevariya",
                "profileUrl": null,
                "active": true
            }
        ]
}
```

Get Technician By userId :

Description :

This API is used to **fetch the details of a specific technician** based on the provided `userId`. It returns complete profile information including personal details, assigned roles, branch mapping, reporting hierarchy, and account status.

This endpoint is accessible to authorized users such as admins and managers.

Request

- Method: GET
- Authentication: Required
- Authorization: Admin / Manager / Authorized User
- Request Parameter: Required

GET - /api/v1/user/byId?userId=13

Request Parameter is required.
Response :

```
{
```

```
    "status": 200,
    "message": "Technician Find !!",
    "data": {
        "id": 13,
        "firstName": "Ravi",
        "lastName": "Joshi",
        "email": "ravi.joshi05@gmail.com",
        "phoneNo": 8899776655,
        "schemaName": "tenant_1_shini90_gmail_com",
        "createdAt": "2026-01-09",
        "lastModifiedAt": "2026-01-09",
        "roleNames": [
            "TECHNICIAN",
            "EMPLOYEE"
        ],
        "branchId": 1,
        "reporterId": 2,
        "branchName": "Shrini West Bengaluru",
        "moduleName": null,
        "reportingTo": "Deep Ghevariya",
        "fullName": "Ravi Joshi",
        "profileUrl": null,
        "active": true
    }
}
```

User Filter Wise Designation :

Description :

This API is used to **fetch users based on filter criteria**, such as **designation**. It allows admins and authorized users to retrieve a filtered list of users that match specific conditions, making it easier to search and manage users by role or designation (e.g., Technician, Manager).

The API returns the total count of matched users along with their detailed profile information.

Request

- Method: POST
- Authentication: Required
- Authorization: Admin / Manager / Authorized User
- Request Body: Required

POST - /api/v1/user/filter

Request :


```
{
    "filterColumns":{
        "designation":"TECHNICIAN"
    }
}
```


Response :


```
{
    "status": 200,
    "message": "Filter Wise Fetched Users Successfully",
    "data": {
        "count": 23,
        "results": [
            {
                "id": 7,
                "firstName": "Meet",
                "lastName": "Gajera",
                "email": "meetgajera21@gmail.com",
                "phoneNo": 7874591247,
                "schemaName": "tenant_1_shini90_gmail_com",
                "createdAt": "2026-01-09",
                "lastModifiedAt": "2026-01-09",
                "roleNames": [
                    "TECHNICIAN",
                    "EMPLOYEE"
                ],
                "branchId": 1,
                "reporterId": 2,
                "branchName": "Shrini West Bengaluru",
                "moduleName": null,
                "reportingTo": "Deep Ghevariya",
                "fullName": "Meet Gajera",
                "profileUrl": null,
                "active": true
            }
        ]
}
```

Get Current User Assigned Modules :

It takes the current user implicitly.

Description :

This API is used to **fetch the list of modules assigned to the currently logged-in user** based on their roles and permissions. It returns all system modules that the user has access to, along with their corresponding module identifiers.

No request parameters or request body are required. The user is identified using the authentication token.

Request

- Method: GET
- Authentication: Required
- Authorization: Role-Based Access
- Request Body: Not required

GET - /api/v1/user/role/module

Request not Required.

Response :


```
{
    "status": 200,
    "message": "User Role Module Fetched !!",
    "data": {
        "User Management": 1,
        "Branch Management": 2,
        "Product": 3,
        "Tax": 4,
        "Services Management": 5,
        "Leads": 6,
        "Follow Ups": 7,
        "Customers": 8,
        "Quotations": 9,
        "Sales Orders": 10,
        "AMC": 11,
        "Invoices": 12,
        "Payments": 13,
        "Receipts": 14,
        "Task": 15,
        "Live Tracking": 16,
        "Customer Support": 17,
        "Performance Management": 18,
        "Check In Check Out": 19,
        "Attendance": 20,
        "Salary": 21,
        "Leave": 22,
        "Profile": 23
    }
}
```

Managers Drop Down :

It gives only manager details, except subuser technician and employee.

Description :

This API is used to **fetch a list of managers for a specific branch**, formatted for dropdown selection in the UI. It returns minimal manager details (ID and display name) to support features such as **assigning reporters, managers, or supervisors**.

The managers are filtered based on the provided **branch ID**.

Request

- **Method: GET**
- **Authentication: Required**
- **Authorization: Authorized Users**
- **Request Parameter: Required**

GET - /api/v1/user/dropdown?id=1

Request parameter required , it takes branchId.

Response :


```
{
    "status": 200,
    "message": "Manager Drop Down List",
    "data": {
        "count": 6,
        "results": [
            {
                "id": 2,
                "value": "Deep Ghevariya"
            },
            {
                "id": 3,
                "value": "Rashmin Patel"
            },
            {
                "id": 4,
                "value": "Pal Gabani"
            },
            {
                "id": 5,
                "value": "Vishal Kholakiya"
            },
            {
                "id": 31,
                "value": "testing2 full"
            },
            {
                "id": 32,
                "value": "testing3 full"
            }
        ]
    }
}
```

Technician Drop Down Branch Wise :

It gives drop down for branch wise of technician only.

Description :

This API is used to **fetch a list of technicians belonging to the same branch as the currently logged-in user**, formatted specifically for dropdown selection in the UI. It returns minimal technician details (ID and display name) to support assignment workflows such as **task allocation, service assignment, and job scheduling**.

No request parameters are required. Branch filtering is applied automatically based on the authenticated user.

Request

- Method: GET
- Authentication: Required
- Authorization: Authorized Users
- Request Body: Not required

GET - /api/v1/user/technician/dropdown

Request not required.

Response :


```
{
    "status": 200,
    "message": "Technician Drop Down !!",
    "data": {
        "count": 23,
        "results": [
            {
                "id": 7,
                "value": "Meet Gajera"
            },
            {
                "id": 8,
                "value": "Prince Gajera"
            },
            {
                "id": 9,
                "value": "Amit Sharma"
            }
       ]
}
```

Soft delete User :

Description :

This API is used to **soft delete (deactivate) a user account** in the system. Instead of permanently removing the user's data, the API marks the user as inactive, ensuring historical records, reporting data, and references remain intact.

This operation is typically performed by **admin or authorized management users**.

Request

- Method: DELETE
- Authentication: Required
- Authorization: Admin / Authorized User
- Request Body: Required

DELETE – /api/v1/users

Request :


```
{
    "id": "2"
}
```


Response :


```
{
    "status": 200,
    "message": "User delete Successfully !!",
    "data": {
        "id": 1,
        "firstName": "Raj",
        "lastName": "Patel",
        "email": "raj123@gmail.com",
        "phoneNo": 7874591247,
        "schemaName": "tenant_2_aryanraval299_gmail_com",
        "createdAt": "2026-01-12",
        "lastModifiedAt": "2026-01-12",
        "roleNames": [
            "BRANCHADMIN",
            "EMPLOYEE"
        ],
        "branchId": 1,
        "reporterId": 1,
        "branchName": "East Zone Service Center",
        "moduleName": null,
        "reportingTo": null,
        "fullName": "Raj Patel",
        "profileUrl": null,
        "active": false
    }
}
```

Hard Delete User ：

Description :

This API is used to **permanently delete a user from the system**. Unlike soft delete, this operation **completely removes the user and all associated references** from the database. Once deleted, the user **cannot be restored**.

This endpoint is highly restricted and should be accessible **only to admin or super-admin users**.

Request

- Method: DELETE
- Authentication: Required
- Authorization: Admin / Super Admin Only
- Request Parameter: Required

DELETE - /api/v1/user/hard-delete?userId=2

Request Paramater required as a UserId.

Response :


```
{
    "status": 200,
    "message": "User delete Successfully !!",
    "data": {
        "id": 9,
        "firstName": "Aryan",
        "lastName": "Raval",
        "email": "aarugaming2005gmail.com",
        "phoneNo": 7874591247,
        "schemaName": "tenant_1_aryanraval299_gmail_com",
        "createdAt": "2026-01-12",
        "lastModifiedAt": "2026-01-12",
        "roleNames": [
            "BRANCHADMIN",
            "EMPLOYEE"
        ],
        "branchId": 1,
        "reporterId": 2,
        "branchName": "GreenTechTech ServiceService Point (Koramangala)",
        "moduleName": null,
        "reportingTo": null,
        "fullName": "Aryan Raval",
        "profileUrl": null,
        "active": true
    }
}
```

Description :-

The **User Management** feature is fully **role-based**.
Each logged-in user can create **only specific types of users**, depending on their role.
The frontend must **show or hide UI elements** (buttons, forms, dropdown options) strictly based on the logged-in user's role.

Common Permissions (Applies to All Above Roles)

- All roles can:
    - Create user records (within allowed role types)
    - Update user details
    - Delete user details
- These actions should be available in the UI **only for users they are allowed to manage**

User Creation Rules (Very Important for UI Logic)

1. Company Admin

- Can create **only Branch Admin**
- In the "Create User" screen:
    - Role dropdown should show **only "Branch Admin"**
- Cannot create:

    - Managers
    - Technicians
    - Employees

---

2. Branch Admin

- Can create **Managers only**
- Allowed manager roles:
    - Technician Manager
    - Sales Manager
    - Account Manager
- In the “Create User” screen:
    - Role dropdown should show **only manager roles**
- Cannot create:
    - Technicians
    - Employees

---

3. Technician Manager

- Can create **Technicians only**
- In the “Create User” screen:
    - Role dropdown should show **only “Technician”**
- Cannot create any other role

---

4. Sales Manager

- Can create **Sales Employees only**
- In the “Create User” screen:
    - Role dropdown should show **only “Sales Employee”**
- Cannot create technicians or managers

---

5. Account Manager

- Can create **Account Employees only**
- In the “Create User” screen:
    - Role dropdown should show **only “Account Employee”**
- Cannot create technicians or managers

User Meta Details

User Meta Details

---

1. Module Overview

1.1 Purpose of the Module

The User Meta Details module serves as the core configuration layer for employee-specific professional and financial parameters. It extends the basic `User` entity by storing critical HR and payroll data.

1.2 Real-world Business Problem It Solves

- **Payroll Standardization**
  Defines base salaries and working hour expectations for different employment types.

- **Leave Management**
  Sets entitlements (Casual, Medical, Paid leaves) and tracks usage per employee.

- **Employment Governance**
  Formalizes employment types (Full-time, Contract, etc.) and branch-wise staffing rules.

1.3 How This Module is Used in the System

This module is **mandatory** after the initial user creation. While the `User` table stores credentials and basic identity, `UserMetaDetails` is required to enable attendance tracking, salary calculation, and leave request workflows.

---

2. Functional Responsibilities

- **Employment Profiling**
  Records the `employType` (e.g., Permanent, Probation).

- **Attendance Configuration**
  Defines `monthlyWorkingDays`, `perDayWorkingHours`, and `halfDayWorkingHours` for accurate attendance verification.

- **Salary Benchmarking**
  Maintains the `monthlyBaseSalary` used by the payroll engine.

- **Leave Entitlement**
  Manages annual caps for different leave categories:
    - Casual Leave Entitled vs. Used
    - Medical Leave Entitled vs. Used
    - Paid Leave Entitled vs. Used

- **One-to-One User Mapping**
  Ensures that each employee record has exactly one professional profile.

- **Branch-Specific Scoping**
  Associates employees with specific organizational branches for local HR management.

---

3. Database Design

3.1 Primary Table

Table Name: `user_meta_details`
Entity / Model: `UserMetaDetails.java`

Field Name	Type	Purpose
`id`	PK (Long)	Unique profile identifier.
`user_id`	FK (Long)	**Mandatory** One-to-One link to the `User` table.
`branch_id`	FK (Long)	Link to the organizational `Branch`.
`employ_type`	String	Type of employment (e.g., FULL_TIME, CONTRACT).
`monthly_base_salary`	Decimal	Fixed base pay for payroll calculations.

`monthly_working_days`	Integer	Expected days per month (Default: 26).
`per_day_working_hours`	Integer	Regular shift duration.
`half_day_working_hours`	Integer	Minimum hours to consider for a half-day.
`casual_leave_entitled`	Integer	Total annual casual leaves allowed.
`medical_leave_entitled`	Integer	Total annual medical leaves allowed.
`paid_leave_entitled`	Integer	Total annual paid leaves allowed.
`casual_leave_used`	Integer	Running count of casual leaves taken.
`medical_leave_used`	Integer	Running count of medical leaves taken.
`paid_leave_used`	Integer	Running count of paid leaves taken.

---

4. API Documentation

4.1 Controller

Controller Name: `UserMetaDetailsController`

4.2 Create Meta Details

- **Endpoint**: `POST /api/v1/user-meta-detail`
- **HTTP Method**: `POST`
- **Purpose**: Initial setup of professional details for a new user.

- **Parameters**: `UserMetaDetailsRequestDto` (Request Body).
- **Business Rule**: This must be called immediately after user registration to enable employee functionality.

4.3 Update Meta Details

- **Endpoint**: `PUT /api/v1/user-meta-detail/update`
- **HTTP Method**: `PUT`
- **Purpose**: Modify employment terms, salary, or leave caps.
- **Parameters**:
    - `id` – (Request Parameter): Primary key of the metadata record.
    - `UserMetaDetailsRequestDto` – (Request Body): Updated fields.

4.4 Fetch by User ID

- **Endpoint**: `GET /api/v1/user-meta-detail/user`
- **HTTP Method**: `GET`
- **Purpose**: Retrieves the professional profile for a specific employee.
- **Parameters**:
    - `userId` – (Request Parameter).

4.5 Branch-wise Scoped Fetch

- **Endpoint**: `GET /api/v1/user-meta-detail/getAll/branchWise`
- **HTTP Method**: `GET`
- **Purpose**: Allows branch managers to view all employee profiles within their specific branch context.
- **Parameters**: None (scoping handled via security context).

4.6 Get by Profile ID

- **Endpoint**: `GET /api/v1/user-meta-detail`
- **HTTP Method**: `GET`
- **Purpose**: Fetch details using the primary key of the meta record.
- **Parameters**:
    - `Id` – (Request Parameter).

---

5. Module Workflow (End-to-End Flow)

1. **Post-Creation Phase**
   After a new `User` is registered in the system, HR/Admin must call the `POST` API to create meta-details.
2. **Validation Phase**
    - The system verifies the `user_id` exists.
    - Ensures a meta-profile doesn't already exist for this user (Unique constraint).
    - Checks the validity of `branch_id`.

3. **Active Employment Phase**
    - Once metadata exists, the Attendance module uses `perDayWorkingHours` to mark presents/absents.
    - The Leave module checks the `leave_entitled` fields before approving any leave requests.
4. **Payroll Phase**
   At month-end, the Salary Slip generator pulls the `monthlyBaseSalary` and `monthlyWorkingDays` to calculate final payouts.
5. **Profile Lifecycle**
   If an employee is promoted or their terms change, the `update` API is used to adjust salary or leave structures.

---

6. Dependency & Integration

- **User Module**
  Provides the foundation identity record.
- **Attendance Module**
  Depends on this module for shift timings and working day definitions.
- **Leave Module**
  Depends on this module for leave balances and entitlement caps.
- **Payroll Module**
  Direct dependency on `monthlyBaseSalary` for salary slip generation.
- **Branch Module**
  Used for regional HR filtering and reporting.

---

7. Important Notes for Developers

- **Mandatory Creation**
  Failure to create metadata after user creation will break attendance and salary workflows for that user.

- **Unique Mapping**
  The relationship with the `User` table is `@OneToOne`. Trying to create multiple meta records for a single user ID will result in a database error.

- **Default Values**
  `monthlyWorkingDays` defaults to 26; ensure this aligns with the client's labor policies.

- **Leave Usage**
  The `used` fields (casual, medical, paid) are typically updated by the Leave Request module, not manually via this controller.

Branch Management

Branch Management

Whole Branch Management Can Do Only Admin

Creating Branch :

Description :

This API is used to **create a new branch in the system**. Only **admin users** have permission to perform this operation. The API captures essential branch details such as name, contact info, type, status, and location.

After creation, the branch details are returned along with metadata like **createdAt** and **editedBy**.

Request

- Method: POST
- Authentication: Required
- Authorization: Admin Only
- Request Body: Required

Only Admin can Create Branch

POST - /branch

Request :

```
{
  "branchName": "East Zone Service Center",
  "contactInfo": "opseast@zencorp.com",
```

```
  "phoneNumber": "+91 93345 88990",
  "branchStatus": "ACTIVE",
  "branchType": "SERVICE",
  "pincode": "700091",
  "city": "Kolkata",
  "state": "West Bengal",
  "location": "Salt Lake Sector V"
}
```


Response :


```
{
    "status": 201,
    "message": "Branch Created",
    "data": {
        "branchId": 4,
        "branchName": "East Zone Service Center",
        "contactInfo": "opseast@zencorp.com",
        "phoneNumber": "+91 93345 88990",
        "branchStatus": "ACTIVE",
        "branchType": "SERVICE",
        "editedBy": "Palak Dave",
        "pincode": "700091",
        "city": "Kolkata",
        "state": "West Bengal",
        "location": "Salt Lake Sector V",
        "createdAt": "2026-01-12T16:30:13.3233182"
    }
}
```

Updating Branch :

Description :

This API is used to **update an existing branch's details**. Only **admin users** have permission to perform this operation. You can update branch metadata such as **name, contact info, type, status, location, and address details**.

The API returns the **updated branch details** along with metadata like `editedBy` and `createdAt`.

Request

- Method: PUT
- Authentication: Required
- Authorization: Admin Only
- Request Body: Required

PUT - /branch/update

```
Request :
{
  "id":4,
  "branchName": "GreenTechTech ServiceService Point (Koramangala)",
  "contactInfo": "greentech.koramangala@gmail.com",
  "phoneNumber": "+91 98765 43210",
  "branchStatus": "INACTIVE",    // Must Be ACTIVE / INACTIVE
  "branchType": "STORAGE_UNIT",    // Must Be SERVICE / RETAIL / STORAGE_UNIT
  "pincode": "560034",
```

```
  "city": "Bangalore",
  "state": "Karnataka",
  "location": "1st Floor, Beta Complex, 5th Block, Koramangala"
}
```


Response ;


```
{
    "status": 200,
    "message": "Branch Updated Successfully!",
    "data": {
        "branchId": 4,
        "branchName": "GreenTechTech ServiceService Point (Koramangala)",
        "contactInfo": "greentech.koramangala@gmail.com",
        "phoneNumber": "+91 98765 43210",
        "branchStatus": "INACTIVE",
        "branchType": "STORAGE_UNIT",
        "editedBy": "Palak Dave",
        "pincode": "560034",
        "city": "Bangalore",
        "state": "Karnataka",
        "location": "1st Floor, Beta Complex, 5th Block, Koramangala",
        "createdAt": "2026-01-12T16:30:13.323318"
    }
}
```

Delete Branch :

Description :

This API is used to **soft delete (deactivate) a branch in the system**. Instead of permanently removing the branch, the API marks it as inactive, ensuring historical records, user assignments, and reports remain intact.

Only **admin users** can perform this operation.

Request

- Method: DELETE
- Authentication: Required
- Authorization: Admin Only
- Request Body: Required

DELETE - /branch/delete

Request :

```
{
    "id": 10
}
```

Response :

```
{
    "status": 200,
    "message": "Branch Deleted Successfully!",
```

```
    "data": {
        "branchId": 4,
        "branchName": "GreenTechTech ServiceService Point (Koramangala)",
        "contactInfo": "greentech.koramangala@gmail.com",
        "phoneNumber": "+91 98765 43210",
        "branchStatus": "INACTIVE",
        "branchType": "STORAGE_UNIT",
        "editedBy": "Palak Dave",
        "pincode": "560034",
        "city": "Bangalore",
        "state": "Karnataka",
        "location": "1st Floor, Beta Complex, 5th Block, Koramangala",
        "createdAt": "2026-01-12T16:30:13.323318"
    }
}
```

Branch Get By Id :

Description :

This API is used to **fetch detailed information of a specific branch** using its **branch ID**. Only admin users are authorized to access this endpoint.

It returns complete branch metadata, including contact information, status, type, location, and admin details, which can be used for branch management, reporting, or updates.

Request

- Method: POST
- Authentication: Required
- Authorization: Admin Only
- Request Body: Required

POST - /branch/byid

Request ;

```
{
    "id":1
}
```

Response ;

```
{
    "status": 200,
```

```
    "message": "Branch Found Successfully",
    "data": [
        {
            "branchId": 1,
            "branchName": "GreenTechTech ServiceService Point (Koramangala)",
            "contactInfo": "greentech.koramangala@gmail.com",
            "phoneNumber": "+91 98765 43210",
            "branchStatus": "INACTIVE",
            "branchType": "STORAGE_UNIT",
            "editedBy": "Palak Dave",
            "pincode": "560034",
            "city": "Bangalore",
            "state": "Karnataka",
            "location": "1st Floor, Beta Complex, 5th Block, Koramangala",
            "createdAt": "2025-12-04T17:45:22.600505"
        }
    ]
}
```

Get All Branches :

Description :

This API is used to **fetch the list of all branches** in the system. Only admin users are authorized to access this endpoint.

The response includes all branch details such as name, contact info, status, type, location, and admin metadata. This API is useful for branch management, reporting, and administrative operations.

Request

- Method: GET
- Authentication: Required
- Authorization: Admin Only
- Request Body: Not Required

GET - /branch/all

Request not required.

Response :


```
{
    "status": 200,
    "message": "All Branch Successfully",
    "data": {
        "count": 1,
        "results": [
```

```
            {
                "branchId": 2,
                "branchName": "Jindagi",
                "contactInfo": "aurifex@gmail.com",
                "phoneNumber": null,
                "branchStatus": "ACTIVE",
                "branchType": null,
                "editedBy": "Aryan",
                "pincode": null,
                "city": null,
                "state": null,
                "location": "Mumbai",
                "createdAt": "2025-12-05T15:31:55.684942"
            }
        ]
}
```

Get All Branches Via Filters :

Description :

This API is used to **retrieve branches based on filtering, searching, pagination, and sorting criteria**. Only admin users can access this endpoint.

It allows admins to:

- Filter branches by **type, status, and creation date range**
- Search branches by **name, location, or contact info**
- Apply **pagination** for large datasets
- Sort results by **branch metadata** (e.g., `createdAt`, `branchId`)

The response returns the filtered and paginated list of branch details.

Request

- Method: POST
- Authentication: Required
- Authorization: Admin Only
- Request Body: Required

POST - /branch/filter

Request :

```
{
  "paginationRequest":
  {
    "pageNumber": 0,
    "pageSize": 10
  },
  "filterColumns":
  {
    // "branchType":"SERVICE", // RETAIL, STORAGE_UNIT
    // "branchStatus": "INACTIVE" // ACTIVE
    // "startDate": "2025-10-21T00:00:00",
    // "endDate": "2025-10-22T23:59:59"
  },
  "searchColumns":
  {
    // "branchName":"NextGen Tech Hub",
    // "location":"Ground Floor, Gamma Tower, 7th Block, Marathahalli",
    // "contactInfo":"support@nextgentech.com" // This field annotate as a email
address
  },
  "orderByColumns":
  {
    "createdAt": "desc"
    // "branchId":"asc"
  }
}
```

Response :

```
{
    "status": 200,
    "message": "Branches retrieved successfully!",
    "data": {
        "count": 3,
        "results": [
            {
                "branchId": 3,
                "branchName": "North Zone Operations",
                "contactInfo": "opsnorth@zencorp.com",
                "phoneNumber": "+91 84477 99881",
                "branchStatus": "ACTIVE",
```

```
            "branchType": "SERVICE",
            "editedBy": "Aryan",
            "pincode": "110089",
            "city": "New Delhi",
            "state": "Delhi",
            "location": "Sector 8, Rohini Depot",
            "createdAt": "2025-12-05T17:39:48.386898"
        }
    ]
}
```

Get Branch Drop Down :

Description :

This API is used to **fetch a simplified list of all branches** suitable for use in dropdowns, forms, or selection inputs. Only admin users are authorized to access this endpoint.

The response provides **branch IDs and names** to populate dropdown menus in the frontend.

Request

- Method: GET
- Authentication: Required
- Authorization: Admin Only
- Request Body: Not Required

GET – /branch/dropdown

Request Not Required.

Response :


```
{
    "status": 200,
```

```
    "message": "Branch Drop Down List",
    "data": {
        "count": 2,
        "results": [
            {
                "id": 2,
                "value": "Jindagi"
            },
            {
                "id": 3,
                "value": "North Zone Operations"
            }
        ]
    }
}
```

Branch Management – Role-Based Access Description

The Branch Management module is strictly controlled and centralized at the company level to ensure consistency, security, and proper governance across the organization. Only the Company Admin has authority to manage branches, while all other roles have no access to branch-level administrative actions.

This clear separation prevents unauthorized changes and maintains standardized branch operations.

Authorized Role

Company Admin

- Has full control over Branch Management
- Can:
    - Create new branches
    - Update existing branch details
    - Delete branches
- Can create, update, or delete Branch Admins linked to respective branches

Unauthorized Roles

The following roles do not have any permissions in the Branch Management module:

- Branch Admin
- Technician Manager

- Sales Manager
- Account Manager

These roles:

- Cannot create branches
- Cannot edit branch details
- Cannot delete branches
- Should not see any branch management actions or controls in the UI

Access Control Notes

- Branch Management features must be visible only to Company Admin
- All branch-related buttons, forms, and actions should be hidden or disabled for other roles
- Backend will enforce permissions, but frontend must strictly follow role-based visibility

Inventory

Inventory Management

Only ADMIN and BRANCHADMIN CAN ACCESS

Add Inventory :

Description :

This API is used to **add a new inventory** item into the system. Both **Admin** and **Branch Admin** users are authorized to access this endpoint.

It supports detailed inventory creation including:

- **Item details**: name, description, brand, category, HSN/SKU/EAN codes
- **Stock management**: stock quantity, low stock threshold, unit type, expiry date
- **Pricing**: selling price, purchase price, price type
- **Rental options**: rental rate, deposit, insurance, and rental quantity (if applicable)
- **Media attachments**: upload images or documents for the inventory item

The response returns the added inventory item details along with document URLs.

Request

- Method: POST
- Authentication: Required
- Authorization: Admin and Branch Admin Only
- Request Body: Required

POST - /api/v2/inventory

Request :


```
{
  "itemName": "Neem Antibacterial Powder",
  "itemDescription": "Herbal powder widely used for medicinal, cosmetic, and organic
pest-control applications",
  "lowStockThreshold": 12.0,
  "brandName": "EcoHerb Organics",
  "productCategories": "CHEMICALS",
  "hsnCode": "1211",
  "skuCode": "EH-NM-210",
  "ean": "8906677889903",
  "returnable": false,
  "taxId": 2,
  "productStatus": "ACTIVE",
  "branchId": 1,
  "rentable": false,
  "variants": [
    {
      "stockQuantity": 95,
      "sellingPriceType": "RETAIL",
      "sellingPrice": 320.00,
      "purchasePriceType": "WHOLESALE",
      "purchasePrice": 260.00,
      "unitType": "KG",
      "expiryDate": "2026-11-20T00:00:00"
    }
  ],
  "defaultRentalRate": 2500.00,
  "rentalRateFrequency": "MONTHLY",
  "defaultDepositAmount": 10000,
  "insuranceValue": 5000,
  "rentalProductQuantity": 10,
  "rentalProductStatus": "ACTIVE",

  "files":[
    {
```

```
        "fileName":"ca abhi dholakiya.png",
        "fileType":"image/png",
        "fileData":"
      }
}
```

Response :


```
{
    "status": 200,
    "message": "Inventory Added Successfully!!",
    "data": {
        "count": 1,
        "results": [
            {
                "itemId": 27,
                "itemName": "Neem Antibacterial Powder",
                "itemDescription": "Herbal powder widely used for medicinal, cosmetic,
and organic pest-control applications",
                "lowStockThreshold": 12.0,
                "createdAt": "2026-01-12T16:46:35.2549017",
                "lastModifiedAt": "2026-01-12T16:46:35.2549017",
                "branchId": 1,
                "brandName": "EcoHerb Organics",
                "productCategories": "CHEMICALS",
                "hsnCode": "1211",
                "skuCode": "EH-NM-210",
                "ean": "8906677889903",
                "taxId": 2,
                "productStatus": "ACTIVE",
                "active": true,
                "stockQuantity": 95.0,
                "sellingPriceType": "RETAIL",
                "sellingPrice": 320.0,
                "purchasePriceType": "WHOLESALE",
                "purchasePrice": 260.0,
                "unitType": "KG",
                "unitTypeValue": 0.0,
                "measurementType": null,
                "measurement": 0.0,
                "expiryDate": "2026-11-20T00:00:00",
                "rentable": false,
                "defaultRentalRate": null,
```

```
                "rentalRateFrequency": null,
                "defaultDepositAmount": null,
                "insuranceValue": null,
                "rentalProductQuantity": null,
                "rentalProductStatus": null,
                "documentsUrls": [

"https://seravion3960.s3.amazonaws.com/tenant_1_aryanraval299_gmail_com/inventory/331e
1811-bc62-48ec-b4d0-bab802d640b0_caabhidholakiya.png?X-Amz-Algorithm=AWS4-HMAC-SHA256&
X-Amz-Date=20260112T111640Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credentia
l=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=8915
50e561d4137b82f8288910061a20aa5434aabd630142aaf222b36c8b5d49",

"https://seravion3960.s3.amazonaws.com/tenant_1_aryanraval299_gmail_com/inventory/ff46
949b-d86b-437e-91c7-8ac270b4da8d_WhatsAppImage2024-12-05at16.46.35_76d6a594.jpg?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T111640Z&X-Amz-SignedHeaders=host&X-Amz-
Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_r
equest&X-Amz-Signature=8f9aabf41b979d1025eec59193633ac3998b24e78a78b5f6225654aa3114fbd
0"
                ],
                "returnable": false
            }
        ]
    }
}
```

Delete Particular Inventory :

Description :

This API is used to **delete a specific inventory item** from the system. Only **Admin** and **Branch Admin** users are authorized to perform this action.

The deletion is **permanent**, and the response returns the details of the deleted inventory item for confirmation.

Request

- Method: DELETE
- Authentication: Required
- Authorization: Admin and Branch Admin Only

DELETE - /api/v2/inventory?itemId=29

Request Parameter Required as an InventoryId.

Response ;


```
{
    "status": 200,
    "message": "Inventory Deleted Successfully!!",
    "data": {
        "itemId": 29,
        "itemName": "Neem Antibacterial Powder",
        "itemDescription": "Herbal powder widely used for medicinal, cosmetic, and
organic pest-control applications",
        "lowStockThreshold": 12.0,
        "createdAt": "2026-01-12T16:47:41.546814",
        "lastModifiedAt": "2026-01-12T16:47:41.546814",
        "branchId": 1,
```

```
        "brandName": "EcoHerb Organics",
        "productCategories": "CHEMICALS",
        "hsnCode": "1211",
        "skuCode": "EH-NM-210",
        "ean": "8906677889903",
        "taxId": 2,
        "productStatus": "ACTIVE",
        "active": true,
        "stockQuantity": 95.0,
        "sellingPriceType": "RETAIL",
        "sellingPrice": 320.0,
        "purchasePriceType": "WHOLESALE",
        "purchasePrice": 260.0,
        "unitType": "KG",
        "unitTypeValue": 0.0,
        "measurementType": null,
        "measurement": 0.0,
        "expiryDate": "2026-11-20T00:00:00",
        "rentable": false,
        "defaultRentalRate": null,
        "rentalRateFrequency": null,
        "defaultDepositAmount": null,
        "insuranceValue": null,
        "rentalProductQuantity": null,
        "rentalProductStatus": null,
        "documentsUrls": [

"https://seravion3960.s3.amazonaws.com/tenant_1_aryanraval299_gmail_com/inventory/5a117312-a5ea-45b3-b09a-2f4b615b7849_caabhidholakiya.png?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T111916Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=f9298eacad1f1a884f388dd3ffc48d2c9db44676eca2f4812dffe80d5207a0b3",

"https://seravion3960.s3.amazonaws.com/tenant_1_aryanraval299_gmail_com/inventory/81602a7b-4d2f-43fc-8a04-f1313c253e0d_WhatsAppImage2024-12-05at16.46.35_76d6a594.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T111916Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=da62f32f8bc6a82803d3c1dce57946f88ca50e7af00a8fcdaabf4f64a3e2b8d0"
        ],
        "returnable": false
    }
```

```
}
```

Update Inventory :

Description :

This API allows **updating an existing inventory item** in the system. It can be used to modify general product information, stock details, pricing, or rental settings. Both **Admin** and **Branch Admin** users are authorized to access this endpoint.

This API supports updating both regular inventory items and rental inventory items with respective fields.

Request

- Method: PUT
- Authentication: Required
- Authorization: Admin and Branch Admin Only

PUT - /api/v2/inventory

```
{
  "itemId":26,
  "itemName": "Herbal Chamomile 11",
  "itemDescription":"Used For 11=kin care, also in making natural medicines",
  "lowStockThreshold":11.0,
  "brandName": "GreenLeaf 11",
  "productCategories": "CHEMICALS",
  "hsnCode": "0914",
  "skuCode": "GL-CH-311",
  "ean": "8901234567891",
```

```
  "returnable": false,
  "taxId": 5,
  "productStatus": "ACTIVE",
  "branchId": 1,
  "active":true,
  "rentable":false,

      "stockQuantity": 100,
      "sellingPriceType": "RETAIL",
      "sellingPrice": 350.00,
      "purchasePriceType": "WHOLESALE",
      "purchasePrice": 360.00,
      "unitType": "Gram",
      "unitTypeValue": 370,
      "measurementType": "Weight",
      "measurement": 380.00,
      "expiryDate": "2026-01-31T00:00:00"
}

// For updating equipment
// {
//   "itemId":26,
//   "itemName": "Herbal Chamomile 18",
//   "itemDescription":"Used For Skin care, also in making natural medicines",
//   "lowStockThreshold":11.0,
//   "brandName": "GreenLeaf",
//   "productCategories": "CHEMICALS",
//   "hsnCode": "0912",
//   "skuCode": "GL-CH-302",
//   "ean": "8901234567823",
//   "returnable": true,
//   "taxId": 4,
//   "productStatus": "ACTIVE",
//   "branchId": 1,
//   "rentable":true,
//   "defaultRentalRate":6.5,
//   "rentalRateFrequency":"DAILY",
//   "defaultDepositAmount":1500,
//   "insuranceValue":400,
//   "rentalProductQuantity":4,
//   "rentalProductStatus":"AVAILABLE"
// }
```

Response :


```
{
    "status": 200,
    "message": "Inventory Updated Successfully!!",
    "data": {
        "itemId": 26,
        "itemName": "Herbal Chamomile 11",
        "itemDescription": "Used For 11=kin care, also in making natural medicines",
        "lowStockThreshold": 11.0,
        "createdAt": "2026-01-09T15:45:49.172858",
        "lastModifiedAt": "2026-01-09T15:46:32.271812",
        "branchId": 1,
        "brandName": "GreenLeaf 11",
        "productCategories": "CHEMICALS",
        "hsnCode": "0914",
        "skuCode": "GL-CH-311",
        "ean": "8901234567891",
        "taxId": 5,
        "productStatus": "ACTIVE",
        "active": true,
        "stockQuantity": 100.0,
        "sellingPriceType": "RETAIL",
        "sellingPrice": 350.0,
        "purchasePriceType": "WHOLESALE",
        "purchasePrice": 360.0,
        "unitType": "Gram",
        "unitTypeValue": 370.0,
        "measurementType": "Weight",
        "measurement": 380.0,
        "expiryDate": "2026-01-31T00:00:00",
        "rentable": false,
        "defaultRentalRate": null,
        "rentalRateFrequency": null,
        "defaultDepositAmount": null,
        "insuranceValue": null,
        "rentalProductQuantity": null,
        "rentalProductStatus": null,
        "documentsUrls": [

"https://seravion3960.s3.amazonaws.com/tenant_1_shini90_gmail_com/inventory/b1a51985-f
```

5da-48d7-ac01-30ba279b90b0_WhatsAppImage2026-01-09at1.16.33PM.jpeg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T112231Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=0b062d7a0a462ff9c0eb6062732e1bc8c34ffed4c6337da4fe0d1941d35f9871"
        ],
        "returnable": false
    }
}

Get Inventory By Id :

Description :

This API fetches **the complete details of a specific inventory item** using its unique ID. It returns all related information including stock details, pricing, rental settings, and attached documents. Only Admin and Branch Admin users have access to this endpoint.

Request

- Method: GET
- Authentication: Required
- Authorization: Admin and Branch Admin Only

GET - /api/v2/inventory/byId?id=26

Request Parameter as Inventory Id .

Response :


```
{
    "status": 200,
    "message": "Expected Inventory Fetched !!",
    "data": {
        "itemId": 26,
        "itemName": "Herbal Chamomile 11",
        "itemDescription": "Used For 11=kin care, also in making natural medicines",
        "lowStockThreshold": 11.0,
        "createdAt": "2026-01-09T15:45:49.172858",
        "lastModifiedAt": "2026-01-12T11:22:31.79539",
```

```
        "branchId": 1,
        "brandName": "GreenLeaf 11",
        "productCategories": "CHEMICALS",
        "hsnCode": "0914",
        "skuCode": "GL-CH-311",
        "ean": "8901234567891",
        "taxId": 5,
        "productStatus": "ACTIVE",
        "active": true,
        "stockQuantity": 100.0,
        "sellingPriceType": "RETAIL",
        "sellingPrice": 350.0,
        "purchasePriceType": "WHOLESALE",
        "purchasePrice": 360.0,
        "unitType": "Gram",
        "unitTypeValue": 370.0,
        "measurementType": "Weight",
        "measurement": 380.0,
        "expiryDate": "2026-01-31T00:00:00",
        "rentable": false,
        "defaultRentalRate": null,
        "rentalRateFrequency": null,
        "defaultDepositAmount": null,
        "insuranceValue": null,
        "rentalProductQuantity": null,
        "rentalProductStatus": null,
        "documentsUrls": [

"https://seravion3960.s3.amazonaws.com/tenant_1_shini90_gmail_com/inventory/b1a51985-f5da-48d7-ac01-30ba279b90b0_WhatsAppImage2026-01-09at1.16.33PM.jpeg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T112307Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=33933fc90bb6b15573352811aea1a268f3633f94c4bc6d28b9bf4353bb61a5b4"
        ],
        "returnable": false
    }
}
```

Get Inventory By Id for Task Form :

Description :

This API fetches the **complete details of a specific inventory item** using its unique ID. It returns all related information including stock details, pricing, rental settings, and attached documents. Only Admin and Branch Admin users have access to this endpoint.

Request

- Method: GET
- Authentication: Required
- Authorization: Admin and Branch Admin Only

GET - /api/v2/inventory/form/byId?id=23

Request Parameter Required as a Inventory Id

Response ;


```
{
    "status": 200,
    "message": "Expected Inventory Fetched !!",
    "data": {
        "itemId": 23,
        "itemName": "Cockroach Trap Box",
        "skuCode": "PX-UN-003",
        "sellingPrice": 95.0,
        "taxRate": 7.00,
        "documentUrls": [
```

"https://seravion3960.s3.amazonaws.com/tenant_1_shini90_gmail_com/inventory/6bb8edd4-69e9-4e8f-b259-461ab75b7adb_caabhidholakiya.png?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T112542Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=567ba4ad4af4324f26b022b5fffd3d831df7603f3cb855ff78ffcdb59d5555ca",

"https://seravion3960.s3.amazonaws.com/tenant_1_shini90_gmail_com/inventory/0016d55f-edb8-4acb-a00b-fdde456812aa_WhatsAppImage2024-12-05at16.46.35_76d6a594.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T112542Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=45ac5f1bdc426676251fbc837bf6b1a832662f69ff50dcdbd3004d542a62cb55"
        ]
    }
}

Get Inventories For ADMIN APIs :

Description :

This API retrieves a **list of all inventory items** in the system. It includes details such as stock, pricing, product information, rental status, and associated documents. Access is restricted to Admin and Branch Admin users.

Request

- Method: GET
- Authentication: Required
- Authorization: Admin and Branch Admin Only

GET - /api/v2/inventory

Request not Required.
Respons e;


```
{
    "status": 200,
    "message": "Inventory Added Successfully!!",
    "data": {
        "count": 14,
        "results": [
            {
                "itemId": 30,
                "itemName": "Neem Antibacterial Powder",
                "itemDescription": "Herbal powder widely used for medicinal, cosmetic,
and organic pest-control applications",
                "lowStockThreshold": 12.0,
                "createdAt": "2026-01-12T16:48:18.166426",
                "lastModifiedAt": "2026-01-12T16:48:18.166426",
                "branchId": 1,
```

```
            "brandName": "EcoHerb Organics",
            "productCategories": "CHEMICALS",
            "hsnCode": "1211",
            "skuCode": "EH-NM-210",
            "ean": "8906677889903",
            "taxId": 2,
            "productStatus": "ACTIVE",
            "active": true,
            "stockQuantity": 95.0,
            "sellingPriceType": "RETAIL",
            "sellingPrice": 320.0,
            "purchasePriceType": "WHOLESALE",
            "purchasePrice": 260.0,
            "unitType": "KG",
            "unitTypeValue": 0.0,
            "measurementType": null,
            "measurement": 0.0,
            "expiryDate": "2026-11-20T00:00:00",
            "rentable": false,
            "defaultRentalRate": null,
            "rentalRateFrequency": null,
            "defaultDepositAmount": null,
            "insuranceValue": null,
            "rentalProductQuantity": null,
            "rentalProductStatus": null,
            "documentsUrls": [

"https://seravion3960.s3.amazonaws.com/tenant_1_aryanraval299_gmail_com/inventory/74e1c243-0eb9-41a4-8cd4-8f88d57bfca4_caabhidholakiya.png?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T112740Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=edc542866a2bbc5ae7fcfee5a11b9750d183cad659a42956a31908d93b2058c8",

"https://seravion3960.s3.amazonaws.com/tenant_1_aryanraval299_gmail_com/inventory/744c4fcb-d943-471a-821b-468e36a91c31_WhatsAppImage2024-12-05at16.46.35_76d6a594.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T112740Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=b74038e1ca88a7ba864a7053a2064642ec40ebf2eda84c1b79f9b2c8c31df80d"
            ],
            "returnable": false
        }
```

```
        ]
}
```

Get Inventories Branch Wise :

For Sub User but only Access By BRANCHADMIN

`GET` – /api/v2/inventory/branchWise

Request not required.

Response :


```
{
    "status": 200,
    "message": "Inventory Fetched Successfully!!",
    "data": {
        "count": 26,
        "results": [
            {
                "itemId": 1,
                "itemName": "Neem Antibacterial Powder",
                "itemDescription": "Herbal powder widely used for medicinal, cosmetic,
and organic pest-control applications",
                "lowStockThreshold": 12.0,
                "createdAt": "2026-01-09T08:56:46.871908",
                "lastModifieddAte": "2026-01-09T08:56:46.871908",
                "branchId": 1,
                "brandName": "EcoHerb Organics",
                "productCategories": "CHEMICALS",
                "hsnCode": "1211",
                "skuCode": "EH-NM-210",
                "ean": "8906677889903",
                "taxId": 2,
                "productStatus": "ACTIVE",
                "active": true,
                "stockQuantity": 95.0,
                "sellingPriceType": "RETAIL",
                "sellingPrice": 320.0,
                "purchasePriceType": "WHOLESALE",
```

```
                "purchasePrice": 260.0,
                "unitType": "KG",
                "unitTypeValue": 0.0,
                "measurementType": null,
                "measurement": 0.0,
                "expiryDate": "2026-11-20T00:00:00",
                "rentable": false,
                "defaultRentalRate": null,
                "rentalRateFrequency": null,
                "defaultDepositAmount": null,
                "insuranceValue": null,
                "rentalProductQuantity": null,
                "rentalProductStatus": null,
                "documentsUrls": [

"https://seravion3960.s3.amazonaws.com/tenant_1_shini90_gmail_com/inventory/9d9f11ce-5eeb-4157-a362-698dbcf9644a_caabhidholakiya.png?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T113024Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=512ef6da94358a82339ecc9b194144cef419b4c67f640641807e2f29684864d6",

"https://seravion3960.s3.amazonaws.com/tenant_1_shini90_gmail_com/inventory/0a4acf08-b95b-415e-8dd7-26687ac74610_WhatsAppImage2024-12-05at16.46.35_76d6a594.jpg?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20260112T113024Z&X-Amz-SignedHeaders=host&X-Amz-Expires=600&X-Amz-Credential=AKIAYD2RZA5W7MTHOXVI%2F20260112%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Signature=15ef372f6dc9318782ee4bc39eedfb7057e4c127f5febf6715792d875f0789fa"
                ],
                "returnable": false
            }
        }
}
```


Get Drop Down of Product (Rentable False):

GET - /api/v2/inventory/dropdown

Request not Required.

Response ;


```
{
    "status": 200,
    "message": "Product Drop Down Fetched !!",
```

```
    "data": {
        "count": 26,
        "results": [
            {
                "id": 1,
                "value": "Neem Antibacterial Powder"
            },
            {
                "id": 2,
                "value": "Cockroach Control Powder"
            },
            {
                "id": 3,
                "value": "Ant Control Powder"
            }
       }
}
```


Get Drop Down of Equipment (Rentable True) :

`GET` - /api/v2/inventory/dropdown/equipment

Request not Required.

Response :


```
{
    "status": 200,
    "message": "Equipment Drop Down Fetched !!",
    "data": {
        "count": 0,
        "results": [
             {
                "id": 3,
                "value": "Ant Control Powder"
            }
       ]
    }
}
```

Role-Wise Inventory Permissions

Company Admin

- Has **full control** over inventory across all branches
- Can:
    - Create inventory records **branch-wise**
    - Update inventory details for any branch
    - Delete inventory records for any branch
- Oversees overall stock availability and distribution at the company level

Branch Admin

- Has full inventory control **only for their assigned branch**
- Can:
    - Add new inventory items
    - Update existing inventory details
    - Delete inventory records
- Cannot access or modify inventory of other branches

Technician Manager

- Has **no access** to the Inventory Management module
- Cannot:
    - View inventory
    - Add, update, or delete inventory

Sales Manager & Account Manager

- Have **view-only access** to inventory
- Can:
    - View inventory details and stock levels
- Cannot:
    - Add inventory
    - Update inventory
    - Delete inventory

Access Control Notes

- Inventory actions (Add / Edit / Delete) must be shown **only** to authorized roles
- View-only roles should see inventory data **without action buttons**
- Technician Manager should not see the inventory module at all
- Backend validation will enforce rules, but frontend must control visibility and user experience

Tax

Tax management

Postman : Inventory -> Tax
Admin Only Access :

Add Tax :
POST - /tax
-> Fill the form, only does not require any other resources.

Update Tax :
PUT - /tax/update
-> Update Tax Details using Tax Id.

Get By Id Tax ;
POST - /tax/byid
-> Get specific details of tax via tax id.
-> Pass tax id as a requestBody.

Delete Tax By Id :
DELETE - /tax/delete
-> Delete specific tax by tax id.
-> pass tax id as a requestBody.

Get All Taxes :
GET - /tax/all
-> Gives all tax details.

Get All Taxes Via Filter :

GET - /tax/find

-> Give All tax details via filter matching.

Request :

```
{
    "paginationRequest":
    {
        "pageNumber":0,
        "pageize":5
    },
    "filterColumns":
    {
        "taxType":"PERCENTAGE"  // PERCENTAGE, FIXED_AMOUNT
        // "taxName": "IGST" // CGST, SGST, UTGST, IGST, LUXURY_TAX, OTHER
        // "minRate":5.0,   // for Rate Range
        // "maxRate":8.0
        // "startDate":"2025-10-21 13:46:08.053505",    // for date range
        // "endDate":"2025-10-21 13:46:28.66149"
    },
    "searchColumns":
    {
        // "taxName":"OT"
    },
    "orderByColumns":
    {
        // "createdAt":"desc",
        "taxRate":"asc"
    }
}
```

GET TAX DROP DOWN :

Creation remaiming

Tax Management – Role-Based Access Description

The Tax Management module is designed to centrally manage all tax-related configurations used across the system. To maintain accuracy, compliance, and control, access to this module is strictly limited based on user roles.

Only authorized roles can manage or view tax details, while other roles are completely restricted from this module.

Role-Wise Tax Permissions

Company Admin

- Has full control over the Tax Management module
- Can:
    - Add new tax configurations
    - Update existing tax details
    - Delete tax records
- Responsible for maintaining correct and up-to-date tax settings for the entire company

Branch Admin

- Has view-only access to tax details
- Can:
    - View tax configurations
- Cannot:

    - Add, update, or delete tax records

Account Manager

- Has view-only access to tax details
- Can:
    - View tax configurations
- Cannot:
    - Modify or delete tax records

Technician Manager & Sales Manager

- Have no access to the Tax Management module
- The tax module should be:
    - Hidden
    - Disabled
      for these roles

Access Control Notes

- Only Company Admin should see Add / Edit / Delete tax actions
- View-only roles should see tax data without action buttons
- Disabled roles should not see the tax module in navigation

- Frontend must enforce visibility based on role, with backend validation as support

Service Management

Service Management

Create Service :

POST - /service

-> ADMIN and BRANCHADMIN can create a service, and does not require any other resource fill the form only.

-> Select Branch Name from Branch Drop Down, only one Branch.

=> GET - /branch/dropdown (Enhanced ERP -> Branch -> Branch Dropdown)

-> Select Multiple Inventories From Inventory Drop Down.

=> GET - /api/v2/inventory/dropdown (Enhanced ERP -> Inventory -> Product -> Get Product Drop Down)

Update Service :

PUT - /service/update

-> ADMIN and BRANCHADMIN can update service, does not require any other resource fill the form only.

-> Branch does not update .

-> Select Multiple Inventories From Inventory Drop Down.

=> GET - /api/v2/inventory/dropdown (Enhanced ERP -> Inventory -> Product -> Get Product Drop Down)

=> Already existing inventories removed and new are updated.

Delete Service :

DELETE - /service/delete

-> ADMIN and BRANCHADMIN can delete service.

-> Pass ServiceId into request for delete specific Service.

Get All Services With Filter :

POST - /service/filter

-> ADMIN and BRANCHADMIN can filter all service via this request.

Request like :

```
{
  "paginationRequest":
  {
    "pageNumber": 0,
    "pageSize": 10
  },
```

```
  "filterColumns":
  {
    // "serviceStatus":"ACTIVE",            // INACTIVE
    // "serviceCategory":"COMMERCIAL",      // RESIDENTIAL
    // "startDate":"2025-10-29T11:53:00",   // Date Range
    // "endDate":"2025-10-30T00:00:00"      // Date Range
    // "startPrice":100.0,                     // Price Range
    // "endPrice":300.0                        // Price Range
  },
  "searchColumns":
  {
    // "serviceName":"om"                   // Search By Service Name
  },
  "orderByColumns":
  {
    // "createdAt":"asc",                   // Sorting Via created Time
    "servicePrice":"desc"                   // Sorting Via Service Price
  }
}
```

Get All Service :

GET - /service/all

-> Using this api, ADMIN and BRANCHADMIN can access all services.

Get All Service Branch Wise For Managers :

GET - /service/branchWise

-> Get All services Branch Wise. It takes require branchId from the backend. And give all services via branch wise.

-> Only BRANCHADMIN can access. But other users can use it when required.

Service Drop Down BranchWise :

GET - /service/dropdown

-> Get All service Drop Down Branch Wise. It takes require branchId from the backend. And give all service drop down via branch wise.

-> Only BRANCHADMIN can access. But other users can use it when required.

Get Service by Id :

GET - /service/byid?id=100

-> ADMIN and BRANCHADMIN can access this api for update service.

-. When an update form opens for a particular service, all details are received via this.

Notes :
Other users can also access this service module when users have permission to access this module with actions.

Services Management – Role-Based Access Description

The **Services Management** module controls how services are created, maintained, and managed across branches and technicians. Access to this module is role-based to ensure services are handled by the right level of authority while maintaining operational clarity and accountability.

Each role interacts with services differently, based on responsibility and scope.

Role-Wise Services Permissions

Company Admin

- Has **view-only access** to services
- Can:
    - View services **branch-wise** across the entire company
- Cannot:
    - Create services

    - Update services
    - Delete services
- Uses this access mainly for monitoring and reporting purposes

Branch Admin

- Has **full control** over services within their assigned branch
- Can:
    - Create new services for the branch
    - Update existing branch services
    - Delete branch services
- Cannot manage services of other branches

Technician Manager

- Has **full control** over services assigned to their technicians
- Can:
    - Create services for technicians
    - Update technician-related services
    - Delete technician-related services
- Service access is limited to their managed technicians only

Sales Manager & Account Manager

- Have **no access** to the Services Management module
- The services module should be:
    - Disabled
    - Hidden
      for these roles

Access Control Notes

- Service actions (Add / Edit / Delete) must be visible **only** to Branch Admin and Technician Manager
- Company Admin should see services in **read-only mode**
- Disabled roles should not see the services module in navigation
- Frontend must enforce role-based visibility; backend will validate permissions

CRM

CRM Management – Role-Based Access Description

The **CRM (Customer Relationship Management)** module is divided into three core sections: **Leads**, **Follow-ups**, and **Customers**. Access to each section is strictly controlled based on user roles to ensure focused responsibilities, clean workflows, and data security.

Each role interacts with CRM data differently depending on their operational involvement in sales, customer handling, and financial processes.

Role-Wise CRM Permissions

Company Admin

- Has **view-only access** to CRM data
- Can:
    - View **Leads**
    - View **Customers**
- Cannot:
    - Create or edit Leads
    - Create or edit Customers
    - Access Follow-ups
- **Follow-up module is disabled** for Company Admin

Branch Admin

- Has limited CRM access
- Can:
    - View **Leads**
    - Create and edit **Customers**
- Cannot:
    - Create or edit Leads
    - Access Follow-ups
- **Follow-up module is disabled** for Branch Admin

Technician Manager

- Has **no access** to the CRM module
- Leads, Follow-ups, and Customers should be:
    - Hidden
    - Disabled

Sales Manager

- Has **full access** to the CRM module

- Can:
    - Create, view, update **Leads**
    - Create, view, update **Follow-ups**
    - Create, view, update **Customers**
- Responsible for complete CRM lifecycle management

Account Manager

- Has limited, finance-oriented CRM access
- Can:
    - View **Customers**
- Cannot:
    - Access Leads
    - Access Follow-ups
- **Leads and Follow-up modules are disabled**

Access Control Notes

- CRM sections must be **enabled or disabled individually** (Leads / Follow-ups / Customers) based on role
- Sales Manager should see **all CRM actions**

- View-only roles should not see create/edit/delete buttons
- Technician Manager should not see the CRM module at all
- Frontend must control visibility; backend will enforce final permissions

Leads

Postman : Enhanced ERP -> CRM -> Leads

Create Lead :

POST - /api/v1/leads/add
-> Sub Users can access which have permission to access this module.
-> Fill the form
-> ADMIN can also access but not recommended
-> Select Multiple Inventories Products that have demand from inventories drop down.
=> GET - /api/v2/inventory/dropdown (Enhanced ERP -> Inventory -> Product -> Get Product Drop Down)
-> Select Branch Name from Branch Drop Down, only one Branch.
=> GET - /branch/dropdown (Enhanced ERP -> Branch -> Branch Dropdown)
-> When Lead Status is Product, backend take productList and when service backend take serviceList.

Update Lead :

PUT - /api/v1/leads/review
-> Sub Users can access which have permission to access this module.
-> Change only lead status, and update lead
-> if you provide lead status lost, then add lost reason also.
-> ADMIN can also access but not recommended.
-> does not change leadType.
-> when lead status converted then with leadId open the customer details form with same details that provide with leads.
=> call Customer : POST - /api/v1/customers/add (Enhanced ERP -> CRM -> Customers) when lead status converted.
-> add additional details to customer and save it. Automatically specific lead status set converted.

Get All Leads :

GET - /api/v1/leads
-> Only Admin can access This APIs, It provides all branched leads.

Get All Leads Branch Wise :

GET - /api/v1/leads/branchWise
-> Only Sub Users can access who have permission to access this module.
-> In backend, it takes current user’s branch and show relative to its all leads details.

Get All Leads Drop Down Branch Wise :

GET - /api/v1/leads/dropdown

-> Only Sub Users can access who have permission to access this module.

-> In backend, it takes current user’s branch and show relative to its all leads Drop Down with Id and value(name of lead).

Get Lead By Id :

GET - /api/v1/leads/byId?id=8

-> Sub Users can access this APIs. ADMIN also can access but not recommended.

-> Pass lead id as a request parameter to get lead details.

Delete Lead By Id :

DELETE - /api/v1/leads?id=8

-> sub users can access this API who have permission to access this module.

-> Pass lead id as a request parameter to delete specific Lead.

Follow Ups

Postman : Enhanced ERP -> CRM -> Follow Ups Details

Create Follow Ups :

Post - /api/v1/followup/add

-> Sub Users can access this module who have permission to access this module.
-> ADMIN also can access, but not recommended.
-> fill the form.
-> Select Lead from Lead DropDown (GET - /api/v1/leads/dropdown , Postman : CRM -> Leads -> Lead Drop Down Branch Wise)
-> Select Quotation from Quotation DropDown (GET - DROP DOWN CREATION REMAIMING)
-> Select Branch from Branch DropDown (GET - /branch/dropdown, Postman : Enhanced ERP -> Branch -> Branch DropDown)

Get All Follow Ups :

GET - /api/v1/followup

-> ADMIN also can access, but not recommended.
-> it gives all follow ups details of whole company.

Get All Follow Ups Branch Wise :

GET - /api/v1/followup/branchWise

-> Sub Users can access this module who have permission to access this module.
-> it gives all follow ups details of specific Branch. It takes branch from currently logged in User's details.

Get Follow Ups By Id :

GET - /api/v1/followup/byId?id=1

-> Sub Users can access this module who have permission to access this module.
-> ADMIN also can access, but not recommended.
-> Pass follow up Id to request parameter to get follow ups details.

Delete Follow Ups By Id :

DELETE - /api/v1/followup?id=1

-> Sub Users can access this module who have permission to access this module.
-> ADMIN also can access, but not recommended.
-> Pass follow up Id to request parameter to delete follow ups details.

Update Follow Up By Id :

PUT - /api/v1/followup

-> Sub Users can access this module who have permission to access this module.
-> ADMIN also can access, but not recommended.
-> Can update all details of Follow up.
-> when status goes to lost , set lost reason also.
-> We store remarks for getting information about follow ups.

Customers

Postman : Enhanced ERP -> CRM -> Customer Details

Create Customer :

POST - /api/v1/customers/add

-> Sub Users can access this module who have permission to access this module.
-> ADMIN also can access, but not recommended.
-> Customer can create independently. Or from lead details (When lead converted).
-> Add Customer details into form.
-> when customerType is Product it takes only productList. When customerType is Service it takes only serviceList.
-> Select Multiple Inventories Products that have demand from inventories drop down.
=> GET - /api/v2/inventory/dropdown (Enhanced ERP -> Inventory -> Product -> Get Product Drop Down)
-> Select Branch Name from Branch Drop Down, only one Branch.
=> GET - /branch/dropdown (Enhanced ERP -> Branch -> Branch Dropdown)

Get Customer By Id :

GET - /api/v1/customers/byId?id=17

-> Sub Users can access this module who have permission to access this module.
-> ADMIN also can access, but not recommended.
-> Pass customer id as a request parameter to get specific customer details.

Delete Customer By Id :

DELETE - /api/v1/customers?id=1

-> Sub Users can access this module who have permission to access this module.
-> ADMIN also can access, but not recommended.
-> Pass customer id as a request parameter to delete specific customer details.

Update Customer By Id :

PUT - /api/v1/customers/byId/update?customerId=8

-> Sub Users can access this module who have permission to access this module.

-> ADMIN also can access, but not recommended.
-> does not change customertype. So assigned product and service can not changed. If first product assigned then we can update it. If service assigned first then you can update only services.
-> other details can be updated.

Get All Customers List :

GET - /api/v1/customers

-> ADMIN can only access this API.
-> it gives all customers list.

Get All Customer Branch Wise :

GET - /api/v1/customers/branchWise

-> Sub Users only can access this API who have permission to access this module.
-> It takes implicitly of current user logged in branch id and its customer details list.

Get Customer Drop Down Branch Wise :

GET - /api/v1/customers/dropdown

-> Sub Users only can access this API who have permission to access this module.
-> It takes implicitly of current user logged in branch id and its customer details drop down.
-> drop down list can be used by other forms.

Get Customer All History :

POST - /api/v1/customers/byId/track

-> Sub Users only can access this API who have permission to access this module.
-> Admin can also access, but not recommended.
-> It gives all services that consumes by specific customers.
-> In request , it takes customerId that mentioned in Postman collection.

Sales Management

Sales Management – Role-Based Access Description

The **Sales Management** module handles all sales-related activities and is divided into three core sections: **Quotations**, **Sales Orders**, and **AMC (Annual Maintenance Contracts)**. Access to each section is controlled based on user roles to ensure proper authorization, operational clarity, and data security.

Each role interacts with sales data according to its responsibility within the organization.

Role-Wise Sales Management Permissions

Company Admin

- Has **view-only access** across the Sales Management module
- Can:
    - View **Quotations**
    - View **Sales Orders**
    - View **AMC**
- Cannot:
    - Create, update, or delete any sales records

Branch Admin

- Has **full access** to Sales Management for their branch
- Can:

    - Create, view, update, and delete **Quotations**
    - Create, view, update, and delete **Sales Orders**
    - Create, view, update, and delete **AMC**
- Responsible for managing all sales activities at the branch level

---

Technician Manager

- Has **limited access** to Sales Management
- Can:
    - View **AMC** details
- Cannot:
    - Access **Quotations**
    - Access **Sales Orders**
- **Quotation and Sales Order modules are disabled**

Sales Manager

- Has **full access** to Sales Management
- Can:
    - Create, view, update, and delete **Quotations**

    - Create, view, update, and delete **Sales Orders**
    - Create, view, update, and delete **AMC**
- Manages the complete sales lifecycle

Account Manager

- Has restricted, finance-focused access
- Can:
    - View **Sales Orders**
- Cannot:
    - Access **Quotations**
    - Access **AMC**
- **Quotation and AMC modules are disabled**

Access Control Notes

- Each sales sub-module (**Quotation / Sales Orders / AMC**) must be enabled or disabled independently based on role
- Full-access roles should see all action buttons (Add / Edit / Delete)
- View-only roles should not see modification actions
- Disabled modules should not appear in navigation

- Frontend must enforce role-based visibility; backend will validate permissions

Quotation

Quotation Module

---

1. Module Overview

Purpose

The **EnhanceQuotation** module is the central engine for managing business proposals and quotations. It bridges the gap between the **Pre-Sales (Lead)** phase and the **Sales (Customer)** phase.

Real-World Problem Solved

In many ERPs, "Leads" and "Customers" are disconnected. This module solves that by:

1. Allowing quotations to be sent to **Leads** (prospects) *before* they become actual customers.

2. **Automating Conversion**: When a Lead accepts a quotation, the system automatically converts them into a **Customer**, creates their account, and moves the relationship forward without manual data entry.

System Usage

This module is used by Sales Representatives to:

- Generate formal price quotes for Products or Services.
- Track the status of these quotes (Sent, Accepted, Rejected).
- Define recurring billing terms (e.g., Monthly Maintenance Contracts).

---

2. Functional Responsibilities

The module handles the following core operations:

Quotation Creation

- Can be created for an existing **Lead** OR an existing **Customer**.
- **Validation**: A quotation can contain **Products** OR **Services**, but NEVER both in the same document.

Financial Calculation

- Auto-calculates Subtotal, Tax, Discount, and Grand Total.
- Supports different calculation logic for Products (Quantity * Price) vs. Services (Sqft based or fixed).

Status Management & Automation

- Tracks status flow:
  `DRAFT` → `SENT` → `ACCEPTED` → `CONVERTED`
- **CRITICAL TRIGGER**: Updating status triggers the **Lead-to-Customer conversion** process.

Recurring Quotations

- Handles setup for subscription-based quotes (Start Date, End Date, Intervals, Cycles).

---

3. Database Design

Primary Table: `enhance_quotation`

Stores the header information for the quotation.

Field Name	Data Type	Purpose	Relationship
`id`	BIGINT	Primary Key	

`lead_id`	BIGINT	Reference to the prospect (if not yet a customer)	FK to `leads`
`customer_id`	BIGINT	Reference to the existing customer (if applicable)	FK to `customer_details`
`branch_id`	BIGINT	Branch/Office user belongs to	FK to `branch`
`quotation_number`	VARCHAR	Unique ID (e.g., "QT-1001")	
`quotation_type`	VARCHAR	`PRODUCT` or `SERVICE`	
`service_type`	VARCHAR	`RESIDENTIAL` or `COMMERCIAL`	
`status`	VARCHAR	Current state (e.g., `ACCEPTED`)	
`sqft`	DECIMAL	Area size (Specific to Service quotes)	
`subtotal`	DECIMAL	Total before tax/discount	
`grand_total`	DECIMAL	Final payable amount	

`is_recurring`	BOOLEAN	Flag for subscription quotes
`recurring_type`	VARCHAR	`MONTHLY`, `YEARLY`

Mapping Tables (Line Items)

1. `quotation_product_mapper`

Links specific products and quantities to a quotation.

- **Fields**: `id`, `quotation_id`, `product_id`, `quantity`

2. `quotation_service_mapper`

Links specific services to a quotation.

- **Fields**: `id`, `quotation_id`, `service_id`

---

4. API Documentation

Based on `EnhanceQuotationController`, the following APIs are available:

---

1. Add Quotation

- **Endpoint**: `POST /api/v1/quotation/add`
- **Purpose**: Creates a new quotation.

Request Body (Important Fields)

- `leadId` OR `customerId`: **Mandatory** (One must be present).
- `products`: List of `{ productId, quantity }` (If `quotationType` is PRODUCT).
- `services`: List of `serviceIds` (If `quotationType` is SERVICE).
- `quotationType`: `"PRODUCT"` or `"SERVICE"`.
- `sqft`: Area coverage (Required for Service quotes).
- `financials`: `subtotal`, `taxAmount`, `grandTotal`.
- `recurring`: `isRecurring`, `startDate`, `endDate`.

Validations

- Throws `BadRequestException` if both `leadId` and `customerId` are missing.
- Throws `BadRequestException` if *both* products and services are provided (Exclusive Logic).

Response

- Returns the created `QuotationResponseDto` with the generated `quotationNumber`.

---

2. Update Status (Trigger)

- **Endpoint**: `PATCH /api/v1/quotation/update-status`
- **Purpose**: Updates the status and **triggers the Lead Conversion workflow** if conditions are met.

Request Body

- `id`: Quotation ID.

- `status`: New status string (e.g., `"ACCEPTED"`).

Internal Logic

- Updates the status in `enhance_quotation`.
- **Event Trigger**: If status changes and `leadId` is present, it publishes `QuotationAcceptedEvent`.

---

3. Get Quotation By ID

- **Endpoint**: `GET /api/v1/quotation/getById`
- **Request**: Body containing `{ "id": 123 }`.
- **Response**: Full details including mapped products **or** services.

---

4. Get All Quotations

- **Endpoint**: `GET /api/v1/quotation/getAll`
- **Response**: List of all quotations in the system.

---

5. Get Quotations By Branch (Branch Wise Fetch)

- **Endpoint**: `GET /api/v1/quotation/getByBranch` (Assumed/Planned)
- **Purpose**: Fetches all quotations belonging to a specific branch.
- **Request**: Body containing `{ "branchId": 45 }`.
- **Response**: List of `QuotationResponseDto` for that branch.

- **Logic**: Filters `enhance_quotation` table records where `branch_id` matches.

---

5. Module Workflow (End-to-End)

Scenario: Lead to Customer Conversion

1. **Request**: User maps a quotation to a **Lead** (e.g., `leadId: 50`) and sets items.
2. **Creation**: System validates exclusivity (Product vs Service), calculates totals (or accepts frontend totals), and saves to `enhance_quotation`. Status is `SENT`.
3. **Update**: User calls `/update-status` changing status to `ACCEPTED`.
4. **Event Firing**: `EnhanceQuotationServiceImpl` detects the change and fires `QuotationAcceptedEvent`.
5. **Event Handling (`CustomerDetailsServiceImpl`)**:
    - Listener catches the event.
    - Fetches the **Lead** data (Name, Phone, etc.).
    - **Checks**: Is Lead already `CONVERTED`? If yes, stops/errors.
    - **Action**: Creates a new entry in `customer_details` copying data from Lead and Quotation.
    - **Migration**: Copies Lead's Product/Service preferences to the Customer profile.
    - **Lead Update**: Sets Lead status to `CONVERTED`.
    - **Quotation Update**: Links the specific Quotation to the **New Customer ID**.

---

6. Dependencies & Integration

Lead Module (`LeadRepository`, `LeadService`)

- Source of truth for initial data.
- Updated upon conversion.

Customer Module (`CustomerDetailsRepository`, `CustomerDetailsService`)

- Destination for converted data.
- Contains the **Event Listener** logic (Lead Conversion).

Branch Module

- All quotations are linked to a specific Branch for data isolation.

---

7. Important Notes for Developers

⚠️ [WARNING] Product vs. Service Exclusivity

The system strictly enforces that a quotation cannot contain mixed items. It must be purely **Product-based** or **Service-based**. Ensure your frontend UI respects this rule to avoid `400 Bad Request` errors.

⚠️ [CAUTION] Status Trigger Logic

The current Event Trigger logic in `updateStatus` fires on **ANY** status change if the ID exists. It presumes the business process is "moving forward". Be careful if implementing "Backward" status transitions (e.g., `"Accepted"` → `"Draft"`).

ℹ️ [NOTE] Financial Calculations

While the backend has calculation utilities (`AmountCalculationUtil`), the `addQuotation` endpoint currently accepts financial totals (`grandTotal`, `taxAmount`) directly from the DTO. Ensure the Frontend calculation logic matches the backend expectations.

Sales oder

Sales oder

---

1. Module Overview

Purpose

The **SalesOrder** module serves as the **official confirmation of a sale**. It formalizes the transaction between the company and the **Customer**.

Real-World Business Problem Solved

- **Order Confirmation**: Converts a proposal (Quotation) or a direct request into a binding order.
- **Workflow Automation**: Acts as the bridge between Sales and Finance/Operations. Confirming an order automatically triggers **Invoice generation** and **AMC (Annual Maintenance Contract) creation**, reducing manual data entry and errors.

System Usage

- **User**: Sales Managers / Representatives.
- **Trigger**: Created when a Customer agrees to a Quotation or places a direct order.
- **Output**: Generates a confirmed Sales Order, a Draft Invoice, and optionally an AMC Contract.

---

2. Functional Responsibilities

The module handles the following core operations:

Customer-Centric Creation

- Sales Orders are **strictly mapped to a Customer ID**.

- If coming from a Lead, the Lead *must* be converted to a Customer (handled in the Quotation module) before a Sales Order can be created.

Status Lifecycle

`DRAFT` → `PENDING` → `CONFIRMED` → `CANCELLED`

Automatic Invoice Generation

- **Trigger**: When Status changes to `CONFIRMED`.
- **Action**: Creates a new Invoice Record.
- **Defaults**:
    - Invoice Status: `DRAFT`
    - Payment Status: `UNPAID`
- **Data Mapping**: Copies Customer details, Branch, Financials (Subtotal, Tax, Grand Total), and Line Items (Products/Services).

Automatic AMC Creation

- **Trigger**: When Status changes to `CONFIRMED` **AND** `is_amc` flag is true.
- **Logic**:
    - If `is_amc = true`: System reads AMC fields (e.g., Start Date, Frequency) from the request DTO and creates a new **Contract**.
    - If `is_amc = false`: No contract is created.

---

3. Database Design

Primary Table: `sales_orders`

Stores the order details.

Field Name	Data Type	Purpose	Relationship
`sales_order_number`	BIGINT	Primary Key (Auto Identity)	
`customer_id`	BIGINT	**Mandatory** reference to Customer	FK to `customer_details`
`quotation_id`	BIGINT	Linked Quotation (Optional)	FK to `enhance_quotation`
`branch_id`	BIGINT	Branch processing the order	FK to `branch`
`status`	VARCHAR	`CONFIRMED`, `DRAFT`, etc.	
`sales_order_type`	VARCHAR	`PRODUCT` or `SERVICE`	
`sqft`	DECIMAL	Area (for Service orders)	
`subtotal`	DECIMAL	Amount before tax	
`tax_amount`	DECIMAL	Tax calculated	
`grand_total`	DECIMAL	Final payable amount	
`created_at`	TIMESTAMP	Audit timestamp	

Mapping Tables

1. `sales_order_product_mapper`

- **Purpose**: Stores Product Line Items.
- **Fields**: `saled_order_id`, `product_id`, `quantity`.

2. `sales_order_service_mapper`

- **Purpose**: Stores Service Line Items.

- **Fields**: `sales_order_id`, `service_id`.

---

4. API Documentation

All APIs are prefixed with `/api/v1/sales-orders`.

---

1. Create Sales Order

- **Endpoint**: `POST /`
- **Method**: `POST`
- **Purpose**: Creates a new Sales Order.
- **Request Body**:
    - `customerId`: **Required**
    - `quotationId`: Optional
    - `soType`: `PRODUCT` or `SERVICE`
    - `salesOrderRequest`: List of Items
    - `financials`: `subtotal`, `grandTotal`, etc.
- **Logic**: Validates existence of Customer and Branch. Saves order in the database.

---

2. Update Sales Order (Confirmation Trigger)

- **Endpoint**: `PUT /{id}`
- **Method**: `PUT`

- **Purpose**: Updates order details and **processes Confirmation logic**.
- **Request Body**:
    - `status`: **Crucial**. Set to `CONFIRMED` to trigger workflows.
    - `isAmc`: Boolean flag for AMC creation.
    - `amcDetails`: (If `isAmc` is true) `{ startDate, frequency }`

Business Logic

1. Updates the `SalesOrder` entity.
2. **IF Status becomes `CONFIRMED`**:
    - **Invoice**: Checks if exists → If not, creates new Invoice (Status: `DRAFT`).
    - **AMC**: Checks `isAmc` → If true, creates `Contract` entity using AMC details.

---

3. Get By ID

- **Endpoint**: `GET /{id}`
- **Method**: `GET`
- **Response**: Returns full Sales Order details including Products/Services lists.

---

4. Get All

- **Endpoint**: `GET /`
- **Method**: `GET`
- **Response**: Lists all Sales Orders.

---

5. Get Sales Orders By Branch

- **Endpoint**: `GET /branch/{branchId}`
- **Method**: `GET`
- **Purpose**: Fetches all Sales Orders belonging to a specific branch.
- **Request**: Path Variable `branchId`.
- **Response**: List of Sales Order DTOs for that branch.
- **Logic**: Filters `sales_orders` table records where `branch_id` matches.

---

6. Delete Sales Order

- **Endpoint**: `DELETE /{id}`
- **Method**: `DELETE`
- **Purpose**: Removes a Sales Order.
- **Validation**: Should ideally check if Linked Invoices exist (Dependency).

---

5. Module Workflow (End-to-End)

Step 1: Creation

- Sales Rep sends a `POST` request with `customerId` and items.
- System validates data and creates a `SalesOrder` record with status `DRAFT` (or as sent).

Step 2: Confirmation & Invoice Auto-Creation

- Authorized user sends a `PUT` request with `status: "CONFIRMED"`.
- **System Action**:
    - Updates Order Status.
    - **Invoice Service**: Creates a corresponding **Invoice**.
        - Invoice Status: `DRAFT`
        - Common fields (Customer, Branch, Totals) are copied.
    - **Payment Service**: Creates a `PENDING` payment record linked to the Invoice.

Step 3: AMC Auto-Creation (Conditional)

- During the same `PUT` (Confirmation) request:
    - System checks the `isAmc` boolean in the DTO.

Case A (`isAmc = true`)

- System reads `amcStartDate` and `amcEndDate` from request.
- Calls `ContractService` or `AmcRepository` to create a new Contract.

Case B (`isAmc = false`)

- Skips Contract creation.

Step 4: Completion

- The Sales Order is now `CONFIRMED`.
- Finance team takes over the **Draft Invoice**.

- Support team takes over the **AMC Contract** (if created).

---

6. Dependency & Integration

- **Customer Module**: Core dependency. Orders cannot exist without a valid Customer ID.
- **Invoice Module**: Downstream. Sales Order pushes data *to* Invoice tables (`invoice_master`).
- **Contract (AMC) Module**: Downstream. Sales Order triggers creation of Contracts.
- **Payment Module**: Sales Order triggers initialization of Payment records.

---

7. Important Notes for Developers

- **Customer ID is Mandatory**: Unlike Quotations (which can be for Leads), Sales Orders are strictly for Customers.
- **Invoice logic is Idempotent**: The system checks if an invoice already exists for the `salesOrderId` before creating a new one. This prevents duplicate invoices if the `CONFIRMED` status is sent multiple times.
- **AMC Logic relies on DTO**: The `isAmc` flag and details must be present in the `Update` request DTO to trigger contract creation.
- **Data Consistency**: Ensure that `Grand Total` on the Sales Order matches the generated Invoice amount exactly.

AMC

AMC (Annual Maintenance Contract)

---

1. Module Overview

Purpose of the Module

The AMC module is designed to manage long-term service commitments after a sales agreement. It automates the lifecycle of Annual Maintenance Contracts, ensuring that recurring services and billing are handled systematically without manual intervention.

Real-world Business Problem It Solves

In service industries, tracking when a customer is due for their next service or invoice is error-prone when done manually. The AMC module solves:

- **Missed Revenue**: Automatically generates invoices on scheduled dates.
- **Service Consistency**: Ensures service cycles are tracked and executed as per the contract.
- **Contract Tracking**: Provides visibility into the remaining value and cycles of a contract.

How This Module is Used in the System

The module starts with a **Sales Order**. If a Sales Order is flagged as an AMC, the system automatically creates an AMC record. This record then resides in the background, monitored by a daily **Cron Job** that triggers billing cycles based on the defined frequency (`MONTHLY`, `QUARTERLY`, `YEARLY`).

---

2. Functional Responsibilities

- **Automated Creation**: Creates an AMC record when a Sales Order is saved with `isAmc = true`.

- **Cycle Calculation**: Automatically calculates total service cycles and total contract value based on contract duration and recurring interval.
- **Background Execution**: A scheduler runs daily to identify AMCs due for the next execution cycle.
- **Invoice Generation**: Automatically creates a Proforma or Final Invoice when a cycle is triggered.
- **Status Management**: Tracks the AMC lifecycle through `DRAFT`, `ACTIVE`, `PAUSED`, `TERMINATED`, and `COMPLETED`.

---

3. Database Design

Primary Table

`amc_contract`
(Model: `Amc.java`)

Field Name	Type	Purpose
`amc_id`	PK (Long)	Unique Identifier
`amc_number`	String	Auto-generated unique number (e.g., AMC001)
`amc_status`	String	`DRAFT`, `ACTIVE`, `PAUSED`, `TERMINATED`, `COMPLETED`
`recurring_type`	String	`MONTHLY`, `QUARTERLY`, `YEARLY`
`recurring_interval`	Integer	Frequency interval from frontend request
`total_cycle`	Integer	Total service cycles in contract
`complete_cycle`	Integer	Cycles already completed
`remain_cycle`	Integer	Cycles remaining

`contract_start_date`	Date	Contract start date
`contract_end_date`	Date	Contract end date
`next_invoice_date`	Date	Next scheduled execution date
`last_invoice_date`	Date	Last executed cycle date
`per_cycle_amount`	Decimal	Amount per cycle
`contract_total_value`	Decimal	`per_cycle_amount × total_cycle`
`customer_id`	FK	Customer reference
`sales_order_id`	FK	Sales Order reference
`branch_id`	FK	Branch reference

Tracking Table

`amc_tracking`
(Model: `AmcTracking.java`)

Used to track execution of each AMC service cycle.

Field Name	Type	Purpose
`amc_tracking_id`	PK (Long)	Unique tracking ID
`amc_id`	FK (Long)	Parent AMC reference
`invoice_id`	FK (Long)	Generated Invoice ID
`invoice_create_date`	Date	Invoice draft date

`invoice_confirm_date`	Date	Invoice confirmation date
`invoice_amount`	Decimal	Invoice value
`payment_id`	FK (Long)	Payment record reference
`payment_create_date`	Date	Payment creation date
`payment_paid_date`	Date	Actual paid date
`payment_amount`	Decimal	Payment amount
`payment_status`	String	`PENDING`, `COMPLETED`, etc.
`payment_method`	String	Cash, Online, etc.
`payment_delay_days`	Long	Delay between invoice and payment
`receipt_id`	FK (Long)	Final receipt reference

---

4. API Documentation

Controllers: `AmcController`, `AmcTrackingController`

1. Update AMC Status

- **Endpoint**: `PUT /api/v1/amc/update`
- **Purpose**: Pause, terminate, or resume an AMC
- **Request Body (`AmcRequestDto`)**:
    - `amcId`
    - `amcStatus`

    - `pauseReason` / `terminationReason`

- **Internal Logic**: Updates AMC status. `PAUSED` records are skipped by the scheduler.

---

2. Get All AMCs

- **Endpoint**: `GET /api/v1/amcs`
- **Purpose**: Fetch all AMC records (Admin view)
- **Response**: `ResultDto<AmcResponseDto>`

---

3. Get AMC by ID

- **Endpoint**: `GET /api/v1/amc`
- **Parameters**: `amcId` (Request Parameter)
- **Purpose**: Fetch complete AMC details

---

4. Get Branch-wise AMCs

- **Endpoint**: `GET /api/v1/amcs/getAll/branchWise`
- **Purpose**: Fetch AMCs scoped to the current user's branch

---

5. Get AMC Tracking History

- **Endpoint**: `GET /api/v1/amc/{amcId}/tracking`

- **Purpose**: Fetch full execution timeline for an AMC
- **Logic**:
    - Queries `amc_tracking` by `amcId`
    - Ordered by `invoiceCreateDate` (ASC)
- **Response**: `ResultDto<AmcTrackingResponseDto>`

---

6. Manual Cron Trigger (Dev Only)

- **Endpoint**: `POST /api/v1/dev/cron/run-amc`
- **Purpose**: Manually triggers AMC scheduler for testing

---

5. Module Workflow (End-to-End)

1. **Trigger Phase**
   Sales Order created with `isAmc = true`
2. **Creation Phase**
    - Generates `amc_number`
    - Sets status to `DRAFT`
    - Maps customer, branch, and sales order
3. **Calculation Phase**
    - Calculates `total_cycle`
    - Calculates `contract_total_value`

4. **Scheduler Phase (Daily 01:00 AM)**
    - Identifies AMCs where:
        - `next_invoice_date == today`
        - Status is `ACTIVE` or initial `DRAFT`
5. **Execution Phase (AmcTxService)**
    - Invoice creation (DRAFT)
    - Tracking record creation
    - `next_invoice_date` moved forward
    - **CRITICAL**: AMC counters are NOT updated here
6. **Payment Phase**
    - On payment `PAID`:
        - Increment `complete_cycle`
        - Decrement `remain_cycle`
        - Update total completed and remaining amounts
        - Finalize tracking entry
7. **Finalization**
    - If `remain_cycle == 0`, AMC status becomes `COMPLETED`

---

6. Dependency & Integration

- **Sales Order Module**: Source for AMC creation

- **Invoice Module**: Generates billing documents
- **Security Context**: Used for branch-wise filtering

---

7. Important Notes for Developers

- **Date Accuracy**: Scheduler relies on `LocalDate.now()`
- **Draft to Active Transition**: Happens on `contractStartDate`
- **Deferred Accounting Rule**: AMC counters update only after payment, not invoice creation
- **Auto-Generate Flag**: Invoices are generated only if `autoGenerateInvoice == true`

Billing & Finance

Billing & Finance – Role-Based Access Description

The **Billing & Finance** module manages all financial transactions and records within the system. It is divided into three core sections: **Invoices**, **Payments**, and **Receipts**. Due to the sensitive nature of financial data, access to this module is strictly controlled and granted only to authorized roles.

Each role is provided access based on its financial responsibility and involvement in revenue management.

Role-Wise Billing & Finance Permissions

Company Admin

- Has **view-only access** to financial data
- Can:
    - View **Invoices**
    - View **Payments**
    - View **Receipts**
- Cannot:
    - Create, update, or delete any financial records

Branch Admin

- Has **view-only access** to Billing & Finance
- Can:

    - View **Invoices**
    - View **Payments**
    - View **Receipts**
- Cannot:
    - Modify or manage financial records

Account Manager

- Has **full access** to the Billing & Finance module
- Can:
    - Create, view, update, and delete **Invoices**
    - Create, view, update, and delete **Payments**
    - Create, view, update, and delete **Receipts**
- Responsible for complete financial operations

Technician Manager & Sales Manager

- Have **no access** to the Billing & Finance module
- Invoices, Payments, and Receipts should be:
    - Hidden

    - Disabled

Access Control Notes

- Financial actions (Add / Edit / Delete) must be visible **only** to Account Manager
- View-only roles should not see action buttons
- Disabled roles should not see Billing & Finance in navigation
- Frontend must enforce role-based access; backend will validate permissions

Invoices

Invoice Module

---

1. Module Overview

Purpose of the Module

The Invoice module is responsible for the final billing phase of the ERP system. It serves as the primary document for revenue recognition and payment collection, bridging the gap between service/product delivery and financial accounting.

Real-world Business Problem It Solves

- **Inaccurate Billing: Ensures that every Sales Order or service provided under AMC is billed correctly based on agreed prices.**
- **Workflow Automation: Eliminates manual invoice entry by pulling direct data from Sales Orders and Background Schedulers.**
- **Payment Lifecycle Tracking: Integrates with the Payment module to track which invoices are `PAID`, `UNPAID`, or `PARTIALLY PAID`.**

How This Module is Used in the System

Invoices are the culmination of the sales workflow. Every invoice tracks customer details, service categories (Service or Product), taxes, and grand totals. Once an invoice is confirmed, it triggers ledger entries and payment records.

---

2. Functional Responsibilities

- **Multi-Source Generation: Handles invoice creation from three distinct triggers:**
  - **Manual API**
  - **Sales Order status change**
  - **AMC Cron Job**
- **Status Management: Manages invoices through states:**
  - `DRAFT`
  - `CONFIRM`
  - `CANCELLED`

- **Financial Accounting: Calculates:**
    - `Grand Total`
    - `Tax Amount`
    - `Discount Amount`
    - `Balance Amount`
- **Integration Hooks:**
    - **Updates `CustomerDetails` (Total Invoices count)**
    - **Triggers `AmcTracking` updates upon confirmation**

---

3. Database Design

Primary Table: `invoices`

Model: `Invoice.java`

Field Name	Type	Purpose
`id`	PK (Long)	Unique Identifier for the invoice record.
`invoice_number`	String	Unique generated number (e.g., INV001).
`customer_id`	Long	Source customer from `CustomerDetails`.
`sales_order_id`	Long	Reference to the originating Sales Order.
`status`	Enum	Current stage: `DRAFT`, `CONFIRM`, `CANCELLED`.
`payment_status`	String	Tracks collection: `UNPAID`, `PAID`, `PARTIAL`.

`service_category`	Enum	Type of service (e.g., General, AMC-linked).
`invoice_is_for`	Enum	Distinguishes between `SERVICE` and `PRODUCT`.
`grand_total`	Decimal	Final billing amount after taxes and discounts.
`amount_paid`	Decimal	Total amount received against this invoice.
`balance_amount`	Decimal	Remaining amount to be collected.
`sqft`	Decimal	Measurements used for service-based pricing.
`branch_id`	FK (Long)	Linking to the operating `Branch`.

---

4. API Documentation

Controller

- `InvoiceOrderController`

4.1 Manual Invoice Creation

- **Endpoint:** `POST /api/v1/invoices/add`
- **Purpose: Direct manual invoice generation via UI.**
- **Logic:**
    - **Generates a new `invoice_number`**
    - **Sets invoice status to `DRAFT`**

4.2 Update / Confirm Invoice

- **Endpoint:** `POST /api/v1/invoices/update`

- **Purpose: Update invoice details or change status to `CONFIRM`.**

Business Logic

- **If status changes to `CONFIRM`:**
    - **Publishes an `InvoiceConfirmedEvent`**
    - **Automatically creates a `Payment` record in `PENDING` state**
    - **Updates `AmcTracking` if the invoice was generated via AMC**

4.3 Fetch Invoices

Admin Access

- **`GET /api/v1/invoices/all` — Fetch all records**
- **`GET /api/v1/invoices/{id}` — Fetch specific record details**

Branch-wise Scoped Access

- `GET /api/v1/invoices/getAll/branchWise`
    - Primary endpoint for branch staff
    - Uses the logged-in user's security token to identify branch
    - Returns only branch-specific invoices

4.4 Delete Invoice

- Endpoint: `DELETE /api/v1/invoices/{id}`
- Purpose: Removes an invoice record from the system.

---

5. Module Workflow (End-to-End)

The Invoice module supports three distinct entry methods.

Method 1: Manual API Generation

1. User submits `InvoiceRequestDto` via `/api/v1/invoices/add`.

2. `InvoiceServiceImplement` validates the branch and customer.

3. A `DRAFT` invoice is created with a unique sequence number.

Method 2: Sales Order Status Trigger

1. `SalesOrder` status is updated to `CONFIRMED` (and it is NOT an AMC).

2. `SalesOrderServiceImpl.addOrUpdateInvoice` is automatically triggered.

3. System fetches an existing invoice for that Sales Order or creates a new one.

4. Financial totals (Subtotal, Tax, Grand Total) are mapped directly from the Sales Order.

Method 3: AMC Scheduler Trigger

1. `AmcScheduler` runs daily and identifies active AMCs due for service.

2. `AmcTxService` calls `invoiceService.createFromAmc(amc)`.

3. System generates a `SERVICE` type invoice using the AMC's `perCycleAmount`.

4. An `AmcTracking` entry is created to link the invoice to the specific contract cycle.

---

6. Dependency & Integration

- Amc Module: Provides recurring billing data for the scheduler-based invoice generation.

- Sales Order Module: Acts as the source for one-time product and service invoices.

- Payment Module: Invoice confirmation automatically generates a `PENDING` payment.

- **Customer Module: Automatically increments the `totalInvoices` field in the customer profile.**

- **Inventory Module: Indirectly linked via Sales Order line items that define invoice totals.**

---

7. Important Notes for Developers

- **Confirmation Logic:**
  Never manually create a Payment record. Always update Invoice status to `CONFIRM`, which triggers `addOrUpdatePayment`.

- **AMC Detection:**
  An invoice is identified as AMC-related by checking the `amc_tracking` table using `invoice_id`.

- **Branch Context Security:**
  All data retrieval uses the current user's branch ID from `UserIdentity` to prevent cross-branch access.

- **Grand Total Consistency:**
  Always ensure:
  `Grand Total = Subtotal + Tax - Discount`

- **Branch Scope Enforcement:**
  All production-facing GET APIs must use branch-wise filtering to maintain data privacy.

Payments

Payment

---

1. Module Overview

Purpose of the Module

The Payment module manages the financial collection process within the ERP. It handles the transition of an outstanding Invoice into a settled transaction, ensuring that revenue is accurately tracked, receipts are generated, and contract balances are updated.

Real-world Business Problem It Solves

- **Reconciliation Issues**: Tracks which invoices are settled and which are pending, preventing revenue leakage.
- **Service-Payment Lag**: Specifically tracks *Payment Delay Days* for AMC services to analyze customer payment behavior.
- **Financial Document Automation**: Automatically generates a formal Receipt once a payment is confirmed as `PAID`.

How This Module is Used in the System

Payments are typically initialized automatically when an Invoice is confirmed. The user then updates the Payment record with the actual amount received, the payment method (Cash, Online, etc.), and the transaction reference.

---

2. Functional Responsibilities

- **Lifecycle Settlement**: Transitions an invoice from `UNPAID` to `PAID`.
- **Receipt Automation**: Automatically triggers the creation of a `Receipt` record upon successful payment confirmation.

- **AMC Financial Sync**: Updates the `Total Complete Amount` and `Total Remain Amount` in the AMC module when a cycle's payment is completed.
- **Ledger Integration**: Triggers ledger entries based on the payment event for accounting accuracy.
- **Performance Tracking**: Calculates the number of days between invoice confirmation and payment receipt (`paymentDelayDays`).

---

3. Database Design

Primary Table: `payments`

Model: `Payment.java`

Field Name	Type	Purpose
`id`	PK (Long)	Unique Identifier for the payment record.
`invoice_id`	Long	Reference to the Invoice being settled.
`customer_id`	Long	Reference to the Customer.
`invoice_amount`	Decimal	The original amount billed.
`amount_paid`	Decimal	The amount received in the current transaction.
`total_paid_till_now`	Decimal	Cumulative amount paid towards the invoice.
`balance_amount`	Decimal	Remaining amount due on the invoice.
`payment_status`	Enum	`PENDING`, `PARTIAL`, `PAID`, `CANCELLED`.
`payment_method`	Enum	`CASH`, `CHEQUE`, `ONLINE`, `BANK_TRANSFER`.
`transaction_reference`	String	UTR number, Cheque number, or Transaction ID.
`payment_date`	Timestamp	Actual date the payment was received.

`branch_id`	FK (Long)	Branch scoping for financial reporting.

---

4. API Documentation

Controller

- `PaymentController`

4.1 Update Payment (Settle Transaction)

- **Endpoint**: `PUT /api/v1/payments/{id}`
- **Purpose**: Record receipt of funds and settle the invoice.

Business Logic

- Updates payment status to `PAID`
- Updates the linked Invoice status to `PAID`
- Calls `ledgerEntryService.createLedgerBasedOnInvoicePaid(invoice)`
- Triggers `addReceipt(savedPayment)` to generate a receipt
- If linked to an AMC, triggers `updateAmcAmounts`

4.2 Get Payment Details

Standard Fetch

- `GET /api/v1/payments/{id}` — Fetch specific payment details

- `GET /api/v1/payments/invoice/{invoiceId}` — Fetch all payment attempts/history for a specific invoice

Branch-wise Scoped Fetch

- `GET /api/v1/payments/getAll/branchWise`
    - Primary endpoint for branch-level financial tracking
    - Filters results based on the logged-in user's branch ID

4.3 Delete Payment

- **Endpoint**: `DELETE /api/v1/payments/{id}`
- **Note**: Generally restricted to Admin roles to preserve financial audit trails.

---

5. Module Workflow (End-to-End Flow)

1. **Generation Phase**
   A `Payment` record is automatically created in `PENDING` status when an Invoice is updated to `CONFIRM` (via `InvoiceService`).
2. **Collection Phase**
   The user receives payment from the customer and calls `PUT /api/v1/payments/{id}` with payment details.
3. **Settlement Logic (`PaymentServiceImpl`)**
    - When status is set to `PAID`:
        - **Step 3.1 (Invoice Sync)**: Linked Invoice is marked as `PAID`.

        - **Step 3.2 (Receipt Creation)**: A new Receipt record is generated with a unique `RCT` number.
        - **Step 3.3 (AMC Integration)**:
            - `totalCompleteAmount` is increased
            - `totalRemainAmount` is decreased
        - **Step 3.4 (Tracking Update)**:
          `AmcTracking` is updated with:
            - `paymentPaidDate`
            - `receiptId`
            - `paymentDelayDays`

4. **Accounting Phase**
   Ledger entries are automatically posted to reflect:
    - Cash/Bank inflow
    - Reduction in Accounts Receivable

---

6. Dependency & Integration

- **Invoice Module**: Payments cannot exist without a parent Invoice.
- **Amc Module**: AMC cycle payments update contract-level financial balances.
- **Receipt Module**: Payment module acts as the factory for Receipt generation.
- **Ledger Module**: Ensures all payment events are reflected in accounting records.

---

7. Important Notes for Developers

- **AMC Payment Trigger**:
  `updateAmcAmounts(invoice)` is the **only** point where AMC financial balances are modified after contract creation.

- **Delay Calculation**:
  Calculated as the difference between:

  - `invoice_confirm_date`

  - `payment_paid_date`
    Ensure both are present to avoid `NullPointerException`.

- **Branch-Based Filtering**:
  All payment queries are scoped using `branch_id` from the user session.

- **Automatic Sync Warning**:
  Setting payment status to `PAID` is a **one-way trigger** affecting:

  - Invoice

  - Receipt

  - Ledger

  - AMC
    Use caution with bulk or rollback operations.

Receipts

Receipt Module

---

1. Module Overview

1.1 Purpose of the Module

The Receipt module is the final documentation stage of a financial transaction. It represents the official proof of payment issued to a customer after a payment record has been marked as settled (PAID).

1.2 Real-world Business Problem It Solves

- **Audit Compliance**: Provides a permanent, non-editable record of funds received.
- **Customer Disputes**: Acts as legal proof that a specific invoice cycle (Regular or AMC) has been fully or partially paid.
- **Reference Integrity**: Links the received cash/online transfer back to the specific Payment ID and Invoice ID for zero-error accounting.

1.3 How This Module is Used in the System

Receipts are mostly generated automatically by the system when a `Payment` is updated to the `PAID` status. However, the module also provides manual APIs to create or update receipts if corrections are needed.

---

2. Functional Responsibilities

- **Proof of Payment**
  Generates a unique `RCT` prefixed number for every successful transaction.
- **Cross-Module Linking**
  Maintains a mandatory 3-way link between `PaymentId`, `InvoiceId`, and `CustomerId`.

- **AMC History Tracking**
  Specifically updates the `AmcTracking` table with the `receiptId` when the payment belongs to an AMC cycle.

- **Branch-Level Scoping**
  Ensures receipts are segregated by branch for localized financial reporting.

---

3. Database Design

3.1 Primary Table

Table Name: `receipts`
Entity / Model: `Receipt.java`

Field Name	Type	Purpose
`id`	PK (Long)	Unique Identifier for the receipt record.
`receipt_number`	String	Unique auto-generated number (e.g., RCT001).
`payment_id`	FK (Long)	The source Payment record ID.
`invoice_id`	FK (Long)	The Invoice ID the payment was made against.
`customer_id`	FK (Long)	The Customer who made the payment.
`amount_received`	Decimal	Total amount recorded in this specific receipt.
`payment_method`	Enum	The method used (CASH, ONLINE, etc.).
`receipt_date`	Timestamp	Date and time the receipt was generated (default: now).
`branch_id`	FK (Long)	Branch identifying where the funds were received.

---

4. API Documentation

4.1 Controller

Controller Name: `ReceiptController`

4.2 Create Receipt

- **Endpoint**: `POST /api/v1/receipts`
- **HTTP Method**: `POST`
- **Purpose**: Manual generation of a receipt (used if auto-trigger fails or manual override is needed).
- **Validations**:
    - `paymentId` must be valid
    - `invoiceId` must be valid
    - `customerId` must be valid
    - `amountReceived` must be greater than 0

4.3 Update Receipt

- **Endpoint**: `PUT /api/v1/receipts/{id}`
- **Purpose**: To update notes or correction in details.

4.4 Fetch Receipts

Available Endpoints

- `GET /api/v1/receipts`
  Fetch all receipts.

- `GET /api/v1/receipts/{id}`
  Detailed view of a single receipt.

Branch-wise Scoped Fetch

- `GET /api/v1/receipts/getAll/branchWise`
  Main endpoint for branch users to see their collection history.

4.5 Delete Receipt

- **Endpoint**: `DELETE /api/v1/receipts/{id}`

---

5. Module Workflow (End-to-End Flow)

1. **Trigger Phase**
   The process typically starts in the **Payment Module**. When `PaymentServiceImpl.updatePayment` sees a transition to `PAID`.

2. **Generation Phase (`addReceipt`)**
   - The system creates a new `Receipt` entity.
   - Uses `NumberGeneratorUtil` to create a unique `RCT` number based on the current count.
   - Automatically pulls `amountPaid` and `paymentMethod` from the `Payment` record.

3. **Cross-Update Phase**
   - If the payment is linked to an AMC, the system finds the corresponding `AmcTracking` entry.

    - The `receiptId` and `paymentPaidDate` are updated in the tracking record.
    - This allows the AMC tracking screen to show the exact receipt issued for each service cycle.

4. **Completion Phase**
    - The record is saved.
    - A response is returned.
    - Typically triggers the UI to allow PDF download of the receipt.

---

6. Dependency & Integration

- **Payment Module**
  The primary trigger for receipt creation.
- **Amc Module**
  Integrated via the tracking system to provide a paper trail for service cycles.
- **Invoice Module**
  References the invoice for which the proof of payment is issued.
- **Branch Module**
  Segregates collections for branch-wise audit purposes.

---

7. Important Notes for Developers

- **Auto-Generation Logic**
  Most of the receipt logic is technically housed in `PaymentServiceImpl.addReceipt` but managed via the `ReceiptRepository`.
- **Audit Lock**
  Receipts should ideally not be editable or deletable once sent to the customer, although the controller currently allows these operations for administrative flexibility.

- **Validation**
  The `ReceiptServiceImpl` has a strict internal `validateRequest` method that ensures no receipt is created without a positive amount or missing primary keys.

Task Management

Postman : Task Technician Management -> Task Management

Get All Tasks with Minimal Information : (not useful)

GET - /api/v1/tasks

-> sub users can access who have permission from the task module.
-> ADMIN can also be accessed, but not recommended.
-> It gives only taskId, status, assignedDate, serviceLocation, TaskName.

Get All Task Details Branch Wise User

GET - /api/v1/task/all

-> except ADMIN, all users can access who have permission from the task module.
-> it gives full details of the task if assigned, technician details, material details, etc.

Get Task By Id for Task Details View (Profile)

GET - /api/v1/task/byId?taskId=3

-> sub users can access who have permission from the task module.
-> ADMIN can also be accessed, but not recommended.
-> It gives full details of specific tasks with technician details, material details, images, etc.
-> It requires a task id as a request parameter.

Task Material List Based On TaskId

GET - /api/v1/task/material?taskId=12

-> sub users can access who have permission from the task module.
-> ADMIN can also be accessed, but not recommended.
-> It provided a list useful in task assigned form. Append material list to form data.
-> it requires a task id as a request parameter.

Task Not Assigned Details

GET - /api/v1/task/assign

-> only sub users can access who have permission from the task module.
-> It provides data for specific branches that have not assigned tasks.
-> it provides full task details.
-> from this, we assigned a task to a technician.
-> specific row, attach to task assigned form with taskId.

Assign Task To Technician

PUT - /api/v1/task

-> sub users can access who have permission from the task module.
-> ADMIN can also be accessed, but not recommended.
-> when this form open
    => Get Task By Id hit and fill the form with its details.
    => Task Material List Based on TaskId hit and append its details.
    => This form have technician drop down (GET - /api/v1/user/technician/dropdown, Postman : Enhanced ERP -> User -> Technician Drop Down)
-> FIll the form and assign the task to a particular technician.

Task Management – Role-Based Access Description

The **Task Management** module is used to plan, assign, and track tasks across the organization. The primary actions in this module are **assigning tasks** and **updating task status or details**. Access is controlled based on user roles to ensure tasks are managed only by authorized personnel.

Each role interacts with tasks according to its operational responsibility.

Role-Wise Task Management Permissions

Company Admin

- Has **view-only access** to the Task Management module
- Can:
    - View all tasks across branches
- Cannot:
    - Assign tasks
    - Update task details or status

Branch Admin

- Has **full access** to Task Management within their branch
- Can:
    - Assign tasks to users
    - Update task details and status
- Responsible for branch-level task coordination

Technician Manager

- Has **full access** to Task Management
- Can:
    - Assign tasks to technicians
    - Update task details and status
- Manages task execution at the technician level

Sales Manager & Account Manager

- Have **no access** to the Task Management module
- Task Management should be:
    - Hidden
    - Disabled
      for these roles

Access Control Notes

- Task action buttons (Assign / Update) must be visible **only** to Branch Admin and Technician Manager
- Company Admin should see tasks in **read-only mode**
- Disabled roles should not see the Task Management module in navigation

- Frontend must enforce role-based visibility; backend will validate permissions

Live Tracking, (Technician Tracking)

Postman : Enhanced ERP -> Technician Tracking

Track Technician Location :
POST - /api/v1/track/update

-> this is for only technician users.
-> it updated automatically form application side. Every 15 minutes, technician location update.
-> for track of every technician where is he.

Get Technicain Location :
GET - /api/v1/track?technicianId=6

-> This is for only TECHNICIAN MANAGER
-> he can show where specific technician.
-> We have one Technician dashboard, with each row it attach to google form with given location for tracking technicnai location.
-> Required technianId as a request parameter.

Live Tracking – Role-Based Access Description

The **Live Tracking** module provides real-time location visibility of technicians to support operational monitoring, task coordination, and service efficiency. Access to live tracking data is strictly controlled based on user roles to ensure privacy and proper usage of location information.

Each role can track technicians only within its permitted scope.

Role-Wise Live Tracking Permissions

Company Admin

- Has **full visibility** of live tracking across the organization

- Can:
    - View real-time location of **all technicians**
    - Track technicians across **all branches**
- Uses this access for centralized monitoring and performance oversight

---

Branch Admin

- Has branch-level live tracking access
- Can:
    - Track the real-time location of **all technicians within their branch**
- Cannot track technicians from other branches

Technician Manager

- Has branch-level tracking access
- Can:
    - Track the live location of **technicians under their management**
    - Monitor technician movement and task execution within the branch

---

Sales Manager & Account Manager

- Have **no access** to the Live Tracking module
- Live Tracking should be:
    - Hidden
    - Disabled
      for these roles

Access Control Notes

- Live tracking data must be filtered based on **branch and manager scope**
- Only authorized roles should see the tracking map and technician list
- Disabled roles should not see Live Tracking in navigation
- Frontend must enforce role-based visibility; backend will validate permissions

Performacae Management

Postman : Task Technician Management -> Task Management

Technician Performance Report

GET - /api/v1/task/performance/reports

-> sub users can access who have permission of task module.
-> ADMIN can also access, but not recommended.
-> It provides each technician performance details.
-> witht overall rating, total task completed, ranking, technician name, etc.

Performance Management – Role-Based Access Description

The **Performance Management** module provides insights into technician productivity and operational efficiency through structured performance reports. These reports are used for monitoring, evaluation, and decision-making, and access is strictly controlled to ensure data is visible only to authorized roles.

The module focuses on **technician performance reports**, with visibility limited by branch and managerial scope.

Role-Wise Performance Management Permissions

Company Admin

- Has **view-only access** to performance reports
- Can:
    - View **technician performance reports branch-wise** across the entire company
- Uses this data for organization-wide performance monitoring

---

Branch Admin

- Has **view-only access** to performance reports for their branch
- Can:
    - View **technician performance reports** for technicians within their branch

---

Technician Manager

- Has **view-only access** to performance reports
- Can:
    - View **technician performance reports** for technicians under their management within the branch

---

Sales Manager & Account Manager

- Have **no access** to the Performance Management module
- Performance reports should be:
    - Hidden
    - Disabled
      for these roles

Access Control Notes

- All access in this module is **read-only**
- Reports must be filtered by **branch and manager scope**
- Disabled roles should not see Performance Management in navigation
- Frontend must enforce role-based visibility; backend will validate permissions

Customer Support

Postman : Task Technician Management -> Customer Support

Customer Support : Ticket Management :

Access ONLY TECHNICIAN MANAGER

Create Ticket / Raise Ticket :
POST - /api/tickets/Create

-> Raise ticket against specific task which got issue.
-> customer call here, technician manager create raise according to issue.
-> with this technician manager, add new technician, new assigned date and time with to task.

Delete Ticket By Id :
DELETE - /api/tickets?ticketId=3

-> for deleting ticket , we require ticketId as a request parameter.

Ticker Filter on Dashboard :
POST - /api/tickets/search

-> for filtering data of ticket, we use this API.
Request :

```
{
  "filters": {
    "customerId": 1,
    "priority": "HIGH",
    "ticketStatus": "OPEN"
  },
  "sortBy": "id",
  "sortDirection": "DESC",
  "page": 0,
  "size": 10
}
```

Get ALL Tickets :
GET - /api/tickets

-> Get all Tickets only with Ticket Details.

Get ALL Tickets With ALL Details for Dashboard :
GET - /api/tickets/dash

-> Get all Tickets with Technician and other Details, that required in Dashboard fields.

Customer Support – Role-Based Access Description

The **Customer Support** module is used to **raise, track, and manage support call tickets** related to customer issues and service requests. Access to this module is role-based to ensure that support tickets are handled only by authorized operational roles while maintaining visibility for monitoring purposes.

Each role interacts with support tickets according to its responsibility level.

Role-Wise Customer Support Permissions

Company Admin

- Has **view-only access** to customer support tickets
- Can:
    - View all support tickets across branches
- Cannot:

    - Create new tickets
    - Update or manage tickets

Branch Admin

- Has **view-only access** to customer support tickets
- Can:
    - View support tickets related to their branch
- Cannot:
    - Create or manage tickets

Technician Manager

- Has **full access** to the Customer Support module
- Can:
    - Create support tickets
    - View support tickets
    - Assign, update, and manage ticket status
- Responsible for resolving and managing customer issues

Sales Manager & Account Manager

- Have **no access** to the Customer Support module
- Support tickets should be:
    - Hidden
    - Disabled
      for these roles

Access Control Notes

- Ticket creation and management actions must be visible **only** to Technician Manager
- View-only roles should not see ticket action buttons
- Disabled roles should not see Customer Support in navigation
- Frontend must enforce role-based visibility; backend will validate permissions

Technician APK

Postman : Technician APK
Only TECHNICIAN CAN ACCESS

Technician Login :
POST - /api/v1/login

Same as User Login User. (Authentication : User Login)

For Punch IN/OUT :
PATCH - /api/v1/user-attendance/punch-status?status=PUNCH_OUT

-> for technician, attendance and storing punch in and punch out time.
-> attendance time useful when counting salary.

Specific Technician's Task :
POST - /api/v1/task/search

-> It required technician Id in request body.
-> it gives task details of particular technician.
-> from application side, we can filter pending, in_progess, Completed.

Request :

```
{
    "paginationRequest":{
        "pageNumber":0,
        "pageSize":10
    },
    "filterColumns":{
       "technicianId":"6"
    }
}
```

Get Task Details By Id :
GET - /api/v1/task/byId?taskId=3

-> same as Task Get by Id (Postman : Task Technician Management -> Task Management -> Task Get By Id)

Start Particular Task :

POST - /api/v1/task/start?taskId=3

-> with multiple selfie of technician, we have to start the task.
-> required taskId as a Request parameter.

Completed Task Images Upload :

POST - /api/v1/task/completed/images?taskId=3

-> add before task images of where the task is done.
-> add after task images of where the task is done.
-> required taskId as a Request parameter.

Task Complete Material Update

POST - /api/v1/task/completed?taskId=3

-> pass used material details to this api.
-> after updating materials.
-> it sends otp on customerPhone from task details automatically.
-> redirect to otp verification page.
-> it required a taskId as a Request Parameter.

Send OTP if you want RESEND

POST - /api/v1/otp/send-otp?mobile=7874591247

-> it is used for resending otp on specific mobile devices.

Task Complete Feedback With Otp verification :

POST - /api/v1/task/completed/feedback?taskId=12

-> required TaskId as a Request Parameter.

REquest :

```
{
    "comment":"Good Service !",
    "rating": 4.5,
    "otp": "123456",
    "mobileNo": "7874591247"
}
```

-> we have to put all the information like the above request.

Track Technician Location :

POST - /api/v1/track/update (postman : Enhanced ERP -> Technician APK)

-> same as a Live Tracking -> Track Technician Location

Technician Calendar :

POST - /api/v1/task/daily/track

-> We have to pass requests like this, particular date and technician, so we got technician's daily work.

Request :

```
{
    "paginationRequest":{
        "pageNumber":0,
        "pageSize":10
    },
    "filterColumns":{
        "technicianId":"13",
        "date":"2026-01-10"
    }
}
```

Get technician By Email :

GET - /api/v1/users/byEmail

-> same as User Get By Email (Postman -> Enhanced ERP -> User -> get by user email)

Update Technician Details :

PUT - /api/v1/user/profile/update

-> same as User Update Profile (Postman -> Enhanced ERP -> User -> User Profile)

FOr Storing Device Token :

PUT - /api/v1/technician/devicetoken

-> Storing technician Device token for push notification.
-> on request , it required technicianId, devoceToke, OS.

Technician Apply Leave :

POST - /leave/apply

-> Applying leave we required this type of information.
Request :

```
{
  "userId": 8,  - you get from profile
  "firstName": "John",  - you get from profile
  "lastName": "Doe", - you get from profile
  "startDate": "15-11-2025",
  "endDate": "17-11-2025",
  "leaveType": "SICK",
  "reason": "Fever and rest advised by doctor",
  "status": "PENDING"
}
```

Technician Log Out :

POST - /api/v1/root/logout

-> same as Authentication log out API.
-> Generic API for all users.

Tab 25