SAP - APPROVAL WORKFLOW

PREPARED FOR

Rane Madras limited

ADMIN MANUAL STRUCTURE

1. Introduction

- **Purpose of the Application**

  The SAP Approval Workflow application is designed to streamline and digitize the material request approval process within an organization. It allows users to raise requests and enables multi-level approvals based on user roles and hierarchies.

- **Target audience of the manual (Admins, IT support, etc.)**

- **Technologies Used**

  Frontend: React.js

  Backend: Node.js with Express

3. Folder Structure and Paths

Frontend:

- **Frontend: React (`SAP-Frontend`)**
- Computer IP :

- **Path**: `D:\Vishnu Priya\SAP\SAP-Frontend`
- **Port**: `30001`
- **Start Command**: `npm start`
- **Build Command**: `npm run build`

Backend:

- **Backend**: Node.js/Express (`SAP-Backend`)
- **Path**: `D:\Vishnu Priya\SAP\SAP-Backend`
- **Port**: `3000`
- **Start Command**: `npm start`

4. Installation & Setup

Backend Setup:

- **Navigate to backend folder**:`cd D:\Vishnu Priya\SAP\SAP-Backend`

1. **Install dependencies**: `npm install`
2. **Start the backend**: `npm start`

Frontend Setup:

- **Navigate to frontend folder**:`cd D:\Vishnu Priya\SAP\SAP-Frontend`

1. **Install dependencies**:`npm install`
2. **Start the frontend**: `npm start`
3. **Build for production**: `npm run build`

5. Application Overview

Approval Workflow Levels: User Roles & Approval Levels

The SAP Approval Workflow supports multiple roles across 4 main approval levels. Here's how they're defined:

Level 1 – Requester

- **Role_ID:** 1
- **Access:** Can create material requests.

Level 2 – Department/Plant Level Approvers

- **Role_IDs:** 2 (PLANT MMD HEAD), 3 (PLANT FINANCE HEAD), 4 (PLANT MRPC)
- **Access:** Can approve/reject L1 requests.

Level 3 – Functional/Plant Heads

- **Role_IDs:** 5 (PLANT HEAD), 6 (CORP FINANCE HEAD), 7 (CORP MRPC)
- **Access:** Approves after L2.

Level 4 – Business Head

- **Role_ID:** 8
- **Access:** Final approval authority.

Admin / Others

- **CORP ADMIN (9):** Manages users and roles.
- **Other Roles (10, 11, 14, 15):** Informational or task-based roles, not in core approval loop.

- **Project Creation and Project details API:**

1. Home Page
    - **Route path : http://localhost:3001/home/Home**
        - Based On Employee

```
import { decryptSessionData } from "../../controller/StorageUtils";
```

2. Dashboard
    - **Route path : http://localhost:3001/home/dashboard**
        - Based On Employee

```
import { getLogin } from "../../controller/Masterapiservice";
import {encryptSessionData,decryptSessionData,}
from "../../controller/StorageUtils";
import { AuthContext } from "../../Authentication/AuthContext";
```

- # Masters

1. Module: Company Master

1. **Frontend Details**

Description: Add, update and view company master data

- **Route path: http://localhost:3001/home/company**
- **Component path: src/Master/Company.jsx**

API Service File:

- **src/controller/CompanyMasterapiservice.jsx**
    - **API Functions Used**
        - **getdetails() → GET /CompanyMaster/get_details_Company**
        - **getAdd(data) → POST /CompanyMaster/Get_Add**
        - **getUpdates(data) → PUT /CompanyMaster/get_Updates**

- User_Img(data) → POST /CompanyMaster/Upload_Image

2. **Backend Details**

Backend File: server/routes/CompanyMaster.Router.js

- SQL Tables: Mst_Company

Purpose → Stores master data of companiesSQL

- Procedures Used:

1. GetMst_Company → Fetch all company master records
2. InsertCompany → Insert new company record
3. ImageUpload → Update existing company record
4. UpdateCompany → Upload company logo image

2. Module: Business Division Master

1. **Frontend Details**

Description: Add, update, view business division and link to company

- Route path: http://localhost:3001/home/BusinessDivision
- Component path: src/Master/BusinessDivision.jsx

API Service File:

- src/controller/BusinessDivisionMasterapiservice.jsx
  - API Functions Used
    - getdetails() → GET /BusinessDivisionMaster/get_details_BusinessDivision
    - getAdd(data) → POST /BusinessDivisionMaster/Get_Add

- getUpdates(data) → PUT /BusinessDivisionMaster/get_Updates
- getCompany() → GET /BusinessDivisionMaster/Get_Company

2. **Backend Details**

Backend File:
server/routes/BusinessDivisionMaster.Router.js

- SQL Tables: Mst_Business_Division
- SQL Fetch Other Table: Mst_Company

Purpose → Stores business division master data

- Procedures Used:

1. GetMst_Business_Division → Fetch all business division records
2. InsertBusinessDivision → Insert new business division
3. UpdateBusinessDivision → Update business division info (name, address, status)
4. GetActiveCompany → Fetch all active companies for dropdown/listing

3. Module: Plant Master

1. **Frontend Details**

Description: The Plant Master module allows you to add, update, and manage plant records.

Each plant is linked to a company and includes code, name, and status details.

- Route path: http://localhost:3001/home/Department
- Component path: src/Master/Plant.jsx

API Service File:

- src/controller/PlantMasterapiservices.jsx
  - API Functions Used
    - getdetails() → GET /PlantMaster/get_details_Plant
    - getAdd(data) → POST /PlantMaster/Get_Add
    - getUpdates(data) → PUT /PlantMaster/get_Updates
    - getCompany() → GET /PlantMaster/Get_Company

2. Backend Details

Backend File:

server/routes/BusinessDivisionMaster.Router.js

- SQL Tables: Mst_Plant
- SQL Fetch Other Table: Mst_Company

Purpose → Stores business division master data

- Procedures Used:

1. GetMst_Plant → Fetches all plant master records from the Mst_Plant table
2. InsertPlant → Inserts a new plant record including company link, name, and status
3. UpdatePlant → Updates existing plant details (name, short name, status); restricts update if plant is inactive

4. GetActiveCompany→ Fetches all active companies for use in dropdowns or company association

4. Module: Department Master

1. **Frontend Details**

Description: The Department Master module is used to add, update, and manage department records. Each department includes a code, name, and active status. It helps define departments for request and approval workflows.

- Route path: http://localhost:3001/home/Department
- Component path: src/Master/Department.jsx

API Service File:

- **src/controller/DepartmentMasterapiservice.jsx**
  - **API Functions Used**
    - **getdetails() → GET /DepartmentMaster/get_details_Department**
    - **getAdd(data) → POST /DepartmentMaster/Get_Add**
    - **getUpdates(data) → PUT /DepartmentMaster/get_Updates**

2. Backend Details

Backend File: server/routes/DepartmentMaster.Router.js

- **SQL Tables: Mst_Department**
- **SQL Fetch Other Table: Mst_Company**

Purpose → Stores all department master data

- Procedures Used:

1. GetMst_Department → Fetches all department records
2. InsertDepartment → Inserts a new department (code, name, status, user ID)
3. UpdateDepartment → Updates an existing department; prevents update if inactive

5. Module: User Master

1. **Frontend Details**

Description: Manage users by adding, updating, and fetching user records including their plant, department, role, user level, and status.

- Route path: http://localhost:3001/home/UserMaster
- Component path: src/Master/UserMaster.jsx

API Service File:

- src/controller/UserMasterapiservice.jsx
  - API Functions Used
    - getdetails() → GET /UserMaster/get_details
    - getPlants() → GET /UserMaster/Get_Plants

- getDepartment() → GET /UserMaster/Get_Department
- getRole() → GET /UserMaster/Get_Role
- getUserLevel() → GET /UserMaster/Get_UserLevel
- getAdd(data) → POST /UserMaster/Get_Add
- getUpdates(data)→ PUT /UserMaster/get_Updates

2. **Backend Details**

Backend File: server/routes/UserMaster.Router.js

- SQL Tables: Mst_User
- SQL Fetch Other Table:
  - Mst_Company
  - Mst_Plant
  - Mst_Role
  - Mst_UserLevel
  - Mst_Department

Purpose → Stores all user master data

- Procedures Used:

1. GetUserDetails → Fetches all user records
2. GetUserActivePlants → Fetches all active plants
3. GetActiveDepartment → Fetches all active department
4. GetActiveRole → Fetches all active roles
5. GetActiveUserLevel → Fetches all active user levels
6. Insert_Mst_User → Inserts a new user record

7. UpdateUserMaster → Updates an existing user; blocks update if inactive

6. Module: Role Master

1. **Frontend Details**

Description: Manages roles by allowing add, update, and fetch operations with role name and active status to control user permissions.

- Route path: http://localhost:3001/home/Role
- Component path: src/Master/Role.jsx

API Service File:

- src/controller/Roleapiservices.jsx
  - API Functions Used
    - getdetails() → GET /RoleMaster/get_details_Role
    - getAdd() → POST /RoleMaster/Get_Add
    - getUpdates()→ PUT /RoleMaster/get_Updates

2. Backend Details

Backend File: server/routes/RoleMaster.Router.js

- SQL Tables: Mst_Role

Purpose → Stores role master data including role names and their active status to manage user permissions.

- Procedures Used:

1. GetMst_Role → Fetches all role records
2. Insert_Mst_Role → Inserts a new role

3. Insert_Mst_Role → Updates an existing role, prevents updates if inactive

7. Module: Material Master

1. **Frontend Details**

Description: Manages materials by allowing add, update, and fetch operations with material details such as material name, type, plant, and status.

- Route path: http://localhost:3001/home/Material
- Component path: src/Master/Material.jsx

API Service File:

- src/controller/Masterapiservice.jsx
  - API Functions Used
    - getdetails() → GET /Master/get_details
    - getAdd() → POST /Master/Get_Add
    - getUpdates()→ PUT /Master/get_Updates
    - getPlants() → GET /Master/Get_Plants
    - getMaterialType() → GET /Master/Get_Material_Type
    - MaterialMaster(data)→ POST /Master/File

2. **Backend Details**

Backend File: server/routes/MaterialMaster.Router.js

- SQL Tables: Mst_Material
- SQL Fetch Other Table:

- Mst_Material_Type
- Mst_Plant

Purpose → Stores material information including codes, types, and plant associations, managing inventory and procurement processes.

■ Procedures Used:

1. GetActiveMaterialInfo → Fetches all active material records
2. InsertMstMaterial → Inserts a new material record
3. UpdateMaterial → Updates existing material details
4. GetSupvCode → Retrieves supervisor codes based on PlantCode
5. GetActiveMaterialType → Fetches all active material types
6. GetActivePlants → Fetches all active plants
7. UploadMaterial1 → Handles material-related file uploads

8. Module: Vendor Master

1. **Frontend Details**

Description: The Vendor Master module allows adding, updating, and fetching vendor information including vendor code, name, address, associated plant, and active status..

- Route path: http://localhost:3001/home/Vendor
- Component path: src/Master/Vendor.jsx

API Service File:

- src/controller/VendorMasterapiservice.jsx
  - API Functions Used
    - getdetails() → GET /VendorMaster/get_details_Vendor
    - getAdd(data)→ POST /VendorMaster/Get_Add
    - getUpdates(data) → PUT /VendorMaster/get_Updates
    - getPlants()→ GET /VendorMaster/Get_Plants
    - VendorMaster(data)→ POST /VendorMaster/File

2. Backend Details

Backend File: server/routes/VendorMaster.Router.js

- SQL Tables: Mst_Vendor
- SQL Fetch Other Table:
  - Mst_Plant

Purpose → Stores vendor details including vendor code, name, address, plant association, and status, enabling supplier management across plants.

- Procedures Used:

1. GetMst_Vendor → Fetches all vendor records
2. InsertVendor → Inserts a new vendor with plant, code, name, and address
3. UpdateVendor → Updates existing vendor information
4. GetActivePlants → Fetches all active plants

5. UploadVendor → Handles file upload and batch insert for vendor data

9. Module: Customer Master

1. **Frontend Details**

Description: The Customer Master module manages customer data by enabling create, update, view, and bulk upload operations. Each record stores customer code, name, address, plant code, and active status.

- Route path: http://localhost:3001/home/Customer
- Component path: src/Master/Customer.jsx

API Service File:

- src/controller/CustomerMasterapiservice.jsx
  - API Functions Used
    - getdetails() → GET /CustomerMaster/get_details_Customer
    - getAdd(data)→ POST /CustomerMaster/Get_Add
    - getUpdates(data) → PUT /CustomerMaster/get_Updates
    - getPlants()→ GET /CustomerMaster/Get_Plants
    - CustomerMaster(data)→ POST /CustomerMaster/File

2. **Backend Details**

Backend File:

server/routes/CustomerMasterMaster.Router.js

- SQL Tables: Mst_Customer
- SQL Fetch Other Table:
  - Mst_Plant

Purpose → Stores customer master data including code, name, address, and associated plant. This supports customer-linked workflows in supply chain, sales, and delivery operations.

- Procedures Used:

1. GetMst_Customer → Fetches all customer records
2. InsertCustomer → Inserts a new customer record
3. UpdateCustomer → Updates existing customer information
4. GetActivePlants → Fetches all active plants
5. UploadCustomer → Handles file upload and inserts customer data in bulk

10. Module: Storage Location Master

1. **Frontend Details**

Description: The Storage Location Master module manages the creation, update, and mapping of storage locations to supervisors. Each storage location is linked to a plant, has a unique 4-digit code, and can be associated with multiple supervisors.

- Route path: http://localhost:3001/home/StorageLocation
- Component path: src/Master/StorageLocation.jsx

API Service File:

- src/controller/StorageLocationapiservice.jsx
  - API Functions Used
    - getdetails() → GET /StorageLocation/get_details
    - getAdd(data)→ POST /StorageLocation/Get_Add
    - getUpdates(data) → PUT /StorageLocation/get_Updates
    - getPlants()→ GET /StorageLocation/Get_Plants
    - getSupvCode(plantId)→ GET /StorageLocation/Get_SupvCode?PlantCode={id}
    - getSupvMappingsBySLocId(id) → GET /StorageLocation/supv-mapping/{SLoc_ID}
    - MappingData()→ GET /StorageLocation/Get_SupvCode_Mappings
    - getStorageDownload()→ GET /StorageLocation/get_StorageDownload

2. Backend Details

Backend File:

server/routes/StorageLocationMaster.Router.js

- SQL Tables: Mst_Storage_Location
- SQL Fetch Other Table:
  - Mst_Plant
  - Mst_SupvCode_Mapping

Purpose → Maintains records of storage locations and their supervisor mappings. Each location is plant-specific and supports mapping to one or more supervisors for operational workflows like inventory management.

■ Procedures Used:

1. Insert_Mst_StorageLocation → Inserts a new storage location and returns the Sloc_ID
2. UpdateStorageLocation → Updates the storage location details
3. GetSupvCode → Fetches available supervisor codes by PlantCode
4. Get_SupvCode_Mappings → Retrieves all supervisor-location mappings
5. GetStorageDownloadData → Used for downloading complete storage location dataset
6. Inline Query (by SLoc_ID)→ Gets all active supervisor mappings for a specific storage location
7. Custom Query (Deactivate)→ Marks existing mappings as inactive before remapping
8. Custom Query (Insert/Update Mapping)→ Inserts new or reactivates existing supervisor mappings for a location

11. Module: Movement Type Master

1. **Frontend Details**

Description: Manages movement types by allowing operations to add, update, and fetch movement codes and names used in inventory or material tracking.

- Route path: http://localhost:3001/home/Movement_Type
- Component path: src/Master/MovementType.jsx

API Service File:

- src/controller/MovementType.jsx
  - API Functions Used
    - getdetails() → GET /MovementType/get_details
    - getAdd(data)→ POST /MovementType/Get_Add
    - getUpdates(data) → PUT /MovementType/get_Updates
    - getPlants()→ GET /MovementType/Get_Plants

2. Backend Details

Backend File:
server/routes/MovementType.RouterMaster.js

- SQL Tables: Mst_Movement_Type

Purpose → Stores movement type definitions, including code, name, and active status, used for inventory tracking, stock movement, and reporting.

- Procedures Used:

1. GetMovementType → Fetches all movement type records
2. Insert_Mst_MovementType → Inserts a new movement type record

3. UpdateMovementType → Updates an existing movement type

12. Module: Movement List Item Master

1. **Frontend Details**

Description: Manages Movement List Items, allowing operations to add, update, and view records. Movement list items are subcategories of a movement type, used for detailed tracking.

- Route path: http://localhost:3001/home/MVT_LIST_ITEM
- Component path: src/Master/MovementListItem.jsx

API Service File:

- src/controller/MovementListItem.jsx
  - API Functions Used
    - getdetails() → GET /MovementListItem/get_details
    - getAdd(data)→ POST /MovementListItem/Get_Add
    - getUpdates(data) → PUT /MovementListItem/get_Updates
    - getActiveMovementType()→ GET /MovementListItem/Get_MovementType

2. **Backend Details**

Backend File:
server/routes/MovementListItemMaster.Router.js

- SQL Tables: Mst_Movement_List_Item

- SQL Fetch Other Table:
  - Mst_Movement_Type

Purpose → Stores subcategories of movement types, such as movement reasons or sub-movements, for more granular tracking and classification.

- Procedures Used:

1. GetMovementListItem→ Fetch all movement list item records
2. Insert_Mst_MovementListItem→ Insert a new movement list item record
3. UpdateMovementListItem→ Update existing movement list item
4. GetActiveMovementType → Fetch all active movement types for selection dropdowns

13. Module: Cost Center Master

1. **Frontend Details**

Description: Manage Cost Centers linked to Plants. Supports listing, adding, and updating cost centers.

- Route path: http://localhost:3001/home/CostCenter
- Component path: src/Master/CostCenter.jsx

API Service File:

- src/controller/CostCenterapiservices.js
  - API Functions Used

- getdetails() → GET /CostCenter/get_details
- getPlants()→ GET /CostCenter/Get_Plants
- getAdd(data) → POST /CostCenter/Get_Add
- getUpdates(data)→ PUT /CostCenter/get_Updates

2. Backend Details

Backend File: server/routes/CostCenterMaster.Router.js

- SQL Tables: Mst_Cost_Center
- SQL Fetch Other Table:
  - Mst_Plant

Purpose → The Cost Center Master module manages plant-specific cost centers used for tracking and controlling departmental expenses. It supports budgeting, financial reporting, and accurate cost allocation across operations.

- Procedures Used:

1. GetCostCenter → Fetch all cost center records
2. GetUserActivePlants → Fetch all active plants
3. Insert_Mst_Cost_Center → Insert new cost center
4. UpdateCostCenter → Update existing cost center

14. Module: SupvCode Master

1. **Frontend Details**

Description: Manage Supervisor Codes (SupvCode) linked to specific plants. Supports listing, adding, and updating supervisor records with lead time settings.

- **Route path: http://localhost:3001/home/SupvCode**
- **Component path: src/Master/SupvCode.jsx**

API Service File:

- **src/controller/SupvCodeMasterapiservice.js**
  - **API Functions Used**
    - **getdetails() → GET /SupvCodeMaster/get_details**
    - **getAdd(data) → POST /SupvCodeMaster/Get_Add**
    - **getPlants() → GET /SupvCodeMaster/Get_Plants**
    - **getUpdates(data)→ PUT /SupvCodeMaster/get_Updates**

2. Backend Details

Backend File: server/routes/SupvCodeMaster.js

- **SQL Tables: Mst_SupvCode**
- **SQL Fetch Other Table:**
  - **Mst_Plant**

Purpose → The SupvCode Master maintains supervisor code details for each plant, including lead time settings. It ensures correct mapping of

supervisors to support plant-specific operational planning and oversight.

- Procedures Used:

1. GetSupvCodeMaster → Fetch all supervisor code records
2. GetUserActivePlants → Fetch all active plant records
3. Insert_Mst_SupvCode → Add a new supervisor code
4. UpdateMstSupvCode → Update existing supervisor code

15. Module: Module Master

1. **Frontend Details**

Description: Manage module names associated with departments and plants. Allows listing, adding, and updating modules.

- Route path: http://localhost:3001/home/Module
- Component path: src/Master/ModuleMaster.jsx

API Service File:

- src/controller/ModuleMasterapiservice.js
  - API Functions Used
    - getdetails() → GET /ModuleMaster/get_details
    - getPlants() → GET /ModuleMaster/Get_Plants

- getDepartment() → GET /ModuleMaster/Get_Department
- getAdd(data) → POST /ModuleMaster/Get_Add
- getUpdates(data)→ PUT /ModuleMaster/get_Updates

2. Backend Details

Backend File: server/routes/ModuleMaster.js

- SQL Tables: Mst_Module
- SQL Fetch Other Table:
  - Mst_Plant
  - Mst_Department

Purpose → The Module Master manages modules linked to departments and plants for streamlined operations and role-based access control.

- Procedures Used:

1. GetModule → Fetch all module records
2. GetUserActivePlants → Get active plants list
3. GetActiveDepartment → Get active departments
4. Insert_Mst_Module → Insert a new module
5. UpdateMst_Module → Update existing module

15. Module: Submenu Page (Inside Role Page)

1. **Frontend Details**

Description:Handles screen-level access (submenu permissions) under a selected role. Allows assigning and removing submenu access for each role, improving granular control over role-based UI rendering and security.

- **Route path: http://localhost:3001/home/Role/:roleId?role=Admin&menu_name=ScreenType**
- **Component path: src/Master/Submenu.jsx**

API Service File:

- **src/controller/AdminMasterapiservice.jsx**
  - **API Functions Used**
    - **getdetailssub() → GET /AdminMaster/SubMenuNames?Role_id={role}&menuName={menu}**
    - **get_Sub_Menu_List() → GET /AdminMaster/SubMenuList?Role_id={role}&menuName={menu}**
    - **AddMenuAccess() → POST /AdminMaster/AddAccessMenu**
    - **Delete_Menu()→ PUL /AdminMaster/Delete_Access?Access_Id={id}&EmployeeId={uid}**

2. Backend Details

Backend File: server/routes/AdminMaster.js

- **SQL Tables: Mst_Screen**

- SQL Fetch Other Table:
  - Mst_Access
  - Mst_Role

- Procedures Used:

1. GetModule → Retrieves all menu permissions by role
2. SubMenusRole → Returns available submenus not yet assigned to the role
3. GetAvailableMenu → Fetches current submenu names under a menu for the role
4. AddAccess → Adds submenu/screen access to the role
5. DeleteAccess → Soft deletes (revokes) submenu access

- **Report**

1. Module: Report 309 – Excel Export

1. **Frontend Details**

Description: Generates and downloads Excel report for 309 movement transactions within a selected date range. Includes date validations and styled Excel output.

- Route path: http://localhost:3001/home/Report3
- Component path: src/Reports/Report3.jsx

API Service File:

- src/controller/Report309apiservice.jsx
  - API Functions Used
    - getTransactionData(from, to) → GET /Report3/download_data

2. Backend Details

Backend File: server/routes/Report309.Router.js

- SQL Tables: Trn_SapTransfer_Records

Purpose → Validates date inputs, runs a SQL stored procedure (GetExistingTrnSap309Movt), and returns 309 transaction data for Excel export.

- Excel Formatting:

Named: Trn_309_Movement_List.xlsx.

- Procedures Used:

1. GetExistingTrnSap309Movt → Retrieves movement transactions in the 309 category based on date range.

2. Module: Report 201 – Excel Export

1. **Frontend Details**

Description: Generates and downloads Excel report for 201 movement transactions within a selected date range. Includes date validations and styled Excel output.

- Route path: http://localhost:3001/home/Report4
- Component path: src/Reports/Report4.jsx

API Service File:

- src/controller/Report201apiservice.jsx
  - API Functions Used
    - getTransactionData(from, to) → GET /Report4/download_data

2. Backend Details

Backend File: server/routes/Report201.Router.js

- SQL Tables: Trn_SapTransfer_Records

Purpose → Validates date inputs, runs a SQL stored procedure (GetExistingTrnSap201Movt), and returns 201 transaction data for Excel export.

- Excel Formatting:

Named: Trn_201_Movement_List.xlsx.

- Procedures Used:

1. GetExistingTrnSap201Movt → Retrieves movement transactions in the 201 category based on date range.

3. Module: Report 202– Excel Export

1. **Frontend Details**

Description: Generates and downloads Excel report for 202 movement transactions within a selected date range. Includes date validations and styled Excel output.

- Route path: http://localhost:3001/home/Report5
- Component path: src/Reports/Report5.js

API Service File:

- src/controller/Report202apiservice.jsx
  - API Functions Used
    - getTransactionData(from, to) → GET /Report5/download_data

2. Backend Details

Backend File: server/routes/Report202.Router.js

- SQL Tables: Trn_SapTransfer_Records

Purpose → Validates date inputs, runs a SQL stored procedure (GetExistingTrnSap202Movt), and returns 202 transaction data for Excel export.

- Excel Formatting:

Named: Trn_202_Movement_List.xlsx.

- Procedures Used:

1. GetExistingTrnSap202Movt → Retrieves movement transactions in the 202 category based on date range.

4. Module: Report 551 – Excel Export

1. **Frontend Details**

Description: Generates and downloads Excel report for 551 movement transactions within a selected date range. Includes date validations and styled Excel output.

- Route path: http://localhost:3001/home/Report6
- Component path: src/Reports/Report6.jsx

API Service File:

- src/controller/Report551apiservice.jsx
  - API Functions Used
    - getTransactionData(from, to) → GET /Report6/download_data

2. Backend Details

Backend File: server/routes/Report551.Router.js

- SQL Tables: Trn_SapTransfer_Records

Purpose → Validates date inputs, runs a SQL stored procedure (GetExistingTrnSap551Movt), and returns 551 transaction data for Excel export.

- Excel Formatting:

Named: Trn_551_Movement_List.xlsx.

- Procedures Used:

2. GetExistingTrnSap551Movt → Retrieves movement transactions in the 551 category based on date range.

5. Module: Report 311 – Excel Export

1. **Frontend Details**

Description: Generates and downloads Excel report for 311 movement transactions within a selected date range. Includes date validations and styled Excel output.

- Route path: http://localhost:3001/home/Report7
- Component path: src/Reports/Report7.jsx

API Service File:

- src/controller/Report311apiservice.jsx
  - API Functions Used
    - getTransactionData(from, to) → GET /Report7/download_data

2. Backend Details

Backend File: server/routes/Report311.Router.js

- SQL Tables: Trn_SapTransfer_Records

Purpose → Validates date inputs, runs a SQL stored procedure (GetExistingTrnSap311Movt), and returns 311 transaction data for Excel export.

- Excel Formatting:

Named: Trn_311_Movement_List.xlsx.

- Procedures Used:

3. GetExistingTrnSap311Movt → Retrieves movement transactions in the 311 category based on date range.

6. Module: Report Rs.1 – Excel Export

1. **Frontend Details**

Description: Generates and downloads Excel report for Rs.1 movement transactions within a selected date range. Includes date validations and styled Excel output.

- Route path: http://localhost:3001/home/Report8
- Component path: src/Reports/Report8.jsx

API Service File:

- **src/controller/ReportConversionapiservice.jsx**
  - **API Functions Used**
    - getTransactionData(from, to) → **GET** /Report7/download_data

2. Backend Details

Backend File: server/routes/ReportRs1.Router.js

- SQL Tables: Trn_SapTransfer_Records

Purpose → Validates date inputs, runs a SQL stored procedure (GetExistingConversionRs1Movt), and returns Rs.1 transaction data for Excel export.

- Excel Formatting:

Named: Trn_Rs.1_Movement_List.xlsx.

- Procedures Used:

4. **GetExistingConversionRs1 → Retrieves movement transactions in the Rs.1 category based on date range.**