

AFBL SFA

**System Document**

Date: 2024-09-25

Version: 1.0

# Table of Contents



[Table of Contents 2](#_Toc179389690)

[1. Introduction 3](#_Toc179389691)

[2. System Architecture Overview 4](#_Toc179389692)

[2.1 Microservices Architecture 4](#_Toc179389693)

[3. Microservices Breakdown 5](#_Toc179389694)

[3.1 Service 1: Authentication Service 5](#_Toc179389695)

[3.3.1 Auth Module Overview 5](#_Toc179389696)

[3.2 Service 2: App Service 5](#_Toc179389697)

[3.2.1 Asset Requisition Module Overview 5](#_Toc179389698)

[3.2.2 Attendance Module Overview 6](#_Toc179389699)

[3.2.3 Auth Module Overview 6](#_Toc179389700)

[3.2.4 Bill Module Overview 7](#_Toc179389701)

[3.2.5 Common Module Overview 7](#_Toc179389702)

[3.2.6 Leave Module Overview 8](#_Toc179389703)

[3.2.7 Localization Module Overview 8](#_Toc179389704)

[3.2.8 Maintenance Module Overview 8](#_Toc179389705)

[3.2.9 Movement Module Overview 9](#_Toc179389706)

[3.2.10 Promotion Module Overview 9](#_Toc179389707)

[3.2.11 Retailer Module Overview 10](#_Toc179389708)

[3.2.12 RTC Module Overview 10](#_Toc179389709)

[3.3.13 Sales Module Overview 10](#_Toc179389710)

[3.3.14 Section Change Module Overview 11](#_Toc179389711)

[3.3.15 Static File Module Overview 11](#_Toc179389712)

[3.3.16 Sync File Module Overview 12](#_Toc179389713)

[3.3 Service 3: Master Data Service 12](#_Toc179389714)

[3.3.1 Asset Management Module Overview 12](#_Toc179389715)

[3.3.2 Bill Module Overview 13](#_Toc179389716)

[3.3.3 Common Module Overview 14](#_Toc179389717)

[3.3.4 Cron Job Module Overview 14](#_Toc179389718)

[3.3.5 Field Force Module Overview 15](#_Toc179389719)

[3.3.6 Leave Movement Module Overview 16](#_Toc179389720)

[3.3.7 Location List Module Overview 17](#_Toc179389721)

[3.3.8 Location Tree Module Overview 17](#_Toc179389722)

[3.3.9 Maintainance Module Overview 18](#_Toc179389723)

[3.3.10 Menu Module Overview 19](#_Toc179389724)

[3.3.11 Menu Access Module Overview 19](#_Toc179389725)

[3.3.12 Outlet Module Overview 20](#_Toc179389726)

[3.3.13 POSM Module Overview 20](#_Toc179389727)

[3.3.14 Product Tree Module Overview 21](#_Toc179389728)

[3.3.15 Distributor ROI Module Overview 22](#_Toc179389729)

[3.3.16 Section Change Module Overview 23](#_Toc179389730)

[3.3.17 Target Configuration Module Overview 23](#_Toc179389731)

[3.3.18 User PJP Module Overview 24](#_Toc179389732)

[3.3.19 User Module Overview 25](#_Toc179389733)

[3.3.20 Vehicle Load Module Overview 25](#_Toc179389734)

[3.4 Service 4: Trade Marketing Service 26](#_Toc179389735)

[3.4.1 Promotion Module Overview 26](#_Toc179389736)

[3.5 Service 5: Stock management Service 27](#_Toc179389737)

[3.5.1 QC Management Module Overview 27](#_Toc179389738)

[3.5.2 Shipment API Module Overview 27](#_Toc179389739)

[3.5.3 Stock API Module Overview 28](#_Toc179389740)

[3.6 Service 6: Sales Management Service 29](#_Toc179389741)

[3.6.1 Web Entry Module Overview 29](#_Toc179389742)

[3.7 Service 7: Dashboard Service 29](#_Toc179389743)

[3.7.1 Dashboard Module Overview 29](#_Toc179389744)

[3.7.2 Cron Job Module Overview 30](#_Toc179389745)

[3.8 Service 8: Survey Service 30](#_Toc179389746)

[3.8.1 Digital Learning Module Overview 30](#_Toc179389747)

[3.8.2 Survey Configuration Module Overview 31](#_Toc179389748)

[3.9 Service 9: Report Service 31](#_Toc179389749)

[3.9.1 Bulk Attendance Report Module Overview 31](#_Toc179389750)

[3.9.2 Sales Master Data Report Module Overview 32](#_Toc179389751)

[3.9.3 Non-Order Outlet Report Module Overview 33](#_Toc179389752)

[3.9.4 Sales By Geo Level Report Module Overview 34](#_Toc179389753)

[3.9.5 Field Force Sales Performance Report Module Overview 35](#_Toc179389754)

[3.9.6 Route-Wise Delivery Report Module Overview 35](#_Toc179389755)

[3.9.7 Inventory Management Report Module Overview 36](#_Toc179389756)

[3.9.8 Current Stock Report Module Overview 37](#_Toc179389757)

[3.9.9 Stock Management Report Module Overview 38](#_Toc179389758)

[3.9.10 Order Performance Report Module Overview 39](#_Toc179389759)

[3.9.11 SO Attendance Report Module Overview 39](#_Toc179389760)

[3.9.12 TSM Attendance Report Module Overview 40](#_Toc179389761)

[3.9.13 PJP Compliance Report Module Overview 41](#_Toc179389762)

[3.9.14 SO Login Log Report Module Overview 42](#_Toc179389763)

[3.9.15 Registered Outlet Report Module Overview 42](#_Toc179389764)

[3.9.16 Outlet Wise Damage Report Module Overview 43](#_Toc179389765)

[3.9.17 Route Wise Damage Report Module Overview 44](#_Toc179389766)

[3.9.18 Outlet Wise Preorder Live Report Module Overview 45](#_Toc179389767)

[3.9.19 Route Wise Preorder Live Report Module Overview 46](#_Toc179389768)

[3.9.20 Outlet Wise Sales Report Module Overview 47](#_Toc179389769)

[3.9.21 Pre Order vs Delivery Report Module Overview 47](#_Toc179389770)

[3.9.22 Memo Report Module Overview 48](#_Toc179389771)

[3.9.23 Trade Offer Report Module Overview 49](#_Toc179389772)

[3.9.24 Secondary Trade Adjustment Report Module Overview 50](#_Toc179389773)

[3.9.25 By Point Pending Report Module Overview 50](#_Toc179389774)

[3.9.26 Secondary Sales Report Module Overview 51](#_Toc179389775)

[3.9.27 PSR Sales Report Module Overview 52](#_Toc179389776)

[3.9.28 Achievement by Rank Report Module Overview 53](#_Toc179389777)

[3.9.29 Point Pending Report Module Overview 54](#_Toc179389778)

[3.11 Web frontend service 55](#_Toc179389779)

[3.11.1 Login Overview 55](#_Toc179389780)

[3.11.3 Dashboard Overview 55](#_Toc179389781)

[3.11.4 Retailers Overview 56](#_Toc179389782)

[3.11.5 Field Force Management Overview 57](#_Toc179389783)

[3.11.6 Distributor ROI Overview 57](#_Toc179389784)

[3.11.7 Section Overview 58](#_Toc179389785)

[3.11.8 Vehicle Overview 58](#_Toc179389786)

[3.11.9 Stock Management New Shipment Overview 58](#_Toc179389787)

[3.11.10 Stock Management Reports Overview 59](#_Toc179389788)

[3.11.11 Damage Management Overview 60](#_Toc179389789)

[3.11.12 Product Management Overview 61](#_Toc179389790)

[3.11.13 Data Entry Overview 62](#_Toc179389791)

[3.11.14 Survey Overview 62](#_Toc179389792)

[3.11.15 TA/DA Overview 63](#_Toc179389793)

[3.11.16 Vehicle Load Overview 64](#_Toc179389794)

[3.11.17 Target Overview 64](#_Toc179389795)

[3.11.18 Trade Promotion Overview 65](#_Toc179389796)

[3.11.19 Journey Plan Overview 66](#_Toc179389797)

[3.11.20 Leave Management Overview 66](#_Toc179389798)

[3.11.21 Asset Management Overview 67](#_Toc179389799)

[3.11.22 User Journey Overview 68](#_Toc179389800)

[3.11.23 Master Entry Overview 68](#_Toc179389801)

[3.11.24 Digital Learning Overview 69](#_Toc179389802)

[3.11.25 Credential Overview 69](#_Toc179389803)

[4. Mobile App: Flutter 71](#_Toc179389804)

[4.1 Module wise overview 71](#_Toc179389805)

[4.1.1 Login Module 71](#_Toc179389806)

[4.1.2 Verification Controller 71](#_Toc179389807)

[4.2.1 Outlet Module 72](#_Toc179389808)

[4.2.2 Outlet UI 72](#_Toc179389809)

[4.2.3 Outlet Controller 73](#_Toc179389810)

[4.3 Preorder Module 73](#_Toc179389811)

[4.4 Memo Module 74](#_Toc179389812)

[4.4 Delivery Module 75](#_Toc179389813)

[4.5.1 Attendance Module 77](#_Toc179389814)

[4.5.2 Attendance UI 77](#_Toc179389815)

[4.5.3 Attendance Controller 78](#_Toc179389816)

[4.5.4 Check In/Out 78](#_Toc179389817)

[4.5.5 Geo Fencing 78](#_Toc179389818)

[4.6.1 Summary Module 79](#_Toc179389819)

[4.6.2 Sales Summary UI 79](#_Toc179389820)

[4.6.3 Sales Summary Controller 80](#_Toc179389821)

[4.7.1 Asset Module 80](#_Toc179389822)

[4.7.2 Asset UI 80](#_Toc179389823)

[4.7.3 Asset Controller 81](#_Toc179389824)

[4.7.4 Asset Requisition 81](#_Toc179389825)

[4.7.5 Asset Installation 81](#_Toc179389826)

[4.7.6 Asset Pull Out 82](#_Toc179389827)

[4.8 Sales Submit Module 82](#_Toc179389828)

[4.8.1 Leave Management Module 83](#_Toc179389829)

[4.8.2 Leave UI 83](#_Toc179389830)

[4.8.3 Leave Controller 84](#_Toc179389831)

[4.9.1 Allowance Management Module 84](#_Toc179389832)

[4.9.2 Allowance UI 84](#_Toc179389833)

[4.9.3 Movement Management 85](#_Toc179389834)

[4.9.4 TA/DA Management 85](#_Toc179389835)

[4.9.5 Allowance Approval 86](#_Toc179389836)

[4.9.6 Allowance History 86](#_Toc179389837)

[4.10 Digital Learning Module 86](#_Toc179389838)

[4.11 PJP Plan Module 87](#_Toc179389839)

[4.12 Maintenance Module 87](#_Toc179389840)

**Technical Documentation: Microservices Architecture**

**Project: Akij SFA Solution**

**Backend: NodeJS (NestJS)**

**Frontend: ReactJS (NextJS)**

**Mobile App: Flutter**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 1. Introduction

**Overview of Microservices Architecture**

Microservices architecture is an approach to building a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. These services are built around business capabilities and independently deployable by fully automated deployment machinery.

**Purpose of Each Microservice**

1. **API Gateway:** Acts as an entry point for clients. It routes requests to the appropriate microservices, handles authentication, and can also perform load balancing, caching, and request shaping.
2. **Authentication Service:** Manages user authentication and authorization. It issues tokens that clients use to access other services securely.
3. **App Service:** Manages all the APIs related to the mobile app, ensuring seamless interaction and data flow between the app and backend services.
4. **Master Data Service:** Handles all requests related to master data and web APIs, ensuring consistent and accurate data across the system.
5. **Trade Marketing Service:** Manages trade configuration and scope upload features, facilitating effective trade marketing strategies and operations.
6. **Stock Management Service:** Keeps track of product inventory levels, updates stock quantities, and notifies other services when stock levels change, ensuring accurate inventory management.
7. **Sales Management Service:** Manages house sell-related features, supporting the sales process and ensuring efficient sales operations.
8. **Dashboard Service:** Manages all recurring tasks for calculating dashboard data and other features related to the dashboard, providing valuable insights and analytics.
9. **Survey Service:** Manages all features related to digital learning and surveys, enabling the creation, distribution, and analysis of surveys.
10. **Product Service:** Manages product-related APIs, handling operations such as adding new products, updating product details, and retrieving product information.

**Interaction Between Microservices**

* **API Gateway** receives client requests and routes them to the appropriate microservice.
* **Authentication Service** verifies user credentials and issues tokens used by other services to authenticate requests.
* **Stock Management Service** updates stock levels and notifies the **Order Service** when stock changes.

**Importance in Ensuring a Scalable, Maintainable, and Modular Application**

* **Scalability**: Each microservice can be scaled independently based on its load. For example, the **App Service** can be scaled up during peak hours without affecting other services.
* **Maintainability**: Smaller codebases are easier to manage. Each microservice can be developed, tested, and deployed independently, reducing the complexity of the overall system.
* **Modularity**: Microservices promote a modular approach where each service focuses on a specific business capability. This modularity allows teams to work on different services simultaneously without interfering with each other.
* **Fault Isolation**: Failures in one microservice do not necessarily bring down the entire system. For example, if the **Survey Service** fails, the core functionality of the application remains unaffected.
* **Technology Diversity**: Different microservices can be built using different technologies that are best suited for their specific tasks, allowing for greater flexibility and optimization.

By breaking down the application into smaller, manageable pieces, microservices architecture helps in building robust, scalable, and maintainable systems.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 2. System Architecture Overview

## 2.1 Microservices Architecture

**Microservices Overview:**

Microservices architecture is a design pattern where an application is composed of small, independent services that communicate over a network. Each service is focused on a specific business function and can be developed, deployed, and scaled independently. Here’s a detailed description of how microservices are structured and deployed, along with their benefits:

**Structure of Microservices**

1. **Service Independence**:
   * Each microservice is a standalone application with its own database and data model.
   * Services communicate with each other using lightweight protocols like HTTP/REST or messaging queues.
2. **Domain-Driven Design**:
   * Microservices are often designed around business domains, ensuring that each service corresponds to a specific business capability.
3. **Decentralized Data Management**:
   * Each microservice manages its own database, which helps in maintaining data consistency and integrity within the service.
4. **API Gateway**:
   * An API Gateway often sits in front of the microservices, handling requests from clients and routing them to the appropriate service.

**Deployment of Microservices**

1. **Containerization**:
   * Microservices are typically deployed in containers (e.g., Docker), which encapsulate the service and its dependencies, ensuring consistency across different environments.
2. **Orchestration**:
   * Tools like Kubernetes are used to manage the deployment, scaling, and operation of containerized applications, ensuring that services are resilient and can recover from failures.
3. **Continuous Integration/Continuous Deployment (CI/CD)**:
   * Automated pipelines are set up to build, test, and deploy microservices, enabling rapid and reliable delivery of new features and updates.

**Benefits of Microservices**

1. **Scalability**:
   * Each microservice can be scaled independently based on its load and performance requirements. This allows for more efficient use of resources compared to scaling a monolithic application.
2. **Modularity**:
   * Microservices promote a modular architecture where each service is developed and maintained independently. This modularity makes it easier to manage and update individual services without affecting the entire system.
3. **Fault Isolation**:
   * Since microservices are independent, a failure in one service does not necessarily impact other services. This isolation improves the overall resilience and reliability of the application.
4. **Technology Diversity**:
   * Different microservices can be built using different technologies and programming languages, allowing teams to choose the best tools for each specific task.
5. **Faster Time to Market**:
   * Independent development and deployment of microservices enable teams to release new features and updates more quickly, responding faster to market demands and user feedback.
6. **Improved Maintainability**:
   * Smaller, focused codebases are easier to understand, test, and maintain. This reduces technical debt and improves the overall quality of the software.

In summary, microservices architecture offers significant advantages in terms of scalability, modularity, and fault isolation, making it a popular choice for building large, complex applications that need to be agile and resilient.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 3. Microservices Breakdown

## 3.1 Service 1: Authentication Service

### 3.3.1 Auth Module Overview

**Overview**

The AuthModule in this microservice is responsible for managing user authentication and authorization processes. It includes several key components and endpoints to handle login, logout, and token refresh functionalities.

**Key Responsibilities**

User Authentication: Validates user credentials and provides access tokens.

User Authorization: Ensures that only authenticated users can access certain endpoints.

Token Management: Handles the generation, validation, and refreshing of JWT tokens.

**Key Endpoints**

POST /auth/login: Authenticates a user and returns an access token.

POST /auth/logout: Logs out the authenticated user.

POST /auth/refreshtoken: Generates a new access token using a refresh token.

## 3.2 Service 2: App Service

### 3.2.1 Asset Requisition Module Overview

**Overview**

The Asset Requisition Module is a crucial component of a microservice architecture designed to manage the lifecycle of assets within an organization. This module handles the creation, installation, pull-out, and approval processes for asset requisitions. It ensures that assets are efficiently allocated, tracked, and maintained across various outlets or departments.

**Key Responsibilities**

Asset Requisition Management: Handle the creation, installation, pull-out, and approval processes for asset requisitions.

Asset Tracking and Reporting: Provide detailed lists of assets for specific outlets, track the status and history of asset requisitions, and generate reports for asset management and auditing purposes.

Integration with Other Services: Communicate with other microservices for inventory management, user authentication, and notification services, ensuring data consistency and integrity across the microservice ecosystem.

**Key Endpoints**

POST /asset-requisition/new-requisition: Creates a new asset requisition.

POST /asset-requisition/new-installation: Handles the installation of new assets.

POST /asset-requisition/pull-out: Processes the pull-out of assets from outlets.

GET /asset-requisition/get-outlet-asset-list: Retrieves a list of assets for a specific outlet.

GET /asset-requisition/get-outlet-asset-requisition/:id: Retrieves details of a specific asset requisition by its ID.

### 3.2.2 Attendance Module Overview

**Overview**

The Attendance Module is a vital part of a microservice architecture designed to manage employee attendance within an organization. This module handles the check-in and check-out processes, ensuring accurate tracking of employee working hours and attendance records. It also manages the upload and storage of clock-in and clock-out images for verification purposes.

**Key Responsibilities**

Attendance Management: Handle the check-in and check-out processes for employees.

Image Upload and Storage: Manage the upload and storage of clock-in and clock-out images for verification purposes.

Attendance Reporting: Provide detailed attendance records and reports for employees.

**Key Endpoints**

POST /attendance: Handles employee check-in with image upload.

PATCH /attendance: Handles employee check-out with image upload.

GET /attendance: Retrieves attendance records based on query parameters.

GET /attendance/tsm: Retrieves attendance records for TSM based on query parameters.

### 3.2.3 Auth Module Overview

**Overview**

The Auth Module is a critical part of a microservice architecture designed to manage authentication and authorization within an organization. This module handles user login, token generation, and password management, ensuring secure access to the system's resources.

**Key Responsibilities**

User Authentication: Handle user login processes and validate credentials.

Token Management: Generate and manage JWT tokens for authenticated sessions.

Password Management: Provide functionalities for changing and resetting passwords.

**Key Endpoints**

POST /ff-auth/login: Handles user login and returns a JWT token.

POST /ff-auth/change-password: Allows users to change their password.

POST /ff-auth/refresh-token: Refreshes the JWT token for an authenticated session.

### 3.2.4 Bill Module Overview

**Overview**

The Bill Module is an essential part of a microservice architecture designed to manage billing processes within an organization. This module handles the retrieval and disbursement of outdoor bills, ensuring accurate and efficient billing operations.

**Key Responsibilities**

Bill Retrieval: Handle the retrieval of outdoor bills based on query parameters.

Bill Disbursement: Manage the disbursement of outdoor bills by updating their status.

**Key Endpoints**

GET /bill/outdoor-bill: Retrieves outdoor bills based on query parameters.

PATCH /bill/disburse-outdoor-bill/:id: Disburses an outdoor bill by updating its status.

### 3.2.5 Common Module Overview

**Overview**

The Common Module is a versatile part of a microservice architecture designed to handle various common functionalities within an organization. This module manages geo-tracking data and file uploads, ensuring efficient handling of these operations.

**Key Responsibilities**

Geo-Tracking Management: Handle the saving and management of geo-tracking data.

File Upload Management: Manage the upload and storage of files with specific filters and storage configurations.

**Key Endpoints**

POST /common/save-geo-tracking-data: Saves geo-tracking data.

POST /common/upload-file: Uploads a file with specific filters and storage configurations.

GET /common/latest-app-version: Retrieves the latest app version.

DELETE /common/delete-pda-file: Deletes a PDA file based on provided criteria.

### 3.2.6 Leave Module Overview

**Overview**

The Leave Module is an essential part of a microservice architecture designed to manage employee leave within an organization. This module handles the creation, retrieval, and calendar view of leave requests, ensuring efficient leave management and tracking.

**Key Responsibilities**

Leave Management: Handle the creation and retrieval of leave requests.

Leave Calendar: Provide a calendar view of leave data for better visualization and planning.

Leave Querying: Allow querying of leave data based on various parameters.

**Key Endpoints**

POST /leave: Creates a new leave request.

GET /leave: Retrieves leave requests based on query parameters.

GET /leave/calendar: Retrieves leave data in a calendar format.

### 3.2.7 Localization Module Overview

**Overview**

The Localization Module is a crucial part of a microservice architecture designed to manage localization data within an organization. This module handles the retrieval and management of localization data, ensuring that the application can support multiple languages and regional settings efficiently.

**Key Responsibilities**

Localization Data Management: Handle the retrieval and management of localization data.

Error Handling: Ensure robust error handling for localization data operations.

Service Integration: Integrate with the localization service to fetch and update localization data.

**Key Endpoints**

GET /localization: Retrieves localization data.

### 3.2.8 Maintenance Module Overview

**Overview**

The Maintenance Module is a vital part of a microservice architecture designed to manage maintenance tasks within an organization. This module handles the retrieval of maintenance request lists and the updating of maintenance tasks, ensuring efficient maintenance operations and tracking.

**Key Responsibilities**

Maintenance Request Management: Handle the retrieval of maintenance request lists for users.

Task Update Management: Manage the updating of maintenance tasks with detailed information.

**Key Endpoints**

GET /maintainance/request-list/:user\_id: Retrieves the maintenance request list for a specific user.

POST /maintainance/task-update: Updates maintenance tasks with detailed information.

### 3.2.9 Movement Module Overview

**Overview**

The Movement Module is a critical part of a microservice architecture designed to manage employee movements within an organization. This module handles the creation, retrieval, and updating of movement records, ensuring efficient tracking and management of employee movements.

**Key Responsibilities**

Movement Creation: Handle the creation of new movement records.

Movement Retrieval: Retrieve movement records based on various query parameters.

PJP Management: Manage the creation, retrieval, and updating of PJP (Planned Journey Plan) records.

**Key Endpoints**

POST /movement: Creates a new movement record.

GET /movement: Retrieves movement records based on query parameters.

GET /movement/pjp: Retrieves PJP records based on query parameters.

PATCH /movement/update\_pjp: Updates PJP records.

### 3.2.10 Promotion Module Overview

**Overview**

The Promotion Module is an essential part of a microservice architecture designed to manage promotional activities within an organization. This module handles the creation, retrieval, and updating of promotion data, ensuring efficient management and tracking of promotional activities.

**Key Responsibilities**

Promotion Data Management: Handle the creation and updating of promotion data.

QPS Enrollment: Manage the enrollment data for QPS (Quarterly Promotion Scheme).

Sales Data Management: Retrieve and manage QPS-wise sales data.

**Key Endpoints**

POST /promotion/save-qps-enroll-data: Saves QPS enrollment data.

POST /promotion/qps-wise-sale-data: Retrieves QPS-wise sales data.

### 3.2.11 Retailer Module Overview

**Overview**

The Retailer Module is a crucial part of a microservice architecture designed to manage retailer-related operations within an organization. This module handles the creation, updating, and bulk disabling of retailer records, ensuring efficient management and tracking of retailer information.

**Key Responsibilities**

Retailer Creation: Handle the creation of new retailer records with associated images.

Retailer Update: Manage the updating of existing retailer records with new information and images.

Bulk Retailer Management: Handle the bulk disabling of retailer records based on provided criteria.

**Key Endpoints**

POST /retailer: Creates a new retailer record with associated images.

PUT /retailer/bulk-disable: Bulk disables retailer records based on provided outlet codes.

PUT /retailer/:outlet\_code: Updates an existing retailer record with new information and images.

### 3.2.12 RTC Module Overview

**Overview**

The RTC (Digital Learning) Module is an essential part of a microservice architecture designed to manage digital learning related data. This module handles the saving of RTC surveys, synchronization of RTC RAG scores, and uploading of survey scope files, ensuring efficient management and tracking of RTC data.

**Key Responsibilities**

RTC Survey Management: Handle the saving of RTC survey data.

RTC RAG Score Synchronization: Synchronize RTC RAG scores based on department, section, and date.

Survey Scope File Upload: Manage the upload and storage of survey scope files.

**Key Endpoints**

POST /rtc/save-survey: Saves RTC survey data.

GET /rtc/sync-rtc-rag-score: Synchronizes RTC RAG scores based on query parameters.

POST /rtc/upload-survey-scope: Uploads survey scope files with specific configurations.

### 3.3.13 Sales Module Overview

**Overview**

The Sales Module is a critical part of a microservice architecture designed to manage sales-related operations within an organization. This module handles the saving of sales and delivery data, retrieving sales summaries, and managing unsold sales data, ensuring efficient management and tracking of sales activities.

**Key Responsibilities**

Sales Data Management: Handle the saving of sales data for various outlets.

Delivery Data Management: Manage the saving of delivery data for various outlets.

Sales Summary Retrieval: Provide detailed sales summaries for specific outlets.

Unsold Sales Data Management: Handle the saving and management of unsold sales data.

Stock Check Management: Manage stock check operations for various outlets.

Try Before You Buy Management: Handle the data for the "Try Before You Buy" program.

**Key Endpoints**

POST /sales/save-sales-data: Saves sales data for an outlet.

POST /sales/save-delivery-data: Saves delivery data for an outlet.

GET /sales/get-outlet-wise-sale-data: Retrieves sales data for a specific outlet.

POST /sales/save-unsold-sales-data: Saves unsold sales data.

POST /sales/stock-check: Manages stock check operations.

POST /sales/try-before-you-buy: Handles data for the "Try Before You Buy" program.

### 3.3.14 Section Change Module Overview

**Overview**

The Section Change Module is an essential part of a microservice architecture designed to manage section change requests within an organization. This module handles the creation and retrieval of section change request statuses, ensuring efficient management and tracking of these requests.

**Key Responsibilities**

Section Change Request Creation: Handle the creation of new section change requests.

Request Status Retrieval: Retrieve the status of section change requests based on query parameters.

**Key Endpoints**

POST /section-change: Creates a new section change request.

GET /section-change/request-status: Retrieves the status of section change requests based on query parameters.

### 3.3.15 Static File Module Overview

**Overview**

The Static File Module is a crucial part of a microservice architecture designed to manage the retrieval of static files within an organization. This module handles the serving of static files from various directories, ensuring efficient access and management of these files.

**Key Responsibilities**

Static File Retrieval: Handle the retrieval of static files based on query parameters and path variables.

File Listing: Provide a list of all available static files for synchronization purposes.

**Key Endpoints**

GET /static-file: Retrieves a static file based on query parameters.

GET /static-file/app-sync-file-assets: Retrieves a list of all available static files for synchronization purposes.

GET /static-file/retailer/:folder\_name/:fileId: Retrieves a retailer-specific static file based on folder name and file ID.

GET /static-file/:folder\_name/:fileId: Retrieves a static file based on folder name and file ID.

### 3.3.16 Sync File Module Overview

**Overview**

The Sync File Module is a crucial part of a microservice architecture designed to manage the synchronization of files within an organization. This module handles the creation, updating, and retrieval of sync files, ensuring efficient and accurate synchronization of data.

**Key Responsibilities**

Sync File Creation: Handle the creation of new sync files.

Mid-Day Configuration Update: Manage the updating of mid-day configuration data.

Sync File Retrieval: Retrieve sync file data based on provided criteria.

Retailer List Management: Retrieve retailer lists for specific sections and sales dates.

Pre-Order Management: Retrieve available pre-order data for specific sales representatives and dates.

**Key Endpoints**

POST /sync-file: Creates a new sync file.

POST /sync-file/mid-day-configuration-update: Updates mid-day configuration data.

POST /sync-file/save-sync-file: Saves sync file data for web entry.

GET /sync-file/get-retailer-list/:sectionId/:salesDate: Retrieves retailer lists for specific sections and sales dates.

GET /sync-file/parents: Retrieves SKU-wise parent data.

GET /sync-file/auto-generate-sync-file: Automatically generates a sync file.

GET /sync-file/get-available-pre-order/:srId/:salesDate: Retrieves available pre-order data for specific sales representatives and dates.

## 3.3 Service 3: Master Data Service

### 3.3.1 Asset Management Module Overview

**Overview**

The asset-management module is responsible for managing various asset-related operations within the microservice. This includes functionalities for assigning assets, retrieving requisition lists, rejecting requisitions, handling asset pull-outs, and fetching requisition asset information.

**Key Responsibilities**

Assign Assets: Provides functionality to assign assets to different entities or users.

Retrieve Requisition Lists: Allows retrieval of asset requisition lists based on specific query parameters.

Reject Requisitions: Supports the rejection of asset requisitions by their ID.

Handle Asset Pull-Outs: Manages the process of pulling out assets from inventory.

Fetch Requisition Asset Information: Retrieves detailed information about a specific asset requisition by its ID.

**Key Endpoints**

- POST /assign-asset: Assigns an asset to a specific entity or user based on the provided details.

- GET /get-requisition-list: Retrieves a list of asset requisitions based on specific query parameters.

- GET /reject-requisition/:id: Rejects an asset requisition by its ID.

- POST /asset-pull-out: Handles the process of pulling out assets from inventory based on the provided details.

### 3.3.2 Bill Module Overview

**Overview**

The bill module is responsible for managing various billing operations within the microservice. This includes functionalities for creating and retrieving bill categories, handling outdoor bills, and updating bill information. The module ensures that billing data is managed efficiently and securely.

**Key Responsibilities**

- Create Bill Categories: Provides functionality to create new bill categories, such as electricity bill categories.

- Retrieve Bill Categories**:** Allows retrieval of existing bill categories.

- Create Outdoor Bills: Supports the creation of outdoor bills, capturing necessary billing details.

- Retrieve Outdoor Bills: Enables retrieval of outdoor bills based on specific query parameters, such as department ID and date.

- Update Bills: Facilitates the updating of existing bill information.

**Key Endpoints**

- POST /bill/electricity-bill-category: Creates a new electricity bill category based on the provided details.

- GET /bill/electricity-bill-category: Retrieves the list of existing electricity bill categories.

- POST /bill/outdoor-bill: Creates a new outdoor bill with the provided billing details.

- GET /bill/outdoor-bill: Retrieves outdoor bills based on specific query parameters, such as department ID and date.

- PATCH /bill/outdoor-bill: Updates an existing outdoor bill with the provided details.

### 3.3.3 Common Module Overview

**Overview**

The common-module is responsible for handling common operations within the microservice. It includes functionalities for retrieving master categories, fetching SBU (Strategic Business Unit) lists, and saving user activity logs.

**Key Responsibilities**

- Retrieve Master Categories: Provides functionality to retrieve master categories based on a given slug.

- Fetch SBU List: Allows retrieval of the list of Strategic Business Units.

- Save User Activity Logs: Supports saving user activity logs to track user actions within the system.

**Key Endpoints**

- GET /common-module/master-categories: Retrieves master categories based on the provided slug.

- GET /common-module/sbu-list: Fetches the list of Strategic Business Units.

- POST /common-module/save-user-activity-log: Saves a user activity log with the provided details.

### 3.3.4 Cron Job Module Overview

**Overview**

The cron-job module is responsible for managing scheduled tasks and background jobs within the microservice. It includes functionalities for synchronizing shipment and point information, and preparing sales data by distribution point and section.

**Key Responsibilities**

- Synchronize Shipment Information: Provides functionality to synchronize point shipment information.

- Synchronize Point Information: Allows synchronization of point information.

- Prepare Sales Data: Supports the preparation of sales data by department and section based on a given date.

**Key Endpoints**

- GET /cron-job/sync-point-shipment-info: Synchronizes point shipment information.

- GET /cron-job/sync-point-info: Synchronizes point information.

- GET /cron-job/prepear-sales-by-dep/:date: Prepares sales data by department for a given date.

- GET /cron-job/prepear-trans-sales-by-section/:date: Prepares sales data by section for a given date.

### 3.3.5 Field Force Module Overview

**Overview**

The field-force module is responsible for managing operations related to field force personnel within the microservice. It includes functionalities for saving field force information, retrieving sales representative and supervisor information, and updating field force details.

**Key Responsibilities**

- Save Field Force Information: Provides functionality to save new field force personnel information.

- Retrieve Sales Representative and Supervisor Information: Allows retrieval of sales representative and supervisor information based on department and SBU (Strategic Business Unit) IDs.

- Update Field Force Details: Supports updating the details of existing field force personnel.

- Disable Field Force: Provides functionality to disable a field force personnel by their ID.

- Retrieve Section-Wise Field Force List: Allows retrieval of field force personnel based on sections.

- Assign Routes to Field Force: Supports assigning routes to field force personnel.

- Unassign Routes from Field Force: Provides functionality to remove field force personnel from assigned routes.

- Export Field Force Data: Allows exporting field force data based on specific query parameters.

- Reset Field Force Password: Supports resetting the password for field force personnel.

**Key Endpoints**

- POST /master-entry/field-force/save-field-force: Saves new field force personnel information based on the provided details.

- GET /master-entry/field-force/get-field-force-list: Retrieves a list of field force personnel based on query parameters.

- POST /master-entry/field-force/update-field-force: Updates the details of existing field force personnel based on the provided information.

### 3.3.6 Leave Movement Module Overview

**Overview**

The leave-movement module is responsible for managing leave and movement operations within the microservice. It includes functionalities for creating leave movements, retrieving leave movement records, updating leave and movement details, and handling bulk leave uploads.

**Key Responsibilities**

Create Leave Movements: Provides functionality to create new leave movement records.

Retrieve Leave Movements: Allows retrieval of leave movement records based on specific query parameters.

Update Leave Details: Supports updating the details of existing leave records.

Update Movement Details: Supports updating the details of existing movement records.

Bulk Leave Upload: Provides functionality to upload leave records in bulk.

Approve or Reject Ta/Da: Manages the approval or rejection of travel allowances (Ta/Da).

Download Leave Balance Sample: Allows downloading a sample file for leave balance uploads.

Delete Attendance Records: Supports deleting attendance records based on specific criteria.

**Key Endpoints**

POST /leave-movement: Creates a new leave movement record based on the provided details.

GET /leave-movement: Retrieves leave movement records based on specific query parameters.

PATCH /leave-movement/leave/:id: Updates the details of an existing leave record by its ID.

PATCH /leave-movement/movement/:id: Updates the details of an existing movement record by its ID.

POST /leave-movement/bulk-upload: Uploads leave records in bulk using a provided file.

### 3.3.7 Location List Module Overview

**Overview**

The location-list module is responsible for managing location-related operations within the microservice. It includes functionalities for retrieving categories by type, creating sections, disabling routes, adding new routes to sections, and fetching section and point geo-location data.

**Key Responsibilities**

Retrieve Categories by Type: Provides functionality to retrieve location categories based on a given type.

Create Sections: Allows the creation of new sections within the location list.

Disable Routes: Supports disabling routes within a specific section.

Add New Routes to Sections: Provides functionality to add new routes to existing sections.

Retrieve Section Data: Allows retrieval of section data by its ID.

Retrieve Section List Tree: Supports fetching a hierarchical list of sections.

Retrieve Point Geo-Location: Provides functionality to retrieve geographical location data for points.

**Key Endpoints**

GET /location-list/category-by-type: Retrieves location categories based on the provided type.

POST /location-list/section: Creates a new section within the location list.

DELETE /location-list/route/:section\_id: Disables a route within a specific section by its ID.

PATCH /location-list/section: Adds a new route to an existing section based on the provided details.

GET /location-list/section/:id: Retrieves section data by its ID.

GET /location-list/section-list-tree: Retrieves a hierarchical list of sections based on specific query parameters.

GET /location-list/point-geo-location: Retrieves geographical location data for points based on specific query parameters.

### 3.3.8 Location Tree Module Overview

**Overview**

The location-tree module is responsible for managing hierarchical location data within the microservice. It includes functionalities for retrieving location trees, route lists, active route lists, and section lists.

**Key Responsibilities**

Retrieve Location Tree: Provides functionality to retrieve a hierarchical tree structure of locations.

Retrieve Route Lists: Allows retrieval of route lists based on specific query parameters.

Retrieve Active Route Lists: Supports fetching active route lists.

Retrieve Section Lists: Provides functionality to retrieve lists of sections.

Retrieve All Locations: Allows retrieval of all location data.

Retrieve Location by ID: Supports fetching location data by its ID.

**Key Endpoints**

POST /location-tree: Retrieves a hierarchical tree structure of locations based on the provided details.

GET /location-tree/get-route-list: Retrieves route lists based on specific query parameters.

GET /location-tree/get-active-route-list: Retrieves active route lists based on specific query parameters.

GET /location-tree/get-section-list: Retrieves lists of sections based on specific query parameters.

GET /location-tree: Retrieves all location data.

GET /location-tree/:id: Retrieves location data by its ID.

### 3.3.9 Maintainance Module Overview

**Overview**

The maintainance module is responsible for managing maintenance operations within the microservice. It includes functionalities for creating maintenance requests, retrieving maintenance request lists, viewing maintenance details, and assigning users to maintenance tasks.

**Key Responsibilities**

Create Maintenance Requests: Provides functionality to create new maintenance requests.

Retrieve Maintenance Request Lists: Allows retrieval of maintenance request lists based on specific query parameters.

View Maintenance Details: Supports viewing detailed information about a specific maintenance request.

Retrieve Maintenance User Lists: Provides functionality to retrieve lists of users available for maintenance tasks based on department ID.

Assign Users to Maintenance Tasks: Supports assigning users to specific maintenance tasks.

**Key Endpoints**

POST /maintainance/create-new-request: Creates a new maintenance request based on the provided details.

GET /maintainance/request-list: Retrieves a list of maintenance requests based on specific query parameters.

GET /maintainance/view-details/:id: Retrieves detailed information about a specific maintenance request by its ID.

GET /maintainance/maintainance-user-list/:dep\_id: Retrieves a list of users available for maintenance tasks based on department ID.

POST /maintainance/assign-user: Assigns a user to a specific maintenance task based on the provided details.

### 3.3.10 Menu Module Overview

**Overview**

The menu module is responsible for managing menu items within the microservice. It includes functionalities for creating, retrieving, updating, and deleting menu items, ensuring that the menu structure is maintained and updated as needed.

**Key Responsibilities**

Retrieve All Menu Items: Provides functionality to retrieve a list of all menu items.

Retrieve Dynamic Menu Items: Allows retrieval of menu items based on specific query parameters.

Create Menu Items: Supports the creation of new menu items.

Update Menu Items: Provides functionality to update existing menu items.

Delete Menu Items: Supports the deletion of menu items based on specific criteria.

**Key Endpoints**

GET /menu/list: Retrieves a list of all menu items.

GET /menu: Retrieves dynamic menu items based on specific query parameters.

POST /menu: Creates a new menu item based on the provided details.

### 3.3.11 Menu Access Module Overview

**Overview**

The menu-access module is responsible for managing access control to menu items within the microservice. It includes functionalities for creating menu access permissions, ensuring that users have the appropriate access to various menu items.

**Key Responsibilities**

Create Menu Access Permissions: Provides functionality to create new menu access permissions.

**Key Endpoints**

POST /menu-access/create: Creates a new menu access permission based on the provided details.

### 3.3.12 Outlet Module Overview

**Overview**

The outlet module is responsible for managing operations related to outlets within the microservice. It includes functionalities for getting outlets, retrieving outlet classifications, generating reports and outlet list.

**Key Responsibilities**

Retrieve Outlet Classifications: Allows retrieval of outlet classifications based on specific query parameters.

Retrieve Outlet Lists: Supports fetching lists of outlets based on provided criteria.

Generate Reports: Provides functionality to generate reports for outlets and export them.

Generate Outlet list: For faster outlet list retrival.

**Key Endpoints**

GET /outlets/classification: Retrieves outlet classifications based on specific query parameters.

POST /outlets/list: Retrieves a list of outlets based on the provided criteria.

POST /outlets/generate-report: Generates a report for outlets and exports it based on the provided criteria.

### 3.3.13 POSM Module Overview

**Overview**

The POSM (Point of Sale Materials) module is responsible for managing operations related to POSM stock within the microservice. It includes functionalities for retrieving and adding POSM stock by department, section, and region, as well as generating POSM reports and managing POSM statuses.

**Key Responsibilities**

Retrieve POSM Stock by Department: Provides functionality to retrieve POSM stock levels by department.

Add POSM Stock by Section: Allows adding POSM stock levels by section.

Retrieve POSM Stock by Section: Supports fetching POSM stock levels by section.

Retrieve POSM Status: Provides functionality to retrieve the status of POSM items.

Generate POSM Reports: Supports generating reports for POSM stock levels and statuses.

Add POSM Stock by Region: Allows adding POSM stock levels by region.

Nill POSM Stock by Section: Provides functionality to mark POSM stock as nil by section.

**Key Endpoints**

GET /posm/stock-by-dep: Retrieves POSM stock levels by department based on the provided query parameters.

POST /posm/stock-by-section: Adds POSM stock levels by section based on the provided details.

GET /posm/stock-by-section: Retrieves POSM stock levels by section based on the provided query parameters.

GET /posm/posm-status: Retrieves the status of POSM items based on the provided query parameters.

GET /posm/posm-report: Generates a report for POSM stock levels and statuses based on the provided query parameters.

POST /posm/nill-stock-by-section: Marks POSM stock as nil by section based on the provided details.

POST /posm/stock-by-region: Adds POSM stock levels by region based on the provided details.

### 3.3.14 Product Tree Module Overview

**Overview**

The product-tree module is responsible for managing operations related to the product hierarchy within the microservice. It includes functionalities for creating, updating, and removing products, as well as retrieving product lists with prices and updating product prices.

**Key Responsibilities**

Create Products: Provides functionality to create new product records.

Retrieve Product Lists with Prices: Provides functionality to retrieve lists of products along with their prices.

Update Product Prices: Supports updating the prices of existing products.

**Key Endpoints**

POST /product-tree/product: Creates a new product record based on the provided details.

GET /product-tree/product/list-with-price: Retrieves a list of products along with their prices based on specific query parameters.

PUT /product-tree/update-product-price: Updates the prices of existing products based on the provided details.

### 3.3.15 Distributor ROI Module Overview

**Overview**

The distributor-roi module is responsible for managing operations related to the Return on Investment (ROI) for distributors within the microservice. It includes functionalities for adding, retrieving, and updating income and expense data, as well as fetching income and expense types and lists.

**Key Responsibilities**

Retrieve Income and Expense Types: Provides functionality to retrieve different types of income and expenses.

Add Income and Expense Data: Allows adding new income and expense data for distributors.

Retrieve Income and Expense Data: Supports fetching detailed income and expense data based on specific query parameters.

Retrieve Income and Expense Lists: Provides functionality to retrieve lists of income and expense records.

Update Income and Expense Data: Supports updating existing income and expense data.

Retrieve Sales Commission by Point: Allows retrieval of sales commission data by point.

**Key Endpoints**

GET /roi/income-expense-types: Retrieves different types of income and expenses.

POST /roi/income-expense: Adds new income and expense data based on the provided details.

GET /roi/income-expense: Retrieves detailed income and expense data based on specific query parameters.

GET /roi/income-expense-list: Retrieves lists of income and expense records based on specific query parameters.

PATCH /roi/income-expense: Updates existing income and expense data based on the provided details.

GET /roi/by-point-sales-commision: Retrieves sales commission data by point based on specific query parameters.

### 3.3.16 Section Change Module Overview

**Overview**

The section-change module is responsible for managing operations related to section change requests within the microservice. It includes functionalities for retrieving section change request lists and updating the status of section change requests.

**Key Responsibilities**

Retrieve Section Change Request List: Provides functionality to retrieve a list of section change requests based on specific query parameters.

Update Section Change Request Status: Allows updating the status of existing section change requests.

Manage Section Change Requests: Supports the overall management of section change requests, ensuring that requests are processed efficiently.

**Key Endpoints**

GET /section-change: Retrieves a list of section change requests based on specific query parameters.

PATCH /section-change: Updates the status of an existing section change request based on the provided details.

### 3.3.17 Target Configuration Module Overview

**Overview**

The target-configuration module is responsible for managing operations related to target configurations within the microservice. It includes functionalities for creating targets, retrieving target configurations, and managing target data from various sources such as Excel files.

**Key Responsibilities**

Create Targets: Provides functionality to create new target records.

Retrieve Target Configurations: Allows retrieval of target configurations based on specific query parameters.

Manage Target Data from Excel: Supports the creation and management of target data imported from Excel files.

Retrieve Target Data: Provides functionality to retrieve target data based on specific criteria.

**Key Endpoints**

POST /target/create-target: Creates a new target record based on the provided details.

POST /target/create-target-from-excel: Creates target records from data imported from Excel files.

POST /target/get-target: Retrieves target data based on the provided request details.

POST /target/get-target-configuration: Retrieves target configurations based on the provided request details.

### 3.3.18 User PJP Module Overview

**Overview**

The user-pjp module is responsible for managing Planned Journey Plans (PJP) for users within the microservice. It includes functionalities for creating PJP plans, retrieving PJP categories and lists, approving or rejecting PJP requests, and exporting PJP data.

**Key Responsibilities**

Create PJP Plans: Provides functionality to create new PJP plans for users.

Retrieve PJP Categories: Allows retrieval of PJP categories.

Retrieve PJP Lists: Supports fetching lists of PJP plans based on specific query parameters.

Approve or Reject PJP Requests: Provides functionality to approve or reject PJP requests.

Export PJP Data: Supports exporting PJP data based on specific criteria.

**Key Endpoints**

POST /pjp\_plan/save\_plan\_data: Creates a new PJP plan based on the provided details.

GET /pjp\_plan/category: Retrieves PJP categories.

GET /pjp\_plan/pjp-list: Retrieves a list of PJP plans based on specific query parameters.

PATCH /pjp\_plan/approve-reject-pjp: Approves or rejects a PJP request based on the provided details.

GET /pjp\_plan/export-pjp-data: Exports PJP data based on specific query parameters.

### 3.3.19 User Module Overview

**Overview**

The user module is responsible for managing user-related operations within the microservice. It includes functionalities for creating, updating, retrieving, and deleting user information, as well as managing field force personnel and retrieving various user lists.

**Key Responsibilities**

Create Users: Provides functionality to create new user records.

Update Users: Allows updating existing user records.

Delete Users: Supports the deletion of user records.

Retrieve User Information: Provides functionality to retrieve user information by ID.

Manage Field Force Personnel: Supports creating, updating, retrieving, and deleting field force personnel information.

Retrieve User Lists: Allows retrieval of various user lists, including TSM (Territory Sales Manager) lists and field force lists.

**Key Endpoints**

PATCH /field-force/:id: Updates an existing field force personnel record by its ID.

DELETE /field-force/:id: Deletes a field force personnel record by its ID.

GET /field-force/:id: Retrieves a specific field force personnel record by its ID.

GET /tsm-list: Retrieves a list of Territory Sales Managers (TSM) based on specific query parameters.

GET /field-force-and-tsm-list: Retrieves a combined list of field force personnel and TSMs based on specific query parameters.

### 3.3.20 Vehicle Load Module Overview

**Overview**

The vehicle-load module is responsible for managing operations related to vehicle loads within the microservice. It includes functionalities for retrieving vehicle lists, creating and retrieving vehicle load summaries, and fetching route-wise pre-order lists.

**Key Responsibilities**

Retrieve Vehicle Lists: Provides functionality to retrieve a list of vehicles based on specific query parameters.

Create Vehicle Load Summaries: Allows creating new vehicle load summary records.

Retrieve Vehicle Load Summaries: Supports fetching vehicle load summaries based on specific query parameters.

Retrieve Route-Wise Pre-Order Lists: Provides functionality to retrieve pre-order lists based on routes.

Retrieve Load Summary Details: Supports fetching detailed information about vehicle load summaries.

**Key Endpoints**

GET /vehicle-load/vehicle-list: Retrieves a list of vehicles based on specific query parameters.

POST /vehicle-load/load-summary: Creates a new vehicle load summary record based on the provided details.

GET /vehicle-load/load-summary: Retrieves vehicle load summaries based on specific query parameters.

GET /vehicle-load/route-wise-preorder: Retrieves route-wise pre-order lists based on specific query parameters.

GET /vehicle-load/load-summary-details: Retrieves detailed information about vehicle load summaries based on specific query parameters.

## 3.4 Service 4: Trade Marketing Service

### 3.4.1 Promotion Module Overview

**Overview**

The promotion module is responsible for managing operations related to promotions within the microservice. It includes functionalities for creating promotions, uploading promotion scopes, and handling various types of promotions such as slab promotions and coupons.

**Key Responsibilities**

Create Promotions: Provides functionality to create new promotions based on the provided details.

Create Slab Promotions: Allows creating new slab promotion records.

Upload Promotion Scopes: Supports uploading promotion scope files.

Create Coupons: Allows creating new coupon records.

Create Gifts: Supports creating new gift records.

Retrieve All Promotions: Provides functionality to retrieve a list of all promotions based on specific query parameters.

Retrieve Promotions by Points: Supports fetching promotions based on points.

**Key Endpoints**

POST /promotions/create-promotion: Creates a new promotion record based on the provided details.

POST /promotions/create-slab-promotion: Creates a new slab promotion record based on the provided details.

POST /promotions/upload-promotion-scope: Uploads a promotion scope file.

POST /promotions/create-coupon: Creates a new coupon record based on the provided details.

POST /promotions/create-gift: Creates a new gift record based on the provided details.

GET /promotions/all: Retrieves a list of all promotions based on specific query parameters.

GET /promotions/by-points: Retrieves promotions based on points.

## 3.5 Service 5: Stock management Service

### 3.5.1 QC Management Module Overview

**Overview**

The QC Management module is responsible for managing operations related to quality control (QC) within the microservice. It includes functionalities for inserting QC data, retrieving QC faults, retrieving QC summaries, and deleting QC records.

**Key Responsibilities**

Insert QC Data: Provides functionality to insert new QC records.

Retrieve QC Faults: Supports fetching a list of QC faults based on specific query parameters.

Retrieve QC Summaries: Allows retrieving QC summary records based on specific query parameters.

Delete QC Records: Provides functionality to delete QC records based on specific query parameters.

**Key Endpoints**

POST /qc-management/qc-insert: Inserts new QC records based on the provided details.

GET /qc-management/get-qc-faults: Retrieves a list of QC faults based on specific query parameters.

GET /qc-management/get-qc-summary: Retrieves QC summaries based on specific query parameters.

GET /qc-management/delete-qc: Deletes QC records based on specific query parameters.

### 3.5.2 Shipment API Module Overview

**Overview**

The Shipment API module is responsible for managing operations related to shipments within the microservice. It includes functionalities for creating shipments, synchronizing shipment data, updating shipment information, and retrieving pending shipment data.

**Key Responsibilities**

Create Shipments: Provides functionality to create new shipment records.

Synchronize Shipment Data: Supports synchronizing shipment data based on specific query parameters.

Update Shipment Information: Allows updating shipment information based on specific criteria.

Retrieve Pending Shipment Data: Supports fetching data for pending shipments based on specific query parameters.

**Key Endpoints**

POST /shipment-api/save: Creates new shipment records based on the provided details.

POST /shipment-api/shipment-sync: Synchronizes shipment data based on the provided details.

GET /shipment-api/update-shipment-invoice: Updates shipment information based on specific criteria.

GET /shipment-api/get-pending-shipment-data: Retrieves pending shipment data based on specific query parameters.

### 3.5.3 Stock API Module Overview

**Overview**

The Stock API module is responsible for managing operations related to stock within the microservice. It includes functionalities for retrieving on-hand stock, distributor stock, transferable stock etc.

**Key Responsibilities**

Retrieve Department On-Hand Stock: Provides functionality to retrieve on-hand stock for a specific department.

Retrieve Distributor On-Hand Stock: Supports fetching on-hand stock for distributors.

Retrieve Department Stock by Department and SKU: Allows retrieving stock information for a specific department and SKU.

Retrieve Distributor Transferable Stock: Supports fetching transferable stock for distributors.

Retrieve Pending IGT List: Provides functionality to retrieve a list of pending inter-gate transfers.

Retrieve Pending IGT Details: Supports fetching detailed information about pending inter-gate transfers.

**Key Endpoints**

GET /stock-api/get-dep-onhand-stock: Retrieves on-hand stock for a specific department based on query parameters.

GET /stock-api/get-distributor-onhand-stock: Retrieves on-hand stock for distributors based on query parameters.

GET /stock-api/get-dep-onHandStock-byDep-bySku: Retrieves stock information for a specific department and SKU based on query parameters.

GET /stock-api/get-distributor-transferable-stock: Retrieves transferable stock for distributors based on query parameters.

GET /stock-api/get-pending-igt-list: Retrieves a list of pending inter-gate transfers based on query parameters.

GET /stock-api/get-pending-igt-details: Retrieves detailed information about pending inter-gate transfers based on query parameters.

## 3.6 Service 6: Sales Management Service

### 3.6.1 Web Entry Module Overview

**Overview**

The web-entry module is responsible for managing operations related to web entries within the microservice. It includes functionalities related to submitting house sell.

**Key Responsibilities**

Add Sales Data: Allows adding sales data to the web entry module.

**Key Endpoints**

POST /web-entry/add-sales: Adds sales data to the web entry module based on the provided details.

## 3.7 Service 7: Dashboard Service

### 3.7.1 Dashboard Module Overview

**Overview**

The dashboard module is responsible for managing operations related to the AFBL dashboard within the microservice. It includes functionalities for retrieving volume data, outlet information, preorder delivery information, and monthly target achievements.

**Key Responsibilities**

Retrieve Volume Data: Provides functionality to retrieve volume data based on specific query parameters.

Retrieve Outlet Information: Allows fetching detailed information about outlets.

Retrieve Preorder Delivery Information: Supports fetching preorder delivery information.

Retrieve Preorder Delivery Information by Section: Provides functionality to retrieve preorder delivery information based on sections.

Retrieve Monthly Target Achievement: Supports fetching monthly target achievement data.

Retrieve Date-wise Monthly Target Achievement: Provides functionality to retrieve monthly target achievement data based on specific dates.

**Key Endpoints**

POST /dashboard/get-volume: Retrieves volume data based on the provided details.

POST /dashboard/get-outlet-info: Retrieves detailed information about outlets based on the provided details.

POST /dashboard/get-preorder-delivery-info: Retrieves preorder delivery information based on the provided details.

POST /dashboard/get-preorder-delivery-info-by-section: Retrieves preorder delivery information by section based on the provided details.

GET /dashboard/monthly-target-archievement: Retrieves monthly target achievement data based on specific query parameters.

GET /dashboard/by-date-monthly-target-archievement: Retrieves monthly target achievement data based on specific dates.

### 3.7.2 Cron Job Module Overview

**Overview**

The cron job module is responsible for managing scheduled tasks related to the AFBL dashboard within the microservice. It includes functionalities for preparing sales summary data and preparing target visited data.

**Key Responsibilities**

Prepare Sales Summary Data: Provides functionality to prepare and update sales summary data.

Prepare Target Visited Data: Allows preparing and updating target visited data.

**Key Endpoints**

GET /afbl-dashboard-cron-job/prepare-summary-data: Prepares and updates sales summary data.

GET /afbl-dashboard-cron-job/target-visited: Prepares and updates target visited data.

## 3.8 Service 8: Survey Service

### 3.8.1 Digital Learning Module Overview

**Overview**

The digital-learning module is responsible for managing operations related to digital learning configurations within the microservice. It includes functionalities for retrieving question configurations, inserting survey configurations, and handling various survey-related operations.

**Key Responsibilities**

Retrieve Question Configurations: Provides functionality to retrieve a list of question configurations based on specific query parameters.

Insert Survey Configurations: Allows inserting new survey configuration records.

Store Survey Answers: Supports storing answers for surveys.

Upload Files: Provides functionality to upload files related to surveys.

**Key Endpoints**

GET /digital\_learning/get-question-configuration: Retrieves a list of question configurations based on specific query parameters.

POST /digital\_learning/survey-config-insert: Inserts new survey configuration records based on the provided details.

POST /digital\_learning/store-survey-answer: Stores answers for surveys based on the provided details.

POST /digital\_learning/upload-file: Uploads files related to surveys.

### 3.8.2 Survey Configuration Module Overview

**Overview**

The survey-configuration module is responsible for managing operations related to survey configurations within the microservice. It includes functionalities for retrieving question configurations, inserting survey configurations, sending mid-day updates, and handling various survey-related operations.

**Key Responsibilities**

Retrieve Question Configurations: Provides functionality to retrieve a list of question configurations based on specific query parameters.

Insert Survey Configurations: Allows inserting new survey configuration records.

Send Mid-Day Updates: Supports sending mid-day updates for surveys.

Store Survey Configurations: Handles storing survey configurations based on provided details.

Key Endpoints

GET /survey-configuration/get-question-configuration: Retrieves a list of question configurations based on specific query parameters.

GET /survey-configuration/send\_mid\_day\_update: Sends mid-day updates for surveys based on provided headers.

POST /survey-configuration/survey-config-insert: Inserts new survey configuration records based on the provided details.

POST /survey-configuration: Stores survey configurations based on the provided details.

## 3.9 Service 9: Report Service

### 3.9.1 Bulk Attendance Report Module Overview

**Overview**

The Bulk Attendance Report Module provides an endpoint designed to fetch attendance data for employees within specific departments and business units over a selected date range. This module is essential for generating comprehensive attendance reports based on various filters such as department, role, and timeframe, making it easier for administrators to track and manage attendance records in bulk.

**Key Responsibilities**

* **Attendance Data Retrieval**: Fetch detailed attendance information for employees based on specified business units and departments within a given date range.
* **Filter by Role and Date Range**: Provide flexibility in querying data by allowing filtering by role, department, and business unit, ensuring the relevant data is retrieved for reporting purposes.
* **Comprehensive Reports**: Generate bulk attendance reports to support organizational needs for tracking employee attendance.

**Key Endpoint**

**GET /get-attendance-format**

Retrieves attendance records in bulk for specific business units and departments, filtered by date range and optional role parameters.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which attendance is being fetched.
  + dep\_ids (string): A comma-separated list of department IDs to filter the attendance records.
  + start\_date (date): The start date for the attendance report.
  + end\_date (date): The end date for the attendance report.
  + data\_for (string): The purpose for which the attendance data is being retrieved (e.g., "Excel", "Table View").
  + role (string, optional): An optional filter to retrieve attendance based on specific employee roles within the selected departments.

### 3.9.2 Sales Master Data Report Module Overview

**Overview**

The Sales Master Data Report Module is designed to generate detailed sales reports based on routes, including both house sales and other sales types and promotion data. This module is crucial for understanding sales performance across different routes, departments, and product types, providing a comprehensive view of sales data within a selected date range.

**Key Responsibilities**

* **Sales Data Retrieval by Route**: Fetch sales data for different routes, offering insight into actual sales performance across various routes within the specified timeframe.
* **Filter by Product and Location**: Allow filtering of sales data by product types, specific products, and locations, ensuring accurate and relevant report generation.
* **Route-Wise Analysis**: Include house sales along with other route-based sales for a holistic view of performance, helping in better sales strategy formulation.

**Key Endpoint**

**GET /route-wise-actual-sales**

Retrieves actual sales data by delivery route, filtered by departments, product types, locations, and date range.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which sales data is being fetched.
  + dep\_ids (string): A comma-separated list of department IDs to filter the sales data.
  + start\_date (date): The start date for the sales report.
  + end\_date (date): The end date for the sales report.
  + product\_type (string): The type of product for which the sales report is generated (e.g., "10 for SKU").
  + location\_type (string): The type of location where the sales occurred (e.g., "section", "route").
  + product\_ids (string): A comma-separated list of specific product IDs to filter the sales data.
  + data\_for (string, optional): Specifies the purpose of the data retrieval (e.g., "Excel", "View").

### 3.9.3 Non-Order Outlet Report Module Overview

**Overview**

The Non-Order Outlet Report Module provides an endpoint designed to generate sales reports for outlets that are not placed any order. This report helps track the sales performance of non-order outlets across different departments, locations, and product types within a specified date range. It is crucial for analyzing sales trends, providing insights into direct or on-demand sales at various outlets.

**Key Responsibilities**

* **Non-Order Sales Data Retrieval**: Fetch sales data specifically for outlets that do not participate in the pre-order system.
* **Filter by Product and Location**: Allow filtering of sales data by product types, specific products, and outlet locations to ensure relevant data is included in the report.
* **Outlet-Specific Sales Analysis**: Provide detailed sales performance information for non-order outlets, aiding in the assessment of these outlets' contributions to overall sales.

**Key Endpoint**

**GET /non-pre-order-outlet-report**

Retrieves sales data for non-pre-order outlets, filtered by departments, product types, locations, and date range.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which sales data is being fetched.
  + dep\_ids (string): A comma-separated list of department IDs to filter the sales data.
  + start\_date (date): The start date for the sales report.
  + end\_date (date): The end date for the sales report.
  + product\_type (string): The type of product for which the sales report is generated (e.g., "10 for SKU").
  + location\_type (string): The type of location where the sales occurred (e.g., "outlet").
  + product\_ids (string): A comma-separated list of specific product IDs to filter the sales data.
  + data\_for (string, optional): Specifies the purpose of the data retrieval (e.g., "Excel", "View").

### 3.9.4 Sales By Geo Level Report Module Overview

**Overview**

The Sales By Geo Level Report Module provides an endpoint designed to generate sales reports across various locations such as regions, areas, territories, and specific points. It allows for reporting by both sales volume and value, giving a comprehensive view of sales performance. This module is crucial for analyzing sales across multiple dimensions, including location type and report type, and includes house sales data for a holistic analysis of overall sales performance.

**Key Responsibilities**

* **Location-Based Sales Data Retrieval**: Fetch sales data across multiple location types such as region, area, territory, and point, offering insights into sales performance in different geographic or operational segments.
* **Filter by Product and Report Type**: Allow filtering of sales data by product type and report type, enabling the generation of sales reports by either volume or value, depending on the analysis requirement.
* **Comprehensive Sales Analysis**: Provide detailed sales reports, including house sales, to ensure all types of sales are captured and analyzed for performance tracking.

**Key Endpoint**

**GET /get-sales-view –** Table View of Data

**GET /** **get-sales-report –** Excel Download

Retrieves sales data across various locations (region, area, territory, point), filtered by departments, product types, locations, and report type (volume, value).

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which sales data is being fetched.
  + dep\_ids (string): A comma-separated list of department IDs to filter the sales data.
  + start\_date (date): The start date for the sales report.
  + end\_date (date): The end date for the sales report.
  + product\_type (string): The type of product for which the sales report is generated (e.g., "10 for SKU").
  + location\_type (string): The type of location where the sales occurred (e.g., "region", "area", "territory", "point").
  + type (string, optional): Specifies the report type, either sales volume or sales value.

### 3.9.5 Field Force Sales Performance Report Module Overview

**Overview**

The Field Force Sales Performance Report Module is designed to generate sales performance reports for field forces across various departments and locations. This module is crucial for tracking and analyzing the sales activities of individual or multiple field forces over a specified time range. By filtering data based on product types and locations, the module provides a detailed view of how each field force contributes to overall sales performance.

**Key Responsibilities**

* **Field Force Sales Data Retrieval**: Fetch sales performance data for specific field force members , providing insights into their contribution to sales.
* **Filter by Product and Location**: Allow filtering of sales performance data by product type and location to focus on relevant segments of the market.
* **Performance Analysis**: Enable detailed performance tracking of field force members across different regions, allowing for targeted sales management and strategic planning.

**Key Endpoint**

**GET /field-force-sales-view – Table View**

**GET /** **field-force-sales-excel – Excel Download**

Retrieves sales performance data for field forces, filtered by departments, product types, locations, and field force IDs.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which sales data is being fetched.
  + dep\_ids (string): A comma-separated list of department IDs to filter the sales data.
  + start\_date (date): The start date for the sales performance report.
  + end\_date (date): The end date for the sales performance report.
  + ff\_ids (string): A comma-separated list of field force IDs to filter the sales performance data.
  + product\_type (string): The type of product for which the sales report is generated (e.g., "10 for SKU").
  + location\_type (string): The type of location where the sales occurred (e.g., "route").

### 3.9.6 Route-Wise Delivery Report Module Overview

**Overview**

The Route-Wise Delivery Report Module is designed to generate detailed reports on deliveries made on routes. This module is essential for analyzing delivery performance by examining various factors such as product types and locations. It provides valuable insights into the efficiency of delivery operations across different routes within a specified date range.

**Key Responsibilities**

* **Delivery Data Retrieval by Route**: Fetch delivery data specifically associated with designated routes, enabling businesses to track performance and efficiency along those routes.
* **Filter by Product and Location**: Allow filtering of delivery data based on product types and location types to ensure that the report focuses on the relevant segments of interest.
* **Comprehensive Delivery Analysis**: Provide a detailed view of deliveries made along specific routes, helping identify trends and areas for improvement in delivery operations.

**Key Endpoint**

**GET /route-wise-delivery-view**

Retrieves delivery data for specific routes, filtered by departments, product types, locations, and date range.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which delivery data is being fetched.
  + dep\_ids (string): A comma-separated list of department IDs to filter the delivery data.
  + start\_date (date): The start date for the delivery report.
  + end\_date (date): The end date for the delivery report.
  + product\_type (string): The type of product for which the delivery report is generated (e.g., "10 for SKU, 9 for Brnad").
  + location\_type (string): The type of location associated with the deliveries (e.g., "section", "route").
  + product\_ids (string): A comma-separated list of specific product IDs to filter the delivery data.

### 3.9.7 Inventory Management Report Module Overview

**Overview**

The Stock Management Report Module is essential for overseeing and analyzing stock levels within an organization. This module provides a comprehensive report on stock management, allowing businesses to track their inventory, intransit, shipments, sales, opening, closing, promotion and other critical metrics over a specified period. It aids in optimizing stock levels and improving supply chain efficiency.

**Key Responsibilities**

* **Stock Level Monitoring**: Retrieve and monitor stock levels across different departments, regions, and product types, ensuring effective management of inventory.
* **Detailed Stock Analysis**: Provide insights into various stock-related metrics, including opening and closing stock amounts, shipments, sales, promotions, and damaged stock.
* **Customizable Reporting**: Allow users to customize reports based on pack type (e.g., 'piece' or 'case') and report type (e.g., 'volume' or 'value') to meet specific analytical needs.

**Key Endpoint**

**GET /stock-management-report-** Table View

**GET /stock-management-report-excel –** Excel Download

Generates a detailed report on stock management for specified business units and departments, filtered by date range and product specifications.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which the stock management report is generated.
  + dep\_ids (string): A comma-separated list of department IDs to filter the stock data.
  + start\_date (date): The start date for the stock management report.
  + end\_date (date): The end date for the stock management report.
  + pack\_type (string, optional): The type of packaging for stock items (default is 'piece').
  + type (string, optional): The type of report to generate, either 'volume' or 'value' (default is 'volume').

### 3.9.8 Current Stock Report Module Overview

**Overview**

The Current Stock Report Module is designed to provide real-time visibility into the inventory levels across various departments and locations within an organization. This module allows users to track the current status of stock facilitating effective inventory management and decision-making.

**Key Responsibilities**

* Real-Time Stock Monitoring: Track current stock levels across different SKUs ensuring accurate inventory visibility at all times.
* Detailed Inventory Insights: Provide insights into the quantities of stock available.
* Inventory Discrepancy Reporting: Identify and report discrepancies in stock levels, allowing for quick resolution and minimizing potential stock shortages or overages.

**Key Endpoint**

**GET /stock-closing-report –** Table View

**GET /stock-closing-report-excel –** Excel Download

Generates a report detailing the current stock levels for specified strategic business units (SBUs) and departments.

* Parameters:
  + sbu\_id (string): The identifier for the strategic business unit for which the current stock report is generated.
  + dep\_ids (string): A comma-separated list of department IDs to filter the stock data.
  + start\_date (date): The start date for the stock management report.
  + end\_date (date): The end date for the stock management report.
  + pack\_type (string, optional): The type of packaging for stock items (default is 'piece').

### 3.9.9 Stock Management Report Module Overview

**Overview**

The Stock Management Report Module is essential for overseeing and analyzing stock levels within an organization. This module provides a comprehensive report on stock management, allowing businesses to track their inventory, intransit,shipments, sales, opening, closing, promotion and other critical metrics over a specified period. It aids in optimizing stock levels and improving supply chain efficiency.

**Key Responsibilities**

* **Stock Level Monitoring**: Retrieve and monitor stock levels across different departments, regions, and product types, ensuring effective management of inventory.
* **Detailed Stock Analysis**: Provide insights into various stock-related metrics, including opening and closing stock amounts, shipments, sales, promotions, and damaged stock.
* **Customizable Reporting**: Allow users to customize reports based on pack type (e.g., 'piece' or 'case') and report type (e.g., 'volume' or 'value') to meet specific analytical needs.

**Key Endpoint**

**GET /stock-management-report-** Table View

**GET /stock-management-report-excel –** Excel Download

Generates a detailed report on stock management for specified business units and departments, filtered by date range and product specifications.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which the stock management report is generated.
  + dep\_ids (string): A comma-separated list of department IDs to filter the stock data.
  + start\_date (date): The start date for the stock management report.
  + end\_date (date): The end date for the stock management report.
  + pack\_type (string, optional): The type of packaging for stock items (default is 'piece').
  + type (string, optional): The type of report to generate, either 'volume' or 'value' (default is 'volume').

### 3.9.10 Order Performance Report Module Overview

**Overview**  
The Order Performance Report Module is designed to provide insights into the performance of field force operations over a specified period. It helps assess compliance and effectiveness in fulfilling orders across various strategic business units (SBUs) and departments. This module allows users to generate detailed reports that analyze various metrics, such as ordered outlets, non-ordered outlets, memo counts, and overall productivity rates.

**Key Responsibilities**

* **Performance Tracking:** Evaluate the compliance of the field force in fulfilling orders by analyzing ordered and non-ordered outlets.
* **Detailed Metrics Analysis:** Calculate and report on key performance metrics, including strike rate, non-billed percentages, cooler order taking percentages, and per-ticket value.
* **Flexible Reporting:** Generate reports for specific time frames, SBUs, and departments, allowing for tailored insights.

**Key Endpoint**  
**GET /** **get-pjp-compliance-report**  
Retrieves an order performance report for specified SBUs and departments within a given date range.

**Parameters:**

* **sbu\_id (string):** The identifier for the strategic business unit for which the report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the report data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.
* **ff\_id (string, optional):** The field force ID to filter the report; if not provided, all field forces for the specified SBU and departments will be considered.
* **data\_for (string, optional):** Specifies the context or audience for the report, defaulting to None.

### 3.9.11 SO Attendance Report Module Overview

**Overview**

The SO Attendance Report Module provides an endpoint designed to fetch attendance data for employees within specific departments and business units over a selected date range and field force. This module is essential for generating comprehensive attendance reports based on various filters such as department, role, and timeframe, making it easier for administrators to track and manage attendance records.

**Key Responsibilities**

* **Attendance Data Retrieval**: Fetch detailed attendance information for employees based on specified business units and departments within a given date range.
* **Filter by Role and Date Range**: Provide flexibility in querying data by allowing filtering by role, department, and business unit, ensuring the relevant data is retrieved for reporting purposes.
* **Comprehensive Reports**: Generate bulk attendance reports to support organizational needs for tracking employee attendance.

**Key Endpoint**

**GET /get-attendance-format**

Retrieves attendance records in bulk for specific business units and departments, filtered by date range and optional role parameters.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which attendance is being fetched.
  + dep\_ids (string): A comma-separated list of department IDs to filter the attendance records.
  + start\_date (date): The start date for the attendance report.
  + end\_date (date): The end date for the attendance report.
  + ff\_id(string): field force id

### 3.9.12 TSM Attendance Report Module Overview

**Overview**

The TSM Attendance Report Module provides an endpoint designed to fetch attendance data for employees within specific departments and business units over a selected date range and field force. This module is essential for generating comprehensive attendance reports based on various filters such as department, role, and timeframe, making it easier for administrators to track and manage attendance records.

**Key Responsibilities**

* **Attendance Data Retrieval**: Fetch detailed attendance information for employees based on specified business units and departments within a given date range.
* **Filter by Role and Date Range**: Provide flexibility in querying data by allowing filtering by role, department, and business unit, ensuring the relevant data is retrieved for reporting purposes.
* **Comprehensive Reports**: Generate bulk attendance reports to support organizational needs for tracking employee attendance.

**Key Endpoint**

**GET /get-attendance-format**

Retrieves attendance records in bulk for specific business units and departments, filtered by date range and optional role parameters.

* **Parameters**:
  + sbu\_id (string): The strategic business unit identifier for which attendance is being fetched.
  + dep\_ids (string): A comma-separated list of department IDs to filter the attendance records.
  + start\_date (date): The start date for the attendance report.
  + end\_date (date): The end date for the attendance report.
  + ff\_id(string): field force id

### 3.9.13 PJP Compliance Report Module Overview

**Overview**

The PJP Compliance Report Module is designed to provide detailed insights into the compliance and operational efficiency of the field force in adhering to their Planned Journey Plan (PJP). This module allows users to generate comprehensive reports that track attendance, outlet visits, and distance covered by the field force over a specified time period. The report highlights compliance metrics, such as check-in times, outlet coverage, and average time spent per outlet, across different regions and strategic business units (SBUs).

**Key Responsibilities**

* **Field Force Attendance Tracking**: Monitor the attendance of field force personnel for the given period, including check-in times and locations.
* **Outlet Visit Analysis**: Track the first and last outlet visits each day, including the time and location of those visits.
* **Compliance Metrics Calculation**: Calculate and report metrics such as the number of outlets within and beyond a 25-meter radius and the average time spent per outlet.
* **Detailed Regional Breakdown**: Provide insights across various regions and SBUs, allowing for comprehensive performance evaluation.

**Key Endpoint**

**GET /get-main-pjp-compliance-report**  
Retrieves the PJP compliance report for specified SBUs and departments within a given date range.

**Parameters**

* **sbu\_id** (string): The identifier for the strategic business unit for which the report is generated.
* **dep\_ids** (string): A comma-separated list of department IDs to filter the report data.
* **start\_date** (date): The start date for the reporting period.
* **end\_date** (date): The end date for the reporting period.
* **ff\_id** (string, optional): The field force ID to filter the report; if not provided, all field forces for the specified SBU and departments will be considered.
* **data\_for** (string, optional): Specifies the context or audience for the report, defaulting to None (can be set to 'view' for JSON response or omitted for Excel download).

### 3.9.14 SO Login Log Report Module Overview

**Overview**

The SO Login Log Module is designed to track and report the login activities of sales officers over a specified period. This module provides insights into the attendance and engagement levels of sales personnel across various strategic business units (SBUs) and departments. Users can generate detailed reports that analyze login status, times, and related metrics to evaluate the performance and compliance of sales officers during their operational hours.

**Key Responsibilities**

* **Attendance Monitoring**: Track the login status of sales officers, indicating whether they logged in on a specific date.
* **Time Tracking**: Record and report the login times for each sales officer, providing insights into their engagement levels.
* **Detailed Metrics Analysis**: Analyze the login patterns of sales officers across different regions and departments, allowing for targeted performance evaluations.
* **Flexible Reporting**: Generate reports for specific date ranges and filter by SBUs and departments for tailored insights.

**Key Endpoint**

**GET /get-so-login-log**  
Retrieves the SO login log for specified SBUs and departments within a given date range.

**Parameters**

* **sbu\_id** (string): The identifier for the strategic business unit for which the report is generated.
* **dep\_ids** (string): A comma-separated list of department IDs to filter the report data.
* **start\_date** (date): The start date for the reporting period.
* **end\_date** (date): The end date for the reporting period.
* **data\_for** (string, optional): Specifies the context or audience for the report, defaulting to None (can be set to 'view' for JSON response or omitted for Excel download).

### 3.9.15 Registered Outlet Report Module Overview

**Overview**

The Registered Outlet Module is designed to provide insights into the outlets registered by sales officers (SOs) over a specified period. This module enables users to track the number of outlets each sales officer has registered, allowing for an evaluation of performance and effectiveness in managing customer relationships. By analyzing registered outlets, businesses can assess the productivity of their sales force and make informed decisions regarding resource allocation and sales strategies.

**Key Responsibilities**

* **Outlet Registration Tracking**: Count and report the number of outlets registered by each sales officer, providing insights into individual performance.
* **Performance Analysis**: Evaluate the effectiveness of sales officers in registering new outlets, contributing to overall sales growth.
* **Flexible Reporting**: Generate reports for specific time frames and filter by sales officer IDs to provide targeted insights.
* **Data Integration**: Integrate outlet registration data with other performance metrics to create comprehensive reports for strategic analysis.

**Key Endpoint**

**GET /registered-outlet**  
Retrieves the number of outlets registered by each sales officer for specified strategic business units (SBUs) and departments within a given date range.

**Parameters**

* **sbu\_id** (string): The identifier for the strategic business unit for which the report is generated.
* **dep\_ids** (string): A comma-separated list of department IDs to filter the report data.
* **start\_date** (date): The start date for the reporting period.
* **end\_date** (date): The end date for the reporting period.

### 3.9.16 Outlet Wise Damage Report Module Overview

**Overview**

The Outlet Wise Damage Report Module provides detailed insights into the quality control faults associated with outlets over a specified period. This module allows users to track and analyze the performance of various products across different outlets, facilitating effective quality management. By assessing QC data, businesses can identify trends, evaluate product performance, and enhance overall quality assurance processes.

**Key Responsibilities**

* **Fault Tracking**: Monitor and report on the types and frequencies of quality control faults associated with products in each outlet.
* **Volume and Value Analysis**: Calculate the total volume and value of products with QC issues, enabling effective resource allocation for remediation.
* **Flexible Reporting**: Generate customized reports based on strategic business units (SBUs), departments, product types, and date ranges.
* **Data Integration**: Combine QC data with outlet and product information to provide comprehensive insights for decision-making.

**Key Endpoint**

**GET /qc-report-by-outlet-view –** Table View

**GET /qc-report-by-outlet –** Excel Download

Retrieves a quality control report for specified outlets, products, and departments within a given date range.

**Parameters**

* **sbu\_id** (string): The identifier for the strategic business unit for which the report is generated.
* **dep\_ids** (string): A comma-separated list of department IDs to filter the report data.
* **start\_date** (date): The start date for the reporting period.
* **end\_date** (date): The end date for the reporting period.
* **product\_type** (string): The type of product to filter the report data.
* **product\_ids** (string): A comma-separated list of product IDs to filter the report data.
* **qc\_type** (int, optional): The type of quality control; defaults to 1 if not specified.
* **type** (string, optional): Specifies whether to report by volume or value; defaults to 'volume' if not provided.

### 3.9.17 Route Wise Damage Report Module Overview

**Overview**

The Route Wise Damage Report Module provides detailed insights into the quality control faults associated with routes over a specified period. This module allows users to track and analyze the performance of various products across different outlets, facilitating effective quality management. By assessing QC data, businesses can identify trends, evaluate product performance, and enhance overall quality assurance processes.

**Key Responsibilities**

* **Fault Tracking**: Monitor and report on the types and frequencies of quality control faults associated with products in each route.
* **Volume and Value Analysis**: Calculate the total volume and value of products with QC issues, enabling effective resource allocation for remediation.
* **Flexible Reporting**: Generate customized reports based on strategic business units (SBUs), departments, product types, and date ranges.
* **Data Integration**: Combine QC data with outlet and product information to provide comprehensive insights for decision-making.

**Key Endpoint**

**GET /qc-report-by-section-view –** Table View

**GET /** **qc-report-by-section –** Excel Download  
Retrieves a quality control report for specified outlets, products, and departments within a given date range.

**Parameters**

* **sbu\_id** (string): The identifier for the strategic business unit for which the report is generated.
* **dep\_ids** (string): A comma-separated list of department IDs to filter the report data.
* **start\_date** (date): The start date for the reporting period.
* **end\_date** (date): The end date for the reporting period.
* **product\_type** (string): The type of product to filter the report data.
* **product\_ids** (string): A comma-separated list of product IDs to filter the report data.
* **qc\_type** (int, optional): The type of quality control; defaults to 1 if not specified.
* **type** (string, optional): Specifies whether to report by volume or value; defaults to 'volume' if not provided.

### 3.9.18 Outlet Wise Preorder Live Report Module Overview

**Overview**

The Outlet Wise Preorder Live View Module provides a comprehensive overview of live preorder data for specific outlets. This module allows users to analyze preorder volumes and values of various products, facilitating effective sales forecasting and inventory management. By aggregating sales data across different outlets, businesses can make informed decisions to optimize their preorder strategies and enhance customer satisfaction.

**Key Responsibilities**

* **Live Data Monitoring**: Access real-time preorder data for specified outlets and products, allowing for dynamic business adjustments.
* **Volume and Value Analysis**: Calculate and display total preorder volumes and values for effective inventory planning.
* **Customizable Reporting**: Generate reports based on strategic business units (SBUs), departments, product types, and specific date ranges.
* **Geolocation Integration**: Incorporate geolocation data to enhance the context of preorder analysis, including cooler images and purity scores.

**Key Endpoint**

**GET /outlet-wise-preorder-live-view –** Table View

**GET /** **outlet-wise-preorder-live-excel –** Excel Download  
Retrieves live preorder data for specified outlets, products, and departments within a given date range.

**Parameters**

* **sbu\_id** (string): The identifier for the strategic business unit for which the preorder data is generated.
* **dep\_ids** (string): A comma-separated list of department IDs to filter the preorder data.
* **start\_date** (date): The start date for the reporting period.
* **end\_date** (date): The end date for the reporting period.
* **product\_type** (string): The type of product to filter the preorder data.
* **location\_type** (string): The type of location to filter the preorder data.
* **product\_ids** (string): A comma-separated list of product IDs to filter the preorder data.
* **type** (string, optional): Specifies whether to report by volume or value; defaults to 'volume' if not provided.

### 3.9.19 Route Wise Preorder Live Report Module Overview

**Overview**

The Route Wise Preorder Live View Module provides a comprehensive overview of live preorder data for specific route. This module allows users to analyze preorder volumes and values of various products, facilitating effective sales forecasting and inventory management. By aggregating sales data across different outlets, businesses can make informed decisions to optimize their preorder strategies and enhance customer satisfaction.

**Key Responsibilities**

* **Live Data Monitoring**: Access real-time preorder data for specified route and products, allowing for dynamic business adjustments.
* **Volume and Value Analysis**: Calculate and display total preorder volumes and values for effective inventory planning.
* **Customizable Reporting**: Generate reports based on strategic business units (SBUs), departments, product types, and specific date ranges.
* **Geolocation Integration**: Incorporate geolocation data to enhance the context of preorder analysis, including cooler images and purity scores.

**Key Endpoint**

**GET /route-wise-pre-order-live-view–** Table View

**GET /** **route-wise-pre-order-live-excel –** Excel Download  
Retrieves live preorder data for specified outlets, products, and departments within a given date range.

**Parameters**

* **sbu\_id** (string): The identifier for the strategic business unit for which the preorder data is generated.
* **dep\_ids** (string): A comma-separated list of department IDs to filter the preorder data.
* **start\_date** (date): The start date for the reporting period.
* **end\_date** (date): The end date for the reporting period.
* **product\_type** (string): The type of product to filter the preorder data.
* **location\_type** (string): The type of location to filter the preorder data.
* **product\_ids** (string): A comma-separated list of product IDs to filter the preorder data.
* **type** (string, optional): Specifies whether to report by volume or value; defaults to 'volume' if not provided.

### 3.9.20 Outlet Wise Sales Report Module Overview

**Overview**

The Outlet Wise Delivery Report Module generates detailed reports on delivery performance across various outlets. This module enables businesses to monitor delivery volumes and values for specific products within defined timeframes, allowing for better inventory management and strategic decision-making.

**Key Responsibilities**

* **Delivery Data Retrieval**: Access and aggregate delivery data from specified outlets based on product types and date ranges.
* **Volume and Value Reporting**: Provide insights into total delivery volumes and values to aid in resource allocation and forecasting.
* **Custom Reporting Options**: Generate reports based on strategic business units (SBUs), departments, product types, and selected date ranges, with options for either volume or value metrics.
* **Excel Export**: Create and return an Excel file containing the delivery report for offline analysis.

**Key Endpoint**

**GET /outlet-wise-delivery-report**  
Retrieves delivery data for specified outlets, products, and departments within a given date range, with the option to generate a view or an Excel report.

**Parameters**

* **sbu\_id** (string): The identifier for the strategic business unit for which the delivery report is generated.
* **dep\_ids** (string): A comma-separated list of department IDs to filter the delivery data.
* **start\_date** (date): The start date for the reporting period.
* **end\_date** (date): The end date for the reporting period.
* **product\_type** (string): The type of product to filter the delivery data.
* **location\_type** (string): The type of location to filter the delivery data.
* **product\_ids** (string): A comma-separated list of product IDs to filter the delivery data.
* **data\_for** (string, optional): Specifies the purpose of the data, such as 'view' for on-screen display.
* **type** (string, optional): Specifies whether to report by volume or value; defaults to 'volume' if not provided.

### 3.9.21 Pre Order vs Delivery Report Module Overview

**Overview**  
The Pre Order vs Delivery Report Module generates comprehensive reports detailing pre-order quantities and delivery performance across various outlets. This module allows businesses to assess the effectiveness of their pre-order strategies, monitor product availability, and evaluate sales performance within specific timeframes, enhancing inventory management and decision-making processes.

**Key Responsibilities**

* **Pre Order Data Retrieval:** Fetch and aggregate pre-order data from specified outlets based on department IDs, product types, and date ranges.
* **Delivery Data Integration:** Access and compile delivery data corresponding to pre-orders to provide a complete view of sales performance.
* **Custom Reporting Options:** Generate reports tailored to specific strategic business units (SBUs) and departments, allowing for analysis based on selected date ranges.
* **Excel Export Capability:** Facilitate the creation of downloadable Excel reports for offline data analysis.

**Key Endpoint**  
GET /get-report-view  
Retrieves pre-order and delivery data for specified SBUs, departments, and products within a defined date range, with the option to generate a comprehensive view.

**Parameters**

* **sbu\_id (string):** The identifier for the strategic business unit for which the pre-order report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the pre-order and delivery data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.

### 3.9.22 Memo Report Module Overview

**Overview**  
The Memo Report Module provides detailed insights into memo quantities across various sections or routes. This module allows businesses to analyze track memo data by location, and evaluate product availability within specific date ranges, enhancing inventory management and sales strategies.

**Key Responsibilities**

* **Memo Data Retrieval:** Access and aggregate memo data from specified outlets based on product types, department IDs, and date ranges.
* **Custom Reporting Options:** Generate reports tailored to specific strategic business units (SBUs) and departments, with options for detailed views or downloadable Excel files.
* **Excel Export Capability:** Create and return an Excel file containing the memo report for offline analysis, including formatted headers and totals.

**Key Endpoint**  
**GET /memo-report**  
Retrieves memo data for specified SBUs, products, and departments within a defined date range, providing options for on-screen display or Excel report generation.

**Parameters**

* **sbu\_id (string):** The identifier for the strategic business unit for which the memo report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the memo data.
* **product\_type (string):** The type of product to filter the memo data.
* **product\_ids (string):** A comma-separated list of product IDs to filter the memo data.
* **location\_type (string):** The type of location to filter the memo data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.
* **data\_for (string, optional):** Specifies the purpose of the data, such as 'view' for on-screen display.

### 3.9.23 Trade Offer Report Module Overview

**Overview**  
The Trade Offer Report Module facilitates the retrieval of trade adjustment data, providing insights into adjustments made across various strategic business units (SBUs) and departments. This module allows businesses to analyze trade adjustments within specified date ranges, supporting informed decision-making and effective inventory management.

**Key Responsibilities**

* **Trade Adjustment Data Retrieval:** Access and aggregate trade adjustment data based on SBUs, department IDs, and specified date ranges.
* **Custom Reporting Options:** Generate tailored reports based on report types, with options for detailed views or downloadable formats.
* **Excel Export Capability:** Create and return an Excel file containing the trade adjustment report for offline analysis, including formatted headers and summaries.

**Key Endpoint**  
**GET /trade-adjustement-report**  
Retrieves trade adjustment data for specified SBUs and departments within a defined date range, providing options for on-screen display or Excel report generation.

**Parameters**

* **sbu\_id (string):** The identifier for the strategic business unit for which the trade adjustment report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the trade adjustment data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.
* **report\_type (string, optional):** The type of report to generate, allowing customization based on business needs(Summery, Trade Offer, Special Offer).
* **data\_for (string, optional):** Specifies the purpose of the data, such as 'view' for on-screen display.

### 3.9.24 Secondary Trade Adjustment Report Module Overview

**Overview**  
The 23 Secondary Trade Adjustment Report Module facilitates the retrieval of trade adjustment data, providing insights into adjustments made across various strategic business units (SBUs) and departments. This module allows businesses to analyze trade adjustments within specified date ranges, supporting informed decision-making and effective inventory management.

**Key Responsibilities**

* **Trade Adjustment Data Retrieval:** Access and aggregate trade adjustment data based on SBUs, department IDs, and specified date ranges.
* **Custom Reporting Options:** Generate tailored reports based on report types, with options for detailed views or downloadable formats.
* **Excel Export Capability:** Create and return an Excel file containing the trade adjustment report for offline analysis, including formatted headers and summaries.

**Key Endpoint**  
**GET /trade-adjustment-summary**

Retrieves trade adjustment data for specified SBUs and departments within a defined date range, providing options for on-screen display or Excel report generation.

**Parameters**

* **sbu\_id (string):** The identifier for the strategic business unit for which the trade adjustment report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the trade adjustment data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.
* **report\_type (string, optional):** The type of report to generate, allowing customization based on business needs.
* **data\_for (string, optional):** Specifies the purpose of the data, such as 'view' for on-screen display.

### 3.9.25 By Point Pending Report Module Overview

**Overview**  
The By Point Pending Report Module enables the retrieval of pending sales data by points across various locations. This module allows businesses to track pending volumes for specific departments and SBUs, providing valuable insights for monitoring sales performance and pending deliveries within a specified date range.

**Key Responsibilities**

* **Pending Data Retrieval:** Aggregate pending sales data based on departments, SBUs, and product lists within a defined date range.
* **Custom Reporting Options:** Generate reports tailored to specific SBUs and departments, with options for detailed on-screen views or downloadable Excel reports.
* **Cumulative Totals Calculation:** Calculate and display cumulative totals for product sales across different locations.
* **Excel Export Capability:** Create and return an Excel file containing the pending sales report, with formatted headers and detailed product-level data for offline analysis.

**Key Endpoint**  
GET /by-point-pending-report  
Retrieves pending sales data by points for specified SBUs, departments, and products within a defined date range, with options for on-screen display or Excel report generation.

**Parameters**

* **sbu\_id (string):** The identifier for the strategic business unit for which the pending report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the pending sales data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.
* **data\_for (string, optional):** Specifies the purpose of the data, such as 'view' for on-screen display.

### 3.9.26 Secondary Sales Report Module Overview

**Overview**  
Secondary Sales Report retrieves date-wise secondary sales data for specific products over a given date range and performs various calculations such as cumulative sales, daily averages, targets, achievements, and growth comparisons with the previous year.

**Key Responsibilities:**

1. **Fetch Product Sales Data**:  
   Retrieves daily sales data from the salesReportService, including current sales, house sales, target data, and last year's sales.
2. **Cumulative and Date-Wise Calculations**:  
   Performs cumulative sum calculations for volumes and values of product sales over the specified date range.
3. **Target and Achievement Calculations**:  
   Calculates the achievement percentage based on the sales vs. targets and also calculates RADT.
4. **Growth Comparison with Last Year**:  
   Compares the current year sales with the same period in the previous year to calculate growth percentages.
5. **Special Handling for Sales Days**:  
   Handles specific conditions such as counting the number of working days (excluding Fridays).

**Key Endpoints and Parameters:**

* **GET /date-wise-secondary-sales**
  + **Parameters**:
    - sbu\_id (str): Strategic Business Unit ID to filter sales data.
    - dep\_ids (str): Department IDs to filter sales data.
    - start\_date (date): Start date for the sales data query.
    - end\_date (date): End date for the sales data query.
    - type (str, optional): Specifies the type of data ('volume' by default).
    - data\_for (str, optional): Specifies if the data is for 'view'.

### 3.9.27 PSR Sales Report Module Overview

**Overview**  
The PSR Sales Report Module provides detailed insights into sales performance across various sections or routes. It enables businesses to analyze sales data for different products and locations within a specific date range, offering both volume and value-based reporting. This module supports strategic planning and inventory optimization by delivering customizable reports.

**Key Responsibilities**

* **Sales Data Retrieval:** Access and aggregate sales data based on SBUs, department IDs, product types, and locations within a given date range.
* **Custom Reporting Options:** Generate detailed reports based on volume or value for on-screen display or downloadable formats.
* **Excel Export Capability:** Create and export an Excel report for offline analysis, with formatted headers and totals for easy review of sales data.

**Key Endpoint**  
**GET /psr-sales-report**  
Retrieves route-wise or location-wise sales data for specified SBUs, products, and departments within a defined date range, providing options for on-screen display or Excel report generation.

**Parameters**

* **sbu\_id (string):** The identifier for the strategic business unit for which the PSR sales report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the sales data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.
* **product\_type (string):** The type of product to filter the sales data.
* **location\_type (string):** The type of location (e.g., route, section) for which the sales data is reported.
* **product\_ids (string):** A comma-separated list of product IDs to filter the sales data.
* **data\_for (string, optional):** Specifies the purpose of the data, such as 'view' for on-screen display.
* **type (string, optional):** Specifies whether the report should be based on 'volume' or 'value'. Default is 'volume'.

### 3.9.28 Achievement by Rank Report Module Overview

**Overview**  
The Achievement by Rank Report Module provides a comprehensive analysis of sales achievements segmented by rank across various locations and departments against last year . This module allows businesses to evaluate their sales performance against set targets over a specific date range, facilitating strategic planning and performance management. The reports can be generated based on volume or value metrics and are available for on-screen display or downloadable Excel formats.

**Key Responsibilities**

* **Sales Achievement Calculation:** Retrieve and calculate sales achievements based on current and last year's data for specific SBUs and departments within a defined date range.
* **Target Data Aggregation:** Access target data to compare actual sales against set goals, providing insights into performance gaps.
* **Location-Based Insights:** Generate location-specific sales performance reports, enabling businesses to identify high-performing areas and those needing improvement.
* **Excel Export Capability:** Create and export an Excel report with formatted headers and calculated totals for offline analysis of sales achievements.

**Key Endpoint**  
**GET /achievement-by-rank**  
Retrieves sales achievement data by rank for specified SBUs and departments, offering insights into sales performance against targets over a defined date range.

**Parameters**

* **sbu\_id (string):** The identifier for the strategic business unit for which the achievement report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the sales data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.
* **location\_type (string):** The type of location (e.g., route, section) for which the sales data is reported.
* **type (string, optional):** Specifies whether the report should be based on 'volume' or 'value'. Default is 'volume'.
* **data\_for (string, optional):** Specifies the purpose of the data, such as 'view' for on-screen display.

### 3.9.29 Point Pending Report Module Overview

**Overview**  
The Point Pending Report Module provides a detailed analysis of pending stock across various points (distributors) for specific strategic business units (SBUs) and departments. This module enables businesses to assess stock levels, shipments, and remaining quantities against sales targets over a specified date range. It supports both on-screen display and downloadable Excel reports for offline analysis, facilitating effective inventory management and decision-making.

**Key Responsibilities**

* **Stock Data Retrieval:** Access and aggregate stock data from external sources based on the specified SBUs and departments within a given date range.
* **Target Data Aggregation:** Retrieve target data to compare actual stock against targets for informed decision-making.
* **Location-Based Insights:** Generate location-specific reports that highlight stock levels, shipments, and remaining quantities across different distributors.
* **Excel Export Capability:** Create and export an Excel report with formatted headers and calculated totals for offline analysis of stock and sales achievements.

**Key Endpoint**  
GET /point-pending-report  
Retrieves pending stock data by distributor for specified SBUs and departments, offering insights into stock availability against sales targets over a defined date range.

**Parameters**

* **sbu\_id (string):** The identifier for the strategic business unit for which the point pending report is generated.
* **dep\_ids (string):** A comma-separated list of department IDs to filter the stock data.
* **start\_date (date):** The start date for the reporting period.
* **end\_date (date):** The end date for the reporting period.
* **data\_for (string, optional):** Specifies the purpose of the data, such as 'view' for on-screen display.

## 3.11 Web frontend service

### 3.11.1 Login Overview

**Overview**

The Login component is a crucial part of the authentication system in a React application. It provides a user interface for logging into the application, handles form submission, and manages the login process including browser information collection and IP address tracking.

**Key Responsibilities**

1. User Authentication: Handle user login by collecting username and password.

2. Form Management: Utilize Ant Design's Form component for input validation and submission.

3. State Management: Use Redux for managing authentication state.

4. Browser Information Collection: Gather and store browser details using Bowser library.

5. IP Address Tracking: Collect and store user's IP address information.

6. Error Handling: Display error notifications for failed login attempts.

**Key Functions**

- handleLogin(values): Manages the login process, including:

- Collecting browser information

- Retrieving IP address data

- Dispatching login action

- Storing browser and IP information in localStorage

- handleChange(name, event): Updates the local state with form input changes.

- onFinishFailed(errorInfo): Logs form submission failures to the console.  
  
  
3.11.2 AccessGuard Overview

**Overview**

The AccessGuard component is a crucial part of a React application designed to manage access control based on user permissions. This component ensures that users can only access routes they have permission for, enhancing security and user experience.

**Key Responsibilities**

1. Permission Checking: Verify if the current route is accessible based on the user's menu permissions stored in local storage.

2. Conditional Rendering: Render child components only if the user has the necessary permissions.

3. Error Handling: Display an error page when access is denied.

**Key Functions**

- isMenuRouteExist(): Checks if the current path exists in the user's menu permissions stored in local storage. It iterates through the menu data, comparing the current path against menu URLs based on the specified pathCheckingType.

This AccessGuard component enhances application security by ensuring that users can only access routes they have permission for, based on their menu data stored in local storage.

### 3.11.3 Dashboard Overview

**Overview**

The Dashboard component is a comprehensive React-based dashboard that integrates multiple sections to provide a holistic view of sales performance, outlet information, and location tracking. It served as the main hub for displaying various business metrics and data visualizations.

**Key Responsibilities**

1. Data Management: Fetch and manage data from multiple API endpoints using custom hooks and services.

2. State Management: Handle complex state for filters, API responses, and user interactions.

3. Layout Composition: Integrate multiple section components (FirstSection, SecondSection, ThirdSection, ForthSection) to create a cohesive dashboard layout.

4. Data Distribution: Pass fetched data to child components for rendering specific dashboard sections.

5. Filtering: Implement a flexible filtering system using TopDropdown component for date ranges and hierarchical selections.

6. Visualization: Incorporate various charts and maps for data representation.

**Key Features**

1. Sales Performance: Display sales metrics, targets, and achievements with interactive charts.

2. Outlet Information: Show visited outlets, order statistics, and geographic distribution.

3. Numeric Distribution: Visualize SKU distribution data with bar charts.

4. Location Tracking: Integrate Google Maps to display real-time location data of field forces.

5. Strike Rate Analysis: Present strike rate and related metrics in card format.

6. Responsive Design: Ensure the dashboard is responsive across different device sizes.

**Key Components**

1. TopDropdown: Manages filters for date ranges and hierarchical data selection.

2. FirstSection: Displays sales performance charts and metrics.

3. SecondSection: Shows outlet visit statistics and summary data.

4. ThirdSection: Presents numeric distribution data with bar charts.

5. ForthSection: Combines strike rate information and location tracking map.

6. LocationMap: Integrates Google Maps for visualizing location data.

7. MapComponent: Handles the rendering of Google Maps with custom markers and info windows.

This Dashboard component provides a comprehensive overview of business operations, combining sales data, outlet information, and location tracking in a single, interactive interface. It enhances decision-making by presenting complex data in an easily digestible format through various charts, maps, and summary cards.

### 3.11.4 Retailers Overview

**Overview**

The GeneratedOutletList component is a React-based module designed to display and manage a list of generated outlets or retailers. It provides a comprehensive interface for viewing, filtering, and exporting outlet data, enhancing the management of retail locations within a larger system.

**Key Responsibilities**

1. Data Fetching: Retrieve generated outlet list data from an API based on selected filters.

2. State Management: Handle complex state for table data, pagination, and filter settings.

3. Data Presentation: Display outlet data in a paginated, sortable, and filterable table format.

4. Search Functionality: Implement column-specific search capabilities for efficient data lookup.

5. Data Export: Provide functionality to export the outlet list data to Excel format.

6. Filtering: Integrate a multi-select dropdown for department-based filtering.

**Key Features**

1. Table Display: Present outlet data in a responsive, paginated table with sortable columns.

2. Advanced Filtering: Allow users to filter data based on multiple departments using a custom dropdown.

3. Search Capability: Provide column-specific search functionality for precise data retrieval.

4. Excel Export: Enable users to download the filtered data in Excel format.

5. Loading States: Display loading indicators during data fetching and export operations.

6. Responsive Design: Ensure the component is usable across various device sizes.

**Key Functions**

1. getGeneratedOutletList(): Fetch outlet data based on selected department IDs.

2. getExcelData(): Handle the export of data to Excel format.

3. handleSearch(): Manage the search functionality for individual columns.

4. getColumnSearchProps(): Configure search properties for table columns.

5. getDepId(): Process department selection data from the filter dropdown.

3.11.5 Field Force Management Overview  
**Overview**

1. CreateRecord Component:

- Handles creation and editing of field force records

- Manages form state for various fields (name, contact, email, etc.)

- Integrates with location data for SBU and territory selection

- Handles form submission, including data validation and API calls

- Provides different behavior for create and edit modes

2. FieldForceMapping Component:

- Manages the assignment of field forces to sections

- Displays a table of sections with assigned field forces

- Allows selection and assignment of field forces to sections

- Provides functionality to remove field force assignments

- Integrates with department filtering for data retrieval

**Key Features:**

- Form-based data entry for field force details

- Dynamic dropdown population based on hierarchical data (SBU, Territory)

- Integration with Redux for state management

- API integration for data fetching and submission

- Error handling and user feedback using Swal (SweetAlert)

- Responsive design using Ant Design components

- Table-based display of section-field force mappings

- Filtering capabilities for data retrieval

### 3.11.6 Distributor ROI Overview

**Overview:**

These components are part of a Distributor ROI (Return on Investment) management system in a React application. Create.js handles the creation and editing of Distributor ROI records, while Details.js displays detailed information about a specific Distributor ROI in a modal.

**Key Responsibilities:**

1. Create and edit Distributor ROI records (Create.js)

2. Display detailed Distributor ROI information (Details.js)

3. Manage form state and data submission

4. Handle user interactions with ROI data

5. Calculate and display income, expense, and profit/loss information

6. Integrate with backend services for data operations

**Key Functions:**

- getData() (both files): Fetches ROI data from an API

- handleAmountChange() (Create.js): Updates income or expense amounts

- handleSaveClick() (Create.js): Saves or updates ROI data

- getByPointSalesCommission() (Create.js): Retrieves commission data

- expenseTypeDataColumns() and incomeTypeDataColumn() (both files): Define table structures for displaying data

### 3.11.7 Section Overview

**Overview**:

The Sections and Create components are part of a larger React application, likely used for managing organizational structures, sections, or territories. The Sections component displays a list of sections in a table, allowing users to view and edit them. The Create component is a form interface for creating or editing sections, including configuring routes and schedules.

**Key Responsibilities:**

1. Sections Component:

- Fetch and display section data in a table.

- Handle editing of sections.

- Manage user interactions with the section list.

2. Create Component:

- Handle section creation and editing.

- Manage form state and submission.

- Configure routes and schedules for sections.

**Key Functions:**

1. Sections Component:

- `getData()`: Fetches section data from an API.

- `handleOnEditBtnClick()`: Handles the edit button click for a section.

- `getColumns()`: Defines the structure of the table columns.

2. Create Component:

- onFinish(): Handles form submission for creating or updating a section.

- handleAddRoute(): Adds a new route to the section.

- fetchEditData(): Retrieves existing section data for editing.

- handleEnableDisable(): Toggles the status (enabled/disabled) of a route.

### 3.11.8 Vehicle Overview

**Overview**:

These components are part of a vehicle management system in a React application. VehicleDetails.js displays detailed information about a specific vehicle in a modal, while Create.js handles the creation and editing of vehicle records.

**Key Responsibilities:**

1. Display vehicle details (VehicleDetails.js)

2. Create and edit vehicle records (Create.js)

3. Manage form state and data submission

4. Handle user interactions with vehicle data

5. Integrate with backend services for data operations

**Key Functions:**

- getData() (VehicleDetails.js): Fetches vehicle details from an API

- onFinish() (Create.js): Handles form submission for creating or updating a vehicle

- fetchFFList() (Create.js): Retrieves vehicle type options

- fetchLineList() (Create.js): Fetches line (SBU) data

- fetchPointList() (Create.js): Retrieves distribution point options based on the selected line

### 3.11.9 Stock Management New Shipment Overview

**Overview**:

The NewShipment component is a complex React component used to manage stock shipments or open stock entries in an inventory management system. It provides a user interface for viewing and updating stock quantities for various SKUs (Stock Keeping Units).

**Key Responsibilities:**

1. Display and manage stock data for multiple SKUs

2. Handle user input for shipment quantities or opening stock entries

3. Sync data with external systems (via API calls)

4. Validate and submit shipment or opening stock data

5. Manage UI states (loading, errors, modals)

**Key Functions:**

- updateList(): Updates the SKU list with stock-in-hand data

- handleInputChange(): Manages changes in shipment quantity inputs

- handleFilter(): Fetches and processes on-hand stock data

- handleSyncDataClick(): Syncs data with external systems

- handleSubmit(): Submits the shipment or opening stock data

**Notable Features:**

- Uses a modal for confirming shipment or stock entry submission

- Implements a table for displaying SKU data with editable shipment quantities

- Provides options for syncing data and managing invoices

- Calculates and displays total volumes for on-hand stock and shipment quantities

- Implements conditional rendering based on whether it's a new shipment or opening stock entry

This component is part of a larger inventory management system, providing crucial functionality for managing stock movements and initial stock setup. It integrates with backend services for data retrieval and submission, and uses various UI components to create an interactive and user-friendly interface.

### 3.11.10 Stock Management Reports Overview

**Overview**:

These components are part of a reporting system & create new shipments in a React application. They handle different types of reports: Shipment Report, Stock Management Report, and Pending Shipment Report. Each component provides a user interface for filtering data, displaying results in a table format, and exporting data to Excel or CSV.

**Key Responsibilities:**

1. Fetch and display report data based on user-selected filters

2. Render data in tabular format

3. Provide options for exporting data (Excel/CSV)

4. Handle date range selections

5. Manage loading states and error handling

**Key Functions:**

- getFilterData(): Fetches report data based on selected filters (all components)

- handleExportBtn() / getExcelData(): Handles exporting data to Excel (ShipmentReport and StockManagementReportComponent)

- generateTable(): Generates table data for Stock Management Report

- onDateChange() / onDateRangeChange(): Handles date range selection

**Common Features:**

- Use of Ant Design components (Table, DatePicker, Spin, etc.)

- Integration with backend services for data fetching

- Error handling using SweetAlert2

- Responsive table designs

- Date range selection for report filtering

### 3.11.11 Damage Management Overview

#### 1. Point Damage Entry component:

**Overview:**

The QCEntry component is a complex React component used for managing Quality Control (QC) entries or damage reports in an inventory management system. It provides a user interface for viewing and updating QC data for various SKUs (Stock Keeping Units) across different fault categories.

**Key Responsibilities:**

1. Display and manage QC data for multiple SKUs and fault types

2. Handle user input for QC quantities

3. Fetch and display route information

4. Validate and submit QC entry data

5. Manage UI states (loading, errors, date ranges)

**Key Functions:**

- handleSubmit(): Processes and submits the QC entry data

- updateTableBasedOnRouteAndPoint(): Fetches and processes SKU and QC data to populate the table

- handleInput(): Manages changes in QC quantity inputs

- handleFilter(): Triggers data fetch based on selected filters

- getRoutes(): Fetches available routes for the selected distribution point

**Notable Features:**

- Uses a dynamic table structure that adapts to different fault categories

- Implements date range selection for QC entries

- Provides validation against the last submission date

- Calculates and submits total values based on SKU prices

- Supports different page types (e.g., "Damage Entry" vs other QC types)

#### 2. Damage Verification component

**Overview**:

The DamageVerification component is a complex React component used for verifying and managing damage reports in an inventory management system. It provides a user interface for reviewing, updating, and submitting verified damage data for various SKUs (Stock Keeping Units) across different fault categories.

**Key Responsibilities:**

1. Display and manage damage verification data for multiple SKUs and fault types

2. Handle user input for verified damage quantities and remarks

3. Generate dynamic table columns based on fault categories

4. Validate and submit verified damage data

5. Manage UI states (loading, errors, date selection)

**Key Functions:**

- getColumns(): A memoized function that generates dynamic table columns based on fault categories

- handleInput(): Manages changes in verified damage quantities and remarks inputs

- handleSubmit(): Processes and submits the verified damage data

- getData(): Fetches and processes SKU and damage data to populate the table

#### 3. Browse Verified Damage component

**Overview:**

The BrowseVerifiedDamage component is a React component designed to display and manage verified damage reports in an inventory management system. It provides a user interface for viewing and exporting damage data for various SKUs (Stock Keeping Units) across different fault categories.

**Key Responsibilities:**

1. Fetch and display verified damage data for multiple SKUs and fault types

2. Generate dynamic table columns based on fault categories

3. Provide date-based filtering of damage reports

4. Enable exporting of damage data to Excel format

5. Calculate and display totals for each fault category

**Key Functions:**

- getColumns(): Generates dynamic table columns based on fault categories and damage data

- getData(): Fetches and processes verified damage data to populate the table

- getExcelData(): Prepares and exports damage data to Excel format

- handleClick(): Triggers the Excel export functionality

- onDateChange(): Handles date selection for filtering damage reports

#### 4. Damage Report Damage component

**Overview:**

The DamageReport component is a React component designed to generate and display damage reports in an inventory management system. It provides a user interface for filtering, viewing, and exporting damage data across various SKUs and locations.

**Key Responsibilities:**

1. Fetch and display damage report data based on user-selected filters

2. Generate dynamic table columns based on the report structure

3. Handle different report types (e.g., by piece or by value)

4. Enable exporting of damage report data to Excel format

5. Manage loading states and error handling

**Key Functions:**

- getFilterData(): Fetches damage report data based on selected filters and processes it for display

- getExcelData(): Prepares and exports damage report data to Excel format

- handleColumnGeneration(): Generates dynamic table columns based on the report data structure

- processDataForDisplay(): Transforms the API response data into a format suitable for the table

### 3.11.12 Product Management Overview

**Overview**:

These components are part of a product management system in a React application. They provide different views for browsing and visualizing product-related data, including brands, SKUs, and a hierarchical product tree.

**Key Responsibilities:**

1. Fetch and display product data (brands, SKUs, product hierarchy)

2. Render data in various formats (tables, tree structure)

3. Handle loading states and error scenarios

4. Provide interactive elements for data exploration

**Key Functions:**

- getBrands() / getSKUs() / getAllProductTree(): Fetch respective product data from backend services

- returnBadge() (BrowseProducts): Generates status badges for brands

- updateData() (BrowseSkus): Processes SKU data for display

- renderForeignObjectNode() (BrowseProductsTree): Renders custom nodes for the product tree

Common Features:

- Use of Ant Design components (Table, Card, Spin)

- Integration with Redux for accessing global state (sbu\_id, dep\_ids)

- Responsive design considerations

- Error handling and loading state management

Unique Aspects:

- BrowseProducts: Displays brand information with status badges

- BrowseSkus: Processes complex price data structures for display

- BrowseProductsTree: Utilizes react-d3-tree for an interactive, hierarchical view of products

### 3.11.13 Data Entry Overview

**Overview**:

These components are part of a stock management and sales reporting system in a React application. WebEntry handles stock rotation and sales entry, while HouseSalesReport generates and displays house sales reports.

**Key Responsibilities:**

1. WebEntry:

- Manage stock rotation and sales entry

- Handle complex promotion calculations

- Update and display current stock data

- Process and submit sales data

2. HouseSalesReport:

- Fetch and display house sales report data

- Generate dynamic table columns based on report structure

- Enable exporting of report data to Excel format

**Key Functions:**

- handleSingleTableSaleInput() (WebEntry): Processes sales input and calculates promotions

- updateList() and getCurrentStockData() (WebEntry): Fetch and update current stock data

- getFilterData() (HouseSalesReport): Fetches and processes house sales report data

- getExcelData() (HouseSalesReport): Prepares and exports report data to Excel

**Common Features:**

- Use of Ant Design components (Table, Spin)

- Integration with backend services for data retrieval and submission

- Error handling using SweetAlert2

- Loading state management

**Unique Aspects:**

- WebEntry: Complex promotion calculation logic, stock management

- HouseSalesReport: Dynamic column generation, Excel export functionality

### 3.11.14 Survey Overview

**Overview**:

These components form a comprehensive survey management system within a React application. They handle survey creation, listing, and reporting functionalities.

**Key Components:**

1. ServeyMain:

- Manages survey creation and configuration

- Handles checkbox-based options for survey setup

- Integrates with FilterDropDownMulti for location selection

2. SurveyList:

- Displays a list of existing surveys

- Provides functionality for cloning, updating, and deleting surveys

- Implements a modal for survey scope configuration

- Handles bulk upload and location-wise survey assignment

3. SurveyReportComponent:

- Generates survey reports based on selected filters

- Implements date range selection for report generation

- Allows export of survey data to Excel format

**Key Features:**

- Dynamic form generation for survey configuration

- Integration with backend services for data retrieval and submission

- Complex state management for multi-step processes

- Date range selection for filtering survey data

- Excel export functionality for survey reports

- Modal-based interfaces for additional actions (e.g., scope configuration)

### 3.11.15 TA/DA Overview

**Overview**:

These components form a Travel Allowance and Daily Allowance (TA/DA) management system within a React application. They handle browsing existing TA/DA entries and creating new ones.

**Key Components:**

1. BrowseTada:

- Displays a list of existing TA/DA entries

- Allows editing of TA/DA amounts for individual entries

- Implements date-based filtering of TA/DA data

- Handles saving of edited TA/DA amounts

2. CreateTada:

- Provides an interface for creating new TA/DA entries

- Allows bulk entry of TA/DA amounts for multiple routes/employees

- Implements filtering by distribution points

- Handles saving of new TA/DA entries

**Key Features:**

- Dynamic table generation for TA/DA entries

- In-line editing of TA/DA amounts

- Integration with backend services for data retrieval and submission

- Date-based and location-based filtering of data

- Confirmation dialogs for important actions (e.g., saving data)

**Key Functions:**

- handleAmountChange(): Manages changes in TA/DA amounts

- handleEdit(), handleSave(), handleCancel(): Control the editing flow for TA/DA entries

- getData(): Fetches TA/DA data from the backend

- tableColumnData(): Generates dynamic table columns

### 3.11.16 Vehicle Load Overview

**Overview:**

These components form a Vehicle Load Summary system within a React application. They handle browsing existing load summaries and creating new ones for vehicle deliveries.

**Key Components:**

1. LoadSummaryList:

- Displays a list of existing vehicle load summaries

- Provides filtering options by date and location

- Allows viewing detailed information for each load summary

- Enables printing of delivery chalans and outlet-wise memos

2. Create:

- Manages the creation of new vehicle load summaries

- Handles selection of vehicles, routes, and pre-orders

- Processes SKU-level data for each order

- Implements stock checking and overstock warnings

- Submits load summaries to the backend

**Key Features:**

- Dynamic table generation for load summaries and SKU details

- Complex filtering and search functionality

- Date-based and location-based data retrieval

- Integration with backend services for data fetching and submission

- Stock level checks and warnings for overstocking

- PDF generation for delivery chalans and memos

- CSV export of load summary data

**Key Functions:**

- getColumnSearchProps(): Implements column-level search functionality

- handleInput(): Manages changes in SKU quantities and calculates remaining stock

- fetchVehicleInfo(): Retrieves vehicle and route information

- getPreOrderDetails(): Fetches and processes pre-order data

- submitLoadSummary(): Handles the submission of load summaries with stock checks

### 3.11.17 Target Overview

**Overview:**

These components form a comprehensive target management system within a React application. They handle target creation, reporting, and editing functionalities.

**Key Components:**

1. Targets (not fully visible in the provided snippets):

- Likely handles the creation and management of targets

2. TargetReport:

- Generates and displays target achievement reports

- Allows filtering by date, target type, and location

- Provides Excel export functionality for reports

3. EditTargetList:

- Displays a list of existing target configurations

- Allows editing and deletion of target configurations

- Provides Excel export functionality for individual target data

**Key Features:**

- Dynamic table generation for target reports and lists

- Complex filtering and search functionality

- Date-based and location-based data retrieval

- Integration with backend services for data fetching and submission

- Excel export capabilities for reports and target data

- Deletion of target configurations with confirmation dialogs

**Key Functions:**

- getData() (TargetReport): Fetches and processes target report data

- getExcelData() (TargetReport & EditTargetList): Prepares and exports data to Excel format

- handleDelete() (EditTargetList): Manages the deletion of target configurations

- generateTable() (EditTargetList): Fetches and processes target list data

- getColumnSearchProps() (EditTargetList): Implements column-level search functionality

### 3.11.18 Trade Promotion Overview

**Overview:**

These components form a comprehensive Trade Promotion Management system within a React application. They handle different types of promotions including gift promotions, regular promotions (with bulk upload capability), and try-before-you-buy promotions.

**Key Components:**

1. GiftPromotion:

- Manages the creation of gift promotions

- Uses BasicSetup, PromotionSetup, and ScopeSetup components

- Allows multiple promotions to be added

- Handles saving of promotion data to the backend

2. RegularPromotionBulkUpload:

- Handles bulk upload of regular promotions via Excel file

- Provides file upload functionality with validation

- Manages scope setup for uploaded promotions

- Includes basic setup and scope setup

3. TryBeforeBuy:

- Manages the setup of try-before-you-buy promotions

- Manages the creation of "Try Before You Buy" promotions

- Uses BasicSetup, PromotionSetup, and ScopeSetup components

- Handles saving of promotion data with specific default values

- Manages scope setup for the promotion

**Key Functions:**

- handleSave(): Manages the saving process for promotions

- handleExcelUpload(): Handles the upload and processing of Excel files for bulk promotions

- handleFinalDataChange(): Updates the final data state with new values

Unique Aspects:

- GiftPromotionConfiguration: Allows for multiple promotion configurations

- RegularPromotionBulkUpload: Includes file upload and preview functionality

- TryBeforeBuy: Has predefined default data for try-before-you-buy promotions

### 3.11.19 Journey Plan Overview

**Overview**:

This component displays a list of journey plans and provides functionality for viewing, filtering, and managing these plans.

**Key Features:**

1. Date selection for filtering journey plans

2. Status filtering (All, Pending, Approved, Rejected)

3. Table display of journey plans with detailed information

4. Modal for viewing detailed information about a specific journey plan

5. Integration with FilterDropDownMulti for additional filtering options

**Key Functions:**

1. getColumns(): Defines the structure of the table columns

2. handleOnViewBtnClick(): Opens a modal with detailed information about a selected journey plan

3. getData(): Fetches journey plan data from the server based on selected filters

4. onDateChange(): Handles date selection for filtering

### 3.11.20 Leave Management Overview

**Overview:**

These components form a comprehensive Leave Management system within a React application. They handle viewing existing leave records, creating new leave balances, and managing bulk uploads of leave data.

**Key Components:**

1. LeaveManagement:

- Displays a list of leave records

- Provides filtering options by date range, leave type, and status

- Allows viewing detailed information for each leave record

2. CreateRecord (Leave Balance):

- Manages the creation and editing of leave balances for field forces

- Handles bulk upload of leave balance data

- Provides individual leave balance entry for different leave types

**Key Features:**

1. LeaveManagement:

- Dynamic table generation for leave records

- Complex filtering and search functionality

- Date range selection for filtering leave data

- Excel export of leave management data

- Modal for viewing detailed leave information

2. CreateRecord:

- Bulk upload functionality for leave balances

- Sample file download for bulk upload

- Individual leave balance entry for multiple leave types

- Dynamic form generation based on selected field force and date range

- Validation and error handling for form submission

**Key Functions:**

1. LeaveManagement:

- getData(): Fetches leave management data based on selected filters

- getExcelData(): Prepares and exports data to Excel format

- handleOnViewBtnClick(): Opens a modal with detailed leave information

- getColumns(): Defines the structure of the table columns

2. CreateRecord:

- handleExcelUpload(): Processes and submits bulk upload data

- fetchCategoryList(): Retrieves leave types and existing leave balances

- handleInput(): Manages changes in leave balance inputs

- onFinish(): Handles form submission for individual leave balance entry

- sampleDownloadBtn(): Generates and downloads sample Excel file for bulk upload

### 3.11.21 Asset Management Overview

**Overview:**

These components form part of a React application's Asset Management and Reporting system. They handle viewing asset details, generating reports for refurbished cooler services, and managing asset-related data.

**Key Components:**

1. ViewAsset:

- Displays a list of assets with detailed information

- Allows filtering of assets by department

- Provides a modal for viewing detailed asset information

2. RefurbishCoolerServiceReport:

- Generates reports for refurbished cooler services

- Allows filtering by date range and department

- Provides Excel export functionality

3. AssetManagementReport:

- Generates asset management reports

- Allows filtering by department

- Provides Excel export functionality

**Key Features:**

1. Dynamic table generation for asset lists and reports

2. Complex filtering functionality using custom dropdown components

3. Modal for detailed asset information viewing

4. Excel export capabilities for reports

5. Integration with backend services for data fetching and report generation

6. Loading state management with spinners

7. Error handling and user feedback using SweetAlert2

**Key Functions:**

1. ViewAsset:

- tableColumnData(): Defines the structure of the asset table columns

- getData(): Fetches asset data based on selected filters

- handleOnViewBtnClick(): Opens a modal with detailed asset information

2. RefurbishCoolerServiceReport:

- getFilterData(): Fetches report data based on selected filters

- getExcelData(): Prepares and exports data to Excel format

3. AssetManagementReport:

- getFilterData(): Fetches report data based on selected filters

- getExcelData(): Prepares and exports data to Excel format

### 3.11.22 User Journey Overview

**Overview:**

These components form part of a User Journey Plan and Reporting system within a React application. They handle configuring journey plans, listing existing plans, and generating punch history reports.

1. UserJourneyPlanConfigure:

Key Responsibility: Configuring and creating user journey plans.

Key Functions:

- handleSampleDownload(): Downloads a template for journey plan configuration.

- removeFile(): Removes an uploaded file.

2. UserJourneyPlanList:

Key Responsibility: Displaying and managing a list of user journey plans.

Key Functions:

- handlePjpDownload(): Downloads a specific user's journey plan as an Excel file.

- getColumns(): Defines the structure of the table columns for journey plans.

- fetchPjpList(): Retrieves the list of journey plans based on selected filters.

- onDateChange(): Handles date range selection for filtering.

3. PunchHistoryReport:

Key Responsibility: Generating and displaying punch history reports.

Key Functions:

- getFilterData(): Fetches report data based on selected filters.

- getExcelData(): Prepares and exports data to Excel format.

Common Features:

- Use of Ant Design components for UI (Table, DatePicker, Spin, etc.)

- Integration with backend services for data fetching and report generation

- Excel export capabilities

- Complex filtering options (date ranges, multi-select dropdowns)

- Error handling and user feedback using SweetAlert2

These components demonstrate advanced React techniques, including:

- Complex state management for large datasets

- Dynamic rendering of table columns and data

- Integration with backend services for data fetching and report generation

- File download capabilities for Excel reports and templates

- Implementation of reusable components for filtering and layout

Together, these components provide a comprehensive system for managing user journey plans and generating related reports, allowing for efficient planning and monitoring of user activities within an organization.

### 3.11.23 Master Entry Overview

**Overview**:

These components form part of a Product Entry and Price Management system within a React application. They handle listing and managing point geo-locations, SKUs (Stock Keeping Units), and product prices.

1. PointGeoLocation List:

**Key Responsibility**: Displaying a list of point geo-locations.

**Key Functions**:

- getData(): Fetches geo-location data based on selected filters.

- getColumns(): Defines the structure of the table columns for geo-locations.

2. MasterSku:

**Key Responsibility**: Managing SKU entries for products.

**Key Functions**:

- getSkus(): Retrieves the list of SKUs.

- handleNewProduct(): Initiates the process of adding a new product.

- renderModal(): Renders the modal for adding or editing SKU details.

3. PriceModule List:

**Key Responsibility**: Displaying and managing product prices.

**Key Functions:**

- getData(): Fetches product price data based on selected filters.

- getColumns(): Defines the structure of the table columns for product prices.

- handleOnEditBtnClick(): Navigates to the edit page for a specific product price.

- fetchLineList(): Retrieves the list of lines (departments) for filtering.

### 3.11.24 Digital Learning Overview

**Overview**:

These components form a comprehensive Digital Learning management system within a React application. They handle listing, creating, and configuring digital learning content, as well as managing the scope and distribution of learning materials.

**Key Components:**

1. DigitalLearning (Main):

**Key Responsibility**: Creating and editing digital learning content.

**Key Functions**:

- updateArrIdToString(): Converts array IDs to strings for consistent data handling.

- onCheckboxChange(): Handles changes in configuration checkboxes.

- submitScope(): Submits the scope and content of a digital learning item.

2. DigitalLearningList:

**Key Responsibility**: Displaying and managing a list of digital learning content.

**Key Functions:**

- generateTableForShowCurrent(): Generates table data based on selected filters.

- handleClone(): Clones an existing digital learning item.

- submitScope(): Submits the scope configuration for a digital learning item.

**Key Features:**

1. Multi-step configuration process (Basic Configuration, Questions Setup, Preview Questions)

2. File upload functionality for digital learning materials (PDF, video, images)

3. Location-wise or bulk upload options for content distribution

4. Preview functionality for created questions

5. Integration with backend services for data submission and retrieval

6. Complex state management for various configuration options

### 3.11.25 Credential Overview

**Overview**:

This component is responsible for handling the password change functionality within the application. It provides a user interface for users to change their password securely.

**Key Responsibilities:**

1. Render a form for users to input their old password, new password, and confirm the new password.

2. Validate user inputs for password change.

3. Submit the password change request to the server.

4. Display success or error messages based on the server response.

**Key Functions:**

1. onFinish(values):

- Handles form submission

- Prepares the data for the password change request

- Calls the changePassword service function

- Displays success or error messages using SweetAlert2

- Reloads the page on successful password change

2. onFinishFailed(errorInfo):

- Handles form validation failures (currently empty, but can be used for additional error handling)

Key Features:

1. Uses Ant Design's Form and Input components for form handling and UI

2. Implements password confirmation validation

3. Utilizes Redux for accessing user and SBU information

4. Integrates with a backend service (changePassword) for processing the password change request

5. Provides user feedback using SweetAlert2 for success and error messages

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 4. Mobile App: Flutter

## 4.1 Module wise overview

### 4.1.1 Login Module

**Overview**

The login module is responsible for managing operations related to user authentication within the mobile application. It includes functionalities for logging in users, checking app versions, and handling various login-related operations.

**Key Responsibilities**

1. **Login:** Provides functionality to log in users based on their username and password.
2. **Check App Version:** Checks if the app version is up-to-date and prompts the user to update if necessary.
3. **Check Asset Exists:** Checks if the required assets are downloaded and available for use.
4. **Delete PDA from Server:** Deletes PDA files from the server.

**Key Endpoints**

* POST/login: Logs in users based on their username and password.
* GET/app-version: Checks if the app version is up-to-date.
* GET/asset-exists: Checks if the required assets are downloaded and available for use.
* DELETE/pda-delete: Deletes PDA files from the server.

### 4.1.2 Verification Controller

**Overview**

The verification controller is responsible for managing operations related to user verification within the mobile application. It includes functionalities for checking login status, logging in users, and handling various verification-related operations.

**Key Responsibilities**

1. **Check Login Status:** Checks the login status of the user and redirects them to the dashboard or login page accordingly.
2. **Do Login:** Logs in users based on their username and password.
3. **Do Login with Sync File:** Logs in users using a sync file.
4. **Do Login with Sync File and Go to PDA Upload:** Logs in users using a sync file and redirects them to the PDA upload page.
5. **Check App Version:** Checks if the app version is up-to-date and prompts the user to update if necessary.
6. **Check Asset Exists:** Checks if the required assets are downloaded and available for use.
7. **Delete PDA from Server:** Deletes PDA files from the server.

**Key Endpoints**

* POST/login: Logs in users based on their username and password.
* GET/sync-file-login: Logs in users using a sync file.
* GET/sync-file-login-and-upload: Logs in users using a sync file and redirects them to the PDA upload page.
* GET/app-version: Checks if the app version is up-to-date.
* GET/asset-exists: Checks if the required assets are downloaded and available for use.
* DELETE /pda-delete: Deletes PDA files from the server.

### 4.2.1 Outlet Module

**Overview**

The outlet module is responsible for managing operations related to outlet management within the mobile application. It includes functionalities for managing outlet information, handling outlet-related operations, and providing outlet data to other modules.

**Key Responsibilities**

1. **Outlet Management:** Provides functionality to manage outlet information, including creating, updating, and deleting outlets.
2. **Outlet List:** Retrieves a list of outlets and their corresponding information.
3. **Outlet Details:** Retrieves detailed information about a specific outlet.
4. **Outlet Sync:** Synchronizes outlet data with the server to ensure data consistency.

**Key Endpoints**

* GET/outlets: Retrieves a list of outlets and their corresponding information.
* GET/outlet/{id}: Retrieves detailed information about a specific outlet.
* POST/outlet: Creates a new outlet.
* PUT/outlet/{id}: Updates an existing outlet.
* DELETE/outlet/{id}: Deletes an outlet.
* POST/outlet-sync: Synchronizes outlet data with the server.

### 4.2.2 Outlet UI

**Overview**

The outlet UI module is responsible for providing a user-friendly interface for managing outlet information. It includes functionalities for displaying outlet lists, outlet details, and handling outlet-related operations.

**Key Responsibilities**

1. **Outlet List UI:** Displays a list of outlets and their corresponding information.
2. **Outlet Details UI:** Displays detailed information about a specific outlet.
3. **Outlet Sync UI:** Synchronizes outlet data with the server to ensure data consistency.

**Key Endpoints**

* GET/outlet-list: Retrieves a list of outlets and their corresponding information.
* GET/outlet-details/{id}: Retrieves detailed information about a specific outlet.
* POST/outlet-sync: Synchronizes outlet data with the server.

### 4.2.3 Outlet Controller

**Overview**

The outlet controller module is responsible for managing operations related to outlet management within the mobile application. It includes functionalities for handling outlet-related operations, providing outlet data to other modules, and synchronizing outlet data with the server.

**Key Responsibilities**

1. **Outlet Management:** Provides functionality to manage outlet information, including creating, updating, and deleting outlets.
2. **Outlet Sync:** Synchronizes outlet data with the server to ensure data consistency.

**Key Endpoints**

* POST/outlet: Creates a new outlet.
* PUT/outlet/{id}: Updates an existing outlet.
* DELETE/outlet/{id}: Deletes an outlet.
* POST/outlet-sync: Synchronizes outlet data with the server.

### 4.3 Preorder Module

**Overview**

The Preorder module is a critical component of the mobile application, enabling users to select products, calculate promotions, and navigate to the examine UI to view the overall memo. This module is designed to streamline the preorder process, ensuring that users can efficiently and accurately place orders.

**Key Responsibilities**

1. **Product Selection:** Allows users to select products individually and calculate the total cost.
2. **Promotion Calculation:** Calculates the applicable promotions for each selected product, including different types of promotions such as normal, entire memo, multi-buy, and QPS.
3. **Promotion Type:** Determines the promotion type for each product, including payable promotion types such as absolute cash, percentage of value, product discount, and gift.
4. **Examine UI Navigation:** Navigates the user to the examine UI to view the overall memo.
5. **Preorder Saving:** Saves the preorder data, either to the PDA if the device is offline or to both the PDA and server if the device is online.

**Key Endpoints**

* GET /products: Retrieves a list of products and their corresponding information.
* GET /product/{id}: Retrieves detailed information about a specific product.
* POST /preorder: Creates a new preorder.
* PUT /preorder/{id}: Updates an existing preorder.
* DELETE /preorder/{id}: Deletes a preorder.
* GET /promotions: Retrieves a list of promotions and their corresponding information.
* GET /promotion/{id}: Retrieves detailed information about a specific promotion.
* POST /preorder/examine: Navigates the user to the examine UI to view the overall memo.
* POST /preorder/save: Saves the preorder data to the PDA and/or server.

**Key Utils**

* promotion\_utils.dart: Provides functions for calculating promotions and determining promotion types.

**Key Screens**

1. **Examine UI Screen:** Displays the overall memo and allows users to view the preorder details.
2. **Preorder Saving Screen:** Saves the preorder data to the PDA and/or server.

### 4.4 Memo Module

**Overview**

The Memo module is an extension of the Preorder module, allowing users to view and edit memos for outlets that have already completed a preorder. This module is designed to streamline the memo editing process, ensuring that users can efficiently and accurately update memos.

**Key Responsibilities**

1. **Outlet Selection:** The Memo module allows users to select an outlet that has already completed a preorder, and view the corresponding memo.
2. **Memo Editing:** Once an outlet is selected, the module enables users to edit the memo, following the same process as the preorder module.
3. **Product Selection:** Users can select products to add to the memo, with the ability to search, filter, and sort products by various criteria such as product name, category, and price.
4. **Promotion Calculation:** The module calculates the applicable promotions for each selected product, taking into account various promotion types such as normal, entire memo, multi-buy, and QPS.
5. **Promotion Type:** The module determines the promotion type for each product, including payable promotion types such as absolute cash, percentage of value, product discount, and gift.
6. **Memo Saving:** The module saves the updated memo data, either to the PDA if the device is offline or to both the PDA and server if the device is online.

**Key Benefits**

* Improved Efficiency: The Memo module streamlines the memo editing process, reducing the time and effort required to update memos.
* Increased Accuracy: The module's promotion calculation and promotion type determination features ensure that memos are accurate and up-to-date.
* Enhanced User Experience: The module's intuitive interface and navigation features provide a seamless user experience, making it easy for users to edit memos.

**Key Dependencies**

* Preorder Module: The Memo module relies on the Preorder module to retrieve preorder data and calculate promotions.
* Product Module: The Memo module relies on the Product module to retrieve product information and calculate promotions.
* Promotion Module: The Memo module relies on the Promotion module to determine promotion types and calculate promotion amounts.
* PDA Module: The Memo module relies on the PDA module to save memo data when the device is offline.
* Server Module: The Memo module relies on the Server module to save memo data when the device is online.

**Key Screens**

1. Memo UI: The Memo UI screen allows users to select an outlet and view the corresponding memo.
2. Product Selection Screen: The Product Selection screen allows users to select products to add to the memo.
3. Memo Editing Screen: The Memo Editing screen allows users to edit the memo, including selecting products and calculating promotions.
4. Memo Saving Screen: The Memo Saving screen saves the updated memo data to the PDA and/or server.

### 4.4 Delivery Module

**Overview**

The Delivery module is an extension of the Preorder and Memo modules, allowing users to view and sync delivery data from the server for outlets where a load summary has already been created. This module is designed to streamline the delivery process, ensuring that users can efficiently and accurately confirm deliveries.

**Key Features**

1. **Outlet Selection:** The Delivery module allows users to select an outlet where a load summary has already been created, and view the corresponding delivery data.
2. **Sync from Server:** The module enables users to sync delivery data from the server, ensuring that the latest information is available.
3. **Product Changes:** Users can make changes to the products for the selected outlet, including updating quantities or removing products.
4. **Confirm Delivery:** Once changes have been made, users can confirm the delivery for the selected outlet.
5. **Bulk Delivery:** Users can select multiple outlets and save bulk delivery data, streamlining the process for multiple deliveries.
6. **Save Delivery:** The module saves the delivery data, either to the PDA if the device is offline or to both the PDA and server if the device is online.

**Key Benefits**

* Improved Efficiency: The Delivery module streamlines the delivery process, reducing the time and effort required to confirm deliveries.
* Increased Accuracy: The module's ability to sync data from the server ensures that the latest information is available, reducing errors and discrepancies.
* Enhanced User Experience: The module's intuitive interface and navigation features provide a seamless user experience, making it easy for users to confirm deliveries.

**Key Dependencies**

* Preorder Module: The Delivery module relies on the Preorder module to retrieve preorder data.
* Memo Module: The Delivery module relies on the Memo module to retrieve memo data.
* Product Module: The Delivery module relies on the Product module to retrieve product information.
* PDA Module: The Delivery module relies on the PDA module to save delivery data when the device is offline.
* Server Module: The Delivery module relies on the Server module to save delivery data when the device is online.

**Key Screens**

1. Delivery UI: The Delivery UI screen allows users to select an outlet and view the corresponding delivery data.
2. Outlet Selection Screen: The Outlet Selection screen allows users to select multiple outlets for bulk delivery.
3. Product Changes Screen: The Product Changes screen allows users to make changes to the products for the selected outlet.
4. Confirm Delivery Screen: The Confirm Delivery screen allows users to confirm the delivery for the selected outlet.
5. Save Delivery Screen: The Save Delivery screen saves the delivery data to the PDA and/or server.

### 4.5.1 Attendance Module

**Overview**

The attendance module is responsible for managing operations related to attendance management within the mobile application. It includes functionalities for managing attendance information, handling attendance-related operations, and providing attendance data to other modules.

**Key Responsibilities**

1. **Attendance Management:** Provides functionality to manage attendance information, including creating, updating, and deleting attendance records.
2. **Attendance List:** Retrieves a list of attendance records and their corresponding information.
3. **Attendance Details:** Retrieves detailed information about a specific attendance record.
4. **Attendance Sync:** Synchronizes attendance data with the server to ensure data consistency.

**Key Endpoints**

* GET/attendance: Retrieves a list of attendance records and their corresponding information.
* GET/attendance/{id}: Retrieves detailed information about a specific attendance record.
* POST/attendance: Creates a new attendance record.
* PUT/attendance/{id}: Updates an existing attendance record.
* POST/attendance-sync: Synchronizes attendance data with the server.

### 4.5.2 Attendance UI

**Overview**

The attendance UI module is responsible for providing a user-friendly interface for managing attendance information. It includes functionalities for displaying attendance lists, attendance details, and handling attendance-related operations.

**Key Responsibilities**

1. **Attendance List UI:** Displays a list of attendance records and their corresponding information.
2. **Attendance Details UI:** Displays detailed information about a specific attendance record.
3. **Attendance Sync UI:** Synchronizes attendance data with the server to ensure data consistency.

**Key Endpoints**

* GET/attendance-list: Retrieves a list of attendance records and their corresponding information.
* GET/attendance-details/{id}: Retrieves detailed information about a specific attendance record.
* POST/attendance-sync: Synchronizes attendance data with the server.

### 4.5.3 Attendance Controller

**Overview**

The attendance controller module is responsible for managing operations related to attendance management within the mobile application. It includes functionalities for handling attendance-related operations, providing attendance data to other modules, and synchronizing attendance data with the server.

**Key Responsibilities**

1. **Attendance Management:** Provides functionality to manage attendance information, including creating, updating, and deleting attendance records.
2. **Attendance Sync:** Synchronizes attendance data with the server to ensure data consistency.

**Key Endpoints**

* POST/attendance: Creates a new attendance record.
* PUT/attendance/{id}: Updates an existing attendance record.
* DELETE/attendance/{id}: Deletes an attendance record.
* POST/attendance-sync: Synchronizes attendance data with the server.

### 4.5.4 Check In/Out

**Overview**

The check in/out module is responsible for managing operations related to checking in and out of attendance. It includes functionalities for handling check in/out operations and providing check in/out data to other modules.

**Key Responsibilities**

1. **Check In:** Provides functionality to check in attendance.
2. **Check Out:** Provides functionality to check out attendance.

**Key Endpoints**

* POST/check-in: Checks in attendance.
* POST/check-out: Checks out attendance.

### 4.5.5 Geo Fencing

**Overview**

The geo fencing module is responsible for managing operations related to geo fencing attendance. It includes functionalities for handling geo fencing operations and providing geo fencing data to other modules.

**Key Responsibilities**

1. **Geo Fencing:** Provides functionality to manage geo fencing attendance.

**Key Endpoints**

* POST/geo-fencing: Manages geo fencing attendance.

### 4.6.1 Summary Module

**Overview**

The summary module is responsible for managing operations related to sales summary management within the mobile application. It includes functionalities for managing sales summary information, handling sales summary-related operations, and providing sales summary data to other modules.

**Key Responsibilities**

1. **Sales Summary Management:** Provides functionality to manage sales summary information, including creating, updating, and deleting sales summary records.
2. **Sales Summary List:** Retrieves a list of sales summary records and their corresponding information.
3. **Sales Summary Details:** Retrieves detailed information about a specific sales summary record.
4. **Sales Summary Sync:** Synchronizes sales summary data with the server to ensure data consistency.

**Key Endpoints**

* GET/sales-summary: Retrieves a list of sales summary records and their corresponding information.

### 4.6.2 Sales Summary UI

**Overview**

The sales summary UI module is responsible for providing a user-friendly interface for managing sales summary information. It includes functionalities for displaying sales summary lists, sales summary details, and handling sales summary-related operations.

**Key Responsibilities**

1. **Sales Summary List UI:** Displays a list of sales summary records and their corresponding information.
2. **Sales Summary Details UI:** Displays detailed information about a specific sales summary record.
3. **Sales Summary Sync UI:** Synchronizes sales summary data with the server to ensure data consistency.

**Key Endpoints**

* GET/sales-summary-list: Retrieves a list of sales summary records and their corresponding information.

### 4.6.3 Sales Summary Controller

**Overview**

The sales summary controller module is responsible for managing operations related to sales summary management within the mobile application. It includes functionalities for handling sales summary-related operations, providing sales summary data to other modules, and synchronizing sales summary data with the server.

**Key Responsibilities**

1. **Sales Summary Management:** Provides functionality to manage sales summary information, including creating, updating, and deleting sales summary records.
2. **Sales Summary Sync:** Synchronizes sales summary data with the server to ensure data consistency.

### 4.7.1 Asset Module

**Overview**

The asset module is responsible for managing operations related to asset management within the mobile application. It includes functionalities for managing asset information, handling asset-related operations, and providing asset data to other modules.

**Key Responsibilities**

1. **Asset Management:** Provides functionality to manage asset information, including creating, updating, and deleting asset records.
2. **Asset List:** Retrieves a list of asset records and their corresponding information.
3. **Asset Details:** Retrieves detailed information about a specific asset record.
4. **Asset Sync:** Synchronizes asset data with the server to ensure data consistency.

**Key Endpoints**

* GET/assets: Retrieves a list of asset records and their corresponding information.
* GET/assets/{id}: Retrieves detailed information about a specific asset record.
* POST/assets: Creates a new asset record.

### 4.7.2 Asset UI

**Overview**

The asset UI module is responsible for providing a user-friendly interface for managing asset information. It includes functionalities for displaying asset lists, asset details, and handling asset-related operations.

**Key Responsibilities**

1. **Asset List UI:** Displays a list of asset records and their corresponding information.
2. **Asset Details UI:** Displays detailed information about a specific asset record.
3. **Asset Sync UI:** Synchronizes asset data with the server to ensure data consistency.

**Key Endpoints**

* GET/assets-list: Retrieves a list of asset records and their corresponding information.
* GET/assets-details/{id}: Retrieves detailed information about a specific asset record.

### 4.7.3 Asset Controller

**Overview**

The asset controller module is responsible for managing operations related to asset management within the mobile application. It includes functionalities for handling asset-related operations, providing asset data to other modules, and synchronizing asset data with the server.

**Key Responsibilities**

1. **Asset Management:** Provides functionality to manage asset information, including creating, updating, and deleting asset records.
2. **Asset Sync:** Synchronizes asset data with the server to ensure data consistency.

**Key Endpoints**

* PUT/assets/{id}: Updates an existing asset record.
* DELETE/assets/{id}: Deletes an asset record.

### 4.7.4 Asset Requisition

**Overview**

The asset requisition module is responsible for managing operations related to asset requisition within the mobile application. It includes functionalities for handling asset requisition operations and providing asset requisition data to other modules.

**Key Responsibilities**

1. **Asset Requisition:** Provides functionality to manage asset requisition information.

**Key Endpoints**

* GET/asset-requisition: Retrieves asset requisition information.
* POST/asset-requisition: Creates a new asset requisition record.
* PUT/asset-requisition/{id}: Updates an existing asset requisition record.
* DELETE/asset-requisition/{id}: Deletes an asset requisition record.

### 4.7.5 Asset Installation

**Overview**

The asset installation module is responsible for managing operations related to asset installation within the mobile application. It includes functionalities for handling asset installation operations and providing asset installation data to other modules.

**Key Responsibilities**

1. **Asset Installation:** Provides functionality to manage asset installation information

**Key Endpoints**

* GET/asset-installation: Retrieves asset installation information.
* POST/asset-installation: Creates a new asset installation record.
* PUT/asset-installation/{id}: Updates an existing asset installation record.
* DELETE/asset-installation/{id}: Deletes an asset installation record.

### 4.7.6 Asset Pull Out

**Overview**

The asset pull out module is responsible for managing operations related to asset pull out within the mobile application. It includes functionalities for handling asset pull out operations and providing asset pull out data to other modules.

**Key Responsibilities**

1. **Asset Pull Out:** Provides functionality to manage asset pull out information.

**Key Endpoints**

1. GET/asset-pull-out: Retrieves asset pull out information.
2. POST/asset-pull-out: Creates a new asset pull out record.

### 4.8 Sales Submit Module

**Overview**

The sales submit module is responsible for managing operations related to sales data synchronization within the mobile application. It includes functionalities for managing sales data, handling sales data-related operations, and providing sales data to other modules.

The sales submit UI module is responsible for providing a user-friendly interface for managing sales data synchronization. It includes functionalities for displaying sales data, handling sales data-related operations, and synchronizing sales data with the server.

The sales submit controller module is responsible for managing operations related to sales data synchronization within the mobile application. It includes functionalities for handling sales data-related operations, providing sales data to other modules, and synchronizing sales data with the server.

The sync all data module is responsible for managing operations related to synchronizing all sales data within the mobile application. It includes functionalities for handling sales data-related operations and providing sales data to other modules.

The check if sync is successful module is responsible for managing operations related to checking if sales data synchronization is successful within the mobile application. It includes functionalities for handling sales data-related operations and providing sales data to other modules.

**Key Responsibilities**

1. **Sales Data Management:** Provides functionality to manage sales data, including creating, updating, and deleting sales records.
2. **Sales Data UI:** Displays sales data information.
3. **Show Alert For Sync:** Provides functionality to display alerts for sales data synchronization.
4. **Sales Data Sync:** Synchronizes sales data with the server to ensure data consistency.
5. **Sync All Data:** Provides functionality to synchronize all sales data.

**Key Endpoints**

1. POST/sales-submit: Synchronizes sales data with the server.
2. GET/sales-data: Retrieves sales data information.

### 4.8.1 Leave Management Module

**Overview**

The leave management module is responsible for managing operations related to leave management within the mobile application. It includes functionalities for managing leave data, handling leave-related operations and providing leave data to other modules.

**Key Responsibilities**

1. **Leave Management:** Provides functionality to manage leave data, including creating, updating, and deleting leave records.
2. **Leave Sync:** Synchronizes leave data with the server to ensure data consistency.

**Key Endpoints**

1. POST/leave-submit: Synchronizes leave data with the server.
2. GET/leave-data: Retrieves leave data information.

### 4.8.2 Leave UI

**Overview**

The leave UI module is responsible for providing a user-friendly interface for managing leave data. It includes functionalities for displaying leave data, handling leave-related operations, and synchronizing leave data with the server.

**Key Responsibilities**

1. **Leave UI:** Displays leave data information.
2. **Leave Sync UI:** Synchronizes leave data with the server to ensure data consistency.

**Key Endpoints**

1. GET/leave-list: Retrieves leave data information.
2. POST/leave-submit: Synchronizes leave data with the server.

### 4.8.3 Leave Controller

**Overview**

The leave controller module is responsible for managing operations related to leave management within the mobile application. It includes functionalities for handling leave-related operations, providing leave data to other modules, and synchronizing leave data with the server.

**Key Responsibilities**

1. **Leave Management:** Provides functionality to manage leave data, including creating, updating, and deleting leave records.
2. **Leave Sync:** Synchronizes leave data with the server to ensure data consistency

**Key Endpoints**

* POST/leave-submit: Synchronizes leave data with the server.
* GET/leave-data: Retrieves leave data information.

### 4.9.1 Allowance Management Module

**Overview**

The allowance management module is responsible for managing operations related to allowance management within the mobile application. It includes functionalities for managing allowance data, handling allowance-related operations, and providing allowance data to other modules.

**Key Responsibilities**

1. **Allowance Management:** Provides functionality to manage allowance data, including creating, updating, and deleting allowance records.
2. **Allowance Sync:** Synchronizes allowance data with the server to ensure data consistency.

**Key Endpoints**

* POST/allowance-submit: Synchronizes allowance data with the server.
* GET/allowance-data: Retrieves allowance data information.

### 4.9.2 Allowance UI

**Overview**

The allowance management module is responsible for managing operations related to allowance management within the mobile application. It includes functionalities for managing allowance data, handling allowance-related operations, and providing allowance data to other modules.

**Key Responsibilities**

1. **Allowance Management:** Provides functionality to manage allowance data, including creating, updating, and deleting allowance records.
2. **Allowance Sync:** Synchronizes allowance data with the server to ensure data consistency.

**Key Endpoints**

* POST/allowance-submit: Synchronizes allowance data with the server.
* GET/allowance-data: Retrieves allowance data information.

### 4.9.3 Movement Management

**Overview**

The movement management module is responsible for managing operations related to movement management within the mobile application. It includes functionalities for handling movement-related operations and providing movement data to other modules.

**Key Responsibilities**

1. **Movement Management:** Provides functionality to manage movement data, including creating, updating, and deleting movement records.

**Key Endpoints**

* GET/movement-list: Retrieves movement data information.
* POST/movement-submit: Creates a new movement record.
* PUT/movement/{id}: Updates an existing movement record.
* DELETE/movement/{id}: Deletes a movement record.

### 4.9.4 TA/DA Management

**Overview**

The TA/DA management module is responsible for managing operations related to TA/DA management within the mobile application. It includes functionalities for handling TA/DA-related operations and providing TA/DA data to other modules.

**Key Responsibilities**

1. **TA/DA Management:** Provides functionality to manage TA/DA data, including creating, updating, and deleting TA/DA records.

**Key Endpoints**

* GET/ta-da-list: Retrieves TA/DA data information.
* POST/ta-da-submit: Creates a new TA/DA record.
* PUT/ta-da/{id}: Updates an existing TA/DA record.

### 4.9.5 Allowance Approval

**Overview**

The allowance approval module is responsible for managing operations related to allowance approvals within the mobile application. It includes functionalities for handling allowance approval-related operations and providing allowance approval data to other modules.

**Key Responsibilities**

1. Allowance Approval: Provides functionality to manage allowance approvals.

**Key Endpoints**

* GET/allowance-approvals: Retrieves allowance approval information.
* POST/allowance-approvals: Creates a new allowance approval record.
* PUT/allowance-approvals/{id}: Updates an existing allowance approval record.
* DELETE/allowance-approvals/{id}: Deletes an allowance approval record.

### 4.9.6 Allowance History

**Overview**

The allowance history module is responsible for managing operations related to allowance history within the mobile application. It includes functionalities for handling allowance history-related operations and providing allowance history data to other modules.

**Key Responsibilities**

1. **Allowance History:** Provides functionality to manage allowance history.

### 4.10 Digital Learning Module

**Overview**

The digital learning module is responsible for managing operations related to digital learning within the mobile application. It includes functionalities for managing digital learning items, handling digital learning-related operations, and providing digital learning data to other modules.

**Key Responsibilities**

1. **Digital Learning Item Management:** Provides functionality to manage digital learning items, including creating, updating, and deleting items.
2. **Digital Learning Item List:** Retrieves a list of digital learning items and their corresponding information.
3. **Digital Learning Item Details:** Retrieves detailed information about a specific digital learning item.
4. **Digital Learning Item Playback:** Provides playback functionality for digital learning items, including video, image, and PDF playback.
5. **Digital Learning Survey:** Provides survey functionality for digital learning items, allowing users to complete surveys after completing an item.

**Key Endpoints**

* GET /digital-learning-item/file/{id}: Retrieves a digital learning item file.
* POST /digital-learning-item/file/{id}: Uploads a digital learning item file.

### 4.11 PJP Plan Module

**Overview**

The PJP Plan module is a critical component of the mobile application, specifically designed for Territory Sales Managers (TSMs). This module provides a comprehensive monthly visiting plan for TSMs, ensuring that they have a structured approach to their sales activities.

**Key Responsibilities**

1. **Monthly Visiting Plan**: The PJP Plan module generates a detailed monthly visiting plan for TSMs, outlining the outlets to be visited, the dates, and the products to be focused on.
2. **TSM Attendance**: The module tracks TSM attendance, which is directly dependent on the PJP plan. If a TSM has a PJP plan for the day, their attendance is fully dependent on this plan.
3. **Outlet Selection**: The module allows TSMs to select outlets to visit, based on the PJP plan.
4. **Product Focus**: The module highlights the products to be focused on during each visit, ensuring that TSMs are aware of the key products to promote.

**Key Endpoints**

* **Improved Planning**: The PJP Plan module provides TSMs with a structured approach to their sales activities, ensuring that they have a clear plan for the month.
* **Increased Efficiency**: By having a clear plan, TSMs can optimize their time and resources, reducing the time spent on non-essential activities.
* **Enhanced Productivity**: The module's focus on specific products ensures that TSMs are promoting the right products to the right outlets, increasing sales and revenue.

### 4.12 Maintenance Module

**Overview**

The Maintenance module is a critical component of the mobile application, specifically designed for Technicians. This module enables Technicians to perform maintenance operations on assets that have technical faults, ensuring that issues are resolved efficiently and effectively.

**Key Responsibilities**

1. **Technical Fault Reporting**: The Maintenance module allows Technicians to report technical faults in assets, triggering the maintenance process.
2. **Maintenance Scheduling**: The module schedules maintenance operations, ensuring that Technicians are allocated to resolve issues in a timely manner.
3. **Asset Information**: The module provides Technicians with access to asset information, including asset history, technical specifications, and maintenance records.
4. **Maintenance Task Management**: The module enables Technicians to manage maintenance tasks, including assigning tasks, tracking progress, and updating task status.
5. **Parts and Materials Management**: The module allows Technicians to manage parts and materials required for maintenance operations, including tracking inventory levels and ordering replacement parts.

**Key Screens**

* **Asset Module**: The Maintenance module relies on the Asset module to retrieve asset information and technical specifications.
* **Technician Module**: The Maintenance module relies on the Technician module to retrieve Technician information and allocate maintenance tasks.
* **Inventory Module**: The Maintenance module relies on the Inventory module to manage parts and materials required for maintenance operations.

**Key Dependencies**

* **Maintenance UI**: The Maintenance UI screen allows Technicians to view and manage maintenance operations, including scheduling, task management, and parts and materials management.
* **Technical Fault Reporting Screen**: The Technical Fault Reporting screen allows Technicians to report technical faults in assets, triggering the maintenance process.
* **Asset Information Screen**: The Asset Information screen provides Technicians with access to asset information, including asset history, technical specifications, and maintenance records.
* **Maintenance Task Management Screen**: The Maintenance Task Management screen enables Technicians to manage maintenance tasks, including assigning tasks, tracking progress, and updating task status.