* **Tech Stack:**
* Winston
* Node.js
* Express.js
* Sequelize (MySQL)
* Dotenv
* Express Rate Limit
* Morgan
* sanitizeInput
* bcryptjs
* Redis
* Csv-parser
* Path
* Fs
* Multer
* Aws
* Mongodb
* Jwt
* FCM
* Nodemailer
* Axios
* **Authentication Micro Service:**

**API Name:** Admin Login API (/admin/login): POST

**File:** adminAuth.js

**Function Use**: adminSingIn()

**Description:** The adminSignIn API facilitates administrator authentication. Upon receiving a POST request to /admin/login, it verifies the provided credentials, checks if the user account is banned, retrieves role-based permissions, generates authentication tokens, updates the administrator's login details, logs the login activity, and responds with the generated tokens upon successful authentication. If authentication fails, it returns an error message indicating the reason for failure.

**API Name:** User Login API (/user/login): POST

**File:** userAuth.js

**Function Use**: userSingIn ()

**Description:** The userSignIn API handles user authentication. Upon receiving a POST request to /user/login, it verifies the provided credentials, checks if the user account is banned, generates an authentication token, updates the user's login details, logs the login activity, and responds with the generated token upon successful authentication. If authentication fails, it returns an error message indicating the reason for failure.

* **Admin Micro Service:**

**API Name:** Get all admins without pagination: (/): GET

**File:** admin.js

**Function Use**: getAllAdmins ()

**Description:** This API fetches all admins from the database without applying pagination. It retrieves the list of admins and returns them in the response. If no admins are found, it returns a 404 status code along with an error message indicating that no admins were found. If an error occurs during the process, it returns a 400 status code along with an error message.

**API Name:** Get all admins with pagination: (/all): GET

**File:** admin.js

**Function Use**: getAllAdminsWithPagination ()

**Description:** This API fetches admins from the database with pagination support. It allows filtering, ordering, and searching for admins. The pagination parameters such as offset, perPage, and currentPage are used to retrieve a subset of admins based on the provided criteria. If the isDownload query parameter is set to true, it returns all admins without pagination for downloading purposes. Otherwise, it returns paginated data along with metadata such as total count and total pages.

**API Name:** Get One Admin: (/adminId): GET

**File:** admin.js

**Function Use**: getAdminById ()

**Description:** This API retrieves a single admin based on the provided adminId parameter. It fetches the admin details from the database and returns them in the response. If the admin is found, it returns a success message along with the admin data. If the admin is not found, it returns a 404 status code along with an error message indicating that the admin was not found. If an error occurs during the process, it returns a 400 status code along with an error message.

**API Name:** Admin Creation API (/create): POST

**File:** admin.js

**Function Use**: adminCreate ()

**Description:** Before creating a new admin, the API checks if an admin with the provided email already exists in the system. If the email is not already in use, the API hashes the provided password using bcrypt for enhanced security. Required parameters for admin creation include email, password, first name, last name, mobile, and role. After hashing the password, the API creates a new admin record in the database with the provided details.

**API Name:** Update Admin (/:adminId): PUT

**File:** admin.js

**Function Use**: updateAdmin ()

**Description:** This API is used to update an existing admin's information. It accepts the adminId parameter to identify the admin to be updated and the updateData object containing the fields to be modified. If a new password is provided in the update data, it is hashed using bcrypt and the passwordUpdatedAt field is set to the current time. Similarly, if the email is updated, the emailUpdatedAt field is also updated. After preparing the update data, the API performs the update operation using the updateAdmin function. If the update is successful, it returns a success message with a 200 status code. If an error occurs during the update process, it returns a 400 status code along with an error message.

**API Name:** Admin Status update API (/status/:adminId): POST

**File:** admin.js

**Function Use**: updateAdminStatus ()

**Description:** This API updates the status of an admin account specified by the adminId. It expects the adminId in the URL parameter and the new account status in the request body (under 'accountStatus'). Upon receiving the request, it extracts the adminId and accountStatus, then calls updateAdmin(adminId, accountStatus) to update the admin record.

**API Name:** Delete Admin (/status/: adminId): DELETE

**File:** admin.js

**Function Use**: deleteAdmin ()

**Description:** This API is used to delete an existing admin from the system. It accepts the adminId parameter to identify the admin to be deleted. The API performs a bulk delete operation to remove the admin from the database. Upon successful deletion, it returns a success message with a 200 status code. If an error occurs during the deletion process, it returns a 400 status code along with an error message.

**API Name:** Bulk Delete Admins API (/bulk-delete): POST

**File:** admin.js

**Function Use**: bulkDeleteAdmins ()

**Description:** This API is designed to perform bulk deletion of admin accounts. It expects an array of adminIds in the request body under the key adminIds. Upon receiving the request, the API extracts the adminIds array from the request body.It then performs a bulk delete operation using the deleteAdmin() function, passing an object specifying the condition for deletion.

**API Name:** Get Permissions API (/permission/): GET

**File:** permissions.js

**Function Use:** getPermissions()

**Description:** This API fetches all top-level permissions along with their associated sub-items in a hierarchical structure. It retrieves the permissions and sub-items, merges them, and returns the hierarchical data. If no permissions are found, it returns a success status with a corresponding message. Otherwise, it returns the hierarchical data. If an error occurs, it logs the error and returns an error status with a message.

**API Name:** Get Permissions with Pagination API (/permission/all): GET

**File:** permissions.js

**Function Use:** getAllPermissionswithPagination()

**Description:** This API facilitates the retrieval of permissions with pagination, enabling efficient management of large datasets. It parses query parameters to extract sorting, filtering, and pagination criteria, then constructs a where clause for filtering based on provided search terms. The API fetches permissions with pagination and their associated sub-items, merging them into a hierarchical structure. It handles requests for data download separately, returning all permissions without pagination if requested.

**API Name:** Get One Permission (/ permission/: permissionId): GET

**File:** permissions.js

**Function Use:** getPermissionById()

**Description:** This API retrieves a single permission identified by its permissionId. it extracts the permissionId from the request parameters. It then attempts to fetch the permission with the specified ID using the getAllPermission function. Additionally, it retrieves sub-items associated with the permission. Subsequently, it merges the permission data with its corresponding sub-items, creating a hierarchical structure. If the permission is found, it returns a success status along with the permission data. In case of any error, it logs the error and returns an error status with a corresponding message

**API Name:** Get Parent Permission (/ permission/parent): GET

**File:** permissions.js

**Function Use:** getParentPermissions()

**Description:** The Get Parent Permission API retrieves parent permissions within the system. Upon receiving a GET request to /permission/parent, it fetches permissions with a parent value of 0, indicating top-level permissions. If successful, it returns a success status along with the parent permissions. If an error occurs during the process, it logs the error and returns an error status with a corresponding message.

**API Name:** Create Permission (/ permission/): POST

**File:** permissions.js

**Function Use:** createPermission()

**Description:** The Create Permission API facilitates the creation of new permissions within the system. Upon receiving a POST request to /permission/, it extracts the name and parent values from the request body and attempts to create the permission using the provided data. Upon successful creation, it returns a success status. If an error occurs during the process, such as invalid data or database issues, it logs the error and returns an error status with a corresponding message.

**API Name:** Update Permission (/ permission/:permissionId): PUT

**File:** permissions.js

**Function Use:** updatePermission()

**Description:** The Update Permission API updates an existing permission identified by its permissionId. Upon receiving a PUT request to /permission/:permissionId, it extracts the permissionId from the request parameters and the updated permission data from the request body. It then attempts to update the permission with the provided data. Upon successful update, it returns a success status. If an error occurs during the process, such as invalid data or database issues, it logs the error and returns an error status with a corresponding message.

**API Name:** Delete Permission (/ permission/: permissionId): DELETE

**File:** permissions.js

**Function Use:** deletePermission()

**Description:** The Delete Permission API deletes a permission identified by its permissionId. Upon receiving a DELETE request to /permission/: permissionId, it attempts to delete the permission with the specified ID. Upon successful deletion, it returns a success status. If an error occurs during the process, such as invalid permissions or database issues, it logs the error and returns an error status with a corresponding message.

**API Name:** Bulk-Delete Permission (/ permission/ bulk-delete): POST

**File:** permissions.js

**Function Use:** bulkDeletePermissions()

**Description:** The Bulk-Delete Permission API allows for the bulk deletion of permissions within the system. It expects an array of permissionIds in the request body and deletes permissions matching those IDs. Upon successful deletion, it returns a success status. If an error occurs during the process, such as invalid permissions or database issues, it returns an error status with a corresponding message.

**API Name:** Get Role with Pagination API (/role/all): GET

**File:** role.js

**Function Use:** getAllRolewithpagination()

**Description:** The Get Role with Pagination API retrieves roles with pagination, allowing for efficient handling of large datasets. It parses query parameters to extract sorting, filtering, and pagination criteria. Then, it constructs a where clause for filtering based on provided search terms. The API fetches roles with pagination and applies the specified filter. It returns paginated data with information about the total count, current page, total pages, and per-page data. If requested, it also supports downloading all roles without pagination.

**API Name:** Get Role without Pagination API (/role/): GET

**File:** role.js

**Function Use:** getAllRoles()

**Description:** The Get Role without Pagination API retrieves all roles without pagination. Upon receiving a GET request to /role/, it fetches all roles from the database. It returns a success status along with the role data. If an error occurs during the process, it logs the error and returns an error status with a corresponding message.

**API Name:** Get One Role API (/role/:roleId): GET

**File:** role.js

**Function Use:** getRoleById ()

**Description:** The Get One Role API retrieves a single role identified by its roleId. Upon receiving a GET request to /role/: roleId, it extracts the roleId from the request parameters and attempts to fetch the role with the specified ID. It returns a success status along with the role data if found. If an error occurs during the process, it logs the error and returns an error status with a corresponding message.

**API Name:** Create Role API(/role/): POST

**File:** role.js

**Function Use:** createRole()

**Description:** The Create Role API facilitates the creation of a new role within the system. Upon receiving a POST request to /role/, it extracts the name and permission values from the request body and attempts to create the role using the provided data. It returns a success status upon successful creation. If an error occurs during the process, such as invalid data or database issues, it logs the error and returns an error status with a corresponding message.

**API Name:** Update Role API (/role/:roleId): PUT

**File:** role.js

**Function Use:** updateRole ()

**Description:** The Update Role API updates an existing role identified by its roleId. Upon receiving a PUT request to /role/:roleId, it extracts the roleId from the request parameters and the updated role data from the request body. It then attempts to update the role with the provided data. It returns a success status upon successful update. If an error occurs during the process, it logs the error and returns an error status with a corresponding message.

**API Name:** Delete Role API (/role/:roleId): DELETE

**File:** role.js

**Function Use:** deleteRole()

**Description:** The Delete Role API deletes a role identified by its roleId. Upon receiving a DELETE request to /role/:roleId, it attempts to delete the role with the specified ID. It returns a success status upon successful deletion. If an error occurs during the process, it logs the error and returns an error status with a corresponding message.

**API Name:** Bulk Delete Roles API (/role/bulk-delete): POST

**File:** role.js

**Function Use:** bulkDeleteRoles()

**Description:** The Bulk Delete Roles API allows for the bulk deletion of roles within the system. It expects an array of roleIds in the request body and deletes roles matching those IDs. It returns a success status upon successful deletion. If an error occurs during the process, it returns an error status with a corresponding message.

* **User Micro Service:**

**API Name:** User Register API (/register): POST

**File:** users.js

**Function Use**: userSingUp()

**Description:** This API allows users to register in the system. Upon receiving a POST request with user registration details, including first name, last name, email, password, mobile, device information, and IP address, it checks if the user with the provided email already exists. If not, it hashes the password for security, creates the user record in the database along with device information, generates an OTP for email verification, and returns a success status along with an authentication token.

**API Name:** Get All Users With Pagination API (/pagination): GET

**File:** users.js

**Function Use**: getAllUserwithpagination ()

**Description:** This API retrieves a list of all users with pagination support. It accepts query parameters for sorting, searching, and filtering users. It constructs a WHERE clause based on the provided search and filter criteria, fetches users from the database with pagination, and returns the paginated user data along with metadata such as total count, current page, and total pages.

**API Name:** Get One User API: (/:userId): GET

**File:** admin.js

**Function Use**: getUserById()

**Description:** This API retrieves a specific user by their user ID. It expects the user ID as a URL parameter. Upon receiving a GET request, it fetches the user data from the database based on the provided user ID and returns the user details.

**API Name:** Get All Users Without Pagination API (/): GET

**File:** users.js

**Function Use**: getAllUsers ()

**Description:** This API retrieves a list of all users without pagination. Upon receiving a GET request, it fetches all user records from the database and returns the user data.

**API Name:** Update User API (/:userId): PUT

**File:** users.js

**Function Use:** updateUser ()

**Description:** This API updates an existing user's information. It expects the user ID as a URL parameter and updated user data in the request body. If the update includes a new password, it hashes the password using bcrypt, logs the password update activity, and updates the user record in the database. It returns a success status upon successful update.

**API Name:** Delete User API (/:userId): DELETE

**File:** users.js

**Function Use:** deleteRole()

**Description:** This API deletes an existing user from the system. It expects the user ID as a URL parameter. Upon receiving a DELETE request, it deletes the user record from the database and returns a success status upon successful deletion.

**API Name:** Download Xls file API (/) : GET

**File:** users.js

**Function Use:** downloadUsersAsXLS ()

**Description:** This API allows downloading user data in Excel format. Upon receiving a GET request, it retrieves user data from the database, generates an Excel file with user details, and sends the file as a downloadable attachment in the response.

**API Name:** Update User Status API (/status/:userId): PUT

**File:** users.js

**Function Use:** updateStatus ()

**Description:** This API updates the status of a user account. It expects the user ID as a URL parameter and the new user status and banned reason in the request body. It validates the input data, updates the user status and banned reason in the database, and returns a success status upon successful update.

**API Name:** Bulk-Delete Users API (/bulk-delete): POST

**File:** users.js

**Function Use:** bulkDeleteUsers ()

**Description:** This API allows bulk deletion of user accounts. It expects an array of user IDs in the request body. Upon receiving a POST request, it performs a bulk delete operation using Sequelize's Op.in operator, deletes the specified user records from the database, and returns a success status upon successful deletion.

* **Product Micro Service:**

**API Name:** Get All Categories without Pagination (/categories/): GET

**File:** categories.js

**Function Use**: getCategory ()

**Description:** The getCategory API retrieves all categories without pagination. It first attempts to fetch data from the cache using the "getAllparentCategories". Then, it fetches sub-items. After merging the data, it constructs a hierarchical structure where each category may have children categories. If no data is found, it responds with an error message indicating that categories are not found.

**API Name:** Get All Categories with Pagination (/categories/all): GET

**File:** categories.js

**Function Use**: getAllCategorieswithPagination ()

**Description:** The getAllCategorieswithPagination API retrieves all categories with pagination. It allows filtering, sorting, and searching categories based on different criteria. It fetches categories with their associated products and provides pagination information. If no data is found, it responds with an appropriate error message.

**API Name:** Get Parent Categories (/categories/parent): GET

**File:** categories.js

**Function Use**: getParentCategories ()

**Description:** The getParentCategories API retrieves only the parent categories without their children. It fetches data directly from the database without pagination. If no parent categories are found, it responds with an appropriate error message.

**API Name:** Get Category by ID (/categories/:categoryId): GET

**File:** categories.js

**Function Use**: getCategoryById ()

**Description:** The getCategoryById API retrieves a single category by its unique ID. It fetches the category data along with its children categories, if any. If the requested category does not exist, it responds with an appropriate error message.

**API Name:** Create Category **(**/categories/): POST

**File:** categories.js

**Function Use:** createCategory ()

**Description:** The createCategory API allows the creation of a new category. It expects category data in the request body and validates it before insertion. If the category is created successfully, it responds with a success message along with the newly created category data. If there is any error during the creation process, it responds with an appropriate error message.

**API Name:** Update Category (/categories/:categoryId): PUT

**File:** categories.js

**Function Use:** updateCategory()

**Description:** The updateCategory API allows updating an existing category. It expects the category ID in the request parameters and updated category data in the request body. It validates the data and performs the update operation. If the update is successful, it responds with a success message along with the updated category data. If the category does not exist or there is an error during the update process, it responds with an appropriate error message.

**API Name:** Delete Category /categories/:categoryId): DELETE

**File:** categories.js

**Function Use:** deleteCategory ()

**Description:** The deleteCategory API allows deleting a category by its ID. It expects the category ID in the request parameters. If the category is successfully deleted, it responds with a success message. If the category does not exist or there is an error during the deletion process, it responds with an appropriate error message.

**API Name:** Update Category Status (/categories/status/:categoryId): PUT

**File:** categories.js

**Function Use:** updateCategoryStatus()

**Description:** The updateCategoryStatus API allows updating the status of a category by its ID. It expects the category ID and the new status in the request parameters and body, respectively. If the status is updated successfully, it responds with a success message along with the updated category data. If the category does not exist or there is an error during the update process, it responds with an appropriate error message.

**API Name:** Bulk Delete Categories (/categories/bulk-delete): POST

**File:** categories.js

**Function Use:** bulkDeleteCategories ()

**Description:** The bulkDeleteCategories API allows deleting multiple categories at once. It expects an array of category IDs in the request body. It performs the bulk delete operation and responds with a success message if all categories are deleted successfully. If there is any error during the deletion process, it responds with an appropriate error message.

**API Name:** Get All Products with Pagination (/all): GET

**File:** products.js

**Function Use**: getAllProductsWithPagination()

**Description:** The getAllProductsWithPagination API retrieves all products with pagination. It allows filtering, sorting, and searching products based on different criteria. It fetches products and provides pagination information. If no data is found, it responds with an appropriate error message.

**API Name:** Get Product Keys (/keys): GET

**File:** products.js

**Function Use**: getProductKeys()

**Description:** The getProductKeys API retrieves all product keys. It fetches and returns a list of product keys used for identifying products in the system.

**API Name:** Get All Products with Categories (/parent): GET

**File:** products.js

**Function Use**: getAllProductsWithCategories ()

**Description:** The getAllProductsWithCategories API retrieves all products along with their associated categories. It fetches and returns a list of products with their respective categories.

**API Name:** Search Products (/search): GET

**File:** products.js

**Function Use**: searchProducts ()

**Description:** The searchProducts API allows searching for products based on the provided search query. It retrieves products matching the search query and responds with the search results.

**API Name:** Get All Products without Pagination (/): GET

**File:** products.js

**Function Use**: getAllProducts ()

**Description:** The getAllProducts API retrieves all products without pagination. It fetches and returns a list of all products available in the system.

**API Name:** Get Product by ID (/: productId): GET

**File:** products.js

**Function Use:** getProductById ()

**Description:** The getProductById API retrieves a single product by its unique ID. It fetches and returns the product data associated with the specified product ID.

**API Name:** Create Product (/): POST

**File:** products.js

**Function Use:** createProduct ()

**Description:** The createProduct API allows creating a new product in the system. It expects product data in the request body and validates it before insertion. If the product is created successfully, it responds with a success message along with the newly created product data.

**API Name:** Update Product (/:productId): PUT

**File:** products.js

**Function Use:** updateProduct ()

**Description:** The updateProduct API allows updating an existing product. It expects the product ID in the request parameters and updated product data in the request body. It validates the data and performs the update operation. If the update is successful, it responds with a success message along with the updated product data.

**API Name:** Delete Product (/:productId): DELETE

**File:** products.js

**Function Use:** deleteProduct ()

**Description:** The deleteProduct API allows deleting a product by its ID. It expects the product ID in the request parameters. If the product is successfully deleted, it responds with a success message. If the product does not exist or there is an error during the deletion process, it responds with an appropriate error message.

**API Name:** Bulk Import Products (/bulk-import): POST

**File:** products.js

**Function Use:** bulkImportProducts ()

**Description:** The bulkImportProducts API allows importing multiple products at once using a CSV file. It expects the CSV file containing product data in the request body. It converts the CSV data to JSON format and imports the products into the system. If the import is successful, it responds with a success message along with the imported product data.

**API Name:** Update Product Status (/status/:productId): PUT

**File:** products.js

**Function Use:** updateProductStatus ()

**Description:** The updateProductStatus API allows updating the status of a product by its ID. It expects the product ID and the new status in the request parameters and body, respectively. If the status is updated successfully, it responds with a success message along with the updated product data.

**API Name:** Bulk Delete Products (/bulk-delete): POST

**File:** products.js

**Function Use:** bulkDeleteProducts ()

**Description:** The bulkDeleteProducts API allows deleting multiple products at once. It expects an array of product IDs in the request body. It performs the bulk delete operation and responds with a success message if all products are deleted successfully. If there is any error during the deletion process, it responds with an appropriate error message.

* **Upload Micro Service:**

**API Name:** Upload Image (/upload/image): POST

**File:** upload.js

**Function Use:** uploadImage ()

**Description:** Accepts a single image file in the request body and uploads it. Returns the final path of the uploaded image.

**API Name:** Upload Multiple Images (/upload/images): POST

**File:** upload.js

**Function Use:** uploadMultipleImages ()

**Description:** Accepts multiple image files in the request body and uploads them. Returns an array containing the final paths of all uploaded images.

**API Name:** Upload Document (/upload/document): POST

**File:** upload.js

**Function Use:** uploadDocument()

**Description:** Accepts a single document file in the request body and uploads it. Returns the final path of the uploaded document.

**API Name:** Upload Multiple Documents (/upload/ documents): POST

**File:** upload.js

**Function Use:** uploadMultipleDocuments ()

**Description:** Accepts multiple document files in the request body and uploads them. Returns an array containing the final paths of all uploaded documents.

**API Name:** Move Images and Documents (/upload/ move): POST

**File:** upload.js

**Function Use:** moveImagesAndDoc ()

**Description:** Moves files specified in the request body to a specified folder.

**API Name:** Upload CSV to JSON (/upload/ csv-to-json): POST

**File:** upload.js

**Function Use:** uploadCSVToJson ()

**Description:** Reads a CSV file specified in the request body, converts it to JSON format, and returns the product data extracted from the CSV.

* **Report Micro Service:**

**API Name:** Dashboard API (/dashboard): GET

**File:** dashboard.js

**Function Use:** updateDashboard ()

**Description:** This API endpoint updates the dashboard with various counts and metrics related to user registrations, active users, product counts, order counts, etc.

**API Name:** get Last 15 Days Order status API (orders/last15daysorder): GET

**File:** orderReport.js

**Function Use:** getLast15DaysOrderStatus ()

**Description:** The "getLast15DaysOrderStatus" API retrieves order statistics for the last 15 days. It calculates the date 15 days ago from today and then iterates through each day to count the orders created on that day. The API constructs a response object containing the order counts for each day in the last 15 days and returns it as JSON.

**API Name:** Get Orders By Filters API(orders / filter): GET

**File:** orderReport.js

**Function Use:** getOrdersByFilters ()

**Description:** The getOrdersByFilters API retrieves orders based on specified filters. It accepts query parameters for ordering, filtering, pagination, and downloading options. Upon receiving a request, it parses the query parameters, constructs a where clause based on the filters provided, and fetches orders accordingly. If the isDownload query parameter is set to 'true', it returns all orders without pagination for downloading purposes. Otherwise, it returns paginated order data. The response includes formatted order details, such as order date, creation date, order ID, customer ID, order amount, order status, coupon code, and associated products. If an error occurs during the process, it returns an error message with details.

**API Name:** get Products By Creation Date API (products/date): GET

**File:** productReport.js

**Function Use:** getProductsByCreationDate ()

**Description:** The "getProductsByCreationDate" API retrieves products created within a specified date range. It accepts query parameters for ordering, filtering, pagination, and downloading data. Upon execution, it queries the database for products based on the provided date range and returns either paginated data or the complete set of products, depending on the "isDownload" parameter. If no products are found within the specified dates, it returns a success response with an empty data array.

**API Name:** get Products By Status API (products/status): GET

**File:** productReport.js

**Function Use:** getProductsByStatus()

**Description:** The "getProductsByStatus" API retrieves products based on their active or inactive status. Similar to the previous API, it supports ordering, filtering, pagination, and downloading options. It constructs a where clause based on the provided status and other filter criteria, fetches products from the database, and returns the results accordingly. If no products match the specified status, it responds with a success message indicating the absence of matching products.

**API Name:** get Products By Quantity Range API (products/range): GET

**File:** productReport.js

**Function Use:** getProductsByQuantityRange ()

**Description:** The "getProductsByQuantityRange" API fetches products within a specified quantity range. It parses the provided minimum and maximum quantity parameters, constructs a where clause accordingly, and queries the database for products meeting the specified criteria. Like the previous APIs, it handles ordering, filtering, pagination, and downloading options and returns the results based on the provided parameters. If no products are found within the specified quantity range, it returns a success response with an empty data array.

**API Name:** get Products Less Than 10 Quantity API (products/lessthan10): GET

**File:** productReport.js

**Function Use:** getProductsLessThan10Qty ()

**Description:** The "getProductsLessThan10Qty" API retrieves products with a quantity less than 10 in stock. It fetches products using Sequelize's model method, specifying a condition where the quantity is less than 10. Upon execution, it returns the fetched products along with a success message indicating the successful retrieval of products with less than 10 quantity in stock.

**API Name:** get Out-of-Stock Products API (products/outofstock): GET

**File:** productReport.js

**Function Use:** getOutOfStockProducts ()

**Description:** The "getOutOfStockProducts" API retrieves out-of-stock products by querying the database for products with an inactive status and a quantity of 0. If any out-of-stock products are found, it returns them along with a success message indicating the successful retrieval of out-of-stock products. If no out-of-stock products are found, it responds with a success message indicating the absence of such products.

**API Name:** get Users By Creation Date API (users/date): GET

**File:** usersReport.js

**Function Use:** getUsersByRegisterDate ()

**Description:** Retrieves all users based on their registration date within a specified date range.

**API Name:** Get All Users By Status API (users/status): GET

**File:** usersReport.js

**Function Use:** getUsersByStatus ()

**Description:** Retrieves users based on their status within a specified date range. It supports filtering, sorting, and pagination.

**API Name:** Get All Register Vs Active Users API (users/registervsactive): GET

**File:** usersReport.js

**Function Use:** getRegisterVsActiveUsers ()

**Description:** Compares the number of registered users to active users within a specified date range and returns the statistics. This API calculates the count of registered users and active users for each day within the specified range and returns the data in a structured format.

* **Others Micro Service:**

**API Name:** Get All Activity Log with Pagination (activitylogs/all): GET

**File:** activityLog.js

**Function Use:** getAllActivityLogswithPagination()

**Description:** The Get All Activity Log with Pagination API retrieves activity logs with pagination support. It allows clients to fetch activity logs within a specified range and apply filtering if needed. Additionally, clients can specify sorting parameters like orderBy and order to customize the order of returned logs. Filtering based on search criteria is supported, enabling clients to retrieve logs matching specific keywords across various fields like tableName, activityType, and details. If the client requests to download the data, the API provides all activity logs without pagination. Otherwise, it returns paginated data along with metadata. In case of any errors during the retrieval process, appropriate error messages are returned to the client.

**API Name:** Get All Activity Logs without Pagination (activitylogs/): GET

**File:** activityLog.js

**Function Use:** getAllActivityLogs ()

**Description:** Retrieves all activity logs without pagination. It returns all activity logs present in the system without any limitations. This endpoint is useful when clients need to retrieve the entire set of activity logs at once, without the need for pagination. If the operation is successful, it returns a JSON response with the retrieved data. If an error occurs during the retrieval process, an appropriate error message is returned along with a status code.

**API Name:** Get One Activity Log (activitylogs/: logId): GET

**File:** activityLog.js

**Function Use:** getActivityLogById ()

**Description:** Retrieves a single activity log by its ID. Clients can specify the ID of the activity log they want to retrieve in the request parameters. If the log exists, it returns the log data in the response. If the specified log ID does not exist, or if there is an error during the retrieval process, appropriate error messages are returned along with a status code.

**API Name:** Create Activity Log (activitylogs/): POST

**File:** activityLog.js

**Function Use:** createActivityLog()

**Description:** Creates a new activity log in the system. Clients need to provide the necessary details of the activity log in the request body, including userID, tableName, activityType, timestamp, and details. Upon successful creation, it returns a success message along with a status code. If there are any validation errors or issues during the creation process, appropriate error messages are returned along with a status code.

**API Name:** Delete Activity Log (activitylogs/: logId): DELETE

**File:** activityLog.js

**Function Use:** deleteActivityLog ()

**Description:** Deletes an activity log from the system based on its ID. Clients need to specify the ID of the activity log they want to delete in the request parameters. If the deletion is successful, it returns a success message along with a status code. If the specified log ID does not exist or if there is an error during the deletion process, appropriate error messages are returned along with a status code.

**API Name:** Backup Creation (/backup/getbackup): POST

**File:** backup.js

**Function Use:** getBackup ()

**Description:** The Backup Creation API allows users to trigger the creation of backups. When invoked, the API calls the createBackup() function from the backup service to initiate the backup creation process. Upon successful backup creation, it returns a JSON response with a success message indicating that the backups were created successfully, along with the URLs to access the backups. If an error occurs during the backup creation process, the API returns an appropriate error message with a status code.

**API Name:** Get all posts with pagination (/posts/all): GET

**File:** posts.js

**Function Use:** getPostsWithPagination()

**Description:** This API retrieves posts with pagination support, allowing clients to fetch posts within a specified range and apply filtering based on search criteria. Clients can specify sorting parameters such as orderBy and order to customize the order of returned posts. The API constructs a where clause for filtering based on the provided search query, matching keywords across fields like title, content, and postStatus. If the client requests to download the data, the API returns all posts without pagination; otherwise, it returns paginated data along with metadata. In case of any errors during the retrieval process, appropriate error messages are returned.

**API Name:** Get All Posts without Pagination (/posts/): GET

**File:** posts.js

**Function Use**: getAllPosts()

**Description:** This API retrieves all posts without pagination, allowing clients to access the entire collection of posts at once. Clients can utilize this endpoint to retrieve all posts stored in the system. Upon a successful retrieval, the API responds with a JSON object containing the retrieved posts along with a success message indicating that all posts were retrieved successfully. In case of any errors during the retrieval process, appropriate error messages are returned with the corresponding status code.

**API Name:** Get One Post (/posts/:postId): GET

**File:** posts.js

**Function Use**: getPostById()

**Description:** The Get One Post API retrieves a single post identified by its unique postId. Clients can specify the postId in the request parameters to retrieve the specific post. Upon successful retrieval, the API responds with a JSON object containing the retrieved post along with a success message indicating that the post was found successfully. If the specified postId does not exist or if an error occurs during the retrieval process, appropriate error messages are returned with the corresponding status code.

**API Name:** Create Post (/posts/): POST

**File:** posts.js

**Function Use:** createPost()

**Description:** The Create Post API allows clients to create a new post by providing necessary details such as title, content, post category, post type, etc., in the request body. Upon successful creation of the post, the API responds with a JSON object containing a success message indicating that the post was created successfully. If any validation errors occur during the creation process, appropriate error messages are returned with the corresponding status code.

**API Name:** Update Post (/posts/:postId): PUT

**File:** posts.js

**Function Use:** updatePost()

**Description:** The Update Post API allows clients to update an existing post identified by its unique postId. Clients can specify the postId in the request parameters and provide updated post details in the request body. Upon successful update of the post, the API responds with a JSON object containing a success message indicating that the post was updated successfully. If the specified postId does not exist or if any validation errors occur during the update process, appropriate error messages are returned with the corresponding status code.

**API Name:** Delete Post (/posts/:postId): DELETE

**File:** posts.js

**Function Use:** deletePost ()

**Description:** The Delete Post API allows clients to delete a post identified by its unique postId. Clients can specify the postId in the request parameters to delete the specific post. Upon successful deletion of the post, the API responds with a JSON object containing a success message indicating that the post was deleted successfully. If the specified postId does not exist or if an error occurs during the deletion process, appropriate error messages are returned with the corresponding status code.

**API Name:** Bulk Delete Post (/posts/bulk-delete): POST

**File:** posts.js

**Function Use:** bulkDeletePosts ()

**Description:** The Bulk Delete Post API allows clients to delete multiple posts at once by providing an array of postIds in the request body. Upon successful deletion of the specified posts, the API responds with a JSON object containing a success message indicating that the posts were deleted successfully. If any validation errors occur during the deletion process or if the specified postIds do not exist, appropriate error messages are returned with the corresponding status code.

**API Name:** Get All Posts Category without Pagination (/post/category/): GET

**File:** postsCategory.js

**Function Use:** getAllPostsCategory ()

**Description:** This API retrieves all posts categories without pagination, allowing clients to access the entire collection of posts categories at once. Clients can utilize this endpoint to retrieve all posts categories stored in the system. Upon a successful retrieval, the API responds with a JSON object containing the retrieved posts categories along with a success message indicating that all posts categories were retrieved successfully. In case of any errors during the retrieval process, appropriate error messages are returned with the corresponding status code.

**API Name:** Get All Posts Category with Pagination (/post/category/all): GET

**File:** postsCategory.js

**Function Use:** getPostCategoryWithPagination ()

**Description:** This API retrieves posts categories with pagination support, allowing clients to fetch posts categories within a specified range and apply filtering based on search criteria. Clients can specify sorting parameters such as orderBy and order to customize the order of returned posts categories. The API constructs a where clause for filtering based on the provided search query, matching keywords across fields like name and categoryStatus. If the client requests to download the data, the API returns all posts categories without pagination; otherwise, it returns paginated data along with metadata. In case of any errors during the retrieval process, appropriate error messages are returned.

**API Name:** Get One Post Category (/post/category/:postCategoryId): GET

**File:** postsCategory.js

**Function Use**: getPostCategoryById ()

**Description:** The Get One Post Category API retrieves a single post category identified by its unique postCategoryId. Clients can specify the postCategoryId in the request parameters to retrieve the specific post category. Upon successful retrieval, the API responds with a JSON object containing the retrieved post category along with a success message indicating that the post category was found successfully. If the specified postCategoryId does not exist or if an error occurs during the retrieval process, appropriate error messages are returned with the corresponding status code.

**API Name:** Create Post Category (/posts/ category/): POST

**File:** postsCategory.js

**Function Use:** createPostCategory ()

**Description:** The Create Post Category API allows clients to create a new post category by providing necessary details such as name, parentId, categoryImage, categoryStatus, etc., in the request body. Upon successful creation of the post category, the API responds with a JSON object containing a success message indicating that the post category was created successfully. If any validation errors occur during the creation process, appropriate error messages are returned with the corresponding status code.

**API Name:** Update Post Category (/posts/ category/:postCategoryId): PUT

**File:** postsCategory.js

**Function Use:** updatePostCategory ()

**Description:** The Update Post Category API allows clients to update an existing post category identified by its unique postCategoryId. Clients can specify the postCategoryId in the request parameters and provide updated post category details in the request body. Upon successful update of the post category, the API responds with a JSON object containing a success message indicating that the post category was updated successfully. If the specified postCategoryId does not exist or if any validation errors occur during the update process, appropriate error messages are returned with the corresponding status code.

**API Name:** Update Post Category Status (/post/category/status/:postCategoryId): PUT

**File:** postsCategory.js

**Function Use:** updatePostCategoryStatus ()

**Description:** The Update Post Category Status API allows clients to update the status of a post category identified by its unique postCategoryId. Clients can specify the postCategoryId in the request parameters and provide the updated categoryStatus in the request body. Upon successful update of the post category status, the API responds with a JSON object containing a success message indicating that the post category status was updated successfully. If the specified postCategoryId does not exist or if any validation errors occur during the update process, appropriate error messages are returned with the corresponding status code.

**API Name:** Delete Post Category (/posts/ category/:postCategoryId): DELETE

**File:** postsCategory.js

**Function Use:** deletePostCategory ()

**Description:** The Delete Post Category API allows clients to delete a post category identified by its unique postCategoryId. Clients can specify the postCategoryId in the request parameters to delete the specific post category. Upon successful deletion of the post category, the API responds with a JSON object containing a success message indicating that the post category was deleted successfully. If the specified postCategoryId does not exist or if an error occurs during the deletion process, appropriate error messages are returned with the corresponding status code.

**API Name:** Bulk Delete Post Category (/posts/ category/: bulk-delete): POST

**File:** postsCategory.js

**Function Use:** bulkDeletePostsCategory ()

**Description:** The Bulk Delete Post Category API allows clients to delete multiple post categories at once by providing an array of postCategoryIds in the request body. Upon successful deletion of the specified post categories, the API responds with a JSON object containing a success message indicating that the post categories were deleted successfully. If any validation errors occur during the deletion process or if the specified postCategoryIds do not exist, appropriate error messages are returned with the corresponding status code.

**API Name:** Get All Posts Types without Pagination (/post/type/): GET

**File:** postsTypes.js

**Function Use:** getAllPostsTypes ()

**Description:** This API retrieves all posts types without pagination, allowing clients to access the entire collection of posts types at once. Clients can utilize this endpoint to retrieve all posts types stored in the system. Upon a successful retrieval, the API responds with a JSON object containing the retrieved posts types along with a success message indicating that all posts types were retrieved successfully. In case of any errors during the retrieval process, appropriate error messages are returned with the corresponding status code.

**API Name:** Get All Posts Types with Pagination (/post/type/all): GET

**File:** postsTypes.js

**Function Use**: getPostTypesWithPagination ()

**Description:** This API retrieves posts types with pagination support, allowing clients to fetch posts types within a specified range and apply filtering based on search criteria. Clients can specify sorting parameters such as orderBy and order to customize the order of returned posts types. The API constructs a where clause for filtering based on the provided search query, matching keywords across fields like name and slug. If the client requests to download the data, the API returns all posts types without pagination; otherwise, it returns paginated data along with metadata. In case of any errors during the retrieval process, appropriate error messages are returned.

**API Name:** Get One Posts Types (/post/type/:postTypeId)GET

**File:** postsTypes.js

**Function Use**: getPostTypeById ()

**Description:** The Get One Posts Types API retrieves a single post type identified by its unique postTypeId. Clients can specify the postTypeId in the request parameters to retrieve the specific post type. Upon successful retrieval, the API responds with a JSON object containing the retrieved post type along with a success message indicating that the post type was found successfully. If the specified postTypeId does not exist or if an error occurs during the retrieval process, appropriate error messages are returned with the corresponding status code.

**API Name:** Create Posts Types (/posts/ type/): POST

**File:** postsTypes.js

**Function Use:** createPostType ()

**Description:** The Create Posts Types API allows clients to create a new post type by providing necessary details such as name in the request body. Upon successful creation of the post type, the API responds with a JSON object containing a success message indicating that the post type was created successfully. If any validation errors occur during the creation process, appropriate error messages are returned with the corresponding status code.

**API Name:** Update Posts Types (/posts/ type/:postTypeId): PUT

**File:** postsTypes.js

**Function Use:** updatePostType ()

**Description:** The Update Posts Types API allows clients to update an existing post type identified by its unique postTypeId. Clients can specify the postTypeId in the request parameters and provide updated post type details in the request body. Upon successful update of the post type, the API responds with a JSON object containing a success message indicating that the post type was updated successfully. If the specified postTypeId does not exist or if any validation errors occur during the update process, appropriate error messages are returned with the corresponding status code.

**API Name:** Delete Posts Types (/posts/ type/:postTypeId): DELETE

**File:** postsTypes.js

**Function Use:** deletePostType ()

**Description:** The Delete Posts Types API allows clients to delete a post type identified by its unique postTypeId. Clients can specify the postTypeId in the request parameters to delete the specific post type. Upon successful deletion of the post type, the API responds with a JSON object containing a success message indicating that the post type was deleted successfully. If the specified postTypeId does not exist or if an error occurs during the deletion process, appropriate error messages are returned with the corresponding status code.

**API Name:** Bulk Delete Posts Types (/posts/ type/bulk-delete): POST

**File:** postsTypes.js

**Function Use:** bulkDeletePostsTypes ()

**Description:** The Bulk Delete Posts Types API allows clients to delete multiple post types at once by providing an array of postTypeIds in the request body. Upon successful deletion of the specified post types, the API responds with a JSON object containing a success message indicating that the post types were deleted successfully. If any validation errors occur during the deletion process or if the specified postTypeIds do not exist, appropriate error messages are returned with the corresponding status code.

**API Name:** Get All Settings with Pagination (/setting/all): GET

**File:** setting.js

**Function Use:** getAllSettingwithPagination ()

**Description:** This API retrieves all settings with pagination support, allowing clients to fetch settings within a specified range and apply filtering based on search criteria. Clients can specify sorting parameters such as orderBy and order to customize the order of returned settings. The API constructs a where clause for filtering based on the provided search query, matching keywords across fields like key and value. If the client requests to download the data, the API returns all settings without pagination; otherwise, it returns paginated data along with metadata. In case of any errors during the retrieval process, appropriate error messages are returned.

**API Name:** Get All Settings without Pagination (/setting/): GET

**File:** setting.js

**Function Use**: getAllSettings ()

**Description:** The Get All Settings without Pagination API retrieves all settings without pagination, specifically targeting inactive settings. Clients can utilize this endpoint to retrieve all inactive settings stored in the system. Upon successful retrieval, the API responds with a JSON object containing the retrieved inactive settings along with a success message indicating that the inactive settings were retrieved successfully. In case of any errors during the retrieval process, appropriate error messages are returned.

**API Name:** Get One Setting (/setting/:settingId):GET

**File:** setting.js

**Function Use**: getSettingById ()

**Description:** The Get One Setting API retrieves a single setting identified by its unique settingId. Clients can specify the settingId in the request parameters to retrieve the specific setting. Upon successful retrieval, the API responds with a JSON object containing the retrieved setting along with a success message indicating that the setting was found successfully. If the specified settingId does not exist or if an error occurs during the retrieval process, appropriate error messages are returned.

**API Name:** Create Setting (/setting/): POST

**File:** setting.js

**Function Use:** createSetting ()

**Description:** The Create Setting API allows clients to create a new setting by providing necessary details such as key, value, label, description, isEditable, and isActive in the request body. Upon successful creation of the setting, the API responds with a JSON object containing a success message indicating that the setting was created successfully. If any validation errors occur during the creation process, appropriate error messages are returned.

**API Name:** Update Setting (/setting/:settingId): PUT

**File:** setting.js

**Function Use:** updateSetting ()

**Description:** The Update Setting API allows clients to update an existing setting identified by its unique settingId. Clients can specify the settingId in the request parameters and provide updated setting details in the request body. Upon successful update of the setting, the API responds with a JSON object containing a success message indicating that the setting was updated successfully. If the specified settingId does not exist or if any validation errors occur during the update process, appropriate error messages are returned.

**API Name:** Delete Setting (/setting/:settingId): DELETE

**File:** setting.js

**Function Use:** deleteSetting ()

**Description:** The Delete Setting API allows clients to delete a setting identified by its unique settingId. Clients can specify the settingId in the request parameters to delete the specific setting. Upon successful deletion of the setting, the API responds with a JSON object containing a success message indicating that the setting was deleted successfully. If the specified settingId does not exist or if an error occurs during the deletion process, appropriate error messages are returned.

**API Name:** Bulk Update Setting (/setting/bulk-update): POST

**File:** setting.js

**Function Use:** bulkUpdateSetting()

**Description:** The Bulk Update Setting API allows clients to update multiple settings at once by providing an array of settingsToUpdate in the request body. Each setting must contain a settingId for updating. Upon successful update of the specified settings, the API responds with a JSON object containing a success message indicating that the settings were updated successfully. If any validation errors occur during the update process or if any settingId is missing, appropriate error messages are returned.

* **Communication Micro Service:**

**API Name:** Send Email (/email/send-email): POST

**File:** email.js

**Function Use:** sendEmail ()

**Description:** The Send Email API allows clients to send an email using a specified email template and user data. Clients need to provide the templateName and userData in the request body. Upon receiving the request, the API calls the sendEmail function, passing the provided templateName and userData. If the email is sent successfully, the API responds with a JSON object containing a success message indicating that the email was sent successfully. If any errors occur during the email sending process, appropriate error messages are returned.

**API Name:** Get All Notifications with pagination (/notification/all): GET

**File:** notification.js

**Function Use:** getAllNotificationwithpagination ()

**Description:** The getAllNotificationwithpagination API allows clients to retrieve all notifications with pagination. Clients can specify search parameters such as orderBy, order, and search string. The API returns paginated data containing the count, rows, perPage, currentPage, and totalPages. If the isDownload parameter is set to 'true', the API returns all notifications without pagination.

**API Name:** Get Notifications without pagination (/notification/): GET

**File:** notification.js

**Function Use:** getNotifications ()

**Description:** The getNotifications API allows clients to retrieve all notifications without pagination. The API returns all notifications in the response.

**API Name:** Create Notification (/notification/create): POST

**File:** notification.js

**Function Use:** createNotification ()

**Description:** The createNotification API allows clients to create a new notification. Clients need to provide the title, message, and optionally an image in the request body. Upon receiving the request, the API creates the notification and sends it to all active users who have registered their Firebase tokens. The API also checks the environment variable NOTIFICATION\_TYPE to determine the type of notification to be sent. If the variable is set to 'notification', the notification is sent via Firebase. If it is set to 'oneSignal', the notification is sent via OneSignal. If no environment variable is set, the notification is sent via both Firebase and OneSignal. After sending the notification, the API responds with a success message.

**API Name:** Delete Notification (/notification/:nId): DELETE

**File:** notification.js

**Function Use:** deleteNotification ()

**Description:** The deleteNotification API allows clients to delete a notification by its ID. Clients need to provide the notification ID in the route parameter. Upon receiving the request, the API finds the notification by its ID and deletes it. The API responds with a success message upon successful deletion.

**API Name:** Bulk Delete Notifications (/notification/bulk-delete): POST

**File:** notification.js

**Function Use:** bulkDeleteNotifications ()

**Description:** The bulkDeleteNotifications API allows clients to delete multiple notifications in bulk. Clients need to provide an array of notification IDs in the request body. Upon receiving the request, the API performs a bulk delete operation for all the provided notification IDs. The API responds with a success message upon successful deletion.

**API Name:** Send SMS (/send-sms): POST

**File:** sms.js

**Function Use:** sendSMS()

**Description:** The sendSMS API allows clients to send SMS messages. Clients need to provide the mobile number and message content in the request body. Upon receiving the request, the API calls the sendSMS function, passing the mobile number and message content. The sendSMS function sends the SMS message using the provided mobile number and message content. The API responds with the result of the sendSMS function call.

* **Order Micro Service:**

**API Name:** Get All Coupon with pagination (coupon/all): GET

**File:** coupon.js

**Function Use:** getAllCouponswithPagination ()

**Description:** Retrieves all coupons with pagination. Clients can specify the sorting order, search criteria, and pagination parameters in the query string. If the isDownload query parameter is set to true, it returns all coupons without pagination. Each coupon contains information such as its code, discount type, discount amount, expiry date, isActive status, usage count, and description.

**API Name:** Get Coupon without pagination (/coupon /): GET

**File:** coupon.js

**Function Use:** getAllCoupons ()

**Description:** Retrieves all coupons without pagination. Each coupon contains information such as its code, discount type, discount amount, expiry date, isActive status, usage count, and description.

**API Name:** Get Coupon By Id (/coupon /: couponId): GET

**File:** coupon.js

**Function Use:** getCouponById ()

**Description:** Retrieves a single coupon by its unique ID. Clients need to provide the coupon ID in the URL parameter. The response contains detailed information about the coupon, including its code, discount type, discount amount, expiry date, isActive status, usage count, and description.

**API Name:** Create Coupon (/coupon): POST

**File:** coupon.js

**Function Use:** createCoupon ()

**Description:** Creates a new coupon. Clients need to provide coupon details such as coupon code, discount type, discount amount, expiry date, isActive status, usage count, and description in the request body. Upon successful creation, the API returns a success message along with the newly created coupon details.

**API Name:** Update Coupon (/coupon/: couponId): PUT

**File:** coupon.js

**Function Use:** updateCoupon ()

**Description:** Updates an existing coupon. Clients need to provide the coupon ID in the URL parameter and the updated coupon details in the request body. Supported updateable fields include coupon code, discount type, discount amount, expiry date, isActive status, usage count, and description. Upon successful update, the API returns a success message.

**API Name:** Update Coupon Status (/coupon/ status/: couponId): PUT

**File:** coupon.js

**Function Use:** updateCouponStatus ()

**Description:** Updates the status of a coupon (isActive). Clients need to provide the coupon ID in the URL parameter and the updated isActive status in the request body. This API is useful for activating or deactivating coupons. Upon successful status update, the API returns a success message.

**API Name:** Delete Coupon (coupon/: couponId): DELETE

**File:** coupon.js

**Function Use:** deleteCoupon ()

**Description:** Deletes a coupon by its ID. Clients need to provide the coupon ID in the URL parameter. Upon successful deletion, the API returns a success message indicating that the coupon has been removed from the system.

**API Name:** Bulk Delete Coupon (/coupon /bulk-delete): POST

**File:** coupon.js

**Function Use:** bulkDeleteCoupons ()

**Description:** Performs bulk deletion of coupons. Clients need to provide an array of coupon IDs in the request body to delete multiple coupons at once. Upon successful bulk deletion, the API returns a success message indicating that the specified coupons have been removed from the system.

**API Name:** Get All Order list with pagination (/list): GET

**File:** order.js

**Function Use:** listOrders ()

**Description:** Retrieves a list of orders with pagination. Clients can specify various filters such as order status, start date, end date, and sorting order. Pagination parameters like offset, perPage, and currentPage can also be provided in the query string. Additionally, clients can choose to download all orders without pagination by setting the isDownload query parameter to true.

**API Name:** Create Order (/coupon): POST

**File:** order.js

**Function Use:** addOrder ()

**Description:** Creates a new order. Clients need to provide details such as customer ID, billing and shipping details, order items, and an optional coupon code in the request body. The API validates the order items, checks product availability, calculates the total order amount, and creates the order along with associated order items. It also updates the product quantities and sends an email notification to the customer upon successful order creation.

**API Name:** Update Order (/:orderId): PUT

**File:** order.js

**Function Use:** updateOrder ()

**Description:** Updates an existing order identified by its ID. Clients need to provide the order ID in the URL parameter and the updated order details in the request body. The API handles updating order items, recalculating the order amount, adjusting product quantities, and updating the order status. It also sends email notifications for order status updates and creates activity logs for the order.

**API Name:** Update Order Status (/status/: orderId): PUT

**File:** order.js

**Function Use:** updateOrderStatus ()

**Description:** Updates the status of an existing order identified by its ID. Clients need to provide the order ID in the URL parameter and the updated order status along with user email and username in the request body. The API updates the order status, sends email notifications to the customer, and creates activity logs for the order status update.

**API Name:** Delete Order (/:orderId): DELETE

**File:** order.js

**Function Use:** deleteOrder()

**Description:** Deletes an existing order identified by its ID. Clients need to provide the order ID in the URL parameter. The API deletes the order along with associated order items, increments product quantities for each order item, and sends an email notification to the customer upon successful order deletion.

**API Name:** Bulk Delete Order (/bulk-delete): POST

**File:** order.js

**Function Use:** bulkDeleteOrders ()

**Description:** Deletes multiple orders specified by their IDs in bulk. Clients need to provide an array of order IDs in the request body. The API validates the existence of all orders, deletes each order along with associated order items, increments product quantities for each order item, and sends email notifications to the customers upon successful order deletion.

* **Payment Micro Service:**

**API Name:** Get All PaymentGateways with pagination (payment-gateway/all): GET

**File:** paymentGateway.js

**Function Use:** getAllPaymentGatewaywithPagination ()

**Description:** Retrieves all payment gateways with pagination. Clients can specify the sorting order, search criteria, and pagination parameters in the query string. If the isDownload query parameter is set to true, it returns all payment gateways without pagination. Each payment gateway contains information such as its name, URL, image, mode, ID, secret, merchant details, client details, and additional information.

**API Name:** Get All PaymentGateways without pagination (/payment-gateway /): GET

**File:** paymentGateway.js

**Function Use:** getAllPaymentGateways ()

**Description:** This API retrieves all payment gateways from the database with the status set to "active". It internally calls the getAllPaymentGateways function with the filter criteria { status: 'active' }. Upon successful retrieval of payment gateways, it returns a JSON response with a 200 status code containing the fetched data. If an error occurs during the process, it logs the error and returns a JSON response with a 400 status code along with an error message.

**API Name:** Get One PaymentGateway (/payment-gateway/:paymentGatewayId): GET

**File:** paymentGateway.js

**Function Use:** getPaymentGatewayById ()

**Description:** Retrieves a single payment gateway by its ID. Clients must provide the ID of the payment gateway in the URL path. The response includes detailed information about the payment gateway, including its name, URL, image, mode, ID, secret, merchant details, client details, and additional information.

**API Name:** Create PaymentGateway (/payment-gateway/): POST

**File:** paymentGateway.js

**Function Use:** createPaymentGateway ()

**Description:** Creates a new payment gateway. Clients must provide details such as the payment gateway's name, URL, image, mode, ID, secret, merchant details, client details, and additional information in the request body. Upon successful creation, a confirmation message is returned.

**API Name:** Update PaymentGateway (/payment-gateway/:paymentGatewayId): PUT

**File:** paymentGateway.js

**Function Use:** updatePaymentGateway ()

**Description:** Updates an existing payment gateway. Clients must provide the ID of the payment gateway to be updated in the URL path and the updated details in the request body. Upon successful update, a confirmation message is returned.

**API Name:** Update PaymentGateway Status (/payment-gateway/status/:paymentGatewayId): PUT

**File:** paymentGateway.js

**Function Use:** deletePaymentGateway ()

**Description:** This API endpoint updates the status of a PaymentGateway in the database. It expects the paymentGatewayId as a route parameter and the new status value in the request body. Upon receiving the request, it updates the status of the PaymentGateway with the provided ID to the new status value. If the operation is successful, it returns a success message along with a status code of 200. If any error occurs during the process, it returns an error message along with a status code of 400.

**API Name:** Get All Transactions with Pagination (/transactions/ all): GET

**File:** transaction.js

**Function Use:** getAllTransactionwithPagination ()

**Description:** Retrieves all transactions with pagination support. This API allows fetching transactions with optional filtering, sorting, and pagination parameters. It accepts query parameters such as orderBy, order, search, isDownload, and filter to customize the result set. If downloading is requested (isDownload set to true), it returns all transactions without pagination. Otherwise, it returns paginated data including total count, current page, total pages, and transaction details. This API ensures flexibility in querying and retrieving transaction data efficiently.

**API Name:** Rozerpay Webhook API (/transactions/ rozerpay): POST

**File:** rozerpay.js

**Function Use:** webHook()

**Description:** Handles webhook events from Razorpay. This API is responsible for processing payment-related events such as successful payment capture and failed payments. Upon receiving a webhook event, it processes the event payload and updates the transaction status accordingly in the database. Additionally, it calls the callUpdateOrderStatusAPI function to update the order status based on the payment event. This API ensures synchronization between payment transactions and order status updates.

**API Name:** Paypal Webhook API (/transactions/paypal): POST

**File:** paypal.js

**Function Use:** paypalWebhook ()

**Description:** Handles webhook events from PayPal. This API is responsible for processing payment-related events such as completed payment capture and denied payments. Upon receiving a webhook event, it processes the event payload and extracts relevant information such as payment ID, amount, status, order ID, and customer ID. It then fetches additional customer and order details using the extracted IDs. The API determines the transaction status based on the PayPal event and creates a new transaction in the database with the appropriate details. Additionally, it updates the order status based on the transaction status, ensuring synchronization between payment transactions and order status updates. If the payment event indicates a denied transaction, the order status is set to 'Cancelled'.

**API Name:** PaytmWebhook API (/transactions/paytm): POST

**File:** paytm.js

**Function Use:** paytmWebhook ()

**Description:** Handles webhook events from Paytm. This API is responsible for processing payment-related events such as successful payment capture and failed payments. Upon receiving a webhook event, it processes the event payload and extracts relevant information such as ORDERID, TXNID, TXNAMOUNT, STATUS, BANKTXNID, BANKNAME, and customerId. It then fetches additional customer and order details using the extracted ORDERID and customerId. The API determines the transaction status based on the Paytm status ('TXN\_SUCCESS', 'TXN\_FAILURE', etc.). It creates a new transaction in the database with the extracted details and updates the order status based on the transaction status. If the transaction status indicates success, the order status is set to 'Processing'; otherwise, it's set to 'Cancelled'.

**API Name:** Payumoney Webhook API (/transactions/payumoney): POST

**File:** payumoney.js

**Function Use:** payUmoneyWebhook ()

**Description:** Handles webhook events from PayUmoney. This API is responsible for processing payment-related events such as successful payment and failed payments. Upon receiving a webhook event, it processes the event payload and extracts relevant information such as txnid, amount, status, orderId, and customerId. It then fetches additional order details using the orderId and customer details using the customerId. The API determines the transaction status based on the PayUmoney event ('success' or 'failed'). It creates a new transaction in the database with the extracted details and updates the order status based on the transaction status. If the event indicates a failed payment, the order status is set to 'Cancelled'; otherwise, it's set to 'Processing'.

**API Name:** Stripe Webhook API (/transactions/stripe): POST

**File:** stripe.js

**Function Use:** stripeWebhook ()

**Description:** Handles webhook events from Stripe. This API is responsible for processing payment-related events such as successful payment and failed payments. Upon receiving a webhook event, it processes the event payload and extracts relevant information such as id, amount, status, charges, and customer. It then fetches additional order details using orderId extracted from charges metadata and customer details using the customerId. The API determines the transaction status based on the Stripe event ('succeeded' or 'failed'). It creates a new transaction in the database with the extracted details and updates the order status based on the transaction status. If the event indicates a failed payment, the order status is set to 'Cancelled'; otherwise, it's set to 'Processing'.