**API Documentation**

**🔄 Standard API Response Structure**

All API responses follow a consistent structure defined by the Response<T> wrapper.

**✅ Response Format**

{

"statusCode": 200,

"message": "Success",

"data": { /\* Actual response payload \*/ },

"responseStatus": "SUCCESS"

}

**📌 Fields Description**

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| statusCode | int | HTTP-like status code representing the result (e.g., 200 for success, 400 for bad request). |
| message | String | A human-readable message describing the result. |
| data | T | The actual data returned by the API (e.g., Product, List<Product>, Boolean, etc.). |
| responseStatus | ResponseStatus | Enum representing status such as SUCCESS, FAILURE, ERROR, etc. |

**🧾 Sample Responses**

**✔ Success Response (Product Retrieval)**

{

"statusCode": 200,

"message": "Product retrieved successfully",

"data": {

"id": "123",

"name": "Product A",

"price": 99.99,

...

},

"responseStatus": "SUCCESS"

}

**❌ Error Response (Product Not Found)**

{

"statusCode": 404,

"message": "Product with id '123' not found",

"data": null,

"responseStatus": "FAILURE"}

## 📦 ProductRepo Java API Documentation

### Overview

The ProductRepo interface defines methods for managing products in a system. It provides functionality to add, update, delete, and retrieve products, along with handling bulk updates and inventory modifications.

To use this API, simply import the JAR file containing the ProductRepo interface and get its instance by ProductRepository.getInstance().

### ****1. addProduct(Product product)****

* **Description:** Adds a new product to the repository.
* **Parameters:**
  + product (Product): The product object to be added.
* **Returns:** Response<Product> – A response containing the added product.

**Response Object Example:**

Response<Product> response = productRepo.addProduct(new Product("123", "Product A", 99.99, 10));

Product addedProduct = response.getData();

### ****2. updateProduct(Product product)****

* **Description:** Updates an existing product in the repository.
* **Parameters:**
  + product (Product): The product object with updated details.
* **Returns:** Response<Product> – A response containing the updated product.

**Response Object Example:**

Response<Product> response = productRepo.updateProduct(new Product("123", "Updated Product", 89.99, 20));

Product updatedProduct = response.getData();

### ****3. deleteProduct(String id)****

* **Description:** Deletes a product based on its unique ID.
* **Parameters:**
  + id (String): The UUID of the product to be deleted.
* **Returns:** Response<Boolean> – A response indicating whether the deletion was successful (true) or not (false).

**Response Object Example:**

Response<Boolean> response = productRepo.deleteProduct("123");

boolean isDeleted = response.getData();

### ****4. getAllProducts()****

* **Description:** Retrieves a list of all products from the repository.
* **Returns:** Response<List<Product>> – A response containing the list of products.

**Response Object Example:**

Response<List<Product>> response = productRepo.getAllProducts();

List<Product> allProducts = response.getData();

### ****5. findProductById(String id)****

* **Description:** Finds a product based on its unique ID.
* **Parameters:**
  + id (String): The UUID of the product to be retrieved.
* **Returns:** Response<Product> – A response containing the product if found, or an appropriate error message if not found.

**Response Object Example:**

Response<Product> response = productRepo.findProductById("123");

Product product = response.getData();

### ****6. updateProducts(List<Product> list)****

* **Description:** Updates multiple products in the repository at once.
* **Parameters:**
  + list (List<Product>): A list of product objects with updated details.
* **Returns:** Response<Object> – A response indicating the success or failure of the bulk update operation.

**Response Object Example:**

List<Product> productList = Arrays.asList(new Product("123", "Product A", 99.99, 10));

Response<Object> response = productRepo.updateProducts(productList);

Object result = response.getData();

### ****7. updateProductInventory(String uuid, float updateQuantity)****

* **Description:** Updates the inventory of a product by the given quantity.
* **Parameters:**
  + uuid (String): The UUID of the product.
  + updateQuantity (float): The quantity to be added or subtracted from the existing inventory.
* **Returns:** Response<Boolean> – A response indicating whether the inventory update was successful (true) or not (false).

**Response Object Example:**

Response<Boolean> response = productRepo.updateProductInventory("123", 5.0f);

boolean isUpdated = response.getData();

### Dependencies

To use the ProductRepo API, include the following dependencies in your project:

1. **ProductRepo Interface** - Import the JAR file that contains the ProductRepo interface and its implementation ProductRepository.
2. **Product Class** - The Product class must have properties such as id, name, price, and quantity (or similar fields depending on your system).
3. **Response Class** - A generic class Response<T> to wrap all responses, including success or error status.

**📦 ClientRepo Java API Documentation**

**Overview**

The ClientRepo interface provides methods for managing **Client** entities in a repository. These methods allow you to retrieve clients, add new clients, update client details, and fetch clients based on specific criteria (e.g., with associated orders).

To use this API, import the JAR containing the ClientRepo interface and get its instance by ClientRepository.getInstance().

**1. getAllClient()**

* **Description:** Retrieves all clients from the repository.
* **Returns:** Response<List<Client>> – A response containing a list of all clients.

**Response Object Example:**

Response<List<Client>> response = clientRepo.getAllClient();

List<Client> allClients = response.getData();

**2. addClient(Client client)**

* **Description:** Adds a new client to the repository.
* **Parameters:**
  + client (Client): The client object to be added.
* **Returns:** Response<Client> – A response containing the added client.

**Response Object Example:**

Client newClient = new Client("123", "John Doe", "johndoe@example.com");

Response<Client> response = clientRepo.addClient(newClient);

Client addedClient = response.getData();

**3. updateClient(Client client)**

* **Description:** Updates an existing client's details in the repository.
* **Parameters:**
  + client (Client): The client object with updated details.
* **Returns:** Response<Client> – A response containing the updated client.

**Response Object Example:**

Client updatedClient = new Client("123", "John Doe", "newemail@example.com");

Response<Client> response = clientRepo.updateClient(updatedClient);

Client client = response.getData();

**4. findById(String id)**

* **Description:** Retrieves a client by their unique ID.
* **Parameters:**
  + id (String): The unique ID of the client to be fetched.
* **Returns:** Response<Client> – A response containing the client if found, or an appropriate error message if not found.

**Response Object Example:**

Response<Client> response = clientRepo.findById("123");

Client client = response.getData();

**5. getClientWithOrderList(String id)**

* **Description:** Retrieves a client along with a list of orders associated with them.
* **Parameters:**
  + id (String): The unique ID of the client whose orders are to be fetched.
* **Returns:** Response<Client> – A response containing the client and their associated order list, if available.

**Response Object Example:**

Response<Client> response = clientRepo.getClientWithOrderList("123");

Client clientWithOrders = response.getData();

**Dependencies**

To use the ClientRepo API, include the following dependencies in your project:

1. **ClientRepo Interface** - Import the JAR file that contains the ClientRepo interface and its implementation.
2. **Client Class** - The Client class must include properties such as id, name, email, and associated orders (if applicable).
3. **Response Class** - A generic class Response<T> to wrap all responses, including success or error status.

**📦 OrderRepo Java API Documentation**

**Overview**

The OrderRepo interface provides methods for managing **Order** entities in a repository. It includes functionality to retrieve orders, update them, delete them, check out orders, and verify whether an order can be checked out.

To use this API, simply import the JAR file containing the OrderRepo interface and get its instance from OrderRepository.getInstance()method.

**1. getAllOrder()**

* **Description:** Retrieves a list of all orders from the repository.
* **Returns:** Response<List<Order>> – A response containing the list of all orders.

**Response Object Example:**

Response<List<Order>> response = orderRepo.getAllOrder();

List<Order> allOrders = response.getData();

**2. findByOrderId(String id)**

* **Description:** Retrieves an order by its unique ID.
* **Parameters:**
  + id (String): The unique ID of the order to be fetched.
* **Returns:** Response<Order> – A response containing the order if found, or an appropriate error message if not found.

**Response Object Example:**

Response<Order> response = orderRepo.findByOrderId("123");

Order order = response.getData();

**3. deleteOrder(String id)**

* **Description:** Deletes an order by its unique ID.
* **Parameters:**
  + id (String): The unique ID of the order to be deleted.
* **Returns:** Response<Boolean> – A response indicating whether the deletion was successful (true) or not (false).

**Response Object Example:**

Response<Boolean> response = orderRepo.deleteOrder("123");

boolean isDeleted = response.getData();

**4. updateOrder(Order order)**

* **Description:** Updates an existing order in the repository.
* **Parameters:**
  + order (Order): The order object with updated details.
* **Returns:** Response<Order> – A response containing the updated order.

**Response Object Example:**

Order updatedOrder = new Order("123", "Product A", 2, 50.0);

Response<Order> response = orderRepo.updateOrder(updatedOrder);

Order order = response.getData();

**5. doCheckOut(Order order)**

* **Description:** Performs the checkout process for a given order.
* **Parameters:**
  + order (Order): The order to be checked out.
* **Returns:** Response<Order> – A response containing the order after the checkout process.

**Response Object Example:**

Order orderToCheckout = new Order("123", "Product A", 2, 50.0);

Response<Order> response = orderRepo.doCheckOut(orderToCheckout);

Order checkedOutOrder = response.getData();

**6. isAbleToCheckOut(Order order)**

* **Description:** Checks whether the given order can be checked out based on business rules (e.g., payment status, inventory availability).
* **Parameters:**
  + order (Order): The order to be checked for checkout eligibility.
* **Returns:** Response<Boolean> – A response indicating whether the order can be checked out (true) or not (false).

**Response Object Example:**

Response<Boolean> response = orderRepo.isAbleToCheckOut(order);

boolean canCheckout = response.getData();

**Dependencies**

To use the OrderRepo API, include the following dependencies in your project:

1. **OrderRepo Interface** - Import the JAR file that contains the OrderRepo interface and its implementation OrderRepository.
2. **Order Class** - The Order class must include properties such as id, productName, quantity, price, and any additional relevant order details.
3. **Response Class** - A generic class Response<T> to wrap all responses, including success or error status.

**🏬 StoreRepo Java API Documentation**

**Overview**

The StoreRepo interface provides methods for **authenticating store users**, retrieving the authenticated store user, accessing the service layer, and logging out. This interface is part of a Java API and is intended for internal service-to-service communication or embedding within Java-based enterprise applications.

**1. Authenticate(String username, String password)**

* **Description:** Authenticates the user using the provided username and password.
* **Parameters:**
  + username (String): The username of the store user.
  + password (String): The corresponding password.
* **Returns:** Response<Object> – A response object that includes authentication status or token data (such as a JWT).

**Example Usage:**

Response<Object> response = storeRepo.Authenticate("admin", "admin@123");

if (response.getResponseStatus() == ResponseStatus.SUCCESS) {

System.out.println("Login successful!");

} else {

System.out.println("Authentication failed: " + response.getMessage());

}

**2. getUser()**

* **Description:** Returns the currently authenticated store user.
* **Returns:** Response<Store> – A response containing the authenticated Store object.

**Example Usage:**

Response<Store> response = storeRepo.getUser();

Store loggedInStore = response.getData();

System.out.println("Logged in user: " + loggedInStore.getStoreName());

**3. getService()**

* **Description:** Returns the internal Service object used for validating and authorizing service access (e.g., contains JWT token or session).
* **Returns:** Service – A Service object used for API security and request signing.

**Example Usage:**

Service service = storeRepo.getService();

String jwtToken = service.getPassKey(); // Example usage

**4. logout()**

* **Description:** Logs out the current user and invalidates any session or token associated with them.
* **Returns:** void

**Example Usage:**

storeRepo.logout();

System.out.println("User logged out successfully.");

**Supporting Classes**

**✅ Response<T>**

A generic wrapper for API responses. This class contains:

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| statusCode | int | Status code (e.g., 200 for success) |
| message | String | Descriptive message |
| data | T | Returned data object (e.g., Store, Object) |
| responseStatus | ResponseStatus | Enum indicating success/failure/error |

**🧾 Store**

Represents a store user with fields like:

* String storeId
* String storeName
* String email
* String role
* Other metadata (e.g., creation date, permissions, etc.)

**🛡️**

**Service**

An object that carries session-specific data such as authentication credentials or access tokens.

| **Method** | **Return Type** | **Description** |
| --- | --- | --- |
| getPassKey() | String | Returns the JWT or session token |
| getName() | String | Returns the ID of the currently authenticated user |
| getEmail() | String | Returns the user's email |
| getUUID | String | Returns the user’s UUID |

**Dependencies**

* JAR file containing the StoreRepo interface and related classes like Store, Response<T>, and Service.

**📦 InventoryTransactionsRepo Java API Documentation**

**Overview**

The InventoryTransactionsRepo interface provides a contract for performing **CRUD operations** and **bulk updates** on inventory transaction records. It is designed for internal use in service layers, libraries, or applications that consume inventory services via Java API (not REST).

**🧾 Method Documentation**

**1. createTrasaction(InventoryTransactions transactions)**

* **Description:** Creates a new inventory transaction.
* **Parameters:**
  + transactions – InventoryTransactions object containing transaction details.
* **Returns:** Response<Object> – Response indicating success or failure, and optionally contains the created transaction or a message.

**2. updateTransaction(InventoryTransactions transactions)**

* **Description:** Updates an existing inventory transaction.
* **Parameters:**
  + transactions – InventoryTransactions object with updated fields.
* **Returns:** Response<Object> – Response indicating the result of the update operation.

**3. deleteTransaction(String uuid)**

* **Description:** Deletes a specific inventory transaction using its unique UUID.
* **Parameters:**
  + uuid – String: The UUID of the transaction to delete.
* **Returns:** Response<Integer> – Response with number of rows deleted (typically 1 or 0).

**4. findAllTransactionsByProductUUID(String uuid)**

* **Description:** Retrieves all inventory transactions associated with a specific product UUID.
* **Parameters:**
  + uuid – String: The UUID of the product.
* **Returns:** Response<List<InventoryTransactions>> – List of matching transactions.

**5. createTrasactions(List<InventoryTransactions> list)**

* **Description:** Creates multiple inventory transactions in a single batch operation.
* **Parameters:**
  + list – List<InventoryTransactions>: List of transactions to create.
* **Returns:** Response<Object> – Response with batch creation result.

**6. updateTransactionWithInvoiceNo(List<InventoryTransactions> list, String invoiceNo)**

* **Description:** Updates multiple inventory transactions associated with a specific invoice number.
* **Parameters:**
  + list – List<InventoryTransactions>: Transactions to update.
  + invoiceNo – String: The invoice number for grouping.
* **Returns:** Response<Object> – Response indicating the outcome of the batch update.

**7. findfindAllTransactionsByInvoiceNo(String invoiceNo)**

* **Description:** Retrieves all transactions linked to a given invoice number.
* **Parameters:**
  + invoiceNo – String: The invoice number to search by.
* **Returns:** Response<List<InventoryTransactions>> – List of transactions associated with the invoice.

📝 **Note:** Method name has a typo: findfindAllTransactionsByInvoiceNo. It should likely be findAllTransactionsByInvoiceNo.

**8. deleteTransactions(List<InventoryTransactions> list)**

* **Description:** Deletes multiple inventory transactions in bulk.
* **Parameters:**
  + list – List<InventoryTransactions>: Transactions to delete.
* **Returns:** Response<Integer> – Number of records deleted.

**✅ Supporting Types**

**InventoryTransactions**

Represents a transaction for a product in the inventory. Typical fields may include:

* String transactionId
* String productUUID
* float quantity
* String invoiceNo
* Date transactionDate
* TransactionType type (e.g., IN, OUT)

**Response<T>**

A wrapper for method responses that includes status and result.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| statusCode | int | Status code (e.g., 200, 400) |
| message | String | Message string |
| data | T | Response data |
| responseStatus | ResponseStatus | Status enum (SUCCESS, FAILURE, etc.) |

**🛠 Example Usage**

InventoryTransactionsRepo repo = InventoryTransactionsRepository.getInstance();

// Create a new transaction

InventoryTransactions txn = new InventoryTransactions(...);

Response<Object> createRes = repo.createTrasaction(txn);

// Update with invoice number

List<InventoryTransactions> txns = List.of(txn1, txn2);

Response<Object> updateRes = repo.updateTransactionWithInvoiceNo(txns, "INV123");

// Delete multiple

Response<Integer> deleteRes = repo.deleteTransactions(txns);