|  |  |
| --- | --- |
| **Service Name** | Product CRUD |
| **Service Version** | 1.0 |
| **Repo Name** | https://github.com/AllenGeo/Product\_CRUD |
| **Artifact ID** | product-crud |
| **Maven Group ID** | com.ust. service. product |
| **Overall Purpose and Requirement** | The E-commerce business operating with a set of RESTful services to manage its product inventory efficiently. These services include the ability to retrieve item details using an item\_id, fetch all items under a specific category\_id, and filter special products with an optional parameter. Additionally, there is a service to add new items to the inventory, checking for duplicates before insertion, and an update service to modify the stock details for multiple items. These services ensure smooth operations, accurate inventory management, and an enhanced shopping experience for customers. |

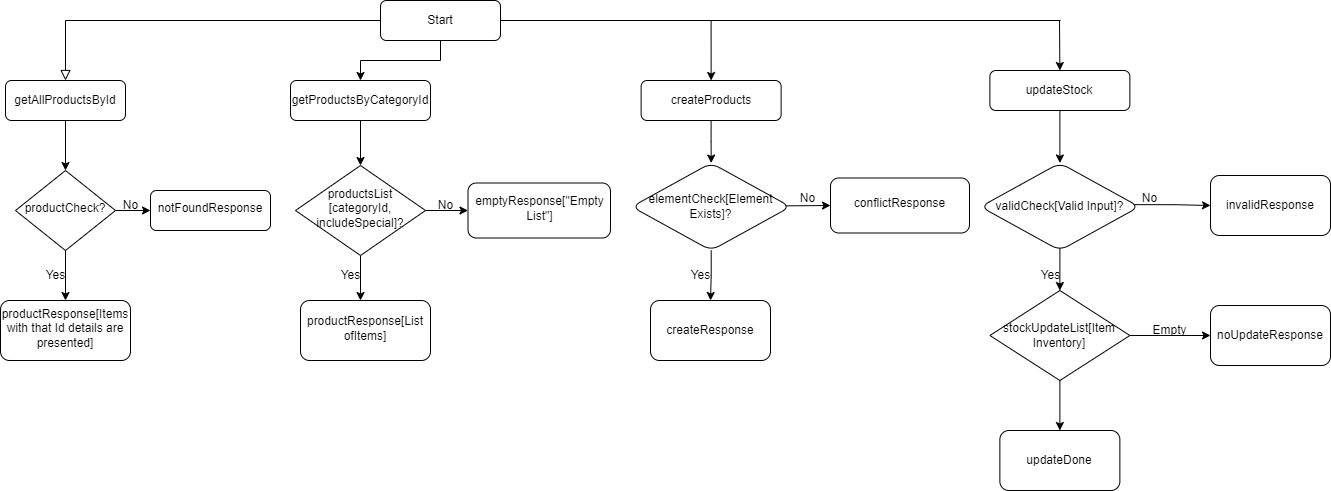
# Tech Spec Overview

# Document Version History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Date** | **Document Version** | **Revised By** | **Description** | **Story or Change Number** | **Contact(s)** |
| 24-07-2023 | 1.0 | Allen Jacob George (UST,IN) | An initial draft of product services |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# 

# Architecture Diagram



# Service Design

**Design**: Product CRUD

|  |  |  |
| --- | --- | --- |
| **Database Table(s)** | | |
| **Name** | **Type** | **Comment(s)** |
| Database – Itemsdb  Collection - Item | MongoDB |  |

|  |  |  |
| --- | --- | --- |
| **ReST Endpoint** | | |
| **Attribute** | **Value** | **Comment** |
| URL (APIGEE) | <Apigee domain >/catalog |  |
| URL | [http://localhost:8080/products/{\_id}](http://localhost:8080/products/%7b_id%7d)  [http://localhost:8080/products/category/{ categoryId }?includeSpecial=true](http://localhost:8080/products/category/%7b%20categoryId%20%7d?includeSpecial=true)  <http://localhost:8080/products>  <http://localhost:8080/products/updateStock> | GET [By getting \_id]  GET [By getting categoryId]  POST []  PUT [] |
| HTTP Method | POST, GET, PUT |  |
| Request Format(s) | XML, JSON |  |
| Response Format(s) | XML, JSON |  |
| HTTP Response Codes | * 200 – Valid request * 500 – Internal Error |  |

**JPA**

|  |  |  |
| --- | --- | --- |
| **Properties <Artifact\_ID.properties>** | | |
| **Name** | **Value** | **Comment(s)** |
| spring.data.mongodb.host | localhost |  |
| spring.data.mongodb.port | 27017 |  |
| spring.data.mongodb.collection | Item |  |
| spring.data.mongodb.database | Itemsdb |  |
| server.port | 8080 |  |

# Reference Document(s)

|  |  |
| --- | --- |
|  |  |
| **Document Name** | **Attachment** |
| Data Mapping |  |
| Request Payload |  |
| Response Payload |  |
| MongoDB model |  |
| API Doc |  |
| Sample MongoDB Query |  |

# Process

|  |
| --- |
| **Programming Objects** |
| **getAllProductsById service**   * HTTP Request: The service handles a GET request with a path variable \_id, representing the unique identifier of the product. * Product Found: If the product exists in the database, a 200 OK HTTP response is returned with the product data in the response body. * Product Not Found: If the product is not found, a 404 Not Found HTTP response is returned with an error message indicating that the product with the given \_id was not found. |
| **getProductsByCategoryId service**   * HTTP Request: The service is designed to handle GET requests with the path variable categoryId, which represents the category identifier. The service can also accept an optional query parameter includeSpecial, which is a boolean value. * Response with Products: If products are found in the database (!products.isEmpty()), the service returns a 200 OK response with the list of products in the response body using ResponseEntity.ok(products). The list of products is serialized into JSON format. * No Products Found: If no products are found for the given categoryId, the service returns a 404 Not Found response with an empty list in the response body using ResponseEntity.status(HttpStatus.NOT\_FOUND).body(Collections.emptyList()). |
| **createProduct service**   * HTTP Request: The service handles a POST request to the endpoint "/products". * Product Retrieval: The service first attempts to check if a product with the same \_id as provided in the ProductRequest already exists in the database. It does this by calling productRepository.findById(productRequest.get\_id()). * Decision Point: The service checks whether a product with the same \_id exists in the database or not. If the \_id is unique (i.e., !element.isPresent()), it proceeds with product creation. * Response with New Product: After creating the new product, the service saves it to the database using this.productRepository.save(product). It then returns a 200 OK response with the newly created product in the response body using ResponseEntity.ok(). The response will contain the product data in JSON format. * Conflict Handling: If a product with the same \_id already exists in the database (i.e., element.isPresent()), the service returns a 409 Conflict response with a null response body. This indicates that a product with the provided \_id cannot be created as it already exists. |
| **updateStock service**   * HTTP Request: The service handles a PUT request to the endpoint "/products/updateStock". * Validation: The service starts by checking if the request body contains the "items" key with a non-empty list. If the request format is invalid (no "items" key or empty list), it returns a 400 Bad Request response with an error message. * Product Retrieval: For each ItemInventory object, the service extracts the \_id (product identifier) and the updated stock details (soldOut and damaged quantities). * Stock Calculation: The service retrieves the product from the database using productRepository.findById(\_id). If the product exists in the database, it calculates the new available stock quantity by deducting the soldOut and damaged quantities from the current available stock. * Stock Update: The service updates the product's stockDetails with the newly calculated available stock quantity and saves the product back to the database using productRepository.save(items). * Response: After updating the stock details for all applicable products, the service returns a response. If at least one product was successfully updated (i.e., updated is true), it returns a 200 OK response with the message "Stock details updated successfully." If no products were found for updating (i.e., updated is false), it returns a 404 Not Found response with the message "No products found to update." |

# Test Cases

|  |  |
| --- | --- |
| **Test Case(s)** | |
| **Case** | **Expected Result** |
|  |  |
|  |  |

**Performance Document**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Performance** |  |  |  |  |  |  |
| **Acceptance Criteria** | | | | | |  |
| **Type** | **Volume** | **In Service Time** | **DownStream Time** | **Response Time** | **Requestor Approved** |  |
| *(Calls per Second)* | *(90th Percentile -- in Milliseconds)* | | |  |
| Acceptance Criteria |  |  |  |  |  |  |
| Current Baseline |  |  |  |  |  |  |
| Goal/Target Metrics |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **Measurements** | | | | | | |
| **Date of Performance Measurement** | **Measurement Method** | **Volume** | **In Service Time** | **DownStream Time** | **Response Time** | **Growth Pattern** |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| **Performance Document** | *<Attach the performance document here>* |