**Final Solution**

**Core Functionalities:**

**Item management:** Expand on the basic CRUD operations to include additional item details like description, category etc.

**Transactions:** Enhance transaction capabilities to include purchase orders, returns, and transfers between locations.

**Item history:** Implement a detailed item history view, including transaction type, date, quantity, and price.

**Filtering:** Improve filtering options to include multiple criteria (e.g., category, price range, quantity).

**User interface:** Maintain a user-friendly Angular interface with enhanced features like data visualization for inventory analysis.

**Backend:** Utilize Spring Boot for robust backend services, including REST APIs, data persistence with Spring Data JPA.

**Additional Features**

**Filtering:** In this feature we can search the product in the system.

**Technology Stack**

**Frontend:** Angular, Angular Material, TypeScript

**Backend:** Spring Boot, Spring Data JPA

**Database:** h2

**Deployment:** Docker, Docker Compose

**Development Approach**

**Iterative development:** Prioritize core functionalities and add features incrementally.

**Testing:** Implement unit, integration, and end-to-end tests to ensure quality.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Solution 1** | **Solution 2** | **Final Solution** |
| Item Management | Basic CRUD | Basic CRUD with selling/inserting | Comprehensive CRUD, additional details, supplier, category |
| Transactions | None | Selling, Inserting | Purchase orders, returns, transfers, history |
| Item History | None | None | Detailed history with transaction types, dates, quantities, prices |
| Filtering | None | By ID or name | Multiple criteria, advanced search |
| User Interface | Angular | Angular | Enhanced UI with data visualization |
| Backend | Spring Boot | Spring Boot | Spring Boot with advanced features, potential message queue |

### **Components**

1. **Define the Problem:** This is the initial stage where the problem or challenge is identified and clearly defined.
2. **Do Background Research:** Gathering information and understanding the context of the problem is crucial for informed decision-making.
3. **Specify Requirements:** Outlining the specific needs, goals, and constraints of the solution is essential for guiding the development process.
4. **Brainstorm, Evaluate, and Choose Solution:** Generating multiple ideas, assessing their feasibility, and selecting the most promising approach is a critical step.
5. **Develop and Prototype Solution:** Creating a preliminary version of the solution allows for testing and refinement before final implementation.
6. **Test Solution:** Evaluating the solution's effectiveness, identifying any shortcomings, and making necessary adjustments is an ongoing process.
7. **Communicate Results:** Sharing the findings, outcomes, and insights with stakeholders is essential for knowledge transfer and impact.

**Main Purpose of Each Component:**

**Define the Problem:** Clearly articulates the challenge to be addressed.

**Do Background Research:** Provides context and informs decision-making.

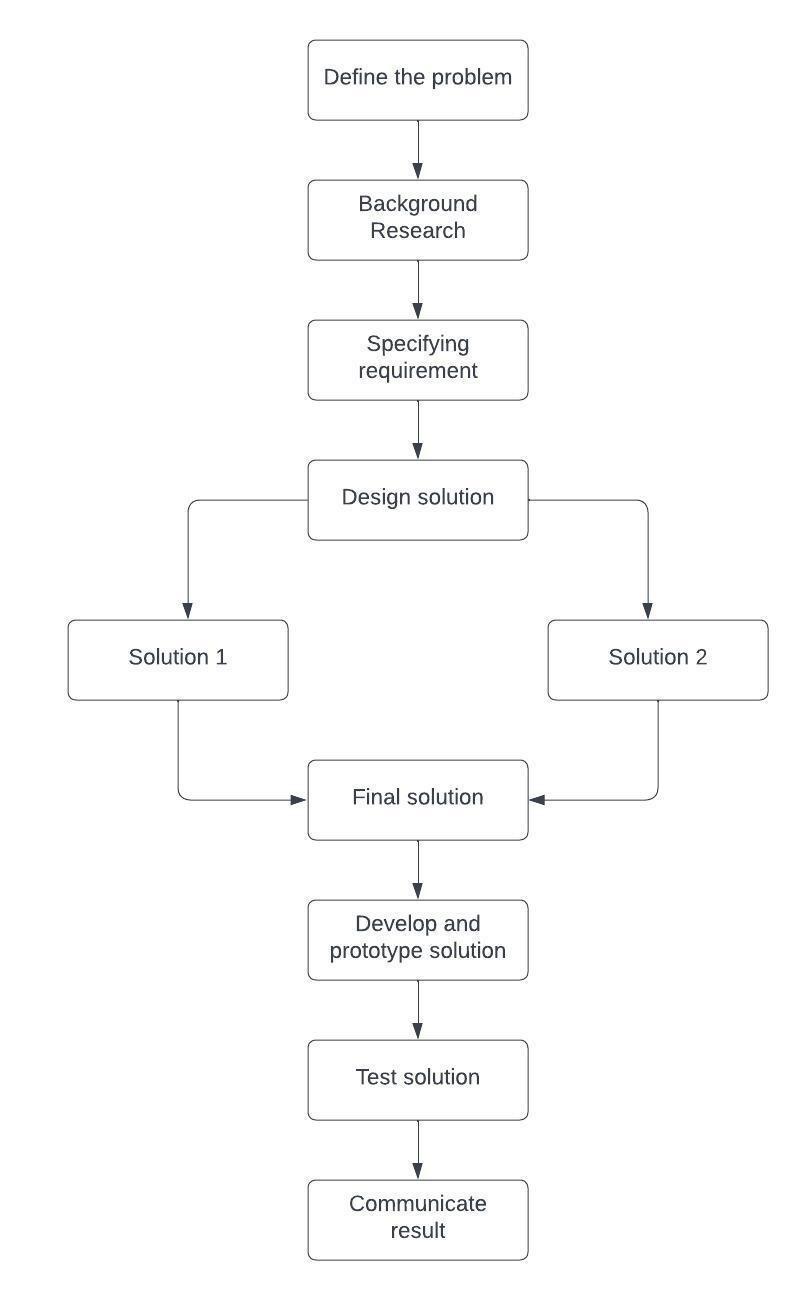
**Specify Requirements:** Establishes clear guidelines for the solution.

**Brainstorm, Evaluate, and Choose Solution:** Generates and selects the best possible approach.

**Develop and Prototype Solution:** Creates a tangible representation of the solution for testing.

**Test Solution:** Ensures the solution meets the specified requirements and performs effectively.

**Communicate Results:** Disseminates information and promotes knowledge sharing.



**Features**

* Comprehensive CRUD operations (create, read, update, delete), including detailed item information (ID, name, description, cost, price, quantity, category, supplier).
* Handles various transaction types such as sales, purchases, returns, and transfers between locations.
* Provides a detailed record of item transactions, including date, quantity, price, and transaction type.
* Handles inventory storage, picking, and packing processes.
* Offers a user-friendly interface with data visualization capabilities for inventory analysis.
* Leverages Spring Boot for robust backend services, including REST APIs, data persistence, and potential integration with a message queue.

**Environmental, Societal, Safety, and Economic Considerations**

* **Environmental Considerations**

**Reduced Paper Consumption:** By digitizing inventory records, the system can significantly reduce paper usage, thereby conserving resources and reducing waste.

**Optimized Inventory Levels:** Effective inventory management can prevent overstocking, which reduces the need for excessive production and transportation, both of which contribute to greenhouse gas emissions.

* **Societal Considerations**

**Improved Supply Chain Efficiency:** Accurate and timely inventory data can optimize supply chain operations, leading to reduced product shortages and improved customer satisfaction.

**Job Creation:** The implementation of the system might require additional personnel for data entry, system administration, or analysis, contributing to job creation.

* **Economic Considerations**

**Return on Investment (ROI):** The system should demonstrate a clear ROI by improving inventory turnover, reducing stockouts, and optimizing purchasing decisions.

**Economic Impact:** The system can contribute to economic growth by enabling businesses to operate more efficiently and competitively.

**Limitation**

* As the system grows, managing and maintaining the complexity of features and integrations can become challenging.
* Reliance on human input for data entry can lead to inaccuracies and affecting inventory levels.
* The system's functionality relies heavily on technology, and disruptions or failures can impact operations.
* Implementing and maintaining the system involves ongoing costs for hardware, software, and personnel.
* Effective training and support are crucial for user acceptance and system utilization.

**Test cases for ItemActionServiceImpl.java**

1. **Boundary Value testing**

Test Case 1: Boundary Value testing for ‘getItemActions’

* Minimum ‘itemId’:

‘itemId = 1’

* Small valid ‘itemId’:

‘itemId = 10’

* Maximum ‘itemId’:

‘itemId = Long.MAX\_VALUE’

Test Case 2: Boundary Testing for ‘saveItemAction’

* Minimum ‘itemId’ and ‘itemActionQuantity’:

‘itemId = 1’, ‘itemActionQuantity = 0’

* Small valid itemId and ‘itemActionQuantity’:

‘itemId = 10’, ‘itemActionQuantity = 5’

* Maximum itemId and ‘itemActionQuantity’:

‘itemId = Long.MAX\_VALUE’, ‘itemActionQuantity= Long.MAX\_VALUE’

Test Case 3: Boundary Testing for ‘getItemAction

* Minimum ‘itemId’ and ‘itemActionId’:

‘itemId = 1’, ‘itemActionId = 1’

* Small valid itemId and ‘itemActionId’:

‘itemId = 10’, ‘itemActionId= 5’

* Maximum itemId and ‘itemActionId’:

‘itemId = Long.MAX\_VALUE’, ‘itemActionId= Long.MAX\_VALUE’

Test Case 4: Boundary Testing for ‘replaceItemAction’

* Minimum ‘itemId’, ‘itemActionId’, and ‘itemActionQuantity’:

‘itemId= 1’, ‘itemActionId= 1’, ‘itemActionQuantity = 0’

* Small valid ‘itemId’, ‘itemActionId’, and ‘itemActionQuantity’:

‘itemId= 10’, ‘itemActionId= 5’, ‘itemActionQuantity= 5’

* Maximum ‘itemId’, ‘itemActionId’, and ‘itemActionQuantity’:

‘itemId= Long.MAX\_VALUE’, ‘itemActionId= Long.MAX\_VALUE’, ‘itemActionQuantity= Long.MAX\_VALUE’

Test Case 5: Boundary Testing for deleteItemAction

* Minimum ‘itemId’ and ‘itemActionId’:

‘itemId = 1’, ‘itemActionId = 1’

* Small valid ‘itemId’ and ‘itemActionId’:

‘itemId = 10’, ‘itemActionId = 5’

* Maximum ‘itemId’ and ‘itemActionId’:

‘itemId = Long.MAX\_VALUE’, ‘itemActionId = Long.MAX\_VALUE’

1. **Equivalence class testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Equivalence Class** | **Use Case** | **Test Case Description** | **Expected Outcome** |
| Item Exists | Valid Item ID | Provide a valid itemId referencing an existing item. | - Method successfully retrieves related item actions. - Returned CollectionModel contains a list of EntityModel objects for each item action. - Self-link (linkTo(methodOn(ItemActionController.class).getItemActions(itemId)).withSelfRel()) included. |
| Item Exists | Non-existent Item ID | Provide a non-existent itemId. | - Method throws a ResourceNotFoundException. |
| Number of Item Actions | Existing Item Actions | Test with a valid itemId that has existing item actions. | - Method retrieves all associated item actions. - Size of the list within the returned CollectionModel matches the number of retrieved item actions. |
| Number of Item Actions | No Item Actions | Test with a valid itemId that has no associated item actions. | - Method retrieves an empty list. - Returned CollectionModel contains an empty list but still includes the self-link. |

1. **Decision table testing**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule n** |
| Item exists | Yes | Yes | No | No |
| Item has item actions | Yes | No | Yes | No |
| itemId format valid | Yes | Yes | Yes | No |
| Database connection | Successful | Successful | Successful | Failed |
| Database response time | Normal | Slow | Normal | Timeout |
| ItemActionRepository behavior | Normal | Throws exception | Normal | Returns unexpected data |
| ItemModelAssembler behavior | Normal | Throws exception | Normal | Returns incorrect data |
| User permissions | Authorized | Unauthorized | Authorized | Authorized |
| System load | Normal | High | Normal | Low |
| Expected Outcome | Returns list of item actions | Returns empty list | Throws ResourceNotFoundException | Throws UnauthorizedException |

1. **State transition testing**

States:

* ItemActive: Item is active.
* ItemInactive: Item is inactive.

Transitions:

* ActivateItem: Transition from ItemInactive to ItemActive.
* DeactivateItem: Transition from ItemActive to ItemInactive.

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Event** | **Next State** | **Actions** |
| ItemActive | GetItemActions | ItemActive | Returns list of item actions |
| ItemInactive | GetItemActions | ItemInactive | Returns empty list |

1. **Use case testing**

Main Scenario:

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | A: Request to retrieve all actions for an item. |
| 2 | S: Retrieve item from the repository. |
| 3 | S: Retrieve item actions for the item. |
| 4 | S: Return the item actions. |

Add item action:

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | A: Request to add a new action to an item. |
| 2 | S: Validate the item ID. |
| 3 | S: Validate the item action details. |
| 4 | S: Save the new item action and update item quantity. |
| 5 | S: Return the saved item action. |

Update item action:

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | A: Request to update an existing action for an item. |
| 2 | S: Validate the item ID and action ID. |
| 3 | S: Validate the new item action details. |
| 4 | S: Update the item action and item quantity. |
| 5 | S: Return the updated item action. |

Delete item action:

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | A: Request to delete an action for an item. |
| 2 | S: Validate the item ID and action ID. |
| 3 | S: Delete the item action and update item quantity. |
| 4 | S: Return confirmation of deletion. |

**Test cases for ItemSummaryServiceImpl.java**

1. **Boundary value testing**

Quantity:

Minimum positive quantity: ‘1’

Maximum positive quantity: Large value close to ‘Long.MAX\_VALUE’

Minimum negative quantity: ‘-1’

Maximum negative quantity: Large value close to ‘Long.MIN\_VALUE’

Zero quantity: ‘0’

Price:

Minimum price: ‘0’

Small positive price: ‘0.01’

Large positive price: Large value close to ‘BigDecimal.MAX\_VALUE’

CreatedDate Range:

Start of day: ‘startDate = LocalDateTime.now().withHour(0).withMinute(0).withSecond(0)’

End of day: ‘endDate = LocalDateTime.now().withHour(23).withMinute(59).withSecond(59)’

1. **Equivalence class testing for getItemSummary() method:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Equivalence Class** | **Description** | **Test Cases** | **Expected Outcome** |
| Item Action Dates | Dates of retrieved Item Actions relative to the current day. | Valid Dates: Item Actions have created dates within the current day (between startDate and endDate).  Pre-dated Actions: Item Actions have created dates before the current day. Future-dated Actions (Optional): Item Actions have created dates after the current day. | Valid Dates: - Calculation includes only Item Actions with valid dates. - dailyProfit, soldCount, and insertedCount reflect these actions. Pre-dated Actions: - Excludes pre-dated Item Actions from calculations. - dailyProfit, soldCount, and insertedCount reflect actions for the current day only.  Future-dated Actions (Optional): - Ignores future-dated actions (optional behavior). - dailyProfit, soldCount, and insertedCount reflect current day actions. |
| Number of Item Actions | Number of Item Actions retrieved from the repository. | Existing Actions: There are some Item Actions retrieved for the current day.  No Actions: There are no Item Actions retrieved for the current day (empty list). | Existing Actions: - Calculation includes all retrieved Item Actions. - dailyProfit, soldCount, and insertedCount reflect calculations based on retrieved actions.  No Actions: - Calculations result in zero dailyProfit, soldCount, and insertedCount. |
| Item Action Quantity | Values of quantity in retrieved Item Actions. | Positive Quantities: Item Actions have positive quantity values (representing insertions).  Negative Quantities: Item Actions have negative quantity values (representing sales). Zero Quantities: Item Actions have zero quantity values. | Positive Quantities: - insertedCount is incremented by the sum of positive quantities. - soldCount remains zero. Negative Quantities: - soldCount is incremented by the absolute value of negative quantities. - insertedCount remains zero.  Zero Quantities: - No change to insertedCount or soldCount. |

1. **Decision tables testing**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
| Item Actions exist | Yes | Yes | No | No |
| Item Action dates (current day) | Valid | Invalid (before) | Valid | Invalid (after) |
| Expected Outcome | Calculates item summary | Excludes pre-dated actions | Calculates empty summary | Ignores future actions (optional) |

1. **State transition testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Event** | **Next State** | **Actions** |
| NormalOperation | Successful Data Processing | NormalOperation | Returns ItemSummary |
| NormalOperation | Data Processing Error | DataProcessingError | Logs error, handles gracefully |
| DataProcessingError | Error Resolved | NormalOperation | Resumes normal operation |

Test Cases:

Test Case 1: Verify that the system returns a correct ItemSummary when data processing is successful.

Test Case 2: Verify that the system handles database connection errors gracefully and logs the error.

Test Case 3: Verify that the system can recover from a data processing error and resume normal operation.

1. **Use case testing.**

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | A: Request to retrieve daily item summary. |
| 2 | S: Determine start and end date for the current day. |
| 3 | S: Retrieve item actions created between the start and end date. |
| 4 | S: Calculate the total daily profit, sold count, and inserted count. |
| 5 | S: Return the item summary. |

**Test cases for ItemServiceImpl.java**

1. **Equivalence class testing**

|  |  |  |
| --- | --- | --- |
| **Equivalence Class** | **Description** | **Test Cases** |
| Get Items | Results: Empty list (no items in the database).  Non-empty list (various numbers of items). | Verify retrieval of an empty list when no items exist. Verify retrieval of a populated list with different numbers of items. |
| Save Item | Item Code: Unique code (not present in the database). Duplicate code (already exists). Empty or null code. Cost and Price: Valid positive values. Negative values. Zero values. Quantity: Valid positive or negative values. Zero value. | Verify successful saving with a unique code. Verify ResourceUniqueViolationException for duplicate code. Verify error handling for empty/null code (potentially using validation annotations). Test saving with valid positive, negative, and zero values for cost, price, and quantity. |
| Get Item, Replace Item | Item ID: Existing item ID. Non-existent item ID. | Verify retrieval of an existing item by ID. Verify ResourceNotFoundException for a non-existent ID.  Test replacement logic with valid data for existing items. |
| Delete Item | Item ID: Existing item ID. Non-existent item ID. | Verify successful deletion with an existing ID. Verify ResourceNotFoundException for a non-existent ID. |

1. **Decision tables testing**

|  |  |  |
| --- | --- | --- |
| **Condition** | **Rule 1** | **Rule 2** |
| Item Code exists | Yes (Unique) | Yes (Duplicate) |
| Expected Outcome | Save item successfully | Throw ResourceUniqueViolationException |

1. **State transition testing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Current State** | **Event** | **Next State** | **Actions** |
| Normal Operation | Successful CRUD operation | Normal Operation | Returns appropriate response (created, updated, deleted, etc.) |
| Normal Operation | Database error | Error State | Logs error, handles gracefully |
| Error State | Error resolved | Normal Operation | Resumes normal operation |

1. **Use case testing.**

Retrieve all item - Main success scenario:

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | A: Request to retrieve all items. |
| 2 | S: Retrieve all items from the repository. |
| 3 | S: Return the collection of items. |

Add new item – Main success scenario:

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | A: Request to add a new item. |
| 2 | S: Validate item details. |
| 3 | S: Save the new item. |
| 4 | S: Return the saved item. |

Retrieve Specific Item – main success scenario:

|  |  |
| --- | --- |
| **Step** | **Description** |
| 1 | A: Request to retrieve an item by ID. |
| 2 | S: Retrieve the item from the repository by ID. |
| 3 | S: Return the item. |