Inventory Management System

Joshua Rozenberg, Craig Sirota, Jonnelin Marzielli Leonardo, Michael Yang

1. System Requirements

Functional Requirements

IN - 001	Users should be able to scan an item and find information about that item, specified by the system administrators.
IN - 002	Users should be allowed to modify the amount of an item in a list.
IN - 003	Users should be allowed to manually add new items.
IN - 004	Users should be allowed to add categories to organize the items by.
IN - 005	Users should be able to filter items to make looking for items easier.

OT - 001	The application shall return all the items data after an item is scanned.
OT - 002	The system shall send inventory reports to all system administrators on a specified schedule.
OT - 003	The application shall prompt the user if the scanned item is not found in the database.
OT - 004	A notification should be sent out to the appropriate individuals regarding a change.
OT - 005	Show items sorted in alphabetical order and based on categories and filter settings.

PT - 001	The application shall allow users to scan existing items and retrieve.
PT - 002	The application shall allow the admin to generate basic statistical reports on the inventory.
PT - 003	The system should be configurable by the admin. (custom categories and fields for different applications)
PT - 004	The application shall allow the admin to generate a sales analysis to predict future inventory requirements.
PT - 005	The application shall allow all users to search, sort, and filter inventory results.
PT - 006	The Application shall allow admins to assign user roles

Non-Functional Requirements

IN - 101	Users should not be allowed to access supply listings that are outside their level of permissions.
IN - 102	Users should not be allowed to modify the quantity of an item without correct permissions.
IN - 103	Items being added should not be in the system already.
IN - 104	Item addition should only be done when all required fields are filled out.
IN - 105	Modifying information in the database should be logged.

OT - 101	Data that is output should always be complete, correct, and consistent with the database.
OT - 102	The system should catch and handle any data errors or SQL exceptions on output.
OT - 103	Document output and user requesting output and store information in a report.
OT - 104	Output data should always be exactly consistent with the data request or an error shall be reported.

PT - 101	Access to the data should be exactly consistent with the defined permissions.
PT - 102	The system shall monitor storage capacity before each save and send alerts if storage is 85% full.
PT - 103	The software shall display the UI responsively between different android devices.
PT - 104	The system shall use multi-threading to separate background tasks and UI tasks.
PT - 105	The system shall not take longer than 10 seconds to retrieve and display proper information when network connection is not an issue.
PT - 106	The system shall store all data in a DBMS on an external server.
PT - 107	Multiple users should be able to access the database at the same time.

2. Test Design

Test Case ID	T01
Purpose	Check that basic scanning and item data retrieval works properly
Pre-conditions	The application has permission to access the camera
Inputs	From Inventory Listing Activity, user clicks scan button User uses mobile device to scan product
Expected Outputs	View Item activity application is displayed or message saying item was not found
Post-conditions	N/A
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T02
Purpose	Check that allowed users can modify item quantity
Pre-conditions	User has either employee or admin permissions The item is already in the database
Inputs	From the View Item activity, user clicks edit button User edits the appropriate field User clicks save button
Expected Outputs	The View Item activity goes back to its default state, out of edit mode
Post-conditions	Database entry of item is updated
Design Technique	Boundary analysis

Test Case ID	T03
Purpose	Ensure that allowed users can manually add new items.
Pre-conditions	User has either employee or admin permissions The Inventory Listing Activity is active Item does not already exist in database
Inputs	The user clicks the add item button The user fills out at least the required information fields The user clicks save button
Expected Outputs	The application goes back to the Inventory Listing Activity with the new item in the list
Post-conditions	Database is updated with new item
Design Technique	Boundary analysis

Test Case ID	T04
Purpose	Check that allowed users can add categories to organize the items by those categories.
Pre-conditions	User has either employee or admin permissions The application has the Category Listing Activity active
Inputs	User clicks new category button User enters the name of the new category User clicks the save button
Expected Outputs	The application goes back to the Category Listing Activity with the updated list
Post-conditions	The categories table of the database is updated
Design Technique	Equivalence class, reviewing requirement

Test Case ID	T05
Purpose	Check item organization and filter operation are accurate.
Pre-conditions	List of items The application has the Inventory Listing Activity active
Inputs	User clicks filter button User selects desired filter(s) from category and/or quantity range dropdown menus User clicks filter
Expected Outputs	Inventory Listings Activity displays desired subset of inventory
Post-conditions	N/A
Design Technique	Equivalence class, reviewing requirement

Test Case ID	T06
Purpose	Ensure data is properly returned after an item is scanned
Pre-conditions	The application has permission to access the camera The application has the Inventory Listing Activity active
Inputs	User clicks scan button User uses mobile device to scan product

Expected Outputs	View Item activity application is displayed or message saying item was not found
Post-conditions	N/A
Design Technique	Reviewing requirement

Test Case ID	T07
Purpose	Ensure the system sends inventory reports to all system administrators on a specified schedule.
Pre-conditions	The system has an interval specified List system administrators
Inputs	Inventory Report
Expected Outputs	Notify system of success or failure to send
Post-conditions	N/A
Design Technique	Reviewing requirement

Test Case ID	T08
Purpose	Ensure the application prompts the user if the scanned item is not found in the database.
Pre-conditions	The code being scanned is not already associated with an item stored in the database The application has permission to access the camera The application has the Inventory Listing Activity active
Inputs	User clicks scan button User uses mobile device to scan product
Expected Outputs	Error message displayed, alerting the user the item was not found
Post-conditions	N/A
Design Technique	Equivalence class, boundary analysis

Test Case ID	T09
Purpose	Ensure a notification is sent out to the appropriate individuals regarding a change.
Pre-conditions	A change has been made to the database

Inputs	None
Expected Outputs	Admin accounts receive a notification
Post-conditions	Change to the database has been implemented
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T10
Purpose	Ensure items are sorted in alphabetical order and based on categories and filter settings.
Pre-conditions	Inventory Listing Activity is open and running
Inputs	User scrolls
Expected Outputs	All items appear in alphabetical order items beginning with a number will appear in the list in numeric order after all alphabetic items are displayed
Post-conditions	N/A
Design Technique	Boundary analysis

Test Case ID	T11
Purpose	Ensure concurrent edits to database are handled in appropriate order
Pre-conditions	The application is installed on multiple devices.
Inputs	Both devices attempt to edit the same data
Expected Outputs	Information displayed to users is the information stored in database
Post-conditions	All devices running the application see the same information from the database
Design Technique	Boundary analysis

Test Case ID	T12
Purpose	Ensure the admin can generate basic statistical reports on the inventory.
Pre-conditions	The user has admin level access.
Inputs	Generate Statistics Report button is clicked
Expected Outputs	Correctly generated report on the inventory
Post-conditions	N/A
Design Technique	Reviewing requirement

Test Case ID	T13
Purpose	Ensure the system is configurable by the admin.
Pre-conditions	The user has admin permissions
Inputs	The admin selects either the edit/delete or add user option from the User List Activity To add/edit: The admin changes/adds desired information The admin clicks the save button To delete: The admin selects the delete option from the edit pop-up menu
Expected Outputs	The admin is returned to the updated user list.
Post-conditions	The user list in the database is updated
Design Technique	Boundary analysis, reviewing requirement

Test Case ID	T14
Purpose	Ensure the admin can generate a sales analysis to predict future inventory requirements.
Pre-conditions	The user has admin permissions
Inputs	The user clicks on the navigation drawer button The user selects the “Reports” option The user selects “Generate Sales Report”
Expected Outputs	The admin is sent a sales report, based on recent sales compared to historic trends
Post-conditions	N/A
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T15
Purpose	Ensure all users can search, sort, and filter inventory results.
Pre-conditions	The device is connected to the database
Inputs	From Inventory Listings Activity, user clicks filter button User selects desired filter(s) from category and/or quantity range dropdown menus User clicks filter

Expected Outputs	Inventory Listings Activity displays desired subset of inventory
Post-conditions	N/A
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T16
Purpose	Ensure admins can assign user roles.
Pre-conditions	The user has admin permissions
Inputs	The admin selects either the edit or add user option from the User List Activity The admin sets the user's role desired information The admin clicks the save button
Expected Outputs	The admin is returned to the updated user list.
Post-conditions	The user list in the database is updated
Design Technique	Boundary analysis, reviewing requirement

Test Case ID	T17
Purpose	Ensure users can not access items in categories that are outside their project scope.
Pre-conditions	Projects A and B, with restricted scopes, are in progress. User is on project A, but not project B.
Inputs	The user opens Inventory Listing Activity
Expected Outputs	The user sees items in the scope of project A, but not items that are only in the scope of project B.
Post-conditions	N/A
Design Technique	Reviewing requirement

Test Case ID	T18
Purpose	Ensure users can not modify the quantity of an item without correct permissions.
Pre-conditions	The user has customer permissions
Inputs	The user selects the item edit button

Expected Outputs	An error message saying the user does not have the correct permission is displayed.
Post-conditions	No change is made to database
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T19
Purpose	Ensure duplicate items cannot be created.
Pre-conditions	An item with name N or code C already exists in the database The user is on the Add/Edd Item Activity
Inputs	The user enters information for an item including at least name N or code C. The user clicks the save button.
Expected Outputs	The system displays a message over the Add/Edit Item Activity, alerting the user that the name and/or code is already taken
Post-conditions	No change is made to the database
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T20
Purpose	Ensure an item cannot be added without all required fields.
Pre-conditions	The user is on the Add/Edd Item Activity The name, quantity, and/or code are blank
Inputs	The user clicks the save button.
Expected Outputs	The system displays a message over the Add/Edit Item Activity, alerting the user that the item could not be added because critical information was missing
Post-conditions	No change is made to the database
Design Technique	Boundary analysis, reviewing requirement

Test Case ID	T21
Purpose	Ensure any database modifications are logged
Pre-conditions	A change is made to the database
Inputs	None

Expected Outputs	System records database modification
Post-conditions	A user who did not make the change, on a device other than the one the change was made on, can see the update
Design Technique	Boundary analysis, reviewing requirement

Test Case ID	T22
Purpose	Ensure output data is complete, correct, and consistent
Pre-conditions	Non-empty list of items
Inputs	Item Query
Expected Outputs	Data displayed in application matches data in database
Post-conditions	N/A
Design Technique	Reviewing requirement

Test Case ID	T23
Purpose	Ensure SQL exceptions report an error in the application
Pre-conditions	Specified data is not in the database
Inputs	Invalid SQL query
Expected Outputs	System admin is notified and exception is entered into the system log
Post-conditions	N/A
Design Technique	Boundary analysis, reviewing requirement

Test Case ID	T24
Purpose	Ensure the data access log is working properly.
Pre-conditions	A series of events or changes in the system
Inputs	A change is made to the database
Expected Outputs	Changes to database should appear in application
Post-conditions	Log reflects time of change, users involved in change, and content of change
Design Technique	Reviewing requirement

Test Case ID	T25
Purpose	Ensure errors are reported if the output data is not consistent with the data request.

Pre-conditions	The output data is not consistent with the data request
Inputs	The user requests data from the database
Expected Outputs	Error notification which describes the inconsistency.
Post-conditions	Inconsistency is added to the system log
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T26
Purpose	Ensure permissions to the data are working properly
Pre-conditions	Users have different permission levels
Inputs	Users try to access similar data
Expected Outputs	Data if permission level is met, otherwise error message saying lack of permission
Post-conditions	N/A
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T27
Purpose	Ensure the system monitors storage capacity before each save and sends alerts if storage is 85% full.
Pre-conditions	The system is set below 85% full
Inputs	Items generated until the the 85% capacity threshold is passed
Expected Outputs	An alert sent to all admins that the system is nearing its maximum capacity.
Post-conditions	No change is made
Design Technique	Reviewing requirement

Test Case ID	T28
Purpose	Ensure the software displays the UI responsively between different android devices.
Pre-conditions	The devices are running up-to-date versions of the Android OS
Inputs	User clicks through the UI screens
Expected Outputs	No UI should look distorted, misaligned, covered up, or go off screen
Post-conditions	No change is made
Design Technique	Boundary analysis, reviewing requirement

Test Case ID	T29
Purpose	Ensure separation between UI and background tasks
Pre-conditions	A non-empty list of items
Inputs	Report generation request
Expected Outputs	Application performance
Post-conditions	N/A
Design Technique	Equivalence class, boundary analysis, reviewing requirement

Test Case ID	T30
Purpose	Ensure the system does not take longer than 10 seconds to retrieve and display proper information.
Pre-conditions	network connection is not an issue.
Inputs	A request for data from the database has been made
Expected Outputs	Expected data is displayed
Post-conditions	N/A
Design Technique	Boundary analysis, reviewing requirement

Test Case ID	T31
Purpose	Ensure we can write data to the DBMS on the remote server
Pre-conditions	The user has permission to edit the database
Inputs	A change is made to the database
Expected Outputs	The change is displayed in the application
Post-conditions	The system log reflects the change
Design Technique	Boundary analysis, reviewing requirement

JUnit Tests

Test Case ID(s)	T02, T03, T12, T13, T14, T16, T17, T18, T26
Purpos	Check if user has the expected permissions

e	

JUnit Method	
	```
	@Test
	void checkUserPermission(User user, int[] expectedPermissionLevels) {
	    for (int level : expectedPermissionLevels) {
	        assertEquals(user.permission_level, expectedPermissionLevel);
	    }
	}
	```

Test Case ID(s)	T02
Purpose	Check that the item exists in the database

JUnit Method	
	```
	@Test
	void checkItemExistsDB(Item targetItem) {
	    Item[] items = /* Get all items from database */
	    boolean itemExists = false;
	    for (Item item : items) {
	        if (item.product_id == targetItem.product_id) {
	            itemExists = true;
	        }
	    }
	    assertTrue(itemExists);
	}
	```

Test Case ID(s)	T03, T08, T19
Purpose	Check that the item does not exist in the database

JUnit Method	
	```
	@Test
	void checkItemExistsDB(Item targetItem) {
	    Item[] items = /* Get all items from database */
	    boolean itemExists = false;
	    for (Item item : items) {
	        if (item.product_id == targetItem.product_id) {
	            itemExists = true;
	        }
	    }
	```

	assertFalse(itemExists);
	}

Test Case ID(s)	T10, T15
Purpose	Check items are in alphabetical order by comparing it to a list that is already alphabetical

JUnit Method	@Test
	void checkItemsAlphabetical(Item[] listToTest, Item[] listSortedAlphabetically) {
	    for (int i = 0; i < listSortedAlphabetically.length(); i++) {
	        assertEquals(listSortedAlphabetically[i].name, listToTest[i].name);
	    }
	}

Test Case ID(s)	T10, T15
Purpose	Check all items have the expected category

JUnit Method	@Test
	void allItemsSameExpectedCategory(Item[] listItems, Category expectedCategory)
	{
	    for (Item item : listItems) {
	        for (Category itemCategory : item.category_list) {
	            assertEquals(itemCategory.name, expectedCategory.name);
	        }
	    }
	}

Test Case ID(s)	T10, T15
Purpose	Check item quantity is within a certain range
JUnit	@Test

Method	void itemQuantityWithinRange(Item item, int lowerBoundRange, int higherBoundRange) {
	    boolean withinRange = false;
	    if (item.quantity >= lowerBoundRange && item.quantity <= higherBoundRange) {
	        withinRange = true;
	    }
	    assertTrue(withinRange);
	}

Test Case ID(s)	T13
Purpose	Check that a user exists in the database

JUnit Method	@Test
	void userExistDB(User targetUser) {
	    User[] listUsers = /* Get list of users from DB ... */
	    boolean userExists = false;
	    for (User user : listUsers) {
	        if (targetUser.ID_number == user.ID_number) {
	            userExists = true;
	        }
	    }
	    assertTrue(userExists);
	}

Test Case ID(s)	T13
Purpose	Check that a user does not exist in the database

JUnit Method	@Test
	void userExistDB(User targetUser) {
	    User[] listUsers = /* Get list of users from DB ... */
	    boolean userExists = false;
	    for (User user : listUsers) {
	        if (targetUser.ID_number == user.ID_number) {
	            userExists = true;
	        }

	}
	   assertFalse(userExists);
	}

Test Case ID(s)	T16
Purpose	Check the user fields have the expected values (for checking if the user saved correctly after edit)

JUnit Method	@Test
	void checkUserFieldsExpected(User user, String expectedName, int expectedPermissionLevel) {
	   assertTrue(user.name, expectedName);
	   assertTrue(user.permssion_level, expectedPermissionLevel);
	}

Test Case ID(s)	T20
Purpose	Check all required fields in item is filled

JUnit Method	@Test
	void checkAllItemFieldsFilled(String itemName, int itemQuantity, QRCode itemCode) {
	   assertNotNull(itemName);
	   assertNotNull(itemQuantity);
	   assertNotNull(itemCode);
	   assertNotEquals(itemName, "");
	}

Test Case ID(s)	T22
Purpose	Check that the item list is not empty
JUnit	@Test

Method	void itemListNotEmpty(Item[] listItems) {
	    assertNotEquals(listItems.length() == 0);
	}

Test Case ID(s)	T30
Purpose	Make sure that the intended method/program/execution does not take more than the specified maxSeconds

JUnit Method	@Test
	void checkExecutionBelowMaxTime(int maxSeconds) {
	    boolean belowMaxTime = false;
	    long startTime = System.nanoTime();
	    /* ... Run program/part of program to time ... */
	    /* ... End running program ... */
	    long endTime = System.nanoTime();
	    long duration = endTime - startTime;
	    int durationSeconds = (int) duration / 1000000000;
	    if (durationSeconds < maxSeconds) {
	        belowMaxTime = true;
	    }
	    assertTrue(belowMaxTime);
	}

3. Traceability

Test Case Number	List of the Requirements tested
T01	IN-001/PT-001
T02	IN-002
T03	IN-003
T04	IN-004
T05	IN-005
T06	OT-001
T07	OT-002
T08	OT-003
T09	OT-004
T10	OT-005
T11	PT-107

T12	PT-002
T13	PT-003
T14	PT-004
T15	PT-005
T16	PT-006
T17	IN-101
T18	IN-102
T19	IN-103
T20	IN-104
T21	IN-105
T22	OT-101
T23	OT-102
T24	OT-103
T25	OT-104
T26	PT-101
T27	PT-102
T28	PT-103
T29	PT-104
T30	PT-105
T31	PT-106