TEST SPECIFICATION

G2J: Gloria Nzoh-Ndem, Jamison Brennan, John Kubas

1.0 Introduction

This test plan covers test procedures, testing strategy, environment, schedule, and specific test cases to validate the G2J Software system's behavior. This project is a cash register and inventory management system designed for grocery stores. In this project, the system processes real-time customer purchases through barcode scans, updates the inventory levels, and automates reordering to maintain stock based on demand. The test process makes sures that the software meets the functional, performance, and reliability requirements while addressing the risks that were previously identified in the project plan.

1.1 Goals and objectives

The primary goals of the test process are:

- Functional Correctness: Ensures that the software accurately processes the transactions, updates the inventory, and generates the orders as it was defined in the project scope.
- Performance and Reliability: Ensures that the system handles concurrent transactions (transactions happening at the same time) from multiple cash registers during peak hours with minimal latency and no data loss.
- Scalability: Confirms that the system can initially manage a small sample of products and is extensible for larger datasets.
- Risk Mitigation: Addresses the risks such as software underperformance, storage underestimation, and data integrity issues through rigorous testing.
- User Satisfaction: Ensures that the user interface is intuitive and aligns with the stakeholder's requirements, as guided by input from the user representative.

1.2 Statement of scope

The testing scope covers the essential functions, features, and behaviors of the G2J software.

In Scope:

Transaction Processing: This includes barcode scanning, sales calculations, and inventory updates.

Inventory Management: This involves monitoring stock levels (CurrentStockUnits, CurrentStockCases) and updating the inventory based on sales or restocking events.

Order Generation: This automatically generates the orders when inventory drops below the ReorderThreshold.

User Interface: This includes the data entry for products, inventory display, pending orders, and report generation.

Data Persistence: This makes sures that the data is stored and retrieved across sessions using a relational database.

Concurrency: This supports the handling of simultaneous transactions (transactions happening at the same time) from multiple registers.

Out of Scope:

External Supplier Integration: Integration with external supplier systems is not included due to budget constraints; manual logging of external order fulfillment is expected.

Full Product Catalog: Testing will be limited to a small sample of products as per resource limitations.

Hardware Compatibility: Barcode scanners are assumed to deliver standard input; specific hardware testing is excluded from this scope.

2.0 Test Plan

This section describes the overall testing strategy and the project management issues that are required to properly execute effective tests.

2.1 Software to be tested

Software Name: G2J Cash Register and Inventory Management System.

Components:

- User Interface (GUI) for input and output.

- Inventory Processor for transaction and reorder logic.
- Data Manager for database interactions.

Exclusions:

- Supplier ordering system (future feature, not implemented in initial scope).
- Non-core features like advanced analytics or third-party integrations.

2.3 Testing tools and environment

Test Environment:

- Development Workstation: A MacBook Pro (e.g., 16GB RAM, 2.6 GHz multi-core CPU, macOS) used for code creation, initial testing, and development of the Python-based application and MySQL database interactions. This workstation serves as the primary development and unit testing environment.
- Client Terminals: Multiple personal computers (PCs) simulating cash registers, each with a minimum configuration of 8GB RAM, 2.0 GHz CPU, and running Windows, Linux, or macOS. These imitate the cash register hardware described in the external machine interface.
- Database Server: A single server with 16GB RAM and a 4-core CPU to host the MySQL database, ensuring robust data persistence and concurrent access.

Tools:

Unit Testing: unittest (Python's built-in testing framework) is used to test individual components, such as UPC processing logic and inventory updates, executable on the MacBook Pro.

- Purpose: Validates backend functions like barcode parsing and stock calculations.

Integration Testing: pytest is used with plugins for testing interactions between the GUI, Inventory Processor, and Data Manager, runnable on the MacBook Pro and client terminals.

- Purpose: Ensures seamless integration of backend logic with the database and GUI (Weeks 9–12).

Performance Testing: Apache JMeter is used to simulate concurrent transactions from multiple cash registers, testing the system's ability to process barcode files within 1 second for up to 5-10 entries.

- Purpose: Verifies performance under high load, addressing the risk of size underestimation.

Database Testing: MySQL Workbench and custom SQL scripts are used which makes it accessible from the MacBook Pro, to validate data integrity, persistence, and transactional updates.

- Purpose: Ensures accurate storage and retrieval of inventory data, mitigating storage underperformance risks.

GUI Testing: PyQtTest or Selenium (if a web-based GUI is implemented) is used to automate user interface interactions, such as product entry and report generation, executable on the MacBook Pro.

- Purpose: Validates usability and responsiveness for stakeholders like the Inventory Control Specialist (ICS) and Store Manager.

Simulators: These are used to custom Python scripts, and they are developed and executed on the MacBook Pro, to emulate barcode scanner inputs, generating text files with UPC codes to mimic cash register outputs.

- Purpose: Simulates hardware inputs for testing without physical scanners, given the project's constraints.

Test Files: These are in the format of CSV files containing sample product data (e.g., ProductID, UnitsPerCase, ReorderThreshold) and transaction logs, created and managed on the MacBook Pro.

- Purpose: Provides controlled inputs for testing barcode processing and inventory updates.

Resources:

- Security Testing: Tools like sqlmap, run from the MacBook Pro, to test the MySQL database connections for vulnerabilities, ensuring compliance with SSL-based secure connections.
- Documentation: Test case templates and logs stored in a shared repository for team collaboration, accessible from the MacBook Pro.
- Error Logging: Python's logging module to capture and analyze errors during testing, integrated into the codebase on the MacBook Pro, supporting reliability requirements.

2.4 Test schedule

A detailed schedule for testing is described.

Weeks 1–6: Requirement Analysis

Objectives/ Our goal for the project:

- Gather and analyze stakeholder requirements.
- Identify critical functionalities and system constraints.
- Define success criteria for each module.

Testing Activities (Things we did):

- Validate requirements through stakeholder review sessions.
- Develop preliminary test cases and acceptance criteria.
- Identify dependencies and create the initial test plan.

Weeks 7–8: UPC Processing Logic (Development and Testing)

Development Focus:

- Implement barcode scanning, sales calculation, and inventory update logic.

Testing Activities (Things we did):

- Unit test UPC input parsing, validation, and error handling.
- Verify accurate sales calculations and inventory updates.
- Test edge cases (e.g., invalid UPCs, duplicate entries).

Weeks 8–9: Database Schema Design and Creation

Development Focus:

- Design and create tables for products, sales, inventory, and orders.

Testing Activities (Things we did):

- Validate schema design against functional requirements.
- Perform unit tests for table constraints, relationships, and queries.
- Load sample data and verify data consistency and normalization.

Weeks 9–10: Backend Integration (UPC Logic + Database)

Development Focus:

- Connect UPC logic to the database to support real-time data storage and retrieval.

Testing Activities (Things we did):

- Conduct integration tests to ensure seamless interaction between logic and database.
- Validate transaction processing: item scan ➔ sale ➔ inventory update ➔ order generation.
- Test for data persistence and rollback on transaction failure.

Weeks 10–11: GUI Development and Testing

Development Focus:

- Build interfaces for entering product details, viewing inventory, generating reports, and reviewing pending orders.

Testing Activities (Things we did):

- Perform usability testing for intuitiveness and accessibility.
- Validate GUI functionality against backend responses.
- Conduct cross-platform/browser testing if applicable.

Weeks 11–12: Full System Integration (Backend + GUI)

Development Focus:

- Merge GUI with backend to form the complete application.

Testing Activities (Things we did):

- Perform end-to-end testing (scan ➔ database update ➔ visual confirmation in GUI).
- Execute concurrency tests with multiple simultaneous transactions.
- Conducted regression testing.
- Final user acceptance testing (UAT) based on stakeholder scenarios.

3.0 Test Cases

This section enumerates a complete list of test cases for the software. A template for test cases is as follows.

1. Successful query of an existing item by name

ID	**1**
Test Input	Scan barcode "123456789012" (CurrentStockUnits = 50)
Expected Output	CurrentStockUnits = 49, transaction recorded, UI updates quantity
Description	Tests Cashier barcode scan and inventory update.

2. Successful query of an item by name, using a non-unique term and updating the item successfully

ID	**2**
Test Input	Select "Eggs (a dozen)" (ProductID ="345678901234", Current Quantity = 20), update Current Quantity to 30 on configuration screen, click "Save Changes"
Expected Output	Current Quantity updates to 30, "Product updated successfully" message appears, dashboard "Recent Products" shows 30
Description	Tests updating product quantity via the configuration screen

3. Searching an invalid item by UPC, handled properly

ID	**3**
Test Input	Enter UPC= "1234567890123" in the search bar on the dashboard, click "Search"
Expected Output	"No products found matching the term" message appears, "Recent Products" list remains unchanged
Description	Tests searching for a non-existent product by UPC and verifies error handling

4. Searching for a valid item by UPC

ID	**4**
Test Input	From dashboard, select "Milk (1 gallon)" (UPC ="123456789012") from "Recent Products" list
Expected Output	Product configuration screen loads with ProductID = 1, UPC= "123456789012", Product Name="Milk (1 gallon)", Description "Whole milk, 1 gallon", Category "Dairy", Current Quantity is 50, Case Size 12, Unit Price $3.99
Description	Tests navigation to product configuration screen and verifies product details

5. Adding an item with invalid fields, handled properly

ID	**5**
Test Input	Click "Enter New Item", input UPC "" (empty), Product Name ="Pepsi 20oz bottle", Description "single bottle for cold case by register", Category "Soda", Current Quantity is 24, Case Size 24, Unit Price $2.29, click "Save"
Expected Output	UPC field highlights with "This is an invalid field", clicking "Save" shows error: "Data row too long for column 'UPC' at row 1", product not added, "Recent Products" list unchanged
Description	Tests error handling for invalid UPC when adding a new product

6. Adding an item with valid fields

ID	**6**
Test Input	Click "Enter New Item", input UPC "41525", Product Name "Pepsi 20oz bottle", Description "single bottle for cold case by register", Category "Soda", Current Quantity 24, Case Size 24, Unit Price $2.29, click "Save"
Expected Output	"New product added successfully" message appears, dashboard "Recent Products" list updates with UPC "41525", Product Name "Pepsi 20oz bottle", Quantity 24
Description	Tests adding a new product via the "Enter New Item" screen