BigCommerce Plugin for ChessWorld SKU Management

Quality Assurance Plan

Version 4.0

Written by:

Rui En Koe

Alex Chan

Ashley Warden

Liangdi Wang

Luqmaan Yurzaa

Wen Cheng Huong

MONASH University × UpScale × AUSTRALIA CHESS WORLD THE ART OF CHESS

Document History and Version Control

Date (dd/mm/yy)	Version	Author	Description
24/03/2025	1.0	Wen Cheng Huong	Initial Quality Assurance Plan draft.
17/04/2025	2.0	Wen Cheng Huong	Carried forward previous Quality Assurance Plan.
18/08/2025	3.0	Wen Cheng Huong	Update QAP: 1. Updated document to include sections on code review and product testing.
27/10/2025	4.0	Wen Cheng Huong	Update QAP: 1. Removed edge and test cases section 2. Removed automated load testing section

Contents

1. Project Overview **4**
1.1 Project Description 4
1.2 Problem Statement 4
1.3 Project High Level Objectives 4
1.4 Project Vision 5
2. Quality Assurance Overview **6**
2.1 Who will be responsible for this process 6
2.2 Extent of software testing 6
2.3 Definition of Done 6
2.4 Code artefacts 6
2.5 CI/CD software and tools 6
3. Github Repository and Branching Model **7**
3.1 Commits 7
3.2 Branch 7
3.3 Merge Requests 8
4. Non Functional Requirements **9**
4.1 Maintainability 9
4.2 Performance 9
4.3 Scalability 9
4.4 Security 9
4.5 Compatibility 9
4.6 Usability 9
5. Git Rules to Follow **10**
6. Product Testing **11**
6.1 Automated Unit Testing 11
6.2 Automated End-to-end Testing (E2E) 12
6.3 Automated Load Testing 13
6.4 Test Cases and Edge Cases 14

1. Project Overview

1.1 Project Description

Chessworld, a premium online retailer specializing in chess-related products, requires a robust inventory management solution for its **BigCommerce** store. The project aims to develop a **custom plugin** that enables accurate **SKU-level inventory tracking** while supporting product bundling. The plugin will ensure that bundled products—composed of multiple SKUs—are correctly tracked to prevent overselling. By seamlessly integrating with BigCommerce's existing product and order management system, the solution will provide an intuitive interface for managing bundled SKUs efficiently. This will enhance Chessworld's ability to manage inventory in real-time, ensuring a seamless shopping experience for customers while maintaining operational efficiency.

1.2 Problem Statement

Chessworld sells chess sets that often consist of multiple SKUs from different manufacturers, such as chess boards and chess pieces, which can be sold separately or as part of a bundle. However, **BigCommerce lacks native support for SKU-level inventory tracking within bundled products**, leading to **overselling risks** when a shared SKU runs out of stock. The current system does not synchronize inventory across bundles and individual SKUs, creating potential stock discrepancies. This gap in inventory management affects order fulfillment accuracy and operational efficiency. Chessworld needs a solution that allows **independent SKU tracking within bundles**, ensuring inventory accuracy and preventing overselling while maintaining seamless integration with BigCommerce.

1.3 Project High Level Objectives

- Develop a BigCommerce plugin to manage bundled products with accurate SKU tracking.
- Enable seamless SKU synchronization across individual products and bundles.
- Prevent overselling when a shared SKU is out of stock.
- Provide a simple and intuitive user interface for inventory management.

1.4 Project Vision

Our plugin is designed to support Chess World by enabling independent SKU tracking within bundled products. With real-time SKU-level tracking and automatic stock synchronization between individual SKUs and bundles, it removes the need for manual updates and significantly reduces the risk of overselling. Unlike Smart Manager, our plugin offers automated, SKU-level inventory tracking tailored specifically for bundled products.

Seamlessly integrated with BigCommerce, the plugin improves order fulfillment accuracy and streamlines operations, directly addressing Chess World's current challenges in managing SKU-level inventory.

In addition, the plugin features a clean, spreadsheet-style interface that makes inventory management intuitive and efficient. Its simple, minimalist design delivers an all-in-one solution that prioritizes ease of use and operational excellence for Chess World.

2. Quality Assurance Overview

2.1 Who will be responsible for this process

Quality assurance will be the responsibility for every member of the Scrum team. The lead tester will be responsible for ensuring the team meets all the standards detailed in this document. All team members are responsible for facilitating the enforcement of all the standards detailed in this document for the project. Any updates or changes to the quality assurance plan will need to be further discussed with the team and reflected in the updated plan every milestone.

2.2 Extent of software testing

The primary type of testing that will be used throughout the project is unit testing. This is due to their ease of implementation and execution. Unit testing will be implemented using CI/CD tools so that it will be executed continually and ensure that new features implemented in the system do not deteriorate the application's current functionality.

2.3 Definition of Done

1. Code passes all automated unit tests.
2. Code passes code review.
3. Integration testing (if required).
4. Non-functional requirements are met.
5. Product Manager assesses the feature to ensure it meets the user story.

2.4 Code artefacts

Code artefacts are deliverables for architecture, testing pipelines and code would be an entire team effort, given that it ranges from all deliverables.

2.5 CI/CD software and tools

GitHub CI/CD tools will be used throughout the development of the application as it will be developed using GitHub as a Git repository. CI/CD pipelines will be configured and deployed on every commit so that every addition of code is tested, and checked to ensure that proper code standards and conventions defined within the project will be maintained.

Within these pipelines, code building, code testing with chosen testing frameworks, and code deploying to the main branch will occur to effectively manage continuous integration of new features and code from all members of the team.

3. Github Repository and Branching Model

3.1 Commits

Our team should follow commit guidelines aimed at maintaining consistency, clarity, and organisation within our codebase. We encourage frequent commits to promote regular integration of code changes and minimise merge conflicts. Commit messages should provide concise yet descriptive summaries of the changes introduced, focusing on the "what" and "why" rather than the "how."

For every commit and push, check that unnecessary files are not being pushed by making sure that they are included in the '.gitignore' file. Attempt to keep commits atomic, involving one task or one fix whenever possible to allow for quicker debugging and reverting if any mistakes occur.

3.2 Branch

The naming of branches should follow a strict convention throughout the entire development. Branch names should begin with a category; **feature** (adding or removing a feature), **bugfix** (fixing a bug), **hotfix** (temporary code change/solution) or **test** (experimenting). After the category, there should be a '/', followed by the reference of the issue or 'no-ref' if there is no reference. The next part of the branch name should be the description of the branch with '-' for whitespace. An example of a branch name could be: *feature/no-ref/new-menu*. Each branch should use tags to denote priority of the user story or feature the branch is related to.

Development will be done using branches so that members can make changes without affecting the main code line. The main branch will be protected by ensuring that merge requests to the main branch require approval and reviewing from at least one other member.

When new features or changes are made to the main branch and are relevant to the feature/development branch a team member is working on, they should rebase their feature branch into main so the feature branch begins on the tip of the main branch with its new changes (both branches will still exist and main branch will not be affected). However, it is **crucial** that team members **do not** rebase the main branch onto their feature branch as this can cause git workflow issues.

3.3 Merge Requests

When merging to the main branch, the team member will need to request a merge which will be reviewed by other team members. The merge request should be clear and illustrate the changes made to the repository, as well as any notes for any additional information/possible issues that they may have.

When merge conflicts appear, the team member should notify the other team members of the issue, and devise a solution to resolve the conflict.

Labels and tags can be used to help section out which part of the application was changed, and if it may also affect other parts of code from other team members. This can help identify the changes to sections of the application in case team members would like to go back to see the different changes. Tags can also be used to highlight milestones within the development, associating them with issues or pull requests, and helping to track the progress of features or fixes.

After everything has been checked and we have a certain number of members thoroughly review the change, will it then be merged. When the merge request has been approved, the source branch should be deleted.

4. Non Functional Requirements

4.1 Maintainability

The plugin will be developed with maintainability in mind, including clear, modular code and comprehensive documentation. This documentation will cover system architecture, API usage, setup instructions, and guidelines for adding new features. Code will follow consistent standards and include inline comments to aid future developers in understanding and updating the system.

4.2 Performance

The plugin must be optimized to ensure minimal impact on store performance, including page load times and overall responsiveness. It should execute background operations asynchronously where applicable and leverage efficient data handling practices to avoid blocking the main thread.

4.3 Scalability

The plugin will be designed to scale seamlessly with the growth of the store's catalog. It must support thousands of products and SKUs without performance degradation or instability. This will be achieved through efficient querying, pagination, and indexing strategies. Stress tests will be conducted to validate consistent performance across varying data volumes.

4.4 Security

Security is a core priority for the plugin. All API communication will be secured using industry-standard encryption protocols (e.g., HTTPS, OAuth 2.0). Sensitive data will be handled in compliance with best practices, including data minimization and secure storage (where applicable). Regular security reviews and vulnerability scans will be performed to identify and address potential risks.

4.5 Compatibility

To ensure smooth integration, the plugin will maintain compatibility with the current BigCommerce APIs and storefront features. It will be designed to coexist with other commonly used plugins without conflict. Compatibility testing will be part of the QA process to identify and resolve any integration issues early in the development lifecycle.

4.6 Usability

The plugin will provide a clean, intuitive, and responsive user interface that supports streamlined inventory management. UI/UX best practices will be followed to ensure that users can easily access features and perform tasks with minimal training. The interface will be tested across a variety of devices and screen sizes to ensure a consistent experience.

5. Git Rules to Follow

Code	Subject	Description	If violated
QA1	Merge Requests	Delete the source branch after merging.	Contact the team member to delete the source branch after merging.
QA2	Merge Conflicts	Team members should notify their team of the issue to devise a solution.	Other team members should train the team member on how to manage merge conflicts appropriately to avoid future problems.
QA3	Main Branch	Commits to the main branch must be approved by at least one other member.	The team member committing into the main branch must be reminded to not make commits to the main branch.
QA4	Naming Conventions	Following naming conventions for branches e.g. *feature/no-ref/new-menu*.	Team members must be reminded to rename the branch using the correct naming conventions.
QA5	Branch Development	Software development will be done using branches to protect the main code line/branch	Any changes made directly to the main branch without review will be reverted.
QA6	Description	Descriptions must be concise but descriptive summaries of the commit.	Team members are encouraged to point out any descriptions that are too long or not descriptive enough, asking the commit owner to update their commit description.
QA7	Commit Contents	Keeping commits atomic and only involving one task or fix when possible.	Reminders to commit owners will be made.
QA8	Git Ignore	Commits must use .gitignore to avoid committing unnecessary files to the repository.	Team members are encouraged to point out this violation. The commit will be reverted, and a .gitignore must be implemented.
QA9	Rebasing	Team members should not rebase the main branch onto their feature branch as this can cause git workflow issues for that team member.	The team member should run ‘git reflog’ to see when the rebase happened in history (get the reference e.g. ‘HEAD@2), then use ‘git reset {reference}’ to revert the rebase.

6. Product Testing

6.1 Automated Unit Testing

Jest will be used as the testing framework for automated unit testing of the plugin. Jest is a widely adopted JavaScript testing library that provides fast, reliable, and maintainable test execution.

Objectives

- Validate individual functions, modules, and components of the plugin in isolation.
- Ensure that logic within the plugin behaves as expected under various inputs and edge cases.
- Detect regressions early during development by running tests automatically on code changes.
- Maintain confidence in code quality as new features are added.

Scope

Jest will be used to test:

- **Core plugin functions** (e.g., pricing calculations, validation logic).
- **APIs and service calls** (mocked to ensure isolated testing).
- **Error handling** for invalid inputs or unexpected scenarios.
- **Utility/helper functions** that support the plugin.

Approach

- **Isolated tests**: Each function or module is tested independently, without reliance on external systems.
- **Mocking**: External dependencies (e.g., APIs, databases) are mocked to ensure predictable, reliable tests.
- **Assertions**: Jest's built-in matchers will be used to confirm expected outputs and behavior.
- **Automation**: Tests are run automatically during development, and integrated into CI/CD pipelines.

Benefits

- Fast and lightweight testing, suitable for continuous feedback.
- Easy-to-read syntax and rich debugging output.
- Code coverage tracking to measure testing completeness.
- Early defect detection before integration or E2E testing.

6.2 Automated End-to-end Testing (E2E)

Playwright will be used as the primary tool for automated end-to-end (E2E) testing. Playwright is a modern framework developed by Microsoft that enables testing across multiple browsers (Chromium, Firefox, WebKit) and devices in a reliable and scalable way.

Objectives

- Validate that the product functions correctly from the perspective of an end user.
- Ensure that core business workflows (e.g., bundle creation in BigCommerce) work as expected.
- Catch regressions early during development and deployment cycles.
- Maintain a consistent, repeatable testing process that reduces reliance on manual testing.

Scope

Playwright will be used to test key workflows, including:

- **Authentication flows**: login with credentials and handling two-factor authentication.
- **Bundle creation**: simulating a merchant creating, updating, and deleting product bundles.
- **Storefront validation**: ensuring that bundles appear correctly to customers.
- **Checkout process**: verifying that a customer can add a bundle to cart and successfully complete a purchase.

Approach

- **Automated browser interactions**: Playwright scripts will simulate user actions such as clicking, typing, and navigating through the store.
- **Assertions and validations**: Tests will confirm the presence of UI elements, correct calculations, and expected user feedback.
- **State cleanup**: Each test will include teardown steps (e.g., deleting created bundles) to avoid polluting the environment.
- **Test data management**: Sandbox store and test credentials will be used to ensure no production data is impacted.

Benefits

- Cross-browser coverage (Chrome, Edge, Safari, Firefox).
- CI/CD integration for automated runs on commits and deployments.
- Consistent reproducibility for both developers and QA testers.

Limitations

- Must be run in headed mode due to limitations with the BigCommerce platform's Two-Factor-Authentication (2FA).