e-store

Software Quality Assurance Plan

Version 0.4.0, 11 March 2016

Project Team
Ebarle, Roselle M.
Maglasang, Catherine M.
Yee, Mar Rynner
Esin, Dexter D.

Project Manager
Ebarle, Roselle M.

Senior Manager
Maglasang, Catherine M.

Advisor
Prof. Orven Llantos Ebarle

April 5, 2016

Abstract

This document is about the Software Quality Assurance Plan ($SQAP$) of the system, e-store which is an e-commerce platform for online merchants. It provides the power to grow your web business, reach more customers and sell more products and services. It enables businesses to experience an integrated workflow for their business: Sales, Inventory and Order Management and Customer Service under one platform.

Contents

Document Status Sheet

Document Title	**Software Quality Assurance Plan**
Document Identification	e-store/Documents/Management/SQAP/0.5.0
Author(s)	Ebarle, Maglasang, Esin, Yee
Version	0.4.0
Document Status	draft

Version	**Date**	**Author(s)**	**Summary**
0.0.0	05-02-2016	R. Ebarle, C, Maglasang, M. Yee, D. Esin	Document Start
0.1.0	13-02-2016	R. Ebarle, C, Maglasang, M. Yee, D. Esin	Edited Chapter 1, Started Chapter 2
0.2.0	13-02-2016	R. Ebarle, C, Maglasang, M. Yee, D. Esin	Added Chapter 3
0.3.0	13-02-2016	R. Ebarle, C, Maglasang, M. Yee, D. Esin	Added Chapter 4
0.4.0	11-03-2016	R. Ebarle, C, Maglasang, M. Yee, D. Esin	Added Chapter 5

1 Introduction

1.1 Purpose

The purpose of this plan is to define the e-store Software Quality Assurance (SQA) organization, SQA tasks and responsibilities; provide reference documents and guidelines to perform the SQA activities; provide the standards, practices and conventions used in carrying out SQA activities; and provide the tools, techniques, and methodologies to support SQA activities, and SQA reporting.

1.2 Scope

This plan establishes the SQA activities performed throughout the life cycle of the e-store project.

This plan shall implement a project that follows the RESTful architectural style. The project shall be developed using the Flask microframework. There will be a clear separation of concerns between the client and the server for easy maintenance and scalability.

1.3 List of Definitions

Term	Definition
ATDD	Acceptance Test Driven Development
TDD	Test Driven Development
BDD	Behavior-Driven Development
SQA	Software Quality Assurance
UML	Unified Modeling Language
ERD	Entity Relationship Diagram
MSU-IIT	Mindanao State University - Iligan Institute of Technology
REST	Representational State Transfer
FLASK	Web Framework
Python	Programming Language
SCS	School of Computer Studies
Sales Inventory	The list of items such as the goods that are in stock
E-commerce	The buying and selling of goods over an electronic network, primarily the internet
Customer	The person who transacts in the store page of the business
Admin	The owner of the products sold in the store.
Product	The items being sold in the website
Cart	The list of items the customer is going to buy
Checkout	The process in which the customer is going to buy and pay the items inside the cart
Gherkin	Business Readable, Domain Specific Language that lets you describe software's behaviour without detailing how that behaviour is implemented.
QAM	Quality Assurance Manager
SQAP	Software Quality Assurance Plan
SQMP	Software Quality Management Plan
PM	Project Manager
CM	Configuration Manager
AD	Architectural Design
DD	Detailed Design
CI	Configuration Items
UML	Unified Modeling Language

Table 1: List of Definitions

1.4 List of References

- [*SQAP*] Software Quality Assurance Plan, SPINGRID team, TU/e, 0.1.3, June 2006
- Saleh H. (2013). Javascript Unit Testing. Packt Publishing
- Osmani A., (2012). Javascript Learning Design Patterns
- Zlobin G., (2013). Learning Python Design Patterns
- Sale D., (2014). Testing Python
- IEEE Standard for Software Quality Assurance Processes, IEEE Std 730-2014
- Clean Code Cheat Sheet
- Test-Driven Development, Dr. Christoph Steindl, Senior IT Architect and Method Exponent, Certified ScrumMaster
- Best Practices, Development Methodologies, and the Zen of Python, Valentin Haenel
- Test-Driven Development, Gary Brown
- Detailed Design, (2006). Parametric Technology Corporation (PTC)
- Configuration Items, $http://www.chambers.com.au/glossary/configuration_item.php$
- $http://flask.pocoo.org/docs/0.10/styleguide/$
- $http://explore-flask.readthedocs.org/en/latest/conventions.html$

2 Management

This section describes each major element of the organization that influences the quality of the software.

2.1 Organization

The team shall follow the agile approach, and adhere to the scrum approach in development. The team shall consist of the Scrum Master, Product Owner and the Development Team. Throughout each iteration, SQA activities should be headed by the Scrum Master who shall also serve as the Quality Assurance Manager. The team shall follow the Behaviour-driven approach in development as an extension to the Test-driven development approach. Prior to coding any functionality, the individual responsible for the feature shall create just enough acceptance tests, unit tests and code to pass the tests.

2.2 Tasks

The SQA team's main task is to check whether the procedures are followed and that standards are handled correctly as defined in the [SQAP]. Additionally, the SQA team inspects whether all group members fulfill their tasks according to the parts of the [SQAP] applying to their specific tasks.

Besides the described main task, the SQA team has to check the consistency and coherence between documents.

2.3 Responsibilities

The responsibility of Quality Assurance shall be vested in all of the members of the development team. Each one shall serve as a tester and developer at the same time. However, a Software Quality Assurance Manager (QAM) shall be the one to oversee that the BDD approach is followed, and that tests cover 100 percent coverage throughout the system, in order to avoid any bleeds or regression. The agile team shall be self-organizing individuals and take full responsibility in the feature or story assigned to them. The team shall not only be concerned with the product quality but also with the process quality and relationship between them. Should there be any major problems, the QAM shall take over and plan as needed.

3 Documentation

The documents to be delivered in the specific phases of the project will be based in Chapter 4. Document standards are described in the same chapter.

4 Standards, Practices, Conventions And Metrics

4.1 Documentation Standards

Documentations may be in the form of a test, a docstring, or any formal document. If possible, the code should serve as enough documentation for the system. Throughout the project, PEP 8 style guide convention shall be used.

4.1.1 PEP 8 basically commands developers the following practices:

- Indentation: Indent with 4 real spaces (no tabs)
- Maximum line lenght: 79 characters with a soft limit for 84 if absolutely necessary. Try to avoid too nested code by cleverly placing break, continue and return statements.
- Continuing long statements: To continue a statement you can use backslashes in which case you should align the next line with the last dot or equal sign, or indent four spaces.

4.1.2 Docstrings

All docstrings shall be formatted with reStructuredText as understood by Sphinx. Depending on the number of lines in the docstring, they are laid out differently. If it's just one line, the closing triple quote is on the same line as the opening, otherwise the text is on the same line as the opening quote and the triple quote that closes the string on its own line:

4.1.3 Comments

Rules for comments are similar to docstrings. Both shall be formatted with reStructuredText. If a comment is used to document an attribute, put a colon after the opening pound sign (#).

4.2 Design Standards

The system shall follow the restful-architectural style. The API backend shall be built with Flask, while the frontend (running in a different port) shall be built with AngularJS. The following table describes the API model of the project:

The project requires sufficient skill in Python, Flask, and front end technologies like Angular, jQuery and Ajax. The learning curve throughout the project development has been steep and required training from the Advisor and co-team members.

14 Risk management

14.1 Categories of risks

The following are categories of risks that are relevant to the project:

14.1.1 Risks with respect to the work to be done

1. Miscommunication
Probability: High

Prevention: Daily stand ups or quick huddle should be done by the team on a regular basis. Major weekly meetings are done to keep address pressing issues in development. Team members should not hesitate to ask and re ask questions if things are unclear in order to avoid bottlenecks in the progress of the system. With regards to the customer, bi-monthly face-to-face meetups should be done to update and keep track of progress. If any confusions arise, the team may opt to use other communication mediums like phone calls, or emails to clear up problems.
Correction: When it becomes clear that miscommunication is causing problems, the team members involved and the customer are gathered in a meeting to clear things up.
Impact: High

14.1.2 Risks with respect to the customer

Resource	URI	HTTP Method	Description
Inventory Module			
Item	/api/v1/items/	GET	Retrieve all items
	/api/v1/items/:id	GET	Retrieve single item
	/api/v1/items/:id	PUT	Update item
	/api/v1/items/	POST	Create new item
Type	/api/v1/types/	GET	Retrieve all item types
	/api/v1/types/:id	GET	Retrieve all types
	/api/v1/types/:id	PUT	Update type
	/api/v1/types/	POST	Create new type
Attribute	/api/v1/types/:id/attributes/	GET	Retrieve all attributes under the type id
	/api/v1/types/:id/attributes/:id/	GET	Retrieve single attribute under the type id
	/api/v1/types/:id/attributes/	POST	Create new attribute under the type id
	/api/v1/types/:id/attributes/:id/	PUT	Update attribute details under the type id
AttributeValue	/api/v1/items/:id/attributes/	GET	Retrieve all type-attribute pair values for each item assigned to a particular type
	/api/v1/items/:id/attributes/:id/	GET	Retrieve a single type-attribute pair value for an item
	/api/v1/items/:id/attributes/:id/	PUT	Update the type-attribute pair value
	/api/v1/items/:id/attributes/	POST	Create new attribute value based on the type assigned to an item
Image	/api/v1/items/:id/images/	GET	Retrieve all images for a single item
	/api/v1/items/:id/images/:id/	GET	Retrieve single image for an item
	/api/v1/items/:id/images/:id/	PUT	Update image for an item
	/api/v1/items/:id/images/	POST	Create new image for an item
Supplier	/api/v1/suppliers/	GET	Retrieve all suppliers
	/api/v1/suppliers/:id/	GET	Retrieve single supplier
	/api/v1/suppliers/:id/	PUT	Update a supplier
	/api/v1/suppliers/	POST	Create new supplier
Cart & POS Module			
Cart	/api/v1/carts/	POST	Create new cart instance
	/api/v1/carts/:id/	PUT	Update Cart
CartItem	/api/v1/carts/:id/items/	GET	Retrieve all cart items
	/api/v1/carts/:id/items/:id/	GET	Retrieve single cart item
	/api/v1/carts/:id/items/:id/	PUT	Update single cart item
	/api/v1/carts/:id/items/	POST	Add an item to cart
Order	/api/v1/orders/	GET	Retrieve all orders
	/api/v1/orders/:id/	GET	Retrieve single order
	/api/v1/orders/:id/	PUT	Update single order
	/api/v1/orders/	POST	Create new order
OrderItem	/api/v1/orders/:id/items/	GET	Retrieve all order items
	/api/v1/orders/:id/items/:id/	GET	Retrieve single order item
	/api/v1/orders/:id/items/:id/	PUT	Update single order item
	/api/v1/orders/:id/items/	POST	Add new item in order
Wishlist	/api/v1/wishlists/	GET	Retrieve all created wishlists
	/api/v1/wishlists/:id/	GET	Retrieve single wishlist
	/api/v1/wishlists/:id/	PUT	Update wishlist
	/api/v1/wishlists/	POST	Create new wishlist
WishlistItem	/api/v1/wishlists/:id/items/	GET	Retrieve all items under a single wishlist
	/api/v1/wishlists/:id/items/:id/	GET	Retrieve single wishlist item
	/api/v1/wishlists/:id/items/:id/	PUT	Update wishlist item
	/api/v1/wishlists/:id/items/	POST	Add new item in wishlist
User	/api/v1/users/	GET	Retrieve all users
	/api/v1/users/:id/	GET	Retrieve single user
	/api/v1/users/:id/	PUT	Update user
	/api/v1/users/	POST	Create new user
Group	/api/v1/groups/	GET	Retrieve all groups
	/api/v1/groups/:id/	GET	Retrieve single group
	/api/v1/groups/:id/	PUT	Update group
	/api/v1/groups/	POST	Create new group
	/api/v1/groups/:id/users/	GET	Retrieve all users under a group
	/api/v1/groups/:id/users/:id/	GET	Retrieve a user under a group
	/api/v1/groups/:id/users/:id/	PUT	Update a user in a group (e.g. Permission)
	/api/v1/groups/:id/users/	POST	Add a user to a group
Customer	/api/v1/site/:id/customers/	GET	Retrieve all customers in a site
	/api/v1/site/:id/customers/:id/	GET	Retrieve single customer in a site
	/api/v1/site/:id/customers/:id/	PUT	Update customer in a site
	/api/v1/site/:id/customers/	POST	Register new customer in site or business

Table 2: REST API Model