Automated Testing & Deployment Plan

Document Status: DRAFT

Document Date: 03/31/2015

Owners:

Yiran Mao

Jesse Rafalko

Design Principle

The web infrastructure project conforms to the professional web development procedure with proper source control under collaborative development environment, automatic testing and continuous integration deployment.

This document will mainly cover the following content:

- Source control and branch management
- Automatic testing development & implementation
- Continuous automatic integration of the system
- Scale-out deployment

Source control and branch management

Overview

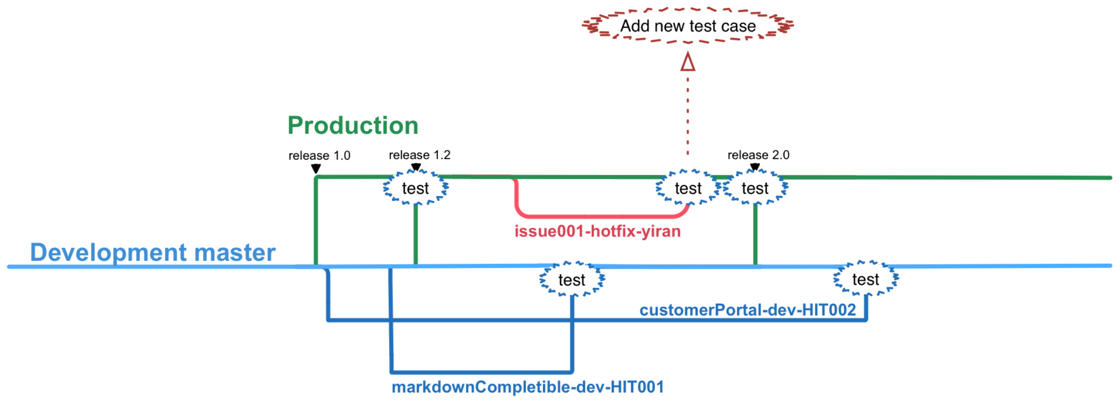
The project utilizes git as the source control tool.

The git repository for the mainline corporate website project is git-sym: sio_cms.
The git repository for the testing module has not been assigned yet.

Git Branch Design

There will be four major components for git branch:
- Production
- Development master
- Feature development
- Hot fix

The following diagram best describe the relationship for the branches.



Production

The production branch is the online code for the corporate website. Production branch should be merged and only be merged with development master branch through automatic continuous integration program and is not designed to be merged with any other branches manually **EXCEPT HOT FIX BRANCHES**.

The continuous integration of the production branch is planned to schedule on Sunday 12:00AM per week. The reasons for not schedule it on daily bases but that specific time point are: firstly we don't want frequent UI development confuse our customers and secondly we can have a quick response for hot fix if we indeed find some problem on Monday. The schedule is subject to change during deployment stage but the general idea should remain unchanged.

Development master

The development master serves as the mainline development branch and the source for production branch continuous integration.

The initial development stage will be occurred directly on development master branch. However, once we reach the codes freeze stage, further scrum on feature improvements should be forked from it and happen on their own branches and thus the development master branch should remain untouched. Once the feature development completes, it will be accepted to merge back to development master branch only if it can pass all testing cases. Otherwise, the merge request will be rejected to ensure the "ready-to-deploy" stage of development master branch.

Feature development

The feature development branch is further feature development associate to the web infrastructure.

On creation, the branch creator should pull the latest code on development master branch and fork its own branch named by "*featureName-dev-teamName*" format, for example "customerPortal-dev-HIT002".

During development stage, the owner of the branch should continuously pull the latest code from development master branch, merge and solve any conflicts that occurred within their feature branch.

Once the feature development is done, the owner should first pull latest code from development master and resolve any existing code conflicts. After that, they should run automatic test to ensure all-testing cases are passed and then merge back to development master branch. On merge occurs, the automatic testing will be triggered before the real merge happens and the merge will only succeed once the code can pass all testing cases.

Hot fix

Even we try to avoid deficits through thorough manual and automatic testing, there will still be bug from time to time that we cannot cover. The hot fix branches serve for correcting any bugs that appears on production branch that expose severe threats to the website.
It should be forked from the latest production branch and named by "issueID-hotfix-ownerName", for example "issue001-hotfix-yiran", where issue ID refers to the ticket that filed in JIRA for detail information reference.

Once finished, the hotfix branch should be merged to **BOTH PRODUCTION AND DEVELOPMENT MASTER BRANCH** to avoid code conflict. Conceptually, any non-urgent bug should not generate a hotfix branch but a short-term feature development branch and hotfix branch should be active only within 24 hours.

ANY SOLVED HOTFIX SHOULD ALSO GENERATE A CORRESPONDING TEST CASE IN THE SYSTEM.

Automatic testing development & implementation

Testing Development Plan

Initial test development
For the initial development of the corporate website, we will run testing mainly for the following aspects:

- Frontend
  - DOM element validation
  - Data validation
- Backend
  - Route existence
  - User authentication
  - Data validation
  - Response format

(The following two parts will be completed in depth once the initial development is complete)
Regular module development

- Frontend
  - DOM element validation
  - Data validation
  - Cross browser compatibility
  - Responsiveness on screen size
- Backend
  - Route existence
  - User authentication
  - Data validation
  - Response format
- Integration

On bug fix

Test Running Hooks

Feature development

Feature unit test should be run on feature development completion. Besides that, integration test should also be run for feature development branch to ensure the system steadiness before merge back to development master branch.

Automatic test

Automatic integration test run will be triggered under following two situation:
- On merge-to-development-master
- On merge-to-production of development master and hot fix

Once succeed, the merge can take place.

Continuous automatic integration of the system

The continuous integration will be handled by Jenkins task.
During the process, Jenkins task will

- Stop parts of the servers but remain minimal level to serve customer requests
- Pull the latest code from production and development master
- Run automatic integration test before merge
- Merge development master to production
- Restart the server and put them back to service
- Update the rest servers

Scale-out deployment

The rough plan for scale out have been mentioned in <*Server Architecture Design Diagrams*> and as we mentioned, we will investigate into Docker see if it is suitable for our scale-out plan later on.