Quality Assurance

Quality Assurance plan and procedures

The section describes the quality assurance procedures and plans to ensure high product quality.

The quality assurance plan will be divided into the following phases:

- Verification phase
- Validation phase
- Code Review phase

Verification phase

The verification phase should contain the following procedures:

1. Prepare the acceptance test to ensure that the functionality described in the user story works properly and as required by the user story.
2. Test the functionality using scripts
3. Test the functionality using manual testing where usage of scripts is irrelevant.
4. In case there found irrelevant behaviour in the program functionality, record the behaviour on the Confluence and let the responsible team/person know about a bug.

Validation phase

The validation phase should contain the following procedures:

1. Identify the requirements the client needs for the current sprint.
2. The assumptions made by the team during the implementation which were not verified by the client must be recorded on the confluence page.
3. If the meeting with the client has occurred, record client's feedback and comments about the assumptions.
4. Make adjustment (if any) to the previous sprint requirements based on client's feedback.

The Code Review phase.

Note: the code review is highly recommended to be completed by several people in each team to identify the possible mistakes and confusions in the code.

The code review should contain the following procedures:

- Check the completed code for the major/critical mistakes and make sure that the code is easy to understand for another person who reads it.
- If there found an unclear code part or part which potentially lead to unreliable behaviour, contact the developing person for the explanation. If the code leads to unintentional behaviour, record it on the Confluence page.

Coding Standards

This section describes the coding standards for the project for consistent standards across the whole project

General Guidelines:

1. For **Javascipt Code** Please adhere to the Airbnb javascript style guide

Models:

The models' names should be exported using the **Upper Camel Case** for the models created specifically for this project.

E.g: user model has to be exported for usage in controllers as "User" with the upper case as the first letter.

Controllers:

The controller names should be constructed using the **Lower Camel Case** for procedures which are exported to the routes for future usage.

Routes:

The routes names for URL addresses should be built using **Lower Camel Case** for usage together with Views

The exported **models** used in the route has to use **Upper Camel Case** for development.

The exported **controllers** used in the route has to use **Lower Camel Case** for development.

Git Workflow

Motivation

A git workflow is an agreed-upon approach used to using git tools within a team. This is important from both a Quality Assurance and a Dev-Ops perspective. For Quality Assurance it allows us to ensure that unstable/untested code isn't mixed with a branch that is designed to hold stable code. From a Dev-Ops standpoint, it allows us to have set conventions as to which branch represents what environment (Development, Staging, Production)

Gitflow

Gitflow is a git workflow suited for collaboration amongst a team.

It is made of several types of branches

- **Master branch**: This is used for production (The product after a sprint).
- **Develop branch**: It is used to hold stable, tested code that we plan to deliver by the conclusion of the sprint we are in.
- **Feature branch**: This is where developers commit their code for the feature they are building. This is where the unstable, untested and incomplete code resides. Once stable, tested and complete the developer makes a pull request to the Develop branch. After our code review process we then merge the branch to develop.
- **Release Branch:** At the end of a sprint we create a release branch from the develop branch to apply final changes before merging to master. This could be bug-fixes or removing features that aren't ready. This branch is then merged to master.
- **Hotfix branch:** Bugfixes for issues in master branch.

A video explanation is given here

Naming Conventions

Feature Branch

Named **feature/<JIRA_ID>-<Short Description>**

- **Where the short description is optional**

For Example: if we have a Jira Task with the ID **CE90013-001** for **adding a Home Page**. Some Appropriate examples are:

1. **feature/CE90013-001**
2. **feature/CE90013-001-Home Page**
3. **feature/CE90013-001-Add-Home-page**

Release Branch

Named after the Sprint that it is being used for. **i.e: At the conclusion of sprint 2 the branch should be called release/sprint-2**

If multiple release branches are required in a given sprint then a letter should be appended to denote additional branches i.e: **release/sprint-2, release/sprint-2b, release/sprint-2c, ...**

Workflow Conventions

Developing a feature

1. Developers building a feature should only be working in the relevant **feature branch.**
2. Once a feature is considered ready for review, a pull request is made to merge the feature in the **develop branch.**
3. If the pull request is approved the feature is merged to the develop branch. These new changes on the develop branch **must** now be merged into **all feature branches.**

Creating a release

1. Towards the conclusion of the sprint, a release branch is made by the **deployment team**. The deployment team will check for any issues before deployment to a production environment and talk to the product owner, front-end and back-end team leads about the removal of any features if necessary.
2. Once the release branch is ready, it is then merged to **master** and **develop** (only the bugfixes for develop)
3. A release will then be created. The release tag must include
    a. Some general notes on the changes that have occurred
    b. A list of all the User stories and their subtasks that were added since the previous release

Bugs In production

1. If any issues are found in the production environment, the deployment team **must** be consulted. A hotfix branch will then be made from the master branch by the relevant developer. Once the fix is ready, a pull request is made where **at least one** reviewer needs to be on the deployment team.
2. Once the hotfix is approved it is merged to both **master** and **develop** branches.

Review process

The review process occurs during the **review** phase of the task on **Jira** platform.

Review process of the project coding tasks

Placing the coding task under review

The assigning process includes the following steps:

1) Creating a **pull request** on the GitHub platform to the develop branch

2) Assign a person using Jira to complete the review process before emerging with the **develop** branch. Note: it is preferred to choose a person who is currently working or worked on the similar/same feature before and has an idea of how the feature was implemented for better testing results. Otherwise, choose a person who is responsible for testing or leader of the team.

3) For consistent communication between team members, it is advised to also notify the reviewer via Slack.

Reviewing process of coding tasks:

The reviewing process includes the following:

1) Using testing scripts provided by the testing team on the GitHub platform

2) Test the new feature implemented in the pull request for correct functionality

3) Test other features which might potentially be related to the current feature and ensure that it does not cause new errors.

4) If the user story has more than 5 points, it is required for 2 team members to review the current process

Review process of the project documentation

Placing the documentation task under review

The assigning process includes the following steps:

1) Placing the task in under review on **Jira** platform and assigning a person who will review it. Note: it is preferred to assign a person who is familiar with the similar job for better results and better efficiency.

2) For consistent communication between team members, it is advised to also notify the reviewer via Slack

Reviewing process of documentation tasks:

1) Check the documentation on **Confluence** platform which was assigned to the team member.

2) Check the documentation consistency with other documents and ensure that it is clear and understandable for all team members to follow the documentation guidelines.

3) Check the documentation for grammatical mistakes and typos to ensure the high quality of the document.

4) Check the reference links for correctness and clear explanations for the corresponding document.

5) Write a review as a comment on the JIRA task (or, as a comment in Confluence, in which case leave a comment on the JIRA task linking to the Confluence comment).

Team members responsibility

To ensure the consistency of the review process each team member is required to follow the next process:

1) Check the **Jira** and **Slack** platforms at least once a day to identify the required review goals assigned to each team member

2) Make a review for the task assigned to the person

3) Comment the pull request and approve/reject the request based on performance.