Test Plan

Artwurk Project

Team Dudagon

Team Leader: Aaron Manithep

Team Members:

Anton Amstadt,

Chris Tran,

Edwin Bulnes,

Joshua Quibin

# Table of Contents

[1 Table of Contents 2](#_Toc86245550)

[2 History 2](#_Toc86245551)

[3 Overview 2](#_Toc86245552)

[Purpose 2](#_Toc86245553)

[Scope 2](#_Toc86245554)

[4 Test Strategy 2](#_Toc86245555)

[Test Strategy 3](#_Toc86245556)

[Test Objective 3](#_Toc86245557)

[Test Assumptions 3](#_Toc86245558)

[5 Test Execution 3](#_Toc86245559)

[Test Creation 3](#_Toc86245560)

[Test execution 4](#_Toc86245561)

[Defect Management 4](#_Toc86245562)

[Projected Test Schedule 4](#_Toc86245563)

[Risks 6](#_Toc86245564)

# History

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Description of Change |
| 1 | 10/27/2014 | Team Dudagon | Submitted as part of Milestone 2 |

# Overview

## Purpose

The purpose of this document is to provide the information and framework required to plan and perform all testing and validation for the Artwurk web app.

## Scope

As outlined in our Project Plan, we will be using AWS to host our web app. AWS will handle all our hardware and networking needs. Consequently, we are not planning on doing any automated performance testing. Of course, like all things, this may change if we feel that performance testing is necessary as we begin developing the app and experiencing performance issues. We plan to use small EC2 spot instances for our build pipeline’s test environment and scale these up to larger instances as needed for our production environment.

# Test Strategy

## Test Strategy

The goal is for all the use cases outlined in our BRD to be covered by automated tests. This would be accomplished largely by unit and integration tests written as we develop the application. Since we consist of a team of 5 that all expect to write code and help develop functionality, this approach would disperse the responsibility of testing and maintaining quality code to everyone on the team. We are hoping that this approach, combined with the typical code reviews will helps us quickly develop a stable and usable test suite.

## Test Objective

The objective of the tests is to verify that the Artwurk web app works according to our specification documents.

The final product after all testing is complete is:

* Production-ready software that is bug-free
* A set of robust and stable test scripts than can be reused for any future testing and enhancements

## Test Assumptions

* Set of test scripts will be run against all new code check-ins
* Any defects found will be tracked using GitHub Issues
* The developer whose change caused a failure in the build pipeline will be initially responsible for logging the defect and fixing it
  + Both logging and fixing can be deferred to other team members depending on capacity and availability on any given sprint
  + This would be discussed and identified during sprint standups and any additional meetings would be schedule as necessary
* Any UI defects logged will have a screenshot of the issue
* Weekly code reviews to be done on any code done during that week
  + Frequency of code reviews may increase as needed
  + Code reviews to include review of test scripts to ensure that a test is actually verifying what it purports to

# Test Execution

## Test Creation

Each developer is responsible for writing integration and unit tests around the feature that they are currently developing that sprint. The expectation is that along with working, functional code, team members will also produce tests that verify that the code produces the expected result.

In addition to unit tests that verify an individual unit (i.e., method) of the applications, developers will also be responsible for creating integration tests that test the interactions between features and other features. Just like with unit tests, these tests will be run automatically by the build pipeline. Any failures need to not only be fixed, but tests must also be updated to ensure that the particular scenario is covered

Additionally, end-to-end test cases will be created to test the functionality outlined in each of the use cases defined in our BRD. That is, once a feature is complete and is expected to be working, additional tests will be created testing the functionality from beginning to end. These tests will be rolled into the recurring tests of our pipeline Thus, if future changes break functionality, we would have a set of tests to account for this.

## Test execution

All tests are to be executed automatically via our build pipeline when a change is checked in to our test environment on AWS. Additionally, since tests consist of a series of test scripts, developers can also run these locally in their own Dev environment using the test runner in VS code. This allows developers to test their code locally as needed while our build pipeline enforces test execution.

## Defect Management

As mentioned earlier, it will be the responsibility of each developer to log any bugs found by the testing suite to GitHub issues. Each bug will be linked to the corresponding test case that failed and the bug will be assigned a severity based on the following scale:

|  |  |
| --- | --- |
| **Severity** | **Impact** |
| 1 (Critical) | * This bug is critical enough to crash the system, cause data corruption, or cause potential data loss * It causes the application to hang and requires re-booting the client or server. |
| 2 (High) | * It causes a lack of vital program functionality with workaround. |
| 3 (Medium) | * This Bug will degrade the quality of the System. However, there is an intelligent workaround for achieving the desired functionality - for example through another screen. * This bug prevents other areas of the product from being tested. * This is likely the default for most bugs |
| 4 (Low) | * There is an insufficient or unclear error message, which has minimum impact on product use. * There is a UI problem that affects usability, but there are reasonable workarounds |
| 5(Cosmetic) | * There is an insufficient or unclear error message that has no impact on product use. * There is UI problem that does not prevent use of the app |

Work for these defects will then form a part of our regular backlog and will be assigned out alongside other work. Team will review bugs weekly (more frequently if needed). If it is determined that a given behavior is not a bug or will not be fixed, the test cases will be updated appropriately to account for this.

## Projected Test Schedule

Based on the schedule defined in the Project Plan document, the projected test schedule is as follows:

Sprint 6: (1st sprint of spring semester):

* Set up test runner on IDE for each team member
* Team to complete training on testing (can be completed earlier)
* Set up AWS CodePipeline and GitHub Issues
* Create a simple build pipeline in AWS and configure the target EC2 instances to be used as resources

Sprint 7:

* Configure AWS S3 Databases and create logging for any DB constraints violations
* Set up notifications for failures for both Build pipeline and DB constraint failures
* Write unit tests around login/register pages
* Write integration tests on login/register feature and its interactions with DB and rest of system
* Write E2E tests verifying login use cases
* Bug and configuration fixes

Sprint 8:

* Write unit tests around user profiles - basic following, followers
* Write integration tests user profiles
* Write E2E tests verifying profile creation use cases
* Write unit tests around posting an image
* Write integration tests around posting an image with focus on interactions with profile and DB
* Write E2E tests verifying posting an image use cases
* Write unit tests around social collaboration functionality - followers/following, community groups
* Write integration tests around social collaboration functionality - followers/following, community groups
* Write E2E tests verifying group use cases
* Write E2E tests verifying following use cases
* Bug fixes

Sprint 9:

* Write unit tests around the edit tree
* Write integration tests around the edit tree’s interaction with other features
* Write E2E tests verifying the edit tree
* Write unit tests around posting an edit
* Write integration tests around posting an edit
  + NOTE: The edit tree and posting an edit are closely related features so it is possible that work here may spill over to next sprint of both are not ready as expected
* Write E2E tests verifying posting an image use cases
* Bug fixes

Sprint 10:

* Verify current tests account for all identified error scenarios of current features
  + Expand the error scenario unit/integration tests if necessary
* Develop set-up test script to populate database with a known list of popular images
* Write unit and integration tests that verify the list of images returned for popular images
* Bug fixes
* Write unit tests for the collaboration feature
* Write integration tests for the collaboration feature and verify interactions with other features (i.e., DB, Profiles, Posting an image.)

Sprint 11:

* Verify tests cover all UI components- Login Page, Profiles, Followers, Following, Images, Edit Tree
* Verify E2E tests cover all use cases in the BRD

Sprint 12:

* Finalize project test documentation
* Bug fixes

Sprint 13:

* Conduct UAT with team members for Artwurk
* Final debugging before launch

## Risks

There are several risks that we considered. We divided these into internal and external risks.

Internal Risks are risks that are inherent and specific to our team. The risks we identified to our team are as follows:

* Lack of team knowledge with unit and integration testing
  + Solutions: Team training
    - The team will spend the first week of sprint 6 (this would be the first sprint of the spring semester) going over the following training material:
      * [Unit Testing for C# Developers](https://www.udemy.com/course/unit-testing-csharp/)
      * Material will be available earlier than this sprint, so Team members are free to review it sooner. However, the first week of Sprint 6 is when we budgeted for training alongside the environment setup that will be done that sprint
* Conflicts in code changes from different team members might lead to brittle tests
  + Solutions: We are implementing a simple CI pipeline
    - Part of the reasoning for using AWS is the relative ease in implementing pipelines as this would help ensure that each check-in is run against our growing suite of must tests
      * Additionally, we are using AWS CodePipeline to orchestrate our build pipeline
      * One of the benefits of this service is that it integrates easily with Jenkins and other build management services
        + While we do not currently foresee the need of having such a complex pipeline as to require these tools, the option is there if we find that we need it later in development
      * Similarly, AWS CodePipeline also integrates with test automation frameworks such as Jest. While we do not currently anticipate needing to use such a framework, we would still have the ability to do so if we find that it becomes necessary during development.
    - Thus, when anyone checks in code, the build pipeline would run and determine if the code fails existing tests
* Project can get too expensive
  + Solutions: We will be thoroughly monitoring the costing of AWS services and will set up alerts to notify when services reach thresholds
  + We should be able to use the free hours and educational credits to cover most of the costs related to development and implementation
    - Additionally, we have set aside a budget of $500 to come from Edwin’s tuition reimbursement provided by his work to cover any unexpected costs

External Risks are risks that outside of our team that would endanger our project:

* Pandemic situation could worsen
  + Solutions: There is little we individually could do to mitigate this situation. We will continue using Agile processes throughout our sprints to foster frequent communication which should help us deal with and mitigate unexpected changes to availability or capacity of the team members
* Earthquake or other natural disaster
  + Solutions: We will have our code repository on GitHub and use AWS as our cloud service. We will select a different availability zone within the US than our current location in California.
    - The Availability Zone is to be determined later as costs associated can frequently vary.
      * Currently, an AZ in the Ohio area seems to be the most cost-effective but that may not be the case by Sprint 6 in the Spring semester.
    - Selecting an AZ in a different geographic area than us will allow us to have surviving infrastructure for our project if a natural disaster hampers our local area
    - Since all these services would be available remotely, we would just need a computer and internet service to get back to work on our project