**Test Plan**

**Dropp**

**Prepared by:**

Steven McCracken

Pongsakorn Cherngchaosil

Pongsathron Cherngchaosil

Alan Dao

Joel Lee

**TABLE OF CONTENTS**

**1.0 Introduction**

**2.0 Objectives and Tasks**

2.1 Objectives

2.2 Tasks

**3.0 Scope**

**4.0 Testing Strategy**

4.1 Unit Testing

4.2 System and Integration Testing

4.3 Performance and Stress Testing

4.4 Beta Testing

**5.0 Hardware Requirements**

**6.0 Environment Requirements**

6.1 Test Environment

6.2 Workstation

**7.0 Test Schedule**

**8.0 Features to Be Tested**

**9.0 Features Not to Be Tested**

**10.0 Dependencies**

**11.0 Risks/Assumptions**

**12.0 Tools**

**1.0 Introduction**

Dropp allows users to post messages and photos anchored to a specific location. It also allows others to view the messages that are posted around them. These messages can be viewed individually as they appear, or in a feed as they travel throughout the world.

**2.0 Objectives and Tasks**

**2.1 Objectives**

Dropp needs to flow seamlessly from page to page for the user, and the data needs to be retrieved quickly. We will ensure this through unit tests for application modules, integration tests, performance tests, and beta tests.

**2.2 Tasks**

* Unit Testing
* System and Integration Testing
* Performance and Stress Testing
* Beta Testing

**3.0 Scope**

Dropp’s server-side, client-side, and database performance will be tested. The server-side functions include updating and retrieving from the database, responding to client requests, and maintaining connection with clients. The client-side functions include displaying the data from the server, and responding to user input to navigate in the application.

**4.0 Testing Strategy**

We will utilize test-driven development when creating modules for our application. There will be four types of tests that we conduct throughout the development of Dropp.

**4.1 Unit Testing**

**Definition:** Verify that the modules we create produce the output that we expect. The unit tests are specific to each function for a module in our application.

**Participants:** All members of our development team.

**Methodology:** We will incorporate a testing framework along with our test modules to verify that each module works as designed. Each developer that works on a function will write the unit test for that function and make sure it passes for all edge cases.

**4.2 System and Integration Testing**

**Definition:** End-to-end testing will be performed when a new module is incorporated into the application. This will ensure that the new functionality works properly with the rest of the application and makes progress towards our final design.

**Participants:** All members of our development team.

**Methodology:** We will incorporate a testing framework along with our application test to verify that all of the modules work as intended after the integration. We will also test to verify that no bugs occur through the interaction of the new module with other existing modules.

**4.3 Performance and Stress Testing**

**Definition:** To ensure that a significantly high amount of users uploading data to the servers concurrently does not cause problems for our servers or any user.

**Participants:** All members of our development team.

**Methodology:** We will write test scripts to simulate a heavy load of users. The script will simulate multiple users and attempt to perform actions on the server and database. We will document all the errors that may occur, as well as the conditions of the server and database. We will write test scripts to test specific components of the server and database. The sequence of testing will be as follows:

1. Script will spawn threads (number of thread will be defined by the tester). Each thread will simulate a user’s activities.
2. Each simulated user will perform various actions with the server and database.
3. The tester will monitor and log the health and condition of the server along with any found bugs.

**4.4 Beta Testing**

**Definition:** Release our application to a specific set of users for them to interact with it how they please, and provide feedback about problems related to performance, usage, or look and feel.

**Participants:** Selected users of the general public.

**Methodology:** We will use Apple’s TestFlight and Android’s Google Play Developer Console Beta Testing systems to invite users to use our application. Their interactions with the application will provide us with valuable feedback. We will track issues that may occur with different versions of operating systems and user actions.

**5.0 Hardware Requirements**

To perform these tests, we will need smartphones: Android and iOS devices. We will also need servers with at least 100gb of storage to store Dropp information like messages and pictures.

**6.0 Environment Requirements**

**6.1 Test Environment**

The servers have to be connected to the internet. A developer’s computer will be used to host a server process. Servers on the connection should have at least 20 Mb/s download and upload rate.

**Mode of usage:** Servers have to be up. Phone app has to be open.

**Level of security:** Low. No remote access of servers is permitted.

**6.2 Workstation**

Developer computers running android and iOS virtual machines.

**7.0 TEST SCHEDULE**

**Unit testing:** Will be performed throughout the development of each module of the application. Whether the module is new or updated, the unit test must be performed.

**Integration testing:** Will be performed for every new module that is integrated into the application.

**Performance and Stress testing:** Will be performed in several stages, mainly towards the end of the development of the application. However, we will also perform this test midway through development to ensure our design and implementation is sound.

**Beta testing:** Will be performed towards the end of development to track and solve any small, outstanding issues discovered by users.

**8.0 Features to be tested**

All of them

**9.0 Features not to be tested**

None of them

**10.0 Dependencies**

The significant constraints on testing are the internet connection and amount of users we gather during the beta testing. Also, deadlines we set throughout the semester will constrain our testing abilities to what we are currently developing at that time.

**11.0 Risks/Assumptions**

While writing this test plan, we are taking the risk that the automation tools and testing frameworks will not provide the functionality that we need to properly and thoroughly test our app. Also, third party services that we use to automate our testing may be down and deemed unusable, either temporarily or indefinitely.

**12.0 Tools**

Travis CI syncs with our GitHub repository to automate our tests and track issues created from the tests. We will use Travis CI for automation and issue tracking. We will also make use of Trello and GitHub Issues to manage testing development, tasks, and procedures. To test the User Interface interactions, we will use Xcode for iOS development and the Espresso testing framework in the Android Testing Support Library for Android development.