TEST PLAN

for the application

CLOUD COMMANDER

carried out as part of

CO421 SOFTWARE TESTING

12-09-2019

TARUN ANAND
16CO147
ARCHIT PANDEY
16CO153

TABLE OF CONTENTS

1. INTRODUCTION **2**

2. OBJECTIVES AND TASKS **3**

2.1. Objectives 3

2.2. Tasks 3

3. SCOPE **3**

4. OUT OF SCOPE **4**

5. TEST ENVIRONMENT **4**

6. SCHEDULES **5**

7. ROLES AND RESPONSIBILITIES **5**

8. DELIVERABLES **5**

9. TOOLS **5**

9.1. Mocha 5

9.2. Selenium 5

10. RISKS AND RISK MANAGEMENT **6**

10.1. RISK 1 - Schedule Risks 6

10.2. RISK 2 - Requirements Inflation 6

10.3. RISK 3 - Technical Risks 6

11. EXIT CRITERIA **6**

1. INTRODUCTION

Cloud Commander is a file manager for the web with a console and an editor. Cloud Commander includes a command line console and a text editor. Cloud Commander helps in managing servers and remote work with files, directories and programs in a web browser from any computer, mobile or tablet. Cloud Commander is open source, works on Windows, Linux, Mac OS and Android, and is written using NodeJS. This test plan documents the steps and processes involved in the testing of Cloud Commander.

2. OBJECTIVES AND TASKS

2.1. Objectives

This document describes the plan for testing the Cloud Commander software. This test plan document aims to fulfill the following objectives:

- Identify the existing project information and the software to be tested.
- List the high level recommended test requirements.
- Recommend and describe the high level testing strategies to be employed.
- Identify the required resources and provide an estimate of the test efforts.
- List the deliverable elements of the test activities.

2.2. Tasks

The following testing tasks are to be carried out-

Functional Testing-

- Unit Testing
- Black Box Testing

Non-functional Testing-

- Performance Testing
- Load Testing

3. SCOPE

The test plan describes the various tests and strategies to be employed in the testing of Cloud Commander.

Unit testing will be employed to gain extensive coverage of source code, and testing of all module interfaces. Unit testing will be done in a black-box manner. The unit testing will attempt to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

Since CloudCommander is a file-browser, performance will be compared to a terminal based browser and a GUI based browser.
The most critical performance measures to test are:

1. Response time for file copying.
2. Response time to access a specified directory.
3. Response times when 10 tabs are open.

4. OUT OF SCOPE

Due to various constraints and issues the following aspects of testing are outside the scope of our testing plan-

1. Integration Testing
2. System Testing
3. Usability Testing
4. Security Testing

5. TEST ENVIRONMENT

We plan to run tests in two environments:

1. NodeJS environment for unit testing, black box testing
2. Automated browser based environment for performance testing, load testing

All tests will be run on a machine with the following specs:

- Intel i7 8th Gen
- 16GB Memory
- 512 GB Solid State Drive
- High-speed internet access

Other details of the test environment:

- Firefox Web-Browser

- Gnome based GUI
- Access to documentation at CloudCMD.io

6. SCHEDULES

We plan to divide the work equally among both the members. We have identified certain tasks as well as a series of milestones to mark our progress. We have estimated the various requirements needed for the tasks and set appropriate deadlines.

Milestone Task	Responsibility	Deadline
Preparing Test Plan	Archit, Tarun	12-9-19
Generation of Test Cases	Archit,Tarun	25-9-19
Unit Test Cases Execution	Archit	10-10-19
Performance Test Cases Execution	Tarun	10-10-19
Load Test Cases Execution	Archit	18-10-19
Black Box Test Case Execution	Tarun	18-10-19
Test Report	Archit,Tarun	25-10-10

7. DELIVERABLES

The various test artifacts to be delivered as part of the testing process are -

- The relevant Test Strategy along with Test scenarios
- Appropriate project Test cases
- The associated test metrics
- Comprehensive Test Report

- Comprehensive test-suite for CloudCommander written using Mocha in Javascript.

8. TOOLS

8.1. Mocha

Mocha is a JavaScript test framework for Node.js programs, featuring browser support, asynchronous testing, test coverage reports, and use of any assertion library.

8.2. Selenium

Selenium is a web automation framework. It starts a web browser and any task that can be done typically on the web.

9. RISKS AND RISK MANAGEMENT

9.1. RISK 1 - Schedule Risks

Given the intangible nature and uniqueness of software , it is inherently difficult to correctly estimate and predict a schedule beforehand. However, in order to mitigate this risk we started working on the schedule earlier and both members were involved in planning and estimating.

9.2. RISK 2 - Requirements Inflation

As the testing progresses, more and more cases that were not identified earlier may turn up and can disrupt the entire testing schedule. Unforeseen situations can affect the entire testing process. In order to ameliorate this risk there needs to be regular trade-off discussions as well as prioritisation sessions that allow changes to proceed and allow initially envisioned testing plans to be superseded.

9.3. RISK 3 - Technical Risks

Technical risks can lead to failure of functionality and performance. Some of the causes of technical risks include changing requirements, unavailability of required tools, unexpected complexity in project progress as well as difficulties in project integration. These issues can be handled by discussing the deliverables, revisiting and revising the objectives when required, open sharing of testing related information along with a daily plan of action.

10. EXIT CRITERIA

The exit criteria is the set of criteria used to determine whether a given test activity has been successfully completed or not. The various exit criterion under consideration are as follows-

- Ensuring all critical Test Cases are passed
- Covering the entire Requirements
- Achieving complete Functional Coverage
- Identifying and fixing all the high-priority defects
- Fixing all the ‘Show Stopper defects’ or ‘Blockers’ and ensuring that none of the identified Critical/Severity 1 defects are in Open Status
- Re-testing and closing all the high-priority defects to execute corresponding Regression scenarios successfully
- No critical defect is left out.

Achievement of all the criteria listed above will be necessary to successfully complete the testing of Cloud Commander.