**Chapter 5: Software Testing Documentation**

5.1 Introduction

5.1.1 Purpose

This chapter’s primary goal is to create testing plans and execute the defects detection and prevention procedures, which may cause software malfunctioning. Another objective of this chapter is to provide details about the software quality and to ensure that the end result meets the business and user requirements. This chapter consists of these following parts:

* Scope of Testing.
* Testing Tool and Environment.
* Resources & responsibilities.
* Test strategy: Test approach, test stages.
* Test schedule.
* Feature to be tested.
* Feature not to be tested.
* Defect Log.
* Test report.

5.1.2 Scope of testing

* Stages of testing:

There are 3 phases in the Testing Process: Unit testing, Integration testing, System testing.

|  |  |  |
| --- | --- | --- |
| **ID** | **Test Stages** | **Description** |
| 1 | Unit test | Unit testing is a software testing method by which small units of source code are tested to determine whether they meet the requirements. |
| 2 | Integration testing | Integration testing is a software testing method in which individual software modules are combined and tested as a group. Integration test’s input modules that have been unit tested are aggregated and undergoes integration test plan and delivers an output that is ready for system testing. |
| 3 | System testing | System Testing is the testing of a complete and fully integrated software product. System Testing is actually a series of different tests whose purpose is to exercise the full computer-based system |

* Type of testing

The test team has to test the following type on Google Chrome

* GUI test
* Performance test
* Regression test
* Range of testing
* Team performs all functions defined in the SRS based on the approved version.

5.2 Test plan

5.2.1 Testing tools and environment

5.2.1.1 Testing tools

5.2.1.1.1 Trip-Sharing Front-end and Project testing

Chrome Developer Tools: To view logs, inspect elements.



Figure 5-1: Chrome Dev Tools

Trello and Backlog: bug control service to log, manage and resolve bugs.

Microsoft Excel: To manage test cases



Figure 5-2: Microsoft Excel

Postman: A tool to test API endpoints and returned results.



Figure 5-3: Postman

5.2.1.1.2 Trip-Sharing API testing

5.2.1.2 Testing environment

|  |  |  |
| --- | --- | --- |
| **Type of testing** | **Software** | **Hardware** |
| System test | Chrome version 71.0.3578.98 (Official Build 64-bit) | Personal computer for developing with the minimum configuration:  - Windows 10 Education 64-bit.  - Intel® Core™ i7 7700K.  - Installed memory (RAM): 16.00GB |
| Unit test | Visual Studio | Personal computer for run testing with minimum configuration  - Visual Studio 2017 or 2019  - Windows 10 Education 64-bit.  - Intel® Core™ i7 7700K.  - Installed memory (RAM): 16.00GB |
| API testing | Postman | Personal computer for run testing with minimum configuration  - Windows 10 Education 64-bit.  - Intel® Core™ i7 7700K.  - Installed memory (RAM): 16.00GB |

5.2.2 Resources and responsibilities

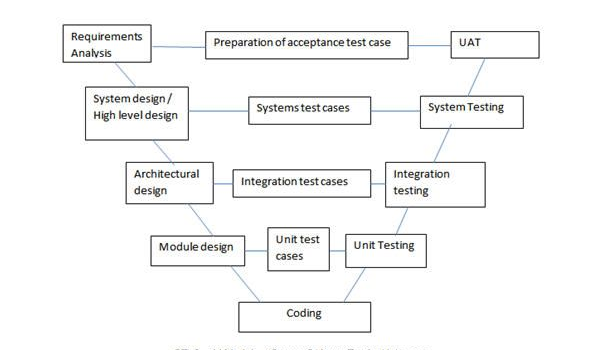
|  |  |  |
| --- | --- | --- |
| **ID** | **Resources** | **Responsibilities** |
| 1 | Project Manager | * Responsible for Project Schedules and overall success of the project. * Review Test-case and report. |
| 2 | Tester | * Preforming the actual system testing. * Manage test resource and assign test tasks. * Create Test Plan. * Create Test Cases. * Create Test Report. * Execute Test. * Test Log report. |
| 3 | Developer | * Create unit test and integration test scripts. * Fix bugs. |

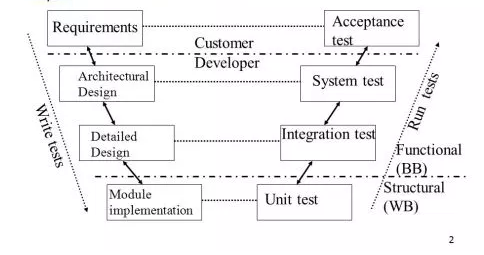
5.2.3 Test strategy

5.2.3.1 Test model

Overall, Trip-Sharing’s deploys the Iterative and Incremental Software Process Model, and the entire system is comprised of 2 main systems: backend API services & front-end services.

Trip-Sharing use the V-Model is an enhanced version of the classic waterfall model whereby each level of the development lifecycle is verified before moving on to the next level. With this model, testing explicitly starts at the very beginning, i.e. as soon as the requirements are written. Here, by testing we mean verification by means of reviews and inspections, i.e. static testing. This helps in identifying errors very early in the lifecycle and minimizes potential future defects appearing in the code later in the lifecycle. In the development time, whenever we add a new feature or change the old features, we will add/modify the tests first, then write code to make the test pass then refactor the code and refactor the test at the last.





Trip-Sharing API has 2 levels of test:

* Unit testing: Develper will write unit test to cover logic of Models and Functions
* API testing: Use Postman tool tests that involve testing APIs directly to determine whether APIs return the correct response for a broad range of feasible requests, react properly to cases such as failures and unexpected/extreme inputs.

5.2.3.2 Test types

* Unit testing:
* Testing individual methods, functions, model class and library class.
* Unit test also includes database testing to verify constraint, transaction, default value, data types, data format, and check null and junk characters which are mentioned in database design and software requirement.
* Test case will have to cover all logic branch that function or method could execute with difference data input. Another alternative logic branch should be covered if not, that logic branch should be detected at API testing level.
* API testing:
* Involves testing APIs directly to determine if they meet expectations for functionality, reliability, performance, and security. API testing will test all of individual implemented API of Trip-Sharing API.
* Test case will verify constraint of data which be mention in Business rule
* Basically, almost all API test cases are executed as automation test. After that all API with standard sample datasets will be saved and confirmation tests will be executed by using Postman with developer’s local database.
* UI testing
* User Interface testing verifies a user’s interaction with the software. The goal of GUI testing is to ensure that the GUI provides the user with an appropriate access and navigation through the functions of the target-of-test. In addition, GUI testing ensures that the objects within the GUI function as expected and conform to requirement o GUI test will be performed fully on all screens.
* This test targets to cover the verification of the overall look and feel of the Trip-Sharing system including initial position, font, text size, color, focus, initial button, tab order, label, screen sizes and sentences width.
* All GUI elements for size, position, width, length and acceptance of characters or numbers. Text font is readable.
* Images have high resolution and properly adjusted for different screen sizes.
* GUI elements are properly aligned for different screen resolutions.
* Regression testing
* Regression testing is a form of software testing to see if its old and new functions are still functioning correctly after changing the system (update new function, fix bug, ...)

5.2.3.3 Test schedule

Table below is the Test Schedule for Trip-Sharing Project

|  |  |  |
| --- | --- | --- |
| **Test Schedule** | **Start Date** | **End Date** |
| Phase 1: | 06/06/2019 | 15/07/2019 |
| Unit Testing and API Testing | 10/06/2019 | 20/06/2019 |
| User Interface Testing | 12/06/2019 | 24/06/2019 |
| System Testing | 25/06/2019 | 05/07/2019 |
| Regression Tests | 06/07/2019 | 12/07/2019 |
| Phase 2: | 20/07/2019 | 15/08/2019 |
| Unit Testing and API Testing | 23/07/2019 | 05/08/2019 |
| User Interface Testing | 25/07/2019 | 05/08/2019 |
| System Testing | 01/08/2019 | 09/08/2019 |
| Regression Tests | 09/08/2019 | 15/08/2019 |

Table 5-1: Test schedule

5.2.4 Features to be tested

All features are listed in the use case list.

5.2.5 Features not to be tested

All features are listed in 1.5.1.3 above will not to be tested.

5.3 Test Case

5.3.1 Unit testing and API testing

Unit testing and API testing will be done by the developers and approved by the development team leader.

The Trip-Sharing development team embrace these features to gain the following advantages:

* Reduce the number of bugs in production code.
* Save development & testing time.
* Automation tests can be run frequently.
* Make it easier to change and refactor code by improving the design of code especially with Test-Driven Development.
* Easier to maintain than GUI tests which are difficult to maintain with the short release cycles and frequent changes and with a complex system
* Reduce cost of resource to corresponding GUI testing.

5.3.1.1 Unit testing framework

* For API testing and Unit testing, we use NUnit is a unit testing framework for .NET. It is the most used framework for writing unit test cases and JetBrains dotCover is a .NET unit test runner and code coverage tool.
* For front-end unit testing, we use Jasmine testing framework with Karma as the test runner. Coverage reporting is done via istabul library.
* All testing frameworks and libraries will be installed to Trip-Sharing front-end automatically by using npm package manager.
* Unit tests use a separate environment which is similar to production environment and includes: Database, Google Application ID, Google Cloud Storage, Google PubSub. For testing complex code that requires running external libraries, unittest.mock library is used to simulate operations.
* Unit testing scripts are created manually and saved to ProjectName.Tests directory of Trip-Sharing API services



Figure 5-5: Test directory structure

* Unit tests focus on individual functions in a class and are created as in the picture.



Figure 5-6: Unit test case sample

This factory function used to bookmark a post and add this post to list bookmark

* Coverage report: (Thêm ảnh chụp coverage)

5.3.2 System testing

Detailed Test cases will be described in TestCase\_Final.xlsx file.

As a standard definition, Trip-Sharing Project defines that a test case is:

* A set of test data and test programs (test scripts) and their expected results. A test case validates one or more system requirements and generates a pass or fail
* A good test case should follow two basic aspects, the Contents and the Style. Test cases for functional testing are derived from the target of test's use cases Test cases should be developed for each use case scenario. The use case scenarios are identified by describing the paths through the use case that traverse the basic flow and alternate flows start to finish through the use case.
* By using good automation test and using, Trip-Sharing Project System testing will not focus on common logic of system like length of text but focus on behavior of website and aims to validate that all software module dependencies are functionally correct, and that data integrity is maintained between separate modules for the entire solution.

5.3.3 Defect Log

* Trip Sharing project used http://www.trello.com in phase 1 and http://www.backlog.com in phase 2 to manager tasks and defects.
* Every member of Trip-Sharing project creates an account backlog and trello to take part in activities: control bugs, fix bugs, re-test bugs and close bug. Bug will be log by tester or developer in develop progress.

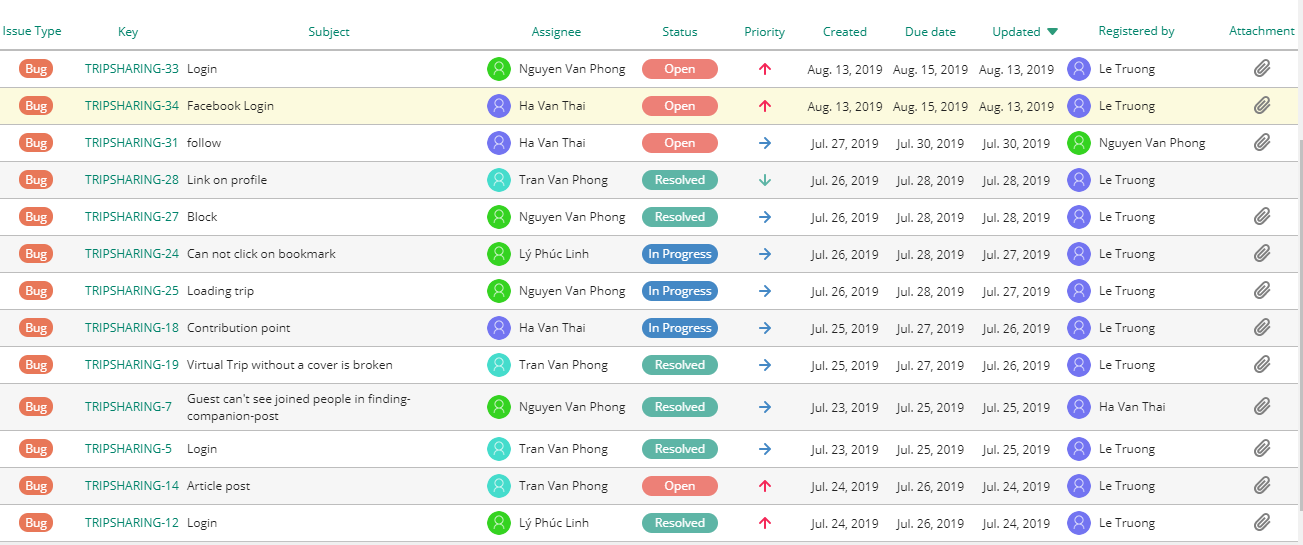


Figure 5-8: Control task and bug with Backlog



Figure 5-8: Control task and bug with Trello

5.4 Test Report

5.4.1 Unit test case report

The contents of the Unit Test Case Report are shown in the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Service Name** |  | **Pass** | **Fail** | **Not available** | **Number of Test Case** |
| Chat Service | Chat Controller | 15 | 0 | 0 | 15 |
| Chat Service | 10 | 0 | 0 | 10 |
| Conversation Repository | 11 | 0 | 0 | 11 |
| Message Repository | 1 | 0 | 0 | 1 |
| User Repository | 4 | 0 | 0 | 4 |
| **Total** | **41** | **0** | **0** | **41** |
| Email Service |  |  | 0 | 0 |  |
|  |  | 0 | 0 |  |
|  |  | 0 | 0 |  |
| Identity Provider | Account Controller | 16 | 0 | 0 | 16 |
| Account Service | 15 | 0 | 0 | 15 |
| Account Repository | 7 | 0 | 0 | 7 |
| BlacklistTokenRepository | 2 |  | 00 | 2 |
| **Total** | **40** | **0** | **0** | **40** |
| Notification Service | Notification Controller | 2 | 0 | 0 | 2 |
| Notification Repository | 4 | 0 | 0 | 4 |
| Notification Service | 2 | 0 | 0 | 2 |
| User Repository | 3 | 0 | 0 | 3 |
| **Total** | **11** | **0** | **0** | **11** |
| Post Service | Article Controller | 11 | 0 | 0 | 11 |
| Article Service | 8 | 0 | 0 | 8 |
| Author Service | 2 | 0 | 0 | 2 |
| Author Repository | 4 | 0 | 0 | 4 |
| Comment Controller | 6 | 0 | 0 | 6 |
| Commnet Service | 5 | 0 | 0 | 5 |
| Comment Repository | 8 | 0 | 0 | 8 |
| Companion Controller | 13 | 0 | 0 | 13 |
| Companion Post Service | 15 | 0 | 0 | 15 |
| Like Controller | 2 | 0 | 0 | 2 |
| Like Service | 4 | 0 | 0 | 4 |
| Post Controller | 3 | 0 | 0 | 3 |
| Post Service | 5 | 0 | 0 | 5 |
| Post Repository | 9 | 0 | 0 | 9 |
| Report Controller | 6 | 0 | 0 | 6 |
| Report Service | 5 | 0 | 0 | 5 |
| Topic Controller | 6 | 0 | 0 | 6 |
| Topic Service | 4 | 0 | 0 | 4 |
| Topic Repository | 4 | 0 | 0 | 4 |
| Upload File Controller | 2 | 0 | 0 | 2 |
| Upload File Service | 1 | 0 | 0 | 1 |
| Virtual Trip Controller | 7 | 0 | 0 | 7 |
| Virtual Trip Service | 7 | 0 | 0 | 7 |
| Virtual Trip Repository | 9 | 0 | 0 | 9 |
|  | **Total** | **146** | 0 | 0 | **146** |
| User Service | Block Controller | 5 | 0 | 0 | 5 |
| Block Service | 3 | 0 | 0 | 3 |
| Block Repository | 2 | 0 | 0 | 2 |
| Bookmark Controller | 6 | 0 | 0 | 6 |
| Bookmark Service | 4 | 0 | 0 | 4 |
| Bookmark Repository | 4 | 0 | 0 | 4 |
| Follow Controller | 10 | 0 | 0 | 10 |
| Follow Service | 7 | 0 | 0 | 7 |
| Follow Repository | 4 | 0 | 0 | 4 |
| Photo Controller | 3 | 0 | 0 | 3 |
| Photo Service | 2 | 0 | 0 | 2 |
| Report Service | 10 | 0 | 0 | 10 |
| User Controller | 19 | 0 | 0 | 19 |
| User Service | 10 | 0 | 0 | 10 |
| User Repository | 2 | 0 | 0 | 2 |
| **Total** | **91** | 0 | 0 | **91** |
| Api Gateway |  |  | 0 | 0 |  |
|  |  | 0 | 0 |  |
|  |  | 0 | 0 |  |
| **Total of Test Case** | | **326** | **0** | **0** | **326** |

Table 5-2: Unit test case report

5.4.2 Unit test report

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case** | **Phase 1** | | **Phase 2** | | **Final** |
| **Pass** | **Fail** | **Pass** | **Fail** |
| Chat Service | 41 | 0 | 41 | 0 | 41 |
| Email Service |  |  |  |  |  |
| Identity Provider | 40 | 0 | 40 | 0 | 40 |
| Notify Service | 11 | 0 | 11 | 0 | 11 |
| Post Service | 135 | 0 | 146 | 0 | 146 |
| User Service | 65 | 0 | 91 | 0 | 91 |
| Api Getway |  |  |  |  |  |

Table 5-3: Unit test report

|  |  |  |
| --- | --- | --- |
| **Service Name** | **Coverage** | |
| **Phase 1** | **Phase 2** |
| Chat Service | 46% |  |
| Email Service |  |  |
| Identity Provider | 66% |  |
| Notify Service | 32% |  |
| Post Service | 51% |  |
| User Service | 66% |  |
| Api Getway |  |  |

Table 5-4: Unit test coverage report

5.4.3 System test case report

5.4.4 System test report