FIT History Tours Master Test Plan

10/4/2022

Table of Contents

1. Introduction

1.1. Scope

This test plan concerns the FIT History Tours app project, and will cover the functional requirements and specified requirements from performance and external interface sections of the Software Requirements Specification. The manner in which the application will be tested along with how the tests will be categorized and errors reported are detailed below. This document serves to ensure that the reader understands what will be tested and how the overall plan for testing those items will be carried out. This document will continue to evolve as the project begins and will be updated according to the guidelines outlined in 3.2.

1.2. References

This is a collection of all items referenced in the FIT History Tours Master Test Plan. This section is broken into 'internal' references which originate from the documents created by our group and the Florida Institute of Technology. The other 'external' references are from other sources.

- 1.2.1. **External References**
  - 1.2.1.1. **829-2008 - IEEE Standard for Software and System Test Documentation**: Link
  - 1.2.1.2. **Selenium Documentation**: Link
  - 1.2.1.3. **Markdown Documentation**: Link
- 1.2.2. **Internal References**
  - 1.2.2.1 **FIT History Tours SRS**: Link

1.3. Software Overview

The developed software will allow for students, staff, and the public to enter and learn about the history of the Florida Institute of Technology in-person or virtually. The user will be able to interact with a map of the campus allowing them to go on "guided tours" or free-roam with a scrollable timeline to keep them interacting with the campus. The virtual tour will allow off-campus users to enter into a 360° view of Florida Tech and experience the buildings and environment while learning about the rich history of the University. The software will use geolocation to keep track of the user and have a historical database of factoids about the campus that can be navigated through by the user. This software will *not* replace the current admissions tour process nor serve the purpose of introducing prospective students to the campus prior to enrollment.

1.4. Test Overview

The application will have a diverse feature set with many different areas to test but by using a black box and white box testing we will aim to test all the major features of the application. This white and black box testing will be completed after the main goals of the current milestone are completed by group members. While a solid schedule can't be provided, it would be ideal to get a week before each milestone is due to test so improvements and fixes can be implemented. Selenium and other frameworks will be utilized to keep all testing as consistent across versions of the application as new features are implemented. Each group member will be responsible for testing their own additions to the application but during the final week, each member will dynamically test parts of the entire application.

1.4.1. **Organization**

FIT History Tours is a group of students in completing their Senior Design Project at Florida Tech. The group members communicate through text chat in the appropriate channel in a private Discord server for development and interactions. Testing will be conducted throughout the development cycle of the application. Each individual group member will be responsible for testing the features and functionality added to the application while it is being developed. Once that change has been pushed to the core repository, other group members will check the code before it is added to the main production branch. A week before each milestone is due, the group plans to do a thorough series of tests split among the group to check the features added by others. If there is a bug or error found in implemented features, it should be reported on the main repository and will be posted automatically into the #bug-finds channel on Discord. This will alert the individual to the need to investigate the bug that was found. All members can be reached by one another and there is no hierarchy of reporting.

1.4.2. **Master Test Schedule**

During the development of new features and patches, all code added or removed will be built into a new version of the application the developer will test before committing it to the main code repository. The code will then be committed to the main repository and a member of the group will be assigned to check over that code and approve its addition to the production codebase. Full group testing will hopefully be done the week before a milestone is due to ensure that any bugs are found and errors can be fixed before the feature-set is demonstrated to instructors and clients. Previous tests should be run again when a new feature is implemented to ensure it hasn't broken previously implemented functionality.

1.4.3. **Resources Summary**

All tests will be primarily within the group but external clients may be asked to test different requirements depending on the need for external input. One of the tools that will be utilized for testing is Selenium which allows for automation of testing for web applications. This functionality will allow automated test suites to run after code changes are pushed or ensure that functionality is still the same after an update. Other automation and testing tools may be utilized and this document will be updated accordingly. All group members have access to all systems that will be utilized to make the application run (virtual machine, database, public DNS records) so that all members can test any area of the application they may be working on.

1.4.4. **Responsibilities**

All group members are responsible for testing their own code bases before committing to the main codebase. Another randomly assigned member will look that code over to ensure that it is correct. During the main testing, all group members will complete a test suite to ensure that as much of the application is thoroughly tested as possible.

1.4.5. **Tools, Techniques, Methods, and Metrics**

Throughout the testing process, the group will utilize black box and white box testing to ensure that all areas of the user interface and codebase are effectively tested. A variety of physical testing will need to be done during black box testing to ensure that the application's geolocation features are both accurate and working correctly while the user moves around the Florida Tech campus. The “virtual tours” feature will also need extensive black box testing to ensure the data collected from the Google Maps API and database geolocation points provide and correct historical facts and mimic moving around the Florida Tech campus. The white box testing conducted will need to ensure that the internal functions of the applications are passing data smoothly between different pieces of the application while making sure that data is error checked. This white and black box testing will be completed after the main goals of the current milestone are completed by group members. These techniques will help the group to keep track of all elements of the codebase and user interface. To prevent all of the testing from being manual, the use of Selenium will help with some black box testing to keep automated test cases for simple and scriptable tasks within the application. The metrics will be recorded in decision tables for black box testing and both linting and unit testing documentation built after testing. These metrics will be recorded along with the central codebase and other updates to the project during that time.

2. Details of the Master Test Plan

2.1. Test Processes and Test Level Definitions

The processes applicable to our development lifecycle include:

2.1.1. **Management**

The administration of our project in its entirety as well as the testing efforts performed in parallel to development necessitate the presence of a MTP (this document) and evaluation of the level tests and their corresponding documentation periodically throughout the project lifecycle.

2.1.2. **Development**

During the development process the codebase will be created. Prior to acceptance and commitment to the shared repository, code will be statically tested on the following criteria:

2.1.2.1. The code is complete and all syntactical requirements are met.
2.1.2.2. The code is annotated to explain functionality and purpose.
2.1.2.3. The code is traceable to requirements found in the SRS and the SRS is updated if code generates new requirements.
2.1.2.4. The code is modular, i.e. each independent component is cohesive and components are coupled at less than or equal to two locations with a probability of 75%.
2.1.2.5. The code is readable and follows a consistent style.

2.1.3. **Operation**

Operational testing of prototypes will generate the greatest number of test cases in the pursuit of requirements verification. The testing phase will map as directly to the requirements as possible, with our naming convention referring to the corresponding “shall” statement from the SRS by number. Our test suite includes:

2.1.3.1. **Functional Testing**
Test Set (3.2.2)

Test Case (3.2.2.1., 3.2.2.2)	
Goal	The user starts the app and initiates a walking tour.
Precond.	The app is installed on mobile or a compatible browser version is in use. The user is on the Florida Tech campus.
Input/Output	Accept guided tour prompt/Geolocation data, UI updates
Postcond.	The app will begin guiding the user on a tour following the preset tour order. A guideline will display on the roadmap.
Comments	This test case can fail safely by exiting to Free Roam mode on error.

Test Case (3.2.2.1.2, 3.2.2.2.2)	
Goal	The user starts the app and initiates a virtual tour.
Precond.	The app is installed on mobile or a compatible browser version is in use.

Input/Output	Reject location service or Located off campus/UI updates, Text and Images
Postcond.	The app will begin guiding the user on a tour following the preset tour order. A point of interest will begin expositing information.
Comments	This case ensures that the alternative tour experience is accessible on start-up

Test Set (3.2.3)

Test Case (3.2.3.1.)	
Goal	The user can navigate to a point of interest using campus pathways.
Precond.	Application is open and a guided tour has been initiated.
Input/Output	Geolocation data, Accelerometer data/UI updates
Postcond.	The user arrives at point of interest, software stops giving directions to the point of interest.
Comments	This case is looking to verify map display accuracy.

Test Case (3.2.3.2., 3.2.3.3, 3.2.3.4)	
Goal	The user can activate a point of interest and continue a tour.
Precond.	Guided tour in progress.
Input/Output	Geolocation data, Accelerometer data/UI, Text, Images
Postcond.	The software displays information about the nearby point of interest and suggests next location to explore.
Comments	

Test Set (3.2.4.)

Test Case (3.2.4.1., 3.2.4.4)	
Goal	The user can begin a tour from Free Roam.
Precond.	App is open, No Tours in progress.
Input/Output	Select Guided or Virtual Tour/UI mode change

Postcond.	The software begins a walking or virtual tour.
Comments	

Test Case (3.2.4.2., 3.2.4.3.)	
Goal	The user can select any point of interest for information or directions.
Precond.	App is open, No tours in progress.
Input/Output	Select Point of Interest on Map/UI update, Text, Images
Postcond.	The software displays historical content for the point of interest, and directs the user if requested.
Comments	

Test Set (3.2.5.)

Test Case (3.2.5.1., 3.2.5.2)	
Goal	The software guides users through a virtual tour using a predetermined start point.
Precond.	App is open.
Input/Output	Select Virtual Tour/Map UI updates, Historical Content Display
Postcond.	The Clemente Center is targeted and historical content for the point of interest is displayed.
Comments	

Test Case (3.2.5.2.1., 3.2.5.3.)	
Goal	The user can navigate a virtual tour to learn more.
Precond.	App is Open, Virtual Tour in Progress
Input/Output	Navigate timeline or Select Next button/UI updates, Content Display updates
Postcond.	The user is shown new historical content on the selected point of interest, or moves onto next point of interest.
Comments	

Test Set (3.2.6)

Test Case (3.2.6.1.)	
Goal	The user can begin a scavenger hunt on-campus.
Precond.	App is Open, Geolocation accepted, User is on campus grounds.
Input/Output	Select Scavenger Hunt, Geolocation data/UI hints and objectives
Postcond.	The user is guided to identify details about the nearest point of interest.
Comments	

Test Case (3.2.6.2.2.)	
Goal	The user discovers the nearest point of interest as part of the scavenger hunt.
Precond.	App is Open, Scavenger hunt in progress, >15 meters away from point of interest.
Input/Output	Geolocation data/UI hints and guidance
Postcond.	The user can locate and approach the targeted point of interest by hints or gps directions.
Comments	

Test Case (3.2.6.2.1.)	
Goal	The user explores around a point of interest to learn historical content.
Precond.	App is Open, Scavenger hunt started, <=15 meters away from point of interest.
Input/Output	Select answers to question or objectives/UI updates and hints
Postcond.	The user discovers a fact about the targeted point of interest, and a new location is targeted.
Comments	

2.1.3.2. **Performance Testing**

Test Set (3.3.1)

Test Case (3.3.1.1., 3.3.1.2.)	
Goal	>=20 users can access the appserver simultaneously for tours and historical content.
Precond.	>=20 user test devices or clients open the app and initiate a tour.
Input/Output	Client GET requests/Server responses
Postcond.	The server can service all users asynchronously in tours across the campus.
Comments	

Test Case (3.3.1.3.)	
Goal	Users can stream audio files from the server without loss of service.
Precond.	User device compatible and has a speaker, App is open
Input/Output	Client audio requests/Server audio stream
Postcond.	Server can supply stream bandwidth of >= 8Mbps.
Comments	

Test Set (3.3.2.)

Test Case (3.3.2.1., 3.3.2.4.)	
Goal	Location services can be processed frequently and quickly.
Precond.	Client connection with geolocation services enabled.
Input/Output	Client location information/Server update response
Postcond.	Clients can receive updated location representation in <= 1.5 seconds utilizing <= 20% processing power of local device.
Comments	

Test Case (3.3.2.2., 3.3.2.3.)	
Goal	Users receive rapid response from the client application when supplying input.

Precond.	Device is compatible, App is installed
Input/Output	User opens app or interacts with UI/UI response and update
Postcond.	Client-side input response time <= 1 second.
Comments	

Test Case (3.3.2.5., 3.3.2.6.)	
Goal	The user can scroll quickly and the app will load information at pace.
Precond.	Device compatible, App is open and timeline enabled.
Input/Output	User scrolls along timeline/UI loads info and images
Postcond.	Textual and Image content should display in <= 1 second when pulled from server.
Comments	

2.1.4. **Maintenance**

Following the deployment of our application, the capacity to repair and update our implementation is paramount. The Anomaly Reporting and Resolution policy [2.3.1.] is the initiative by which modifications to production code can be proposed and evaluated. Any changed or newly implemented code resulting from an Anomaly Report will necessarily undergo the same static scrutinization defined in the Development process [2.1.2.] prior to verification by Operational test cases [2.1.3.]. Tests traced to a requirement that is refined through Maintenance and Anomaly Reporting will receive updates to reflect any changes and remain consistent.

2.2. Test Documentation Requirements

All testing documentation should be able to be described as clear and thorough. The purpose of the test documentation is to ensure that all members of the group understand and know what parts of the software have been tested and what parts of the software need to be tested. The test documentation should clearly state what parts of the software is being tested, how it's been tested (including inputs if applicable), and the results of said tests (including the outputs if applicable). In theory, any other member of the group should be able to follow another member's documentation to reproduce the results, as so they too can understand it, especially if the results are unexpected or error prone. The testing documentation should also serve to make sure that two members of the group don't test the same aspect of the software. This is to save on time and resources, as unless needed, individual members should work on different aspects of the software to ensure that as much of it is working as expected and free of errors that hurt the user experience.

2.3. Test Administration Requirements

2.3.1. **Anomaly Reporting and Resolution**

If there are any anomalies found in the software while it is being tested, an issue should be opened on the main repository with the following information:

- What version of the application are you running?
- What browser/device and version are you using?
- What were you trying to do?
- What did you expect to happen?
- What actually happened?
- What are the steps to reproduce this error?
- Anything else?

These questions will provide the developer with the necessary information to begin investigating the reported bug in the application. Among the group, we will tag the corresponding member that will go solve the issue in the codebase or explain to the group why it may not actually be correct. An anomaly should be closed within a week of reporting but this may be extended if the group member has other external circumstances that have been shared with the group. If the issue isn't something to be publicly shared, the group should still be informed that it won't be completed on time but specifics are never required.

2.3.2. **Task Iteration Policy**

The fixed anomaly should continue to be tested for the next milestone and depending on the severity of the anomaly on the application, potentially more milestones. If testing the anomaly will cause extra strain on the hosting platform, it should be discussed with the group to prevent extra costs from being incurred.

2.3.3. **Deviation Policy**

This policy may be deviated from if a member needs immediate help with an issue while they are working on the production codebase or if the development codebase has stopped working with recent updates. Inquiring the developer by a direct tag in Discord based on the latest commit to the code and working backwards will be the main way that the group will work to get the codebase working properly once again. Tagging an individual in Discord should only be done in the case of great urgency or if the group is collectively working together during a specified time.

2.3.4. **Control Procedures**

The production environment should be used to try and recreate the anomaly that was reported. Other information such as operating system, browser, and other information reported on the anomaly should be mimicked as closely as possible. Due to budget constraints and personal devices being different, the issue may need to be replicated on the same device for the developer if the anomaly can't be recreated by them locally.

2.3.5. **Standards, Practices, And Conventions**

All reporting should be fair and not attack members of the group under any circumstances. Repeated comments and reports of this sort should be reported to the professor of the class or another member of the Florida Tech faculty.

2.4. Test Reporting Requirements

All testing reports should be relayed into the group Discord for archival and communication purposes to the rest of the group. This channel will allow all members to see the test report and go back to it at later times to ensure it is accurate and something that had previously been tested to prevent adding it to the test suite multiple times at different locations. These should be properly formatted in a file format that is directly readable in the default Discord text channel. The preferred report would be posted in Markdown with information correctly formatted in the desired method for the type of testing run. These reports should have logs of the test and the anomalies reported. Any issues found should be referenced by codebase issue number so that as anomalies get fixed the test reports can reflect that information.

3. General

This section contains a glossary of terms referenced throughout the Master Test Plan and the correct method for updating and versioning this document.

3.1. Glossary

UI	User Interface
Florida Tech, FIT	Abbreviation for Florida Institute of Technology
Hosting Platform	Platform used to host the web application (Amazon Web Services, Google Cloud Compute, DigitalOcean)

3.2. Document Change Procedures and History

Document history will be kept by Google Documents automatic versioning history. Upon a version being updated to finished state with updated information, the document will be redistributed to our website with an updated version and date on the title page. The old documents will be stored on the website but not directly addressable to the public via the included hyperlinks provided on the front page.