Test Plan Document

Public Vehicle Tracking (PVT) Application

Version 1.0

Team TKG

Shashank Misra

Devendra Rajput

Shashank Shukla

Vinita Sharma

Shefali Jain

**Table of Contents**

Table of Contents.................................................................................................................................. 2

**1. Introduction................................................................................................................................3**

1.1 Test Plan View............................................................................................................. ....... 3

**2. Objectives and Tasks...................................................................................................................3**

2.1 Objectives............................................................................. ............................................. 3

2.2 Process Overview............................................................................................................... 3

2.3 Testing Process................................................................................................................... 4

**3. Testing Strategy ........................................................................................................................6**

**4. Unit Testing................................................................................................................................6**

4.1 White Box Testing...............................................................................................................6

4.2 Black Box Testing................................................................................................................ 7

4.3 Integration Testing............................................................................................................. 8

4.4 System Testing...................................................................................................................10

4.5 Performance Testing..........................................................................................................10

4.6 Interface Testing………........................................................................................................11

**5. Hardware Requirements...........................................................................................................11**

**6. Deliverables...............................................................................................................................12**

Introduction

**1.1 Test Plan View**

Public Vehicle Tracking (PVT) is a user friendly android application which can be the best solution to the travelling passenger. User of this application just has to possess an android device. Using this application user can track his target vehicle saving his valuable time. The Test Plan for this application has been created to communicate the test approach to team members. It includes the objectives, scope, testing strategy, hardware requirements, environmental requirements, schedule, risks and approach. This document will clearly identify what the test deliverables will be and what is deemed in and out of scope.

Objectives and Tasks

**2.1 Objectives**

Test Plan of Public Vehicle Tracking Application will have an approach to maintain an error free and accurate output application. This can be done by focusing on all aspects of the target app. The test team is responsible for testing the product and ensuring it meets their needs. The test team is both the customer and the tester in this project.

**2.2 Process Overview**

The following represents the overall flow of the testing process:

1. Identify the requirements to be tested. All test cases shall be derived using the current Program Specification.
2. Identify which particular test(s) will be used to test each module.
3. Review the test data and test cases to ensure that the unit has been thoroughly verified and that the test data and test cases are adequate to verify proper operation of the unit.
4. Identify the expected results for each test.
5. Document the test case configuration, test data, and expected results.
6. Perform the test(s).
7. Document the test data, test cases, and test configuration used during the testing process. This information shall be submitted via the Unit/System Test Report (STR).
8. Successful unit testing is required before the unit is eligible for component integration/system testing.
9. Unsuccessful testing requires a Bug Report to be generated. This document shall describe the test case, the problem encountered, it’s possible cause, and the sequence of events that led to the problem. It shall be used as a basis for later technical analysis.
10. Test documents and reports shall be submitted. Any specifications to be reviewed, revised, or updated shall be handled immediately.

**2.3 Testing Process**

**a.** Organize Project

**b.** Design System Test

**c.** Design/Build Test Proc.

**d.** Organize Project

**e.** Design/Build Test Proc.

**f.** End Plan

**Test Process Flow**

The diagram above outlines the Test Process approach that will be followed.

**a.** **Organize Project** involves creating a System Test Plan, Schedule & Test Approach, and assigning responsibilities.

**b.** **Design/Build System Test** involves identifying Test Cycles, Test Cases, Entrance & Exit Criteria, Expected Results, etc. In general, test conditions/expected results will be identified by the Test Team in conjunction with the Development Team. The Test Team will then identify Test Cases and the Data required. The Test conditions are derived from the Program Specifications Document.

**c.** **Design/Build Test Procedures** includes setting up procedures such as Error Management systems and Status reporting.

**d.** **Build Test Environment** includes requesting/building hardware, software and data set-ups.

**e. Execute System Tests –** The tests identified in the Design/Build Test Procedures will be executed. All results will be documented and Bug Report Forms filled out and given to the Development Team as necessary.

**f.** **End Plan** - End Plan happens when all pre-defined exit criteria have been achieved.

Testing Strategy

Testing strategy will include following points:

1. Test Metadata (such as Tested by, Test type, Test Case Description etc)
2. Item(s) to be tested
3. Specifications :

It will include

1. Input
2. Expected Result/Output
3. Procedural Steps

Unit Testing

Unit Testing is done at the source or code level for language-specific programming errors such as bad syntax, logic errors, or to test particular functions or code modules. The unit test cases shall be designed to test the validity of the programs correctness.

**4.1 White Box Testing**

In white box testing, the UI is bypassed. Inputs and outputs are tested directly at the code level and the results are compared against specifications. This form of testing ignores the function of the program under test and will focus only on its code and the structure of that code. Test case designers shall generate cases that not only cause each condition to take on all possible values at least once, but that cause each such condition to be executed at least once. To ensure this happens, we will be applying Branch Testing. Because the functionality of the program is relatively simple, this method will be feasible to apply.

By using branch testing we can test almost every unit of coding. The testing can be done on following blocks in branch by branch i.e. controlled flow manner.

* ANDROID DEVICE INTERFACE BACK END CODING BLOCK
* AREA MAP BLOCK
* SERVER TO DEVICE CODING BLOCK
* VEHICLE GPS DEVICE TO SERVER BLOCK

As all these blocks are inter related to each other by a sequential flow (which can be seen in UML diagrams) so we can separately test them.

**4.2 Black Box Testing**

Black box testing typically involves running through every possible input to verify that it results in the right outputs using the software as an end-user would. We have decided to perform Equivalence Partitioning and Boundary Value Analysis testing on our application.

4.2.1 Equivalence Partitioning

In considering the inputs for our equivalence testing, the following types will be used:

* Legal input values – Test values within boundaries of the specification equivalence classes. This shall be input data the program expects and is programmed to transform into usable values.
* Illegal input values – Test equivalence classes outside the boundaries of the specification. This shall be input data the program may be presented, but that will not produce any meaningful output.

The equivalence partitioning technique is a test case selection technique in which the test designer examines the input space defined for the unit under test and seeks to find sets of input that are, or should be, processed identically. The following table represents our equivalence classes, both valid and invalid.

|  |  |  |
| --- | --- | --- |
| **Input/Output Event** | **Valid Equivalence Classes** | **Invalid Equivalence Classes** |
| Input maximum number of allowed value | 1 value | > 1 values |
| Input characters | Characters between a-z, A-Z  Integers | Non- characters  Decimal values |
| Load external file | File exists | File does not exist |
| Store external file | File exists | File does not exist |

4.2.2 Boundary Value Testing

The acceptable range of values for this application was set by the development team. Due to the limitations of the GUI, the developers also limited the size of the input values to characters & Integers.

**4.3 Integration Testing**

### 

### 4.3.1 Incremental Testing

There are two primary modules that will need to be integrated: the Graphic User Interface module and the Server & Database Module (back-end). The two components, once integrated, will form the complete Public Vehicle Tracking Application. The following describes these modules as well as the steps that will need to be taken to achieve complete integration. We will be employing an incremental testing strategy to complete the integration.

**Module 1 - Graphic User Interface (GUI) Module**

This module provides a simple GUI where the user can perform the different actions (functions). This module will be tested separate from the backend to check if each interface (e.g. insert button) is functioning properly, and in general, to test if the pointer-event actions are working properly. The testing will be performed by writing a stub for each element in the interface.

**Module 2 – Acknowledgement Server & Database Backend Module**

The “Acknowledgement Server & Database” provides the storage for the data elements and implements the algorithms and associated functionality of the tracking application. This module will be tested separate from the GUI by printing out the results to the Console. In testing this module we will follow the incremental testing method i.e. testing one function first and then keep adding additional function and test it again until all the required functions are tested.

When the GUI is combined with the backend module, we will have a complete Public Vehicle Tracking Application. To achieve complete integration of these two modules, we will test each element in the GUI by replacing the stubs with the appropriate function from the back end. The results will be displayed within the GUI instead of through the Console. In testing the combined modules, we will follow the incremental testing method. Each stub will be replaced one at a time and tested. This will be done until all stubs have been replaced by the appropriate functions from the backend.

## 4.4 System Testing

The goals of system testing are to detect faults that can only be exposed by testing the entire integrated system or some major part of it. Generally, system testing is mainly concerned with areas such as performance, security, validation, load/stress, and configuration sensitivity. But in our case well focus only on function validation and performance. And in both cases we will use the black-box method of testing.

### Function Validation Testing

The integrated “Public Vehicle Tracking Application” will be tested based on the requirements to ensure that we built the right application. In doing this test, we will try to find the errors in the inputs and outputs, that is, we will test each function to ensure that it properly implements the algorithms, and that the result displays the values in the proper location graphically. .

In addition, we will test:

* The interfaces to ensure they are functioning as desired (i.e. check if each interface is behaving as expected, specifically verifying the appropriate action is associated with each click event).
* The interaction between the GUI and the backend repository of server. In this case the data will be inserted and check if they are processed in the backend and give the expected output.

### 4.5 Performance testing

This test will be conducted to evaluate the fulfillment of a system with specified performance requirements. It will be done using black-box testing method. And this will be performed by:

* Storing the maximum data in the files & Data Base and trying to insert, and observe how the application will perform when it is out of boundary.
* Deleting data and check if it follows the Tracking algorithm to give the resulting data or output.
* Trying to store new data and check if it over writes the existing once.
* Trying to load the data while they are already loaded.

**4.6 Interface Testing**

Usually done at integration stage when modules or sub-systems are combined . Objective is to detect error or invalid assumption about interface between modules. Reason these are not shown up in unit testing is that test case may perpetuate same incorrect assumption made by the designer.

Three common kinds of interface errors:

1. Interface misuse: User gives invalid destination
2. Interface misunderstanding: User misunderstanding of called component and receives data in illegal and unexpected form
3. Timing error: User/Server data operate at different speeds and data is accessed before being ready. “RACE CONDITION”.

HARDWARE REQUIREMENTS

The Hardware Interfaces will be requiring the basic and necessary devices which will be required for the functioning of the PVT application. The Hardware Requirements will include:

* User Android Device
* Vehicle GPS Tracker Hardware
* Acknowledgement Server

Other Hardware requirements will be according to the advanced functionality of the application.

DELIVERABLES

* Program function specifications
* Program source code
* Test plan document - this document should address testing objectives, criteria, standards, schedule and assignments, and testing tools.
* Unit Testing Plan
* Integration Plan
* System Testing Plan
* Test Design Document
* Test report document