PiSonal Trainer: Weight Lifting Performance Tracker
Test Plan Version 0

Birunthaa Umamahesan   Micaela Estabillo   Simarpreet Singh

November 2, 2016

Prepared for Computer Science 4ZP6: Capstone Project
Instructor: Dr. Wenbo He Fall/Winter 2016-2017

Contents

List of Tables

Revision History

Date	Version	Primary Author	Comment
28/10/2016	1.00	Micaela Estabillo	Created document and drafted system testing section
31/10/2016	1.00	Birunthaa Umamahesan	Drafted overview section and non-functional test plan
01/11/2016	1.00	Simarpreet Singh	Drafted proof of concept testing and functional requirements test plan
02/11/2016	1.00	Micaela Estabillo	Final editing and proofreading for revision 0

Table 1: Revision history

1 Overview

The purpose of this document is to provide a detailed plan for the testing of the application Pi-Sonal Trainer. The following brief outline gives an overview of what is covered in this document:

- A proof of concept test is described in **section 2**
- The set of tests used in testing the system is described in **section 3**.
- The set of tests used to ensure that the software requirements specifications are met is described in **section 4**.
- A timeline of the test plan is given in **section 5**.

Test # :	**Test name**
Description:	A description of what is being tested
Type:	The type of test
Tester(s):	The people who will run manual test
Initial State:	The initial state of the system being tested
Input:	The input that will change the state of the system
Output:	The relevant output that is checked
Pass:	The description of the pass criteria for test

Table 2: Test Case Format

1.1 Automated Testing

Automated testing will be used for testing the application's system, specifically the backend processing of data and transmitting it to the mobile application. The automated testing will encompass both unit testing and coverage analysis. The test cases have been created such that unit testing can be performed on independent components of the application, allowing testing to run alongside the development of the application.

1.1.1 Testing Tools

The software tools that will be used to perform the automated testing are listed in the following table.

Tool	**Description**	**Use**
UnitTests	Python unit testing framework	Unit testing
Nose	Python test coverage analyzer	Analysis of unit test coverage

Table 3: List of Testing tools

1.2 Manual Testing

Manual tests will be conducted throughout the development of the application. It will be used for testing the application where automation testing is not feasible or efficient. The developers will carry out these tests as part of their development process.

1.2.1 User Experience Testing

Manual testing will also be used to assess the overall user experience of the application. The user experience testing will be conducted by allowing a group of individuals who were not involved in the development of the application to complete end-to-end testing of the application. The testers will be given a copy of the application and will be asked to provide feedback. The same individuals will be asked to complete the test twice; once in mid-February and once in mid-March. This will allow us to implement the feedback provided in February and re-test to see if the changes meet users' expectations.

1.3 List of Constants

Constant	Value	Description
α	99%	Test coverage target
β	8	Number of testers
λ	20%	Phase II testing improvement target
γ	1%	Counter increment for repetition and set count.

Table 4: List of constants

2 Proof of Concept Testing

Proof of Concept testing is to be carried out before the developing the rest of the project. In this section, the project's significant risks, demonstration plan and proof of concept test plan will be discussed.

2.1 Significant Risks

The successful completion of the project depends on overcoming the following significant risks:

- The products's exercise counting mechanism must be reliable before it is completed. It needs to be able to accurately count exercise repetitions.
- The product needs to give quick feedback showing the user's performance statistics on their phone once a they finish a set. Designing an effective database schema would reduce the processing time between storing user data and showing it to the user. It would also significantly lower the chance of errors and make them easier to detect while inserting data in using an inefficient database schema.
- The camera plays a huge role in the success of the project because it is used to detect the motion of the user. Low resolution images, large image distance from the camera, and undetectable markers pose a significant risk to the quality of motion detection generated.
- The user's privacy of information is also an important factor in implementing this project since losing trust may cause failure. Finding ways to protect users' data is a significant risk.
- Availability of the product is also required for the completion of this project. Thus, having an uptime service of 99.9% is essential. Such expectations present a significant risk if the server goes down due overload or a bugs.

2.2 Demonstration Plan

For proof of concept we will be implementing a simple algorithm as a prototype to track the movement of a dumbbell in a hand. As the hand moves up and down the program will count the repetition of bicep curls.

We will be using the camera of our laptops to demonstrate this plan. The scope of this plan is limited to demonstrating that the significant risks can be overcome.

As a user does bicep curls in front of the laptop our prototype will count repetitions of the exercise. This will show that the algorithm accurately counts the repetitions. This prototype will also give a good understanding of the minimum resolution of an image that is required for the final product to actually detect objects. Moreover, it will provide us an estimate of how far away objects can be from the camera to be detectable.

The prototype will also include a backend server conencted to a database with a well-designed schema. The backend server will be extensively load-tested by hitting it with mock requests to demonstrate that it can handle stress and overloads due to computation and multiple requests.

2.3 Proof of Concept Test

The proof of concept is given in test case format to adhere with the presentation of the other tests in this document.

Test 2.3.1:	**Proof of Concept**
Description:	Tests whether significant risk to the completion of the project can be overcome.
Type:	Proof of Concept (manual)
Tester(s):	Gym users and/or prototype developers
Pass:	Successful detection of objects in different scenarios and a stable backend with little processing time and 99.9

3 System Testing

Each component of the project will be tested using black-box techniques through automated unit testing and manual user input. Test coverage tools will also be utilised to identify portions of code that lack unit tests. After each component is tested, they will be verified as one integrated system. The individual components to be tested are object detection, counting algorithms, database and backend server operations, and the mobile application interface; the specific test cases are defined in this section.

3.1 Object Detection

Manual user testing will be used as the primary method for testing object detection using a camera. Equivalent unit tests with mocked input values will also be used to test the software. The following test cases relate to the software's ability to recognize an object and to retrieve its location.

Test 3.1.1:	**Detect a non-moving marker**
Description:	Tests if objects are detected when they are not moving
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	Camera is powered on and connected to software
Input:	A stationary object within the camera's view
Output:	A value representing the object's position
Pass:	Position for marker is detected

Test 3.1.2:	**Detect a moving marker**
Description:	Tests whether multiple position values representing object's path can be tracked
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	Camera is powered on and connected to software
Input:	Simulate an object moving within the camera's view
Output:	A set of values representing the object's path
Pass:	Multiple positions for marker are detected

Test 3.1.3:	**Do not detect when there are no markers**
Description:	Tests whether the camera software can distinguish between marker and the environment
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	Camera is powered on and connected to software
Input:	A view in which there is no marker
Output:	No marker position
Pass:	No positions for markers are detected

Test 3.1.4:	**Handle multiple markers in a single frame**
Description:	Tests whether the camera software can detect and distinguish between multiple markers in a single frame
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	Camera is powered on and connected to software
Input:	A view wherein more than one markers are present
Output:	Sets of position values for each marker
Pass:	The number of sets of position values match the number of markers

3.2 Counting Algorithm

Unit testing will be the main method of testing the movement-counting algorithms. The following test cases relate to the software's ability to detect patterns in the object's movement and record corresponding repetition and set counts.

Test 3.2.1	**Count one repetition of each defined movement**
Description:	Tests whether a repetition based on changes in object's position value over time can be inferred
Type:	Unit (dynamic, automated), Manual
Tester(s):	Software developers
Initial State:	Object is not moving
Input:	Object moves to perform one repetition
Output:	Repetition count
Pass:	Repetition count is equal to one

Test 3.2.2	**Count multiple repetitions of each defined movement**
Description:	Tests if multiple exercise repetitions based on object's position values over time can be counted
Type:	Unit (dynamic, automated), Manual
Tester(s):	Software developers
Initial State:	Object is not moving
Input:	Object moves to perform multiple repetitions
Output:	Repetition count
Pass:	Repetition count is equal to the number of performed repetitions

Test 3.2.3	**Determine the end of a set**
Description:	Tests to see when a set is considered finished based on whether the object has stopped moving for a given amount of time
Type:	Unit (dynamic, automated), Manual
Tester(s):	Software developers
Initial State:	Object is moving
Input:	Object stops moving for at least one minute
Output:	Set count
Pass:	The set count increased by 1

3.3 Database Manipulation and Backend Server Operations

Unit and manual testing will be used to verify performance and scalability of the application's database and server. The following test cases relate to the software's ability to adapt to different loads, interruptions and failures during typical execution.

Test 3.3.1:	**Store data for single user**
Description:	Tests if server is able to store data for a single user on the database
Type:	Unit (dynamic, automated)
Tester(s):	Software developers
Initial State:	The database contains a schema for the application
Input:	Server inserts a row into database
Output:	Database status notification
Pass:	The database contains the new data inserted

Test 3.3.2:	**Load data for single user**
Description:	Tests if server is able to load data for a single user from the database
Type:	Unit (dynamic, automated)
Tester(s):	Software developers
Initial State:	The database contains rows of data pertaining to user
Input:	Server is used to query user data
Output:	Data is returned
Pass:	The data returned matches what is expected

Test 3.3.3:	**Store data for multiple users**
Description:	Tests whether the database and server are able to handle large loads of insert requests
Type:	Unit (dynamic, automated)
Tester(s):	Software developers
Initial State:	Database has enough space for at least 200 more entries
Input:	Scripts that add values (200 in total) to database are executed at the same time
Output:	Database status notification for each insertion
Pass:	All insertions succeed and database values are inserted as expected

Test 3.3.4:	**Load data for multiple users**
Description:	Tests whether the database and server are able to handle large loads of select requests
Type:	Unit (dynamic, automated)
Tester(s):	Software developers
Initial State:	Database contains at least 200 rows added by a script
Input:	Scripts that retrieve values from database and load them on the server are executed at the same time
Output:	Data is returned
Pass:	The data returned matches the 200 rows added by a script

Test 3.3.5:	**Database operation interrupted by network outage**
Description:	Tests how the system mitigates unexpected network disconnection and recovers data
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	Database statement is ready for execution
Input:	Kill server's connection to database
Output:	Failure notification, data is not sent or received
Pass:	Server intermittently retries operation until timeout or a network connection is restored

3.4 Mobile Application Interface

Manual testing will be used to verify usability of the software's mobile user interface; other aspects of the mobile app is tested during system integration. The following test cases relate to the mobile app's ability to react to user manipulation and various errors.

Test 3.4.1:	**Scan QR code**
Description:	Test whether the scanning software recognizes image input
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	A QR code that is associated with a weight machine and a phone with scanner software
Input:	Phone scans QR code
Output:	Scanner notification
Pass:	Phone scanner notification indicates successful recognition

Test 3.4.2:	**Scan QR code that is not recognized by the application**
Description:	Test whether the scanning software distinguishes between QR codes associated with gym equipment and all other QR codes
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	A QR code that is not associated with a weight machine and a phone with scanner software
Input:	Phone scans QR code
Output:	Scanner notification
Pass:	Phone scanner notification indicates successful scan but failed recognition

Test 3.4.3:	**Scan QR code that is recognized by the application, but is distorted**
Description:	Test whether the scanning software can recognize QR codes after some wear
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	A worn out QR code that is associated with a weight machine and a phone with scanner software
Input:	Phone scans QR code
Output:	Scanner notification
Pass:	Phone scanner notification indicates successful recognition

Test 3.4.4:	**Pop up error messages**
Description:	Tests whether the app prevents unwanted actions when user enters wrong credentials, scans unrecognized QR or tries to register with a duplicate email
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	Mobile login/registration and QR code scanner are implemented
Input:	Perform unexpected action as specified in *Description*
Output:	Pop up error message
Pass:	Error message appears and prevents access to other pages until it is acknowledged or issue is resolved

Test 3.4.5:	**Update visualizations of user's performance when requested**
Description:	Tests whether user performance charts are updated as soon as they log new workouts
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	A workout has just been logged
Input:	Navigate to the data visualizations page
Output:	Graphs and actionable data is presented
Pass:	The data includes the latest workout logged by the user

3.5 System Integration

Interactions between the first four systems will be tested using manual testing. The specific test cases are outlined in this subsection.

Test 3.5.1:	**Camera capture should not start until after a QR code is scanned and user logs in**
Description:	Tests whether the mobile app, server and camera are able to communicate and control access
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	Mobile app, camera and server are connected via a network, and user has logged in but has not scanned QR code
Input:	Object moves
Output:	Set of position values
Pass:	No position values are recorded

Test 3.5.2:	**Require user registration before using the app**
Description:	Tests whether app prevents data from being inserted into database unless it is associated with a user
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Initial State:	Mobile app login and registration page are implemented
Input:	Attempt to see the menu and other functions
Output:	Error message
Pass:	Error message appears and prevents access to other pages until user logs in or registers

Test 3.5.3:	**Logged workouts should be instantaneously available as visualizations on mobile app**
Description:	Tests how fast the camera sends data to server and back to the user's phone
Type:	Unit (dynamic, automated)
Tester(s):	Software developers
Initial State:	Workout logging has finished and set/rep counts are ready for transmission
Input:	Mobile app is used to save the session
Output:	Data visualizations are generated for viewing
Pass:	There is a maximum of 200 ms delay between saving a workout session and having the statistics included in the data visualizations

Test 3.5.1:	**Mobile app should cache a user's latest statistics**
Description:	Tests whether the system is able to recover user's latest session in case of unexpected failures that prevent data from being uploaded to server for storing in the database
Type:	Unit (dynamic, automated)
Tester(s):	Software developers
Initial State:	Workout logging has finished and set/rep counts are ready for transmission
Input:	Mobile app is used to save the session but a power/network interruption is triggered
Output:	Database status
Pass:	Session saving fails and database is not updated, but the data is cached on the mobile phone

4 Requirements Testing

4.1 Functional Requirements Testing

The following tests will be performed to ensure adherence of the functional requirements stated in the software requirements specification.

Test 4.1.1:	**Input workout data testing**
Description:	To make sure the user can manually log workouts using the PiSonal Trainer App.
Type:	Functional (dynamic, manual)
Tester(s):	Users
Pass:	Users can log their workouts manually

Test 4.1.2:	**Output of workout statistics testing**
Description:	To make sure the user can view their workout statistics using the PiSonal Trainer App.
Type:	Functional (dynamic, manual)
Tester(s):	Users
Pass:	Users can view their workout statistics

Test 4.1.3:	**Getting fitness data from the server testing**
Description:	To make sure the PiSonal Trainer app can receive fitness data from the backend server.
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	The PiSonal Trainer app can receive fitness data from the server in the background.

Test 4.1.4:	**Requesting fitness data from the server testing**
Description:	To make sure the PiSonal Trainer app can send a request to receive fitness data to the backend server.
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	The PiSonal Trainer app can send a request to receive fitness data to the server in the background.

Test 4.1.5:	**Transmitting repetition count from the camera to the server esting**
Description:	To make sure the camera software is able to transmit the count of repetitions to the backend server.
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	Camera software is able to transmit the count of repetitions to the backend server

Test 4.1.6:	**Reading data from the database testing**
Description:	To make sure the backend server can extract/read data from the database on demand.
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	Backend server can extract/read data from the database on demand.

Test 4.1.7:	**Writing data to the database testing**
Description:	To make sure the backend server can insert/write data to the database on demand.
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	Backend server can insert/write data to the database on demand

4.2 Non-Functional Requirements Testing

The following tests will be carried out to ensure completeness and accuracy of the non-functional requirements given in the software requirements specification.

Test 4.2.1:	**Textual content on the screen check**
Description:	The mobile application uses minimal textual content on each screen
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	Textual content on the application is only 20%

Test 4.2.2:	**Application simple to use**
Description:	The mobile application must be simple to use
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	The time taken for users to use the application must be less than the time for manual entry of their work out.

Test 4.2.3:	**Capacity check**
Description:	The mobile application must be used concurrently by multiple users on their devices
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	The applications output must not be impacted by the number of users currently using it.

Test 4.2.4:	**Scalability and Extensibility**
Description:	The camera should be able to track at least one user at any time
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	The camera gives the accurate count of moving objects in the screen (at least one when there is someone working out).

Test 4.2.5:	**Operating System support**
Description:	The mobile application can be accessible on both Android and IOS
Type:	Functional (dynamic, manual)
Tester(s):	Software developers
Pass:	The application is usable and passes on each platform.

4.2.1 User Experience Testing

The following non-functional requirements will be tested via a user experience survey administered to a testing group.

Test 4.2.1.1:	**Camera is non-obtrusive**
Description:	The camera blends in with the environment and does not negatively affect the aesthetic of the gym.
Type:	Functional (dynamic, manual)
Tester(s):	Testing group
Pass:	The camera is approved by the users confirming it does not negatively impact the aesthetic of the gym.

Test 4.2.1.2:	**Personalization and Internalization**
Description:	The product shall become the user's preferred tracking method after the trial period.
Type:	Functional (dynamic, manual)
Tester(s):	Testing group
Pass:	After a trial period of one week, the user should agree that they would rather se this product than manually track their muscle training progress.

Test 4.2.1.2:	**Personalization and Internalization**
Description:	The product shall become the user's preferred tracking method after the trial period.
Type:	Functional (dynamic, manual)
Tester(s):	Testing group
Pass:	After a trial period of one week, the user should agree that they would rather se this product than manually track their muscle training progress.

Test 4.2.1.3:	**Learning Period**
Description:	The product should be easy to learn by users who have never tracked their workout before.
Type:	Functional (dynamic, manual)
Tester(s):	Testing group
Pass:	fter a trial period of one week, the rate of errors that the users make while using the application should decrease to at most 5%.

5 Timeline

The timeline below is structured roughly to match the anticipated chronology of testing. Note: Dates are anticipated to change.

Task	**Responsible Party**	**Test Creation Date**	**Test Completion Date**
Completion of Proof of Concept demo	Development team	15/11/2016	20/11/2016
Mobile application output data test cases	Birunthaa	01/12/2016	05/12/2016
QR code identification test cases	Micaela	05/01/2017	12/01/2017
Image recognition test cases	Simar	09/01/2017	16/01/2017
Sets and repetitions test cases	Simar	18/01/2017	27/01/2017
Database retrieval and manipulation test cases	Micaela	01/02/2017	10/02/2017
Mobile application data from the server test cases	Simar	13/02/2017	20/02/2017
User experience survey phase I test cases	Birunthaa	22/02/2017	06/03/2017
User experience survey phase II test cases	Micaela	15/03/2017	27/03/2017

Table 5: Timeline of testing