System Test Documentation

for

Hypnos

Version 1.0 approved

Prepared by

Ozge Lule
Esref Ozturk
Baris Ozkuslar
Oguzhan Unlu

Newline

1. Introduction

1.1. Problem Definition

This software product will be built in five main parts, and these parts will have their own design fashions. These parts are wearable, fixed, cloud, mobile application and web respectively. The project aims to develop a sleep tracking and feedback system so that, designed physiological sensing system can provide reliable vital signs measurements and incorporating real-time decision support. The project can be used to track a user for a period of time and to detect immediate changes on individual that can be sign of an emergency. Modularity of this project will also provide different configurations for different targets.

1.2 Purpose and Scope

The purpose of this document is to provide the test cases of the Hypnos project. It defines the objective, scenario, expected outcomes and procedural requirements for each test case. It also includes a table showing which test case is related to which one. The software will be tested using guidance of this document. Although it covers all the test cases specifically in detail, a little portion of the details is subject to change in test phase.

1. Details for system test plan

This section describes the specific items to be tested at different levels and provides a Test Traceability Matrix that links the items to be tested with the requirements.

1.1. Test Items and Their Identifiers

Since the system consists of five subsystems which can be identified as components or levels, each subsystem is an object of tests. Integration of these components shall be included to tests as well.

1.2. Features to be Tested

Software in Arduino Board is going to be tested deployed with different combinations of sensors. Bandwidth of Xbee module and serial port of Arduino will be in the features to be tested.

Database in cloud will be tested for different cases. Data collection software will be tested connected to the database. Analysis tests are going to be held with differentiating conditions.

Web application will have its individual tests. Phone application tests will be held for notifications and availability.

Protocols between components of system are subjects to the test. System-wide integration can be included in features to be tested as well.

1.3. Features not to be Tested

Device dependency tests for hardware architectures other than already stated ones will be omitted. Arduino board is able to employ more than three sensors but no additional sensors will be tried. Rationale behind that is lack of available hardware components and undecided design.

Third-party open source libraries usually conduct their own tests, so tests for those libraries will only take place in integration and unit tests of software implemented.

User stress tests for server will not exceed a certain limit because, for a brand new product, hardware requirements for scaling are subjects of successive test documents. Security tests will also be postponed for this version of document.

For each level, tests will be held using simulated inputs flow originated from lower levels since real world inputs are not sufficiently required.

1.4. Approach

Every level of test have its individual approach. Detailed information can be found in chapter 3.

1.5. Item Pass/Fail Criteria

There are several test categories which specify fail criteria for different tests:

- Integrity Checks: If integrity of values are not preserved during relay operations, test is failed.
- Connectivity Tests: If packages or connection is lost, test is failed.
- Analysis Accuracy Checks: If analysis accuracy is lower than 70%, test is failed.
- Integration Tests: For integration tests, predecessor component shall be available for test status to be passed, otherwise it is failed.
- Performance Tests: There is no specification of delays that is not negligible in this version of document.
- Unit Tests: Unit crash or permanent failure means that test is failed.

1.6. Suspension Criteria and Resumption Requirements

During integration tests, if one lower level fails, tests associated with the lower level are invalidated. In such a case, tests for both levels shall be repeated. For system integration tests, any type of component crash requires this test to be suspended. Unit tests for crashed item shall be conducted again.

2. Test Management

Best way to test the whole system is creating subsystem tests. This approach is superior to complete system testing because subcomponents are pretty big already. Their interactions and communications with each other should be well defined and work steadily. Finally the complete system integration and testing must be performed. Thus the workload of testing schema is as follows:

2.1. Testing a sub component within itself

This process ensures that the component is working without any interaction by any other sub components. To be able to conduct this test procedure for the wearable subsystem, in addition to cores of the hardware such as micro-controller, pulse sensor, temperature sensor, accelerometer, we need a serial connection to a workstation. This will allow us to be able analyze & debug the embedded code much more efficiently. On the other hand, remaining subsystems do not depend on specialized hardware/software parts. Nevertheless, their unit tests are written by corresponding programming languages (Python / C) and deployed individually on them.

2.2. Testing the communication protocols and interactions between adjacent sub-system components

Purpose of this part is to stabilize inter subsystem communications. Techniques are mostly composed of erroneous situation generation. As defects reveals the real life situations more realistically. Sending an invalid JSON object from fixed component to cloud server might be an example. In fact, when exactly an exception occurs and how subsystems reacts those are the heart of this test activity.

2.3. Integration of the complete system and testing

This is the final test scenario and describes how will the complete system work when it is deployed in real life. Thus, conducting stress tests for the system is necessary obviously. To make stress testing effective, several duplicate simulators will be designed and executed. Those

simulators mimic realistic behaviors of the subsystems. As an example, wearable component simulators generate data for the fixed component.

3. System Test Levels

3.1. Wearable Component Level

3.1.1. Wearable Test 1

Test Case Identifier	WEARABLE-TEST-01
Objective	Correctness of temperature sensor measurements
Scenario	Heating and cooling the temperature sensor via external inputs
Input	Holding the analog temperature sensor for a while and blowing the cold air to the sensor afterwards.
Outcome	Temperature output of the micro-controller smoothly rises first and falls later on.
Requirements	Avoidance of extraordinary noise interruption

3.1.2. Wearable Test 2

Test Case Identifier	WEARABLE-TEST-02
Objective	Correctness of pulse sensor measurements
Scenario	Breathing heavily when equipped with pulse sensor on the finger
Input	Increasing rate of the person's heart rate
Outcome	Heart beat per minute data output of the micro-controller rises
Requirements	Equipping the pulse sensor properly so that no extra environmental light disturbs the led of the sensor

3.1.3. Wearable Test 3

Test Case Identifier	WEARABLE-TEST-03
Objective	Correctness of accelerometer measurements
Scenario	Tilting, pushing, pulling and moving the accelerometer
Input	Changing accelerometer data with respect to the 3 axises
Outcome	Rapid changes for the accelerometer output of the micro-controller
Requirements	Calibration and sensitivity settings should be properly configured for the digital accelerometer sensor

3.1.4. Wearable Test 4

Test Case Identifier	WEARABLE-TEST-05
Objective	Xbee connection establishment

Scenario	Bringing the peer device closer and send it away later on
Input	Increasing the power level of the xbee signal, then decreasing it.
Outcome	First, micro-controller sets up a xbee connection with the peer and sends measured data to it. Whenever the peer is far away from the micro-controller, connectivity drops. The whole scenario is looped again when the peer is getting closer.
Requirements	Necessary configurations on Xbees prior to connection

3.2. Fixed Component Level

3.2.1. Fixed Component Test 1

Test Case Identifier	FIXED-TEST-02
Objective	Getting Sensor Data
Scenario	Sensor data is obtained from wearable device.
Input	Data queue of a sensor periodically enqueued with new sensor data
Outcome	Periodically obtaining the same data from queue concurrently
Requirements	A simulator which generates sensor data or wearable device input is required

3.2.2. Fixed Component Test 2

Test Case Identifier	FIXED-TEST-02
Objective	Regular Sender Periodic Send Operation
Scenario	Given a period by configuration file, regular sender thread works every N seconds to wipe out the output sensors, pack the sensor data and send them to cloud.
Input	Regular sensor data.
Outcome	Threads work concurrently and without starvation
Requirements	FIXED-TEST- 01 should be successful

3.2.3. Fixed Component Test 3

Test Case Identifier	FIXED-TEST-03
Objective	Urgent Sender Immediately Send Operation
Scenario	In case of emergency identified by analyzer, urgent sender thread sends the emergent situation data immediately to the server.
Input	Urgent sensor data.
Outcome	If network delays are ignored, there is no delay when sending data to cloud

Requirements	FIXED-TEST- 01 should be successful

3.3. Cloud Component Level

3.3.1. Cloud Component Test 1

Test Case Identifier	CLOUD-TEST-01
Objective	To test the register component of the system
Scenario	User sends register info via mobile application
Input	name, surname, address,phone email, password, device id
Outcome	A new user is created in the system.
Requirements	The mobile application should be started

3.3.2. Cloud Component Test 2

Test Case Identifier	CLOUD-TEST-02
Objective	To test the login component of the system with valid data
Scenario	User sends login info via mobile application
Input	username, password
Outcome	The user is logged in to the application.
Requirements	The mobile application should be started and user must be registered.

3.3.3. Cloud Component Test 3

Test Case Identifier	CLOUD-TEST-03
Objective	To test the logout functionality.
Scenario	User sends logout info via mobile application
Input	None
Outcome	The session should be invalidated.
Requirements	The mobile application should be started and user must be logged in.

3.4. Mobile Application Component Level

3.4.1. Mobile Test 1

Test Case Identifier	MOBILE-TEST-01
Objective	Testing the Emergency Notification Systems' Registration procedure
Scenario	When a user installs the mobile application and runs it, he or she will be required to enter an emergency contact number
Input	Providing the emergency contact number

Outcome	Emergency contact number will be registered to cloud automatically
Requirements	-

3.4.2. Mobile Test 2

Test Case Identifier	MOBILE-TEST-02
Objective	Testing the Emergency Notification
Scenario	An event, such as ‘dangerously high temperature’, occurred that needs a notification to be sent to emergency contact. Then for such cases, our server sends messages to emergency contacts.
Input	-
Outcome	Emergency contact will be texted.
Requirements	-

3.5. Web Application Component Level

3.5.1. Web Test 1

Test Case Identifier	WEB-TEST-01
Objective	Testing display of real time sensor values
Scenario	User opens dashboard and real time sensor values are shown on line charts
Input	User logins to web application
Outcome	User sees real time sensor values
Requirements	-

3.6. System Integration Level

3.6.1. System Test 1

Test Case Identifier	SYSTEM-TEST-01
Objective	Testing the whole data flow from sensor to cloud
Scenario	Sensors on hardware are working and sampling data with an arbitrary rate.
Input	Sensor’s Input
Outcome	Data is transmitted to the cloud
Requirements	User has sensor hardware, fıxed device working. Cloud is running.