

# ***My Diet***

## **Test Strategy (TS)**

---

Version 1.0  
7/10/13

### **Primary Authors:**

Ana G. Hernandez Reynoso

Alba A. Magallanes García

Julio César Roa Gil

Diego R. Torres Lafuente

Victor M. Rodriguez Bahena

### **Approvals:**

<b>Role</b>	<b>Signature</b>	<b>Date</b>
Project Manager	Victor M. Rodríguez	7/10/13
QA Manager	Ana G. Hernández R.	7/10/13

# Revision History

Version	Date	Author's Full Name	Reason for Changes

# Abstract

Software testing is a critical job. With the size and complexity of today's software, it's imperative that software testing be performed professionally and effectively. Too much is at risk. In the following document, you'll learn more about the big picture of how software testing fits in "My Diet" App project. This document describes the strategy used to test the application with the name "My Diet". This knowledge is critical to helping you apply the specific test techniques you need.

This document includes the external results that we will use to test the application. The test scope will be on both orientations: static and dynamic tests and white and black box testing. This document will contain: assumptions when testing the application, list of test cases, list of features to be tested, approach to use for testing, list of deliverables that need to be test, test approaches .and tests scope.

The main focus will be the functionality and usability for this Test Strategy Document. Nonetheless other important areas will be tested as well: performance, reliability and maintenance.

One of the challenges will be to achieve the fully test of it, find all the bugs, and assure that the software is perfect. Unfortunately, this isn't possible, even with the simplest programs, due to four key reasons:

- The number of possible inputs is very large.
- The number of possible outputs is very large.
- The number of paths through the software is very large.
- The software specification is subjective. You might say that a bug is in the eye of the beholder.

One approach we will follow is the automation of test cases and random generation of user data input. Although the validation of My Diet App seems to be a difficult task it will be performed with high quality standards and always trying to satisfy the costumer's requirements.

# Table of Contents

## **1 Introduction**

- 1.1 Purpose and Scope of this Document
- 1.2 Definitions
- 1.3 Team
- 1.4 Related Documents
- 1.5 Test Deliverables
- 1.6 Roles and Responsibilities
  - 1.6.1 Project Manager
  - 1.6.2 System Analyst Lead
  - 1.6.3 Software Design Lead
  - 1.6.4 Test Lead
  - 1.6.5 Quality Assurance Lead

## **2 Test Scope**

- 2.1 Product Requirements
  - 2.1.1 Usability
    - 2.1.1.1 Navigation
    - 2.1.1.2 Discrimination
    - 2.1.1.3 Touch screen use
    - 2.1.1.4 Screen repainting
    - 2.1.1.5 Consistency
    - 2.1.1.6 UI & Graphics
    - 2.1.1.7 Visual Quality
    - 2.1.1.8 Notification and error messages
    - 2.1.1.9 Multiple format input handling
    - 2.1.1.10 Spelling errors
  - 2.1.2 Reliability
    - 2.1.2.1 Self-monitoring caloric intake
    - 2.1.2.2 Goal setting
    - 2.1.2.3 Non-food rewards
    - 2.1.2.4 Summary of reliability features
  - 2.1.3 Performance
    - 2.1.3.1 Performance Requirements
    - 2.1.3.2 Battery Life

- 2.1.3.3 Responsiveness
- 2.1.3.4 Suspend / Resume
- 2.1.3.5 Multi-tasking and effect on other functions
- 2.1.3.6 User state / Application state
- 2.1.3.7 Resource sharing and database
- 2.1.3.8 Other programming Do's and Don'ts
- 2.1.4 Supportability
- 2.1.5 Maintainability
- 2.1.6 Manufacturability
  - 2.1.6.1 Installation
  - 2.1.6.2 Application permissions
  - 2.1.6.3 Long launch time
  - 2.1.6.4 Memory and file storage during run
- 2.1.7 Testability
- 2.1.8 Extensibility
- 2.2 Feature Matrix/ Requirement Matrix
- 2.3 Use Case Matrix
- 2.4 Configuration Matrix
- 2.5 Relevant Policies, Process, and Guidelines to follow
- 2.6 Priority Matrix

### **3 Test Approaches**

- 3.1 Test Approaches Matrix
- 3.2 Test Case Design Matrix
- 3.3 Test Tools
- 3.4 Risk Management
- 3.5 Defect Management
- 3.6 Logistics
- 3.7 Test Management
- 3.8 Additional Standards and Procedures

# 1. Introduction

## 1.1 Purpose and Scope of this Document

This document describes the strategy used to test the application with the name “My Diet”, the resources needed, the testing environment and the limitations of the tests. It includes: assumptions when testing the application, a list of test cases, a list of features to be tested, an approach to use for testing, and a list of deliverables that need to be tested.

## 1.2 Definitions

Term	Definition
SRS	Software Specification Requirement
UI	User Interface
API	Application programming interface
OS	Operating System
GPS	Global Positioning System
SDK	Software Development Kit
ADT	Android Development tools
IDE	Integrated Development environment
JAR	Java archive

## 1.3 Team

- A. Alba A. Magallanes García
- B. Julio César Roa Gil
- C. Víctor M. Rodríguez Bahena
- D. Diego R. Torres Lafuente
- E. Ana G. Hernández Reynoso

## 1.4 Related Documents

Author	Version	Title	Location
A,B,C,D,E	1	Vision and Scope document	<a href="https://docs.google.com/document/d/1ug8xIAaEgF4Rx3k61YfGOYLqDfiU84tSqJ1oHvA0Ifg/edit?usp=sharing">https://docs.google.com/document/d/1ug8xIAaEgF4Rx3k61YfGOYLqDfiU84tSqJ1oHvA0Ifg/edit?usp=sharing</a>
A,B,C,D,E	1	Software Specification Requirement (SRS)	<a href="https://docs.google.com/document/d/1DPqJ8cnn9RcbocG3E-alXEwPtVa0eJgneKgqcUFwQGg/edit?usp=sharing">https://docs.google.com/document/d/1DPqJ8cnn9RcbocG3E-alXEwPtVa0eJgneKgqcUFwQGg/edit?usp=sharing</a>

## 1.5 Test Deliverables

- Test strategy version 1 - week 40
- Test strategy version 2 - week 44
- Test strategy final - week 47

## 1.6 Roles and Responsibilities

### 1.6.1 Project Manager

Victor M. Rodríguez Bahena (C): determines objectives, schedule and resource budgets, design a software project management plan, create and sustain focused and motivated teams, determine the team's procedures, reporting systems and communication infrastructure, accomplish project objective within time and budget, monitor performance against the plan, resolve technical conflicts, control changes in the project, report on project activities to upper management, keep the client informed and committed, contribute to the team members and performance approval.

### 1.6.2 System Analyst Lead

Diego R. Torres Lafuente (D): assisting with the business case, making high-level feasibility studies, gathering of the requirements, designing and/or reviewing test cases, processing change

requests, tracing the requirements during implementation, manage project scope, acceptance installation, and deployment.

### **1.6.3 Software Design Lead**

Julio César Roa Gil (B): concerned with facets of the software development process: research, design, development and testing software. May take part on the design, programming, or software project management, contribute to the overview of the project on the application level rather than component-level or individual programming tasks.

### **1.6.4 Test Lead**

Alba A. Magallanes García (A): prepare the software test plan, review the test case document, system, preparing integration and user acceptance, analyze requirements, keep track of the new requirements, forecast project future requirements, arrange the hardware and software requirement for the test setup, develop and implement test plans, escalate the issues about project requirements to project manager, escalate issues in the application to the client.

### **1.6.5 Quality Assurance Lead**

Ana G. Hernández Reynoso (E): assign task to all testing team members and ensure that all of them have sufficient work in the project, send status report, frequent status report, point of contact between developers and testers for iterations, testing and deployment activities, track and report upon testing activities, assist in performing any applicable maintenance, document, implement, monitor, and enforce all processes and procedures for testing, review reports prepared by test engineers, check for timely delivery of different milestones.



## 2. Test Scope

### 2.1 Product Requirements

According to the Software Requirements Specification Document we have 3 main features to cover:

- Feature 1: Self-monitoring of caloric intake
  - Track calories.
  - Add new meals.
- Feature 2: Goal setting
  - Determine ideal caloric intake.
- Feature 3: Non food rewards
  - Report balance of ingested meals.

Based on these main features we can design.

#### 2.1.1 Usability

The usability is better described in the section 2.3 User Classes and Characteristics. This section specifies the classes of end users:

- Patients diagnosed with obesity or overweight: most of these users will have a nutritionist caloric intake recommendation. They will use the app on a daily basis and must enter all their meals. The app will be used as a tool for them to lose weight by tracking their caloric intake in each meal. This class will be the main market of the app.
- Patients with no weight problems who want to monitor their intake: the second class of users doesn't have weight problems, and use the app only to stay in shape and monitor their consumption. They could use the app on a daily basis or just as informative tool to know if they're eating correctly.
- Summary of usability features:
  - Use the app on a daily basis and must enter all their meals.
  - Used as a tool for them to lose weight by tracking their caloric intake in each meal.

- Use the app only to stay in shape and monitor their consumption.
- Use the app on a daily basis or just as informative tool to know if they're eating correctly.

Some other good requirements to be taken into consideration are described in the document “Best Practice Guidelines for producing high quality mobile applications” (Aqua, 2013).

#### **2.1.1.1 Navigation**

- Only hide the status bar where the application's user experience is better without it.
- Use native icons consistently. The application should not replace a system icon with a completely different icon if it triggers the standard UI behavior.
- Don't override the platform menu button. Instead, put menu options behind the menu button.
- Respect user expectations for navigation. Where the platform provides a Back button, it should always navigate back through previously seen screens
- Always support trackball navigation if the platform supports it. Understand your navigation flow when the platform permits the entry point to be a notification or widget.
- The application must support the standard system navigation, and must not make use of any custom on-screen prompts or buttons that attempt to replace system navigation functions.
- On Android all dialogs must be dismissible using the Back button. If the platform supports a Home button, pressing this at any point must navigate to the expected default or home screen on the device.

#### **2.1.1.2 Discrimination**

Don't make assumptions about screen size, resolution, orientation or input. Never hard-code string values in code. Use Relative Layouts and device independent pixels. Optimize assets for different screen resolutions, and use reflection to determine what APIs are available.

#### **2.1.1.3 Touch screen use**

For applications used in a touch screen device without stylus, on-screen elements should be of sufficient size and responsiveness to provide a good user experience.

#### **2.1.1.4 Screen repainting**

The application screens should be correctly repainted, including cases when edit boxes and dialog boxes are dismissed. Also, there should be no blinking of moving objects and background. If the application objects overlap they must still render correctly.

#### **2.1.1.5 Consistency**

The application UI should be consistent and understandable throughout, e.g. displaying a common series of actions, action sequences, terms, layouts, soft button definitions and sounds that are clear and understandable.

#### **2.1.1.6 UI & Graphics**

The application should support both landscape and portrait orientations if possible. The application should expose largely the same features and actions in both orientations, and preserve functional parity

#### **2.1.1.7 Visual Quality**

The application should display graphics, text and other UI elements without noticeable distortion, blurring or pixilation.

#### **2.1.1.8 Notification and error messages**

Notifications should follow the design guidelines for the platform. Multiple notifications should be stacked into a single notification object where the platform supports this. Notifications should only be persistent if related to on-going events (such as music playback or a phone call).

Notifications must not contain advertising or content unrelated to the core function of the application, unless the user has made a conscious choice to accept such data via an opt-in option offered by the application.

The application should only use notifications to:

- Indicate a change in context relating to the user personally (such as an incoming message)
- Expose information or controls relating to an on-going event (such as music playback or a phone call).

Any error messages in the application should be clearly understandable. Error messages should clearly explain to a user the nature of the problem, and indicate what action needs to be taken (where appropriate).

#### **2.1.1.9 Multiple format input handling**

Where the device and application can accept input in multiple formats (e.g. external touch screen / external keypad / internal touch screen / internal keypad / QWERTY layout / 12-key layout and others), it should work correctly with all supported input methods.

#### **2.1.1.10 Spelling errors**

The Application should be free of spelling or language errors unless they are part of a deliberate design concept.

### **2.1.2 Reliability**

The reliability is described more in detail on section 3 (System Features) of the System Requirements Specification Document. The following is a brief summary of the features we are planning to release:

#### **2.1.2.1 Self-monitoring caloric intake**

- Track calories: the user is able to add a list of all the meals ingested during the day and the app calculates the number of calories ingested.
- Add new meals:
  - If an ingredient is not present in the default database, the user can add it by indicating the number of calories present in it.
  - If a meal is not present in the database, the user can add it by either indicating the total calories in it; or indicating the meal ingredients from the actual database.
  - End users can not share new meals or ingredients among each other

#### **2.1.2.2 Goal setting**

The user is able to set a daily caloric intake range according to the instructions given to them by their nutritionist or calculated by the app. If the user does not set a goal, an 1800 - 2600 calorie intake goal is set by default.

#### **2.1.2.3 Non-food rewards**

The app provides a feedback system to the user, promoting positive reinforcement by a point system in which the app gives points to the user everyday that the goal is met; on the other hand it removes points when the opposite happens.

If the caloric intake exceeds the goal, more points are removed from the user; if the caloric intake doesn't reach the inferior limit, fewer points are removed.

#### **2.1.2.4 Summary of Reliability Features**

- User will be able to add a list of all the meals ingested during the day. These features should always work.
- The app calculates the number of calories. The computation should be accurate.
- Add new food indicating the amount of calories or indicating the meal ingredients from the actual database. Database connection should always work
- The user is able to set a daily caloric intake range according to the instructions given to them by their nutritionist or calculated by the app. User should always be able to add the caloric intake range. The caloric intake calculated by the app should always be accurate.
- The app provides a feedback system to the user. The computation should be accurate according to the caloric intake of the end user.

#### **2.1.3 Performance**

These requirements are described more deeply in section 5 (Other Nonfunctional Requirements) of the Software Requirements Specification Document. The following is a brief summary of those requirements:

##### **2.1.3.1 Performance Requirements**

As on every mobile application, having a good performance is one of the most important aspects to care about in order to give a good user experience.

- Every app operation (database query, data storing or any calculation) should take less than 1 second to process.

### **2.1.3.2 Battery Life**

The app will use the Android Battery Management System and allow the power-saving features of the device to be used, including sleep functions for the screen and the device itself. The management of connectivity can have a huge impact on battery life

### **2.1.3.3 Responsiveness**

Always update the user on progress. Render the main view and fill in data as it arrives. Always respond to the user input within 5 seconds. Users perceive a lag longer than 110 - 200 ms adversely.

In Android with Strict Mode enabled, no red flashes (performance warnings) should be visible when exercising the app, including during game play, animations and UI transitions, or any other part in the app.

### **2.1.3.4 Suspend / Resume**

Where an OS environment supports 'suspend / resume', ensure that the application suspends and resumes at a point that does not impair the user experience.

### **2.1.3.5 Multi-tasking and effect on other functions**

In a multi-tasking environment, remember to release used resources or functionality for other applications to use when not in use by that application.

An application should correctly handle situations where following user input, or some external event (e.g. a phone call), it is switched to the background by the terminal. Upon returning to the foreground the application should resume its execution correctly.

While in the background, the application should not intrude on any way with the operation of other applications or handset functions, unless there is an explicit design requirement to do so and it is clearly explained in an accessible help file.

When `pauseApp()` or `hideNotify()` is called by the system, the My Diet should do the following:

1. Pause the application.
2. Save application state.
3. Release any resources the app won't need while paused (like sound resources).

Upon the system calling `startApp()` should reallocate the resources it needs and renew the app from its saved state.

For apps which are meant to run in the background, it's not necessary to do all of the above, but apps should at least stop the drawing loop as drawing while in the background is a waste of system resources.

#### **2.1.3.6 User State / Application State**

The application should not leave any services running when it is in the background (unless needed for a core capability of the app). Generally speaking, the application should not leave services running to maintain a network connection, or to maintain a Bluetooth connection, or to keep GPS location powered-on.

The application should correctly preserve and restore the user or app state; particularly, it should preserve the user or app state when leaving the foreground, and prevent accidental data loss due to back-navigation or other state changes.

When returning to the foreground, the application must restore the preserved state, and any significant state-dependant transaction that was pending (such as changes to editable fields, game progress, menus, videos, and other sections of the application or game).

When the application is resumed from the background or from sleep, it should return the user to the exact state in which it was last used, unless that state has been invalidated by the passage of time (e.g. the expiry of a limit related to actual calendar date / time rather than elapsed time in the application).



When the application is re-launched from an app launcher or home screen, it should restore its state as closely as possible to the previous state (e.g. if the application is already running and the user navigates away to another application or system function, then re-launches the app, it should aim to continue the existing session rather than start a completely new instance, if this is possible).

If the platform provides a dedicated Back key or function (and this is selected), the application should then give the user the option of saving any app- or user-state that would be otherwise lost on back navigation.

#### **2.1.3.7 Resource sharing and Database**

Where there are multiple applications that make use of a common database (for example, calendar synchronization and calendar viewing), the database should be properly shared between those applications.

#### **2.1.3.8 Other programming Do's and Don'ts**

Don't update widgets too frequently, or update your location unnecessarily or use Services to try to override users or the system as these adversely impact data usage and battery life. But do share data to minimize duplication and let users manage how often they update, and whether or not they want to allow updates to happen at all

#### **2.1.4 Supportability**

This is described more in detail on section 2.4 (Operating Environment) of the System Requirements Specification Document.

- The main development work of the application is going to reside on the Application level.
- It will use a few parts of the existing Applications Framework (Content Provider -> My SQL database).
- It does not require the development of new libraries or drivers.
- This application is going to be supported from version 2.3 and up of Android OS.

### **2.1.5 Maintainability**

Maintainability is defined as the probability of performing a successful repair action within a given time. In other words, maintainability measures the ease and speed with which a system can be restored to operational status after a failure occurs.

This is similar to system reliability analysis except that the random variable of interest in maintainability analysis is time-to-repair rather than time-to-failure. For example, if it is said that a particular component has 90% maintainability for one hour, this means that there is a 90% probability that the component will be repaired within an hour. When you combine system maintainability analysis with system reliability analysis, you can obtain many useful results concerning the overall performance (availability, uptime, downtime, etc.) that will help you to make decisions about the design and/or operation of a repairable system.

This is not described on the SRS document. However in the case of My Diet App it's not possible to obtain the maintainability because the app does not require connection to the internet and there is no personal technical support. In case of critical failure the end user just can:

- Reboot the application
- Re-install the application

### **2.1.6 Manufacturability**

Design for manufacturability is the general engineering art of designing products in such a way that they are easy to manufacture.

The basic idea exists in almost all engineering disciplines, but of course the details differ widely depending on the manufacturing technology. This design practice not only focuses on the design aspect of a component but also on the productivity. In simple language it means relative ease to manufacture a product, part or assembly.

In case of My Diet App this will be really straight forward following minor rules described on the document "Best Practice Guidelines for producing high quality mobile applications" (Aqua, 2013).

Some of them are described listed below:

#### **2.1.6.1 Installation**

The application should install from the intended distribution channel, and the icon for the application should be found in the expected location on the device.

#### **2.1.6.2 Application Permissions**

The application should only request the absolute minimum permissions needed to support its core functionality.

#### **2.1.6.3 Long Launch Time**

All applications should notify the user if there's going to be a long launch time. If the Application takes longer than five seconds to be ready for use, a progress bar or a message should be displayed to tell the user what is happening.

#### **2.1.6.4 Memory and file storage during run**

For an application writing to the file system, it must ensure that it can handle out-of-space exceptions correctly during execution, and it gives a meaningful warning to the user advising about lack of space when a file is trying to be stored.

### **2.1.7 Testability**

Software testability is the degree to which a software artifact supports testing in a given test context. In case of my Diet we decided to follow the Android testing framework, an integral part of the development environment, that provides the architecture and powerful tools to test every aspect of your application at every level from unit to framework.

The testing framework has these key features:

- Android test suites are based on JUnit. You can use plain JUnit to test a class that doesn't call the Android API, or Android's JUnit extensions to test Android components. If you're new with Android testing, you can start with general-purpose test case classes such as

AndroidTestCase and then move to use more sophisticated classes.

- The Android JUnit extensions provide component-specific test case classes. These classes provide helper methods to create mock objects and methods that help you control the lifecycle of a component.
- Test suites are contained in test packages that are similar to main application packages, so you don't need to learn a new set of tools or techniques for designing and building tests.
- The SDK tools for building and tests are available in Eclipse with ADT, and also in command-line form for use with other IDEs. These tools get information from the project of the application under test and use this information to automatically create the build files, manifest file, and directory structure for the test package.
- The SDK also provides *monkeyrunner*, an API for testing devices with Python programs, and *UI/Application Exerciser Monkey*, a command-line tool for stress-testing UIs by sending pseudo-random events to a device.

### 2.1.8 Extensibility

An *extensible* application refers to an application able to extend easily without modifying its original code base. You can enhance its functionality with new plug-ins or modules. Developers, software vendors, and even customers can add new functionality or application programming interfaces (APIs) by simply adding a new Java Archive (JAR) file into the application classpath or into an application-specific extension directory.

The article “Creating Extensible Applications with the Java Platform” (O’Conner, 2007), describes two ways to create applications with extensible services, which allows to provide service implementations that require no modifications to the original application. By designing an extensible application, you will provide an easy way to upgrade or enhance specific parts of a product without changing the core application.

## 2.2 Feature Matrix / Requirements Matrix

Feature	Considered part of the testing effort
Feature 1: Self-monitoring of caloric intake	True
A. Track calories	True
B. Add new meals	True
Feature 2: Goal setting	True
A. Determine ideal caloric intake	True
Feature 3: Non food rewards	True
A. Report balance of ingested meals	True
B. Point-based system	True

## 2.3 Use Case Matrix

Use Case
1. Install Application
2. Open Application
3. Add caloric intake from existing ingredient
4. Add caloric intake from existing meal
5. Add caloric intake from new ingredient
6. Add caloric intake from new meal
7. Create meal
8. Create ingredient
9. Check records daily

10. Check records monthly
11. Modify ingredient
12. Delete ingredient
13. Modify meal
14. Delete meal
15. Close application

## 2.4 Configuration Matrix

The app does not use specific hardware, such as the camera or any other hardware devices. All the application uses are on a regular Android Smartphone.

## 2.5 Relevant policies, processes and guidelines

TBD

## 2.6 Priority Matrix

Priority	Test Effort Scope
Required	Test all code (based on the test cases) and generate tests for the code committed.
Significant	Test each feature in different environments and conditions (not all conditions and not all environments).
Moderate	Test if the software specifications were correctly implemented.

## 3. Test Approaches

### 3.1 Configuration Matrix

My Diet will be validated through **White box testing** methodology in order to achieve the best quality software development. In deep, White-box testing is the use of these techniques as guidelines to create an error free environment by examining any fragile code. It is a method of testing the application at the level of the source code.

The matrix shows up the level methodology for each feature and its use case:

Test Appr. ID	Item	Methodology Level
<b>Features</b>		
TA_1	Feature 1: Self-monitoring of caloric intake	Integration testing
TA_2	Track calories	Unit testing
TA_3	Add new meals	Unit testing
TA_4	Feature 2: Goal setting	Integration testing
TA_5	Determine ideal caloric intake	Unit testing
TA_6	Feature 3: Non food rewards	Integration testing
TA_7	Report balance of ingested meals	Unit testing
TA_8	Point-based system	Unit testing
<b>Uses Cases</b>		
TA_9	Open Application	Unit testing
TA_10	Add caloric intake from existing ingredient	Unit testing
TA_11	Add caloric intake from existing meal	Unit testing

TA_12	Add caloric intake from new ingredient	Unit testing
TA_13	Add caloric intake from new meal	Unit testing
TA_14	Create meal	Unit testing
TA_15	Create ingredient	Unit testing
TA_16	Check records daily	Unit testing
TA_17	Check records monthly	Unit testing
TA_18	Modify ingredient	Unit testing
TA_19	Delete ingredient	Unit testing
TA_20	Modify meal	Unit testing
TA_21	Delete meal	Unit testing
TA_22	Close application	Unit testing
<b>Application</b>		
TA_23	Feature 1 and Feature 2	Integration testing, Regression testing
TA_24	Feature 1, Feature 2 and Feature 3	Integration testing, Regression testing



## 3.2 Test Case Design Matrix

Test Case ID	Item	Test dep.	Pre-conditions	Expected Result
<b>Features</b>				
TC_1	Feature 1: Self-monitoring of caloric intake	TC_9,T C_2, TC_3, TC_10	Exits records on the local database.	Auto tracking caloric inputs. Visualize in the progress bar after one or more records added.
TC_2	Track calories	TC_9	Exits records on the local database.	Show the progress bar with records added.
TC_3	Add new meals	TC_9	Connection with the local database.	Show the success message.
TC_4	Feature 2: Goal setting	TC_9, TC_5	Exits records on the local database.	Visualize the goal setting.
TC_5	Determine ideal caloric intake	TC_9	Exits records on the local database.	Based on the user profile (Active, Moderate, Sedentary) visualize the ideal caloric intake.
TC_6	Feature 3: Non food rewards	TC_9,T C_7, TC_8	Exits records on the local database.	Visualize the daily report with the added reports and points gained.
TC_7	Report balance of ingested meals	TC_9,T C_1	Exits records on the local database.	Visualize daily report based on records added.
TC_8	Point-based system	TC_9,T C_2	Exits records on the local database.	Visualize points gained based on goal setting.

Uses Cases				
TC_9	Open Application	N/A		Show the main menu.
TC_10	Add caloric intake from existing ingredient	TC_9	Exits records on the local database.	Show the message "Ingredient added."
TC_11	Add caloric intake from existing meal	TC_9	Exits records on the local database.	Show the message "Meal added."
TC_12	Add caloric intake from new ingredient	TC_9	Exits records on the local database.	Show the message "New Ingredient added."
TC_13	Add caloric intake from new meal	TC_9	Exits records on the local database.	Show the message "New Meal added."
TC_14	Create meal	TC_9	Connection with the local database.	Show the message "Meal successfully created."
TC_15	Create ingredient	TC_9	Connection with the local database.	Show the message "Ingredient successfully created."
TC_16	Check records daily	TC_9	Exits records on the local database.	Show in a list all records in the current day.
TC_17	Check records monthly	TC_9	Exits records on the local database.	Show in a list all records per day in the month chosen.
TC_18	Modify ingredient	TC_9	Exits records on the local database.	Show the message "Ingredient successfully modified."

TC_19	Delete ingredient	TC_9	Exits records on the local database.	Show the message "Ingredient successfully deleted."
TC_20	Modify meal	TC_9	Exits records on the local database.	Show the message "Meal successfully modified."
TC_21	Delete meal	TC_9	Exits records on the local database.	Show the message "Meal successfully deleted."
TC_22	Close application	TC_9		Show the user's application menu.
<b>Application</b>				
TC_23	Feature 1 and Feature 2	TC_9, TC_1	Exits records on the local database.	Self monitoring of caloric intake and visualize the goal setting.
TC_24	Feature 1, Feature 2 and Feature 3	TC_9, TC_23	Exits records on the local database.	Self monitoring of caloric intake, report balance of ingested meals based on the goal setting and visualization of points earned.

### 3.3 Test Tools

The following table shows up the testing tools that My Diet is going to use:

Testing Tool	Description	Testing Coverage
GIT	Unit Test Case source control	Unit testing
Android Testing Framework	Unit test case framework (based on JUnit)	Unit testing, Functional testing (Service, Activity, Content Provider, Accessibility and UI)
Jenkins	Continuous Integration server	Integration testing, Regression testing
Android Studio	IDE for development and testing	Unit testing
Apache Ant	Software tool to make automated build process	Unit testing

### 3.4 Risk Management

The most important risk that we have in this project is finishing the application with fewer features than the ones stated on the Software Requirements Specification Document. Another important risk is to delivering the software on time.

To manage this, our plan is to use an Agile Methodology to develop the code. Using this tool our development team will use feedback to make constant adjustments in a highly collaborative environment, enforcing good code practices and establishing daily goals. This will allow the project manager to have a very detailed status about the progress of the project, how well or not the milestones are being completed and make adjustments if needed.

### 3.5 Defect Management

Defining a defect as a failure on the code to conform to specifications, we plan to use a systematic approach to debugging. To accomplish this, we are planning to have a good software toolkit as mentioned in the previous section (3.3 – Test Tools).

We will use three types of tests (regression tests, unit tests and system tests) to improve our chance to find bugs on early development stages.

Once a defect has been found, a defect tracking system will be used, giving to the project manager the capability for reporting. It is worth mentioning that due to the size of the project we are going to use a simple spreadsheet to track our defects.

### **3.6 Logistics**

The testing logistics will be coordinated by the Test Lead, which is Alba Magallanes in this case. Video conferencing software, document sharing platforms and several communication systems will be used to ensure that all the team members are aware of the work they have to do and the deadline to complete those tasks.

### **3.7 Test Management**

When required, a testing agenda will be created for all the team members containing the specific testing areas for each of them, the expected results and the way to report the issues to the developers for further fixing.

### **3.8 Additional Standards and Procedures**

Given that we are students and in most of the cases we are working full time, if a group member could not assist to a certain meeting, the procedure to handle this type of situation is to first try to assist remotely (using a video call or an online shared document) and if this is not possible review the meeting agenda and via email debrief the other group members and do all pending tasks assigned to that person.