Prototype of an Image Recognition Based Calorie Recording Application

Waqqas Sheikh
H00137508

Talal Sheikh
Project Supervisor
Declaration

I, Waqqas Sheikh, confirm that this work submitted for assessment is my own and expressed in my own words. Any uses made within it of the works of other authors in any form (e.g. ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

..............................................................
Signature

15th August 2013
Acknowledgements

I would like to thank my supervisor, Talal Shaikh, for his guidance, support and patience throughout the time I was writing this research paper. I would also like to express my gratitude towards Steve Gill; his lectures on how to conduct thorough research and writing an effective report were very useful.

Lastly, I am deeply grateful for the support and encouragement from my parents, family and friends.
Abstract

The goal of this project is to design a prototype of a smartphone calorie-counting application that uses image recognition to identify food/food products. The application allows a user to take a picture of food using the smartphone's camera; it then attempts to identify the food using Content Based Image Recognition (CBIR) and subsequently retrieve its calories from an online web-service. The user can share the photos online on social networks such as Facebook. The premise is that by allowing the user to take and share photos while recording calories in the background, calorie counting can be made fun, interactive and painless.
Contents

Table of Contents

List of Figures and Tables ........................................................................................................ 6
1. Introduction ........................................................................................................................ 7
   1.1 Dissertation Structure ................................................................................................. 8
2. Objectives .......................................................................................................................... 10
   2.1 Aims ............................................................................................................................. 10
   2.2 Stakeholders ............................................................................................................... 10
3. Literature Review ............................................................................................................... 11
   3.1 Introduction ................................................................................................................. 11
   3.2 Dieting ......................................................................................................................... 11
      3.2.1 Calorie Diets vs. Other kinds of diets ............................................................... 11
      3.2.2 Calculating Recommended Daily Calorie Intake ............................................. 13
   3.3 Content Based Image Recognition and Techniques .................................................. 14
      3.3.1 Content Based Image Recognition and MPEG-7 ............................................. 14
      3.3.2 CBIR Techniques .............................................................................................. 15
   3.4 CBIR Libraries ............................................................................................................ 21
      3.4.1 imgSeek ............................................................................................................. 21
      3.4.2 LiRe ................................................................................................................... 22
      3.4.3 Evaluation ......................................................................................................... 22
   3.5 Web Services ............................................................................................................... 23
      3.5.1 Different Approaches of Web Services .............................................................. 23
      3.5.2 Web Service Response Formats ....................................................................... 25
      3.5.3 Nutrition Web Services ................................................................................... 26
   3.6 Servlets and Tomcat ..................................................................................................... 27
   3.7 The Android Framework ......................................................................................... 28
   3.8 Smartphone Application Design & Usability ......................................................... 30
   3.9 Critical Analysis ........................................................................................................ 31
   3.10 Conclusion ................................................................................................................ 32
4. Requirements Analysis ...................................................................................................... 34
   4.1 Functional Requirements ......................................................................................... 34
      4.1.1 User Use Case Diagram .................................................................................. 36
   4.2 Non-Functional Requirements .................................................................................. 37
      4.2.1 Client Side Requirements ............................................................................... 37
      4.2.2 Server Side Requirements .............................................................................. 38
   4.3 Image Recognition Requirements .......................................................................... 40
   4.4 Optional Requirements ............................................................................................. 41
5. Professional, Legal, Ethical and Social Issues ............................................................... 42
6. Project Plan ...................................................................................................................... 43
   6.1 Project Deliverables and Tasks ................................................................................. 43
   6.2 Gantt Chart ............................................................................................................... 45
   6.3 Methodology .............................................................................................................. 46
Appendix III: Questionnaire & Responses.................................................................101
Appendix IV: Source Code..........................................................................................110
Appendix V: Functionality Test Screenshots...............................................................120
Appendix VI: Image Recognition Test Screenshots....................................................124
Bibliography..................................................................................................................127

List of Figures and Tables

Figure 1 Estimates Overweight and Obesity in males above 15 ................................7
Figure 2 Estimates Overweight and Obesity in females above 15 ......................... 7
Figure 3 Different Kinds of edges detected by EdgeHistogram Descriptor ..............16
Figure 4 Auto colour correlogram test images 1 and 2. ...........................................17
Figure 5 Auto colour correlogram results for image 1 and 2. ....................................17
Figure 6 Flowchart of the CEDD process .................................................................20
Figure 7 Auto color correlogram description of a strawberry ..................................21
Figure 8 Servlet application architecture ...............................................................28
Figure 9 Android Lifecycle .......................................................................................29
Figure 10 Requirements Use Case Diagram .............................................................36
Figure 11 Gant Chart for Project .............................................................................45
Figure 12 Server Side Architecture .........................................................................50
Figure 13 Server side database schema .................................................................56
Figure 14 Application navigation state diagram ......................................................65
Figure 15 Client side database structure .................................................................66
Figure 16 Sever Side Factory Pattern for parsing FatSecret JSON .........................70
Figure 17 Client side factory pattern for parsing web service JSON .....................71
Figure 18 Observer pattern in SearchActivity .........................................................72
Figure 19 Generic Results in FullScreenImageActivity ..........................................73
Figure 20 Specific Results in ResultsActivity ........................................................74
Figure 21 RGB Colour Space [31] ...........................................................................97
Figure 22 Colour Wheel [33] ..................................................................................97
Figure 23 Histogram of test scores .........................................................................99
Figure 24 Quantised Histogram .............................................................................100
Figure 25 Normalised Histogram ..........................................................................100

Table 1 Daily Recommended Calories .................................................................13
Table 2 Activity Factors for different Activity Lifestyles .........................................13
Table 3 Values of N corresponding to edges ..........................................................19
Table 4 Values of M corresponding to hues .........................................................19
Table 5 Activity states in Android Lifecycle .........................................................29
Table 6 Summary of User Evaluation Responses .................................................87
1. Introduction

Obesity has almost doubled since the 1980s and it is the fifth highest cause of annual deaths, at around 2.8 million adult deaths each year.\(^1\) The World Health Organisation defines being overweight as having a BMI greater than 25 kg/m\(^2\) and being obese as having a BMI greater than 30 kg/m\(^2\).\(^1\) The map below shows the percentage of population with a BMI of 25 kg/m\(^2\) or more around the world. What's clear is that in more than three-quarters of the world, at least 35% of the population is either overweight or obese.
In order to combat obesity, the WHO recommends that individuals reduce their calorie intake from fat and sugar, and engage in regular exercise amounting to at least 150 minutes per week. [1]

Calorie counting, i.e. recording the number of calories consumed in a day as well as burnt through exercise, is a proven technique to maintain a healthy weight [3]. Over the last few years, a number of smartphone calorie counting applications have emerged such as myFitnessPal, CalorieKing and LoseIt.

These applications require the user to look for his meal in a database before the calories can be recorded which is both, a time consuming and a mundane process; it would be a lot more interesting and simpler if the user could take a picture of his meal and the calories are logged automatically.

This is the goal of the project: to build an application that lets the user simply take a picture of a food or a snack product and the calories are determined and recorded in the background. Content Based Image Recognition can be used to identify the food from the photo and an online web-service can be used to retrieve its calories.

1.1 Dissertation Structure
The structure of this dissertation is as follows:

Chapter 1: Introduction
Presents the goal of this project in the context of global issues and existing calorie recording applications.

Chapter 2: Objectives
States the goals of the project and how the project is relevant to stakeholders.

Chapter 3: Literature Review
Presents background research for key concepts involved in the project. Introduces and evaluates technologies that will be used.

Chapter 4: Requirement Analysis
Lists functional, non-functional and optional requirements for the application. Provides examples of foods the application is expected to identify.

Chapter 5: Professional, Legal, Ethical and Social Issues
Describes the Professional, Legal, Ethical and Social Issues involved in the project and how they will be alleviated.
Chapter 6: Project Plan
Breaks the project down into stages and presents a project timeline in the form a Gantt chart. An evaluation strategy is introduced and possible risks are assessed.

Chapter 7: Design
Describes the server-side and client side architecture of the application, including user interface mock-ups,

Chapter 8: Implementation
Explains how the core requirements of the application were implemented with code samples. Lists issues faced in the development stage.

Chapter 9: Testing & Evaluation
Presents the results of functionality tests and image recognition tests done on final application. Summarizes responses from user evaluation.

Chapter 10: Conclusion
Summaries the purpose of the report, limitations of the application, lessons learnt from the project and future improvements.

Appendix I: Colour Spaces
Defines colour spaces and illustrates the RGB and HSV colour space.

Appendix II: Histograms
Explains the terms and concepts relating to histograms that were mentioned in the literature review.

Appendix III: Questionnaire
A copy of the original questionnaire that the users were asked to fill.

Appendix IV: Source Code
Code samples referenced by the Implementation Chapter.

Appendix V: Functionality Test Screenshots
Screenshot of functionality test results.

Appendix VI: Image Recognition Test Screenshots
Screenshot of image recognition test results.

Bibliography
Complete list of references.
2. Objectives

2.1 Aims
The aim of this project is to develop a prototype of a smartphone application that merges the enjoyable aspect of taking pictures of food (and sharing them online) with the comparatively mundane task of recording calories in order to maintain a healthy weight.

By designing an application that brings photography and online sharing to the forefront and moves weight watching to the background, the goal is to get users interested in maintaining a healthy diet.

In summary, the objectives of this project are to develop an application that:

1. Identifies food or food products from photos and records their calories
2. Informs users whether their calorie intake is within the recommended daily range.
3. Allows users to share photos on a social network.
4. Motivates users to be conscientious and proactive about maintaining a healthy diet.
5. Encourage users to make healthier food choices.

2.2 Stakeholders
The stakeholders for this application are:

1. Myself
2. My supervisor
3. Users looking for a painless way to lose weight
4. Doctors, Nutritionists and other Health Specialists
5. Individuals interested in the use of technology to solve everyday problems.

Users who wish to monitor their food intake would be interested in such an application because it would provide a quick and easy way to monitor calories.

Doctors and Health specialists can recommend the application to their patients.

Technology Enthusiasts may be interested in the concept of the project and consider implementing the future recommendations to improve the accuracy.
3. Literature Review

3.1 Introduction
The calorie application contains 3 core aspects:
- Creating a web service that uses image recognition to identify food.
- Communicating with a nutrition web service to get information for a certain food.
- Developing a client application that communicates with the web service and maintains a record of calories consumed.

The purpose of this chapter is to develop a better understanding of the concepts and technologies involved in developing such an application.

This chapter begins by describing the different kinds of diets and how to calculate the recommended daily calorie intake for an individual. Afterwards, a study is made of Image recognition and its techniques. This is followed by a look at web-services and how they work, concepts involved in developing an application for an Android smartphone and comparing the proposed application to other existing applications. The chapter concludes by overviewing how the different technologies will work together to provide the desired solution.

3.2 Dieting

Nutrition information isn’t based solely on calories; it involves other metrics such as carbohydrates, proteins and fat. The section compares calorie diets with other forms of diets and then presents formulae by which the recommended daily calories for an individual may be calculated.

3.2.1 Calorie Diets vs. Other kinds of diets.
Dieting is where an individual controls his food intake in order to maintain a healthy weight. Irene Strycher classified diets into 4 types and conducted research to compare their effectiveness and adverse effects. [4]

Low-Calorie Diets:
A calorie is a measure of the energy contained in food. Specifically, it is the energy needed to raise the temperature one gram of water by one degree Celsius.

Low-Calorie diets are low in fat but high in carbohydrates. In these diets, the individual’s daily calorie intake is between 1200-1400 for men and 1000-1200 for women. Low-calorie diets can lower total body weight by an average of 8% in the short term. These diets are not known for having adverse side effects and the weight loss is typically maintained for more than 5 years. [4]
Very-Low-Calorie Diets:
Very-low calorie diets involve an intake of less than 800 calories per day through the consumption of specific foods. Weight loss occurs at a much faster rate than low-calorie diets; average weight loss between weeks 12 and 16 is 20 kg for Very-Low-Calorie diets compared to 8 kg with low-calorie diets. This kind of diet, however, has several adverse effects such as reduction of lean body mass, cholelithiasis, ketosis and in some cases, sudden death. [4]

This diet is not recommended for an excess of 16 weeks, and is generally prescribed to people whose Body Mass Index (BMI) is greater than 30 kg/m². [4]

Low-Carbohydrate Diets:
These diets are low in carbohydrates but high in fat and protein. In two 6-month studies, it was observed that a low-carbohydrate diet resulted in almost double weight loss than a low calorie diet however these diets are not often recommended; the high intake of saturated fat may lead to heart-related chronic illnesses. [4]

Very-Low-Fat Diets:
Very-Low-Fat diets are designed to reverse or prevent heart disease: the patient is encouraged to eat foods that are high in carbohydrates and fibre but very little meat. Experiments conducted by Ornish and associates [5] revealed that patients following these diets experienced a weight loss of 10.9 kg in the first year.

However, these diets are not recommended because the high carbohydrate content increases triglyceride levels and the high consumption of fibre leads to a decrease in the absorption of vital minerals: zinc, calcium and iron. [4]

Strycher recommends low-calorie diets over other forms of dieting because: [4]
- They are closest to an individual’s normal eating habits i.e. the patient eats carbohydrates, proteins, fibre and fat but in measured amounts.
- There is little risk of adverse health effects
- Weight loss is maintained for a fairly long period of time (up to 5 years).

The table below lists the range of recommended daily calories for individuals between 19 and 30 of different activity lifestyles:
### Table 1 Daily Recommended Calories

<table>
<thead>
<tr>
<th>Activity Life Style</th>
<th>Male (19-30 years)</th>
<th>Female (19-30 years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sedentary</td>
<td>2400 - 2600</td>
<td>1800 – 2000</td>
</tr>
<tr>
<td>- Individuals who do not engage in daily exercise.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Moderately Active</td>
<td>2600 - 2800</td>
<td>2000 – 2200</td>
</tr>
<tr>
<td>- Individuals who exercise an equivalent of walking 2.5 - 5 km a day at 1.3 - 1.8 m/s.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Active</td>
<td>3000</td>
<td>2400</td>
</tr>
<tr>
<td>- Individuals who exercise an equivalent of walking 4.8km a day at 1.3 - 1.8 m/s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 3.2.2 Calculating Recommended Daily Calorie Intake

Calculating the daily caloric needs of an individual is not a precise science because every individual has a different metabolic rate. The formulas to calculate caloric needs are simply best guesses. They determine the energy expended by an individual in a sedentary state, known as the R.E.E (Resting Energy Expenditure) or B.M.R (Basic Metabolic Rate). This value is multiplied by an activity factor to account for the individual’s activity lifestyle. The resulting value is the Daily Recommended Calorie Intake. [7]

The table below lists activity factors [8]

### Table 2 Activity Factors for different Activity Lifestyles

<table>
<thead>
<tr>
<th>Activity Lifestyle</th>
<th>Activity Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sedentary</td>
<td>1.200</td>
</tr>
<tr>
<td>Lightly Active</td>
<td>1.375</td>
</tr>
<tr>
<td>Moderately Active</td>
<td>1.550</td>
</tr>
<tr>
<td>Very Active</td>
<td>1.725</td>
</tr>
<tr>
<td>Extra Active</td>
<td>1.900</td>
</tr>
</tbody>
</table>
Two well-known formulae for BMR are:

**Mifflin-St. Jeor Equation**[^8]

\[
\text{BMR (male)} = 10(\text{weight in kg}) + 6.25(\text{height in cm}) - 5(\text{age in years}) + 5 \quad (\text{Eq. 1})
\]

\[
\text{BMR (female)} = 10(\text{weight in kg}) + 6.25(\text{height in cm}) - 5(\text{age in years}) - 161 \quad (\text{Eq. 2})
\]

**Harris Benedict Equation**[^9]

\[
\text{BMR (male)} = 66.5 + 13.75(\text{weight in kg}) + 5.003(\text{height in cm}) - 6.775(\text{age in years}) \quad (\text{Eq. 3})
\]

\[
\text{BMR (female)} = 65.09 + 9.563(\text{weight in kg}) + 1.86(\text{height in cm}) - 4.676(\text{age in years}) \quad (\text{Eq. 4})
\]

### 3.2.2.1 Comparison of BMR Equations

A study was conducted in 2005 to determine which of several BMR equations most closely approximated the Resting Energy Expenditure of a test group. The report concluded that the Mifflin St-Jeor equation was most accurate, being able to estimate the measured value to within 10%.[^9]

### 3.3 Content Based Image Recognition and Techniques

#### 3.3.1 Content Based Image Recognition and MPEG-7

Content Based Image Recognition (CBIR) is the retrieval of similar images based on the content of a search image. The term ‘content’ here refers to: colours, shapes, textures and possibly textual descriptions in the image’s metadata. This differs from semantic search which is the retrieval of images based on manually-assigned keywords describing the image’s content.[^10]

In recent years, interest in the field of CBIR has grown. This is due to the increasing rate at which images are uploaded to the Internet. With so many images online, finding a particular image has become an almost impossible task for stakeholders such as journalists looking for photos of a certain event, historians looking for images from a certain era, artists looking for paintings by certain painter etc.[^10]

Previously, semantic search was used to retrieve images: Images would be assigned keywords by which they could be searched. The main drawback of this technique is that different stakeholders may perceive a single image differently. This is how Besser[^10] describes the problem:

“A set of photographs of a busy street scene a century ago might be useful to historians wanting a ‘snapshot’ of the times, to architects looking at buildings, to urban planners looking at traffic patterns or building shadows, to cultural historians looking at changes in fashion, to medical researchers looking at female smoking habits, to sociologists looking at class distinctions, or to students looking at the use of certain photographic processes or techniques.”[^10]
CBIR solved this problem by being able to retrieve similar images given a ‘search’ image.

A new multimedia standard, MPEG-7, was introduced specifying content that can be used to identify images and how this content should be formatted:

**Description Definition Language:** This is an XML based language used to define Description Schemes and Descriptors.\[^{[11]}\]

**Description Scheme:** Similar to an XML Schema, this defines the semantics with by the content will be described.\[^{[11]}\]

**Descriptor:** The descriptor actually describes the content according to the semantics in the Description Scheme.\[^{[11]}\]

### 3.3.2 CBIR Techniques

The MPEG-7 includes a number of standard descriptors to describe content in terms of colour, shape and texture. These include\[^{[11]}\],

- ColorLayout
- EdgeHistogram

Other descriptors, not part of the standard, include:

- Auto Color Correlogram
- CEDD (Color and Edge Directivity Descriptor)

This section will briefly explain how these descriptors extract and summarise the content; an in-depth mathematical treatment would be beyond the scope of this project. The concepts of ‘Colour Space’ and ‘Histograms’ are explained in Appendix I and II respectively.

#### 3.3.2.1 ColorLayout

The ColorLayout Descriptor describes how the predominant colours in an image are spatially arranged. In order to extract this descriptor:

- The image is divided into an 8 x 8 grid consisting of 64 blocks. This makes the descriptor independent of the image size.
- A representative colour in the YCbCr colour space is selected from each block. This is achieved by taking an average of all the colours in a given block.
- Next, each coefficient of the representative colour (Y, Cb and Cr) in each block undergoes Discrete Cosine Transformation. This gives three 8x8 arrays, one array for each coefficient (DCTY, DCTCb and DCTCr).
- The three matrices are scanned to remove high frequency and low frequency values. The resulting 3 arrays (DY, DCT and DCr) are the ColorLayout Descriptor of the image.\[^{[12]}\]
In order to compare the CLD of two images, the following formula is used:

$$D = \sqrt{\sum_i w_{y_i} (DY_i - DY'_i)^2} + \sqrt{\sum_i w_{b_i} (DCb_i - DCb'_i)^2} + \sqrt{\sum_i w_{r_i} (DCr_i - DCr'_i)^2}$$

(Eq. 5)\textsuperscript{[12]}

The ColorLayout descriptor’s advantages are\textsuperscript{[12]}:
- Extracted very quickly.
- Compact (only 12 coefficients stored).
- Fast comparison

The main disadvantage is that it only stores information about the colours in the image and no information about texture or shape

3.3.2.2 EdgeHistogram

The Edge Histogram records the frequency of 5 different edges in a given image. These edges are: vertical, horizontal, 45-degree diagonal, 135-degree diagonal and non-directional. \textsuperscript{[13]}

![Figure 3 Different kinds of edges detected by the EdgeHistogram descriptor \textsuperscript{[1]3}](image)

Steps:
- The image is divided into 4x4 blocks. This makes the descriptor independent of the size of the image.
- For each block, a histogram is created. This histogram has 5 bins corresponding to the 5 kinds of edges.
- Each block is scanned for all different edges. Each time a certain kind of edge is found, the appropriate bin is incremented.
- The 5-bin histograms of each block are joined together to make a single 80 bin histogram.
- The histogram is normalised and quantised so that each bin is represented using 3 bits.
- These bits are the EHD for the image. \textsuperscript{[13]}

The advantages of this descriptor are:
- Compact
- Fast Comparison

16
- Robust against noise e.g. brightness changes, orientation.

The disadvantages are:
- It only records the frequency of edges in an image. It does not record any information about colour or texture.
- Details are stored for 8x8 blocks. Small texture details might be missed.

### 3.3.2.3 Auto colour correlogram

An auto colour correlogram describes how the distance between two colours changes from a single point within the image.\(^{[14]}\)

- The auto colour correlogram is a HashMap, where the key is a colour pair and the value is a list of probabilities.
- The Nth entry of the list contains the probability of finding one colour at a distance N from the second colour.

The diagram below graphs the contents of an auto colour correlogram:

![Figure 4 Auto colour correlogram test images 1 and 2.\(^{[14]}\)](image)

![Figure 5 Auto colour correlogram results for image 1 and 2.\(^{[14]}\)](image)

- The x-axis represents the horizontal distance.\(^{[14]}\)
● Line B is the probability of the grey background colour appearing at a various distances from the white colour in image 2. \[14\]

● Line A is the probability of the grey background colour appearing at various distances from the white colour in image 1. \[14\]

● Line C is the probability of the white colour appearing at various distances from the grey background colour in image 1.

● Line D is the probability of the white foreground colour appearing at various distances from the grey background colour in image2. \[14\]

The benefits of this descriptor are:
1. It is robust against large changes in appearance due to orientation, lighting, camera zoom, background changes.
2. The size of the descriptor is fairly small
3. The descriptor is easy to compute \[14\]

Huang and associates \[14\] conducted an experiment to compare the performance of an auto colour correlogram with other histogram-based descriptors. 77 queries were executed on a database consisting of 14,554 images. Each query had a unique correct answer. The result of their experiment is summarized in their report as follows:

_from a user’s point of view, these results can be interpreted as follows: given a query, the user is guaranteed to locate the correct answer by just checking the top two search results (on average) using autocorrelograms. On the other hand, the user needs to check at least the top 80 search results (on average) to locate the correct answer in the case of histogram._ \[14\]

3.3.2.4 Content Edge Directivity Descriptor

The Content Edge Directivity Descriptor (CEDD) describes the colour and the shapes within an image. \[15\]

_steps:_
- The image is broken down into 1600 blocks.
- Each block goes through a ‘Texture Unit’ in the Luminescence colour space (YIQ) and the ‘Colour Unit’ in the HSV Color space.
- The Texture unit returns a value \( N \) which is between 0 and 5, corresponding to the kind of edge in the block: \[15\]
Table 3 Values of N corresponding to edges\textsuperscript{[15]}

<table>
<thead>
<tr>
<th>N</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Non edge</td>
</tr>
<tr>
<td>1</td>
<td>Non-directional edge</td>
</tr>
<tr>
<td>2</td>
<td>Horizontal edge</td>
</tr>
<tr>
<td>3</td>
<td>Vertical edge</td>
</tr>
<tr>
<td>4</td>
<td>45 degree edge</td>
</tr>
<tr>
<td>5</td>
<td>135 degree edge</td>
</tr>
</tbody>
</table>

- The Colour Unit returns a value M between 0 and 23, corresponding to the hue in the block: \textsuperscript{[15]}

Table 4 Values of M corresponding to hues \textsuperscript{[15]}

<table>
<thead>
<tr>
<th>M</th>
<th>Hue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>Grey</td>
</tr>
<tr>
<td>2</td>
<td>White</td>
</tr>
<tr>
<td>3</td>
<td>Dark Red</td>
</tr>
<tr>
<td>4</td>
<td>Red</td>
</tr>
<tr>
<td>5</td>
<td>Light Red</td>
</tr>
<tr>
<td>6</td>
<td>Dark Orange</td>
</tr>
<tr>
<td>7</td>
<td>Orange</td>
</tr>
<tr>
<td>8</td>
<td>Light Orange</td>
</tr>
<tr>
<td>9</td>
<td>Dark Yellow</td>
</tr>
<tr>
<td>10</td>
<td>Yellow</td>
</tr>
<tr>
<td>11</td>
<td>Light Yellow</td>
</tr>
<tr>
<td>12</td>
<td>Dark Green</td>
</tr>
<tr>
<td>13</td>
<td>Green</td>
</tr>
<tr>
<td>14</td>
<td>Light Green</td>
</tr>
<tr>
<td>15</td>
<td>Dark Cyan</td>
</tr>
<tr>
<td>16</td>
<td>Cyan</td>
</tr>
<tr>
<td>17</td>
<td>Light Cyan</td>
</tr>
<tr>
<td>18</td>
<td>Dark Blue</td>
</tr>
<tr>
<td>19</td>
<td>Blue</td>
</tr>
<tr>
<td>20</td>
<td>Light Blue</td>
</tr>
<tr>
<td>21</td>
<td>Dark Magenta</td>
</tr>
<tr>
<td>22</td>
<td>Magenta</td>
</tr>
<tr>
<td>23</td>
<td>Light Magenta</td>
</tr>
</tbody>
</table>

- The final histogram consists of 144 bins (6 possible edges x 24 possible hues)
- For each block, the bin corresponding to its N x 24 + M is incremented.
- This histogram is normalised and quantised at 3 bits per pin.

The flow chart below summarises the process. \textsuperscript{[15]}
The advantages of this descriptor are\cite{15}:
- Higher performance compared to Edge Histogram
- Less computation power required in order to extract descriptor.
- Records shape and colour. As the image is broken down to 1600 blocks, minor details of texture may be recorded but eliminated in the quantisation process.

3.3.2.5 Evaluation of CBIR Techniques

To decide on the most suitable descriptor for the application, a set of metrics needs to be outlined. The requirements set here are:

1. The descriptor should store information that is capable of recognising food. Humans identify objects through shape, colour and texture. The most reliable descriptor would be one that stores all 3 values.

2. The descriptor should be able to withstand background noise: Users are likely to take pictures in a variety of situations: different background scenes, different zoom settings and different orientation.

3. The descriptor should have reasonable accuracy in spite of orientation of food.

The ColorLayout Descriptor describes only the spatial arrange of dominant colours in the image. It does not contain information about shapes or texture, and it is susceptible to background noise. Since it only records dominant colour, it is also prone to false matches e.g. a strawberry and a tomato have approximately the same colour yet very different shapes and textures.

The Edge Histogram Descriptor stores information on shape, but due to the large block sizes: it misses information about texture. It also does not store information about colour.

This leaves Auto Colour Correlogram and CEDD, both of which are strong candidates as they store more than one aspect of an image’s content. CEDD stores the colours and the edges in a shape. Auto colour correlogram records the probability that two colours will appear at a certain distance from one another.
From the two, Auto Colour Correlogram seems to be the more suitable choice. If we consider the example of a strawberry, the CEDD would simply create a histogram describing colour and edges in the image. The Auto colour correlogram would record the changes in colour with distance, which is indicative of shape, texture and, of course, colour.

Additionally, experiments conducted by Huang and associates \(^{[14]}\) have shown that it can withstand noise such as changes in background, orientation, lighting and zooming.

Hence, the Auto Colour Correlogram will be used as the image descriptor for this application.

### 3.4 CBIR Libraries

In this section, two CBIR Libraries are compared to decide which of the two is most suited to the needs of the application.

#### 3.4.1 imgSeek

ImgSeek is an open-source database server that allows any website or application to add Content Based Image Recognition services. The server uses optimised C++ code for image processing and provides a simple API (in Java, PHP and Ruby) for performing tasks such as \(^{[16]}\):

- Fast indexing of images
- Associating keywords with images for semantic retrieval
- Performing image similarity queries
- Retrieving similar images from a database. \(^{[16]}\)

ImgSeek retrieves similar images using Fast Multi-resolution Image Querying. A technique that is similar to the ColorLayout descriptor \(^{[17]}\). The library is open-source so the image-recognition implementation can be modified as needed. \(^{[16]}\)
The library is hosted on a server and handles the job of indexing the images and retrieving similar ones. The developer has only to set up a file system for the images and a database storing their ids and any other relevant information. [16]

3.4.2 LIRe

LIRe (Lucene Image Retrieval) is an open-source Java library for content-based image recognition based on the Lucene search index library. It supports a wide range of descriptors including [18]:

- ColorLayout
- EdgeHistogram
- RGB Histogram
- CEDD
- Auto Colour Correlogram

**Mechanism**

LIRe is based on Lucene, which is an open-source text search engine.

Lucene takes a document and divides it into fields. A field is a name-value pair. The value either contains text from within the document or keywords. The name identifies the text. Each field has a field-object, which could be one of either: indexed, stored or tokenised. [19]

- **Indexed** Indexed fields are indexed for later retrieval.
- **Stored** Stored fields contains text that is displayed with the search results e.g. a Summary.
- **Tokenised** Tokenised text is broken down into keywords, each of which is then indexed.

Each field is indexed along with the documents it appears in; such an index is known as an ‘Inverted Index’. [19]

LIRe treats an image as a Lucene document; the descriptors extracted from the image and the metadata are stored as text and treated as fields. These fields are indexed. [18]

During image retrieval, LIRe creates a document for the query image and uses indexes to retrieve similar images.

3.4.3 Evaluation

The following are the features desired from the Image Recognition library:

1. **Free**.

2. **Well-Documented**:
The methods in the CBIR library should be listed and well-documented, ideally with examples. This would reduce time spent learning to use the library.

3. **Provide a suitable descriptor: CEDD or Auto Colour Correlogram**

   As discussed earlier in this chapter, the CEDD and Auto Colour Correlogram descriptors are best suited to recognise and retrieve similar images. The library should provide either of these descriptors or, at least, an equally capable one.

Both imgSeek and LIRe are well-documented libraries. Both provide a web page listing methods; both mention best practices when using the API and both provide open-source demo applications showing the code in action.

In using imgSeek, however, there are two drawbacks:

1. ImgSeek needs to be set up on a server. The purchase, operational and maintenance cost of a dedicated server would be too high for this project.

2. ImgSeek uses Fast Multi-resolution Image Querying, a technique similar to ColorLayout, which was dismissed earlier for not being suitable for the needs of this project. It is possible to implement the Auto Colour Correlogram however LIRe has the functionality built right in.

As such, LIRe has been selected as the CBIR library for this project.

### 3.5 Web Services

This project will create an online web service, which will store uploaded images, retrieve food names from similar images and fetch nutrition information of the food via a second nutrition web service.

This section looks at two different approaches to web services and compares various nutrition services to select the most appropriate one.

#### 3.5.1 Different Approaches of Web Services

A Web service is an application whose methods are accessible online. The data returned by the method is in a standardised format e.g. XML, JSON or plain text. The benefit of web services is that part of the functionality can be moved online and shared between different applications, even if the client applications are written in different languages. This reduces development time.

There are two main approaches to web services: SOAP and REST.
3.5.1.1 SOAP

SOAP is an acronym for Simple Object Access Protocol. Messages are written in XML in a standardised format and transported using the HTTP or SMTP Protocol. Each message contains a header and a body. The header contains the ID of the transaction while the body names the method to be invoked and includes any parameters the method requires. The response from the web service returns the result of the method.\[^{20}\]

The methods of SOAP web services are exposed in Web Service Description Language (WSDL). This is an XML file that stored at a known location on the server and can be accessed by a simple HTTP GET request.\[^{20}\]

The code below is an example quoted from Programming Web Services with SOAP\[^{20}\]. It shows an example of a SOAP request being sent to get the stock quote of IBM. The SOAP response returns the value.

SOAP Request:\[^{20}\]
\[
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Header>
    <m:transaction xmlns:m="soap-transaction" s:mustUnderstand="true">
      <transactionID>1234</transactionID>
    </m:transaction>
  </s:Header>
  <s:Body>
    <n:getQuote xmlns:n="urn:QuoteService">
      <symbol xsi:type="xsd:string">IBM</symbol>
    </n:getQuote>
  </s:Body>
</s:Envelope>
\]

SOAP Response:\[^{20}\]
\[
<s:Envelope xmlns:s="http://www.w3.org/2001/06/soap-envelope">
  <s:Body>
    <n:getQuoteRespone xmlns:n="urn:QuoteService">
      <value xsi:type="xsd:float">98.06</value>
    </n:getQuoteRespone>
  </s:Body>
</s:Envelope>
\]
3.5.1.2 REST

REST is an acronym for Restless State Transfer. It was introduced by Roy Fielding in his thesis wherein he argued that web services could be implemented using the existing HTTP Protocol rather than extending it the way SOAP does.\[^{[21]}\]

REST consists of resources. Each resource is identified by URIs (Uniform Resource Identifier). Resources can be manipulated using standard HTTP Methods\[^{[21]}\]:

- **GET**: retrieves/queries resource identified by a URI
- **POST**: places resource on the server
- **PUT**: updates resource on the server
- **DELETE**: deletes resource on the server.

Other methods exist, but these are the main ones. The result can be in any format: XML, JSON, HTTP or even plain text.\[^{[21]}\]

The World Wide Web itself is an example of REST architecture. Each online resource is given a URL (Uniform Resource Locator). A GET request is used to retrieve the resources and a POST request is used to post data to the server. The response is formatted in HTML.\[^{[21]}\]

The web service created in this project will follow a REST architecture for the following reasons:

1. REST is minimal. No extra tools or frameworks are needed to set up a REST web Service. Such is not the case with a SOAP service.
2. In relation to the first point, no extra set-up is record for a client to invoke a REST method.
3. REST has the benefit of URL Caching\[^{[21]}\]. This means that a response to a request can be recycled thus making efficient use of resources.

3.5.2 Web Service Response Formats

With SOAP web services, the request and the response must be encoded in XML\[^{[20]}\]. On the other hand, with REST services, there is no defined standard for the encoding of the responses. XML and JSON, however, are the two most commonly used formats.\[^{[21]}\]

XML stands for Extensible Mark-Up Language. A schema defines how the data is to be structured in a hierarchy and the data types involved. The XML file can be validated against its schema. It can be transformed easily into other formats via XSLT and queried using tools such as XPath or XQuery\[^{[21]}\]. The main drawback of XML is its wordiness. A start tag and an end tag surround each data element, and quite often the tag takes up more memory than the data being transported. Thus, transporting data via XML may consume considerably more bandwidth.\[^{[21]}\]
JSON, on the other hand, is very minimal. It simply consists of name-value pairs. Data can be represented as objects, arrays and maps. JSON is not as extensible as XML, nor can it be queried or transformed easily into other formats. However, for this application - the extra features of XML are not required. The data simply needs to be transported. [21]

Since JSON is simpler and more efficient at transporting data than XML in terms of bandwidth, the calorie app web service will encode its responses in JSON.

3.5.3 Nutrition Web Services
This section presents and compares three nutrition web-services.

FatSecret Platform API
The FatSecret Platform is geared towards providing reliable product nutrition information so that consumers can make informed choices. The API provides developers access to their comprehensive database of nutritional information through REST and JavaScript. The API is free for commercial use, and is well documented online with code samples.

The API uses OAuth (Open Authentication) to verify the authenticate requests and has a limit of 5000 API calls per day. If an application expects more calls, support can be provided. [22]

MyNetDiary Food Service
MyNetDiary provides access to a nutritional database of about 182,000 foods through two methods: [23]

<table>
<thead>
<tr>
<th>Find Food</th>
<th>This method returns food name and id based on search criteria. The results are sorted by popularity. Each query costs 0.5 cents for production usage.</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetFoodDetails</td>
<td>This service returns nutritional information given an id. Invoking this method is free but the FindFood service needs to be used to get the id for the food.</td>
</tr>
</tbody>
</table>

Both REST and SOAP APIs are available, and output is formatted in JSON or XML respectively. [23]

MyFitnessPal
MyFitnessPal is an online calorie logging service. Its free API provides access to a database of about 2.5 million different foods that is updated continuously. Several Apps have already been developed using the API such as BodyMedia FIT and DigiFit iCardio. [24]

The API is free to use however access is limited to approved developers. A request to use the service was sent, however no response has been received so far.
3.5.3.1 Comparison of Nutrition Web Services

The choice of nutrition web service is based on the following criteria:

1. **Service should provide an extensive database**
   A larger database means the application is more scalable and more useful to potential users.

2. **Free to use**

3. **Well-documented**
   The web service should be well documented, presenting a list of functions available and examples of how to use them. This reduces development time and effort.

All three web-services provide access to an extensive database; however, MyNetDiary puts a price on Food ID queries.

Whether MyFitnessPal and MyNetDiary are well documented could not be determined because both platforms review individual requests to use their APIs. Requests have been sent to both platforms with no response.

The FatSecret API is open for use by the public and is well documented, providing clear examples of how the REST API can be used to retrieve calorie information.

As the FatSecret API is the only one of the three web services that appears to meet the set criteria, it has been selected for use in this project.

3.6 Servlets and Tomcat

Java will be used as the server side programming language. This is because the LIRe library itself is written in Java and so the library can be easily added to the build path.

Servlets are Java classes that incorporate server behaviour: they receive requests and generate responses.\(^{[25]}\)

In order to deploy a Servlet, a Servlet container is needed. The container is responsible for loading the servlet, passing requests from the client to the servlet and communicating responses from the servlet back to the client. Tomcat is an open source servlet container based on the Apache Web Server.\(^{[25]}\)

The diagram below illustrates architecture of a Servlet application.\(^{[25]}\)
When the Servlet Container receives an HTTP Request, it checks if an instance of the Servlet already exists. If it does not, it initialises a new instance of the servlet and calls the init() method which configures the Servlet. The container also creates an HttpServletRequest object, containing details of the request, and an HttpServletResponse object, which will contain the response. These are passed as parameters to the initialised servlet.\(^{[25]}\)

A separate thread handles each HTTP request; this avoids creation of multiple servlets, which would consume considerable sources. The service() method of the Servlet is called which identifies the type of request (GET / POST / PUT / DELETE etc) and invokes the corresponding method (doGet, doPost, doPut, doDelete etc). The method processes the request and generates a response. This response is communicated to the user.\(^{[25]}\)

### 3.7 The Android Framework

The client application will be developed for Android smartphones. This section presents basic fundamentals of the Android framework.

Android applications typically consist of Activities. An activity corresponds to a single screen on the User Interface. A typical application will consist of a number of activities, each performing a different role. One activity can start another activity to perform a certain task. The launched activity can be in the same application or even in a different application. Data can be transferred between two activities via a Bundle object. The bundle object stores data in key-value pairs. The key is the name of the data and the value is a primitive data type or any object that implements the Parcelable interface.\(^{[26]}\)

The Android operating System maintains a stack of launched activities, known as the “Back Stack”. When one activity launches a second activity, the first activity is paused and the second activity is activated and pushed to the top of the stack. When the user presses the back button of the device, the current activity is popped off the top of the stack and stopped. The previous activity is then resumed.\(^{[26]}\)

As such, Android activities have a life cycle consisting of the following states. A callback method is associated with each change of state:
Table 5 Activity states in Android Lifecycle[26]

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
<th>Callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>Created</td>
<td>Activity is launched and initialized in memory.</td>
<td>onCreate()</td>
</tr>
<tr>
<td>Resumed</td>
<td>The Activity is visible on screen and has user focus.</td>
<td>onResume()</td>
</tr>
<tr>
<td>Paused</td>
<td>The Activity is still visible on screen but not in focus e.g. a modal Progress Dialog is being displayed.</td>
<td>onPause()</td>
</tr>
<tr>
<td>Stopped</td>
<td>The Activity is no longer visible on the screen but its state is preserved in memory</td>
<td>onStop()</td>
</tr>
<tr>
<td>Destroyed</td>
<td>The Activity and its resources are deallocated from memory.</td>
<td>onDestroy()</td>
</tr>
</tbody>
</table>

The state diagram below summarizes the Activity Life Cycle:

![Android Activity Lifecycle Diagram](image_url)
3.8 Smartphone Application Design & Usability

The smartphone environment imposes a number of constraints that make smartphone application design more challenging than desktop applications. These constraints are:

1. **Limited Screen Space**
   Smartphone screens are considerably smaller than desktop screens meaning less content can be displayed. [27]

2. **Awkward Input**
   Scrolling and tapping is used for navigating content on a mobile device. Typing on the small keyboards is error-prone and uncomfortable. [27]

3. **Different Screen Sizes**
   Each smartphone manufacturer produces phones of different sizes and an effort needs to be made to ensure that the same experience is provided across the majority of platforms. [27]

A well-designed application not only has to make the most out of the limited space available, but it must also be usable. Usability is defined as how well a user learns to use a product to achieve his goal. [27] Jacob Nielson, a noted expert on web usability, stated 5 points that constitute good usability:

1. **Ease of Learning**
   This is a measure of how easy it is for a new inexperienced user to accomplish basic tasks. [27]

2. **Efficiency of Use**
   This is a measure of the speed by which an experienced user accomplishes basic tasks. [27]

3. **Memorability**
   Memorability measures how easy it is for a user to remember to do basic tasks after he hasn’t used the application for some time. [27]

4. **Error Frequency and Severity**
   This is a measure of how frequent an error of a certain severity is. [27] Severity itself is a measure of how easy it is to return from an error. A typographical error is of low severity as it can be fixed by pressing the backspace key. Permanently erasing the contents of a user database is of high severity. [27]

5. **Subjective Satisfaction**
   This is a measure of how satisfied the user is with the system on the whole. A system that is intuitive and presents fewer opportunities for errors is one that consumers are likely to be content with. [27]

Based on the limitations of smartphones and on the principles of good usability outlined above, the guidelines for good smartphone application design are:
1. Keep it simple
The actions on a mobile device should be simple and intuitive. Long complex procedures that require greater use of the awkward input methods, especially the keyboard, should be avoided. On the other hand, scrolling and hand gestures should be taken advantage of.

2. Prioritise the content
Due to limited screen real estate, it’s important to identify the content the user wants to see and to make it easily accessible and viewable. Images should be enlarged to show as much detail as possible. The application should avoid having the user search for content, as typing in a search field is error-prone, uncomfortable and frustrating.

3. Scroll Vertical Only
Smartphones have limited width so it makes more sense to arrange the content vertically. Scrolling in one direction also simplifies usability, as navigating in two directions can be an inconvenience.

4. Avoid Repeating the Navigation
Navigation panels on each screen can take up considerable space leaving less for content. It is therefore better to dedicate one screen for navigation (e.g. a Main Menu screen) and have all other screens link back to the navigation screen. Alternatively, the navigation panel could be hidden by default and brought into view through an intuitive method.

3.9 Critical Analysis
This section explains the benefits of calorie logging applications and then compares the intended calorie app with similar kinds.

Dieting by keeping track of calories consumed and burnt (through exercise) is a proven technique. Research has shown that individuals who monitor their calorie intake may lose up to twice as much as those who don’t:

“After adjusting for race, gender, and initial weight, greater weight loss was associated with more frequent attendance at the group sessions, number of food records kept per week, and minutes of moderate-intensity physical activity per week.”

On the other hand, calorie counting can be quite cumbersome. It requires an individual to diligently find out the nutrition content of the meals he’s eaten and note them down regularly, ensuring that he has not eaten more than a set amount. This inconvenient process puts quite a few people off of monitoring their caloric intake in the first place. Smartphone applications solve this issue by automating the process. Applications provide a built-in searchable database of nutrition information and the calorie record is maintained automatically by the application. A 2012 study into the effectiveness of integrating technology...
with weight loss treatment concluded that individuals who used a Personal Digital Assistant lost 3.1% more weight than the control group. Furthermore, individuals who used apps to assist weight loss had greater odds of losing 5% of their weight in the first 3 months. The study involved adults in their late 50s with an average BMI of 35kg/m². [29]

> Our participants used the PDA effectively after a brief training session, which accords with other evidence that older adults increasingly use technology. For example, 81% of 55- to 64-year-olds and 56% of those who are 65 years or older own a cell phone. Mobile devices and apps have become accessible to a degree that was unimaginable previously. Therefore, interventions, such as the present one, that use technology to augment an existing care system hold great potential to advance population health. [29]

MyFitnessPal and Calorie King are two examples of such calorie logging apps. These applications provide a vast database of nutrition information. The user searches for his meal in the database and adds the associated calories to his online food journal. This approach is highly convenient but searching through the database is still not all that convenient. The databases are large so it takes time to find the item the desired item.

The intended application overcomes this problem to some extent. The user takes a picture of the food rather than type it in. If the food in the picture can be identified, then the search for the item in the database will be performed automatically. If the food isn’t identified, then the user will have to manually search through the database, but in future situations, the application should be able to identify the item. Additionally, taking pictures of food is a far more interesting input method than is typing on a compact keyboard. This is further augmented with the option to share pictures on online social networks.

In short, the main feature points of the application are interactive and automated calorie recording with the ability to share and connect. The main limitation is the reliability of image recognition technology to identify food.

### 3.10 Conclusion

This section summarises how the technologies described and selected in this chapter will work together to meet the basic requirements.

The overall project consists of 3 elements:

- The client application (named Caloricam),
- The Caloricam web service,
- The FatSecret web service.

At the client side, the user takes a picture of food that is uploaded to the server.
At the server, there is a file system containing images of various foods along with a database of their IDs and names. CBIR technology is used to find images of foods similar to the one uploaded. The food names associated with the images are read from a food database. The Auto colour correlogram was selected for this task because it retrieves images based on colour, shape and texture and is robust against background noise, lighting, zooming and orientation changes.

This project will use the LIRe library in order to add image recognition functionality. This library was selected because it is free, well documented, easy-to-integrate and features the auto colour correlogram descriptor.

Once possible food names are determined, the nutritional information is retrieved using the FatSecret API. This API was chosen because it is open to developers, well documented and has a database sizeable for the needs of this project.

The data is sent back to the client device and presented to the user. The user selects the matching food and the calories are logged in the journal. The user has the option to share the picture on any social network application installed on his device.

By merging calorie counting seamlessly with social sharing, the belief is that maintaining a healthy weight will be effortless and this will motivate users to pay greater attention to their eating habits.
4. Requirements Analysis

This chapter will present the functional, non-functional and optional requirements of the application as well as detailing the expectations of the image-recognition feature.

The requirements are based on the aims of the projects described in chapter 1 and the research presented in chapter 3.

4.1 Functional Requirements

Functional requirements are the features that will be available to the user. These are described below:

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-APP-001</td>
<td><strong>Create Profile</strong></td>
</tr>
<tr>
<td></td>
<td>● When the user launches the application, he must create a profile. The user must enter his age, sex, height, weight, activity factor, and weight loss goal.</td>
</tr>
<tr>
<td></td>
<td>● The physical characteristics will be used to determine the recommended daily calorie intake using the Mifflin-St. Jeor method.</td>
</tr>
<tr>
<td></td>
<td>● The profile details will be saved on the device</td>
</tr>
<tr>
<td>F-APP-002</td>
<td><strong>View Profile</strong></td>
</tr>
<tr>
<td></td>
<td>● The user should be able to view profile details from within the application.</td>
</tr>
<tr>
<td>F-APP-003</td>
<td><strong>Edit Profile</strong></td>
</tr>
<tr>
<td></td>
<td>● The user should be able to update profile details as necessary.</td>
</tr>
<tr>
<td></td>
<td>● Changes made to the profile should be saved on device and reflected throughout the application.</td>
</tr>
<tr>
<td>F-APP-004</td>
<td><strong>Take Photos</strong></td>
</tr>
<tr>
<td></td>
<td>● The user should be able to take photos using the device’s built-in camera.</td>
</tr>
<tr>
<td></td>
<td>● Photos should be saved on the device.</td>
</tr>
<tr>
<td>F-APP-005</td>
<td><strong>View Photos</strong></td>
</tr>
<tr>
<td></td>
<td>● The user should be able to view all photos taken with application in a gallery.</td>
</tr>
<tr>
<td></td>
<td>● Clicking on a photo in the gallery should display an enlarged view.</td>
</tr>
<tr>
<td>F-APP-006</td>
<td><strong>Delete Photos</strong></td>
</tr>
<tr>
<td></td>
<td>● The user should be able to delete photos from the gallery</td>
</tr>
<tr>
<td>F-APP-007</td>
<td>Share Photos</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------</td>
</tr>
<tr>
<td>● The user should be able to share photos on social media from within the gallery.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F-APP-008</th>
<th>Identify Food in Photos</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Photos of food taken by the application should be uploaded to the Caloricam web service, which attempts to identify the food item.</td>
<td></td>
</tr>
<tr>
<td>● If identification yields results, food names and calories are presented to the user so that the user can select the closest match.</td>
<td></td>
</tr>
<tr>
<td>● If identification fails, the user should still be able to search for the food item in the food database.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F-APP-009</th>
<th>Search Food in Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>● The user should be able to search for a food name in the food database and get possible matches and nutrition information.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F-APP-010</th>
<th>Add Entry to Journal</th>
</tr>
</thead>
<tbody>
<tr>
<td>● The user should be able to log the details of a meal i.e. food name, calories (determined either by F-APP-008 or by F-APP-009) into a calorie journal.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F-APP-011</th>
<th>View Entries in Journal</th>
</tr>
</thead>
<tbody>
<tr>
<td>● The user should be able to view the total calories consumed on each day of a given month.</td>
<td></td>
</tr>
<tr>
<td>● The user should be able to view all meal entries for a given day.</td>
<td></td>
</tr>
<tr>
<td>● The application should indicate if the user ate more, less or equal to the recommended daily calorie intake on a given day.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>F-APP-012</th>
<th>Delete Entries from Journal</th>
</tr>
</thead>
<tbody>
<tr>
<td>● The user should be able to delete meal entries for a given day.</td>
<td></td>
</tr>
</tbody>
</table>
4.1.1 User Use Case Diagram

Figure 10 Requirements Use Case Diagram
## 4.2 Non-Functional Requirements

### 4.2.1 Client Side Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Description</th>
</tr>
</thead>
</table>
| NF-APP-001     | **Profile data must be encrypted.**  
  - Profile data is of a personal nature and should be stored in encrypted form. |
| NF-APP-002     | **Easy and Intuitive Navigation**  
  - The application should have a flat structure so that navigation is quick and easy to learn (Discussed in the Usability section of the Literature Review) |
| NF-APP-003     | **Intuitive Design**  
  - The stakeholders of this application vary in age and technical literacy; therefore the application should have a really intuitive design that makes it easy to use and learn.  
  - Application should provide visual cues to indicate what the user should do next.  
  - Processes should not involve many unnecessary steps.  
  - The design of the application should not allow for many accidental steps (e.g. accidental clicks)  
  - If an accidental step does happen, recovery should be quick and painless. |
| NF-APP-004     | **Simple UI**  
  - The design of the application should be simple, so as not to cause too much confusion, but at the same time, it has to be appealing.  
  - The application must make optimal use of limited screen real estate. |
| NF-APP-005     | **Optimal Performance**  
  - The application should be execute basic operations in no more than 2 seconds. The user should not have to wait.  
  - Heavy operations (i.e. image recognition and identification) should be done on server side to preserve device’s battery. User should be informed of undergoing steps so avoid frustration.  
  - The application should be immediately responsive to touches and gestures. |
4.2.2 Server Side Requirements

4.2.2.1 Web Service Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF-SER-001</td>
<td><strong>Upload Pictures</strong></td>
</tr>
<tr>
<td></td>
<td>• The server should be able to receive image uploads from client side application.</td>
</tr>
<tr>
<td></td>
<td>• Uploaded images should be saved in the ‘images’ directory.</td>
</tr>
<tr>
<td></td>
<td>• If uploaded is successful, the request should be forwarded to the Indexer which indexes the image.</td>
</tr>
<tr>
<td></td>
<td>• If an uploaded image already exists on the server (i.e. has the same name), it should not be uploaded. This still counts as a successful upload.</td>
</tr>
<tr>
<td></td>
<td>• If upload fails, response will consist of an error code and message.</td>
</tr>
<tr>
<td>NF-SER-002</td>
<td><strong>Index Pictures</strong></td>
</tr>
<tr>
<td></td>
<td>• Uploaded pictures should be indexed using the Auto Colour Correlogram method.</td>
</tr>
<tr>
<td></td>
<td>• Indexing should be thread-safe in light of the multi-threaded environment of servlets (discussed in the Servlets section of the Literature Review Chapter).</td>
</tr>
<tr>
<td></td>
<td>• If indexing is successful, an OK status code should be sent to the client.</td>
</tr>
<tr>
<td></td>
<td>• If indexing fails, response should contain an error code and message.</td>
</tr>
<tr>
<td>NF-SER-003</td>
<td><strong>Find Similar Pictures</strong></td>
</tr>
<tr>
<td></td>
<td>• The web service should be able to retrieve images that are similar to a given picture by comparing indexes.</td>
</tr>
<tr>
<td></td>
<td>• Minimum similarity for an index should be configurable.</td>
</tr>
<tr>
<td>NF-SER-004</td>
<td><strong>Identify Food in Pictures</strong></td>
</tr>
<tr>
<td></td>
<td>• The web service should be capable of generically identifying food items in pictures. (see image recognition requirements)</td>
</tr>
<tr>
<td></td>
<td>• If identification is successful, the response should consist of an OK status code and a JSON Map of food names and similarity indexes.</td>
</tr>
<tr>
<td></td>
<td>• If identification is unsuccessful, the response should consist of an OK status code but an empty JSON Map.</td>
</tr>
<tr>
<td></td>
<td>• If identification failed, the response should consist of an error code and error message.</td>
</tr>
<tr>
<td>NF-SER-005</td>
<td><strong>Fetch Nutrition Information for Generic Food Item</strong></td>
</tr>
</tbody>
</table>


The Caloricam web service should be able to fetch a list of food items and their nutrition information (name, calories, proteins, fats, carbs) given a generic food name.

- If successful, the response should consist of an OK status code and a JSON Array of Nutrition Information.
- If unsuccessful, the response should consist of an OK status code but an empty JSON Array.
- If the operation fails, the response should consist of an error code and error message.

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
</tr>
</thead>
</table>
| NF-SER-006     | **Link Image With Food**  
  - Each image uploaded to the server must be linked with a food name. |
| NF-SER-007     | **Error Logging**  
  - The web service should log errors and associated details. |

### 4.2.2.2 Admin Panel Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
</tr>
</thead>
</table>
| NF-ADM-001     | **Authentication Required**  
  - The admin panel requires a username and password for authentication. |
| NF-ADM-002     | **View Uploaded Images**  
  - Admin can view all uploaded images |
| NF-ADM-003     | **Delete Uploaded Images**  
  - Admin can delete uploaded images.  
  - When admin deletes image, accompanying record in database should also be deleted. |
| NF-ADM-004     | **View Indexes**  
  - Admin can view indexes created by LIRe Index Writer. |
| NF-ADM-005     | **Delete Indexes**  
  - Admin can delete indexes. |
| NF-ADM-006     | **View Logs**  
  - Admin is able to view logs. |
4.3 Image Recognition Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
</tr>
</thead>
</table>
| NF-IR-001      | **The Application is only expected to identify food generically, not specifically.**  
  - For example, the application should be able to identify a sandwich, but not that it is, say, a Turkey Club Sandwich.  
  - Precise identification is handled by the search functionality of the FatSecret web service. |
| NF-IR-002      | **Application should be able to recognise easily distinguishable food products with high accuracy.**  
  See table below |
| NF-IR-003      | **Application is expected to recognise a single item of food in a photo.**  
  - The photo should contain only one item of food. The application is not expected to distinguish between different items of food in a picture. |
| NF-IR-004      | **Application is expected to recognise the food if a close-up picture is taken.**  
  - The photo must be zoom-in on the food. The application is not expected to detect food from a large photo. |

The table below describes and exemplifies the sort of food items the application is expected to identify and not identify:

<table>
<thead>
<tr>
<th>Can Identify</th>
<th>Can’t Identify</th>
</tr>
</thead>
</table>
| **1. Large objects of distinctive shape and colour**  
  - Fruits (Apple, Orange, Banana)  
  - Vegetables (Carrots, Tomatoes) | **1. Small indistinguishable objects (in a cluster)**  
  - Peas  
  - Sweet Corn  
  - A single potato chip. |
| **2. Packaged Branded Food Item**  
  - A tin of Campbell's© Tomato Soup  
  - A branded juice product e.g. Minute Maid© Orange drink  
  - A box of lasagne sheets | **2. Open food item that is difficult to distinguish**  
  - A bowl of soup  
  - A glass of juice / cup of tea  
  - A slab of lasagne. |
4.4 Optional Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O-APP-001</td>
<td><strong>Augmented Reality</strong></td>
</tr>
<tr>
<td></td>
<td>• The user would be able point the device’s camera at the food.</td>
</tr>
<tr>
<td></td>
<td>• Frames from the camera feed would be uploaded to the web service for identification in real time</td>
</tr>
<tr>
<td></td>
<td>• Identification results would be overlaid on the screen, next to the item of food.</td>
</tr>
</tbody>
</table>
5. Professional, Legal, Ethical and Social Issues

This chapter lists the potential professional, legal, ethical and social issues involved in this project as well as measures that can be taken to mitigate such issues.

1. Stored User Data

This application stores the user’s age, weight, height and activity lifestyle in order to calculate the recommended daily calorie intake. Additionally, it maintains a database of food and calories consumed by the user each day.

This personal data needs to be kept secure and, according to the Data Protection Act (1998):

- User should be able to view and edit the data
- Data should be relevant
- Data should be kept for no longer than is needed.

In order to mitigate these issues, the application should

- Encrypt personal data
- Allow user to view and update his age, height, weight and activity statistics
- Only store relevant data
- Delete user data when the user deletes the application.
- Inform the user of what information is being stored.

2. Questionnaire Results

After the prototype has been developed, a group of users will be asked to test the application and then fill out a questionnaire concerning the usability and usefulness of the app.

The answers that the testers provide may include details about their age, weight, eating and exercising habits, which are of a personal nature. Therefore, the questionnaires will be anonymous and will be disposed of once the data has been collated.
6. Project Plan
This chapter will present the following:
- Project Deliverables and Tasks
- Project Schedule
  - Methodology
  - Evaluation Plan
- Risk Management and Mitigation.

6.1 Project Deliverables and Tasks
The deliverables for this project and their deadlines are as follows:

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research Report</td>
<td>15th April 2013</td>
</tr>
<tr>
<td>Prototype Application</td>
<td>15th August 2013</td>
</tr>
<tr>
<td>Dissertation</td>
<td>15th August 2013</td>
</tr>
<tr>
<td>Poster Presentation</td>
<td>22nd August 2013</td>
</tr>
</tbody>
</table>

The tasks involved in developing this prototype and their expected duration are:

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Task</th>
<th>Description</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Research Report</strong></td>
<td>Literature Review</td>
<td>- Research the relevant background (dieting, calorie counting).</td>
<td>2 weeks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Evaluation of existing applications.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Evaluation and selection of technologies</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Requirement Analysis</td>
<td>- Determine functional user requirements.</td>
<td>2 days</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Categorise minimum mandatory requirements and optional requirements.</td>
<td></td>
</tr>
<tr>
<td>Professional, Legal, Ethical and Social Issues</td>
<td></td>
<td>- Research compliance of application with data laws.</td>
<td>2 days</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Professional issues raised by the application e.g. data security, privacy</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Social impact of application</td>
<td></td>
</tr>
<tr>
<td>Planning</td>
<td></td>
<td>- List deliverables</td>
<td>3 days</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Determine Tasks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create Gantt Chart</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Consider Risks</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create Evaluation Strategy</td>
<td></td>
</tr>
<tr>
<td>Examination &amp; Vacation</td>
<td></td>
<td>4 weeks</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------</td>
<td>---------</td>
<td></td>
</tr>
</tbody>
</table>
| **Prototype** | Design | - Draw mock up of UI screens  
- Navigation between screens  
- Server-side architecture  
- Database structure | 1 week |
| | Implementation | - Server-side coding  
- Database development  
- Client side development  
- Optional features | 6 week |
| | Testing | - Iterative Testing  
- Functionality Testing  
- Smoke Testing | 7 week |
| | Evaluation | - Accuracy Testing  
- Questionnaires | 1 week |
| **Dissertation** | Documentation | - Documentation of each of step in the prototype stage | 1.5 weeks |
| **Poster Presentation** | Poster Preparation | - Creating a poster. | 3 days |
| | Slides Preparation | - Creating slides | 3 days |
| | Rehearsal | - Rehearsal slide delivery | 3 days |
|------------------------------------------------|----------|------|----------|----------|----------|----------|----------|----------|
| 1) Literature Review                            |          |      |          |          |          |          |          |          |
| 2) Requirement Analysis                         |          |      |          |          |          |          |          |          |
| 3) Professional, Legal, Ethical and Social Issues|          |      |          |          |          |          |          |          |
| 4) Project Plan                                 |          |      |          |          |          |          |          |          |
| 5) Submission of Research Report                |          |      |          |          |          |          |          |          |
| 6) Examination and Vacation                     |          |      |          |          |          |          |          |          |
| 7) Design                                       |          |      |          |          |          |          |          |          |
| 8) Implementation                               |          |      |          |          |          |          |          |          |
| 9) Testing                                      |          |      |          |          |          |          |          |          |
| 10) Evaluation                                  |          |      |          |          |          |          |          |          |
| 11) Prototype Developed                         |          |      |          |          |          |          |          |          |
| 12) Dissertation                                |          |      |          |          |          |          |          |          |
| 13) Poster Preparation                          |          |      |          |          |          |          |          |          |
| 14) Slides Preparation                          |          |      |          |          |          |          |          |          |
| 15) Rehearsal                                   |          |      |          |          |          |          |          |          |
| 16) Submission of Documentation                 |          |      |          |          |          |          |          |          |
| 17) Poster Presentation                         |          |      |          |          |          |          |          |          |
6.3 Methodology

In the implementation step, development of the software will be done using the Iterative Incremental Development methodology. In this process, the requirements are divided into increments and each increment goes through the cycle of requirement analysis, design, implementation and testing before work begins on the next increment.

This methodology has several benefits:

1. **Simplification**
   Requirements are divided into increments; increments are divided into tasks and these tasks can be set as daily goals.

2. **Better Risk Management**
   Each developed increment is tested before work begins on the next increment meaning that, even if the project is not completed in time, partial functionality can be provided.

3. **Better Software Structure**
   Each increment is planned so the end product is more organised and can be made more efficient.

The disadvantages, however, include:

1. **Good Planning is required**
   The whole requirements of the system need to be well understood and studied in order for a comprehensive plan to be drawn up.

2. **Planning is time consuming**

3. **Limited flexibility to changing requirements**
   For good planning, a good understanding of all the requirements is essential. If these requirements change then the previous plan may no longer apply.

In spite of these limitations, this methodology seems appropriate for the task at hand: the requirements are unlikely to change and even though it will take time to plan each iteration, a stable build will still be provided in the end.
6.4 Evaluation Strategy

In the evaluation step, the aim is to assess whether the application has met the intended objectives:

1. Identifies food or food products from photos and retrieves their calories.
2. Informs users whether their calorie intake is within the recommended daily range.
3. Allows users to share photos on a social network.
4. Motivates users to be conscientious and proactive about maintaining a healthy diet.
5. Encourages users to make healthier choices.

Based on these objectives, the evaluation strategy should determine:

1. Whether the application is reasonably accurate at identifying foods.
2. Is the application easy to use?
3. How useful would the application be to the user in his/her everyday life?
4. Does the user like the concept; does the user feel that such an application should exist?

In order to answer these questions, the following evaluation strategy has been devised:

1. Image Recognition Test

Once the application is developed, it will undergo an Image Recognition test. In this test, the application will be used to recognise a few items of foods in different configurations i.e. different orientations, lighting and positions. The tests will quantify the application’s ability to identify food items in everyday situations.

2. Questionnaires

The application will be distributed to a small set of testers who monitor their calorie intake on a regular basis either using a diary or an app. The testers will be asked to spend a few days testing the app in various scenarios.

After the test period is over, each tester will be asked to fill out a questionnaire. This questionnaire will ask the user to rate the usability, navigability and design of the application as well as how useful they think the application would be in their everyday lives and whether they believe such an application should exist.
6.5 Risk Management

The following table lists risks involved in the development of this project and steps taken to mitigate them.

<table>
<thead>
<tr>
<th>Risk</th>
<th>Probability</th>
<th>Impact</th>
<th>Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project isn’t completed in time.</td>
<td>High</td>
<td>High</td>
<td>- Incremental Iterative Development has been adopted so that partial functionality is provided even when time runs out.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Increments will focus on mandatory requirements first and then optional ones.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Server-side functionality will be implemented before client-side functionality.</td>
</tr>
<tr>
<td>Existing technologies are not capable of recognising foods (project's goals were too optimistic)</td>
<td>High</td>
<td>High</td>
<td>- As explained in the literature review chapter, the auto colour correlogram descriptor retrieves images based on similar shape, colour and texture. An item of food should be fairly recognisable based on these 3 attributes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Multiple images of the same item of food on the server will improve accuracy of matches.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Users will be instructed to take well lit and close up pictures to improve accuracy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- This application is a proof of concept and is not expected to have high accuracy.</td>
</tr>
<tr>
<td>High level of complexity</td>
<td>Moderate</td>
<td>Moderate</td>
<td>- 6 weeks have been scheduled for project implementation providing sufficient time to get help from online sources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- If a requirement is too complex, it will be simplified, documented and included in list of future extensions.</td>
</tr>
<tr>
<td>Lack of Familiarity with Technology (LIRε and Tomcat)</td>
<td>Moderate</td>
<td>Moderate</td>
<td>- Vacation time will be used for familiarisation with both technologies.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Technologies are well documented.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- A number of online forums are available from where assistance can be sought.</td>
</tr>
<tr>
<td>Users do not like the final application</td>
<td>Moderate</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>----------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>- A number of steps have been taken to improve accuracy and usability.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- Evaluation questionnaire will ask users to input suggestions for improvements.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. Design
This chapter presents the server side and client side design. The server side design includes server side architecture, web service methods, their parameters, their responses and database schema. The client side design lists activities in the android application and their responsibilities. It describes the navigation between the activities and includes UI mock-ups.

7.1 Server Side Design

7.1.1 Server Side Architecture

The diagram below illustrates the server side architecture. (A ‘test’ servlet exists but is not shown in the diagram):
### 7.1.2 Web Service Methods

The Web Service exposes 6 methods:

<table>
<thead>
<tr>
<th>Method Name</th>
<th>/upload</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTP Method</strong></td>
<td>POST</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>FileInputStream</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The Upload method is implemented in the Upload Servlet. This method receives image uploads from the client smartphone. The uploaded images are saved in the ‘images’ folder (unless an image with the same name already exists). Once uploading is complete, the request is automatically forwarded to the Index servlet.</td>
</tr>
<tr>
<td><strong>Response</strong></td>
<td>If file uploads successfully, response will be provided by indexer method. If upload fails, response will consist of error code and error message.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Name</th>
<th>/index?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HTTP Method</strong></td>
<td>GET</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>image_name: String (required)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The index method is implemented in the ‘Index’ servlet and serviced by the Indexer class. The method takes one parameter: the image name. The image, if found, is indexed using the auto colour correlogram method. The index files are saved in the indexes folder.</td>
</tr>
<tr>
<td><strong>Response</strong></td>
<td>If indexing is successful, then an OK response is returned to the client. Otherwise, the response consists of an error code and an error message.</td>
</tr>
<tr>
<td>Method Name</td>
<td>/identify?</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>GET</td>
</tr>
<tr>
<td>Parameters</td>
<td>image_name: String (required)</td>
</tr>
<tr>
<td></td>
<td>min_similarity: float (optional)</td>
</tr>
<tr>
<td></td>
<td>max_hits : int (optional)</td>
</tr>
<tr>
<td>Description</td>
<td>The identify method is implemented by the Identify servlet and is serviced by the Identifier class. This method takes one mandatory parameter, image name, and two optional parameters: minimum similarity index and maximum number of hits. If the image is found, this method will retrieve images similar to the given one. Images with a similarity index less than the one specified are ignored. For the remaining images, the corresponding food name is retrieved from the food database.</td>
</tr>
<tr>
<td>Response</td>
<td>If the method executes successfully, it will return an OK code and a JSON map of food name and similarity index. Otherwise, the response consists of an error code and an error message.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Name</th>
<th>/nutrition_info?</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP Method</td>
<td>GET</td>
</tr>
<tr>
<td>Parameters</td>
<td>food_name: String (required)</td>
</tr>
<tr>
<td>Description</td>
<td>The nutrition_info method is implemented by the GetNutritionInfo servlet and serviced by the FSWebService class. The method takes one mandatory parameter: food_name. This method invokes the ‘search’ method of the FatSecret Web Service passing the food name as the search term. If the food item is found in the FatSecret database, the response will consist of a list of matching food items and their nutrition information (name, type, URL, calories, carbohydrates, fats and proteins).</td>
</tr>
<tr>
<td>Response</td>
<td>If the method executes successfully, it will return a JSON Array of Nutrition Information. If the method executes unsuccessfully, it will return an empty JSON Array. If the method fails, the response will consist of an error code and error message.</td>
</tr>
<tr>
<td>Method Name</td>
<td>/link?</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------</td>
</tr>
<tr>
<td>HTTP Method</td>
<td>GET</td>
</tr>
<tr>
<td>Parameters</td>
<td><strong>image_name</strong>: String (required)</td>
</tr>
<tr>
<td></td>
<td><strong>food_name</strong>: String (required)</td>
</tr>
<tr>
<td>Description</td>
<td>Each image of food uploaded to the server must be linked with a food name. This should improve the accuracy of the application. The linking task is handled by the link method, which takes two mandatory parameters: image name and food name.</td>
</tr>
<tr>
<td>Response</td>
<td>If the update method executes successfully, it will return an OK response: <code>{&quot;message&quot;:&quot;Success&quot;,&quot;code&quot;:0}</code>. Otherwise, if execution fails, it will return an error code and error message.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method Name</th>
<th>/recognize?</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP Method</td>
<td>GET</td>
</tr>
<tr>
<td>Parameters</td>
<td><strong>image_name</strong>: String (required)</td>
</tr>
<tr>
<td></td>
<td><strong>min_similarity</strong>: float (optional)</td>
</tr>
<tr>
<td></td>
<td><strong>max_hits</strong>: int (optional)</td>
</tr>
<tr>
<td>Description</td>
<td>The recognize method is implemented by the Recognize servlet and serviced by the Identifier and FSWebService class. This is a convenience method: it rolls the functionality of the identify method and the nutrition_info method into a single call.</td>
</tr>
<tr>
<td>Response</td>
<td>If the method executes successfully, it will return a JSON Array of Nutrition Information. If the method executes unsuccessfully, it will return an empty JSON Array. If the method fails, the response will consist of an error code and error message.</td>
</tr>
</tbody>
</table>
7.1.3 Service Classes

Service classes implement the main functionality of a given Servlet. The benefit of creating service classes is that they provide loose coupling: the underlying implementation of how the task is performed can be changed without affecting the servlet code.

Indexer and Identifier classes
The indexer class indexes images using the auto colour correlogram method. The identifier class compares image indexes to retrieve similar images. Both classes use a singleton design pattern and the index/identify methods are synchronized in order to cope with the multi-threaded environment of Servlets (discussed in the Literature Review chapter).

FSWebService
The FSWebService presents Java equivalent of the FatSecret REST methods. The REST method responses are encoded in JSON and the factory design pattern is used to convert the JSON Responses to Java Response Objects.

Linker
The linker class facilitates the Link servlet. It links a given image with a given food name.

7.1.4 Admin Servlets

AdminLogin
The AdminLogin servlet loads the login JSP page and authenticates the username and password. If authentication is successful, a session is created for the administrator and the request is forwarded to the AdminPanel servlet.

AdminPanel
The AdminPanel servlet loads the admin JSP page if a valid session exists; otherwise it sends the request back to the AdminLogin servlet.

AdminLogout
The AdminLogout servlet terminates the administrator’s session if it exists, and forward the request to the AdminLogin servlet.

AdminImages
The AdminImages servlet loads the images JSP page and populates it with a list of images on the server, along with their size. It handles view and delete requests for each image.

AdminIndexes
The AdminIndexes servlet loads the indexes JSP page and populates it with a list of indexes in the indexes folder. It handles reindex and delete requests.
AdminLog
The AdminLog servlet prints out the contents of the Log file.

7.1.5 Response Syntax
The Calorie App Web Service encodes its responses in JSON. The response will consist of two parts: a status code and a message. If the method executes successfully, the status code is 0 and the message will contain the relevant results e.g. A JSON Array of nutrition information or a JSON Map of food names and similarity. If the method execution fails, the response will consist of an error code (any number greater than 0 ) and an error message

Success:
```json
{"message":"[{"id":36308,"fat":0.12,"name":"Cabbage","carbohydrates":5.58,"calories":24.0,"type":"Generic","proteins":1.44}],&quot;code&quot;:0}
```

Fail:
```json
{"message":"Insufficient arguments provided for this operation. : {string: image_name (required) },{float: min_similarity(optional) },{int: max_hits(optional)}],&quot;code&quot;:9}
```

7.1.6 Database Schema
There are three tables on the server side database: images, foods and admins. Each image entry can only have one food entry, however one food entry can serve multiple image entries.

The admin table records username and passwords of administrators. It is independent of the other two tables.

The image id is the IMEI number of the device that uploaded the image concatenated with the timestamp of when the picture was taken. The combination of these two properties results in a consistently unique value. The size attribute is the size in bytes of the image.

The food name is generic, for example: cheese or sandwich. This generic food name is sent to the FatSecret Search method, which returns a detailed list of specific foods and their nutrition information.
Figure 13 Server side database schema
7.2 Client Side Design

7.2.1 Activities
The Android Application consists of the following activities:

1. StartActivity
The start activity is the first activity to run when the user launches the application. It searches for the profile JSON file in the file system. If it is found, then it launches the MainMenuActivity; otherwise it will assume that this is the first time the user has launched the application and will launch the ProfileActivity in welcome mode.

2. ProfileActivity

The profile activity presents a form where the user enters his/her physical characteristics: age, sex, height, weight, activity and weight loss goal. Once the user submits the form, the recommended daily calorie intake is calculated. The data is encrypted and stored in a JSON file on the device and loaded each time the user launches the application. If the profile.json file isn’t found, then this activity is launched in welcome mode with default values for physical characteristics. The only difference between the welcome mode and normal mode is in the appearance. The user’s profile data is encapsulated by a Profile object. This object is shared globally by different activities.
3. MainMenuActivity

The main menu activity contains direct links to other activities. This design approach simplifies navigation, as mentioned in the Usability section of the Literature Review.

4. IdentifyTaskActivity

IdentifyTaskActivity is a superclass inherited by CameraActivity and FullScreenImageActivity. It uploads the displayed image to the Caloricam web service and calls the recognize method to get possible food matches and their nutritional information. Once a response is received from the web service, the food names are displayed on a list that dynamically appears over the child Activity. The last item on the list on the list is always “No match”. Clicking on a food item launches ResultsActivity, which shows a list of specific food items and their calories. Clicking on ‘No Match’ launches the SearchActivity.
5. CameraActivity

The camera activity displays a live camera feed to the user. When the user clicks on the ‘Take Picture’ button at the bottom of the screen, the picture is saved on the device with a filename of the device’s IMEI number appended with the timestamp of when the photo was taken. Once the picture is saved, two menu icons become visible: Get Calories and Try Again. Clicking on ‘Try Again’ resets the camera feed allowing the user to take another picture. Clicking on Get Calories uploads the image for identification and shows a list of possible food matches (see IdentifyTaskActivity)
5. ResultsActivity

As mentioned earlier, the IdentifyTaskActivity displays a list of generic food items in a list. When the user clicks on a food item, the ResultsActivity is launched. This activity displays specific food names and their nutrition information. The user selects an item from the list to add it to his journal. There is a confirmation dialog at the bottom, so that the user doesn't select an item accidentally while scrolling through the list.

The action bar consists of a ‘Done’ button and a ‘No match’ button. Clicking on the ‘Done’ button launches the JournalEntryActivity. Clicking on the ‘No match’ button launches the SearchActivity.
6. SearchActivity

The search activity presents a single text entry field where the user enters a food name. Upon submission, the activity makes a single nutrition info REST call to retrieve the nutrition information for the search item. If any results are found, they are displayed in an ExpandableListView. The user clicks on a food item in the list to add it to the journal. If the SearchActivity was launched from the ResultsActivity then, when the user clicks on an item to add it to the journal, an update call is made to link the food in the picture with the food item the user selected. The action bar in the search activity has a single item: "Done" which launches the JournalEntryActivity.
7. JournalActivity

The journal activity consists of a Calendar widget that displays the total calories consumed for each day of a given month. Days where the user consumed more than the recommended amount are highlighted red, while days where the user ate within the recommended amount are highlighted green. Clicking on a particular day launches the JournalEntry activity.

8. JournalEntryActivity

The JournalEntry activity lists all the meals logged on a particular day along with their calories. The uploaded picture is displayed as a thumbnail if it is available. The user can delete individual meal entries by swiping on the list items. The total calories consumed for the day are displayed at the bottom.
9. GalleryActivity

The Gallery Activity shows thumbnails of all pictures taken by the user. The gallery is scrollable in a vertical direction. Clicking on a thumbnail launches the FullScreenImageActivity.

10. FullScreenImageActivity

The FullScreenImage Activity shows a selected image in full screen. The action bar consists of a share button, a delete button and a get calories button. The share button allows the user to share the photo across all social applications installed on the device. The delete button deletes the photo and the get calories button uploads the image for identification and shows a list of possible food matches (see IdentifyTaskActivity)
11. AboutActivity

The AboutActivity states basic information about the application and includes step-by-step instructions on how to use the application.
7.2.2 Navigation State Diagram

The state diagram below summarizes navigation between activities. Pressing the device’s back button or the action bar’s ‘up’ button returns to the previous activity (except in the case of the StartActivity and GetCaloriesActivity).

![Application navigation state diagram]

Figure 14 Application navigation state diagram
### 7.2.3 Database Schema

The client side database consists of three tables: journals, images and nutrition. Each journal entry must have a nutrition information entry and may have an image entry. A nutrition information entry can serve many journal entries. There can only be one image corresponding to a single journal entry.

![Database Schema Diagram]

**Figure 15**

*Client side database structure*
8. Implementation

This chapter describes the implementation of the main features of the application. The server side section displays the code behind the indexing, identification and fetching nutrition information from the FatSecret Web service. The client side section covers the details of recognizing an image and linking a picture with a food name. The chapter concludes by discussing issues faced during the implementation stage.

8.1 Development Increments

The requirements were divided among a number of increments. The division is tabulated below. Core aspects of the application were developed first, while auxiliary and optional features were developed towards the final iterations.

<table>
<thead>
<tr>
<th>Iteration Number</th>
<th>Requirement ID</th>
<th>Tasks</th>
</tr>
</thead>
</table>
| 1                | NF-SER-001, NF-SER-002, NF-SER-003 | ● Create JSP page consisting of a form to upload images.  
● Create Upload servlet to receive uploaded images and store them in images folder.  
● Create Index servlet to index received image using the auto colour correlogram method.  
● Create Identify servlet to retrieve similar images for a given image  
● Host servlets online  
● Testing |
| 2                | NF-SER-004, NF-SER-007, NF-ADM-001, NF-ADM-002, NF-ADM-003, NF-ADM-004, NF-ADM-005, NF-ADM-006 | ● Create database tables for Images, Foods and Admin  
● Create Data Access Classes for each table  
● Create Admin panel with pages to view indexes, images and log.  
● Host changes  
● Testing |
| 3                | NF-SER-005, NF-SER-007 | ● Create API to interact with FatSecret REST service.  
● Create GetNutritionInfo Servlet to retrieve nutrition information from FatSecret Servlet.  
● Create Error Servlet to handle any unhandled exceptions |
| 4                | F-APP-004, F-APP-008, F-APP-009 | ● Create Camera Activity to take photos and save on device.  
● Upload photo to server and use Caloricam’s web service to identify food items and fetch nutrition information.  
● Display nutrition information results to user in ResultsActivity |
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
|   |   | - Create SearchActivity that displays nutritional information for a given food item.  
| 5 | F-APP-010, F-APP-011, F-APP-012 | - Create Database tables for Journal, Images and Nutrition Information.  
|   |   | - Create Data Access Objects for each table  
|   |   | - Create JournalActivity to view total calories consumed in a month  
|   |   | - Create JournalEntryActivity to view all meals for a given day (with thumbnail of picture, if available)  
|   |   | - Testing |
| 6 | F-APP-001, F-APP-002, F-APP-003, NF-APP-001 | - Create ProfileActivity where the user enters his/her physical characteristics.  
|   |   | - Create Welcome UI for ProfileActivity.  
|   |   | - Use profile data to highlight journal entries:  
|   |   |   - Green, if total calories consumed is within recommended intake.  
|   |   |   - Red, if total calories consumed is beyond recommended intake.  
|   |   | - Store profile data file in encrypted form. |
| 7 | F-APP-005, F-APP-006, F-APP-007 | - Create GalleryActivity that displays all pictures taken by the user.  
|   |   | - Create FullScreenImageActivity that displays a selected picture in full screen.  
|   |   | - Add controls to delete pictures.  
|   |   | - Add control to identify pictures from FullScreenActivity  
|   |   | - Implement ‘Share’ functionality so that pictures can be shared on social applications installed on device. |
| 8 | NF-SER-006, NF-APP-003, NF-APP-004 | - Link images with food names.  
|   |   | - Application Icon design  
|   |   | - Application UI design and improvements |
| 9 | NF-APP-005 | - Code Refactoring, Redesign |
8.2 Server Side Implementation

8.2.1 Indexing Images

Indexing is handled by the Index servlet and serviced by the Indexer class. The servlet receives the image name as a parameter. This image is loaded from the image directory. The Indexer class uses the LIRe library to create an auto colour correlogram index for the image. The index is saved in the indexes directory.

As the method involves resources being written and read by multiple servlets (and thereby, threads), it is necessary to provide a measure of thread safety. Thus the method itself is synchronized and the Indexer class implements a singleton pattern.

Indexing code is provided in Appendix IV.1

8.2.2 Identifying Images

Image identification is handled by the identify servlet and serviced by the Identifier class. The servlet receives three parameters: image name, minimum similarity and maximum hits. The identifier class uses the LIRe library to find all images similar to the given image. The food names associated with the matched images are read from the foods database. These food names are returned in the response along with their images’ similarity values.

Identification code is provided in Appendix IV.2

8.2.3 Getting Nutrition Information

Nutrition Information is fetched by calling the search method of the FatSecret REST API. This functionality is divided into two layers:

- FatSecret API Layer
- FatSecret Web Service Layer

FatSecret API Layer
Each FatSecret method call needs to be authenticated with an OAuth signature. The FatSecret Platform provides a Java API to perform this task. This class has been extended to perform two additional tasks:

- Generate the URL to invoke the REST methods.
- Send GET request and store the response.

Fat Secret Web Service Layer
The FSWebService class wraps the FatSecretAPI so that there is a Java method corresponding to each REST method.
The JSON responses received by the FatSecret API class are parsed into Java Objects. The JSON response will contain a foods entry if the REST method was successful or it will contain an error entry if something went wrong.

The factory pattern was used to parse the JSONs into appropriate Java object, illustrated by this UML diagram:

![UML Diagram](image)

Figure 16 Sever Side Factory Pattern for parsing FatSecret JSON

One of the issues faced during development was the unpredictability of the response from the FatSecret REST API. This issue is discussed in the Development Issues section at the end of this chapter.

Code samples from the FatSecretAPI class and FSWebService class can be found in Appendix IV.3 and 4 respectively.

8.2.4 Linking Image with Food

Linking of an image is handled by the Link servlet and serviced by the Linker class. The Link servlet takes two parameters: food name and image name. The Linker class creates a record for the food (unless it already exists) and then associates the food’s ID with the image.

Linking code can be found in Appendix IV.5.
8.3 Client-Side Implementation

8.3.1 Identifying Food

Identification of images is done in the IdentifyTask class and serviced by the CAWebService class. The CAWebService class wraps the Caloriam web service methods inside Java methods. As such, the class contains a Java equivalent for each method exposed by the service:

```java
public class CAWebService {
    private String WEBAPP_URL;
    private String SERVLET_UPLOAD;
    private String SERVLET_RECOGNIZE;
    private String SERVLET_LINK;

    public boolean upload(String image)
    public List<NutritionInfo> recognize(String image)
    public List<NutritionInfo> getNutritionInfo(String food)
    public boolean link(String food, String category);
}
```

The IdentifyTask class first calls the upload() method to upload the image and then calls the recognize() method to get a list of matching foods and their nutrition information (See Appendix IV.6).

The CAWebService class uses a factory pattern to parse the JSON responses into Java objects. The diagram below summarizes how the Abstract Factory Pattern is used on the client side:

![Diagram](image)

Figure 17 Client side factory pattern for parsing web service JSON
8.3.2 Search Activity
The SearchActivity consists of an input field at the top and a ViewSwitcher at the bottom. In the **Loading** mode, the SearchActivity shows a loading icon. The **Results** mode consists of an ExpandableListView that displays food names and their nutrition information.

When the user submits a search term in the input field, the CAWebService’s GetNutritionInfo method is used to fetch the nutrition information from the FatSecret Database. Once the data is received, the observer pattern is used to update the view mode and the list.

The getNutritionInfo method returns a list of Nutrition Information. This list is used to populate a SearchResultsModel. The model notifies its subscribers: the SearchActivity and the ExpandableListViewAdapter. Consequentially, the Activity changes its view mode from **Loading** to **Results**, and the ExpandableListView updates itself to show the new items in the list.

The diagram below illustrates how this design pattern is implemented:

![Observer pattern in SearchActivity](image)

8.3.3 Linking Image With Food
Linking is performed asynchronously in the LinkTask class (Appendix IV.8). The class is provided the image name and the generic food name by the invoking activity. The CAWebService’s link method is used to associate the food name with the image.

Each picture of food that the user uploads to the server must to be associated with a generic food name. This, in theory, should improve the accuracy of the application. If the food is successfully identified in the ResultsActivity, then the linking takes place within that activity. If the food item is not identified in the ResultsActivity (either the recognize method did not return any results or the user did not find a matching item) then the SearchActivity is launched where the user can enter the name of the food manually. In such a situation, linking is done in the SearchActivity.
The following section explains how the generic food name is determined in both activities. The necessity of determining a generic name is not ideal, and is discussed in the Development Issues section. Code samples can be found in Appendix IV.7

8.3.3.1 Linking in ResultsActivity

The IdentifyTask calls the CAWebService’s recognize method. This method returns a HashMap where the key is a generic food name and the value is a list of nutrition information for specific food items.

The generic food items are displayed in an overlay list in the invoking activity (either the CameraActivity or the FullScreenActivity).

![Possible Matches:](image)

Figure 19 Generic Results in FullScreenImageActivity

When the user clicks on a specific food item, the ResultsActivity is launched. This consists of an ExpandableListView. The parent item is the generic food name and the child item is the list of specific food items and their accompanying nutrition information.
When the user adds a child item to the journal, the corresponding parent item is set as the generic food name for the picture. In case of the image shown above, if the user clicks on any item in the list, the image will be linked with the name ‘Snack Crackers’.

### 8.3.3.2 Linking in SearchActivity

The ResultsActivity launches the SearchActivity if no match is found. The activity consists of an input field, where the user enters the food name, and an ExpandableListView, which displays the search results.

In general, the search term itself can be used as a generic food name e.g. ‘cheese’, ‘pizza’ etc. but the user might decide to truncate the search term to ‘ch’ or ‘pizz’. In order to resolve this problem, each part of the search term that matches with the selected item’s name is used as the generic food name. For example, if the user enters ‘Pizz’ and then adds ‘Pepperoni Pizza’ to the journal, then ‘Pizza’ will be used as the generic name. However, if the user enters ‘Ch’ and then selects ‘Cheese Chips’, ‘Cheese Chips’ will be used as the generic name.

If no part of the search term matches with the selected item, the selected item’s name will be used as the generic name.
This is not a very reliable approach and a further discussion is presented in the Development Issues section.

8.3.4 Encrypting Profile Data

In the Political, Legal, Ethical and Social Issues chapter, it was mentioned that the profile data must be stored in an encrypted format to guard the privacy of the user’s data. In keeping with this requirement, before the profile JSON is saved, it is encrypted using the AES algorithm with a 128-bit key.

The Encryption code is provided in Appendix IV.9

8.4 Development Issues

This section presents the issues faced during the development stage and their resolutions, if any.

1. Linking in SearchActivity

If a matching food is not found in the ResultActivity, the user can search for the food in the SearchActivity. The users search term is used as the generic food name.

This approach serves the purpose of attaching a food name to every uploaded image but it is not an ideal approach. A much better approach would be to create a database that caters to the specific needs of this application. Such a database would have a table of generic food names and another table of specific food items.

If the user could not find the matching food item in the IdentifyTask list, he would search for the item in the specific foods table. The generic food name corresponding to the selected item would then be associated with the image.

In the initial stages of development, it was felt that the application could be made to work without creating a dedicated nutrition information database. As development proceeded, it was realised that this was a mistaken assumption.

A dedicated database has not been implemented due to a shortage of time, but this solution would have to be implemented before the application is sold commercially.

2. Recognize Method

Originally, the recognize method did not exist. The client application would first upload the image, then call the identify method to get a list of foods and their similarity indexes. Finally, it would call the nutrition_info method for each of the foods whose similarity was greater than desired minimum.
It was later realised that this was an inefficient approach because the client device would have to process the responses for each of these requests. A much better idea would be to perform the identification and fetching tasks on the server and send the final outcome to the client. This would conserve the smartphone battery and improve performance.

Hence, the recognize method was introduced to make better use of resources on the client side.

3. Code Duplication between SearchActivity and ResultsActivity
SearchActivity and ResultsActivity have a lot in common: they both have a loading mode and a results mode that consists of an ExpandableListView. The code to toggle between view modes and to populate the ListView is identical. Thus, there is a great deal of code duplication between the two activities.

This issue can be resolved by creating a superclass containing the common methods. The SearchActivity and ResultsActivity would extend this superclass.

The solution has not been implemented because it would be time-consuming in both implementation and especially debugging.

4. Fetching Nutrition Information from FatSecret API
The FatSecret REST API uses the OAuth standard to authenticate method calls. Each request must pass an OAuth signature parameter determined by converting the parameters string in to base64 and then hashing it with the SHA-1 algorithm. The FatSecret Platform provides a Java API to handle this signature generation.

An issue that was faced was that only one of 3 calls would be accepted; the other two would be denied due to an invalid OAuth signature. This issue is most likely caused by the nonce parameter not being random enough. Making relevant changes to the randomisation code did not make much difference.

The temporary solution to the problem is to call the methods repeatedly until it is accepted but no more than 3 times.

5. Slow Indexing Issue
Indexing an image usually takes less than 5 seconds but one issue that was faced was that, after the final refactoring iteration, indexing began to take much longer: between 45 seconds up to a whole minute.
It was eventually determined that instead of indexing the uploaded image; all images in the images directory were being indexed. This was caused due to an accidental method deletion during the refactoring stage.
The method was recovered from the git repository and indexing returned to its normal speed.
6. Android Calendar Widget

In the design chapter, it was mentioned that the JournalActivity would display a calendar that would display the total calories eaten by the user on each day of that month. The Android framework does provide a calendar widget, but it does not allow for text to be added in the cells for a given date nor does it have a method to change the background of the cells.

In order to solve this problem, an online search was conducted to see if a library or open source code was available to perform the task. A library was found but it did not provide detailed documentation on how to install it. Code shared by other developers was available but it was neither well documented nor very extensible.

In the end, a Calendar widget was designed for this application using a GridLayout. The design goal for this widget was that it would be extensible, well-documented and adequately commented.

The Calendar widget consists of a Calendar view XML file, an adapter class and a CalendarEvent class. The Adapter class displays the list of calendar events on the calendar view. It also provides methods to determine the date on a given grid cell. The CalendarEvent class has methods to set the background colour, image and text for an event. These methods are designed to make the widget easy to use for other developers.

The code will be shared online on a public repository once the project has been completed.
9. Testing & Evaluation
This chapter lists the results of the functionality and image recognition test. It presents the feedback received from test users.

9.1 Functionality Tests

9.1.1 Testing Strategy

There are a number of testing techniques, such as:

- **Unit Testing**: This involves isolating a single module of code e.g. a class or a method and performing tests to ensure that the module works as intended.\(^{[30]}\)

- **Integration Testing**: This is when two or more related modules are combined to see that they function correctly as a group.\(^{[30]}\)

- **Black Box Testing**: This is when the functionality of the application is tested without looking at how it is implemented i.e. the software is tested from a user’s perspective.\(^{[30]}\)

- **White Box Testing**: As the name implies, white box testing is the opposite of Black-Box Testing: the internal code of the application is tested programmatically using a range of input values.\(^{[30]}\)

Testing was carried out at the end of each iteration, using all 4 of the techniques described above.

9.1.2 Test Cases

The following test cases go through a single run of the application from the users perspective i.e. uploading an image, getting the results, searching, adding item to journal, linking, and re-identification.

The test food item is a jar of Ovaltine, which has never been linked before.
<table>
<thead>
<tr>
<th>TC-001</th>
<th>Taking a picture.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>- User takes a picture of food item.</td>
</tr>
<tr>
<td><strong>Prerequisite</strong></td>
<td>Active Internet Connection</td>
</tr>
</tbody>
</table>
| **Expected Results** | - Picture is saved with name: IMEI number + Timestamp  
- Picture is uploaded to server.  
- Picture is saved in images directory on server.  
- Picture is indexed on server. |
| **Actual Results** | - The file is saved as 35937204245399320130730201132.jpg.  
- The image is uploaded to the server successfully.  
- The image is saved in images directory.  
(Screenshot: Appendix V.1) |
| **Status** | Pass |

<table>
<thead>
<tr>
<th>TC-002</th>
<th>Identification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>- User waits for the application to present possible food matches</td>
</tr>
</tbody>
</table>
| **Prerequisite** | - Active Internet Connection  
- An image must have been uploaded. |
| **Expected Results** | - Application presents list of food items.  
- Last item on list should be ‘No Match’  
- Application should not be able to identify item because it has not been linked before. |
| **Actual Results** | - A list of possible matches appears  
- The last item in list is ‘No match’  
- Item is not identified because it has not been linked before.  
(Screenshot: Appendix V.2) |
<p>| <strong>Status</strong> | Pass |</p>
<table>
<thead>
<tr>
<th>TC-003</th>
<th>Search</th>
</tr>
</thead>
</table>
| **Description** | - User clicks on ‘No match’ from the results of TC-002  
- User should be taken to SearchActivity  
- User enters ‘Ovaltine’ in search field  
- User adds Ovaltine to journal.  
- User clicks done. |
| **Prerequisite** | - Active Internet Connection  
- An image must have been uploaded.  
- Results of image identification are displayed. |
| **Expected Results** | - When user clicks on ‘No match’, SearchActivity is launched  
- Submitting ‘Ovaltine’ returns list of matching food items.  
- When user clicks on add to journal, image must be linked with photo  
- When user clicks on done, JournalEntryActivity is launched. |
| **Actual Results** | - Search Activity launched when ‘No match’ clicked.  
- Results for Ovaltine and nutrition information displayed.  
- Image is linked with ‘Chocolate Malt Mix’. The following is the log entry on the Admin servlet.  
(Screenshot: Appendix V.3) |
<p>| <strong>Status</strong> | Pass |</p>
<table>
<thead>
<tr>
<th>TC-004</th>
<th>Journal Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>- User clicks on ‘done’ from search activity.</td>
</tr>
</tbody>
</table>
| **Prerequisite** | - Active Internet Connection  
- An image must have been uploaded.  
- Item was added to journal from SearchActivity. |
| **Expected Results** | - User sees a list of meals eaten that day.  
- List of meal should include the Ovaltine entry with a thumbnail of photo and nutrition information.  
- Total calories should be displayed at bottom of screen.  
- Total calories sum should be correct.  
- If total calories are less or equal to the recommended daily calorie intake, background should be green. Otherwise, the background should be red. |
| **Actual Results** | - A list of meals eaten that day is displayed  
- Thumbnails are present where available  
- Total calories is displayed at the bottom of the screen  
- The sum of total calories is correct.  
- Current recommended daily calories: 2236.1 calories/day. Calories consumed are less than recommended amount so background is green.  
(Screenshot: Appendix V.4) |
| **Status** | **Pass** |

<table>
<thead>
<tr>
<th>TC-005</th>
<th>Gallery &amp; Full Screen</th>
</tr>
</thead>
</table>
| **Description** | - User clicks on ‘Gallery’ from main menu  
- User clicks on the picture of the Ovaltine jar taken in TC-001 |
| **Prerequisite** | - Active Internet Connection  
- An image must have been taken. |
| **Expected Results** | - User should find his image in the gallery.  
- When user clicks on image, it should be displayed in full screen. |
| **Actual Results** | - Image found in gallery.  
- Clicking on image loads it in full screen.  
(Screenshot: Appendix V.5) |
| **Status** | **Pass** |
### TC-006  Sharing

**Description**  
- User clicks on ‘share’ from FullScreenActivity.  
- User shares photo on Facebook.

**Prerequisite**  
- Active Internet Connection  
- Image was selected from gallery

**Expected Results**  
- A menu displays all available social media.  
- When user clicks on ‘Facebook’, a share dialogue appears on the screen  
- Shared photo and caption should appear on Facebook profile.

**Actual Results**  
- Clicking on share item displays list of all social media installed on application.  
- Selecting Facebook launches Facebook activity with attached picture. User can enter a caption and share.  
  (Screenshot: Appendix V.6)

**Status**  
Pass

### TC-007  Re-identification

**Description**  
- User clicks on ‘Get Calories’ from FullScreenActivity

**Prerequisite**  
- Active Internet Connection  
- Image was selected from gallery  
- TC-001

**Expected Results**  
- User should see a list of matching food items  
- The list should now include Ovaltine because the picture has been linked.

**Actual Results**  
- A List of results displayed.  
- ‘Chocolate Malt Mix’ i.e. Ovaltine, is now included in results.  
  (Screenshot: Appendix V.7)

**Status**  
Pass
<table>
<thead>
<tr>
<th>TC-008</th>
<th>Profile Activity</th>
</tr>
</thead>
</table>
| **Description** | - User clicks on ‘Profile’ from Main Menu.  
- User updates his physical characteristics |
| **Prerequisite** | None. |
| **Expected Results** | - User can set his age, sex, height, weight, weight loss goal, and activity life style factor.  
- Clicking on update button updates the daily recommended calories  
- Daily recommended calories value is correct according to Mifflin St-Jeor method.  
- Changes made to profile persist upon re-launch of application. |
| **Actual Results** | - User can change physical characteristics.  
- Clicking on update button updates recommended daily calorie intake.  
- Changes made to profile are persistent.  
- Calculation Validation:  
  B.M.R = 10(weight in kg) + 6.25(height in cm) − 5(age) + 5  
  B.M.R = 10(66 − 2) + 6.25(177) − 5(25) + 5 = 1626.25 kCal  
  Daily Calories = B.M.R x ActivityFactorDaily Calories = 1626.25 x 1.375 = 2236.093✓  
  (Screenshot: Appendix V.8) |
| **Status** | Pass |

### 9.2 Image Recognition Tests

Image recognition tests were done to see how well the application could identify items in different situations: different orientation, lighting, zoom and background.

The tests were performed on a chicken burger, fries and an orange soda. The pictures were taken beforehand, and the tests were done later. The original photos can be found in Appendix VI.1.
<table>
<thead>
<tr>
<th>TC-001</th>
<th>Image recognition with zoom-in.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Take zoomed-out pictures of test food items.</td>
</tr>
<tr>
<td><strong>Prerequisite</strong></td>
<td>Upload and link original pictures of test food.</td>
</tr>
</tbody>
</table>
| **Actual Results** | - The hamburger and fries were correctly identified.  
- The orange soda can was **not** identified correctly.  
- (Screenshots: VI.2) |
| **Results** | 2 / 3 |

<table>
<thead>
<tr>
<th>TC-002</th>
<th>Image recognition with zoom-out.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Take zoomed-out pictures of test food items with white backdrop.</td>
</tr>
</tbody>
</table>
| **Prerequisite** | Upload and link original pictures of test food.  
- Upload and link zoomed-in pictures |
| **Actual Results** | - The chicken burger and chips were recognised correctly.  
- The orange soda can was **not** identified correctly.  
- (Screenshots: VI.3) |
| **Status** | 2/3 |

<table>
<thead>
<tr>
<th>TC-003</th>
<th>Image recognition with orientation change.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Take pictures of food in an orientation that is different than in the original pictures.</td>
</tr>
</tbody>
</table>
| **Prerequisite** | Upload and link original pictures of test food.  
- Upload and link zoomed-in pictures  
- Upload and link zoomed-out pictures |
<p>| <strong>Actual Results</strong> | All items were identified successfully (Screenshots: VI.4) |
| <strong>Status</strong> | 3/3 |</p>
<table>
<thead>
<tr>
<th>TC-004</th>
<th>Image recognition with lighting change</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>- Take pictures of food in different lighting conditions</td>
</tr>
</tbody>
</table>
| **Prerequisite** | - Upload and link original pictures of test food.  
- Upload and link zoomed-in pictures.  
- Upload and link zoomed-out pictures.  
- Upload and link different-orientation pictures. |
| **Actual Results** | All Items were identified successfully (Screenshots: VI.5) |
| **Status** | 3/3 |

<table>
<thead>
<tr>
<th>TC-005</th>
<th>Image recognition with same meal but on different occasion.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>- Take pictures of the same meal but on a different occasion.</td>
</tr>
<tr>
<td><strong>Prerequisite</strong></td>
<td>- Upload and link original pictures of test food.</td>
</tr>
<tr>
<td><strong>Actual Results</strong></td>
<td>All items were identified successfully (Screenshots: VI.6)</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>3/3</td>
</tr>
</tbody>
</table>

**Results Summary:**

<table>
<thead>
<tr>
<th>Test</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom-out</td>
<td>2</td>
</tr>
<tr>
<td>Zoom-in</td>
<td>2</td>
</tr>
<tr>
<td>Orientation</td>
<td>3</td>
</tr>
<tr>
<td>Lighting</td>
<td>3</td>
</tr>
<tr>
<td>Different Meal</td>
<td>3</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>13/15 (87%)</td>
</tr>
</tbody>
</table>
9.2.1 Reflection
The food identification is done using the recognize method which takes, among others, a minimum similarity parameter. When the image recognition tests were initially done, the minimum similarity was set to 0.3. With this configuration, none of the tests passed. After investigating the issue, it was found that the foods were being recognised in the tests but the similarity was a really low value ranging from 0.05 to 0.15. The tests were repeated with a minimum similarity of 0 and this yielded much better results, as seen above.

The results of the image recognition test are very promising; they indicate that the application is capable of recognising food items in a variety of situations. This adds weight to the decision to use the auto colour correlogram descriptor.

One oddity in the results is that the tests usually failed in the case of recognising the soda can. One would expect that the soda can, which has a definite shape, colour and texture, would be more identifiable than the burger, with its complex shape and texture. The result, however, seem to indicate otherwise.
9.3 Evaluation
The evaluation strategy was described earlier in the Project Plan chapter. The strategy was twofold:

1. Perform image recognition test to see how well the application can identify food in various circumstances.
2. Let a test group use the application for a short period and then fill a questionnaire.

The results of the image recognition tests were covered in the previous section. This section will present the feedback from users who tested the application.

9.3.1 User Evaluation
User Evaluation was conducted by a group of 6 people: 4 males, 2 females. One male and one female candidate were above 30 years of age; the rest were between 20-30 years old. Each tester was given a demo on how to use the application and asked to use the application for 1-2 days, taking pictures of the food with the app. Users were encouraged to take close up pictures on a white background (as per the instructions in the About Activity). At the end of the two-day session, the users were asked to fill in an online survey hosted at www.freeonlinesurveys.com. The questions and their responses are tabulated below. The original questionnaire and response bar charts can be found in Appendix III.

<table>
<thead>
<tr>
<th>1. Age</th>
<th>13 – 20 years</th>
<th>0%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20 – 30 years</td>
<td>66.7%</td>
</tr>
<tr>
<td></td>
<td>30 – 50 years</td>
<td>33.3%</td>
</tr>
<tr>
<td></td>
<td>50 – 70 years</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>70 – 100 years</td>
<td>0%</td>
</tr>
<tr>
<td>2. Sex</td>
<td>Male</td>
<td>66.7%</td>
</tr>
<tr>
<td></td>
<td>Female</td>
<td>33.3%</td>
</tr>
<tr>
<td>3. Do you normally count the total calories you consume in a day in order to maintain a healthy diet?</td>
<td>Yes</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>100%</td>
</tr>
<tr>
<td>4. Do you use a smartphone application to assist you in keeping track of your daily calories intake?</td>
<td>Yes</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>No (If not please explain)</td>
<td>100%</td>
</tr>
</tbody>
</table>

Reasons given:
- “Too much work. I eat healthy and exercise regularly to maintain healthy diet.”
5. In total, how many times did you use the calorie camera feature of the application?  
<table>
<thead>
<tr>
<th>Frequency</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>0 %</td>
</tr>
<tr>
<td>1 – 2 times</td>
<td>33.3 %</td>
</tr>
<tr>
<td>3 – 5 times</td>
<td>66.7 %</td>
</tr>
<tr>
<td>6 – 10 times</td>
<td>0%</td>
</tr>
<tr>
<td>More than 10 times</td>
<td>0%</td>
</tr>
</tbody>
</table>

6. How frequently was the calorie camera able to correctly identify the food in the pictures?  
<table>
<thead>
<tr>
<th>Frequency</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>100%</td>
<td>0%</td>
</tr>
<tr>
<td>90 – 99%</td>
<td>33.3%</td>
</tr>
<tr>
<td>80 – 89%</td>
<td>66.7%</td>
</tr>
<tr>
<td>70 – 79%</td>
<td>0%</td>
</tr>
<tr>
<td>60 – 69%</td>
<td>0%</td>
</tr>
<tr>
<td>50 – 59%</td>
<td>0%</td>
</tr>
<tr>
<td>&lt; 50%</td>
<td>0%</td>
</tr>
</tbody>
</table>

7. How fast was the performance of the food identification process?  
<table>
<thead>
<tr>
<th>Speed</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Fast</td>
<td>33.3%</td>
</tr>
<tr>
<td>Fast</td>
<td>66.7%</td>
</tr>
<tr>
<td>Reasonable</td>
<td>0%</td>
</tr>
<tr>
<td>Slow</td>
<td>0%</td>
</tr>
<tr>
<td>Very Slow</td>
<td>0%</td>
</tr>
</tbody>
</table>

8. How easy was it for you to learn to use the application?  
<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Easy</td>
<td>33.3 %</td>
</tr>
<tr>
<td>Easy</td>
<td>66.7%</td>
</tr>
<tr>
<td>OK</td>
<td>0%</td>
</tr>
<tr>
<td>Difficult</td>
<td>0%</td>
</tr>
<tr>
<td>Very Difficult</td>
<td>0%</td>
</tr>
</tbody>
</table>

9. How easy was it for you to navigate between the different screens in the application?  
<table>
<thead>
<tr>
<th>Difficulty</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Easy</td>
<td>83%</td>
</tr>
<tr>
<td>Easy</td>
<td>17%</td>
</tr>
<tr>
<td>OK</td>
<td>0%</td>
</tr>
<tr>
<td>Difficult</td>
<td>0%</td>
</tr>
<tr>
<td>Very Difficult</td>
<td>0%</td>
</tr>
</tbody>
</table>

10. How would you rate the appearance and the 'Look & Feel' of the application?  
<table>
<thead>
<tr>
<th>Quality</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excellent</td>
<td>33.3%</td>
</tr>
<tr>
<td>Good</td>
<td>50%</td>
</tr>
<tr>
<td>OK</td>
<td>16.7%</td>
</tr>
<tr>
<td>Bad</td>
<td>0%</td>
</tr>
<tr>
<td>Very Bad</td>
<td>0%</td>
</tr>
</tbody>
</table>

11. Would you be interested in using this application to keep a track of your daily calories?  
<table>
<thead>
<tr>
<th>Interest</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>0%</td>
</tr>
<tr>
<td>Maybe</td>
<td>33.3%</td>
</tr>
<tr>
<td>No (Please explain)</td>
<td>66.7%</td>
</tr>
</tbody>
</table>

Reasons given:  
- “I don’t like taking pictures of food.”  
- “I use an app for running, not eating.”
12. Do you find the concept of the application (taking a picture of food to find out its calories) to be interesting?

<table>
<thead>
<tr>
<th>Yes</th>
<th>100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>0%</td>
</tr>
</tbody>
</table>

13. Finally, Do you have any comments, complaints or suggestions to improve the application?

- “The design for tablets could be improved.”
- “Serving size is not mentioned.”
- “I like the calorie journal idea. It’s fun to look at a picture of what you ate yesterday.”

9.3.2 Reflection

Overall, user reviews of the application have been quite positive. Unfortunately, a user who regularly counts calories could not be found to test the application.

The testers did not test the calorie camera feature as much as would be hoped but their responses indicate that the application was able to correctly identify the food in the photos between 80 and 89% of the time, which coincides with the results of the image recognition tests.

Most testers agreed that the user experience was good: the application was quick, easy to navigate through and well designed. One user mentioned that the UI was not optimised for tablets, which is a valid point. It was decided in the project planning stage that designing a consistent UI for the majority of Android devices would be a very time consuming task, so the focus was largely put on smartphones. This issue can be remedied in the future.

Another issue mentioned by one of the testers is that the serving size is not listed for some of the food items, leading to an inaccurate calorie record. This is a serious issue. Serving size information is received from the FatSecret response but it is not parsed.

Unfortunately, even though all of the testers felt the concept of the application was good, it did not motivate any of the testers to use the application to keep track of their calories, which is one of the original aims of the project. However, it must be noted that none of the testers count calories in the first place. It is quite possible that the opinion of those who do routinely count calories is different and more favourable.

Another very useful input by one of the testers is that he used an app to track his exercise but not his food. Daily exercise logging could be a future feature for the project, which would appeal to a larger audience.

Overall, the user review has been positive and constructive. The application has met with 4 of its 5 objectives. Though it didn’t meet the objective of encourage users to be more diligent about the calories they consume, the user reviews have given positive ideas by which this can be achieved in future versions.
9.3.3 Requirements Evaluation
This sections ticks off those requirements that were met and those that failed based on the results of the Accuracy Tests and User Evaluation questionnaires.

**Functional Requirements**

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>F-APP-001</td>
<td>Create Profile</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-002</td>
<td>View Profile</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-003</td>
<td>Edit Profile</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-004</td>
<td>Take Photos</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-005</td>
<td>View Photos</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-006</td>
<td>Delete Photos</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-007</td>
<td>Share Photos</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-008</td>
<td>Identify Food in Photos</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-009</td>
<td>Search Food in Database</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-010</td>
<td>Add Entry to Journal</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-011</td>
<td>View Entries in Journal</td>
<td>✓</td>
</tr>
<tr>
<td>F-APP-012</td>
<td>Delete Entries from Journal</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Non-Functional Client Side Requirements**

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF-APP-001</td>
<td>Profile data must be encrypted</td>
<td>✓</td>
</tr>
<tr>
<td>NF-APP-002</td>
<td>Easy and Intuitive Navigation</td>
<td>✓</td>
</tr>
<tr>
<td>NF-APP-003</td>
<td>Intuitive Design</td>
<td>✓</td>
</tr>
<tr>
<td>NF-APP-004</td>
<td>Simple UI</td>
<td>✓</td>
</tr>
</tbody>
</table>
Non-Functional Server Side Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>NF-SER-001</td>
<td>Upload Pictures</td>
<td>✓</td>
</tr>
<tr>
<td>NF-SER-002</td>
<td>Index Pictures</td>
<td>✓</td>
</tr>
<tr>
<td>NF-SER-003</td>
<td>Find Similar Pictures</td>
<td>✓</td>
</tr>
<tr>
<td>NF-SER-004</td>
<td>Identify Food in Pictures</td>
<td>✓</td>
</tr>
<tr>
<td>NF-SER-005</td>
<td>Fetch Nutrition Information for Generic Food Item</td>
<td>✓</td>
</tr>
<tr>
<td>NF-SER-006</td>
<td>Link Image With Food</td>
<td>✓</td>
</tr>
<tr>
<td>NF-SER-007</td>
<td>Error Logging</td>
<td>✓</td>
</tr>
<tr>
<td>NF-ADM-001</td>
<td>Authentication Required</td>
<td>✓</td>
</tr>
<tr>
<td>NF-ADM-002</td>
<td>View Uploaded Images</td>
<td>✓</td>
</tr>
<tr>
<td>NF-ADM-003</td>
<td>Delete Uploaded Images</td>
<td>✓</td>
</tr>
<tr>
<td>NF-ADM-004</td>
<td>View Indexes</td>
<td>✓</td>
</tr>
<tr>
<td>NF-ADM-005</td>
<td>Delete Indexes</td>
<td>✓</td>
</tr>
<tr>
<td>NF-ADM-006</td>
<td>View Logs</td>
<td>✓</td>
</tr>
</tbody>
</table>

Optional Requirements

<table>
<thead>
<tr>
<th>Requirement ID</th>
<th>Requirement Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>O-APP-001</td>
<td>Augmented Reality</td>
<td>✗</td>
</tr>
</tbody>
</table>

It can be seen from the table above that all mandatory requirements have been adequately met. The Augmented Reality requirement could not undertaken due to a shortage of time. However, the infrastructure necessary to implement this feature is mostly present and the feature can be added in future versions.
10. Conclusion
This concluding chapter will list the limitations of the final project and how it may be improved in future versions. It will surmise what has been learnt and accomplished from working on this project.

10.1 Limitations
Though most of the requirements were met, the software does have limitations. These are listed below:

<table>
<thead>
<tr>
<th>Application Crashes</th>
<th>Test users did report that the application crashed in the following scenario:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. The user selects a picture from the gallery.</td>
</tr>
<tr>
<td></td>
<td>2. The user clicks on the ‘get calories’ button.</td>
</tr>
<tr>
<td></td>
<td>3. The user presses back to return to the gallery.</td>
</tr>
<tr>
<td></td>
<td>4. The user selects a different picture.</td>
</tr>
<tr>
<td></td>
<td>At this stage, the two screens seem to overlap each other and the</td>
</tr>
<tr>
<td></td>
<td>application crashes. It is unclear what is causing the issue, and it</td>
</tr>
<tr>
<td></td>
<td>has yet to be resolved.</td>
</tr>
<tr>
<td></td>
<td>Another user reported a crash when adding an item to his journal.</td>
</tr>
<tr>
<td></td>
<td>The scenario could not be reproduced and so it is unknown what</td>
</tr>
<tr>
<td></td>
<td>caused it.</td>
</tr>
<tr>
<td></td>
<td>Users did not report frequent crashes, excluding the reproducible</td>
</tr>
<tr>
<td></td>
<td>issue described earlier.</td>
</tr>
</tbody>
</table>

| Unsorted Results    | The generic food name list that is displayed after a picture is         |
|                     | recognised is unsorted. This means that the food with the highest      |
|                     | similarity might not be at the top of the list. The results would be    |
|                     | more intuitive if the list were sorted according to food similarity.   |

| Serving size notlisted for all food items | The FatSecret Web Service does send serving size with the calorie       |
|                                          | information but it is not displayed to the user.                        |

| Camera does not have a lot of features | The calorie camera does not have a lot of useful features such as        |
|                                         | zoom-in, zoom-out, focus and the ability to shoot in landscape mode.    |
|                                         | This was done to save time and focus on completing the essential        |
|                                         | requirements.                                                          |
|                                         | Adding these features might improve the reliability of the              |
|                                         | application and will provide a more engaging experience for users        |
|                                         | who enjoy taking pictures.                                             |
UI is not designed for landscape mode.

Android applications can be used in both portrait and landscape mode. Some users prefer to type in landscape mode where the keyboard is more spaced out. The framework itself encourages app developers to make both orientations available. This application, however, is only available in portrait mode.

Incorporating landscape mode would require some extra code in order to persist data during orientation change. This was avoided to save time.

Consistent UI on multiple devices

Android devices come in a variety of screen sizes. It is a time-consuming challenge to design a UI that is consistent across a range of screen sizes. This application was developed with only a 4-inch touch device in mind (because this is the device the application was tested with).

10.2 Future Improvements

In addition to resolving the limitations listed earlier, a number of enhancements can be added to the application. These are listed below:

Dedicated Nutrition Database

This idea was mentioned earlier in the Implementation chapter, in the Development Issues section. The application associates each photo with a generic food name. This name is used to fetch a list of specific food items and their information from the FatSecret web service.

The generic food name is ‘guessed’ from the users search term and search results selection. This is sort of a hacky solution.

It would be better to develop a nutrition database dedicated to the needs of the application. This would consist of a table of generic food names that references a table of specific food items and their nutrition information.

This would be a far more reliable and scalable approach.

Cropping Functionality

At the moment, the user is encouraged to take pictures up close and with a white backdrop. Ideally, the user should not have to do any ‘housework’ for the application.

As such, a cropping feature could be added so that the user can take pictures of food however he likes. The user can then encircle all the items of food. Each food item would be cropped, uploaded and recognised and the results would appear next to each food item in the photo.
Exercise Journal

One of the test users mentioned that he used an app to monitor his exercise regimen, but not his eating habits. This feature can be added to the application whereby the user can monitor and log calories burnt through exercise on a given day. This would expand the potential market for this application.

Other smartphone platforms.

Another way to expand the market for this application is to port it to other smartphone platforms e.g. iOS and Windows Phone.

This could either be done by writing the application using a cross-platform framework, for example PhoneGap, or by developing native versions for each platform.

Online profiles

At the moment, the application stores user’s profile details and journal in the device’s file system. In future versions, this data can be stored online and retrieved via the web service.

The benefit of this feature is that the users data would be synchronised between multiple devices. It would also reduce the threat of data being lost.

Web Service Authentication

If the online profile feature is implemented, the Caloriecam web service will need to authenticate requests in order to protect user data.

Augmented Reality

Augmented Reality was an optional requirement for this project that could not be fulfilled. It could be implemented in future versions.

10.3 Lessons Learnt

This project has been a valuable learning experience. Several lessons have been learnt that can be applied to future projects, such as:

REST Web Services

This is the first time I have used a web service for one of my projects and I’m certain it won’t be the last.

Web services are particularly useful when developing an application that is to be ported to several platforms. Having to rewrite the same code in different languages for different platforms can be rather cumbersome.

Using a web service allows the core functionality to be implemented and optimised in one area and shared by different applications running on different platforms.

This is particularly advantageous in the burgeoning field of Smartphone apps, where the major smartphone vendors each have a different language for their platforms: Objective-C for
<table>
<thead>
<tr>
<th><strong>iPod, C# for Windows Phone and Java for Android.</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Servlets</strong></td>
</tr>
</tbody>
</table>
| **Design Patterns** | Despite having developing for a number of years, I’d only heard of design patterns recently but their benefits were immediately apparent. Design patterns provide a very clean and elegant way to write code for complex problems. I had only studied a few design patterns for this project: Model-View-Controller, Observer Pattern, Data Access Object Pattern and the Factory Pattern. 

After the completion of this project, I intent to learn and implement others. |
| **Image Recognition** | Conducting research on the Image Recognition helped me to understand the apparent magic behind it. I had a few personal ideas about how it might work but reading about the efficient ways that have been developed to perform the task has been enlightening. It’s a good feeling to services such as Google Images with some idea of what’s going on in the background. |
| **Planning & Agile Methodology** | This is the largest project I’ve ever worked on and it would not have been successful had it not been thoroughly planned. The project taught me good planning techniques: The requirements were broken down into iterations, the iterations into major tasks and the major tasks into daily goals. This proved to a very productive method of development. |
| **User Attitudes** | The user response was one of the most interesting parts of the project. It was interesting to see how different users, with different levels of technical competence, used the application differently. This exercise has given me a broader idea of user attitudes and how to design applications that are easy to use for people with different levels of technical skill. 

For instance, older users prefer to have some sort of written instruction on what should be done on each screen whereas younger users can usually figure it out on their own. Older users tend to get more confused by icons than younger users. |
| **GIMP** | I designed most of the icons in the application. This required learning to use a scalable vector drawing software. I chose GIMP because it’s free and there are several tutorials available. |
10.4 Conclusion

This report documented the requirement analysis, design, implementation and evaluation of the development of a new kind of calorie recording application: one that uses image recognition to identify food. The goals of this project were to make calorie recording a more enjoyable and intuitive task and thereby encourage individuals to monitor their diet and make healthier choices.

All the mandatory requirements of the project were met and the results of functionality tests indicate that the image recognition feature is quite successful; user feedback has shown that they found the concept of the application to be an interesting one. The future improvements suggested in this chapter should enhance the appeal and marketability of the application even further.

The project is the largest I have ever undertaken and it has proven to be an invaluable learning experience in terms of planning, problem solving and understanding user perspectives. The lessons I’ve learnt from this project are ones that I shall implement in many projects to come.
Appendix I: Colour Spaces

A colour space is a way of describing colours numerically in terms of proportion of their constituents. The constituents could be RGB: Red, Green and Blue or HSV: Hue, Saturation and Value.

A colour space can be viewed as an x-y-z graph where x represents red, y represents blue and z represents green. All possible colours have a unique position in this 3-d graph corresponding to different amounts of the primary constituents. [31]

![RGB Colour Space](image)

**Figure 21 RGB Colour Space** [31]

**RGB Colour Space**
The RGB Colour space represents a colour in terms of different amounts of Red, Green and Blue.

**HSV Colour Space**
The HSV Colour space represents colours in terms Hue, Saturation and Value.

**Hue**  
Describes a colour in terms of its angular position in a colour wheel [32]

![Colour Wheel](image)

**Figure 22 Colour Wheel** [33]
<table>
<thead>
<tr>
<th><strong>Saturation</strong></th>
<th>Numeric representation of distance of the colour from grey. Saturated colours tend to be pure whereas de-saturated colours are grey.[^{[32]}]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Value</strong></td>
<td>Numeric measure of how bright the colour is. [^{[32]}]</td>
</tr>
</tbody>
</table>
Appendix II: Histograms

A histogram is a bar chart that displays how frequently a value occurs in a data set. Each value in the data set is given a bin on the x-axis and the y-axis value indicates its frequency.

Consider the following tabulated results of a test (with a maximum score of 10):

<table>
<thead>
<tr>
<th>Score</th>
<th>Number of Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

A histogram would show the frequency of each test score

![Figure 23 Histogram of test scores](image-url)
**Quantisation**
Quantisation is when a large range of values is converted to a small range of values. This reduces the size of the histogram by summarising the information. The student table shown above can be quantised into two bins as follows:

Original number of bins : 10
New number of bins : 2
Range of each bin: $\frac{10}{2} = 5$.

![Quantised Histogram](image)

**Normalisation**
Normalisation is when the count of each bin is divided by the total number of counts so that the sum of all frequencies is one. Such a histogram shows the relative frequency of the bins.

The histogram above can be normalised as follows:
Count of Bin 1: 3
Count of Bin 2: 17
Total Count : 20

![Normalised Histogram](image)
Appendix III: Questionnaire & Responses

The original user evaluation questionnaire and the bar chart of responses are shown below. The questionnaire was hosted on [www.freeonlinesurveys.com](http://www.freeonlinesurveys.com).

---

**Caloricam User Experience**

Thank you for trying out Caloricam!

Please fill out this questionnaire to let us know about your experience.

---

1. **Age**
   - 13-20
   - 20-30
   - 30-50
   - 50-70
   - 70-100

2. **Sex**
   - Male
   - Female

3. Do you normally count the total calories you consume in a day in order to maintain a healthy diet?
   - Yes
   - No (Please Specify): ______________________

4. Do you use a smartphone application to assist you in keeping track of your daily calories intake?
   - Yes
   - No

5. In total, how many times did you use the calorie camera feature of the application?
   - Never
   - 1-3 times
   - 3-5 times
   - 5-10 times
   - more than 10 times

6. How frequently was the calorie camera able to correctly identify the food in the pictures?
   - 100%
   - 90-99%
   - 80-89%
   - 70-79%
   - 60-69%
   - 50-59%
   - <50%

7. How fast was the performance of the food identification process?
   - Very Fast
   - Fast
   - Reasonable
   - Slow
   - Very Slow
8. How easy was it for you to learn to use the application?
   - Very Easy
   - Easy
   - OK
   - Difficult
   - Very Difficult

9. How easy was it for you to navigate between the different screens in the application?
   - Very Easy
   - Easy
   - OK
   - Difficult
   - Very Difficult

10. How would you rate the appearance and the "Look & Feel" of the application?
    - Excellent
    - Good
    - OK
    - Bad
    - Very Bad

11. Would you be interested in using this application to keep a track of your daily calories?
    - Yes
    - Maybe
    - No (Please explain): ______________

12. Do you find the concept of the application (taking a picture of food to find out its calories) to be interesting?
    - Yes
    - No

13. Finally, Do you have any comments, complaints or suggestions to improve the application.

   ______________
1. Age

![Age Chart]

2. Sex:

![Sex Chart]
3. Do you normally count the total calories you consume in a day in order to maintain a healthy diet?

Reasons given:
- “Too much work. I eat healthy and exercise regularly to maintain healthy diet.”

4. Do you use a smartphone application to assist you in keeping track of your daily calories intake?
5. In total, how many times did you use the calorie camera feature of the application?

6. How frequently was the calorie camera able to correctly identify the food in the pictures?
7. How fast was the performance of the food identification process?

8. How easy was it for you to learn to use the application?
9. How easy was it for you to navigate between the different screens in the application?

10. How would you rate the appearance and the 'Look & Feel' of the application?
11. Would you be interested in using this application to keep a track of your daily calories?

Reasons given:
- “I don’t like taking pictures of food.”
- “I use an app for running, not eating.”
12. Do you find the concept of the application (taking a picture of food to find out its calories) to be interesting?

13. Finally, Do you have any comments, complaints or suggestions to improve the application?
   - “The design for tablets could be improved.”
   - “Serving size is not mentioned.”
   - “I like the calorie journal idea. It’s fun to look at a picture of what you ate yesterday.”
Appendix IV: Source Code

1. Indexing images on server side.

```java
/**
 * @param imageFile image to index
 * @param indexesDir directory where index will be stored.
 * @return true if image was indexed successfully.
 * @throws IOException
 */
public boolean indexImage(File imageFile, File indexesDir) throws IOException {
    synchronized(lock) {
        if(imageFile != null && !imageFile.isFile())
            throw new IllegalArgumentException("The Image File provided is not a file.");
        if(indexesDir != null && !indexesDir.isDirectory())
            throw new IllegalArgumentException("The indexes File provided is not a directory.");

        ArrayList<String> images = new ArrayList<String>();
        images.add(imageFile.getAbsolutePath());
        return this.indexImages(images, indexesDir);
    }
}

/**
 * Indexes all images using selected document builder.
 * @param images paths of images to index.
 * @param indexesDir directory where generated indexes are to be stored
 * @return true, if all images were indexed successfully
 * @throws IOException
 */
private boolean indexImages(ArrayList<String> images, File indexesDir) throws IOException {
    boolean success = false;

    IndexWriterConfig config = new IndexWriterConfig(Version.LUCENE_40, new WhitespaceAnalyzer(Version.LUCENE_40));
    IndexWriter indexer = null;
    Directory indexesDirectory = null;
    try {
        // Initialise indexer with output location.
        indexesDirectory = FSDirectory.open(indexesDir);
        indexer = new IndexWriter(indexesDirectory, config);

        for (Iterator<String> it = images.iterator(); it.hasNext();)
```
{  
  String imagePath = it.next();  
  // load image  
  BufferedImage img = ImageIO.read(new FileInputStream(imagePath));  
  // create lucene document containing descriptor  
  Document document = this.getDocumentBuilder().createDocument(img, imagePath);  
  // index document  
  indexer.addDocument(document);  
  
  success = true;  
}  

} catch (FileNotFoundException e)  
{
  throw e;
} catch (LockObtainFailedException e)  
{
  logger.error("Indexer. Lock ObtainFailedException. Will Unlock", e);
  if (indexesDirectory != null)  
  {
    // force write unlock on indexes directory.  
    IndexWriter.unlock(indexesDirectory);
  }
} catch (IOException e)  
{
  throw e;
} finally  
{
  // safely close index writer.
}

return success;
2. Identifying images on server side.

```java
/**
 * @param imageUri path to image file
 * @param indexesDir path to indexes file
 * @param maximumHits maximum number of hits to be determined
 * @return Map of image name (key) and similarity index (value).
 * @throws IOException
 */
private Map<String, Float> findSimilarImages(String imageUri, String indexesDir, int maximumHits)
    throws IOException {
  Map<String, Float> imageSimilarityMap = new HashMap<String, Float>();

  //load file into image.
  BufferedImage image = ImageIO.read(new File(imageUri));

  //load indexes
  IndexReader reader = DirectoryReader.open(FSDirectory.open(new File(indexesDir)));

  //create searcher to compare auto color correlogram descriptors
  ImageSearcher searcher = ImageSearcherFactory.createAutoColorCorrelogramImageSearcher(maximumHits);

  //search for images similar to given image
  ImageSearchHits hits = searcher.search(image, reader);

  int limit = hits.length() > maximumHits? maximumHits : hits.length();

  for (int i = 0; i < limit; i++)
  {
    String fileName = hits.doc(i).getValues(DocumentBuilder.FIELD_NAME_IDENTIFIER)[0];
    fileName = fileName.substring(fileName.lastIndexOf(File.separator)+1);
    imageSimilarityMap.put(fileName, hits.score(i));
  }

  return imageSimilarityMap;
}
```
private String doRestCall(Method method, HashMap<Parameter,String> methodParameters, Format responseFormat)
  throws IOException
{
  HashMap<String, String> requestParameters = new HashMap<String, String>();
  // add method name to parameters
  requestParameters.put(Parameter.METHOD.getName(), method.getName());
  // add response type to parameters
  requestParameters.put(Parameter.FORMAT.getName(), responseFormat.value);

  // add method parameters to request parameters
  for (Entry<Parameter, String> e : methodParameters.entrySet())
  requestParameters.put(e.getKey().getName(), e.getValue());

  // append methodParameters and request parameters to url
  String urlWithParameters = addParametersToUrl(URL, requestParameters);

  // create Result object which will store URL with oauth signature
  Result result = new Result();

  // generate OAuth Signature
  OAuthBase oauthenticator = new OAuthBase();
  URL url = new URL(urlWithParameters);
  oauthenticator.generateSignature("GET", url, consumerKey, consumerSecret, null, null, result);

  // make rest call
  String signedUrl = result.getUrl();
  return HttpClient.get(signedUrl);
}
IV.4 FatSecret Web Service Layer

/**Constructor
 * @param consumerKey private key for FatSecret REST API
 * @param sharedKey public key for FatSecret REST API
 */
public FSWebService(String consumerKey, String sharedKey)
{
  this.apilayer = new FatSecretAPI(consumerKey, sharedKey);
}

/**Searches for food in FatSecret food database
 * @param foodName food name
 * @return list of nutrition info for matching items
 * @throws IOException
 * @throws ParseException
 */
public List<NutritionInfo> searchFood(String foodName)
throws IOException, ParseException
{
  //Weird Problem:
  //Due to the nonce, the OAuth signature isn't always accepted.
  //Multiple tries need to be made.
  List<NutritionInfo> nutritionInfoList = new ArrayList<NutritionInfo>();
  for (int i = 0; i < NUM_TRIES; i++)
  {
    String json = this.apilayer.foodsSearch(foodName);
    FSAbstractResponseFactory factory = FSResponseFactory.getFactory(json);
    FSRresponse response = factory.createResponseFromJSON(json);
    if (response instanceof FSFoods)
    {
      nutritionInfoList = ((FSFoods) response).getNutritionInfoList();
      break;
    }
  }

  return nutritionInfoList;
}
IV.5 Linking Images on Server side

```java
/**
 * Links food in image with given food name.
 * @param foodName
 * @param imageFile
 * @return true if linking was succesful.
 * @throws DataAccessObjectException
 */
public boolean linkImageWithFood(String foodName, File imageFile) throws DataAccessObjectException
{
    boolean success = false;
    String imageName = imageFile.getName();

    //get id of food in food database
    //getFoodId will create record if it doesnt already exist.
    //likewise for imageId.
    long foodId = getFoodId(foodName);
    String imageId = getImageId(imageFile);
    if (foodId != -1 && imageId != null)
    {
        //set image.foodId = food.id
        success = linkImageWithFood(imageName, foodId);
    }
    return success;
}
```
/**Invokes the Calorie REST service recognize method.
 *  
 *  @param imageName name of image to be recognized
 *  @param minSimilarity minimum similarity index for matching images
 *  @param maximumHits maximum number of results
 *  @return Map with key: generic food name, value: specific food items and their
 *  nutrition info
 *  @throws IOException
 */

public static Map<
  String,
  List<
  NutritionInfo
  >>
  recognize
  ( String imageName,
  float
  minSimilarity,
  int
  maximumHits
  )
  throws
  IOException {

  Map<
  String,
  List<
  NutritionInfo
  >>
  result
  =
  new
  HashMap<string,List<NutritionInfo>>();

  Map<string,string> params =
  new
  HashMap<string,string>();
  params.put ( PARAM_IMAGE_NAME, imageName );
  params.put ( PARAM_MIN_SIMILARITY, ""+minSimilarity );
  params.put ( PARAM_MAX_HITS, ""+maximumHits );
  String url = HttpClient.appendGetParameters(WEBAPP_URL+SERVLET_RECOGNIZE,params);

  String json = HttpClient.get ( url );
  Log.i (TAG, json );

  try
  {
    CAAbstractResponseFactory factory =
    new
    CARecognizeResponseFactory ();
    CARecognizeResponse response = (CARecognizeResponse)
    factory.createResponseFromJSON ( json );
    result = response.getNutritionInfo ();
  }
  catch ( ParseException e )
  {
    Log.e ( TAG, "" + e.getMessage ( ) );
  }

  return result;
}
IV.7 Determining Generic Food Name

Determining generic food name in ResultsActivity

```java
/**Callback when a list item is clicked.
 *
 */
class OnListItemClicked implements ExpandableListView.OnChildClickListener {
    @Override
    public boolean onChildClick(ExpandableListView parent, View v, int groupPosition, int childPosition, long id) {
        NutritionInfo selectedItem = adapter.getChild(groupPosition, childPosition);
        genericFoodName = adapter.getGroup(groupPosition).getGenericFoodName();
        selectedFood = selectedItem;
        setTextConfirm(TextConfirmState.CONFIRM_ADD);
        return true;
    }
}
```

Determining generic food name in SearchActivity

```java
private String estimateGenericFoodNameFromSearchTerm() {
    String searchTerm = this.searchResultsModel.getSearchTerm();
    String selectedFood = this.selectedItem.getName();

    //Each part of the food name that contains the search term
    //should be added to final name
    //e.g.1. Search Term: Pizza
    // Result: Pepperoni and Cheese Pizza
    // Generic Name: Pizza
    //e.g.2. Search Term: Ch
    // Result: Cheddar Cheese Chips
    // Generic Name: Cheddar Cheese Chips
    //This is not an ideal approach!

    String sPattern = "(?i)\b\"+searchTerm\"(?::.\+?)\b\";  
    Pattern pattern = Pattern.compile(sPattern);  
    Matcher matcher = pattern.matcher(selectedFood);  
    String result = "";  
    while (matcher.find()) {  
        result += matcher.group() + " ";  
    }  
    return result;
}
```
public class LinkTask extends AsyncTask<String, Void, Boolean> {
    @Override
    protected Boolean doInBackground(String... params) {
        boolean success = false;
        if (params.length >= 2) {
            try {
                String imageName = params[0];
                //Encode whitespaces between words.
                String foodName = URLEncoder.encode(params[1], "UTF-8");

                Log.e(TAG, "Linking " + foodName + " with " + imageName);
                success = CAWebService.link(imageName, foodName);
            }
            catch (IOException e) {
                Log.e(TAG, e.getMessage());
            }
        } else {
            Log.e(TAG, "Insufficient Parameters.");
        }
        return success;
    }
}
IV.9 Encrypting Profile Data

```java
/** Encrypts given text with encryptionKey. Key must be 128-bits (16 characters long).
   * @param text text to be encrypted
   * @param encryptionKey 128-bit key
   * @return encrypted data
   * @throws CryptoException
   */

public static byte[] AES ( String text, String encryptionKey ) throws CryptoException {
    byte[] encrypted = null;
    try {
        Cipher cipher = Cipher.getInstance("AES");
        SecretKeySpec key = new SecretKeySpec(encryptionKey.getBytes("UTF-8"), "AES");
        cipher.init(Cipher.ENCRYPT_MODE, key);
        encrypted = cipher.doFinal(text.getBytes("UTF-8"));
    } catch (InvalidKeyException e) {
        throw new CryptoException(e);
    } //..more catch statements
    return encrypted;
}
```
Appendix V: Functionality Test Screenshots

V.1 Uploading Image

V.2 Identifying Image
V.3 Food Search

Log entry of Linking:

Link: 78 - Link Request. Linking Chocolate Malt Mix with /var/lib/openshift/51600645f8399320130730201132.jpg
Linker: 45 - Linker. ImageName: 35937204245399320130730201132.jpg, FoodId: 19, Link: 84 - Link Request. Success: true

V.4 Journal Entry

Tuesday, 30 July 2013

Snack Cracker 502.0 cal
Bananas (Large) 121.0 cal
Cocoa Cereal 110.0 cal
Chocolate Malt Mix 80.0 cal

Total Calories Consumed: 813.0 Cal
V.7 Re-identification

Possible Matches:
- Snack Cracker
- Chocolate Malt Mix
- Juice
- No match

V.8 Profile

Enter your age, height, weight and activity lifestyle to calculate your recommended calorie intake.

<table>
<thead>
<tr>
<th>Age</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sex</td>
<td>Male</td>
</tr>
<tr>
<td>Activity</td>
<td>Little or No Exercise</td>
</tr>
<tr>
<td>Weight Loss Goal (kg)</td>
<td>2</td>
</tr>
<tr>
<td>Height(cm)</td>
<td>177.0</td>
</tr>
<tr>
<td>Weight(kg)</td>
<td>66.0</td>
</tr>
</tbody>
</table>

Recommended: 1951.5 calories/day
Appendix VI: Image Recognition Test Screenshots

VI.1 Original Images

VI.2 Zoomed-in Images
VI.3 Zoomed-out Images

VI.4 Differently Orientation
VI.5 Different Lighting

Possible Matches:
- Chicken Burger
- French Fries
- Mango
- Barbeque Potato Chips
- Honey
- No match

Possible Matches:
- Chicken Burger
- French Fries
- Fanta
- Honey
- Bananas
- No match

Possible Matches:
- Chicken Burger
- Lime
- Orange Soda
- Juice
- No match

6 Different Meal

Possible Matches:
- Chicken Burger
- French Fries
- No match

Possible Matches:
- Chicken Burger
- Fanta
- French Fries
- No match

Possible Matches:
- French Fries
- No match
Bibliography


[10]: John Eakens, Margaret Graham (October 1999), Content Based Image Retrieval, University of Northumbria at Newcastle.


[12]: Jens-Rainer Ohm, Lezsek Cieplinski, Kim Heon Jun, Santhana Krishnamachari, B.S. Manjunath, Dean S Messiing, Akio Yamada MPEG-7 Color Descriptors
[13]: Balasubramani R, Dr V. Kannan, Efficient use of MPEG-7 Color Layout and Edge Histogram Descriptors in CBIR Systems, Global Journal of Computer Science and Technology


[16]: imgSeek Official website http://www.imgseek.net/isk-daemon (Retrieved on 7-Feb-2013)


[18]: Mathias Lux, Savvas A. Chatzichristofis (October 2008) LIRe: Lucene Image Retrieval - An extensible Java CBIR Library, Association for Computer Machinery


[20]: Doug Tidwell, James Snell, Pavel Kulchenko (December 2001) Programming Web Services with SOAP, O'Reilly Media Inc, CA, USA

[21]: Leonard Richardson, Sam Ruby (2007) RESTful Web Services, O'Reilly Media Inc., CA, USA

[22]: FatSecret, Platform API http://platform.fatsecret.com/api/ (retrieved on: 18-Feb-2013)


[29]: Bonnie Spring PhD, Jennifer M. Duncan PsyD, Amy Janke PhD, Andrea T. Kozak PhD, H. Gene McFadden, Andrew DeMott, Alex Pector, Leonard H. Epstein PhD, Juned Siddique PhD, Christine A Pelligrini PhD, Joanna Buscemi PhD, Donald Hedeker PhD (December 2012) Integrating Technology into Standard Weight Loss Treatment, American Medical Association


