Context-Aware Geo-tagging, Indexing and Querying of Smartphone Image Galleries

A thesis submitted in partial fulfilment of the requirements for
the award of the degree of

B.Tech.

in

Computer Science and Engineering

by

Arun Sai K (Roll No. 106112013)
Sujay Vetrivelan (Roll No. 106112093)

COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
TIRUCHIRAPALLI - 620015
MAY 2016

BONAFIDE CERTIFICATE

This is to certify that the project titled **Context-aware Geo-tagging, Indexing and Querying of Smartphone Image Galleries** the work done by:

Arun Sai K (Roll No. 106112013)
Sujay Vetrivelan (Roll No. 106112093)

in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering** at the **NATIONAL INSTITUTE OF TECHNOLOGY, TIRUCHIRAPPALLI,** during the years 2012-2016.

Dr. Mary Saira Bhanu
Project Guide

Dr. Leela Velusamy
Head of the Department

Project viva-voce held on _____

Internal Examiner

External Examiner

ABSTRACT

The last decade has seen a complete overhaul in the domain of image capture. From capturing images using film cameras in the early part of the century to the advent of digital photography, the sphere of image capture has undergone drastic developments. People no longer have to wait while their film rolls are developed at their local photo studio. Instead, they can now view thousands of digitally captured images at their leisure. The smartphone revolution has transformed public perception of the image capture process. With a smartphone, it is possible to take high resolution photos, comparable to those taken on a dedicated digital camera. The ubiquitous nature of smartphones means that individuals have perpetual access to a high quality camera wherever they travel. The proliferation of social networking sites such as Facebook [4], Instagram [2], and Twitter [6] has streamlined the process of sharing images with ones acquaintances and the general public. This has acted as added incentive to capture and share images that promote discussion and convey emotion. As a result of these factors, we have witnessed an unprecedented increase in the rate at which images are being captured. As pervasive computing comes to fruition, the number of images captured will only continue to grow. Hence, the next challenge in the digital image sphere comes in the form of efficient organization and classification of this multitude of images.

Through this entire revolution, the one aspect that has remained largely unchanged is the digital photo format. A large percentage of images being captured today are taken on devices with the potential to capture a significant amount of contextual information. This information, if recorded along with images, could provide rich functionality in areas such as image organization and querying for an image or a set of images on the basis of multiple parameters. An example is the location data captured by the GPS system of a smartphone. If this data was to be recorded along with its associated image, a culinary reviewer could easily retrieve all images taken at restaurants in a recent visit to Trichy. Adding contextual data other than the implicit data captured by the digital device allows for further customization specific to the use-case. The reviewer could retrieve images of the highest rated desserts containing chocolate, but not containing dairy, to create a catalog of the best outlets to buy desserts for lactose-intolerant individuals. At a more general level, such data recorded along with an image would allow for effective cataloging, easy documentation, and retrieval. Our proposed framework, named “ConWare Cam”, provides an

extensible structure to add contextual data to captured images through an intuitive and seamless interface. This data is encoded into the images on capture, and thus remains portable with the image and can be decoded on any platform to which the image may be transferred. Our application also provides rich functionality to query for both images as well as contextual data on the basis of multiple attribute values. ConWare Cam is a robust framework that comprehensively addresses the issue of adding associated contextual data to images.

ACKNOWLEDGEMENTS

The contribution of many different people, in their different ways, has made this project possible. We would like to extend our appreciation to the following :-

Dr. S. Mary Saira Bhanu, Associate Professor in the Department of Computer Science and Engineering, our project guide, for making this project possible with her infectious enthusiasm, constant support, invaluable guidance and pain-staking effort in proofreading the drafts. Indeed, without her guidance, we would not have been able to successfully complete this project.

Dr. R. Leela Velusamy, Head of the Department in the Department of Computer Science and Engineering, for the valuable insights she has shared from time to time and steadfast encouragement to complete this project.

Dr. Srinivasan Sundarrajan, Director, National Institute of Technology, Tiruchirappalli, for providing all facilities to carry out this project.

Finally, we are extremely thankful to all other faculty members, as well as our friends and family for their help and encouragement during the execution of this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The image capture process has taken many forms in the last decade, from capture using film cameras, to digital cameras, to the present day scenario using smartphone device cameras. With the rise of smartphones, there has been an unprecedented increase in the rate at which images are being captured. The smartphone image capture process is outlined below in Fig 1. The image sensor captures the object to be photographed, and this data is processed and refined by the image signal processor. As can be seen, there is no facility to capture or record contextual data in the existing scenario.

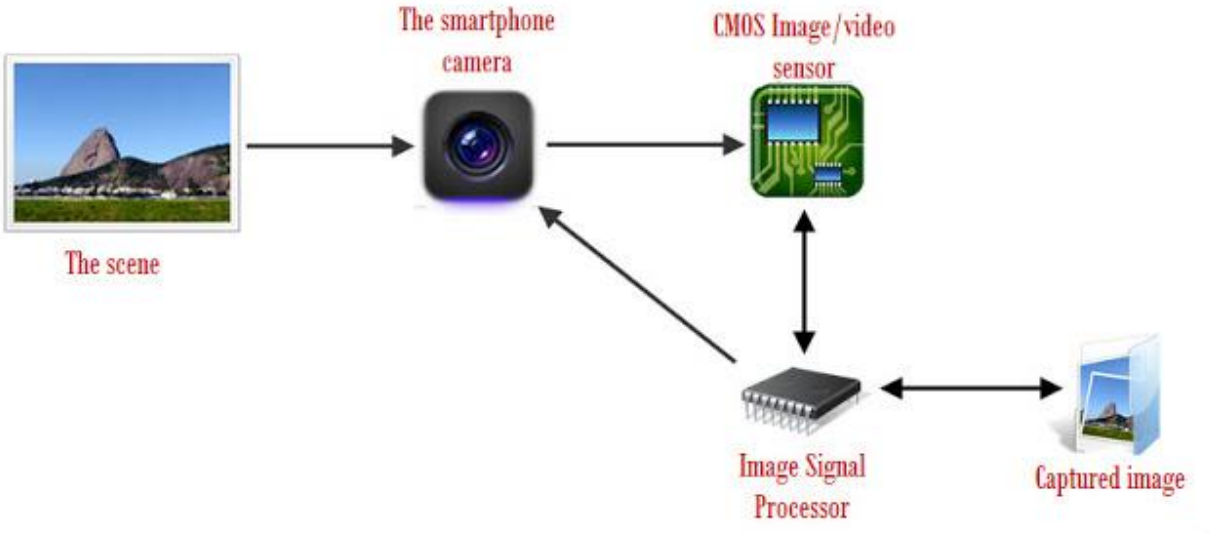

Fig 1. Image Capture in Smartphones.

In order to effectively query and access this abundance of images, we require an efficient method to organize and classify captured images. Our solution is to store the context information associated with an image along with the image data itself.

We broadly classify context-data into two categories: implicit and explicit context-data. We define implicit context-data as the data which can be captured implicitly by the smartphone while capturing an image. Contextual information such as the location, time of image capture, weather, camera orientation, type of camera (front or back) and proximity to objects (obtained from the

proximity sensor) qualifies as implicit context. Explicit context is defined as the context which can be added to the image by the user alone and not implicitly by the smartphone itself. For a wildlife photographer, the species, gender, lifespan, habitat and diet of the animal captured in the image may constitute the associated explicit context information.

Another issue to be considered is the manner of representation of this contextual data. We require a framework that is both flexible and that allows for economical storage. Templatizing the context attributes to be captured helps in building an extensible and robust system to capture and classify various classes of images and their respective context attributes. Furthermore, we have proposed a novel hierarchical template-based context attribute definition model that enables the users to define their own context templates to instantiate a new class of images.

Currently, there is no solution to adding context to images such that the contextual information is constituted in the image itself. Our solution uses a novel method to make this contextual information inherent to the image. We utilize the meta-data headers of the file to encode the contextual information into the image. We have also employed an alternate method to achieve the same by appending the Base64 encoded string version of the context-data to the Base64 encoded string version of the image itself.

Context data for all images is additionally stored in a relational database that is local to the device. This is done to improve performance and usability, as otherwise, the data would have to be decoded on every access. As a further optimization, we have indexed tables based on commonly searched attribute sets. Querying and retrieval is done using an intuitive GUI. We have also exposed API's for developer support.

CHAPTER 2

RELATED WORK AND EXISTING SOLUTIONS

Currently, there is no comprehensive solution to tackle the problem of adding context to images. However, there are a few products that address a small part of the issues pertaining to associating images with a context. Google Photos [1], Instagram [2] and Snapchat [3] are some of the products present in the market currently that address some aspects of the bigger problem of attaching contextual information to images. In this section, we discuss the current products in the market and their limitations.

2.1 PRODUCTS IN THE MARKET

Following are the existing solutions in the market:

2.1.1 Google Photos

Google Photos is an application that serves as a home for all the images captured by a user. All these photos are aggregated and a variety of services and features such as visual search, unlimited cloud backup storage, shared albums and instant image sharing are offered by the application. The visual search feature makes images searchable by places and things appearing in the images. This is a feature which associates locational context alone to images. The other contextual data captured in this application is user information. Other features such as shared albums and instant image sharing serve as a platform for collaborative image based social networks.

2.1.2 Instagram

Instagram is an online mobile photo-sharing, video-sharing, and social networking service that enables its users to take pictures and videos, and share them either publicly or privately on the app, as well as through a variety of other social networking platforms. Users can upload photographs and short videos, follow other users' feeds and geotag images with longitude and latitude coordinates, or the name of a location. Location and information about the person sharing the image are the contextual data captured by this service. Similar to Google Photos, Instagram is primarily a social networking platform for the social exchange of images.

2.1.3 Snapchat

Snapchat is an ephemeral image messaging application software product. It is primarily used for creating multimedia messages referred to as "snaps"; snaps can consist of a photo or short video, and can be edited to include filters and effects, text captions, and drawings. A feature known as "Geofilters" allows special graphical overlays to be available if the user is within a certain geographical location, such as a city, event, or destination. The "Selfie lens" feature allows users to add real-time effects using face detection into their snaps. Users can also send text messages to their friends along with the images and videos. Snapchat is different from the previous two products due to its ephemeral nature which results in fleeting images. The only contextual information stored in this product is again location and user information.

2.2 LIMITATIONS OF EXISTING PRODUCTS

- From the descriptions of the existing solutions in the market, we clearly see that the contextual information added to the images is very limited and primarily includes only locational and user context. There is a significant amount of other contextual information surrounding every image a user captures using a smartphone which can go a long way in helping develop a robust system to organize, sort and query these images effectively. Weather conditions, proximity to user, type of camera used (front or back) and orientation of the smartphone are just some examples of the additional context data that can be captured to help in effective organization and retrieval of the images captured.
- The contextual information which is captured is not constituent in the image itself. Hence, when the image is shared with someone or transferred to another system, the recipient does not gain any contextual knowledge from the image. Thus, it is critical that the captured contextual image is made constituent in the image to ensure portability and seamless access of context-related information from the images.
- There is no provision for the user to add custom context data to the images in these systems. A flexible and scalable system to enable the user to define and add custom context attributes to images is a key feature which is not present in any of the current solutions in the market.

CHAPTER 3

CONWARE CAM: OUR PROPOSED FRAMEWORK

We propose to address the issue of associating context to images through our novel framework named ConWare Cam. Our proposed solution is radically different from existing solutions as we have reworked the way images are captured in the smartphone in order to enable seamless and intuitive infusion of context data in the captured images.

In traditional camera applications, the image is captured from the digital camera mounted on the smartphone and directly stored in an image store as shown in Fig 2. These applications directly store the captured images in the smartphone's gallery and no context related information is captured in this process.

In our novel framework, we have an additional context-wrapper layer around the digital camera which gathers and embeds the required context information into the image being captured. The image is stored in an image store and the context data is also stored in a context-store after it has been embedded in the image. The query module is used to query the captured images based on contextual data and it retrieves the queried images after interacting with the context and image stores. This system is illustrated in Fig 3.

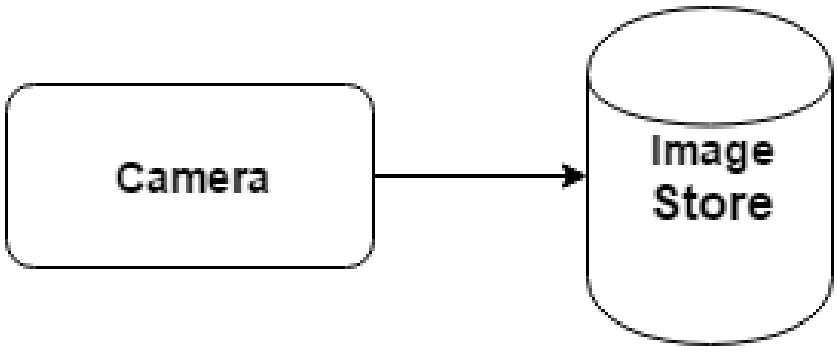

Fig 2. Traditional Smartphone Camera Applications

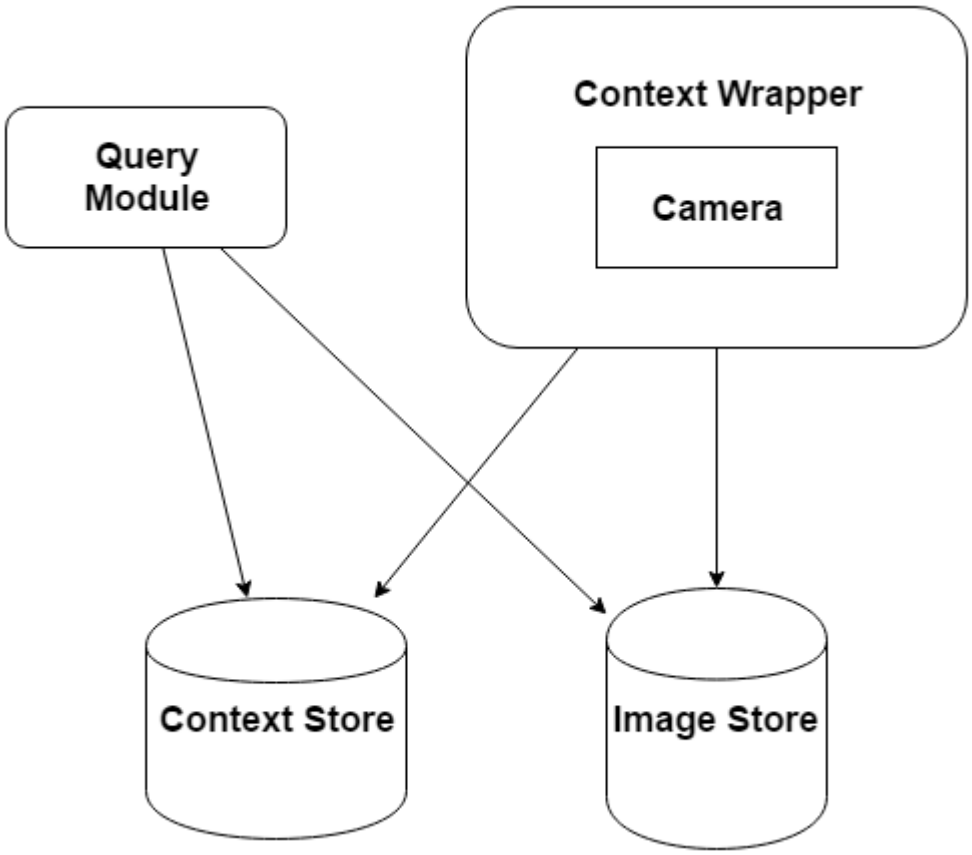

Fig 3. ConWare Cam: Our Novel Approach

Further, we can visualize our context wrapper as a layered structure as shown in Fig 4. The captured images go through all these layers in the context-wrapper before it is ready to be stored in the image store. Following are the five layers in the context wrapper which wraps around the smartphone camera module:

- **Layer 1 - Hierarchical Template-Based Context Model Definition:**
  This layer enables the user to define custom context-model templates which give the user the flexibility to store varied types of contextual information based on the type of image being captured.

- **Layer 2 - Dynamic Table Instantiation:**
  This layer is responsible for dynamically creating new relational tables in the context store as and when new context-attribute templates are defined by a user to capture the context-data associated with different classes of images.

- **Layer 3 - Context-Data Capture:**

  This layer is where the real time context data is captured and prepared to be added to the image in the subsequent layers.

- **Layer 4 - Encode Context Data to Image:**

  This is the layer where the context data captured in the previous step is encoded into the image itself to ensure seamless transfer and portability of the context data associated with the image.

- **Layer 5 - Database Interface:**

  This layer is responsible for the interaction with the image and context stores. It acts as an interface to the databases.

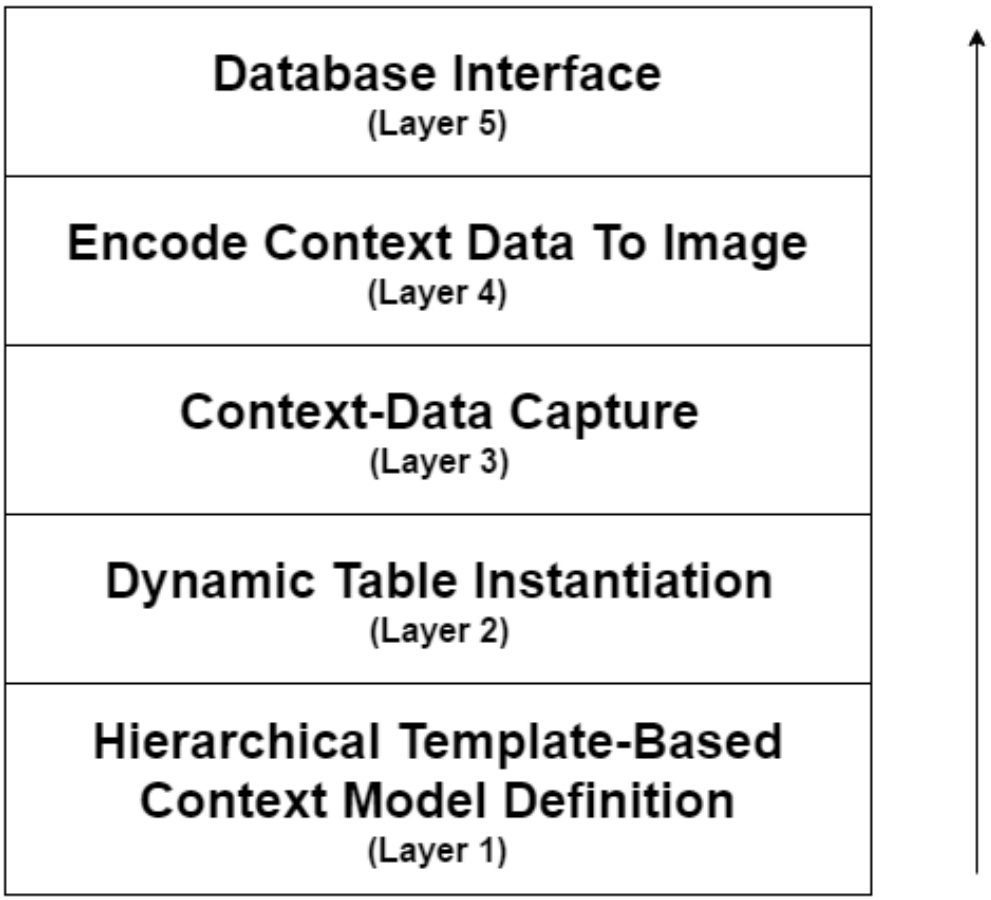

Fig 4. Context Wrapper Layers

The detailed architectural design of our proposed framework is illustrated in Fig 5. Our design consists of four high level modules, each performing a unique and crucial role in our framework. The modules are listed as follows:

- **Hierarchical Template-Based Model for Context Attribute Definition:**
  This module is responsible for defining templates for context attributes of various classes of images which may be captured by the user.

- **Encoding Context-Data to Images:**
  The context data captured in the application is embedded in the image through this module. This enables seamless transfer and sharing of the contextual information along with the image.

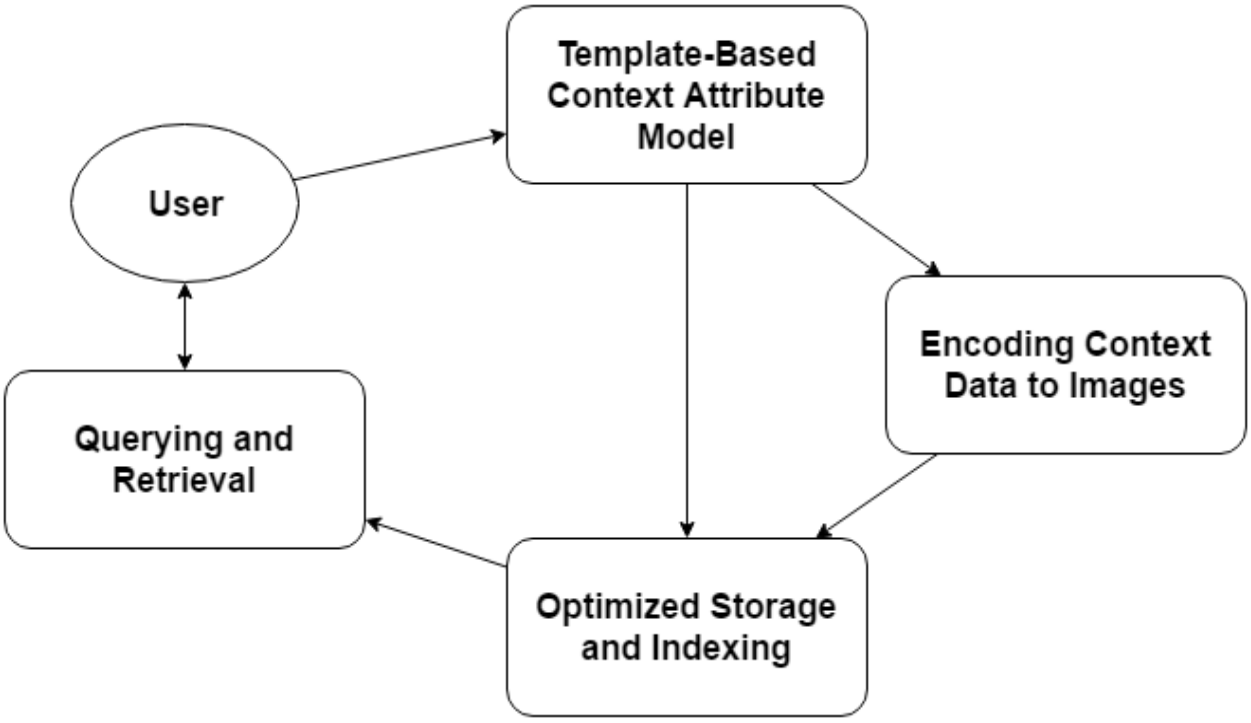

Fig 5. Architecture Diagram.

- **Optimized Storage and Indexing of Context-Data:**
  This module is responsible for the effective storage of images and the associated context-data. It operates on the image and data stores. Efficient indices are maintained on various attributes in order to ensure high speed querying and retrieval of context-data and images.

- **Querying and Retrieval of Images and Context-Data:**
  Intuitive interfaces are provided for querying and viewing the retrieved images or associated context-data. The queries can return the images or the associated context-data. This flexibility allows users and developers to work with both images and context-data retrieved by querying the image and context stores.

In this chapter, we have given a broad overview of the structure and functioning of our novel framework. We describe the structure and function of the components of our framework in detail in the subsequent chapters.

CHAPTER 4

HIERARCHICAL TEMPLATE-BASED CONTEXT-ATTRIBUTE MODEL

The flexibility to define new templates and context-attribute models is a crucial feature while supporting context-data capture for multitude of image classes. This is enabled in our framework through a hierarchical template-based context-attribute model, which is a highly flexible and extensible module. It allows the user to dynamically create new context templates on the go. It also enables the user to extend existing templates, much like inheritance in object oriented programming. An extensible hierarchical structure is developed with super-template and sub-template relationships. The parent template is called the super-template, while the child template is called the sub-template. The sub-template inherits the features of the super-template while also having additional features of its own. In this chapter, we discuss the functioning and structure of this module in detail.

The context information that is to be stored has to be in a format that is:

- **Lightweight:**
  It should be lightweight as the limited processing power of the smartphone should be preserved for more user-intensive operations.

- **Portable:**
  It should be portable in order to ensure that this information can be transferred, decoded and read seamlessly across several platforms.

- **Extensible:**
  It should be extensible to ensure that additional contextual information can be added to the image at any point of time without majorly disrupting the storage structure of the contextual information.

Based on these requirements, we choose a simple and extensible data exchange format such JavaScript Object Notation (JSON) or Extensible Markup Language (XML) for storage and exchange of the contextual information associated with the images being captured.

Inheritance is enabled and hierarchy is preserved in this module through a tree-like structure for context-template definition. The broad outline of this tree-like structure is illustrated in Fig 6. The root template is the “Image” template. All the implicit context data such as latitude, longitude, proximity to user, time and weather is captured in this template. Other templates like “Animals” and “Plants” extend the “Image” template. Thus, “Animals” and “Plants” inherit the context-attributes from “Image” while also having context-attributes of their own which are defined by the user. A more detailed hierarchical tree structure will be illustrated and discussed later in this chapter.

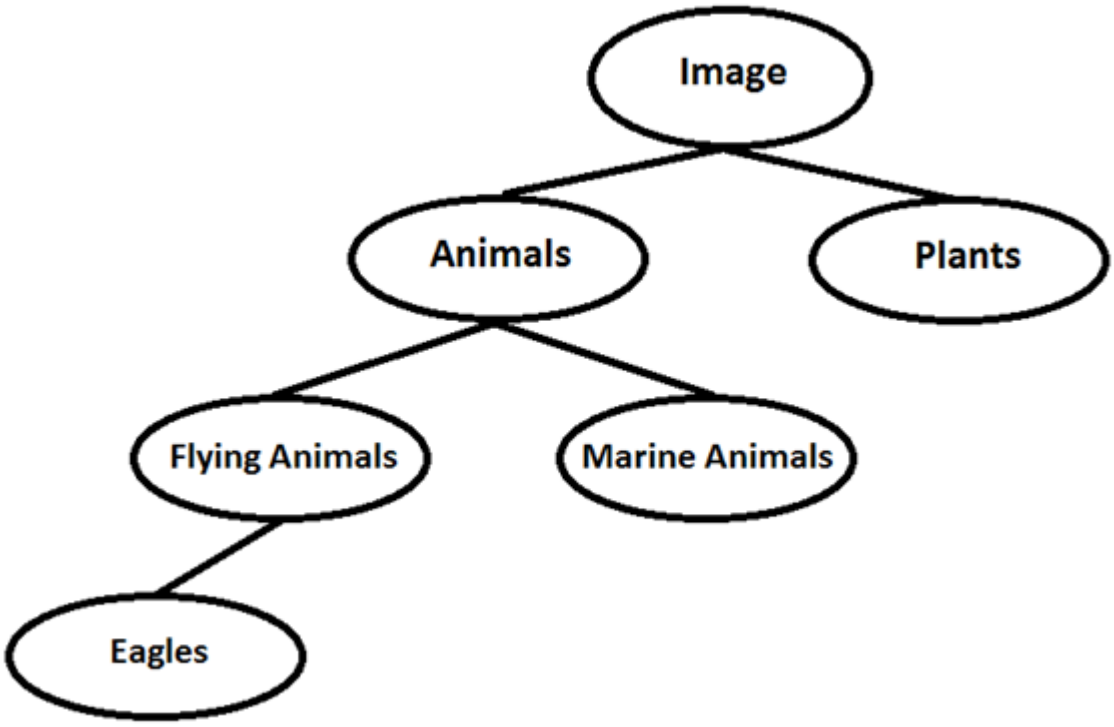

Fig 6. Overview of Hierarchical Tree-like Structure.

A sample structure of how to store contextual information is shown in Fig 7. In this illustration, we use JavaScript Object Notation (JSON) to show how to store the data in a simple and extensible data exchange format. The illustration provides the sample structure for storing contextual information of the “Animals” template which has been shown above. The “Animals” template extends “Image” and hence inherits all the attributes from the “Image” template.

Additionally, it also has templates of its own, namely: “species” which accepts only string values and “expected_lifespan” which accepts only integer values.

```
{
        “template_name” : “Animals”,
        “extends” : “Image”,
        “attribute_list” :
                [
                        {
                                “attr_name” : “species”, “attr_type” : “string”
                        },
                        {
                                “attr_name” : “expected_lifespan”, “attr_type” : “int”
                        },
                ]
}
```

Fig 7. Sample JSON Storage Structure for Contextual Data.

The contextual information can be stored in smartphones in the following two ways of persistent data storage:

- Relational Database.
- SharedPreferences.

The overhead associated with handling transactions in a relational database negates the advantages it provides by way of reliability and structure. The contextual data must be quickly retrieved on-demand. SharedPreferences is a persistent storage solution in smartphones (Android) which utilizes a key-value format to store data, thereby facilitating quick read-write operations. This makes it the favorable context-data storage option considering our constraints.

A detailed example of the hierarchical tree like structure along with the attributes involved is illustrated in Fig 8.

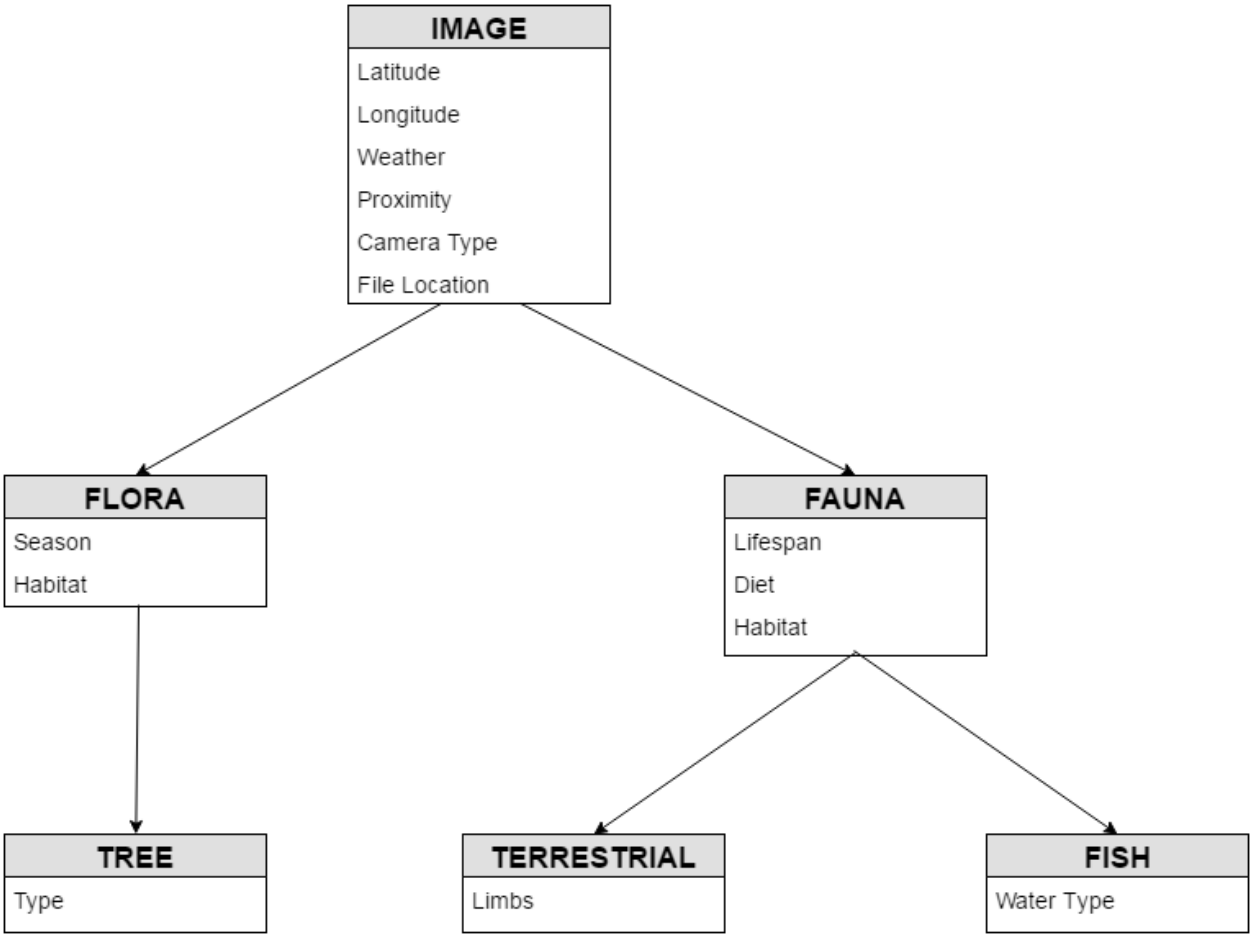

Fig 8. Detailed Hierarchical Tree-like Structure.

In the above example, the "Image" template is extended by the "Flora" and "Fauna" templates. Flora can be a "Tree" which has a type associated with it such as coniferous, deciduous or evergreen. Fauna can be categorized as "Terrestrial" and "Fish". Terrestrial fauna has an attribute named limbs, which gives the number of limbs of the animal. Fish has an attribute named water type which gives the type of water in which the fish lives: fresh water or marine.

Thus, we conclude the discussion of the hierarchical template-based context-attribute model in this chapter.

CHAPTER 5

INCORPORATING CONTEXT-DATA IN IMAGES

Encoding and decoding context-data to and from images is a crucial component of our proposed framework. We encode the contextual data into the image to enable portability and interoperability across platforms and devices. Making the context information constituent in the image enables seamless access and utilization of the contextual information associated with the images. We decode this information from images when we need to extract the contextual information.

Encoding context-data into images can be done in two ways:

- Appending Base64 string of context data to Base64 string version of the image.
- Storage in meta-data headers (EXIF headers).

The above two methods are discussed in detail in the following sections.

5.1 METHOD A - APPENDING BASE64 ENCODED STRINGS

The method of appending Base64 encoded strings to encode context data into images is as shown in Fig 9.

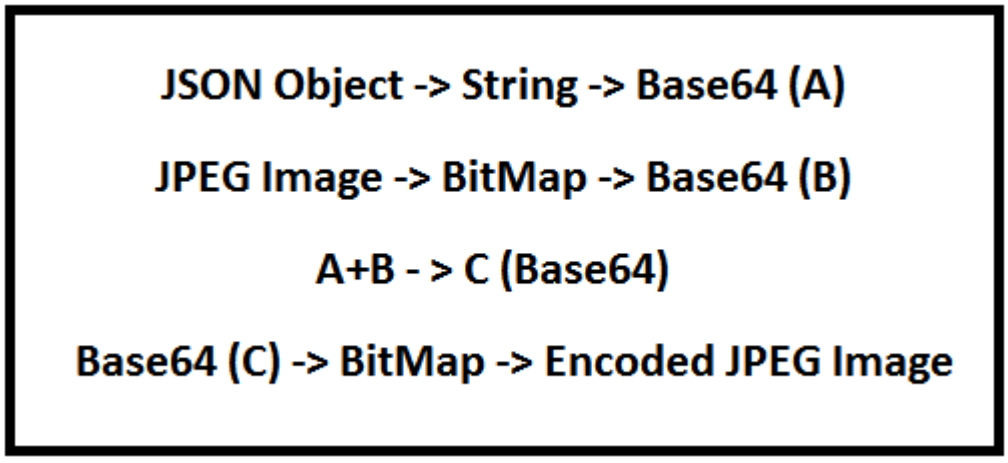

Fig 9. Method A – Appending Base64 Encoded Strings.

The JSON object containing the context information is converted to a string. This string is then converted to a Base64 encoded string (A). The JPEG file of the image is then converted to a Bitmap which is in-turn converted to a Base64 encoded string (B). These two Base64 strings are appended to get a third Base64 encoded string (C), which is then converted to a Bitmap and then to a JPEG file. Thus, this JPEG file consists of the encoded context-data as well as the original image itself.

An illustration of the working of this method is shown below in Fig 10 and Fig 11. Fig 10 shows an image and its equivalent Base64 encoded string. Fig 11 shows the JSON object of the contextual data associated with the image and its corresponding Base64 encoded string. These two Base64 encoded strings are then appended and then converted to a JPEG file which contains both the original image and the context-data associated with it.

Fig 10. JPEG Image to Base64 Encoded String.

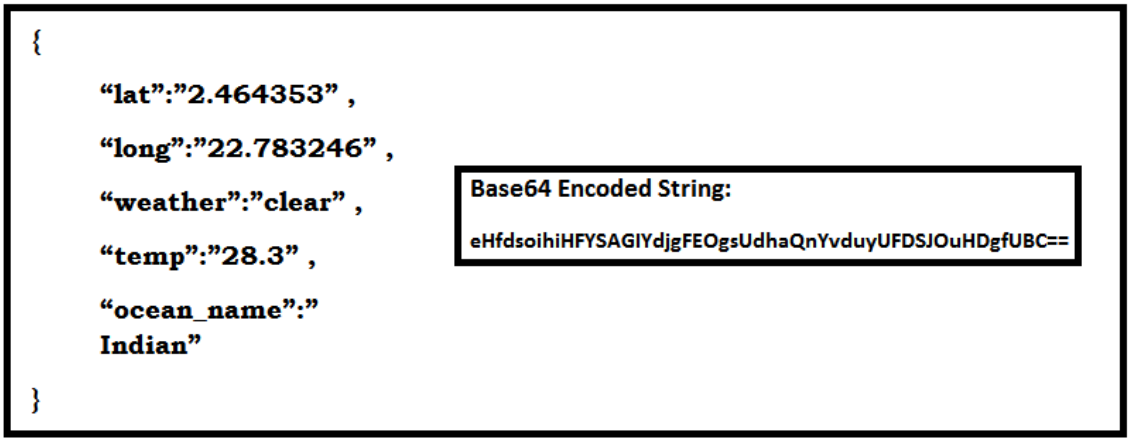

Fig 11. JSON Context-data to Base64 Encoded String.

5.2 METHOD B – STORAGE IN META-DATA HEADERS (EXIF HEADERS)

Exchangeable image file format (Exif) [7] is a standard that specifies the formats for images, sound, and ancillary tags used by digital cameras (including smartphones), scanners and other systems handling image and sound files recorded by digital cameras. The specification uses the following existing file formats with the addition of specific metadata tags: JPEG discrete cosine transform (DCT) for compressed image files and TIFF Rev. 6.0 (RGB or YCbCr) for uncompressed image files. When Exif is employed for JPEG files, the Exif data are stored in one of JPEG's defined utility Application Segments.

The metadata tags defined in the Exif standard cover a broad spectrum:

- Date and time information. Digital cameras will record the current date and time and save this in the metadata.
- Camera settings. This includes static information such as the camera model and make, and information that varies with each image such as orientation (rotation), aperture, shutter speed, focal length, metering mode, and ISO speed information.
- A thumbnail for previewing the picture on the camera's LCD screen, in file managers, or in photo manipulation software.
- Descriptions.
- Copyright information.

For our purposes, we use the reserved tags in the Exif format such as “UserComment” to store our context-data information string (obtained from the corresponding JSON object) in with the JPEG file of the captured image. When the image file is transferred or shared, the new device can simply extract the context-data associated with the image by accessing the “UserComment” Exif meta-data tag. Thus, this method enables seamless mobility of the images and associated context-data, while also enabling interoperability across different smartphone platforms such as Android, iOS, Symbian and Windows.

CHAPTER 6

OPTIMIZED STORAGE AND INDEXING

6.1 NEED FOR IMAGE AND CONTEXT STORES

Once the image has been captured and the context data has been recorded and encoded in the image, the next step is to store the images in the memory or disk of the device. For this model to qualify as a feasible solution, the retrieval of the captured image and its associated context data must be instantaneous. As covered in chapter 5, the framework employs two broad strategies to encoding context data within the image. One strategy is to use the meta-data headers of the file to encode the contextual information into the image. An alternative strategy appends the Base64 encoded string version of the context-data to the Base64 encoded string version of the image itself. Both of these methods are efficient approaches when dealing with a single image. Consider however, the situation when a user queries for all images on the basis of an attribute value. Executing this operation translates to decoding the context data (using the methods detailed in chapter 5) and checking for attribute equality for potentially thousands of images. Although today's devices possess multiple processing units that allow for parallel processing, executing several such queries would represent a significant burden to the mobile device's limited resources. This manifests itself as poor user experience, especially considering that users may use multiple parameters to query for thousands of images. Hence, an alternative approach to storage and querying must be employed to guarantee the feasibility of this solution.

An approach that would eliminate the need to decode context data on every access of the image would require an alternative mapping of images to their respective context data. Storing this mapping on device storage would represent an additional cost on device storage. However, there are numerous factors to suggest that this overhead is not as big a hurdle as may initially appear. The mapping for a single image represents a storage overhead of only a few bytes. Exponential growth in storage technology has resulted in mobile devices having storage capacities up to hundreds of gigabytes. Considering these factors, the cost of storing this mapping separate from images on the mobile device is minimal.

Querying for images on the basis of this mapping gives the mapping a large significance. The correctness of the query must not be compromised in pursuit of additional efficiency. The mapping must always be consistent with the context data encoded in images. Furthermore, this mapping must be durable and survive system failures. It must be ensured that the mapping guarantees these minimum requirements to qualify as a potential solution.

6.2 CONTEXT STORE: STRUCTURE AND FUNCTION

The mapping itself can be realized using a number of data structures. However, considering the requirements touched upon previously, the structure that lends itself most naturally to the situation is the database management system. A DBMS that guarantees the ACID (Atomicity, Consistency, Isolation and Durability) properties is an ideal candidate to store this mapping. Our framework uses a relational DBMS to store image context data and map it to its respective image.

One possibility is to store all mappings in a single relation. This approach would satisfy the minimum requirements specified. However, in practice, executing a query on this relation would mean iterating through and performing equality comparisons with thousands of tuples. While this is significantly less time consuming than decoding the context data for thousands of images, a better design can be proposed that does not require nearly as many operations.

An alternative to the above is to dynamically instantiate a new relation for each template (described in chapter 4). Adopting this model has numerous advantages. Often times, the template is a parameter included when querying for images. This reduces the tuples to be searched to just those in the specified relation, a substantial decrease from the previous approach. Consider a template “Eagle”, which has attributes “ImageID”, “SubFamily”, “Diet”, and “NativeLocation”. When this template is created, a new relation is dynamically instantiated to store the mapping of all images of template “Eagle” with their respective context data. The relation for the “Eagle” template is shown in Table 1.

Consider an example query executed by the user, which is intended to retrieve all images of template “Eagle” belonging to SubFamily “Haliaeetinae”. The DBMS recognizes the presence of template name “Eagle” as a parameter, and hence restricts its search to only those tuples in the “Eagle” relation. Now, the DBMS iterates through these tuples, and returns the ImageID

corresponding to each tuple with SubFamily value “Haliaeetinae”. The device then retrieves the images corresponding to the returned ImageIDs, and these images are presented to the user as the result of the query.

ImageID	SubFamily	Diet	NativeLocation
0xADIF	Haliaeetinae	Fish	Northern Africa
0xAC35	Aquilinae	Marsupials	South Asia

Table 1. Sample Relational Table.

Further optimizations can be made to the current model. By indexing the relations on certain attributes, the query time can be substantially reduced. Indexing however, adds overhead in terms of maintenance of the index. Every insertion, deletion and some updations to the relation would require a corresponding change to the index. When considering multiple indices for each relation, this overhead cannot be disregarded. Hence, creation of indices must only be done when the index is guaranteed to be used often enough to compensate the associated maintenance overhead. To this effect, our solution indexes relations geospatially [10][11][12][13], temporally, and on the basis of file location, a set of attributes that we believe will be commonly used in queries.

CHAPTER 7

QUERYING AND RETRIEVAL

Once the images are well organized according to their context, the only task that remains is the querying and retrieval of these images. Users can query for images through a simple and intuitive graphical user interface. This querying system will also support complex queries with multiple attributes to ensure easy access of images according to very specific contexts.

Some researchers and developers would be more interested in the context data than the image itself. To support this, we have also opened up an Application Programming Interface (API) to access the context data of the queried images in a suitable data exchange format.

The results of the query can be retrieved in the following forms:

- **Images:**
  Images which satisfy the query condition.
- **Context-data:**
  Context-data associated with the images satisfying the query condition.

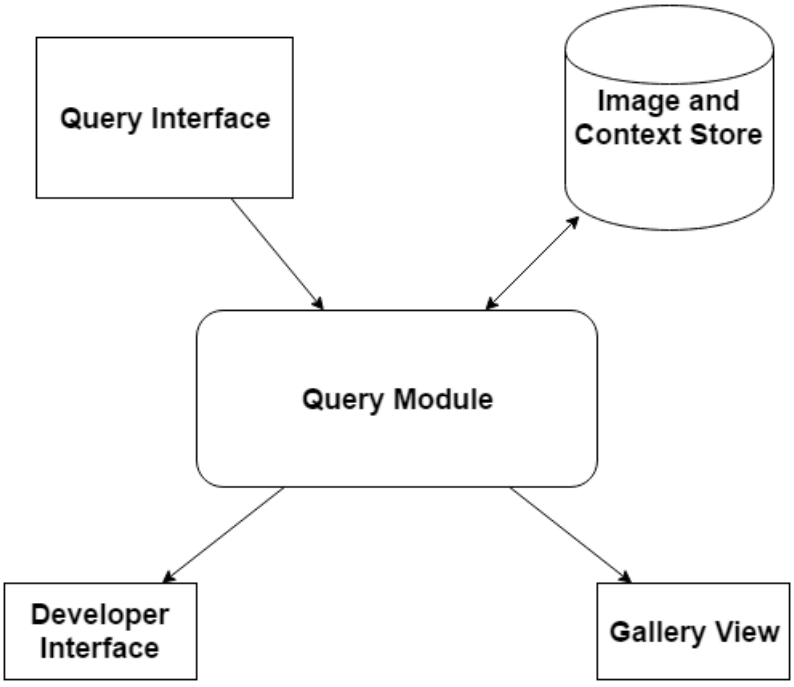

Fig 12. Block Diagram – Querying and Retrieval.

The structure and functioning of the querying and retrieval component in our novel framework is illustrated in Fig 12.

The various components of the querying and retrieval component of our framework are described below:

- **Query Interface:**
  This is an intuitive graphical user interface where the user queries for images based on complex query conditions. This component helps the user easily specify complex query conditions to search for a very specific image / set of images.
- **Image and Context Store:**
  The images and the context-data associated with them are stored in the image and context store of our framework.
- **Gallery View:**
  The images which satisfy the conditions specified by the query are displayed in a gallery view, which is a suitable interface to display a set of images.
- **Developer Interface:**
  The context-data associated with the images which satisfy the condition specified by the query are exposed to researchers and developers through an Application Programming Interface (API) supported by the developer interface.

CHAPTER 8

IMPLEMENTATION AND RESULTS

8.1 IMPLEMENTATION DETAILS

For the purposes of prototyping and proof of concept, we have implemented our solution as an application for the Android operating system. Android, with more than 60% of the mobile operating system market share, is currently being deployed on approximately 1.5 billion devices by users around the world. By building our prototype for the Android OS, we have shown that our solution can be adopted at scale.

The application was built using the Android SDK (Software Development Kit), using the Android Studio IDE (Integrated Development Environment) [8] [9] developed by Google and based on JetBrains' IntelliJ IDEA (IDE Advanced) software. The Java programming language was used to implement the functionality of the application. Layout design and implementation was done using the EXtensible Markup Language (XML).

Template structure, including name and various attributes, is stored using the JavaScript Object Notation (JSON) . JSON is a lightweight interchange format, and its use allows for speed of processing and high readability. JSON is also used in our application as notation for the context data encoded along with their respective images. Our developer interface returns context data in the JSON format.

Encoding the context data within images is done through two methods (described in chapter 5), each of which employs different technologies. In the first approach, EXIF (Exchangeable Image File Format) metadata tags are used to store the image context data. The alternative approach uses the Base64 encoded string representation of the image as well as the context data.

The template model is persistently stored by representing the model in JSON notation and storing it in Android SharedPreferences. SharedPreferences is an API (Application Programming Interface) from the Android SDK to store and retrieve sets of data values persistently. It is well suited to storing key-value sets such as the template model.

Mappings from context data to their respective images are stored persistently using the SQLite RDBMS. SQLite is ACID-compliant and provides transaction processing facilities.

The source code of the prototype has been open sourced using GitHub. As this research area remains in its nascent stage, open sourcing the project provides an opportunity to collaboratively address this vast problem and the issues associated with it.

8.2 RESULTS

Our framework has been successfully implemented in the form of an application on the Android smartphone platform. The steps to capturing and adding context data to an image using our application have been outlined in Fig 13.

Images are captured using the application's camera. Both the front and back cameras can be used, and images can additionally be taken in a panoramic format. Features including exposure control and auto/manual focus have been incorporated.

Once an image has been captured, the user must either assign it an existing template or create a new template for it. If the user chooses to create a new template, the new template must be assigned a name and a variable number of attributes. Attributes are defined by their name and data type.

Once the template is assigned to the captured image, the user must enter values for the attributes pertaining to that template. This includes those attributes that the template inherits from its parent templates. Once this is done, the context information is encoded in the image and the image is stored in memory.

Querying for images can be done on the basis of a single attribute or multiple attributes. The returned results can be either in the form of the images themselves or their associated context data in the JSON format.

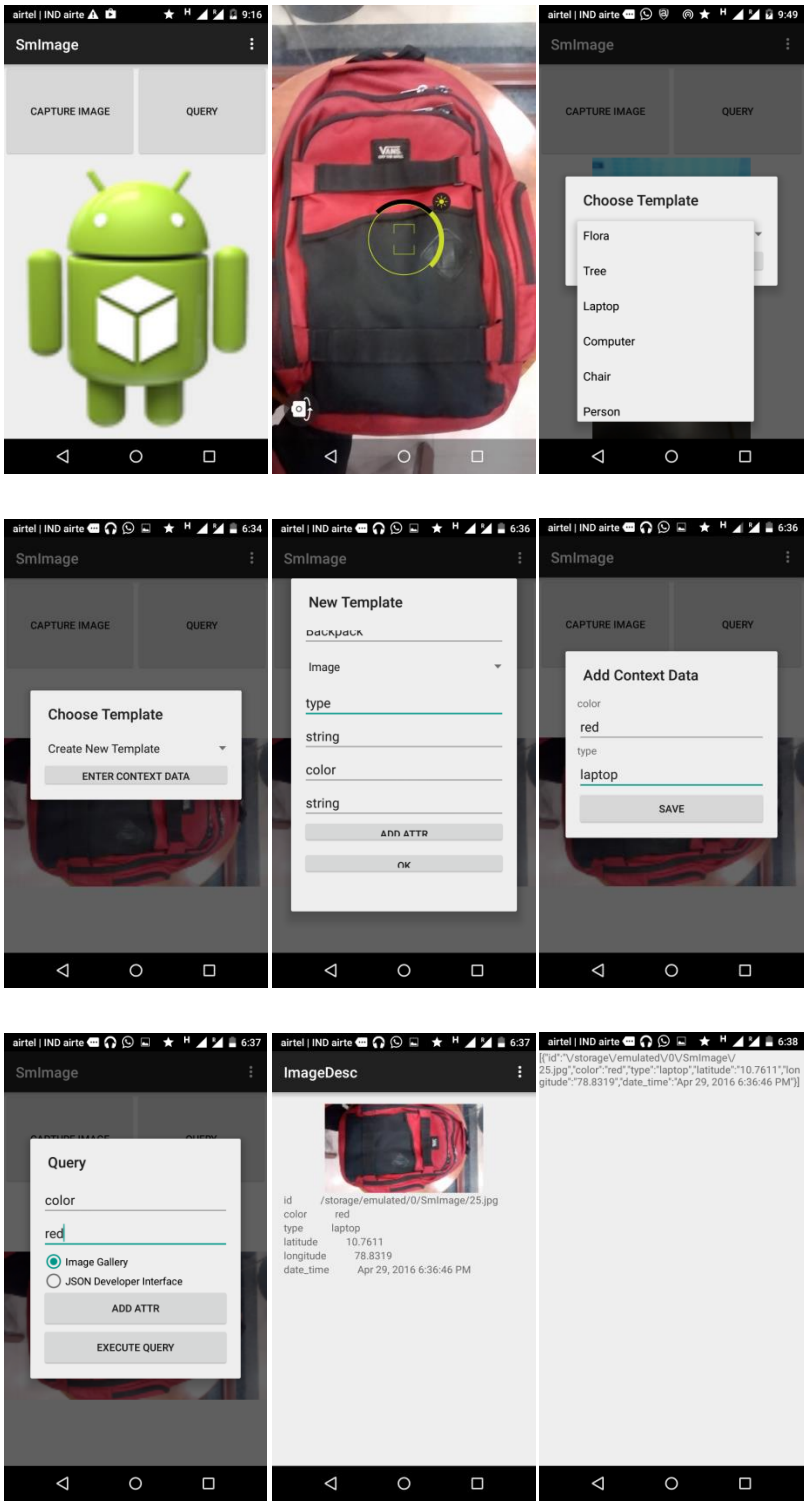

Fig 13. Application Walkthrough.

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 CONCLUDING REMARKS

With the onset of ubiquitous computing, the rate at which images are being captured will continue to grow. The need of the hour is a framework which helps organize and catalog this multitude of images. Our solution has successfully addressed this issue at a broad level, by providing extensive functionality with a seamless user experience. By building a prototype of our concept for the Android OS, we have made the application available to the 1.5 billion devices currently running the OS and thereby have shown that our solution can be adopted at scale.

The template based context attribute model provides a hierarchical, extensible structure to organize context data. Inheritance of attributes from parent templates allows for economy of storage and representation, two important factors when considering that users capture thousands of images with their devices. Context data is encoded in the image itself, using two broad approaches – one employing EXIF metadata tags, and the other using the Base64 encoded string representation of the image and context data. When executing a query, encoding and decoding the context data for every image is a time consuming operation that manifests itself as poor user experience. To tackle this, we store the mapping of images with their respective context data separately from the image. This mapping is stored persistently on device storage using the SQLite RDBMS. Performance is further enhanced by storing each template as a separate relation, and indexing the relations on commonly queried attributes. The querying itself can be done on the basis of multiple attributes. The images satisfying the query can be returned as is, or the context data associated with the images can be returned in the JSON notation. Thus, the solution effectively addresses the broad issues associated with the problem, by providing the functionality required in an environment that can be replicated at a mass scale.

9.2 FUTURE WORK

By open sourcing our application, this incipient area can receive the mass attention that it requires. Our solution has tackled the broad issues associated with the problem, and has shown that it can be feasibly addressed. There are many improvements and optimizations, which, if

made to our base application, could enhance the usability and encourage the commercialization of this product. These are detailed below.

Currently, implicit contextual information is captured automatically by the device and encoded with the image. All other data that is not implicitly captured by the devices sensors has to be entered manually by the user. This represents a time consuming operation that has to be performed meticulously. By deploying image processing techniques as an additional layer on our application, much of this context data entry can be automated by the device. As an additional optimization, machine learning methods can be used to suggest attributes for template definition, and to automate similar tasks.

Our solution has addressed the issue of providing a framework for encoding contextual data in images. Multimedia formats are currently undergoing a revolution that has seen the introduction of virtual reality technologies to a mainstream audience. In light of these advancements, our framework must be extended to other media formats, to remain relevant in the current environment. The recent proliferation of applications such as Vine[5], Instagram, and Snapchat, coupled with increasing network bandwidths and storage capabilities, has seen an explosion in the capture and sharing of video content. Our framework could be extended to include contextual data with both video and audio formats.

REFERENCES

1. (2016) Google Photos [Online]. Available: https://photos.google.com
2. (2016) Instagram [Online]. Available: https://www.instagram.com/
3. (2016) Snapchat [Online]. Available: https://www.snapchat.com/
4. (2016) Facebook [Online]. Available: https://www.facebook.com/
5. (2016) Vine [Online]. Available: https://vine.co/
6. (2016) Twitter [Online]. Available: https://twitter.com/
7. (2016) Exif – Wikipedia [Online]. Available: https://en.wikipedia.org/wiki/Exchangeable_image_file_format
8. (2016) Android Documentation [Online]. Available: developer.android.com/
9. (2016) Android Studio [Online]. Available: developer.android.com/sdk/index.html
10. Debopam Acharya and Vijay Kumar. 2005. Location based indexing scheme for DAYS. In*Proceedings of the 4th ACM international workshop on Data engineering for wireless and mobile access* (MobiDE '05). ACM, New York, NY, USA, 17-24. DOI=http://dx.doi.org/10.1145/1065870.1065874
11. W. S. Ku, R. Zimmermann and H. Wang, "Location-Based Spatial Query Processing with Data Sharing in Wireless Broadcast Environments," in *IEEE Transactions on Mobile Computing*, vol. 7, no. 6, pp. 778-791, June 2008. doi: 10.1109/TMC.2007.70791
12. S. Geetha and S. Anu Velavan, “Optimization of Location Based Queries Using Spatial Indexing,” in *ICTACT Journals*
13. Chuan Qin, Xuan Bao, Romit Roy Choudhury, and Srihari Nelakuditi. 2011. TagSense: a smartphone-based approach to automatic image tagging. In Proceedings of the 9th international conference on Mobile systems, applications, and services (MobiSys '11). ACM, New York, NY, USA, 1-14. DOI=http://dx.doi.org/10.1145/1999995.1999997

APPENDIX - SOURCE CODE

MainActivity.java

```
package com.conware.smimage;

import android.app.Dialog;
import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.location.LocationManager;
import android.os.Bundle;
import android.support.v7.app.ActionBarActivity;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.Spinner;
import android.widget.TextView;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.text.DateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;


public class MainActivity extends ActionBarActivity{

    private static final int CAMERA_REQUEST = 1888;
    Button captureImage,queryButton;
    ImageView capturedImage;
    TemplateFormat tf;
    String fname;
    String i;

    List<String> templates;
    List<String> copyTemplates;
    int selectedPosition;
    int selectedExtendsPosition;
```

```
    JSONObject globalContextJSON;
    DatabaseHandler db;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        boolean containsIsInit = Prefs.containsBool(MainActivity.this, "isInitialised");
        if(containsIsInit == false)
        {
            Prefs.setMyBoolPref(MainActivity.this, "isInitialised", false);
        }

        db = new DatabaseHandler(MainActivity.this);
        tf = new TemplateFormat(MainActivity.this);
        captureImage = (Button) this.findViewById(R.id.captureImage);
        queryButton = (Button) this.findViewById(R.id.query);
        capturedImage = (ImageView) this.findViewById(R.id.capturedImage);

        captureImage.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                captureImageFromCamera();
            }
        });
        queryButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                queryDialog();
            }
        });

    }

    public void captureImageFromCamera() {
        Intent cameraIntent = new
Intent(android.provider.MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(cameraIntent, CAMERA_REQUEST);
    }

    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == CAMERA_REQUEST && resultCode == RESULT_OK) {
            Bitmap mphoto = (Bitmap) data.getExtras().get("data");
            i = Base64Utilities.bitmapToBase64(mphoto);
            capturedImage.setImageBitmap(mphoto);
            fname = ImageUtilities.saveImage(mphoto, getApplicationContext());
            ImageUtilities.galleryAddPic(fname, MainActivity.this);
            createEnterContextDialog();
        }
    }

    public ArrayList<String> captureImplicitContext(){
```

```
        double latitude,longitude;
        LocationManager lm = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
        Location location = lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        double longitude = location.getLongitude();
        double latitude = location.getLatitude();

        String currentDateTimeString = DateFormat.getDateTimeInstance().format(new Date());

        ArrayList<String> res = new ArrayList<String>();
        res.add(String.valueOf(latitude));
        res.add(String.valueOf(longitude));
        res.add(currentDateTimeString);
        return res;
    }

    public void createEnterContextDialog(){
        final Dialog dia = new Dialog(MainActivity.this);
        dia.setContentView(R.layout.enter_context);
        dia.setTitle("Choose Template");

        templates = ContextTemplateUtilities.getTemplateNames(MainActivity.this);
        copyTemplates = templates;

        copyTemplates.add("Create New Template");
        for(int i=0;i<copyTemplates.size();i++)
            Log.i("Check", copyTemplates.get(i));

        Log.i("Check",copyTemplates.toString());

        Spinner spinner = (Spinner) dia.findViewById(R.id.spinner);
        spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {

//Toast.makeText(MainActivity.this,templates.get(position),Toast.LENGTH_SHORT).show();
                selectedPosition=position;
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {

            }
        });

        ArrayAdapter <String> dataAdapter = new ArrayAdapter <String> (this,
android.R.layout.simple_spinner_item, copyTemplates);

        // Drop down layout style - list view with radio button

dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

        // attaching data adapter to spinner
        spinner.setAdapter(dataAdapter);
```

```
        Button addData = (Button) dia.findViewById(R.id.add_data);
        addData.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                if(selectedPosition == copyTemplates.size()-1){
                    create_dialog();
                }
                else{
                    addInstanceData(copyTemplates.get(selectedPosition));
                }
                dia.dismiss();
            }
        });

        dia.show();
    }

    public void addInstanceData (final String tName){

        final JSONObject addedContextData = new JSONObject();
        try {
            addedContextData.put("id",fname);
        } catch (JSONException e) {
            e.printStackTrace();
        }

        ArrayList<Pair> attrData =
ContextTemplateUtilities.getAttributeList(tName,ContextTemplateUtilities.getContextDataFrom
SharedPrefs(MainActivity.this));
        Log.i("ContextMatch",attrData.toString());
        final Dialog dialog = new Dialog(MainActivity.this);
        dialog.setContentView(R.layout.add_instance);
        dialog.setTitle("Add Context Data");

        LinearLayout ll = (LinearLayout) dialog.findViewById(R.id.ll);

        final TextView[] textView = new TextView[attrData.size()-3];
        final EditText[] editText = new EditText[attrData.size()-3];
        int i;
        for (i=0;i<attrData.size()-3;i++){
            Log.i("Pairs",attrData.get(i).getLeft()+"    "+attrData.get(i).getRight());
            TextView temp = new TextView(MainActivity.this);
            temp.setText(attrData.get(i).getLeft());
            ll.addView(temp);
            textView[i]=temp;

            EditText tempet=new EditText(MainActivity.this);
            tempet.setHint(attrData.get(i).getRight());
            ll.addView(tempet);
            editText[i]=tempet;
        }

        if(i==0)
```

```
        {
            TextView temp = new TextView(MainActivity.this);
            temp.setText("All implicit attributes will be stored");
            temp.setTextSize(14);
            ll.addView(temp);
        }

        Button submit=new Button(MainActivity.this);
        submit.setText("Save");

        submit.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                for(int i=0;i<textView.length;i++){
                    Log.i("TextView", textView[i].getText().toString());
                    Log.i("EditText", editText[i].getText().toString());
                    try {

addedContextData.put(textView[i].getText().toString(),editText[i].getText().toString());

                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
                ArrayList<String> impCon = captureImplicitContext();
                try {
                    addedContextData.put("latitude",impCon.get(0));
                    addedContextData.put("longitude",impCon.get(1));
                    addedContextData.put("date_time",impCon.get(2));
                } catch (JSONException e) {
                    e.printStackTrace();
                }
                globalContextJSON=addedContextData;
                Log.i("StoredJSON",addedContextData.toString());

                db.setTableName(tName);
                db.addRow(addedContextData);
                Log.i("check1", addedContextData.toString());
                ImageUtilities.setExifData(addedContextData,fname);
                dialog.dismiss();
            }
        });

        ll.addView(submit);
        dialog.show();
    }

    public void queryDialog(){
        final Dialog dialog = new Dialog(MainActivity.this);
        dialog.setContentView(R.layout.query_dialog);
        dialog.setTitle("Query");

        final ArrayList<EditText> aNames=new ArrayList<EditText>();
        final ArrayList<EditText> aTypes=new ArrayList<EditText>();
```

```
final LinearLayout ll=(LinearLayout) dialog.findViewById(R.id.linlay2);

final RadioGroup radiodisp = (RadioGroup) dialog.findViewById(R.id.radiodisp);

Button add = (Button) dialog.findViewById(R.id.add2);
add.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
      EditText tempAttr = new EditText(MainActivity.this);
      tempAttr.setHint("Attr Name");
      EditText tempType = new EditText(MainActivity.this);
      tempType.setHint("Attr Value");
      ll.addView(tempAttr, 0);
      ll.addView(tempType, 1);
      aNames.add(tempAttr);
      aTypes.add(tempType);
   }
});

Button dialogButton = (Button) dialog.findViewById(R.id.confirm_button2);
// if button is clicked, close the custom dialog
dialogButton.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) {

      int selectedID = radiodisp.getCheckedRadioButtonId();
      RadioButton radiomethod = (RadioButton) dialog.findViewById(selectedID);

      AttributeList af = new AttributeList();
      List<AttributeList> al = new ArrayList<AttributeList>();

      JSONObject queryObj = new JSONObject();

      for(int i=0;i<aNames.size();i++){
            try {
               queryObj.put(aNames.get(i).getText().toString(),
aTypes.get(i).getText().toString());
            } catch (JSONException e) {
               e.printStackTrace();
            }
      }

      Log.i("QUERYOBJ", queryObj.toString());
      JSONArray queryResults = db.getRow(queryObj);
      Log.i("Result", queryResults.toString());

      String[] fPaths = new String[queryResults.length()];
      for(int i=0;i<queryResults.length();i++){
         try {
            String temp=queryResults.getJSONObject(i).getString("id");
            fPaths[i]=temp;
         } catch (JSONException e) {
            e.printStackTrace();
```

```
            }
        }

        if(radiomethod.getText().equals("Image Gallery")) {
            Bundle b = new Bundle();
            b.putStringArray("images", fPaths);
            b.putString("devdata", queryResults.toString());
            Intent i = new Intent(MainActivity.this, GridLayoutActivity.class);
            i.putExtras(b);
            startActivity(i);
        }
        else {
            Bundle b = new Bundle();
            b.putString("devdata", queryResults.toString());
            Intent i = new Intent(MainActivity.this, DevView.class);
            i.putExtras(b);
            startActivity(i);
        }

        dialog.dismiss();
    }
  });

  dialog.show();

}

public void create_dialog()
{
  // custom dialog
  final Dialog dialog = new Dialog(MainActivity.this);
  dialog.setContentView(R.layout.dialog_layout);
  dialog.setTitle("New Template");

  int ind = 2;

  final ArrayList<EditText> aNames=new ArrayList<EditText>();
  final ArrayList<EditText> aTypes=new ArrayList<EditText>();

  final LinearLayout ll=(LinearLayout) dialog.findViewById(R.id.linlay);

  Button add = (Button) dialog.findViewById(R.id.add);
  add.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        EditText tempAttr = new EditText(MainActivity.this);
        tempAttr.setHint("Attr Name");
        EditText tempType = new EditText(MainActivity.this);
        tempType.setHint("Attr Type");
        ll.addView(tempAttr, 2);
        ll.addView(tempType, 3);
        aNames.add(tempAttr);
        aTypes.add(tempType);
    }
```

```
        });

        Spinner template_extends = (Spinner) dialog.findViewById(R.id.template_extends);
        template_extends.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
{
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
                selectedExtendsPosition=position;
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {

            }
        });

        ArrayAdapter<String> dataAdapter = new ArrayAdapter <String> (this,
android.R.layout.simple_spinner_item, copyTemplates);

        // Drop down layout style - list view with radio button

dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

        // attaching data adapter to spinner
        template_extends.setAdapter(dataAdapter);


        Button dialogButton = (Button) dialog.findViewById(R.id.confirm_button);
        // if button is clicked, close the custom dialog
        dialogButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {

                // set the custom dialog components - text and button
                EditText template_name = (EditText) dialog.findViewById(R.id.template_name);

                AttributeList af = new AttributeList();
                List<AttributeList> al = new ArrayList<AttributeList>();

                for(int i=0;i<aNames.size();i++){
                    AttributeList temp=new AttributeList();
                    temp.setAttr_name(aNames.get(i).getText().toString());
                    temp.setAttr_type(aTypes.get(i).getText().toString());
                    al.add(temp);
                }

                tf.setTemplate_name(template_name.getText().toString());
                tf.setExt(templates.get(selectedExtendsPosition));
                tf.setAttribute_list(al);
                JSONArray ar = tf.convertJsonStore();
                Log.i("convertJsonStore", ar.toString());

                db.setTableName(template_name.getText().toString());
                db.createTable();
```

```
            addInstanceData(template_name.getText().toString());
            dialog.dismiss();
        }
    });

    dialog.show();

}



@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}

}
```

DatabaseHandler.java

```
package com.conware.smimage;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class DatabaseHandler extends SQLiteOpenHelper {

    String Table_Name;
    Context cont;
    Activity act;
    // All Static variables
    // Database Version
    private static final int DATABASE_VERSION = 2;
    // Database Name
    private static final String DATABASE_NAME = "templatesManager";

    public DatabaseHandler(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
        cont = context;
        act = (Activity) cont;
    }

    public void setTableName(String tableName)
    {
        Table_Name = tableName;
    }

    // Creating Tables
    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_IMAGETEMPLATE_TABLE = "CREATE TABLE " + "Image" + "("
                + " id" + " TEXT PRIMARY KEY," + "latitude" + " TEXT," + "longitude" + "
TEXT," + "date_time" + " TEXT" + ")";
        db.execSQL(CREATE_IMAGETEMPLATE_TABLE);
    }

    // Upgrading database
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
        // Drop older table if existed
        db.execSQL("DROP TABLE IF EXISTS " + Table_Name);

        // Create tables again
        onCreate(db);
    }

    public void createTable()
    {
        SQLiteDatabase db = this.getWritableDatabase();
        int i;
        Log.i("TABLE_NAME", Table_Name);
        Log.i("PREFSJSON",
ContextTemplateUtilities.getContextDataFromSharedPrefs(act).toString());
        ArrayList<Pair> attrs = ContextTemplateUtilities.getAttributeList(Table_Name,
ContextTemplateUtilities.getContextDataFromSharedPrefs(act));
        String CREATE_TEMPLATES_TABLE = "CREATE TABLE " + Table_Name + " (" +
"id TEXT PRIMARY KEY,";
        for(i = 0; i<attrs.size() - 1; i++)
        {
            CREATE_TEMPLATES_TABLE = CREATE_TEMPLATES_TABLE +
attrs.get(i).getLeft();
            if(attrs.get(i).getRight() == "string")
            {
                CREATE_TEMPLATES_TABLE = CREATE_TEMPLATES_TABLE + " TEXT,";
            }
            else
            {
                CREATE_TEMPLATES_TABLE = CREATE_TEMPLATES_TABLE + "
INTEGER,";
            }
        }
        if(attrs.get(i).getRight() == "string")
        {
            CREATE_TEMPLATES_TABLE = CREATE_TEMPLATES_TABLE +
attrs.get(i).getLeft() + " TEXT)";
        }
        else
        {
            CREATE_TEMPLATES_TABLE = CREATE_TEMPLATES_TABLE +
attrs.get(i).getLeft() + " INTEGER)";
        }
        Log.i("CREATETABLESTMT", CREATE_TEMPLATES_TABLE);
        db.execSQL(CREATE_TEMPLATES_TABLE);
        db.close();
    }

    public void addRow(JSONObject obj)
    {
        SQLiteDatabase db = this.getWritableDatabase();

        ContentValues values = new ContentValues();
        Iterator<String> iter = obj.keys();
        while (iter.hasNext()) {
```

```
        String key = iter.next();
        try {
            Object value = obj.get(key);
            values.put(key, value.toString());
        } catch (JSONException e) {
            // Something went wrong!
                e.printStackTrace();
        }
    }

    // Inserting Row
    db.insert(Table_Name, null, values);
    Log.i("INSERTTABLEVALUES", values.toString());
    db.close(); // Closing database connection

}

public JSONArray getRow(JSONObject queryparams)
{
    SQLiteDatabase db = this.getReadableDatabase();

    int k = 0;
    List<String> queryparamsattrs = new ArrayList<String>();
    List<String> queryparamsvalues = new ArrayList<String>();

    Iterator<String> iter = queryparams.keys();
    while (iter.hasNext()) {
        String key = iter.next();
        queryparamsattrs.add(key);
        try {
            Object value = queryparams.get(key);
            queryparamsvalues.add((String) value);
            k++;
        } catch (JSONException e) {
            // Something went wrong!
        }
    }
    Log.i("QUERYPARAMSVALUES", queryparamsvalues.toString());
    String sql = new String();
    Cursor cursor = null;

    //CASE 1

    if(queryparams.length() == 0)
    {
        List<String> table_names = new ArrayList<String>();
        table_names = ContextTemplateUtilities.getTemplateNames((Activity) cont);

        JSONArray arr = new JSONArray();

        int v;
        for(v = 0; v<table_names.size(); v++)
        {
            sql = "SELECT * from " + table_names.get(v);
```

```
            Log.i("QUERYSQL1", sql);
            cursor = db.rawQuery(sql, null);

            ArrayList<Pair> attrs = ContextTemplateUtilities.getAttributeList(table_names.get(v),
ContextTemplateUtilities.getContextDataFromSharedPrefs(act));
            String[] querycols = new String[50];
            querycols[0] = "id";
            for(int i = 0; i<attrs.size(); i++)
            {
               querycols[i+1] = attrs.get(i).getLeft();
            }

            try {
               if (cursor.moveToFirst()) {
                  do{
                     try {
                        JSONObject obj = new JSONObject();
                        for (int i = 0; i < attrs.size() + 1; i++) {
                           Log.i("querycols[i]", querycols[i] + " " + cursor.getString(i));
                           obj.put(querycols[i], cursor.getString(i));
                        }
                        arr.put(obj);
                     } catch (JSONException e) {
                        e.printStackTrace();
                     }
                  }while (cursor.moveToNext());
               }
            }finally{
               cursor.close();
            }

         }
         return arr;
      }
      else {
         if(queryparamsattrs.contains("template_name"))
         {

            //CASE 2


            if(queryparams.length() > 1) {
               JSONArray arr = new JSONArray();
               try {
                  sql = "SELECT * from " + queryparams.get("template_name").toString() + "
WHERE ";
               } catch (JSONException e) {
                  e.printStackTrace();
               }
               int j;
               queryparamsattrs.remove("template_name");
               for (j = 0; j < k - 2; j++) {
                  sql = sql + queryparamsattrs.get(j) + " = ? AND ";
               }
```

```
            sql = sql + queryparamsattrs.get(j) + " = ?";

            Log.i("QUERYSQL2", sql);
            try {
               queryparamsvalues.remove(queryparams.getString("template_name"));
            } catch (JSONException e) {
               e.printStackTrace();
            }
            String[] queryparamsvaluesarray = new String[queryparamsvalues.size()];
            queryparamsvaluesarray = queryparamsvalues.toArray(queryparamsvaluesarray);
            cursor = db.rawQuery(sql, queryparamsvaluesarray);

            ArrayList<Pair> attrs = null;
            try {
               attrs =
ContextTemplateUtilities.getAttributeList(queryparams.get("template_name").toString(),
ContextTemplateUtilities.getContextDataFromSharedPrefs(act));
            } catch (JSONException e) {
               e.printStackTrace();
            }
            String[] querycols = new String[50];
            querycols[0] = "id";
            for(int i = 0; i<attrs.size(); i++)
            {
               querycols[i+1] = attrs.get(i).getLeft();
            }


            try {
               if (cursor.moveToFirst()) {
                  do{
                     try {
                        JSONObject obj = new JSONObject();
                        for (int i = 0; i < attrs.size() + 1; i++) {
                           Log.i("querycols[i]", querycols[i] + " " + cursor.getString(i));
                           obj.put(querycols[i], cursor.getString(i));
                        }
                        arr.put(obj);
                     } catch (JSONException e) {
                        e.printStackTrace();
                     }
                  }while (cursor.moveToNext());
               }
            }finally{
               cursor.close();
            }

            return arr;

         }

         //CASE 3
```

```
            else
            {
                JSONArray arr = new JSONArray();
                try {
                    sql = "SELECT * from " + queryparams.get("template_name").toString();
                } catch (JSONException e) {
                    e.printStackTrace();
                }
                Log.i("QUERYSQL3", sql);
                cursor = db.rawQuery(sql, null);

                ArrayList<Pair> attrs = null;
                try {
                    attrs =
ContextTemplateUtilities.getAttributeList(queryparams.get("template_name").toString(),
ContextTemplateUtilities.getContextDataFromSharedPrefs(act));
                } catch (JSONException e) {
                    e.printStackTrace();
                }
                String[] querycols = new String[50];
                querycols[0] = "id";
                for(int i = 0; i<attrs.size(); i++)
                {
                    querycols[i+1] = attrs.get(i).getLeft();
                }


                try {
                    if (cursor.moveToFirst()) {
                        do{
                            try {
                                JSONObject obj = new JSONObject();
                                //Log.i("Querycols", String.valueOf(querycols.length));
                                //Log.i("Cursor", String.valueOf(cursor.getColumnCount()));
                                for (int i = 0; i < attrs.size() + 1; i++) {
                                    Log.i("querycols[i]", querycols[i] + " " + cursor.getString(i));
                                    obj.put(querycols[i], cursor.getString(i));
                                }
                                arr.put(obj);
                                //j++;
                            } catch (JSONException e) {
                                e.printStackTrace();
                            }
                        }while (cursor.moveToNext());
                    }
                }finally{
                    cursor.close();
                }
                return arr;
            }
        }

        //CASE 4
```

```
        else
        {

            List<String> table_names = new ArrayList<String>();
            table_names = ContextTemplateUtilities.getTemplateNames((Activity) cont);

            JSONArray arr = new JSONArray();

            int v;
            for(v = 0; v<table_names.size(); v++) {

                ArrayList<Pair> attrs =
ContextTemplateUtilities.getAttributeList(table_names.get(v),
ContextTemplateUtilities.getContextDataFromSharedPrefs(act));
                String[] querycols = new String[50];
                querycols[0] = "id";
                for (int i = 0; i < attrs.size(); i++) {
                    querycols[i + 1] = attrs.get(i).getLeft();
                }
                int outerflag = 1;
                for (int j = 0; j < queryparamsattrs.size(); j++) {
                    int flag = 0;
                    for (int d = 0; d < attrs.size() + 1; d++) {
                        if (querycols[d].equals(queryparamsattrs.get(j))) {
                            flag = 1;
                            break;
                        }
                    }
                    if (flag == 0) {
                        outerflag = 0;
                        break;
                    }
                }
                if (outerflag == 1) {

                    sql = "SELECT * from " + table_names.get(v) + " WHERE ";
                    int q;
                    for (q = 0; q < k - 1; q++) {
                        sql = sql + queryparamsattrs.get(q) + " = ? AND ";
                    }
                    sql = sql + queryparamsattrs.get(q) + " = ?";
                    Log.i("QUERYSQL4", sql);

                    String[] queryparamsvaluesarray = new String[queryparamsvalues.size()];
                    queryparamsvaluesarray = queryparamsvalues.toArray(queryparamsvaluesarray);
                    cursor = db.rawQuery(sql, queryparamsvaluesarray);


                    try {
                        if (cursor.moveToFirst()) {
                            do {
                                try {
                                    JSONObject obj = new JSONObject();
```

```
                        for (int i = 0; i < attrs.size() + 1; i++) {
                            Log.i("querycols[i]", querycols[i] + " " + cursor.getString(i));
                            obj.put(querycols[i], cursor.getString(i));
                        }
                        arr.put(obj);
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                } while (cursor.moveToNext());
            }
        } finally {
            cursor.close();

        }

    }

    return arr;
  }
 }
}
```

ContextTemplateUtilities.java

```
package com.conware.smimage;

import android.app.Activity;
import android.util.Log;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.List;

public class ContextTemplateUtilities {

   final static String SHARED_PREF_KEY = "Templates";
   final static String DEFAULT_PREF_VALUE = "Error";
   final static String ATTR_LIST_TAG = "attribute_list";

   public static JSONArray getContextDataFromSharedPrefs(Activity act){
      try {
         Log.i("SharedPref",Prefs.getMyStringPref(act.getApplicationContext(),
SHARED_PREF_KEY));

if(Prefs.getMyStringPref(act.getApplicationContext(),SHARED_PREF_KEY).equals("default"))
         {
            JSONArray templatear = new JSONArray();
            JSONObject temp = new JSONObject();
            JSONArray tempar = new JSONArray();
            try {
               JSONObject t = new JSONObject();
               t.put("attr_name", "latitude");
               t.put("attr_type", "String");
               tempar.put(t);
               JSONObject t1 = new JSONObject();
               t1.put("attr_name", "longitude");
               t1.put("attr_type", "String");
               tempar.put(t1);
               JSONObject t2 = new JSONObject();
               t2.put("attr_name", "date_time");
               t2.put("attr_type", "String");
               tempar.put(t2);

               temp.put("template_name", "Image");
               temp.put("extends", "None");
               temp.put("attribute_list", tempar);
               templatear.put(temp);
            } catch (JSONException e) {
               e.printStackTrace();
            };
            return templatear;
         }
         else {
```

```
            return new
JSONArray(Prefs.getMyStringPref(act.getApplicationContext(),SHARED_PREF_KEY));
        }
      } catch (JSONException e) {
        e.printStackTrace();
        return null;
      }
   }

   public static List<String> getTemplateNames(Activity act){
      JSONArray jsonArray = getContextDataFromSharedPrefs(act);

      List<String> res = new ArrayList<String>();
      try{
        for(int i=0;i<jsonArray.length();i++){
          res.add(jsonArray.getJSONObject(i).getString("template_name"));
        }
      } catch (JSONException e){
        e.printStackTrace();
      }

      return res;
   }

   public static ArrayList<Pair> getAttributeList(String tName, JSONArray contextTemplates){

      ArrayList<String> attrNames = new ArrayList<String>();
      ArrayList<String> attrTypes = new ArrayList<String>();

      try {
        JSONObject matchedContextTemplate = getMatchedContextTemplate(tName,
contextTemplates);
        JSONArray attrList = matchedContextTemplate.getJSONArray(ATTR_LIST_TAG);
        for (int i=0;i<attrList.length();i++){
          attrNames.add(attrList.getJSONObject(i).getString("attr_name"));
          attrTypes.add(attrList.getJSONObject(i).getString("attr_type"));
        }

        //From Ancestors
        if(!matchedContextTemplate.getString("extends").equals("None")){
          ArrayList<Pair> parentData =
getAttributeList(matchedContextTemplate.getString("extends"),contextTemplates);
          for (int i=0;i<parentData.size();i++){
            attrNames.add(parentData.get(i).getLeft());
            attrTypes.add(parentData.get(i).getRight());
          }
        }


      } catch (JSONException e) {
        e.printStackTrace();
      }

      //Set Pairs
```

```
        ArrayList<Pair> result = new ArrayList<Pair>();
        for (int i=0;i<attrNames.size();i++){
            Pair temp = new Pair(attrNames.get(i),attrTypes.get(i));
            result.add(temp);
        }

        return result;
    }

    private static JSONObject getMatchedContextTemplate(String tName, JSONArray
contextTemplates) {
        for (int i=0; i<contextTemplates.length(); i++){
            try {
                if (contextTemplates.getJSONObject(i).getString("template_name").equals(tName))
                    return contextTemplates.getJSONObject(i);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

}
```

ImageUtilities.java

```
package com.conware.smimage;

import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.media.ExifInterface;
import android.net.Uri;
import android.os.Environment;

import org.json.JSONObject;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Calendar;

public class ImageUtilities {
   static final String ATTRIBUTE_USED = "UserComment";
   static final String EXTRACT_ERROR = "Error";

   //Save image to SD Card with filename as date and time of image capture
   public static String saveImage(Bitmap bm, Context con){
      String root = Environment.getExternalStorageDirectory().toString();
      String baseDirectory = root + "/SmImage";
      File myDir = new File(baseDirectory);
      if (!myDir.isDirectory()){
         myDir.mkdirs();
      }

      Calendar c = Calendar.getInstance();
      int seconds = c.get(Calendar.SECOND);
      String fname = Integer.toString(seconds)+".jpg";

      File file = new File (myDir, fname);
      if (file.exists ()){
         file.delete ();
      }

      try {
         FileOutputStream out = new FileOutputStream(file);
         bm.compress(Bitmap.CompressFormat.JPEG, 100, out);
         out.flush();
         out.close();

      } catch (Exception e) {
         e.printStackTrace();
      }

      String fpath = baseDirectory+"/"+fname;
      galleryAddPic(fpath,con);
      return fpath;
   }
```

```
    //Adds the JSONObject of context data into the JPEG file as an exif tag's data
    public static void setExifData(JSONObject exifData, String filePath){
        ExifInterface exif;
        try {
            exif = new ExifInterface(filePath);
            exif.setAttribute(ATTRIBUTE_USED, exifData.toString());
            exif.saveAttributes();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //Extracts context string from image
    public static String getExifData(String filePath){
        ExifInterface exif;
        String res;
        try {
            exif = new ExifInterface(filePath);
            return exif.getAttribute(ATTRIBUTE_USED);
        } catch (IOException e) {
            e.printStackTrace();
            return EXTRACT_ERROR;
        }
    }

    //Adds Image To Android Gallery
    public static void galleryAddPic(String mCurrentPhotoPath, Context con) {
        Intent mediaScanIntent = new Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE);
        File f = new File(mCurrentPhotoPath);
        Uri contentUri = Uri.fromFile(f);
        mediaScanIntent.setData(contentUri);
        con.sendBroadcast(mediaScanIntent);
    }
}
```

ContextTemplateUtilities.java

```
package com.conware.smimage;

import android.app.Activity;
import android.util.Log;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.List;

public class ContextTemplateUtilities {

   final static String SHARED_PREF_KEY = "Templates";
   final static String DEFAULT_PREF_VALUE = "Error";
   final static String ATTR_LIST_TAG = "attribute_list";

   public static JSONArray getContextDataFromSharedPrefs(Activity act){
      try {
         Log.i("SharedPref",Prefs.getMyStringPref(act.getApplicationContext(),
SHARED_PREF_KEY));

if(Prefs.getMyStringPref(act.getApplicationContext(),SHARED_PREF_KEY).equals("default"))
         {
            JSONArray templatear = new JSONArray();
            JSONObject temp = new JSONObject();
            JSONArray tempar = new JSONArray();
            try {
               JSONObject t = new JSONObject();
               t.put("attr_name", "latitude");
               t.put("attr_type", "String");
               tempar.put(t);
               JSONObject t1 = new JSONObject();
               t1.put("attr_name", "longitude");
               t1.put("attr_type", "String");
               tempar.put(t1);
               JSONObject t2 = new JSONObject();
               t2.put("attr_name", "date_time");
               t2.put("attr_type", "String");
               tempar.put(t2);

               temp.put("template_name", "Image");
               temp.put("extends", "None");
               temp.put("attribute_list", tempar);
               templatear.put(temp);
            } catch (JSONException e) {
               e.printStackTrace();
            };
            return templatear;
         }
         else {
```

```
            return new
JSONArray(Prefs.getMyStringPref(act.getApplicationContext(),SHARED_PREF_KEY));
        }
      } catch (JSONException e) {
        e.printStackTrace();
        return null;
      }
   }

   public static List<String> getTemplateNames(Activity act){
      JSONArray jsonArray = getContextDataFromSharedPrefs(act);

      List<String> res = new ArrayList<String>();
      try{
        for(int i=0;i<jsonArray.length();i++){
          res.add(jsonArray.getJSONObject(i).getString("template_name"));
        }
      } catch (JSONException e){
        e.printStackTrace();
      }

      return res;
   }

   public static ArrayList<Pair> getAttributeList(String tName, JSONArray contextTemplates){

      ArrayList<String> attrNames = new ArrayList<String>();
      ArrayList<String> attrTypes = new ArrayList<String>();

      try {
        JSONObject matchedContextTemplate = getMatchedContextTemplate(tName,
contextTemplates);
        JSONArray attrList = matchedContextTemplate.getJSONArray(ATTR_LIST_TAG);
        for (int i=0;i<attrList.length();i++){
          attrNames.add(attrList.getJSONObject(i).getString("attr_name"));
          attrTypes.add(attrList.getJSONObject(i).getString("attr_type"));
        }

        //From Ancestors
        if(!matchedContextTemplate.getString("extends").equals("None")){
          ArrayList<Pair> parentData =
getAttributeList(matchedContextTemplate.getString("extends"),contextTemplates);
          for (int i=0;i<parentData.size();i++){
            attrNames.add(parentData.get(i).getLeft());
            attrTypes.add(parentData.get(i).getRight());
          }
        }


      } catch (JSONException e) {
        e.printStackTrace();
      }

      //Set Pairs
```

```
        ArrayList<Pair> result = new ArrayList<Pair>();
        for (int i=0;i<attrNames.size();i++){
            Pair temp = new Pair(attrNames.get(i),attrTypes.get(i));
            result.add(temp);
        }

        return result;
    }

    private static JSONObject getMatchedContextTemplate(String tName, JSONArray
contextTemplates) {
        for (int i=0; i<contextTemplates.length(); i++){
            try {
                if (contextTemplates.getJSONObject(i).getString("template_name").equals(tName))
                    return contextTemplates.getJSONObject(i);
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        return null;
    }

}
```

Base64Utilities.java

```
package com.conware.smimage;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.util.Base64;

import org.json.JSONObject;

import java.io.ByteArrayOutputStream;

public class Base64Utilities {

    //Convert Bitmap to Base64 String
    public static String bitmapToBase64(Bitmap bm){
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        bm.compress(Bitmap.CompressFormat.JPEG, 100, baos); //bm is the bitmap object
        byte[] byteArrayImage = baos.toByteArray();
        return Base64.encodeToString(byteArrayImage, Base64.DEFAULT);
    }

    //Convert JSONObject to Base64String
    public static String jsonToBase64(JSONObject jo){
        String jsonString = jo.toString();
        return Base64.encodeToString(jsonString.getBytes(), Base64.DEFAULT);
    }

    //Append two Base64 Strings
    public static String appendBase64Stings(String str1, String str2){
        return str1+str2;
    }

    //Convert Base64 to Bitmap
    public static Bitmap convertBase64toBitmap(String encodedImage){
        byte[] decodedString = Base64.decode(encodedImage, Base64.DEFAULT);
        return BitmapFactory.decodeByteArray(decodedString, 0, decodedString.length);
    }

}
```

Prefs.java

```
package com.conware.smimage;

import android.content.Context;
import android.content.SharedPreferences;
import android.util.Log;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class Prefs {

   private static SharedPreferences getPrefs(Context context) {
      return context.getSharedPreferences("myprefs", 0);
   }

   public static String getMyStringPref(Context context, String name) {
      //Log.i("isInitialised", String.valueOf(isInitialised));
      if(Prefs.getMyBoolPref(context, "isInitialised") == false)
      {
         JSONArray templatear = new JSONArray();
         JSONObject temp = new JSONObject();
         JSONArray tempar = new JSONArray();
         try {
               JSONObject t = new JSONObject();
               t.put("attr_name", "latitude");
               t.put("attr_type", "String");
               tempar.put(t);
               JSONObject t1 = new JSONObject();
               t1.put("attr_name", "longitude");
               t1.put("attr_type", "String");
               tempar.put(t1);
               JSONObject t2 = new JSONObject();
               t2.put("attr_name", "date_time");
               t2.put("attr_type", "String");
               tempar.put(t2);

               temp.put("template_name", "Image");
               temp.put("extends", "None");
               temp.put("attribute_list", tempar);
               templatear.put(temp);
               Prefs.setMyStringPref(context, "Templates", templatear.toString());
               Prefs.setMyBoolPref(context, "isInitialised", true);
         } catch (JSONException e) {
            e.printStackTrace();
         };
         return(templatear.toString());
      }
      else
      {
         Log.i("NamefromPrefs", name);
         Log.i("ArrayfromPrefs", getPrefs(context).getString(name, "default").toString());
```

```
            return getPrefs(context).getString(name, "default");
        }
    }

    public static void setMyStringPref(Context context,String name, String value) {
        // perform validation etc..
        getPrefs(context).edit().putString(name, value).commit();
    }

    public static boolean getMyBoolPref(Context context,String name) {
        // perform validation etc..
        return getPrefs(context).getBoolean(name, false);
    }

    public static void setMyBoolPref(Context context,String name, boolean value) {
        // perform validation etc..
        getPrefs(context).edit().putBoolean(name, value).commit();
    }

    public static void deletePref(Context context)
    {
        getPrefs(context).edit().clear().commit();
    }

    public static void delPref(Context context, String s)
    {
        getPrefs(context).edit().remove(s).commit();
    }

    public static boolean containsBool(Context context, String name)
    {
        return getPrefs(context).contains(name);
    }

}
```

Pair.java

```
package com.conware.smimage;

public class Pair {
   private final String left;
   private final String right;

   public Pair(String left, String right) {
      this.left = left;
      this.right = right;
   }

   public String getLeft() { return left; }
   public String getRight() { return right; }

   @Override
   public int hashCode() { return left.hashCode() ^ right.hashCode(); }

   @Override
   public boolean equals(Object o) {
      if (!(o instanceof Pair)) return false;
      Pair pairo = (Pair) o;
      return this.left.equals(pairo.getLeft()) &&
            this.right.equals(pairo.getRight());
   }
}
```

ImageAdapter.java

```
package com.conware.smimage;

import android.content.Context;
import android.content.Intent;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.os.Bundle;
import android.view.View;
import android.view.ViewGroup;
import android.widget.BaseAdapter;
import android.widget.GridView;
import android.widget.ImageView;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.io.File;

public class ImageAdapter extends BaseAdapter{
    private Context mContext;
    private String[] mpaths;
    private JSONArray mData;

    // Constructor
    public ImageAdapter(Context c, String[] paths, JSONArray data){
        mContext = c;
        mpaths=paths;
        mData=data;
    }

    @Override
    public int getCount() {
        return mpaths.length;
    }

    @Override
    public Object getItem(int position) {
        return mpaths[position];
    }

    @Override
    public long getItemId(int position) {
        return 0;
    }

    @Override
    public View getView(final int position, View convertView, ViewGroup parent) {
        ImageView imageView = new ImageView(mContext);

        imageView.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
            public void onClick(View v) {
                try {
                    JSONObject toSend=mData.getJSONObject(position);
                    Bundle b = new Bundle();
                    b.putString("imageData",toSend.toString() );
                    b.putString("imagePath",mpaths[position]);
                    Intent i = new Intent(mContext, ImageDesc.class);
                    i.putExtras(b);
                    mContext.startActivity(i);
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        });

        File imgFile = new  File(mpaths[position]);

        if(imgFile.exists()){

            Bitmap myBitmap = BitmapFactory.decodeFile(imgFile.getAbsolutePath());
            imageView.setImageBitmap(myBitmap);

        }
        imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
        imageView.setLayoutParams(new GridView.LayoutParams(160, 160));
        return imageView;
    }


}
```

GridLayoutActivity.java

```
package com.conware.smimage;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.widget.GridView;

import org.json.JSONArray;
import org.json.JSONException;

public class GridLayoutActivity extends Activity{
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.grid_view);

      Bundle b=this.getIntent().getExtras();
      String[] array=b.getStringArray("images");
      JSONArray allJSON = new JSONArray();
      try {
         allJSON = new JSONArray(b.getString("devdata"));
      } catch (JSONException e) {
         e.printStackTrace();
      }

      GridView gridView = (GridView) findViewById(R.id.grid_view);

      Log.i("Passed Images",array.toString());
      Log.i("Passed Images",array[0]);

      // Instance of ImageAdapter Class
      gridView.setAdapter(new ImageAdapter(GridLayoutActivity.this,array,allJSON));
   }
}
```

DevView.java

```
package com.conware.smimage;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class DevView extends Activity{
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.dev_view);

      Bundle b=this.getIntent().getExtras();
      String devdata=b.getString("devdata");

      TextView tv=(TextView) findViewById(R.id.devtext);
      tv.setText(devdata);
   }
}
```

ImageDesc.java

```
package com.conware.smimage;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.TextView;

import org.json.JSONException;
import org.json.JSONObject;

import java.io.File;
import java.util.ArrayList;
import java.util.Iterator;


public class ImageDesc extends ActionBarActivity {

    ImageView iv;
    ArrayList<Pair> ap;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_image_desc);

        iv=(ImageView)findViewById(R.id.image_view);
        ap=new ArrayList<Pair>();

        Bundle b=this.getIntent().getExtras();
        try {
            JSONObject iData = new JSONObject(b.getString("imageData"));
            Iterator<String> iter = iData.keys();
            while(iter.hasNext()){
                String key = iter.next();
                String val = iData.getString(key);
                Pair temp=new Pair(key,val);
                ap.add(temp);
            }

            //Code to dynamically populate views
            final LinearLayout descll= (LinearLayout) findViewById(R.id.descll);
            File imgFile = new  File(b.getString("imagePath"));

            if(imgFile.exists()){

                Bitmap myBitmap = BitmapFactory.decodeFile(imgFile.getAbsolutePath());
                iv.setImageBitmap(myBitmap);
```

```
            }

            for(int i = 0; i<ap.size(); i++) {
                LinearLayout attrll = new LinearLayout(ImageDesc.this);
                attrll.setOrientation(LinearLayout.HORIZONTAL);
                TextView tv1 = new TextView(ImageDesc.this);
                tv1.setText(ap.get(i).getLeft() + "           ");
                TextView tv2 = new TextView(ImageDesc.this);
                tv2.setText(ap.get(i).getRight());
                attrll.addView(tv1);
                attrll.addView(tv2);
                descll.addView(attrll);
            }


        } catch (JSONException e) {
            e.printStackTrace();
        }
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_image_desc, menu);
        return true;
    }

}
```

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity"
   android:orientation="vertical"
   android:weightSum="4">

   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:orientation="horizontal"
      android:weightSum="2"
      android:layout_weight="1">

         <Button
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:id="@+id/captureImage"
            android:layout_weight="1"
            android:text="Capture Image" />

         <Button
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:id="@+id/query"
            android:layout_weight="1"
            android:text="Query" />

   </LinearLayout>

      <ImageView
         android:id="@+id/capturedImage"
         android:layout_width="match_parent"
         android:layout_height="0dp"
         android:layout_weight="3"
         android:src="@drawable/ic_launcher"
         android:layout_gravity="center"/>


</LinearLayout>
```

dialog_layout.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent"
    android:orientation="vertical"
    android:weightSum="5"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:minWidth="300dp"
    android:minHeight="400dp"
    android:id="@+id/linlay">

    <EditText
        android:id="@+id/template_name"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_marginBottom="10dp"
        android:hint="Template Name"/>

    <Spinner
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_marginBottom="10dp"
        android:id="@+id/template_extends"></Spinner>

    <Button
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="Add Attr"
        android:layout_marginBottom="10dp"
        android:id="@+id/add"/>

    <Button
        android:id="@+id/confirm_button"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="OK"/>

</LinearLayout>
```

query_dialog.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent" android:layout_height="match_parent"
   android:orientation="vertical"
   android:weightSum="3"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:id="@+id/linlay2">

   <RadioGroup
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:id="@+id/radiodisp">

      <RadioButton
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Image Gallery"
         android:checked="true"
         android:id="@+id/radioimg"/>

      <RadioButton
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="JSON Developer Interface"
         android:id="@+id/radiodev"/>

   </RadioGroup>

   <Button
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:text="Add Attr"
      android:id="@+id/add2"/>

   <Button
      android:id="@+id/confirm_button2"
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:layout_weight="1"
      android:text="Execute Query"/>

</LinearLayout>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.conware.smimage" >

   <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
   <uses-permission android:name="android.permission.CAMERA" />
   <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
   <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
   <uses-feature
      android:name="android.hardware.camera"
      android:required="true" />

   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name=".MainActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity
         android:name=".DevView"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.DEVVIEW" />

            <category android:name="android.intent.category.DEFAULT" />
         </intent-filter>
      </activity>
      <activity
         android:name=".GridLayoutActivity"
         android:label="@string/app_name" >
         <intent-filter>
            <action android:name="android.intent.action.GRIDLAYOUTACTIVITY" />

            <category android:name="android.intent.category.DEFAULT" />
         </intent-filter>
      </activity>
      <activity
         android:name=".ImageDesc"
         android:label="@string/title_activity_image_desc" >
      </activity>
   </application>

</manifest>
```