SDSU DINING – AN ANDROID MOBILE APPLICATION SOLUTION FOR SDSU DINING SERVICES

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Priya Jayaprakash

Fall 2014
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Priya Jayaprakash:

SDSU Dining – An Android Mobile Application Solution for
SDSU Dining Services

Roger Whitney, Chair
Department of Computer Science

Alan Riggins
Department of Computer Science

Eric Frost
Graduate Program in Homeland Security

Oct 24, 2014
Approval Date
Copyright © 2014
by
Priya Jayaprakash
All Rights Reserved
DEDICATION

I dedicate my thesis to my parents – Dr. Jayaprakash Valiakolleri and Dr. Meena Panthayi, and my brother – Rajesh Jayaprakash for their unceasing support and encouragement throughout the course of this thesis. I would also like to dedicate this thesis to my professors who have been a constant source of guidance and inspiration.
It always seems impossible until it is done

– Nelson Mandela
ABSTRACT OF THE THESIS

SDSU Dining – An Android Mobile Application Solution for SDSU Dining Services

by

Priya Jayaprakash

Master of Science in Computer Science
San Diego State University, 2014

Mobile technology and the use of mobile devices have increased tremendously within the past few years. San Diego State University (SDSU) Dining Services has decided to create a native Android application to reach out, inform, and promote their diverse restaurants, catering services, and many other offerings. To keep up with the high student demand for “search on-the-go” applications, students will now have the ability to search and stay informed on SDSU Dining Services without having to go on their website. The information is now available at their fingertips. The six main categories developed and featured during the initial release of this application include Restaurants, Farmer’s Market, Coupons, SDSU Catering, Sweet, and Contact Us. Push notification feature has also been incorporated which notifies users on spontaneous events and promotions offered by the Dining Services. The primary focus of this native Android application is to provide users with the ability to access dining information on-the-go with ease and simplicity.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
</tr>
</tbody>
</table>

## CHAPTER

<table>
<thead>
<tr>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 INTRODUCTION</td>
</tr>
<tr>
<td>2 PROJECT REQUIREMENTS AND RESEARCH</td>
</tr>
<tr>
<td>2.1 Application Requirements</td>
</tr>
<tr>
<td>2.2 Research</td>
</tr>
<tr>
<td>2.2.1 Web Service APIs</td>
</tr>
<tr>
<td>2.2.2 Image Loading APIs</td>
</tr>
<tr>
<td>3 TECHNOLOGIES</td>
</tr>
<tr>
<td>3.1 Android</td>
</tr>
<tr>
<td>3.1.1 Android Platform</td>
</tr>
<tr>
<td>3.1.2 Android Architecture</td>
</tr>
<tr>
<td>3.2 AQuery API</td>
</tr>
<tr>
<td>3.3 Web Service Emulator</td>
</tr>
<tr>
<td>3.4 Restful JSON Data</td>
</tr>
<tr>
<td>3.5 Android SQLite Database</td>
</tr>
<tr>
<td>3.6 Google Cloud Messaging for Android</td>
</tr>
<tr>
<td>4 APPLICATION DESIGN</td>
</tr>
<tr>
<td>4.1 Project Timeline</td>
</tr>
<tr>
<td>4.2 Storyboard</td>
</tr>
<tr>
<td>4.3 App Flowchart</td>
</tr>
<tr>
<td>5 IMPLEMENTATION</td>
</tr>
<tr>
<td>5.1 Development Environment</td>
</tr>
<tr>
<td>5.2 App Launch</td>
</tr>
</tbody>
</table>
5.3 Restaurants

5.4 Coupons

5.5 Farmer’s Market, SDSU Catering, Sweet, and Contact Us

5.6 Push Notification

5.7 Application Demo Phases

6  CHALLENGES OVERCOME

7  VERSION 2.0 FUTURE ENHANCEMENTS

8  CONCLUSION

REFERENCES
LIST OF TABLES

Table 4.1. Detailed Reference of Figure 4.1 ................................................................. 17
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Android architectural layers</td>
<td>8</td>
</tr>
<tr>
<td>4.1</td>
<td>Draft project timeline</td>
<td>16</td>
</tr>
<tr>
<td>4.2</td>
<td>Storyboard vs. Version 1.0 comparison</td>
<td>18</td>
</tr>
<tr>
<td>4.3</td>
<td>SDSU Dining app flowchart</td>
<td>19</td>
</tr>
<tr>
<td>5.1</td>
<td>Splash screen</td>
<td>22</td>
</tr>
<tr>
<td>5.2</td>
<td>Home screen</td>
<td>25</td>
</tr>
<tr>
<td>5.3</td>
<td>Restaurants selection</td>
<td>26</td>
</tr>
<tr>
<td>5.4</td>
<td>Restaurant timings status</td>
<td>26</td>
</tr>
<tr>
<td>5.5</td>
<td>Coupons screen</td>
<td>28</td>
</tr>
<tr>
<td>5.6</td>
<td>Reusable fragments</td>
<td>28</td>
</tr>
<tr>
<td>5.7</td>
<td>SectionsPagerAdapter fragment creation</td>
<td>30</td>
</tr>
<tr>
<td>5.8</td>
<td>SDSU Dining push notification</td>
<td>31</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Dr. Roger Whitney, my graduate advisor and thesis chair, for his support, guidance, and patience through the progression of my thesis. His invaluable advice and expertise in mobile technology, object-oriented programming, and software development have aided me tremendously for the success of this thesis. I would also like to thank Professor Alan Riggins and Dr. Eric Frost for providing additional advice and for accepting to be a part of my thesis committee. Last but definitely not least, I would like to extend my thanks to Jahan Jamshidi, Keli Hazelton, Kane Luong, and Ben Eisenstein of the SDSU Dining Services team for their valuable inputs, assistance, and support for the delivery of this project.
CHAPTER 1

INTRODUCTION

The primary objective of this thesis is to develop and deliver a functional, robust, and modular Android mobile application catered towards the needs and requirements of the SDSU Dining Services. The Android operating system (OS) is based on Linux and it is developed and maintained by Google. It is a very versatile operating system and is supported by many platforms. According to Android Developers, “Every day more than 1 million new devices are activated worldwide” and analysts predict that by 2017 Android will be shipping a billion handsets a year [1, 2]. The deciding factor on whether to develop an Android application or an Apple iOS application was primarily based on Android being open source. Another reason is because Apple enforces strictly controlled application development rules. Google on the other hand, simply provides a set of Android best practices. With these factors in mind, the SDSU Dining Services has pursued to build an Android mobile application as its first choice. As part of the Version 1.0 release, deliverables were broken down to six different categories: Restaurants, Farmer’s Market, Coupons, SDSU Catering, Sweet, and Contact Us. The app provides a listing of all the various eateries around SDSU campus along with the eatery hours and status. It also provides contact information, coupons and promotions for various eateries, catering and sweet, and notifications of flash events that would occur throughout the year. Since this project acts as an underlying framework, the app has been designed with various enhancement hooks and capabilities. Additional features that can be implemented are covered in depth under the Version 2.0 Future Enhancements chapter. SDSU Dining Services definitely has an advantage for building a native Android application – not only can they reach out to all the students, parents, and visitors of San Diego State University, but it also opens a wide range of possibilities of a fine college dining experience for the users. The positives out way the cost of building a native mobile application and SDSU Dining Services has been thoroughly supportive of the idea. Engineers and programmers must understand that coding and launching a deliverable application will not happen overnight. In this thesis, I will focus on the many different aspects that
encompass software engineering, project design and architecture, and core milestones for a successful launch.
CHAPTER 2

PROJECT REQUIREMENTS AND RESEARCH

2.1 APPLICATION REQUIREMENTS

Application requirements always change. Throughout the entire course of this thesis, the requirements have modified numerous times. For example, the initial postulation was that the Farmer’s Market category would have a similar layout as Restaurants. A complete listing of the market vendors would be provided, and users would have the option to explore details on each vendor. However this was later changed due to the fact that vendors change very often, some as often as every week. This would lead to high maintenance costs in order keep vendor data up-to-date and prevent false advertising to end users. Another example was the change in design from using SOAP web service to RESTful data. The reason behind this change is explained in detail in Section 2.2.1. Finally, the SDSU Dining Services team required an expedited incorporation of push notification feature from release 2.0 to release 1.0. Programmers must be prepared for code alterations for the entire lifetime of their application. Applications must be functional, robust, and user-friendly. After all, the success of an application is highly dependent on the positive feedback of users. After working closely with the Dining Services engineering, graphics design, and marketing teams the final specifications for the Version 1.0 release are as follows:

1. Create six main categories – Restaurants, Farmer’s Market, Coupons, SDSU Catering, Sweet, and Contact Us.
2. On app launch event, load a Splash screen displaying the SDSU Dining Services logo before redirecting to the Home screen.
3. Any layout with list-type entries will follow a consistent ListView layout
4. Font face will be Century Gothic and Roboto, and must stay consist for the entire application.
5. For consistency throughout the app, the following are standardized information that must be provided to users whenever applicable:
- Contact must contain phone number, fax, email, website, and address
- Hours must display hours Monday through Sunday for the item displayed
- About must provide a useful informational snippet of the item displayed

6. Restaurant logo images will reside within the application as opposed to keeping it on the server.

7. Restaurants screen will consist of a ListView of all the different restaurants around the SDSU campus area.

8. Restaurants screen will contain a Filter Location button which will list all the different eatery locations throughout campus. This will provide the user with the ability to narrow their search to a particular location.

9. Based on the location selected by the user, display a ListView of all the different restaurants at the particular location.

10. When a particular eatery is selected, dynamically create Fragments with a swipeable Tab Selected ActionBar corresponding to each fragment. Each tab represents a unique location of the eatery throughout the SDSU campus area.

11. Upon selecting an eatery, display a snapshot on the restaurant status; ie. Open, Closed, or \( x \) mins to Close if less than an hour till close. Also display the eatery hours, phone number, and website.

12. Farmers Market, SDSU Catering, Sweet, and Contact Us will each contain three fragments with swipeable Tab Selected ActionBar corresponding to each fragment – Contact, Hours, and About.

13. Coupons will provide a ListView of all the coupons currently provided by SDSU Dining Services. Coupons will be ordered by expiring date with most recent on top.

14. The app will also provide Push Notification capabilities. This will be the primary source for communicating various flash events and deals offered throughout the year.

### 2.2 Research

A very crucial process of any application development is research. Throughout the course of this thesis various researches were conducted to acquaint with the multitude of technologies and open source Application Programming Interfaces (APIs) available. It is also equally important to be able to narrow down the technologies and APIs to the ones most applicable and suited for the project.

#### 2.2.1 Web Service APIs

During the initial design and planning phase of the app, SDSU Dining Services had decided to use Simple Object Access Protocol (SOAP) web service calls for fetching and displaying back-end data. SOAP is important for application development to allow Internet
communication between programs. A SOAP message is an ordinary XML document with data that is transferred via HTTP [3]. After researching thoroughly, I found that most of the web service Android application examples use KSOAP2-Android API. Instead of reinventing the wheel and writing my own SOAP layer, I decided to use the KSOAP2 API which is licensed under MIT and can therefore be included in commercial applications. The API provides a lightweight and efficient SOAP client library for the Android platform. It is comprised of an XML pull parser, a de/serializer, and a transport layer. The de/serializer is responsible for mapping the object representation to the XML representation and back again. The transport layer provides the mechanism on which the SOAP messages are exchanged between server and client [4]. Using the KSOAP2 API, I was able to successfully implement a functional lightweight SOAP web service data fetcher catered specifically for the SDSU Dining application.

Several months into the implementation of this project, the SDSU Dining Services teams opted to step away from using SOAP web service and instead pursue using Representational State Transfer (REST) data. The reason was because REST data is simple to understand, secure, stateless, scalable, and leveraged by all major Web 2.0 applications. REST is the architecture of the World Wide Web. It can be accessed using Uniform Resource Identification (URI) and has a uniform interface for all resources: GET, POST, PUT, and DELETE [5]. Additionally, Java supports the use of Apache’s HttpClient for Android. It provides an efficient, up-to-date, and feature-rich package which can be implemented on the client side with the most recent HTTP standards and recommendations [6]. The use of REST data has been a significantly positive decision because now, an external API is no longer required and the feature is available using Java’s import. This has not only reduced the size of the application by 2KB without the KSOAP2 API dependency, but also the complexity of the data fetcher class has reduced significantly by using Apache’s HttpClient package.

2.2.2 Image Loading APIs

When it comes to promoting restaurants and eateries, the adage “A picture is worth a thousand words” holds very true and relevant. Most apps classified under the Food & Restaurants category have lots of pictures. It is important to have a smart app which can make decisions on how and when to download these large quantities of image files. The app
must also be smart to download and display the best image size for the various and diverse mobile screen sizes. After researching thoroughly, the API best suited for smart, smooth, and efficient image loading would be the Android-Query (AQuery) API. AQuery is a lightweight library for doing asynchronous tasks and manipulating UI elements in Android [7]. It is licensed under Apache License 2.0 and is granted for redistribution as long as there is no code modification done directly to the API. I do not intend to make any code changes to the API but simply use it as is. AQuery supports easy asynchronous image loading from network, with automatic file and memory caching. In addition, it also handles optimal image size display for various screen sizes, provides the capability to use a fallback image if download speed is slow, and smooth image display and transitions. Needless to say, the AQuery API plays a huge part in the success of the release of the SDSU Dining mobile application.
CHAPTER 3

TECHNOLOGIES

Several technologies were used in the successful delivery of this project. Each of these is explained in detail in the following subsections.

3.1 ANDROID

A fundamental prerequisite of mobile application development is to select the mobile operating system. For this thesis, the target mobile OS is Android. Sections 3.1.1 and 3.1.2 provide an overview of the Android platform and its architecture.

3.1.1 Android Platform

The Android Platform is a software stack and is currently developed by Google. Android Inc. was founded by Andy Rubin in 2003 and it was primarily designed to support mobile devices such as phones and tablets. It is built on the open Linux Kernel and delivers a complete set of software for mobile devices: an operating system, middleware, and key mobile applications. Android is open source and the platform will continue to evolve as the developer community works together to build innovative mobile applications. In 2005 Google acquired Android Inc. to begin dwelling into mobile technology [8]. Android does not differentiate between the phone’s core applications and third-party applications. They can all be built to have access to a phone’s capabilities providing users with a broad spectrum of applications and services. With devices built on the Android Platform, users are able to fully tailor the phone to their interests. They can swap out the phone’s home screen, the style of the dialer, or any of other the applications. They can even instruct their phones to use their favorite photo application to handle the viewing of all photos [9].

3.1.2 Android Architecture

Android’s software stack is organized into several layers. As indicated in Figure 3.1, the layers are:
Figure 3.1. Android architectural layers.

- Linux Kernel
- System Libraries
- Application Framework
- Application

The Linux Kernel layer is the lowest layer of software in the Android Platform. This layer provides the core services that any Android computing device will rely on. Android’s Linux Kernel, just like any Linux kernel, provides generic operating system services. For example, it provides security permissions architecture so access to data and services can be restricted to processes with the appropriate authorizations. It supports memory and process management. It handles low level file and network I/O. It also allows device drivers to be plugged in so Android can communicate with a wide range of low level hardware components that are often coupled to computing devices such as memory, cameras, etc. Android’s Linux Kernel also contains Android-specific components. This includes power
management services, memory sharing features, inter-process communication, and many more.

The System Libraries layer includes various native libraries that are written in C and C++. These libraries handle core performance sensitive activities on a device, such as quickly rendering web pages, updating the display, high performance graphics, SQLite for managing in memory database, etc. The Android Runtime System supports writing and running Android applications. The two main components of the Android Runtime System are the Core Java libraries and the Dalvik Virtual Machine (VM). Android is written in the Java programming language and Android provides a number of reusable Java building blocks such as java.*, javax.*, org.*. The Dalvik VM is the software that executes the Android applications. It was designed for resource-constrained environments, which is typical of mobile devices.

The Application Framework layer contains reusable software that many mobile applications are likely to need. It keeps track of all the applications that are currently installed on a device. Various applications can now request services or obtain data from other applications. It also manages the windows comprising an application. The View System of the Application Framework contains many common user interface (UI) elements such as icons, text entry boxes, buttons, etc. This layer also manages non-compiled resources like strings, graphics, and layout files. Application Framework manages the app lifecycle and navigation stack, and provides location and moment information [10].

The last and topmost layer of the Android software stack is the Android Applications layer. Android ships with a standard set of applications such as the home screen, phone dialer, web browser, etc. These apps are not hardcoded into the system. A third party app can be substituted for any of these apps. An average user of the Android device would mostly interact with this layer [11].

3.2 AQuery API

AQuery is a light-weight library for doing asynchronous tasks and manipulating UI elements in Android. It is primarily used in the SDSU Dining app for downloading, caching, and loading images. AQuery provides various features such as AJAX Callbacks, Chaining, XML Parsing, etc. but the feature we are most interested in is the Image Loading feature.
AQuery Image Loading allows for cache control, fallback image, preloading, aspect ratio, list view display, and many more [7].

A simple example of loading an image to an ImageView from network, and caching the image to file and memory is as follows:

```java
aq.id(R.id.image1).image("http://www.vikispot.com/z/images/vikispot/android-w.png");
```

Another more relevant example of loading images to a ListView that is currently used in the SDSU Dining app is as follows:

```java
AQuery rootAQuery = new AQuery(this);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this, R.layout.activity_restaurant_list_row, entries){
    @Override
    public View getView(int position, View convertView, ViewGroup parent){
        if(convertView == null){
            convertView = getLayoutInflater().inflate(R.layout.activity_restaurant_list_row, null);
        }

        AQuery listRowAQuery = new AQuery(convertView);
        String url = entries.get(position);
        int id = getApplicationContext().getResources().getIdentifier(url, "drawable", getPackageName());
        Bitmap bm = listRowAQuery.getCachedImage(id);
        listRowAQuery.id(R.id.restaurantListViewImage).image(bm, AQuery.RATIO_PRESERVE);
        return convertView;
    }
};
rootAQuery.id(R.id.restaurantsListView).adapter(adapter);
```

When working with AQuery, we have to ensure that the views we want to operate can be found by the view or activity created with the AQuery object. In this case, I had to create an AQuery object with the item container view in order to render the list items.

### 3.3 Web Service Emulator

A web service is a method of transferring and communicating data between two devices over a network. At the time of writing this paper, the web service implementation is still not functional and available. SDSU Dining Services will be handling and implementing back-end database queries and secure data transfer. Due to security measures, I do not have authorization to view and handle sensitive back-end database information. Thus, the entire web service delivery will be managed by the SDSU Dining Services team.
In the meantime, a placeholder Web Service Emulator has been implemented for the delivery of the app. The web service emulator contains plain text “JSON” strings which were manually created from information available at the SDSU Dining Services website.\(^1\) These files currently reside on a student SDSU Rohan account.\(^2\) The static plain text files are accessed by the mobile device as a means of fetching data. Once the SDSU Dining Services web service is functional and available, the data will be dynamically fetched from the back-end database and displayed as JSON objects. This means the information displayed on the app will be accurate and up-to-date.

### 3.4 RESTFUL JSON DATA

JavaScript Object Notation (JSON) is a lightweight data-interchange format. It is easy for humans to read and write, and it is easy for machines to parse and generate [12]. JSON is a syntax for storing and exchanging data and is an easier to use alternative to XML [13]. JSON is built on two structures:

- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

These are universal data structures and virtually all modern programming languages support them in one form or another [12].

The Web Service Emulator uses JSON objects and arrays as a means of communicating data to the mobile device. The SDSU Dining app uses a HTTP GET call to access and retrieve these JSON objects with relevant information which gets displayed to the users. Below are two sample JSON objects currently used in the application:

- **Restaurant JSON array with its respective name/value pairs**

```json
{
    "restaurants": [
        {
            "restaurant_id": "100-1",
            "active": "true",
            "lastModified": "2014-09-21:13:00:00",
```  

\(^1\) [www.eatatsdsu.com](http://www.eatatsdsu.com)

\(^2\) [http://www-rohan.sdsu.edu/~jayaprak/sdsuDining](http://www-rohan.sdsu.edu/~jayaprak/sdsuDining)
"restaurant_name": "Aztec Market & Convenience Store",
"image": "aztecmarket",
"location_id": "100",
"location_name": "East Commons",
"phone": "(619) 594-7619",
"website": "http://www.eatatsdsu.com/Locations.aspx"
},

{  
  "restaurant_id": "100-2",
  "active": "true",
  "lastModified": "2014-09-21:13:00:00",
  "restaurant_name": "Vinnie's Pizza & Pasta Bar",
  "image": "vinnies",
  "location_id": "100",
  "location_name": "East Commons",
  "phone": "(619) 594-0413",
  "website": "http://www.eatatsdsu.com/Locations.aspx"
}
]

- **Hours JSON array with its respective name/value pairs**

```
{  
  "hours": [  
    {  
      "id": "100-1-Monday",
      "active": "true",
      "lastModified": "2014-09-21:13:00:00",
      "restaurant_id": "100-1",
      "day": "Monday",
      "open": "7:00 AM",
      "close": "10:00 PM"
    },  
    {  
      "id": "100-1-Tuesday",
      "active": "true",
      "lastModified": "2014-09-21:13:00:00",
      "restaurant_id": "100-1",
      "day": "Tuesday",
      "open": "7:00 AM",
      "close": "10:00 PM"
    }
  ]
}
```

### 3.5 Android SQLite Database

SQLite is an open source database. SQLite supports standard relational database features like SQL syntax, transactions, and prepared statements [14]. The SQLite library is a core part of the Android environment. Java applications and content providers access SQLite using the interface in the android.database.sqlite namespace. Applications use these database
management classes to manage its own private database [15, 16]. Android ships with the sqlite3 database tool therefore external APIs or administration of the database is not required in order to use the SQLite database.

The SDSU Dining app uses the Android SQLite database to store the JSON objects fetched from the web service into local storage. Additionally, various database queries have been written to retrieve the minimal desired information required to be displayed. An example of a query currently used by the app to retrieve restaurants at a particular location is as follows:

```java
public ArrayList<HashMap<String, String>> getRestaurantsAt(String locationName) {
    ArrayList<HashMap<String, String>> restaurants = new ArrayList<HashMap<String, String>>();
    String query = "SELECT " + RESTAURANT_ID + ", " + RESTAURANT_NAME + ", " + RESTAURANT_IMAGE + " FROM " + RESTAURANT_TABLE + " WHERE " + RESTAURANT_LOCATION_NAME + "=\"" + locationName + "\"; 
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor cursor = db.rawQuery(query, null);
    //Move to first row
    cursor.moveToFirst();
    while(cursor.getCount() > 0 && !cursor.isAfterLast()){
        HashMap<String, String> entry = new HashMap<String, String>();
        entry.put(RESTAURANT_ID, cursor.getString(0));
        entry.put(RESTAURANT_NAME, cursor.getString(1));
        entry.put(RESTAURANT_IMAGE, cursor.getString(2));
        restaurants.add(entry);
        cursor.moveToNext();
    }
    cursor.close();
    db.close();
    return restaurants;
}
```

### 3.6 Google Cloud Messaging for Android

Google Cloud Messaging for Android (GCM) is a service that allows you to send data from your server to your users’ Android-powered device. The GCM service handles all aspects of queuing of messages and delivery to the target Android application running on the target device. GCM is completely free and it is the key technology used to implement push notifications [17]. The three components that play a primary role in GCM are as follows:

- **Client App:** The GCM-enabled Android application that is running on the device. The application must use the GoogleCloudMessaging API provided through the Google Play Services software development kit (SDK).

- **3rd-party Application Server:** At the time of writing this paper, the application server and underlying database is still not functional and available. SDSU Dining Services
will be handling and implementing encrypted registration ID transfer and storage. Due to security measures, I do not have authorization to implement and access sensitive back-end infrastructure information. Thus, the 3rd-party application server delivery will be managed by the SDSU Dining Services team. In the meantime, a simple PHP application has been developed which is capable of saving registration IDs and sending messages.

- **GCM Connection Servers**: The Google-provided servers involved in taking messages from the 3rd-party application server and sending them to the device.

Prior to implementing the client app and the 3rd-party application server, a Google API project must be created which allows authorized application server connection, generates the server application API key, and provides the Android application project ID. With this information, the client app can register with the GCM servers and receive messages. Similarly, the 3rd-party application server can communicate with the GCM servers and send messages to the client applications.
CHAPTER 4

APPLICATION DESIGN

The second phase of the software development process is Design. Designing feasible, scalable, versatile, and user-friendly software is the key for a successful application. I believe research and design are the most important activities pertaining to the delivery of any software. Hence, as shown in Section 4.1, about 37% of the project delivery timeline was spent on research and design.

Most classes within the entire application are fairly short and straightforward. Various object-oriented best practices and design patterns were used throughout to accomplish simple, easy to read, and flexible code. The key idea which I always kept in mind was to ensure that each class encapsulated one idea. The moment I realized a class was getting lengthy I would pause to verify whether the class was staying consistent to one idea. If not, it was an indication that I should refactor, remodel, or redesign the class and its structure. During the refactoring process, I would try to list and understand the tasks that need to be accomplished and what the end goal was. Often by doing this, the draft thumbnail sketches provide key words which hints towards a particular design pattern which would help me accomplish the end goal. This has enabled me on countless occasions to write clean, reusable, and flexible code.

4.1 PROJECT TIMELINE

The first thing I did was create a project timeline. This helped me not only organize my thoughts on a checklist of what needs to be done prior to a successful app release, but also helped me create an outline and estimate the duration of the project. Figure 4.1 provides a timeline overview of the various key events and Table 4.1 provides a detailed reference.

4.2 STORYBOARD

The first task of the project was to create a storyboard for the application. Creating a storyboard can be useful because it outlines what the product will look like with the various
Figure 4.1. Draft project timeline.
Table 4.1. Detailed Reference of Figure 4.1

<table>
<thead>
<tr>
<th></th>
<th><strong>Date</strong></th>
<th><strong>Timeline Details</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Kick Off</strong></td>
<td>November 26, 2013</td>
<td>SDSU Dining app project kick off</td>
</tr>
<tr>
<td>A</td>
<td>November 27, 2013</td>
<td>Research and Design:</td>
</tr>
<tr>
<td></td>
<td>1. Android features and best practices</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. APIs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. Storyboard and flowchart</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>December 2, 2013</td>
<td>Development environment setup</td>
</tr>
<tr>
<td>C</td>
<td>March 2014</td>
<td>Deliver web service emulator and basic UI layout</td>
</tr>
<tr>
<td>D</td>
<td>March 7, 2014</td>
<td>First day coding the SDSU Dining app</td>
</tr>
<tr>
<td>E</td>
<td>April 2014</td>
<td>Deliver asynchronous parsers and database helper</td>
</tr>
<tr>
<td>Milestone I</td>
<td>April 1, 2014</td>
<td>Completion of Research and Design phase</td>
</tr>
<tr>
<td>Milestone II</td>
<td>April 11, 2014</td>
<td>Phase One demo – user experience. Attendees: SDSU Dining Service engineering and graphics design teams</td>
</tr>
<tr>
<td>F</td>
<td>May – Mid June 2014</td>
<td>Deliver <em>Restaurants, Filter Location</em>, and dynamic restaurants pager adapter</td>
</tr>
<tr>
<td>G</td>
<td>Mid June – August 2014</td>
<td>Deliver <em>Farmer’s Market, SDSU Catering, Sweet, Contact Us</em>, dynamic sections pager adapter, and <em>Coupons</em></td>
</tr>
<tr>
<td>Milestone III</td>
<td>August 28, 2014</td>
<td>Phase Two demo – mid-year progress. Attendees: SDSU Dining Services engineering, graphics design, and marketing teams</td>
</tr>
<tr>
<td>H</td>
<td>September 30, 2014</td>
<td>Deliver push notification feature</td>
</tr>
<tr>
<td>Milestone IV</td>
<td>October 17, 2014</td>
<td>Phase Three demo – product deliverable. Attendees: SDSU Dining Services engineering, graphics design, and project management teams. Code freeze with teams’ approvals</td>
</tr>
<tr>
<td>Milestone V</td>
<td>October 24, 2014</td>
<td>Thesis defense and Version 1.0 release ready for launch</td>
</tr>
</tbody>
</table>

app requirements established. The main focus of a mobile application storyboard is to sketch a rough draft of what the user interface of the app would look like. The storyboard and flowchart work hand-in-hand in the initial app design; and these were shown to the SDSU Dining Services teams for their feedback and approval. I had created a Microsoft PowerPoint presentation to show what this application would possibly look like. Over the course of this project, several of these sketches were modified; however, the storyboard established a foundation on a starting point for the development of the app. For example, in the Version
1.0 release, the *Home* view is delivered exactly as the sketch in Figure 4.2-A. However, the layout of *Farmer’s Market* has changed completely from what is depicted in Figure 4.2-B.

![Figure 4.2. Storyboard vs. Version 1.0 comparison. (A) Home view. Left: Storyboard sketch. Right: Version 1.0 screenshot. (B) Farmer’s Market view. Left: Storyboard sketch. Right: Version 1.0 screenshot.](image.png)

**4.3 APP FLOWCHART**

With any software, it must have a relevant and coherent flow. A flowchart is a type of diagram that represents an algorithm, workflow, or process. It outlines the sequential steps in a process, the relationship between the steps, the individual or group responsible for or involved in each step, and decision points with their associated process steps. Flowcharting also allows individuals or groups to understand complex processes, identify start and end points of a process, reveal problems or delays, understand optimal process steps, and leverage opportunities for improvements [18]. In our case, the individual involved in all the decision points is the user of the app and the associated process steps are handled by the app itself. Figure 4.3 illustrates the high-level diagram of the app’s process flow. I chose to create an app flowchart to provide an overview of the app’s process flow primarily for all the various stake-holders in the delivery of this app; namely, the SDSU Dining Services teams and the thesis committee members.
Figure 4.3. SDSU Dining app flowchart.
CHAPTER 5

IMPLEMENTATION

This section details the coding process and steps during the implementation of the project and its features.

5.1 DEVELOPMENT ENVIRONMENT

The first step was to setup the development environment in order to begin coding. The integrated development environment (IDE) used is Eclipse Juno Version 4.2.0. Eclipse is cross-platform supported and for this project was installed on a Windows 7 64-bit system. Alongside Eclipse, the Android Development Tools (ADT) plugin was used within the Eclipse environment. Because ADT is a plugin for Eclipse, developers get the functionality of a well-established IDE, along with Android-specific features that are bundled with ADT. It provides a suite of tools which are integrated with the Eclipse IDE. It offers developers many features that help develop Android applications quickly. ADT provides GUI access to many of the command link SDK tools as well as UI design tool for rapid prototyping, designing, and building of the application’s user interface [19]. The target SDK for this project is Jelly Bean Version 4.3 with API level 18. However, the minimum SDK required to run this project bug-free is Honeycomb Version 3.0 with API level 11.

5.2 APP LAUNCH

On app launch, there are several checklist items that the app must complete prior to displaying the Home screen. During an app launch event, the app loads the Splash screen and must verify whether network connectivity is available. If network is not available, the app displays a Dialog message alerting the user that there is no internet connection and the app failed to fetch data from the server (Figure 5.1-A). If there is no network available on the very first launch of the application, then the app will only display the six main categories, but all succeeding views will have no information to display. The app requires at least one successful attempt of fetching and parsing data in order to display something. For all future
If network connectivity failures, the app will continue to function as expected; however, the information displayed will be stale data from the local mobile device database.

If network connectivity is available, a progress activity circle, otherwise known as a busy-wait icon, is displayed as shown in Figure 5.1-B. The application asynchronously proceeds verify if it is already registered with the GCM servers. If not, it registers the device, saves the registration ID on the local device database, and sends the registration ID to the external push notification server. While verifying GCM registration, the app simultaneously proceeds to fetch and parse data provided by the web service emulator. Seven different threads, one for each database table, asynchronously fetch and parse web service data, and stores the data to its respective table in the local SQLite database.

SDSU_DiningParser is an abstract class which has an abstract method called getAsyncJsonStringParser() which is implemented by the respective concrete classes. SDSU_DiningParser also contains a protected inner class called AsyncWebServiceCall which extends Android’s AsyncTask. On instantiation of a concrete object, the constructor calls the parent’s AsyncWebServiceCall execute(url) method. The execute() method instantiates a DataFetcher object which fetches JSON data from the URL passed in the parameter. On
onPostExecute() of the AsyncWebServiceCall class, the object in turn calls the respective concrete object’s getAsyncJsonStringParser() method. This method is again asynchronous and executes in the background. Here, the JSON string returned by the AsyncWebServiceCall class is parsed by the respective table parser and is either inserted, updated, or deleted from the local SQLite database depending on the JSON object’s active state and lastModified timestamp. All parsers extend Java’s Observerable class which notifies the Splash activity observer upon its respective completion. Below is the code snippet of SDSUDiningParser:

```java
public abstract class SdsuDiningParser {
    protected class AsyncWebServiceCall extends AsyncTask<String, Void, Void> {
        ...
        @Override
        protected void onPreExecute() {}
        protected Void doInBackground(String... params) {
            DataFetcher df = new DataFetcher(params[0]);
            jsonString = df.fetch();
            ...
        }
        @Override
        protected void onProgressUpdate(Void... values) {}
        @Override
        protected void onPostExecute(Void result) {
            AsyncJsonStringParser asyncParse = getAsyncJsonStringParser();
            asyncParse.execute(jsonString);
        }
    }
    abstract class AsyncJsonStringParser extends AsyncTask<String, Void, Void> {
        protected abstract AsyncJsonStringParser getAsyncJsonStringParser();
    }
}
```

Below is the code snippet for the concrete RestaurantsParser class:

```java
public RestaurantsParser(String url, Context context, Observer observer) {
    this.context = context;
    this.addObserver(observer);
    ...
    AsyncWebServiceCall ws = new AsyncWebServiceCall();
    ws.execute(url);
}
```

```java
@Override
protected AsyncJsonStringParser getAsyncJsonStringParser() {
    return new AsyncJsonStringParser() {
        @Override
        protected Void doInBackground(String... params) {
            String jsonString = params[0];
            if(jsonString != null){
                try {
```
```java
JSONObject jsonObj = new JSONObject(jsonString);
restaurants = jsonObj.getJSONArray(RESTAURANTS_OBJECT_TAG);
SdsuDBHelper db = new SdsuDBHelper(context);
for(int i=0; i<restaurants.length(); i++){
    ...
    if(!active){
        db.removeFromRestaurantsTable(id);
    } else{
        db.addToRestaurantTable(id, name, image, locationId, locationName, phone, website, lastModified);
    }
}...
```

```java
@Override
protected void handleObservers() {
    ...
    notifyObservers();
    ...
}
```

It is an Android best practice to reduce heavy computations on the main UI thread. This prevents the app from having long lags or delays, and appear glitchy to the user. GCM registration and all the fetching and parsing tasks are done asynchronously in the background; hence, they are multithreaded. Keeping this feature as multithreaded not only improves efficiency and reduces process time, but also delegates the heavy computations to background tasks. After various timed tests, this entire process takes about 4 to 11 seconds to complete depending on the strength of the network connection and number of database table writes. Once the checklist items have completed successfully, the Home screen is displayed and the user can navigate and begin using the app. This view is a ListView of the six main categories. Figure 5.2 illustrates what the Version 1.0 release Home screen looks like.
5.3 Restaurants

The Restaurants category lists the wide selection of eateries currently serving on SDSU campus. As shown in Figure 5.3-A, the user has the option of either selecting a unique eatery to view the eatery information or the user can tap one any of the filter buttons to narrow down the search. Currently only the Filter Location feature is available in Version 1.0. Close To Me and Open Now will be future enhancements detailed in Chapter 7. If Filter Location is selected, a ListView of all the eatery locations will be displayed as shown in Figure 5.3-B. The user now has the option to view eateries at a specific location. If, for example, the East Commons location is selected, the app now displays only the eateries which are located at East Commons (Figure 5.3-C). When an eatery is selected, the RestaurantsPagerAdapter dynamically calculates the number of locations of the eatery. It creates the necessary number of ActionBar tabs and restaurant information fragments at runtime and based on the location selected, it auto-opens the respective location tab (Figure 5.4-A).

The restaurant information fragments display the restaurant status, daily hours, phone number, and website. Tapping the phone number automatically opens Android’s phone dialer.
Figure 5.3. Restaurants selection. (A) Restaurant listing. (B) Location filter listings. (C) Restaurant listings at East Commons location.

Figure 5.4. Restaurant timings status. (A) Restaurant with status x mins to Close. (B) Restaurant with status Open. (C) Restaurant with status Closed.
app or tapping the website automatically opens Android’s web browser app. One of the key features of the Version 1.0 release is displaying the accurate restaurant status. Figure 5.4-A, Figure 5.4-B, and Figure 5.4-C show the various restaurant status options. The DiningHoursCalculator class uses Java’s TimeZone class with the time zone setting of America/Los_Angeles. This handles the daylight savings time changes and provides the accurate local time in Pacific Standard Time (PST). Also, by using Java’s TimeZone class, the restaurant status displays the correct status irrespective of the user’s location. Based on the local PST time, the DiningHoursCalculator class determines whether the restaurant is open or closed on a particular day. It also calculates the time difference between open and close in order to set the closure countdown timer.

5.4 COUPONS

The Coupons category lists all the various coupons offered by the SDSU Dining Services. Coupons are ordered by the date of expiration with the most recent being on top. Expired coupons will not be shown to users. Currently the SDSU Dining app is not linked to any user account, hence there are no means established for tracking coupon usage and coupons may be reused multiple times until its expiration. This feature will be a future enhancement elaborated in Chapter 7. As shown in Figure 5.5-A, if there are no coupons available, a message is displayed to notify the user. Otherwise, a ListView of coupon images is displayed with various eatery promotional coupons (Figure 5.5-B). The user can simply open the Coupons screen and show the respective eatery cashier to get their discount. Not only is the user benefitting from using the coupons, but the SDSU Dining Services also benefits from promoting the diverse eateries as a marketing strategy.

5.5 FARMER’S MARKET, SDSU CATERING, SWEET, AND CONTACT US

Farmer’s Market, SDSU Catering, Sweet, and Contact Us categories follow a similar layout structure. As shown in Figure 5.6-A, Figure 5.6-B, Figure 5.6-C, and Figure 5.6-D, all four categories have three ActionBar tabs titled Contact, Hours, and About. The fragment layouts of each of these tabs are consistent across the four categories as well. The reuse of fragments/view/layouts is widely used throughout the entire SDSU Dining application. Not only does it create a consistent “look and feel” throughout the entire application, but it also
Figure 5.5. Coupons screen. (A) Coupons category with no coupons available. (B) Coupons category with two coupons from different eateries.

Figure 5.6. Reusable fragments. (A) Farmer’s Market layout. (B) SDSU Catering layout. (C) Sweet layout. (D) Contact Us layout.
promotes for view reuse which reduces the number of lines of code written and reduces computation time as well.

The SectionsPagerAdapter class handles the view creation of the four categories mentioned above. Based on the category selected, the SectionsPagerAdapter creates and displays the respective view. Below is the code snippet of the Contact Us view creation.

```java
@override
public Fragment getItem(int position) {
    if(lableString.equals(context.getString(R.string.farmersMarketString))){
        ...
    } else if(lableString.equals(context.getString(R.string.cateringString))){
        ...
    } else if(lableString.equals(context.getString(R.string.sweetString))){
        ...
    } else{
        dbList = getContactDBDetails();
        return getFragement(position);
    }
}
```

Depending on which tab is selected by the user, the SectionsPagerAdapter returns the appropriate fragment, as shown in Figure 5.7-A, Figure 5.7-B, and Figure 5.7-C. It handles fragment creation and reuse at runtime which allows for smooth transitions between the tabs. Below is the code snippet of the tab fragment creation.

```java
private Fragment getFragement(int position){
    if(position == 1){
        return getHoursFragment();
    }
    ...

    Bundle args = new Bundle();
    Fragment fragment;
    if(position == 0){
        ...
        fragment = new DisplayContactFragment();
        fragment.setArguments(args);
        return fragment;
    } else{
        ...
        fragment = new DisplayAboutFragment();
        fragment.setArguments(args);
        return fragment;
    }
}
```

As an example, the DisplayAboutFragment class is coded as follows:
public class DisplayAboutFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState){
        View rootView = inflater.inflate(R.layout.fragment_display_about,
        container, false);
        TextView aboutText = (TextView) rootView.findViewById(R.id.aboutText);
        aboutText.setText(getArguments().getString(getActivity().getResources().getString(R.string.about)));
        return rootView;
    }
}

5.6 PUSH NOTIFICATION

The push notification feature is an expedited component of the Version 1.0 release. Initially this feature was scheduled for Version 2.0 release; however after much deliberation, the SDSU Dining Services teams decided that carrying forward this feature sooner than later would be very beneficial for them and users. Push notification provides SDSU Dining Services the capability to promote and communicate various flash events and deals that would be occurring on a short notice. Flash events such as the Tiki BBQ cook-off which was held on August 27th, 2014 from 5-9pm at the Cuicacalli dining room for new students and parents during welcome week would be perfect candidates for the push notification feature. It
will be a quick and easy way to reach out to the vast majority of students and users of the app.

As mentioned in Section 5.2, the application automatically verifies and registers the app with the GCM servers and the external push notification server. When a Dining Services personnel sends a notification via the push notification server, the message is queued by the GCM servers and delivered to the respective targeted devices. The app’s `GCMBroadcastReceiver` receives the message and the `GCMIntentService` creates the notification. The notification first appears as an icon in the notification area (Figure 5.8-A) and the user is notified with sound, light, and/or vibration depending on the user’s settings. If the notification drawer is opened, a more detailed overview of the notification is provided (Figure 5.8-B) and if the user clicks on the notification, the SDSU Dining app is automatically launched and the notification message is displayed as shown in Figure 5.8-C.

![Figure 5.8. SDSU Dining push notification. (A) Notification icon displayed in the notification area. (B) Notification overview displayed in the notification drawer. (C) Notification details displayed within the SDSU Dining app.](image-url)
5.7 Application Demo Phases

The SDSU Dining application has been piloted by various individuals, both technical and not, using multiple devices. The app has been demoed on:

1. HTC Sensation running Android 4.0 Ice Cream Sandwich
2. Nexus 4 running Android 4.4 Kit Kat
3. Nexus 5 running Android 4.4 Kit Kat

The app was initially developed and tested on the HTC Sensation device. The app gracefully handles both portrait and landscape modes.

A total of three demos phases were presented to the SDSU Dining Services teams during the lifetime of this project. During Phase One demo, I received feedback on modifying the UI to provide a better user experience. Suggestions such as changing the heading text style to bold and text color to the SDSU red scheme, redesigning the Restaurants filter button, etc., were implemented. During Phase Two demo, I received feedback to expedite the push notification feature to be included in Version 1.0 release. They were thrilled and satisfied with the quick load time of the application, smooth view transitions, and especially the restaurants status feature. During the final Phase Three demo, they seemed satisfied and pleased with the app and no further modifications were made. This indicated the app is ready for launch and submission to the Google Play Store as soon as the SDSU Dining Services delivers the web service and 3rd-party push notification server implementations.
CHAPTER 6

CHALLENGES OVERCOME

Like with any other software product delivery, there were challenges faced with this project as well. Hurdles were encountered but were overcome throughout the lifetime of the thesis. I believe the biggest challenge was research. With the multitude of technologies and APIs available, trying to find the best, most efficient, and lightweight program suited for the tasks I was trying to accomplish was very time consuming and tedious. Once I finalized on using a particular software, the process of mastering the tool was also a challenge. There were numerous failed implementations; either the approach did not accomplish the end goal or the approach was inefficient. For example, creating a database helper class for each table within a single database proved to be an erroneous approach due to database version collisions within each helper class. Designing and redesigning, coding and refactoring became a normal practice throughout.

Another challenge faced was being able to write and implement efficient dynamic fragment creation and action bar layouts. It took over five different redesigns before the appropriate classes, pager adapters, fragments, and views were created to handle its designated tasks. For example, with previous approaches, issues such as the correct number of tabs were not getting created; there was noticeable delay between when the category button was tapped and when the tabs and fragments were displayed; and code was very hard to read therefore would make it difficult for future development.

A third major hurdle overcome was implementing the push notification feature. Incorporating push notifications was a particularly difficult challenge since it involved establishing successful interaction between three different applications where the central go-in-between component are the GCM servers – which developers cannot access to debug failures. For example, when a message sent from the 3rd-party application server is never received by the client app, it is sometimes unclear whether the interaction failed between the application server and GCM servers or between the GCM servers and client app. While implementing the push notification feature, three issues were encountered. After a few weeks
of researching and debugging, the causes were as follows: the application server communication was denied by the GCM servers, the client app devices did not have Google Play Services application installed, and the ExternalServerConnector failed to send registration IDs to the application server. Once the issues were fixed, push notification was working as expected. Like any other software product delivery, these challenges were overcome and as a result a functional application is available for use.
CHAPTER 7

VERSION 2.0 FUTURE ENHANCEMENTS

At the conclusion of this project, the SDSU Dining application is ready for Version 1.0 release. All the requirements established as deliverables by the SDSU Dining Services team have been completed to their satisfaction and approval. Without doubt this application opens a wide range of possibilities on what the SDSU Dining Services can do and incorporate as future enhancements. Several Version 2.0 release enhancements which we have established already are listed below:

1. Restaurants category Close To Me and Open Now features must be implemented. Close To Me will use the GPS location of the user to determine which eateries are closest. Open Now will filter a listing of eateries which are currently open. Closed eateries will not be displayed.

2. Depending on which eatery is selected by the user, provide an option to Take Me There which will auto link to the Google Maps application and provide a step-by-step navigation outline.

3. Marketing promotions for Farmer’s Market, SDSU Catering, and Sweet by incorporating images. Hooks for displaying images with asynchronous image downloads and smooth display and transitions are already in place.

4. Create an additional category for Meal Plan lookup. The user would enter his Red ID and the app will perform an asynchronous web service call. The result displayed to the user will be the daily remaining balance of his meal plan allowance.

5. User registration using accounts such as Facebook, Gmail, etc. to back-end server using OAuth. This will be used for managing coupon usage, linking restaurant listings and views according to user preferences, and many more.

6. Coupons category must incorporate Mark As Used feature which handles coupon usage associated to the user registered with the app.

7. Incorporate a tool for crash report collection.

8. Incorporate a user feedback and ratings tracker. Eventually these feedbacks will be associated to the various restaurants. This will also serve as metrics for restaurant evaluation by SDSU Dining Services.

9. Enhance and deliver a sleeker UI with user customizable elements.

10. Incorporate beacon tracking when the feature is available for Android devices.
CHAPTER 8

CONCLUSION

The SDSU Dining application is a simple to use mobile solution for SDSU Dining Services to reach out to the student, faculty, and staff of San Diego State University as well as the general public. Most people, especially the younger generation, in today’s day and age always have a mobile device handy. During a busy day, it is very convenient for the individual to pull out his/her device, open the SDSU Dining app, and decide on a spot to “grab a quick bite to eat”. This application provides this luxury and many other additional features. Users now have immediate access to restaurant and facility hours, contact information, and various SDSU Dining Services offerings. Users also have numerous coupons and discounts at their disposal. They can also get immediate notification of various flash events that occur on short notice. This thesis concludes with a functional Version 1.0 app release and serves as a foundation for a wide range of additional features and enhancements scheduled for future releases.
REFERENCES


