JOBS – ANDROID BASED JOB LISTING/SEARCH APPLICATION

A Project

Presented to the faculty of the Department of Computer Science

California State University, Sacramento

Submitted in partial satisfaction of
the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

by

Rachitha Prasad

SPRING
2013
JOBS – ANDROID BASED JOB LISTING/SEARCH APPLICATION

A Project

by

Rachitha Prasad

Approved by:

__________________________________, Committee Chair
Jinsong Ouyang, Ph.D.

__________________________________, Second Reader
Meiliu Lu, Ph.D.

____________________________
Date
Student: Rachitha Prasad

I certify that this student has met the requirements for format contained in the University format manual, and that this project is suitable for shelving in the Library and credit is to be awarded for the project.

__________________________, Graduate Coordinator
Behnam Arad, Ph.D.                     _________________

Date

Department of Computer Science
Modern world is seeing a number of Smart phones and Tablet devices, which are getting integrated into our daily lives. This drives the need to make available various kinds of data/solutions through these devices. Mobile applications provide a great way to share data and also provide customizable mobile experience. Android being the biggest market for smart devices is an ideal platform for building Applications.

“Jobs” is a simple Android based application for job search. It mainly helps students and professionals looking for jobs in different companies with varied skill set based on their field of study. In the current economic condition, finding jobs can be challenging. The application aims at consolidating jobs from different sites into a single location. It achieves this through RSS feeds interface exposed by the job sites. The users will also have the flexibility to add additional feed sources into the application to expand the horizon of job search. The user will also be able to apply for the jobs listed through the application.

Some of the features it provides are:

* Search – based on job area/skill, categorization on job type, company and location
* Notifications – Status Bar notifications when there are new jobs available
• Data Management - Data Refresh Interval, Data Retention period
• Search history and Statistics

_______________________, Committee Chair
Jinsong Ouyang, Ph.D.

_______________________
Date
ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Jinsong Ouyang for giving me an opportunity to work on this Android based mobile application. I thank him for his time and effort in guiding me to complete this project successfully. His advice and comments were instrumental to my work on the project.

I would like to thank Dr. Meiliu Lu for being my second reader. I would also like to thank Dr. Behnam Arad, the Acting Graduate Coordinator of the Computer Science Department and the staff of Computer Science Department for their support.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>vii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xi</td>
</tr>
<tr>
<td>Chapter</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Purpose</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Scope</td>
<td>2</td>
</tr>
<tr>
<td>2. FUNCTIONAL AND SYSTEM REQUIREMENTS</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Functional Requirements</td>
<td>3</td>
</tr>
<tr>
<td>2.1.1 Get Job Listing</td>
<td>3</td>
</tr>
<tr>
<td>2.1.2 Apply for Jobs</td>
<td>4</td>
</tr>
<tr>
<td>2.1.3 Status Bar Notification</td>
<td>4</td>
</tr>
<tr>
<td>2.1.4 Data Management</td>
<td>4</td>
</tr>
<tr>
<td>2.1.5 Search History</td>
<td>4</td>
</tr>
<tr>
<td>2.1.6 Statistics</td>
<td>5</td>
</tr>
<tr>
<td>2.1.7 Manual Data Sync</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Use Case Diagram</td>
<td>6</td>
</tr>
<tr>
<td>2.3 System Requirements</td>
<td>7</td>
</tr>
</tbody>
</table>
3. ANDROID FRAMEWORK AND APPLICATION DEVELOPMENT .................. 8

3.1 Android Framework .................................................................. 8
   3.1.1 Architecture of Android ................................................. 9
   3.1.2 Android Components .................................................... 10

3.2 Environment Setup and Getting the Required Tools .................... 19
   3.2.1 Android SDK ............................................................ 19
   3.2.2 Installing Eclipse IDE .................................................. 20
   3.2.3 Installing Android Development Tool plug-in for Eclipse ... 21
   3.2.4 Creating Android Virtual Devices (AVDs) ....................... 22

3.3 Creating and Running Android Application with Eclipse .............. 24
   3.3.1 Creating Android Application ....................................... 24
   3.3.2 Application files and folders ........................................ 25
   3.3.3 Building and Running Android Application ...................... 26

4. DESIGN OF JOBS APPLICATION ........................................... 27
   4.1 Design ............................................................................. 27
   4.2 Development Environment .............................................. 29

5. IMPLEMENTATION .................................................................... 31
   5.1 Getting the Job Feeds ....................................................... 31
      5.1.1 Entering the Search Filter ......................................... 31
      5.1.2 Getting the Jobs ....................................................... 32
      5.1.3 Displaying the jobs .................................................. 36
### LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1: Jobs Use Case Diagram</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2: Android Architecture</td>
<td>9</td>
</tr>
<tr>
<td>Figure 3: Activity Life Cycle</td>
<td>11</td>
</tr>
<tr>
<td>Figure 4: Fragment Life Cycle</td>
<td>15</td>
</tr>
<tr>
<td>Figure 5: Android SDK Manager</td>
<td>20</td>
</tr>
<tr>
<td>Figure 6: Installing ADT</td>
<td>21</td>
</tr>
<tr>
<td>Figure 7: Creating AVD</td>
<td>22</td>
</tr>
<tr>
<td>Figure 8: Android Emulator</td>
<td>23</td>
</tr>
<tr>
<td>Figure 9: Creating Android Application</td>
<td>24</td>
</tr>
<tr>
<td>Figure 10: Steps Involved in Building and Running an Android Application</td>
<td>26</td>
</tr>
<tr>
<td>Figure 11: High Level Design of Jobs Application</td>
<td>27</td>
</tr>
<tr>
<td>Figure 12: Search Filter Screen</td>
<td>32</td>
</tr>
<tr>
<td>Figure 13: All Jobs Display Screen</td>
<td>37</td>
</tr>
<tr>
<td>Figure 14: Jobs Display By Company Screen</td>
<td>38</td>
</tr>
<tr>
<td>Figure 15: Jobs Display By Location Screen</td>
<td>39</td>
</tr>
<tr>
<td>Figure 16: Jobs Display By Job Type Screen</td>
<td>40</td>
</tr>
<tr>
<td>Figure 17: Job Details Display Screen</td>
<td>41</td>
</tr>
<tr>
<td>Figure 18: Settings/Preferences Screen</td>
<td>44</td>
</tr>
<tr>
<td>Figure 19: Notification Setting Screen</td>
<td>46</td>
</tr>
</tbody>
</table>
Figure 20: Data Management Settings Screen .......................................................... 47
Figure 21: Refresh Rate Selection Dialog ................................................................. 48
Figure 22: Network Connection Selection Dialog .................................................... 52
Figure 23: Search History Display Screen .................................................................. 54
Figure 24: Statistics Display Screen ........................................................................... 55
Chapter 1

INTRODUCTION

1.1. Purpose

There are a number of job sites today, which post hundreds of different jobs every day. There are also job search applications in android from these individual job sites like indeed.com, simplyhired.com, monster etc. that enhance the user experience in viewing and applying for these jobs on a mobile phone. These applications list all the jobs that are posted based on certain user entered search filters. However, there is no application that provides a consolidated view of all these jobs posted on different job sites with the ability to view the new jobs getting into the market. This application “Jobs” aims at achieving that.

“Jobs” is an Android based mobile application, which integrates jobs from multiple job sites into one consolidated view. A number of job sites provide RSS feeds, which the users can subscribe to get the latest job updates. The application uses this technology of RSS feeds to pull the job feeds from the job sites. Once the feeds are fetched, they are stored locally in the database, which the users can then view. The users can view all the details related to a job and also apply to the job through the application.

The application provides the ability to specify search filters (job area/skill). Multiple filters can be specified at the same time. The application then pulls only the jobs that match the search filter.
The application also categorizes the jobs based on Job Type – Full Time, Part Time, Internship, Contract, and Temporary. It also provides the ability to view jobs by specific Company or Location.

The job feeds are pulled asynchronously and when new jobs are fetched, status bar notification is sent if enabled by the user. This helps the user to quickly know and look at the new jobs.

The application also provides Data Management features with which the user can specify how often the jobs will be fetched, what is the retention period of these jobs in the database, and on what network the jobs can be fetched – either on Wi-Fi or any network. The application also aims at providing useful statistics to the users like the Search history and number of jobs applied. This can be useful for the user to evaluate if the search filters are accurate for the jobs they are interested in or modify accordingly.

1.2. **Scope**

The application fetches the jobs based on RSS feed URL’s. Hence, it can only get jobs from the job sites that provide RSS feed subscription and RSS URL’s.

The data received from the feeds is limited and does not include the entire details of a job. The user can get to the full description provided on the job sites through the application. The users will also be able to apply to the jobs of interest through the application.

The application is developed using Eclipse IDE and for Android framework API level 15 and developed mainly for mobile form factor.
Chapter 2

FUNCTIONAL AND SYSTEM REQUIREMENTS

The main objective of the application is to provide a consolidated view of various jobs listings from different job sites.

2.1. Functional Requirements

This section lists all the functional requirements of the application.

2.1.1 Get Job Listing

Provide a consolidated list of the jobs from different job sites that match the user entered job area/skill search criteria. The user can specify multiple search filters. The search filters are stored as a user preference. The jobs are fetched from RSS feed URLs provided by the job sites.

- The application provides a view where the jobs are categorized based on Job Type – Full Time, Part Time, Internship, Contract and Temporary
- The application provides a view where jobs are grouped by Company and jobs for specific company can be viewed
- The application provides a view where jobs are grouped by Location and jobs specific to a location can be viewed
2.1.2 Apply for Jobs

The users after viewing the job listings can also apply for the jobs from within the application.

2.1.3 Status Bar Notification

The application can send status bar notifications when new jobs are available for the user to be viewed. The user will be able to enable or disable the notifications. Clicking on the notification should open the application with the new jobs listed.

2.1.4 Data Management

- Refresh rate: The user can specify how often the job feeds should be fetched. This data is stored as a user preference.
- Retention period (In days): The user can specify when the jobs stored locally on the mobile can be deleted. This data is stored as a user preference.
- Network Connection to download the jobs data: The user can specify on which network – “Wi-Fi” or “Any cellular network”, the data should be fetched. This data is stored as a user preference.

2.1.5 Search History

The user can view the previous search filters specified for getting the jobs. Every filter specified by the user is stored as a preference which can be viewed by the user by selecting the “Search History” menu.
2.1.6  Statistics

The user can view statistics on how many jobs he/she have applied to through the application. This shows the total number of jobs the user has applied to and the breakdown for each job type.

2.1.7  Manual Data Sync

The users will be able to fetch jobs from the job sites manually as needed by clicking on the Refresh button.
2.2. Use Case Diagram

Below is the use case diagram for the application, which shows user interaction with the application.

![Jobs Use Case Diagram](image)

**Figure 1: Jobs Use Case Diagram**
2.3. **System Requirements**

- Application developed as an Android mobile application using Android API level 15
- Requires Wi-Fi or cellular network and GPS for fetching the jobs
- Eclipse IDE with ADT plugin
- Android SDK installed
- Android mobile for testing the application
Chapter 3

ANDROID FRAMEWORK AND APPLICATION DEVELOPMENT

3.1. Android Framework

Android is one of the most popular mobile operating system today. It is a modified version of Linux and is an open source. This makes it a very customizable and appealing operating system. It also provides a very unified approach to applications, which means the developed applications will be runnable on a number of Android based devices.

Android supports a number of features, which makes application development easy:

- **Storage**: Provide SQLite database, lightweight relational database for storing data.
- **Messaging**: Provides SMS and MMS capabilities.
- **Web browser**
- **Connectivity**: provides support for Bluetooth, Wi-Fi, GS/EDGE, CDMA, LTE etc.
- **Multi-tasking and Multi-touch**
- **Media Support**: Supports different media formats like H.264, H.263, JPEG, MPEG-4, WAV etc.
- **Hardware support**: Camera, GPS, various sensors
- **Virtual machine**
- **Development environment with Emulator support**
3.1.1 Architecture of Android

The architecture primarily has four layers:

- Applications: which consists of all the built-in application on android
- Application Framework: which provides framework and support to build applications
- Libraries: support the key features of Android like database, media support
- Linux Kernel: this is the Android Kernel which has all the device driver implementation for the hardware supported

Figure 2: Android Architecture [2]
3.1.2 Android Components

This section describes some of the main features/components of the Android Framework used to develop the application.

3.1.2.1 Activity

Activities hold the user interface of an application. There can be any number of activities for an application. Since it represents the UI window, the users interact with the activities to provide and receive data. The activity stack is used to manage the activities.

The following diagram describes the lifecycle of an activity:

- onCreate(): this method gets called when activity is first created
- onStart(): this method gets called when the activity becomes visible
- onResume(): this method gets called when the activity starts interacting with the user
- onPause(): this method gets called when the activity is paused
- onStop(): this method gets called when the activity is not visible to the user
- onDestroy(): this method gets called when the activity is to be destroyed
- onRestart(): this method gets called when the activity is restarting after it was stopped
Figure 3: Activity Life Cycle [2]
An activity can be created by extending the “Activity” base class.

```java
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

The main layout for the UI will be specified in the `main.xml` file.

```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:id="@+id/job_details_fragment_container">
    <Button android:id="@+id/btn1"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </Button>
</LinearLayout>
```

### 3.1.2.2 Intent

Intent is an object that can be used to navigate and connect different activities. It can be used with `startActivity` to launch an Activity, `broadcastIntent` to send it to any interested `BroadcastReceiver` components, and `startService(Intent)` or `bindService(Intent, ServiceConnection, int)` to communicate with a background Service [2].

Some of the attributes of an Intent include:

- action: specifies the action to be performed
  - ACTION_VIEW: display some information to the user
- ACTION_DELETE: delete the data specified from its component
- ACTION_DIAL: dial the specified number
- ACTION_SEARCH: perform a search

- data: the data on which the action is to be performed
- category: specifies additional information with respect to the action to be performed
  - CATEGORY_DEFAULT
  - CATEGORY_LAUNCHER
  - CATEGORY_PREFERENCE
  - CATEGORY_HOME

- type: specifies the type of the intent data
- component: Specifies the component class to be used for the intent
- extras: this is used to provide additional information to the component. This is specified as a Bundle object

```xml
<intent-filter>
  <action android:name="com.project.jobs.MAIN" />
  <category android:name="com.project.jobs.LAUNCHER" />
</intent-filter>
```

### 3.1.2.3 Fragment

Fragments are like mini activities that are embedded in an activity. A single activity can have multiple fragments to build a multi-pane UI and fragments can be reused in multiple activities. A fragment has its own life cycle with its own input events that can be statically or dynamically added or removed when an activity is running.

Life cycle of a Fragment:
- `onCreate()`: called when creating the fragment
- `onCreateView()`: called when the fragment has to draw its UI
- `onPause()`: called when the user is leaving the fragment
- `onAttach()`: called when the fragment is associated with the activity
- `onActivityCreated()`: called when the `onCreate` method of the activity has been returned
- `onDestroyView()`: called when the view of the fragment has been removed
- `onDetach()`: called when the fragment is detached from the activity
Figure 4: Fragment Life Cycle [2]
There are some subclasses of Fragment which can be used as well – ListFragment, PreferenceFragment, DialogFragment.

A fragment can be added to an activity in two ways:

- Declaring the fragment in the layout file of the activity

```xml
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.project.jobs.JobFeedListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

- Adding programmatically using FragmentManager

```java
FragmentManager fragmentMgr = getFragmentManager();
FragmentTransaction fragmentTxn=fragmentMgr.beginTransaction();
fragmentTxn.add(R.id.fragment_container, fragment);
fragmentTxn.commit();
```

### 3.1.2.4 Services

Service is a component that can be used to perform asynchronous background tasks that do not require user interaction.

- Starting a service: A service can be start as a “Bound” or “Started” service. A service is “Started” by calling startService(). After the service is started, it can run the background indefinitely even after the component that started it is destroyed. A “Bound” service is started by calling a bindService(). The binding allows the service to interact with other components to send and receive any data.
Stopping a service: A service can be stopped by calling stopService() from another component or when the service itself calls stopSelf().

A service must be declared in the manifest file like activities as follows:

```xml
<application … >
  <service android:name="com.project.jobfinder.JobFeedReaderService"/>
</application>
```

### 3.1.2.5 Notifications

Android provides the notification service to display any information or message to the user outside of the application’s UI. Notifications can be in the form of status bar, flashing LEDs on the device, flashing the backlight, playing a sound or vibrating.

Notifications can be issued by creating a “Notification” object and then calling NotificationManager.notify().

Creating a notification object:

```java
NotificationManager nmgr = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
Notification notif = new Notification(R.drawable.ic_launcher, "New Job Feeds", System.currentTimeMillis());
PendingIntent ni = PendingIntent.getActivity(getApplicationContext(), 0, new Intent(getBaseContext(), DatabaseActivity.class), 0);
nmgr.notify(NOTIFY_FEED_ID, notif);
```

### 3.1.2.6 Data Storage

The following Android storage options are used in the application:

- **Shared Preference**: Key value pairs which store primitive data
- **SQLite Databases**: Private database which can store structured data
3.1.2.7 Location Based Service

- Geo coding: Android provides the “Geocoder” class to get the latitude and longitude of a location from the address.

```java
Geocoder geocoder = new Geocoder(getBaseContext());
Address address = geocoder.getFromLocation(streetAddress, 1).get(0);
Address.getLongitude();
Address.getLatitude();
```

- Reverse Geo coding: The “Geocoder” class can be used to obtain the address of a location given its latitude and longitude.

```java
Geocoder geocoder = new Geocoder(getBaseContext());
List<Address> address = geocoder.getFromLocation(latitude, longitude, 1);
```

3.1.2.8 Broadcast Receiver

Broadcast receiver can be used to receive any broadcast/intents sent within the application. These broadcasts can be synchronous or asynchronous. They can be registered dynamically with Context.registerReceiver() or statically by specifying the <receive> tag in the manifest file. The sendBroadcast() can be used to send broadcasts.

3.1.2.9 Android Emulator

The Android SDK includes an android emulator that can be used to emulate android devices with different configurations such as screen resolutions, LCD density, monitor size etc. The emulator has an option to create snapshots and emulate the existence of an SD card. It also provides the capability to emulate voice, SMS messages or make phone
calls. It can also be used to emulate GPS data and to transfer any files into and out from the DDMS perspective.

3.2. Environment Setup and Getting the Required Tools

This section describes how to setup the environment for developing Android application and get all the necessary tools.

3.2.1 Android SDK

This is the most important software that has all the libraries, tools, emulator, debugger and tutorials for Android development.

The SDK can be downloaded from http://developer.android.com/sdk/index.html. The SDK is zipped and needs to be unzipped into a folder.

Configuring the Android SDK Manager:
The SDK Manager can be used to download and install the Android SDK platform tools and other API’s. Once the needed version and API level are chosen, clicking on the Install button will install the packages.

3.2.2 Installing Eclipse IDE

The Eclipse IDE can be installed from [http://www.eclipse.org/downloads/](http://www.eclipse.org/downloads/). This application development was done with Eclipse Indigo.
3.2.3 Installing Android Development Tool plug-in for Eclipse

ADT plug-in supports the creation of Android applications. It has the emulator and tools for accessing real mobile devices. It also has a debugger to debug applications. It also facilitates running of the application on emulator or a real device.

![Install Android Development Tool plug-in](image)

**Figure 6: Installing ADT**

To install the ADT from Eclipse, go to Help->Install New Software. In the dialog that opens, enter the following URL: [https://dl-ssl.google.com/android/eclipse](https://dl-ssl.google.com/android/eclipse) and hit enter (above figure). Then a list of Developer Tools appears all of which can be selected and
installed. Then there will be an option to accept the license agreements, accepting that and clicking on Finish, installs the ADT.

3.2.4 Creating Android Virtual Devices (AVDs)

AVDs are emulator instances that model a real Android device. They are used for testing the developed application. Any number of AVDs can be created to model different real Android device configurations.

Figure 7: Creating AVD
AVDs can be created in Eclipse as follows: Window->Android Virtual Device Manager. In the dialog that opens, click ‘New’. This shows the AVD dialog (above figure) to specify the name of the device, select Target and other configurations. After entering all the data, clicking on ‘Create’, creates a new AVD. After the AVD is created, clicking on ‘Start’ starts the emulator.

Figure 8: Android Emulator
3.3. Creating and Running Android Application with Eclipse

3.3.1 Creating Android Application

In Eclipse do the following: File -> New -> Project. From the dialog that comes up, select Android -> Android Application Project. Enter the Application, Project and Package names. Enter the minimum and Target SDK version (figure below). Clicking finish creates a new application.

![New Android Application](image)

**New Android Application**

- **Application Name:** HelloWorld
- **Project Name:** HelloWorld
- **Package Name:** com.test.helloworld
- **Minimum Required SDK:** API 8: Android 2.2 (Froyo)
- **Target SDK:** API 15: Android 4.0.3 (IceCreamSandwich)
- **Compile With:** API 15: Android 4.0.3 (IceCreamSandwich)
- **Theme:** Holo Light with Dark Action Bar

The package name must be a unique identifier for your application. It is typically not shown to users, but it "must" stay the same for the lifetime of your application; it is how multiple versions of the same application are considered the "same app." This is typically the reverse domain name of your organization plus one or more application identifiers, and it must be a valid Java package name.

**Figure 9: Creating Android Application**
3.3.2 Application files and folders

An Android application in general has the following structure:

- **src**: contains the source files that are .java.

- **gen**: contains R.java file, which is compiler generated and has the references of all the resources used in the project.

- **References Libraries**: contains a list of all the libraries used.

- **assets**: contains all the assets used by the application such as database files, text files etc.

- **bin**: contains the files generated after the build is completed. It has the .apk file, which is the application binary that is installed on the device for the application to run.

- **doc**: contains any document files added.

- **res**: contains all the resource files used by the application. It can have sub-folders as follows:
  - **layout**: contains the xml files which defines the layout of the different UI screens
  - **drawable**: contains images used in the application
  - **menu**: contains the xml files which defines any menu used in the application
  - **AndroidManifest.xml**: this is the manifest file for the application. It contains all the activities in the application, any permissions needed and service defined.
3.3.3 Building and Running Android Application

When the application is compiled and packaged with Eclipse, it generates a .apk, which contains the application binary.

![Diagram of building and running Android application](image)

Figure 10: Steps Involved in Building and Running an Android Application [2]

After the .apk is available, the application needs to be signed which helps identify the author of the application and trust the application source. With Eclipse, the signing happens automatically when the application is compiled.

The application can be run on the Emulator or real android device. To run the application on a real device, the device should be connected to the system via a USB connection.

To run the application in Eclipse, right-click on the project and select Run As -> Android application. This will run the application on the Emulator or device depending on what is available. If both are running, the user can choose the device to run the application on.
Chapter 4
DESIGN OF JOBS APPLICATION

4.1. Design

At a high level, the application consists of the following modules:

- **Main UI:** This component is the main user interface, which the user uses to enter any data and view the data. When the application starts, the user can specify the job skill/area filters in the UI. Based on the search data entered, the application gets the jobs that match the user’s criteria and displays them as a list. The user can also view jobs categorized by job type, company or location.

---

**Figure 11: High Level Design of Jobs Application**
- **Job Feed Reader Service**: This is the background service that runs asynchronously and fetches the job feeds from the RSS URL’s. After the jobs are fetched, it stores them into the SQLite database. It also sends notifications about the new jobs to the user if enabled.

- **XML Parser**: The job feeds fetched are RSS feeds that are in XML format. These feeds are parsed to get specific job details using the ROME XML parser. It is a free XML Parser that integrates with Android.

- **Android Alarm Service and Receivers**: Alarms are registered to be triggered at regular intervals for fetching the jobs from the RSS URL’s and also to delete expired jobs. Alarm receivers receive these alarms. The application has two alarm receivers – one starts the background Feed Receiver Service to fetch jobs and the other starts the DBDataFeedManager Service, which deletes jobs from database that are very old.

- **Job Preference Module**: Maintains the user preferences as:
  - Search filters and Search history
  - Notifications enabled/disabled
  - Data Refresh Interval
  - Data Retention Period
  - Network to download data
  - Statistics – Number of jobs applied

- **SQLite Database**: Stores the data on all the jobs fetched from the RSS URL’s. The database has the following tables:
“feeds”: This is the primary table which holds all the information on the jobs fetched

CREATE TABLE feeds
(
category TEXT, job_type TEXT, company TEXT, fpub_date TEXT,
address_line TEXT, zipcode NUMERIC, city NUMERIC, state TEXT,
country_name TEXT, country_code TEXT, _id INTEGER PRIMARY KEY,
longitude NUMERIC, latitude NUMERIC, description TEXT, pub_date
TEXT, link TEXT, title TEXT
);

“feed_urls”: This table holds the URL’s from which the jobs are fetched

CREATE TABLE feed_urls (url_id NUMERIC, url TEXT);

“jobs_applied”: This table holds the statistics on the jobs applied by the user through the application

CREATE TABLE jobs_applied (applied_date NUMERIC, job_type TEXT);

4.2. Development Environment

The application is developed on Mac OS X and using Eclipse IDE. There is an Android Development Tools (ADT) plug-in for Eclipse that supports developing Android applications. The Google API level 15 is used with Android 4.0 operating System (Ice Cream Sandwich).
The application is tested using the Android Emulator and Android handset - Samsung Galaxy S2 running Ice Cream Sandwich.
Chapter 5

IMPLEMENTATION

This chapter describes the implementation details of the application. It explains the interaction and flow from one UI to the next. It also describes how the Android Framework features are used.

5.1. Getting the Job Feeds

This section describes how the job feeds are fetched by the application and how they are stored locally and displayed to the user. This is primary objective of the application.

5.1.1 Entering the Search Filter

The user has the option to specify the job area/skill they are interested in seeing the jobs from. Multiple such filters can be added by clicking on the “+” button. The search filters are then stored by the application as a Shared Preference object.

Clicking the “Go” button, starts the main functionality of the application of fetching the job feeds from the RSS URLs and storing it in the database and displaying to the user. This is described in more detail in the following sections.

Layout consists of following elements:

<EditText android:id="@+id/filterTxt" />
<Button android:id="@+id/clearBtn"/>
<Button android:id="@+id/addBtn"/>
<Button android:id="@+id/goBtn"/>
5.1.2 Getting the Jobs

5.1.2.1 Setting up the Database

SQLite Database provided by Android is used to store all the data on the jobs fetched from the RSS URLs. When the application is opened for the first time, there is no database. Hence, this needs to be setup.

The database tables are Pre-Created using SQLite Database Browser tool and the file is stored under “assets” folder of the application. When the application runs for the first time, the database is read from this pre-created file and created in the mobile phone in
the SQLite Database. This is accomplished as a file InputStream.read and
OutputStream.write operations.

```java
String destPath = "/data/data/" + getPackageName() + "/databases";
File f = new File(destPath);
if(!f.exists()) {
    f.mkdirs()
    f.createNewFile();
    copyJobsDB(getBaseContext().getAssets().open("jobsdb"), new
FileOutputStream(destPath + "/JobsDB"));
}

public void copyJobsDB(InputStream input, OutputStream output) throws
IOException {
    byte [] buffer = new byte[1024];
    int length;
    while((length = input.read(buffer)) > 0) {
        output.write(buffer, 0, length);
    }
    input.close();
    output.close();
}
```

This creates two database tables – “feeds” which stores all the details on the jobs
fetched and “feed_urls” which is pre loaded with all the RSS URLs from which the jobs
will be fetched. This step is implemented in the JobTabMenu class

5.1.2.2 Starting the background Service

Now that the database is setup with all the necessary tables, the next step is to fetch
the job feeds. The JobTabMenu class after setting up the database, creates the
JobFeedReaderService intent and starts the service as below:

```java
Intent servIntent = new Intent(this, JobFeedReaderService.class);
```
startService(servIntent);

It also registers a Broadcast Receiver which listens to broadcast from the background service to indicate if the jobs are available in the database. This is used to avoid fetching data when there is none in the database.

```
intentFilter = new IntentFilter();
intentFilter.addAction("FEED_FETCH_ACTION");
registerReceiver(intentReceiver, intentFilter);
```

Till the data is available the JobTabMenu displays a ProgressBar that shows the progress of the operation.

### 5.1.2.3 JobFeedReaderService – getting the job feeds

This class extends the Android “Service” class. It also has defines a private Background AsyncTask class which fetches the Job feeds as a background thread.

```
private class FeedReaderBg extends AsyncTask<HashMap<Integer, ArrayList<String>>, Void, Boolean> {
}
```

The Service does the following things:

- Reads the Shard Preference Object to check if the Network Connection preference matches the current active network the user is in. If it matches it proceeds with more processing

```
SharedPreferences sharedPrefs = PreferenceManager.getDefaultSharedPreferences(getBaseContext());
ConnectivityManager connMgr = (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo activeInfo = connMgr.getActiveNetworkInfo();
```
Reads the RSS URLs from the “feed_urls” table by calling a database method implemented in the JobsDBAdapter class

```java
Cursor feedURLS = dba.getJobFeedURLIDs();
```

It reads the search filters entered by the user from the Preference Object and embeds this data in the RSS URLs as part of the query. This is done to fetch only the jobs which are of interest to the user.

It then calls the SyndFeedInput.build(feedURL) method in the ROME parser framework to get the Feeds

```java
SyndFeed feed = new SyndFeedInput().build(new XmlReader(new URL(feedUrl)));
```

The job feeds are in the form of SyndFeed objects. All the job details are extracted from the feed object – title, description, company, published date, job type, job link, longitudinal and latitudinal coordinates

The service uses the Geocoder framework to get the location information – (city, state, country) from the longitude and latitude coordinates

```java
Geocoder geocoder = new Geocoder(getBaseContext());
List<Address> addrs = geocoder.getFromLocation(lat, lon, 1);
if((addrs != null) && (addrs.size() > 0)) {
    a = addrs.get(0);
    country = a.getCountryName();
    city = a.getLocality();
    state = a.getAdminArea();
    zipcode = a.getPostalCode();
}
```

It then inserts all the job details into the database by calling insertToDatabase() method defined in the JobsDBAdapter class
➢ It then sends a Broadcast to the JobTabMenu class if needed, to inform that jobs are available in the database and can be fetched and displayed

```
Intent broadcastIntent = new Intent();
broadcastIntent.setAction("FEED_FETCH_ACTION");
getBaseContext().sendBroadcast(broadcastIntent);
```

### 5.1.3 Displaying the jobs

The display of the jobs is handled by the JobTabMenu class which is a FragmentActivity. It adds four tabs to display the jobs as per different criteria – “All Jobs”, “By Job Type”, “By Company”, “By Location”. The JobsLoader class handles the loading of the jobs, which is also a FragmentActivity.

```
public class JobTabMenu1 extends FragmentActivity implements
ListFragmentItemClickListener {

...

  tabHost = (FragmentTabHost)findViewById(android.R.id.tabhost);
  tabHost.addTab(tabHost.newTabSpec("byjobtype").setIndicator("By JobType"),
    JobsLoaderJTFrag.class, b1);

  ...

  }
```
The JobsLoader class has a JobsLoaderList class which is a ListFragment. This class lists all the jobs in a ListView. When the JobsLoaderList class is started, depending on the “tab” selected, it calls the corresponding database method in the JobsDBAdapter class to fetch the jobs. The database read is performed by a background asynchronous task.

```java
public class JobsLoader extends FragmentActivity {
    public static class JobsLoaderList extends ListFragment {}
```
If the tab is “By Company” it also loads a Spinner component with all the Company names the jobs are from. The user can use this drop down, to select and view jobs from specific Company.

![Figure 14: Jobs Display By Company Screen](image)

```java
adapter = new ArrayAdapter(getActivity(), android.R.layout.simple_spinner_item, android.R.id.text1, companies.toArray());
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);
```
If the tab is “By Location” it also loads a Spinner component with all the Location – city, state names the jobs are from. The user can use this drop down, to select and view jobs from specific location.

![Jobs Display By Location Screen](image)

**Figure 15: Jobs Display By Location Screen**

If the tab is “By Job Type”, five more sub-tabs are created for different job types - Full Time, Part Time, Internship, Contract, Temporary. Clicking on theses tabs fetches the appropriate jobs for the job type it represents.
The jobs are listed in a ListView which uses a SimpleCursorAdapter as the data source. The information shown for each job is limited to – title, company, location and published date.

```
jobAdapter = new JobCursorAdapter(getActivity(), R.layout.jobs_info, curFeeds, columns, to, 0);
setListAdapter(jobAdapter);
```

When a list item is clicked, the JobTabMenu class calls the JobDetailActivity, which instantiates a Fragment which calls a database method to display more details on a particular job as below.

![Figure 16: Jobs Display By Job Type Screen](image-url)
Intent intent = new Intent(this, JobDetailActivity.class);
intent.putExtra("rowid", position);
intent.putExtra("tabindex", tabHost.getCurrentTab());
startActivity(intent);

When the “Apply” button is clicked, the WebViewActivity class is called which embeds a WebView. It then uses the data in the “link” field of the job to load the actual source website where the job is posted. The user can then apply to the job as from any job site opened in a browser.

Intent webintent = new Intent(getActivity().getBaseContext(), WebViewActivity.class);
webintent.putExtra("url", cursor.getString(6));
startActivity(webintent);

![Image of Job Details Display Screen]

Figure 17: Job Details Display Screen
5.2. User Preferences/Settings

The application provides a few preferences/settings that the user can set. These settings are provided as a Menu option and stored as Preferences.

Layout file contains the following hierarchical structure where each Preference is a Preference Screen:

```xml
<PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:title="Settings">
    <PreferenceScreen
        android:title="@string/title_notifications"
        android:icon="@android:drawable/ic_notification_overlay">
        <CheckBoxPreference android:key="key_statusbar_notif"
            android:DefaultValue="false" />
    </PreferenceScreen>
    <PreferenceScreen
        android:title="@string/data_preferences"
        android:icon="@drawable/data_mgmt">
        <ListPreference android:key="key_data_ref_rate"
            android:DefaultValue="0"
            android:DialogTitle="@string/dialog_title_data_ref_rate"
            android:Enabled="true"
            android:Selectable="true">
            <Entries android:entries="@array/entries_data_ref_rate_interval"
                android:entryValues="@array/values_data_ref_rate_interval" />
        </ListPreference>
        <EditTextPreference android:key="key_data_ret_period"
            android:DefaultValue="30"
            android:Selectable="true" android:Enabled="true" />
    </PreferenceScreen>
    <ListPreference android:key="key_network_conn"
        android:Entries="@array/entries_network_conn"
        android:EntryValues="@array/values_network_conn"
        android:DialogTitle="@string/dialog_title_network_conn" />
</PreferenceScreen>
```
The preferences are implemented by "PreferenceActivity" class, which also implements "OnSharedPreferenceChangeListener" to capture any changes in the preferences and perform suitable action.

```java
public class JobFeedPreferencesActivity extends PreferenceActivity implements OnSharedPreferenceChangeListener {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.job_feed_preferences);
        sharedPref = PreferenceManager.getDefaultSharedPreferences(getBaseContext());
        sharedPref.registerOnSharedPreferenceChangeListener(this);
    }
}
```

There are mainly two sets of Preferences:

- Notifications
- Data Management
5.2.1 Notifications

The user can choose to enable/disable “Status Bar Notifications”. When the notification is enabled, the application will send a status bar notifications when new jobs are pulled from the RSS URLs. This helps the user to be aware of new jobs without the need to manually check for new jobs every time.

When the “Status Bar” checkbox is set, the value is stored as a Shared Preference for the application – “KEY_PREF_STATUSBAR_NOTIF”. When the
JobFeedReaderService fetches new jobs, it checks for the value of this shared preference. If it is set, the Service uses the NOTIFICATION_SERVICE, to create the notification object and set the appropriate message to be shown. It then sends the notification to JobsHomeActivity.class, which is the first activity of the application.

```java
NotificationManager nmgr = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
Notification notif = new Notification(R.drawable.ic_launcher, "New Job Feeds", System.currentTimeMillis());
PendingIntent ni = PendingIntent.getActivity(getApplicationContext(), 0, new Intent(getBaseContext(), DatabaseActivity.class), 0);
nmgr.notify(NOTIFY_FEED_ID, notif);
```

When the user clicks on the notification, the application will be opened listing the new jobs.
When this preference is not enabled, user will not receive any notifications when new jobs are fetched.

5.2.2 Data Management

The Data Management Preferences provide control over managing the jobs data fetched and stored locally.
5.2.2.1 Refresh Rate

This preference value determines how often and at what intervals the job feeds will be fetched from the RSS URLs. The Values are from a fixed set – “never”, “An Hour”, “Half day”, “once a day”.

Figure 20: Data Management Settings Screen
If the value is set to “Never” for the first time, then the job feeds will not be fetched automatically. Instead the user will have to use the Manual Sync option to pull the job feeds manually each time.

When a refresh rate other than “Never” is set for the first time or changed to a new value, the following events take place:

- The Alarm Service is used to get the Alarm Manager class

  ```java
  AlarmManager mgr = (AlarmManager) ctx.getSystemService(Context.ALARM_SERVICE);
  ```

- Any existing alarms which is set is cancelled
boolean isAlarmSet = (PendingIntent.getBroadcast(ctx, 0, new Intent(ctx, JobFeedAlarmReceiver.class), PendingIntent.FLAG_NO_CREATE) != null);
if(isAlarmSet) {
    mgr.cancel(PendingIntent.getBroadcast(ctx, 0, new Intent(ctx, JobFeedAlarmReceiver.class), PendingIntent.FLAG_NO_CREATE));
}

- A new Repeating alarm is created which gets triggered after every refresh rate interval specified by the user

    mgr.setRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
    System.currentTimeMillis() + interval, interval, sender);

    - If the value is changed to “Never” from a different value, then any existing registered alarms are cancelled and there will be no alarms triggered

When the alarm is triggered, it is received by the JobFeedAlarmReceiver class, which is a Broadcast Receiver listening for this specific alarm. On receiving this alarm, it starts the background JobFeedReaderService, which in turn queries the RSS URLs for new jobs and fetches and stores them in the database.

public class JobFeedAlarmReceiver extends BroadcastReceiver{
    private static PowerManager.WakeLock wakeLock = null;
    private static final String LOCK_TAG = "com.project.jobfinder.FeedReader";
    public static synchronized void getWakeLock(Context ctx) {
        if(wakeLock == null) {
            PowerManager pmgr = (PowerManager)
            ctx.getSystemService(Context.POWER_SERVICE);
            wakeLock =
            pmgr.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK,
            LOCK_TAG);
            wakeLock.setReferenceCounted(true);
        }
        wakeLock.acquire();
    }
    public void onReceive(Context context, Intent intent) {
        getWakeLock(context);
5.2.2.2 Retention Period

This preference is used to specify for how long the job feeds fetched will be stored in the database, in other words when the jobs are deleted from the database. The value is in terms of “number of days”. Any job fetched before this interval will be deleted periodically.

When the user specifies this value, a repeating alarm is set which gets triggered once a day.

```
Intent i = new Intent(ctx, DBFeedDataMgrAlarmReceiver.class);
PendingIntent sender = PendingIntent.getBroadcast(ctx, 1, i,
PendingIntent.FLAG_CANCEL_CURRENT);
mgr.setRepeating(AlarmManager.ELAPSED_REALTIME_WAKEUP,
System.currentTimeMillis(), interval, sender);
```

DBFeedDataMgrReceiver receives this alarm, which is a Broadcast Receiver. The receiver in turn starts a background service – DBFeedDataMgrService that calls a database method to delete jobs older than the value specified by the user.

```
public void onReceive(Context context, Intent intent) {
    getWakeLock(context);
    Intent feedDelService = new Intent(context, DBFeedDataMgrService.class);
    context.startService(feedDelService);
}
```

```
public class DBFeedDataMgrService extends Service {
    JobsDBAdapter dba = JobsDBAdapter.getInstance(this);
    @Override
```
public int onStartCommand(Intent intent, int flags, int startId) {
    new FeedDataMgrBg().execute();
    return START_NOT_STICKY;
}

private class FeedDataMgrBg extends AsyncTask<Void, Void, Void> {

    @Override
    protected Void doInBackground(Void... params) {
        int count = dba.delJobFeeds();
        return null;
    }
}

5.2.2.3 Network Connection to download jobs

This preference is used to specify on what network the data should be downloaded from the RSS URLs. There two options supported: “Wi-Fi” and “Any Network”. When the value is set to “Wi-Fi”, job feeds are downloaded only when the user is in the Wi-Fi zone. When “Any Network” is specified, job feeds are always fetched when the background service is run.
This preference value is used in the JobFeedReaderService, when it is started to fetch the job feeds. Before fetching the jobs, the service gets the ConnectionManager provided by the CONNECTIVITY_SERVICE. The Connection Manager is used to obtain information on the current active network of the user. If the network of the user, matches the value specified in the Preference, then the service goes ahead with fetching the jobs from the RSS URLs, else service stops running.
5.3. **Search History and Statistics**

The application maintains the list of all the search filters entered by the user as a preference. When the user hits the “Search History” menu, this list is displayed.

When the “Search History” option is clicked, it starts the SearchHistoryActivity. The activity fetches the SharedPreferences object of the application to get all the previous search filters stored. It then displays all the search filters as a list using the ListView. An ArrayAdapter containing the preference data is set as the adapter to the ListView.

```java
public class SearchHistoryActivity extends ListActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        SharedPreferences settings = getSharedPreferences("JobsPrefsFile", 0);
        HashSet<String> searches = (HashSet<String>) settings.getStringSet("searchCategories", new HashSet<String>());
        String[] array = searches.toArray(new String[0]);

        ListView searchHistList = getListView();
        searchHistList.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, array));
    }
}
```
The only statistics information provided by the application is the “Number of Jobs Applied”. The user can specify the start date and the end date for the duration the statistics needs to be viewed. The application gives a count of the number of jobs applied to by the user through the application. It lists the total number of jobs applied, as well as the jobs applied for different job types.

Figure 23: Search History Display Screen
For any job when the user hits the “Apply” button in the “JobDetailsFrag” after looking at the job details, the Fragment inserts the “date applied” and “job type” data into the “jobs_applied” database table by calling the method insertJobsApplied(JobType) in JobsDBAdapter.

When the user clicks on “Info” menu option, the “jobs_applied” table is read to get the different statistics based on the chosen start and end dates. The DatePicker dialog is used to display the dates selection to the user.

```
dba.getJobsAppliedTotal(startDate, endDate);
```
public static class DatePickerFragment1 extends DialogFragment
    implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        return new DatePickerDialog(getActivity(), this, year, month, day);
    }
}

The Search History and Statistics help the user to fine tune their search filters to find more appropriate jobs. They can look at the previous filters and look at the number of jobs it has provided which are of interest to the user by comparing against the Number of job applied statistics. The user can then change the search filter accordingly.

5.4. Manifest file

The application, in order to use the Internet, network, receive device boot event and location services, must specify in the manifest file the below set of permissions.

<uses-permission android:name="android.permissionINTERNET" />
<uses-permission android:name="android.permissionWAKE_LOCK" />
<uses-permission android:name="android.permissionRECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permissionACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permissionACCESS_NETWORK_STATE"/>

It also specifies all the activities, services and receivers implemented by the application.
<activity
    android:name="com.project.jobfinder.JobsHomeActivity"
    android:label="@string/app_name"
    android:launchMode="singleTask">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>

<service
    android:name="com.project.jobfinder.JobFeedReaderService"
    android:process=":jobfeeds_background"/>

<receiver
    android:name="com.project.jobfinder.JobFeedAlarmReceiver"
    android:process=":jobfeeds_background"/>
Chapter 6

CONCLUSION AND FUTURE WORK

The application gets the jobs only from job sites that provide RSS feeds. Hence the data is limited. Many other job sites and company websites do not expose RSS feeds. Some of the job sites have started to provide public API’s that can be used to get the same job details. There is also a need to explore in other ways the jobs can be fetched from these sites.

The application is developed as a mobile application and tested with android mobile phone. There are many more different android devices available today in different form factors. The application may need to be enhanced to seamlessly work on all these devices.

The jobs fetched by the application from different websites are stored locally on the mobile phone running the application. All the job details are inserted into the database, which is stored on the device. As the amount of data increases, it may not be efficient to store all the data locally. As such the application can be enhanced as a Client/Server application. All the jobs fetched can be stored on the server. The server can then send push notifications to the client on the availability of new jobs. On receiving the notification, the client can pull the new jobs. Google provides “Google Cloud Messaging for Android”, a service that allows a server to send data to android-powered device. This technology can be explored here.
The application today provides search filters for job skill/area, categorization on job types, company and location. The future work may involve adding more search filters and customization to enhance user experience.
BIBLIOGRAPHY


[7] Customizing Android Fragment TabHost