MOBILE-PLATFORM-BASED REAL-TIME IN-MUSEUM ARTWORK RECOGNITION SYSTEM

A Project
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
Naga Viswanathan Malepati
Spring 2011
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the
Project of Naga Viswanathan Malepati:

Mobile-Platform-Based Real-Time In-Museum Artwork Recognition System

William Root, Chair
Department of Computer Science

Carl Eckberg
Department of Computer Science

Robert D. Grone
Department of Mathematics and Statistics

4/4/2011
Approval Date
Copyright © 2011
by
Naga Viswanathan Malepati
All Rights Reserved
ABSTRACT OF THE THESIS

Mobile-Platform-Based Real-Time In-Museum Artwork Recognition System
by
Naga Viswanathan Malepati
Master of Science in Computer Science
San Diego State University, 2011

In accordance with their educational mission, art museums and art galleries generally display their artworks accompanied by explanatory signage. Such signage must necessarily be limited in size in order not to detract from the visual experience provided by the artwork itself. Unfortunately this correspondingly limits the amount of educational information the signage can convey. The problem is even more acute with respect to international visitors to galleries and museums, since space limitations typically preclude displaying multiple copies of even minimal signage in multiple natural languages.

In this thesis project an application was developed to mitigate the above problem. The application enables a camera-equipped mobile device to act as a real-time multilingual guide for art museum and art gallery visitors. The mobile device’s camera captures machine readable codes (QR codes) mounted unobtrusively on or near each artwork, that identify the artwork and provide an index into a device-hosted multilingual multimedia database containing supplementary textual, audio and video information in the visitor’s language of choice. Since the application is hosted entirely upon the mobile device and requires no modification to the museum/gallery environment except the mounting of very small, passive, non-powered QR code tags on or near each artwork, the application provides the equivalent functionality of a complete location-based service while requiring no costly modification or enhancement of the museum’s or gallery’s existing infrastructure.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT ............................................................................................................................. iv</td>
</tr>
<tr>
<td>LIST OF FIGURES ................................................................................................................ vii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS ................................................................................................... viii</td>
</tr>
<tr>
<td>CHAPTER</td>
</tr>
<tr>
<td>1 INTRODUCTION .........................................................................................................1</td>
</tr>
<tr>
<td>2 COMPARISON OF ALTERNATIVE SOLUTIONS ...................................................3</td>
</tr>
<tr>
<td>2.1 WI-FI Based Localization..................................................................................3</td>
</tr>
<tr>
<td>2.1.1 Pros ...........................................................................................................4</td>
</tr>
<tr>
<td>2.1.2 Cons ..........................................................................................................4</td>
</tr>
<tr>
<td>2.2 RFID Based Identification .................................................................................4</td>
</tr>
<tr>
<td>2.2.1 Pros ...........................................................................................................4</td>
</tr>
<tr>
<td>2.2.2 Cons ..........................................................................................................4</td>
</tr>
<tr>
<td>2.3 Image Template Matching .................................................................................4</td>
</tr>
<tr>
<td>2.3.1 Pros ...........................................................................................................5</td>
</tr>
<tr>
<td>2.3.2 Cons ..........................................................................................................5</td>
</tr>
<tr>
<td>3 REQUIREMENTS SPECIFICATION ..............................................................................6</td>
</tr>
<tr>
<td>3.1 Use Cases ...........................................................................................................6</td>
</tr>
<tr>
<td>3.1.1 Language Selection ...................................................................................6</td>
</tr>
<tr>
<td>3.1.2 Information about the Artwork .................................................................7</td>
</tr>
<tr>
<td>3.2 User Interface .....................................................................................................7</td>
</tr>
<tr>
<td>3.2.1 Language Selection Screen .......................................................................7</td>
</tr>
<tr>
<td>3.2.2 Detail Screen .............................................................................................7</td>
</tr>
<tr>
<td>4 TECHNOLOGY ............................................................................................................9</td>
</tr>
<tr>
<td>4.1 Android ..............................................................................................................9</td>
</tr>
<tr>
<td>4.1.1 Activity .....................................................................................................9</td>
</tr>
<tr>
<td>4.1.2 View ..........................................................................................................10</td>
</tr>
<tr>
<td>4.1.3 Services ..................................................................................................10</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

PAGE

Figure 1.1. System architecture. ................................................................................................2
Figure 2.1. WI-FI Based indoor localization (a:fingerprinting phase, b:matching phase). ............................................................................................................................3
Figure 2.2. Architecture from SIFT based approach in Ref 2. ..................................................5
Figure 3.1. Use case diagram. ....................................................................................................6
Figure 3.2. Main screen. ............................................................................................................7
Figure 3.3. Details screen...........................................................................................................8
Figure 4.1. Sample QR code. ...................................................................................................10
Figure 5.1. Activity diagram. ...................................................................................................12
Figure 5.2. Class diagram. .......................................................................................................14
Figure 5.3. Launch sequence. ..................................................................................................16
Figure 5.4. Language selection sequence. ...............................................................................17
Figure 5.5. Scan artwork sequence. .........................................................................................17
Figure 6.1. Generating QR code. .............................................................................................19
ACKNOWLEDGEMENTS

I am indebted to a number of people who have contributed to the successful completion of my Master’s program. First, I would like to thank my supervisor William Root for his valuable suggestions and for guiding me through my thesis. My thanks also go to Professor Carl Eckberg and Professor Robert Grone for taking time to serve as members of my thesis committee.

Finally, I would like to express my profound gratitude to my parents and family, for their love and incredible support.
CHAPTER 1

INTRODUCTION

In accordance with their educational mission, museums and art galleries generally display their artworks accompanied by explanatory signage. Such signage must necessarily be limited in size in order not to detract from the visual experience of the art itself. Unfortunately this correspondingly limits the amount of educational information the signage can convey. The problem is even more acute with respect to foreign visitors, since space limitations typically preclude displaying multiple copies of even minimal signage in multiple languages. Museums and art galleries around the world increasingly acknowledge the need for devices and/or systems that can provide enhanced informational services to their visitors; with an increasingly global population of visitors, the need for such informational systems that can provide localized/multilingual information is even more essential.

In this thesis project an application was developed to mitigate the above problem. The application enables a camera-equipped mobile device to act as a real-time multilingual guide for museum and art gallery visitors. The mobile device’s camera captures machine readable codes (QR codes) mounted unobtrusively on or near each artwork, that identify the artwork and provide an index into a device-hosted multilingual multimedia database containing supplementary textual, audio and video information in the visitor’s language of choice. Since the application is hosted entirely upon the mobile device and requires no modification to the museum/gallery environment except the mounting of very small, passive, non-powered QR code tags on or near each artwork, the application provides the equivalent functionality of a complete location-based service while requiring no costly modification or enhancement of the museum’s or gallery’s existing infrastructure.

There has been some previous work in this area – using camera-enabled devices to capture images of displayed artworks and then using image processing algorithms such as the Scale Invariant Feature Transform (SIFT) to match the captured image to a database of reference images. However, results have been mixed, for a number of reasons: image distortion due to uneven illumination, lack of focus, difficult/variable viewing angles,
obscuring crowds, etc. Furthermore, it is not clear that such image matching algorithms would be effective in distinguishing among the very large numbers of visually relatively similar artworks typical of some museum/gallery contexts.

The method used in this thesis project requires the museum authority to catalog its artworks and to create a database in which a unique identifier is assigned to each artwork. A two-dimensional bar code (called a “QR code”) is generated that encodes the identifier for each artwork. Users capture the QR code with the camera on the mobile device, which in turn decodes the image and presents the user with contextually appropriate multimedia information from its onboard database based on the code. In this case artwork images are not matched to identify the current artwork of interest, as in the SIFT method; instead, the QR code for the current artwork of interest is merely captured and decoded; hence identification of the artwork of interest is accomplished much more quickly. The high-level architecture of this project’s system may therefore be summarized pictorially as shown in Figure 1.1.

![Figure 1.1. System architecture.](image-url)
CHAPTER 2

COMPARISON OF ALTERNATIVE SOLUTIONS

The following alternative identification paradigms were analyzed and assessed before arriving at the current approach for artwork identification.

2.1 WI-FI BASED LOCALIZATION

WI-FI based localization [1] depends on identifying the location of a mobile device by matching signal strength fingerprints. This process involves capturing the signal strength of multiple wireless access points (at least 3 or more) at known locations within the museum/gallery building, and creating a database of that information. This is similar to indoor localization using Bluetooth technology [2]. In the identification phase, signal strengths of various wireless access points are captured and that information is matched against the database to determine the current location of the device. Figure 2.1 illustrates one such system [3].

![Figure 2.1. WI-FI Based indoor localization (a:fingerprinting phase, b:matching phase).](image-url)
2.1.1 Pros

This solution scales well, since once a database of fingerprints is created they can be matched relatively quicker. In addition, the setup is unobtrusive such that three or more wi-fi access points that it requires can be hidden from the visitors’ view. Furthermore, no user interaction is required in order for the device/application to detect the artwork currently closest to the user.

2.1.2 Cons

Wi-Fi Signal strength like any other RF signals is affected by environment [4]. There could be more losses if there are more people and could vary based on the direction the user is facing as well. In additional the strength will also depend on the type of hardware and potentially restricts visitors using their own mobile devices.

2.2 RFID BASED IDENTIFICATION

This system requires a RFID emitter placed next to each artwork which encodes the identifier for the artwork [5]. Visitors can browser around the gallery with a RFID capable mobile device which will capture the RFID signal and provide the code to the application. This will in turn look up the contextual information based on it.

2.2.1 Pros

The system is scalable and requires RFID tags to be placed close to the artwork. It requires no user action to detect the artwork closest to the user.

2.2.2 Cons

Passive RFID tags have very small ranges and hence are un-usable. While active RFID tags do provide much higher ranges, it’s comparatively expensive to have active tags. Also, this would require museums to have special hardware that are RFID capable.

2.3 IMAGE TEMPLATE MATCHING

This solution is based on “Weighted Color Spectrum Correlation”, where database of artwork images is created [6, 7]. The images are processed and a color histogram of the images is created. At runtime, users capture part of the artwork via cameras in the mobile
devices and this is compared against the histogram data to identify the artwork. Figure 2.2 shows the architecture of one such prototype system [6].

Figure 2.2. Architecture from SIFT based approach in Ref 2.

2.3.1 Pros

This solution scales well users can capture whole or part of the artwork from a reasonable distance which will be matched against a database of images. This can potentially allow users use their camera enable devices to get information.

2.3.2 Cons

Indoor environment could be a potential hindrance because insufficient lighting and also depends on user to get a good quality image to be able to match well. Image matching could potentially take longer time because of complex image processing [8, 9] and hence the system might not be interactive.
CHAPTER 3

REQUIREMENTS SPECIFICATION

This chapter details the use-cases and user interface screens for the system.

3.1 USE CASES

Figure 3.1 illustrates the high level use-cases to be supported. The details of the use cases are given below.

3.1.1 Language Selection

The system should allow users to select a language of their choice from a given list of supported languages. Once a user picks a language, artwork information both textual and audio should be provided in that language.

Figure 3.1. Use case diagram.
3.1.2 **Information about the Artwork**

The information about the closest artwork should be presented to the user in form of text and audio. The user is expected to capture the visual code via the device to get information about it.

3.2 **USER INTERFACE**

Application provide interface for users to make language selection and view artifact information as illustrated below.

3.2.1 **Language Selection Screen**

The main screen of the application provides users the option to select a language of their choice as illustrated in Figure 3.2.

![Figure 3.2. Main screen.](image)

3.2.2 **Detail Screen**

The Detail screen provides information about the artwork in the language of user’s choice and plays the audio file related to the artwork as well as illustrated in Figure 3.3.
Figure 3.3. Details screen.
CHAPTER 4

TECHNOLOGY

The mobile platform solution for artwork recognition is based on the Android software stack and a two dimensional bar code standard called QR Code.

4.1 ANDROID

Android [10] is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language. Android offers developers the ability to build extremely rich and innovative applications by utilizing the Application Framework and libraries provided by the runtime. The application architecture is designed to simplify the reuse of components; any application can publish its capabilities and any other application may then make use of those capabilities (subject to security constraints enforced by the framework). This same mechanism allows components to be replaced by the user. Some of key concepts are defined below

4.1.1 Activity

An activity is an application component that provides screen that the users can interact with. Each application typically consists of one “main” activity which is launched when an application starts. An application can have one or more activities to perform different actions. Activity is given a window to draw its user interface. An activity is created by creating a subclass of android.app.Activity class.
4.1.2 View

The user interface is built using Views and ViewGroups. Android provides a set of predefined widgets that are subclasses of View. ViewGroup serves as the base class for layout related classes. Activity UI is built by nesting view and viewgroups as needed. See Appendix A for XML based views defined for the application.

4.1.3 Services

A service is an application component that can perform long running operations in the background. Services do not provide an user interface and generally can bound by other application components to interact with. These are generally used to perform I/O, play music in the background etc.

4.2 QR Codes

QR Codes are two dimensional bar codes that consists of black modules arranged in a square pattern on a white background. The two dimensional code was first devised by Denso-Wave in 1994. QR is short-form for ‘Quick Response,’ appropriately suggestive of the fact that the code can be decoded at very high speeds. Standard and open source libraries are available to generate and decode QR codes. A sample QR code is illustrated in Figure 4.1.

![Sample QR code](qrcode.png)

4.2.1 Capacity

QR Code has the following capacity

- Numeric Data: Max 7089 Chars
- Alphanumeric Data: Max 4296 Chars
- Binary (8 bits) Data: Max 1817 Chars
4.2.2 ZXing

There are quite a few algorithms [11] that recognize QR codes. ZXing [12] (pronounced "zebra crossing") is an open-source, multi-format 1D/2D barcode image processing library implemented in Java. ZXing utilizes the camera built in mobile devices to capture images of the codes and processes the image to decode the codes. ZXing library supports QR code standards and provides library/application for android devices and also provides a web based code generator.
CHAPTER 5
DESIGN

The artwork recognition system provides contextual information about the artworks to users in the language of their choice. The activity diagram below in Figure 5.1 illustrates the high level functions and state transitions supported by the system.

![Activity diagram](image)

Figure 5.1. Activity diagram.

As illustrated in Figure 1.1, the system consists of a core application component, a data store component and an image scanner component. The following sections describe them in detail.
5.1 DATA STORE

For ease of managing data, the information about the artworks will be stored in the file-system on a removable storage disk. The structure of the data on the file system will be as below

- Data (top level folder)
  - Language (* Top level language folder)
    - Flag.jpg (Icon for each language)
    - Artwork_id (* Folder with artwork Id)
      - info.txt
      - {audio}.mp3

5.2 CORE APPLICATION CLASSES

Figure 5.2 illustrates the core application classes and their relationship to some of core Android classes, which are illustrated in grey color. See Appendix B for Java sources.

5.2.1 BaseActivity Class

BaseActivity class that extends Activity and encapsulates common shared functionality related to launching the scanner application to capture barcode. And once the code is captured, provides extending classes to define the behavior. The extending classes are required to implement the following abstract method

\[
\text{abstract void scanComplete();}
\]

This class provides static helper methods to set and get the data directory from the file system as well.

5.2.2 Main Class

This class extends the BaseActivity class and is the main Activity that is launched by android platform when the application is launched. The class uses the view generated by android platform off of main.xml. The view contains a list view that displays the list of supported languages for users to choose from. The list of supported languages is provided by the FlagListAdapter class which is associated to the list view during the onCreate phase of the activity.
Figure 5.2. Class diagram.

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    initialize();
    languageList=(ListView)findViewById(R.id.lang_list);
    adapter=new FlagListAdapter(this);
    languageList.setAdapter(adapter);
    languageList.setTextFilterEnabled(true);
    languageList.setOnItemClickListener(new LanguageItemClickListener(this));
}
```

### 5.2.3 LanguageItemClickListener Class

This class implements the OnItemClickListener to handle itemClick event on the language selection list view. When a language selection is made, it is stored as a global constant and triggers the scanner application.
5.2.4 FlagListAdapter Class

This class extends the BaseAdapter to provide the following to the list view:

1. Custom View for Item that has an icon and text.
2. Data for the list

The view for the item is provided by overriding the getView method. We use a XML based view to generate a view for the item and set the icon and text appropriately.

```java
public View getView(int position, View convertView, ViewGroup parent) {
    View vi=convertView;
    ViewHolder holder;
    if(convertView==null){
        vi = inflater.inflate(R.layout.list_item, null);
        holder=new ViewHolder();
        holder.text=(TextView)vi.findViewById(R.id.item_text);
        holder.image=(ImageView)vi.findViewById(R.id.item_image);
        vi.setTag(holder);
    }else
        holder=(ViewHolder)vi.getTag();
    holder.text.setText(data[position]);
    holder.image.setTag(data[position]);
    imageLoader.displayImage(data[position], activity, holder.image);
    return vi;
}
```

The constructor initializes the list of languages as shown below:

```java
data = folder.list(new FilenameFilter(){
    public boolean accept(File arg0, String arg1) {
        File flagFile = new File(arg0, arg1 + "/flag.jpg");
        return flagFile.exists();
    }});
```

5.2.5 ImageLoader Class

This class encapsulates the aspects of reading the image from the file system and decoding it into a bitmap as required by Android. Additionally, it caches the generated bitmaps for subsequent usage.

5.3 SEQUENCE DIAGRAMS

The main application flows are described in the sequence diagrams.
5.3.1 Application START Sequence

Android platform instantiates and initializes MainActivity class when the application is launched. MainActivity in turn creates instances of ImageAdapter & LanguageItemClickListener and assigns them to the language selection list view as illustrated in Figure 5.3.

![Figure 5.3. Launch sequence.](image)

5.3.2 Selecting Language Sequence

Figure 5.4 illustrates the various api invocations that happen when the user makes a language selection from main screen. The Android platform fires the event which is handled by the application and launches the QR code scanner utility. Upon successful identification of QR Code, the details of the artifact are displayed in the selected language.

5.3.3 Scanning QR Code

Figure 5.5 shows the sequence of actions that happen when users click on the ‘Scan’ button from the details screen in the application.
Figure 5.4. Language selection sequence.

Figure 5.5. Scan artwork sequence.
CHAPTER 6

DEPLOYMENT AND USAGE

6.1 CREATING DATABASE

The data associated with each of the artworks should be copied into an external storage card supported by the device in the structure described in the Data Model section above. Once the data is copied, the storage card must be inserted into the mobile device at all times during the application’s use.

6.2 DEPLOYING

Build the application and generate the distributable (apk) files. The compiled distributable should be deployed along with the zxing apk to the device.

6.3 GENERATING QR CODES

A QR code for each artwork must be generated using a QR code generator [13], and permanently mounted on or near the artwork. QR codes can be generated online from the link provided in the references. A screen shot of the QR Code generator [13] provided by ZXing is shown in Figure 6.1.
Figure 6.1. Generating QR code.
CHAPTER 7

CONCLUSIONS

We evaluated different alternative artwork identification paradigms for “Mobile-Platform-Based Real-Time In-Museum Artwork Recognition” and selected a paradigm based on an optical-machine-readable-standard. The system provides contextual information with little interaction from the user. The solution developed is scalable to support multiple languages without any changes other than to the database provided to the system.

The system presented here utilizes a local onboard database for storing supplemental multimedia artwork information. This requires the museum authorities to manage storage cards for the mobile platforms, which might not be compatible with mobile devices that museum/gallery visitors themselves might already be carrying. This application could in the future be converted to a client/server model so that supplementary information about the various artworks could be transferred wirelessly on demand from a central server, eliminating the need for local onboard datqastore and allowing visitors to use their own personal mobile devices to interface with the system.
REFERENCES


APPENDIX A

XML VIEW CONFIGS
XML view definition for main screen that provides language selection options for the user.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Top level layout container -->
<LinearLayout android:id="@+id/linearLayout1"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Table layout to display 2 rows one for title and other for language selection list -->
    <TableLayout android:layout_width="fill_parent"
        android:id="@+id/tableLayout1" android:layout_height="fill_parent">
        <!-- Container for title -->
        <TableRow android:id="@+id/tableRow1" android:layout_width="fill_parent"
            android:layout_height="wrap_content">
            <LinearLayout android:layout_width="fill_parent"
                android:id="@+id/linearLayout2" android:layout_height="wrap_content"
                android:orientation="vertical">
                <TextView android:text="Museum Guide"
                    android:id="@+id/autoCompleteTextView1" android:layout_height="wrap_content"
                    android:layout_width="fill_parent" android:layout_weight="1" android:layout_gravity="center"></TextView>
            </LinearLayout>
        </TableRow>
        <!-- Container for language selection list -->
        <TableRow android:id="@+id/tableRow2" android:layout_width="fill_parent"
            android:layout_height="wrap_content">
            <LinearLayout android:id="@+id/linearLayout3" android:layout_width="fill_parent"
                android:layout_height="wrap_content">
                <ListView android:id="@+id/lang_list" android:layout_width="fill_parent"
                    android:layout_height="wrap_content" android:layout_weight="1"></ListView>
            </LinearLayout>
        </TableRow>
    </TableLayout>
</LinearLayout>
```
INFO.XML

XML view definition for details screen.

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Top level layout container -->
<TableLayout android:id="@+id/tableLayout1"
android:layout_width="fill_parent" android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android">
  <!-- Container for title and scan button -->
  <TableRow android:id="@+id/tableRow1"
android:layout_width="fill_parent" android:layout_height="wrap_content">
    <LinearLayout android:id="@+id/relativeLayout1"
android:layout_width="fill_parent" android:layout_height="wrap_content">
      <TextView android:layout_width="wrap_content"
android:id="@+id/titleTextView1" android:text="AutoCompleteTextView"
android:layout_height="wrap_content"></TextView>
      <Button android:layout_width="wrap_content"
android:id="@+id/scanButton" android:text="Scan"
android:layout_height="wrap_content"></Button>
    </LinearLayout>
  </TableRow>

  <!-- Container for description text -->
  <TableRow android:id="@+id/tableRow2"
android:layout_width="fill_parent" android:layout_height="wrap_content">
    <TextView android:layout_width="fill_parent"
android:id="@+id/descriptionTextView1" android:text=""
android:layout_height="wrap_content" android:layout_alignParentLeft="true"></TextView>
  </TableRow>
</TableLayout>
```
LIST_ITEM.XML

Custom XML view for list item in the language selection list.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">
    <ImageView
        android:id="@+id/item_image"
        android:layout_width="98dip"
        android:layout_height="72dip"
        android:src="@drawable/stub"
        android:scaleType="centerCrop"/>
    <TextView
        android:id="@+id/item_text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_gravity="left|center_vertical"
        android:textSize="20dip"
        android:layout_marginLeft="10dip"/>
</LinearLayout>
```
APPENDIX B

JAVA SOURCE FILES
BASEACTIVITY.JAVA

package com.naga.artwork;
import java.io.File;
import android.app.Activity;
import android.content.Intent;

/**
 * BaseActivity class that extends Activity. This class provides 
 * common shared functionality related to 
 * launching the scanner application to capture barcode. And once 
 * the code is captured, provides extending 
 * classes to define the behavior. 
 */

public abstract class BaseActivity extends Activity {

    //Static global constants
    private static File DATA_FOLDER = null;
    public static String SELECTED_LANGUAGE = "English";
    public static String SELECTED_ARTWORK = "1";
    private static String BASE_DIRECTORY = null;

    /*
     * (non-Javadoc)
     * @see android.app.Activity#onActivityResult(int, int, 
     * android.content.Intent)
     * Method called back by android platform once the calling 
     * activity returns a result
     */
    public void onActivityResult(int requestCode, int resultCode, Intent intent) {
        if (requestCode == 0) {
            if (resultCode == RESULT_OK) {
                String contents = intent.getStringExtra("SCAN_RESULT");
                String format = intent.getStringExtra("SCAN_RESULT_FORMAT");
                // Handle successful scan
                SELECTED_ARTWORK = contents;
                scanComplete();
            } else if (resultCode == RESULT_CANCELED) {
                // Handle cancel
            }
        }
    }

    /*
Abstract void method to be implemented by child classes to define behavior on successful scan

abstract void scanComplete();

Static helper to launch the scanner application that captures the image and decodes it.

public static void launchScanner(Activity activity) {
    Intent intent = new Intent("com.google.zxing.client.android.SCAN");
    intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
    activity.startActivityForResult(intent, 0);
}

Static helper to get the BaseDirectory where the data is stored.

public static File getBaseDirectory() {
    if (DATA_FOLDER == null) {
        if (BASE_DIRECTORY == null)
            DATA_FOLDER = android.os.Environment.getExternalStorageDirectory();
        else
            DATA_FOLDER = new File(android.os.Environment.getExternalStorageDirectory(),
                                  BASE_DIRECTORY);
    }
    return DATA_FOLDER;
}

public static void setBaseDirectory(String dir) {
    DATA_FOLDER = null;
    BASE_DIRECTORY = dir;
}

public static String getBaseDirectoryName() {
    return BASE_DIRECTORY;
}

public static File getArtworkFolder() {
    return new File(getBaseDirectory(), "/" + SELECTED_LANGUAGE + "/" + SELECTED_ARTWORK);
}
public class Main extends BaseActivity {
    /** Called when the activity is first created. */
    ListView languageList;
    FlagListAdapter adapter;

    /**
     * (non-Javadoc)
     * @see android.app.Activity#onCreate(android.os.Bundle)
     * Overrides the onCreate method to set view and lookup languageList view.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Setting the view of the activity to main.xml
        setContentView(R.layout.main);
        initialize();

        languageList=(ListView)findViewById(R.id.lang_list);
        //Creating and setting new FlagListAdapter to provide items for list view
        adapter=new FlagListAdapter(this);
        languageList.setAdapter(adapter);
        languageList.setTextFilterEnabled(true);
        //Setting on click listener on list view
        languageList.setOnItemClickListener(new LanguageItemClickListener(this));
    }

    /*
     * Initializing the language and baseFolder from prefs
     */
    private void initialize() {
    }
}
SharedPreferences prefs = this.getSharedPreferences(MODE_WORLD_READABLE);
    BaseActivity.SELECTED_LANGUAGE = prefs.getString("language", "English");
    BaseActivity.setBaseDirectory(prefs.getString("baseFolder", "data"));
}

/**
  * (non-Javadoc)
  * @see android.app.Activity#onStop()
  * Overriding the onStop method to persist preferences on exit.
  */
protected void onStop(){
    super.onStop();
    SharedPreferences settings =
    getPreferences(MODE_WORLD_READABLE);
    SharedPreferences.Editor editor = settings.edit();
    if (BaseActivity.SELECTED_LANGUAGE != null)
        editor.putString("language", SELECTED_LANGUAGE);
    if (BaseActivity.getBaseDirectoryName() != null)
        editor.putString("baseFolder",
        BaseActivity.getBaseDirectoryName());
    else
        editor.remove("baseFolder");
    // Commit the edits!
    editor.commit();
}

/**
 * abstract method implementation. Launches Details activity on
 * successful scanning a QR code
 */
@Override
void scanComplete() {
    try {
        Thread.sleep(100);
    }
    catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    Intent myIntent = new Intent(this, DetailActivity.class);
    startActivity(myIntent);  
}

/*
 * ItemClickListener class that handles language list item
 * selection
 */
private class LanguageItemClickListener implements OnItemClickListener {
    private Activity parentActivity;
    LanguageItemClickListener(Activity parent) {
        this.parentActivity = parent;
    }

    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        // When clicked, show a toast with the TextView text
        if (view instanceof LinearLayout) {
            view = view.findViewById(R.id.item_text);
        }
        if (view instanceof TextView) {
            BaseActivity.SELECTED_LANGUAGE = ((TextView)view).getText().toString();
            BaseActivity.launchScanner(parentActivity);
        }
    }
}
package com.naga.artwork;

import java.io.File;
import java.io.FileInputStream;
import java.io.FilenameFilter;
import java.io.IOException;
import android.app.Activity;
import android.content.Intent;
import android.media.AsyncPlayer;
import android.media.AudioManager;
import android.net.Uri;
import android.os.Bundle;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

/*
* Activity for Details screen
*/
public class DetailActivity extends BaseActivity {

    private AsyncPlayer player;
    private TextView titleTxt = null;
    private TextView descriptionTxt = null;

    /*
     * (non-Javadoc)
     * @see android.app.Activity#onCreate(android.os.Bundle)
     * Initializes the audio player and sets onClickhandler for scan button to launch scanner
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.info);
        player = new AsyncPlayer("artwork player");
        titleTxt = (TextView)findViewById(R.id.titleTextView1);
        descriptionTxt = (TextView)findViewById(R.id.descriptionTextView1);
        //Call to initialize information for current selection
        this.initializeArtworkInfo();
        final Activity top = this;
Button button = (Button) findViewById(R.id.scanButton);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Perform action on click
        BaseActivity.launchScanner(top);
    }
});

private void initializeArtworkInfo() {
    // Looks up fully qualified path for selected artwork
    File dir = getArtworkFolder();
    // Searching for audio files in it
    File audio[] = dir.listFiles(new FilenameFilter() {
        public boolean accept(File dir, String filename) {
            return filename.toLowerCase().endsWith(".mp3");
        }
    });
    if (audio != null && audio.length > 0) {
        player.play(getApplicationContext(),
                Uri.fromFile(audio[0]), false, AudioManager.STREAM_MUSIC);
    }
    // Looking up info.txt
    File info = new File(dir, "info.txt");
    if (info.exists()) {
        FileInputStream fis = null;
        try {
            fis = new FileInputStream(info);
            StringBuffer title = new StringBuffer();
            StringBuffer description = new StringBuffer();
            StringBuffer ref = title;
            int chr;
            while ((chr = fis.read()) != -1) {
                if (chr != '\n') {
                    ref.append((char)chr);
                } else {
                    ref = description;
                }
            }
            titleTxt.setText(title.toString());
            descriptionTxt.setText(description.toString());
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            if (fis != null) {
                fis.close();
            }
        }
    }
}
try {
    fis.close();
}

catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

}
package com.naga.customlist;

import java.io.File;
import java.io.FilenameFilter;
import com.naga.artwork.BaseActivity;
import com.naga.artwork.R;
import android.app.Activity;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.BaseAdapter;
import android.widget.ImageView;
import android.widget.TextView;

/**
 * Custom adapter that provides a custom view for list items and data
 * for the list.
 */
public class FlagListAdapter extends BaseAdapter {

    private Activity activity;
    private String[] data;
    private static LayoutInflater inflater=null;
    public ImageLoader imageLoader;

    public FlagListAdapter(Activity a) {
        activity = a;
        File folder = BaseActivity.getBaseDirectory();
        data = folder.list(new FilenameFilter(){
            public boolean accept(File arg0, String arg1) {
                File flagFile = new File(arg0, arg1 + "/flag.jpg");
                return flagFile.exists();
            }
        });
        inflater = (LayoutInflater)activity.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        imageLoader=new ImageLoader(activity.getApplicationContext());
    }

    public int getCount() {
        return data.length;
    }

    public Object getItem(int position) {
        return data[position];
    }

    public View getView(int position, View convertView, ViewGroup parent) {
        View v = convertView;
        if (v == null) {
            v = inflater.inflate(R.layout.flag_list, null);
        }
        ImageView imageView = (ImageView)v.findViewById(R.id.flag_image);
        TextView textView = (TextView)v.findViewById(R.id.flag_text);
        imageView.setImageResource(data[position] + "/flag.jpg");
        textView.setText(data[position]);
        return v;
    }
}

FLAGLISTADAPTER.JAVA
return position;
}

public long getItemId(int position) {
    return position;
}

public static class ViewHolder{
    public TextView text;
    public ImageView image;
}

/*@ (non-Javadoc)
@see android.widget.Adapter#getView(int, android.view.View, android.view.ViewGroup)
* Creates a custom view for list item and sets the image and text for the item
*/
public View getView(int position, View convertView, ViewGroup parent) {
    View vi=convertView;
    ViewHolder holder;
    if(convertView==null){
        vi = inflater.inflate(R.layout.list_item, null);
        holder=new ViewHolder();
        holder.text=(TextView)vi.findViewById(R.id.item_text);
        holder.image=(ImageView)vi.findViewById(R.id.item_image);
        vi.setTag(holder);
    }else
    holder=(ViewHolder)vi.getTag();
    holder.text.setText(data[position]);
    holder.image.setTag(data[position]);
    imageLoader.displayImage(data[position], activity, holder.image);
    return vi;
}
package com.naga.customlist;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.HashMap;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.widget.ImageView;

import com.naga.artwork.BaseActivity;
import com.naga.artwork.R;

/**
 * Bitmap cache for images. This class encapsulates loading of bitmap from filesystem
 * and provides cache of loaded bitmaps.
 */
public class ImageLoader {

    //the simplest in-memory cache implementation. This should be replaced with something like SoftReference or BitmapOptions.inPurgeable(since 1.6)
    private HashMap<String, Bitmap> cache=new HashMap<String, Bitmap>();

    private File dataFolder;

    public ImageLoader(Context context){
        //Make the background thread low priority. This way it will not affect the UI performance
        dataFolder= BaseActivity.getBaseDirectory();
    }

    final int stub_id=R.drawable.stub;

    /**
     * Api to push an image into an imageView
     */
    public void displayImage(String folder, Activity activity, ImageView imageView) {
    }
String url = folder + "/flag.jpg";
imageView.setImageBitmap(getBitmap(url));

/*
 * Looks up the cache for given file. If not found loads it from
 * file system.
 */
private Bitmap getBitmap(String filename)
{
    //from SD data folder
    File f = new File(dataFolder, filename);
    Bitmap b;
    if(!cache.containsKey(filename)) {
        b = decodeFile(f);
        cache.put(filename, b);
    } else {
        b = cache.get(filename);
    }
    return b;
}

//decodes image and scales it to reduce memory consumption
private Bitmap decodeFile(File f){
    try {
        //decode image size
        BitmapFactory.Options o = new BitmapFactory.Options();
        Bitmap bitmap = BitmapFactory.decodeStream(new
        FileInputStream(f), null, o);
        return bitmap;
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } return null;
}

public void clearCache() {
    //clear memory cache
    cache.clear();

    //clear SD cache
    File[] files = dataFolder.listFiles();
    for(File f: files)
        f.delete();
}