Ahmedabad City App

Rushabh Picha

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd

Part of the Computer and Systems Architecture Commons, and the Other Computer Engineering Commons

Recommended Citation


This Project is brought to you for free and open access by the Office of Graduate Studies at CSUSB ScholarWorks. It has been accepted for inclusion in Electronic Theses, Projects, and Dissertations by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.
AHMEDABAD CITY APP

A Project
Presented to the
Faculty of
California State University,
San Bernardino

by
Rushabh Picha
December 2021

Approved by:
Dr. Yan Zhang, Advisor, Computer Science and Engineering
Dr. Jennifer Jin, Committee Member
Dr. Amir Ghasemkhani, Committee Member
ABSTRACT

The Ahmedabad City App is a city guide app that provides information on the city's accessible resources. The project's goal is to provide a concrete, one-stop platform for finding information on all of Ahmedabad's accessible resources. The main goal is to simplify the railway's schedule and make it easier for customers to get from one point to another swiftly and safely. Emergency connections such as the blood bank, fire department, police station, and hospitals will be included in the app. Restaurants and picnic areas are also included in the rejuvenation process.

The proposed system uses SQLite as the database and uses Android SDK and XML to build the frontend of the app. It exploits the MVC Design Pattern as an architectural pattern to foster separation of concerns in the app.

The Ahmedabad City App will foster a sense of belonging and connection among the city's residents, bridging the divide.

*Keywords: Android, SQLite, Design Pattern, MVC, XML, City App*
ACKNOWLEDGEMENTS

I'd want to express my gratitude to Dr. Yan Zhang, my project adviser, and committee members Dr. Jennifer Jin and Dr. Amir Ghasemkhani for believing in me and agreeing to join the committee. I am grateful for the trust they have placed in me while working on this project. I'd also like to express my gratitude to all the University's professors for assisting me in getting to this point in my academic career.

I'm also grateful to my coworkers, manager, and mentor at Audible Inc., where I was able to complete my internship and learn more about Android and designing modern applications.

I owe a debt of gratitude to my sister and parents for paving the way for me to pursue a career in computer science.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
<tr>
<td>CHAPTER ONE: INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>Background</td>
<td>1</td>
</tr>
<tr>
<td>Purpose</td>
<td>1</td>
</tr>
<tr>
<td>CHAPTER TWO: SYSTEM ANALYSIS</td>
<td>2</td>
</tr>
<tr>
<td>Proposed System</td>
<td>2</td>
</tr>
<tr>
<td>CHAPTER THREE: TECHNOLOGIES</td>
<td>3</td>
</tr>
<tr>
<td>Android SDK (Software Development Kit) and Android Studio</td>
<td>3</td>
</tr>
<tr>
<td>SQLite Database</td>
<td>4</td>
</tr>
<tr>
<td>XML</td>
<td>5</td>
</tr>
<tr>
<td>Declare UI Elements in the XML</td>
<td>5</td>
</tr>
<tr>
<td>Android Studio Layout Editor</td>
<td>6</td>
</tr>
<tr>
<td>CHAPTER FOUR: USER INTERFACE</td>
<td>8</td>
</tr>
<tr>
<td>ListView inside a Card View</td>
<td>8</td>
</tr>
<tr>
<td>Recycler View</td>
<td>10</td>
</tr>
<tr>
<td>Card Layout</td>
<td>11</td>
</tr>
<tr>
<td>View Holder</td>
<td>11</td>
</tr>
<tr>
<td>Data Class</td>
<td>11</td>
</tr>
<tr>
<td>CHAPTER FIVE: SYSTEM DESIGN</td>
<td>15</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Design Pattern</td>
<td>15</td>
</tr>
<tr>
<td>The Model</td>
<td>16</td>
</tr>
<tr>
<td>The View</td>
<td>18</td>
</tr>
<tr>
<td>The Controller</td>
<td>18</td>
</tr>
<tr>
<td>CHAPTER SIX: EXPLORING THE DATABASE</td>
<td>20</td>
</tr>
<tr>
<td>Database Creation</td>
<td>20</td>
</tr>
<tr>
<td>Database Insertion</td>
<td>21</td>
</tr>
<tr>
<td>Database Fetching</td>
<td>22</td>
</tr>
<tr>
<td>CHAPTER SEVEN: WEB VIEW IN ANDROID</td>
<td>24</td>
</tr>
<tr>
<td>CHAPTER EIGHT: TESTING</td>
<td>27</td>
</tr>
<tr>
<td>Android Linting</td>
<td>27</td>
</tr>
<tr>
<td>CHAPTER NINE: FUTURE ADVANCEMENTS</td>
<td>28</td>
</tr>
<tr>
<td>APPENDIX: CODE OF CRUCIAL PART</td>
<td>29</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>38</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Android Studio downloader page</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Declare UI elements in XML</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Android Studio layout editor</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>Card view with list view code snippet</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>Card view with list view demo example</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>Class Diagram (Railways Section)</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>Demo of recycler view holder</td>
<td>13</td>
</tr>
<tr>
<td>8</td>
<td>Screenshot of the UI</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>Example of MVC</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Sample code snippet of the Data Class</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>MVC Use Case diagram</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>Snapshot of creating the database in SQLite</td>
<td>21</td>
</tr>
<tr>
<td>13</td>
<td>Snapshot of inserting data into the SQLite database</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>Snapshot of retrieving the data from the SQLite database</td>
<td>23</td>
</tr>
<tr>
<td>15</td>
<td>Web view in Android</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>Adding JavaScript to WebView</td>
<td>26</td>
</tr>
</tbody>
</table>
CHAPTER ONE

INTRODUCTION

Background

Ahmedabad is a city located in the state of Gujarat, India. It is the capital of the state and has a population of 5.5 million. The city has a wide network of variety of transport means, picnic places and educational institutions. People use these resources daily and the amount of these resources is increasing rapidly.

Purpose

The purpose of this project is to create a single source of information that people can use to access various resources in the city. People need a quick access to resources like railways, auto fare, emergency contacts, picnic places, schools. Since Android is the most used Mobile Operating System in India, it would be a great way to build an application that utilizes this Operating System to breach the maximum audience. My idea is to build a transparent and reliable system which people can use daily hassle free. This system will help increase a sense of oneness and unity among the people in the city.
CHAPTER TWO
SYSTEM ANALYSIS

Proposed System

The Android application will help people to obtain information about a resource in the city. The city does not have a concrete timetable for railways and any other resources. The main aim of the application is to have a proper system for railways timetable. Apart from the railway timetable, the application incorporates other information like autorickshaw, traffic rules, news, emergency contacts, police station, picnic spots, movies and foodies. This will create a common platform for people to come and get to know more about the city.
CHAPTER THREE

TECHNOLOGIES

Android SDK (Software Development Kit) and Android Studio

The Android SDK tools used alongside with the Android Studio gives us the flexibility to build the application and the User Interface through our own convenience. To download the Android Studio, go to the website developer android studio and download it for the MAC OS users and Windows based on your Operating System [3].

Figure 1 shows a glimpse of how the homepage for the website looks like. All the Android SDK packages are included in the above tool. Install the software at the appropriate path and relaunch the studio.
SQLite Database

SQLite is an opensource SQL database that stores data to a text file on a device. Android comes in with built-in SQLite database implementation [6]. The ‘ite’ in the SQLite stands for lightweight since it stores data in the form of a text file rather than storing it in cloud or a server. This database supports all the features of a relational database (MYSQL, PostgreSQL) and hence querying is done by general SQL queries. We don’t need to create special connection to the JDBC (Java Database Connectivity) or ODBC (Open Database Connectivity), rather we can use inbuilt SQL classes to perform our operations. We will look in detail...
about the SQLite database in the later section.

XML

XML stands for Extensible Markup Language. In Android we can define the layout in two ways:

Declare UI Elements in the XML

This application uses declaring the UI elements in the XML to generate and beautify the views. This type of UI development Android provides a straightforward XML vocabulary which includes the views and the fragments that correspond to the View classes and subclasses, such as those for widgets and layouts. This designing is done by manually writing out attributes for a particular class. As shown in Figure 2, we have used the Image View incorporated into the Linear Layout showcasing the images that we have used on the main page of the app.
Android Studio Layout Editor

The other way to design the UI of the app is using the drag and drop interface. Android Studio provides an in-built ecosystem which adds the code for a particular view whenever a user drags and drops a view into the editor console.
Figure 3 shows the design editor for the drag and drop functionality.

**Figure 3: Android Studio layout editor**
As discussed above we use XML to design the frontend of our application. Let’s see how we utilize some key XML elements in our app to have a better design:

**ListView inside a Card View**

The ListView inside a Card View is a modern way of developing layouts in the Android, making the UI look appealing. Card View is a view that displays views on top of each other. It is applicable to Android SDK version 21 and above. I have used this tool in the emergency, foodies, movie, railways section of the app.

ListView is generally incorporated in the Card View since there must be a view that the Card View needs to show in the form of a card. As shown in the figure below. Figure 4 shows a sample demo of how the Card View along with the ListView looks like.
Figure 4: Card view with list view code snippet
Figure 5 shows the output of the above code and how we have used the card view with the list view to make our UI look better.

![Contacts of Blood Bank](image)

**Figure 5: Card view with list view demo example**

Recycler View

The application extensively uses the Recycler Views to populate the data in the railways section. Recycler View makes it very simple to present huge amounts of data quickly. You just provide data and define how each item should look, and the Recycler View library builds the elements as needed [3]. It was
made to allow the creation of any lists using xml layouts as an item that can be
tweaked further. This functionality helps extensively when there are tons of
objects in the Recycler View. The enhancement is made possible by recycling
views that are no longer visible to the user.

There are three components to any Recycler View:

**Card Layout**

An XML layout is used for the card layout. CardView is a view the allows
many views to be displayed on top of each other. CardView's main purpose is to
assist the UI design have a richer feel and look. CardView's best feature is that it
extends Framelayout and can be displayed on all Android systems.

**View Holder**

The View Holder is the major difference between the listview and
recyclerview. It is recommended to use the View Holder design pattern but is not
compulsory. However, it is mandatory using RecyclerView.ViewHolder class [2].
The View Holder is a Java class that carries a reference to the card layout views
that must be dynamically updated during program execution by a list of data
collected from electronic databases or contributed in some other manner.

**Data Class**

The Data Class is a type of information. It is a custom Java class that serves
as a structure for storing data for each RecyclerView item.

Figure 6 shows the working of the Railways section using a RecyclerView Adapter. In the Railways section, when the user enters the source and the destination stop, the Railways Adapter gets triggered. Thereafter, the Railways data class (POJO class) sends the data to the Railway List Adapter class and each view holder gets attached to a particular item view.

Figure 6: Class Diagram (Railways Section)
Figure 7 shows the demo of the working of the Recycler View Holder and Recycler View Adapter in the background. When the user enters the destination stop in the first picture of the Figure 7, the Railways Adapter triggers the Recycler View Holder and the Recycler View Adapter thereby showing the data populated by the model (SQLite Database). Since the number of trains retrieved could be a very huge number, the recycler view continuously recycles the older data and replaces it with the new data whenever the user scrolls down the Recycler View.

Figure 7: Demo of recycler view holder
Few snapshots of the User Interface have been attached here:

Figure 8: Screenshot of the UI
CHAPTER FIVE

SYSTEM DESIGN

Design Pattern

The MVC (Model View Controller) Design Pattern is used in the Railway module of the application. The backbone of any software development process is design patterns. As to write a good novel, we need to use good grammar, vocabulary, and idioms, similarly, to develop a good software we need to use appropriate design pattern. Design pattern helps in improving the efficiency of the app and increases the testability. The model, the view, and the controller are the three pieces of the pattern. MVC is advantageous to modularized development because of these feathers, and it considerably enhances system development efficiency, maintainability, and code reusability, which can adapt to the design needs of the more complicated multi-tier application system [1]. This pattern is an example of three-way factoring in action, in which objects of various classes take over actions related to the application domain (the model), the display of the program's state (the view), and user interaction with the model and the view (the controller) [4]. Figure 8 showcases a simple demonstration of the MVC Design Pattern.
The Model

The model contains only the application data, and no logic. In Java, it’s called the POJO class (Plain Old Java Objects). The behavior required of models is the ability to have dependents and the ability to broadcast change messages to their dependents (view and controller) in this case [4].

Figure 8 shows a sample data class for the railway’s module which contains instance variables like destination, type, and title of the train. It does not contain any business logic, hence encapsulating the data class completely. This data class is responsible to get and set the data from the database using the getters and setter method described in the code below. The data class transfers the data
to the controller (Railways Adapter) which then passed it to the view. The view is then responsible to showcase it appropriately to the user. Since the data model to not have any direction with the view, it is said that the model and the view in the MVC pattern are loosely coupled unlike other design patterns like the MVP (ModelView Presenter) or the MVVM (Model View View-Model).

```java
package com.rushabh.nasikapp.ahmedabadexplorer.Express;

class Express {
    private String desc;
    private String title;
    private String type;

    public String getType() {
        return this.type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public String getTitle() {
        return this.title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getDesc() {
        return this.desc;
    }

    public Express(String title, String desc, String type) {
        this.title = title;
        this.desc = desc;
        this.type = type;
    }
}

Figure 10: Sample code snippet of the Data Class
The View

All the activities and the layout xml files in the project are a part of the view in the MVC design pattern. In simpler terms, the view is what the user sees on the mobile screen. The View presents the model’s data to the user. It knows how to access the model’s data, but it does not know what this data means or what the user can do to manipulate it.

The Controller

Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate. In the Railway’s section, the adapter acts as a bridge between the activity and the model class. The model gets the data from the SQLite database and the controller thereby forwards the data to the view to display it to the user.

Figure 10 shown below depicts a high-level Use Case diagram of the Railways section of the application. The user provides the input destination stops to the view. The view knocks the controller's door, the controller thereafter asks the model (querying the SQLite database) to retrieve the data asked by the view. The model sends the data back to the controller thereafter sending it to the view. In this whole process, we can ensure that the business logic is separated from the view logic. So, in future when business logic happens to change, the view classes remain unhampered. This is basically the decoupling of the Model and the View by the controller.
Figure 11: MVC Use Case diagram
As discussed before this software application uses the SQLite database which is a Relational Database. Android does have a SQLite database implementation built-in. We will see below how we have used the SQLite database for our advantage by diving deep into the CRUD (Create Remove Update Delete) operations.

Database Creation

All the SQLite libraries are in the android.database.sqlite package. To build a database, just use the openOrCreateDatabase() method with the database name and mode as parameters. It returns a SQLite database instance that you must get in your own object [6]. In the Figure 11 we create the table named express by calling the execSQL ()method of the SQLiteDatabase class. Since the code block might throw an exception if the table do not exist or any column in the table is misspelled, we wrap it with the try and catch block to handle the exception if it arises.
To insert the data into the database, we use the execSQL() method defined in the SQLiteDatabase class. This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given as execSQL(String sql, Object[] bindArgs) [6]. Figure 12 shows the usage of the execSQL() to insert the data into the table nameexpress.
Figure 13: Snapshot of inserting data into the SQLite database

Database Fetching

To retrieve data from the database, we can use the object of the cursor class in Java. We use the method of the class RawQuery and it returns a ResultSet() with the cursor pointing to the table. We can move the cursor forward and move the data along.

The implementation of the cursor class is shown in Figure 13. If the cursor object's count is more than zero, we loop over all of the cursor object's values; otherwise, we simply display a toast message stating that the "train name is not found."
Figure 14: Snapshot of retrieving data from the SQLite database
As mobile devices become increasingly common, web services must be integrated into mobile applications [7]. Web View is a view in an Android application that displays web pages. WebView objects allow you to include web material in your activity layout, but they lack some of the functionality of fully-developed browsers. A WebView is a wonderful solution when you need greater UI control and advanced configuration options. [3]. It allows you to integrate web pages in a custom created environment for your project. We use a web view in the app to retrieve the list of picnic places fetched from an already existing API (Application Programming Interface). To add further security to the webpage, we use the object of the Web Settings class.

```java
String url = "https://www.holidify.com/weekend-getaways/from-ahmedabad.html";

@RequiresApi(api = Build.VERSION_CODES.CUPCAKE)
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_picnic);

    toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setTitle("Picnic places near Ahmedabad");
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    getSupportActionBar().setDisplayShowHomeEnabled(true);

    webView = (WebView) findViewById(R.id.webView1);
    pbar = (ProgressBar) findViewById(R.id.progressBar1);
    webView.setWebViewClient(new Picnic.WebViewClient());
    webView.loadUrl(url);
```

*Figure 15: Web view in Android*
The explanation of the methods declared in the WebSetting() class is given below.

- `setJavaScriptEnabled()`: This method of the Web Settings class tells webview to enable Javascript execution.
- `setBuiltInZoomControl()`: Sets whether the WebView should use its built-in zoom mechanisms.
- `setDisplayZoomControls()`: Sets whether the WebView should display onscreen zoom controls when using the built-in zoom mechanisms.
- `setAppCacheEnabled()`: Sets whether the Application Caches API should be enabled.
- `setDatabaseEnables()`: Sets whether the database storage API is enabled.
- `setDomStorageEnabled()`: Sets whether the DOM storage API is enabled. The default value is false.
- `setUseWideViewPort()`: Sets whether the WebView should enable support for the "viewport" HTML meta tag or should use a wide viewport.
- `setLoadWithOverviewMode()`: Sets whether the WebView loads pages in overview mode, that is, zooms out the content to fit on screen by width [3].
```java
final WebSettings webSettings = webView.getSettings();
webSettings.setJavaScriptEnabled(true);
webSettings.setBuiltInZoomControls(true);
webSettings.setDisplayZoomControls(false);
webSettings.setAppCacheEnabled(true);
webSettings.setDatabaseEnabled(true);
webSettings.setDomStorageEnabled(true);
webSettings.setUseWideViewPort(true);
webSettings.setLoadWithOverviewMode(true);
webView.getSettings().setAllowFileAccess(true);
webView.getSettings().setAllowContentAccess(true);
webView.setScrollbarFadingEnabled(false);
webSettings.setPluginState(WebSettings.PluginState.ON);
```

*Figure 16: Adding JavaScript to WebView*
CHAPTER EIGHT
TESTING

Android Linting

All the development of this application is done successfully by the Android Linting tool. Lint, a code scanning tool provided by Android Studio, can help you find and solve problems with the structural quality of your code. For example, Lint can help us clean up these issues

- XML resource files containing unused namespaces.
- Use of deprecated elements.
- API calls that are not supported by the target API versions, might lead to code failing to run correctly [3].

The project has been continuously verified by using the ./gradlew lint command to check if the lint is showing any errors. All the lint issues are stored in the lint_baseline file in the Android Studio. After running the program if the lint issues are known issues, the Studio ignores them. If the lint issues are unknown, the Studio throws an exception. The lint tool offers paths to the XML and HTML versions of the lint report once it has completed its inspections. The HTML report can then be opened in your browser by navigating to it.
CHAPTER NINE
FUTURE ADVANCEMENTS

• If the database size keeps increasing, shifting to a cloud database would be a good choice.

• Since Kotlin has started replacing Java and is being made an official language for Android Development, further modules should be developed in Kotlin. The older java files can be easily converted to Kotlin since Kotlin, and Java are fully interoperable.

• Incorporating a consumer complain section where people can get help from the government authority quickly.

• Adding a social network to the app where people can connect with each other.

• Adding a movie booking system from the app when the user search for any movie theatres.

• Providing a platform for the event managers in the city to publish and advertise their events or shows.
APPENDIX

CODE OF CRUCIAL PART
Express Adapter class

package com.rushabh.nasikapp.ahmedabadexplorer.Express;

import android.content.Context;
import android.content.Intent;
import android.support.v7.widget.RecyclerView;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import com.rushabh.nasikapp.ahmedabadexplorer.R;
import java.util.ArrayList;

class ExpressAdapter extends RecyclerView.Adapter<ExpressAdapter.MyViewHolder>
{
    ArrayList<Express> express_arraylist;
    Context context;

    public ExpressAdapter(Context context, ArrayList<Express> express_arraylist)
    {
        this.express_arraylist = express_arraylist; this.context = context;
    }

    public class MyViewHolder extends RecyclerView.ViewHolder {
        public TextView heading, subheading;
        public MyViewHolder(View itemView) {
            super(itemView);

            heading = (TextView) itemView.findViewById(R.id.heading);
            subheading = (TextView) itemView.findViewById(R.id.subheading);
        }
    }

    @Override
    public MyViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View itemView = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.list_of_express_main, parent, false);

        return new MyViewHolder(itemView);
    }
}
@Override
public void onBindViewHolder(MyViewHolder holder, final int position) {

Express express=express_arraylist.get(position);
holder.heading.setText(express.getTitle());
holder.subheading.setText(express.getDesc());

holder.heading.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

}})

holder.heading.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

}})
if (((Express)
ExpressAdapter.this.express_arraylist.get(position)).getType().equals("i")) {
    Intent i = new Intent(context, Express_Station_List.class);
    i.putExtra("destination", "Ahmedabad");
    i.putExtra("type", "i");
    context.startActivity(i);
} else if (((Express)
ExpressAdapter.this.express_arraylist.get(position)).getType().equals("a")) {
    Intent i = new Intent(context, ListofExpress.class);
    i.putExtra("source", "All"); i.putExtra("destination", "Trains"); i.putExtra("type", "a");
    context.startActivity(i);  
} else if (((Express)
ExpressAdapter.this.express_arraylist.get(position)).getType().equals("o")) {

    
}};
Intent i=new Intent(context,Express_Station_List.class);i.putExtra("source", "Ahmedabad");
    i.putExtra("type", "o"); context.startActivity(i);
}

holder.subheading.setOnClickListener(new View.OnClickListener()
    {
        @Override
        public void onClick(View v) {
            if (((Express) ExpressAdapter.this.express_arraylist.get(position)).getType().equals("i")) {
                Intent i=new Intent(context,Express_Station_List.class);
                i.putExtra("destination","Ahmedabad");
                i.putExtra("type","i");
                context.startActivity(i);
            } else if (((Express) ExpressAdapter.this.express_arraylist.get(position)).getType().equals("a")) {
                Intent i=new Intent(context,ListofExpress.class);
                i.putExtra("source","All"); i.putExtra("destination","Trains"); i.putExtra("type", "a"); context.startActivity(i);
            } else if (((Express) ExpressAdapter.this.express_arraylist.get(position)).getType().equals("o")) {
                Intent i=new Intent(context,Express_Station_List.class);
            }
        }
    });
i.putExtra("source", "Ahmedabad");i.putExtra("type", "o");
context.startActivity(i);
}
```java
@Override
public int getItemCount() {
return express_arraylist.size();
}
}

SQLite List Adapter Class

package com.rushabh.nasikapp.ahmedadexplorer.Express;

import android.content.Context;
import android.content.Intent; import android.view.LayoutInflater; import android.view.View;
import android.view.ViewGroup; import android.widget.ArrayAdapter; import android.widget.TextView;
import com.rushabh.nasikapp.ahmedadexplorer.BuildConfig;
import com.rushabh.nasikapp.ahmedadexplorer.R;
import java.util.ArrayList;

class SQLiteListAdapter_express extends ArrayAdapter<Array_Express>
{
ArrayList<Array_Express> data = new ArrayList();
int layoutResourceId; Context mContext;

public SQLiteListAdapter_express(Context context, int layoutResourceId, ArrayList<Array_Express> data) {
super(context, layoutResourceId, data);
this.layoutResourceId = layoutResourceId;
this.mContext = context;
this.data = data;
}

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

@Override
public View getView(int position, View child, ViewGroup parent) {
final Holder holder; LayoutInflater layoutInflater;
if (child == null) {
```
layoutInflater = (LayoutInflater)
mContext.getSystemService(Context.LAYOUT_INFLATER_SERVICE); child =
layoutInflater.inflate(R.layout.layout_of_list, null);

holder = new Holder();

holder.name = (TextView) child.findViewById(R.id.train_name);
holder.number = (TextView) child.findViewById(R.id.train_no);
holder.journey = (TextView) child.findViewById(R.id.journey);

child.setTag(holder);

} else {
    holder = (Holder) child.getTag();
}

final Array_Express user = (Array_Express) this.data.get(position);

holder.name.setText(user.getname());
holder.number.setText(user.getnumber());
holder.journey.setText(BuildConfig.FLAVOR + user.getSource() + "-" +
user.getDestination());

holder.name.setOnClickListener(new View.OnClickListener() {

     @Override
    public void onClick(View v) {
        Intent full_schedule = new Intent(mContext, Express_schedule.class);
        full_schedule.putExtra("name", holder.name.getText().toString());
        full_schedule.putExtra("number", holder.number.getText().toString());
        full_schedule.putExtra("source", user.getSource());
        full_schedule.putExtra("destination", user.getDestination());

        mContext.startActivity(full_schedule);
    }
});

holder.number.setOnClickListener(new View.OnClickListener() {

     @Override
    public void onClick(View v) {
        Intent full_schedule = new Intent(mContext, Express_schedule.class);
        full_schedule.putExtra("name", holder.name.getText().toString());
        full_schedule.putExtra("number", holder.number.getText().toString());
        full_schedule.putExtra("source", user.getSource());
        full_schedule.putExtra("destination", user.getDestination());

        mContext.startActivity(full_schedule);
    }
});
holder.journey.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent full_schedule = new Intent(mContext, Express_schedule.class);
        full_schedule.putExtra("name", holder.name.getText().toString());
        full_schedule.putExtra("number", holder.number.getText().toString());
        full_schedule.putExtra("source", user.getSource());
        full_schedule.putExtra("destination", user.getDestination());

        mContext.startActivity(full_schedule);
    }
});

return child;
}

public class Holder {
    TextView name; TextView number; TextView journey;
}

}
REFERENCES


Redundant Structure Based Tokenization Model for XML
Parsing on Android Mobile Devices”, 24 Nov. 2016,