Smart Phone Based Authentication System Using Gestures

Sandeep Ravikanti[1] Rajashekar Deva[2]
[1,2]Assistant Professor
[1,2]Department of Computer Science and Engineering
[1,2]Methodist College of engineering and technology, Hyderabad, Telangana, India

Abstract— Because of the developments in Mobile advancements, Gestures assume a vital part for the Authentication in the cell phones. With expanding prominence of the Authentications, the security dangers are likewise developing and that is clear in the episodes of UnAuthentication as far as different sorts. Along these lines it is vital to have a versatile based Authentication framework [1] for security. Motions permit clients to cooperate with your application by controlling the screen objects [2] you give. This paper presents components that assistance to synchronize with your own information, search for representatives by name, see their points of interest, add them to your contacts, and see their manager and employees, and also call, content, or email them.

Key words: Gestures, Intents

I. Introduction

For traditional web apps (delivered through a browser), Twitter Bootstrap can help. But what about Mobile apps? I explored Android here. Depending on what you are looking for, it may or may not be the right solution: jQM[8] provides mobile skins, but they don't look native. It's also more of a full stack framework than lightweight UI toolkit [9] that you can easily layer on top of your app.

The alternative to using an existing UI toolkit is to roll your own styles to make your application look and behave like a native app[1].

Employee Details always accesses the employee data from a SQLite[7] on your Android Mobile. The information is always available. The application comes with a sample dataset to provide an "out-of-the-box" experience, and with a simple offline synchronization mechanism to sync with your own data.

SQLite is a in-process library that implements a self-contained, serverless, zero configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is currently found in more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases[4], SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures[6]. These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen().

SQLite is a compact library. With all features enabled, the library size can be less than 350KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) If optional features are omitted, the size of the SQLite library can be reduced below 300KiB. SQLite can also be made to run in minimal stack space (4KiB) and very little heap (100KiB), making SQLite a popular database engine choice on memory constrained gadgets such as cellphones, PDAs, and MP3 players. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments.

SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite[7] responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest about all bugs and provides bugs lists including lists of critical bugs and minute-by-minute chronologies of bug reports and code changes.

The SQLite code base is supported by an international team of developers who work on SQLite full-time. The developers continue to expand the capabilities of SQLite and enhance its reliability and performance while maintaining backwards compatibility with the published interface spec, SQL syntax, and database file format.

II. Related work

The Employee Directory in JAVA page flow

(1) Search Page -> Employee Page -> Reports Page -> Employee Page -> …
(2) Search Page -> Employee Page -> Employee Page (manager) -> Reports
(3) Search Page -> Employee Page -> Employee Page (manager) -> Employee Page (manager's manager) -> …

The employee directory in java has o be governed by the external devices for the authentication. It needs a server to run the application where the front end is the HTML.

For storing the data it has to use the large databases like MYSQL and SQL[3].

III. Proposed work

The Employee Directory page flow is more random.

– Gesture Login ->Search Page -> Employee Page -> Reports Page -> Employee Page -> …

- Gesture Login ->Search Page -> Employee Page -> Employee Page (manager) -> Reports
- Gesture Login ->Search Page -> Employee Page -> Employee Page (manager) -> Employee Page (manager's manager) -> …
- Exploring the mechanism for internal and external detection.

Android supports gestures. To use this support your application must use the view "GestureOverlayView"[6]. In this view place you're other views.

Gestures are defined by binary resources which can be created with an example program from the Android SDK[1]. In your activity you can load Gestures via GestureLib.fromRawResource()[6]. If a gesture is detected then the method "onGesturePerformedListener" is called. For this the activity must implement the interface "OnGesturePerformedListener" and must register itself at the GestureOverlayView with the method "addOnGesturePerformedListener()".

Android shows the gestures in yellow for recognized gestures and a ligher yellow for not recognized gestures. You can turn this off, via setGestureColor(Color.TRANSPARENT) or setUncertainGestureColor(Color.TRANSPARENT) on the GestureOverlayView.

If you create the gesture in the Android simulator via the program "GestureBuilder"[6]. You can create several gestures with the same name. That may help you to determine the right one. If you create an Android Emulator for Android 1.6 this application will be preinstalled on your device. Make sure to create a device with sdcard otherwise you cannot save gestures. All gestures will be saved in a file called *gestures* on your emulator.

You can copy the gestures from the emulator via the ADB[9] onto your local machine via the command:

./adb pull /sdcard/gestures ~/test

The gesture file must be copied into your application under "res/raw". Afterwards it can be used in your GestureOverlayView[6].

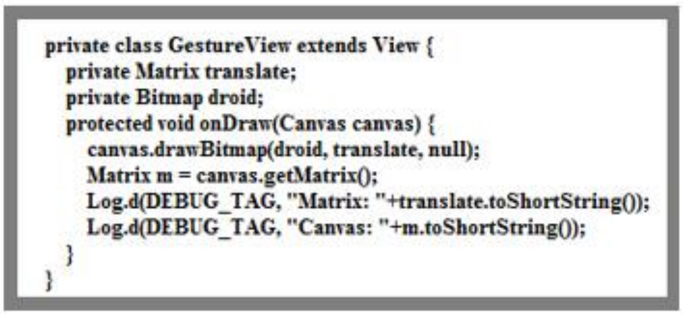
IV. Problem Definition and Assumptions

Authentication is essential in Mobiles. How can we provide the authentication for the mobile devices? Because the biometric is work properly with the mobile devices which use android. How can we manage the Employees in an organization in a compact way? The problem of Authentication is solved by using gestures instead of the biometric and the compact version of employees can be managed by using the Employee details. In Employee details look for employees by name, view their details, add them to your contacts, and see their manager and direct reports, as well as call, text, or email them.

V. Gestures Detection Algorithm

A. Creating a View: Gesture View

An easy way to get a Canvas object to drawing on is by overriding the onDraw() method of a View object. Conveniently, this method has a single parameter: the Canvas object. Drawing a Bitmap graphic on a Canvas object[9][3] is as easy as calling the drawBitmap() method of the Canvas object.

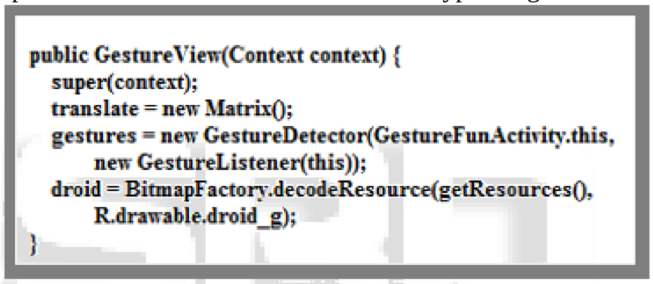


```
private class GestureView extends View {
  private Matrix translate;
  private Bitmap droid;
  protected void onDraw(Canvas canvas) {
    canvas.drawBitmap(droid, translate, null);
    Matrix m = canvas.getMatrix();
    Log.d(DEBUG_TAG, "Matrix: "+translate.toShortString());
    Log.d(DEBUG_TAG, "Canvas: "+m.toShortString());
  }
}
```

Fig. 1: Shows How to create a Gesture View

B. Configuring the Gesture View

A GestureDetector is an Android class that can take motion events, do some mathematical magic to determine what they are, and then delegate calls to a GestureListener object as specific gesture or other motion callbacks. The GestureListener object[5], a class we implement, receives these calls for specific gestures that the GestureDetector[6] recognizes and allows us to react to them as we see fit (in this case, to move a graphic around within our PlayAreaView). Although the GestureDetector[5] handles the detection of certain motions, it doesn't do anything specific with them nor does it handle all types of gestures.

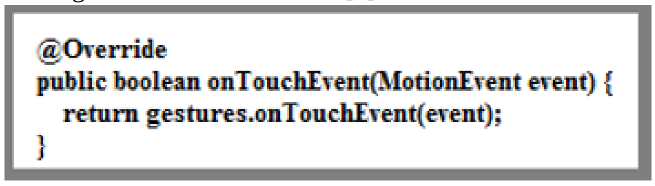


```
public GestureView(Context context) {
  super(context);
  translate = new Matrix();
  gestures = new GestureDetector(GestureFunActivity.this,
      new GestureListener(this));
  droid = BitmapFactory.decodeResource(getResources(),
      R.drawable.droid_g);
}
```

Fig. 2: Shows GestureView Constructor

C. Connecting to a Gesture Detector

GestureDetector[6] object receives the motion data it needs to do its gesture recognizing magic. GestureDetector object called gestures to receive events[1].

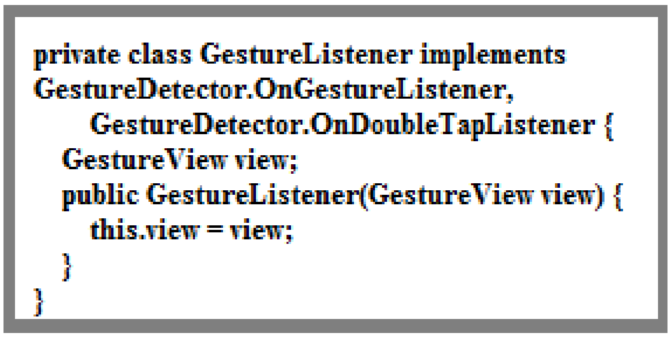


```
@Override
public boolean onTouchEvent(MotionEvent event) {
  return gestures.onTouchEvent(event);
}
```

Fig. 3: Shows Gesture Detection Object

D. Implementing the Gesture Listener

In order to react to the events recognized by the GestureDetector class, we need to implement the GestureListener class. The motion events[2] we are most interested in are double taps and gestures of any kind. To listen for these types of motion events, our GestureListener class must implement both the OnGestureListener and OnDoubleTapListener interfaces[3].

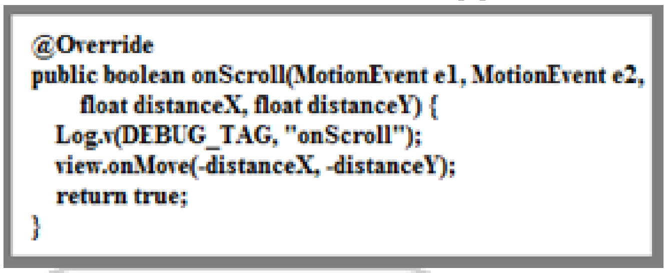


```
private class GestureListener implements
GestureDetector.OnGestureListener,
    GestureDetector.OnDoubleTapListener {
  GestureView view;
  public GestureListener(GestureView view) {
    this.view = view;
  }
}
```

Fig. 4: shows Gesture Listener

E. Handling Motion Events

A scroll event occurs when the user touches the screen and then moves their finger across it. This gesture is also known as a drag event. This event comes in through the onScroll() method of the OnGestureListener interface[5].



```
@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2,
    float distanceX, float distanceY) {
  Log.v(DEBUG_TAG, "onScroll");
  view.onMove(-distanceX, -distanceY);
  return true;
}
```

Fig. 5: Shows onScroll()

VI. IMPLEMENTATION

A. Admin

1) Add Gesture

The admin has the permission to add the Gesture signature of all the Employees and can view all the details of the employee.

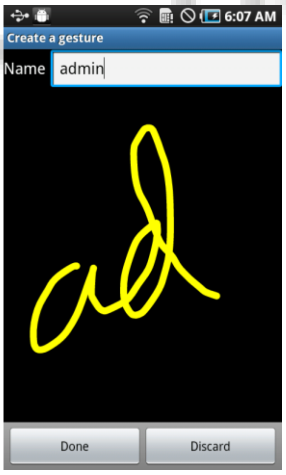
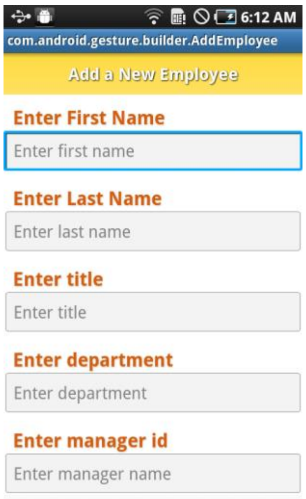


Fig. 6: Adding the Signature

Each and every Employee will have a particular signature. Based on the signature the employee login. The Employee signatures are added by the Admin.

2) Add Employee

The Admin has permissions to add all the Employees. For adding the Employee the following details need to be entered First Name, Last Name, Department, Manager id, Phone Number and Email



com.android.gesture.builder.AddEmployee
Add a New Employee
Enter First Name
Enter first name
Enter Last Name
Enter last name
Enter title
Enter title
Enter department
Enter department
Enter manager id
Enter manager name

Fig. 7: Adding the Details of the Employee

3) Home Activity

In this page there will four sub modules

(1) Employee Reports
(2) Employee Full Details
(3) Add Employee
(4) Save Signature

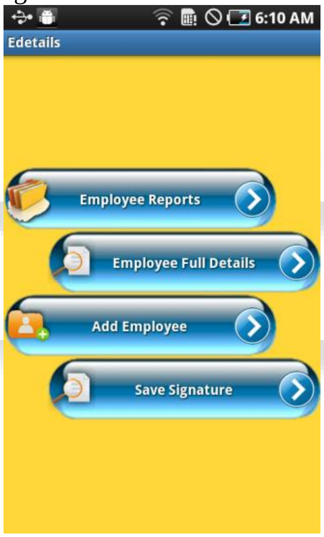


Fig. 8: Other Modules page

B. Employee

The Employee has to enter the Gesture signature for login. If the Employee Login Gesture Signature is correct then the Employee will be redirected to the next page where the employee can view the other employee details along with their details and their manager.

1) Employee Login

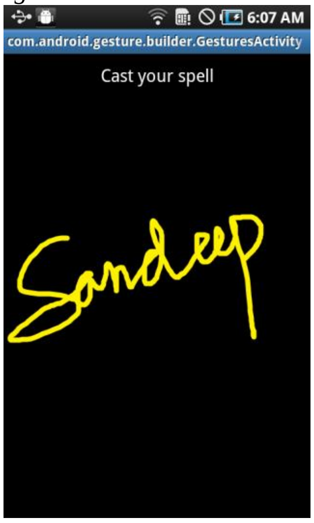


Fig. 9: Signature of the Employee

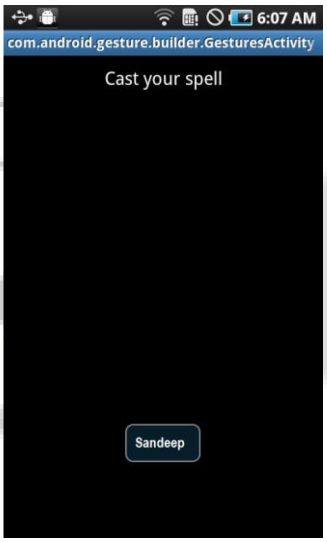


Fig. 10: Employee Login Successful

If the Employee Login Gesture Signature[6] is correct then the Employee will be redirected to the next page where the employee can view the other employee details along with their details and their manager.

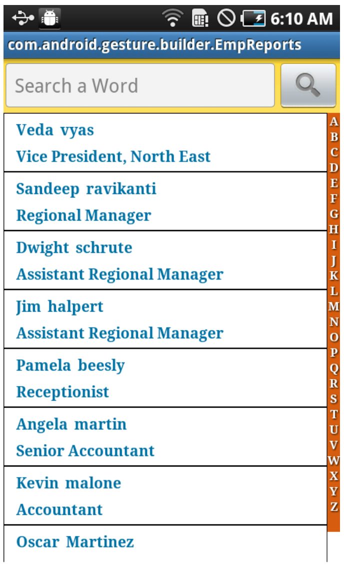
C. *Employee Reports*

The Employee can view all other Employee details and the designation of the Employees. The Employee can even search the other employees. Based on the selection of the Employees the corresponding details will be displayed.

The display of elements in lists is a very common pattern in mobile applications. The user sees a list of items and can scroll through them. If he selects one item, this can update the ActionBar[9] or triggers[3] a detailed screen for the selection.

Android provides the ListView[2] class which is capable of displaying a scrollable list of items. These items can be of any type.

The Alphabetic search bar is used for displaying the list with the selected Alphabet on the right side of the screen.


6:10 AM
com.android.gesture.builder.EmpReports
Search a Word
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Veda vyas
Vice President, North East
Sandeep ravikanti
Regional Manager
Dwight schrute
Assistant Regional Manager
Jim halpert
Assistant Regional Manager
Pamela beesly
Receptionist
Angela martin
Senior Accountant
Kevin malone
Accountant
Oscar Martinez

Fig. 11: Employee Reports.

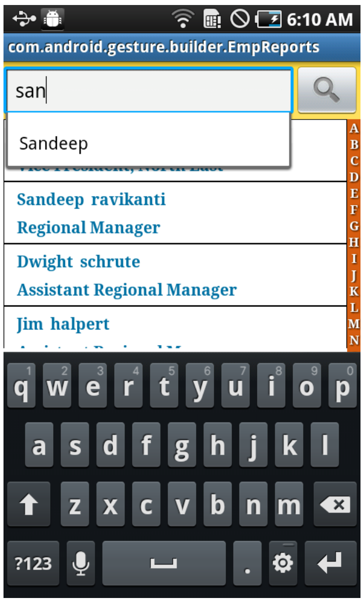
D. *AutoCompleteSearch*

A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

An editable text view that shows completion suggestions automatically while the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box with.

The drop down can be dismissed at any time by pressing the back key or, if no item is selected in the drop down, by pressing the enter/dpad center key[1].

The list of suggestions is obtained from a data adapter and appears only after a given number of characters defined by the threshold.



Sandeep ravikanti
Regional Manager
Dwight schrute
Assistant Regional Manager
Jim halpert

Fig. 12: Auto Complete Search

Based on the search the user can view the list populated down in the ListView

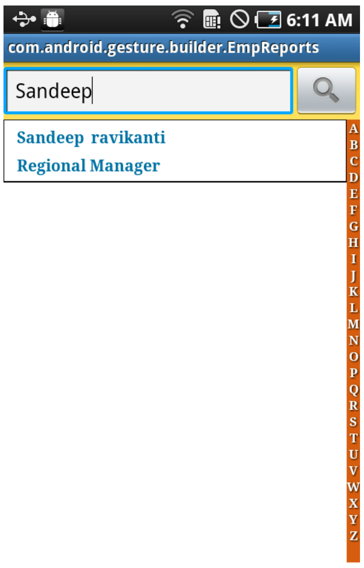

6:11 AM
com.android.gesture.builder.EmpReports
Sandeep
Sandeep ravikanti
Regional Manager
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Fig. 13: Result based on Auto Complete Search

E. Employee Results

The Employee Results shows the Employee name, Designation, Phone Number and view reports.

From this activity we can Call, Send SMS and Send Email.

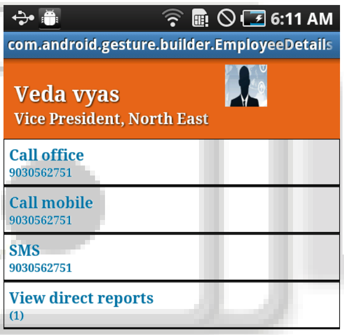


Fig. 14: Employee Results Page.

F. View Reports

We can view the Manager details of a particular Employee and can view the manager details based on the selection of the ListView[2].

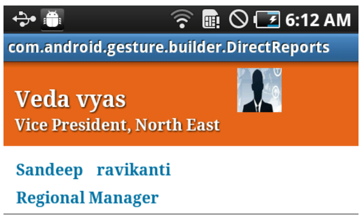
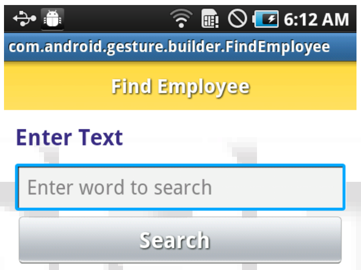

6:12 AM
com.android.gesture.builder.DirectReports
Veda vyas
Vice President, North East
Sandeep ravikanti
Regional Manager

Fig. 15: shows View Reports

G. Employee Full Details

We can search the Employee and can view full details of the Employee.



VII. CONCLUSION

Authentication is Important in the Mobile Technology. This paper presents the Gestures [6] Authentication for Login and compact view of the Employee Details. The effectiveness of the simulation model is tested with simulations using a custom-built[1] simulator developed in Android. The results revealed that the proposed analytical model is effective and can be used in real world applications.

REFERENCES

[1] http://developer.android.com/index.html
[2] http://www.vogella.com/articles/AndroidListView/article.html
[3] http://mobile.tutsplus.com
[4] http://stackoverflow.com
[5] http://www.vogella.com/tutorials/AndroidIntent/article.html

[6] http://www.tutorialspoint.com/android/android_gestures.htm
[7] http://www.androidhive.info/2011/11/android-sqlite-database-tutorial/
[8] http://demos.jquerymobile.com/1.2.0/docs/pages/popup/
[9] http://www.xda-developers.com/