Phone E-Checkout App

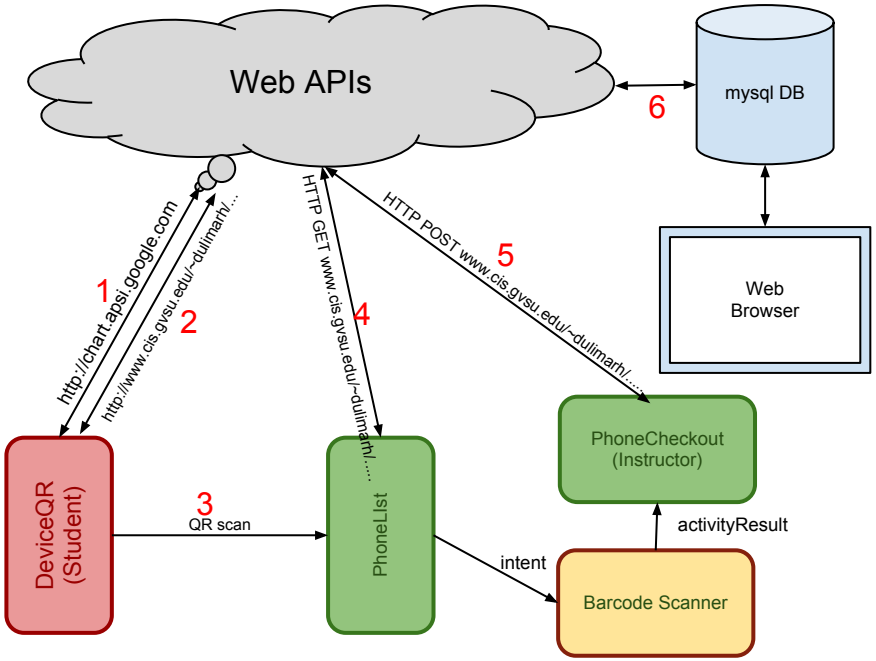
Hans Dulimarta

dulimarh@cis.gvsu.edu

1 Introduction

The Phone E-Checkout program consists of two separate apps.

- `DeviceQR` that runs on the phone being checked out by a student
- `PhoneCheckout` that runs on the instructor's phone. The first screen of this app lists all the phones currently checked out (PhoneID + student userid)



1. Using Android TelephonyManager service, `DeviceQR` obtains the phone DeviceID and shows it using a QR Code generated from http://chart.apis.google.com/chart. This HTTP GET request runs inside an AsyncTask (`URLTask`).

```
HttpResponse res = client.execute(req);
InputStream istr = res.getEntity().getContent();
Bitmap img = BitmapFactory.decodeStream(istr);
result[0] = img;
```

2. Using the same AsyncTask, `DeviceQR` checks if the device was checked out. It makes another HTTP GET to http://.../checkout.php?device=xyz and receives a JSON output that includes the device id, the user id, name of

the student who checked the phone out, and timestamp of checkout. This information is produced by PHP script that runs queries to a MySQL database running on the EOS server.

```
req = new HttpGet(CHECKOUT_URL + "device=" + tm.getDeviceId());
res = client.execute(req);
Scanner scan = new Scanner (res.getEntity().getContent());
String jsonstr = "";
while (scan.hasNextLine()) {
    jsonstr += scan.nextLine();
}
JSONObject obj = new JSONObject(jsonstr);
```

3. To initiate e-checkout, the `PhoneCheckout` app on the instructor's phone scans the QR code with the BarCode Scanner app (invoked via an Intent).

```
@Override
public void onClick(View v) {
    Intent scan = new Intent ("com.google.zxing.client.android.SCAN");
    scan.putExtra("SCAN_MODE", "QR_CODE_MODE");
    startActivityForResult(scan, 0);

}
```

The PhoneID string returned by the BarCode Scanner is then checked using another HTTP GET to query the database.

```
 */
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
            String contents = data.getStringExtra("SCAN_RESULT");
            String format = data.getStringExtra("SCAN_RESULT_FORMAT");
            Intent next = new Intent (PhoneListActivity.this, PhoneCheckoutActivity.class);
            next.putExtra("dev_id", contents);
            startActivity(next);
```

```
        Log.d(TAG, contents + " => " + format);
    }
```

4. During the e-checkout transaction, the student signs his/her name on screen prior to confirming the transcaction.

5. To record the transaction, an HTTP POST is issued and the phoneID, user name, userid, time of checkout, and student signature are recorded in the database.

```
HttpClient client = new DefaultHttpClient();
HttpContext ctx = new BasicHttpContext();
HttpPost poster = new HttpPost(CHECKOUT_URL);
MultipartEntity entity = new MultipartEntity(HttpMultipartMode.BROWSER_COMPATIBLE);
try {
    entity.addPart("name", new StringBody((String) uid.getSelectedItem()));
    entity.addPart("device", new StringBody(deviceId));
    entity.addPart("time", new StringBody(String.valueOf(System.currentTimeMillis())));
    File img = new File (Environment.getExternalStorageDirectory() + "/sig.png");
    entity.addPart("image", new FileBody(img));
    poster.setEntity(entity);
    HttpResponse resp = client.execute(poster, ctx);
```

6. All the HTTP GET and POST requests are translated to mysql SELECT and INSERT queries

2 HTTP POST

When you submit a form on a web browser, the data in the form are transmitted to the server using HTTP POST. To send multiple data items to the server HTTP POST relies on MIME (Multipurpose Internet Mail Extension) encoding. This is exactly the same encoding technique used for sending (multiple) attachments in an email.

The code snippet above posts four different "attachments" (name, device, time, and image). `MultipartEntity`, `StringBody`, `FileBody` are classes from an additional library included in the project (`httpmime-4.1.3.jar`).

3 SignatureView

The signature box is a customized View designed to handle the user scribbles on the screen. The design of this class is mainly based on the `FingerPaint` example found under the APIDemos in the SDK sample. Additional logic was added to the original FingerPaint example to handle dots in the scribble. The main technique used to capture the signature is to override the `onTouchEvent` method. Several factors that determines the design of the `SignatureView` class:

1. A signature may consist of several strokes.

2. A single stroke begins when the user's finger tip touches the screen (`ACTION_DOWN`), and ends when the finger tip leaves the screen (`ACTION_UP`). In between these two events the finger tip moves across the screen (`ACTION_MOVE`).

To record a signature with multiple strokes:

- Each stroke is recorded in a `Path`.
- At the beginning of a signature stroke the path is reset and the starting coordinate of the stroke is recorded.

```
sig.reset();
sig.moveTo(x, y);
```

- At the end of a stroke the path is drawn to a canvas.

```
if (moved) {
    sig.lineTo(x, y);
    sigCanvas.drawPath(sig, sigColor);
}
else {
    sigColor.setStyle(Style.FILL);
    sig.addCircle(x, y, 5, Direction.CW);
    sigCanvas.drawPath(sig, sigColor);
}
```

- As the finger tip scribles a stroke, bezier curves are added to the path.

```
/* use Bezier segments for smoother curves */
sig.quadTo(posx, posy, (x + posx)/2, (y + posy)/2);
```

- While the `onTouch()` method records the fingertip motions the `onDraw()` methods renders the signature using the following technique:
  - The current stroke is rendered from the path
  - The previously recorded strokes are rendered from the canvas

4 Downloading The Source Code

The source code of these two apps are available for download from the CIS git repository. Use the following URI:

```
cis163@eos08.cis.gvsu.edu:PhoneCheckout.git
```

[You can always replace eos08 with any other eosxx hostname]