# Mobile Electronic Training Jacket
Post Documentation
Team Members : Jake Strojny, Tyler McInnis, Robert Huard, Kevin MacAllister

**Table of Content**

## 1. Description of code architecture

Inside of the android application there are three major parts, the activities/displays, network/refresh classes, and the data models. The activities/displays handle user entered data, on-click actions, displaying data to the user and the switching of activities. The network and refresh classes handle communication between the phone and the server, and the storing of entries into the phone's local database. Finally, the data models handle the querying of the local database and structuring of the output for the activities.

Inside of the server there are two servlets, one which handles a request for login and refreshing of data, one which handles a request to register a new phone. Then there is a data processing class which uses data models similar to those on the phone. However, instead of creating human readable output, it creates the ends of SQL Insert statements delimited by newlines and sends them to the servlet, which subsequently sends them to the phone where they are turned into full Insert statements and executed. In this manner the phone updates its local database.

## 2. Android Data Models

There is a data model for each activity which requires queries to be made against the local SQL database. They are all their own class, however they are all designed similarly.

**Variables :**
```
String [] Variable1;
…
String[] VariableN; //Where Variables 1 to N are all the Variables associated with the query
Int NumberOfRecords;
```

**Functions :**
Constructor(String DOD_EDI_PI, SQLiteDatabase Access)

This function is the constructor of the data model and is called when the user enters the activity it is associated with. It queries the local database for the information that the activity needs to display and stores the results in the arrays of variables defined above.

print(ScrollView myScrollView, Context myContext)

This function creates a LinearLayout and appends the results of the query to a number of ExpandableTextViews that is equal to the number of entries returned from the query. The ExpandableTextViews are able to be expanded upon clicking and display only the titles. Then they display the rest of the entry, which is stored in it's child object, when clicked. It also includes a line in order to be more distinguishable.

That LinearLayout is then appended to the ScrollView, this is done because ScrollView can only have one child.

SearchPosition(String SearchString)

This function loops through all of the ExpandableTextViews and compares the SearchString, which is passed from the search box in the associated activity, against the titles of the ExpandableTextViews. Both are cast to uppercase, so searches are not case sensitive. If the ExpandableTextViews title contains the SearchString, it is made visible. If it does not contain it, it is made invisible.

## 3. Android Activities

Each activity has a Bundle object. Bundles are parameters that have a key-value architecture. At the beginning of each activity a new bundle that holds the DOD_EDI_PI value and PIN of the user is created so that it can later be passed if the user enters a new activity.

Also each activity has buttons and other menu items which are found with FindViewById(). The on-click methods for these buttons are defined to do the following :

Menu Button :
Creates a new "Intent" which describes what activity we are to transfer into. Then it passes the bundle into that intent, finally the startActivity function with that intent as a parameter.

Search Button :

Grabs the value from the search box and then passes it to the SearchPosition of the data model created earlier in the activity.

The Data Model associated with the activity is created at the beginning of the activity, being passed the Database and the DOD_EDI_PI value that is stored in the activities bundle. Then the print value of the Data Model created is called, being passed the ScrollView and the Context of the activity. This prints all the values to the screen, later to be filtered with search if desired.

Network and Refresh Activities will be discussed in a later section.

## 4. Activity XML

---

The XML's for each of our activities are fairly simple as most data displayed is created programmatically. The styling options for everything other than ExpandableTextViews is stored in these XML files for the associated activities.

Data Display Activities :

For these there is a ScrollView which has padding above and below. A back button, a search box and a search button.

Also there is a bar with buttons which brings you to any of the menus

Login Activity :

For this there is a text box which prompts the user for their pin, and a button to confirm and login. There is also a TextView to update the user on progress.

Menu Activities :

For these there are a number of buttons which bring you to the menu's associated activities, a back button to go back in the menu tree, and a bar with buttons to bring you to any menu.

Refresh Activity :

For this there is a button to refresh and a TextView to update the user on progress.

## 5. Sign In Functionality and Refresh

The Sign In functionality is split into an activity that prompts the user for their login info, an Asynchronous task that connects the user to the server and accepts information, and a class that handles returned information by using it to refresh the local database.

Login Activity :

The login activity acquires the WLAN_MAC number of the phone, accepts the user input PIN number, then updates the URL of the server with those two arguments and starts communication with the server. When the download is complete, updateFromDownload() is called with a string DownloadReturn containing the information returned from the server.

```
If (no problem with connection to server, correct login information)
{
Flush old information from local database;
Insert new information to local database; //this includes local login tables
Initial bundling of ID and PIN Values;
Call main activity;
}
If (WLAN_MAC does not exist in server)
{
Prompt user for DOD_EDI_PI and old PIN;
//Server generates new PIN for user and sends to their email, assigns new WLAN_MAC to user
User returns to login screen
}
If (Problem with connection to server && login information correct on local database)
{
Send user to main activity without updating information;
}
```

Refresh Activity :

There is also a refresh activity for after the user has already successfully logged in, the user is not prompted for a PIN again, the database is refreshed following the same logic as the Login Activity.

The Refresher class handles all refreshing, the input from download return is segmented into blocks of information for different tables. The first line includes the users DOD_EDI_PI, then the block format is as follows for each of the tables being updated :

A number indicating how many entries for the table are to be inserted

The ends of insert statements for the entries to be inserted into the table

The refresher takes these values and constructs the beginnings of these insert statements and then sends the query to the database until all data has been entered.

## 6. Tomcat Servlets

### MainServlet

The MainServlet for our Tomcat server accepts two arguments -- the users WLAN_MAC and PIN values. These are then send to the data processor class which checks if the WLAN_MAC and PIN exist in the table. If they do, the DOD_EDI_PI number is returned and it is sent to each of the data models constructors. Once the data models have been constructed, the print function for each of the data models is used to create the return string. Once the return string is created, the main servlet uses it to send back to the phone.

If the WLAN_MAC exists, but the PIN is incorrect, the MainServlet sends back a string indicating that the user must retry a different PIN.

If the WLAN_MAC does not exist, the MainServlet sends back a string indicating that the user must register their phone. The phone is then in a new activity where it communicates with a different servlet on our server, the ChangeServlet

### ChangeServlet

The second servlet is the changeServlet, which accepts three arguments -- the user's new WLAN_MAC, PIN, and ID. These arguments are used to access and update the LOGIN table with the user's new WLAN_MAC. The servlet then generates a new random pin and emails it to the user. If all of this happens the servlet sends back a SUCCESS message to the phone

If PIN or ID are incorrect the changeServlet sends back a INVALID_ID_PIN string indicating that the user entered incorrect information.

If the PIN and ID are correct and the servlet successfully updates the database the string SUCCESS is returned, letting the phone know to switch back to the sign in activity.

If the PIN and ID are correct, but the servlet fails to update the database due to some exception then it returns FAILURE and the app allows the user retry.

In order to compile the servlets the following jars must be added to the classpath: mail.jar, activation.jar, and servlet-api.jar.

## 7. Server Data Models

The server data models operate very similarly to the data models on the phone. However, there are a few key differences. Where the phone queries for the codes associated with a user in a given category, and then queries for the description of these codes, the server data models only handle the codes. This is an attempt to reduce the amount of traffic between the server and the client.

Another key difference is that instead of generating a GUI with descriptive entries, the server data models construct the ends of an INSERT statement with new line delimited between entries

For example, where the client will construct an INSERT statement :

INSERT INTO myTable (column1, column2, column3) VALUES (value1, value2, value3)

The server would create :
(value 1, value2, value3)

## 8. Known Bugs

1) Occasionally, when the application attempts to connect to the changeServlet a connection error is returned. After this error the servlets only return empty strings and if the user goes back to the sign in activity they can sign in even if the WLAN is incorrect, but the data models will not build because an ID is not being passed to them.
    a) Possible Solutions: Restarting the Tomcat server appears to fix this problem. It happens very sporadically and after some research it appears to be caused by an interaction between Tomcat and Amazon's snapshot of Ubuntu.