Appendices

A.1. mOpenID Android Application Code.....2
A.2. mOpenID Android Webapp Servlet Code 56
A.3. mOpenID Web Service Code 66

AbstractActivation.java

```
001 package no.ntnu.item.mopenid.app;
002
003 import java.util.Calendar;
004
005 import no.ntnu.item.mopenid.util.OpenID;
006 import no.ntnu.item.mopenid.util.Utils;
007 import android.app.Activity;
008 import android.app.Dialog;
009 import android.app.ProgressDialog;
010 import android.content.BroadcastReceiver;
011 import android.content.Context;
012 import android.content.Intent;
013 import android.content.IntentFilter;
014 import android.graphics.Color;
015 import android.os.AsyncTask;
016 import android.os.Bundle;
017 import android.text.format.DateFormat;
018 import android.util.Log;
019 import android.view.Menu;
020 import android.view.MenuItem;
021 import android.view.View;
022 import android.view.View.OnClickListener;
023 import android.widget.Button;
024 import android.widget.LinearLayout;
025 import android.widget.ProgressBar;
026 import android.widget.TextView;
027
028 public abstract class AbstractActivation extends Activity {
029
030   protected static final int CANCEL_LOGIN_DIALOG = 0;
031   protected static final int SETUP_PROGRESS_DIALOG = 1;
032   protected static final int AUTHENTICATION_DIALOG = 2;
033   protected static final int AUTHENTICATION_CONFIRMATION_DIALOG =
3;
034   protected static final int CLAIMED_ID_MISMATCH = 4;
035   protected static final String INTENT_ACTION_AUTH_REQUEST = "no.
ntnu.item.mopenid.ACTION_AUTH_REQUEST";
036   protected static final String INTENT_ACTION_AUTH_RESPONSE =
"no.ntnu.item.mopenid.ACTION_AUTH_RESPONSE";
037   protected static final String INTENT_ACTION_AUTH_COMPLETED =
"no.ntnu.item.mopenid.ACTION_AUTH_COMPLETED_LOGIN";
038   protected static final String INTENT_ACTION_AUTH_COMPLETELY_
COMPLETED = "no.ntnu.item.mopenid.ACTION_AUTH_COMPLETED_LOGIN_
COMPLETELY";
039
040   protected ProgressDialog progressDialog;
041   protected BroadcastReceiver receiver;
042   boolean linkedWithmOpenID = false;
043   protected String identifier = null;
044   protected Intent receivedIntent;
045   protected int currentShowingDialogId;
046
047   protected void onCreate(Bundle savedInstanceState) {
048     super.onCreate(savedInstanceState);
049     setContentView(R.layout.activation_screen);
050     receiver = null;
051     setUpCancelButtonBehavior();
```

```
052   }
053
054   @Override
055   public boolean onCreateOptionsMenu(Menu menu) {
056     menu.add(Menu.NONE, 1, Menu.NONE, "Simulate authentication
request");
057     return super.onCreateOptionsMenu(menu);
058   }
059
060   @Override
061   public boolean onOptionsItemSelected(MenuItem item) {
062
063     switch (item.getItemId()) {
064     case 1:
065       showDialog(AUTHENTICATION_DIALOG);
066       return false;
067
068     default:
069       return super.onOptionsItemSelected(item);
070     }
071   }
072
073   protected abstract void setUpCancelButtonBehavior();
074
075   @Override
076   protected void onPause() {
077     unregisterReceiver(receiver);
078     super.onPause();
079   }
080
081   @Override
082   protected void onResume() {
083
084     receiver = new BroadcastReceiver() {
085
086       @Override
087       public void onReceive(Context context, Intent intent) {
088         receivedIntent = intent;
089         if((intent.getStringExtra("openid.claimed_id").
equalsIgnoreCase(identifier) && AbstractActivation.this instanceof
StaticIpModeActivation)
090             || (intent.getStringExtra("openid.claimed_id").
endsWith(identifier) && AbstractActivation.this instanceof
MOpenIdModeActivation)) {
091           showDialog(AUTHENTICATION_DIALOG);
092         } else {
093           sendAuthReponse(false, "claimed id mismatch");
094           Log.d("eirik", "openid.identifier: "+intent.
getStringExtra("openid.claimed_id")+ " and identifier: "+identifier);
095           showDialog(CLAIMED_ID_MISMATCH);
096         }
097
098
099       }
100     };
101
102     registerReceiver(receiver, new IntentFilter(INTENT_ACTION_
AUTH_REQUEST));
103     super.onResume();
104   }
105
106   protected abstract void showOpenIDURLAndCancelButton();
107
108
109   protected void setAllTextViewsToSuccessful() {
```

```
110     ProgressBar progressBar = (ProgressBar) findViewById(R.
id.progressbar_activating_id);
111     progressBar.setVisibility(View.GONE);
112     TextView activationHeading = (TextView) findViewById(R.
id.activation_heading);
113     activationHeading.setText(R.string.id_activated);
114     activationHeading.setTextColor(Color.rgb(0, 153, 0));
115
116     LinearLayout layout = (LinearLayout) findViewById(R.
id.layout_activation_heading);
117     layout.requestLayout();
118   }
119
120   protected void startLocalOpenIDProvider() {
121     //
122     Intent intent = new Intent("hello");
123     intent.putExtra(Utils.__PORT, Utils.__PORT_DEFAULT);
124     intent.putExtra(Utils.__NIO, Utils.__NIO_DEFAULT);
125     intent.putExtra(Utils.__SSL, Utils.__SSL_DEFAULT);
126     intent.putExtra(Utils.__CONSOLE_PWD, Utils.__CONSOLE_PWD_
DEFAULT);
127     startService(intent);
128     Log.i("eirik", "Local OP started");
129   }
130
131
132   @Override
133   protected Dialog onCreateDialog(int id) {
134     Dialog dialog;
135     currentShowingDialogId = id;
136     switch (id) {
137     case CANCEL_LOGIN_DIALOG:
138       dialog = new Dialog(this);
139
140       dialog.setContentView(R.layout.popup_disassociation);
141       dialog.setTitle(R.string.closing_application);
142       dialog.setCanceledOnTouchOutside(false);
143       break;
144
145     case SETUP_PROGRESS_DIALOG:
146       progressDialog = new ProgressDialog(this);
147       dialog = progressDialog;
148       dialog.setCancelable(false);
149       dialog.setCanceledOnTouchOutside(false);
150       setupProgressDialog();
151       break;
152
153     case AUTHENTICATION_DIALOG:
154       dialog = new Dialog(this);
155       setupAuthenticationDialog(dialog);
156       break;
157
158     case AUTHENTICATION_CONFIRMATION_DIALOG:
159       dialog = new Dialog(this);
160       setupAuthenticationConfirmationDialog(dialog);
161
162       break;
163
164     case CLAIMED_ID_MISMATCH:
165       dialog = new Dialog(this);
166       setupClaimedIdMismatchDialog(dialog);
167       break;
168     default:
169       dialog = null;
170       currentShowingDialogId = -1;
```

```
171     }
172     return dialog;
173   }
174
175
176   protected abstract void setupProgressDialog();
177
178   private void setupClaimedIdMismatchDialog(Dialog dialog) {
179     dialog.setContentView(R.layout.claimed_id_mismatch_dialog);
180     dialog.setTitle(R.string.invalid_request_heading);
181     dialog.setCancelable(false);
182     dialog.setCanceledOnTouchOutside(false);
183     Button okayButton = (Button) dialog
184     .findViewById(R.id.mismatch_okay_button);
185     okayButton.setOnClickListener(new OnClickListener() {
186       @Override
187       public void onClick(View v) {
188         perfromCancelAction();
189
190       }
191     });
192
193   }
194
195   protected abstract void perfromCancelAction();
196
197   private void setupAuthenticationConfirmationDialog(Dialog
dialog) {
198     dialog.setContentView(R.layout.authentication_confirmation_
dialog);
199     dialog.setTitle(R.string.auth_confirmation_heading);
200     dialog.setCanceledOnTouchOutside(false);
201     dialog.setCancelable(false);
202
203
204     Button okayButton = (Button) dialog
205         .findViewById(R.id.authentication_confirm_okay_button);
206     okayButton.setVisibility(View.GONE);
207     okayButton.setOnClickListener(new OnClickListener() {
208
209       @Override
210       public void onClick(View v) {
211
212       }
213     });
214
215   }
216
217   protected abstract void authenticationFinished();
218
219   private void sendAuthReponse(boolean approved, String reason)
{
220     Intent intent = new Intent(INTENT_ACTION_AUTH_RESPONSE);
221     intent.putExtra("authenticated", approved);
222     intent.putExtra("identifier", (this instanceof
MOpenIdModeActivation) ? Utils.MOPENID_URL + identifier : Utils.
getLocalEndpointURLAsString());
223
224     if(!approved) {
225       intent.putExtra("reason", reason);
226     } else {
227
228       setupAuthenticationCompletelyCompletedReceiver();
229
230
```

```
231       unregisterReceiver(receiver);
232       receiver = new BroadcastReceiver() {
233
234         @Override
235         public void onReceive(Context context, Intent intent) {
236           // update last used
237           if(AbstractActivation.this instanceof
MOpenIdModeActivation) {
238             OpenID openId = new OpenID(identifier);
239             String lastUsed = (String) DateFormat.format("yyyy-
MM-dd kk:mm:ss", Calendar.getInstance());
240             Log.d("eirik", "set last used to: "+lastUsed);
241             openId.setLastUsed(lastUsed);
242             Utils.dbUpdateLastUsed(openId);
243           }
244         }
245       };
246
247       registerReceiver(receiver, new IntentFilter(INTENT_ACTION_
AUTH_COMPLETED));
248     }
249     sendBroadcast(intent);
250     Log.i("openid", "Sent Intent ala "+INTENT_ACTION_AUTH_
RESPONSE);
251   }
252
253   private void setupAuthenticationCompletelyCompletedReceiver()
{
254     BroadcastReceiver completelyReceiver = new
BroadcastReceiver() {
255
256       @Override
257       public void onReceive(Context context, Intent intent) {
258         authenticationFinished();
259       }
260     };
261
262     registerReceiver(completelyReceiver, new
IntentFilter(INTENT_ACTION_AUTH_COMPLETELY_COMPLETED));
263   }
264
265   private void setupAuthenticationDialog(Dialog dialog) {
266     dialog.setContentView(R.layout.authentication_dialog);
267     dialog.setCanceledOnTouchOutside(false);
268     dialog.setCancelable(false);
269
270     TextView realmTextView = (TextView) dialog
271         .findViewById(R.id.auth_realm);
272     realmTextView.setText(receivedIntent.getStringExtra("openid.
realm"));
273     TextView identifierTextView = (TextView) dialog
274         .findViewById(R.id.auth_identifier);
275     identifierTextView.setText(identifier);
276
277     Button yesButton = (Button) dialog
278         .findViewById(R.id.authenticate_yes_button);
279     yesButton.setOnClickListener(new OnClickListener() {
280
281       @Override
282       public void onClick(View v) {
283         Log.i("eirik", "clicked yes for authentication");
284         removeDialog(AUTHENTICATION_DIALOG);
285         sendAuthReponse(true, null);
286         showDialog(AUTHENTICATION_CONFIRMATION_DIALOG);
287       }
```

```
288
289     });
290
291     Button noButton = (Button) dialog
292         .findViewById(R.id.authenticate_no_button);
293     noButton.setOnClickListener(new OnClickListener() {
294
295       @Override
296       public void onClick(View v) {
297         perfromCancelAction();
298         sendAuthReponse(false, "cancelled");
299         Log.i("eirik", "Clicked no button");
300       }
301     });
302
303   }
304
305
306 }
```

MOpenIdModeActivation.java

```
001 package no.ntnu.item.mopenid.app;
002
003 import static no.ntnu.item.mopenid.app.AbstractActivation.
CANCEL_LOGIN_DIALOG;
004 import no.ntnu.item.mopenid.util.Utils;
005 import android.app.ProgressDialog;
006 import android.os.AsyncTask;
007 import android.os.Bundle;
008 import android.util.Log;
009 import android.view.View;
010 import android.view.View.OnClickListener;
011 import android.widget.Button;
012 import android.widget.TextView;
013
014 public class MOpenIdModeActivation extends AbstractActivation {
015
016
017   protected void onCreate(Bundle savedInstanceState) {
018     super.onCreate(savedInstanceState);
019     Bundle extras = getIntent().getExtras();
020     identifier = (String) extras.getString("id");
021   }
022
023   @Override
024   protected void setUpCancelButtonBehavior() {
025     Button cancelButton = (Button) findViewById(R.
id.cancelButton);
026     cancelButton.setOnClickListener(new OnClickListener() {
027
028       @Override
029       public void onClick(View v) {
030         Log.i("eirik", "Cancel button clicked");
031
032         new CancelButtonWithDisassociationTask().
execute(identifier);
033
034       }
035
036     });
```

```
037
038   }
039
040   private class CancelButtonWithDisassociationTask extends
041       AsyncTask<String, Void, Boolean> {
042     private String identifier = null;
043
044     @Override
045     protected void onPreExecute() {
046       removeDialog(currentShowingDialogId);
047       showDialog(CANCEL_LOGIN_DIALOG);
048     }
049
050     protected Boolean doInBackground(String... identifiers) {
051       identifier = identifiers[0];
052       return Utils.disassociateOpenID(identifier);
053     }
054
055     protected void onPostExecute(Boolean
disassociationSuccessful) {
056
057       if (disassociationSuccessful.booleanValue()) {
058         Log.i("eirik", "successful disassociation");
059         removeDialog(CANCEL_LOGIN_DIALOG);
060         setResult(SelectMode.RESULT_AUTHENTICATION_
CANCELLATION);
061         finish();
062       } else {
063         Log.i("eirik", "unsuccessful disassociation");
064         removeDialog(CANCEL_LOGIN_DIALOG);
065       }
066
067     }
068
069   }
070
071   @Override
072   protected void onPause() {
073     super.onPause();
074     Utils.disassociateOpenID(identifier);
075   }
076
077
078   @Override
079   protected void onResume() {
080     showDialog(SETUP_PROGRESS_DIALOG);
081     new ActivateIdentityProviderAndLinkWithmOpenIdTask().
execute();
082     super.onResume();
083   }
084
085   private class AuthenticationFinishedWithDisassociationTask
extends
086       CancelButtonWithDisassociationTask {
087
088     @Override
089     protected void onPreExecute() {
090       removeDialog(AUTHENTICATION_CONFIRMATION_DIALOG);
091     }
092
093     protected void onPostExecute(Boolean
disassociationSuccessful) {
094       if (disassociationSuccessful.booleanValue()) {
095         Log.i("eirik", "successful disassociation");
096         setResult(SelectMode.RESULT_AUTHENTICATION_SUCCESSFUL);
```

```
097         removeDialog(CANCEL_LOGIN_DIALOG);
098         finish();
099       } else {
100         Log.i("eirik", "unsuccessful disassociation");
101       }
102
103     }
104   }
105
106   @Override
107   protected void authenticationFinished() {
108     new AuthenticationFinishedWithDisassociationTask().
execute(identifier);
109
110   }
111
112   private class ActivateIdentityProviderAndLinkWithmOpenIdTask
extends
113       AsyncTask<Void, Void, Void> {
114
115     @Override
116     protected void onPreExecute() {
117       progressDialog.setProgress(0);
118     }
119
120     @Override
121     protected Void doInBackground(Void... params) {
122
123       startLocalOpenIDProvider();
124       progressDialog.setProgress(40);
125       connectToAndRedirectmOpenIDServerToHere();
126       progressDialog.setProgress(100);
127       return null;
128     }
129
130     @Override
131     protected void onPostExecute(Void result) {
132       setAllTextViewsToSuccessful();
133       showOpenIDURLAndCancelButton();
134       removeDialog(AbstractActivation.SETUP_PROGRESS_DIALOG);
135       linkedWithmOpenID = true;
136     }
137
138   }
139
140
141   protected void connectToAndRedirectmOpenIDServerToHere() {
142     linkedWithmOpenID = Utils.linkGivenmOpenIDtoHere(identifier);
143
144   }
145
146   protected void setupProgressDialog() {
147     progressDialog.setProgressStyle(ProgressDialog.STYLE_
HORIZONTAL);
148     progressDialog
149         .setMessage("Activating Identity Provider and linking
with mOpenID server, please wait...");
150     progressDialog.setCancelable(false);
151     progressDialog.setCanceledOnTouchOutside(false);
152   }
153
154   @Override
155   protected void perfromCancelAction() {
156     new CancelButtonWithDisassociationTask().execute(identifier);
157
```

```
158   }
159
160   @Override
161   protected void showOpenIDURLAndCancelButton() {
162     TextView usageInformationTextView = (TextView)
findViewById(R.id.usage_instructions), openidURLTextView = (TextView)
findViewById(R.id.openid_to_use);
163     usageInformationTextView.setVisibility(View.VISIBLE);
164     openidURLTextView.setText(Utils.MOPENID_URL + identifier);
165     openidURLTextView.setVisibility(View.VISIBLE);
166
167     Button cancelButton = (Button) findViewById(R.
id.cancelButton);
168     cancelButton.setVisibility(View.VISIBLE);
169
170   }
171 }
```

StaticIpModeActivation.java

```
001 package no.ntnu.item.mopenid.app;
002
003 import no.ntnu.item.mopenid.util.Utils;
004 import android.content.BroadcastReceiver;
005 import android.content.Context;
006 import android.content.Intent;
007 import android.content.IntentFilter;
008 import android.os.Bundle;
009 import android.util.Log;
010 import android.view.View;
011 import android.view.View.OnClickListener;
012 import android.widget.Button;
013 import android.widget.TextView;
014
015 public class StaticIpModeActivation extends AbstractActivation {
016
017   private static final String INTENT_ACTION_SET_STATIC_IP_MODE =
"no.ntnu.item.mopenid.ACTION_STATIC_IP_MODE";
018   private static final String INTENT_ACTION_SET_STATIC_IP_MODE_
REQUEST = "no.ntnu.item.mopenid.ACTION_STATIC_IP_MODE_REQUEST";
019   private BroadcastReceiver receiver;
020
021
022   public void onCreate(Bundle savedInstanceState) {
023     super.onCreate(savedInstanceState);
024     identifier = Utils.getLocalEndpointURLAsString();
025
026   }
027
028   @Override
029   public void onStart() {
030     super.onStart();
031     registerModeResolverListener();
032   }
033
034   private void registerModeResolverListener() {
035     receiver = new BroadcastReceiver() {
036
037       @Override
038       public void onReceive(Context context, Intent intent) {
039         sendIntentToServlet(true);
```

```
040       }
041     };
042
043     registerReceiver(receiver, new IntentFilter(INTENT_ACTION_
SET_STATIC_IP_MODE_REQUEST));
044
045   }
046
047   private void sendIntentToServlet(boolean isStatic) {
048     Intent intent = new Intent(INTENT_ACTION_SET_STATIC_IP_
MODE);
049     intent.putExtra("staticMode", isStatic);
050     sendBroadcast(intent);
051   }
052
053   @Override
054   protected void onResume() {
055     super.onResume();
056     sendIntentToServlet(true);
057     setAllTextViewsToSuccessful();
058     showOpenIDURLAndCancelButton();
059   }
060
061   @Override
062   protected void onPause() {
063     super.onPause();
064     unregisterReceiver(receiver);
065     sendIntentToServlet(false);
066
067   }
068
069   @Override
070   protected void setUpCancelButtonBehavior() {
071     Button cancelButton = (Button) findViewById(R.
id.cancelButton);
072     cancelButton.setOnClickListener(new OnClickListener() {
073
074       @Override
075       public void onClick(View v) {
076         Log.i("eirik", "Cancel button clicked");
077         perfromCancelAction();
078       }
079     });
080
081   }
082
083   @Override
084   protected void setupProgressDialog() {
085     // Do nothing
086
087   }
088
089   @Override
090   protected void perfromCancelAction() {
091     removeDialog(currentShowingDialogId);
092     setResult(SelectMode.RESULT_AUTHENTICATION_CANCELLATION);
093     finish();
094
095   }
096
097   @Override
098   protected void authenticationFinished() {
099     removeDialog(AUTHENTICATION_CONFIRMATION_DIALOG);
100     setResult(SelectMode.RESULT_AUTHENTICATION_SUCCESSFUL);
101     finish();
```

```
102   }
103
104   @Override
105   protected void showOpenIDURLAndCancelButton() {
106     TextView usageInformationTextView = (TextView)
findViewById(R.id.usage_instructions), openidURLTextView = (TextView)
findViewById(R.id.openid_to_use);
107     usageInformationTextView.setVisibility(View.VISIBLE);
108     openidURLTextView.setText(Utils.
getLocalEndpointURLAsString());
109     openidURLTextView.setVisibility(View.VISIBLE);
110     Button cancelButton = (Button) findViewById(R.
id.cancelButton);
111     cancelButton.setVisibility(View.VISIBLE);
112   }
113
114
115 }
```

SelectMode.java

```
001 package no.ntnu.item.mopenid.app;
002
003 import java.io.File;
004 import java.io.FileNotFoundException;
005 import java.net.URISyntaxException;
006
007 import no.ntnu.item.mopenid.util.BackupUtils;
008 import no.ntnu.item.mopenid.util.CryptoSuite;
009 import no.ntnu.item.mopenid.util.Utils;
010 import android.app.Activity;
011 import android.app.AlertDialog;
012 import android.app.Dialog;
013 import android.content.BroadcastReceiver;
014 import android.content.Context;
015 import android.content.DialogInterface;
016 import android.content.Intent;
017 import android.content.IntentFilter;
018 import android.net.ConnectivityManager;
019 import android.net.NetworkInfo;
020 import android.net.Uri;
021 import android.os.AsyncTask;
022 import android.os.Bundle;
023 import android.os.Handler;
024 import android.os.Message;
025 import android.text.Editable;
026 import android.util.Log;
027 import android.view.Menu;
028 import android.view.MenuInflater;
029 import android.view.MenuItem;
030 import android.view.View;
031 import android.view.View.OnClickListener;
032 import android.view.WindowManager;
033 import android.widget.Button;
034 import android.widget.EditText;
035 import android.widget.TextView;
036
037 public class SelectMode extends Activity {
038
039   private static final int DIALOG_HELP = 0;
040   private static final int DIALOG_NETWORK_CONFIGURATION = 1;
041   private static final int DIALOG_ENTER_DECRYPTION_KEY = 2;
042   public static final int RESULT_AUTHENTICATION_CANCELLATION =
```

```
10;
043   public static final int RESULT_AUTHENTICATION_SUCCESSFUL = 11;
044   private BroadcastReceiver receiver;
045   private Dialog currentShowingDialog;
046   private int currentShowingDialogId = -1;
047   private boolean receiverRegistered = false;
048   private boolean firstStartup = true;
049   private boolean isWaitingForStaticIPModeActivity = false,
050       isWaitingFormOpenIdActivity = false;
051   private String decryptionKey = null, encryptionKey = null;
052   private boolean wifiNetworkReady = false;
053   private boolean mobileNetworkReady = false;
054   private int importMode = -1;
055   private int numberOfNewIdsAdded;
056   private String importUrl;
057   private boolean avoidNotSupportedScreen = false;
058   private InteruptHandler handler;
059
060   @Override
061   public void onCreate(Bundle savedInstanceState) {
062     super.onCreate(savedInstanceState);
063
064     setContentView(R.layout.select_mode);
065     setTitle(R.string.app_full_name);
066     Utils.setApplicationContext(getApplicationContext());
067
068     setupButtons();
069
070     Utils.saveInitialWifiState(getApplicationContext());
071
072   }
073
074   private void setupButtons() {
075
076     Button staticIpButton = (Button) findViewById(R.
id.staticIpButton);
077     staticIpButton.setOnClickListener(new OnClickListener() {
078
079       // @Override
080       public void onClick(View v) {
081         if (!Utils.isPublicAndConnectedAlready(getApplicationCon
text())
082             && !Utils.DEBUGMODE) {
083           launchConfigurationSetupReceiverAndShowDialog();
084           isWaitingForStaticIPModeActivity = true;
085           setupInteruptActionIfTooLongTimeHavePassed();
086
087         } else {
088           Utils.startIJettyService();
089           Intent startActivityIntent = new Intent(v.
getContext(),
090               StaticIpModeActivation.class);
091           startActivityForResult(startActivityIntent, 0);
092
093         }
094       }
095     });
096
097     Button mopenidButton = (Button) findViewById(R.
id.mopenidButton);
098     mopenidButton.setOnClickListener(new OnClickListener() {
099
100       public void onClick(View v) {
101         if (!Utils.isPublicAndConnectedAlready(getApplicationCon
text())
```

```
102             && !Utils.DEBUGMODE) {
103           launchConfigurationSetupReceiverAndShowDialog();
104           setupInteruptActionIfTooLongTimeHavePassed();
105           isWaitingFormOpenIdActivity = true;
106
107         } else {
108           Utils.startIJettyService();
109           Intent startActivityIntent = new Intent(v.
getContext(),
110               SelectMOpenIDMode.class);
111           startActivityForResult(startActivityIntent, 0);
112         }
113
114       }
115     });
116
117     Button helpButton = (Button) findViewById(R.id.button_help);
118     helpButton.setOnClickListener(new OnClickListener() {
119
120       public void onClick(View v) {
121         showDialog(DIALOG_HELP);
122       }
123     });
124
125     Button _exitButton = (Button) findViewById(R.id.button_exit);
126     _exitButton.setOnClickListener(new OnClickListener() {
127
128       public void onClick(View v) {
129         finish();
130       }
131     });
132   }
133
134   @Override
135   protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
136     super.onActivityResult(requestCode, resultCode, data);
137
138     if (resultCode == RESULT_AUTHENTICATION_SUCCESSFUL
139         || resultCode == RESULT_AUTHENTICATION_CANCELLATION) {
140       avoidNotSupportedScreen = true;
141       // go back to initial phone settings
142       if (!Utils.DEBUGMODE) {
143         Utils.restoreApnSettingToNormal(getApplicationConte
xt());
144         Utils.resetToInitialWifiState(getApplicationContext());
145         wifiNetworkReady = mobileNetworkReady = false;
146         Log.i("eirik", "Initial settings reset successful");
147
148       }
149       Utils.stopIJettyService();
150     }
151     Log.i("eirik", "in onActivityResult(). resultcode = " +
resultCode);
152   }
153
154   protected void onResume() {
155     super.onResume();
156     setupButtons();
157     handler = new InteruptHandler();
158
159     if (firstStartup && !Utils.DEBUGMODE) {
160       Log.d("eirik", "First startup TRUE");
161       if (!Utils.isPublicAndConnectedAlready(getApplicationCont
ext())
```

```
162           && !Utils.DEBUGMODE) {
163         if (Utils.apnSettingsSupported()) {
164           showDialog(DIALOG_NETWORK_CONFIGURATION);
165           setupPublicMobileConnectedReceiver();
166           setupInteruptActionIfTooLongTimeHavePassed();
167           new ConfigureNetworkReachabilityTask().execute();
168         } else {
169           showDialog(R.layout.dialog_not_supported);
170         }
171
172       }
173       firstStartup = false;
174       Log.i("eirik", "performed first startup");
175     } else if (!Utils.apnSettingsSupported() &&
!avoidNotSupportedScreen) {
176       showDialog(R.layout.dialog_not_supported);
177     } else if (Utils.isPublicAndConnectedAlready(getApplicationC
ontext())
178         && !Utils.DEBUGMODE) {
179       if (currentShowingDialogId == DIALOG_NETWORK_
CONFIGURATION) {
180         removeDialog(DIALOG_NETWORK_CONFIGURATION);
181         currentShowingDialogId = -1;
182         currentShowingDialog = null;
183       }
184     }
185
186   }
187
188   private class InteruptHandler extends Handler {
189
190     @Override
191     public void handleMessage(Message msg) {
192       if(!(mobileNetworkReady && wifiNetworkReady)) {
193         SelectMode.this.removeDialog(currentShowingDialogId);
194         SelectMode.this.showDialog(R.layout.dialog_mobile_not_
enabled);
195       }
196
197     }
198
199     public void delay(long delayMillis) {
200
201       this.removeMessages(0);
202
203       sendMessageDelayed(obtainMessage(0), delayMillis);
204
205     }
206
207   };
208
209   private void setupInteruptActionIfTooLongTimeHavePassed() {
210     if (handler == null) {
211       handler = new InteruptHandler();
212     }
213     int secondsDelay = 18;
214     handler.delay(1000*secondsDelay);
215   }
216
217   public boolean onSearchRequested() {
218     return false;
219   }
220
221   private void setupPublicMobileConnectedReceiver() {
222     receiver = (Utils.DEBUGMODE) ? null : new
```

```
BroadcastReceiver() {
223
224       @Override
225       public void onReceive(Context context, Intent intent) {
226
227         NetworkInfo networkInfo = (NetworkInfo) intent
228             .getParcelableExtra("networkInfo");
229         Log.i("apn", "received connectivity change: " +
networkInfo);
230         if (networkInfo.getType() == ConnectivityManager.TYPE_
MOBILE
231             && networkInfo.isConnected()) {
232           if (networkInfo.getExtraInfo().contains(".public")) {
233             mobileNetworkReady = true;
234             Log.d("apn", "mobileNetworkReady = true");
235           }
236         }
237
238         if (networkInfo.getType() == ConnectivityManager.TYPE_
WIFI) {
239           if (!networkInfo.isConnectedOrConnecting()) {
240             Log.d("apn", "wifiNetworkReady = true");
241             wifiNetworkReady = true;
242           }
243         }
244
245         if (!Utils.isWifiEnabled()) {
246           Log.d("apn", "wifiNetworkReady = true");
247           wifiNetworkReady = true;
248         }
249
250         if (wifiNetworkReady
251             && mobileNetworkReady
252             && currentShowingDialog != null
253             && currentShowingDialog.isShowing()
254             && currentShowingDialogId == DIALOG_NETWORK_
CONFIGURATION) {
255           performButtonAction();
256         }
257
258       }
259     };
260     registerReceiver(receiver, new IntentFilter(
261         "android.net.conn.CONNECTIVITY_CHANGE"));
262     Log.i("eirik", "set up receiver");
263     receiverRegistered = true;
264   }
265
266   protected void onPause() {
267     super.onPause();
268     Log.i("eirik", "onPause SelectMode");
269   }
270
271   protected void onStop() {
272     super.onStop();
273     if (!Utils.DEBUGMODE && receiverRegistered) {
274       unregisterReceiver(receiver);
275       receiver = null;
276       receiverRegistered = false;
277       Log.i("eirik",
278           "in onStop() in SelectMode. performed unregistration
of receiver");
279     }
280   }
281
```

```
282   protected void onDestroy() {
283     super.onDestroy();
284     if (!Utils.DEBUGMODE) {
285       Utils.restoreApnSettingToNormal(getApplicationContext());
286       Utils.resetToInitialWifiState(getApplicationContext());
287       firstStartup = true;
288     }
289     Utils.stopIJettyService();
290     Log.i("eirik", "onDestroy called in selectMode");
291   }
292
293   protected Dialog onCreateDialog(int id) {
294     Dialog dialog;
295
296     switch (id) {
297     case DIALOG_HELP:
298       dialog = new Dialog(this);
299
300       dialog.setContentView(R.layout.popup_help);
301       dialog.setTitle("mOpenID vs Static IP mode");
302       dialog.setCanceledOnTouchOutside(true);
303
304       Button okButton = (Button) dialog.findViewById(R.id.help_
ok_button);
305       okButton.setOnClickListener(new OnClickListener() {
306
307         public void onClick(View v) {
308           removeDialog(DIALOG_HELP);
309         }
310       });
311       break;
312
313     case R.layout.dialog_mobile_not_enabled:
314       dialog = new Dialog(this);
315       setupDialogMobileNotEnabled(dialog);
316       break;
317
318     case DIALOG_NETWORK_CONFIGURATION:
319       dialog = new Dialog(this) {
320         public boolean onSearchRequested() {
321           return false;
322         }
323       };
324
325       dialog.setContentView(R.layout.popup_network_is_
configurating);
326       dialog.setTitle("Please wait");
327       dialog.setCanceledOnTouchOutside(false);
328       dialog.setCancelable(false);
329       dialog.onSearchRequested();
330
331       break;
332
333     case DIALOG_ENTER_DECRYPTION_KEY:
334       dialog = new Dialog(this);
335       setupDecryptionKeyDialog(dialog);
336       break;
337
338     case R.layout.dialog_pasteview:
339       dialog = new Dialog(this);
340
341       setupPasteViewDialog(dialog);
342       removeDialog(DIALOG_ENTER_DECRYPTION_KEY);
343       break;
344
```

```
345     case R.layout.dialog_encryption_key:
346       dialog = new Dialog(this);
347       setupEncryptionKeyDialog(dialog);
348       break;
349
350     case R.layout.dialog_import_confirmation:
351       dialog = new Dialog(this);
352       setupImportConfirmationDialog(dialog);
353       removeDialog(R.layout.dialog_url);
354       break;
355
356     case R.layout.dialog_url:
357       dialog = new Dialog(this);
358       setupUrlDialog(dialog);
359       removeDialog(DIALOG_ENTER_DECRYPTION_KEY);
360       break;
361
362     case R.layout.dialog_not_supported:
363       dialog = new Dialog(this);
364       setupNotSupportedDialog(dialog);
365       break;
366
367     default:
368       dialog = null;
369     }
370     currentShowingDialog = dialog;
371     currentShowingDialogId = id;
372     return dialog;
373   }
374
375   private void setupDialogMobileNotEnabled(Dialog dialog) {
376     dialog.setContentView(R.layout.dialog_mobile_not_enabled);
377     dialog.setTitle("Mobile network failure");
378     dialog.setCanceledOnTouchOutside(false);
379     dialog.setCancelable(false);
380
381     Button exitButton = (Button) dialog.findViewById(R.id.btn_
exit);
382     exitButton.setOnClickListener(new OnClickListener() {
383
384       public void onClick(View v) {
385         finish();
386       }
387     });
388
389   }
390
391   private void setupNotSupportedDialog(Dialog dialog) {
392     dialog.setContentView(R.layout.dialog_not_supported);
393     dialog.setCanceledOnTouchOutside(false);
394     dialog.setCancelable(false);
395     dialog.setTitle("Not supported");
396     final TextView infoText = (TextView) dialog
397         .findViewById(R.id.not_supported_info_text);
398     String currentApnName = Utils.getCurrentApnName();
399     infoText.setText(infoText.getText().toString()
400         .replaceFirst("%s", currentApnName));
401
402     Button exitButton = (Button) dialog.findViewById(R.id.btn_
exit);
403     exitButton.setOnClickListener(new OnClickListener() {
404
405       @Override
406       public void onClick(View v) {
407         finish();
```

```
408       }
409     });
410
411   }
412
413   private void setupUrlDialog(final Dialog dialog) {
414     dialog.setContentView(R.layout.dialog_url);
415     dialog.setCanceledOnTouchOutside(true);
416     dialog.setTitle("Import from URL");
417
418     final EditText urlTextView = (EditText) dialog
419         .findViewById(R.id.url_encrypted_ids);
420     Button importButton = (Button) dialog.findViewById(R.
id.button_import);
421     importButton.setOnClickListener(new OnClickListener() {
422
423       @Override
424       public void onClick(View v) {
425         importUrl = urlTextView.getText().toString();
426         Uri uri = Uri.parse(importUrl);
427         new DownloadEncryptedIdsTask().execute(uri);
428       }
429     });
430
431     // set the keyboard to automatically show
432     urlTextView.setOnFocusChangeListener(new View.
OnFocusChangeListener() {
433       public void onFocusChange(View v, boolean hasFocus) {
434         if (hasFocus) {
435           dialog.getWindow()
436               .setSoftInputMode(
437                   WindowManager.LayoutParams.SOFT_INPUT_STATE_
ALWAYS_VISIBLE);
438         }
439       }
440     });
441   }
442
443   private void setupImportConfirmationDialog(Dialog dialog) {
444     dialog.setContentView(R.layout.dialog_import_confirmation);
445     dialog.setCanceledOnTouchOutside(true);
446     dialog.setTitle("Import successful");
447     TextView informationTextView = (TextView) dialog
448         .findViewById(R.id.import_information_text);
449
450     String informationText = informationTextView.getText().
toString();
451     String updatedInformationText = informationText.
replaceFirst("%d",
452         Integer.toString(numberOfNewIdsAdded));
453     informationTextView.setText(updatedInformationText);
454     Log.d("eirik", "updated text: " + updatedInformationText);
455
456     Button okayButton = (Button) dialog.findViewById(R.id.button_
okay);
457     okayButton.setOnClickListener(new OnClickListener() {
458
459       @Override
460       public void onClick(View v) {
461         dismissDialog(R.layout.dialog_import_confirmation);
462       }
463     });
464
465   }
466
```

```
467   private void setupPasteViewDialog(Dialog dialog) {
468     dialog.setContentView(R.layout.dialog_pasteview);
469     dialog.setCanceledOnTouchOutside(true);
470     dialog.setTitle(R.string.dialog_pasteview_title);
471
472     final EditText encryptedTextView = (EditText) dialog
473         .findViewById(R.id.pastewindow);
474
475     Button doImportButton = (Button) dialog
476         .findViewById(R.id.button_import_do);
477     doImportButton.setOnClickListener(new OnClickListener() {
478
479       @Override
480       public void onClick(View view) {
481         String encryptedText = encryptedTextView.getText().
toString();
482         importIdsFromEncryptedString(encryptedText);
483       }
484     });
485
486   }
487
488   private void setupDecryptionKeyDialog(final Dialog dialog) {
489     dialog.setContentView(R.layout.dialog_decryption_key);
490     dialog.setTitle(R.string.provide_key);
491     dialog.setCanceledOnTouchOutside(true);
492     final EditText keyView = (EditText) dialog
493         .findViewById(R.id.decryption_key);
494     Button importButton = (Button) dialog.findViewById(R.
id.button_import);
495
496     // set the keyboard to automatically show
497     keyView.setOnFocusChangeListener(new View.
OnFocusChangeListener() {
498       public void onFocusChange(View v, boolean hasFocus) {
499         if (hasFocus) {
500           dialog.getWindow()
501               .setSoftInputMode(
502                   WindowManager.LayoutParams.SOFT_INPUT_STATE_
ALWAYS_VISIBLE);
503         }
504       }
505     });
506
507     importButton.setOnClickListener(new OnClickListener() {
508
509       @Override
510       public void onClick(View view) {
511         decryptionKey = keyView.getText().toString();
512         if (importMode == R.id.from_edittextview)
513           showPasteViewWithDecryptionKey(keyView.getText());
514         else if (importMode == R.id.from_url)
515           showUrlViewWithDecryptionKey(keyView.getText());
516         else
517           return;
518       }
519     });
520   }
521
522   private void importIdsFromEncryptedString(String
encryptedText) {
523     String decryptedText = null;
524     try {
525       Log.d("eirik", "encrypted text: " + encryptedText);
526       decryptedText = CryptoSuite.decrypt(decryptionKey,
```

```
encryptedText);
527     } catch (Exception e) {
528       Log.d("eirik",
529           "exception happened when tried to decrypt (probably
wrong key): "
530               + e.getMessage());
531       dismissDialog(currentShowingDialogId);
532
533       final AlertDialog alertDialog = new AlertDialog.Builder(
534           SelectMode.this).create();
535       alertDialog.setIcon(R.drawable.ic_delete);
536       alertDialog.setTitle("Message");
537       alertDialog
538           .setMessage("An error occured when trying to decrypt.
Is you decryption key correct?");
539       alertDialog.setButton("Dismiss dialog",
540           new DialogInterface.OnClickListener() {
541             public void onClick(DialogInterface dialog, int
which) {
542               alertDialog.dismiss();
543             }
544           });
545       alertDialog.show();
546
547       e.printStackTrace();
548     }
549     if (Utils.isCorrectDecryptionKey(decryptedText)) {
550       Log.d("eirik", "correct decryption key");
551       numberOfNewIdsAdded = BackupUtils
552           .addNewImportedIdsFromDecryptedtext(decryptedText);
553       dismissDialog(currentShowingDialogId);
554       showDialog(R.layout.dialog_import_confirmation);
555     } else {
556       Log.d("eirik", "wrong decryptionkey");
557     }
558
559     Log.d("eirik", "decrypted text: " + decryptedText);
560   }
561
562   private void setupEncryptionKeyDialog(final Dialog dialog) {
563     dialog.setContentView(R.layout.dialog_encryption_key);
564     dialog.setTitle(R.string.provide_enc_key);
565     dialog.setCanceledOnTouchOutside(true);
566     final EditText keyView = (EditText) dialog
567         .findViewById(R.id.encryption_key);
568     Button importButton = (Button) dialog.findViewById(R.
id.button_export);
569
570     // set the keyboard to automatically show
571     keyView.setOnFocusChangeListener(new View.
OnFocusChangeListener() {
572       public void onFocusChange(View v, boolean hasFocus) {
573         if (hasFocus) {
574           dialog.getWindow()
575               .setSoftInputMode(
576                   WindowManager.LayoutParams.SOFT_INPUT_STATE_
ALWAYS_VISIBLE);
577         }
578       }
579     });
580
581     importButton.setOnClickListener(new OnClickListener() {
582
583       @Override
584       public void onClick(View view) {
```

```
585         dismissDialog(R.layout.dialog_encryption_key);
586         encryptionKey = keyView.getText().toString();
587         encryptAndExportIds(encryptionKey);
588       }
589     });
590   }
591
592   protected void showUrlViewWithDecryptionKey(Editable text) {
593     removeDialog(R.id.layout_decryption_key);
594     showDialog(R.layout.dialog_url);
595   }
596
597   protected void showPasteViewWithDecryptionKey(Editable text) {
598     removeDialog(R.id.layout_decryption_key);
599     showDialog(R.layout.dialog_pasteview);
600   }
601
602   private void launchConfigurationSetupReceiverAndShowDialog() {
603     showDialog(DIALOG_NETWORK_CONFIGURATION);
604     if (!receiverRegistered) {
605       setupPublicMobileConnectedReceiver();
606       Log.i("eirik", "set up receiver in the second place");
607       receiverRegistered = true;
608     }
609     new ConfigureNetworkReachabilityTask().execute();
610
611   }
612
613   private void performButtonAction() {
614     Log.d("eirik", "in performButtonAction() method");
615     removeDialog(DIALOG_NETWORK_CONFIGURATION);
616     currentShowingDialog = null;
617     currentShowingDialogId = -1;
618
619     if (isWaitingFormOpenIdActivity ||
isWaitingForStaticIPModeActivity) {
620       Utils.startIJettyService();
621     }
622     if (isWaitingFormOpenIdActivity) {
623       isWaitingFormOpenIdActivity = false;
624       Intent startActivityIntent = new
Intent(getApplicationContext(),
625           SelectMOpenIDMode.class);
626       startActivityForResult(startActivityIntent, 0);
627     } else if (isWaitingForStaticIPModeActivity) {
628       isWaitingForStaticIPModeActivity = false;
629       Intent startActivityIntent = new
Intent(getApplicationContext(),
630           StaticIpModeActivation.class);
631       startActivityForResult(startActivityIntent, 0);
632     }
633   }
634
635   private class ConfigureNetworkReachabilityTask extends
636       AsyncTask<Void, Void, Void> {
637
638     @Override
639     protected Void doInBackground(Void... params) {
640
641       changeAPNSettingsToObtainPublicIPAddress();
642       Utils.disableWifi(getApplicationContext());
643       return null;
644     }
645
646     private void changeAPNSettingsToObtainPublicIPAddress() {
```

```
647       Utils.saveCurrentAndSwitchToPublicAPN(getApplicationConte
xt());
648     }
649   }
650
651   @Override
652   public boolean onCreateOptionsMenu(Menu menu) {
653     MenuInflater inflater = getMenuInflater();
654     inflater.inflate(R.menu.select_mode_menu, menu);
655     return super.onCreateOptionsMenu(menu);
656
657   }
658
659   @Override
660   public boolean onOptionsItemSelected(MenuItem item) {
661     // Handle item selection
662     switch (item.getItemId()) {
663     case R.id.export_option:
664       showDialog(R.layout.dialog_encryption_key);
665       return true;
666
667     case R.id.import_option:
668       // showing submenu - do nothing
669       return true;
670
671     case R.id.from_edittextview:
672       Log.d("eirik", "show paste window");
673       importMode = R.id.from_edittextview;
674       showDialog(DIALOG_ENTER_DECRYPTION_KEY);
675       return true;
676
677     case R.id.from_url:
678       importMode = R.id.from_url;
679       showDialog(DIALOG_ENTER_DECRYPTION_KEY);
680       Log.d("eirik", "show url window");
681       return true;
682
683     default:
684       return super.onOptionsItemSelected(item);
685     }
686   }
687
688   private void encryptAndExportIds(String key) {
689     String savedFilename = BackupUtils
690         .encryptOpenIdsWithPasswordAndWriteToSDCard(key);
691
692     if (savedFilename != null) {
693       Intent intent = new Intent(Intent.ACTION_SEND);
694       intent.putExtra(Intent.EXTRA_STREAM, Uri.fromFile(new
File(
695           "/mnt/sdcard/mopenid/", savedFilename)));
696       intent.setType("text/plain");
697       // intent.get
698       Intent actionChooserIntent = Intent.createChooser(intent,
699           "Export to:");
700       startActivity(actionChooserIntent);
701     } else {
702
703     }
704
705   }
706
707   private class DownloadEncryptedIdsTask extends AsyncTask<Uri,
Void, String> {
708
```

```
709     protected String doInBackground(Uri... uris) {
710
711       try {
712         return Utils.fetchStringFromUri(uris[0]);
713       } catch (URISyntaxException e) {
714         Log.d("eirik", "URISyntaxException when trying to parse
file");
715         e.printStackTrace();
716         return null;
717       } catch (FileNotFoundException e) {
718         Log.d("eirik",
719             "FileNotFoundException when trying to parse file");
720         e.printStackTrace();
721         return null;
722       }
723     }
724
725     protected void onPostExecute(String result) {
726
727       Log.d("eirik", "got the encrypted string: " + result);
728       importIdsFromEncryptedString(result);
729
```

SelectMOpenIDMode.java

```
001 package no.ntnu.item.mopenid.app;
002
003 import java.util.Calendar;
004 import java.util.List;
005
006 import no.ntnu.item.mopenid.util.OpenID;
007 import no.ntnu.item.mopenid.util.Utils;
008 import android.app.AlertDialog;
009 import android.app.ListActivity;
010 import android.content.BroadcastReceiver;
011 import android.content.Context;
012 import android.content.DialogInterface;
013 import android.content.Intent;
014 import android.media.AudioRecord.OnRecordPositionUpdateListener;
015 import android.net.ConnectivityManager;
016 import android.net.NetworkInfo;
017 import android.os.Bundle;
018 import android.text.format.DateFormat;
019 import android.util.Log;
020 import android.view.ContextMenu;
021 import android.view.ContextMenu.ContextMenuInfo;
022 import android.view.LayoutInflater;
023 import android.view.Menu;
024 import android.view.MenuItem;
025 import android.view.View;
026 import android.view.ViewGroup;
027 import android.widget.AdapterView;
028 import android.widget.AdapterView.OnItemClickListener;
029 import android.widget.AdapterView.OnItemLongClickListener;
030 import android.widget.ArrayAdapter;
031 import android.widget.ListView;
032 import android.widget.TextView;
033 import android.widget.Toast;
034
035 public class SelectMOpenIDMode extends ListActivity {
036
```

```
037   public static final String __PORT = "org.mortbay.ijetty.port";
038   public static final String __NIO = "org.mortbay.ijetty.nio";
039   public static final String __SSL = "org.mortbay.ijetty.ssl";
040 }
041   public static final String __CONSOLE_PWD = "org.mortbay.ijetty.
console";
042   public static final String __PORT_DEFAULT = "8080";
043   public static final boolean __NIO_DEFAULT = true;
044   public static final boolean __SSL_DEFAULT = false;
045
046   public static final String __CONSOLE_PWD_DEFAULT = "admin";
047
048   private ArrayAdapter<OpenID> adapter;
049   private OpenID idToDelete = null;
050
051   private static final int MENU_OPTION_DELETE_COMPLETELY = 1;
052   private static final int MENU_OPTION_DELETE_LOCAL_ONLY = 2;
053
054   public void onCreate(Bundle savedInstanceState) {
055     super.onCreate(savedInstanceState);
056     Utils.setApplicationContext(getApplicationContext());
057
058     setContentView(R.layout.select_mopenid);
059     setUpListBehavior();
060
061   }
062
063
064
065   private void setUpListBehavior() {
066     List<OpenID> linkIDs = Utils.getSavedOpenIDs();
067
068     if (linkIDs.size() <= Utils.MAX_NUMBER_OF_IDS)
069       linkIDs.add(new OpenID(">> Create new"));
070     adapter = new MyArrayAdapter<OpenID>(this, R.layout.id_list_
item,
071         linkIDs);
072     setListAdapter(adapter);
073
074     ListView lv = getListView();
075
076     lv.setTextFilterEnabled(true);
077
078     lv.setOnItemClickListener(new OnItemClickListener() {
079
080       public void onItemClick(AdapterView<?> parent, View view,
081           int position, long id) {
082         OpenID openid = (OpenID) parent.
getItemAtPosition(position);
083
084         if (!openid.isCreateNewButton()) {
085
086           Intent intent = new Intent(SelectMOpenIDMode.this,
087               MOpenIdModeActivation.class);
088           intent.putExtra("id", openid.getIdentifier());
089           intent.addFlags(Intent.FLAG_ACTIVITY_FORWARD_RESULT);
090           intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
091           startActivity(intent);
092
093         } else {
094           // open new activity if create new was pushed
095           Intent startActivityIntent = new Intent(view.
getContext(),
096               CreateNew.class);
097           startActivity(startActivityIntent);
```

```
098         }
099
100       }
101
102     });
103
104     registerForContextMenu(lv);
105   }
106
107   @Override
108   public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenuInfo menuInfo) {
109
110     AdapterView.AdapterContextMenuInfo info = (AdapterView.
AdapterContextMenuInfo)menuInfo;
111     List<OpenID> list = Utils.getSavedOpenIDs();
112     idToDelete = list.get(info.position);
113       menu.setHeaderTitle(idToDelete.getIdentifier());
114
115     menu.add(Menu.NONE, MENU_OPTION_DELETE_COMPLETELY , Menu.
NONE, "Delete completely");
116     menu.add(Menu.NONE, MENU_OPTION_DELETE_LOCAL_ONLY , Menu.
NONE, "Delete locally only");
117   }
118
119   @Override
120   public boolean onContextItemSelected(MenuItem item) {
121
122     switch (item.getItemId()) {
123     case MENU_OPTION_DELETE_COMPLETELY:
124       showDeleteAlertDialog(item.getItemId(), true);
125       break;
126
127     case MENU_OPTION_DELETE_LOCAL_ONLY:
128       showDeleteAlertDialog(item.getItemId(), false);
129
130       break;
131     default:
132       break;
133     }
134
135
136     return true;
137   }
138
139   private void showDeleteAlertDialog(final int itemId, final
boolean deleteRemotelyToo) {
140
141     new AlertDialog.Builder(SelectMOpenIDMode.this)
142     .setIcon(R.drawable.ic_delete)
143     .setTitle(R.string.are_you_sure)
144     .setMessage(R.string.delete_info_text)
145     .setPositiveButton(R.string.button_yes, new DialogInterface.
OnClickListener() {
146
147       @Override
148       public void onClick(DialogInterface dialog, int which) {
149         setUpListBehavior();
150          if(Utils.deleteId(idToDelete, deleteRemotelyToo)) {
151                 setUpListBehavior();
152                 Toast.makeText(getApplicationContext(),
(deleteRemotelyToo) ? R.string.deleted : R.string.deleted_locally
,Toast.LENGTH_SHORT).show();
153               } else {
154                 Toast.makeText(getApplicationContext(),
```

```
R.string.id_not_deleted ,Toast.LENGTH_SHORT).show();
155                 }
156
157               idToDelete = null;
158           }
159
160       })
161       .setNegativeButton(R.string.button_no, null)
162       .show();
163
164
165   }
166
167
168   @Override
169   protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
170       super.onActivityResult(requestCode, resultCode, data);
171       setResult(resultCode);
172       finish();
173
174   }
175
176   private class MyArrayAdapter<T> extends ArrayAdapter<T> {
177
178       public MyArrayAdapter(Context context, int
textViewResourceId,
179           List<T> objects) {
180         super(context, textViewResourceId, objects);
181
182       }
183
184       public View getView(int position, View convertView,
ViewGroup parent) {
185         View v = convertView;
186         if (v == null) {
187           LayoutInflater vi = (LayoutInflater)
getSystemService(Context.LAYOUT_INFLATER_SERVICE);
188           v = vi.inflate(R.layout.openid_list_item, null);
189         }
190
191         TextView twId = (TextView) v.findViewById(R.id.openid_
identifier);
192
193         TextView twLastUsed = (TextView) v.findViewById(R.
id.openid_lastused);
194         OpenID openID = (OpenID) getItem(position);
195
196         twId.setText(openID.getIdentifier());
197
198         if (!openID.getIdentifier().equals(">> Create new")) {
199           try {
200             Calendar cal = Utils.parseTimestamp(openID.
getLastUsed());
201             twLastUsed.setText("(last used "
202                 + DateFormat.format("dd.MM.yyyy kk:mm:ss", cal)
203                 + ")");
204           } catch (Exception e) {
205             twLastUsed.setText("(never been used before)");
206           }
207
208         } else {
209           twLastUsed.setText("");
210         }
211         return v;
```

```
212     }
213
214   }
215
216
217
218
219   private class ConnectionChangeReceiver extends
BroadcastReceiver {
220     @Override
221     public void onReceive(Context context, Intent intent) {
222       ConnectivityManager connectivityManager =
(ConnectivityManager) context
223           .getSystemService(Context.CONNECTIVITY_SERVICE);
224       NetworkInfo activeNetInfo = connectivityManager
225           .getActiveNetworkInfo();
226       NetworkInfo mobNetInfo = connectivityManager
227           .getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
228       if (activeNetInfo != null) {
229         Toast.makeText(context,
230             "Active Network Type : " + activeNetInfo.
getTypeName(),
231             Toast.LENGTH_SHORT).show();
232       }
233       if (mobNetInfo != null) {
234         Toast.makeText(context,
235             "Mobile Network Type : " + mobNetInfo.getTypeName(),
236             Toast.LENGTH_SHORT).show();
237       }
238     }
239   }
240 }
```

CreateNew.java

```
001 package no.ntnu.item.mopenid.app;
002
003
004 import no.ntnu.item.mopenid.util.Utils;
005
006 import org.json.JSONException;
007 import org.json.JSONObject;
008
009 import android.app.Activity;
010 import android.content.Intent;
011 import android.graphics.Color;
012 import android.os.AsyncTask;
013 import android.os.Bundle;
014 import android.text.Editable;
015 import android.text.TextWatcher;
016 import android.util.Log;
017 import android.view.View;
018 import android.view.View.OnClickListener;
019 import android.view.WindowManager;
020 import android.widget.Button;
021 import android.widget.EditText;
022 import android.widget.LinearLayout;
023 import android.widget.ProgressBar;
024 import android.widget.TextView;
025 import android.widget.Toast;
```

```
026
027
028
029 public class CreateNew extends Activity {
030
031   private final static int ID_MINIMUM_LENGTH = 5;
032
033   EditText editNewField ;
034   LinearLayout newidLayout;
035
036   @Override
037   public void onCreate(Bundle savedInstanceState) {
038     super.onCreate(savedInstanceState);
039
040     setContentView(R.layout.create_new);
041
042     newidLayout = (LinearLayout) findViewById(R.id.layout_newid);
043     editNewField = (EditText) findViewById(R.id.create_new_field);
044
045     TextView openIdURL = (TextView) findViewById(R.id.openid_
identifier_url);
046     openIdURL.setTextColor(Color.RED);
047     final Button checkAvailabilityButton =
setAvailabilityButtonBehavior();
048     checkAvailabilityButton.setClickable(false);
049     EditText editNewField = setEditTextBehavior(openIdURL,
050         checkAvailabilityButton);
051
052   }
053
054   private Button setAvailabilityButtonBehavior() {
055
056     final Button checkAvailabilityButton = (Button)
findViewById(R.id.check_availability_button);
057
058     checkAvailabilityButton.setOnClickListener(new
OnClickListener() {
059
060 //      @Override
061       public void onClick(View v) {
062         final String desiredOpenID = editNewField.getText().
toString();
063         if(desiredOpenID.length() >= ID_MINIMUM_LENGTH) {
064           CreateNew.this.setButtonOnClickBehavior((Button) v);
065         } else {
066           Toast.makeText(getApplicationContext(), "Minimum "+ID_
MINIMUM_LENGTH+" characters",
067               Toast.LENGTH_SHORT).show();
068         }
069
070       }
071     });
072
073     return checkAvailabilityButton;
074
075   }
076
077
078   private void setButtonOnClickBehavior(Button
checkAvailabilityButton) {
079     checkAvailabilityButton.setVisibility(View.GONE);
080     final ProgressBar progressBar = (ProgressBar) findViewById(R.
id.progressBar_checkid);
081     progressBar.setVisibility(View.VISIBLE);
082
```

```
083 //    final EditText editNewField = (EditText) findViewById(R.
id.create_new_field);
084     final String desiredOpenID = editNewField.getText().
toString();
085
086     new CheckAvailabilityTask().execute(desiredOpenID);
087     Log.i("eirik", "trying to fetch result");
088
089
090
091   }
092
093   private EditText setEditTextBehavior(final TextView openIdURL,
094       final Button checkAvailabilityButton) {
095
096     final EditText editNewField = (EditText) findViewById(R.
id.create_new_field);
097
098     // set the keyboard to automatically show
099     editNewField.setOnFocusChangeListener(new View.
OnFocusChangeListener() {
100       public void onFocusChange(View v, boolean hasFocus) {
101         if (hasFocus) {
102           getWindow().setSoftInputMode(WindowManager.
LayoutParams.SOFT_INPUT_STATE_ALWAYS_VISIBLE);
103         }
104       }
105     });
106
107
108
109     editNewField.addTextChangedListener(new TextWatcher() {
110
111       public void onTextChanged(CharSequence s, int start, int
before,
112           int count) {
113         CharSequence urlTextCurrent = openIdURL.getText();
114         checkAvailabilityButton.setText(R.string.check_
availability);
115         setAvailabilityButtonBehavior();
116
117         if (s.length() == 0) {
118           checkAvailabilityButton.setClickable(false);
119           openIdURL.setTextColor(Color.rgb(255, 0, 0)); // red
120           checkAvailabilityButton.setTextColor(Color.rgb(119,
119,
121               119));// grey
122         } else if (ID_MINIMUM_LENGTH > s.length()) {
123           openIdURL.setTextColor(Color.rgb(255, 215, 0)); //
orange
124           checkAvailabilityButton.setClickable(true);
125           checkAvailabilityButton.setTextColor(Color.rgb(119,
119,
126               119)); // grey
127         } else {
128           openIdURL.setTextColor(Color.rgb(255, 255, 0)); //
yellow
129           checkAvailabilityButton.setClickable(true);
130           checkAvailabilityButton.setTextColor(Color.rgb(0, 153,
0)); // green
131         }
132
133         if (count < before) {
134           // must remove one character
135           openIdURL.setText(urlTextCurrent.toString().
```

```
substring(0,
136                 openIdURL.length() - 1));
137           } else {
138             // must add
139             openIdURL.setText(urlTextCurrent.toString()
140                 + s.toString().substring(start));
141           }
142
143         }
144
145         public void beforeTextChanged(CharSequence s, int start,
int count,
146             int after) {
147
148         }
149
150         public void afterTextChanged(Editable s) {
151
152         }
153       });
154
155       return editNewField;
156     }
157
158     private class CheckAvailabilityTask extends AsyncTask<String,
Void, String> {
159       private static final String STATUS_OK = "no";
160       private static final String STATUS_INVALID = "invalid_id";
161       private static final String STATUS_TAKEN = "yes";
162       private String identifier = null;
163
164       protected String doInBackground(String... identifiers) {
165         identifier = identifiers[0];
166
167
168         return isOpenIDAvailable();
169       }
170
171       private String isOpenIDAvailable() {
172         return Utils.isAvailable(identifier);
173       }
174
175       protected void onPostExecute(String result) {
176         final ProgressBar progressBar = (ProgressBar)
findViewById(R.id.progressBar_checkid);
177         progressBar.setVisibility(View.GONE);
178         final Button checkAvailabilityButton = (Button)
findViewById(R.id.check_availability_button);
179         checkAvailabilityButton.setVisibility(View.VISIBLE);
180
181         try {
182           JSONObject response = new JSONObject(result);
183
184           if(response.get("message").equals(STATUS_OK)) {
185
186             checkAvailabilityButton.setText(R.string.button_
available);
187             TextView openIdURL = (TextView) findViewById(R.
id.openid_identifier_url);
188             openIdURL.setTextColor(Color.rgb(0, 153, 0));
189             checkAvailabilityButton.setTextColor(Color.rgb(0, 153,
0));
190             checkAvailabilityButton.setOnClickListener(new
OnClickListener() {
191
```

```
192             public void onClick(View v) {
193               editNewField.setEnabled(false);
194               editNewField.setFocusable(false);
195               checkAvailabilityButton.setVisibility(View.GONE);
196               progressBar.setVisibility(View.VISIBLE);
197
198               boolean registerSuccess = false;
199               registerSuccess = Utils.registerNewID(identifier);
200
201               if(registerSuccess) {
202                 Log.i("eirik", "insertions success");
203                 Intent selectmOpenIDActivity = new Intent(v.
getContext(), SelectMOpenIDMode.class);
204                 selectmOpenIDActivity.addFlags(Intent.FLAG_
ACTIVITY_CLEAR_TOP);
205                 startActivity(selectmOpenIDActivity);
206               } else {
207                 Log.i("eirik", "insertion failure");
208                 //TODO: handle error
209               }
210
211             }
212
213           });
214
215
216         } else if (response.get("message").equals(STATUS_
INVALID)) {
217           Log.i("eirik", "Identifier is invalid. Only certain
characters are allowed.");
218           Toast.makeText(getApplicationContext(), "Invalid
characters used.",
219               Toast.LENGTH_SHORT).show();
220         } else if (response.get("message").equals(STATUS_TAKEN))
{
221           TextView openIdURL = (TextView) findViewById(R.
id.openid_identifier_url);
222           openIdURL.setTextColor(Color.RED);
223           checkAvailabilityButton.setClickable(false);
224           checkAvailabilityButton.setTextColor(Color.RED);
225           checkAvailabilityButton.setText(R.string.button_
available_taken);
226
227         } else {
228           throw new JSONException("some weird (assumed) json
response received: "+ result);
229         }
230
231       } catch (JSONException e) {
232         e.printStackTrace();
233         Toast.makeText(getApplicationContext(), result,
234             Toast.LENGTH_SHORT).show();
235       }
236
237
238
239     }
240
241
242   }
243 }
```

BackupUtils.java

```
001 package no.ntnu.item.mopenid.util;
002
003 import java.io.File;
004 import java.io.FileWriter;
005 import java.io.IOException;
006 import java.util.ArrayList;
007 import java.util.Date;
008 import java.util.Iterator;
009 import java.util.List;
010
011 import android.content.ContentValues;
012 import android.os.Environment;
013 import android.text.format.DateFormat;
014 import android.util.Log;
015
016 public class BackupUtils {
017
018
019   public static final String ENC_FILE_NAME = "openids_enc_";
020   public static final String OKAY_FLAG = "iamdecrypted";
021
022   public static String encryptOpenIdsWithPasswordAndWriteToSDCar
d(String encryptionKey) {
023     boolean writeOk = false, encryptOk = false;
024     String savedFilename = null;
025     List<OpenID> savedOpenIDs = Utils.getSavedOpenIDs();
026
027     if(savedOpenIDs.size() == 0)
028       return null;
029
030     String unEncryptedString = createStringFromList(savedOpenI
Ds);
031     Log.d("eirik", "formatted openids ready for encryption:
"+unEncryptedString);
032
033     try {
034       String encryptedString = CryptoSuite.
encrypt(encryptionKey, unEncryptedString);
035       encryptOk = true;
036       String time = timestamp();
037       savedFilename = ENC_FILE_NAME + time +".txt";
038       writeOk = writeFileToSD(savedFilename, encryptedString);
039       Log.d("eirik", "Written to file: "+writeOk+ ". Encrypted:
"+encryptOk);
040
041     } catch (Exception e) {
042       // TODO Auto-generated catch block
043       e.printStackTrace();
044       Log.d("eirik", e.toString());
045     }
046
047     return savedFilename;
048   }
049
050   private static String timestamp() {
051     return DateFormat.format("dd.MM.yyyy-kkmmss", new Date()).
toString();
052
053   }
054
055   private static String createStringFromList(List<OpenID>
savedOpenIDs) {
056     StringBuilder result = new StringBuilder();
```

```
057     result.append(OKAY_FLAG+"\n");
058     for (OpenID openId : savedOpenIDs) {
059       result.append(openId.toFormattedString());
060       result.append("\n");
061     }
062     return result.toString();
063   }
064
065   private static boolean writeFileToSD(String sFileName, String
sBody) {
066     try {
067       File root = new File(Environment.
getExternalStorageDirectory(),
068           "mopenid");
069       if (!root.exists()) {
070         root.mkdirs();
071       }
072       File file = new File(root, sFileName);
073       FileWriter writer = new FileWriter(file);
074       writer.append(sBody);
075       writer.flush();
076       writer.close();
077     } catch (IOException e) {
078       e.printStackTrace();
079       return false;
080     }
081     return true;
082   }
083
084   public static int addNewImportedIdsFromDecryptedtext(String
decryptedText) {
085     String[] lines = decryptedText.split("\n");
086     ArrayList<OpenID> ids = new ArrayList<OpenID>();
087     for (int i = 1; i < lines.length; i++) {
088       String[] row = lines[i].split(";");
089       OpenID openIdAtCurrentRow = new OpenID(row[0]);
090       openIdAtCurrentRow.setLastUsed(row[1]);
091       openIdAtCurrentRow.setSecurityToken(row[2]);
092       openIdAtCurrentRow.setCreated(row[3]);
093       openIdAtCurrentRow.setDescription(row[4]);
094       ids.add(openIdAtCurrentRow);
095     }
096     int numberOfInsertedRows = 0;
097     for (Iterator iterator = ids.iterator(); iterator.
hasNext();) {
098       OpenID openID = (OpenID) iterator.next();
099
100       if(!Utils.idAlreadyExists(openID)) {
101         if(Utils.insertNewIdToDatabase(openID)) {
102           numberOfInsertedRows++;
103         }
104       }
105
106     }
107
108     return numberOfInsertedRows;
109   }
110
111
112 }
```

Utils.java

```
001 package no.ntnu.item.mopenid.util;
002
003 import java.io.BufferedReader;
004 import java.io.File;
005 import java.io.FileNotFoundException;
006 import java.io.IOException;
007 import java.io.InputStream;
008 import java.io.InputStreamReader;
009 import java.io.Reader;
010 import java.io.StringWriter;
011 import java.io.Writer;
012 import java.net.InetAddress;
013 import java.net.NetworkInterface;
014 import java.net.SocketException;
015 import java.net.URI;
016 import java.net.URISyntaxException;
017 import java.text.SimpleDateFormat;
018 import java.util.ArrayList;
019 import java.util.Calendar;
020 import java.util.Date;
021 import java.util.Enumeration;
022 import java.util.HashMap;
023 import java.util.List;
024 import java.util.Scanner;
025
026
027 import org.apache.http.client.ClientProtocolException;
028 import org.apache.http.client.HttpClient;
029 import org.apache.http.client.ResponseHandler;
030 import org.apache.http.client.methods.HttpGet;
031 import org.apache.http.impl.client.BasicResponseHandler;
032 import org.apache.http.impl.client.DefaultHttpClient;
033 import org.json.JSONException;
034 import org.json.JSONObject;
035
036 import android.content.ComponentName;
037 import android.content.ContentResolver;
038 import android.content.ContentValues;
039 import android.content.Context;
040 import android.content.Intent;
041 import android.content.SharedPreferences;
042 import android.database.Cursor;
043 import android.database.sqlite.SQLiteDatabase;
044 import android.net.ConnectivityManager;
045 import android.net.NetworkInfo;
046 import android.net.Uri;
047 import android.net.wifi.WifiManager;
048 import android.telephony.TelephonyManager;
049 import android.util.Log;
050
051 public class Utils {
052   public final static String MOPENID_WEBSERVICE_URL = "http://
mopenid.item.ntnu.no/mopenid_ws/"; // TODO:
053                                                    // make
054                                                    // work
055                                                    // for
056                                                    // https
057   public final static String MOPENID_URL = "https://mopenid.item.
ntnu.no/id/";
058   public final static int MAX_NUMBER_OF_IDS = 4;
059
060   public static final String __PORT = "org.mortbay.ijetty.port";
061   public static final String __NIO = "org.mortbay.ijetty.nio";
```

```
062   public static final String __SSL = "org.mortbay.ijetty.ssl";
063
064   public static final String __CONSOLE_PWD = "org.mortbay.ijetty.
console";
065   public static final String __PORT_DEFAULT = "8080";
066   public static final boolean __NIO_DEFAULT = true;
067   public static final boolean __SSL_DEFAULT = false;
068
069   public static final String __CONSOLE_PWD_DEFAULT = "admin";
070
071   private static final String JSON_RESPONSE_INSERTED =
"inserted";
072   private static final String JSON_RESPONSE_OK = "ok";
073   private static final int ACTION_EXISTS = 0;
074   private static final int ACTION_CREATE_NEW_IDENTIFIER = 1;
075   private static final int ACTION_DELETE = 2;
076   private static final int ACTION_ASSOCIATE_IDENTIFIER = 3;
077   private static final int ACTION_DISASSOCIATE_IDENTIFIER = 4;
078   private static final String __MOPENID_WS_NAME = "mopenid";
079   private static final String JSON_LINK_ASSOCIATED_OK = "Link
associated";
080   private static final String JSON_LINK_DISASSOCIATED_OK = "Link
disassociated";
081   private static final String APN_PREFS_NAME = "APN_Preferences";
082   private static SQLiteDatabase dbReadable, dbWriteable;
083   private static final Uri CONTENT_URI = Uri
084       .parse("content://telephony/carriers");
085   private static final Uri PREFERRED_APN_URI = Uri
086       .parse("content://telephony/carriers/preferapn");
087   private static final String APN_DEFAULT_ID_KEY = "apn_current_
id_key";
088   private static final String APN_DEFAULT_NAME_KEY = "apn_
current_name_key";
089   private static final String[] SUPPORTED_APNS = new String[]
{"internet"};
090
091   private static boolean initialWifiStateSet = false;
092   private static final String WIFI_ONSTART_STATE = "wifi_onStart_
state";
093   private static final String WIFI_PREFS = "wifi_prefs";
094   private static WifiManager wifiManager = null;
095
096   private static Context applicationContext;
097
098
099   public static final boolean DEBUGMODE = false;
100
101   public static Context getApplicationContext() {
102     return applicationContext;
103   }
104
105   public static void setApplicationContext(Context
applicationContext) {
106     Utils.applicationContext = applicationContext;
107     MOpenIDDBOpenHelper helper = new MOpenIDDBOpenHelper(applica
tionContext);
108     dbWriteable = helper.getWritableDatabase();
109   }
110
111   public static Calendar parseTimestamp(String timestamp) throws
Exception {
112     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss");
113     Date d = sdf.parse(timestamp);
114     Calendar cal = Calendar.getInstance();
```

```
115     cal.setTime(d);
116     return cal;
117   }
118
119   public static String convertStreamToString(InputStream is)
120       throws IOException {
121     /*
122      * To convert the InputStream to String we use the Reader.
read(char[]
123      * buffer) method. We iterate until the Reader return -1
which means
124      * there's no more data to read. We use the StringWriter
class to
125      * produce the string.
126      */
127     if (is != null) {
128       Writer writer = new StringWriter();
129
130       char[] buffer = new char[1024];
131       try {
132         Reader reader = new BufferedReader(new
InputStreamReader(is,
133             "UTF-8"));
134         int n;
135         while ((n = reader.read(buffer)) != -1) {
136           writer.write(buffer, 0, n);
137         }
138       } finally {
139         is.close();
140       }
141       return writer.toString();
142     } else {
143       return "";
144     }
145   }
146
147   public static boolean registerNewID(String identifier) {
148
149     HttpClient httpclient = new DefaultHttpClient();
150     HttpGet request = new HttpGet(MOPENID_WEBSERVICE_URL +
"associate.php?a="+ACTION_CREATE_NEW_IDENTIFIER+"&id="
151         + identifier);
152     Log.i("eirik", "requesting: " + MOPENID_WEBSERVICE_URL +
"associate.php?a="+ACTION_CREATE_NEW_IDENTIFIER+"&id="
153         + identifier);
154     ResponseHandler<String> handler = new
BasicResponseHandler();
155     String result = "{\"message\":\"failed_std\"}";
156     JSONObject jsonResult = null;
157     try {
158       result = httpclient.execute(request, handler);
159       Log.i("eirik", "got it");
160     } catch (ClientProtocolException e) {
161       e.printStackTrace();
162       Log.i("eirik", "execution failed due to an
ClientProtocolException");
163     } catch (IOException e) {
164       e.printStackTrace();
165       Log.i("eirik", "execution failed due to an IOException");
166     } catch (Exception e) {
167       Log.i("eirik", "execution failed due to an generic
Exception");
168       e.printStackTrace();
169     } finally {
170       try {
```

```
171         jsonResult = new JSONObject(result);
172       } catch (JSONException e) {
173         Log.i("eirik", "JSON parsing failed");
174         e.printStackTrace();
175       }
176     }
177
178     if (jsonResult != null) {
179       try {
180         if (jsonResult.getString("message").equals(
181             JSON_RESPONSE_INSERTED)) {
182           OpenID openId = new OpenID(identifier);
183           openId.setSecurityToken(jsonResult
184               .getString("security_token"));
185           // ContentValues newIdValues = new ContentValues();
186           // newIdValues.put("_id", identifier);
187           // newIdValues.put("security_token",
188           // jsonResult.getString("security_token"));
189           Log.i("eirik", "about to insert");
190           return insertNewIdToDatabase(openId);
191
192         }
193       } catch (JSONException e) {
194         Log.i("eirik", "JSON parsing failed: " + jsonResult.
toString());
195         e.printStackTrace();
196       }
197     }
198
199     return false;
200
201   }
202
203   public static boolean deleteLastCreatedID(boolean
deleteRemotelyToo) {
204     MOpenIDDBOpenHelper helper = new MOpenIDDBOpenHelper(applica
tionContext);
205     dbWriteable = helper.getWritableDatabase();
206     Cursor result = dbWriteable.query(
207         MOpenIDDBOpenHelper.MOPENID_ASSOCIATIONS_TABLE_NAME,
208         new String[] { "_id" }, null, null, null, null, "created
DESC");
209     if (result.moveToFirst()) {
210       return deleteId(result.getString(0), deleteRemotelyToo);
211     }
212     return false;
213   }
214
215   public static boolean insertNewIdToDatabase(OpenID newOpenId)
{
216     ContentValues openIdAsContentValues = newOpenId.
getContentValues();
217     Log.d("eirik","openid to insert: "+newOpenId.
toFormattedString());
218     return (dbWriteable.insert("links", null,
openIdAsContentValues) != -1);
219   }
220
221   public static String isAvailable(String identifier) {
222     HttpClient httpclient = new DefaultHttpClient();
223     String result = "{\"message\":\"taken\"}";
224     HttpGet request = new HttpGet(MOPENID_WEBSERVICE_URL +
"exists.php?id="
225         + identifier);
226     ResponseHandler<String> handler = new
```

```
BasicResponseHandler();
227     try {
228       result = httpclient.execute(request, handler);
229       Log.i("eirik", "got it");
230     } catch (ClientProtocolException e) {
231       e.printStackTrace();
232       Log.i("eirik", "got it not");
233     } catch (IOException e) {
234       e.printStackTrace();
235       Log.i("eirik", "got it never");
236     } catch (Exception e) {
237       Log.i("eirik", "got it huff huff");
238       e.printStackTrace();
239     }
240     httpclient.getConnectionManager().shutdown();
241     return result;
242   }
243
244   public static List<OpenID> getSavedOpenIDs() {
245     Cursor result = dbWriteable.query(
246         MOpenIDDBOpenHelper.MOPENID_ASSOCIATIONS_TABLE_NAME,
247         new String[] { "_id", "last_used", "security_token",
"created",
248             "description" }, null, null, null, null, "created");
249
250     List<OpenID> linkIDs = new ArrayList<OpenID>();
251     if (result.moveToFirst()) {
252       do {
253         OpenID id = new OpenID(result.getString(0), result.
getString(1));
254         id.setLastUsed(result.getString(1));
255         id.setSecurityToken(result.getString(2));
256         id.setCreated(result.getString(3));
257         id.setDescription(result.getString(4));
258         linkIDs.add(id);
259         Log.d("eirik", "got the id as follows: "+id.
toFormattedString());
260
261       } while (result.moveToNext());
262     }
263     result.close();
264     return linkIDs;
265   }
266
267   public static void dbUpdateLastUsed(OpenID openid) {
268     ContentValues updateValues = new ContentValues();
269     updateValues.put("last_used", openid.getLastUsed());
270     dbWriteable.update(MOpenIDDBOpenHelper.MOPENID_ASSOCIATIONS_
TABLE_NAME,
271         updateValues, "_id=?", new String[] { openid.
getIdentifier() });
272
273   }
274
275   private static boolean deleteId(String identifier, boolean
deleteRemotelyToo) {
276     MOpenIDDBOpenHelper helper = new MOpenIDDBOpenHelper(applica
tionContext);
277     boolean deletedRemotely = false;
278     String securityToken = null;
279     dbWriteable = helper.getWritableDatabase();
280     Cursor result = dbWriteable.query(
281         MOpenIDDBOpenHelper.MOPENID_ASSOCIATIONS_TABLE_NAME,
282         new String[] { "security_token"}, "_id=?", new String[]
{identifier}, null, null,"created DESC", "1");
```

```
283
284     if(result.moveToFirst()) {
285       securityToken = result.getString(0);
286       if(deleteRemotelyToo) {
287         JSONObject jsonResult = performServerRequest(identifier,
securityToken, ACTION_DELETE);
288         try {
289           deletedRemotely = (jsonResult!=null && jsonResult.
getString("message").equals("ok"));
290         } catch (JSONException e) {
291           e.printStackTrace();
292         }
293       }
294
295     }
296
297     if(deletedRemotely || !deleteRemotelyToo) {
298       if(!deleteRemotelyToo) {
299         Log.d("eirik", "Deleting the OpenID locally only");
300       }
301       return (-1 != dbWriteable.delete(MOpenIDDBOpenHelper.
MOPENID_ASSOCIATIONS_TABLE_NAME, "_id=? AND security_token=?",
302           new String[] {identifier,securityToken})); //deleted
completely
303     }
304
305     return false;
306
307   }
308
309
310
311   private static String getSecurityTokenForIdentifier(String
identifier) {
312     String securityToken = null;
313     Cursor result = dbWriteable.query(
314         MOpenIDDBOpenHelper.MOPENID_ASSOCIATIONS_TABLE_NAME,
315         new String[] { "security_token" }, "_id=?",
316         new String[] { identifier }, null, null, "created DESC",
"1");
317
318     if (result.moveToFirst())
319       securityToken = result.getString(0);
320     result.close();
321     return securityToken;
322   }
323
324   private static JSONObject performServerRequest(String
identifier,
325       String securityToken, int action) {
326     JSONObject jsonResult = null;
327
328     String phpFile = (ACTION_EXISTS == action) ? "exists" :
"associate";
329     HttpClient httpclient = new DefaultHttpClient();
330
331     HttpGet request = new HttpGet(MOPENID_WEBSERVICE_URL +
phpFile + ".php?a="
332         + action + "&id=" + identifier
333         + ((securityToken != null) ? "&st=" + securityToken :
""));
334     ResponseHandler<String> handler = new
BasicResponseHandler();
335     String response = "{\"message\":\"failed_std\"}";
336     try {
```

```
337       response = httpclient.execute(request, handler);
338     } catch (ClientProtocolException e) {
339       e.printStackTrace();
340       Log.i("eirik", "execution failed due to an
ClientProtocolException");
341     } catch (IOException e) {
342       e.printStackTrace();
343       Log.i("eirik", "execution failed due to an IOException");
344     } catch (Exception e) {
345       Log.i("eirik", "execution failed due to an generic
Exception");
346       e.printStackTrace();
347     } finally {
348       try {
349         jsonResult = new JSONObject(response);
350       } catch (JSONException e) {
351         Log.i("eirik", "JSON parsing failed");
352         e.printStackTrace();
353       }
354     }
355     return jsonResult;
356   }
357
358   public static boolean linkGivenmOpenIDtoHere(String identifier)
{
359     String securityToken = getSecurityTokenForIdentifier(identifi
er);
360     if (securityToken != null) {
361       return linkWithmOpenIDServer(identifier, securityToken);
362     }
363     return false;
364   }
365
366   private static boolean linkWithmOpenIDServer(String identifier,
367       String securityToken) {
368     String localEndpointURL = getLocalEndpointURLAsString();
369     HttpClient httpclient = new DefaultHttpClient();
370     String requestString = "associate.php?a="+ACTION_ASSOCIATE_
IDENTIFIER+"&id=" + identifier + "&st="
371         + securityToken + "&d=" + localEndpointURL;
372     HttpGet request = new HttpGet(MOPENID_WEBSERVICE_URL +
requestString);
373     Log.i("eirik", "requesting: " + MOPENID_WEBSERVICE_URL +
requestString);
374     ResponseHandler<String> handler = new
BasicResponseHandler();
375     String result = "{\"message\":\"failed_std\"}";
376     JSONObject jsonResult = null;
377     try {
378       result = httpclient.execute(request, handler);
379       Log.i("eirik", "got it");
380     } catch (ClientProtocolException e) {
381       e.printStackTrace();
382       Log.i("eirik", "execution failed due to an
ClientProtocolException");
383     } catch (IOException e) {
384       e.printStackTrace();
385       Log.i("eirik", "execution failed due to an IOException");
386     } catch (Exception e) {
387       Log.i("eirik", "execution failed due to an generic
Exception");
388       e.printStackTrace();
389     } finally {
390       try {
391         jsonResult = new JSONObject(result);
```

```
392       } catch (JSONException e) {
393         Log.i("eirik", "JSON parsing failed");
394         e.printStackTrace();
395       }
396     }
397
398     if (jsonResult != null) {
399       try {
400         if (jsonResult.getString("message").equals(
401             JSON_LINK_ASSOCIATED_OK)) {
402           return true;
403
404         }
405       } catch (JSONException e) {
406         Log.i("eirik", "JSON parsing failed: " + jsonResult.
toString());
407         e.printStackTrace();
408       }
409     }
410     return false;
411
412   }
413
414   public static String getLocalEndpointURLAsString() {
415     StringBuilder endpointAddress = new StringBuilder();
416     endpointAddress.append("http://");
417     try {
418       String ip = getPublicIPasString();
419       if (ip == null)
420         throw new Exception("Can't obtain own IP address");
421       endpointAddress.append(ip);
422       endpointAddress.append(":" + Utils.__PORT_DEFAULT);
423       endpointAddress.append("/" + Utils.__MOPENID_WS_NAME);
424       endpointAddress.append("/server");
425
426     } catch (Exception e) {
427       System.out.println("Exception caught =" + e.getMessage());
428       Log.i("eirik", "Failed building endpoint URL");
429       return null;
430
431     }
432
433     return endpointAddress.toString();
434   }
435
436   private static String getPublicIPasString() {
437     String ipaddress = null;
438     try {
439       for (Enumeration en = NetworkInterface.
getNetworkInterfaces(); en
440           .hasMoreElements();) {
441         NetworkInterface intf = (NetworkInterface)
en.nextElement();
442         for (Enumeration enumIpAddr = intf.getInetAddresses();
enumIpAddr
443             .hasMoreElements();) {
444           InetAddress inetAddress = (InetAddress) enumIpAddr
445               .nextElement();
446           if (!inetAddress.isLoopbackAddress()) {
447             ipaddress = inetAddress.getHostAddress().toString();
448             Log.e("ip address", "" + ipaddress);
449           }
450         }
451       }
452     } catch (SocketException ex) {
```

```
453       Log.e("Socket exception in GetIP Address of Utilities",
454           ex.toString());
455     }
456     return ipaddress;
457     //
458     // String ip = null;
459     // try {
460     // URL autoIP = new URL(
461     // "http://www.whatismyip.com/automation/n09230945.asp");
462     // BufferedReader in = new BufferedReader(new
InputStreamReader(
463     // autoIP.openStream()));
464     // ip = (in.readLine()).trim();
465     //
466     // } catch (Exception e) {
467     // e.printStackTrace();
468     //
469     // }
470     // return ip;
471   }
472
473   public static boolean disassociateOpenID(String identifier) {
474     String securityToken = getSecurityTokenForIdentifier(identifi
er);
475     if (securityToken != null) {
476       return unlinkWithmOpenIDServer(identifier, securityToken);
477     }
478     return false;
479
480   }
481
482   private static boolean unlinkWithmOpenIDServer(String
identifier,
483       String securityToken) {
484
485     HttpClient httpclient = new DefaultHttpClient();
486     String requestString = "associate.php?a="+ACTION_
DISASSOCIATE_IDENTIFIER+"&id=" + identifier + "&st="
487         + securityToken;
488     HttpGet request = new HttpGet(MOPENID_WEBSERVICE_URL +
requestString);
489     Log.i("eirik", "requesting: " + MOPENID_WEBSERVICE_URL +
requestString);
490     ResponseHandler<String> handler = new
BasicResponseHandler();
491     String result = "{\"message\":\"failed_std\"}";
492     JSONObject jsonResult = null;
493     try {
494       result = httpclient.execute(request, handler);
495       Log.i("eirik", "got it");
496     } catch (ClientProtocolException e) {
497       e.printStackTrace();
498       Log.i("eirik", "execution failed due to an
ClientProtocolException");
499     } catch (IOException e) {
500       e.printStackTrace();
501       Log.i("eirik", "execution failed due to an IOException");
502     } catch (Exception e) {
503       Log.i("eirik", "execution failed due to an generic
Exception");
504       e.printStackTrace();
505     } finally {
506       try {
507         jsonResult = new JSONObject(result);
508       } catch (JSONException e) {
```

```
509         Log.i("eirik", "JSON parsing failed");
510         e.printStackTrace();
511       }
512     }
513
514     if (jsonResult != null) {
515       try {
516         if (jsonResult.getString("message").equals(
517             JSON_LINK_DISASSOCIATED_OK)) {
518           Log.i("eirik", "disassociated mmkay");
519           return true;
520
521         }
522       } catch (JSONException e) {
523         Log.i("eirik", "JSON parsing failed: " + jsonResult.
toString());
524         e.printStackTrace();
525       }
526     }
527     return false;
528   }
529
530   public static void saveCurrentAndSwitchToPublicAPN(Context
context) {
531     saveCurrentApn(context);
532     SharedPreferences settings = context.getSharedPreferences(
533         APN_PREFS_NAME, 0);
534     long defaultApnId = (long) settings.getLong(APN_DEFAULT_ID_
KEY, -1);
535     String defaultApnName = settings.getString(APN_DEFAULT_NAME_
KEY, "");
536
537     if (apnIsSupported(defaultApnName)  &&
!isPublicAlready(context.getContentResolver())) {
538       setApnAsPublic(context.getContentResolver(), defaultApnId,
539           defaultApnName);
540     } else {
541       Log.d("eirik", "Current APN not supported or is already
public!");
542     }
543
544   }
545
546   public static boolean isPublicAndConnectedAlready(Context
context) {
547     boolean isConnectedToMobileNetwork = isConnectedToMobileNetw
ork(context);
548     boolean isPublicAlready = isPublicAlready(context.
getContentResolver());
549     return isPublicAlready && isConnectedToMobileNetwork;
550   }
551
552   private static boolean isConnectedToMobileNetwork(Context
context) {
553     TelephonyManager telephonyManager = (TelephonyManager)
context
554         .getSystemService(Context.TELEPHONY_SERVICE);;
555         int connId = telephonyManager.getDataState();
556         switch(connId) {
557
558         }
559
560         Log.d("eirik", "");
561     ConnectivityManager connectivityManager =
(ConnectivityManager) context
```

```
562         .getSystemService(Context.CONNECTIVITY_SERVICE);
563     NetworkInfo networkInfo = connectivityManager.
getActiveNetworkInfo();
564     if (networkInfo != null) {
565       return networkInfo.getType() == ConnectivityManager.TYPE_
MOBILE
566           && networkInfo.isConnected();
567     }
568     return false;
569   }
570
571   public static boolean isPublicAlready(ContentResolver
contentResolver) {
572     Cursor cursor = contentResolver.query(CONTENT_URI,
573         new String[] { "_id, apn" }, "current=1", null, null);
574     cursor.moveToFirst();
575     if (!cursor.isAfterLast()) {
576       return cursor.getString(1).contains(".public");
577     }
578     cursor.close();
579     return false;
580   }
581
582   private static void setApnAsPublic(ContentResolver
contentResolver,
583       long defaultApnId, String defaultApnName) {
584     ContentValues values = new ContentValues();
585     String publicApnName = defaultApnName + ".public";
586     values.put("apn", publicApnName);
587     contentResolver.update(CONTENT_URI, values, "_id=?",
588         new String[] { String.valueOf(defaultApnId) });
589
590   }
591
592   private static boolean apnIsSupported(String defaultApnName) {
593     for (int i = 0; i < SUPPORTED_APNS.length; i++) {
594       if (SUPPORTED_APNS[i].equals(defaultApnName.replace(".
public", ""))) {
595         Log.i("apn", "APN is supported ("+defaultApnName+")");
596         return true;
597       } else {
598         Log.i("apn", "APN is NOT supported
("+defaultApnName+")");
599       }
600     }
601     return false;
602   }
603
604   private static void saveCurrentApn(Context context) {
605     SharedPreferences settings = context.getSharedPreferences(
606         APN_PREFS_NAME, 0);
607     if (!settings.contains(APN_DEFAULT_ID_KEY)) {
608       HashMap<Long, String> result = getCurrentApn(context
609           .getContentResolver());
610       long apnId = (long) result.keySet().iterator().next();
611       String apnName = result.values().iterator().next();
612       if (apnName.contains(".public"))
613         return;
614       Log.i("eirik", "this is the current apn id: " + apnId + ""
615           + " and name: " + apnName);
616
617       SharedPreferences.Editor editor = settings.edit();
618       editor.putLong(APN_DEFAULT_ID_KEY, apnId);
619       editor.putString(APN_DEFAULT_NAME_KEY, apnName);
620       editor.commit();
```

```
621     }
622
623   }
624
625   public static HashMap<Long, String>
getCurrentApn(ContentResolver contentResolver) {
626     HashMap<Long, String> result = new HashMap<Long, String>();
627     Cursor cursor = contentResolver.query(CONTENT_URI,
628         new String[] { "_id, apn" }, "current=1", null, null);
629     cursor.moveToFirst();
630     Log.d("apn", "In getCurrentApn() Ð number of apns with
current=1: "+cursor.getCount());
631     if (!cursor.isAfterLast()) {
632       result.put(Long.valueOf(cursor.getLong(0)), cursor.
getString(1));
633     }
634     cursor.close();
635     return result;
636   }
637
638   public static void restoreApnSettingToNormal(Context context)
{
639     SharedPreferences settings = context.getSharedPreferences(
640         APN_PREFS_NAME, 0);
641     long defaultApnId = (long) settings.getLong(APN_DEFAULT_ID_
KEY, -1);
642     String defaultApnName = settings.getString(APN_DEFAULT_NAME_
KEY, null);
643     if (defaultApnName != null &&
apnIsSupported(defaultApnName)) {
644       ContentValues values = new ContentValues();
645       values.put("apn", defaultApnName);
646       context.getContentResolver()
647           .update(CONTENT_URI, values, "_id" + "=?",
648               new String[] { String.valueOf(defaultApnId) });
649       Log.i("eirik", "Restored APN settings to apn=" +
defaultApnName
650           + " at _id=" + defaultApnId);
651     } else {
652       Log.i("eirik", "Unsupported APN name: \"" + defaultApnName
653           + "\". Couldn't restore.");
654     }
655
656   }
657
658   public static boolean isPublic(Context context) {
659     HashMap<Long, String> current = getCurrentApn(context
660         .getContentResolver());
661     return current.values().iterator().next().contains(".
public");
662   }
663
664   public static void saveInitialWifiState(Context context) {
665     if (wifiManager == null)
666       wifiManager = (WifiManager) context
667           .getSystemService(Context.WIFI_SERVICE);
668
669     if (!initialWifiStateSet) {
670       SharedPreferences settings = context.getSharedPreferences(
671           WIFI_PREFS, 0);
672       SharedPreferences.Editor editor = settings.edit();
673       editor.putBoolean(WIFI_ONSTART_STATE, wifiManager.
isWifiEnabled());
674       editor.commit();
675       initialWifiStateSet = true;
```

```
676       Log.i("eirik", "setting the prefs");
677     }
678
679     SharedPreferences settings = context
680         .getSharedPreferences(WIFI_PREFS, 0);
681     if (settings.contains(WIFI_ONSTART_STATE)) {
682       Log.i("eirik",
683           "wifi was enabled on initial launch: "
684               + settings.getBoolean(WIFI_ONSTART_STATE, false));
685     }
686
687   }
688
689   public static void resetToInitialWifiState(Context context) {
690     if (wifiManager == null)
691       wifiManager = (WifiManager) context
692           .getSystemService(Context.WIFI_SERVICE);
693     SharedPreferences settings = context
694         .getSharedPreferences(WIFI_PREFS, 0);
695     if (settings.contains(WIFI_ONSTART_STATE)) {
696       boolean initialState = settings.getBoolean(WIFI_ONSTART_
STATE,
697           false);
698       wifiManager.setWifiEnabled(initialState);
699       SharedPreferences.Editor editor = settings.edit();
700       editor.remove(WIFI_ONSTART_STATE);
701     }
702     initialWifiStateSet = false;
703
704   }
705
706   public static void disableWifi(Context context) {
707     if (wifiManager == null)
708       wifiManager = (WifiManager) context
709           .getSystemService(Context.WIFI_SERVICE);
710     if (wifiManager.getWifiState() != WifiManager.WIFI_STATE_
DISABLED) {
711       wifiManager.setWifiEnabled(false);
712     }
713
714   }
715
716   public static boolean isCorrectDecryptionKey(String
decryptedText) {
717     if (decryptedText == null || decryptedText.length() < 3)
718       return false;
719     String[] lines = decryptedText.split("\n");
720     return lines[0].equals(BackupUtils.OKAY_FLAG);
721
722   }
723
724   public static boolean isWifiEnabled() {
725     return wifiManager.isWifiEnabled();
726   }
727
728   public static boolean idAlreadyExists(OpenID openID) {
729     Cursor result = dbWriteable.query(
730         MOpenIDDBOpenHelper.MOPENID_ASSOCIATIONS_TABLE_NAME,
731         new String[] { "_id" }, "_id=?",
732         new String[] { openID.getIdentifier() }, null, null,
null, null);
733
734     return result.moveToFirst();
735   }
736
```

```
737   public static String fetchStringFromUri(Uri uri) throws
URISyntaxException, FileNotFoundException {
738     String result = null;
739
740     if(uri.getScheme().equalsIgnoreCase("https") || uri.
getScheme().equalsIgnoreCase("http") ) {
741       URI httpuri = URI.create(uri.toString());
742
743       HttpClient httpclient = new DefaultHttpClient();
744
745
746       HttpGet request = new HttpGet(httpuri);
747       ResponseHandler<String> handler = new
BasicResponseHandler();
748       try {
749         result = httpclient.execute(request, handler);
750         Log.i("eirik", "got it");
751       } catch (ClientProtocolException e) {
752         e.printStackTrace();
753         Log.i("eirik", "got it not");
754       } catch (IOException e) {
755         e.printStackTrace();
756         Log.i("eirik", "got it never");
757       } catch (Exception e) {
758         Log.i("eirik", "got it huff huff");
759         e.printStackTrace();
760       }
761       httpclient.getConnectionManager().shutdown();
762       return result;
763     } else if (uri.getScheme().equalsIgnoreCase("file")) {
764       File encryptedFile = new File(new URI(uri.toString()));
765
766       Scanner scanner = new Scanner(encryptedFile).
useDelimiter("\\Z");
767           result = scanner.next();
768     }
769
770
771
772     return result;
773   }
774
775   public static void startIJettyService() {
776
777         Intent intent = new Intent("android.intent.action.
MAIN");
778         intent.setComponent(new ComponentName("org.mortbay.
ijetty", "org.mortbay.ijetty.IJettyService"));
779         intent.addCategory("android.intent.category.LAUNCHER");
780
781
782         intent.putExtra(__PORT,__PORT_DEFAULT);
783         intent.putExtra(__NIO,__NIO_DEFAULT);
784         intent.putExtra(__SSL,__SSL_DEFAULT);
785         intent.putExtra(__CONSOLE_PWD,__CONSOLE_PWD_DEFAULT);
786         applicationContext.startService(intent);
787
788   }
789
790   public static void stopIJettyService() {
791     Intent intent = new Intent();
792     intent.setComponent(new ComponentName("org.mortbay.ijetty",
"org.mortbay.ijetty.IJettyService"));
793     applicationContext.stopService(intent);
794   }
```

```
795
796   public static boolean apnSettingsSupported() {
797     HashMap<Long, String> current = getCurrentApn(applicationCon
text.getContentResolver());
798     if(current.size() > 0) {
799       return apnIsSupported(current.values().iterator().next());
800     }
801     return false;
802   }
803
804   public static String getCurrentApnName() {
805     HashMap <Long, String> current = getCurrentApn(applicationCo
ntext.getContentResolver());
806     if(current.size() == 1) {
807       String apnName = current.values().iterator().next();
808       Log.d("apn", "Current APN name: "+apnName);
809       return apnName;
810     }
811     return null;
812   }
813
814   public static boolean deleteId(OpenID idToDelete, boolean
deleteRemotelyToo) {
815       return deleteId(idToDelete.getIdentifier(),
deleteRemotelyToo);
816   }
817
818   public static boolean dataEnabled() {
819     ConnectivityManager connectivityManager =
(ConnectivityManager) applicationContext
820         .getSystemService(Context.CONNECTIVITY_SERVICE);
821     NetworkInfo[] networkInfo = connectivityManager.
getAllNetworkInfo();
822     if (networkInfo != null) {
823       for (int i = 0; i < networkInfo.length; i++) {
824         Log.d("eirik", "reason: "+networkInfo[i].getReason());
825         if (networkInfo[i].getReason() == null) { continue; }
826         if(networkInfo[i].getType() == ConnectivityManager.TYPE_
MOBILE
827             && (!networkInfo[i].getReason().
equals("dataDisabled"))) { // somewhat bad hack/fix
828           return true;
829         }
830       }
831     }
832     return false;
833   }
834 }
```

MOpenIDDBOpenHelper.java

```
01 package no.ntnu.item.mopenid.util;
02
03 import android.content.Context;
04 import android.database.sqlite.SQLiteDatabase;
05 import android.database.sqlite.SQLiteOpenHelper;
06 import android.util.Log;
07
08 public class MOpenIDDBOpenHelper extends SQLiteOpenHelper {
09   private static final String DATABASE_NAME = "mopenid.db";
10   private static final int DATABASE_VERSION = 9;
11   public static final String MOPENID_ASSOCIATIONS_TABLE_NAME =
"links";
12   private static final String LINKS_TABLE_CREATE = "CREATE TABLE "
13       + MOPENID_ASSOCIATIONS_TABLE_NAME + " ("
14       + "_id TEXT PRIMARY KEY NOT NULL, " + "security_token
TEXT,"
15       + "description TEXT,"+ "last_used TEXT," + "created TEXT
DEFAULT CURRENT_TIMESTAMP"
16       + ");";
17
18   public static final String MOPENID_GRANTED_TABLE_NAME = "sites";
19   public MOpenIDDBOpenHelper(Context context) {
20     super(context, DATABASE_NAME, null, DATABASE_VERSION);
21   }
22
23   public void onCreate(SQLiteDatabase db) {
24     db.execSQL(LINKS_TABLE_CREATE);
25   }
26
27   @Override
28   public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
29     db.execSQL("DROP TABLE IF EXISTS links ");
30     onCreate(db);
31   }
32
33 }
```

OpenID.java

```
01 package no.ntnu.item.mopenid.util;
02
03 import android.content.ContentValues;
04
05 public class OpenID {
06
07   private String identifier, lastUsed, description, created,
securityToken;
08   private boolean isCreateNewButton = false;;
09
10   public OpenID(String id) {
11     identifier = id;
12     if(id.equals(">> Create new")) {
13       isCreateNewButton = true;
14     }
15   }
16
17   public boolean isCreateNewButton() {
18     return isCreateNewButton;
19   }
```

```
20
21    public void setCreateNewButton(boolean isCreateNewButton) {
22      this.isCreateNewButton = isCreateNewButton;
23    }
24
25    public OpenID(String id, String lastUsed) {
26      this(id);
27      this.lastUsed = lastUsed;
28    }
29
30    public String getIdentifier() {
31      return identifier;
32    }
33
34    public void setIdentifier(String identifier) {
35      this.identifier = identifier;
36    }
37
38    public String getLastUsed() {
39      return lastUsed;
40    }
41
42    public void setLastUsed(String lastUsed) {
43      this.lastUsed = lastUsed;
44    }
45
46    public String getDescription() {
47      return description;
48    }
49
50    public void setDescription(String description) {
51      this.description = description;
52    }
53
54
55    public String toString() {
56      return identifier;
57    }
58
59    public void setCreated(String created) {
60      this.created = created;
61    }
62
63    public String getCreated() {
64      return created;
65    }
66
67    public void setSecurityToken(String securityToken) {
68      this.securityToken = securityToken;
69    }
70
71    public String getSecurityToken() {
72      return securityToken;
73    }
74    public String toFormattedString() {
75      return identifier+ ";"+lastUsed+";"+securityToken+";"+created+
";"+description;
76    }
77
78    public ContentValues getContentValues() {
79      ContentValues contentValues = new ContentValues();
80      contentValues.put("_id", this.identifier);
81      contentValues.put("last_used", this.lastUsed);
82      contentValues.put("security_token", this.securityToken);
83      contentValues.put("description", this.description);
```

```
84      return contentValues;
85    }
86
87  }
```

CryptoSuite.java

```
001 package no.ntnu.item.mopenid.util;
002
003 import java.security.MessageDigest;
004 import java.security.NoSuchAlgorithmException;
005
006
007 import java.security.SecureRandom;
008
009 import javax.crypto.Cipher;
010 import javax.crypto.KeyGenerator;
011 import javax.crypto.SecretKey;
012 import javax.crypto.spec.SecretKeySpec;
013
014 /*
015  * This class provides the security keys and helper methods
related to a specific
016  * identifier connection between the mopenID app and server.
017  */
018 public class CryptoSuite {
019
020   public CryptoSuite() {
021   }
022
023   //TODO: not needed atm.
024   public String getHashString(String message) {
025     try {
026       MessageDigest digest = java.security.MessageDigest.
getInstance("SHA-256");
027       digest.update(message.getBytes());
028       byte messageDigest[] = digest.digest();
029
030       // Create Hex String
031       StringBuffer hexString = new StringBuffer();
032       for (int i = 0; i < messageDigest.length; i++)
033         hexString.append(Integer.toHexString(0xFF &
messageDigest[i]));
034       return hexString.toString();
035     } catch (NoSuchAlgorithmException e) {
036       e.printStackTrace();
037     }
038     return "";
039   }
040
041
042   /**
043    * Usage:
044    * <pre>
045    * String crypto = SimpleCrypto.encrypt(masterpassword,
cleartext)
046    * ...
047    * String cleartext = SimpleCrypto.decrypt(masterpassword,
crypto)
048    * </pre>
049    * @author ferenc.hechler
050    *
```

```
051     * Modified by Eirik Stien - eiriksti@stud.ntnu.no
052     * April 2011
053     */
054
055    public static String encrypt(String seed, String cleartext)
throws Exception {
056      byte[] rawKey = getRawKey(seed.getBytes());
057      byte[] result = encrypt(rawKey, cleartext.getBytes());
058      return toHex(result);
059    }
060
061    public static String decrypt(String seed, String encrypted)
throws Exception {
062      byte[] rawKey = getRawKey(seed.getBytes());
063      byte[] enc = toByte(encrypted);
064      byte[] result = decrypt(rawKey, enc);
065      return new String(result);
066    }
067
068    private static byte[] getRawKey(byte[] seed) throws Exception
{
069      KeyGenerator kgen = KeyGenerator.getInstance("AES");
070      SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
071      sr.setSeed(seed);
072        kgen.init(128, sr); // 192 and 256 bits may not be
available
073        SecretKey skey = kgen.generateKey();
074        byte[] raw = skey.getEncoded();
075        return raw;
076    }
077
078
079    private static byte[] encrypt(byte[] raw, byte[] clear) throws
Exception {
080        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
081      Cipher cipher = Cipher.getInstance("AES");
082        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
083        byte[] encrypted = cipher.doFinal(clear);
084      return encrypted;
085    }
086
087    private static byte[] decrypt(byte[] raw, byte[] encrypted)
throws Exception {
088        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
089      Cipher cipher = Cipher.getInstance("AES");
090        cipher.init(Cipher.DECRYPT_MODE, skeySpec);
091        byte[] decrypted = cipher.doFinal(encrypted);
092      return decrypted;
093    }
094
095    public static String toHex(String txt) {
096      return toHex(txt.getBytes());
097    }
098    public static String fromHex(String hex) {
099      return new String(toByte(hex));
100    }
101
102    public static byte[] toByte(String hexString) {
103      int len = hexString.length()/2;
104      byte[] result = new byte[len];
105      for (int i = 0; i < len; i++)
106        result[i] = Integer.valueOf(hexString.substring(2*i,
2*i+2), 16).byteValue();
107      return result;
108    }
```

```
109
110   public static String toHex(byte[] buf) {
111     if (buf == null)
112       return "";
113     StringBuffer result = new StringBuffer(2*buf.length);
114     for (int i = 0; i < buf.length; i++) {
115       appendHex(result, buf[i]);
116     }
117     return result.toString();
118   }
119   private final static String HEX = "0123456789ABCDEF";
120   private static void appendHex(StringBuffer sb, byte b) {
121     sb.append(HEX.charAt((b>>4)&0x0f)).append(HEX.
charAt(b&0x0f));
122   }
123
124 }
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="no.ntnu.item.mopenid.app"
android:versionCode="1"
android:versionName="1.0"
android:multiprocess="true"
android:launchMode="singleInstance">
<uses-sdk android:targetSdkVersion="8"
android:minSdkVersion="8"/>



<application android:icon="@drawable/icon"
android:label="@string/app_full_name">

<activity android:name=".SelectMode"
android:label="@string/app_name"
android:screenOrientation="portrait">
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>

<activity android:noHistory="true"
android:name="SelectMOpenIDMode"
android:label="@string/app_full_name"
android:screenOrientation="portrait">
</activity>

<activity android:noHistory="true"
android:name="StaticIpModeActivation"
android:label="@string/app_full_name"
android:screenOrientation="portrait">
</activity>

<activity android:noHistory="true"
android:name="MOpenIdModeActivation"
android:label="@string/app_full_name"
android:screenOrientation="portrait">
</activity>
```

```
<activity android:noHistory="true"
android:name=".CreateNew"
android:label="@string/app_full_name"
android:screenOrientation="portrait">
</activity>

<activity android:noHistory="true"
android:name="AbstractActivation"
android:label="@string/app_full_name"
android:screenOrientation="portrait">
</activity>

<service android:name="org.mortbay.ijetty.IJettyService"
android:process=":remote" />

</application>


<uses-permission
android:name="android.permission.WRITE_SETTINGS" />
<uses-permission
android:name="android.permission.WRITE_APN_SETTINGS" />
<uses-permission
android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.UPDATE_DEVICE_STATS"/>
<uses-permission
android:name="android.permission.CHANGE_WIFI_STATE"/>
<uses-permission
android:name="android.permission.WAKE_LOCK"/>

</manifest>
```

OpenIDServlet.java

```
001 package no.ntnu.item.mopenid.webapp;
002 import java.io.IOException;
003 import javax.servlet.ServletConfig;
004 import javax.servlet.ServletException;
005 import javax.servlet.ServletOutputStream;
006 import javax.servlet.http.HttpServlet;
007 import javax.servlet.http.HttpServletRequest;
008 import javax.servlet.http.HttpServletResponse;
009 import javax.servlet.http.HttpSession;
010
011 import no.ntnu.item.mopenid.util.Utils;
012
013 import org.openid4java.message.AuthSuccess;
014 import org.openid4java.message.Message;
015 import org.openid4java.message.ParameterList;
016 import org.openid4java.message.VerifyResponse;
017 import org.openid4java.server.ServerException;
018 import org.openid4java.server.ServerManager;
019
020 import android.content.BroadcastReceiver;
021 import android.content.Context;
022 import android.content.Intent;
023 import android.content.IntentFilter;
024 import android.util.Log;
025
026
027
028
029 /* ------------------------------------------------------------
*/
030 /**
031  *
032  */
033 public class OpenIDServlet extends HttpServlet
034 {
035     Context androidContext;
036     public ServerManager manager;
037   private boolean authenticationFromProviderOkay;
038   private boolean associateRequestReceived = false;
039   private Message authenticationResponseMessage;
040   private String identifier;
041   private BroadcastReceiver authenticationReceiver;
042   private HttpSession savedSession;
043   private static final String INTENT_ACTION_SEND_AUTH_REQUEST =
"no.ntnu.item.mopenid.ACTION_AUTH_REQUEST";
044   private static final String INTENT_ACTION_SEND_AUTH_RESPONSE =
"no.ntnu.item.mopenid.ACTION_AUTH_RESPONSE";
045   private static final String INTENT_ACTION_SEND_AUTH_PRESSED_
COMPELETE = "no.ntnu.item.mopenid.ACTION_AUTH_COMPLETED_LOGIN";
046   private static final String INTENT_ACTION_SEND_AUTH_COMPLETED_
COMPLETELY = "no.ntnu.item.mopenid.ACTION_AUTH_COMPLETED_LOGIN_
COMPLETELY";
047   private static final String INTENT_ACTION_SET_STATIC_IP_MODE =
"no.ntnu.item.mopenid.ACTION_STATIC_IP_MODE";
048   private static final String INTENT_ACTION_SET_STATIC_IP_MODE_
REQUEST = "no.ntnu.item.mopenid.ACTION_STATIC_IP_MODE_REQUEST";
049   private static final String IDENTIFIER_PREFIX = "https://
mopenid.item.ntnu.no/id/";
```

```
050   private static boolean staticIpMode = false;
051
052     /* ------------------------------------------------------
---- */
053     public void init(ServletConfig config) throws ServletException
054     {
055       super.init(config);
056       String endpointURL = Utils.getLocalEndpointURLAsString();
057
058       manager = new ServerManager();
059       manager.setOPEndpointUrl(endpointURL);
060
061       manager.setUserSetupUrl("/");
062
063       Log.d("openid", "Set OP endpoint URL to: "+endpointURL);
064       authenticationFromProviderOkay = false;
065         Object o = config.getServletContext().getAttribute("org.
mortbay.ijetty.contentResolver");
066         android.content.ContentResolver resolver = (android.
content.ContentResolver)o;
067         androidContext = (android.content.Context)config.
getServletContext().getAttribute("org.mortbay.ijetty.context");
068
069         setupStaticModeListener();
070         modeResolver();
071
072         Log.i("openid", "server address is: "+endpointURL);
073     }
074
075     private void modeResolver() {
076     Intent intent = new Intent(INTENT_ACTION_SET_STATIC_IP_MODE_
REQUEST);
077     androidContext.sendBroadcast(intent);
078   }
079
080   private void setupStaticModeListener() {
081     androidContext.registerReceiver(new BroadcastReceiver() {
082
083       @Override
084       public void onReceive(Context context, Intent intent) {
085         if(intent.hasExtra("staticMode")) {
086           staticIpMode = intent.getBooleanExtra("staticMode",
false);
087           Log.d("eirik", "Servlet received staticMode intent and
it is: "+staticIpMode);
088         }
089
090       }
091     }, new IntentFilter(INTENT_ACTION_SET_STATIC_IP_MODE));
092
093   }
094
095   /* ------------------------------------------------------
-- */
096     public void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
097     {
098         doGet(request, response);
099     }
100
101     /* ------------------------------------------------------
---- */
102     public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
103     {
```

```
104
105
106       try {
107
108
109
110         // Check to see if the IP address of the smartphone have
changed since last time
111         // Happens only in the rare case of the smartphone IP
going from public to hidden, then back to
112         // public again, without restarting the iJetty
webserver.
113         String endpointURL = Utils.
getLocalEndpointURLAsString(); // quite time consuming for each http
request.
114         if(!endpointURL.equals(manager.getOPEndpointUrl())) {
115           Log.d("eirik", "OP Endpoint URL have changed =>
updating...");
116           manager.setOPEndpointUrl(endpointURL);
117         }
118
119
120         processRequestAndReturnResponse(request, response);
121
122       } catch (Exception e) {
123         Log.i("openid", "ukoselig feilmelding.");
124         e.printStackTrace();
125
126         response.setContentType("text/html");
127         ServletOutputStream out = response.getOutputStream();
128         out.println("error dessverre: "+ e.toString()); //TODO
better error handling
129         out.flush();
130     }
131
132
133     }
134
135   private void processRequestAndReturnResponse(HttpServletReque
st request, HttpServletResponse response) throws Exception{
136
137     // extract the parameters from the request
138         ParameterList requestParameters = new
ParameterList(request.getParameterMap());
139
140
141         String mode = requestParameters.hasParameter("openid.
mode") ?
142                 requestParameters.getParameterValue("openid.
mode") : null;
143
144         Log.d("openid", "Mode is: "+mode);
145
146
147         Message responseMessage;
148         String responseText;
149
150         if(mode == null) {
151           // request comes from "complete login" form, or is a
RP performing OP endpoint localization, or is an xrds request
152
153       if (savedSession == null) {
154         if (staticIpMode) {
155           if (request.getRequestURI().contains("/xrds")) {
156             buildAndPrintXrdsDocument(response);
```

```
157           } else {
158             buildAndPrintIdPage(response);
159           }
160         }
161         return;
162
163       }
164
165
166           Log.d("openid", "is the session the same ? "+request.
getSession().getId().equals(savedSession.getId()));
167
168
169           if(savedSession != null) {
170             requestParameters = (ParameterList) savedSession.get
Attribute("requestParameters");
171             savedSession.removeAttribute("requestParameters");
172           }
173
174 //          Log.d("openid", "requestParameters from session:
"+requestParameters.toString());
175
176           boolean isAuthenticatedAndHaveSameSession =
authenticationFromProviderOkay && request.getSession().getId().
equals(savedSession.getId());
177
178           authenticationResponseMessage = manager.
authResponse(requestParameters,
179                   identifier, identifier,
isAuthenticatedAndHaveSameSession);
180
181
182           if(isAuthenticatedAndHaveSameSession) {
183             //login used, notify mopenid application with intent
184             Intent confirmationIntent = new Intent(INTENT_ACTION_
SEND_AUTH_PRESSED_COMPELETE);
185             androidContext.sendBroadcast(confirmationIntent);
186           }
187
188
189
190           String redirectURL = authenticationResponseMessage.
getDestinationUrl(true);
191
192           Log.d("eirik", "destination url: " + redirectURL);
193           response.sendRedirect(redirectURL);
194           savedSession = null;
195           identifier = null;
196
197           if(authenticationResponseMessage instanceof
AuthSuccess) {
198             if(associateRequestReceived) {
199               new DelayThread(4000).start();
200             }
201           }
202
203           authenticationResponseMessage = null;
204           associateRequestReceived = false;
205         }
206
207
208         else if ("associate".equals(mode))
209         {
210
211
```

```
212             // --- handle an association request ---
213           associateRequestReceived = true;
214             handleAssociationRequest(response,
requestParameters);
215
216         }
217         else if ("checkid_setup".equals(mode))
218         {
219             requestUserAuthenticationAndRegisterReceiver(request
Parameters);
220
221             // --- process an authentication request ---
222 //            AuthRequest authReq = AuthRequest.createAuthReques
t(requestParameters, manager.getRealmVerifier());
223
224             assignIdentifierAndSessionField(request,
requestParameters);
225             printFinalLoginForm(identifier, response);
226
227         }
228         else if("checkid_immediate".equals(mode)) {
229           associateRequestReceived = false;
230           assignIdentifierAndSessionField(request,
requestParameters);
231           Message authImmediateFailureMessage = manager.
authResponse(requestParameters, identifier, identifier, false);
232           String openidNs = authImmediateFailureMessage.
getParameterValue("openid.ns");
233           String openidMode = authImmediateFailureMessage.
getParameterValue("openid.mode");
234
235           String redirectURL = authImmediateFailureMessage.
getDestinationUrl(false) +
236               "&openid.mode="+openidMode+"&openid.ns="+openidNs;
237
238           Log.d("eirik", "destination url: " + redirectURL);
239         response.sendRedirect(redirectURL);
240
241         Log.i("openid", "Sent authImmediateFailureMessage
message to RP");
242
243         }
244         else if ("check_authentication".equals(mode))
245         {
246             // --- processing a verification request ---
247             responseMessage = manager.verify(requestParameters);
248             responseText = responseMessage.
keyValueFormEncoding();
249
250             ServletOutputStream os = response.getOutputStream();
251         os.write(responseText.getBytes());
252         os.flush();
253         Log.i("openid", "Sent verification message to RP");
254         if(responseMessage instanceof VerifyResponse) {
255           VerifyResponse msg = (VerifyResponse) responseMessage;
256             if(!associateRequestReceived && msg.
isSignatureVerified()) {
257
258               DelayThread delayedExecution = new
DelayThread(5000);
259           delayedExecution.start();
260
261           }
262         }
263
```

```
264         }
265         else
266         {
267             // --- error response ---
268 //            responseMessage = DirectError.
createDirectError("Unknown request");
269 //            responseText = responseMessage.
keyValueFormEncoding();
270           Log.d("openid", "unknown openid request");
271
272         }
273
274
275   }
276
277   private class DelayThread extends Thread {
278     private int delay;
279     public DelayThread(int delay) {
280       super();
281       this.delay = delay;
282
283     }
284
285     public void run() {
286       try {
287         sleep(5000);
288       } catch (InterruptedException e) {
289         e.printStackTrace();
290         Log.e("eirik","Interrupted: "+e.getMessage());
291       }
292       Intent allGoodIntent = new Intent(INTENT_ACTION_SEND_AUTH_
COMPLETED_COMPLETELY);
293       androidContext.sendBroadcast(allGoodIntent);
294       Log.d("eirik", "Sent Intent to close confirmation dialog,
"+(int)delay/1000+" seconds after.");
295
296
297     }
298   }
299
300
301   private void buildAndPrintIdPage(HttpServletResponse response)
throws IOException {
302
303     ServletOutputStream os = response.getOutputStream();
304     String htmlDocument = Utils.readFileAsString("/mnt/sdcard/
jetty/webapps/mopenid/id_page.html");
305     String opEndpointURL = manager.getOPEndpointUrl();
306     htmlDocument = htmlDocument.replaceAll("%s", opEndpointURL);
307     htmlDocument = htmlDocument.replaceAll("%x",
opEndpointURL+"/xrds");
308     os.println(htmlDocument);
309     os.flush();
310
311   }
312
313   private void buildAndPrintXrdsDocument(HttpServletResponse
response) throws IOException {
314
315     response.setContentType("application/xrds+xml");
316     ServletOutputStream os = response.getOutputStream();
317     String htmlDocument = Utils.readFileAsString("/mnt/sdcard/
jetty/webapps/mopenid/xrds");
318     String opEndpointURL = manager.getOPEndpointUrl();
319     htmlDocument = htmlDocument.replaceAll("%s", opEndpointURL);
```

```
320     os.println(htmlDocument);
321     os.flush();
322
323   }
324
325
326   private void assignIdentifierAndSessionField(HttpServletRequest
request,
327       ParameterList requestParameters) {
328     identifier = requestParameters.getParameterValue("openid.
identity");
329     String claimedIdentifier = requestParameters.
getParameterValue("openid.claimed_identifier");
330     savedSession = request.getSession();
331     savedSession.setAttribute("identifier", identifier);
332     savedSession.setAttribute("claimedIdentifier",
claimedIdentifier);
333     savedSession.setAttribute("return_to", requestParameters.
getParameterValue("openid.return_to"));
334     savedSession.setAttribute("requestParameters",
requestParameters);
335   }
336
337
338
339   private void printErrorMessage(HttpServletResponse response)
throws IOException {
340     ServletOutputStream os = response.getOutputStream();
341     os.println("<html>" +
342         "<head>" +
343         "  <title>mOpenID</title>" +
344         "</head>" +
345         "<body>Stupid error message" +
346         "</body></html>");
347     os.flush();
348
349   }
350
351   private void printFinalLoginForm(String identifier,
HttpServletResponse response) throws IOException {
352     ServletOutputStream os = response.getOutputStream();
353     String htmlDocument = Utils.readFileAsString("/mnt/sdcard/
jetty/webapps/mopenid/login_screen.html");
354     htmlDocument = htmlDocument.replaceFirst("%s", identifier);
355     os.println(htmlDocument);
356     os.flush();
357
358   }
359
360   private void handleAssociationRequest(HttpServletResponse
response, ParameterList requestParameters) throws IOException {
361     Message responseMessage;
362     String responseText;
363     responseMessage = manager.associationResponse(requestParame
ters);
364     responseText = responseMessage.keyValueFormEncoding();
365
366     //write output to the RP
367
368     ServletOutputStream os = response.getOutputStream();
369     os.write(responseText.getBytes());
370     os.flush();
371     Log.i("openid", "RP sent an associate request, and this is
the reply: "+responseText);
372   }
```

```
373
374    private void requestUserAuthenticationAndRegisterReceiver(Par
ameterList request) throws ServerException
375       {
376
377         setupAndRegisterAuthenticationReceiver();
378
379         Intent infoIntent = new Intent(INTENT_ACTION_SEND_AUTH_
REQUEST);
380         infoIntent.putExtra("openid.realm", request.
getParameterValue("openid.realm"));
381         infoIntent.putExtra("openid.claimed_id", request.
getParameterValue("openid.claimed_id"));
382
383         androidContext.sendBroadcast(infoIntent); //TODO: add
permission required
384
385         Log.i("openid", "Sent Intent ala "+INTENT_ACTION_SEND_
AUTH_REQUEST);
386
387
388
389
390       }
391
392   private void setupAndRegisterAuthenticationReceiver() {
393
394     authenticationReceiver = new BroadcastReceiver() {
395
396
397
398       @Override
399       public void onReceive(Context context, Intent intent) {
400         authenticationFromProviderOkay = intent.getBooleanExtra(
"authenticated", false);
401         identifier = intent.getStringExtra("identifier");
402         Log.i("openid", "recived auth response and it was:
"+authenticationFromProviderOkay);
403         if(!authenticationFromProviderOkay) {
404           String reason = intent.getStringExtra("reason");
405           Log.i("openid", "The reason is: "+reason);
406         }
407         androidContext.unregisterReceiver(authenticationReceiv
er);
408       }
409     };
410
411     androidContext.registerReceiver(authenticationReceiver, new
IntentFilter(INTENT_ACTION_SEND_AUTH_RESPONSE));
412
413
414   }
415
416 }
```

Utils.java

```
01 package no.ntnu.item.mopenid.util;
02
03 import java.io.BufferedReader;
04 import java.io.FileReader;
05 import java.net.InetAddress;
```

```
06 import java.net.NetworkInterface;
07 import java.net.SocketException;
08 import java.util.Enumeration;
09
10 import android.util.Log;
11
12 public class Utils {
13
14   public static final String __PORT_DEFAULT = "8080";
15   private static final String __MOPENID_WS_NAME = "mopenid";
16
17   public static String getLocalEndpointURLAsString() {
18     StringBuilder endpointAddress = new StringBuilder();
19     endpointAddress.append("http://");
20     try {
21       String ip = getPublicIPasString();
22       if (ip == null)
23         throw new Exception("Can't obtain own IP address");
24       endpointAddress.append(ip);
25       endpointAddress.append(":" + Utils.__PORT_DEFAULT);
26       endpointAddress.append("/" + Utils.__MOPENID_WS_NAME);
27       endpointAddress.append("/server");
28
29     } catch (Exception e) {
30       System.out.println("Exception caught =" + e.getMessage());
31       Log.i("eirik", "Failed building endpoint URL");
32       return null;
33
34     }
35
36     return endpointAddress.toString();
37   }
38
39   public static String readFileAsString(String filePath)   throws
java.io.IOException{
40         StringBuffer fileData = new StringBuffer(1000);
41         BufferedReader reader = new BufferedReader(
42                 new FileReader(filePath));
43         char[] buf = new char[1024];
44         int numRead=0;
45         while((numRead=reader.read(buf)) != -1){
46             String readData = String.valueOf(buf, 0, numRead);
47             fileData.append(readData);
48             buf = new char[1024];
49         }
50         reader.close();
51         return fileData.toString();
52     }
53
54   private static String getPublicIPasString() {
55     String ipaddress = null;
56     try {
57       for (Enumeration<NetworkInterface> en = NetworkInterface.
getNetworkInterfaces(); en
58           .hasMoreElements();) {
59         NetworkInterface intf = (NetworkInterface)
en.nextElement();
60         for (Enumeration<InetAddress> enumIpAddr = intf.
getInetAddresses(); enumIpAddr
61             .hasMoreElements();) {
62           InetAddress inetAddress = (InetAddress) enumIpAddr
63               .nextElement();
64           if (!inetAddress.isLoopbackAddress()) {
65             ipaddress = inetAddress.getHostAddress().toString();
66             Log.e("ip address", "" + ipaddress);
```

```
67            }
68          }
69        }
70      } catch (SocketException ex) {
71        Log.e("Socket exception in GetIP Address of Utilities",
72            ex.toString());
73      }
74      return ipaddress;
75    }
76
77
78 }
```

mopenid_ws/associate.php

```
 1 <?php
 2
 3
 4 include_once 'conf/db.conf.php';;
 5 include_once 'conf/common.php';
 6 global $dbconfig;
 7
 8 define("MAXNUMBEROFIDENTIFIERS", 4);
 9
10
11
12 define("CREATE_NEW_IDENTIFIER", 1);
13 define("DELETE_IDENTIFIER", 2);
14 define("ASSOCIATE_IDENTIFIER", 3);
15 define("DISASSOCIATE_IDENTIFIER", 4);
16
17 $link = mysql_connect($dbconfig['host'], $dbconfig['user'],
$dbconfig['password'])
18     or die('Could not connect: ' . mysql_error());
19 mysql_select_db($dbconfig['database']) or die('Could not select
database');
20
21 $response = "";
22
23
24
25 class Association {
26
27     private $result = array("message" => "default msg");
28
29     function __construct() {
30     }
31
32
33     public function createNewIdentifier($identifier) {
34
35         if($this->newAccountAllowed($identifier)) {
36             //first time to create an account
37             $randString = new Local_RandomString(30);
38             $securityToken = strToHex($randString->getRand());
39             $digest = mysql_real_escape_string(securityTokenToDi
gest($securityToken)); //SHA-256 hash
40             $ip = $_SERVER['REMOTE_ADDR'];
41             if($result = @mysql_query("INSERT INTO links (id,
security_token, ip) VALUES ('$identifier', '$digest', '$ip');")) {
42                 $this->result['message'] = "inserted";
43                 $this->result['security_token'] =
$securityToken;
44             } else {
45                 throw new JSONErrorException("Identity already
exists. Please choose another one.");
46             }
47         } else {
48             $this->result['message'] = "failed";
49         }
50
51
52     }
```

```
 53
 54
 55      public function jsonResponse() {
 56          return json_encode($this->result);
 57      }
 58
 59      /*
 60       * @param String $parent
 61       */
 62      private function newAccountAllowed($identifier) {
 63          $result = @mysql_query("SELECT COUNT(*) AS number FROM
links WHERE id = '$identifier'");
 64          if(is_bool($result)) {
 65              throw new JSONErrorException();
 66          }
 67          if(@mysql_num_rows($result) > 0) {
 68              $row = @mysql_fetch_array($result);
 69              return $row['number']+1 < MAXNUMBEROFIDENTIFIERS;
 70          }
 71          return false;
 72      }
 73
 74      public function associate($id, $securityToken, $destination)
{
 75
 76          //first check to see if securitytoken is correct
 77          $digest = mysql_real_escape_string(securityTokenToDigest
($securityToken)); //SHA-256 hash
 78
 79          $result = mysql_query("SELECT security_token FROM links
WHERE enabled = 1 AND id = '$id'");
 80          if(mysql_num_rows($result) == 0) {
 81              @mysql_query("DELETE FROM active_links WHERE
identifier='$id'");
 82              throw new JSONErrorException("Error in
association");
 83          } else {
 84              $row = mysql_fetch_array($result);
 85
 86              if(($token = stripslashes($row['security_token']))
!= $digest) {
 87                    throw new JSONErrorException("Error in
association");
 88              }
 89
 90             // security check okay
 91               //see if there are any active
 92              $result = mysql_query("SELECT id FROM active_links
WHERE identifier = '$id'");
 93              if(mysql_num_rows($result) == 0) {
 94                  $result = mysql_query("INSERT INTO active_
links (identifier, destination, last_updated) VALUES ('$id',
'$destination', CURRENT_TIMESTAMP)");
 95              } else {
 96                  $result = mysql_query("UPDATE active_links SET
active = 1, destination = '$destination', last_updated = CURRENT_
TIMESTAMP WHERE identifier = '$id'");
 97              }
 98
 99              $this->result['message'] = "Link associated";
100
101
102          }
103
104      }
```

```
105
106     public function disassociate($id, $securityToken) {
107          //first check to see if securitytoken is correct
108         $digest = mysql_real_escape_string(securityTokenToDigest
($securityToken)); //SHA-256 hash
109         $result = mysql_query("SELECT security_token FROM links
WHERE enabled = 1 AND id = '$id'");
110         if(mysql_num_rows($result) == 0) {
111             throw new JSONErrorException("Error in
disassociation");
112         }
113         $row = mysql_fetch_array($result);
114
115         if(($token = stripslashes($row['security_token'])) !=
$digest) {
116                 throw new JSONErrorException("Error in
association");
117         }
118         // then update
119         @mysql_query("UPDATE active_links SET active=0 WHERE
identifier = '$id' LIMIT 1");
120         $this->result['message'] = "Link disassociated";
121
122     }
123
124     public function deleteIdentifier($id, $securityToken) {
125         $digest = mysql_real_escape_string(securityTokenToDigest
($securityToken)); //SHA-256 hash
126         $result = mysql_query("DELETE FROM links WHERE id =
'$id' AND security_token = '$digest' LIMIT 1");
127         if(mysql_affected_rows () > 0) {
128             $this->result['message'] = "ok";
129         } else {
130             $this->result['message'] = "failed";
131         }
132     }
133 }
134
135 /*
136  * Code for class Local_RandomString retrieved from http://www.
php.net/manual/en/function.mt-rand.php on 4 April 2011.
137  * @owner: nilesh at itech7 dot com 19-May-2010 07:20
138  */
139 class Local_RandomString {
140
141   protected $_length;
142   protected $_prevRand;
143
144   public function __construct($length = 15) {
145
146     $this->_length = $length;
147
148   }
149
150   public function getRand() {
151
152     $randStr = null;
153     $args[] = 'N' . $this->_length;
154
155     for($i = 0; $i < $this->_length; $i++) {
156       $args[] = mt_rand();
157     }
158
159     $randStr = substr(base64_encode((call_user_func_
array('pack', $args))), 1, $this->_length);
```

```
160     $this->_prevRand = $randStr;
161     return $randStr;
162
163   }
164
165   public function setLength($l) {
166
167     $this->_length = (int) $l;
168
169     if($this->_length <= 0) {
170       throw new Exception('Invalid random string length');
171     }
172
173   }
174
175   public function getPrevRand() {
176
177     return $this->_prevRand;
178
179   }
180
181 }
182
183
184 //this is where the action takes place
185 try {
186
187     $association = new Association();
188
189     $step = trim(mysql_real_escape_
string(htmlspecialchars(strip_tags(@$_GET['a']))));
190     if(!is_null($step)) {
191
192         switch($step) {
193
194             case CREATE_NEW_IDENTIFIER:
195
196                 $id = (@is_null($_GET['id'])) ? NULL :
trim(mysql_real_escape_string(htmlspecialchars(strip_tags($_
GET['id']))));
197                 $association->createNewIdentifier($id);
198                 break;
199
200
201             case DELETE_IDENTIFIER:
202                  $id = (@is_null($_GET['id'])) ? NULL :
trim(mysql_real_escape_string(htmlspecialchars(strip_tags($_
GET['id']))));
203                  $st = (@is_null($_GET['st'])) ? NULL :
trim(mysql_real_escape_string(htmlspecialchars(strip_tags($_
GET['st']))));
204                  $association->deleteIdentifier($id, $st);
205                 break;
206
207              case DISASSOCIATE_IDENTIFIER:
208
209                 $id = (@is_null($_GET['id'])) ? NULL :
trim(mysql_real_escape_string(htmlspecialchars(strip_tags($_
GET['id']))));
210                 $securityToken = (@is_null($_GET['st'])) ? NULL
: $_GET['st'];
211                 if(!is_null($id) && !is_null($securityToken)) {
212                     $association->disassociate($id,
$securityToken);
213                 } else {
```

```
214                     throw new JSONErrorException("Error in
parameters");
215                 }
216
217                 break;
218
219             case ASSOCIATE_IDENTIFIER:
220                 $id = (@is_null($_GET['id'])) ? NULL :
trim(mysql_real_escape_string(htmlspecialchars(strip_tags($_
GET['id']))));
221                 $securityToken = (@is_null($_GET['st'])) ? NULL
: $_GET['st'];
222                 $destination = (@is_null($_GET['d'])) ? NULL :
mysql_real_escape_string($_GET['d']);
223
224                 if(!is_null($id) && !is_null($securityToken) &&
!is_null($destination)) {
225                     $association->associate($id, $securityToken,
$destination);
226                 } else {
227                     throw new JSONErrorException("Error in
parameters");
228                 }
229
230                 break;
231
232             default:
233                 throw new JSONErrorException("Unknown action");
234         }
235
236
237     } else {
238         throw new JSONErrorException("Unknown action");
239     }
240
241
242
243
244     $response = $association->jsonResponse();
245
246 } catch (JSONErrorException $exc) {
247     $response = $exc->jsonResponse();
248 }
249
250
251
252
253
254
255
256
257
258
259 //print the resulting JSON string
260 header('Content-type: application/json');
261 echo $response;
262
263
264
265
266 mysql_close($link);
267
268
269 ?>
270
```

mopenid_ws/exists.php

```
<?php


include_once 'conf/db.conf.php';;
include_once 'conf/common.php';
include_once '../config.php';
global $dbconfig;



$response = array();

if(!isset($_GET['id'])) {
    $response['message'] = "missing_id";
    echo json_encode($response);
    return;
}

if(!preg_match('/[^A-Za-z0-9]/', $_GET['id']) && strlen($_
GET['id']) >= MINIMUM_NUMBER_OF_CHARACTERS_IN_ID) {
    $link = mysql_connect($dbconfig['host'], $dbconfig['user'],
$dbconfig['password'])
    or die('Could not connect: ' . mysql_error());
    mysql_select_db($dbconfig['database']) or die('Could not
select database');
    $id= mysql_real_escape_string($_GET['id']);
    $query = "SELECT id FROM links WHERE id = '$id' AND enabled =
1 LIMIT 1";
    if(isset($_GET['st'])) {
        $securityToken = mysql_real_escape_string($_GET['st']);
        $digest = mysql_real_escape_string(securityTokenToDigest(
$securityToken)); //SHA-256 hash
        $query = "SELECT id FROM links WHERE id = '$id' AND
enabled = 1 AND security_token = '$digest' LIMIT 1";
    }
    $result = @mysql_query($query);
    if(mysql_num_rows($result) > 0) {
        $response['message'] = "yes";
    } else {
        $response['message'] = "no";
    }

    echo json_encode($response);

    mysql_close($link);
} else {
    echo json_encode(array("message"=> "invalid_id"));
}



?>

```

index.php

```
<?php
/*
 * This is the code generating the ID page used when RPs
 * are discovering the OP endpoint URL
 *
 * Modified by Eirik Stien - eiriksti@stud.ntnu.no
 * April 2011
 */

if(isset($_GET['error'])) {
    echo "error 404";
    die();
}

header('Cache-Control: no-cache');
header('Pragma: no-cache');

require_once 'config.php';
require_once 'lib/session.php';
require_once 'lib/actions.php';

if(isset($_GET['id']) && is_string($_GET['id'])) {
    if(!preg_match('/[^A-Za-z0-9]/', $_GET['id']) && strlen($_
GET['id']) >= MINIMUM_NUMBER_OF_CHARACTERS_IN_ID) {

        if(!is_null($destination =
getDestinationifIdIsActiveAndLinked($_GET['id']))) {
            $resp = action_idpage($destination);
            writeResponse($resp);
        } else {
            echo "not an valid id";
        }

    }
} else if(isset($_GET['userXrds']) && is_string($_
GET['userXrds'])) {
    if(!preg_match('/[^A-Za-z0-9]/', $_GET['userXrds']) &&
strlen($_GET['userXrds']) >= MINIMUM_NUMBER_OF_CHARACTERS_IN_ID) {

        if(!is_null($destination =
getDestinationifIdIsActiveAndLinked($_GET['userXrds']))) {

            $resp = action_userXrds($destination);
            writeResponse($resp);
        } else {
            echo "not an valid id";
        }
    }
}

?>

```