CSIS0801 Final Year Project (2011-2012)

Interim Report

FYP11015 A Secure Mobile System to Support Citizen Journalism

Supervisor: Dr. S.M. Yiu

Lam Wan Yin, May 2009148398

Other Group Members:
Li Ka Yau, Ray 2009566813
Lee San Tai, Eric, 2009565273
Lam Harry To Chun, 2009177272
# Tables of Contents

1. **Introduction** ................................................................. 3
   Timeline of my part: ....................................................... 4

2. **System Flow** ............................................................... 4
   A. Traffic Jam in Central ................................................. 4
   B. Andy Lau in Cyberport ............................................... 5
   C. Scene of Fire ........................................................... 6

3. **System Details** ............................................................. 7
   A. Windows Azure Platform ............................................ 8
   B. Web Portal .................................................................. 8
      Access Control ......................................................... 9
      SQL Azure and Azure Blob ....................................... 10
      Bing Map .................................................................... 12
   C. Image Service .......................................................... 14
   D. SQL Azure ............................................................... 15

4. **Further Plan** ............................................................... 17
   A. Transaction ............................................................. 17
   B. Anonymous Upload .................................................. 17
   C. Reporting System ..................................................... 17
1. Introduction

Thanks to the rapid growth of technology, most of us have a smartphone with us all the time and being able to access to Internet wherever and whenever. This gradually trains us to become journalists who love to tell friends or even strangers on what is happening around us. One of the most common ways to share news is by uploading photos to social networking system such as Facebook.

However, for sharing photos on Facebook, the uploading steps are complex and the processing time is tedious. Also, the uploaded photos are not well-organized which makes users hard to retrieve their desired news from Facebook. Moreover, the targeted audience is limited to friends only on Facebook. As a result, we can hardly get the news from stranger and we seldom do that too as it may disclose other personal information which are posted on Facebook.

Furthermore, other than simply sharing the photos with friends, we should be able to do more than that. The photos that we take may have news value and may be desired by news agents. However, many common applications such as Facebook don’t support the selling of photos to those news agents. This not only deprive the chance for each smartphone users to earn money by taking valuable photos, but also make the progress of the data collection of news agents to be slow and ineffective.

Therefore, we develop this secure mobile system to support citizen journalism to solve the above problems. We developed an integrated Windows Phone app which allows users to share their photos to everyone or solely to news agents in a straightforward way. The localized features such as recording the location of the photo taker and allowing users to choose the category of the photo makes the photos to be more organized and easier to access. The security linkage in our apps ensures the integrity and confidentiality of the photos can be ensured so that news agents will have confidence in buying those photos. The web portal of our system provides an alternative way for users to access those photos and buy photos on PC. All of these above reasons make our system special and unique.
Timeline of my part:

<table>
<thead>
<tr>
<th>Task</th>
<th>From</th>
<th>To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Studying and setting up Azure</td>
<td>Early September</td>
<td>Late September</td>
</tr>
<tr>
<td>Set up of the Azure</td>
<td>Early September</td>
<td>Mid October</td>
</tr>
<tr>
<td>Building Image service (phone)</td>
<td>Late September</td>
<td>Late October</td>
</tr>
<tr>
<td>Set up of SQL Azure</td>
<td>Mid Sep</td>
<td>Late October</td>
</tr>
<tr>
<td>Building the web portal</td>
<td>Late October</td>
<td>On going</td>
</tr>
<tr>
<td>Linking up web portal and the SQL Azure and Azure Blob</td>
<td>Late October</td>
<td>Early November</td>
</tr>
<tr>
<td>Implementing the filtering function (Web Portal)</td>
<td>Early November</td>
<td>Mid November</td>
</tr>
<tr>
<td>Implementing the Map function (Web Portal)</td>
<td>Mid November</td>
<td>Late November</td>
</tr>
<tr>
<td>Implementing the “Like”, “Dislike” and “Buy” functions (Web Portal)</td>
<td>Early December</td>
<td>Mid December</td>
</tr>
<tr>
<td>Implementing the Access Control Features (Web Portal)</td>
<td>Mid December</td>
<td>Late December</td>
</tr>
<tr>
<td>Testing on the web portal</td>
<td>Mid December</td>
<td>On going</td>
</tr>
</tbody>
</table>

2. System Flow

In the following paragraph, I will describe different scenarios to explain the system flow clearly.

A. Traffic Jam in Central

Peter got stuck in a traffic jam in Central. He wants to share this news with others and remind them to avoid it. So, he can use his Windows Phone which already installed our app to take a photo of the traffic jam, set the category as “local news” and add comments to it. Then, he can upload the photo and the application will record the GPS location of Peter and save it in the database at the same time.

At the same time, Mary wants to go to Central. But first, she wants to explore the news in Central, opens her app on the Windows Phone. Then, she opens the map in the app and navigates to Central. In this situation, Mary can see the photo uploaded by Peter to show there
is traffic jam in Central. Thus, she can know this news and decide whether she will go to Central or not.

This scenario shows that we can know what’s happening around us by using our apps.

**B. Andy Lau in Cyberport**

Andy Lau, the well-known celebrity of Hong Kong, is in the cinema in Cyberport now. Someone took a photo to prove this and upload it via our app. Quickly all the fans of Andy access our portal to browse this photo and ‘like’ it. They can access our web portal by any PC. Thus, all people, not only Windows Phone users can browse the photo too.

To search the photo easier, his fans can use the date and the category to do the filtering.

And as the same on the phone, users can search the photo by navigating on the map.

After a while, the photo of Andy has many ‘like’ already. For the fans who installed our app on their Windows Phone, they can also find that photo easier by browsing the “Top Like” in the app. As in the web portal, users can select the categories of photos to be shown easier by clicking the button on the application bar at the bottom of the page.

This scenario shows users can find the photos which they are most interesting by the aid of the system of categories, ‘Like’ and ‘Dislike’ and the Map function. Also, the public photos are not only limited Windows Phone users. All people can browse those photos
via the web portal too. Therefore, this system is user-friendly and multi-platform which can serve the requirements of different users.

C. Scene of Fire

Suddenly, the fire alarm is on. The building which Peter in is on fire. Although he is safe now and be able to take some photos on the scene of fire via our Windows Phone app, only he can do so since no one is allowed to go inside the building. Peter realized that those photos are exclusive and many news agents want to buy them to. However, how can the news agents trust those photos are worthiness but at the same time Peter don’t need to disclose the full resolution of the photos as this may lower the value of those photos?

In this situation, Peter can use our app to upload those photos. We will upload the E_SymmetricKey_(Photo), E_PublicKeyofTheBuyer_(SymmetricKey), Hash(Photo) and the thumbnail of the photo and prepare for sale. So the news agents can browse the thumbnail of the photos and decide whether buy them or not. If the news agents buy the photos, he will get the E_SymmetricKey_(Photo), E_PublicKeyofTheBuyer_(SymmetricKey), Hash(Photo). As the private key is saved at the phone of the news agents, only he can decrypt the symmetric key and use the key to decrypt the photos. Moreover, the hash value can be used to check the integrity of the photos.

In another words, the news agents can only view the thumbnail of the photos. This can protect Peter by preventing others to access the full resolution of the photos. And this also can protect the news agents by letting them to preview the photos. So, this is one of the best solutions for Peter to sale his photos easily.
3. System Details

In this system, there are six different components: Windows Phone app, Web Portal, Image Service, SQL Service, SQL Azure, and Azure Blob.

The end users can access our system via the Windows Phone app or via the web portal. Then, these applications will save the records of users, details of photos and details of transaction on SQL Azure. The actual photos (the original, the thumbnail or the encrypted photos) will be saved on the Azure Blob.

And for the Windows Phone, two Windows Communication Foundation projects (SQL Service and Image Service) are needed in order to connect the SQL Azure and the Azure Blob.

In this system, I mainly contributed in these parts:
A. **Windows Azure Platform** *(Contributor: May Lam)*

In our project, we used Windows Azure is one of our main components. However, we are not very familiar with this new platform at first. After studying lots of materials of Windows Azure, I finally come up a way to create a cloud project on Visual Studio and deploy them on Windows Azure Platform. Up to now, we have deployed three web roles (Web Portal, Image Service and SQL Service) They are all hosting on Windows Azure and be able to access at anywhere.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Status</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>team1</td>
<td>Subscription</td>
<td>Active</td>
<td></td>
</tr>
<tr>
<td>team1</td>
<td>Hosted Service</td>
<td>Created</td>
<td></td>
</tr>
<tr>
<td>WebPortal</td>
<td>Role</td>
<td>Ready</td>
<td>Staging</td>
</tr>
<tr>
<td>WebPortal_LIN_0</td>
<td>Instance</td>
<td>Ready</td>
<td>Staging</td>
</tr>
<tr>
<td>SQLService</td>
<td>Role</td>
<td>Ready</td>
<td>Staging</td>
</tr>
<tr>
<td>SQLService_LIN_0</td>
<td>Instance</td>
<td>Ready</td>
<td>Staging</td>
</tr>
<tr>
<td>ImageServices</td>
<td>Role</td>
<td>Ready</td>
<td>Staging</td>
</tr>
<tr>
<td>ImageServices_LIN_0</td>
<td>Instance</td>
<td>Ready</td>
<td>Staging</td>
</tr>
</tbody>
</table>

B. **Web Portal** *(Contributor: May Lam)*

The web portal is an alternative way for users (even non Windows Phone users) to be able to access our system and browse the photos by date, by categories and by locations. The web portal is implemented by ASP.Net and hosted on the Azure as a web role. The web portal also leverages other services such as Access Control on Azure, SQL Azure, Azure Blob and Bing Map.
**Access Control**

One of the main objectives of our system is to make it user friendly but also secured. In order to achieve this goal, I chose to use Access Control to handle the log in system. Access Control is one of the features provided by Windows Azure platform. It allows user to log in the system by different accounts (e.g. Windows Live ID, Yahoo!, Gmail etc.) which most of the users already have created at least one account. Therefore, our goal of user-friendly can be achieved as users don’t need to create another account and remember the other set of password.

After the users type in the log in details (which is done on the page of the service provider), the Access Control will validate the user and issue a secure token to the application. In the process for the users to enter the log in details, the application will not and cannot access any information about the log in details. As a result, the confidential of the log in details can be kept.

Also, by using the Access Control, the web portal doesn’t need to implement a log in system and keep the confidential in our system. Thus, this can enhance our security of the system.

And here is the actual flow of how the web portal uses the Access Control:
1. User go to the URL of the web portal and request to log in
2. The web portal will check whether the user have a secure token or not. If not, it will request one from the Access Control
3. The Access Control will direct the user to the login page of the service provider (Windows Live in this system) and ask the user to log in.
4. The Access Control will validate the user
5. If the user is valid, the Access Control will return a secure token to the web portal.
6. The web portal will check whether the token is valid or not. If it is valid, the web portal will direct the user to the home page of the web portal.
After adding the access control to the web portal, then I added the following code to get the carrier information and the user ID to identify the user:

```csharp
protected void Page_Load(object sender, EventArgs e)
{
    var identity = User.Identity;
    // get provider
    provider = string.Empty; 
    var claimsIdentity = identity as ClaimsIdentity;
    if (claimsIdentity.IsAuthenticated == null)
        provider = string.Empty;
    var providerQuery = from c in claimsIdentity.Claims
                        select c.Value;
    provider = providerQuery.FirstOrDefault();
    // get nameId
    nameId = string.Empty;
    claimsIdentity = identity as ClaimsIdentity;
    if (claimsIdentity.IsAuthenticated == null)
        nameId = string.Empty;
    var nameIdQuery = from c in claimsIdentity.Claims
                       where c.ClaimType == "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier"
                       select c.Value;
    nameId = nameIdQuery.FirstOrDefault();
    addUser();
}
```

**SQL Azure and Azure Blob**

The records of the photos (e.g. uploader id, photo path, photo id etc.) are saved on the SQL Azure. In the main page, the web portal will get those records by dates and tags specified by the users and displays the details of the page (e.g. name, number of likes and dislikes, comments etc.). Then, the web portal will get the photo from Azure Blob by using the path specified on the SQL Azure.

Users can browse the photos, click the photo to enlarge it and the portal will automatically add the number of view of that photo. Users can also like or dislike on particular photos. And user can ‘buy’ a photo via the web portal. But on this stage, web portal will only add a transaction record on the SQL Azure and no further follow up.

To connect the web portal to SQL Azure, I define the connection string as:

```csharp
<add name="MyConnectionStrings"
<add name="Data Source=myDatabase;Initial Catalog=MyDatabase;User ID=myUsername;Password=myPassword;multipleactiveresultsets=true" providerName="System.Data.SqlClient" />
<add name="ApplicationServices" connectionString="data source=\SQLExpress\Integrated Security=SSPI;AttachDBFilename=|DataDirectory|myDatabase.mdf;User Instance" />
```
After defining the connection string, I can use it in List View to show the photos and use it in Configuration Manager to access the database.

To implement the “Like”, “Dislike” and “Buy” functions, I implemented a ListView OnItemCommand to do so.

```csharp
protected void ListView1_OnItemCommand(object sender, ListViewCommandEventArgs e)
{
    string command_str = ConfigurationManager.ConnectionStrings["MREConnectionString"];
    SqlConnection conn = new SqlConnection(command_str);
    conn.Open();
    if (e.CommandName.CompareTo("LikePhoto") == 0)
    {
        string squpdate = "UPDATE tags SET likenum = likenum + 1 WHERE photoId = (SELECT id FROM photo WHERE path = "" + photopath + ")";
        SqlCommand cmdupdate = new SqlCommand(squpdate, conn);
        int row_count = cmdupdate.ExecuteNonQuery();
    }
    if (e.CommandName.CompareTo("DislikePhoto") == 0)
    {
        string squpdate = "UPDATE tags SET dislikenum = dislikenum + 1 WHERE photoId = (SELECT id FROM photo WHERE path = "" + photopath + ")";
        SqlCommand cmdupdate = new SqlCommand(squpdate, conn);
        int row_count = cmdupdate.ExecuteNonQuery();
    }
    if (e.CommandName.CompareTo("Buy") == 0)
    {
        string sqlbuy = "INSERT INTO buy (buyerid, photoid, transid, time) VALUES (" + buyerid + ", (SELECT id FROM photo WHERE path = "" + photopath + ""), newid(), CURRENT_TIMESTAMP)"
        SqlCommand cmdbuy = new SqlCommand(sqlbuy, conn);
        int row_count = cmdbuy.ExecuteNonQuery();
        LabelBuy.Text = "Bought!";
    }
    conn.Close();
    ListView1.DataBind();
}
```

For adding filtering photos by date, I added a selectionchanged event handler for the two calendars.

```csharp
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    string change = changelist;
    enddate = Calendar2.SelectedDate.ToString("dd/MM/yyyy");
    if (begindate != ""
    { changecal = "((time >= convert(date, '"' + begindate + ", 103') and (time <= convert(date, '"' + enddate + ", 103'))";
    }
    else { changecal = "((time <= convert(date, '"' + enddate + ", 103'))
    }
    string sql = "select * from [summary] where " + changecal;
    if (change != ""
    { sql = " and (" + change + ")";
    }
    ListView1.DataSource = sql2;
    ListView1.DataBind();
}
```
To change filter the photos after clicking different tags, I added this SelectIndexChanged function to the ListBox:

```csharp
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    string change = changecal;
    string s = "SELECT * FROM [summary];
    int[] intSelectedIndexes = new int[ListBox1.GetSelectedIndices().Length];
    intSelectedIndexes = ListBox1.GetSelectedIndices();
    changelist = "";
    for (int i = 0; i < ListBox1.GetSelectedIndices().Length; i++)
    {
        if (i == 0)
            s += " where";
        else
            { 
                s += " or";
                changelist += " or";
            }
        int num = intSelectedIndexes[i] + 1;
        s += " (tag1 = " + num + ")";
        changelist += " (tag1 = " + num + ")";
    }
    if (change != "")
    { 
        s += " and " + change;
    }
    ListViewDataSource.SelectCommand = s;
    ListView1.DataBind();
}
```

### Bing Map

The web portal also allows user to view the photos by locations. The web portal will get the locations of all photos from the records on SQL Azure and get the photo from Azure Blob. Then, display those photos on the map by using push pins on recorded location.

For the development this web portal, the basic structure of the web portal is relatively a routine process as I can use ASP.NET web page to construct the web portal. However, I need to cope with different problems while deploying the web portal on Windows Azure Platform.

Although there do have document describing basic steps for how to develop a cloud application, sometimes the basic steps do not help much.
For example, I tried to add the Access Control on the web portal before. It runs perfectly on local but get error while deploying on Windows Azure. Comparing to other web platform such as traditional server, Windows Azure is not that common. Thus, this causes there are very few document or tutorials guiding me or give me inspiration on how to do it. In the above case, after giving much effort, I finally find an article describing I should set something before I deploy the application on Windows Azure which those settings are not mentioned in the official document of MSDN.

In the above case, after giving much effort, I finally find an article describing I should set something before I deploy the application on Windows Azure which those settings are not mentioned in the official document of MSDN.

And I used the Configuration Manager in the Map.aspx.cs page to retrieve the photos records from SQL Azure and pass them to Javascript

protected void Page_Load(object sender, EventArgs e)
{
    string conn_str = ConfigurationManager.ConnectionStrings["WFAREConnectionString"].ConnectionString;
    SqlConnection conn = new SqlConnection(conn_str);
    conn.Open();
    string sqlAddlike = "SELECT * FROM photo WHERE (security = 0)";
    SqlCommand cmdAddlike = new SqlCommand(sqlAddlike, conn);
    SqlDataReader reader = cmdAddlike.ExecuteReader();
    StringBuilder sb = new StringBuilder();
    int i = 0;
    while (reader.Read())
    {
        sb.AppendFormat("lat{i}=(\"{0}\"),\r\n        " + reader["latitude"].ToString() + ",\r\n        " + reader["longitude"].ToString() + ",\r\n        " + reader["photo"].ToString() + ",\r\n        " + reader["uploaderId"].ToString() + ",\r\n        " + reader["uploaderName"].ToString() + ");
    }
    ScriptManager.RegisterClientScriptBlock(this, this.GetType(), "PointArrays", sb.ToString(), true);
    reader.Close();
    reader.Dispose();
    conn.Close();
}
C. **Image Service** (Contributor: May Lam, Harry Lam)

In order to let the user to share their photos, the photos or the encrypted file will be uploaded to Windows Azure Blob Storage. However, as the phone can’t direct access to Windows Azure Blob Storage, we need to build up a Windows Communication Foundation which we named it as Image Service to server as the middle man between the Windows Phone app and the Windows Azure Blob Storage. After creating the image service, then we can add it as a service reference of the phone app and the phone app will be able to upload photos to Windows Azure Blob Storage.

And here’s the flow of photo uploading process:
1. The user takes a photo and the photo will be saved at Isolated Storage which is allocated on the phone
2. The phone app creates a new class called ‘CloudBlobUploader’ and save the photo as a stream file in it.
3. Then the phone app calls the Image Service (which should be deployed on Azure already) and pass the stream file to it.
4. The Image Service connects with the Azure Blob and uploads the stream file on it.

Here’s the code for how we handle the upload process:

```csharp
public void uploadObject()
{
    var client = new UploadServiceClient();
    client.UploadUrlWithSharedAccessSignatureCompleted += UploadUrlWithSharedAccessSignatureCompleted;
    client.UploadUrlWithSharedAccessSignatureAsync(buildNewBuild().ToString());
}

private void UploadUrlWithSharedAccessSignatureCompleted(object sender,
    UploadUrlWithSharedAccessSignatureCompletedEventArgs e)
{
    if (e.Error != null)
        MessageBox.Show(e.Error.Message.ToString());
    if (e.Error == null)
    {
        // Determine upload path - Add filename to container path
        var builder = new UriBuilder(e.Result);
        builder.Path = builder.Path + “/” + filename;
        // Open the image file from isolates storage to read from IsolatedStorageFileStream file =
        IsolatedStorageFile.GetUserStoreForApplication().OpenFile(filename, FileMode.Open, FileAccess.Read);
        // Create the uploader and kick off the uploader
        var uploader = new CloudBlobUploader(file, blobUrl.AbsoluteUri);
        var pos = blobUrl.AbsoluteUrl.ToString().IndexOf(‘?’);
        var url = blobUrl.AbsoluteUrl.ToString().Substring(0, pos);
        uploader.UploadFinished += (s, args) =>
        {
            IsolatedStorageFile.GetUserStoreForApplication().DeleteFile(filename);
            uploadCompleted(this, e);
        };
        uploader.StartUpload();
    }
}
```

For the ease of the implementation, the container of the cloud blob is set to public. That’s mean all people can download the file on the Azure Blob as long as they have the URL of the file. This decision is made after our long consideration and we are quite sure that this will not causer any security problems.
It is because we will only upload the photos and the encrypted files to the Azure Blob. For the photos which are set as public, the photo takers are intended to share the photos with everyone, so it will not cause any problems when the container is set to public and everyone can access it by the URL. For the photos which are set as private, that’s mean the photo takers want to keep the photos private and hold it for sale. In this case, the phone app will only upload the encrypted photos and only people who get the symmetric key can decrypt the file. Therefore, it is meaningless even if other people can download the encrypted file while they get the URL of it.

But now this service is not fully implemented as it is still a public service. In the other words, whoever implemented the Windows Phone app as the way we did and add our Image Service as their Service reference, they will be able to upload files to our Azure Blob. We understand this is a serious problem and will cause numerous security concerns. We will fix this inadequate part as soon as possible by setting the Image Service as a private service.

D. SQL Azure (Contributor: May Lam)

Database acts as an rather important role in this project. It will save all the records of users, details of the photos and the records of transactions. Part of my resopibilities is to build up the SQL Azure (the Database which host on Windows Azure Platform) and implement the details for each tables and view.

Up to now, I have created four tables and three views. And here’s the detials of the tabels:

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>photo</td>
<td>id (int)</td>
</tr>
<tr>
<td></td>
<td>uploaderid (int)</td>
</tr>
<tr>
<td></td>
<td>security (int)</td>
</tr>
<tr>
<td></td>
<td>longitude (text)</td>
</tr>
<tr>
<td></td>
<td>hash (text)</td>
</tr>
<tr>
<td></td>
<td>time (datetime)</td>
</tr>
<tr>
<td></td>
<td>path (varchar 500)</td>
</tr>
<tr>
<td></td>
<td>thumbpath (text)</td>
</tr>
<tr>
<td></td>
<td>keycipher (varbinary (max))</td>
</tr>
<tr>
<td>tags</td>
<td>photoid (int)</td>
</tr>
<tr>
<td></td>
<td>tag1 (int)</td>
</tr>
<tr>
<td></td>
<td>dislikenum (int)</td>
</tr>
<tr>
<td></td>
<td>comment (text)</td>
</tr>
<tr>
<td></td>
<td>viewnum (int)</td>
</tr>
<tr>
<td></td>
<td>likenenum(int)</td>
</tr>
<tr>
<td>user_table</td>
<td>id (int)</td>
</tr>
<tr>
<td></td>
<td>name (varchar 50)</td>
</tr>
<tr>
<td></td>
<td>liveID (varchar 50)</td>
</tr>
<tr>
<td></td>
<td>publickey (varchar(max))</td>
</tr>
<tr>
<td>buy</td>
<td>buyerid (int)</td>
</tr>
<tr>
<td></td>
<td>photoid (int)</td>
</tr>
<tr>
<td></td>
<td>tranid (uniqueidetifier)</td>
</tr>
<tr>
<td></td>
<td>time (datetime)</td>
</tr>
</tbody>
</table>
For the views, I created two which linking the tables of user_table, tags and photo for the further development of “Photo” features on the Windows Phone app and for the data collection of the web portal.

Here’s the screen capture of the portal of SQL Azure:

![Image of SQL Azure portal]

Different from the traditional way, the database is developed and host on the Windows Azure platform. It is rather easy to set up as its components are similar to traditional database. The major difficulty is the SQL Azure is using Transact SQL instead of My-SQL. Although most of the syntax are the same, some are difference. For example, in My-SQL, I will set “AUTO_INCREMENT” as a property of the ID to perform the auto increment. However, in Transact SQL, I need to set the property as “IDENTITY(1,1)” in order to achieve the same goal. But after doing some research, I still be able to overcome this difficulty.

Also, the SQL Azure has a relatively strict requirement on the structure of the tables. For example, tables must have clustered index and primary index before using it. However, SQL Azure doesn’t allow me to drop the clustered index if I create it wrongly. The only solution is to drop the table and create another one with a correct clustered index. This is really annoying sometime for creating and dropping tables repeatly.
4. Further Plan

Although we have already implemented many features in our system, we won’t stop here. We will try our best to add more functions in our system, test it and enhance it at the rest of the project. In the coming months, we planned to do this:

A. Transaction

At this stage, when a user clicks the ‘buy’ button on the web portal, we can only add the records of that transaction in our database with no further process. In the coming months, we will complete the function by implementing the following steps:

1. The seller takes a photo and uploads the E_SymmetricKey_(Photo)
2. Buyer clicks the ‘buy’ button on the web portal
3. The seller uploads E_PublicKey_(Symmetric Key)
4. The buyer receives the E_SymmetricKey_(Photo) and E_PublicKey_(Symmetric Key) so he can use his private key to decrypt the Symmetric Key and use the Symmetric Key to decrypt the photo

B. Anonymous Upload

In some situation, users may not want to disclose his/her identity while uploading the photos. For example, Peter saw a Policeman is abusing a lady and he took a photo on it. Peter wants to share this to the others but don’t want the others know he’s the uploader of that photo to avoid any consequence.

To solve this problem, we will implement the anonymous upload function so that Peter can upload the photos by no one know that’s uploaded by him. To achieve this goal, we will use ring signature to let the user to generate some sets of public keys and private keys. As a result, we can ensure the uploader is a valid user but the system still doesn’t know who the uploader is.

C. Reporting System

As all people who installed our apps on their Windows Phone can upload photos to our system too, some users may try to abuse our system by uploading meaningless photos or even some are pornographies. In order to solve this problem, we are going to use a reporting system on our phone app and also on our web portal. Users can report spam to particular photos and we will removed that photos if we receive a number of report.