Chapter

14

FriendsZone: User Management & Authentication

At the end of this chapter, you will be able to:

- Design a user interface to login a user
- Login to the FriendsZone application
- Design a user interface to create a new user
- Upload an image to the server
- Save a new user

Instruction 1: Design a user interface to login a user

1. Mobile applications, especially those that require the user to connect to a remote server need to be secure. The most basic way to secure the access to a remote server is with a username and password. This is what we will cover in this chapter.

2. In the previous chapter, we had a glimpse of the list of PHP web services that have been written for FriendsZone. We will zoom in on login.php.

3. Start your Web, PHP and MySQL Database Server. If you are running a local XAMPP server, run xampp_start.exe at \xampp\.

4. Open your Internet browser and enter the following URL:

```
http://<your URL>/friendszone/login.php?userid=jimmy&password=jimmy
```

   In the URL above, we are running the login.php script with 2 parameters

   - userid: jimmy
   - password: jimmy

5. The follow JSON string is returned by login.php:

```
[{"result":"1"}]
```

   A result of 1 shows that the correct userid and password has been entered. Try running login.php again with a different password such as "peter". Note that the JSON string returns a 0 value. It means that a wrong userid or password has been entered.

```
[{"result":"0"}]
```

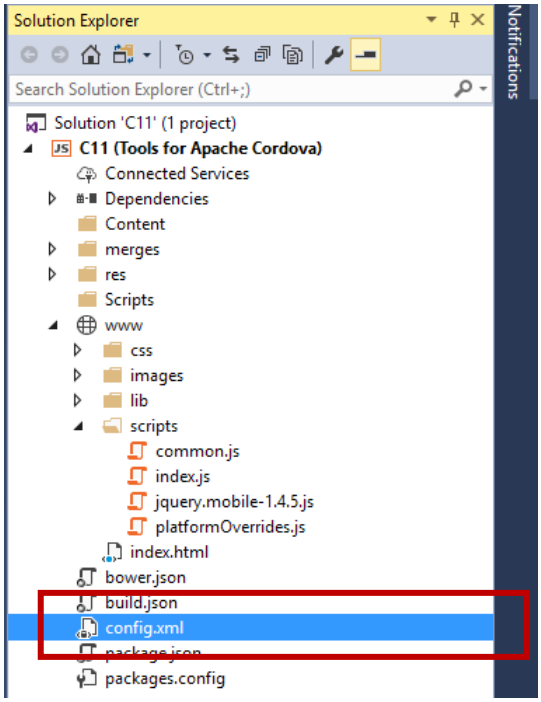
6. We will now develop a Cordova application to make the call to login.php. We will begin by developing the application's user interface.

7. Create a new project, Friendszone.

8. Follow Instruction1 in Chapter 4 to include jQuery and jQuery Mobile libraries into your project.

9. Follow Instruction3 in Chapter 8 to include the jQuery Validation plugin into your project.

10. In Solution Explorer, double click www/config.xml to open it.



11. Click "Plugins", Install the following plugins:

    a. Camera: We will be using the mobile device's camera function

    b. File: Required to upload a picture from the picture gallery of the mobile device to the friendszone server.

    c. File Transfer: Required to upload a picture from the picture gallery of the mobile device to the friendszone server.

    d. Notification: We will use notifications to pop up information and error messages instead of using the alert() function.

12. Enter the following codes into index.html:

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4      <!--
5          Customize the content security policy in the meta tag below as
   needed. Add 'unsafe-inline' to default-src to enable inline JavaScript.
6          For details, see http://go.microsoft.com/fwlink/?LinkID=617521
7      -->
8          <meta http-equiv="Content-Security-Policy" content="default-src
   'self' data: gap: http://jacksonng.org https://ssl.gstatic.com 'unsafe-
   eval'; style-src 'self' 'unsafe-inline'; media-src *">
9
10         <meta http-equiv="content-type" content="text/html; charset=UTF-
   8" />
11         <meta name="format-detection" content="telephone=no">
12         <meta name="msapplication-tap-highlight" content="no">
13         <meta name="viewport" content="user-scalable=no, initial-
   scale=1, maximum-scale=1, minimum-scale=1, width=device-width">
```

```
14
15          <link rel="stylesheet" type="text/css" href="css/index.css">
16          <link rel="stylesheet" href="css/jquery.mobile-1.4.5.css">
17
18          <script src="lib/jquery-1.11.2.min.js"></script>
19          <script src="lib/jquery.mobile-1.4.5.min.js"></script>
20          <script src="lib/jquery.validate.min.js"></script>
21          <script src="scripts/common.js"></script>
22          <script src="scripts/index.js"></script>
23
24          <title>Friendszone</title>
25      </head>
26      <body>
27          <div data-role="page" id="front-page">
28              <div data-role="header" class="header" data-
position="fixed">
29                  <h1 id="app-title">Friendszone</h1>
30              </div>
31
32              <div role="main" class="ui-content" id="main-ui">
33                  <div data-role="content" class="ui-content"
id="loginsection">
34                      <form name="LoginForm" id="LoginForm">
35                          <div data-role="fieldcontainer">
36                              <label for="userid">User ID</label>
37                              <input type="text" name="txtLogin"
id="txtLogin" placeholder="enter user ID" value="jimmy" required>
38                          </div>
39
40                          <div data-role="fieldcontainer">
41                              <label for="password">Password</label>
42                              <input type="text" name="txtPassword"
id="txtPassword" placeholder="enter password" value="jimmy" required>
43                          </div>
44
45                          <div class="ui-grid-a">
46                              <div class="ui-block-a">
47                                  <input type="button" id="btnLogin"
value="Login">
48                              </div>
49                              <div class="ui-block-b">
50                                  <input type="button" id="btnNewUser"
value="New User">
51                              </div>
52                          </div>
53                      </form>
54                  </div>
55              </div>
56
57              <footer data-role="footer" data-position="fixed">
58                  <h4>Copyright 2017 OTBS</h4>
59              </footer>
60          </div>
61          <script type="text/javascript" src="cordova.js"></script>
62          <script type="text/javascript"
63 src="scripts/platformOverrides.js"></script>
```

```
64      </body>
65  </html>
```

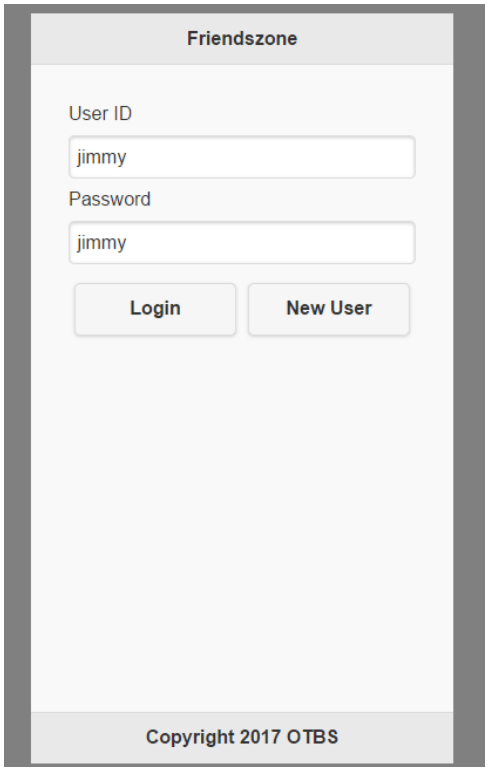
a. Line 8: The app is configured to access only external sites that are stated here. I have stated that it is allowed to access http://jacksonng.org because this is where my friendszone server is. Change this value to the location of your friendszone server (e.g. http://projectsbit.org.

b. Lines 35 to 38 draw the textbox for users to enter a User ID. Note that a default value "jimmy" has been set. We will never do this in a real application, but we are doing it here so that you do not have to re-enter the user ID every time you run the application. You may remove this default value of "jimmy" once the application is tested to work.

c. Lines 40 to 43 draw the textbox for users to enter a password. Note that a value "jimmy" has been entered as a default value for this password field. Again, we will never do this in an actual application. This is done to save us time so that we do not need to retype the password whenever we test our application. In an actual application, we would have used '**input type="password"**' so that what we type cannot be seen on the screen.

d. Lines 45 to 52 draw the 2 buttons [Login] and [New User]

13. Run the app and note the new user interface to enter a User ID and Password.



Instruction 2: Login to the FriendsZone application

1. The URL for our PHP Web Services in our Virtual Private Server is:

```
http://<your URL>/friendszone/login.php?userid=jimmy&password=jimmy
```

If you are running the PHP Web Services in a local XAMPP server, indicate http://10.0.2.2 as the XAMPP Web Server address.

```
http://10.0.2.2/friendszone/login.php?userid=jimmy&password=jimmy
```

2. To save us the trouble of retyping this address every time we make a call to the server, we will save this address as a variable in our common.js file. Open common.js and type in the following function:

```
function serverURL(){
      return "http://<your url>/friendszone";
}
```

3. Add the following function to common.js. The validationMsg() function will pop an error message if the app is unable to connect to your friendszone server.

```
function validationMsg() {
        validationMsgs("Unable to connect to server. Please try again
        later.", "Connection Problem", "OK");
}
```

4. Edit index.js:

```
(function () {
    "use strict";

    var userid;
    var password;

    $(document).ready(function () {

        $("#LoginForm").validate({
            messages: {
                txtLogin: "User ID is required",
                txtPassword: "Password is required",
            },
            focusInvalid: false,
            submitHandler: function () {
                return false;
            },
```

```
18              errorPlacement: function (error, element) {
19                  error.appendTo(element.parent().parent().after());
20              },
21          });
22
23          $("#btnLogin").bind("click", function () {
24              if ($("#LoginForm").valid()) {
25                  login();
26              }
27          });
28
29
30
31
32
33      });
34
35      function login() {
36          var url = serverURL() + "/login.php";
37          var result;
38          userid = $("#txtLogin").val();
39          password = $("#txtPassword").val();
40
41          var JSONObject = {
42              "userid": userid,
43              "password": password
44          };
45
46          $.ajax({
47              url: url,
48              type: 'GET',
49              data: JSONObject,
50              dataType: 'json',
51              contentType: "application/json; charset=utf-8",
52              success: function (arr) {
53                  _getLoginResult(arr);
54              },
55              error: function () {
56                  validationMsg();
57              }
58          });
59      }
60
61      function _getLoginResult(arr) {
62          if (arr[0].result.trim() !== "0") {
63              localStorage.setItem("userid", userid);
64              localStorage.setItem("password", password);
65              validationMsgs("Login OK", "Information", "OK");
66          }
67          else {
68              validationMsgs("Error in Username or Password",
    "Validation", "Try Again");
69          }
70      }
71  } )();
```

a. Lines 4 and 5 define 2 JavaScript variables, userid and password.

```
var userid;
var password;
```

b. Lines 9 to 21 state the validation rules for the LoginForm in index.html. The error messages for txtLogin and txtPassword are defined.

The errorPlacement settings define that error messages will show up just below the corresponding text fields.

```
$("#LoginForm").validate({
    messages: {
        txtLogin: "User ID is required",
        txtPassword: "Password is required",
    },
    focusInvalid: false,
    submitHandler: function () {
        return false;
    },
    errorPlacement: function (error, element) {
        error.appendTo(element.parent().parent().after());
    },
});
```

c. Lines 23 to 27 create an event handler that executes the function login() when the user press the [Login] button. It first checks if the form is valid. A form is valid if the user's entry adheres to the validation rules described in lines 9 to 21.

```
$("#btnLogin").bind("click", function () {
            if ($("#LoginForm").valid()) {
                login();
            }
});
```

d. Lines 35 to 59 define the login() function.

```
function login() {
	var url = serverURL() + "/login.php";
	var result;
	userid = $("#txtLogin").val();
	password = $("#txtPassword").val();

	var JSONObject = {
		"userid": userid,
		"password": password
	};

	$.ajax({
		url: url,
		type: 'GET',
		data: JSONObject,
		dataType: 'json',
		contentType: "application/json; charset=utf-8",
		success: function (arr) {
			_getLoginResult(arr);
		},
		error: function () {
			validationMsg();
		}
	});
}
```

e. Line 36 defines the URL that a server call will be made to, to validate if the user has entered a correct user id and password. It concatenates the result of the function serverURL() with the text "login.php". We have written the function serverURL() to return the value http://<your url>/friendszone in common.js earlier. Thus this operation concatenates http://<your url>/friendszone/ with "login.php" and copies it to the JavaScript variable "url".

f. Line 37 defines a variable "result".

g. Lines 38 and 39 take the values that the user enters in the txtLogin and txtPassword textfields and copies them into the userid and password variable.

h. Lines 41 to 44 forms the parameters that will be used to make the AJAX call to the login.php web service as a JSONObject variable.

i. Lines 46 to 58 will make a call to the friendszone webserver at the URL stated in the variable "url", using the parameters stated in the variable "JSONObject".

   i. Line 47 sets the property of the url to the variable "url".

   ii. Line 48 sets the call to be GET.

   iii. Line 49 sets the property of the data to make the call to the variable "JSONObject".

iv. Lines 50 and 51 sets the datatype of the data to be passed to the server to JSON.

v. Lines 52 to 54 states that the function _getLoginResult() should be executed after a successful call to the friendszone server.

vi. Lines 55 to 57 states that the function validationMsg() will be executed should the call fails. There are a handful of reasons why a server call might fail.

i. The mobile phone has no internet connection.

ii. The friendszone server is down.

vii. This is just like going to the web browser to type the following URL with the parameters of userid=jimmy&password=jimmy.

```
http://<your url>/friendszone/login.php?userid=jimmy&password=jimmy
```

viii. This method of making a web server call is known as AJAX. You may refer to the documentation here: http://api.jquery.com/jquery.ajax/.

j. Lines 61 to 69 states what the program will do when the web server sends its response to the request. The response will state if the result of calling login.php is a 1 or 0.

```
1  function _getLoginResult(arr) {
2          if (arr[0].result.trim() !== "0") {
3              localStorage.setItem("userid", userid);
4              localStorage.setItem("password", password);
5              validationMsgs("Login OK", "Information", "OK");
6          }
7          else {
8              validationMsgs("Error in Username or Password",
   "Validation", "Try Again");
9          }
10     }
```

- Line 1 of the getLoginResult() function above parses the web server's response and turns the JSON string returned by the web server into an array.
- Line 2 checks the value of "result" in the JSON string. If the result is 1, it means that the login is valid.
- Lines 3 and 4 sets 2 localstorage variables userid and password to the values of the userid and password variables. Localstorage variables are accessible across all the HTML pages in the FriendsZone app.
- Line 5 pops a "Login OK" message.
- Line 8 pops a message "Error in User Name or Password" if the result returned is not 1.

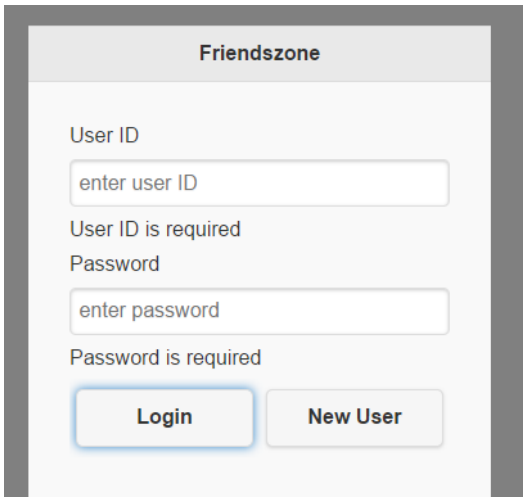
k. Run the app. Try the following:

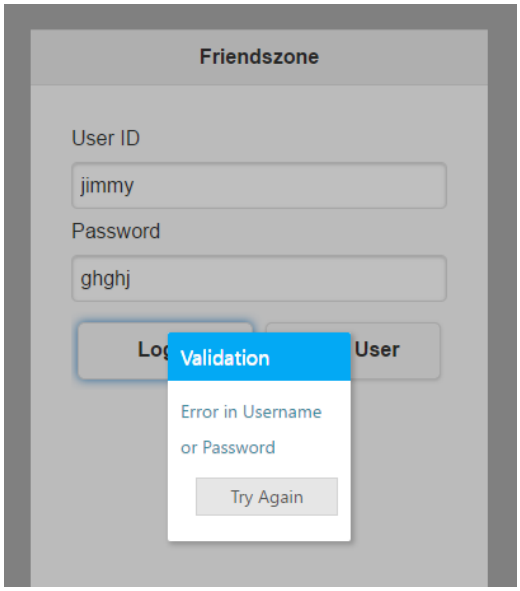
   i. Leave User ID and Password blank.

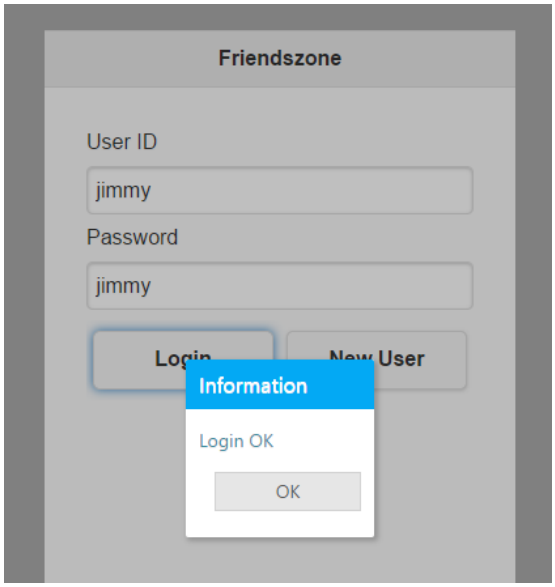
   ii. Enter a correct set of user ID and password

   iii. Enter a wrong set of user ID and password.

Watch how the app reacts differently.







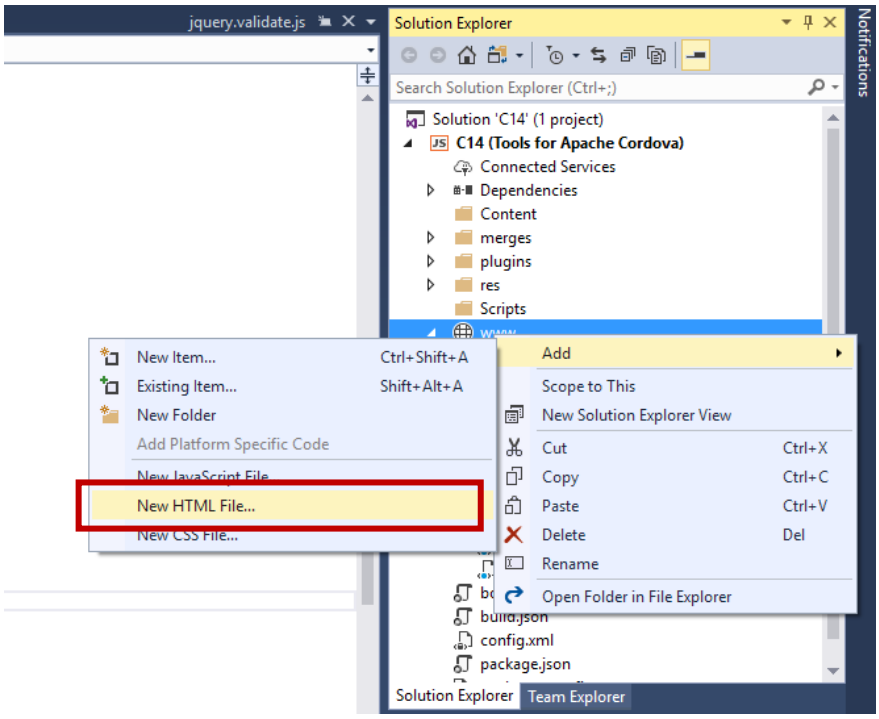
Instruction 3: Design a user interface to create a new user

1. We will design the user interface to create a new user. To create a new user, the user clicks the [New User] button on index.html.

2. The filename for the new user creation form is newuser.html. Before we create this new form, we will modify index.js to link the [New User] button to newuser.html. Make the following changes to index.js within the $(document).ready(function () section.

```
$("#btnNewUser").bind("click", function () {
    window.location = "newuser.html";
});
```

3. Right click on the www folder. Select Add > New HTML File.



4. Add a new HTML file "newuser.html".

5. Right click on the /www/scripts/ folder. Select Add > new JavaScript File.

6. Add a new JavaScript file "newuser.js".

7. Enter the following codes to draw the user interface for newuser.html.

```
<!DOCTYPE html>
<html>
<head>
    <!--
        Customize the content security policy in the meta tag below as needed. Add 'unsafe-inline' to default-src to enable inline JavaScript.
        For details, see http://go.microsoft.com/fwlink/?LinkID=617521
    -->
    <meta http-equiv="Content-Security-Policy" content="default-src 'self' data: gap: http://jacksonng.org https://ssl.gstatic.com 'unsafe-eval'; style-src 'self' 'unsafe-inline'; media-src *">

    <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
    <meta name="format-detection" content="telephone=no">
    <meta name="msapplication-tap-highlight" content="no">
```

```
13      <meta name="viewport" content="user-scalable=no, initial-scale=1,
    maximum-scale=1, minimum-scale=1, width=device-width">
14
15      <link rel="stylesheet" type="text/css" href="css/index.css">
16      <link rel="stylesheet" href="css/jquery.mobile-1.4.5.css">
17
18      <script src="lib/jquery-1.11.2.min.js"></script>
19      <script src="lib/jquery.mobile-1.4.5.min.js"></script>
20      <script src="lib/jquery.validate.min.js"></script>
21      <script src="scripts/common.js"></script>
22      <script src="scripts/newuser.js"></script>
23
24      <title>Friendszone</title>
25  </head>
26  <body>
27      <div data-role="page" id="front-page">
28          <div data-role="header" class="header" data-position="fixed">
29              <h1 id="app-title">Friendszone</h1>
30          </div>
31
32          <div role="main" class="ui-content" id="main-ui">
33              <form name="NewUserForm" id="NewUserForm">
34                  <div data-role="fieldcontainer">
35                      <label for="txtNewName">User ID</label>
36                      <input type="text" name="txtNewName" id="txtNewName"
    placeholder="enter user ID" value="peter" required>
37                  </div>
38
39                  <div data-role="fieldcontainer">
40                      <label for="txtNewEmail">Email</label>
41                      <input type="text" name="txtNewEmail"
    id="txtNewEmail" placeholder="enter email" value="peter@peter.com"
    required>
42                  </div>
43
44                  <div data-role="fieldcontainer">
45                      <label for="txtNewPassword">Password</label>
46                      <input type="text" name="txtNewPassword"
    id="txtNewPassword" placeholder="enter password" value="peter" required>
47                  </div>
48
49                  <div data-role="fieldcontainer">
50                      <label for="txtNewPasswordAgain">Re-Enter
    Password</label>
51                      <input type="text" name="txtNewPasswordAgain"
    id="txtNewPasswordAgain" placeholder="enter password again"
    value="peter" required>
52                  </div>
53
54                  <div data-role="fieldcontainer">
55                      <img id="imgNewUser" height="100">
56                      <input type="button" name="selectimg"
    id="btnSelectImg" value="My Picture">
57                  </div>
58
59                  <div data-role="fieldcontainer">
```

```
60                      <label for="description">About Me:</label>
61                      <textarea rows="8" name="description"
  id="description"></textarea>
62                  </div>
63
64                  <input type="button" id="btnCreateAccount" value="Create
  Account">
65              </form>
66          </div>
67
68          <footer data-role="footer" data-position="fixed">
69              <h4>Copyright 2017 OTBS</h4>
70          </footer>
71      </div>
72      <script type="text/javascript" src="cordova.js"></script>
73      <script type="text/javascript"
  src="scripts/platformOverrides.js"></script>
74 </body>
75 </html>
```

a. Line 33 defines a new form "NewUserForm".

b. Lines 34 to 37 define the txtNewName text field. Note that for convenience, we have set the value of this text field to "peter".

c. Lines 44 and 47 define the txtPassword text field. Note that for convenience, we have set the value of this text field to "peter". We have also used the 'input type="text"' field type instead of 'input type="password"' so that it is easier to see the value in this text field.

d. Lines 49 to 52 define a text field for the user to re-enter his password for confirmation.

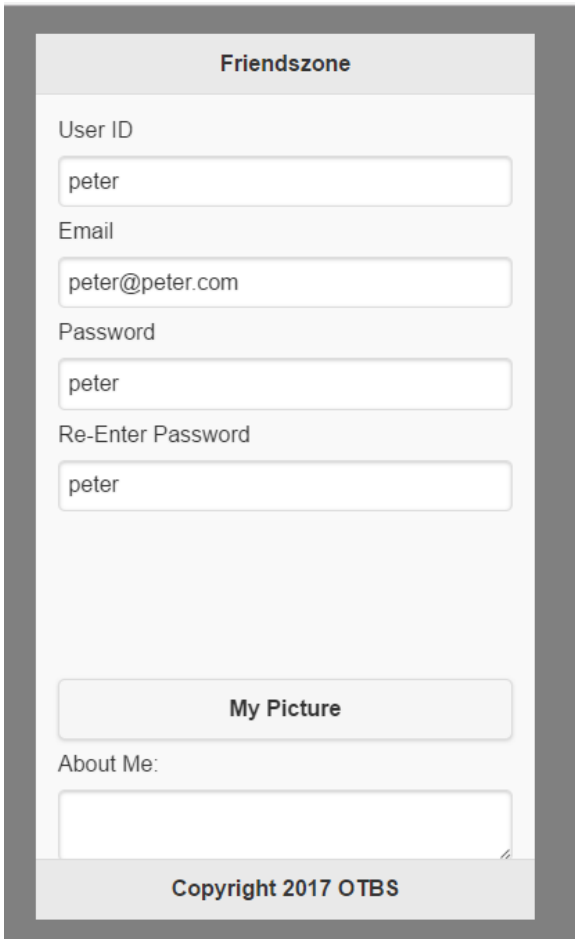
e. Lines 54 to 57 is a placeholder for an image that we are expecting the user to select and upload. It contains nothing now.

f. Line 56 is a button [My Picture]. This allows the user to choose an image from his phone's gallery to upload.

g. Lines 59 to 62 define a text field for the user to enter a description of himself.

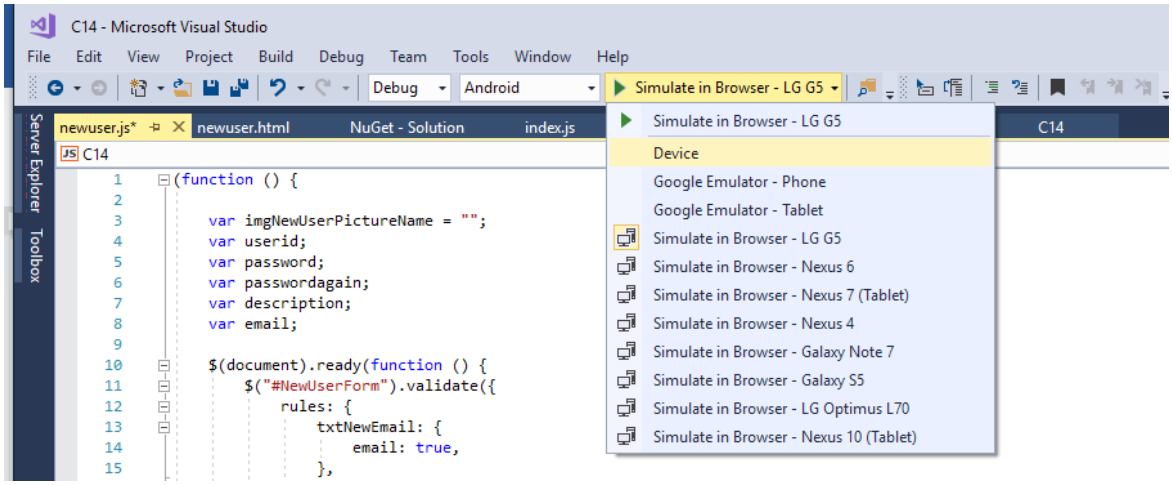
h. Line 64 defines a button [Create Account].

8. Run the program. Click the [New User] button to see the user interface for new user creation.

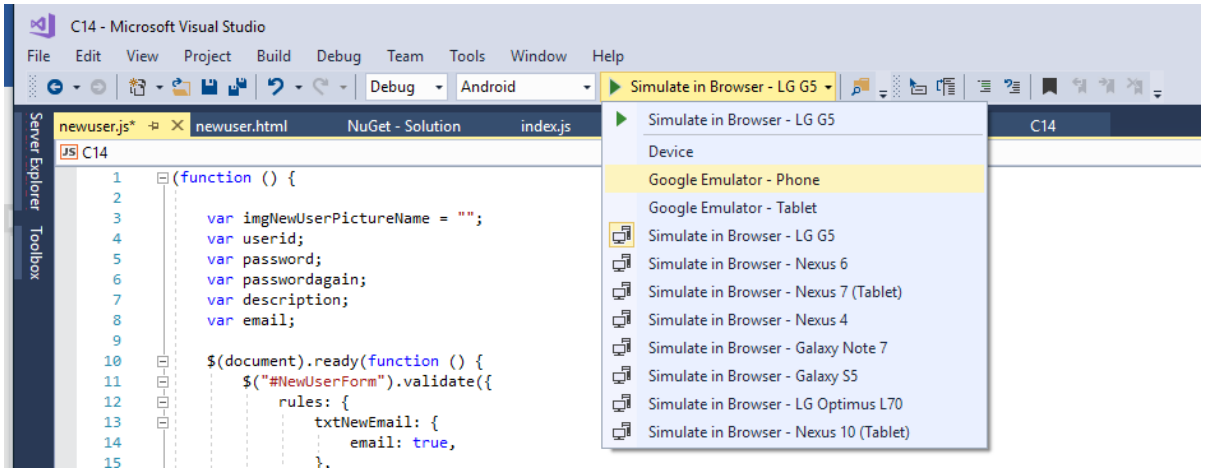


Instruction 5: Upload an image to the server

1. This section must be tested on a physical Android mobile device or an Android emulator. To run this section in a physical Android mobile device, plug in the device and select "Device" when you test the app.



2. To run this section in an Android emulator, select "Google Emulator – Phone".



3. The FriendsZone server application provides 2 sets of PHP function for image uploads namely, upload.php and deleteimg.php.

    a. upload.php is used to upload a new profile image into an image folder in the FriendsZone server. The following URL will be used to upload an image:

```
http://<your url>/friendszone/upload.php
```

    Notice that there are no URL parameters for calling upload.php. This is because this is a file transfer web service.

    b. deleteimg.php is used to delete an existing user image. This is used to delete an old image after a new image has been uploaded. The following URL will be called to delete an image. The URL parameter "imgfile" tells deleteimg.php the filename of the image to delete. B3D4 is the file name of the image to delete.

```
http://<your url>/friendszone/deleteimg.php?imgfile=B3D4
```

4. Edit newuser.js:

```
(function () {

    var imgNewUserPictureName = "";

    $(document).ready(function () {
        $("#btnSelectImg").bind("click", function () {
            capturePhoto();
        });

    });

    //Image Section
    function capturePhoto() {
```

```
14          var source = navigator.camera.PictureSourceType.PHOTOLIBRARY;
15          navigator.camera.getPicture(_onPhotoURISuccess, _failCapture, {
    quality: 50, destinationType: navigator.camera.DestinationType.FILE_URI,
    sourceType: source });
16      }
17
18      function _failUpload(error) {
19          validationMsgs("Error:" + error.code, "Upload Error", "Try
    Again");
20      }
21
22      function _failCapture(message) {
23
24          validationMsgs("Error:" + message, "Image Error", "Try Again");
25      }
26
27      function _onPhotoURISuccess(imageURI) {
28
29          var options = new FileUploadOptions();
30          options.fileKey = "file";
31          options.fileName = imageURI.substr(imageURI.lastIndexOf('/') +
    1);
32          options.mimeType = "image/jpeg";
33
34          var params = new Object();
35          params.value1 = "test";
36          params.value2 = "param";
37          options.params = params;
38          options.chunkedMode = false;
39          options.headers = { Connection: "close" };
40          var ft = new FileTransfer();
41          ft.upload(imageURI, serverURL() + "/upload.php", _winUpload,
    _failUpload, options);
42      }
43
44      function _winUpload(r) {
45          if (imgNewUserPictureName !== "") {
46              _deleteOldImg(imgNewUserPictureName);
47          }
48          var arr = JSON.parse(r.response);
49          imgNewUserPictureName = arr[0].result;
50          $("#imgNewUser").attr("src", serverURL() + "/images/" +
    imgNewUserPictureName + "_s");
51
52      }
53
54      function _deleteOldImg(oldImg) {
55          var url = serverURL() + "/deleteimg.php";
56
57          var JSONObject = {
58              "imgfile": oldImg
59          };
60
61          $.ajax({
62              url: url,
63              type: 'GET',
```

```
64              data: JSONObject,
65              dataType: 'json',
66              contentType: "application/json; charset=utf-8",
67              success: function (arr) {
68                  _deleteImgResult(arr);
69              },
70              error: function () {
71                  validationMsg();
72              }
73          });
74      }
75
76      function _deleteImgResult(arr) {
77          if (arr[0].result !== "1") {
78              validationMsgs("Error deleteing old image", "Upload Error",
   "Try Again");
79          }
80      }
81  })();
```

a. Lines 6 to 8 defines the event handler that executes the function capturePhoto() when the button [My Picture] is pressed.

b. Lines 13 to 16 defines the function getImage().

c. Lines 14 to 15 opens the mobile device's photo gallery (note that this function will not work unless you have installed the Cordova camera plugin (Instruction 1):

```
1 var source = navigator.camera.PictureSourceType.PHOTOLIBRARY;
2 navigator.camera.getPicture(_onPhotoURISuccess, _failCapture, { quality:
  50, destinationType: navigator.camera.DestinationType.FILE_URI,
  sourceType: source });
```

- Line 1 above defines that the user will select a picture from the mobile device's photo library.

- In line 2 above, the Cordova function navigator.camera.getPicture opens the device's default camera application for the user to select a photo. The syntax for this function is as follows:

```
navigator.camera.getPicture( cameraSuccess, cameraError, [ cameraOptions ] );
```

  - cameraSuccess: This function executes if a photo from the gallery is successfully selected. In our codes, the function uploadPhoto will execute on successful selection.

  - cameraError: This function executes if an error occurs when the user tries to select a photo from the gallery. In our codes, a function is defined to pop an alert that says "get picture failed".

  - cameraOptions: 3 options are set in our codes.

- quality:50 – Quality of saved image is expressed in a range of 0-100 where 100 is full resolution with no loss of quality. We set a quality of 10 for our uploaded image.

- destinationType:navigator.camera.DestinationType.FILE_URI - This returns the image file's URI (i.e. the image file's name) to the caller.

- sourceType:source, which was defined in line 1 to be navigator.camera.PictureSourceType.PHOTOLIBRARY – This opens the mobile device's photo library.

Refer to https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-camera/ for more options for the navigator.camera.getPicture function.

```
1  function _onPhotoURISuccess(imageURI) {
2         var options = new FileUploadOptions();
3         options.fileKey = "file";
4         options.fileName = imageURI.substr(imageURI.lastIndexOf('/') +
   1);
5         options.mimeType = "image/jpeg";
6
7         var params = new Object();
8         params.value1 = "test";
9         params.value2 = "param";
10        options.params = params;
11        options.chunkedMode = false;
12        options.headers = { Connection: "close" };
13        var ft = new FileTransfer();
14        ft.upload(imageURI, serverURL() + "/upload.php", _winUpload,
15 _failUpload, options);
16 }
```

a. Lines 1 to 16 define the _onPhotoURISuccess () function that is executed when a photo has been successfully selected from the photo gallery.

b. Line 2 defines a new variable to store file upload options.

c. Line 3 defines that the file to be uploaded is of the "file" type.

d. Line 4 defines the name of the file that has been selected by the user from the photo gallery.

e. Line 5 sets the file type to be a JPEG file.

f. Lines 7 to 12 define the options and parameters for the image file to be uploaded.

g. Line 13 defines a new FileTransfer variable (note that this function will work if you installed the Cordova File and FileTransfer plugin in Instruction1).

h. Line 14 performs the actual file upload activity, stating the following:

   - The file will be received by the upload.php web service.

- The function _winUpload() will be executed on successful upload.
- The function _failUpload() will be executed if upload fails.
- The upload will be executed using the options variable defined between lines 2 to 12.

i. Refer to https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-file-transfer/index.html for more details on the Cordova API for File Transfer.

```
1 function _winUpload(r) {
2         if (imgNewUserPictureName !== "") {
3                 _deleteOldImg(imgNewUserPictureName);
4         }
5         var arr = JSON.parse(r.response);
6         imgNewUserPictureName = arr[0].result;
7         $("#imgNewUser").attr("src", serverURL() + "/images/" +
  imgNewUserPictureName + "_s");
8
9 }
```

a. Lines 1 to 9 defines the function _winUpload() which will execute when the file upload is successful.

b. Lines 2 to 4 check if the variable imgNewUserPictureName contains any value. If it does, the user has uploaded another profile image while he was on this page. This image needs to be deleted since the user is replacing an existing image with a new one. The function _deleteOldImg is executed with the image's filename in the imgNewUserPictureName variable.

c. Line 5 reads the response from upload.php in the web server and copies the response to the variable arr. upload.php returns the name of the file that it has given to image that the user uploaded. The name of this file is copied to the variable imgNewUserPictureName.

d. Line 7 sets the image filename to become the src field in the <img></img> tag imgNewUser defined in index.html for example, '<img id="myimg" height="100" src="B4C3_s">'. The server stores to version of the picture, B4C3_s, which is the thumbnail and B4C3, which is the original high resolution image. This displays the low-resolution version of the image.

```
1 function _failUpload(error) {
2         validationMsgs("Error:" + error.code, "Upload Error", "Try
  Again");
3 }
```

a. Lines 1 to 3 define the function _failUpload() which will execute when the file upload fails.

b. Line 2 pops a notification with an error message and a code that explains why the upload has failed.

```
1  function _deleteOldImg(oldImg) {
2          var url = serverURL() + "/deleteimg.php";
3
4          var JSONObject = {
5              "imgfile": oldImg
6          };
7
8          $.ajax({
9              url: url,
10             type: 'GET',
11             data: JSONObject,
12             dataType: 'json',
13             contentType: "application/json; charset=utf-8",
14             success: function (arr) {
15                 _deleteImgResult(arr);
16             },
17             error: function () {
18                 validationMsg();
19             }
20         });
21     }
```

a. The function _deleteOldImg() will delete an existing profile image in the web server. Line 2 forms the URL to call.

b. Lines 4 to 6 forms the JSONObject that contains the filename of the image to delete based on the image filename 'oldImg' provided. If this was a URL parameter, it will look like the following:

```
http://<your url>/friendszone/deleteimg.php?imgfile=B3D4
```

c. Lines 8 to 20 execute an AJAX call to deleteimg.php, based on the data in the JSONObject.

d. Line 15 states that if the call to deleteimg.php on the friendszone server is successful, the function _deleteImgResult() will execute.

e. Lines 17 to 19 executes the function validationMsg() if the AJAX call fails.

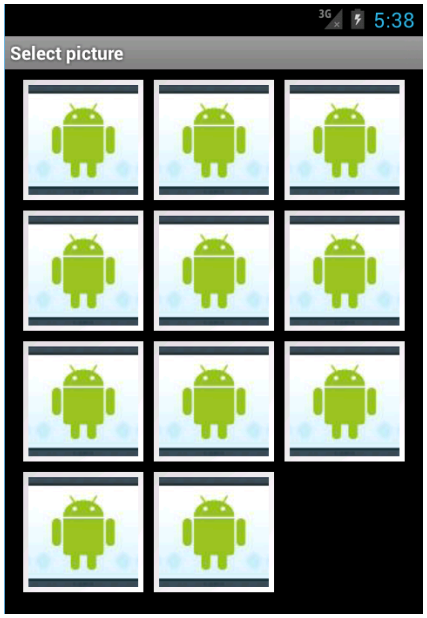
```
1 function _deleteImgResult(arr) {
2         if (arr[0].result !== "1") {
3                 validationMsgs("Error deleteing old image", "Upload
  Error", "Try Again");
4         }
5 }
```

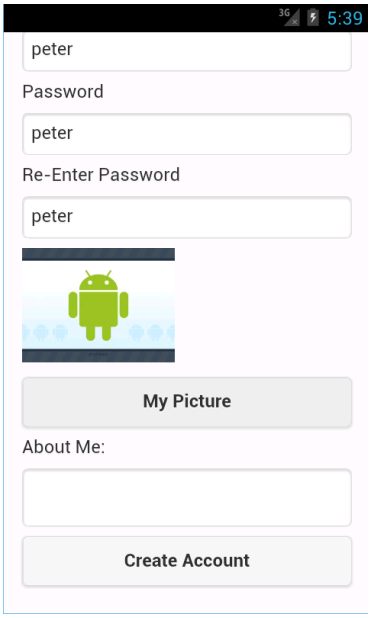
a. Lines 2 to 4 check if deleteimg.php returns the value "1". If it does not, then the old image has not been deleted successfully. A notification pops to say "Error deleting old image.".

5. Run the program. To see how the uploadPhoto() function works, press [New User]. Then press [My Picture] to select a picture from your mobile device's photo gallery.

6. Visit the /friendszone/img folder. Note that a new JPEG file have been uploaded to the server. A "_s" version of the JPEG file has also been created by the friendszone server. The filename that was given to the image is a random text.

7. To see how the _deleteOldImg() function works, press [My Picture] again. Select a different photo from the photo gallery. Note that the previous JPEG file has been deleted and replaced with a new JPEG file and a new random filename.





Instruction 6: Save a new user

1. The FriendsZone server application provides a PHP web service newuser.php to allow new FriendsZone profiles to be created. The following URL will be used to upload an image:

```
http://<your
url>/friendszone/newuser.php?userid=jimmy&password=jimmy&description=hello&pr
ofileimage=B4C5.jpg
```

2. The parameters to call newuser.php are:

    a. userid

    b. Password

    c. Description

    d. profileimage

3. Edit newuser.js:

```
(function () {

    var imgNewUserPictureName = "";
    var userid;
    var password;
    var passwordagain;
    var description;
    var email;

    $(document).ready(function () {
        $("#NewUserForm").validate({
            rules: {
                txtNewEmail: {
                    email: true,
                },
                txtNewPasswordAgain: {
                    equalTo: "#txtNewPassword"
                }
            },
            messages: {
                txtNewName: "new user name is required",
                txtNewEmail: "new email address is required and must be of the format a@b.c",
                txtNewPassword: "new password is required",
                txtNewPasswordAgain: "new password again is required and must be the same as new password",
            },
            focusInvalid: false,
            submitHandler: function () {
                return false;
            },
            errorPlacement: function (error, element) {
                error.appendTo(element.parent().parent().after());
            },
        });

        $("#btnSelectImg").bind("click", function () {
            capturePhoto();
        });

        $("#btnCreateAccount").bind("click", function () {
            savenewuser();
        });
    });


    //Image Section
    function capturePhoto() {
        var source = navigator.camera.PictureSourceType.PHOTOLIBRARY;
        navigator.camera.getPicture(_onPhotoURISuccess, _failCapture, { quality: 50, destinationType: navigator.camera.DestinationType.FILE_URI, sourceType: source });
    }

    function _failUpload(error) {
        validationMsgs("Error:" + error.code, "Upload Error", "Try
```

```
Again");
    }

    function _failCapture(message) {
        validationMsgs("Error:" + message, "Image Error", "Try Again");
    }

    function _onPhotoURISuccess(imageURI) {
        var options = new FileUploadOptions();
        options.fileKey = "file";
        options.fileName = imageURI.substr(imageURI.lastIndexOf('/') +
1);
        options.mimeType = "image/jpeg";

        var params = new Object();
        params.value1 = "test";
        params.value2 = "param";
        options.params = params;
        options.chunkedMode = false;
        options.headers = { Connection: "close" };
        var ft = new FileTransfer();
        ft.upload(imageURI, serverURL() + "/upload.php", _winUpload,
_failUpload, options);
    }

    function _winUpload(r) {
        if (imgNewUserPictureName !== "") {
            _deleteOldImg(imgNewUserPictureName);
        }
        var arr = JSON.parse(r.response);
        imgNewUserPictureName = arr[0].result;
        $("#imgNewUser").attr("src", serverURL() + "/images/" +
imgNewUserPictureName + "_s");
    }

    function _deleteOldImg(oldImg) {
        var url = serverURL() + "/deleteimg.php";

        var JSONObject = {
            "imgfile": oldImg
        };

        $.ajax({
            url: url,
            type: 'GET',
            data: JSONObject,
            dataType: 'json',
            contentType: "application/json; charset=utf-8",
            success: function (arr) {
                _deleteImgResult(arr);
            },
            error: function () {
                validationMsg();
            }
        });
    }
```

```
105
106     function _deleteImgResult(arr) {
107         if (arr[0].result !== "1") {
108             validationMsgs("Error deleteing old image", "Upload Error",
    "Try Again");
109         }
110     }
111
112
113     //New User Saving Section
114     function savenewuser() {
115         if ($("#NewUserForm").valid()) {
116             var profileimage = imgNewUserPictureName;
117
118             userid = $("#txtNewName").val();
119             email = $("#txtNewEmail").val();
120             password = $("#txtNewPassword").val();
121             passwordagain = $("#txtNewPasswordAgain").val();
122             description = $("#txtNewDescription").val();
123
124             if ( validate()) {
125                 var url = serverURL() + "/newuser.php";
126
127                 var JSONObject = {
128                     "userid": userid,
129                     "password": password,
130                     "email": email,
131                     "description": description,
132                     "profileimage": profileimage
133                 };
134
135                 $.ajax({
136                     url: url,
137                     type: 'GET',
138                     data: JSONObject,
139                     dataType: 'json',
140                     contentType: "application/json; charset=utf-8",
141                     success: function (arr) {
142                         _getNewUserResult(arr);
143                     },
144                     error: function () {
145                         validationMsg();
146                     }
147                 });
148             }
149         }
150     }
151
152     function  validate() {
153         var validate = true;
154
155         if (imgNewUserPictureName === "") {
156             validationMsgs("Select a photo", "Validation", "OK");
157             validate = false;
158         }
159         return validate;
```

```
160     }
161
162     function  getNewUserResult(arr) {
163         if (arr[0].result === 1) {
164             localStorage.setItem("userid", userid);
165             localStorage.setItem("password", password);
166             validationMsgs("New User created", "Validation", "OK");
167
168         }
169         else {
170             validationMsgs("User ID already exist", "Validation",
171 "OK");
172         }
173     }
174 })();
```

a. Lines 4 to 8 declare new variables to store userid, password, passwordagain, description and email.

b. Lines 11 to 33 validates the NewUserForm:

```
 1 $("#NewUserForm").validate({
 2         rules: {
 3                 txtNewEmail: {
 4                         email: true,
 5                 },
 6                 txtNewPasswordAgain: {
 7                         equalTo: "#txtNewPassword"
 8                 }
 9         },
10         messages: {
11                 txtNewName: "new user name is required",
                   txtNewEmail: "new email address is required and must be
12 of the format a@b.c",
13                 txtNewPassword: "new password is required",
14                 txtNewPasswordAgain: "new password again is required and
  must be the same as new password",
15         },
16         focusInvalid: false,
17         submitHandler: function () {
18                 return false;
19         },
20         errorPlacement: function (error, element) {
21                 error.appendTo(element.parent().parent().after());
22         },
23 });
```

- Lines 2 to 9 of the codes above sets the rules of validation. The text field txtEmail must be an email address. The text field txtNewPasswordAgain must be the same value as txtNewPassword.

- Lines 11 to 15 set up the messages that will be displayed to the user if any of the text fields are not validated correctly.

c. Lines 39 to 41 defines an event handler to handle the event where the user presses the [Create Account] button.

d. Lines 114 to 150 declare the savenewuser() function:

```
1 function savenewuser() {
2         if ($("#NewUserForm").valid()) {
3                 var profileimage = imgNewUserPictureName;
4
5                 userid = $("#txtNewName").val();
6                 email = $("#txtNewEmail").val();
7                 password = $("#txtNewPassword").val();
8                 passwordagain = $("#txtNewPasswordAgain").val();
9                 description = $("#txtNewDescription").val();
10
11                if (_validate()) {
12                        var url = serverURL() + "/newuser.php";
13
14                        var JSONObject = {
15                                "userid": userid,
16                                "password": password,
17                                "email": email,
18                                "description": description,
19                                "profileimage": profileimage
20                        };
21
22                        $.ajax({
23                                url: url,
24                                type: 'GET',
25                                data: JSONObject,
26                                dataType: 'json',
27                                contentType: "application/json;
  charset=utf-8",
28                                success: function (arr) {
29                                        _getNewUserResult(arr);
30                                },
31                                error: function () {
32                                        validationMsg();
33                                }
34                        });
35                }
36        }
37 }
```

- In the savenewuser() function above, line 2 checks if the form is successfully validated. The function will not proceed unless there are no errors in the form.
- Line 3 copies the value of the filename is imgNewUserPictureName to the variable profileimage.
- Lines 5 to 9 retrieve the values stored in the HTML text fields of txtNewName, txtNewEmail, txtNewPassword, txtNewPasswordAgain and txtNewDescription and copy them into each corresponding JavaScript variable.

- Line 11 performs another validation by executing the _validate() function. The _validate() function does the following:

```
1 function _validate() {
2         var validate = true;
3
4         if (imgNewUserPictureName === "") {
5                 validationMsgs("Select a photo", "Validation", "OK");
6                 validate = false;
7         }
8         return validate;
9 }
```

It checks if the user has uploaded an image before he presses [Create Account]. If imgNewUserPictureName contains no value, the user has not uploaded a picture. A notification pops up to remind him to upload a picture before he creates a new account.

If the function _validate() returns a "true" value, then the user is allowed to continue. Or else, he needs to rectify his error before he can continue.

- Line 12 forms the URL string with the values of userid, password, email, description and profileimage.
- Lines 14 to 20 creates a JSONObject with the values in each variable to be passed to newuser.php to create a new user account.
- Lines 22 to 34 executes the AJAX call to newuser.php on the friendszone server.
- Lines 28 to 30 execute the function _getNewUserResult() if the AJAX call to newuser.php is successful.

e. Lines 162 to 173 declare the _getNewUserResult function. This function is triggered to execute upon successful execution of the savenewuser() function that makes an AJAX call to newuser.php in the FriendsZone web server.

```
 1 function _getNewUserResult(arr) {
 2         if (arr[0].result === 1) {
 3                 localStorage.setItem("userid", userid);
 4                 localStorage.setItem("password", password);
 5                 validationMsgs("New User created", "Validation", "OK");
 6
 7         }
 8         else {
 9                 validationMsgs("User ID already exist", "Validation",
   "OK");
10         }
11 }
```

- In the _getNewUserResult() function above, line 4 checks if arr contains a value of 1. If arr contains a value of 1, newuser.php has responded that a new user has been added to the friendszone server successfully. Line 5 pops a notification that

says "New User created". Lines 3 and 4 store the userid and password values to local storage.

- If arr does not contain the value 1, newuser.php has indicated that the insertion of a new user record has failed. There is only 1 reason for failure – the userid has already been taken. In this case, line 9 pops an alert that tells the user that "User ID already exist".

Exercise

There are no exercises for this chapter.

Codes

Codes for this chapter can be found by searching VC14-User-Management-Authentication on Github.com.