

# Learn2Code – Programming an Instant Messenger

In order to program an instant messaging system, we're going to be using three different types of programming languages: mark-up language, client-side scripting and server-side scripting

Mark-up language is the simplest to understand. This is the code which tells the browser to put a box or an image on the screen and how it should look. The mark-up languages we're going to be using are **HTML** and **CSS**.

Client-side scripting does all the complicated logic on the user's computer, such as displaying message, formatting data etc... for this we're going to use **JavaScript**.

Server-side scripting is the most important part, as all the messages need to be stored on a server. To interact with the server and get information from a database, we're going to be using **PHP** and **MySQL**.

## Setting up the server

Most servers require you to pay a monthly fee for database and hosting space, but we're going to use a free limited server. The website is as follows: <https://byethost.com/>

Navigate to *Free Hosting* and register an account.

Once you have registered, you should get an email with all your login details, please **make a note of these details** as you don't want to lose them.

To log in to your server, visit this website: <http://panel.byethost.com/>  
Enter your cPanel information and you should be able to log in to the control panel.

## Making sure PHP works

Once you're in the control panel, scroll down to the *Files* section, and click on *Online File Manager*. In this view, you can see all the files you are going to be working with. By default, there are a few configuration files, but you don't need to worry about them.

All the website files are inside the *htdocs* folder. Click on this folder name and delete any files in this folder. Make a new file in this folder and call it *index.php*.

Let's see if PHP works correctly on our server by printing today's date and time:

```
1  <?php
2
3  // Tell the server which time zone we're in
4  date_default_timezone_set("Europe/London");
5
6  // Get the current date
7  $date = date("jS M Y");
8
9  // Get the current time
10 $time = date("H:i");
11
12 // Print a string
13 echo "Today is <b>" . $date . "</b> and the time is <b>" . $time . "</b>";
14
15 ?>
```







# Creating the login screen

Before we worry about the logic of the Instant Messenger, we're going to create a basic login screen using the two mark-up languages **HTML** and **CSS**.

Make sure you download all the images from:  
[http://tinyurl.com/learn2code\\_messenger](http://tinyurl.com/learn2code_messenger)

Your folder structure should now look like this:

Open *index.php*, remove the test PHP code and type the following code:

<input type="checkbox"/>		<a href="#">avatar-standard.png</a>	PNG file
<input type="checkbox"/>		<a href="#">close.png</a>	PNG file
<input type="checkbox"/>		<a href="#">icon.png</a>	PNG file
<input type="checkbox"/>		<a href="#">index.php</a>	PHP script
<input type="checkbox"/>		<a href="#">jquery.js</a>	JavaScript file
<input type="checkbox"/>		<a href="#">loading.gif</a>	GIF file

```

1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <link href='http://fonts.googleapis.com/css?family=Montserrat:700,400' rel='stylesheet' />
6      <title>Learn2Code Messenger</title>
7  </head>
8
9  <body>
10
11     <div class="heading">
12         <b>Messenger</b>
13         <br />
14         by Your Name
15     </div>
16
17     <div class="container">
18
19         <div class="login">
20             <h1 class="title">Create an account</h1>
21             <p>
22                 Type a screen name.
23                 <br />You will be remembered on this computer.<br /><br />
24                 <input placeholder="Enter a screen name" class="screen_name" />
25                 <a class="enter">&#10152;</a>
26                 
27             </p>
28         </div>
29
30         <div class="people">
31             <ul>
32             </ul>
33         </div>
34
35     </div>
36
37 </body>
38 </html>

```

Your web application can be viewed at all times at the following URL:  
<http://www.<username>.byethost33.com>  
 Try and preview your page now.

# Designing the login screen

By viewing your login screen, everything should be working but it doesn't look very presentable. We need to hide the loading animation as well as position everything on the screen.

To do all this, we're going to use another mark-up language called **CSS** which stands for **Cascading Style Sheets**.

Create a new file called *style.css* and add this code:

```

1  html, head, body, input, table, span, div, ul, li { margin: 0; padding: 0; }
2  body { font: 14px Arial; background: #2B94D9; color: #fff; }
3
4  .heading { margin: 30px; text-align: center; }
5  .heading b { font-family: 'Montserrat'; font-size: 55px; }
6
7  .container { width: 900px; min-height: 450px; background: #EBEBEB; color: #000; left: 50%;
8              margin-top: 10px; margin-bottom: 40px; position: absolute; margin-left: -480px; }
9
10 .container .people { background: #2B3848; color: #FFF; width: 58px;
11                      height: 100%; position: absolute; }
12 .container .people ul { list-style: none; }
13
14 .container .login { display: block; position: absolute; width: 855px; margin-left: 62px;
15                    text-align: center; font-size: 16px; }
16
17 .container .login .title { font-family: 'Montserrat', sans-serif; font-size: 36px;
18                            margin: 95px 0 35px; font-weight: 400; }
19
20 .container .login p { color: #888; }
21 .container .login .screen_name { padding: 15px 22px; font-family: Arial; font-size: 16px;
22                                  width: 300px; border-radius: 7px; border: 1px solid #aaa;
23                                  -webkit-transition: all 0.5s; margin-top: 15px;
24                                  -moz-transition: all 0.5s; -o-transition: all 0.5s; }
25
26 .container .login .screen_name:focus { border: 1px solid #2B94D9; outline: none; }
27 .container .login a.enter { display: none; position: absolute; font-size: 35px;
28                             margin: 14px 0 0 -45px; }
29
30 .container .login a.enter:hover { cursor: pointer; color: #2B94D9; }
31 .container .login .loading { display: none; position: absolute; height: 30px;
32                             margin: 24px 0 0 -46px; }

```

The CSS is simply a list of declaration and setting properties to various parts of the HTML. Whenever there is a dot (e.g. **.container**), that refers to a class name – the equivalent in HTML is `<div class="container"></div>`.

A word on its own (e.g. **ul**) refers to all the elements with that tag name. Words in blue after a colon are different states (hover or focussed). Finally, the ordering of the words separated by spaces show inheritance in the HTML so the browser knows where to look.

Now before any of that CSS will work, we need to tell our *index.php* page to include this file. We can do this by adding in one line of code into the `<head></head>` tags on **line 6**:

```

6  <link href='/style.css' rel='stylesheet' type='text/css'>

```

Launch the page now and you should see a login screen with an input field to enter a Screen Name. Feel free to modify the CSS and change the colours before we continue.

## Adding Interactivity

We're now going to introduce some **client-side scripting**, which will allow the user to enter their screen name and press "Submit".

Before we write some **JavaScript**, we need to add some more code into the `<head></head>` tags of our HTML mark-up.

Change *index.php* so that the `<head></head>` tags (**lines 4-11**) now look like this:

```
1 <head>
2     <link href='http://fonts.googleapis.com/css?family=Montserrat:700,400' rel='stylesheet'>
3     <link href='/style.css' rel='stylesheet' type='text/css'>
4     <link href="/icon.png" rel="shortcut icon" />
5     <title>Learn2Code Messenger</title>
6     <script type="text/javascript" src="jquery.js"></script>
7     <script type="text/javascript" src="client.js"></script>
8 </head>
```

In this code, we've added a link to the *icon.png* file. This should make the icon in the top corner of the browser to be the messaging icon (you can change this image to whatever you like).

We also load the *jquery.js* file – a **JavaScript library** which allows us to do very useful things.

Finally, we include *client.js* – this is the JavaScript file we are about to create and it will contain all our client side logic.

Let's now create this new file *client.js* and put in it the following:

```
1 // A flag which is true when the user is logged in
2 var loggedIn = false;
3
4 // This is jQuery for "when the page first loads"
5 $(function () {
6     CreateAccount_Events();
7 });
8
9 // Add events for the screen_name input
10 function CreateAccount_Events() {
11     // When user types in input box
12     $('.login input.screen_name').keyup(function (e) {
13         if ($(this).val() === "") {
14             // If the textbox is blank, hide the submit button
15             $('.login .enter').fadeOut();
16         }
17         else {
18             // If the textbox has text, show the submit button
19             $('.login .enter').fadeIn();
20             if (e.keyCode == 13) {
21                 // If the user presses <ENTER>, create the user
22                 CreateAccount_Submit();
23             }
24         }
25     });
26     // When the user clicks on the submit button, create the user
27     $('.a.enter').click(CreateAccount_Submit);
28 }
29
30 function CreateAccount_Submit() {
31     alert("Creating the user!");
32 }
```

## Creating the user

Before we can start to create actual users, we need to set up our database with the correct column names. Go back to the cPanel, under *Databases*, select *MySQL Databases*. Here you can choose a name for your database. Create it and copy the details of it into a file so you don't forget that.

In the cPanel under *Databases* select *phpMyAdmin*. Find your database and click *Connect Now!*.

Congratulations, your database is ready – at the moment though it's empty and you need to create a table. In this application, we'll be using 3 different tables. For now though, start by creating a table called **users**, it will have **5 columns**. Set the table up as follows:

Name	Type	Length	Default	Index	NULL	A_I
userID	INT	10	None	PRIMARY	x	✓
screen_name	VARCHAR	100	NULL	---	✓	x
code	VARCHAR	50	NULL	---	✓	x
lastOnline	DATETIME		NULL	---	✓	x
status	VARCHAR	20	NULL	---	✓	x

Now we can change our function in *client.js* to run a PHP script. **Change from lines 30 onwards:**

```

30  function CreateAccount_Submit() {
31      // Get the screen name and store it in a variable
32      screen_name = $('#screen_name').val();
33      // Hide the submit button and show the loading animation
34      $('#login .enter').fadeOut();
35      $('#login .loading').fadeIn();
36      // Make an AJAX request to insert the user into the database
37      $.post("/create_user.php", { screen_name: screen_name }, function () {
38          // On complete, after 2s reload the page
39          setTimeout(function () {
40              location.reload();
41          }, 2000);
42      });
43  }
```

AJAX is very powerful in JavaScript as it allows us to open a PHP page without the user even knowing! Let's now make a new file *create\_user.php* and tell it to create this new user. When you're done, run the script and then check the database to see the result.

```

1
2  <?php
3  session_start();
4
5  if ($_POST['screen_name']) {
6      // Connect to the database
7      $db = new mysqli("host", "user", "password", "database");
8      $screen_name = $_POST['screen_name'];
9
10     // Generate a random password of 50 letters and numbers
11     $chars = array_merge(range('a', 'z'), range(0, 9));
12     shuffle($chars);
13     $password = implode(array_slice($chars, 0, 50));
14
15     // Insert into database
16     $db->query("INSERT INTO users (screen_name, code, status)
17     VALUES ('$screen_name', '$password', 'offline')");
18
19     // Store this password in a cookie to remember the user
20     // Expire the cookie in 1 year
21     setcookie("learn2code_user", $password, time()+86400*30*12, "/");
22
23 }
```

## Detecting the user

Now that a cookie is on the user's computer, when we refresh the page, we can run some code which checks if this cookie exists. We're going to make a new function and run it on page load. Add a new line to *client.js* to run on page load:

```
7 DetectLoggedIn();
```

So what will this function *DetectLoggedIn* do? Well we're going to make another AJAX call to a PHP script to check if this cookie exists and if it does, we'll show a simple popup box:

```
46 // Run an AJAX call to see if the cookie is set
47 function DetectLoggedIn() {
48     $.post("/detect_user.php", function (data) {
49         // Interpret the response
50         user_data = JSON.parse(data);
51         if (user_data["sn"] != "") {
52             // We have a result!
53             loggedIn = true;
54             alert("You are logged in!");
55         }
56         else {
57             // User is not logged in, show the login screen
58             $('#login').show();
59         }
60     });
61 }
```

Make a new file *detect\_user.php* and this script will try and find the cookie and if it exists, match the password with the user in the database.

```
1 <?php
2 session_start();
3
4 // If the cookie exists
5 if (isset($_COOKIE['learn2code_user'])) {
6
7     // Connect to the database and get the value of the cookie
8     $db = new mysqli("host", "user", "password", "database");
9     $code = $_COOKIE['learn2code_user'];
10    // Query the database to find the user
11    $find = $db->query("SELECT * FROM users WHERE code = '$code'");
12
13    if ($find->num_rows > 0) {
14        // If the user exists, store the ID in a local session and print the result
15        $row = $find->fetch_assoc();
16        $_SESSION['userID'] = $row["userID"];
17        echo json_encode(array("sn" => $row["screen_name"], "id" => $row["userID"]));
18    }
19    else {
20        // If the user is invalid, delete the cookie and return a not found error
21        setcookie("learn2code_user", null, -1, "/");
22        unset($_SESSION['userID']);
23        echo json_encode(array("sn" => "", "error" => "Not found"));
24    }
25
26 }
27 else {
28     echo json_encode(array("sn" => "", "error" => "No cookie"));
29 }
30
31 ?>
```

## Writing the messaging source

We now have a popup telling the user that they are logged in, but now we need to set up the page for them to interact with the application. Let's add the two panels to the *index.php* file, modifying from **lines 40**:

```

40 <div class="messages"><div class="this-user">
41     
42     <div class="name"><i class="status online"></i></div>
43 </div><ul></ul></div>
44
45 <div class="contents">
46     <textarea placeholder="Write a message" class="write"></textarea>
47     
48 </div>

```

And of course, we can't add markup code without giving it styling. Append this code to your *style.css* file:

```

1  .avatar { width: 35px; border-radius: 50%; margin: 5px 0px; border: 1px solid #888; }
2  i.status.online { background: #32A932 !important; }
3  i.status.busy { background: #E2D81F !important; }
4  i.status.unavailable { background: #A90606 !important; }
5  i.status.offline { background: #AAA !important; }
6
7  .container .messages { display:none; width: 300px; position: absolute; height: 100%;
8                          overflow: hidden; margin-left: 58px; background: #E6E9ED;
9                          border-right: 2px solid #9A9EA5;}
10 .container .messages .this-user { height: 50px; background: #D8D8D8;
11                                   border-bottom: 2px solid #63656A;}
12 .container .messages .this-user .avatar { border: none; margin: 9px 14px 0; }
13 .container .messages .this-user .name { position: absolute; font-weight: bold;
14                                         font-size: 18px; margin: -31px 0 0 64px; }
15 .container .messages .this-user .status { width: 13px; height: 13px; border-radius:50%;
16                                             position: relative; margin: 0 0 0 15px; border: 1px solid #999;
17                                             cursor: pointer; display: inline-block; vertical-align: -2px; }
18 .container .messages ul li { border-bottom: 1px solid #C2C9D6; padding: 10px 15px;
19                               list-style: none; height: 40px; }
20 .container .messages ul li:hover { cursor: pointer; background: #E9EEF5; }
21 .container .messages ul li .avatar { width: 30px; float: left; }
22 .container .messages ul li .name { font-weight: bold; display: block; margin: 2px 15px;
23                                   float: left; }
24 .container .messages ul li .recent-message { float: left; display: block;
25 margin: -20px 0 0 46px; white-space: nowrap; text-overflow: ellipsis; width: 225px;
26 overflow: hidden; }
27
28 .container .contents { display:none; width: 540px; position: absolute; height: 100%;
29                       overflow: hidden; margin-left: 360px; background: #E6E9ED; }
30 .container .contents .top-bar { background: #D6D6D6; height: 50px;
31                               border-bottom: 2px solid #CACACA; }
32 .container .contents .top-bar .name { text-align: center; font-size: 18px; margin-top: 16px;
33                                       position: absolute; width: 93%; }
34 .container .contents .top-bar i.status { width: 8px; height: 8px; background: #fff;
35 position: absolute; border-radius: 50%; margin: 5px 17px; border: 1px solid #888; }
36 .container .contents .top-bar .close { height: 22px; position: absolute; right: 0;
37                                       margin: 15px 20px; cursor: pointer; }
38 .container .contents .scrollable { height: 345px; overflow-y: scroll;
39                                   border-bottom: 1px solid #ccc; }
40 .container .contents textarea.write { border: 0; resize: none; font: 14px Arial, sans-serif;
41                                       width: 450px; height: 32px; padding: 10px;
42                                       outline: none; overflow-y:scroll; padding-right:80px; }
43 .container .contents img.loading { display:none; position: absolute; height: 30px;
44                                   margin: -44px 37px 0 0; right: 0; }

```



## Logging in

For this part, we're going to show the relevant panels to the user once they're logged in. First, we need to modify what happens when we detect a logged in user in *client.js*:

```
54  $('.messages').show();
55  $('.messages .this-user .name').prepend(user_data["sn"]);
56  UpdateStatus('online');
57  UpdateOnlineTime();
58  FetchConvoData(true);
59  ClicksOnChatBar();
```

Throughout the rest of the development, we're going to write all these functions. For now, so that we don't get an error, let's create the empty functions.

```
68  function UpdateStatus(text) {}
69
70  function UpdateOnlineTime() {}
71
72  function FetchConvoData(loop) {}
73
74  function ClicksOnChatBar() {}
75
76  function HoversOnChatBar() {}
```

Now if you preview the application now, the panels should be showing but you'll see the login screen behind which shouldn't be there.

Since we already said in the JavaScript that if you are not logged in, it will show the login screen, we change the *style.css* file to say that by default, the login screen is hidden:

```
14  .container .login { display: none;
```

The first function we're going to write is **UpdateOnlineTime**. This function will call a script every 30 seconds which updates the database when we were last online. When the user closes the browser, the script will stop running and so other users will see that we've logged off. Edit *client.js* to add this function:

```
70  function UpdateOnlineTime() {
71      $.post("/still_online.php");
72      setTimeout(UpdateOnlineTime, 30000);
73  }
```

Now we can create this file *still\_online.php* and it will look as follows:

```
1  <?php
2  session_start();
3
4  // Connect to database
5  $db = new mysqli("host", "user", "password", "database");
6  $text = $_POST['status'];
7
8  // Get the user from the session variable
9  $userID = $_SESSION['userID'];
10
11 // When working with date/times, make sure timezone is right
12 date_default_timezone_set('Europe/London');
13 $time = date("Y-m-d H:i:s");
14
15 // Update database
16 $db->query("UPDATE users SET lastOnline = '$time' WHERE userID = '$userID'");
17 ?>
```



## Changing online status

When you're logged in, it would be useful to be able to tell others if you want to talk or if you're unavailable. By default, when a user logs in, they should be online.

Let's make the function **UpdateStatus** which takes a parameter to update the database with this value:

```
68 function UpdateStatus(text) {
69     $.post("/update_status.php", { status: text });
70 }
```

The script which will update the status will be a file called `update_status.php`

```
1  <?php
2  session_start();
3
4  if ($_POST['status']) {
5
6      // Connect to database
7      $db = new mysqli("host", "user", "password", "database");
8      $text = $_POST['status'];
9
10     // Update the status
11     $userID = $_SESSION['userID'];
12     $db->query("UPDATE users SET status='$text' WHERE userID = '$userID'");
13
14 }
15 ?>
```

We also want the user to be able to scroll through the statuses by clicking on the colour of the dot:

```
78 function ChangeStatusEvents() {
79     // When user clicks on status dot
80     $('.container .messages .this-user i.status').click(function () {
81         // Set up list of available statuses
82         var modes = ["online", "busy", "unavailable"];
83         // Change dot to match new status
84         current = $('.container .messages .this-user i.status').attr('class').split(" ")[1];
85         next_index = (modes.indexOf(current) + 1) > (modes.length - 1)
86                     ? 0 : modes.indexOf(current) + 1;
87         $('.container .messages .this-user i.status').removeClass(current);
88         $('.container .messages .this-user i.status').addClass(modes[next_index]);
89         // Update database with new status
90         UpdateStatus(modes[next_index]);
91     });
92 }
```

Finally, we need to trigger this function to run when the page loads:

```
8 ChangeStatusEvents();
```

Now you should be able to change the status of the user and see it update in the database.

## Finding online users

When more than one user is logged in, they need to be able to find others to talk to.

The function **FetchChatData** will start as soon as the page loads – regardless of whether the user is logged in or not and will keep running. Let's add it to the "page load" events:

```
9 FetchChatData();
```

The function should check for users who are online and update the left most panel with pictures which on hover will show their screen name.

```
101 function FetchChatData() {
102     // Make a request to find online users
103     $.post("/fetch_online.php", function (data) {
104         // Clear the current people panel
105         $('.people ul').html('');
106         // Convert the results from object notation to an array
107         online_data = JSON.parse(data);
108         // Loop through results
109         for (x in online_data["information"]) {
110             // The current person in the list
111             result = online_data["information"][x];
112             // Add the HTML to the people panel
113             $('.people ul').append('<li data-id="'+result[0]+'"\
114                 data-hover="'+result[1]+'">\
115                 <i class="status '+result[2]+'"></i>\
116                 \
117                 </li>');
118         }
119         HoversOnChatBar();
120         if (loggedIn) { ClicksOnChatBar(); }
121     });
122     setTimeout(FetchChatData, 30000);
123 }
```

The *fetch\_online.php* file will query the database and provide the appropriate information:

```
1 <?php
2 session_start();
3 $db = new mysqli("host", "user", "password", "database");
4
5 // Create a time for 1 minute ago
6 date_default_timezone_set('Europe/London');
7 $last_online = date("Y-m-d H:i:s", strtotime("1 minute ago"));
8
9 // Query to find all users that are not offline,
10 // who have been online for more than 1 minute and not this user
11 $userID = $_SESSION['userID'];
12 $all_users = $db->query("SELECT * FROM users WHERE status != 'offline'
13 AND lastOnline >= '$last_online' AND userID != '$userID'");
14
15 // Set up results and store in array
16 $output = ["number_online" => $all_users->num_rows, "information" => []];
17 while ($this_user = $all_users->fetch_assoc()) {
18     $output["information"][] =
19         [$this_user["userID"], $this_user["screen_name"], $this_user["status"]];
20 }
21
22 // Output results in object notation
23 echo json_encode($output);
24 ?>
```

## Formatting the chat bar

For this part, we're going to add events for hovering and clicking on the chat bar to start a conversation with that user.

First we need to style the chat bar to show the status icon.  
Add in these lines to the middle of *style.css* from lines 13:

```
13 .container .people li { padding: 10px; height: 43px; border-bottom: 1px solid #37424C; }
14 .container .people li i.status { width: 8px; height: 8px; background: #fff; position: absolute;
15     border-radius: 50%; margin: 31px 1px; border: 1px solid #888; }
```

Add the following code into *client.js* to add a tooltip when you hover over a user in the chat bar:

```
83 function HoversOnChatBar() {
84     // When you hover on the chat bar over a user
85     $('.people ul li[data-hover]').hover(function () {
86         // Create a tooltip and position it at the right place
87         $('.tooltip').remove();
88         $('body').append(
89             '<div class="tooltip">' + $(this).find('i.status')[0].outerHTML
90             + $(this).attr('data-hover') + '</div>'
91         );
92         t = $(this).position().top + $('.container').position().top + 30;
93         l = $(this).position().left + $('.container').position().left - 408;
94         $('.tooltip').css({ top: t, left: l });
95     }, function () {
96         // When off hover, remove tooltip
97         $('.tooltip').remove();
98     });
99 }
```

Finally, let's add the styles for the tooltip. Add this code to the end of *style.css*:

```
82 .tooltip { background: #2B3848; position: absolute; padding: 6px 12px; border-radius: 6px; }
83 .tooltip i.status { width: 10px; height: 10px; background: #FFF; position: relative;
84     border-radius: 50%; display: inline-block; margin: 0 8px 0 0; }
85 .tooltip:before { content: " "; position: absolute; width: 0px; height: 0px; margin: 0 -30px;
86     border: 9px solid rgba(0, 0, 0, 0); border-right: 13px solid #2B3848; display: inline-block; }
```

So what should happen when you click on a user? Well we want to either start a new conversation or open up an existing one. Enter the following code to *client.js* for the function *ClicksOnChatBar*.

```
81 function ClicksOnChatBar() {
82     $('.people ul li[data-id]').click(function () {
83         // Get user id
84         other_id = $(this).attr('data-id');
85         // If conversation already exists, launch it
86         convoExists = $('.messages ul li[data-otherid="'+other_id+'"]');
87         if (convoExists.size()) {
88             convoExists.trigger('click');
89         }
90         else {
91             // Otherwise, create a new conversation
92             NewConversation(other_id);
93         }
94     });
95 }
```

## Making a new conversation

Let's set up what should happen to create a new conversation with a new user.

Let's create the HTML in JavaScript and send it to the browser. We're going to create the function **NewConversation** in *client.js*:

```

155 function NewConversation(user_id) {
156     // Set default variables for other user
157     other_user_status = "offline";
158     other_user_name = "";
159
160     // Try and found the other user's information
161     for (j in online_data["information"]) {
162         if (online_data["information"][j][0] == user_id) {
163             // Get the other user's name and status
164             other_user_status = online_data["information"][j][2];
165             other_user_name = online_data["information"][j][1];
166         }
167     }
168
169     // Set up the HTML
170     $('#.contents').show();
171     $('#.contents .top-bar, .contents .scrollable').remove();
172     $('#.contents').prepend('\
173         <div data-convoid="" data-otherid="'+user_id+'" class="top-bar">\
174         <div class="name">'+other_user_name+'\
175         <i class="status '+other_user_status+'"></i></div>\
176         \
177         </div><div class="scrollable"></div>');
178
179     $('#.contents .top-bar .close').click(function () {
180         $('#.contents').hide();
181     });
182
183     // Set up events to allow user to quickly type in message box
184     EnterInTextarea();
185 }
186
187 function EnterInTextarea() {
188     $('#.contents textarea.write').unbind('keypress');
189     $('#.contents textarea.write').keypress(function (e) {
190         // If user presses <ENTER> key (key code is 13)
191         if (e.keyCode == 13) {
192             SendMessage();
193             e.preventDefault();
194             return false;
195         }
196     });
197 }
```

The **EnterInTextarea** function will call **SendMessage** when the user presses the <ENTER> key.

In the next part, we're going to write the **SendMessage** function and the PHP script which goes with it.

## Sending a message

We're going to send our first message. Before we get started, we need to add 2 tables to the database. Find phpMyAdmin in cPanel. Create tables 'conversations' with 3 columns and 'messages' with 4 columns:

Name	Type	Length	Default	Index	NULL	A_I
convolD	INT	10	None	PRIMARY	x	✓
user0	INT	10	NULL	---	✓	x
user1	INT	10	NULL	---	✓	x
messageID	INT	10	None	PRIMARY	x	✓
convolD	INT	10	NULL	---	✓	x
toUser	INT	1	NULL	---	✓	x
text	LONGTEXT	---	NULL	---	✓	x

Now we can fill in the **SendMessage** function in *client.js*

```

199 function SendMessage() {
200     new_message = $(''.contents textarea.write').val();
201     $(''.container .contents img.loading').fadeIn();
202     // Make an AJAX call to send the message
203     $.post("/send_message.php",
204     { other_user: $(''.contents .top-bar').attr('data-otherid'), text: new_message },
205     function (newConvoID) {
206         // If a conversation was created, set the attribute of the message box
207         if (newConvoID != "") {
208             $(''.contents .top-bar').attr('data-convolid', newConvoID);
209         }
210         $(''.contents textarea.write').val('');
211         // Refresh the message panel
212         FetchConvoData(false);
213     });
214 }
```

Create a new file called *send\_message.php*

```

1  <?php
2  session_start();
3  $db = new mysqli("host", "user", "password", "database");
4
5  $userID = $_SESSION['userID'];
6  $other_user = $_POST['other_user'];
7  $message = $db->real_escape_string($_POST['text']);
8
9  $convo = $db->query("SELECT * FROM conversations WHERE
10 (user0='$userID' AND user1='$other_user') OR (user0='$other_user' AND user1='$userID')");
11
12 if ($convo->num_rows > 0) {
13     $info = $convo->fetch_assoc();
14     $to_user = ($info["user0"] == $userID) ? 1 : 0;
15     $db->query("INSERT INTO messages (convolid, toUser, text)
16     VALUES ('".$info[convolid]."', '$to_user', '$message')");
17 }
18 else {
19     $db->query("INSERT INTO conversations (user0, user1) VALUES ('$userID', '$other_user')");
20     $convolid = $db->insert_id;
21     $db->query("INSERT INTO messages (convolid, toUser, text)
22     VALUES ('$convolid', '1', '$message')");
23     echo $convolid;
24 }
25 ?>
```

## Receiving messages (1)

Now that the message has been sent and it is in the database, we can fill in the last vital function which checks for new messages and displays them. Fill out the **FetchConvoData** function in *client.js*:

```

79 function FetchConvoData(loop) {
80     // AJAX call to get conversations
81     $.post("/fetch_conversations.php", function (data) {
82         // Convert object notation back to JavaScript
83         convoInfo = JSON.parse(data);
84         $('.messages ul').html('');
85         // Loop through all conversations
86         for (x in convoInfo) {
87             id = convoInfo[x]["convoID"];
88             other_user = convoInfo[x]["other_user"];
89             recent_message = convoInfo[x]["messages"][convoInfo[x]["messages"].length-1];
90             // Show a snippet of the most recent message to max 150 characters
91             if (recent_message["text"].length > 150) {
92                 snippet = recent_message["text"].substr(0, 147) + "...";
93             }
94             else {
95                 snippet = recent_message["text"];
96             }
97             // If this user sent the message, put an "arrow" at the start
98             if (recent_message["to_me"] == false) {
99                 snippet = "&#8594; " + recent_message["text"];
100             }
101             // Add the HTML to the page
102             $('.messages ul').append('\
103             <li data-otherid="'+convoInfo[x]["other_id"]+'" data-convoid="'+id+'">\
104                 \
105                 <span class="name">'+other_user+'</span>\
106                 <span class="recent-message">'+snippet+'</span>\
107             </li>');
108         }
109         $('.container .contents img.loading').fadeOut();
110     });
111     // If part of the fetch cycle, run again in 30 seconds
112     if (loop) { setTimeout(function () { FetchConvoData(true) }, 30000); }
113 }

```

When we receive data from the server-side script, after converting it from object notation into an array, the variable *convoInfo* is likely to look something like this:

Array(

```

[0] => Array(
  "convoID" => 1,
  "other_id" => 3,
  "other_user" => "Some random user",
  "messages" => Array(
    [0] => Array("to_me" => true, text => "Hello person"),
    [1] => Array("to_me" => false, text => "Hello back")
  )
),
);

```

In the next part, we're going to write the server-side script which will generate this type of response.

## Receiving messages (2)

Finally, we can create the last server-side script which will read messages from the server. Create **fetch\_conversations.php** – this is another very long script but we'll go through all of it afterwards.

```

1  <?php
2  // Database and session
3  session_start();
4  $db = new mysqli("host", "user", "password", "database");
5
6  // Get user from session and query all messages
7  $this_user = $_SESSION['userID'];
8  $all_messages = $db->query("SELECT conversations.*, messages.* FROM messages
9  LEFT JOIN conversations ON messages.convoID = conversations.convoID
10 LEFT JOIN users ON (conversations.user0=users.userID OR conversations.user1=users.userID)
11 WHERE users.userID = '$this_user'");
12
13 // Loop through all messages
14 $output = [];
15 while ($this_message = $all_messages->fetch_assoc()) {
16
17     // Get information about the other user
18     $other_user = ($this_message["user0"] == $this_user)
19         ? $this_message["user1"] : $this_message["user0"];
20     $get_other_user = $db->query("SELECT screen_name FROM users
21     WHERE userID='$other_user'")->fetch_assoc();
22
23     // Create space for this convoID if first message
24     if (!isset($output[$this_message["convoID"]])) {
25         $output[$this_message["convoID"]] = [
26             "convoID" => $this_message["convoID"],
27             "other_id" => $other_user,
28             "other_user" => $get_other_user["screen_name"],
29             "messages" => []
30         ];
31     }
32
33     // Work out which user the message is directed to
34     $which_user = ($this_message["user0"] == $this_user) ? 0 : 1;
35
36     $output[$this_message["convoID"]]["messages"][] = [
37         "to_me" => ($which_user == $this_message["toUser"]),
38         "text" => $this_message["text"]
39     ];
40 }
41
42 // Send results as object notation
43 echo json_encode(array_values($output));
44
45 ?>

```

Essentially all this script is doing is running a query to receive all the messages and re-formats it all to produce the output on the page before.

If you refresh the page you should see all your messages appear – congratulations!

There's only one more thing to do which is to bring up the message box and allow you to reply, not long to go now!



## Opening up a message

Let's go back into *client.js* and find the **FetchConvoLoop** function. After we get all the information we need to add an event to allow the user to click in the panel to show their conversation, so insert the lines like this:

```

106     ActivateMessages();
107     if ($('.contents .top-bar').size()) {
108         convoId = $('.contents .top-bar').attr('data-convoid');
109         $('.messages ul li[data-convoid="'+convoId+'"]').trigger('click');
110     }

```

The **ActivateMessages** function will create an event to bring up the message box filled with information. The other lines say that if a conversation is already open, trigger its refresh.

```

252 function ActivateMessages() {
253     $('.messages ul li').click(function () {
254         convoID = $(this).attr('data-convoid');
255
256         for (x in convoInfo) {
257
258             // When we reach the requested conversation clicked
259             if (convoInfo[x]["convoID"] == convoID) {
260                 other_user_status = "offline";
261                 for (j in online_data["information"]) {
262                     if (online_data["information"][j][0] == convoInfo[x]["other_id"]) {
263                         other_user_status = online_data["information"][j][2];
264                     }
265                 }
266
267                 $('.contents').show();
268                 $('.contents .top-bar, .contents .scrollable').remove();
269                 $('.contents').prepend('\
270 <div data-convoid="'+convoID+'"\'
271 data-otherid="'+convoInfo[x]["other_id"]+'\' class="top-bar">\
272     <div class="name">'+convoInfo[x]["other_user"]+'\'
273     <i class="status '+other_user_status+'"></i></div>\
274     \
275 </div><div class="scrollable"></div>');
276
277                 for (y in convoInfo[x]["messages"]) {
278                     // Write html of each message
279                     message = convoInfo[x]["messages"][y];
280                     $('.contents .scrollable').append('<li class="message '+
281 (!convoInfo[x]["messages"][y]["to_me"]?"me":"") +'">\
282         \
283         <i class="arrow1"></i><i class="arrow2"></i>\
284         <span class="text">'+convoInfo[x]["messages"][y]["text"]+'</span>\
285 </li>');
286                 }
287
288                 // Allow user to close box
289                 $('.contents .top-bar .close').click(function () {
290                     $('.contents').hide();
291                 });
292
293                 // Scroll to the bottom of the box and leave
294                 $('.scrollable').scrollTop($('.scrollable').height());
295                 EnterInTextarea();
296                 break;
297             }
298         }
299     });
300 }
301 }
302 }

```

## Styling the message

Everything in your code should be working now but message aren't styled particularly well. We want to achieve a nice bubble effect so it looks like people are speaking to each other. Add this last section into your *style.css* file:

```

88 .contents .scrollable { list-style: none; }
89 .contents .scrollable .avatar { margin: 15px 15px 0; position: relative; float: left;}
90 .contents .scrollable .text { max-width: 205px; background: #fff; padding: 7px 14px;
91 margin: 19px 0 15px -5px; display: inline-block; border: 1px solid #ccc; border-radius: 9px; }
92
93 .contents .scrollable .arrow1 { border:8px solid rgba(0, 0, 0, 0); border-right:10px solid #FFF;
94 margin: 27px 0 0 -12px; display: inline-block; position: relative; vertical-align: top;}
95 .contents .scrollable .arrow2 { border:8px solid rgba(0, 0, 0, 0); border-right:10px solid #CCC;
96 margin: 27px 0 0 -20px; display: inline-block; vertical-align: top;}
97
98 .contents .scrollable .me { text-align: right; }
99 .contents .scrollable .me .avatar { float: right; margin: 15px 15px 0 0; }
100 .contents .scrollable .me .text { background: #2B94D9; color: #FFF; margin: 19px 0px 15px 0; }
101
102 .contents .scrollable .me .arrow1 { margin:26px 7px 0 -18px;border-right-color:rgba(0, 0, 0, 0);
103 border-left-color: #2B94D9; float: right;}
104 .contents .scrollable .me .arrow2 { border-right-color:rgba(0, 0, 0, 0); border-left-color:#CCC;
105 float: right; margin: 26px 0 0 -1px;}

```

And you're done!

There are lots of ways you can improve this basic messaging system. A few ideas are:

- Introduce a sound notification when you receive a message
- Include the time when you send message
- Have a "seen" field which tells the other user when you've seen their message
- Include a "typing" feature, which lets the other user know when you start typing a message
- Allow users to change their profile pictures to recognise them on the left panel
- Allow searches through messages
- Users can set their own passwords so they can log in from any computer

Well done on completing your first social network!