

PHP

How to Code a Signup Form with Email Confirmation

by Matt Vickers 23 Sep 2009

f

25

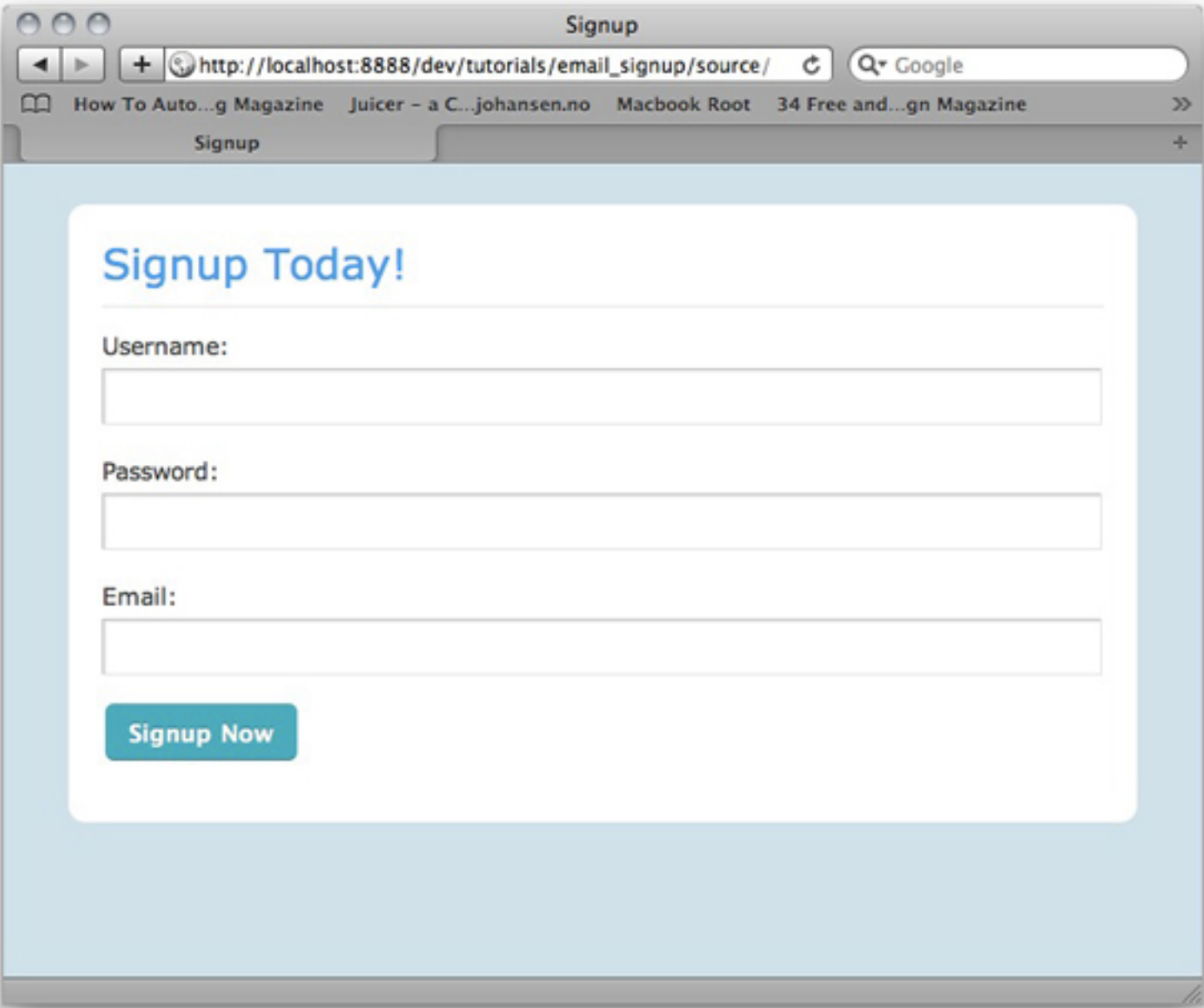
t

8

g+

19

p



In this tutorial, we are going to be creating a user signup form that adds a user to a database, and then sends out a confirmation email that the user must click on before

their account will be activated.

Step 1: The Template

I've included the basic site layout so we aren't wasting time creating the form and making the site looks pretty. We are going to get right into coding which is what you came here for.

Open up the Site Template folder and copy it to either your localhost or web server.

Open up `index.php` and take a quick look. You'll see a simple form with 3 inputs. These are the fields we are going to capture. We want the username, their password as well as their email. You can choose to capture other elements when users are signing up, but these are the 3 barebones elements we need.

id	int	↕	11	<div>u z</div>	NO	↕	NULL	auto_increment
username	varchar	↕	50		NO	↕		
password	varchar	↕	128		NO	↕		
email	varchar	↕	250		NO	↕		
active	binary	↕	1		NO	↕	0	

Step 2: Setting up the MySQL Database

Open up PHPMysqlAdmin or whatever program you use to manage your MySQL database and create a new database. You can name this whatever you like. Now we want to create the rows that are going to hold our user information and confirmation information. For this we create two tables. Users and Confirm.

```
1 CREATE TABLE `users` (  
2   `id` int(11) NOT NULL auto_increment,  
3   `username` varchar(50) NOT NULL default '',  
4   `password` varchar(128) NOT NULL default '',  
5   `email` varchar(250) NOT NULL default '',  
6   `active` binary(1) NOT NULL default '0',  
7   PRIMARY KEY (`id`)  
8 ) ENGINE=MyISAM AUTO_INCREMENT=27 DEFAULT CHARSET=utf8;
```

Our first table has 5 rows. The first is the ID that is given to the user when they signup. This is set to auto increment so that each user is given a unique ID. Next is the username, password and ID. The last row lets us set the users active state. When we first create the user row, the active state will default to 0. This means that the users account is currently inactive. Once the user confirms their account we will set this to 1. This will state that the account is active.

```
1 CREATE TABLE `confirm` (  
2   `id` int(11) NOT NULL auto_increment,  
3   `userid` varchar(128) NOT NULL default '',  
4   `key` varchar(128) NOT NULL default '',  
5   `email` varchar(250) default NULL,  
6   PRIMARY KEY (`id`)  
7 ) ENGINE=MyISAM AUTO_INCREMENT=27 DEFAULT CHARSET=utf8;
```

Our second table is the confirm table. This holds the user's ID and email as well as a randomly generated key that we will use to confirm the users account.

Step 3: Connecting to the MySQL Database

Open up inc/php/config.php.

First we need to make the connect to the database.

```
1 mysql_connect('localhost', 'username', 'password') or die("I could
```

Depending on your setup, we are going to need to change a few variables. So go ahead and fill in everything.

Next we need to tell MySQL which database we want to use.

```
1 mysql_select_db('your_database_name') or die("I couldn't find the
```

Once everything has been edited to fit your database go ahead and point to the index.php file on your server.

If you don't see any errors at the top, we are all connected.

Step 4: Submitting the Form

Ok, now that we are all connected to the database, we need to capture the form data so we can get the user signed up.

I'm going to give you the piece of code and then explain what's going on. After that we are going to make changes and add functionality.

Here is the base; place this right after the first includes at the top of index.php

```
1 //check if the form has been submitted
2 if(isset($_POST['signup'])){
3
4 }
```

This if statement is checking to see if the form has been submitted.

Without this, our script would run every time the page is refreshed and we don't want that.

Note: Depending on your application or just general style of coding this code may be placed in a separate file that is accessed when the form is submitted. I've placed the code all in one file to keep things simple and easy to follow along.

Step 5: Cleaning up and Checking the Variables

We want to make sure that the user has submitted actual content instead of just a blank form, so we are going to perform some quick checks.

The first part is to place the `$_POST` variables into simpler variables and clean them for the database. Place this inside our if statement.

```
1 $username = mysql_real_escape_string($_POST['username']);
2 $password = mysql_real_escape_string($_POST['password']);
3 $email = mysql_real_escape_string($_POST['email']);
```

`mysql_real_escape_string()` makes sure that the user isn't trying to use apostrophes to access our database with MySQL injection. Whenever you want to put information into a database the the user has inputed, please run it through `mysql_real_escape_string()`. For more information on MySQL injection you can [read this article on Wikipedia](#)

So, we've cleaned up our variables, now let's check to see if the user forgot any fields.

```
1 if(empty($username)){ //put code in me please }
2 if(empty($password)){ //put code in me please }
3 if(empty($email)){ //put code in me please }
```

Now we have three if statements that are checking if each field is empty. If the field is empty we are going to assign some variables.

To make things clean we are going to create an array that will hold the status of the signup process as well as any text we need to show the user.

Right above that piece of code, let's create an array and a few variables.

```
1 $action = array();
2 $action['result'] = null;
3
4 $text = array();
```

First we are creating a blank array called action and then setting an array value of result. Result is going to hold a value of either success or error. Next we create another blank array called text. This is going to hold any text we want to show the user during the signup.

Right now, our if statements that are checking our variables aren't executing any code, so let's go ahead and put some code inside the first if statement.

Put this code inside the username if statement.

```
1 | $action['result'] = 'error';
2 | array_push($text, 'You forgot your username');
```

Let's say the user submits the form without a username. Our statement is going to run the code above. First it's going to set the result field of our action array to error.

Then we are going to use `array_push()` to put some text into our text array. We are going to be using this same piece of code for the final two "if" statements so copy and paste that code into the last two if statements. You'll probably want to change the text to match the current if statement.

Note: We are using `array_push()` in case we have multiple errors in the form submission. If all if statements are executed, the text array will look like:

```
1 | Array(
2 |     [0] => 'You forgot your username',
3 |     [1] => 'You forgot your password',
4 |     [2] => 'You forgot your email'
5 | )
```

We now need to check if we have any errors so we can continue on with the signup process.

Step 6: No Errors, Let's Signup the User

We are going to check to see if our action array result value is set to error.

```
1 | if($action['result'] != 'error'){
2 |     //no errors, continue signup
3 |     $password = md5($password);
4 | }
5 |
6 | $action['text'] = $text;
```

We are also running our password through the `md5()` function. This takes the password and returns a 32 character string that looks something like this:

`a3470ce826283eca7ce3360d0f26b230`. It's good practice to run the password

through some sort of hashing function before putting it into the database. This prevents people from viewing the users passwords if your database is hacked.

A quick check of our action result value and we can continue on with the signup. If our result is error we will skip over all this code and output the errors to our user so they can make the necessary changes.

The last piece of this code we are putting the values of your text array into our action array.

Step 7: Adding the User to the Database

Place this code inside our last if statement.

```
01  ...
02  If Statement checking for errors
03  ...
04
05  //add to the database
06  $add = mysql_query("INSERT INTO `users` VALUES(NULL, '$username', '
07
08  if($add){
09
10      //the user was added to the database
11
12  }else{
13
14      $action['result'] = 'error';
15      array_push($text, 'User could not be added to the database. Re
16      =
17  }
```

We use `mysql_query()` and `INSERT` to insert the users information into the database. Next, we create another if statement checking to see if the user was added to the database. We do this by checking if the `$add` variable is true or false.

If the user is added we can continue on with the signup; if not we are going to assign some familiar variables and stop the signup.

When working with MySQL queries, we use the `mysql_error()` function if there are errors because it helps with debugging what is wrong with your queries. It will output text errors when something is wrong. This is good!

Step 8: Confirmation is Needed

The user has submitted the form, everything checks out and they're now living in the database. We want the user to be able to use their account, so let's setup the confirmation.

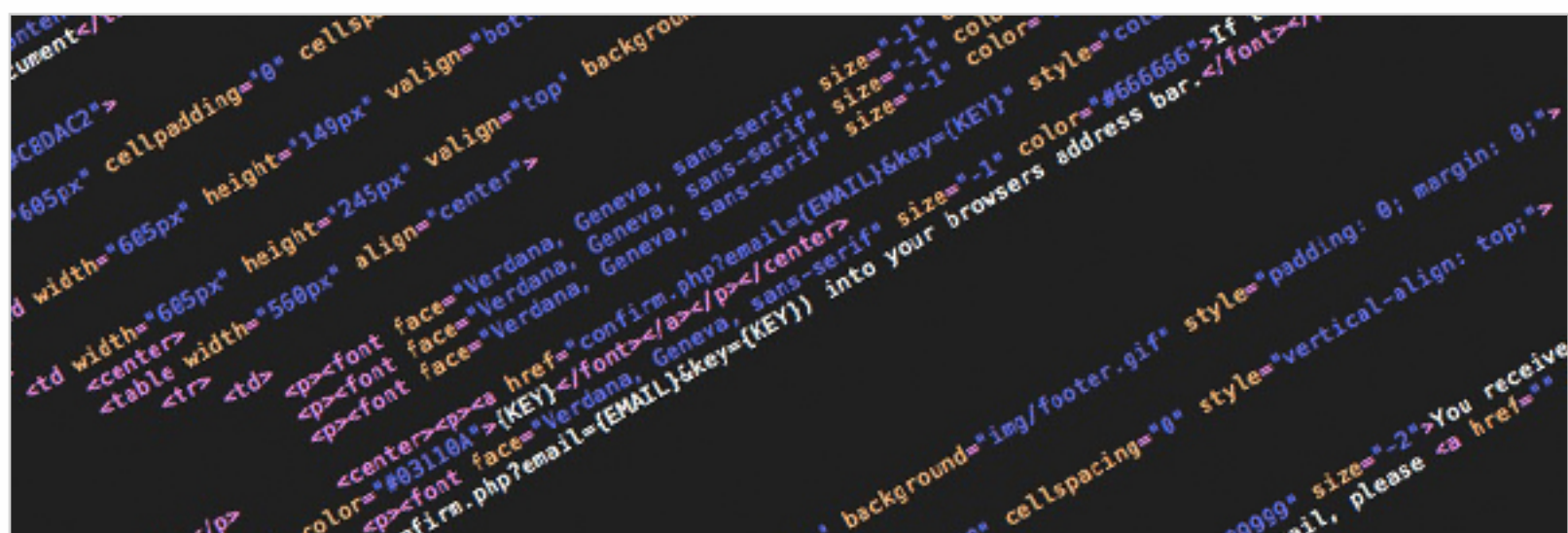
```
01  ...
02  if added check
03  ...
04
05  //get the new user id
06  $userid = mysql_insert_id();
07
08  //create a random key
09  $key = $username . $email . date('mY');
10  $key = md5($key);
11
12  //add confirm row
13  $confirm = mysql_query("INSERT INTO `confirm` VALUES(NULL, '$userid'");
14
15  if($confirm){
16
17      //let's send the email
18
19  }else{
20
21      $action['result'] = 'error';
22      array_push($text, 'Confirm row was not added to the database. ');
23
24  }
```

To make things easy, let's assign the new user id to a variable so we can use it later. We do this by using `mysql_insert_id()`. This will set `$userid` to whatever the

new user's ID is.

Next we create the random key for that specific user. We create a variable named key and fill it with a value of the username, email and date. The string will look like mattmatt@email.com012009. After that we use the md5() function to convert it to a random string that is unique to that user.

Using mysql_query() and INSERT again, we put the new user ID, the key and the users email into the database.



Step 9: Setting up the Email Templates

We are going to take a break from the PHP coding and create two new files. For the sake of being quick and easy we are actually going to use two templates that I've included with this tutorial. The two files we're going to be looking at are signup_template.html and signup_template.txt. Swift lets us assign an HTML as well as a TXT version of the email incase the users email client doesn't support HTML emails.

Open up signup_template.html Note: You can read up on HTML in emails over at carsonified. We aren't going to be editing this file, i'm just going to explain whats going on and then you can play around with it once the tutorial is complete. The most important part of this file is the tags that look like {USERNAME} and confirm.php?email={EMAIL}&key={KEY}. We are going to write a function that uses this template and replaces those tags with the variables from our form.

Step 10: The Template Function

Open up `inc/php/functions.php` and place this code inside.

```
01 function format_email($info, $format){
02
03     //set the root
04     $root = $_SERVER['DOCUMENT_ROOT'].'/dev/tutorials/email_signu
05
06     //grab the template content
07     $template = file_get_contents($root.'/signup_template.'.$format
08
09     //replace all the tags
10     $template = ereg_replace('{USERNAME}', $info['username'], $tem
11     $template = ereg_replace('{EMAIL}', $info['email'], $template
12     $template = ereg_replace('{KEY}', $info['key'], $template);
13     $template = ereg_replace('{SITEPATH}', 'http://site-path.com',
14
15     //return the html of the template
16     return $template;
17
18 }
```

`format_email()` is taking two variables which will be used in index.php. The first is our form information array and the second is format. We have a format variable so we can re-use this array for both the HTML and TXT versions of the template.

First we set the root. This points to the folder that the templates are hosted.

Next we open up the contents of our template and assign it to a variable.

Now we are going to use `ereg_replace()` to replace our `{USERNAME}` tags in our template with the content from our form. It's basically just a super simple template system.

Lastly we return the template variable which holds all the html.

Explanation: In a nutshell, `format_email()` opens up our template files, takes the HTML and assigns it to our variable. This is just a cleaner way then assigning all the

HTML in the function itself.

Step 11: Sending the Email

We are going to write another function to deal with Swift and sending the emails.

```
01 function send_email($info){
02
03     //format each email
04     $body = format_email($info,'html');
05     $body_plain_txt = format_email($info,'txt');
06
07     //setup the mailer
08     $transport = Swift_MailTransport::newInstance();
09     $mailer = Swift_Mailer::newInstance($transport);
10     $message = Swift_Message::newInstance();
11     $message ->setSubject('Welcome to Site Name');
12     $message ->setFrom(array('noreply@sitename.com' => 'Site Name'));
13     $message ->setTo(array($info['email'] => $info['username']));
14
15     $message ->setBody($body_plain_txt);
16     $message ->addPart($body, 'text/html');
17
18     $result = $mailer->send($message);
19
20     return $result;
21
22 }
```

Just like `format_email()`, `send_email()` takes our info array as a variable. The first part of the function we assign two variables, `$body` and `$body_plain_text`. We are using `format_email()` to assign the HTML values of our template to each variable. Now comes the good part. We have setup the swift instance using `Swift_MailTransport::newInstance()` and then setup the mailer using `Swift_Mailer::newInstance($transport)`;

We create a new instance of the Swift message and start to assign some variables to this instance. We set the subject, from email and to email address and then use `setBody()` to assign out text version of the email to the mailer instance. To add the

HTML version we use `addPart()`. The `send()` function takes care of the sending of the email and then we return the result. Alright, we have our email create and send functions written, let's go back to `index.php` and start to wrap up the main signup.

Step 12: Did we Send? Shall we Confirm?

Our last bit should've been the if statement checking if the confirm row was created.

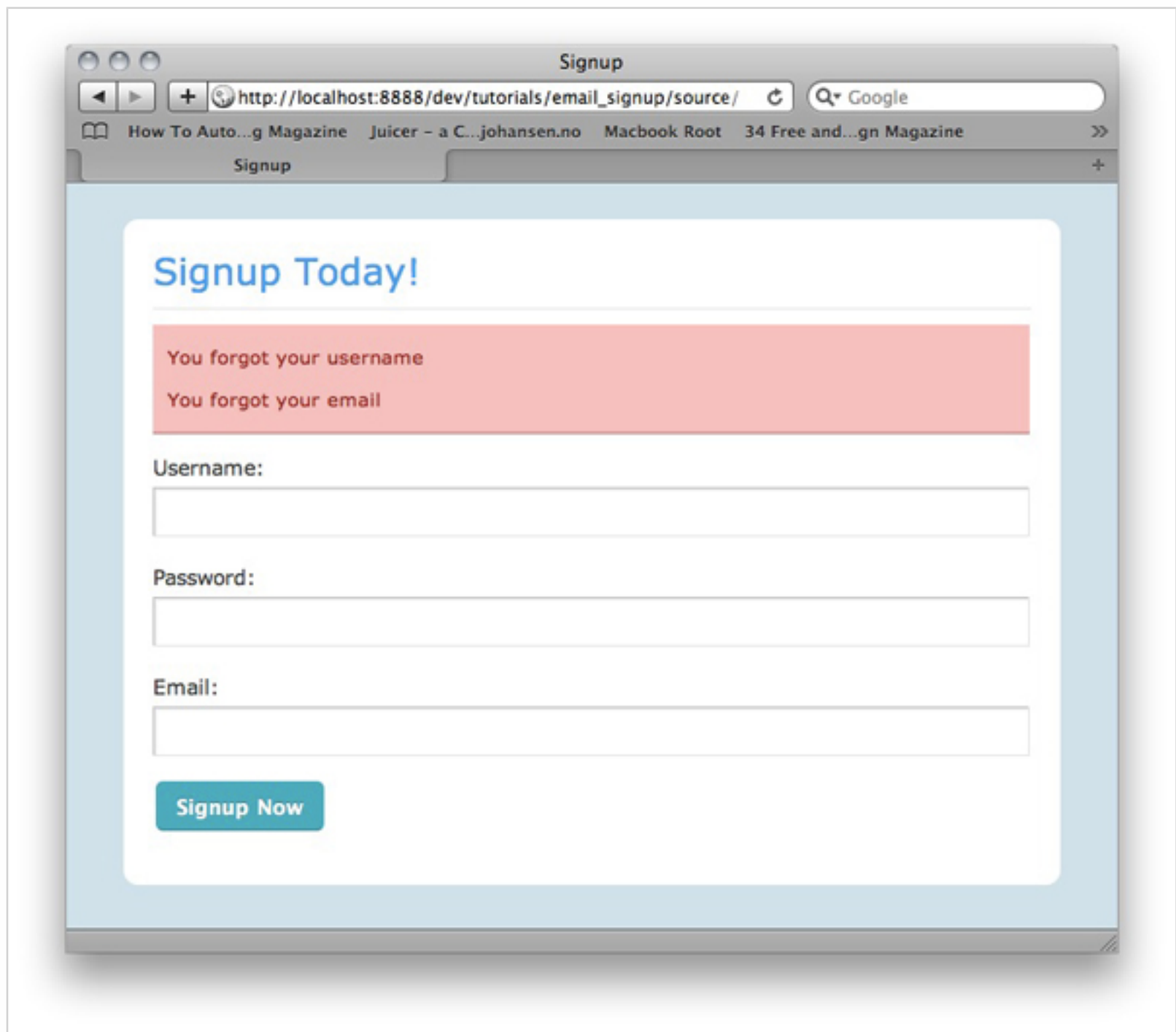
Let's send the email and check if everything went though alright.

```
01  ...
02  if confirm
03  ...
04
05  //include the swift class
06  include_once 'inc/php/swift/swift_required.php';
07
08  //put info into an array to send to the function
09  $info = array(
10      'username' => $username,
11      'email' => $email,
12      'key' => $key
13  );
14
15  //send the email
16  if(send_email($info)){
17
18      //email sent
19      $action['result'] = 'success';
20      array_push($text, 'Thanks for signing up. Please check your em
21
22  }else{
23
24      $action['result'] = 'error';
25      array_push($text, 'Could not send confirm email');
26
27  }
```

Without the Swift class we can't send out any emails, so in our first line, we are including the swift class. We need to send our information to both of our new

functions, so we create a new array and assign our variables to it. I know I know, more if statements, but we need to check for errors to make it easier for the users. You always have to assume that users will make every possible mistake imaginable.

We wrap our `send_email()` function in another if statement as well as passing the `$info` array. If the email is sent we assign a value of success and thank the user for signing up. If there are errors we use the familiar variables. So now, we are almost done with the signup, just one last function needs to be created. Even though we are assigning all these error/success variables and text we haven't displayed this information to the user.



Move back to functions.php and paste this code.

```
01 //cleanup the errors
02 function show_errors($action){
03
```

```

03
04     $error = false;
05
06     if(!empty($action['result'])){
07
08         $error = "<ul class=\"alert $action[result]\">\".\n";
09
10         if(is_array($action['text'])){
11
12             //loop out each error
13             foreach($action['text'] as $text){
14
15                 $error .= "<li><p>$text</p></li>\".\n";
16
17             }
18
19         }else{
20
21             //single error
22             $error .= "<li><p>$action[text]</p></li>\";
23
24         }
25
26         $error .= "</ul>\".\n";
27
28     }
29
30     return $error;
31
32 }

```

This may seem confusing but it's really just making our success/errors looks nice.

First it checks to see if the array is empty so we aren't executing the code when it isn't needed.

Next it creates a ul tag and applies the result as a class. This will either be success or error and is aesthetic only.

We then check to see if the text variable is an array or simply a string. If it's a string, we wrap it in an li. If it's an array we loop through each array item and wrap it in an li.

Lastly, we close the ul and return the entire string.

If we move back to index.php and place this code right after including `header.php` we can wrap up this section.

```
1 ...
2 header include
3 ...
4
5 <?= show_errors($action); ?>
```

A quick little explanation. We are taking all the values of our action array and passing it to the `show_errors()` function. If there is any content it returns a nice unordered list.

Step 13: Confirming the User

We should have a good grip on how the script is functioning; so for this next script I'm going to give you the entire chunk of code and then go through it with you.

Open up `confirm.php` and paste this in-between the header include and your `show_errors()` function.

```
01 //setup some variables
02 $action = array();
03 $action['result'] = null;
04
05 //quick/simple validation
06 if(empty($_GET['email']) || empty($_GET['key'])) {
07     $action['result'] = 'error';
08     $action['text'] = 'We are missing variables. Please double ch
09 }
10
11 if($action['result'] != 'error'){
12
13     //cleanup the variables
14     $email = mysql_real_escape_string($_GET['email']);
15     $key = mysql_real_escape_string($_GET['key']);
16
17     //check if the key is in the database
18     $check_key = mysql_query("SELECT * FROM `confirm` WHERE `email`
```

```

19
20     if(mysql_num_rows($check_key) != 0){
21
22         //get the confirm info
23         $confirm_info = mysql_fetch_assoc($check_key);
24
25         //confirm the email and update the users database
26         $update_users = mysql_query("UPDATE `users` SET `active` :
27         //delete the confirm row
28         $delete = mysql_query("DELETE FROM `confirm` WHERE `id` =
29
30         if($update_users){
31
32             $action['result'] = 'success';
33             $action['text'] = 'User has been confirmed. Thank-You
34
35         }else{
36
37             $action['result'] = 'error';
38             $action['text'] = 'The user could not be updated Reas
39
40         }
41
42     }else{
43
44         $action['result'] = 'error';
45         $action['text'] = 'The key and email is not in our databa
46
47     }
48
49 }

```

Most of this should look very familiar; so I'm going to skip ahead and check if the key is in the database section.

Again, we use `mysql_query()` to get any rows in the database where the email and key are equal to the keys provided by the users email.

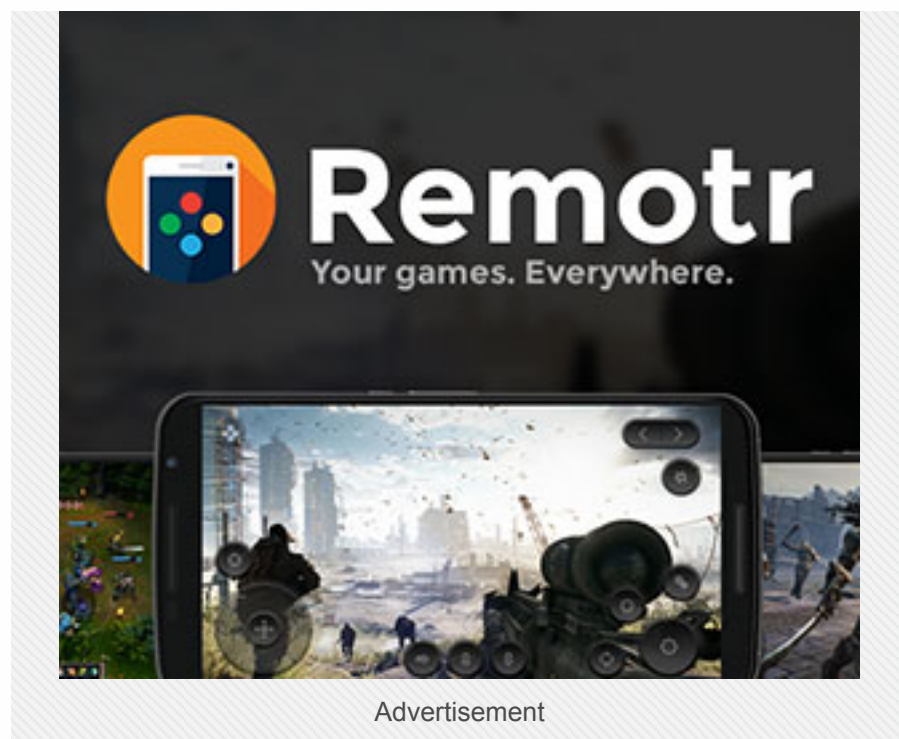
We use `mysql_num_rows()` to check if the number of rows returned is greater than 0.

If the email and key are in the database we grab all the information from the database using `mysql_fetch_assoc()`.

Now that the user has confirmed his account, we need to update the database and set the active row to 1.

We use `mysql_query()` again, but instead of `INSERT` we use `UPDATE` to update the active row to 1 where the user ID is the same as our current users ID.

To clean everything up we use `mysql_query()` and `DELETE` to remove the confirmation row from the database. This makes sure that the user can't come back to this page and reconfirm. It also keeps the database nice and clean.



Conclusion

We've covered many different areas in this tutorial. We downloaded and included a 3rd party script to deal with sending the emails, implemented simple form validation as well as created a super simple template system to style our emails. If you're new to MySQL we've touched on the three most common functions in MySQL so you should have no problem completing some more advanced tutorials.

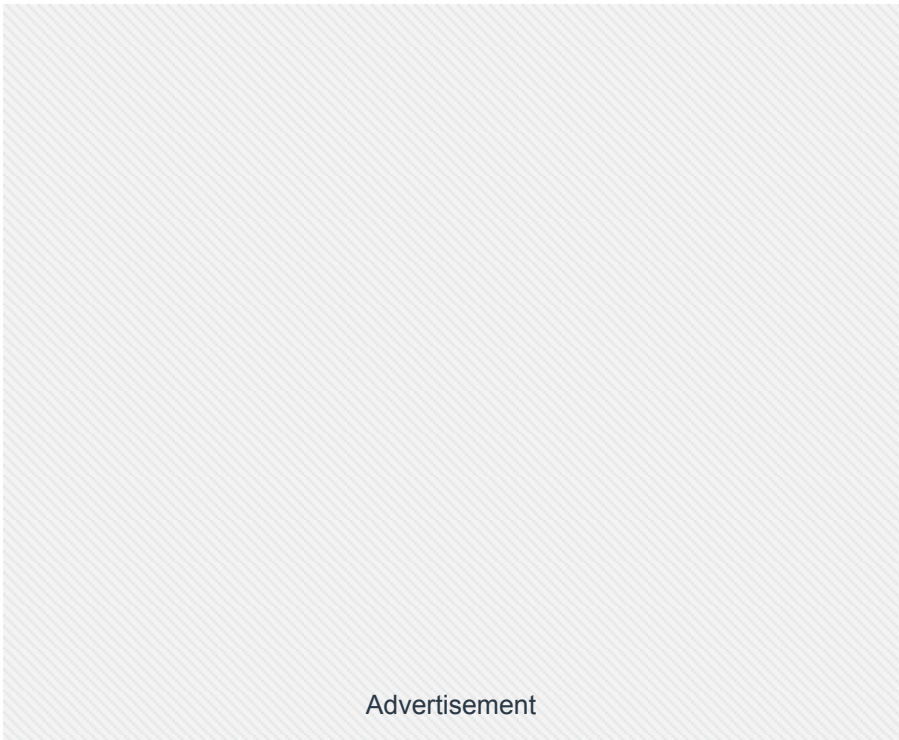
Final Notes

- I've used Swift Mailer as our email deployment script which can be

downloaded here: <http://swiftmailer.org/>

- I've also used button styles provided by Zurb. Be sure to check them out and give them some love. http://www.zurb.com/blog_uploads/0000/0485/buttons-02.html

Thanks for reading and be sure to visit me on [Twitter](#) if you have any questions!



Difficulty:
Beginner

Length:
Short

Categories:

PHP

Web Development

Signup Today

Thanks for signing up. P


Username:

Translations:

Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

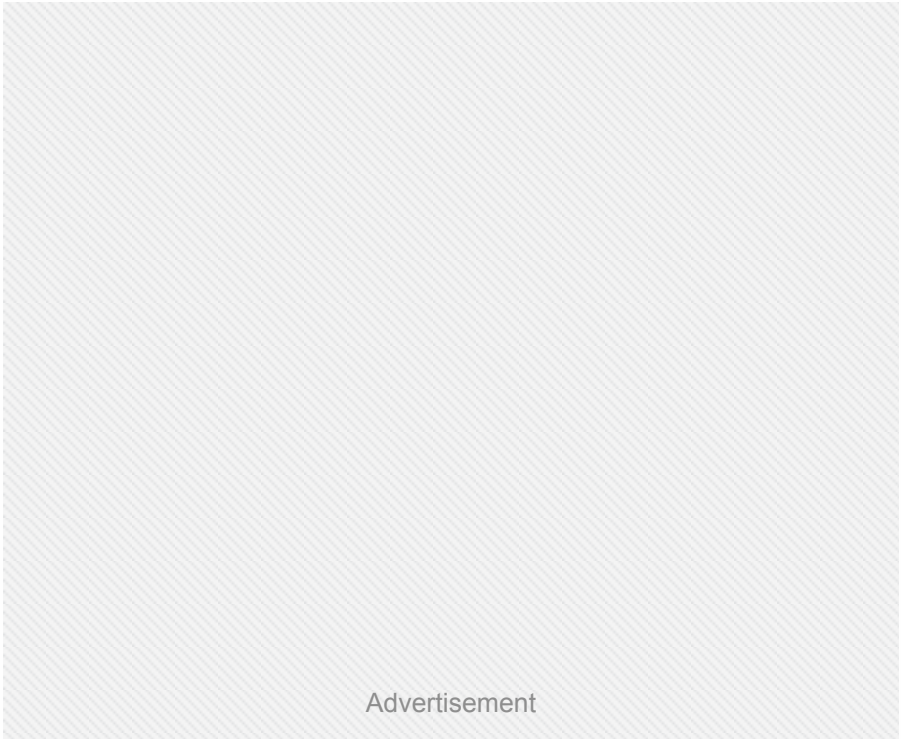
Translate this post

Powered by  native

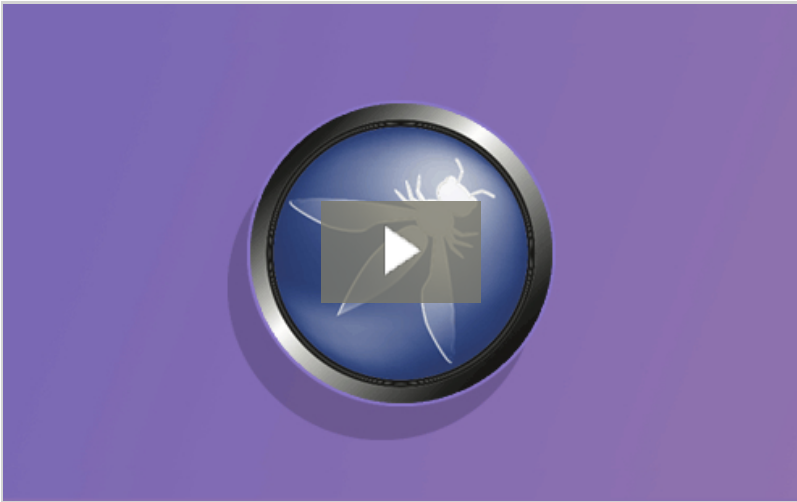
Download Attachment 



Matt Vickers is a coder at heart and a designer by necessity. If it involves coding, he'd probably be into it! He's currently employed at [Vantage Studios Inc.](#) in Winnipeg, MB as a Media Developer. You can also check out his personal site, [Envex Labs](#), where he usually has something interesting to say!



Suggested Tuts+ Course



PHP OWASP Security

\$5

Related Tutorials



Building a Note-Taking Software-as-a-Service Using ASP.NET MVC 5, Stripe, and Azure
[Code](#)

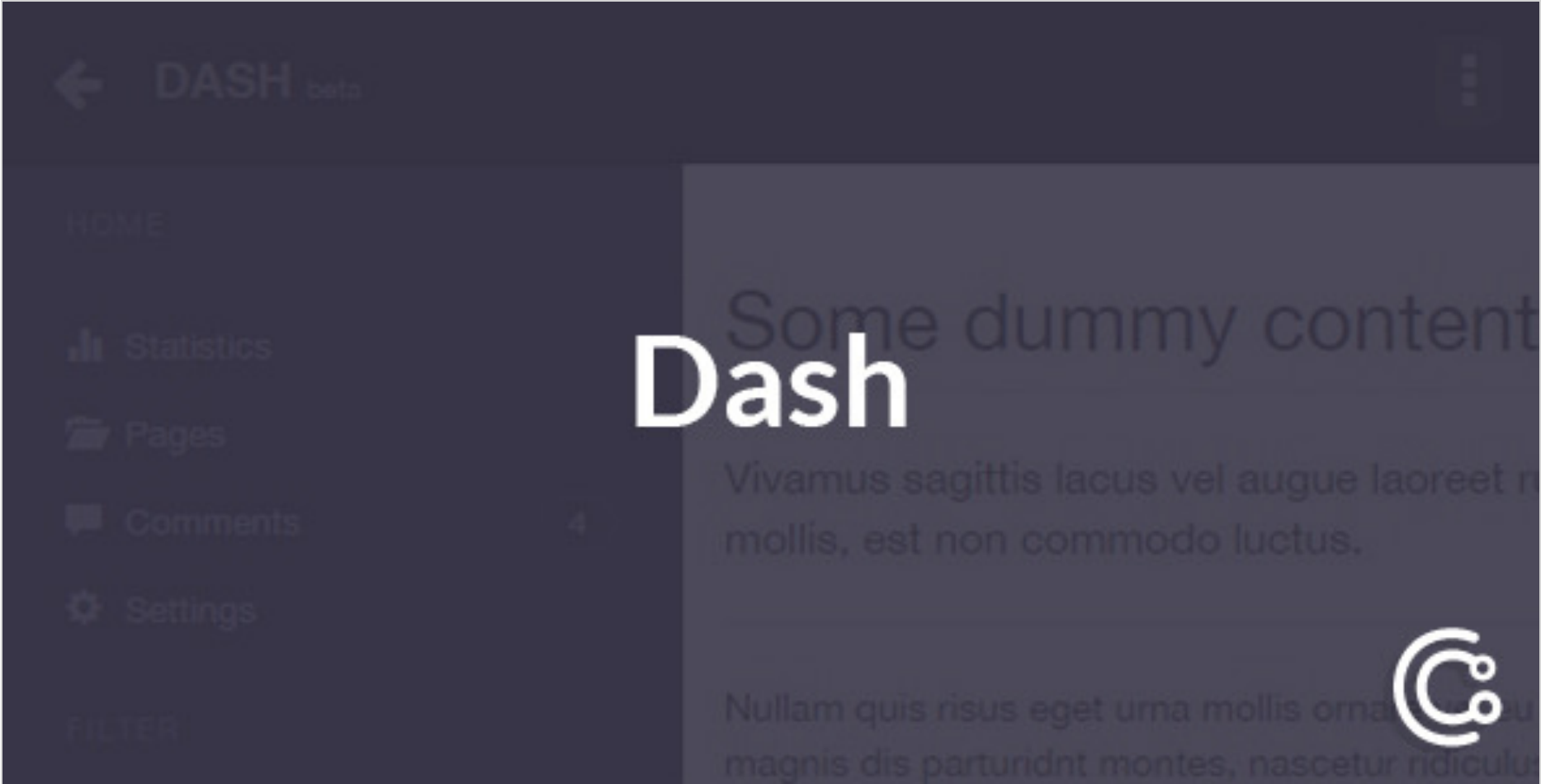


Build a Custom WordPress User Flow — Part 3: Password Reset
[Code](#)



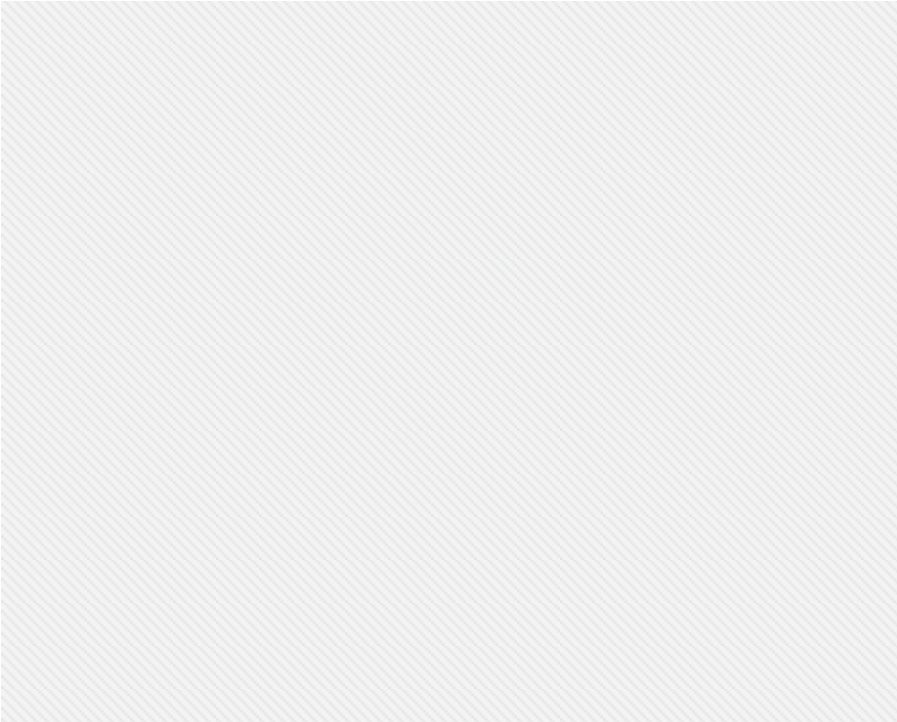
Build a Custom WordPress User Flow — Part 2: New User Registration

Envato Market Item



What Would You Like to Learn?

[Suggest an idea](#) to the content editorial team at Tuts+.



tuts+

Teaching skills to millions worldwide.

20,515 Tutorials **625** Video Courses

Follow Us



Help and Support

[FAQ](#)
[Terms of Use](#)
[Contact Support](#)
[About Tuts+](#)
[Advertise](#)
[Teach at Tuts+](#)
[Translate for Tuts+](#)
[Meetups](#)

Email Newsletters

Get Tuts+ updates, news, surveys & offers.

Subscribe

[Privacy Policy](#)

Custom digital services like logo design, WordPress installation, video production and more.

Check out Envato Studio



New Services!

A square graphic with a dark blue background and a lighter blue diagonal stripe. The text "Pre-Coded PHP" is written in white, sans-serif font.

Pre-Coded PHP

Build anything from social networks to file upload systems. Build faster with pre-coded PHP scripts.

[Browse PHP on CodeCanyon](#)