

# INTRODUCTION

This **codeigniter tutorial** will provide you with all the basic information you need to get up and running with **codeigniter** within 40 minutes. This **codeigniter tutorial** will provide step by step instructions from installation, using models to interact with the database, manipulating this information and finally displaying it to the user.

While this **codeigniter** tutorial will not cover the benefits of the MVC framework in detail you will understand what MVC is and have access to some useful resources to learn more about why you should adopt MVC methodologies.

## WHAT IS CODEIGNITER?

Codeigniter is a PHP MVC framework that aims to make common operations easy while enforcing structured code making it easier to debug, scale and develop as a team.

If you compare **codeigniter** to another framework such as Zend it is not as restrictive in terms of enforcing the MVC structure. This could be seen as a negative or positive, it all depends what you, as the coder wants from **codeigniter** and how you use it.

There are many PHP frameworks available and you should do some research to determine what will work best for you. Here are the main reasons why **codeigniter** is my PHP MVC framework of choice.

- **Lightweight** – Compared to other PHP frameworks **CodeIgniter** is very lightweight, meaning it is easier to create fast and responsive code
- **No Installation** – Unlike some PHP frameworks there is no installation required, you just include the **CodeIgniter** library and off you go
- **Large Community** – **CodeIgniter** has a huge active community so you know there are plenty of people willing to help if you get stuck on a particularly difficult problem
- **Superb Documentation** – **CodeIgniter** provides excellent documentation with examples
- **Flexibility** – **CodeIgniter** allows you to easily extend and expand on its base architecture making it easier to break the mold and produce some really unique web applications

Now you know why I use **CodeIgniter**, lets get started with this **CodeIgniter tutorial** so that you can make use of all these benefits too.

## CODEIGNITER SET-UP

As the introduction stated, there is no installation as such with codeigniter, we just need to get hold of the latest version of the library and place it within our directory structure then change a few configuration files to meet our specific needs.

## DOWNLOAD CODEIGNITER

To start off this **codeigniter tutorial**, head over to <http://ellislab.com/codeigniter> and download the latest version to somewhere you can easily access it.

## CREATE DIRECTORY STRUCTURE FOR CODEIGNITER FILES

Once the download has completed, extract the archive to your desktop for easy access. You should now have a folder that looks like this:

Name	Date modified	Type	Size
application	08/10/2012 07:18	File folder	
system	08/10/2012 07:18	File folder	
user_guide	08/10/2012 07:27	File folder	
index.php	08/10/2012 07:18	PHP File	7 KB
license.txt	08/10/2012 07:18	Text Document	3 KB

Now we need to place these files within our project directory. Within your project directory you should have a "public\_html" or "www" folder which is the web root of the project. Within this directory you should create a "codeigniter" directory where we are going to place the codeigniter files. This is shown below:

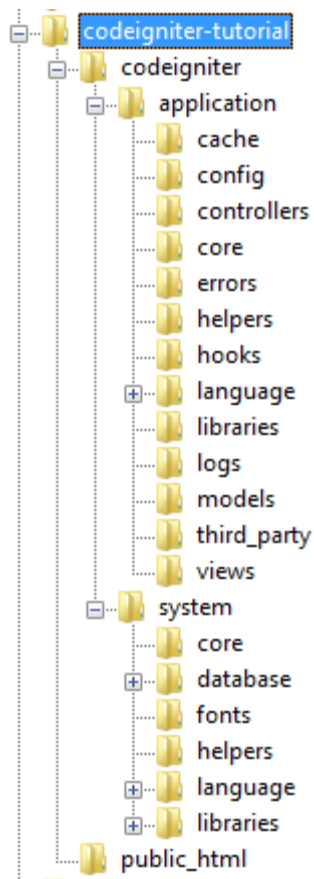
Name	Date modified	Type	Size
codeigniter	28/04/2013 13:59	File folder	
public_html	28/04/2013 13:51	File folder	

We now need to locate the codeigniter system and application directories, shown below:

application	08/10/2012 07:18	File folder	
system	08/10/2012 07:18	File folder	
user_guide			
index.php			7 KB
license.txt	08/10/2012 07:18	Text Document	3 KB

Date created: 28/04/2013 13:54  
Size: 47.7 KB  
Folders: cache, config, controllers, core, errors, helpers, ...  
Files: .htaccess, index.html

Copy both of these folders and paste them into the "/codeigniter" directory of your project folder. You should now have a directory structure like the following:



## EDIT CODEIGNITER CONFIGURATION

Now that we have the CodeIgniter files in the correct place, we need to tell codeigniter where it is going to find its files. Within the extracted codeigniter folder on your desktop, there will be a `index.php` file, shown below:

Name	Date modified	Type	Size
application	08/10/2012 07:18	File folder	
system	08/10/2012 07:18	File folder	
user_guide	08/10/2012 07:27	File folder	
index.php	08/10/2012 07:18	PHP File	7 KB
license.txt	08/10/2012 07:18	Text Document	3 KB

Copy this file and place it within the web root (`public_html`, `www`) of your project and open it in your favorite text editor. From line 49 of this file, you should see the following:

```

/*
 *
 * -----
 *  SYSTEM FOLDER NAME
 * -----
 *
 * This variable must contain the name of your "system" folder.
 * Include the path if the folder is not in the same directory
 * as this file.
 *
 */
    $system_path = 'system';

/*
 *
 * -----
 *  APPLICATION FOLDER NAME
 * -----
 *

```

```

* If you want this front controller to use a different "application"
* folder then the default one you can set its name here. The folder
* can also be renamed or relocated anywhere on your server. If
* you do, use a full server path. For more info please see the user guide:
* http://codeigniter.com/user_guide/general/managing_apps.html
*
* NO TRAILING SLASH!
*
*/
    $application_folder = 'application';

```

\$system\_path and \$application\_folder is where codeigniter will look for the respective folders, you need to update this file so that it is pointing to where you moved the application and system folder. Change this file to read the following:

```

/*
*-----
* SYSTEM FOLDER NAME
*-----
*
* This variable must contain the name of your "system" folder.
* Include the path if the folder is not in the same directory
* as this file.
*
*/
    $system_path = '../codeigniter/system';

/*
*-----
* APPLICATION FOLDER NAME
*-----
*
* If you want this front controller to use a different "application"
* folder then the default one you can set its name here. The folder
* can also be renamed or relocated anywhere on your server. If
* you do, use a full server path. For more info please see the user guide:
* http://codeigniter.com/user_guide/general/managing_apps.html
*
* NO TRAILING SLASH!
*
*/
    $application_folder = '../codeigniter/application';

```

To remove index.php from the URL we also need to create a .htaccess file with the following contents:

```

RewriteEngine on
RewriteCond $1 !^(index\.php|images|robots\.txt)
RewriteRule ^(.*)$ /index.php/$1 [L]

```

Save the .htaccess file within the web root (public\_html, www) with the index.php file.

Finally we need to tell **codeigniter** to auto-load the database library so we don't have to ask for it every time we want to do a database query. Open the file /codeigniter/application/config/autoload.php. From line 43 you should see the following:

```

/*
*-----
* Auto-load Libraries
*-----
*
* These are the classes located in the system/libraries folder
* or in your application/libraries folder.
*
* Prototype:
*
*     $autoload['libraries'] = array('database', 'session', 'xmlrpc');
*/
$autoload['libraries'] = array();

```

All you need to do as add **database** to the array as a string, change the above code to match the code

below:

```
/*
-----
Auto-load Libraries
-----
These are the classes located in the system/libraries folder
or in your application/libraries folder.
Prototype:
    $autoload['libraries'] = array('database', 'session', 'xmlrpc');
*/
$autoload['libraries'] = array('database');
```

Now the database library will always be loaded and available.

That is the first bit done! Providing you have a web server running, if you now go to your web root with a browser you should be greeted with the following message:

## Welcome to CodeIgniter!

The page you are looking at is being generated dynamically by CodeIgniter.

If you would like to edit this page you'll find it located at:

application/views/welcome\_message.php

The corresponding controller for this page is found at:

application/controllers/welcome.php

If you are exploring CodeIgniter for the very first time, you should start by reading the [User Guide](#).

Page rendered in 0.0596 seconds

It is worth mentioning that we have moved the codeigniter files to outside the web root for security reasons, ensuring nobody will be able to access or view your system code.

## MODELS

MVC stands for Model, View and Controller. As previously mentioned, this article will not go into a great amount of detail regarding MVC, just know that MVC is a way of segregating your code to make development easier and to make your code more scalable.

- **Model** – A model is a representation of a database row. Models should be the only part of your application that interacts directly with the database
- **View** – A view is the user facing HTML and server side code. There should be no business logic or database interactions within a view. Views are dumb and should just render and display data to the user
- **Controller** – A controller “controls” all the data that is either requested by the user or data that is

retrieved from the database via a model. All business logic should be within the controllers

If you would like to go beyond this **codeigniter tutorial** and know more about MVC, please find more information and resources [here](#) and [here](#).

Before we create our first model, we need to create a database. On your web server, create a mySQL database called tutorial and import the following SQL:

```
--
-- Table structure for table `user`
--

CREATE TABLE IF NOT EXISTS `user` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `username` varchar(128) NOT NULL,
  `password` varchar(128) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

INSERT INTO `user` (`id`, `username`, `password`) VALUES (NULL, 'Gandalf', 'iamwhite!'),
(NULL, 'Bilbo', 'bagginses!'), (NULL, 'Gollum', 'precious'), (NULL, 'arwen', 'tolodannangalad');
```

The above SQL will create a very simple user table that will represent a user. Before we can create a model, we need to tell **codeigniter** which database to use and how to access it.

In your text editor, navigate to /codeigniter/application/config/database.php and from line 51 you should see the following code:

```
$db['default']['hostname'] = 'localhost';
$db['default']['username'] = '';
$db['default']['password'] = '';
$db['default']['database'] = '';
$db['default']['dbdriver'] = 'mysql';
$db['default']['dbprefix'] = '';
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = '';
$db['default']['char_set'] = 'utf8';
$db['default']['dbcollat'] = 'utf8_general_ci';
$db['default']['swap_pre'] = '';
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
```

The only lines we need to change are the username, password and database. The database should read tutorial as this is the name of the database you just created. And the username and password will be the ones you use to access your mySQL management tool, such as PHPMYAdmin. Change this code to look like the following. Replace "[PASSWORD]" and "[USERNAME]" with your own.

```
$db['default']['hostname'] = 'localhost';
$db['default']['username'] = '[USERNAME]';
$db['default']['password'] = '[PASSWORD]';
$db['default']['database'] = 'tutorial';
$db['default']['dbdriver'] = 'mysql';
$db['default']['dbprefix'] = '';
$db['default']['pconnect'] = TRUE;
$db['default']['db_debug'] = TRUE;
$db['default']['cache_on'] = FALSE;
$db['default']['cachedir'] = '';
$db['default']['char_set'] = 'utf8';
$db['default']['dbcollat'] = 'utf8_general_ci';
$db['default']['swap_pre'] = '';
$db['default']['autoinit'] = TRUE;
$db['default']['stricton'] = FALSE;
```

Now that CodeIgniter knows how to access our database and user table, we can now go ahead and create a model which will represent a user.

```

/**
 * Includes the User_Model class as well as the required sub-classes
 * @package codeigniter.application.models
 */

/**
 * User_Model extends codeigniters base CI_Model to inherit all codeigniter magic!
 * @author Leon Revill
 * @package codeigniter.application.models
 */
class User_Model extends CI_Model
{
    /**
     * A private variable to represent each column in the database
     */
    private $_id;
    private $_username;
    private $_password;

    function __construct()
    {
        parent::__construct();
    }

    /**
     * SET's & GET's
     * Set's and get's allow you to retrieve or set a private variable on an object
     */

    /**
     * ID
     */

    /**
     * @return int [$this->_id] Return this objects ID
     */
    public function getId()
    {
        return $this->_id;
    }

    /**
     * @param int Integer to set this objects ID to
     */
    public function setId($value)
    {
        $this->_id = $value;
    }

    /**
     * USERNAME
     */

    /**
     * @return string [$this->_username] Return this objects username
     */
    public function getUsername()
    {
        return $this->_username;
    }

    /**
     * @param string String to set this objects username to
     */
    public function setUsername($value)
    {
        $this->_username = $value;
    }

    /**
     * PASSWORD
     */

    /**
     * @return string [$this->_password] Return this objects password
     */
    public function getPassword()
    {
        return $this->_password;
    }
}

```

```

/**
 * @param string String to set this objects password to
 */
public function setPassword($value)
{
    $this->_password = $value;
}

/**
 * Class Methods
 */

/**
 * Commit method, this will commit the entire object to the database
 */
public function commit()
{
    $data = array(
        'username' => $this->_username,
        'password' => $this->_password
    );

    if ($this->_id > 0) {
        //We have an ID so we need to update this object because it is not new
        if ($this->db->update("user", $data, array("id" => $this->_id))) {
            return true;
        }
    } else {
        //We dont have an ID meaning it is new and not yet in the database so we
        need to do an insert
        if ($this->db->insert("user", $data)) {
            //Now we can get the ID and update the newly created object
            $this->_id = $this->db->insert_id();
            return true;
        }
    }
    return false;
}
}

```

Save the above code as a file called user\_model.php within the /codeigniter/application/models directory.

And that is all there is to a model! The above code is well commented and should make sense but lets go over it section by section. Lets start with the private variables.

## PRIVATE VARIABLES

```

private $_id;
private $_username;
private $_password;

```

These variables simply provide a place for us to store the data from their matching column in the database, you can call them anything you like but it makes sense to call them the same thing as the database column to avoid confusion.

## SETS & GETS

```

public function getUsername()
{
    return $this->_id;
}

public function setUsername($value)
{
    $this->_username = $value;
}

```

You can look at sets and gets as an interface to the object that the model is representing. We have private



variables as opposed to public so that they cannot be modified outside the models scope. This allows us to use set and get methods to control what can be set and what can be got from these private variables. For example, you could always cast to integer to ensure that a private variable would always be so, something you could not do if you were exposing the local variables to the outside world. Another example is if one of these private variables was a date, you could ensure that a formatted date was always provided via the get method for this variable, saving formatting the value every time you needed it.

## COMMITTING THE OBJECT

```
public function commit()
{
    $data = array(
        'username' => $this->_username,
        'password' => $this->_password
    );

    if ($this->_id > 0) {
        //We have an ID so we need to update this object because it is not new
        if ($this->db->update("user", $data, array("id" => $this->_id))) {
            return true;
        }
    } else {
        //We dont have an ID meaning it is new and not yet in the database so we
        need to do an insert
        if ($this->db->insert("user", $data)) {
            //Now we can get the ID and update the newly created object
            $this->_id = $this->db->insert_id();
            return true;
        }
    }
    return false;
}
```

Obviously at some point we are going to want to save the object's data back to the database. We create a method called commit which simply does that. This method checks to see if it has an ID, if it does then an entry for this object already exists within the database. If the object already exists in the database then it needs to perform an update. If the object doesn't already have an ID then it does not currently have a database entry. If there is no entry it needs to create a new one and then, using the MySQL function insert\_id (which returns the ID of the last inserted row) to then update the object so it will then only perform updates and not insert a new entry every time a commit is performed.

## MODEL FACTORY

We are going to create a very simple factory which is basically a collection of methods within a library that we can reuse to create objects with our model. This separates out methods such as getUser, createUser, etc from within the model, ensuring that the model can strictly represent the database row. Take a look at the following code:

```
if ( ! defined('BASEPATH')) exit('No direct script access allowed');

class UserFactory {
    private $_ci;

    function __construct()
    {
        //When the class is constructed get an instance of codeigniter so we can access it
        locally $this->_ci =& get_instance();
        //Include the user_model so we can use it
        $this->_ci->load->model("user_model");
    }

    public function getUser($id = 0) {
        //Are we getting an individual user or are we getting them all
        if ($id > 0) {
            //Getting an individual user
            $query = $this->_ci->db->get_where("user", array("id" => $id));
        }
    }
}
```

```

        //Check if any results were returned
        if ($query->num_rows() > 0) {
            //Pass the data to our local function to create an object for us and
return this new object
            return $this->createObjectFromData($query->row());
        }
        return false;
    } else {
        //Getting all the users
        $query = $this->_ci->db->select("*")->from("user")->get();
        //Check if any results were returned
        if ($query->num_rows() > 0) {
            //Create an array to store users
            $users = array();
            //Loop through each row returned from the query
            foreach ($query->result() as $row) {
                //Pass the row data to our local function which creates a new
user object with the data provided and add it to the users array
                $users[] = $this->createObjectFromData($row);
            }
            //Return the users array
            return $users;
        }
        return false;
    }
}

public function createObjectFromData($row) {
    //Create a new user_model object
    $user = new User_Model();
    //Set the ID on the user model
    $user->setId($row->id);
    //Set the username on the user model
    $user->setUsername($row->username);
    //Set the password on the user model
    $user->setPassword($row->password);
    //Return the new user object
    return $user;
}
}

```

Once more the above code is well commented and should be easy to follow.

The first thing to notice is as follows:

```

    private $_ci;

    function __construct()
    {
        //When the class is constructed get an instance of codeigniter so we can access it
        $this->_ci =& get_instance();
        //Include the user_model so we can use it
        $this->_ci->load->model("user_model");
    }

```

Here we are creating a private variable called `$_ci`, short for **codeigniter** which is where we are going to store an instance of **codeigniter** so we have access to all its resources. The reason **codeigniter** is like this is because you may want to create libraries that do not require any of the **codeigniter** functionality therefore why include the extra bulk? **Codeigniter** is very lean remember! Learn more about creating libraries in **codeigniter** [here](#).

Then, within the `__construct()` of the library (which is called when the class is instantiated) we grab this **codeigniter** instance and store it in this variable. Now we can access all of codeigniter's magic, for example: `$this->_ci->db->select("*")->from("user")->get();`

We have two methods within our factory, the main method is get user:

```

public function getUser($id = 0) {
    //Are we getting an individual user or are we getting them all
    if ($id > 0) {

```

```

        //Getting an individual user
        $query = $this->_ci->db->get_where("user", array("id" => $id));
        //Check if any results were returned
        if ($query->num_rows() > 0) {
            //Pass the data to our local function to create an object for us and
return this new object
            return $this->createObjectFromData($query->row());
        }
        return false;
    } else {
        //Getting all the users
        $query = $this->_ci->db->select("*")->from("user")->get();
        //Check if any results were returned
        if ($query->num_rows() > 0) {
            //Create an array to store users
            $users = array();
            //Loop through each row returned from the query
            foreach ($query->result() as $row) {
                //Pass the row data to our local function which creates a new
user object with the data provided and add it to the users array
                $users[] = $this->createObjectFromData($row);
            }
            //Return the users array
            return $users;
        }
        return false;
    }
}

```

This method allows us to pass a parameter of \$id so we can retrieve a specific user object, or if we do not provide an ID it will get all of the user objects for us. Lets walk through the more complicated side of this method, getting all users:

```

//Getting all the users
$query = $this->_ci->db->select("*")->from("user")->get();
//Check if any results were returned
if ($query->num_rows() > 0) {
    //Create an array to store users
    $users = array();
    //Loop through each row returned from the query
    foreach ($query->result() as $row) {
        //Pass the row data to our local function which creates a new user object
with the data provided and add it to the users array
        $users[] = $this->createObjectFromData($row);
    }
    //Return the users array
    return $users;
}
return false;

```

Firstly we query the user table and return all rows (\$query = \$this->\_ci->db->select("\*")->from("user")->get();). We then check to see if there are any results using the \$query->num\_rows() method which will return the number of results. If the number is 0 then we return false and end execution. If the number of rows is greater than 0 then we create an array called users and then loop through every row returned in the query: foreach (\$query->result() as \$row) {.

For each returned row we then call our local method:

```

public function createObjectFromData($row) {
    //Create a new user_model object
    $user = new User_Model();
    //Set the ID on the user model
    $user->setId($row->id);
    //Set the username on the user model
    $user->setUsername($row->username);
    //Set the password on the user model
    $user->setPassword($row->password);
    //Return the new user object
    return $user;
}

```

This method takes the row data and creates a new user object using the `user_model` class we created earlier. It then sets the appropriate values from the database row onto this object and then returns the object to the calling method.

Once the calling method receives the new object it adds it to the `$users` array. Once it has iterated through each of the returned rows it returns the `$users` array. Simple!

Now we have a way of representing a database row using a model and a way to get a set of objects represented by the **codeigniter** model using a library, lets move on to something you can see!

**Please note that the user factory needs to be saved within the `/codeigniter/application/libraries` directory.**

## CONTROLLERS

A controller decides what to display and how to display it. Within the `/codeigniter/application/controllers` create a file called `users`, which looks like this:

```
if ( ! defined('BASEPATH')) exit('No direct script access allowed');

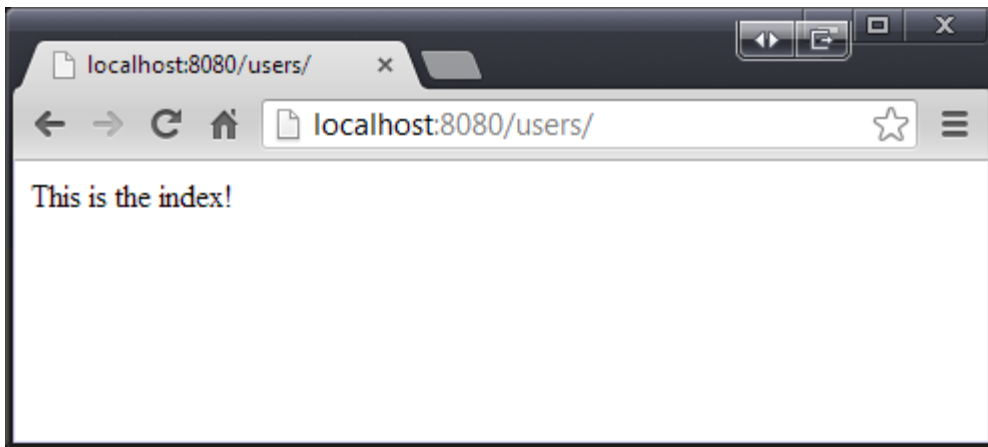
class Users extends CI_Controller {

    public function index()
    {
        echo "This is the index!";
    }

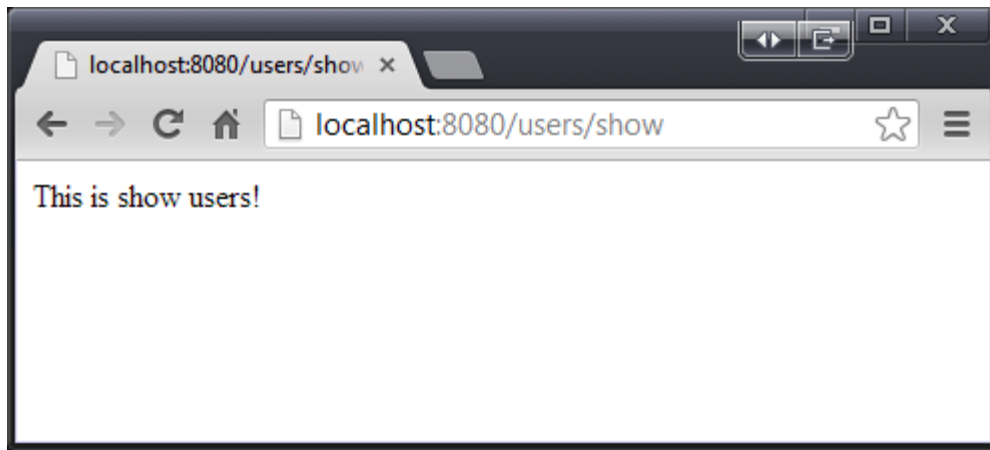
    public function show($userId = 0)
    {
        $userId = (int)$userId;
        echo "This is show users!";
        if ($userId > 0) {
            echo "User ID: {$userId}";
        }
    }

}
```

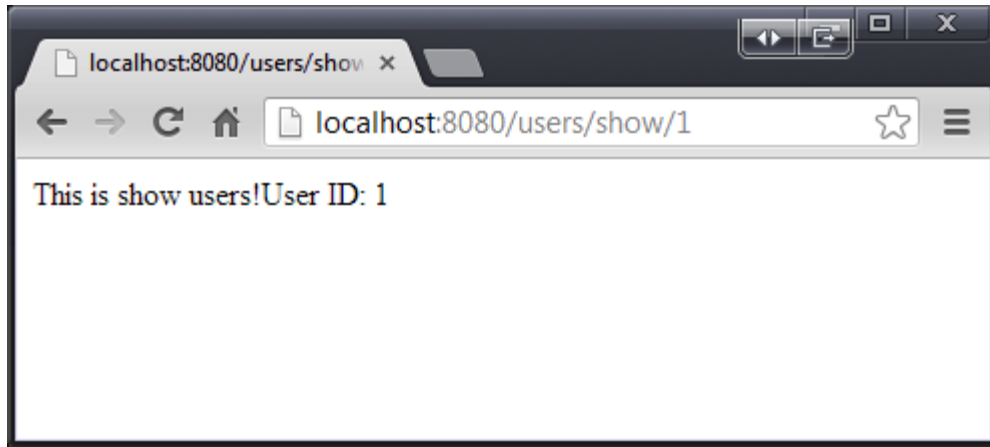
This is not the final version of our controller, we are just going to use this to demonstrate and help you understand how the controllers in **codeigniter** work. If you navigate to your site, lets say the URL you are using is `http://localhost:8080/`, then navigate to: `http://localhost:8080/users` and you should see the following:



The `index` method within the controller is what the controller will fall back on if there has been no other method specified within the URL, so because we just went to `/users` the `index` method is called. Now, navigate to: `http://localhost:8080/users/show` and you should see the following:



Here you can see that by specifying show as the second URL section we are calling the show method within the controller. Now, navigate to: <http://localhost:8080/users/show/1> and see the following:



With the third section we have passed in a user ID which the show method picks up and displays on the screen. We can pass any number of parameters to a **codeigniter** controller method. Now you understand how **codeigniters** controllers work we can move on and make the users controller a little more practical.

Change the show method to look like below:

```
public function show($userId = 0)
{
    //Always ensure an integer
    $userId = (int)$userId;
    //Load the user factory
    $this->load->library("UserFactory");
    //Create a data array so we can pass information to the view
    $data = array(
        "users" => $this->userfactory->getUser($userId)
    );
    //Dump out the data array
    var_dump($data);
}
```

As always the code is well commented so it should be easy to follow. The main thing that this method is doing now, is loading the user factory so we can retrieve user objects using the library and model we created earlier. We store the results within an array which we will eventually pass to a view and then just for now we are dumping the results so we can see how they look. Go ahead and navigate to: <http://localhost:8080/users/show> and you should see the following:

```
array(1) { ["users"]=> array(4) { [0]=> object(User_Model)#22 (3) {
["_id":"User_Model":private]=> string(1) "1" ["_username":"User_Model":private]=> string(7)
"Gandalf" ["_password":"User_Model":private]=> string(9) "iamwhite!" } [1]=> object(User_Model)#23
(3) { ["_id":"User_Model":private]=> string(1) "2" ["_username":"User_Model":private]=> string(5)
"Bilbo" ["_password":"User_Model":private]=> string(10) "bagginses!" } [2]=> object(User_Model)#24
```

```
(3) { ["_id":"User_Model":private]=> string(1) "3" ["_username":"User_Model":private]=> string(6)
"Gollum" ["_password":"User_Model":private]=> string(8) "precious" } [3]=> object(User_Model)#25
(3) { ["_id":"User_Model":private]=> string(1) "4" ["_username":"User_Model":private]=> string(5)
"arwen" ["_password":"User_Model":private]=> string(15) "tolodannangalad" } } }
```

The above output is a dump of an array of user models which are presenting every user within our user table, if you navigate to: <http://localhost:8080/users/show/1> and view the following output:

```
array(1) { ["users"]=> object(User_Model)#19 (3) { ["_id":"User_Model":private]=> string(1) "1"
["_username":"User_Model":private]=> string(7) "Gandalf" ["_password":"User_Model":private]=>
string(9) "iamwhite!" } }
```

You can see that the results are just for the user with user ID 1 as this is what we have specified within the URL, try changing the user ID and see what happens.

Now you should have a pretty good understanding as to how we can request and retrieve data from the database in a complete object oriented manner. Lets move onto the final section and create some views!

## VIEWS

A view is simple how the data is presented to the user, typically consisting of the HTML and other client side code. We are going to create a view which will take the data array we created in our controller in the previous section and list out all the users it is given. Within `/codeingiter/application/views` create a file called `show_users.php` with the following code:

```
//Check to see if users could be found
if ($users !== FALSE) {

    //Create the HTML table header
    echo <<<HTML

    <table border='1'>
        <tr>
            <th>ID #</th>
            <th>Username</th>
            <th>Password</th>
        </tr>

HTML;

    //Do we have an array of users or just a single user object?
    if (is_array($users) && count($users)) {
        //Loop through all the users and create a row for each within the table
        foreach ($users as $user) {
            echo <<<HTML

                <tr>
                    <td>{$user->getId()}</td>
                    <td>{$user->getUsername()}</td>
                    <td>{$user->getPassword()}</td>
                </tr>

HTML;

            }
        } else {
            //Only a single user object so just create one row within the table
            echo <<<HTML

                <tr>
                    <td>{$users->getId()}</td>
                    <td>{$users->getUsername()}</td>
                    <td>{$users->getPassword()}</td>
                </tr>

HTML;

            }
        }
    }
    //Close the table HTML
    echo <<<HTML
    </table>

HTML;
```

```

    } else {
        //Now user could be found so display an error message to the user
        echo <<<HTML
            <p>A user could not be found with the specified user ID#, please try
again.</p>
HTML;
    }

```

Before we take a look at this code in any detail, we need to update our controller to load the view and provide it with the user data it needs. Change the show method within the users controller to be as follows:

```

public function show($userId = 0)
{
    //Always ensure an integer
    $userId = (int)$userId;
    //Load the user factory
    $this->load->library("UserFactory");
    //Create a data array so we can pass information to the view
    $data = array(
        "users" => $this->userfactory->getUser($userId)
    );
    //Load the view and pass the data to it
    $this->load->view("show_users", $data);
}

```

All we have changed is we are now using this **codeigniter** load functionality (`$this->load->view()`) to load the view we have created. The first parameter is the name of the file within the views directory we want to load, the second is an array of data we want to pass to the view. Note that any key within the `$data` array will become a variable within the view. So in this example to access the `$data["users"]` value, we simply use `$users` within the view.

Now we have the controller loading the view and we understand how to access the data we have passed to it from within the view, let's walk through the code.

Firstly we check to see if the `$users` variable is not false, if the factory cannot find a user with the specified ID then it will return false. We handle this and then display an error message to the user: "A user could not be found with the specified user ID#, please try again."

If the `$users` variable is not false and has some data we then need to check to see if it is a single user (an ID has been specified) or it is an array of multiple users. If it is multiple users we then loop through each user and create a row with their data within the HTML table.

if it is just a single user then we just create a single row with the user data. Simple, as, that!

## CONCLUSION

Now you have completed this **codeigniter tutorial** you should now have a good understanding of how **codeigniter** works. You should know how to set-up **codeigniter** including all the configuration, create models, views and controllers along with libraries and know how to load and use them successfully.