Authentication

tutorial #tutorial08

James L. Parry
B.C. Institute of Technology

Tutorial Goals

This tutorial is meant to give you some practice working with simple programmatic authentication, as described in lesson 10.

I have prepared a starter project for you to build on. It is a webapp, which doesn't do much useful other than provide a base for the tutorial.

Revisions & Notes

- Fixed some typos and slides out of order; added end result screenshot

Background

This webapp, for the "Top Secret Government Site", has three controllers and content pages, each intended to be accessible to a different audience.

As it comes, all three pages are visible to anyone.

We will add simple authentication, authorization and access control, to enforce the intended security.

Preparation

I have prepared a starter project.

Fork the github project, and clone it locally to work with, the same as you have done with the previous tutorials.

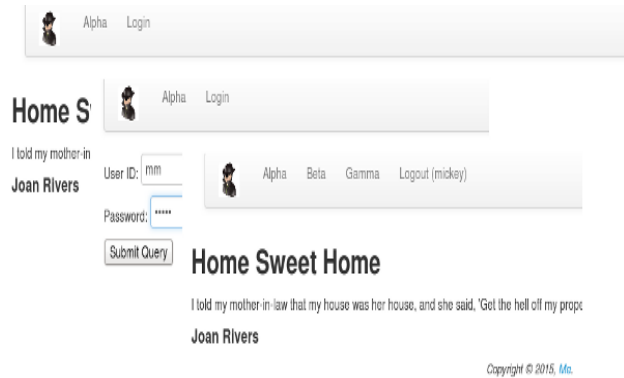When you run it, you should see the homepage to the right.

Home Sweet Home

I told my mother-in-law that my house was her house, and she said, 'Get the hell off my property.'

Joan Rivers

Copyright © 2015, Me.

The End Result

Once we are done, the webapp will have authentication, and it will only show those menu items that the logged in user is allowed to see, shown right.

What Needs Doing?

1. Setup our database
2. Add a login form
3. Add an authentication controller
4. Add authorization constraints
5. Add access control constraints
6. Tailor the output per the user's role

SETUP OUR DATABASE

Perform these tasks.

- Using PHPmyadmin, create a database for this project, called "secrets".
- Configure your webapp to autoload the 'database' library.
  ```
  $autoload['libraries'] =
  array('parser','database');
  ```
- Configure your database to reference the "secrets" database.
  ```
  $db['default']['database'] = 'secrets';
  ```

When run, your webapp should look no different than before. Most importantly, it should not break!

Configure Sessions

Create a `ci_sessions` table, using the `data/sessions.sql` script.

Configure the sessions by modifying `config/config`:

- enable database usage for your sessions
  `$config['sess_driver'] = 'database';`
  `$config['sess_save_path'] = 'ci_sessions';`

Enable Sessions

Enable the sessions by modifying `config/autoload`:

- add 'session' to the libraries autoload, so the facility is always available
  `$autoload['libraries'] = array('parser','database','session');`

When run, your webapp should still look no different than before. How do you know if your sessions are working?

Note: If you encounter an error to the effect that the CI_Session_Driver class cannot be found, that is likely due to a mismatch between the starter's configuration file and your system folder. The config assumes a new version ... download the official CodeIgniter3.0.0 and use its `system` folder to replace your `system3` one!

Test Your Sessions

Verify that the sessions are working properly:

- add a view parameter in `core/MY_Controller`, in the `render()` method just before calling the parser
  `$this->data['sessionid'] = session_id();`
- add a substitution variable to the footer in your views/_template
  `Copyright © 2015, Me. {sessionid}`

When run, your webapp should now show a session ID in the right of the footer. Reload the page to make sure it doesn't change.

Once you have confirmed that your sessions are loading, you can remove this testing code.

Add a Users Table

We also need a users table, to validate against.

Let's keep it simple, per the lesson. I have provided a `data/users.sql` script that you can use.

Using phpMyadmin, import that to your database and insert two records:

- id: dd, username: donald, password: duck, role: user
- id: mm, username: mickey, password: mouse, role: admin

The picture right shows using the "password" function. Don't! It is not the same as the password_hash() function we need to use later. Check the next slide before proceeding!

Column	Type	Function	Null	Value
id	varchar(10)			dd
name	varchar(20)			donald
password	varchar(64)	PASSWORD		duck
role	varchar(20)			user

Go

Fix The Users Table

Oops - there is a bit of a disconnect between phpMyadmin and the password hashing. I am not sure what the "password" function there does, but it is not the same as that referred to in the PHP password hashing documentation.

I wrote a super simple script to echo the intended passwords hashed, "duck" and "mouse". Here is the output:
$2y$10$Rga7t2AYnyhJs5kQIWcDEu5V/x12jOeB5fp1ZivYu5fGQIjODDdVK
$2y$10$MDR86Btj9Iwzc9rQ.HxP1O3SSrcS1G1kEwBqh67QwMVgS8/NmH7y.

Using phpMyadmin, change the password field values for dd and mm to the ones above, respectively :(

This is a one-time thing, and would normally be done through user maintenance by an admin.

Don't Forget a Model for the Users Table

`models/Users`:

```
class Users extends MY_Model {
  public function __construct() {
    parent::__construct('users', 'id');
  }
}
```

Note: You may have to copy `core/MY_Model` from another of your projects.

This model can be loaded in the constructor of any controller that needs it, but it might be a good idea to autoload it.

LOGIN FORM

We will need a login form as part of our authentication.

It doesn't have to be anything fancy, `views/login.php`, for instance something like:

```
<form name="login" method="post" action="/auth/submit">
  UserID: <input type="text" name="userid"></input><br/>
  Password: <input type="password" name="password"></input><br/>
  <input type="submit">Submit</input>
</form>
```

You are welcome to style it any way you like, just keep the field names and the action attribute.

AUTHENTICATION

Let's create a controller, `Auth`, to handle login and logout.

It can be similar to our other controllers, except that the page body it presents by default will be the login page from the previous step.

```
class Auth extends Application {
  function __construct() {
    parent::__construct();
    $this->load->helper('url');
  }
  function index() {
    $this->data['pagebody'] = 'login';
    $this->render();
  }
}
```

Test the Login Form

We can test our login form by typing the URL directly in the location field: "lab08.local/auth", where "lab08.local" is the virtual host domain name you have used.

You should get something like:

Handle the Login Form

Our controller needs to handle the login form submission, with a submit() method.

Without much in the way of error handling, our submit method can be pretty much what was presetned in the lesson:

```
  function submit() {
    $key = $_POST['userid'];
    $user = $this->users->get($key);
    if (password_verify($this->input->post('password'),$user->password)) {
      $this->session->set_userdata('userID',$key);
      $this->session->set_userdata('userName',$user->name);
      $this->session->set_userdata('userRole',$user->role);
    }
    redirect('/');
  }
```

Complete the Auth Controller
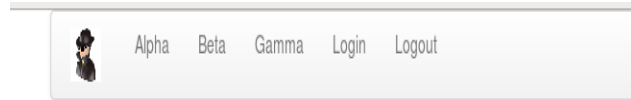
Add the logout() method to your Auth controller.

```
function logout() {
    $this->session->sess_destroy();
    redirect('/');
}
```

Make It Easy to Login

We had better add a link or two to the menu navbar to help with this.

It might be a bit cheesy for now, but we can simply add an additional two entries to our menu choices in config/config:

```
    ...
    array('name' => "Login", 'link' =>
'/auth'),
    array('name' => "Logout", 'link' => '/auth
/logout'),
```

Alpha   Beta   Gamma   Login   Logout

Home Sweet Home

I told my mother-in-law that my house was her house, and she said, 'Get the hell off my property.'

Joan Rivers

Copyright © 2015, Me.

AUTHORIZATION

We have two logged in user roles: "user" to see the alpha and beta pages, and "admin" to see all three content pages beyong the homepage.

For convenience, let's define those in config/constants...

```
define('ROLE_USER','user');
define('ROLE_ADMIN','admin');
```

ACCESS CONTROL

Access control will be enforced programmatically, in our base controller, just like in the lesson. We can add a restrict method to core/MY_Controller, pretty much as shown in the lesson...

```
  function restrict($roleNeeded = null) {
    $userRole = $this->session->userdata('userRole');
    if ($roleNeeded != null) {
      if (is_array($roleNeeded)) {
        if (!in_array($userRole, $roleNeeded)) {
          redirect("/");
          return;
        }
      } else if ($userRole != $roleNeeded) {
        redirect("/");
        return;
      }
    }
  }
}
```

Restrict Access to the Gamma Page

We can now add a rule to block the Gamma page from all but admins, by adding a line to the Gamma constructor:

```
class Gamma extends Application {
  function __construct() {
    parent::__construct();
    $this->restrict(ROLE_ADMIN);
  }
...
```

Try it! Login as "mm" with the password "mouse", and you should see the gamma page. Logged out, or logged in as "dd", you should not.

Repeat For the Beta Page

We can now add a rule to block the Beta page from all but registered users, by adding a line to the its constructor:

```
class Beta extends Application {
  function __construct() {
    parent::__construct();
    $this->restrict(array(ROLE_USER,ROLE_ADMIN));
  } ...
```

Note that the restriction allows either a user or an admin.

TAILORED MENU

Rather than have the same menu presented, whether logged in or not, it makes sense to present a menu with only those pages that the current user has access to. I suggest doing this in the base controller, perhaps as a "makemenu" method to build the array of menu choices. It would also make sense to show the name of the currently logged in user, if appropriate.

The gist is as follows, but I will leave the implementation to the reader.

```
$this->data['menubar'] = $this->makemenu();
...
function makemenu() {
//get role & name from session
// make array, with menu choice for alpha
// if not logged in, add menu choice to login
// if user, add menu choice for beta and logout
// if admin, add menu choices for beta, gamma and logout
// return the choices array
}
```

Congratulations!

You have completed tutorial #tutorial08: Authentication

If you would take a minute to provide some feedback, we would appreciate it!

The next activity in sequence is: lab08 Authentication 2015.04.10 17:30

You can use your browser's back button to return to the page you were on before starting this activity, or you can jump directly to the course homepage, organizer, or reference page.