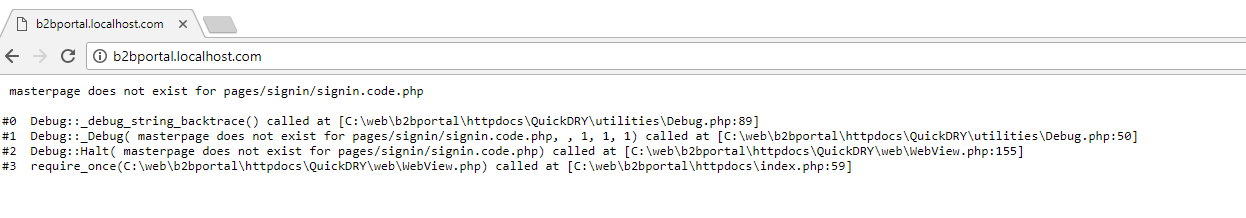
Single Sign-On With Personify

After you complete the walk thru in Getting Started With QuickDRY you end up with this



You will need to open index.php and go to about line 24

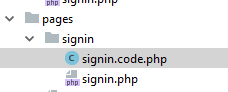


As you can see in the error the site is trying to load signin.code.php and that is because we told it to go to that page by default here. Default\_page is the page a user goes to when they aren’t logged in, default\_user\_page is where they go by default when they are logged in.

And we’re securing MASTERPAGE\_DEFAULT which means you must be logged in (and have proper permissions) to view any page with that master page.

Since we’re building a business to business portal, we’ll go ahead and leave this since everything this site will offer will require a login.

The next step then is to define the sign in page.

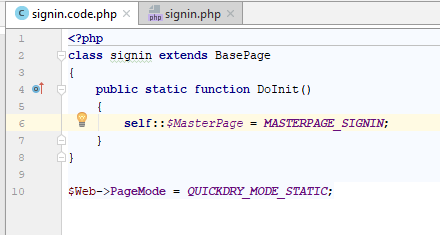


Under pages we create a signin folder and then signin.code.php and signin.php

The names must all match including case. Especially on Linux systems.

# Initial Code Behind (The Controller)

The minimum code required for a page to render is:



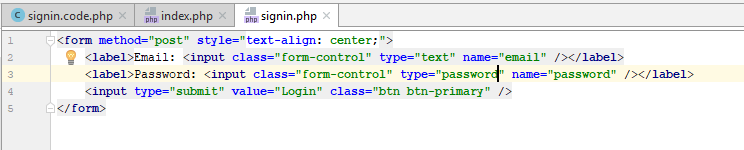
DoInit is called before permissions are checked. This function should generally just set the master page and possibly check some HTTP Request parameters. Any business logic in this function will be executed even if the user is ultimately denied access to the page.

Because we are using the static functions, we are setting the Web PageMode to QUICKDRY\_MODE\_STATIC

There is also QUICKDRY\_MODE\_INSTANCE but that will be covered elsewhere. The typical usage is using statically defined classes to avoid confusing PHP Storm when working on the view.

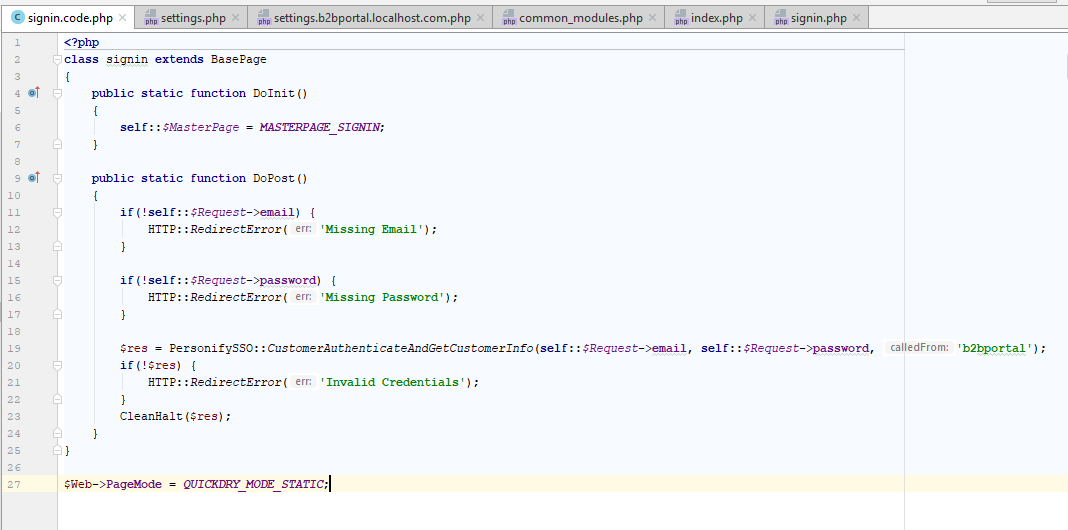
When you refresh the page, it will now load without error.

# The View



All we’re doing on this page is setting up a form for the user to sign in. Personify takes an email address and password to login with. Note that Bootstrap 3 is included in the standard masterpages along with font-awesome, jQuery and Google Charts.

# Handling the POST Request



Since we are using a static Page Model, we use the DoPost method to handle the POST request from the form.  
  
self::$Request gives you access to any GET or POST values with a precedence for POST, then GET.

The global PHP variable $\_REQUEST includes COOKIE values which is not super helpful.

The other advantage of using the Request object is that you do not need to check to see if a value is set. You can, but even if you try to access a variable not passed through the HTTP request, it will simply return null rather than error as is the default behavior of the $\_REQUEST hash variable.

Now if you submit a valid Email address and password, it will display the Master Customer ID from Personify. If not, it will return various errors and return the user to the form.

How the PersonifySSO class works will be covered in another document.

The next step is to create a User object that will persist with the Session so that the logged in Personify user can begin to view their data.

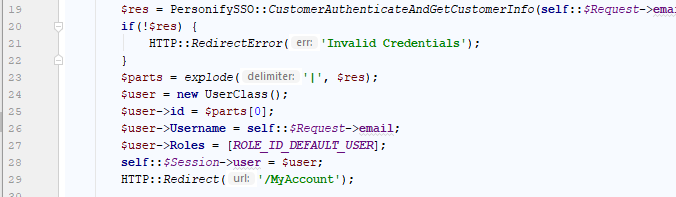
# Creating a Logged In User

Previously we just used CleanHalt to spit out the contents of what Personify returned when using the SSO function. What you get back is something like

000002242452|0

So we need to use explode to break that into two strings. The first string is the Master Customer ID, the second is the Sub Master Customer Id. The leading zeros must be preserved so do not do any sort of math operations on it.

In this use-case, the sub master customer id is ignored. Your system may require it, in which case you may want to skip the explode and just put the original string in the id property of the UserClass.



Notice that the UserClass object is being put into the session. That’s one of the key features of using the Session object rather than the $\_SESSION global. The Session object handles serializing and deserializing values automatically so they can be put into the $\_SESSION global.

The QuickDRY framework expects a UserClass to be put in the user Session parameter to indicate a logged in user.

The UserClass has a user property which is intended to store any custom User object your project requires. You do not need to change the UserClass itself.

The id property of the UserClass is the unique identifier for the user. The Username is generally whatever the user submitted as their username.

If the system you are integrating with has roles for the user, then this would be the place to set them. In this instance, we are just setting them as the ROLE\_ID\_DEFAUL\_USER role which grants them access to any pages that have the same role assigned to it.

The first page we created was MyAccount and they will be redirect there if the sign in is successful.

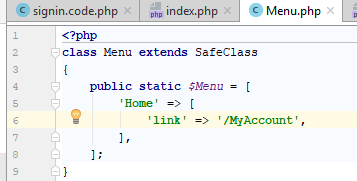
# Updating Index.php

Now that we’ve sorted out what our main page will be for the users, we need to update index.php to reflect that.



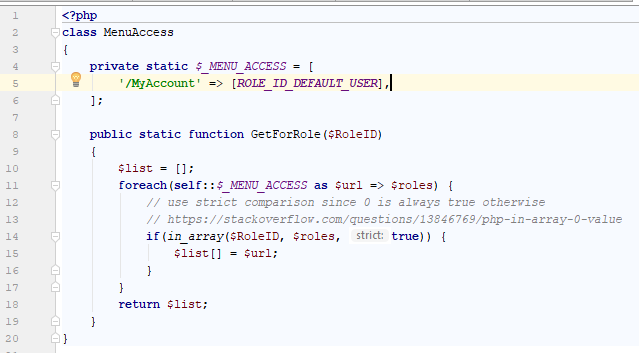
Note, that instead of “admin” the default\_user\_page is now “MyAccount”

We also need to update the menu in QuickDRYInstance/Menu.php



When the user is logged in, the top navigation will show “Home” and clicking it will take the user to MyAccount

We also have to add permissions to this page in MenuAccess.php so that it can be accessed.



Now the page can only be accessed by a user with the role ROLE\_ID\_DEFAULT\_USER

QuickDRYInstance/Defines.php contains all the initial ROLE\_ID definitions.

The convention of putting the use first in the name makes it easier to find things with PHPStorm’s code completion. Start typing ROLE\_ID and all the available ROLE\_ID’s come up quickly.