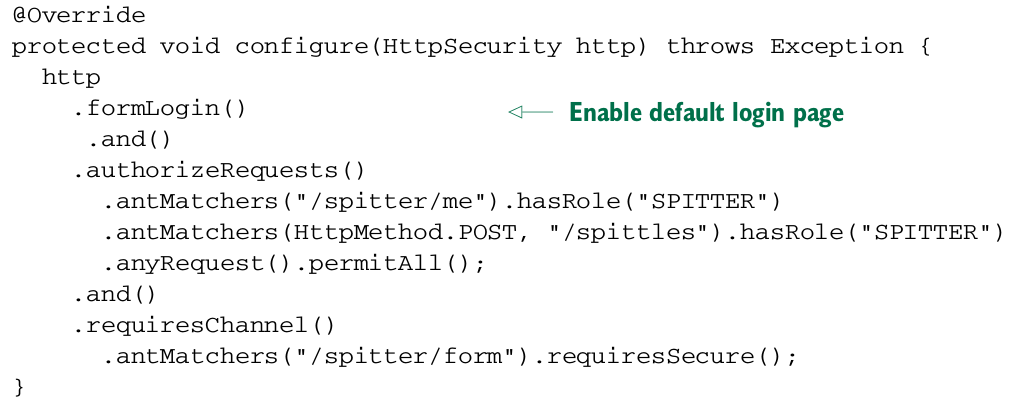
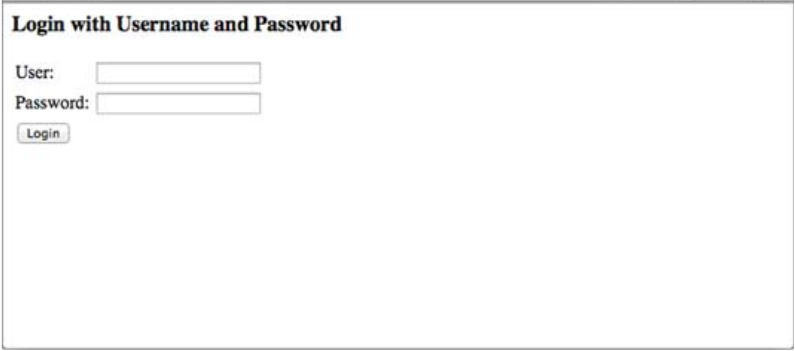
***Authenticating Users***

When you were still using the extremely simple Spring Security configuration in previous listings ,you got a login page for free. In fact, up until you overrode configure(HttpSecurity) , you could count on a plain-vanilla, yet fully functional login page. But as soon as you override configure(HttpSecurity) , you lose that simple login page.

Fortunately, it’s easy enough to get it back. All you need to do is call formLogin() in the configure(HttpSecurity) method, as shown in the following listing.

Notice that, as before, and() is called to chain together different configuration instructions.

If you link to /login in the application, or if the user navigates to a page that requires authentication, then the login page will be shown in the browser. As you can see in figure, the page isn’t very exciting aesthetically, but it does the job it needs to do.



It’d be a shame to have such a plain login page ruin your otherwise beauti fully designed website. No problem. Let’s see how you can add a custom login page to your application.

***Adding a custom login page***

The first step toward creating a custom login page is knowing what you need to

include in the login form. Look no further than the HTML source of the default login

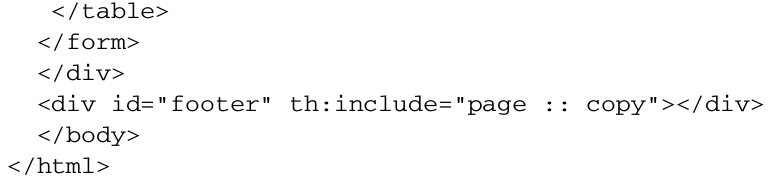
page to see what’s required:



The key thing to note is where the <form> submits to. And make note of the username and password fields; you’ll need those same fields on your login page. Finally, assuming that you’ve not disabled CSRF , you’ll need to be sure to include a \_csrf field with the CSRF token.

The following listing shows a Thymeleaf template that provides a login page within the style of the Spittr application.

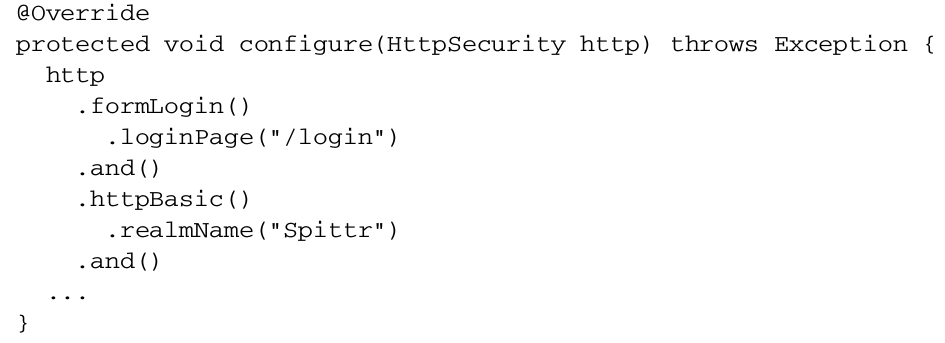




Notice that the Thymeleaf template has both username and password fields, just like the default login page. It also submits to the context-relative /login page. And since this is a Thymeleaf template, the hidden \_csrf field will automatically be added to the form.

***Enabling HTTP Basic authentication***

HTTP Basic authentication is one way to authenticate a user to an application directly in the HTTP request itself. You may have seen HTTP Basic authentication before. When encountered by a web browser, it prompts the user with a plain modal dialog box. But that’s just how it’s manifested in a web browser. In reality, it’s an HTTP 401 response, indicating that a username and password must be presented with the request. This makes it suitable as a means for REST clients to authenticate against the services they’re consuming. Enabling HTTP Basic authentication is as simple as calling httpBasic() on the HttpSecurity object passed into configure() . Optionally, you can specify a realm by calling realmName() . Here’s a rather typical example of Spring Security configuration to enable HTTP Basic:

Notice that once again the and() method is used to chain together different configuration directives in configure() .

Not much customization is available or even required with httpBasic() . HTTP Basic authentication is either turned on or it’s not. So rather than dwell on the topic any further, let’s move on to see how to have a user automatically authenticated via remember-me functionality.