5. Securing Spring

# 5.1 Enabling Spring Security

-Add SB security starter dependency:



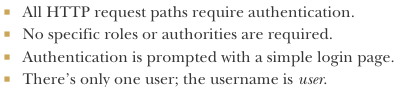
+This dependency is the only thing that’s required to secure an app.

-**Note**: incognito: have a fresh session each time opening a private/incognito window.

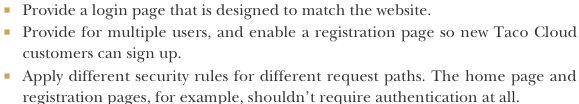
-You need to provide a username + password. User name is *user.* Password is randomly generated and written to app log file:



-Some security features when adding security starter:

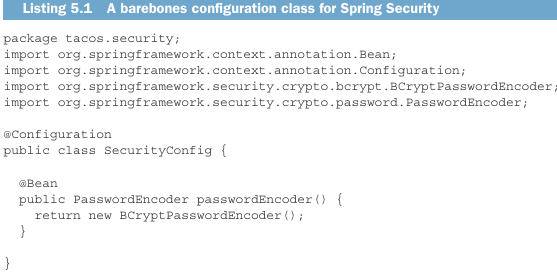


-We’ll configure Spring Security to do:



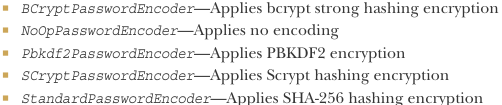
# 5.2 Configuring authentication

-Write the **configuration class**:



+It declares **PasswordEncoder** bean: use when create new users and authenticate users at login.

+**BCryptPasswordEncoder**: password encoders. Others:



+The password in the database must be never decoded. The password that the user enter at login is encoded, it’s then compared with the encoded password in database by PasswordEncoder **matches()**

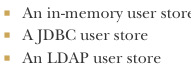
**-**Configure handle more than one user.

**-**Configure a user store for authentication: declare **UserDetailService** bean.



+**loadUserByUsername()**: use username to look up UserDetails object.

+SS offers some out-of-the-box implementation of UserDetailService

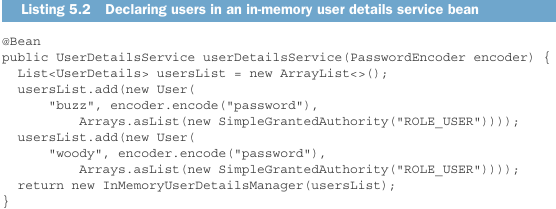


+Or you can create your own implementation.

## 5.2.1 In-memory user details service

-One place where user info can be kept in memory. None of users are likely to change may be defined as part of security configuration.

-Create InMemoryUserDetailManager bean method:



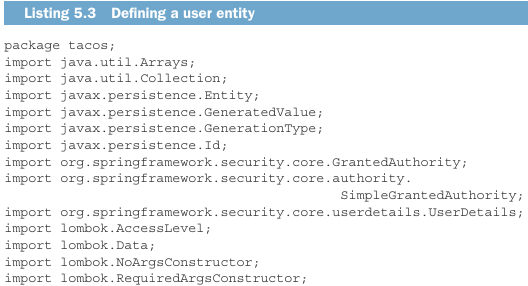
-**in-memory user details service** is **convenient** for **testing purposes** or for very simple apps, but it **doesn’t allow** to easy **edit users.**

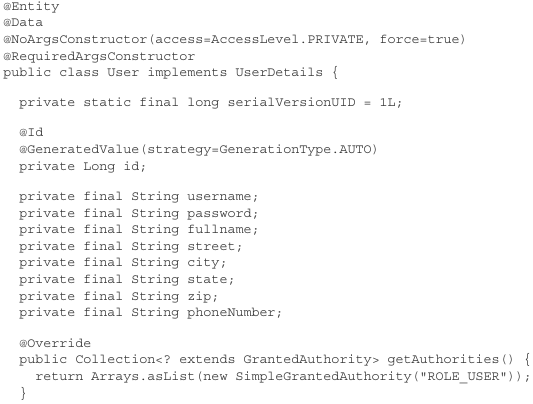
## 5.2.2 Customizing user authentication

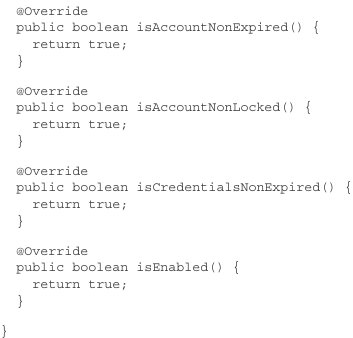
-We use Spring Data JPA repository to store users

-Defining the user domain and persistence

+Create a User class:





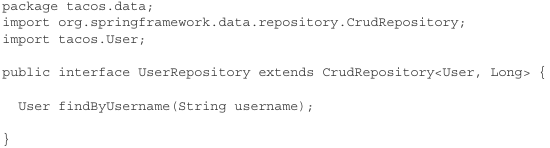


+User also implements **UserDetails** interface: provide some essential user information: what authorities are granted to user and whether user’s account is enabled.

+getAuthorities(): return collection of authorities granted to the user.

+is\*(): whether the user’s account is enabled, locked or expired.

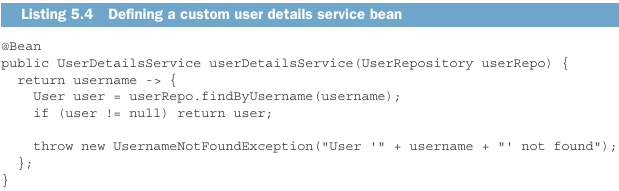
-Define repository interface:



-Creating a user details service

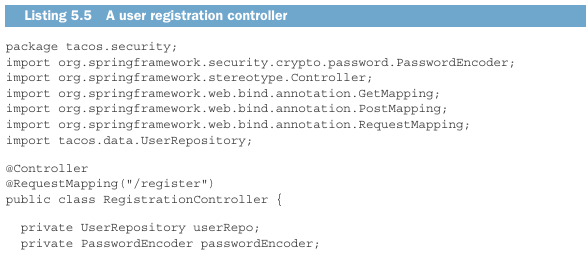
+Custom user details service reads user information via a JPA repository:

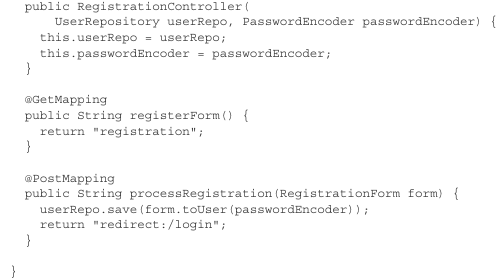
+In SecurityConfig.java



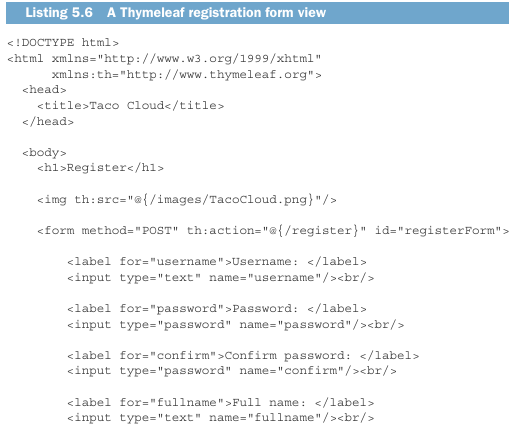
-Registering users

+**RegistrationController**: process registration forms





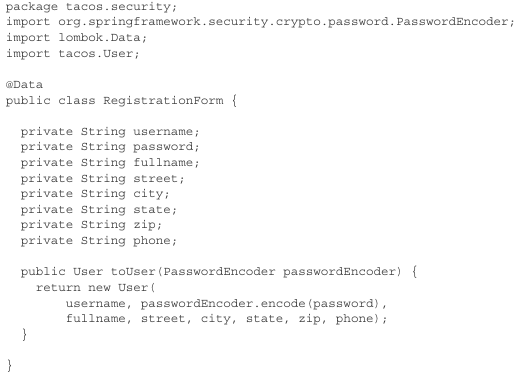
-registration.form







+When the form is submitted, **processRegistration()** handles HTTPS POST request. The form fields will be bound to **RegistrationForm** object and passed into method:



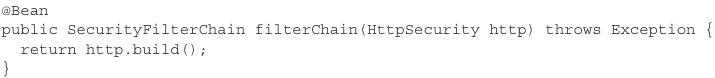
-By **default**, **all requests** require **authentication**.

# 5.3 Securing web requests

-User authentication: design taco, place order.

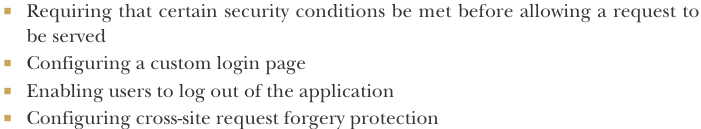
-Unauthentication: home page, login page, registration

-Declare **SecurityFilterChain** bean:



+filterChain(): configure how security is handled at the web level.

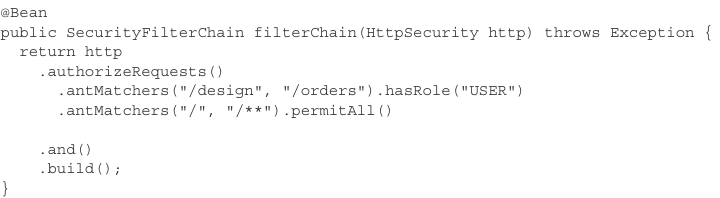
+Things we can configure with HttpSecurity:



+**Intercepting requests** to ensure that the user has **proper authority** is one of the most common things you’ll configure **HttpSecurity** to do.

## 5.3.1 Securing requests

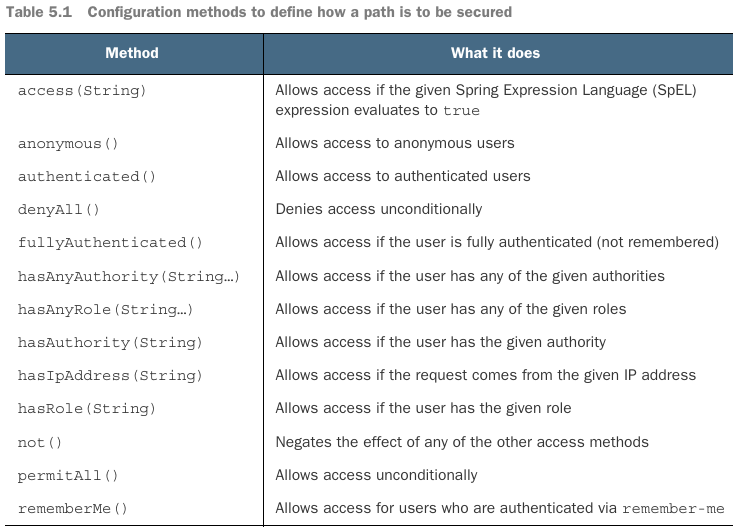
-Requests /design and /orders are available only to authenticated users, all other requests should be permitted for all users:



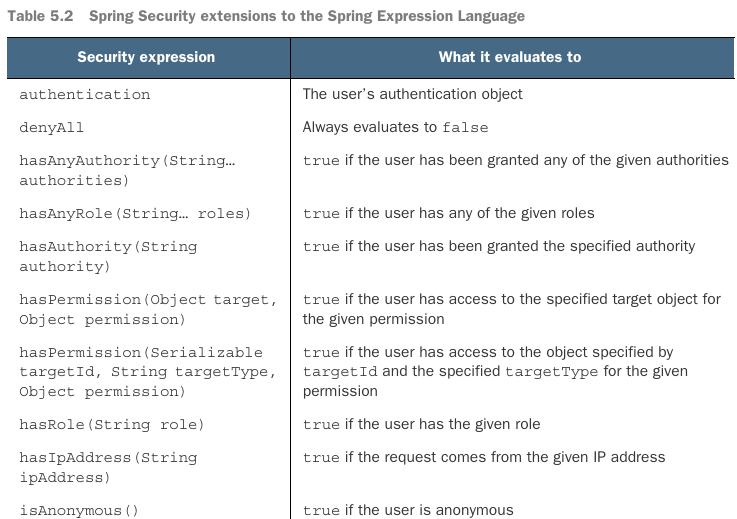
+Note: Use **requireMatchers()** instead of antMachers()

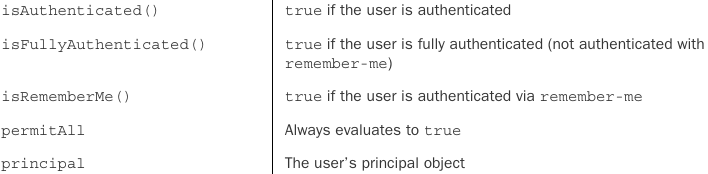
+The order of rules is important. Security rules declared 1st take precedence over those declared lower down.

-Other security requirements for request paths:

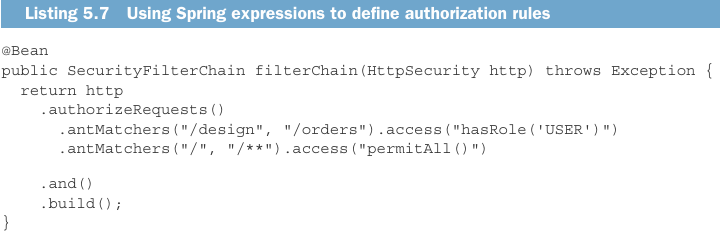


-You can use access() to provide a SpEL expression to declare security rules

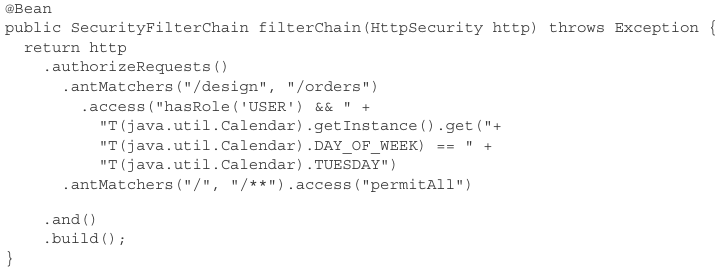




+Use access() to rewrite:



+These expressions are flexible. Example: allow only users with ROLE\_USER to create new tacos on Tuesdays

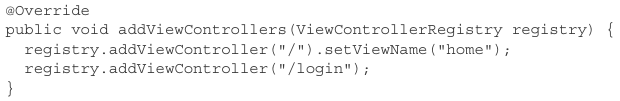


## 5.3.2 Creating a custom login page

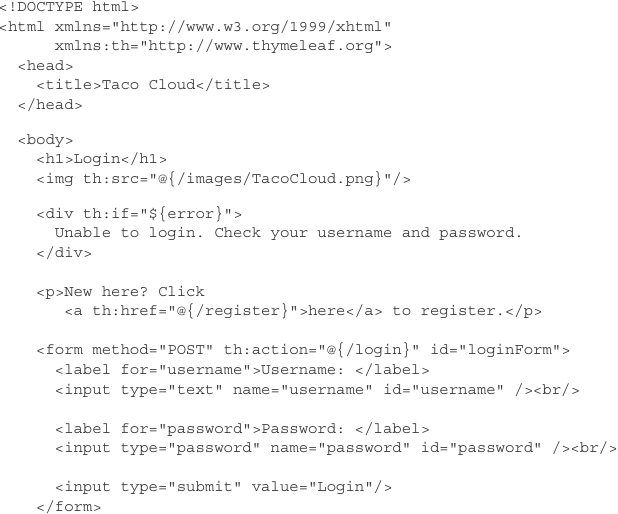
-**formLogin()**: configure the path of custom login page



-Add controller in WebConfig

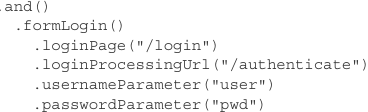


-Define the login page view:

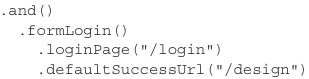


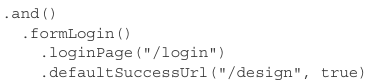


-By default, SS listens for login requests at **/login** and expects that the username and password fields be named **username** + **password**. These are configurable:



-Be default, a successful login will take user to page that they were navigating to when SS determined that they needed to log in. If user directly navigates to login page, login would take them to root path (home page). Change them:





+true: go to design page even if they were navigating elsewhere prior.

## 5.3.3 Enabling third-party authentication

## 5.3.4 Preventing cross-site request forgery

# 5.4 Applying method-level security

# 5.5 Knowing your user

# -Summary