5. Securing Spring

# 5.1 Enabling Spring Security

-Add SB security starter dependency:



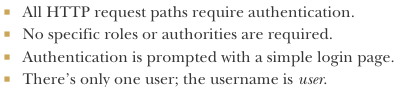
+This dependency is the only thing that’s required to secure an app.

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

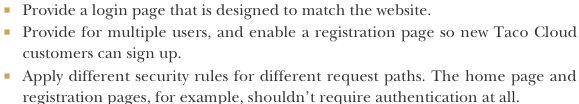
-You need to provide a username + password. User name is *user.* Password is randomly generated and written to app log file:



-Some security features when adding security starter:

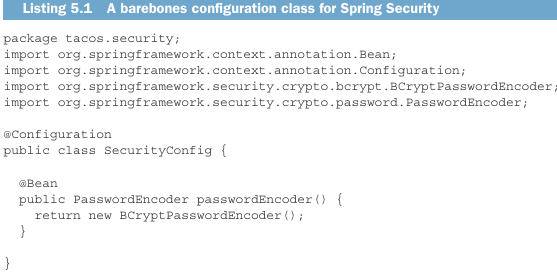


-We’ll configure Spring Security to do:



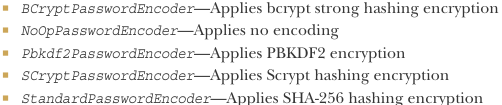
# 5.2 Configuring authentication

-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()**

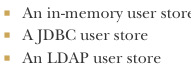
**-**Configure handle more than one user.

**-**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

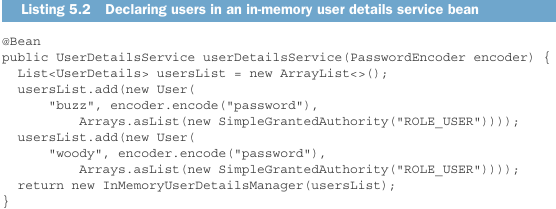


+Or you can create your own implementation.

## 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:



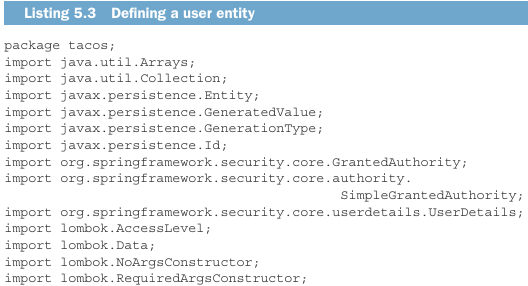
-**in-memory user details service** is **convenient** for **testing purposes** or for very simple apps, but it **doesn’t allow** to easy **edit users.**

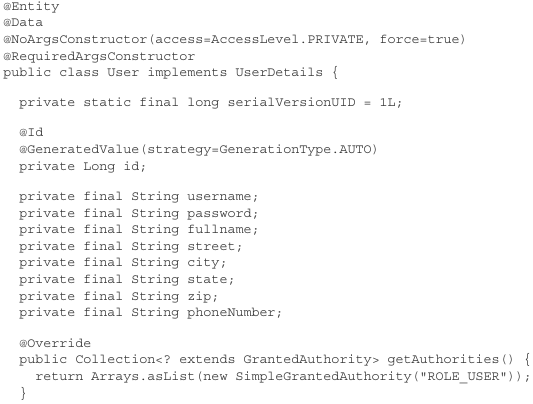
## 5.2.2 Customizing user authentication

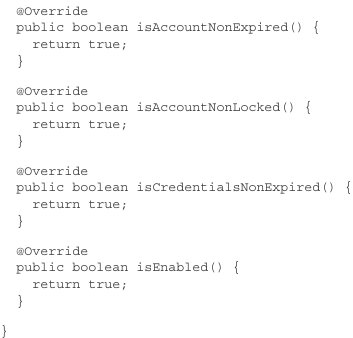
-We use Spring Data JPA repository to store users

-Defining the user domain and persistence

+Create a User class:





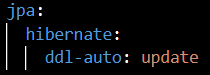


+User also implements **UserDetails** interface: provide some essential user information: what authorities are granted to user and whether user’s account is enabled.

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

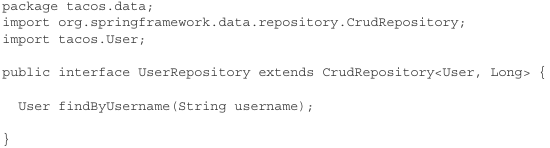
+is\*(): whether the user’s account is enabled, locked or expired.

-Note: When you want to use database like MySQL, add this:



+It will create automatically tables from entities + table \_seq.

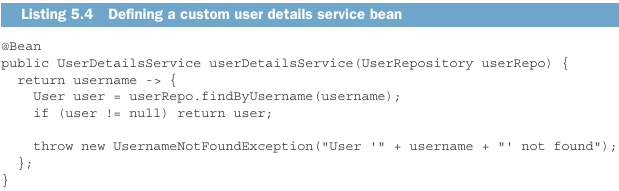
-Define repository interface:



-Creating a user details service

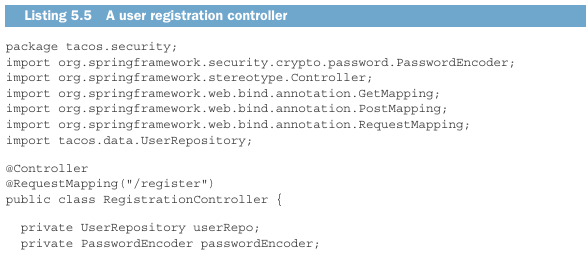
+Custom user details service reads user information via a JPA repository:

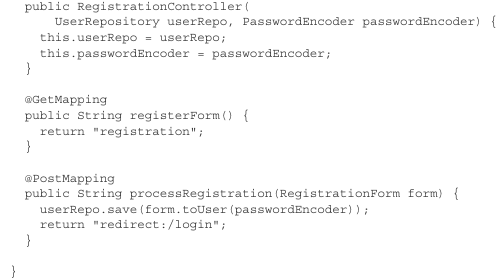
+In SecurityConfig.java



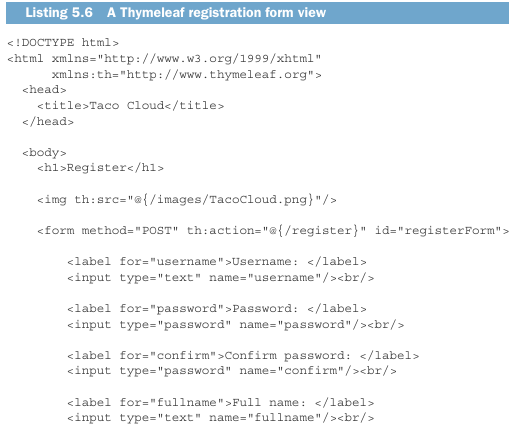
-Registering users

+**RegistrationController**: process registration forms





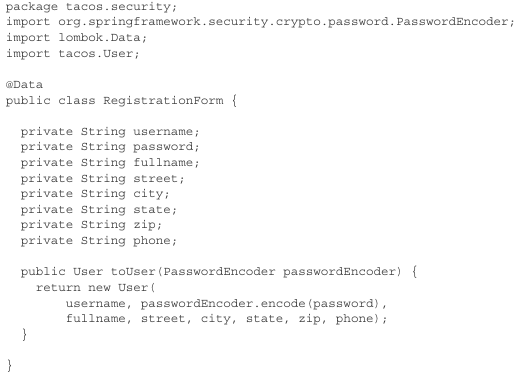
-registration.form







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



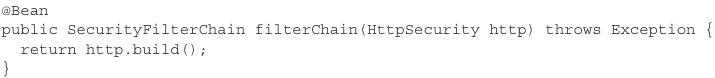
-By **default**, **all requests** require **authentication**.

# 5.3 Securing web requests

-User authentication: design taco, place order.

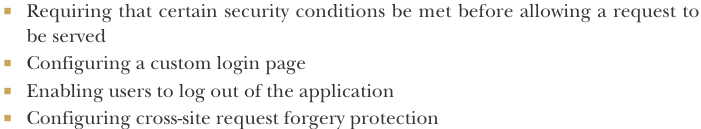
-Unauthentication: home page, login page, registration

-Declare **SecurityFilterChain** bean:



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

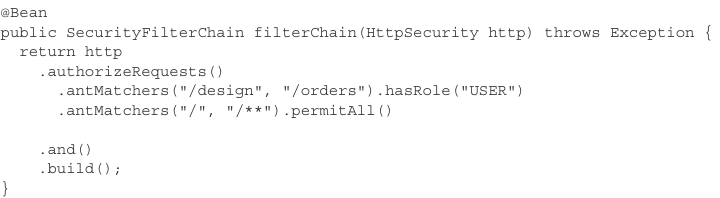
+Things we can configure with HttpSecurity:



+**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

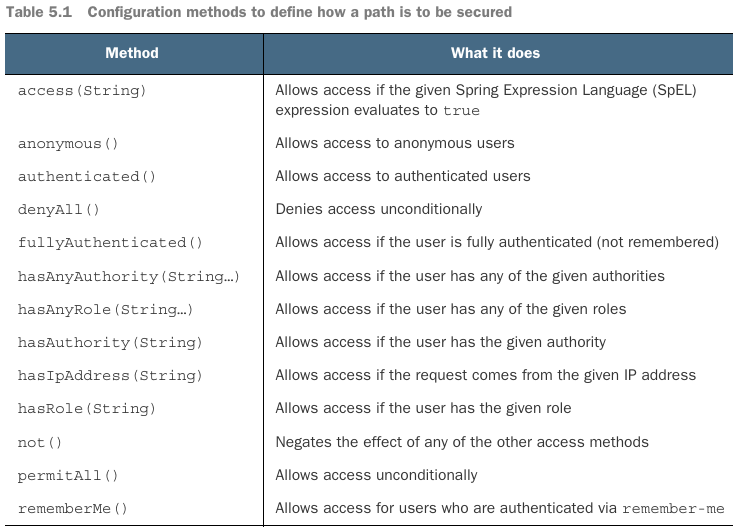
-Requests /design and /orders are available only to authenticated users, all other requests should be permitted for all users:



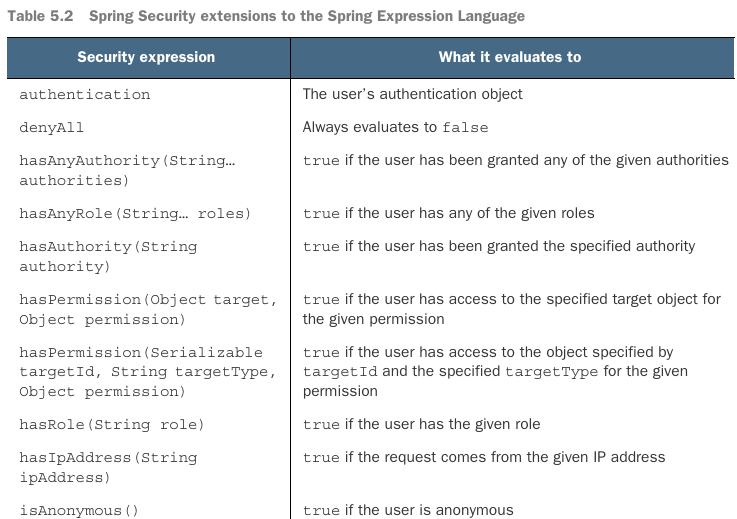
+Note: Use **requireMatchers()** instead of antMachers()

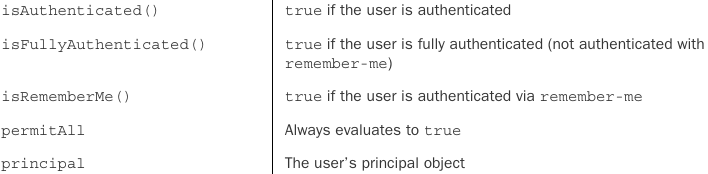
+The order of rules is important. Security rules declared 1st take precedence over those declared lower down.

-Other security requirements for request paths:

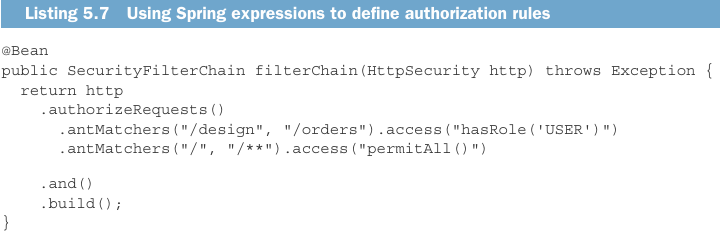


-You can use access() to provide a SpEL expression to declare security rules

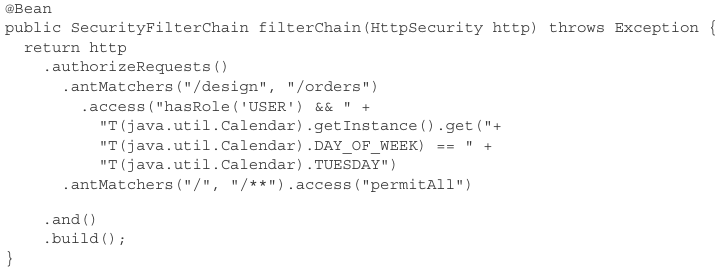




+Use access() to rewrite:



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

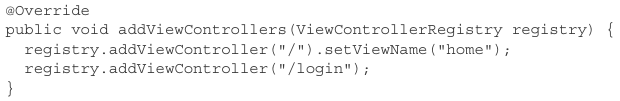


## 5.3.2 Creating a custom login page

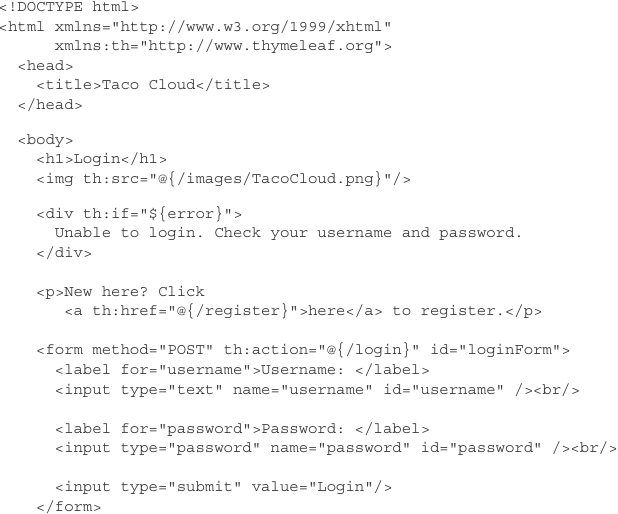
-**formLogin()**: configure the path of custom login page



-Add controller in WebConfig

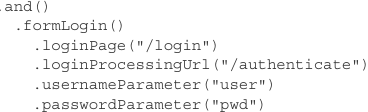


-Define the login page view:

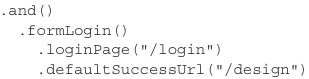


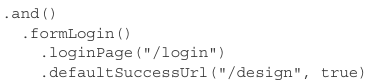


-By default, SS listens for login requests at **/login** and expects that the username and password fields be named **username** + **password**. These are configurable:



-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.

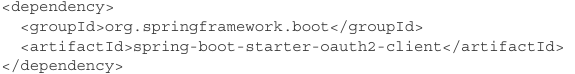
## 5.3.3 Enabling third-party authentication

-Offer a way to sign in via another website like Facebook. This type of authentication is based on OAuth2 or OpenId Connect (OIDC)

+OAuth2 is an authorization specification

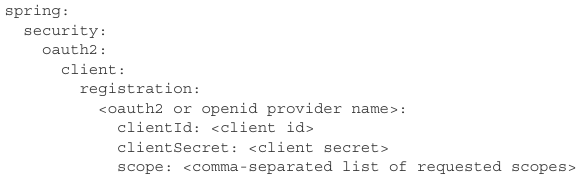
+OCID is another security specification that is based on OAuth2 to formalize the interaction that takes place during a third-party authentication

-Add OAuth2 client starter:

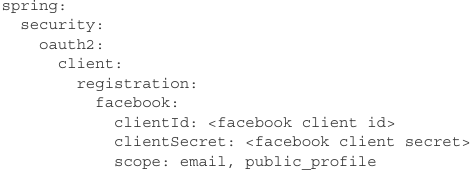


-You’ll need to configure details about one or more OAuth2 or OpenID Connect server. SS supports sigin-in Facebook, Google, GitHub and Okta. You can configure other clients.

-The general set of properties for OAuth2/OpenID Connect client:



+Example: sign in using Facebook in application.yml



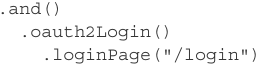
+The **client Id** and **secret** are the credentials that identity the app to Facebook. You obtain a client ID and secret by creating a new app entry at <https://developers.facebook.com/>

+**scope**: specify the access that the app will be granted

-If you customize security to declare SecurityFilterChain bean, you need to enable OAuth2 login:



-Offer both traditional username-password login + third party login:



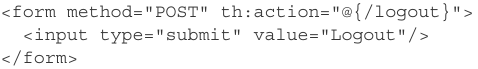
+It takes the user to application-provided login page where they may choose to log in with username +password.

-Provide a link on that same login page to log in Facebook

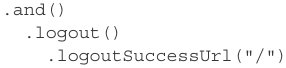


-Enable a user to log out: 

+This sets up a security filter that intercepts **POST** requests to **/logout**. You can add a logout form and button the views in app:



+When user clicks button, the session will be cleared + log out. By default, they will be redirected to login page. You can go to different page:



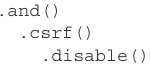
## 5.3.4 Preventing cross-site request forgery

-Cross-site request forgery (CSRF) is a common security attack. It subjects a user to code on a web page that automatically submits a form to another app on behalf of user who is the victim of the attack.

-To protect, app can generate a CSRF token upon displaying a form, place that token in a hidden field, and then stow it for later use on the server. When the form is submitted, the token is sent back to server along with the rest of form data. The request is intercepted by sever and compared with token that was originally generated. If matched, request is proceed.

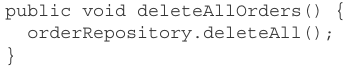
-SS has built-in CSRF protection, it’s enabled by default. Add this hidden field in form:

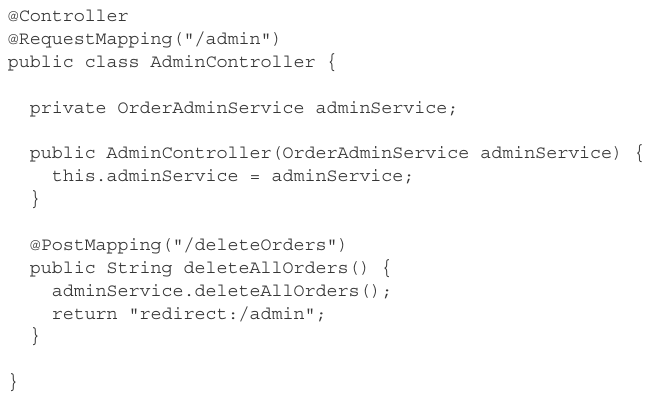


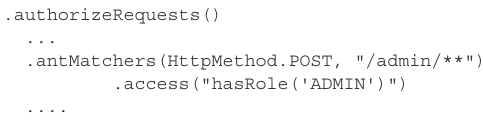
-Disable CSRF protection: 

# 5.4 Applying method-level security

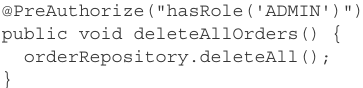
-Sometime it’s better to verify that user is authenticated and has been granted adequate authority at the point where the secured action will be performed.



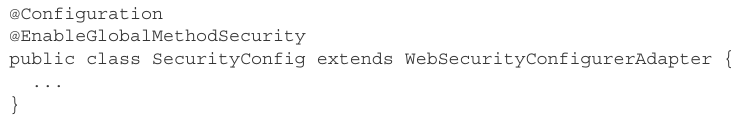




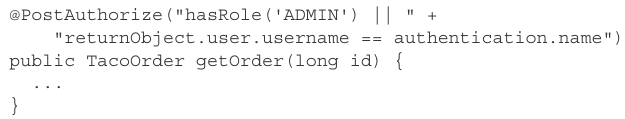
-Apply security on method:



-You need to enable global method security

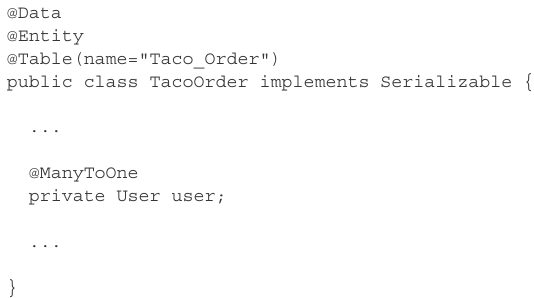


-@PostAuthorize:

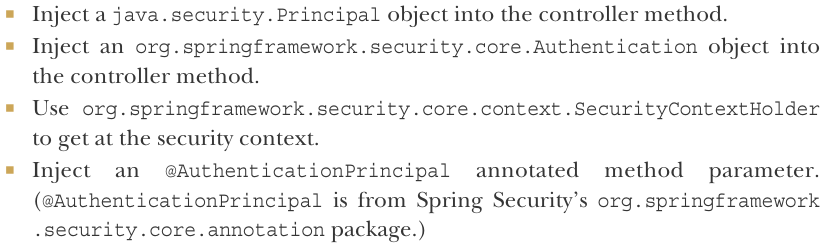


# 5.5 Knowing your user

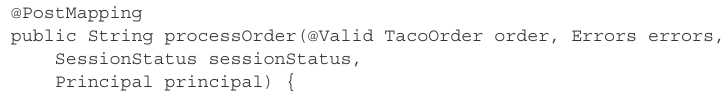
-Add a connection between TacoOrder entity and User entity:

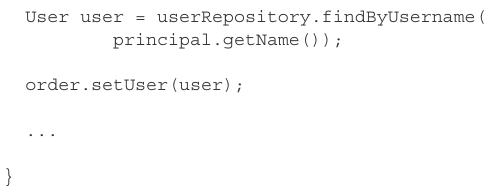


-In processOrder(), we want to determine who the user is. The most common ways:

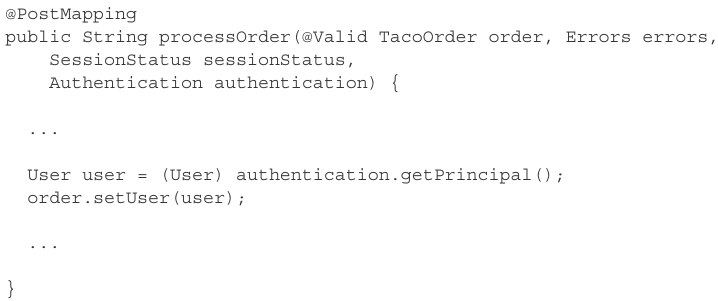


-Use Principal:

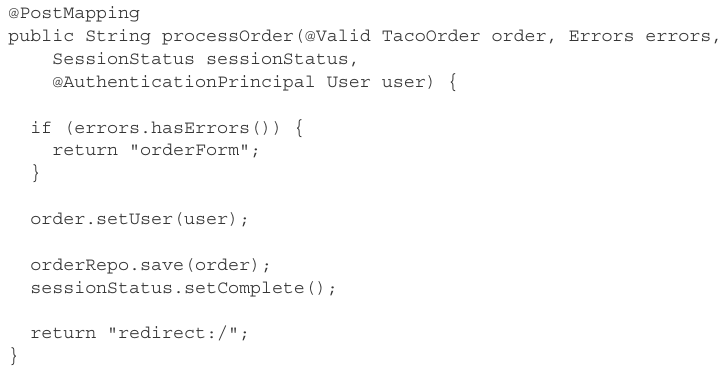




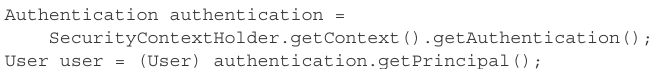
-Use Authentication:



-The cleanest solution is accept a User object with @AuthenticationPrincipal:



-Another way with security-specific code to identify who the user is:



+It can be used anywhere in app, not just in a controller’s handler methods.

# -Summary

