Section 3: A Deeper Dive into Testing

Now that you are done with the videos of section 3, let’s assess your learning. Here, are a few questions, followed by 4 options, out of which 1 is the correct option. Select the right option and validate your learning! The answers are provided in a separate sheet

Q1. You want to load a custom user class in your test, which annotation is the correct way to do that?

a) @WithMockUser(userDetailsService=true, name=“admin”)
b) @WithMockUser(userDetailsService=myUserDetailsService, name=“admin”)
c) @WithUserDetails(name=“admin”)
d) @WithUserDetails(name=“admin”, roles={“ADMIN”,”USER”})

Q2. You are writing a MockMVC based test of your controller. How does the expect matcher look if you want to check for a specific value in your model?

a) .andExpect(model().attribute(“myFieldName”, ENTER_VALUEMATCHER_HERE));.
b) .andExpect(model(“myFieldName”).matches(ENTER_VALUEMATCHER_HERE));.
c) .andExpect(modelAttribute(“myFieldName”, ENTER_VALUEMATCHER_HERE));.
d) .andExpect(model().attribute(“myFieldName”).matches(ENTER_VALUEMATCHER_HERE));.

Q3. How do you check that the HTML of your application contains the correct content using HtmlUnit?

a) You have to get the response as text and use for example regular expressions to verify that the content is correct!
b) You load the page, check that it is a Html page and then query it using xPath or selectors for example
c) You load the page and the use specialised ham crest matchers to verify the state of the page

Q4. You want to test your service with selenium using a real browser, where can you load the driver from?

a) The driver can simply be loaded from the class path.
b) The driver can to be loaded from the filesystem.
c) The driver can be fetched from the web via http.