Learning Consolidation

Test RESTful Services at Controller Layer by Using Testing Tools (JUnit, Mockito)

Menu

00:01:00:00:07 1/7

In this sprint, you have learned to:

- Implement controller layer testing

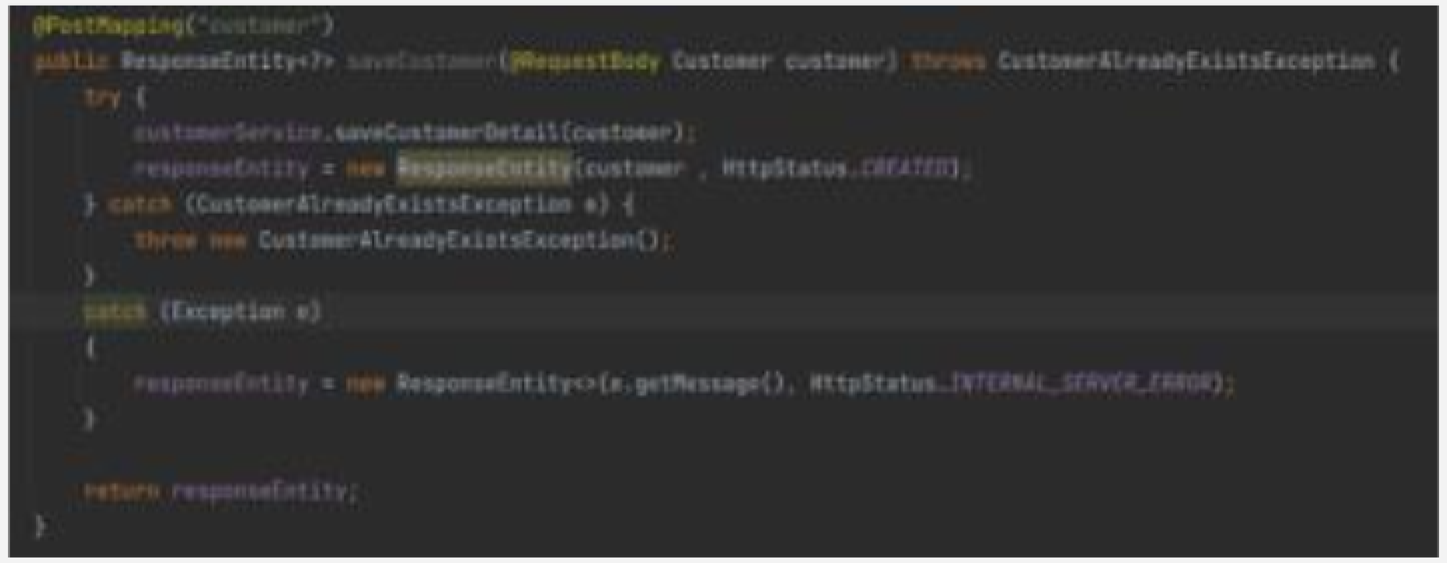
What Is MockMVC?

- MockMvc has been around since Spring 3.2.
- MockMvc is mainly used to test the code of the controller layer.
- It provides a powerful way to mock **Spring** MVC for testing the MVC web applications. Through **MockMvc**, you can send mock HTTP requests to a controller and test how the controller behaves without running the controller within a server.
- MockMvc testing is needed for:
  - Content negotiation headers: To produce only application/JSON content.
  - Response code: To check if the response code matches the expected one.
  - JSON serialization/deserialization: To validate JSON is deserialized and correctly converted into the response body.

Slide Note

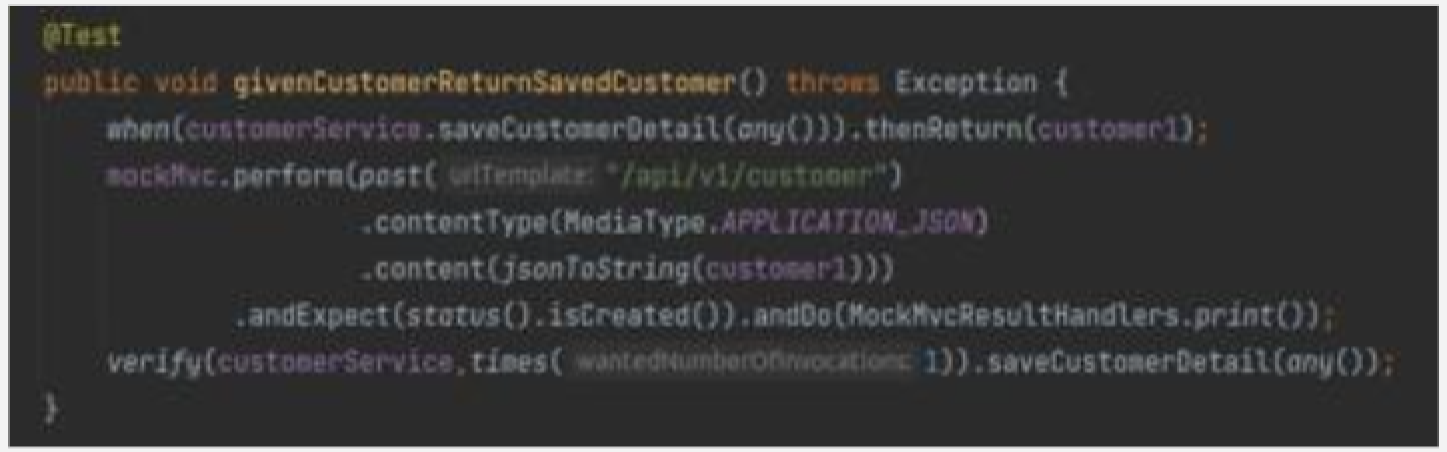
Testing the Controller Layer

- Here, the `when` method of Mockito is used to mock the service layer and set an expectation.
- In this code, the expectation is that when `customerService saveCustomerDetails()` method is called with any customer object as an argument, then it will return the saved object.
- By using the `mockMvc` object, you call "`perform`" method to make a mock call to the API post method.
- Post takes the API endpoint as an argument. Use the same endpoint that was set in RequestMapping in the controller.
- Here, `(.andExpect(`*`status`*`().isCreated())` is a basic check for 201 status.



```java
@PostMapping("customer")
public ResponseEntity<?> saveCustomer(@RequestBody Customer customer) throws CustomerAlreadyExistsException {
    try {
        customerService.saveCustomerDetail(customer);
        responseEntity = new ResponseEntity(customer , HttpStatus.CREATED);
    } catch (CustomerAlreadyExistsException e) {
        throw new CustomerAlreadyExistsException();
    }
    catch (Exception e)
    {
        responseEntity = new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL_SERVER_ERROR);
    }

    return responseEntity;
}
```



```java
@Test
public void givenCustomerReturnSavedCustomer() throws Exception {
    when(customerService.saveCustomerDetail(any())).thenReturn(customer1);
    mockMvc.perform(post( urlTemplate: "/api/v1/customer")
                    .contentType(MediaType.APPLICATION_JSON)
                    .content(jsonToString(customer1)))
            .andExpect(status().isCreated()).andDo(MockMvcResultHandlers.print());
    verify(customerService,times( wantedNumberOfInvocations: 1)).saveCustomerDetail(any());
}
```

Menu

00:04 / 00:00:07 4/7

Slide Note

Quick Check

___________ is used to create and inject a mock for the service class.

1. @Mock
2. @MockMvc
3. @Bean
4. @InjectMocks

Menu

00:05:00:00:07 5/ 7

Slide Note

Quick Check: Solution

__________ is used to create and inject a mock for the service class.

1. @Mock
2. @MockMvc
3. @Bean
4. **@InjectMocks**

Menu

00:06:00:00:07 6/ 7