Spring MockMVC RestController Unit Testing

What is MockMVC?

- Spring (Boot) Framework unit testing tool
- Integrates with jUnit
- Allows testing of @RestControllers
- Supports unit ***and*** integration testing

How Does it Work?

- MockMVC requests to @RestController endpoint with:
  - *Custom payload*
  - *Content type*
  - *Accept type*
  - *HTTP Verb*

- MockMVC sets up expectations for the response

- Is ran with regular maven test phase

Why is this Useful?

- RestControllers aren’t called from a “normal” call context (MockMVC trivializes this invocation)
- The services that a particular RestController relies on can be fully mocked out for testing
- Very realistic testing

What? Unit AND Integration Testing?

- Full mocked out service layer (unit testing)

OR

- True end-to-end testing (integration testing)

Demonstration: RestController Implementation

```java
@RestController
@RequestMapping("/questions")
public class QuestionController {
  @Autowired
  private QuestionAnswerService questionAnswerService;

  @RequestMapping(method = RequestMethod.POST,
      produces = MediaType.APPLICATION_JSON_VALUE, consumes = MediaType.APPLICATION_JSON_VALUE)
  public ResponseEntity<Map<String, Long>> postQuestion(@Validated @RequestBody Question question) {
    Long questionId = questionAnswerService.askQuestion(question);

    Map<String, Long> questionIdResponse = new HashMap<String, Long>();
    questionIdResponse.put("id", questionId);

    return new ResponseEntity<Map<String, Long>>(questionIdResponse, HttpStatus.CREATED);
  }
}
```

Demonstration: Test Setup

```
@RunWith(SpringJUnit4ClassRunner.class)
@WebAppConfiguration
@ContextConfiguration(classes = { MyApplication.class }, loader = SpringApplicationContextLoader.class)
public class QuestionControllerTest {
  private MockMvc mockMvc;

  @Autowired
  @InjectMocks
  private QuestionController questionController;

  @Mock
  private QuestionAnswerService questionAnswerServiceMock;

  @Autowired
  private WebApplicationContext springApplicationContext;

  ...
```

Demonstration: RestController Test

```
@Test
public void postQuestionSuccessfully() throws Exception {
  when(questionAnswerServiceMock.askQuestion(Mockito.any(Question.class)))
    .thenReturn(new Long(1));

  Question questionOne = new Question();
  questionOne.setId(new Long(1));
  questionOne.setContent("question content here");
  questionOne.setEmail("dolor@set.amet");
  questionOne.setTitle("consectetur adipiscing");

  final ObjectMapper mapper = new ObjectMapper();

  mockMvc.perform(MockMvcRequestBuilders.post("/questions")
    .content(mapper.writeValueAsString(questionOne))
    .contentType(MediaType.APPLICATION_JSON)
    .accept(MediaType.APPLICATION_JSON))
    .andExpect(status().is2xxSuccessful())
    .andExpect(jsonPath("$.id", Matchers.is(1)));

  verify(questionAnswerServiceMock, times(1)).askQuestion(Mockito.any(Question.class));
  verifyNoMoreInteractions(questionAnswerServiceMock);
}
```