[Bài đọc] Trình khởi động Test Starter

Quản lý sự phụ thuộc(dependency) là một khía cạnh quan trọng của bất kỳ dự án phức tạp nào. Và làm điều này theo cách thủ công là ít lý tưởng hơn; bạn càng dành nhiều thời gian cho nó thì bạn càng có ít thời gian hơn cho các khía cạnh quan trọng khác của dự án.

Bộ khởi động Spring Boot(Spring Boot starters) được xây dựng để giải quyết chính xác vấn đề này. Starter POMs là một tập hợp các bộ mô tả phụ thuộc thuận tiện mà bạn có thể đưa vào ứng dụng của mình. Bạn nhận được một điểm dừng duy nhất cho tất cả Spring và công nghệ liên quan mà bạn cần, mà không cần phải tìm kiếm mã mẫu và sao chép-dán vô số bộ mô tả phụ thuộc(dependency descriptors).

Chúng ta có sẵn hơn 30 trình khởi động Boot - hãy xem một trong số chúng:

Test Starter

Để testing, chúng ta thường sử dụng bộ thư viện sau: Spring Test, JUnit, Hamcrest và Mockito. Chúng ta có thể bao gồm(include) tất cả các thư viện này theo cách thủ công, nhưng Spring Boot Starters có thể được sử dụng để tự động include các thư viện này theo cách sau:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

Lưu ý rằng bạn không cần chỉ định số phiên bản của một artifact. Spring Boot sẽ tìm ra phiên bản để sử dụng.

Hãy kiểm tra bộ điều khiển mà chúng ta đã tạo trong ví dụ trước.

Có hai cách để kiểm tra bộ điều khiển:

* Sử dụng môi trường giả lập(mock environment)
* Sử dụng Servlet Container được nhúng (như Tomcat hoặc Jetty)

Trong ví dụ này, chúng ta sẽ sử dụng một môi trường giả lập:

@RunWith(SpringJUnit4ClassRunner.class)

@SpringApplicationConfiguration(classes = Application.class)

@WebAppConfiguration

public class SpringBootApplicationIntegrationTest {

@Autowired

private WebApplicationContext webApplicationContext;

private MockMvc mockMvc;

@Before

public void setupMockMvc() {

mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();

}

@Test

public void givenRequestHasBeenMade\_whenMeetsAllOfGivenConditions\_thenCorrect()

throws Exception {

MediaType contentType = new MediaType(MediaType.APPLICATION\_JSON.getType(),

MediaType.APPLICATION\_JSON.getSubtype(), Charset.forName("utf8"));

mockMvc.perform(MockMvcRequestBuilders.get("/entity/all")).

andExpect(MockMvcResultMatchers.status().isOk()).

andExpect(MockMvcResultMatchers.content().contentType(contentType)).

andExpect(jsonPath("$", hasSize(4)));

}

}

Test ở trên gọi “/entity/all” và xác minh rằng phản hồi JSON có chứa 4 phần tử. Để test này vượt qua, chúng ta cũng phải khởi tạo danh sách trong lớp controller:

public class GenericEntityController {

private List<GenericEntity> entityList = new ArrayList<>();

{

entityList.add(new GenericEntity(1l, "entity\_1"));

entityList.add(new GenericEntity(2l, "entity\_2"));

entityList.add(new GenericEntity(3l, "entity\_3"));

entityList.add(new GenericEntity(4l, "entity\_4"));

}

*//...*

}

Điều quan trọng ở đây là chú thích @WebAppConfiguration và MockMVC là một phần của mô-đun Spring Test, hasSize là bộ so khớp Hamcrest và @Before là chú thích JUnit. Tất cả đều có sẵn bằng cách thêm một phụ thuộc(dependendy) duy nhất.