



2.3 – Pruebas de Servicios de Internet

# Tema 2 - Testing unitario y de integración



Tema 2 - Testing unitario y de integración

# Tema 2.1 – Testing con Spring

- **Spring y Spring Boot** ofrece muchas herramientas para facilitar el testing
- Se integra con diversas librerías y ofrece algunas funcionalidades propias:
  - Gestión de mocks (con mockito)
  - Mocks del servidor web
  - Tests de acceso a bases de datos
  - Tests de integración

- **Mock del servidor web**
  - Se pueden simular peticiones web sin ejecutar el servidor web (velocidad y control)
  - Existe un cliente Http mock que se conecta directamente al servidor web mock
  - Se utilizan “plugins” de JUnit para controlar el ciclo de vida de estos elementos de forma muy sencilla

- Mock del servidor web

ejem1

```
@SpringBootTest
@AutoConfigureMockMvc
public class AnuncioControllerTest {

    @Autowired
    private MockMvc mvc;

    @Test
    public void getAnuncioTest() throws Exception {

        mvc.perform(get("/anuncio")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.nombre", equalTo("Pepe")));

    }
}
```

- Mock del servidor web

ejem1

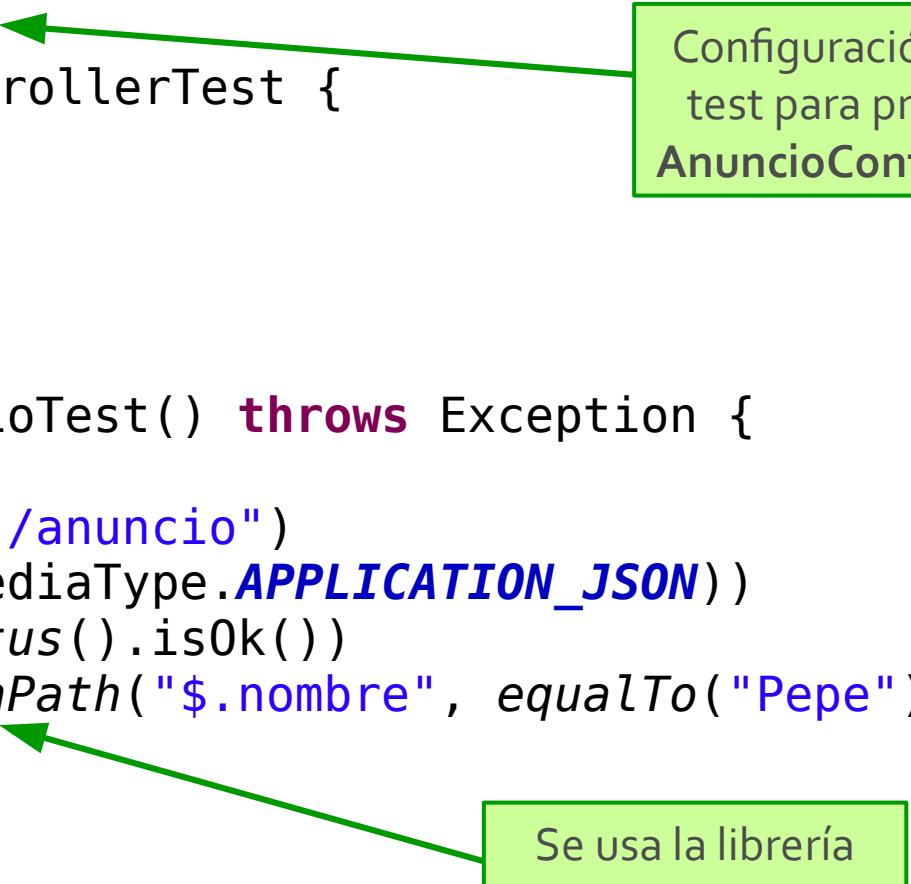
```
@SpringBootTest
@AutoConfigureMockMvc
public class AnuncioControllerTest {

    @Autowired
    private MockMvc mvc;

    @Test
    public void getAnuncioTest() throws Exception {

        mvc.perform(get("/anuncio")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.nombre", equalTo("Pepe")));

    }
}
```



Configuración del test para probar AnuncioController

Se usa la librería jsonPath

- Librerías de testing en el pom.xml

ejem1

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>
</dependencies>
```

# Ejercicio 1

- Implementa el test del controlador de gestión de Items (ItemsController) usando MockMvc
- Código disponible **spring-test-ejer1\_enunciado**



- **Mock de dependencias (servicios)**
  - Lo habitual es que un controlador dependa de uno
  - La librería de testing de Spring facilita la inyección de dobles de esas dependencias



# Testing con Spring

ejem2

```
@SpringBootTest
@AutoConfigureMockMvc
public class UsersControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private UserService userService;

    @Test
    public void getUsersTest() throws Exception {

        List<User> users = Arrays.asList(new User("John"), new User("Peter"));

        when(userService.getUsers()).thenReturn(users);

        mvc.perform(get("/users/")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$", hasSize(2)))
            .andExpect(jsonPath("$[0].name", equalTo("John")));

    }
}
```

# Testing con Spring

ejem2

```
@SpringBootTest
@AutoConfigureMockMvc
public class UsersControllerTest {

    @Autowired
    private MockMvc mvc;

    @MockBean
    private UserService userService;

    @Test
    public void getUsersTest() throws Exception {

        List<User> users = Arrays.asList(new User("John"), new User("Peter"));

        when(userService.getUsers()).thenReturn(users);

        mvc.perform(get("/users/")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$", hasSize(2)))
            .andExpect(jsonPath("$[0].name", equalTo("John")));

    }
}
```

@MockBean crea un mock y lo inyecta como un componente

Configuramos el mock con Mockito

- **Testing de servicios sin dependencias**
  - El servicio es una clase normal
  - Se puede instanciar y probar
  - No hay que usar ninguna herramienta de Spring

ejem2

```
@Service
public class UsersService {

    private List<User> users = Arrays.asList(new User("Pepe"));

    public List<User> getUsers() {
        return users;
    }
}
```

- Testing de servicios sin dependencias

ejem2

```
public class UsersServiceTest {  
  
    @Test  
    public void getUsersTest() throws Exception {  
  
        UsersService service = new UsersService();  
  
        List<User> users = service.getUsers();  
  
        assertThat(users).hasSize(1);  
        assertThat(users.get(0).getName()).isEqualTo("Pepe");  
  
    }  
}
```

# Ejercicio 2

- Haz un test unitario del ItemsRepository
- Código disponible **spring-test-ejer2\_enunciado**



- **Mock de RestTemplate**

- Si el RestTemplate se constuye desde el builder, se puede mockear muy fácilmente
- Eso permite hacer testing unitario de servicios que usan RestTemplate para comunicarse con APIs REST

# Testing con Spring

```
@Service
public class BooksService {

    private RestTemplate restTemplate;

    public BooksService(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build();
    }

    public List<String> getBookTitles(String title) {

        String url = "https://www.googleapis.com/books/v1/volumes?q=intitle:" + title;

        BooksResponse data = restTemplate.getForObject(url, BooksResponse.class);

        List<String> bookTitles = new ArrayList<String>();

        for (Book book : data.items) {
            bookTitles.add(book.volumeInfo.title);
        }

        return bookTitles;
    }
}
```





# Testing con Spring

```
@RestClientTest(BooksService.class)
public class BooksServiceTest {

    @Autowired
    private BooksService service;

    @Autowired
    private MockRestServiceServer booksServer;

    private String jsonResponse = "{...}";

    @Test
    public void bookServiceTest() throws Exception {

        this.booksServer
            .expect(requestTo("https://www.googleapis.com/books/v1/volumes?q=intitle:Java"))
            .andRespond(withSuccess(jsonResponse, MediaType.APPLICATION_JSON));

        List<String> books = this.service.getBookTitles("Java");

        System.out.println(books);

        assertThat(books).hasSize(2);
        assertThat(books).containsExactly("Java a Tope: J2me (java 2 Micro Edition).",
            "Introduccion Al Desarrollo de Programas Con Java");
    }
}
```

# Testing con Spring

```
@RestClientTest(BooksService.class)
public class BooksServiceTest {

    @Autowired
    private BooksService service;

    @Autowired
    private MockRestServiceServer booksServer;

    private String jsonResponse = "{...}";

    @Test
    public void bookServiceTest() throws Exception {

        this.booksServer
            .expect(requestTo("https://www.googleapis.com/books/v1/volumes?q=intitle:Java"))
            .andRespond(withSuccess(jsonResponse, MediaType.APPLICATION_JSON));

        List<String> books = this.service.getBookTitles("Java");

        System.out.println(books);

        assertThat(books).hasSize(2);
        assertThat(books).containsExactly("Java a Tope: J2me (java 2 Micro Edition).",
            "Introduccion Al Desarrollo de Programas Con Java");
    }
}
```

**@RestClientTest**

Inyectamos el mock del servidor REST

Definimos el comportamiento del mock

- **Arquitectura testeable**

- Refactoriza el microservicio de gestión de usuarios para que la lógica de negocio se pueda probar de forma unitaria
- Implementa las pruebas unitarias
- Código disponible **spring-test-ejer3\_enunciado**

