TECNOLÓGICA DEL ORIENTE
INSTITUCIÓN DE EDUCACIÓN SUPERIOR

PRUEBA Y CALIDAD DE SOFTWARE

CREAR UN PROYECTO BÁSICO PARA PRUEBAS DE SISTEMA

CREAR UN PROYECTO BÁSICO PARA PRUEBAS DE SISTEMA

Se trabajará el mismo proyecto usado en las pruebas de integración. La idea es crear un flujo funcional completo: un usuario se registra y luego se consulta, pero esta vez considerando el sistema como una caja negra.

Paso a paso: prueba de sistema de una API REST

Controlador REST (recordatorio)

Figura 1. Código controlador

```java
@RestController
@RequestMapping("/usuarios")
public class UsuarioController {

    @Autowired
    private UsuarioService usuarioService;

    @PostMapping
    public ResponseEntity<Usuario> crear(@RequestBody Usuario usuario) {
        return ResponseEntity.ok(usuarioService.guardar(usuario));
    }

    @GetMapping("/correo/{correo}")
    public ResponseEntity<Usuario> obtener(@PathVariable String correo) {
        return usuarioService.buscarPorCorreo(correo)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
    }
}
```

Prueba de sistema con TestRestTemplate

Figura 2. Prueba de sistema

```java
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class UsuarioSystemTest {

    @Autowired
    private TestRestTemplate restTemplate;

    private static Usuario creado;

    @Test
    @Order(1)
    public void testCrearUsuario() {
        Usuario nuevo = new Usuario();
        nuevo.setNombre("Laura Castillo");
        nuevo.setCorreo("laura@demo.com");

        ResponseEntity<Usuario> respuesta =
            restTemplate.postForEntity("/usuarios", nuevo, Usuario.class);

        assertEquals(HttpStatus.OK, respuesta.getStatusCode());
        creado = respuesta.getBody();
        assertNotNull(creado.getId());
    }
```



```java
    @Test
    @Order(2)
    public void testConsultarUsuarioPorCorreo() {
        ResponseEntity<Usuario> respuesta =
            restTemplate.getForEntity("/usuarios/correo/laura@demo.com", Usuario.class);

        assertEquals(HttpStatus.OK, respuesta.getStatusCode());
        assertEquals("Laura Castillo", respuesta.getBody().getNombre());
    }
}
```

Explicación detallada

- @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT): configura un servidor embebido en un puerto aleatorio, simulando un entorno real de ejecución.

- @TestMethodOrder: asegura que las pruebas se ejecuten en un orden específico, en este caso: primero crear y luego consultar.
- TestRestTemplate: se usa para enviar peticiones POST y GET a los endpoints reales del sistema.
- assertEquals(...): se valida que la respuesta sea exitosa (HTTP 200 OK).
- assertNotNull(...): se asegura que el usuario fue creado correctamente y tiene un ID asignado por la base de datos.