Tests de Integración

Conecta las piezas: el secreto para una app sin fisuras. 🔗

PABLO DEL ÁLAMO

Introducción

¿Alguna vez has hecho cambios en algún punto de tu aplicación, y has temido romper el resto de cosas con las que trabaja en conjunto?

Los tests de integración son ese puente que asegura que las diferentes partes de tu aplicación funcionen perfectmente juntas.

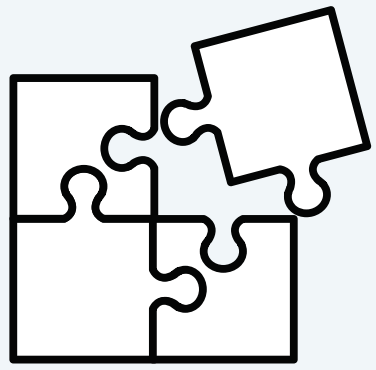
Vamos a ver por qué son esenciales en tu arsenal de testing.

PABLO DEL ÁLAMO

Unit vs. Integration Tests

Mientras que los tests unitarios se centran en partes pequeñas, y funciones específicas de tu código, los tests de integración van un paso más allá.

¡Aquí probamos cómo interactúan ciertos módulos o componentes!



PABLO DEL ÁLAMO

¿Por qué son importantes?

Imagínalo como una orquesta. Cada músico puede tocar su parte a la perfección, pero si no coordinan, ¡el concierto será un desastre!

Aquí es donde entran en juego los tests de integración.

PABLO DEL ÁLAMO

Estrategias generales

Para maximizar su efectividad, sigue un enfoque "Bottom-Up" o "Top-Down", cada uno tiene sus ventajas. ¿Cuál es mejor? Depende de tu proyecto.

PABLO DEL ÁLAMO

Estrategia Bottom-Up

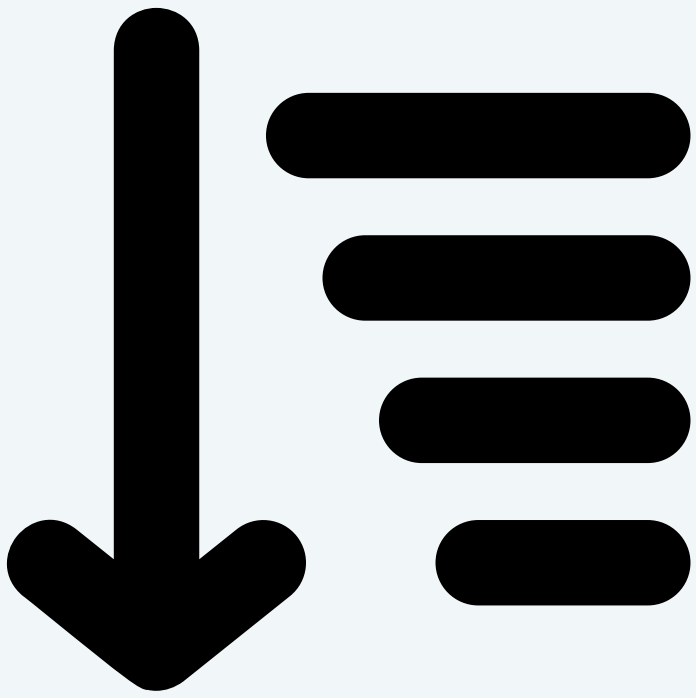
Primero integramos y probamos los componentes de nivel más bajo.

Luego subimos poco a poco hasta completar el sistema.

PABLO DEL ÁLAMO

Estrategia Top-Down

Probamos desde la interfaz arriba, simulando componentes inferiores conforme avanzamos.



PABLO DEL ÁLAMO

Ejecución continua

Los tests de integración son perfectos candidatos para pipelines de CI/CD.

Así garantizas que cada cambio pase por un chequeo exhaustivo.

PABLO DEL ÁLAMO

Ejemplo Práctico: Sistema de Gestión de Usuarios

El sistema tiene una API para:

1. Crear un usuario.
2. Obtener un usuario por ID.

A continuación dejo la estructura del código, para que entiendas cómo van a funcionar los tests de integración.

PABLO DEL ÁLAMO

Controlador (UserController.java)

```java
@RestController
@RequestMapping("/api/users")
public class UserController {

    @Autowired
    private UserService userService;

    @PostMapping
    public ResponseEntity<User> createUser(@RequestBody User user) {
        User createdUser = userService.createUser(user);
        return ResponseEntity.status(HttpStatus.CREATED).body(createdUser);
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
        User user = userService.getUserById(id);
        return ResponseEntity.ok(user);
    }
}
```

Servicio (UserService.java)

```java
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public User createUser(User user) {
        return userRepository.save(user);
    }

    public User getUserById(Long id) {
        return userRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("User not found"));
    }
}
```

Entidad (User.java)

```java
@Entity
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    // Getters and setters
}
```

Repositorio (UserRepository.java)

```java
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
}
```

Tests de Integración

Para este ejemplo, usaremos Spring Boot Test para los tests de integración, pero existen numerosas herramientas que puedes usar, según el lenguaje de programación o framework que estés utilizando.

PABLO DEL ÁLAMO

Clase de Test (UserControllerIntegrationTest.java)

```java
@AutoConfigureMockMvc
@Transactional
public class UserControllerIntegrationTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @Autowired
    private UserRepository userRepository;

    @Test
    public void testCreateUser() throws Exception {
        // Dado un usuario
        User user = new User();
        user.setName("John Doe");
        user.setEmail("john.doe@example.com");

        // Cuando hacemos un POST a /api/users
        mockMvc.perform(post("/api/users")
                        .contentType(MediaType.APPLICATION_JSON)
                        .content(objectMapper.writeValueAsString(user)))
                .andExpect(status().isCreated())
                .andExpect(jsonPath("$.id").exists())
                .andExpect(jsonPath("$.name").value("John Doe"))
                .andExpect(jsonPath("$.email").value("john.doe@example.com"));
    }

    @Test
    public void testGetUserById() throws Exception {
        // Dado un usuario en la base de datos
        User savedUser = userRepository.save(new User("Jane Doe", "jane.doe@example

        // Cuando hacemos un GET a /api/users/{id}
        mockMvc.perform(get("/api/users/" + savedUser.getId())
                        .accept(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(jsonPath("$.id").value(savedUser.getId()))
                .andExpect(jsonPath("$.name").value("Jane Doe"))
                .andExpect(jsonPath("$.email").value("jane.doe@example.com"));
    }
}
```

Explicación

- Configuración:
  - @AutoConfigureMockMvc configura MockMvc para simular peticiones HTTP.
  - @Transactional asegura que los datos insertados en la base de datos durante las pruebas se revierten automáticamente.

- Prueba de Crear Usuario:
  - Se simula un POST con un objeto JSON.
  - Se verifica que la respuesta tenga un código de estado 201 Created y contenga los datos esperados.

PABLO DEL ÁLAMO

- Prueba de Obtener Usuario por ID:
    - Se inserta un usuario en la base de datos directamente.
    - Se simula un GET y se verifica que la respuesta contenga los datos esperados.

PABLO DEL ÁLAMO

Conclusión

Los tests de integración son fundamentales para asegurar que los diferentes módulos de tu aplicación trabajen de manera coherente.

Actúan como un pegamento que garantiza la armonía del ecosistema de tu software, identificando problemas que no se detectan en tests unitarios.

Integrándolos en tu flujo de trabajo, especialmente con herramientas de CI/CD en plataformas como AWS, no solo elevas la calidad sino también la robustez de tus aplicaciones.

Recuerda, un buen test de integración es un paso muy importante hacia un despliegue exitoso y sin sustos. ¡A testear! 🚀

PABLO DEL ÁLAMO

¿Te ha resultado útil?

- Comparte esta guía con tu equipo o amigos desarrolladores.
- Guárdala para tenerla siempre a mano.
- ¡Dale un like o comenta si tienes preguntas!

PABLO DEL ÁLAMO