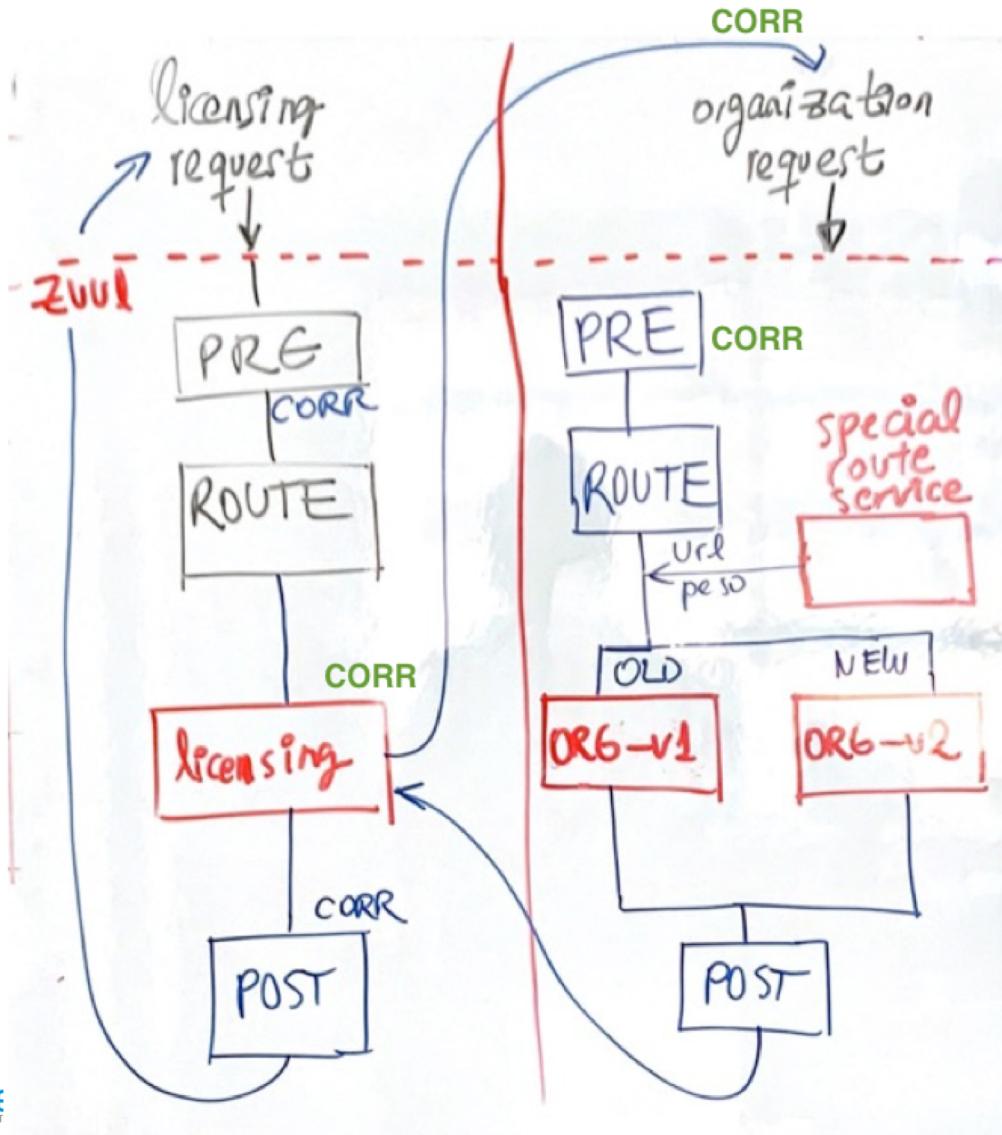


Arquitectura e Implementación de Microservicios

Seguridad en microservicios con Oauth2/JWT



JWT y oauth2

- Oauth2 no hay estandar para token
 - Json web tokens: JWT
- JWT estructura para oauth2 tokens
- Caracteristicas:
 - Pequeños
 - Base64. pasados por http url,header,parameters
 - Firmados criptograficamente
 - Firmado por authentication server
 - Garantiza no esta alterado
 - Auto-contenido
 - Microservicio sabe token es valido con la firma.
 - no llama a authentication server para validar
 - Token validado solo por el microservicio
 - Simplifica autenticacion
 - Extensible
 - Informacion adicional al generar token
 - Microservicio decodifica y recupera informacion

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09o1PSyXnrXCjTwXyr4BsezdI1AVTmud2fU4

JWT

- Usos:
 - Autorizacion
 - comun
 - Intercambio de informacion
 - No alterado
- Estructura
 - Header
 - Tipo
 - Algoritmo firma
 - Payload
 - Claims: data acerca del usuario
 - Signature/firma
 - Verifica contenido no alterado
 - Cifrado utiliza una clave

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

header

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```

payload

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret)
```

signature

Ciprado en JWT

- 2 tipos de cifrado
 - Asimetrico
 - Simetrico

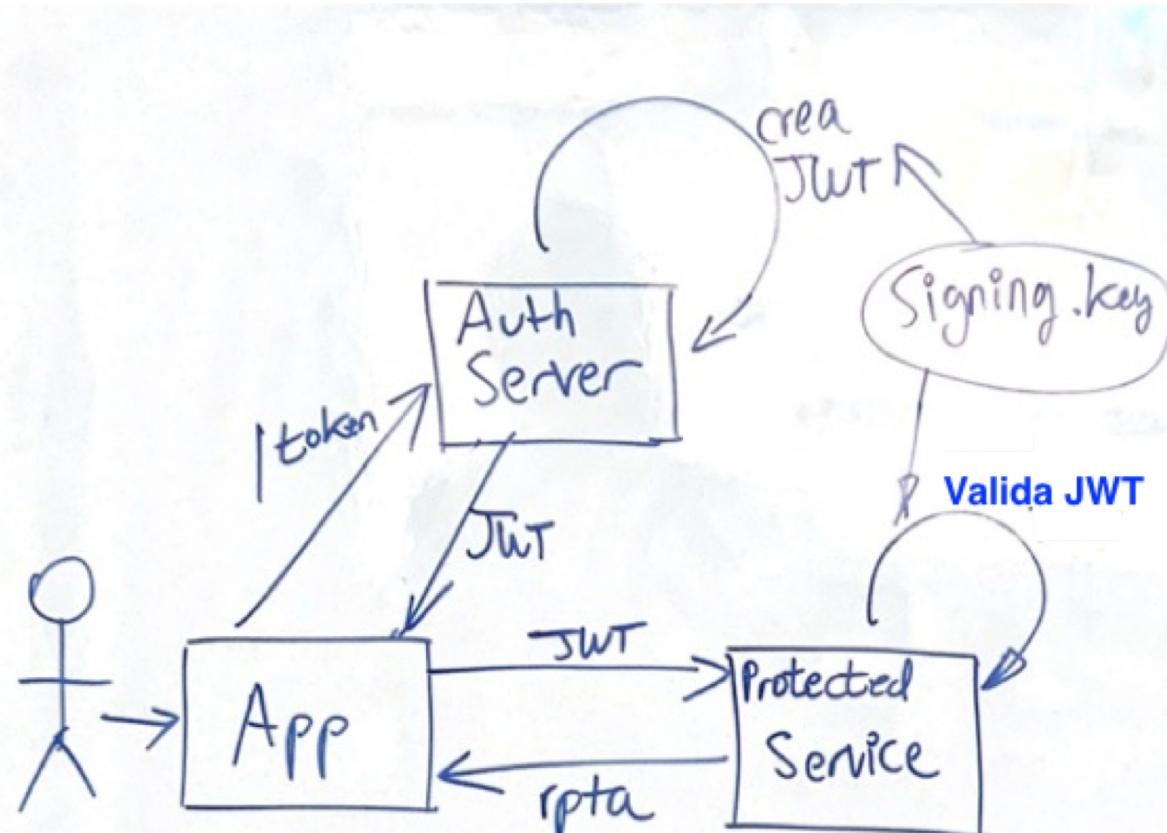


JWT

- 3 cadenas en base64 separados por .

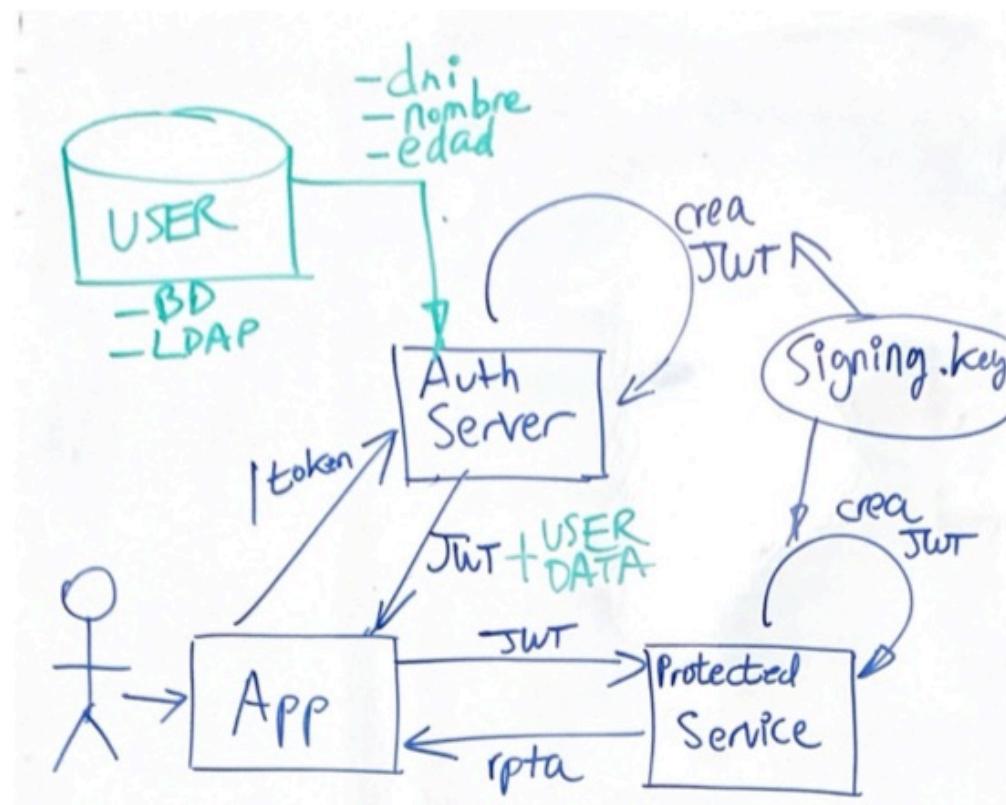
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09o1PSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
- Jwt firmados: no pueden alterarse , pero si leidos
 - no poner secretos en header y payload
 - Token encriptados: JWE
 - <https://tools.ietf.org/html/rfc7516>

Flujo de autenticación usando JWT

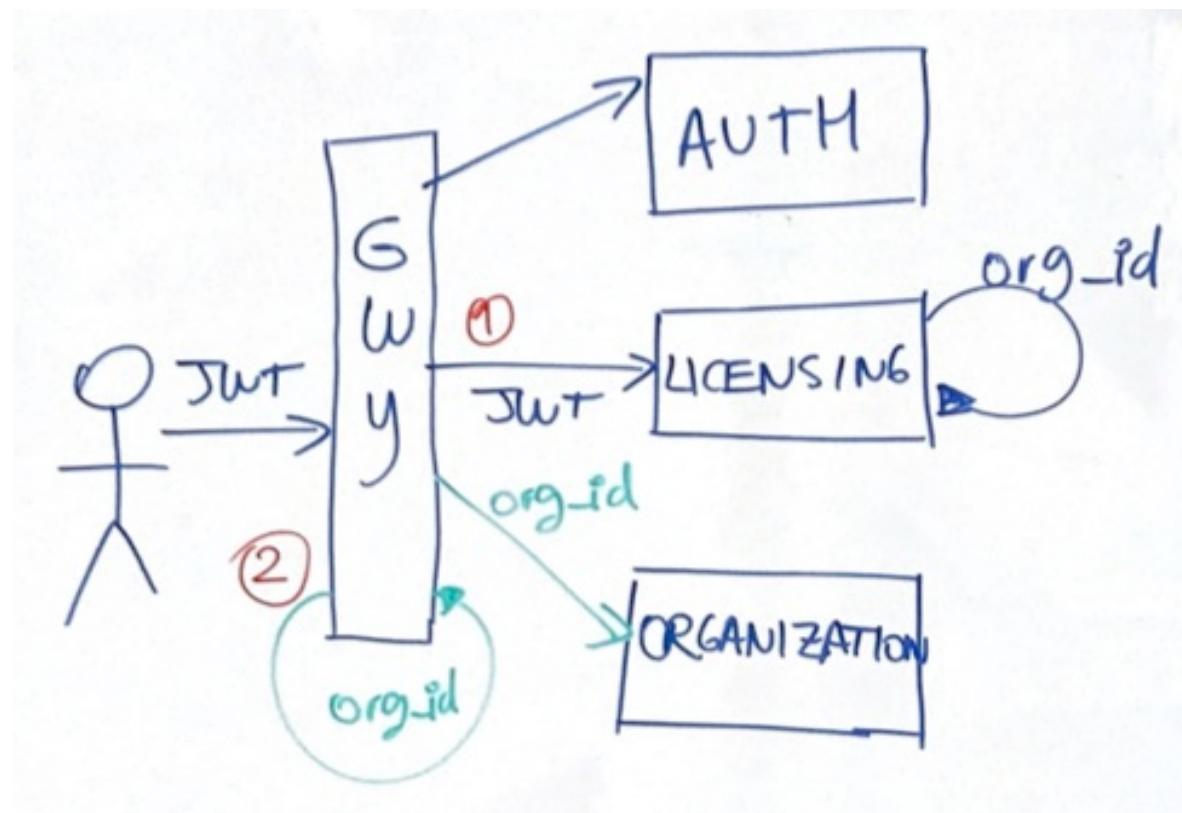


Laboratorio JWT

Autenticación con información adicional



Obtener campo adicional



Testing en microservicios

Importancia

- Comunicación remota es mas importante en microservicios que en monolitos
- Monolito: pocos servicios externos
 - base de datos, servicios web , emails, etc
- Microservicios:
 - Comunicación interprocesos es central ya que están distribuidos
 - Evolucionar: APIs modificadas
 - Esencial escribir test que verifiquen:
 - Funciones internas al servicio
 - Interacción entre servicios

Testing

- No usar método main
- Desventajas:
 - Difícil de leer clase código de pruebas
 - Consideraciones como iniciar la prueba
 - servicio web esta arriba?
 - Cómo se puede probar un método en isolación
 - Clase tiene dependencias en otras clases
 - Dependencia en webservice, base de datos
 - pueden no estar disponibles en el ambiente de prueba.

```
package com.nathanbak.test.main;

public class Application {

    protected Application() { }

    public static void main(String[] args) {
        String string = args[0];
        Application application = new Application();
        try {
            application.run(string);
        } catch (Exception e) {
            System.out.println(e.getMessage());
            System.exit(1);
        }
        System.exit(0);
    }

    protected void run(String string) throws Exception { }

    protected void logException() { }
}
```

Tipos de pruebas

- Unit Tests
 - Prueba de una unidad (e.g. Un método)
 - Permite Regression Testing
 - Permite Integration Unit Tests
 - (e.g. Método de lectura de base de datos con interacción con una base de datos)
- Integration Tests - pruebas de interacciones de componentes.
 - e.g. probar interaccion entre 2 microservicios para completar un request
- System Tests – pruebas por conformidad con sus requerimientos.
 - Blackbox testing
 - Tipos:
 - funcionality test
 - Performance test
 - Security test
 - Installation tests
- Acceptance Tests - pruebas en ambiente casi idéntico a producción.
 - Las ultimas pruebas antes de que el cliente acepte el sistema.

JUnit (<http://junit.org>)

- Uno de los frameworks xUnit más popular
 - Multiples lenguajes
- Ejecutar desde el command prompt o GUI
- Integrado con IDEs (Eclipse, NetBeans)
- Version 4
 - Muy estable
 - Java 7-
- Version 5
 - Compatible con Junit4
 - Sólo Java 8+
- Spring boot por defecto usa junit4

1. Assertions

- Probar Outputs esperados

```
import org.junit.Test;//1
import static org.junit.Assert.*;
public class MoneyTest
{
    //2
    @Test public void simpleAdd()
    {
        Money s12 = new Money(12, "SOL");//3
        Money s2 = new Money(2, "SOL");

        Money expected = new Money(14, "SOL");//4

        Money result = s12.add(s2);//5

        assertEquals(expected, result);//6
    }
}
```

2. Fixtures

A veces las pruebas ejecutan con objetos communes

Para iniciar el fixture

Usar @org.junit.Before

Para limpiar el fixture

Usar @org.junit.After

Para inicializaciones costosas, para iniciar algo sólo una vez por varias pruebas

Usar @org.junit.BeforeClass y @org.junit.AfterClass para anotar un método estático

```
private WebServiceClient client;

//1
@BeforeClass public static void initialize()
{
    startWebService();
}
//2
@AfterClass public static void initialize()
{
    stopWebService();
}
//3
@Before public void setUp()
{
    client = new WebServiceClient();
}

@Test testClient()
{
    String success = client.consumeWebService(); //4
    assertEquals("true", success);
}
```

- **Test Runners**

- Sirve para ejecutar las pruebas
- Ademas genera un reporte con resultados de la prueba

- **Test-suite**

- Se puede ejecutar una colección de pruebas juntas
- Se puede hacer inicializaciones antes de ejecutar el “suite”
 - BeforeClass y AfterClass , para poblar base de datos, subir/bajar servidor JMS

```
@RunWith(Suite.class) //1
@Suite.SuiteClasses(
{
    MoneyTest.class,
    MoreTests.class,
}) //2
public class AllTests
{
//3
}
```

Unit Testing de Componentes Complejos

- Dependencias en recursos externos.
 - e.g. Base de datos o servicio web
- Dependencias en un ambiente de ejecución.
 - e.g. Contenedores para Servlets y EJBs
- Test Doubles.
 - <http://xunitpatterns.com/>
 - Dummy – usado sólo para llenar parámetros
 - e.g. Null, "", new Object()
 - Fake – objetos con comportamiento similar
 - e.g. Base de datos en memoria
 - Stub – objeto que responde con respuestas preparadas para la prueba.
 - Mock – objetos programados con expectativas de las llamadas que esperan recibir.

Considerar:

```
//1
class DateUtils
{
    public static boolean isWeekday()
    {
        Date date = new Date();
        int dayOfWeek = date.getDay();
        return (dayOfWeek != 0 && dayOfWeek != 6); //2
    }
}
```

```
class DateUtils
{
    private Clock clock;//1
    public static void setClock(Clock clock)
    {
        this.clock = clock;
    }
    public static boolean isWeekday()
    {
        Date date = clock.getDate(); //2
        int dayOfWeek = date.getDay();
        return (dayOfWeek != 0 && dayOfWeek != 6);
    }
}

interface Clock
{
    public Date getDate(); //3
}
```

```
//stub con respuestas preparadas  
  
//1  
ClockStub implements Clock  
{  
    Date date;  
    public ClockStub(Date date)  
    {  
        this.date = date;  
    }  
  
    public Date getDate()  
    {  
        return date;//2  
    }  
}  
  
//testclass of dateutils  
public void testIsWeekday()  
{  
    Date tuesday = ...;//get Tuesday  
  
    DateUtils.setClock(new ClockStub(tuesday));//3  
  
    assertTrue(DateUtils.isWeekday());//4  
}
```

Mockito (<http://mockito.org>)

- No tenemos que codificar los mocks y stubs

```
import static org.mockito.Mockito.*;  
//metodo estatico  
List mockedList = mock(List.class); //1  
  
o  
//para usar annotations  
MockitoAnnotations.initMocks(testClass); //1  
@Mock List mockedList //2
```

```
//usar como stub  
  
LinkedList mockedList = mock(LinkedList.class); //1  
when(mockedList.get(0)).thenReturn("first"); //2  
System.out.println(mockedList.get(0)); // imprime "first" //3  
System.out.println(mockedList.get(99)); // imprime "null" //4  
when(mockedList.get(anyInt())).thenReturn("element"); //5  
System.out.println(mockedList.get(99)); // imprime "element" //6
```

- Validaciones mock:
 - validamos cantidad interacciones/colaboracion del mock

```
verify(mockedList).get(anyInt()); //1
```

```
Verify(mockedList, times(2)).get(1); //2
```

```
verify(mockedList, never()).add("two"); //3
```

```

//Class to test

class HolaServlet extends HttpServlet {
//1
    doGet(HttpServletRequest request, HttpServletResponse
response) {
        String name =
request.getParameter("name");//2
        response.getWriter().print("Hola "+name);//3
    }
}

//Testclass
//1
private HolaServlet holaServlet;
//2
@Mock private HttpServletRequest requestMock;
@Mock private HttpServletResponse responseMock;
//3
@Before protected void setUp() {
    holaServlet = new HolaServlet();
}
@Test public void testDoGet() {
    // iniciar
    StringWriter out = new StringWriter();
    //4
    when(requestMock.getParameter("name")).thenReturn("Colin");
    //5
    when(responseMock.getWriter()).thenReturn(out);

    // 6 .la prueba
    holaServlet.doGet(request, response);

    // validaciones
    verify(requestMock).getParameter("name");//7
    verify(responseMock.getWriter());

    assertEquals("Hola Colin", out.toString());//8
}

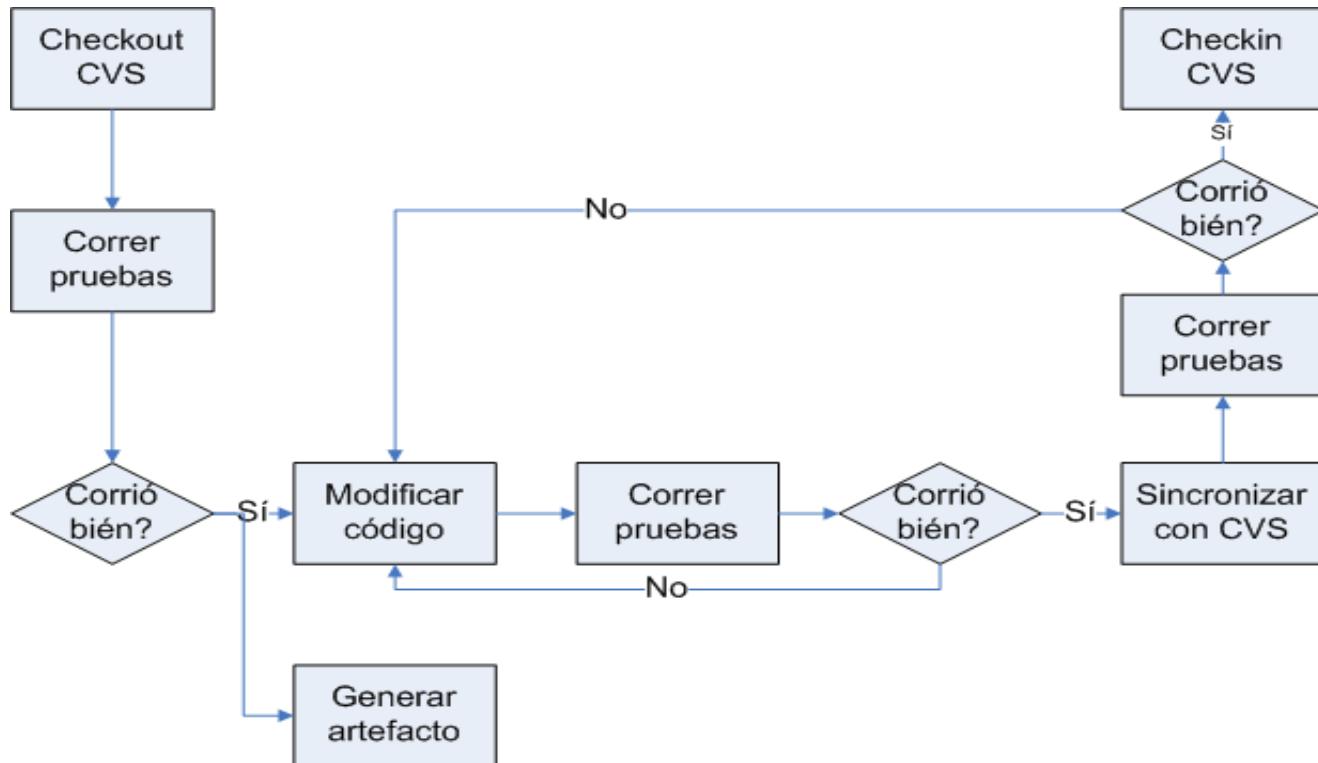
```

Buenas practicas

- Que cosas probamos?
 - Método público
 - Camino feliz” e inputs errados
Una prueba por cada caso
- Dónde poner las pruebas?
Pruebas en otro directorio de fuentes, pero en el mismo paquete que la clase siendo probada.
- No crear métodos públicos sólo para pruebas
- Marcar inputs como final si no deberían ser modificados

```
//1
final String nombre = "Colin";
User user = dao.getUser(nombre);
assertEquals(nombre, user.getNombre());
```
- Pruebas no deberían tener dependencias en otras pruebas

cuando ejecutar pruebas?



- Antes de implementar modificaciones
- Después de modificaciones
- Antes de compartir el código
- Antes de producir un desplegable (e.g. war o jar)

Spring testing

- Spring provee clases para ayudar a las pruebas
- Se integra con Mockito y Junit

Laboratorio

como escribir pruebas en microservicios usando Spring

Ciclo de vida de maven

Default Lifecycle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	package
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	install
generate-test-resources	deploy
processs-test-resources	

Spring Test Slices

- Spring provee anotaciones testing específicas a un slice o capa de tu aplicación:
 - @jsontest, frontend
 - @webmvc, capa controller
 - @datajpatest, capa de datos
 - @restclienttest, servicios remotos