

BLOG.JOEDAYZ.PE

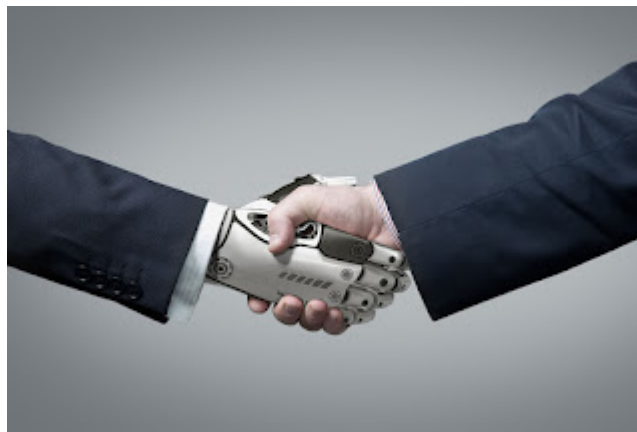
CORAZÓN DE JOE

PÁGINA PRINCIPAL



XORCERY SAMPLES - GREETER

on **noviembre 13, 2023** in **xorcery** with **No hay comentarios.**



We are going to explain our first example with Xorcery called **Greeter**, which you will find in the following [Github repository](#).

With **Xorcery** we can implement the API of service as a REST API (using [JSON-API](#) as a content type) for request/response needs or reactive stream [web sockets](#) (server publishers or subscribers) for streaming needs (like event sourcing or projections or log collection, etc.).

Important fact: The entire implementation of Xorcery uses the Jakarta EE APIs and important libraries in the Java world. Xorcery for example uses HK2 a lightweight and dynamic injection framework.

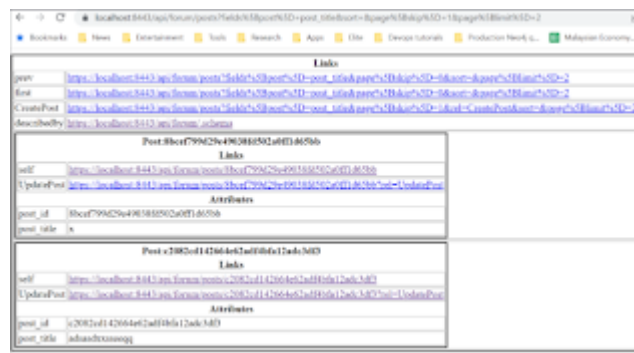
WHY JSON:API



The election in JSON-API and JSONSchema is that then makes it possible to create a REST client, and not an HTTP client, which applications can use.

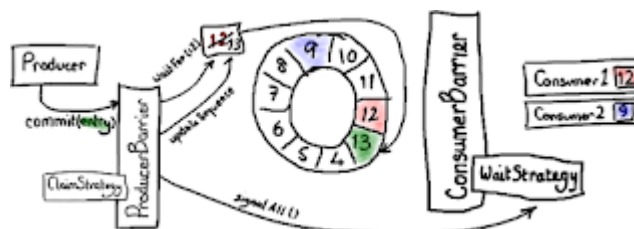
When done properly it leads to the situation that all application code interactions with REST clients are one of these two: follow links and submit forms. That's it. There's no constructing URLs, or figuring out what method to use, or any of that. This is all happening behind the scenes as it is defined by JSON: API and JSONSchema on the server. Application code just needs to bother with what link rel's it cares about, what information it needs to extract from resources in JSON: API, and how to submit forms (which know the URI and method to use).

We currently support URI templates in the JSON:API schemas and HTML sandbox. This way it is easier to test the API in a browser, as well as simplify filling out the URLs as a form.



Free HTML Sandbox stuff

WHY JETTY WEBSOCKETS + DISRUPTOR



Xorcery makes a custom variation of the ReactiveStreams API with WebSockets to send events as headers plus bytes, as well as an integration with the Disruptor API. In the xorcery source code you will find services where the architecture is applied such as **publisher+subscriber**, **metrics events publisher+subscriber**, and **domain event publisher+subscriber** (subscriber maps into hashmap “database”).

GREETER

In the GitHub repository, you will find xorcery-examples where a modular project has been created.

Here you will find the BOM, common dependencies, and plugins.

```
<?xml version="1.0" encoding="UTF-8"
<project xmlns="http://maven.apache.org/maven-v4"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/maven-v4/xsd/maven-4.0.0.xsd"
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.exoreaction.xorcery</groupId>
  <artifactId>xorcery-examples</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <modules>
    <module>xorcery-example-greeter</module>
  </modules>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

    <xorcery.version>0.62.2</xorcery.version>
    <hk2.version>3.0.5</hk2.version>
    <jersey.version>3.1.3</jersey.version>
    <slf4j.version>2.0.7</slf4j.version>
    <log4j.version>2.21.1</log4j.version>

    <junit.version>5.10.0</junit.version>
    <junit.platform.version>1.9.0</junit.platform.version>
  </properties>

  <dependencyManagement>
    <dependencies>
      <dependency>
        <groupId>com.exoreaction.xorcery</groupId>
        <artifactId>xorcery-bom</artifactId>
        <version>1.0-SNAPSHOT</version>
        <scope>import</scope>
      </dependency>
    </dependencies>
  </dependencyManagement>
</project>
```

```
<version>${xorcery.version}
<type>pom</type>
<scope>import</scope>
</dependency>
<dependency>
  <groupId>org.apache.logging
  <artifactId>log4j-bom</arti
  <version>${log4j.version}</
  <type>pom</type>
  <scope>import</scope>
</dependency>

<dependency>
  <groupId>org.hamcrest</grou
  <artifactId>hamcrest</artif
  <version>2.2</version>
  <scope>test</scope>
</dependency>
</dependencies>

</dependencyManagement>

<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven
        <artifactId>maven-compile
        <version>3.11.0</version>
        <configuration>
          <annotationProcessorPat
            <path>
              <groupId>org.glassf
              <artifactId>hk2-met
              <version>${hk2.vers
            </path>
```

```
        </annotationProcessorPa
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven
    <artifactId>maven-depende
    <version>3.6.1</version>
</plugin>
<plugin>
    <groupId>org.apache.maven
    <artifactId>maven-surefir
    <version>3.2.2</version>
    <dependencies>
        <dependency>
            <groupId>me.fabricior
            <artifactId>maven-sur
            <version>1.2.1</versi
        </dependency>
    </dependencies>
    <configuration>
        <reportFormat>plain</re
        <consoleOutputReporter>
            <disable>true</disabl
        </consoleOutputReporter>
        <statelessTestsetInfoRe
    </configuration>
</plugin>
</plugins>
</pluginManagement>
</build>
<repositories>
    <repository>
        <id>cantara-releases</id>
        <name>Cantara Release Reposit
        <url>https://mvnrepo.cantara.
    </repository>
```

```
<repository>
  <id>cantara-snapshots</id>
  <name>Cantara Snapshot Reposi
  <url>https://mvnrepo.cantara.
</repository>
</repositories>

<distributionManagement>
  <repository>
    <id>cantara</id>
    <name>Cantara Release Reposit
    <url>https://mvnrepo.cantara.
  </repository>
  <snapshotRepository>
    <id>cantara</id>
    <name>Cantara Snapshot Reposi
    <url>https://mvnrepo.cantara.
  </snapshotRepository>
</distributionManagement>
</project>
```

It is important that you place the Cantara **repositories** and **distributionManagement** so that Xorcery dependencies can be downloaded without problems.

The test report is presented in a fancy tree output thanks to <https://github.com/fabriciorby/maven-surefire-junit5-tree-reporter>

GREETER

This project has xorcery-examples as its parent project.

```
<parent>
  <groupId>com.exoreaction.xorcery.
  <artifactId>xorcery-examples</art
  <version>1.0-SNAPSHOT</version>
</parent>
```



Then we add the core dependencies:

```
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-core</artifac
</dependency>
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-runner</artif
</dependency>
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-metadata</art
</dependency>
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-configuration
</dependency>
<dependency>
  <groupId>com.exoreaction.xorcery<
```



```
<artifactId>xorcery-json</artifactId>  
</dependency>
```

The dependencies for REST API:

```
<dependency>  
  <groupId>com.exoreaction.xorcery<  
  <artifactId>xorcery-jsonapi-jaxrs  
</dependency>  
<dependency>  
  <groupId>com.exoreaction.xorcery<  
  <artifactId>xorcery-jersey-server  
</dependency>  
<dependency>  
  <groupId>com.exoreaction.xorcery<  
  <artifactId>xorcery-handlebars</a  
</dependency>  
<dependency>  
  <groupId>com.exoreaction.xorcery<  
  <artifactId>xorcery-status-server  
</dependency>
```



Xorcery works with Jetty and Jersey as a Jakarta JAX-RS implementation.

The dependencies for service integration and implementation of reactive streams:

```
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-jersey-client
</dependency>
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-reactivestrea
</dependency>
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-reactivestrea
</dependency>
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-neo4j</artifa
  <scope>compile</scope>
  <exclusions>
    <exclusion>
      <groupId>org.glassfish.jersey
      <artifactId>jersey-container-
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-domainevents-
</dependency>
```




Dependencies for Logging:

```
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-log4j</artifa
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j
  <artifactId>log4j-core</artifactI
</dependency>
<dependency>
  <groupId>org.apache.logging.log4j
  <artifactId>log4j-slf4j2-impl</ar
</dependency>
```



Integration with Junit:

```
<dependency>
  <groupId>com.exoreaction.xorcery<
  <artifactId>xorcery-junit</artifa
  <scope>test</scope>
</dependency>
```



Finally, we add the profile to use `jpacakge` introduced in Java 14 and create the image of an application and install it.

```
<profiles>
  <profile>
    <id>jpackge</id>
    <activation>
      <os>
        <name>linux</name>
        <arch>amd64</arch>
      </os>
    </activation>
    <build>
      <plugins>
        <plugin>
          <groupId>org.apache.maven
          <artifactId>maven-depende
          <executions>
            <execution>
              <id>copy-dependencies
              <phase>package</phase>
              <goals>
                <goal>copy-dependen
              </goals>
              <configuration>
                <includeScope>compi
                <outputDirectory>${
                <overWriteReleases>
                <overWriteSnapshots
                <overWriteIfNewer>t
              </configuration>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```

```
</executions>
</plugin>
<plugin>
  <groupId>org.apache.maven
  <artifactId>maven-antrun-
  <executions>
    <execution>
      <id>copy-modularized-
      <phase>package</phase>
      <goals>
        <goal>run</goal>
      </goals>
      <configuration>
        <target>
          <copy file="${pro
            tofile="${proje
              overwrite="true
          </target>
        </configuration>
      </execution>
    </executions>
  </plugin>
  <plugin>
    <artifactId>maven-resourc
    <version>3.3.1</version>
    <executions>
      <execution>
        <id>copy-app-resource
        <phase>package</phase>
        <goals>
          <goal>copy-resource
        </goals>
        <configuration>
          <outputDirectory>${
          <resources>
            <resource>
```

```
        <directory>src/
        <filtering>true
    </resource>
</resources>
</configuration>
</execution>
</executions>
</plugin>
<plugin>
    <groupId>com.github.akman
    <artifactId>jpackage-mave
    <version>0.1.5</version>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>jpackage</goal>
            </goals>
            <configuration>
                <resourcedir>${proj
                <input>${project.bu
                <mainjar>lib/${proj
                <mainclass>com.exor

                <name>greeter</name>
                <appversion>${proje
                <copyright>Copyrigh
                <description>Descri
                <vendor>eXOReaction
                <installdir>/opt/ex
                <javaoptions>-Dfile
                <dest>${project.bui
            </configuration>
        </execution>
    </executions>
</plugin>
```

```
    </plugins>
  </build>
</profile>
</profiles>
```

At **GreeterResourceTest** we use the Xorcery extension for the configuration that tests that use Xorcery need via `@RegisterExtension`

```
public class GreeterResourceTest {

    @RegisterExtension
    static XorceryExtension xorceryEx
        .configuration(ConfigurationB
            .addYaml(String.format(" "
                                    jetty.server.ht
                                    jetty.server.ss
                                    " " , Sockets.ne
            )
        .build();
```



We tested the GET first of a Rest API by instantiating an Xorcery client.

```
@Test
void updateGreeting() throws Except
```

```
Configuration configuration = xor
URI baseUri = InstanceConfigurati
Client client = xorceryExtension.

{
    String content = client.target(
        .path("/api/greeter")
        .request()
        .get(String.class);
    System.out.println(content);
}

}
```

This is a configuration where we indicate that we want to generate an **SSL certificate** for the local hostname, the **REST API resources**, and the **log4j2 configuration** are declared. Its configuration is located in **resources/xorcery.yaml**.

```
instance.name: "greeter"
instance.home: "{{ SYSTEM.jpackage_
jpackage.app: "{{ SYSTEM.jpackage_a

# So that we can generate a SSL cer
instance.domain: local

# Add local convenience names for y
certificates:
```



```
dnsNames:
- localhost
- "{{ instance.host }}"
ipAddresses:
- 127.0.0.1
- "{{ instance.ip }}"

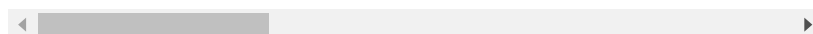
# REST API resources
jersey.server.register:
- com.exoreaction.xorcery.example

log4jpublisher.enabled: false
log4jsubscriber.enabled: false

log4j2.Configuration:
```

Extract from the original

The service **GreeterApplication** permits queries and domain events that are generated and projected into the database.



```
@Service
@Named(GreeterApplication.SERVICE_T
public class GreeterApplication {
    public static final String SERVIC
```

```
private final DomainEventPublishe
private final DomainEventMetadata
private final GraphDatabase graph

@Inject
public GreeterApplication(DomainE
    GraphDatabase graphDatabase)

    this.domainEventPublisher = dom
    this.graphDatabase = graphDatab
    this.domainEventMetadata = new
        .add("domain", "greeter")
        .build());
}

public CompletionStage<String> ge

    return graphDatabase.execute("M
        Map.ofEntries(entry("id
        .thenApply(r ->
        {
            try (GraphResult result =
                return result.getResult
            } catch (Exception e) {
                throw new CompletionExc
            }
        }));
}
}
```

In **GreeterResource** see a full cycle of creating domain events, publishing, then subscribing with `Neo4jDomainEventsService` and projecting into the database, which then can be read through the `GraphDatabase` wrapper.

Reads

```
@Path("api/greeter")
public class GreeterResource extend

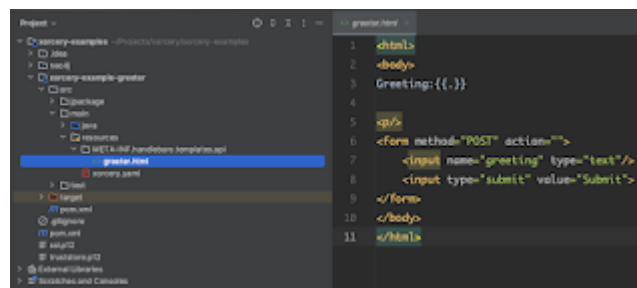
    private GreeterApplication applic

@Inject
public GreeterResource(GreeterApp
    this.application = application;
}

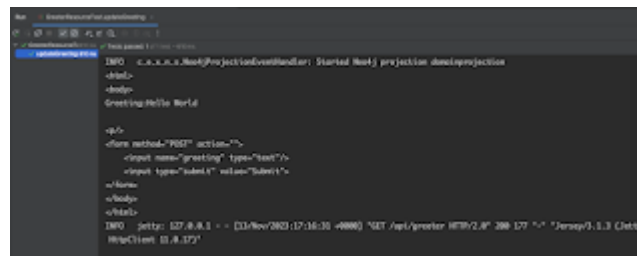
@GET
public CompletionStage<Context> g
    return application.get("greetin
    {
        if (t != null)
        {
            LogManager.getLogger(getCla
            return "";
        } else
        {
            return g;
        }
    }).thenApply(Context::newContext
}
```

}

If we run the test we will see that the call to **/api/greeter** returns the content of the API HTML template:



Tests result:



Writes

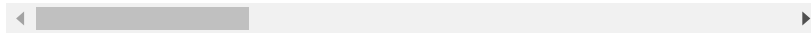
We now test publishing a domain event and projection to neo4j. ▶

```
// Writes
public CompletionStage<Metadata> ha
```

```
Metadata.Builder metadata = new D
    .timestamp(System.currentTimeMillis)

try {
    DomainEvents domainEvents = (Do
    Metadata md = metadata.add("com
    return domainEventPublisher.pub
} catch (Throwable e) {
    return CompletableFuture.failed
}
}

private DomainEvents handle(UpdateG
    return DomainEvents.of(new Update
}
```



We add the necessary domain event and command.

```
public record UpdateGreeting(String
}
```



```
public record UpdatedGreeting(String  
    implements DomainEvent {  
}
```



Finally, if we want to see the HTML sandbox generated thanks to JSON:API and JSONSchema, we have to add the Main.java to use the runner.

```
package com.exoreaction.xorcery.exa  
  
public class Main {  
  
    public static void main(String[]  
  
        com.exoreaction.xorcery.runner.  
    }  
  
}
```



```
2023-11-13 12:24:18,506  
[RunLevelControllerThread-  
1699896252353] INFO
```

c.e.x.j.s.JettyLifecycleService: Started Jetty server

2023-11-13 12:24:18,507

[RunLevelControllerThread-1699896252353] DEBUG

c.e.x.c.RunLevelLogger: Reached run level 18

2023-11-13 12:24:18,507

[RunLevelControllerThread-1699896253228] DEBUG

c.e.x.c.RunLevelLogger: Reached run level 19

2023-11-13 12:24:18,508

[RunLevelControllerThread-1699896253228] DEBUG

c.e.x.c.RunLevelLogger: Reached run level 20

2023-11-13 12:24:18,509 [main] DEBUG

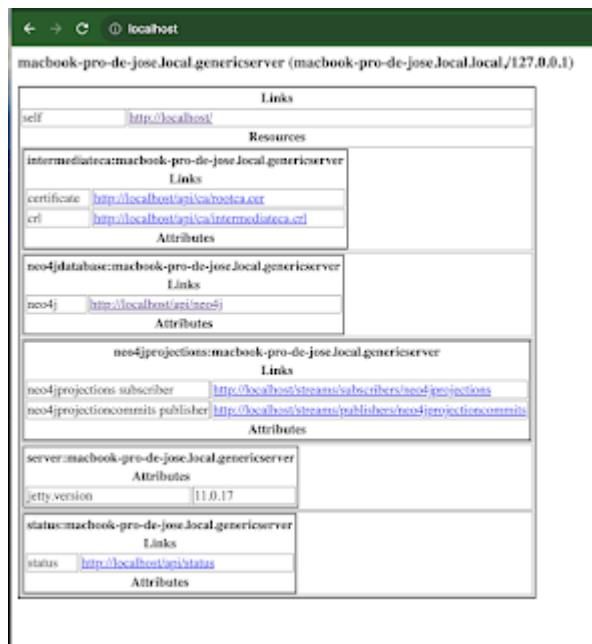
macbook-pro-de-jose.local.genericserver

c.e.x.c.Xorcery: Services:

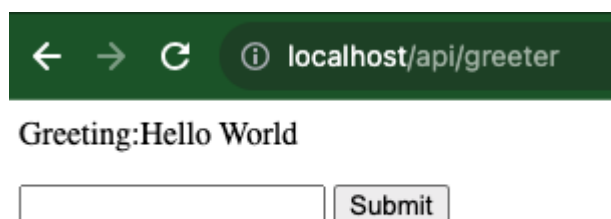
Note: Observe how xorcery executes the level settings when launching the project:

- * 0: Configuration refresh
- * 2: Certificate refresh
- * 4: Servers
- * 6: Server publishers/subscribers
- * 8: Client publishers/subscribers
- * 20: DNS registration

The HTML sandbox:



If we want to publish the domain event and project it to neo4j we can try this URL **<http://localhost/api/greeter>**



In subsequent posts, we will continue examining more examples of using Xorcery.

Enjoy!

Jose

Share:   

RELATED POSTS:



Xorcery implements Reactive Streams - Parte 1



Xorcery uses Jetty & Jersey and mTLS implementation and JWT support



Xorcery implements Reactive Streams - Parte 1



Xorcery Samples - Greeter



Xorcery usa Jetty & Jersey e implementación de mTLS y soporte a JWT

[Página Principal](#)[Entrada antigua](#)

0 COMENTARIOS:

PUBLICAR UN COMENTARIO



Escribir comentario

JOEDAYZ.PE

[Cursos](#)

POPULAR POSTS



Feliz Navidad 2012

Feliz Navidad a todos los que siguen mi blog, a mis amigos, colaboradores, clientes, alumnos. En verdad les doy las gracias por sus saludo...



Xorcery implementa Reactive Streams -

Parte 1

¿Qué es Reactive Streams?
Para poder entender como nos permite Xorcery trabajar

ABOUT



CATEGORIES

#github #java (1)

#guatejug (1)

#historiasdeprogramador (1)

aaii (1)

academia web (2)

acr (1)

Agile (2)

aks (2)

con reactive streams ,
tenemos que saber que es
Reactive ...

BLOG ARCHIVE

▼ 2023 (15)

▼ noviembre (3)

Xorcery Samples -
Greeter

Xorcery
implements
Reactive
Streams - Parte
1

Xorcery
implementa
Reactive
Streams - Parte
1

► octubre (5)

► agosto (2)

► junio (2)

► febrero (1)

► enero (2)

► 2022 (7)

► 2021 (30)

► 2020 (31)

► 2019 (15)

► 2018 (22)

► 2017 (23)

android (3)

angular (11)

AniversarioJoeDayz (2)

apostle (1)

asdf (1)

ASP.NET Core (3)

aspnetcore (4)

aws-ecs (1)

azure (4)

azure-devops (2)

blaze-persistence (1)

blockchain (1)

BluestarEnergy (3)

BMS (2)

bootstrap (1)

Camino Neocatecumenal
(4)

CEVATEC (1)

cide (1)

cloudkarafka (2)

code igniter (2)

code2cloud (1)

codeigniter (1)

comparabien.com (1)

computacion (1)

continuos integration (1)

CoronaVirus (1)

cuba (1)

► **2014** (5)

► **2013** (21)

► **2012** (28)

► **2011** (44)

► **2010** (28)

► **2009** (27)

► **2008** (20)

► **2007** (16)

► **2006** (11)

► **2005** (6)

cursos (1)

darkside (1)

datagrip (1)

deltaspikes (1)

dew (1)

docker (1)

DulceAmorPeru (1)

e-commerce (1)

English (1)

EntityFramework (2)

EPEUPC (3)

eureka (2)

evangelios (1)

eventos (3)

facebook (1)

familia (2)

farmaciaperuanas (1)

firebase (2)

firebase-admin (1)

flutter (2)

functions (1)

gcp (1)

git (1)

github (2)

google-format (1)

google-style (1)

grails (5)

groovy (3)

hangouts (1)**highchart-export-server**
(2)**huacho** (1)**hudson** (1)**hyperledger-composer** (1)**hyperledger-fabric** (1)**i-educa** (1)**iBATIS** (2)**icescrum** (1)**informatica** (1)**Intigas** (1)**ITP_JAVA** (1)**jakartae** (4)**jakartae10** (1)**JasperReports** (1)**java** (3)**JavaCard** (1)**JavaDayUNI** (1)**JavaOne** (1)**jhipster** (2)**jmeter** (1)**joedayz** (46)**JOERP** (4)**jpa** (1)**jquery** (1)**kafka** (3)**kotlin** (2)

Kubernetes (3)**lombok** (1)**m2eclipse** (1)**mac** (2)**Matt Raible** (1)**Maven** (3)**microprofile** (6)**microprofile-jwt**
jakartae (2)**microprofile-jwt jdbc-**
realm jakartae (1)**microprofile-jwt jdbc-**
realm payara (1)**microservicios** (1)**Ministerio del Interior** (1)**MJN** (5)**móvil** (1)**mysql** (1)**namespaces** (1)**navidad** (1)**NET** (4)**Nextel** (1)**Novell** (1)**ocjp** (1)**Opentaps** (2)**Oracle** (1)**oraclecloud** (1)**oraclefunctions** (1)**oracleopenworld** (1)

OSUM (1)**OSX** (1)**p6spy** (1)**Payara** (5)**personal** (1)**perujug jconfperu joedayz**
(2)**php** (1)**play** (1)**PMP** (1)**podcasts** (1)**PostgreSQL** (8)**programacion** (1)**pubsub** (1)**PUCP** (4)**quadim** (2)**quarkus** (1)**rackspace** (1)**rails** (2)**redis** (1)**refactoring** (2)**Reniec** (1)**renovatebot** (2)**Rider** (1)**ruby** (4)**rust** (1)**scala** (1)**SCD2010** (1)

SCJP (1)**Scrum** (3)**Scrum evaluacion** (1)**seminarios** (1)**Setup** (1)**SourceRepo** (1)**spring** (13)**spring 3.1** (1)**spring android** (1)**spring mobile** (1)**spring social** (1)**spring-boot** (9)**spring-boot-admin** (2)**spring-cloud** (1)**spring-cloud-config** (2)**SpringCommunityDay** (1)**SpringRoo** (2)**springsource** (1)**sqlserver** (2)**start-up** (1)**STS** (1)**Subclipse** (1)**Subversion** (1)**SUN** (1)**SUNAT** (2)**synergyj** (2)**Syscom** (1)**Talleres** (21)

- Telefonica (1)
- thedeveloperconf (1)
- thymeleaf (1)
- Trac (1)
- try-with-resources (1)
- twitter (1)
- Tye (1)
- ubuntu (3)
- UNI (3)
- UNMSM (1)
- UPC (1)
- videos (1)
- vimeo (1)
- weblogic (2)
- Workspace (1)
- WPF (1)
- xorcery (7)
- xsd (1)