Taller arquitecturas de servidores de aplicaciones, meta protocolos de objetos, patrón ioc, reflexión

Germán Andrés Ospina Quintero

26 de Febrero del 2021

Contents

1 Glosario

Java: Java es un tipo de lenguaje de programación y una plataforma informática, creada y comercializada por Sun Microsystems en el año 1995. Se constituye como un lenguaje orientado a objetos, su intención es permitir que los desarrolladores de aplicaciones escriban el programa una sola vez y lo ejecuten en cualquier dispositivo. (Content, 2019)

Git: Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. (Wikipedia, 2021)

Maven: Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a Apache Ant, pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. (Wikipedia, 2020)

POJO (Plain Old Java Object): Un POJO es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre de 2000 y utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial. (Wikipedia, 2020)

2 Resumen

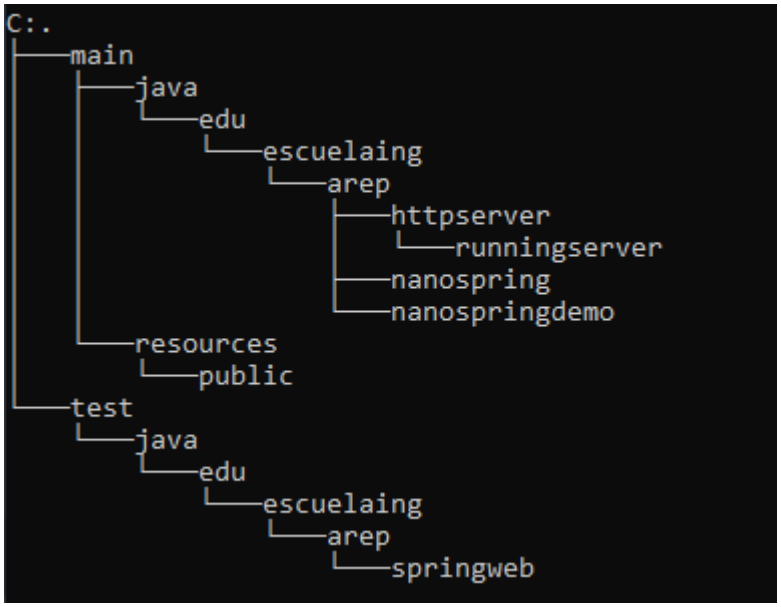
Para este taller, se implementó un servidor capaz de entregar páginas html e imágenes tipo PNG. El servidor provee un framework IoC para la construcción de aplicaciones web a partir de POJOS. Usando el servidor se construyó una aplicación Web de ejemplo y se desplegó en Heroku. El servidor debe atiende múltiples solicitudes no concurrentes. La finalidad de este taller es desarrollar un prototipo mínimo que demuestre las capcidades reflexivas de JAVA y que permita por lo menos cargar un bean (POJO) y derivar una aplicación Web a partir de él.

3 Introducción

Inicialmente, haciendo uso de las herramientas para diseñar el código fuente, desplegarlo y hacer integración continua; maven, Heroku y Circle CI respectivamente, se desarrolló un prototipo de Spring Boot con funcionalidades minimas. Este prototipo fue probado al crear un cliente web con HTML, JavaScript y JQuery

En este documento, se presenta y se describe la estructura, el diseño, la arquitectura de la aplicación y por último se hace un análisis de las pruebas hechas para validar la correctitud de la aplicación

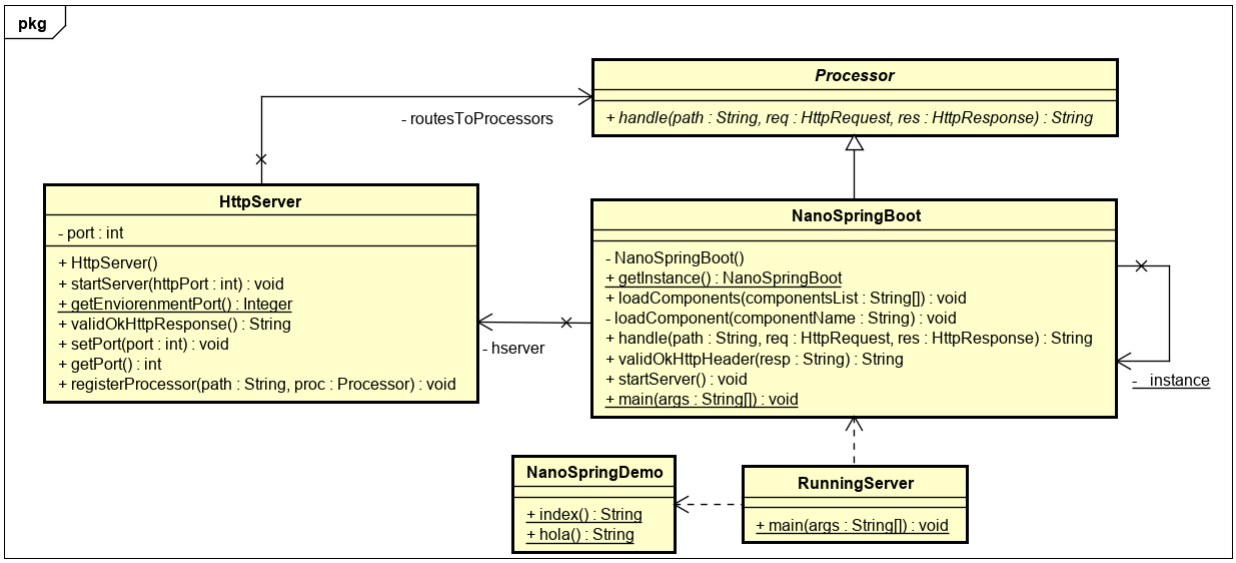
4 Estructura



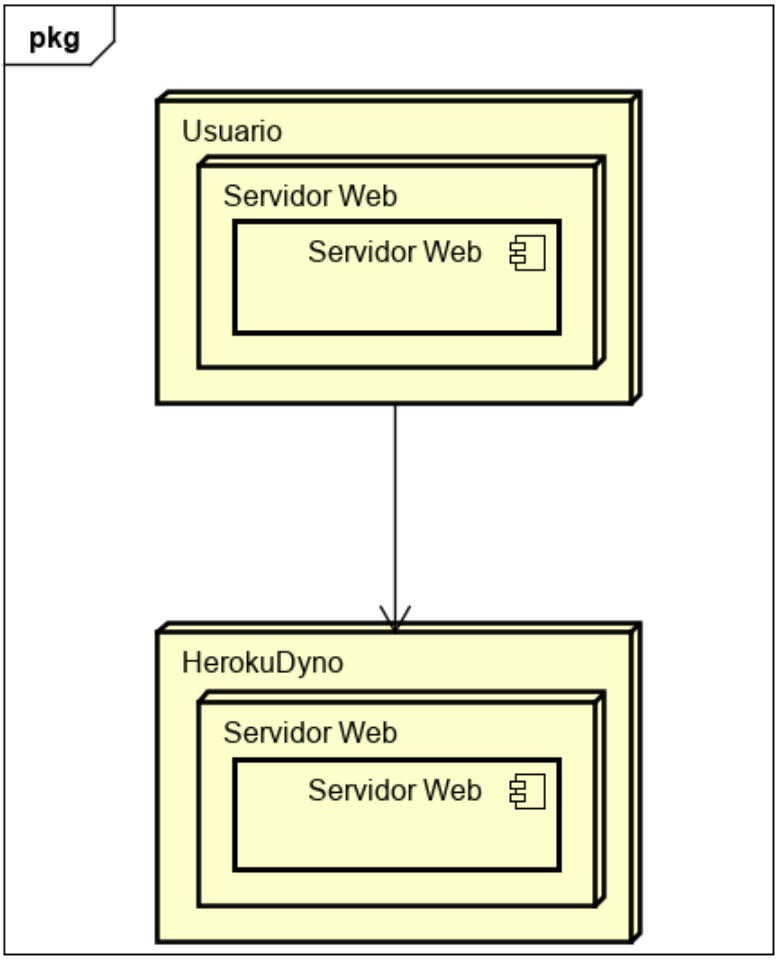
La aplicación presenta dos carpetas principales: main y test. La primera, presenta todo el código fuente de la aplicación, estructurado en la carpeta arep donde se encuentran los paquetes: httpserver, en donde se encuentra la implementación del servidor web, nanospring, con la implementación necesaria para procesar las rutas, y nanospringdemo, en donde se desarrolla la implementación del bean. También, tiene un directorio llamado resources que contiene toda la implementación del cliente web.

La segunda, contiene las cuatro pruebas hechas al código fuente de la aplicación.

5 Diseño


pkg
Processor
+ handle(path : String, req : HttpRequest, res : HttpResponse) : String
- routesToProcessors
HttpServer
- port : int
+ HttpServer()
+ startServer(httpPort : int) : void
+ getEnviorenmentPort() : Integer
+ validOkHttpResponse() : String
+ setPort(port : int) : void
+ getPort() : int
+ registerProcessor(path : String, proc : Processor) : void
- hserver
NanoSpringBoot
- NanoSpringBoot()
+ getInstance() : NanoSpringBoot
+ loadComponents(componentsList : String[]) : void
- loadComponent(componentName : String) : void
+ handle(path : String, req : HttpRequest, res : HttpResponse) : String
+ validOkHttpHeader(resp : String) : String
+ startServer() : void
+ main(args : String[]) : void
- instance
NanoSpringDemo
+ index() : String
+ hola() : String
RunningServer
+ main(args : String[]) : void

6 Arquitectura



7 Pruebas

Como se mencionó anteriormente, se realizaron 4 pruebas para validar la correctitud del funcionamiento de la aplicación

La primera prueba valida que se reciba el mensaje definido en la ruta /hola del bean

```
@Test
public void shouldGetHolaMessage() throws IOException
{
    URL url = new URL("https://stark-ridge-65206.herokuapp.com/springapp/hola");
    URLConnection urlConnection = url.openConnection();
    BufferedReader read = new BufferedReader(new InputStreamReader(urlConnection.getInputStream()));
    String inputLine = read.readLine();
    assertTrue(inputLine.equals("Saludos desde Spring Boot!"+" "));
    read.close();
}
```

La segunda prueba valida que se reciba el mensaje definido en la ruta /hola del bean con el valor del query param

```
@Test
public void shouldGetHolaMessageWithQueryParams() throws IOException
{
    URL url = new URL("https://stark-ridge-65206.herokuapp.com/springapp/hola?value=test");
    URLConnection urlConnection = url.openConnection();
    BufferedReader read = new BufferedReader(new InputStreamReader(urlConnection.getInputStream()));
    String inputLine = read.readLine();
    assertTrue(inputLine.equals("Saludos desde Spring Boot! test"));
    read.close();
}
```

La tercera prueba valida que se reciba el mensaje definido en la ruta /hello del bean

```java
@Test
public void shouldGetHelloMessage() throws IOException
{
    URL url = new URL("https://stark-ridge-65206.herokuapp.com/springapp/hello");
    URLConnection urlConnection = url.openConnection();
    BufferedReader read = new BufferedReader(new InputStreamReader(urlConnection.getInputStream()));
    String inputLine = read.readLine();
    assertTrue(inputLine.equals("Hi "));
    read.close();
}
```

La cuerta prueba valida que se reciba el mensaje definido en la ruta /hello del bean con el valor del query param

```java
@Test
public void shouldGetHelloMessageWithQueryParams() throws IOException
{
    URL url = new URL("https://stark-ridge-65206.herokuapp.com/springapp/hello?value=test");
    URLConnection urlConnection = url.openConnection();
    BufferedReader read = new BufferedReader(new InputStreamReader(urlConnection.getInputStream()));
    String inputLine = read.readLine();
    assertTrue(inputLine.equals("Hi test"));
    read.close();
}
```