**ARQUITECTURA BASADA EN COMPONENTES.**

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas. Esto provee un nivel de abstracción mayor que los principios de orientación por objetos y no se enfoca en asuntos específicos de los objetos como los protocolos de comunicación y la forma como se comparte el estado.

El estilo de arquitectura basado en componentes tiene las siguientes características:

• Es un estilo de diseño para aplicaciones compuestas de componentes individuales.

• Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas.

• Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de interfaces que contienen métodos, eventos y propiedades.

**PRINCIPIOS FUNDAMENTALES**

Un componente es un objeto de software específicamente diseñado para cumplir con cierto propósito. Los principios fundamentales cuando se diseña un componente es que estos deben ser:

• Reusable. Los componentes son usualmente diseñados para ser utilizados en escenarios diferentes por diferentes aplicaciones, sin embargo, algunos componentes pueden ser diseñados para tareas específicas.

• Sin contexto especifico. Los componentes son diseñados para operar en diferentes ambientes y contextos. Información específica como el estado de los datos deben ser pasadas al componente en vez de incluirlos o permitir al componente acceder a ellos.

• Extensible. Un componente puede ser extendido desde un componente existente para crear un nuevo comportamiento.

• Encapsulado. Los componentes exponen interfaces que permiten al programa usar su funcionalidad. Sin revelar detalles internos, detalles del proceso o estado.

• Independiente. Los Componentes están diseñados para tener una dependencia mínima de otros componentes. Por lo tanto los componentes pueden ser instalados en el ambiente adecuado sin afectar otros componentes o sistemas.

**BENEFICIOS**

Los siguientes son los principales beneficios del estilo de arquitectura basado en componentes:

• Facilidad de Instalación. Cuando una nueva versión esté disponible, usted podrá reemplazar la versión existente sin impacto en otros componentes o el sistema como un todo.

• Costos reducidos. El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento.

• Facilidad de desarrollo. Los componentes implementan un interface bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.

• Reusable. El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.

• Mitigación de complejidad técnica. Los componentes mitigan la complejidad por medio del uso de contenedores de componentes y sus servicios. Ejemplos de servicios de componentes incluyen activación de componentes, gestión de la vida de los componentes, gestión de colas de mensajes para métodos del componente y transacciones.

**EJEMPLOS**

Tipos comunes de componentes usados en aplicaciones incluyen:

• Componentes de interfaz de usuario, como grillas, botones, etc., generalmente conocidos como “controles”.

• Componentes de ayuda que exponen un conjunto específico de funciones usados por otros componentes.

• Componentes que se no se usan con mucha frecuencia o son intensivos en recursos y deben ser actividades usando una aproximación de solo en el momento justo (Just in Time (JIT)). Estos son comunes en escenarios de componentes distribuidos o en componentes remotos.

• Componentes encolados, aquellos cuyos métodos pueden ser ejecutados de forma asíncrona usando colas de mensajes del tipo almacenamiento, entrega.

**SPRING**

Spring es un *framework* alternativo al *stack* de tecnologías estándar en aplicaciones JavaEE. Nació en una época en la que las tecnologías estándar JavaEE y la visión "oficial" de lo que debía ser una aplicación Java Enterprise tenían todavía muchas aristas por pulir. Los servidores de aplicaciones eran monstruosos devoradores de recursos y los EJB eran pesados, inflexibles y era demasiado complejo trabajar con ellos. En ese contexto, Spring popularizó ideas como la inyección de dependencias o el uso de objetos convencionales (POJOs) como objetos de negocio, que suponían un soplo de aire fresco. Estas ideas permitían un desarrollo más sencillo y rápido y unas aplicaciones más ligeras. Eso posibilitó que de ser un *framework* inicialmente diseñado para la capa de negocio pasara a ser un completo *stack* de tecnologías para todas las capas de la aplicación.

Las ideas "innovadoras" que en su día popularizó Spring se han incorporado en la actualidad a las tecnologías y herramientas estándar. Así, ahora mismo no hay una gran diferencia entre el desarrollo con Spring y el desarrollo JavaEE "estándar", o al menos no tanta como hubo en su día. No obstante, Spring ha logrado aglutinar una importante comunidad de desarrolladores en torno a sus tecnologías y hoy por hoy sigue constituyendo una importante alternativa al estándar que merece la pena conocer. En la actualidad, las aportaciones más novedosas de Spring se centran en los campos de Big Data/NoSQL, HTML5/móviles y aplicaciones sociales.

Básicamente, la mayor diferencia práctica que podemos encontrar hoy en día entre desarrollar con Spring y con JavaEE estándar es la posibilidad de usar un servidor web convencional al estilo Tomcat para desplegar la aplicación. Las tecnologías JavaEE más sofisticadas requieren del uso de un servidor de aplicaciones, ya que los APIs los implementa el propio servidor, mientras que Spring no es más que un conjunto de librerías portables entre servidores. En otras palabras, usando JavaEE estándar, nos atamos al servidor de aplicaciones y usando Spring nos atamos a sus APIs. Eso sí, los desarrolladores de Spring se han preocupado bastante de armonizar con el estándar en la medida de lo posible, por ejemplo dando la posibilidad de usar anotaciones estándar aun con implementaciones propias por debajo. La idea es obstaculizar lo menos posible una posible portabilidad a JavaEE, idea que es de agradecer en un mundo en que todos los fabricantes intentan de una forma u otra mantener un público cautivo.

Hay una abundante bibliografía sobre Spring, aunque la [documentación del propio proyecto](http://static.springsource.org/spring-framework/docs/) es excelente y bastante exhaustiva, pudiéndose utilizar perfectamente no solo como manual de referencia sino como tutorial detallado. La hemos tomado como referencia básica para la elaboración de estos apuntes.

Desde un punto de vista genérico, Spring se puede ver como un soporte que nos proporciona tres elementos básicos:

* **Servicios *enterprise*:** podemos hacer de manera sencilla que un objeto sea transaccional, o que su acceso esté restringido a ciertos roles, o que sea accesible de manera remota y transparente para el desarrollador, o acceder a otros muchos servicios más, sin tener que escribir el código de manera manual. En la mayoría de los casos solo es necesario anotar el objeto.
* **Estereotipos configurables** para los objetos de nuestra aplicación: podemos anotar nuestras clases indicando por ejemplo que pertenecen a la capa de negocio o de acceso a datos. Se dice que son configurables porque podemos definir nuestros propios estereotipos "a medida": por ejemplo podríamos definir un nuevo estereotipo que indicara un objeto de negocio que además sería cacheable automáticamente y con acceso restringido a usuarios con determinado rol.
* **Inyeccion de dependencias:**: ya hemos visto este concepto cuando se hablaba de CDI de JavaEE. La inyección de dependencias nos permite solucionar de forma sencilla y elegante cómo proporcionar a un objeto cliente acceso a un objeto que da un servicio que este necesita. Por ejemplo, que un objeto de la capa de presentación se pueda comunicar con uno de negocio. En Spring las dependencias se pueden definir con anotaciones o con XML.

**ESTADO DEL ARTE**

No se encontró ningún aplicativo que mescle un sistema recomendador para la lectura de libros con un sistema de gamificacion.

**thefussylibrarian.com :**

Entre los punto a destacar de este aplicativo, envía recomendaciones diarias al correo electrónico, entre los puntos que no tiene son que se basa en la recomendación enfocada por contenido, además la base de datos de libros es estática y es manejada por los administradores del aplicativo y por ultima desventaja se tiene que es sitio solo tiene soporte para el idioma ingles y libros en este idioma.

**quelibroleo.com :**

Este aplicativo tiene un enfoque más de buscador de libros que de recomendador, pero la parte de recomendador es basada en recomendación colaborativo ya que los usuarios colocan libros que quieran que otras personas lean por esto no es muy eficaz el sistema ya que no garantiza que recomiende libros enfocado en cada usuario, el sitio está en español, también tiene el problema es que los libros que alimentan el sistema son administrados por el creador del sitio.

**ebrolis.com :**

El aplicativo como novedad tiene que aparte de recomendador hace de puente con las librerías para comprar el libro o con Amazon.com para libros virtuales, en cuanto al sistema recomendador es basado en contenido ya que según los libros que vaya poniendo en el carrito o haya comprado, el aplicativo está en virios idiomas, igual que los otros aplicativos tienen la base de datos restringida por la alimentación del administrador.

**boolino.es:**

Es un aplicativo muy similar a *quelibroleo.com* ya que se enfoca más en un buscador de libros la recomendación colaborativa y también vende libros, la base de datos no se alimenta por los usuarios, tiene varios idiomas para seleccionar y libros de todo tipo. Pero no deja de ser una tienda de libros con un buscador.

**whatshouldireadnext.com :**

Este es el único aplicativo de los evaluados que cuenta con un sistema que se alimenta de los mismos usuarios, la recomendación es colaborativa**,** pero es solo está en inglés, también cuenta con una base de datos de libros pequeña ya que su comunidad es pequeña, no hay un sistema que atraiga nuevos usuarios o que los antiguos sigan colaborando, además que no es obligatorio registrar libros.

Como se pudo deducir ningún a casi ningún aplicativo que está entre los más usados utiliza la recomendación hibrida, además casi todos se enfocan en la recomendación colaborativa eso quiere decir no se enfocan en lo que consume el usuario, tampoco se encontró un aplicativo que utilice gamificacion

**Referencias**

Extractado de [La Guía de Arquitectura Versión 2.0a](http://www.juanpelaez.com/Blog/ct.ashx?id=1eca3f73-119b-4f9f-8089-80b5b250b84f&url=http%3a%2f%2fwww.codeplex.com%2fAppArchGuide) del grupo de [Patterns and Practices de Microsoft](http://www.juanpelaez.com/Blog/ct.ashx?id=1eca3f73-119b-4f9f-8089-80b5b250b84f&url=http%3a%2f%2fmsdn.microsoft.com%2fen-us%2fpractices%2fdefault.aspx) 2010.

Tomado del sitio oficial de Spring <https://spring.io/docs> Ultima revision 2017