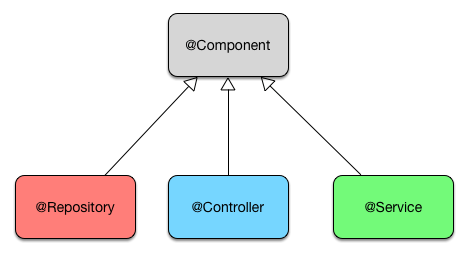
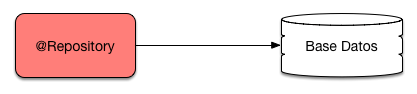
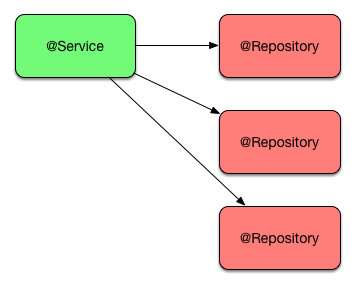
@Component: Es el estereotipo general y permite anotar un bean para que Spring lo considere uno de sus objetos.



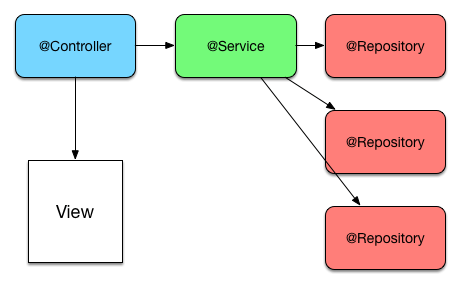
@Repository: es un anotador que da de alta un bean para cualquier clase que sea de acceso a DATOS o persistencia, Al marcar el bean con esta anotación Spring aporta servicios transversales como conversión de tipos de excepciones.



@Service: Esta anotación se encarga de gestionar las reglas de negocio más importantes a nivel de la aplicación y hace llamadas a varios repositorios (@Repository) de forma simultánea. Su tarea fundamental es la de agregador.



@Controller : La última de las anotaciones que realiza las tareas de controlador y gestión de la comunicación entre el usuario y el aplicativo. Para ello se apoya habitualmente en algún motor de plantillas o librería de etiquetas que facilitan la creación de páginas (vistas).



REST es un mecanismo para el intercambio y manipulación de recursos a través de Internet. Es semejante a los Web Services, pero no usa un protocolo concreto para el intercambio de la información.

Inicialmente REST se describe en el contexto de HTTP (aunque no se limita sólo a este protocolo), por lo que se aprovecha de todas sus características: URIs, tipos de media, sesiones, seguridad, cache, etc.

Las operaciones típicas que podemos hacer con REST son:

* GET, para recuperar un recurso. Es idempotente, es decir si la ejecutamos más de una vez siempre devuelve el mismo recurso.
* POST, para añadir recursos. No es idempotente, es decir si la ejecutamos dos veces estaremos añadiendo dos recursos.
* PUT, para modificar un recurso. Es idempotente, si la ejecutamos más de una vez la modificación es siempre la misma (por ejemplo cambiar el nombre de una persona en una agenda).
* DELETE, para borrar un recurso. Es idempotente, si lo ejecutamos más de una vez el resultado es siempre el mismo: el recurso deja de estar en el sistema (la primera vez se borra realmente, las siguientes veces simplemente se ignora la petición, pero no da error).

Otros verbos que tenemos disponibles son:

* HEAD, para pedir un recurso sin recuperarlo, nos sirve para saber si existe o para traer meta información sobre el recurso.
* OPTIONS, para preguntarle a un servidor sobre que otros verbos son aplicables a un recurso determinado.
* PATH, es de reciente adopción (en el 2010), es un intento de expresar de forma estándar actualizaciones parciales de un recurso.

Maven:

* **mvn clean:clean (o mvn clean)**: limpia todas las clases compiladas del proyecto.
* **mvn compile**: compila el proyecto
* **mvn package**: empaqueta el proyecto (si es un proyecto java simple, genera un jar, si es un proyecto web, un war, etc…)
* **mvn install**: instala el artefacto en el repositorio local (/Users/home/.m2)

**Scope:**

* **compile**: es la que tenemos por defecto sino especificamos scope. Indica que la dependencia es necesaria para compilar. La dependencia además se propaga en los proyectos dependientes.
* **provided**: Es como la anterior, pero esperas que el contenedor ya tenga esa libreria. Un claro ejemplo es cuando desplegamos en un servidor de aplicaciones, que por defecto, tiene bastantes librerías que utilizaremos en el proyecto, así que no necesitamos desplegar la dependencia.
* **runtime**: La dependencia es necesaria en tiempo de ejecución pero no es necesaria para compilar.
* **test**: La dependencia es solo para testing que es una de las fases de compilación con maven. JUnit es un claro ejemplo de esto.
* **system**: Es como provided pero tienes que incluir la dependencia explicitamente. Maven no buscará este artefacto en tu repositorio local. Habrá que especificar la ruta de la dependencia mediante la etiqueta *<systemPath>*
* **import**: este solo se usa en la sección *dependencyManagement*. Lo explicaré en otro artículo sobre transitividad de dependencias.

**¿Que es un arquetype?**

**Un arquetype, traducido arquetipo, es, una plantilla**. Cuando creamos un proyecto, como ahora veremos, tenemos que especificar uno. Un arquetipo crea la estructura del proyecto, el contenido del pom.xml, la estructura de carpetas y los ficheros que incluye por defecto.

Estructura de un proyecto maven:

maven-intro-scr01

Tenemos nuestro **pom.xml**, donde se configura el proyecto maven y una carpeta **src** donde se colocan los fuentes. Dentro de src, tenemos dos carpetas: main, donde va todo el código de nuestro proyecto, y test, donde va el código de test. En este caso, como es un proyecto java simple, dentro de main solo tenemos la carpeta java. En ella ya podemos colocar nuestros paquetes y clases java

Diferencia entre Archetype y Artifact