Para la solución del problema planteado, se utilizó un mecanismo de resta de subconjuntos, así al consultar un rango de coordenadas, se empieza desde el subconjunto más grande hasta achicarlo y encontrar el rango especifico, dejando la carga computacional de la consulta al motor de base de datos.

El proyecto java está construido como una aplicación monolítica, compuesta por dos war que exponen cada capa de la aplicación, siendo así, la capa de negocio expuesta por servicios rest que se comunican con la capa de presentación.

Estructura proyectos:

cubeSum-ear (ear)

cubeSum-servicios (war)

cubeSume-web (war)

**cubeSum-servicios**\src\main\java\com\cubeSum

* Common
  + JaxRSActivator – Identifica la URL base para las peticiones Rest.
* Dto
  + CoordenadaDTO – Objeto de trasferencia para los datos de una coordenada, mapea las propiedades de la coordenada.
  + DatosConsultaValorInDTO – Objeto de transferencia que permite ingresar los datos necesarios para la consulta.
  + ResultadoDTO – Objeto de transferencia que contiene atributos informativos sobre la actualización de una coordenada.
  + ResultadoSumCubeDTO - Objeto de transferencia que contiene atributos informativos sobre la consulta de un rango de coordenadas.
* Entidades (**Persistencia**)
  + Coordenada – Entidad que mapea los atributos de BD de la tabla coordenada.

**cubeSum-servicios**\src\main\java\com\cubeSum\servicios

* ConsultasEJB – EJB sin estado, contiene lógica de negocio, con dos métodos encargados de actualizar una coordenada y consultar un rango de coordenadas.
* IConsultasEjbLocal - Interfaz local que expone los métodos de negocio del EJB.
* CORSFilter - Filtro encargado de mediar entre las peticiones entrantes y respuestas salientes.
* ConsultasRest – Clase RESTful que expone los servicios encontrados en la aplicación.

**cubeSum-servicios**\src\test\java\com\cubeSum\servicios\persistence

Conjunto de clases encargadas de construir un contexto de persistencia acorde para las pruebas unitarias, es decir que implemente los métodos propios sin realizar commits permanentes a la base de datos.

* EntityManagerProxy
* TestEntityManager
* UncommitableEntityManager

**cubeSum-servicios**\src\test\java\com\cubeSum\servicios\util

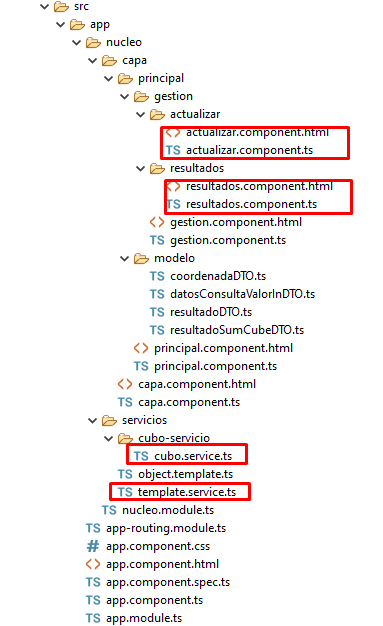
* ComunUnitarias – Clase común para las pruebas unitarias, con un método protegido encargado de obtener instancias de EJB, por reflexión.

**cubeSum-servicios**\src\test\java\com\cubeSum\servicios

Pruebas unitarias de los métodos de negocio.

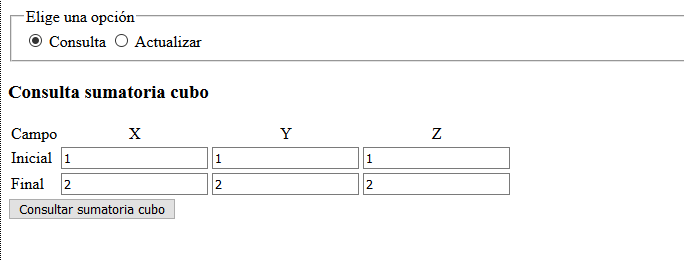
* PruebaUnitariaActualizaCoordenada
* PruebaUnitariaConsultaSuma

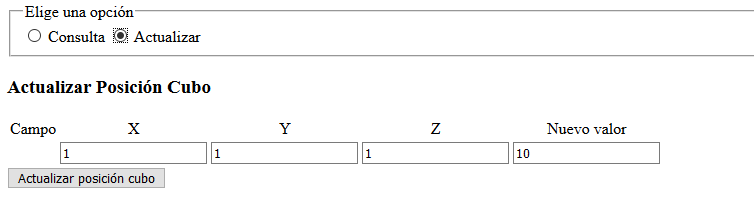
Presentación



Artefactos destacados:

* Template.service.ts – Artefacto encargado de construir las peticiones genéricas html (GET, POST), para la comunicación con negocio vía servicios rest.
* cubo.service.ts - Artefacto que contiene las peticiones especificas para los servicios de negocio.
* Resultados y actualizar, componentes angular encargados de presentar los campos para la gestión del cubo, ejemplo visualización:





Principio de responsabilidad única:

Principio que indica que un módulo dentro del sistema solo debe tener una única razón para cambiar, si se encuentra que existen varias razones para que ese módulo cambie, se estaría rompiendo con este principio, en ese caso se debería buscar un mecanismo para que se garantice que las responsabilidades estén divididas, y cada módulo se encargue de una única cosa, así se mantendría una alta cohesión y un bajo acoplamiento.

Código limpio

Es un código que además de cumplir con el objetivo con el que fue construido, es fácil de leer, mantener y probar, además, puede ser menos propenso a errores, esto se logra, utilizando técnicas o principios como SOLID, una forma de verificar constantemente la calidad del código es usar herramientas de análisis de código como sonar.