Solución Prueba BackEnd – Rappi

Emir Cortés Trujillo

Coding Challenge

- El lenguaje que seleccioné para el reto de hackerrank.com fue Java porque es el que mejor manejo.

Análisis:

- Al leer las restricciones del problema noté que el espacio de soluciones no era muy grande ya que la cantidad de datos a explorar, teniendo en cuenta que el cubo estuviera totalmente lleno, sería apenas 1 millon de elementos y si tuviera que recorrerlos todos se podría hacer en un tiempo razonable. Para esta primera aproximación hice pruebas llenando un cubo de 100x100x100 con numéros de 10 cifras y la suma de todos los números no se demoraba más de 200ms. Sin embargo, a pesar de ser una solución razonable al ejecutarlo contra los casos de prueba en hackerrank.com fallaba por timeouts.
- La observación clave para plantear la siguiente solución es la restricción que especifica que para cada caso de prueba ***no habrá más de m = 1000 operaciones entre actualizaciones y consultas***, lo que implica que en el peor de los casos podría haber 999 datos y una consulta. A partir de esto resultó evidente que era más económico recorrer los datos ingresados en las actualizaciones y ver si se encontraban en el rango de las consultas para posteriormente sumarlos. Con esto en mente la complejidad de cada consulta es lineal - O(n) - respecto al número de datos ingresados y dado que no habrá más de 999 datos es un tiempo eficiente.
- Si el problema fuera un poco difetente y cubo estuviera completamente lleno y se pudieran hacer 1000 consultas la solución sería diferente, en términos resumidos: se podría precalcular la suma de todos subcubos y a partir de esas sumas se podría obtener cualquier consulta en tiempo constante – O(c) - a través de geometría.

Implementación:

- La solución se encuentra en un repositorio de github.com
  - (https://github.com/emirfredy/hackerrank
- Hay dos implementaciones para el problema: la primera es un proyecto por consola que lee desde la terminal e imprime el resultado en pantalla tal cómo lo especificaban las instrucciones

de hackerrank.com, la segunda es un proyecto web que tiene el front-end en AngularJS que consume un API REST.

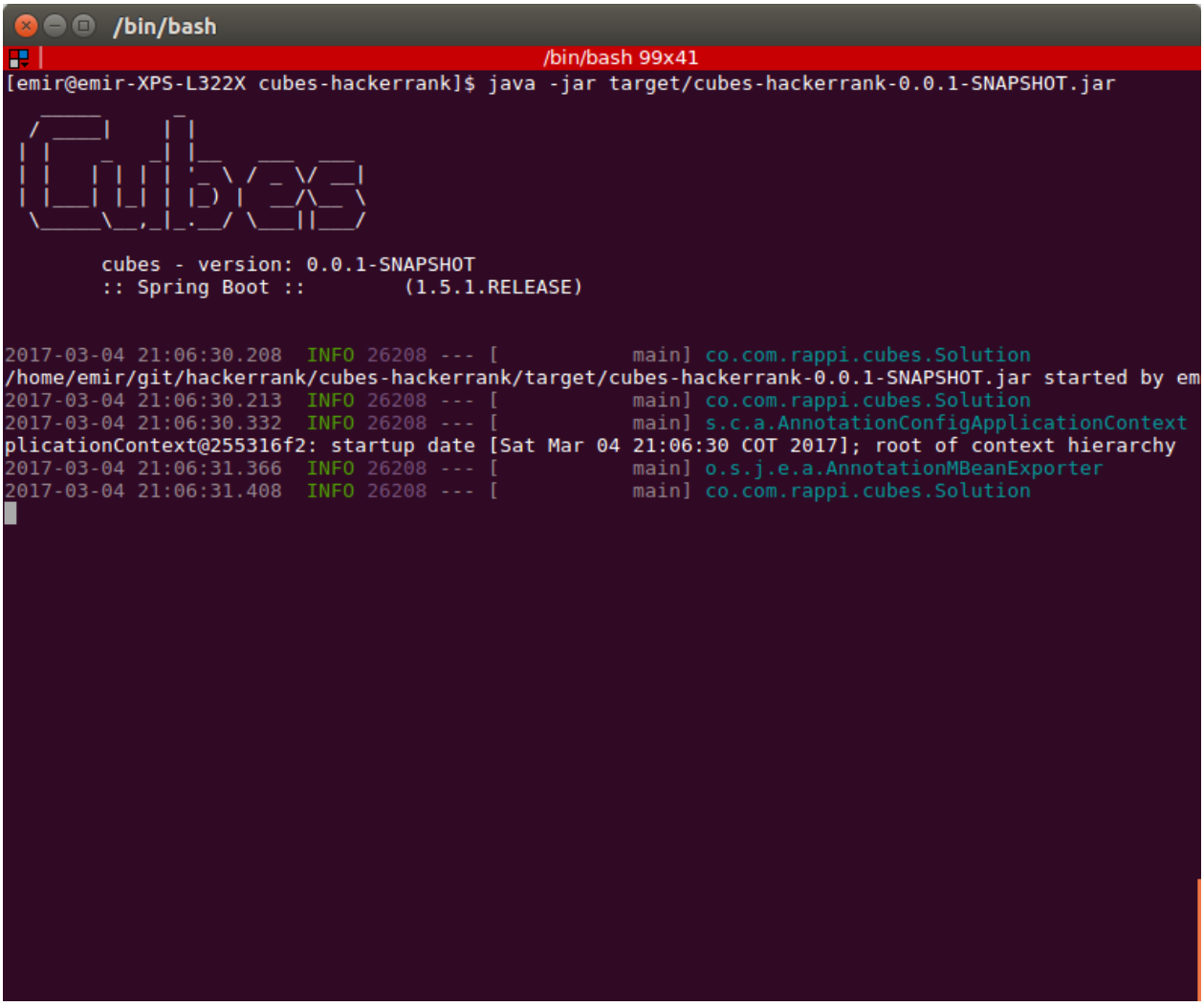
- En el repositorio de github.com hay commits que muestran los pasos más importantes del desarrollo. A continuación veremos una explicación de cada uno.
  - https://github.com/emirfredy/hackerrank/commits/master
- Para la solución por consola se creo un proyecto

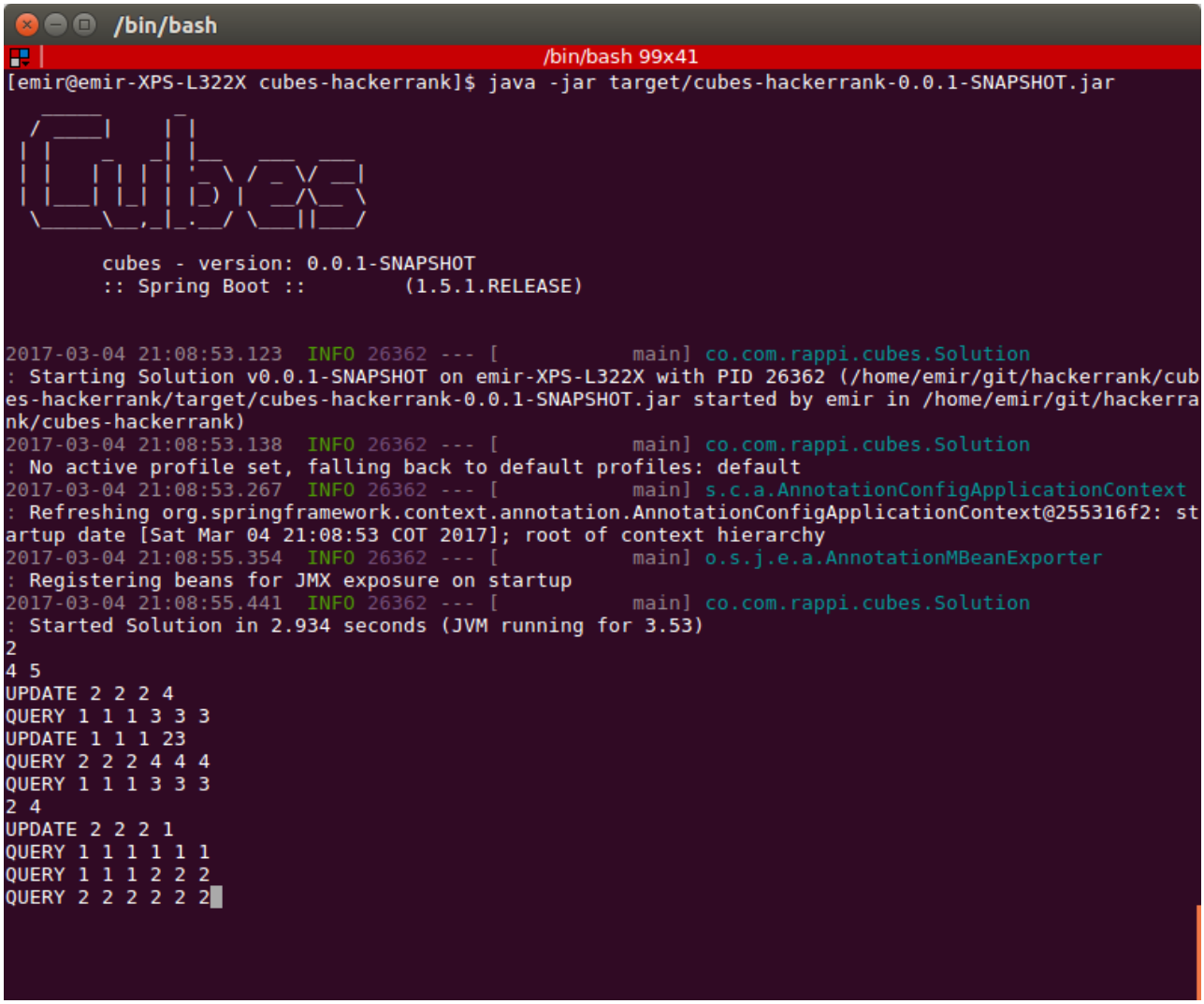
  https://github.com/emirfredy/hackerrank/tree/master/cubes-hackerrank
  - 2f8dc58: Se creó un proyecto básico, usando maven, con la solución al problema. Todas las clases están en un sólo archivo porque en hackerrank.com la solución se presentaba de esa manera y la intención principal era validar el correcto funcionamiento. Hay 4 clases:
    - `Solucion:` es el punto de entrada y sirve para hacer la lectura desde la terminal, para crear los objetos del dominio del problema, para delegar la solución del problema a la clase `CoordinateUtil` y finalmente para imprimir el resultado de las sumas.
    - `Coordinate`: representa un concepto del dominio que representa un punto en un plano 3d
    - `CoordinateUtil`: Es una clase utilitaria encargada de realizar validaciones de las coordenadas y de hacer la suma en un rango
    - `CoordinateUtilTest`: Contiene las pruebas de la clase central del problema `CoordinateUtil`. El nombre de cada método que realiza una prueba es autodescriptivo y por eso no está documentado. La forma general es `given{Escenario}_{metodo}_{resultadoEsperado}`
  - a769259: Se agregó un pantallazo de todos los casos de prueba pasando usando la solución propuesta en el commit anterior
  - e68b321: Cada clase pasó a estar en su propio archivo
  - c14a878: Se pasó a Spring Boot, de manera que se pudiera compilar sin necesidad de tener instalado maven localmente y para poderlo empaquetar fácilmente. Para esto el único requerimiento es tener Java instalado (jdk-8) y tener una variable de entorno llamada JAVA_HOME que apunte a la instación de Java.
    - Con esto en mente, se puede hacer lo siguiente para empaquetar y posteriormente ejecutar, estando ubicado en la carpeta `hackerrank/cubes-hackerrank`:

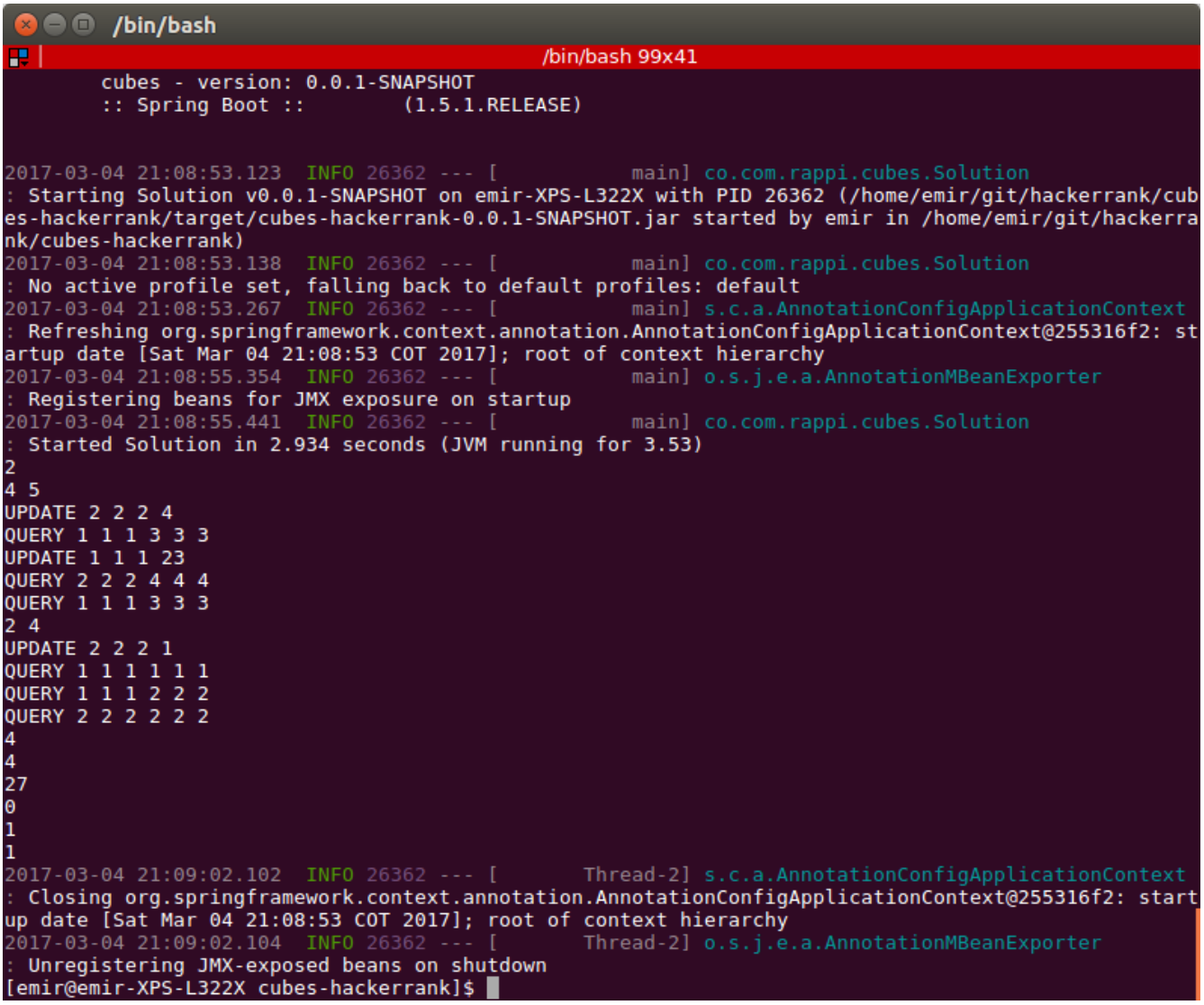
      ```
      ./mvnw clean install package

      java -jar target/cubes-hackerrank-0.0.1-SNAPSHOT.jar
      ```

- Al hacer esto se lanza el programa y se queda esperando para recibir datos por la consola:




/bin/bash
/bin/bash 99x41
[emir@emir-XPS-L322X cubes-hackerrank]$ java -jar target/cubes-hackerrank-0.0.1-SNAPSHOT.jar
cubes - version: 0.0.1-SNAPSHOT
:: Spring Boot :: (1.5.1.RELEASE)
2017-03-04 21:08:53.123 INFO 26362 --- [main] co.com.rappi.cubes.Solution
: Starting Solution v0.0.1-SNAPSHOT on emir-XPS-L322X with PID 26362 (/home/emir/git/hackerrank/cubes-hackerrank/target/cubes-hackerrank-0.0.1-SNAPSHOT.jar started by emir in /home/emir/git/hackerrank/cubes-hackerrank)
2017-03-04 21:08:53.138 INFO 26362 --- [main] co.com.rappi.cubes.Solution
: No active profile set, falling back to default profiles: default
2017-03-04 21:08:53.267 INFO 26362 --- [main] s.c.a.AnnotationConfigApplicationContext
: Refreshing org.springframework.context.annotation.AnnotationConfigApplicationContext@255316f2: startup date [Sat Mar 04 21:08:53 COT 2017]; root of context hierarchy
2017-03-04 21:08:55.354 INFO 26362 --- [main] o.s.j.e.a.AnnotationMBeanExporter
: Registering beans for JMX exposure on startup
2017-03-04 21:08:55.441 INFO 26362 --- [main] co.com.rappi.cubes.Solution
: Started Solution in 2.934 seconds (JVM running for 3.53)
2
4 5
UPDATE 2 2 2 4
QUERY 1 1 1 3 3 3
UPDATE 1 1 1 23
QUERY 2 2 2 4 4 4
QUERY 1 1 1 3 3 3
2 4
UPDATE 2 2 2 1
QUERY 1 1 1 1 1 1
QUERY 1 1 1 2 2 2
QUERY 2 2 2 2 2 2



- Para la solución web se creó un proyecto a parte:

  https://github.com/emirfredy/hackerrank/tree/master/cubes-web

  - 9a3bc9b: Es un proyecto web que usa spring boot y que tiene un servidor de aplicaciones embebido (Apache Tomcat 8) por lo que se puede ejecutar en cualquier parte sin preinstalar nada. También se incluyó una base de datos embebida (H2) que almacena cada problema junto con sus actualizaciones y consultas. En este commit se definió la interfaz REST del API y el resultado fue el siguiente:

Endpoint	Verbo	Responsabilidad
api/problem	POST	Crear un nuevo caso de prueba, recibe el tamaño del cubo como parámetro
api/{id}	GET	Devolver el problema identificado por el id dado
api/{id}/query	PUT	Agregar una consulta al problema dado

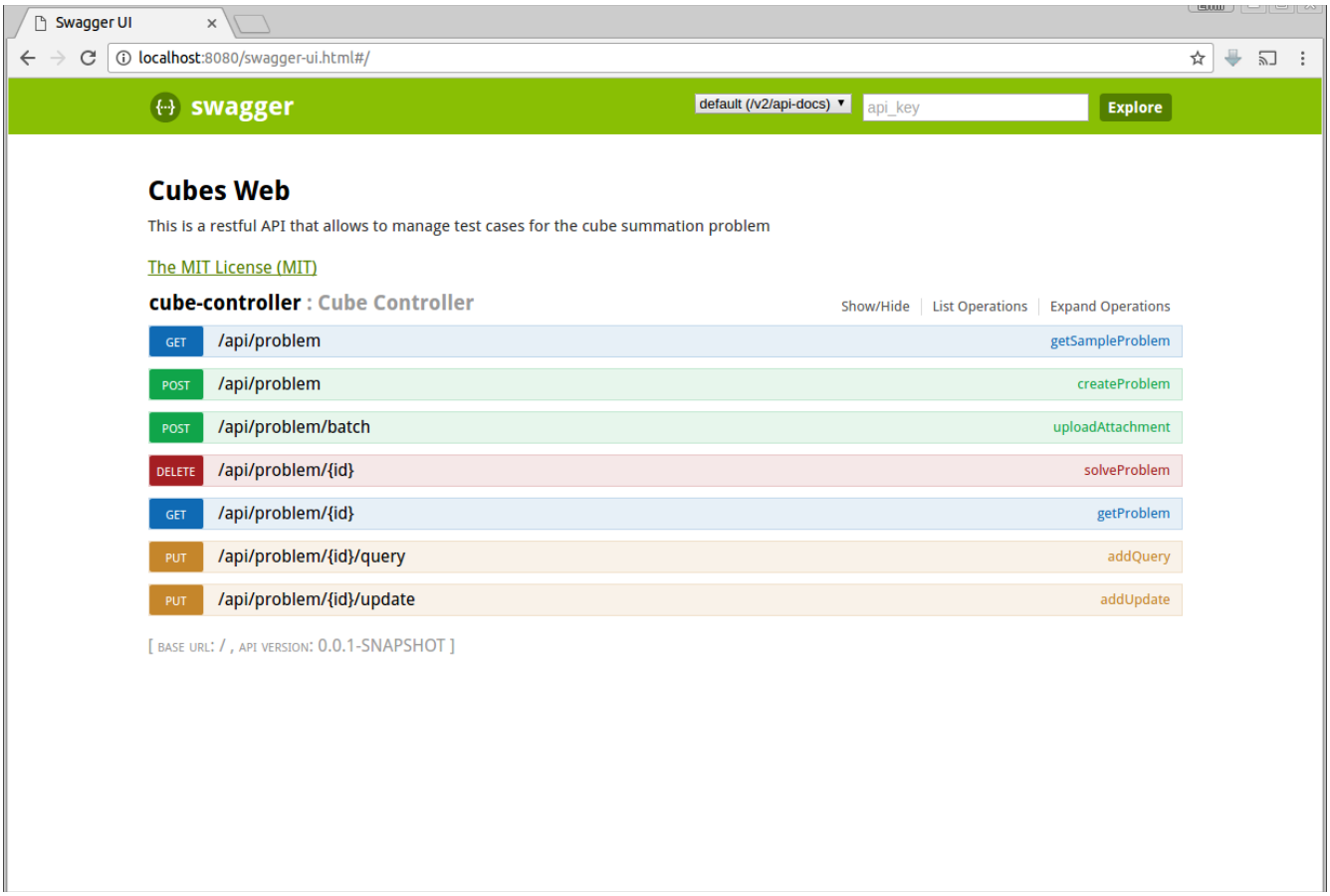
api/{id}/update	PUT	Agregar una actualización al problema dado
api/{id}	DELETE	Resolver el problema, calcular todas las sumas y devolverlas de acuerdo a las actualizaciones y consultas que tenga un problema, y finalmente borrar el problema porque ya no tiene sentido seguir guardandolo después de resuelto
api/	GET	Retornar un JSON con un problema de ejemplo

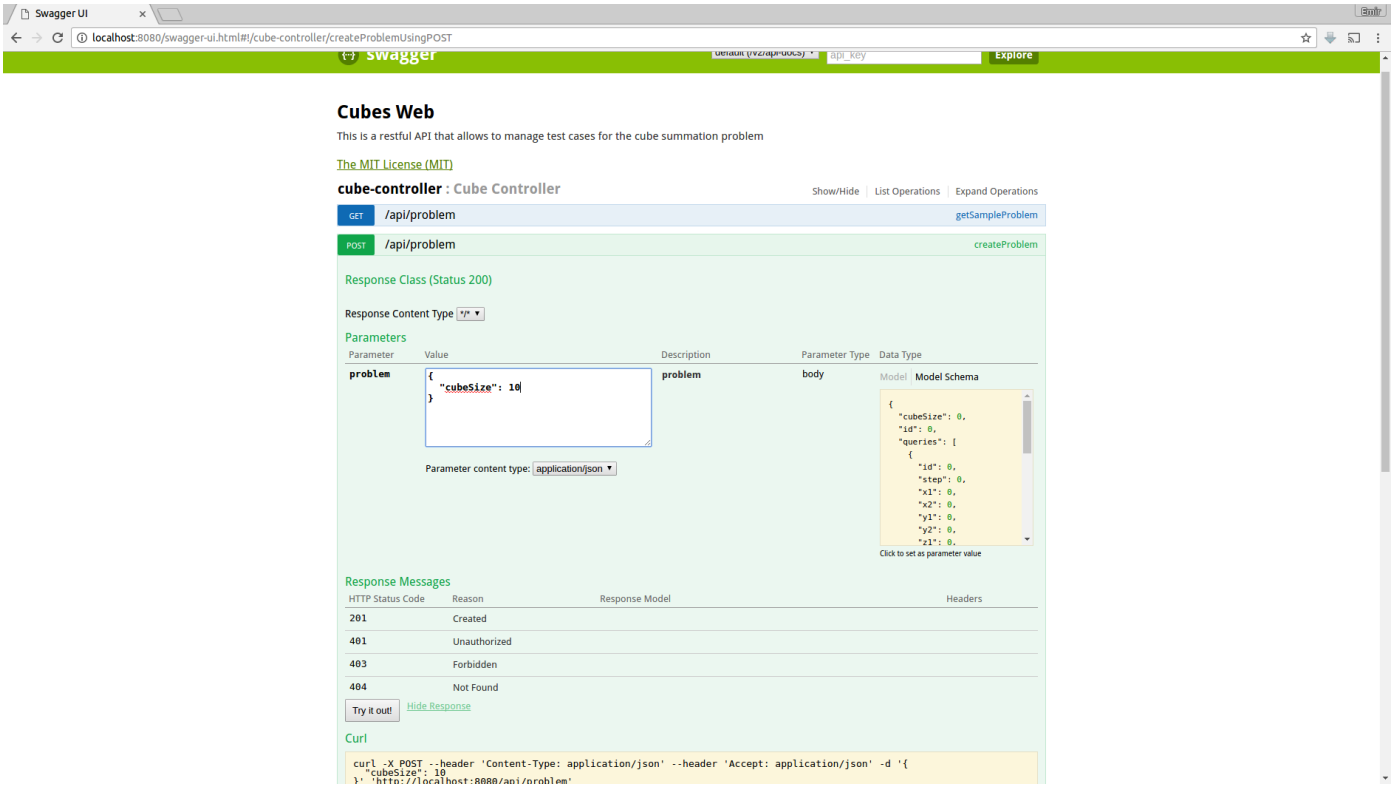
- d211baf: Se agregó la capa de acceso a base de datos
- af2e4ec: Se agregó una capa de servicios que que es invocada por la capa de los controladores
- f50b9f4: Se agregó el servicio que contiene la lógica para resolver los problemas junto con sus pruebas de unidad. En este commit ya se pueden ver las capas de la aplicación. Básicamente es una aplicación multicapa constituida por:
  - Capa de controlador encargada de recibir las peticiones web, dar respuestas e invocar a la siguiente capa, la capa de servicios para que aplique la lógica de negocio.
  - Capa de servicios: Es la encargada de invocar los servicios de la capa de persistencia para recuperar y actualizar los problemas almacenados y también de llamar los otros servicios que saben como resolver los problemas.
  - Capa de acceso a datos: Es a encargada de dar acceso a la base de datos embebida.
  - Capa de modelo: Almacena las clases del dominio del problema. Con el propósito de no extender este documento la ***documentación de cada clase se encuentra en el código***.
- c780da0: Se agregó una UI primitiva que permite crear problemas, hacer consultar y actualizaciones y también resolver problemas usando AngularJS y bootstrap.
- cef6256: En este commit ya se ve una interfaz mucho más pulida con diferentes validaciones, por ejemplo que el problema no pueda ser de n > 100, que cada x, y, z no se mayor que el tamaño del cubo, etc.
- 0801fd0: Se agregó un endpoint al API REST para permitir archivos de texto con el formato especificado en hackerrank.com y obtener su solución:

Endpoint	Verbo	Responsabilidad
api/problem/batch	POST	Procesar un archivo de texto con una suite de casos de prueba y devolver su respuesta

- 0d02812: Se agregó la capacidad de recibir archivos para procesarlos en batch en la UI
- abfca81: Se mejoró la experiencia del usuario en la UI

- 681a4b6: Se agregó soporte para swagger para poder visualizar y probar fácilmente el API REST:



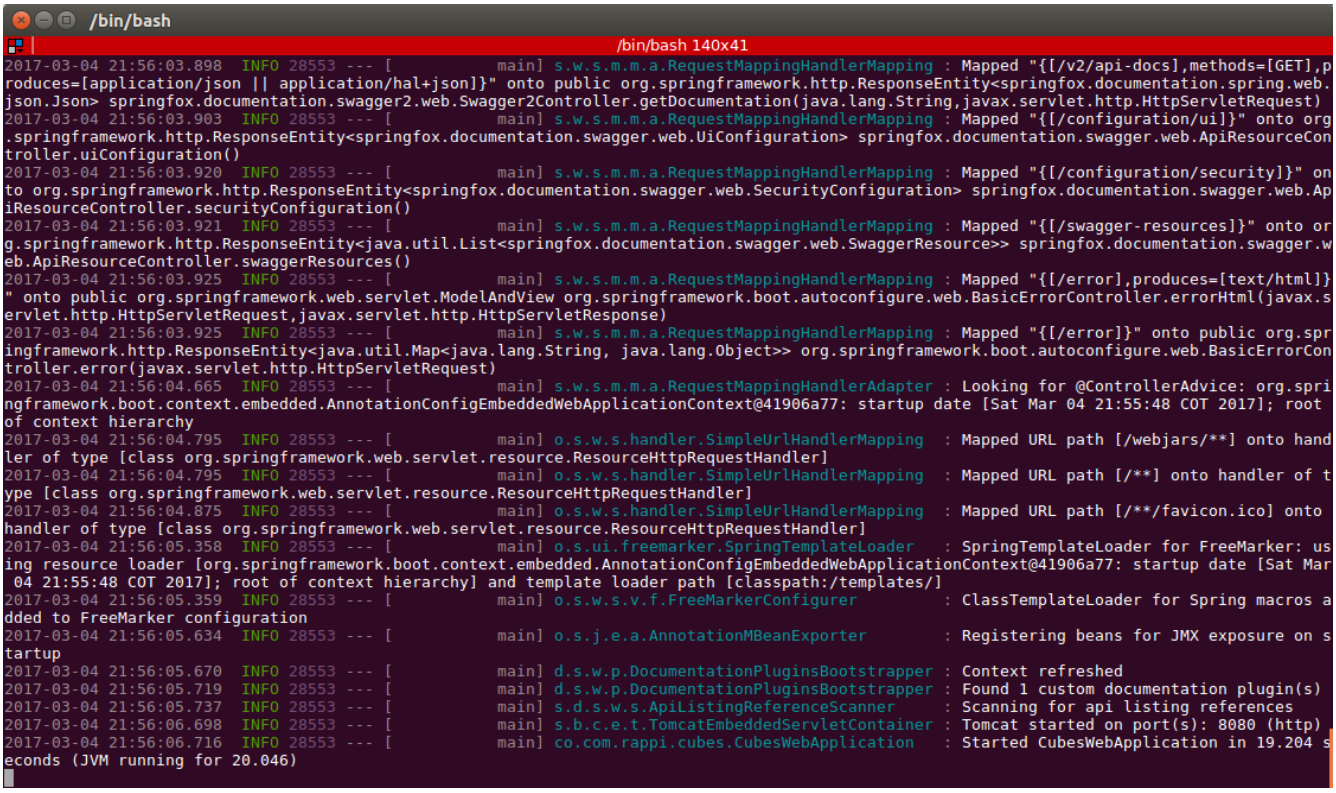


Uso de la aplicación web:

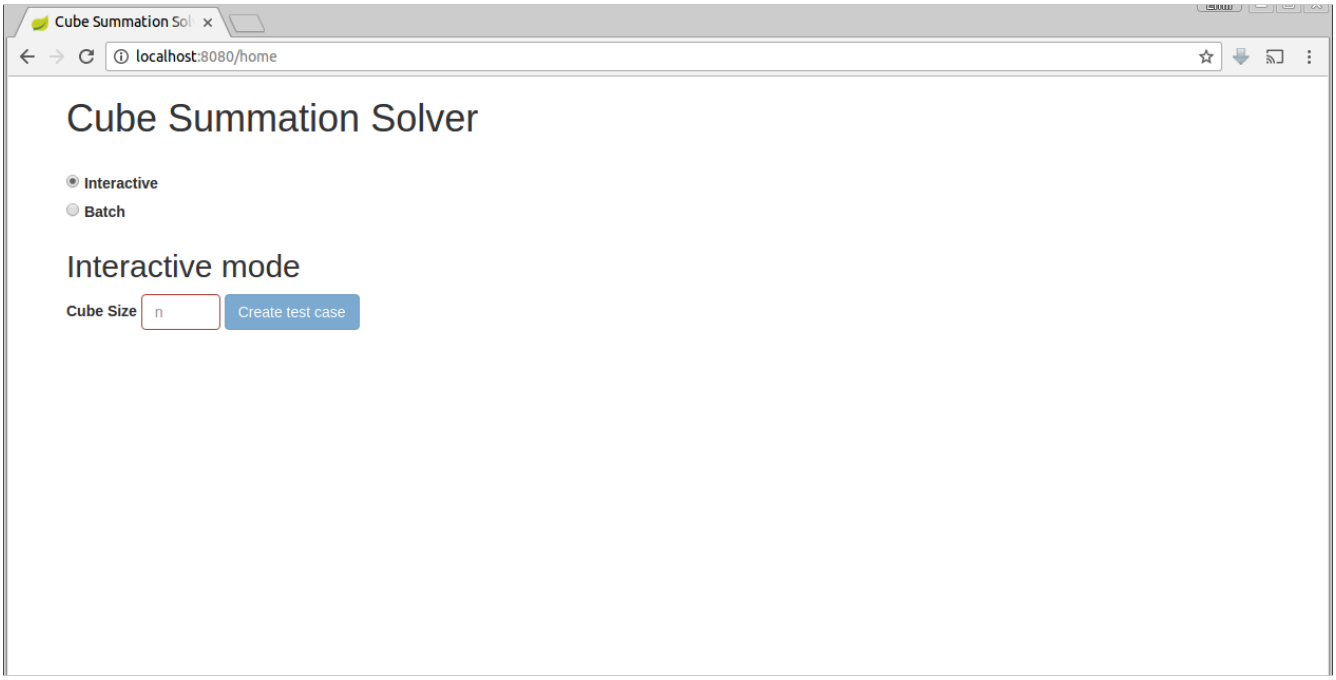
- Para desplegar la aplicación se puede usar el mismo patrón que se usó para desplegar la aplicación de consola, para esto sólo es necesario ubicarse en la carpeta `hackerrank/cubes-web` y luego ejecutar:

```
./mvnw clean install package
```

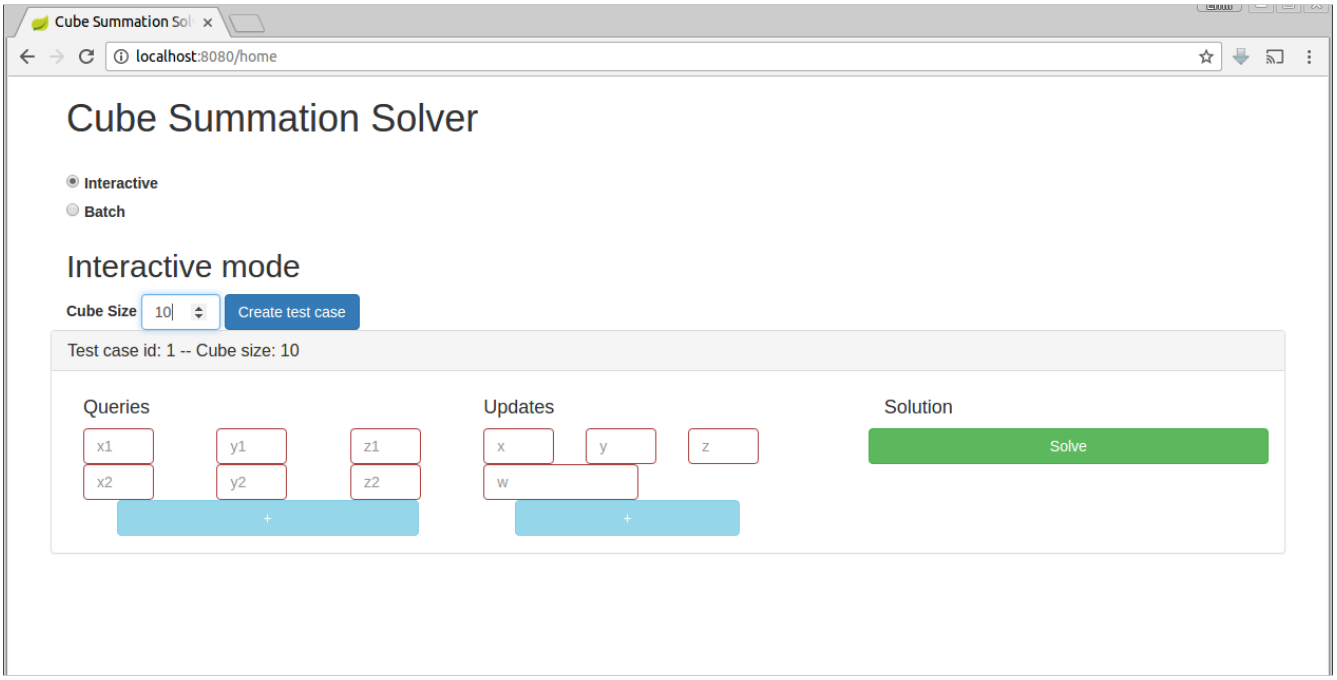
```
java -jar target/cubes-web-0.0.1-SNAPSHOT.jar
```



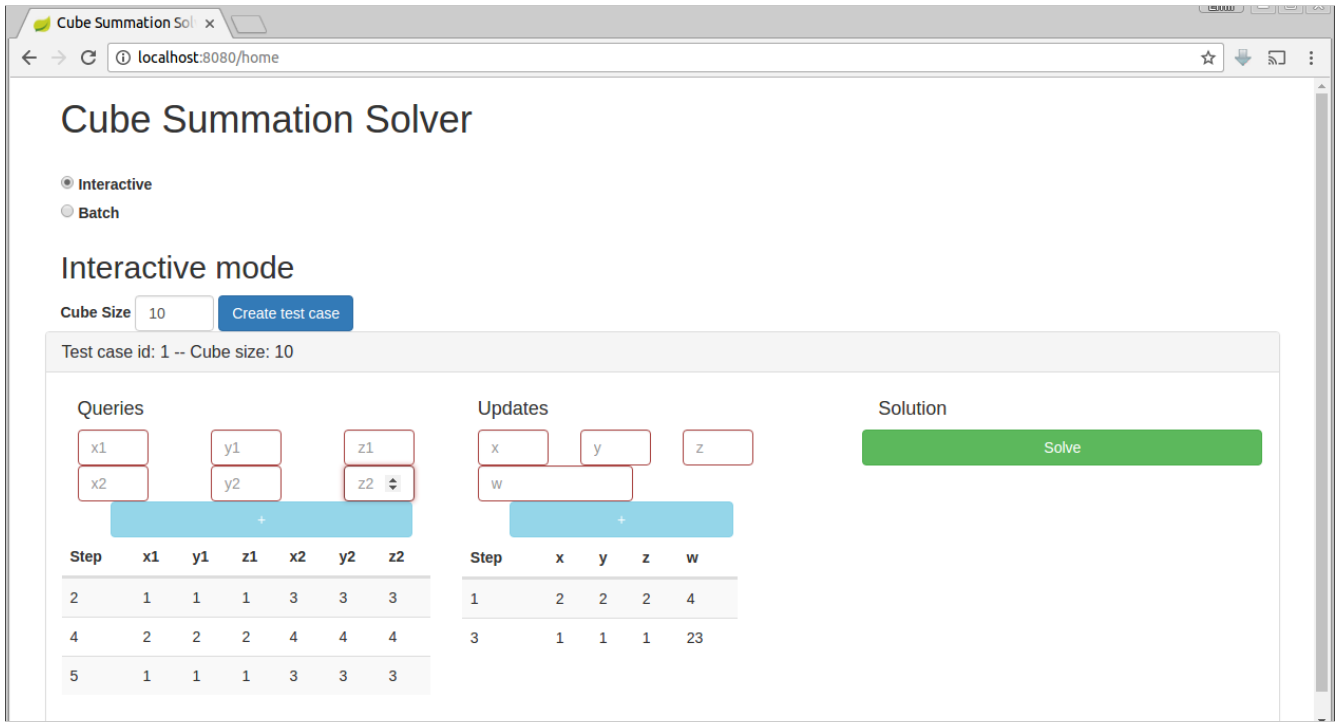
- Enseguida en un navegador web (probado en Chrome) ir la url http://localhost:8080/home, allí se pueden encontrar dos opciones, una para modo interactivo (por defecto) y otra para modo en batch:



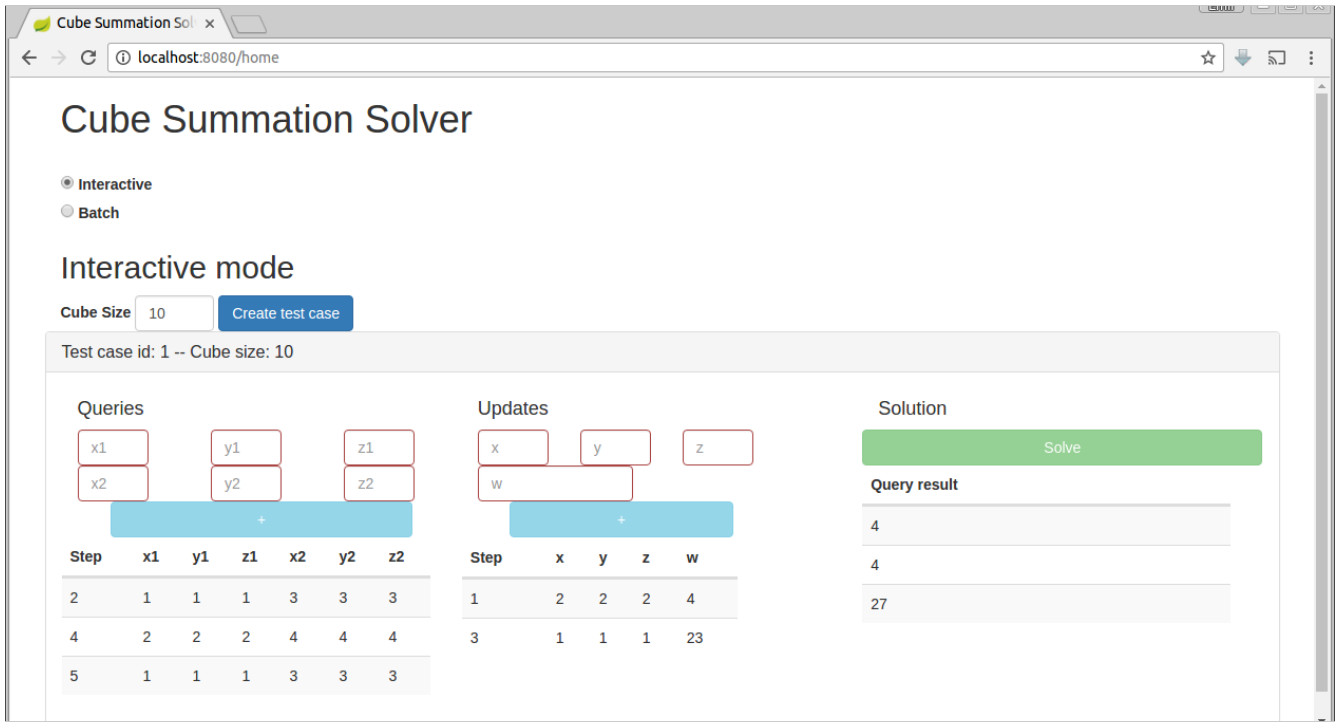
- **En el modo interactivo** lo primero que se debe hacer es ingresar el tamaño, n (siguiendo la notación del problema en hackerrank), del cubo y enseguida se crea un problema al que se le pueden agregar consultas y actualizaciones. n debe estar entre 1 y 100 de no ser así el cuadro de texto se pondrá en rojo y el botón de crear se deshabilitará:



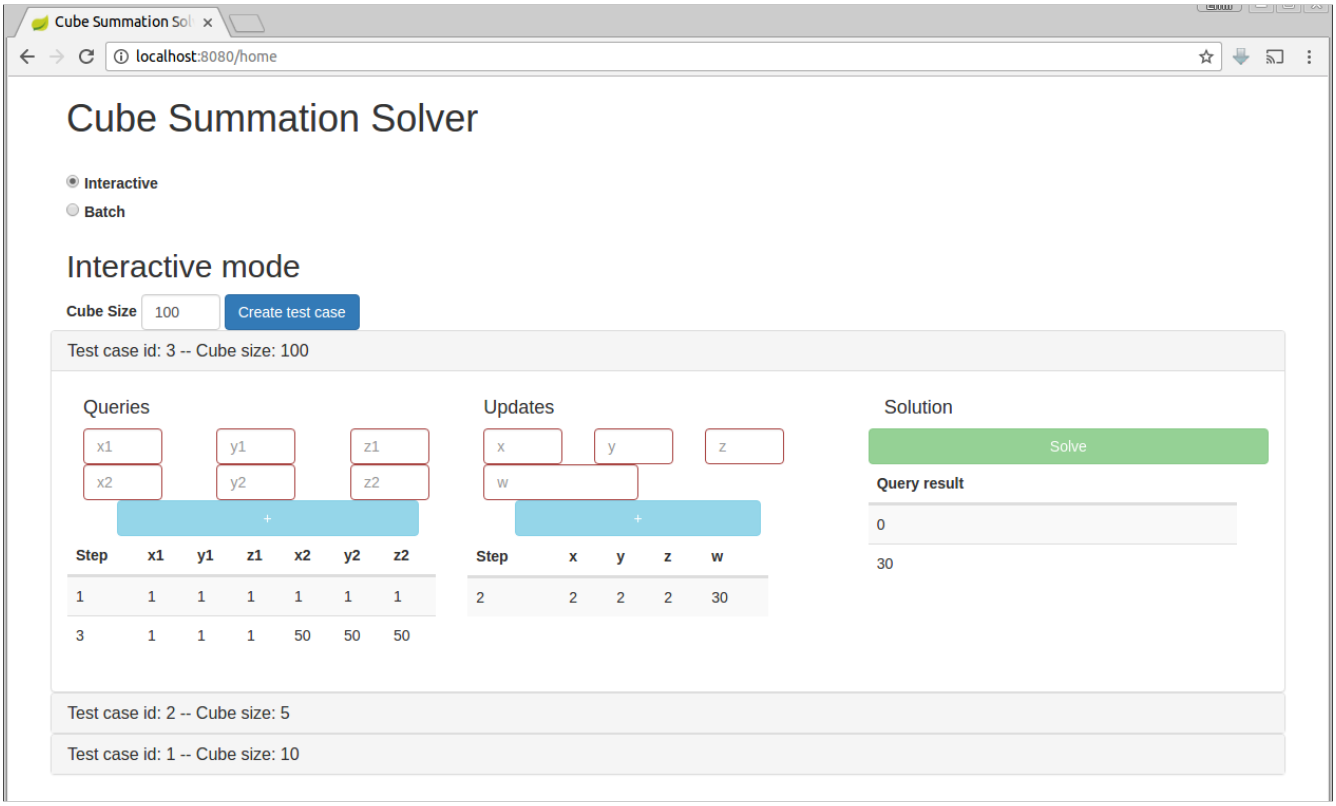
- Al presionar crear se agrega una nueva sección de un acordeon en la que se pueden hacer consultas, actualizaciones y resolver el problema. Todas las coordenas ingresadas son obligatorias para una consulta o para una actualización y deben estar entre 1 y el tamaño del cubo, de otro modo no se pueden agregar y los botones de agregar (+) se deshabilitan.



- Después de agregar todas las consultas y actualizaciones requeridas (en esta versión no hay límite preestablecido de consultas que se puedan realizar) se pude resolver el problema, una vez resuelto el botón de resolver se deshabilita:

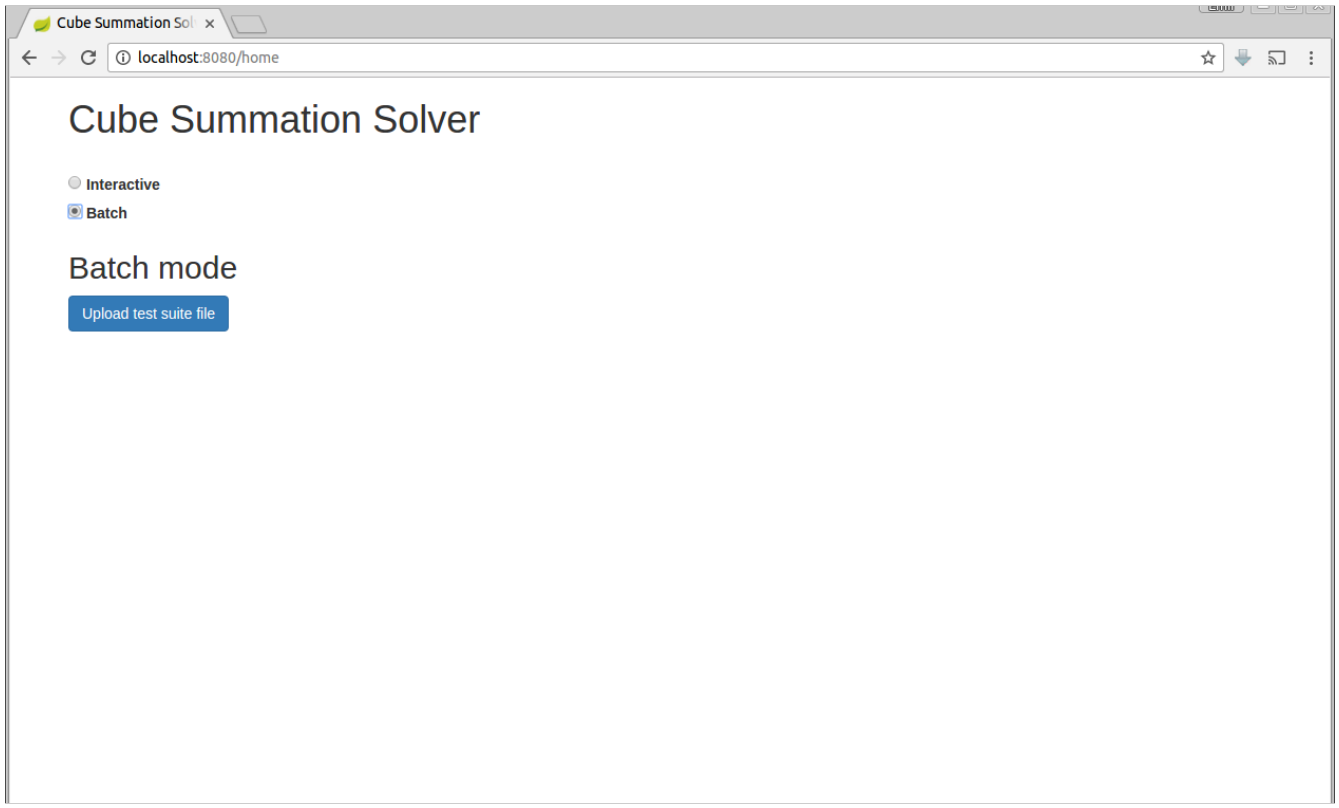


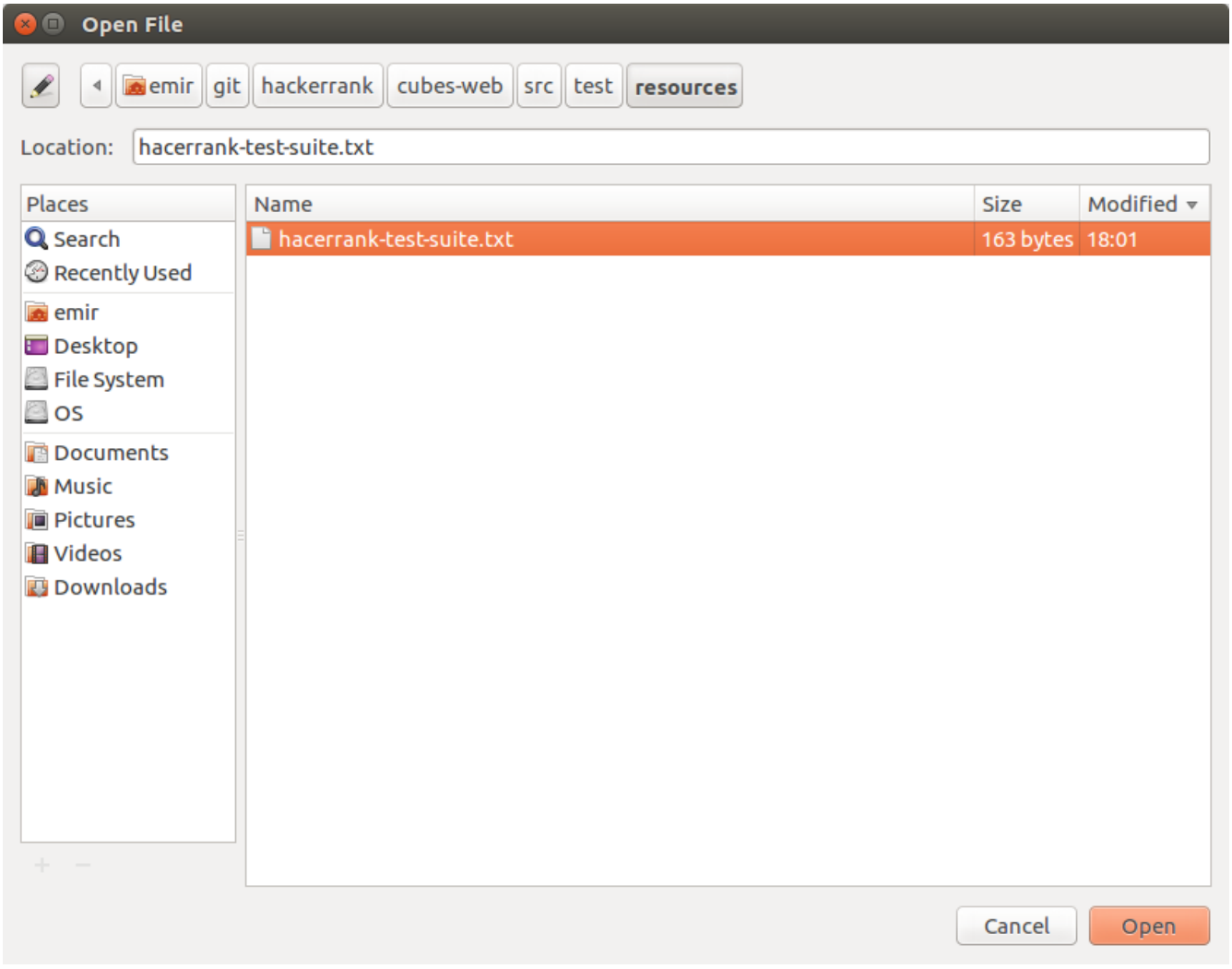
- Se pueden agregar tantas casos de prueba como se desee y no necesariamente se deben resolver antes de agregar uno nuevo. Los casos de prueba se agregan al principio en una nueva sección de acordeon y se ocultan las demas:

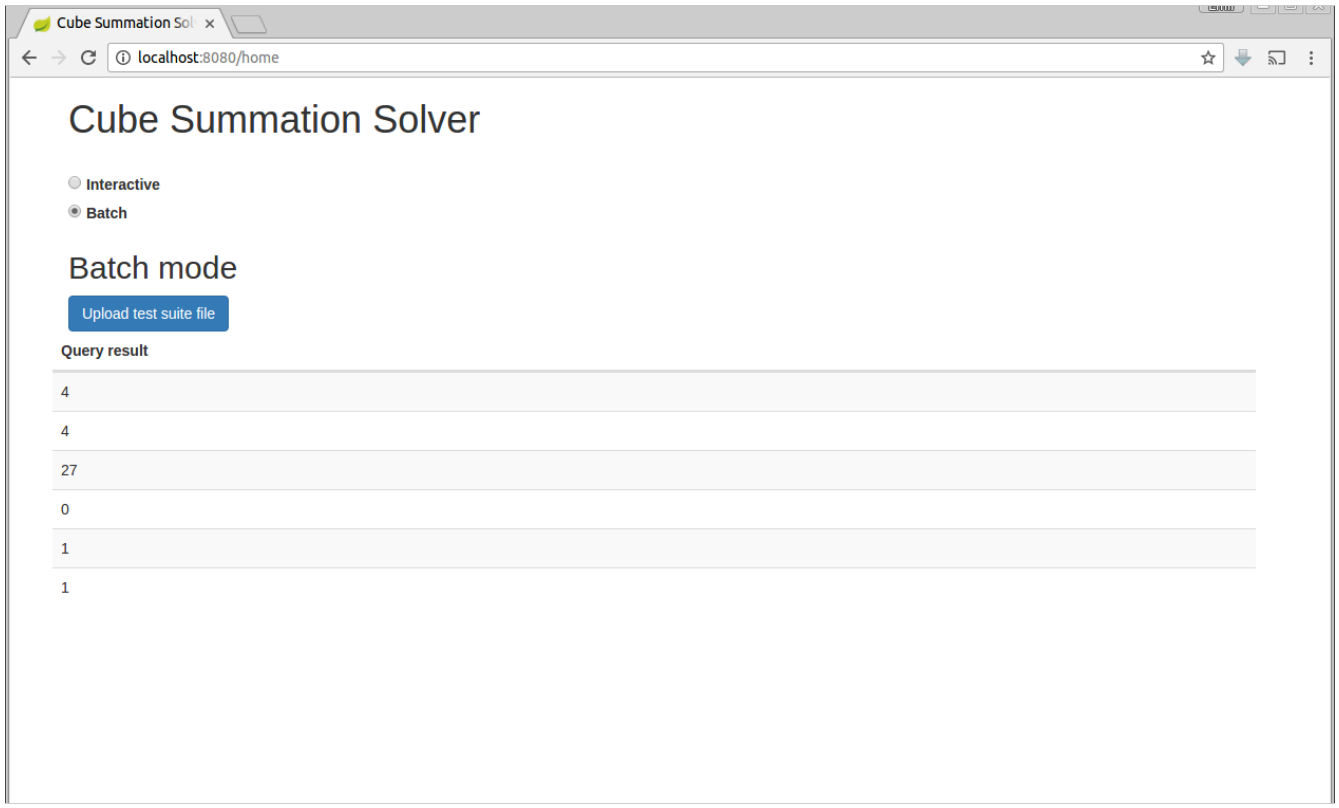


- **El modo batch** es muy sencillo, sólo se debe subir un archivo correctamente formateado para obtener la respuesta. Hay un ejemplo de archivo que se puede usar en:

  `hackerrank/cubes-web/src/test/resources/hacerrank-test-suite.txt`


Cube Summation Sol
localhost:8080/home
Cube Summation Solver
Interactive
Batch
Batch mode
Upload test suite file


Open File
emir
git
hackerrank
cubes-web
src
test
resources
Location:
hacerrank-test-suite.txt
Places
Search
Recently Used
emir
Desktop
File System
OS
Documents
Music
Pictures
Videos
Downloads
Name
Size
Modified
hacerrank-test-suite.txt
163 bytes
18:01
Cancel
Open


Cube Summation Sol
localhost:8080/home
Cube Summation Solver
Interactive
Batch
Batch mode
Upload test suite file
Query result
4
4
27
0
1
1

Code Refactoring

El código presentado tiene como principal problema que todo lo hace en la capa del controlador y eso demuestra que tiene muchas responsabilidades que no debería tener. La capa del controlador sólo debería servir como interfaz entre el cliente que hace la petición y los servicios que saben cómo realizar las lógica de negocio. Los servicios podrían tener a su vez una jerarquía dependiendo de la complejidad de la aplicación pudiendo tratarse de servicios o de fachadas de negocio que hacen llamados a otros servicios y que coordinan diferentes funcionalidades. En este caso particular no habría necesidad de introducir fachadas de negocio porque resultaría superflua. En el servicio debería estar la lógica de negocio y al terminar de realizarla debería llamar otra capa de acceso a datos (DAO) para actualizar las entidades. Otro servicio debería ser el encargado de enviar las notificaciones y la respuesta la debería enviar el controlador.

Preguntas

1. ¿En qué consiste el principio de responsabilidad única? ¿Cuál es su propósito?

El “Single Responsability Principle” es un de los principios fundamentales de los principios SOLID. La idea de este principio es que cada clase y que cada método hagan una sola cosa y que la hagan bien. Evidentemente esa única cosa puede variar de alcance si se trata de una clase o de un método pero la idea sigue siendo la misma, este principio es otra manera de reforzar la alta cohesión del principio alta cohesión y bajo acomplamiento, en este sentido, la intención es que las cosas que haga un método sean independientes de lo demás en el programa y que dentro de él las instrucciones tengan un alcance limitado a un dominio reducido. Otra manera de ver este principio es decir que sólo haya una razón para que los métodos cambien, si el método hace una sola cosa sólo debe haber una razón para cambiarlo. Al ser fieles a este principio se logra código que es fácil de probar porque cada prueba tiene un alcance más reducido y puntual y en caso de errores es mucho más fácil de corregir porque estará limitado a una porción de código que no afecta los demás y es independiente.

2. 2. ¿Qué características tiene según tu opinión “buen” código o código limpio?

Hay muchos rasgos (smells) que hacen que un código demuestre que no está “limpio”, por eso esta respuesta la voy a enfocar más bien a tratar de evitar esos rasgos, de manera que si se está en presencia de un código con pocos o ninguno de esos rasgos lo más seguro es que sea limpio. En concordancia con el punto 1, lo primero es que siga los principios SOLID y en particular el principio de responsabilidad única por las razones ya expuestas.

De otro lado, hay otros principios que el código debería incorporar como DRY (Don’t Repeat Yourself) que básicamente apunta a evitar duplicidad, Least Atonishment Principle que especifica que un API (a cualquier nivel) debería ser claro y que las operaciones que exponen no deberían tener efectos secundarios o la Law of Demeter o principio de menos conocimiento que dice que un método no debería recibir más información de la que necesita.

Pero volviendo a los rasgos negativos, un código “bueno” o “limpio” no debería tener complejidad incidental, es decir que debería tener el nivel de abstracción adecuado para ser entendido rápidamente, por ejemplo, supongamos que se está trabajando en un método que se comunica con un servicio web para traer información de tarifas de vuelos para luego promediar los precios de acuerdo a la distancia

de los vuelos y finalmente los guarda en una base datos, un código “limpio”, al nivel de ese método hipotetico, no debería mostrar los detalles de cómo conectarse al servicio web ni de cómo abrir la conexión a la base datos ni de cómo persistirlo, todos esos detalles deberían estar abstraídos en otros métodos que se puedan invocar.

Otro rasgo negativo, que se debe evitar, es usar número mágicos, es decir números que no expresen claramente la intención, en lugar lo mejor es usar constantes o enumeraciones que muestren el significado dentro del dominio del problema y realmente no sólo aplica para números si no para cualquier literal.

Entre las potenciales fuentes de problemas están los singletons, que tienen un uso claro pero que al no saberlos utilizarlos hacen que el código sea dificil de seguir porque al final lo que hacen es introducir estado glogbal en la aplicación que hace que hacer seguimiento sea más dificil y en ultimas va un poco en contra del principio de encapsulación.

Hay muchos otros rasgos que se pueden encontrar fácilmente en herramientas como sonar que dan claras señale de problemas en el código como la complejidad ciclomática, que no es más que una indicación que un método no tiene una sóla responsabilidad por su tamaño.

En conclusión hay muchas señales que exhiben código malo y que al evitarlas aumentan las posibilidades de tener un código “bueno” o “limpio”, para finalizar quiero mencionar otra señal que hace referencia a no hacer explicitos los “value objects” del negocio y no poner la lógica adecuada dentro de ellos, al no hacerlo se resulta con una arquitectura con un modelo anémico y llena de “clases” que lucen más como servicios que se parecen a funciones en un paradigma imperativo.