**Coding challenge**

1) La aplicación no tiene una capa de seguridad, la vista simplemente muestra un form con valores por defecto que son los del ejemplo de HackerRank para que el usuario ingrese los valores que desee bajo la estructura dada por HackerRank.

La lógica como tal se encuentra en el controlador de test, lo que hace es organizar las entradas en un vector que permite operar fácilmente estas entradas dada la estructura que siguen las mismas y finalmente realizar las operaciones de QUERY y UPDATE requeridas, no cuenta con muchas funciones dado que las operaciones sobre el cubo como tal son sencillas y no considere necesario separar la funcionalidad en más de 2 funciones.

**Me parece importante aclarar que en el controlador dejé unas líneas comentadas**, las cuales están relacionadas todas a lo mismo y es que usé 2 formas para hacer las operaciones de QUERY, iterar sobre el cubo sin importar si sumaba muchos ceros, o como 2da opción sumar solo las posiciones que tenían algún UPDATE (siempre y cuando estuvieran dentro de la zona indicada por los parámetros del QUERY) pero con esta 2da opción no se usaba en si el cubo, como no sé si esto se permitiera, decidí dejar las 2 opciones, dejando sin comentar la opción que entre las 2 hacia menos iteraciones y la que si usaba el cubo pero iteraba mucho más quedó comentada (por si era necesario usar el cubo como tal en el QUERY).

2) No consideré necesario crear clases propias dado que el cubo es una matriz de 3 dimensiones sin propiedades adicionales, la respuesta es una lista sencilla de los resultados de las operaciones realizadas y las entradas como dije anteriormente deben cumplir con una estructura que es fácil de seguir de manera secuencial sin necesidad de hacerles un map a una clase propia (solo las organicé en un vector como dije antes).

**Refactor**

1) Las malas prácticas que encontré fueron:

-Usar la mayoría de variables en español (algunas si están en inglés) cuando las tablas y demás cosas están en inglés.

-Una funcionalidad muy larga dentro del if principal que podría ser realizada en una función (sin que esto afecte la funcionalidad ni requiera hacer más cosas) y así hacer más fácil de leer el código.

-Asigna 2 veces el mismo valor a la variable $servicio.

-La variable $result se guarda pero nunca se usa realmente.

-$pushMessage es una constante que se podría asignar afuera de la lógica para que sea más fácil de cambiarlo por otro mensaje en algún momento o de usarlo en otras partes del código.

-Comentarios que no ayudan a explicar el código, ni permitir intercambiar entre una manera de realizar la funcionalidad y otra (en caso de que se esté probando algo), ni a nada en realidad, simplemente están ahí y ya.

-El if que verifica si el uuid está vacío podría usarse negado para ver si se ejecuta o no la parte de ios y Android 2 en lugar de hacer un return igual al que el código ya iba a ejecutar de todas formas luego del push de ios y Android 2.

2)En el refactor que hice (refactor.txt) traté de darle mi solución a estos problemas nombrando las variables en inglés, creando la función que creí mejoraría la legibilidad del código, ignorando la 2da asignación de $servicio ($service en mi caso), no se guarda la variable result (no se crea realmente), asignando el valor de la constante pushMessage por fuera de la lógica, ignoré los comentarios, e hice que el if que verifica si el uuid está vacío se encargue de determinar solo si se ejecuta lo de ios y Android 2.

**Preguntas escritas**

1) Es que cada clase se encargue de solo una funcionalidad (y lo haga de la totalidad de la misma), su propósito es que una vez dicha funcionalidad esté desarrollada, no se modifique la clase y por lo tanto dicha funcionalidad siga funcionando, válgase la redundancia, de la misma manera (al menos en teoría) así se añadan o no más funcionalidades o procesos a otras partes del código.

2) Que sea fácil de leer (lo cual en mi opinión se facilita dividiendo el código en funciones, y haciéndolo organizado poniendo funcionalidades similares o que dependan unas de otras cerca), un código que no sea acoplado, sino que antes sea fácil de escalar o de adaptar a situaciones similares sin que esto implique hacer muchos cambios (a menos de que desde un principio el proyecto tenga un fin definido, en ese caso no importaría que fuera acoplado si logra llegar a dicho fin).

3) Es una funcionalidad que se encarga de realizar un proceso completo pero pequeño que es solo una parte de un proyecto entero, con el que se debe comunicar de manera estructurada. Su ventaja principal es que permiten hacer proyectos que sean fácilmente escalables ya que en caso de necesitar funcionalidades adicionales se puede desarrollar otro microservicio sin tener que cambiar los ya existentes. Una posible desventaja es que como requiere comunicación estructurada entre sus partes, tal vez para proyectos más pequeños o que de pronto no requiera muchas funcionalidades ejecutándose a la vez represente una arquitectura más compleja de lo que realmente se requiere, o de lo que sería con una arquitectura más interna.