**Challengue Cube Ricardo Mora**

Repositorio que contiene la implementación en Laravel del reto planteado en <https://www.hackerrank.com/challenges/cube-summation>

**Capas de la Aplicación**

**Modelo**

*Cube.php:* Clase encargada de la creación, procesamiento (actualizaciones y consultas) y almacenamiento del cubo.

**Controlador**

CubeController.php: Controlador encargado de procesar las peticiones del usuario, contiene la lógica de la solución de problema, usa la clase Cube.php.

**Vista**

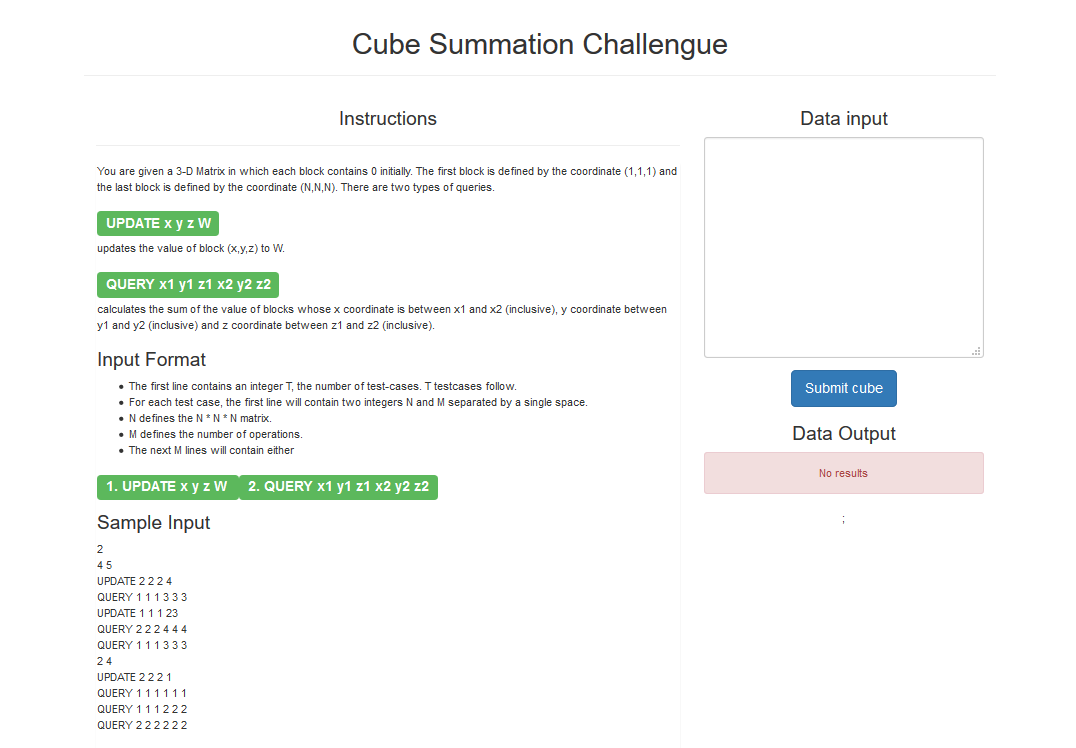
Template.blade.php: Plantilla de la aplicación contiene los estilos y dependencias.

Index.blade.php: Ventana creada con el fin de generar al usuario una vista amigable y uso de la aplicación.

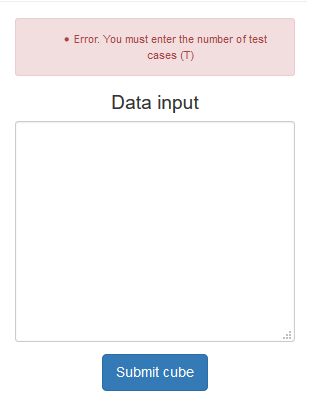
deserror.blade.php: Sección creada para desplegar los mensajes de error resultante de las validaciones.

**Demo**

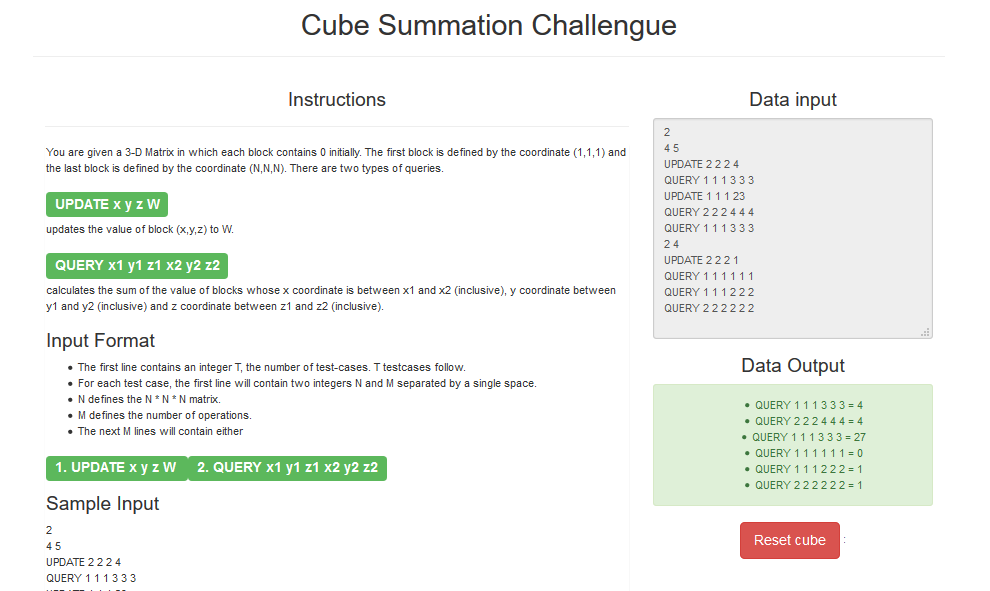
**Demostración entrada de Datos**



**Manejo de Errores**



**Salida de Datos**



**Code Refactoring**

Luego de observar la implementación del código observe varias malas prácticas y realice las siguientes correcciones:

El valor Input::get('driver\_id') es usado múltiples veces, lo mejor es declarar la variable $idDriver = Input::get('driver\_id');

Variables en español y en inglés, se debe mantener uno solo, el más común es el inglés $servicio por $service

Se puede usar !empty() en vez de != null y es más legible.

La documentación del código debe estar al inicio del método, donde se puede especificar los parámetros de entrada de la función y la respuesta esperada, no dentro del código ya que el mismo debería ser entendible por sí mismo.

No se debería tener código comentado, ni variables de error como el helper de laravel dd() al pasar a producción.

Se creó una nueva función para la lógica del envío de las notificaciones, para separarla de método que realiza la confirmación del servicio, debido a que ejecutan acciones diferentes, y se podría invocar desde otro método que requiera enviar notificaciones reusando el dodigo.

El código refactorizado lo puede encontrar en la siguiente ruta del proyecto

**/app/Http/Controllers/DriverController.php**

**Preguntas**

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

Este principio se refiere a que ***una clase, método u objeto debe tener una y solo una única función***, esto nos ayuda en cierto forma proteger el código frente a los cambios ya que solo debería haber un motivo para modificar una clase o método, si un método o clase contiene muchas funciones aumentaremos las razones para modificarlos.

1. ¿Qué características debe tiene según tu opinión código "bueno" o código limpio?

En mi opinión un código limpio debe tener las siguientes características:

* Debe ser legible y expresar su función u objetivo al verlo.
* Mantener los estándares globales para el desarrollo de aplicaciones, usar patrones de diseño, todo esto con la finalidad de que cualquier persona con los conocimientos pueda entenderlo.
* Debe ser conciso y sin redundancias.
* Fácil de mantener y editar