PRUEBA TÉCNICA PARA EL BACKEND DE RAPPI

Ernesto Soto Meléndez
CC 1.015.429.099

Bogotá D.C.
Abril 24 2017

Contents

Coding Challenge

Para este reto se realizó una aplicación web bastante sencilla, sin capa de persistencia ya que no es necesario guardar o cargar información, sólo procesar la entrada e imprimir la salida; además de las pruebas unitarias.

La organización de los archivos es la definida por el framework Laravel, que permite el desarrollo ágil utilizando PHP.

Capas de la aplicación

Como ya se mencionó, la aplicación no cuenta con una capa de Persistencia. Por tanto, se explicarán brevemente la capa de presentación (interfaz gráfica) y la capa de lógica.

Presentación

La capa de presentación se compone de 1 *layout* y 2 vistas que lo extienden. La primera, es el medio para la entrada de los datos.

Ilustración 1 - Vista home

La segunda es la vista de resultado, que muestra la salida de una sumatoria de cubos.

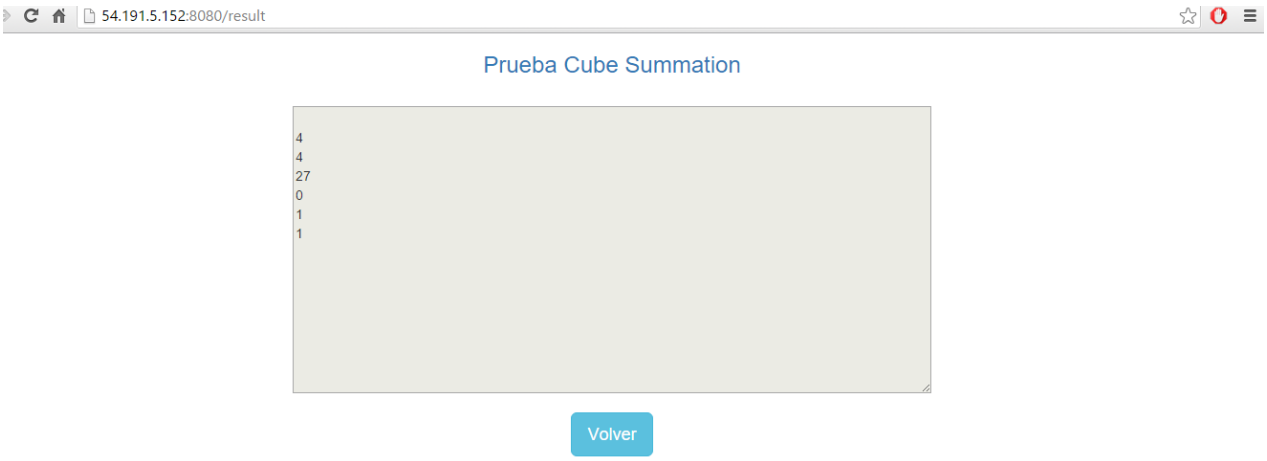

Ilustración 2 - Vista result

Finalmente, en caso que la entrada tenga un formato incorrecto, generará un error; el cual, del mismo modo que el resultado, se informa en la segunda vista, que utiliza la misma ruta del resultado.

Ilustración 3 - Vista error

En cuanto a los estilos, se utilizó Bootstrap para dar algo de color y no dejar la interfaz tan plana para la actual prueba. Estos se encuentran en la carpeta “/public/assets/” del proyecto.

En esta capa existía la posibilidad de utilizar un solo archivo para ingresar los datos de entrada e imprimir el resultado, pero para lograr mayor desacoplamiento y menos responsabilidades en cada componente se utilizaron vistas separadas.

Lógica / Aplicación

Esta capa depende de los controladores y la configuración de las rutas. En el archivo route.php se especifican las rutas del sistema y en el controlador CubeController que se creó, se desarrolla la lógica para dichas rutas.

La ruta *home*, únicamente debe desplegar la vista a través del método GET, sólo ingresando la ruta de la página inicial; mientras que la ruta *result*, que se invoca utilizando el método POST, recibe la entrada para realizar la suma de los cubos.

Para hacer más eficiente el procesamiento, en lugar de utilizar matrices, recorridos complejos y ocupar memoria innecesariamente, se basa el cálculo en un diccionario de datos. El diccionario tiene como llave una composición de las 3 posiciones de un elemento, formateadas a 3 dígitos cada una. El valor en cada elemento del diccionario se asigna cuando se realiza una operación UPDATE.

De esta forma, para realizar una operación QUERY, se recorren los valores del diccionario y se suman los que estén dentro del rango especificado, lo que acorta el recorrido, mantiene la complejidad lineal para la operación, y optimiza el proceso.

Por otro lado, cabe resaltar que, debido a que el procesamiento se realiza directamente a partir de los datos de entrada, y no se requiere almacenamiento, no se utiliza una capa de datos.

Repositorio

Para la integración continua del desarrollo se utilizó Git. El código se mantiene integrado y versionado en un repositorio de GitHub, que se encuentra en la siguiente ruta: https://github.com/sotomelendez/ernesto-soto-back

Creo además, que lo más importante para el buen uso de un sistema de manejo de versiones, es siempre subir código funcional y verificado. Para esto también son útiles las herramientas de integración continua que verifiquen compilación y pruebas unitarias del código que se sube al repositorio.

Pruebas

Dentro de la carpeta *tests* del proyecto se creó el archivo SummationTest.php para la implementación de las pruebas unitarias del proyecto, con base en PHPUnit.

Adicionalmente, se realizaron pruebas de la lógica implementada directamente sobre la página del reto
(), utilizando varias configuraciones diferentes en los datos de entrada y verificando en todos casos los de salida.

Además, se hicieron otros tipos de pruebas. Primero, pruebas de interfaz locales (de caja blanca, teniendo el código a la mano y modificándolo de ser necesario), verificando las vistas y los datos de salida obtenidos para cada entrada.

Luego, pruebas de caja negra o de usuario, donde no se ve el código, desplegando la aplicación en un servidor externo (para el caso, se utilizó una instancia EC2 de Amazon) y verificando el funcionamiento.

Finalmente, pruebas se usuario (ajeno a la implementación), explicando el modelo y pidiendo a un conocido (matemático de profesión) realizar pruebas sobre la aplicación desplegada, luego de explicarle el comportamiento del sistema.