GRABILITY (Mobile Retail, Reinvented)

El siguiente documento contiene la solución a la prueba de conocimientos planteada por la empresa Grability para el cargo desarrollador BACKEND. Consta de 3 secciones para cada una de las cuales se presenta una solución.

Documento presentado por Camilo Alberto Zapata Martínez, Ingeniero de Sistemas y MBA de la Universidad Nacional de Colombia. Cualquier inquietud adicional por favor contactar vía email a cazm013@gmail.com o vía móvil 3112589619.

Fecha: 15/06/2016

Sección 1 – Desarrollo aplicación

Introducción

Se desarrolló una aplicación en el lenguaje PHP planteando una solución al problema de tipo “Coding Challenge” de la página Hackerrank. El problema consiste en desarrollar 2 operaciones sobre una matriz 3D con base en algunos parámetros en entrada. La solución propuesta es totalmente web incluyendo entradas y salidas. Además, se utilizó un servidor SVN para llevar el control de versiones de la aplicación.

Descripción de la aplicación

La aplicación web fue desarrollada en PHP y HTML. Se compone de 3 elementos: index.php, NCasos.php y Matriz.php. Se implementó una arquitectura de 2 capas: VISTA y MODELO.

No se consideró necesario incluir una capa intermedia encargada de la LÓGICA de la aplicación, ya que basados en los requerimientos del algoritmo, la solución era sencilla y no necesitaba una gran implementación a nivel de código. En el caso de haber incluido esta capa de LÓGICA se hubiera aplicado a cabalidad la reconocida arquitectura MVC (Modelo-Vista-Controlador).

En capa **MODELO** se implementó una clase, acorde con el paradigma de programación orientada a objetos, por medio de la cual se modelo una entidad capaz de gestionar los datos requeridos así como las funcionalidades solicitadas. La clase cuenta con 2 atributos, uno para el tamaño de la matriz (N) y otro para la matriz en si (representada por un arreglo multidimensional). Se creó un método constructor el cual recibe un parámetro para el tamaño de la matriz. Además, se crearon 2 métodos los cuales realizan las 2 funcionalidades principales de la aplicación:

- ✓ UPDATE: Actualiza el valor de una posición X,Y,Z en la matriz con el valor W.

- ✓ QUERY: Realiza una sumatoria de los valores desde la posición X_1 Y_1 Z_1 hasta X_2 Y_2 Z_2

El orden establecido para recorrer la matriz se definió de manera secuencial, primero sobre el eje X, luego sobre el eje Y, luego sobre el eje Z. Es decir, Para ir desde la posición (1,1,1) hasta la posición (2,2,2) el algoritmo pasa por las siguientes coordenadas de manera secuencial: (1,1,1) (2,1,1) (3,1,1) (1,2,1) (2,2,1) (3,2,1) (1,3,1) (2,3,1)(3,3,1) (1,1,2) (2,1,2) (3,1,2) (1,2,2) (2,2,2).

Por otra parte, en la capa **VISTA** se desarrolló todo lo referente a la interacción con el usuario final y presentación de la aplicación. Se utilizó tanto código HTML integrando código PHP para darle funcionalidad a la página. Se implementaron 2 componentes: index.php y NCasos.php.

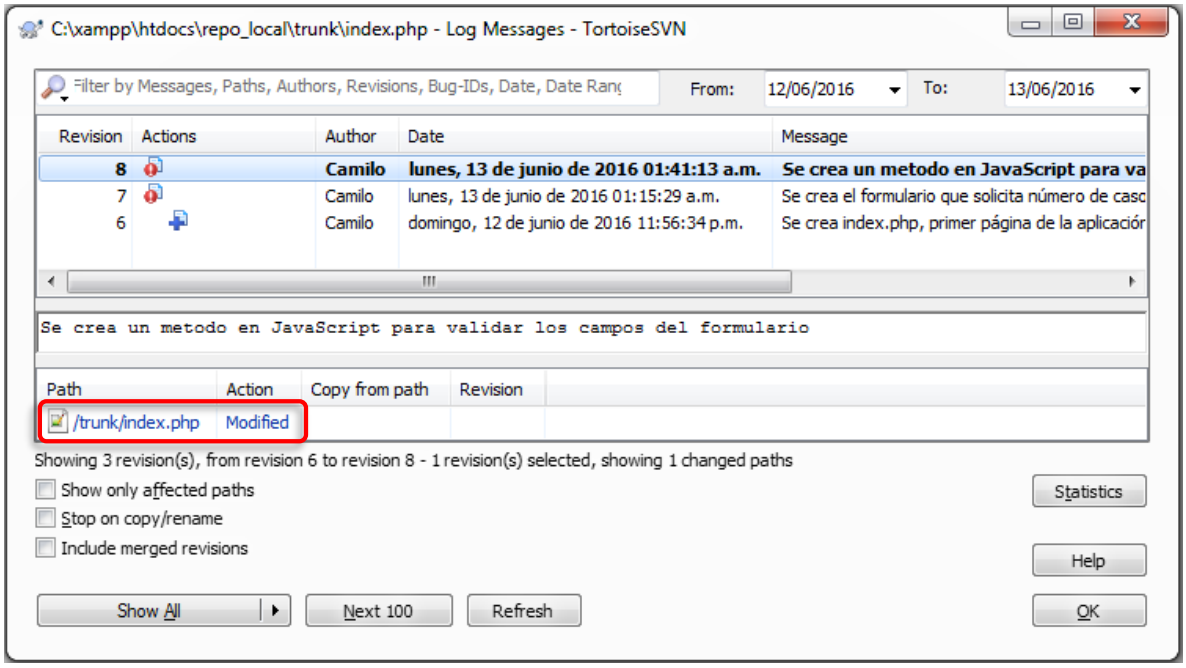
Index.php incluye un formulario para capturar el número de casos definidos por el usuario. Antes de enviar el formulario, por medio del método POST, se valida que este valor no esté vacío y que se encuentre dentro del rango definido por el problema [1 - 50]. El formulario llama al componente NCasos.php.

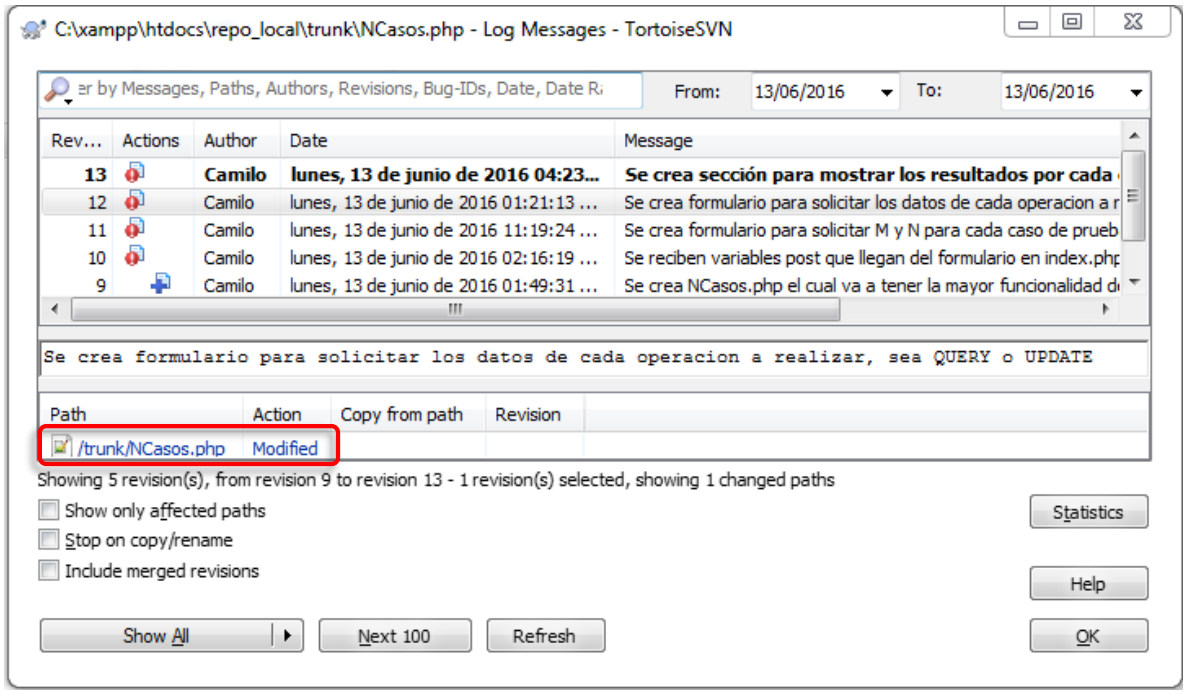
NCasos.php contiene tanto el ingreso de los datos para aplicar las diferentes operaciones sobre la matriz, como el resultado o salida del programa. Consta de 2 formularios independientes, el primero solicita tamaño de la matriz y número de operaciones a realizar y el segundo solicita los datos de cada operación para cada caso T. Finalmente se muestra la salida para cada uno de los casos y un resumen que contiene la salida para todos los casos (emulando la salida sugerida por el planteamiento del problema).

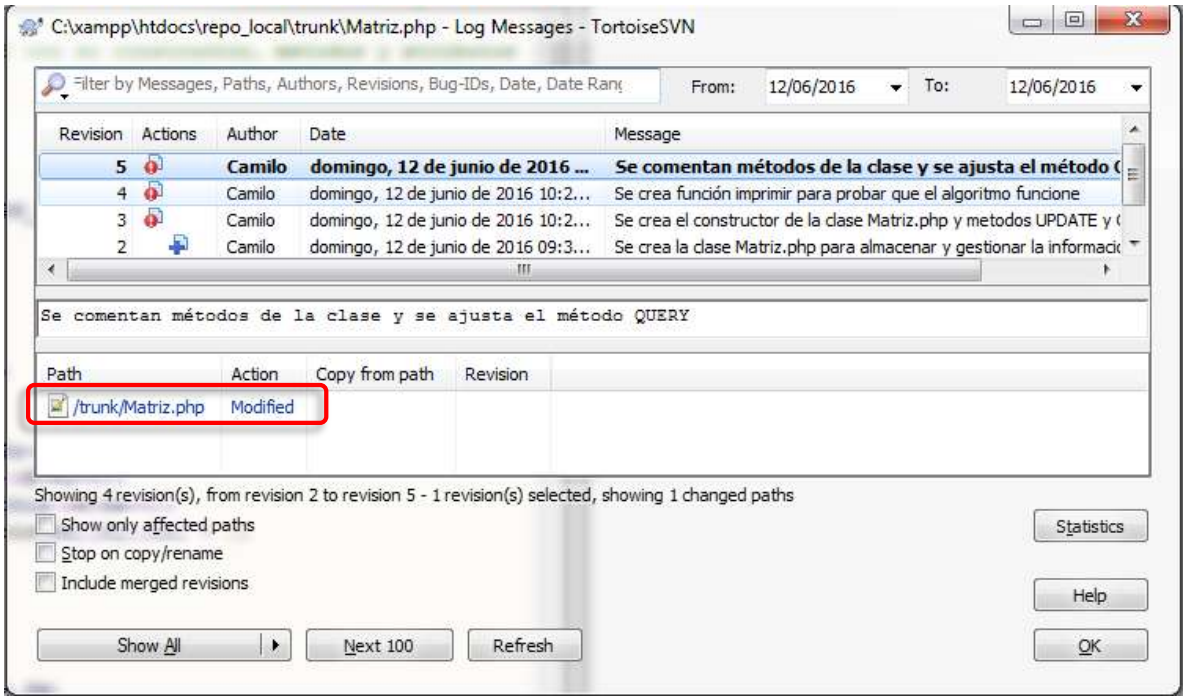
Puntos adicionales

1. Como ya se describió, la aplicación es totalmente web incluyendo entradas y salidas.

2. Se aplicó el concepto de control de versiones por medio de un cliente SVN (Tortoise SVN). A continuación los pantallazos los cuales muestran los cambios para cada componente:






C:\xampp\htdocs\repo_local\trunk\Matriz.php - Log Messages - TortoiseSVN
Filter by Messages, Paths, Authors, Revisions, Bug-IDs, Date, Date Rang
From:
12/06/2016
To:
12/06/2016
Revision
Actions
Author
Date
Message
5
Camilo
domingo, 12 de junio de 2016 ...
Se comentan métodos de la clase y se ajusta el método Q
4
Camilo
domingo, 12 de junio de 2016 10:2...
Se crea función imprimir para probar que el algoritmo funcione
3
Camilo
domingo, 12 de junio de 2016 10:2...
Se crea el constructor de la clase Matriz.php y metodos UPDATE y
2
Camilo
domingo, 12 de junio de 2016 09:3...
Se crea la clase Matriz.php para almacenar y gestionar la informaci
Se comentan métodos de la clase y se ajusta el método QUERY
Path
Action
Copy from path
Revision
/trunk/Matriz.php
Modified
Showing 4 revision(s), from revision 2 to revision 5 - 1 revision(s) selected, showing 1 changed paths
Show only affected paths
Stop on copy/rename
Include merged revisions
Statistics
Help
Show All
Next 100
Refresh
OK

Sección 2 – Refactoring

Luego de revisar el código en la sección 2, se proponen algunos ajustes a nivel de código así como su justificación de por qué dicho cambio favorece la simpleza y legibilidad del código.

- ✓ El llamado para obtener el parámetro se hace 3 veces lo cual es ineficiente y se puede mejorar asignando este valor a una variable al inicio del segmento de código de la función para que almacene este valor.
    - $id_driver = Input::get('driver_id');
- ✓ De acuerdo al paradigma de programación orientada a objetos, los atributos de una clase deben tener un alcance privado y su valor debe ser consultado o modificado mediante métodos getter y setter (Los cuales si tienen alcance público). Es decir que se modifican la forma de acceder a estos atributos en las clases que intervienen, por ejemplo:
    - $servicio->status_id ⟹ $servicios->getStatus_id()

  Lo anterior conduce a hacer la modificación respectiva de la clase, lo cual se sale del alcance de este ejercicio.

- ✓ Para comprobar si una variable u objeto es nulo es recomendable usar la función isset(). Es decir, se realizan los siguientes cambios:
    - $servicio != NULL ⟹ isset($servicio)
    - $servicio->driver_id == NULL ⟹ !isset($servicio->driver_id)
- ✓ Se eliminan los el segmento de código que se encontraba comentado, luego de la sección en donde se notifica el usuario, ya que aparentemente es una versión anterior del segmento de código posterior por lo tanto no es necesario conservar dichos comentarios. Los comentarios son muy útiles en etapa de desarrollo, permiten tener a la mano segmentos de código que puede ser necesario tener a la mano pero una vez se tiene una versión estable lo correcto es eliminar estos comentarios para simplificar el código.

- ✓ Se comentan algunas secciones del código solo con el fin de explicar brevemente en lenguaje cotidiano que hace cada sección. Lo anterior es útil cuando diferentes personas trabajan sobre un mismo segmento de código. Es mucho más fácil entender que hace cierto código en lenguaje cotidiano que analizar el mismo código.

- ✓ Finalmente se indentó el código, agregando algunos espacios entre secciones, haciendo más legible el código.

El código refactorizado se encuentra en el archivo refactor.php cuyo directorio es el mismo que este archivo (documento.pdf)

Sección 3 – Preguntas adicionales

1. El principio de responsabilidad única es uno de los principios fundamentales enunciados por el paradigma de la programación orientada a objetos POO. Consiste en que cada componente, módulo o clase debe cumplir una única y especifica responsabilidad. Si en el modelamiento de un problema, una clase cuenta con varias responsabilidades, es necesario segmentar la clase dejando únicamente la responsabilidad principal, mientras que las otras se pueden modelar en clases distintas. Esto permite que el código sea modular, es decir, si algún elemento necesita ser ajustado, los cambios solo se reflejaran en ese elemento si impactar al resto.

2. Un código limpio, en mi opinión, es aquel código que permite ser leído de forma sencilla. La legibilidad es muy importante en términos del mantenimiento del software. Una solución no solo debe satisfacer la necesidad puntual en el momento de ser entregada, sino debe ser susceptible de obtener el mantenimiento y ajustes posteriores normales en el ciclo de desarrollo. Para que otras personas puedan entender el código, es necesario que se encuentre debidamente indentado, que se sigan estándares para nombrar variables, métodos, clases, literales, etc. Los nombres deben ser nemotécnicos para facilitar su recordación y entendimiento. Además, debe ser modular, separado en cada una de las capas según la arquitectura seleccionada.

   Desde un componente de vista, evitar hacer implementaciones de clases y métodos pertenecientes a la capa de MODELO y LOGICA respectivamente. También considero necesario realizar una capa de ACCESO A DATOS de tal manera que se evite hacer

transacciones en la base de datos con lenguaje SQL desde capas posteriores como la VISTA.

El código debe poder reutilizarse lo más que se pueda y así evitar redundancias. Por ejemplo, si es necesario desarrollar una misma funcionalidad en varias secciones del código, lo correcto es crear un método y solo hacer el llamado tantas veces como sea necesario. Entre más reducido el código, mejor.