Nombre: Wilson Acevedo Samaniego

Coding Challenge

Primera parte el ejercicio de Cube Summation lo quise plantear de forma web, teniendo en cuenta que puedo usar cualquier mecanismo de entrada y salida, analice el problema y para solucionarlo lo dividí en 3 partes.

1. Crear una matriz de 3 dimensiones y llenarlo con 0.
2. Actualizar la matriz indicando las coordenadas w, x, y con un valor en Z.
3. Consultar que deberá sumar todos los valores dentro de la matriz.

Para cada una de las partes pensé en crear un formulario con sus respectivas validaciones en la vista y en controller y los formularios ingresan por el método post, dentro de los formularios existe un campo tipo ("hidden") el cual usare para indicar que acción debe efectuar.

La salida va ser en la vista, con la Session una para la matriz y la otra para el valor de la sumatoria de la matriz.

Las pruebas se realizaron con Selenium y todo el proyecto se subió a repositorio en bitbucket.

Pasos realizados para la solución:

- Se crea un proyecto en laravel 4.2
- Se crea un repositorio en git y se sube el proyecto.
- Template de Boostrap
    - Manejo de blade en las vistas.
- Creación de formulario.
- Creación de Rutas.
- Creación de Controller.
- Validación de datos.
- Implementación de formularios.

1) Las capas de la aplicación (por ejemplo capa de persistencia, vista, de aplicación, etc) y qué clases pertenecen a cual

Por el tipo de naturaleza del ejercicio solo maneje 2 capas aplicación y vista.

Aplicación: Se crea un archivo dentro de la carpeta controllers llamado CubeController.php

En cual cuenta con 4 funciones.

- Función createM : El cual valida en primera instancia si la función llamada es por el método post o get , si es por get retorna a la vista, si es por post entonces primero valida el valor de input de tipo hidden , después utiliza el validator y si pasa entonces va llamar la siguiente función.
- Función crear: Esta va recibir una matriz y debe llenarla con ceros y devuelve una matriz llena de ceros.
- Función actualizar: Recibe la matriz, las coordenadas y valor a actualizar dentro de la matriz y devuelve la matriz con el nuevo valor ingresado.
- Función contar: Recibe la matriz y el tamaño de la matriz, realiza una sumatoria de todos los valores y devuelve un entero con el resultado.

Vista : la vista se llama matriz.blade.php la cual hereda de main.blade.php y en la cual tiene 3 formularios con sus respectivos campos y validaciones pertinentes.

Las capas de Modelo y Persistencia: No se tiene porque no se interactúa con la Base de datos.

2) La responsabilidad de cada clase creada

La clase CuboController no cumple con el principio de responsabilidad única, porque no tiene una única razón de cambio, porque tiene 2 realmente el de crear y el actualizar explicadas en el punto 1.

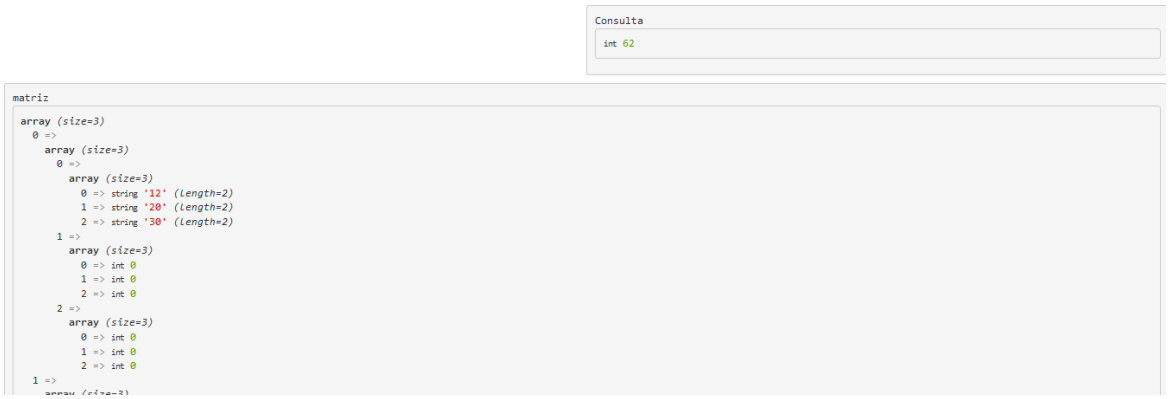
Puntos Extra:

1) El mecanismo usado para la entrada y salida de datos es Web (5 puntos)

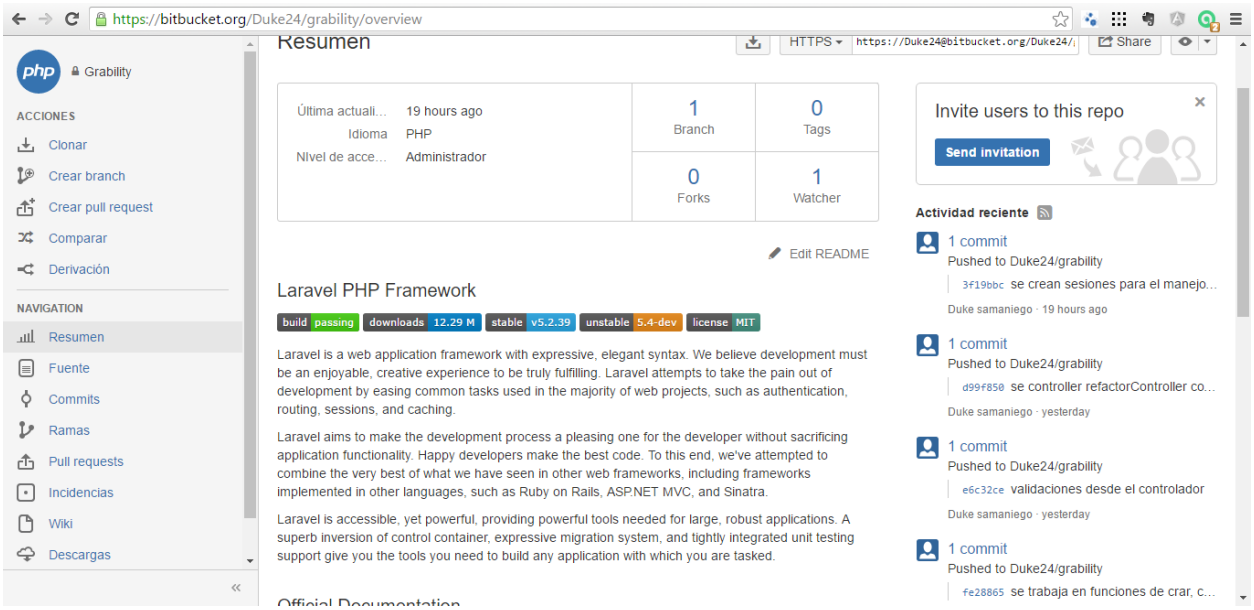
Se utiliza vista, con 3 formularios para la entrada y se pintan los resultados en la vista como salida de datos web.

Entrada

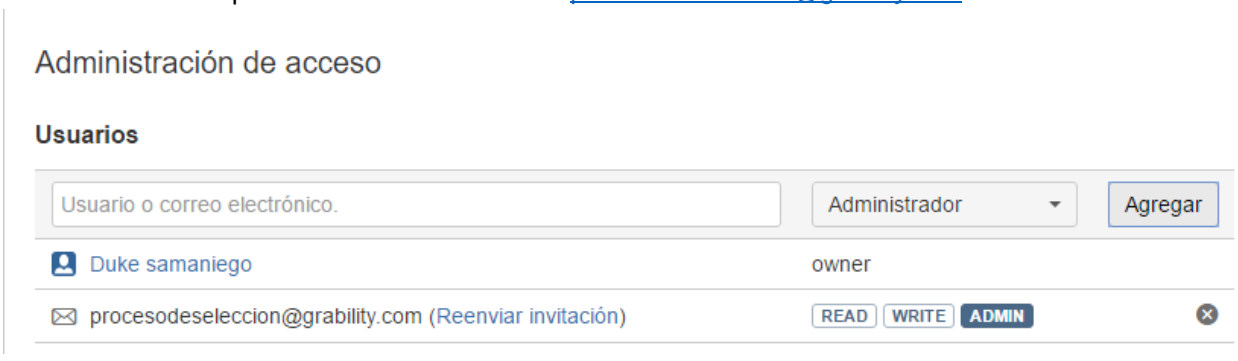
Salida



2) Usas Git o SVN y la historia del programa muestra su progreso en el desarrollo, usando commits con unidades de funcionalidad



Se envía invitación para tener acceso al correo: procesodeseleccion@grability.com

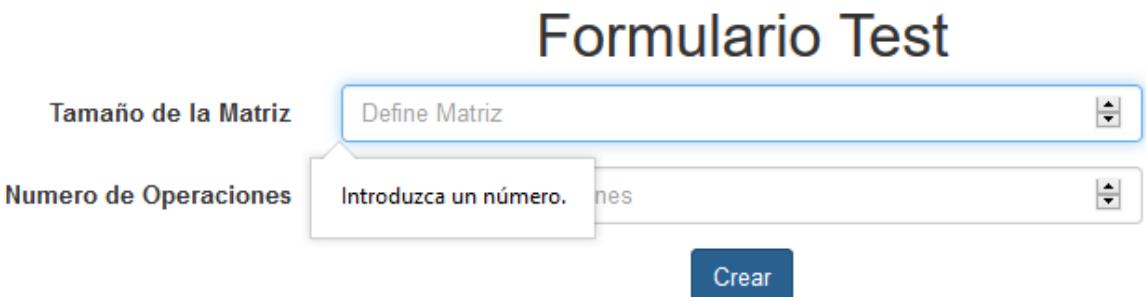


Por tema de confidencialidad de prueba.

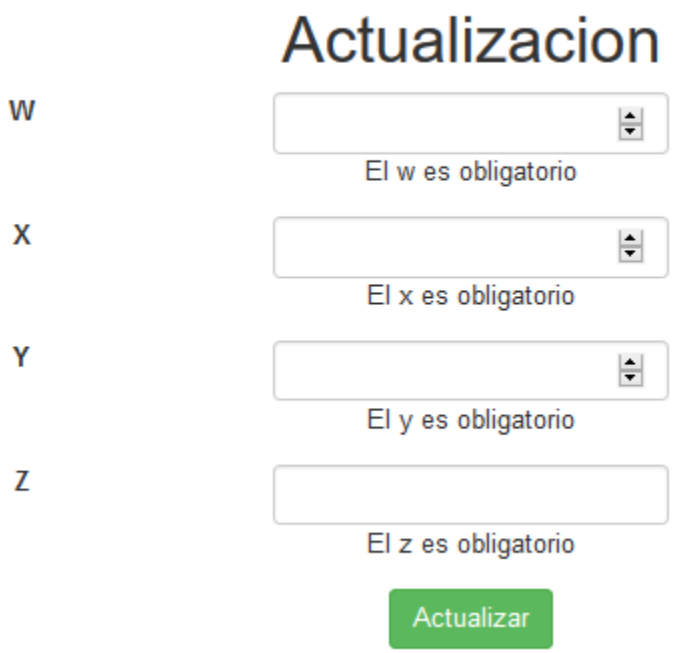
3) Usas pruebas unitarias y/o de validación (acceptance) que demuestren el buen funcionamiento del programa.

Para las pruebas se automatizaron con Selenium, Como no conozco si manejan el mismo plugin , se sube al repositorio un video de las pruebas realizadas con el archivo de selenium por si se desean volver a ejecutar en cualquier computador y se toman pantallazo ejemplo de las validaciones.

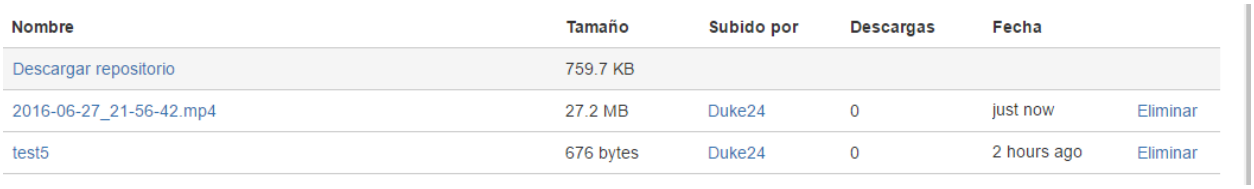
Validaciones por parte de la vista



Validaciones por parte del controlador



Descargar y ver video del ejemplo de la realización de pruebas y ejecución de una de ellas.



Nombre	Tamaño	Subido por	Descargas	Fecha	
Descargar repositorio	759.7 KB				
2016-06-27_21-56-42.mp4	27.2 MB	Duke24	0	just now	Eliminar
test5	676 bytes	Duke24	0	2 hours ago	Eliminar

Archivo llamado “2016-06-27_21-56-42.mp4” es video, y el archive test5 son las todas pruebas automatizadas de Selenium

CODE REFACTORING

1. **Las malas prácticas de programación que en su criterio son evidenciadas en el código**

```
public function post_confirm() {
```

Se recomienda si la función es post y va llamar confirm debería de nombrarse postConfirm() sin ningún tipo de _ y el mayúscula la primera letra del nombre de la fusión seguida del () y {

```
/* $servicio = Service::find($id);
  $push = Push::make();
  if ($servicio->user->type == '1') {//iPhone
  $pushAns = $push->ios($servicio->user->uuid, $pushMessage);
  } else {
  $pushAns = $push->android($servicio->user->uuid, $pushMessage);
  } */
```

Este tipo código que se dejó usar, considero que es una mala práctica dejarlo como comentario. Debería evitarse este tipo de cosas.

```
{//iPhone
```

Este tipo de comentarios no deber ir después de los corchetes si no una línea antes.

Eliminar todo esto que no son comentarios

```
//dd($servicio);
'status_id' => '2'
    //Up Carro
    //,'pwd' => md5(Input::get('pwd'))
    //Up Carro
    //,'pwd' => md5(Input::get('pwd'))
```

Es recomendable siempre usar comillas sencillas

```
"available" => '0'
```

Para mejor compresión del código para otros programadores sugiero usar un solo estándar para nombrar las variables

```
('serviceId'        'service_id'
```

Se recomienda usar variables en minúscula y en caso de ser muy largo usar _

```
}if($servicio->user->uuid== '')
```

Este tipo de validaciones donde usuario en su parte uuid es confuso, no es recomendable adiciona a esto esta validad si es igual a un espacio en blanco, No sé qué tan necesaria sea esta validación, seguramente este id es un auto incrementable y no puede ser null en la base de datos. Y va devolver un solo un mensaje de error.

```
if ($servicio->driver_id == NULL && $servicio->status_id == '1') {
    $servicio = Service::update($id, array(
                'driver_id' => Input::get('driver_id'),
                'status_id' => '2'
                    //Up Carro
                    //,'pwd' => md5(Input::get('pwd'))
    ));
    Driver::update(Input::get('driver_id'), array(
        "available" => '0'
    ));
    $driverTmp = Driver::find(Input::get('driver_id'));
    Service::update($id, array(
        'car_id' => $driverTmp->car_id
            //Up Carro
            //,'pwd' => md5(Input::get('pwd'))
    ));
    //Notificar a usuario!!
    $pushMessage = 'Tu servicio ha sido confirmado!';
    /* $servicio = Service::find($id);
      $push = Push::make();
      if ($servicio->user->type == '1') {//iPhone
      $pushAns = $push->ios($servicio->user->uuid, $pushMessage);
      } else {
      $pushAns = $push->android($servicio->user->uuid, $pushMessage);
      } */
    $servicio = Service::find($id);
    $push = Push::make();
    if ($servicio->user->uuid == '') {
        return Response::json(array('error' => '0'));
    }
    if ($servicio->user->type == '1') {//iPhone
        $result = $push->ios($servicio->user->uuid, $pushMessage, 1, 'honk.wav', 'Open', array('serviceId' => $servicio->id));
    } else {
        $result = $push->android2($servicio->user->uuid, $pushMessage, 1, 'default', 'Open', array('serviceId' => $servicio->id));
    }
    return Response::json(array('error' => '0'));
} else {
```

El if se abre y se cierra donde se seleccionó con el color rojo, es decir que si esa condición de cumple y no ingresa en alguna de los if más internos va a devolver un Json con un error 0, lo cual sería un error.

Considero que si desea hacer este tipo de validaciones seria mejor usar un try cash. Para que caso que ninguno de esas se cumpla se dispare la excepción.

Aunque no conozco si ya en este código el significado de errores, es decir que si nos devuelve un json con un error sea por ejemplo registros null, error numero 2 problemas de autenticación etc. Porque solo existe un comentario que es que se espera que esa función notifique al usuario, de restos solo fue código que no se utiliza como comentario y para otro programador si no entiende la lógica del negocio, es difícil re factorizar u optimizar el código.

2. **Cómo su refactorización supera las malas prácticas de programación**
   Dentro de proyecto de laravel cree un controller llamado FactorController para refactorizar el código , teniendo en cuenta las observaciones del primer punto.

```php
public function postConfirm(){
    try{
        //Recibe dos parámetros por POST: El id de un servicio, el id de un conductor
        $id = Input::get('service_id');
        $servicio = Service::find ($id);
        //verifica que id de servicio sea diferente de null
        if($servicio != NULL){
            if($servicio->status_id == '6'){
                return  Response::json(array('error'=>'2'));
            }
            try{
                if($servicio->driver_id == NULL && $servicio->status_id == '1'){
                    $servicio = Service::update($id,array(
                        'driver_id'  => Input::get('driver_id'),
                        'status_id'  => '2'
                    ));
                    Driver::update(Input::get('driver_id'),array(
                        'available' => '0'
                    ));
                    $driverTmp = Driver::find(Input::get('driver_id'));
                    Service::update($id,array(
                        'car_id' => $driverTmp->car_id

                    ));
                    $pushMessage = 'Tu servicio a sido Confirmado';
                    $servicio = Service::find($id);
                    $push =Push::make();
                    if($servicio->user->uuid==Null){
                        return Response::json(array('error'=>'0'));
                    }
                    if($servicio->user->type == '1'){
                        $result = $push->ios($servicio->user->uuid,$pushMessage,1,'honk.wav','Open',array('service_id' =>$servicio->id));
                    }else{
                        $result = $push->android2($servicio->user->uuid,$pushMessage,1,'default','Open',array('service_id' =>$servicio->id));

                    }
                }else{
                    return Respose::json(array('error'=>'1'));
                }
            }catch(Exception $e){
                return Respose::json(array('error'=>'0'));
            }
        }else{
            return Respose::json(array('error'=>'3'));
        }
    }catch (Exception $e){
        echo "error";
    }
}
```

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

   Principio de responsabilidad única: Consiste en que cada clases debería tener solo y sola una responsabilidad o razón de cambio.

   Propósito: Esto con el fin de que el código sea más fácil de leer, mantener, de testear si clases solo tiene una responsabilidad será más flexible el código.

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

   Un código limpio, es un código que todo programador pueda entenderlo, uno en donde no se repitan líneas iguales de código, un código que sea fácil de testear, un código fácil de efectuar modificaciones cambios, código que se pueda adaptar según las necesidades que surjan en el camino.