CODE REFACTORING

Refactorice y envíe el código y en un documento explique:

Codigo original

```
public function post_confirm() {
    $id = Input::get('service_id');
    $servicio = Service::find($id);
    //dd($servicio);
    if ($servicio != NULL) {
        if ($servicio->status_id == '6') {
            return Response::json(array('error' => '2'));
        }
        if ($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 {
            return Response::json(array('error' => '1'));
        }
    } else {
        return Response::json(array('error' => '3'));
    }
}
```

Codigo refactorizado

```
/**
 * [post_confirm description]
 * @return [type] [description]
 */
public function post_confirm(){
       $id = Input::get('service_id');
       $servicio = Service::find($id);

       //Servicio encontrado
       if ($servicio != NULL){
              if ($servicio->status_id == '6'){
                     // Descripción del error
                     return Response::json(array('error' => '2'));
              }
```

```
                if ($servicio->driver_id == NULL && $servicio->status_id == '1'){
                        $servicio = Service::update($id, array(
                                                'driver_id' => Input::get('driver_id'),
                                                'status_id' => '2'
                        ));

                        //Descripción para la actualización del Driver
                        Driver::update(Input::get('driver_id'), array(
                                "available" => '0'
                        ));
                        $driverTmp = Driver::find(Input::get('driver_id'));

                        //Descripcion para la actualización del servicio dado un conductor
                        Service::update($id, array(
                                'car_id' => $driverTmp->car_id
                        ));

                        //Notificar a usuario!!
                        $pushMessage = 'Tu servicio ha sido confirmado';
                        $servicio = Service::find($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, $psuhMessage, 1,
'honk.wav','Open',array('serviceId' => $servicio->id));
                        }else{//Android
                                $result = $push->android($servicio->user->uuid, $psuhMessage,
1, 'default','Open',array('serviceId' => $servicio->id));
                        }
                        // Descripción del error
                        return Response::json(array('error'=>'0'));
                }else{
                        // Descripción del error
                        return Response::json(array('error'=>'0'));
                }
        }else{
                // Descripción del error
                return Response::json(array('error'=>'3'));
        }
}
```

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

- Existe código comentado en varias partes. Cuando el código tenga que ser extendido o editado por otro programador costará más entenderlo, ya que si se tienen métodos largos comentados se puede perder la continuidad de lo que se hace en el programa.
- No aparece detallada la descripción de lo que la función recibe ni de la información que retorna.

- Almacenar variables tipo STRING en base de datos para determinar identificadores, puede conllevar a una pérdida de espacio y puede hacer la búsqueda de registros más lenta, por lo que es recomendado utilizar variables tipo INTEGER para una mejor búsqueda.
- El atributo del usuario nombrado “uuid” no especifica que contiene o que debería contener la variable, eso es una mala práctica al momento de leer el código o entenderlo. El nombre de una variable debería contener al menos 2 palabras descriptivas de la misma.
- No se describen los tipos de errores arrojados, por lo que el código no se entiende al momento de dar un error
- No se describen correctamente las acciones en cada una de la llamadas a las clases, por lo que se dificulta describir el comportamiento del código.
- Los nombres de las funciones deben detallar su funcionamiento sino se deben describir en un comentario, por ejemplo el nombre de la función $push->android2() posee un número no descriptivo de la misma

2. Cómo su refactorización supera las malas prácticas de programación

La refactorización logra detallar cada paso de la función, lo que la hace más entendible por otros programadores, esto conlleva a que el procedimiento o función, logre ser más escalable.

PREGUNTAS

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

Establece que cada clase contenida en la aplicación debe ser responsabilidad de solo una parte del software. El propósito de este principio es encapsular dicha responsabilidad para que se pueda tratar como un objeto y evitar volver a reescribir código y hacer el código más eficiente y entendible.

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

El código limpio debe poseer varias características para que sea denominado como tal.

- Dicho código no debe ser redundante, todas las funciones deben ser únicas y con un propósito específico en la aplicación.

- No debe poseer partes de códigos comentadas, ya que tiende a confundir la lectura del código con las características de una función o de una variable y no hace amena la lectura.
- El código puede ser entendido por cualquier otro programador con facilidad para la extensión del mismo.
- Debe tener pocas dependencias, ya que si se tiene muchas, puede dificultar la lectura del mismo y el entendimiento