Falta denominar más variables en el Input::get

Denominamos más variables hací es más corto el código y consume menos memoria

class post\_confirm{

public function post\_confirm(){

$id=Input::get('servise\_id');

$servicio=Servise::find($id);

if($servicio !=NULL){

if($servicio->status\_id=='6'){

return Response::json(array('error'=>'2'));

}

if($servicio->driver\_id==NULL && $servicio->status\_id=='1'){

Service::update($id,array(

'driver\_id'=>Input::get('driver\_id'),

'status\_id'=>'2'

));

Driver::update(Input::get('driver\_id'),array(

"available"=>'0'

));

$driverImp=driver::find(Input::get('driver\_id'));

Servise::update($id,array(

'car\_id'=>$driverImp->car\_id

));

$pushMessage='tu servicio ha sido congirmado';

$servicio=Service::find($id);

$push=push::make();

if($servicio->user && $servicio->uuid === ''){

return Response::json(array('error'=>'0'));

}

if($servicio->user && $servicio->type=='1'){

$resul=$push->ios($servicio->user->uui,$pushMessage,1,'honk.wav','open',array('serviceId'=>$servicio->id));

} else {

$resul=$push->android2($servicio->user->uui,$pushMessage,1,'defaul','open',array('servicedId'=>$servicio->id));

}

return Response::json(array('error'=>'0'));

} else {

return Response::json(array('error'=>'1'));

}

} else {

return Response::json(array('error'=>'3'));

}

}

}

Principios fundamentales. La comprensión de dichos principios nos permitirá mejorar la percepción del no siempre fácil campo de la POO, evitando así malas prácticas que la gran flexibilidad que ofrece esta metodología otorga, fundamentalmente a través de los lenguajes y herramientas que la soportan.

Se dice que un subsistema, módulo, clase o incluso una función no debe tener más de una razón para cambiar

Faltan variables pero es un buen código