PRUEBA TECNICA RAPPI

## CODING CHALLENGE:

1. El desarrollo se realiza con lenguaje de programación JAVA. Sobre la plataforma de JDK8 y esta armado el proyecto con MAVEN
2. Los fuentes se alojan en GITHUB <https://github.com/ivanntis/RappiTest.git>.
3. Explicación del código:
   1. Para la prueba se crean 3 clases:
      * ProccessInput : Esta clase se encarga del procesamiento del archivo CSV que tiene el input de datos para las pruebas. 🡪Controlador
      * RappiMatrix : Esta clase tiene toda la lógica del reto colocado por rappi, se realizan los cálculos y el manejo de la matriz 3D. 🡪 Negocios
      * Coordinate : esta clase cumple la función de Objeto de Transferencia de Datos, para la ubicación de elementos en la matriz de 3 dimensiones . 🡪Modelo DTO

## CODE REFACTORING

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

* El código depreciado no debería quedar comentariado, para ello existen sistemas de versionamiento, y aun cuando no contaran con un sistema como git , merculiar, svn …etc. El código comentariado siempre genera suspicacias en los demás desarrolladores.
* El response que retorna la función, debería ser una clase genérica de retorno de datos o una función, están repitiendo constantemente el mismo código para el retorno cuando solo cambia 1 valor.
* El llamado de la línea $servicio = Service::find($id), lo veo repetido sin ningún objetivo aparente. Ya que el servicio fue validado en el primer condicional

1. EL hecho de unificar en un objeto el retorno de la función, permite que en dado caso de una modificación en la estructura de la respuesta, se necesita realizar en un solo lugar.

Al quitar los comentarios se evita que en caso de duda vuelvan a activar códigos que nadie sabe para que existe, y puedan dañar el aplicativo.

## PREGUNTAS:

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

* Este principio nos indica que un objeto debe cumplir una única funcionalidad o una responsabilidad lógica en el programa. Este principio nos ayuda a tener menor acoplamiento y mayor cohesión entre los objeto facilitando el mantenimiento del aplicativo, adicionalmente

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

* Los nombres de los objetos, variables funciones o métodos, deben ser descriptivos con su funcionalidad, con ello reducir los excesivos comentarios dentro del código
* Entre menos acoplamiento tenga con otros objetos, mucho mas fácil de entender.
* No debe estar lleno de códigos viejos sin utilizarse activos o comentariados.
* El código debe ser fácil de leer.
* Evitar que los objetos creados tenga demasiadas líneas de código.