Trabajo de Investigacion #2

Arquitectura Clean

Es aquella que pretende conseguir unas estructuras modulares bien separadas de facil lectura, limpieza de codigo y testabilidad. los sistemas construidos de esta manera son los siguientes.

- **Independientes del framework** las librerias empleadas en la construcción del sistema no deben condicionarlos, han de ser una herramienta mas por utilizar.

- **Testeables** la logica del negocio de la aplicación ha de ser testeable indis-tintivamente de la interfaz grafica. modelo, base de datos o peticiones a una API empleada.

- **Independientes de la interfaz grafica** Debemos de usar patrones que nos permita cambiar facilmente la interfaz grafica. En otras palabras tenemos que evitar acoplar el funcionamiento de la vista con el modelo.

- **Independientes de los origenes de datos** Podemos sustituir nuestro origen de datos facilmente y sin importarnos si esta disponible en una base de datos local, ficheros, base relacional o no relacional.

Principios Solid

Es uno de los acronimos mas famosos en el mundo de la programación este se compone de 5 principios. de la programación orientada a objetos.

- **Principios de responsabilidad unica (SS):** Una clase debe encapsular una unica funcionalidad; en caso de encapsular mas de una funcionalidad, seria nucsario separar la clase en multiples clases.

- **Principio de ser abierto y cerrado (OS):** Debemos preparar nuestro codigo para que este abierto a extensiones y cerrado a modificaciones. La idea es que el codigo ya escrito. y que ha pasado las pruebas unitarias, este cerrado a modificaciones, es decir, que aquello que funciono no se toque.

- **Principio de segregación de interfaz** A medida que vayamos construyendo nuestro sistema, nos encontramos con un problema frecuente en nuestras clases, sobre todo en las extendidas, ya que pueden contener metodos de interfaces, que no implementamos al no ser necesaria su implementacion para la clase en cuestion.

- **Principio de inversión de dependencia** Las dependencias de clase son un problema cuando se introducen cambios en el sistema; instanciar una clase dentro de otra implica tener conocer las clases que se instancián y que de alguna manera realizan su función

- **Patrones de Diseño** El uso de patrones facilita la solucion de problemas comunes existentes en el desarrollo de software. Los patrones de diseño tratan de resolver problemas relacionados con la interacción entre interfaz de usuario, logica de negocio y los datos.

- **Modelo vista controlador** Es uno de los mas conocidos por la comunidad de desarrolladores de software. donde se plantea el uso de 3 capas para separar la interfaz de usuario de los datos y la logica de negocio.

Modelo Esta capa contiene el conjunto de clases que definen la estructura de datos con los que vamos a trabajar en el sistema.

Vista Esta capa contiene la interfaz de usuario de nuestra aplicación. maneja interaccción del usuario con interfaz de usuario para evitar peticiones al controlador.

Controlador Esta es la capa es la intermediaria entre la vista y el modelo. Es capaz de responder a eventos, capturandolos por la interacción de usuarios en la interfaz.

MVP nos permite separar aun mas la vista de la logica de negocio y de los datos. En este patron, toda logica de la presentación de la interfaz reside en el presentador, de forma que este da el formato necesario a los datos y los entrega a la vista para que simplemente pueda mostrarlos sin realizar alguna logica adicional.

Point Observer se basa en 2 objetos con una responsabilidad bien definida; por una parte, los objetos observables y la otra los observadores.