**Programacion orientada a objetos**

Es un paradigma que nos da unas bases o una guía de como trabajar sobre el

**Java es secuencial**

**Que es un objeto**

Es una unidad con un estado y comportamiento que tiene una serie de datos y tareas almacenados, es una instancia de la clase

**Que es una clase**

Es una plantilla que nos servirá para crear objetos la cual recopila información se compone de métodos y atributos

**Que es una función**

Un conjunto de líneas de código que se encargan de realizar una tarea especifica, puede recibir y retornar valores

**Que es una API**

Interfaz de programación de aplicaciones, es un servicio web en el lado del backend consumida por el protocolo http, devuelve los datos en xml o Json

**Que es Rest**

Representational state transfer como se da la aplicación entre dos equipos, trabaja entre cliente servidor, es una lógica de instrucciones para construir la API

**Que es Restful API**

Es una API implementada que fue construida usando la lógica de REST

**Que es xml**

Extensible markup languaje/Lenguaje de marcado extensible, Sirve para la comunicación entre aplicaciones y trae información

**Que es http**

Hipertext transfer protocol, es el protocolo de transmisión de información es decir, el código que se establece para que el computador solicitante y el que contiene la información solicitada puedan “hablar”, si, básicamente diseñado para la comunicación entre los navegadores y servidores web,

**GET**

solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.

**HEAD**

El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.

**POST**

El método POST se utiliza para enviar recursos, causando a menudo un cambio en el estado o efectos en el servidor.

**PUT**

El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.

**DELETE**

El método DELETE borra un recurso en específico.

**CONNECT**

El método CONNECT establece un túnel hacia el servidor identificado por el recurso.

**OPTIONS**

El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.

**TRACE**

El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.

**PATCH**

El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.

**Web services**

Es un intercambio automatico de datos entre dos aplicaciones se basa en normas basadas en internet

**Encapsulamiento**

se denomina encapsulamiento al ocultamiento del estado, es decir, de los datos miembro de un objeto de manera que solo se pueda cambiar mediante las operaciones definidas para ese objeto.

**Herencia**

Se basa en reutilización y la extensibilidad de código, pues para no repetir código y es la capacidad que una clase padre le herede métodos y atributos ósea el comportamiento a una hija

**Polimorfismo**

polimorfismo es la capacidad que tienen los objetos de una clase en ofrecer una respuesta distinta e independiente en función de los parámetros ósea que tiene diferentes implementaciones utilizados durante su invocación. Y puesss dicho de otro modo el objeto como entidad puede contener valores de diferentes tipos durante la ejecución del programa

**Guards en angular**

Los Guards en Angular, son de alguna manera: middlewares que se ejecutan antes de cargar una ruta y determinan si se puede cargar dicha ruta o no. Existen 4 tipos diferentes de Guards (o combinaciones de estos) que son los siguientes:

* (CanActivate) Antes de cargar los componentes de la ruta.
* (CanLoad) Antes de cargar los recursos (assets) de la ruta.
* (CanDeactivate) Antes de intentar salir de la ruta actual (usualmente utilizado para evitar salir de una ruta, si no se han guardado los datos).
* (CanActivateChild) Antes de cargar las rutas hijas de la ruta actual.

**Componentes en angular**

son pequeñas partes lógicas de la aplicación, que van a estar representando a un trozo de la pantalla. Un componente es un bloque, que contiene un template, contiene estilos , contiene lógica.

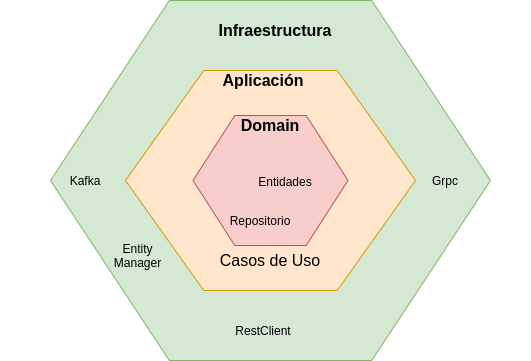
**PRINCIPIOS SOLID**

Son una serie de normas o recomendaciones para programar para que sea mantenible y facilite la adición de codigo

1. Single responsibility (responsabilidad única): que cada clase tenga su propia responsabilidad sobre lo que va a trabajar y no se haga en una clase más de una cosa respecto a un tema
2. Open closed (abierto cerrado): Que está protegida, se debe usar herencia y polimorfismo para implementarlo bien, ósea toca no dejar clases abiertas para agregar porque puede haber bugs
3. Liskov substitution(sustitución de liskov): cada clase hija puede y debe comportarse como el padre si le falta algo por implementar está mal abstraído
4. Interface segregation (segregación de la interfaz): Es mejor tener muchas clases pequeñas y especializadas que una clase enorme para que esta no ocupe espacio de memoria innecesario y este mejor caracterizad
5. Dependency inversión (inversión de la dependencia): Los modulos de alto nivel no deben depender de los de bajo nivel ambos dependen de interfaces y se basa en la abstracciónde que el código no depende de algo en concreto

**Arquitectura hexagonal**

1. Aplicación: es la lógica de negocio y es la que iría en el centro

}

**Angularjs vs angular**

Son distintos, ambos son para front, el angularjs es más viejo y algo obselo y se programa con javascript el angular es el moderno, el que ahora se usa y se trabaja con typescript