TEMA 1

ARQUITECTURAS DE SOFTWARE

- Concepto. Evolución.

La arquitectura de software, indica la estructura, funcionamiento e interacción entre los elementos que forman el software de una aplicación.

Podemos verlo desde dos puntos de vista:

1. Vista lógica: cómo se estructura el software, por ejemplo, su tipo de arquitectura (arquitectura multicapa, arquitectura orientada a servicios SOA).
2. Vista física: cómo se ejecuta el software, por ejemplo, puede ser que se ejecute en una o varias máquinas (arquitectura monolítica y distribuida).

- Evolución de las arquitecturas de software:

Con el paso de los años, las arquitecturas de software han ido evolucionando en el tiempo para superar sus limitaciones. ¿Qué opción elegir?

- Arquitecturas monolíticas: Todo el software se guarda y se ejecuta como un solo proceso en una máquina.

Ventajas: Fácil de testear y debugear, desplegar y desarrollar.
Desventajas: Código enorme (un error puede afectar a todo el funcionamiento), escalabilidad costosa y difícil de incluir nuevos servicios.

- Arquitecturas distribuidas en cliente/servidor: la funcionalidad de la aplicación se divide en dos maquinas, un cliente y un servidor. Los clientes se comunican a través de una red mediante un mecanismo de petición/respuesta. Para esa comunicación se usara algún tipo de protocolo.

Ventajas: cada parte puede escalar de forma independiente usando incluso tecnologías distintas.
Desventajas: es un sistema mas complicado de hacer

¿Cuál se debe utilizar?

Si la aplicación es pequeña, monolítica. Si es grande, microservicios.

- Arquitectura multicapa:

Es un tipo de abstracción, se divide la aplicación en varias capas de modo que cada una desempeñe una función en concreto. SOC (separation of concerns).

- Las capas se colocan una encima de otra.

En modo de capa cerrada, una capa se apoya en la capa inferior y proporciona servicios a la capa superior.

- Cada capa de ser independiente ya que así la aplicación no necesita ser ejecutada en el mismo entorno.
- Las ventajas y desventajas son las mismas que el cliente/servidor.

  - Arquitectura en 3 niveles: añade un nivel intermedio al modelo cliente/servidor. Ahora nos encontramos con una arquitectura cliente/servidor/datos.

Por lo general, las aplicaciones multicapa tienen una capa cliente, varias capas intermedias y una capa de datos.

- Arquitecturas orientadas a servicios (SOA, service oriented architecture).

Es un tipo de arquitectura de software que se apoya en la orientación a servicios. Las aplicaciones se desarrollan como la integración de un conjunto de servicios, independientes de su plataforma o lenguaje de programación.

  - Front-end: es la parte visible de las aplicaciones y sitios web.
  - Back-end: es la parte del servicio con las que los usuarios no tienen contacto.
  - Framework: es una estructura previa que se puede aprovechar para desarrollar un proyecto. El framework es una especie de plantilla que simplifica la elaboración de una tarea, ya que solo es necesario complementarlo de acuerdo a lo que se quiere realizar.
  - API: es una especificación formal que establece cómo un modulo de software se comunica o interactúa con otro para cumplir una o varias funciones.

- Plataforma Java EE (JEE, Java Enterprise)

Es una plataforma de programación, parte de la plataforma Java. Sirve para desarrollar y ejecutar aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

DISEÑO DE APLICACIONES MULTICAPA

- Patrones de diseño de software.

Un patron de diseño es una solución probada que resuelve un tipo especifico de problema en desarrollo de software referente al diseño.

  - **Patron MVC (Model View Controller)**

Este patron permite separar una aplicación en tres capas:

1. Modelo: Representa el acceso a datos: guardar, obtener, actualizar, etc. Además de toda la lógica de negocio.
2. Vista: La presentación de datos del modelo y lo que ve el usuario.
3. Controlador: Funciona como puente entre la vista y el modelo. Recibe instrucciones de la capa de la vista y las direcciona al modelo.

- **Patron DTO**

  Se utiliza para transferir varios atributos entre el cliente y el servidor o viceversa , basicamente consta de dos clases:

  1. Value Object: contiene sus atributos, constructor, getters y setters. Esta clase no tiene comportamiento.
  2. Business Object: se encarga de obtener datos de la base de datos y llenar la clase Value Object y enviarla al cliente o viceversa.

- **Patron DAO**

  Este patron resuelve el problema de acceso a datos. Consiste en tener una aplicación que no esta ligada al acceso de datos. DAO podría sacar los datos independientemente de donde estén alojados.

- Estructura de capas

Una capa define su API de acceso a través de interfaces. La capa superior solo se comunica con la inferior utilizando métodos de la interfaz. De esta forma conseguimos al modificar alguna capa, esta no afecte a las demás.