MVC

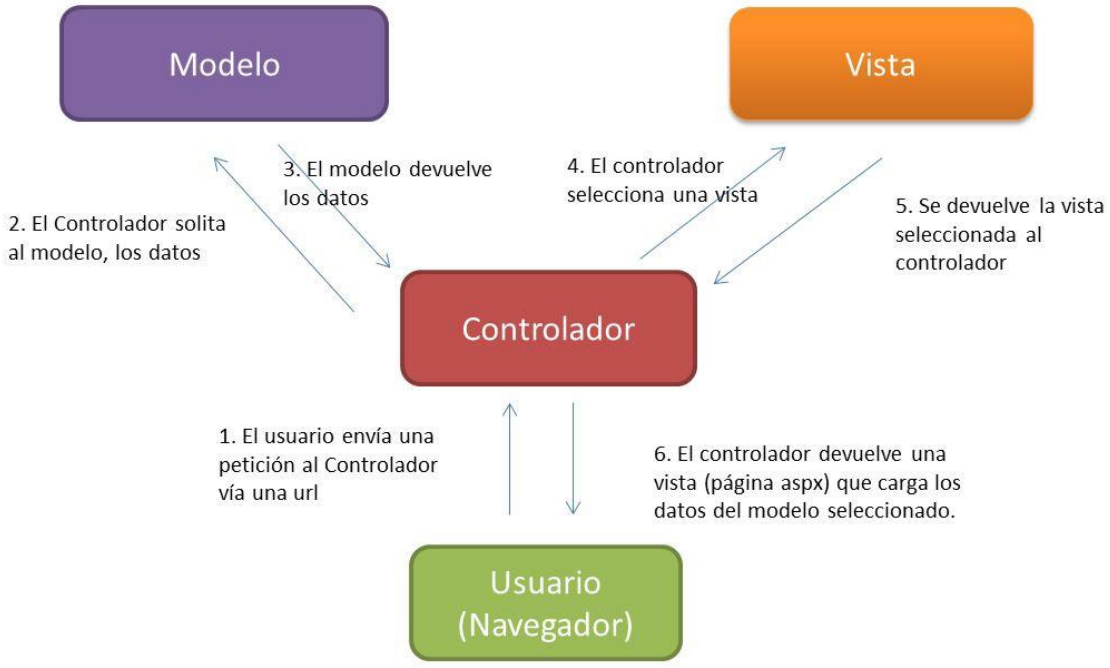
Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

El Modelo representa los datos y la lógica de negocio de la aplicación. Es responsable de la gestión y almacenamiento de la información, y proporciona una interfaz para acceder a ella.

La Vista es la interfaz de usuario de la aplicación. Es responsable de mostrar la información al usuario y recibir las entradas del usuario.

El Controlador actúa como intermediario entre la Vista y el Modelo. Recibe las entradas del usuario desde la Vista, las procesa y actualiza el Modelo en consecuencia. También recupera la información del Modelo y la envía a la Vista para ser mostrada al usuario.

La ventaja de utilizar el patrón MVC es que permite separar las responsabilidades de la aplicación en componentes independientes, lo que facilita el desarrollo, el mantenimiento y la extensión de la aplicación. También ayuda a mejorar la organización del código y aumenta la reutilización de componentes.

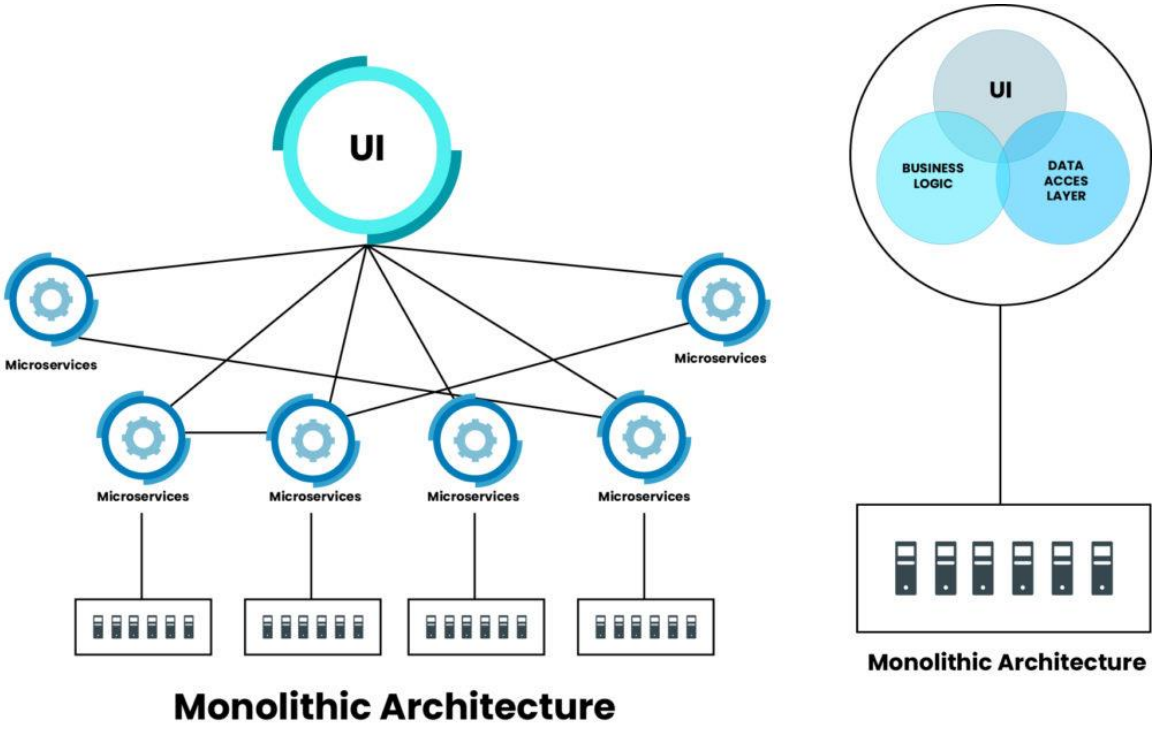


Microservicios

Los microservicios son un enfoque arquitectónico y organizativo para el desarrollo de software donde el software está compuesto por pequeños servicios independientes que se comunican a través de API bien definidas. Los propietarios de estos servicios son equipos pequeños independientes.

Las arquitecturas de microservicios hacen que las aplicaciones sean más fáciles de escalar y más rápidas de desarrollar. Esto permite la innovación y acelera el tiempo de comercialización de las nuevas características.

La idea detrás de los microservicios es que una aplicación compleja se divide en un conjunto de servicios más pequeños y manejables, que se pueden desarrollar, probar y escalar de manera independiente. Esto permite un mayor grado de flexibilidad y escalabilidad, ya que se pueden añadir o eliminar servicios sin afectar al funcionamiento de la aplicación completa.



Excepciones

Las excepciones son el medio que ofrecen algunos lenguajes de programación para tratar situaciones anómalas que pueden suceder cuando ejecutamos un programa. Algunos casos de situaciones anómalas que se pueden citar, son, por ejemplo, invocar a un método sobre un objeto "null", intentar dividir un número por "0", intentar abrir un fichero que no existe para leerlo, quedarnos sin memoria en la JVM al ejecutar (por ejemplo, debido a un bucle infinito), intentar crear un fichero en una carpeta en la que no hay permisos de escritura, tratar de recuperar información de Internet cuando no se dispone de conexión, o simplemente sufrir un apagón eléctrico al ejecutar nuestro código.

Existen varios tipos de excepciones en la programación, algunos de los más comunes son:

- Excepciones de sistema: Son aquellas que son generadas por el sistema operativo o por componentes del sistema en el que se ejecuta el programa. Ejemplos incluyen OutOfMemoryError, cuando no hay suficiente memoria disponible, o FileNotFoundException, cuando no se encuentra un archivo específico.
- Excepciones de lenguaje: Son aquellas que son generadas por el lenguaje de programación utilizado. Ejemplos incluyen NullPointerException, cuando se intenta acceder a un objeto nulo, o IndexOutOfBoundsException, cuando se intenta acceder a un índice fuera de los límites de un arreglo.
- Excepciones de usuario: Son aquellas que son generadas por el programador para indicar una condición de error específica. Ejemplos incluyen InvalidArgumentException, cuando se proporciona un argumento no válido a un método, o LoginFailedException, cuando falla el proceso de inicio de sesión.

Multicatch and try-with-resource

La cláusula "multi-catch" en Java permite manejar varias excepciones diferentes en un solo bloque "catch" en lugar de tener varios bloques "catch" separados. Esto se logra especificando varios tipos de excepción separados por el operador '|'. Por ejemplo:

```
1. try {
2.   // código que puede generar excepciones
3. } catch (IOException | IllegalArgumentException e) {
4.   // manejo de excepción
5. }
6.
```

Esto permite escribir código más limpio y conciso, ya que se pueden manejar varias excepciones relacionadas de manera similar en un solo bloque "catch" en lugar de tener varios bloques "catch" separados.

Una de las novedades que incorporó Java 7 es la sentencia try-with-resources con el objetivo de cerrar los recursos de forma automática en la sentencia try-catch-finally y hacer más simple el código. Aquellas variables cuyas clases implementan la interfaz AutoCloseable pueden declararse en el bloque de inicialización de la sentencia try-with-resources y sus métodos close() serán llamados después del bloque finally como si su código estuviese de forma explícita.

```
1. try (FileInputStream inputStream = new FileInputStream("file.
   txt")) {
2.   // código que utiliza el recurso
3. } catch (IOException e) {
4.   // manejo de excepción
5. }
```

Collections

En Java, existen varios tipos de colecciones diferentes que se pueden utilizar para almacenar y manipular datos. Algunos de los tipos más comunes de colecciones en Java son:

ArrayList: Una implementación de List que utiliza un array para almacenar los datos. Es una de las colecciones más utilizadas en Java debido a su flexibilidad y rendimiento.

LinkedList: Una implementación de List que utiliza un enlace doblemente enlazado para almacenar los datos. Es menos eficiente en términos de rendimiento que ArrayList, pero tiene un mejor rendimiento para operaciones de inserción y eliminación en el medio de la lista.

HashMap: Una implementación de Map que utiliza una tabla hash para almacenar pares clave-valor. Es muy rápido para operaciones de búsqueda y eliminación, pero tiene un peor rendimiento para operaciones de ordenamiento.

TreeMap: Una implementación de Map que utiliza un árbol rojo-negro para almacenar pares clave-valor. Es más lento que HashMap para operaciones de búsqueda y eliminación, pero mantiene los elementos ordenados según su clave.

HashSet: Una implementación de Set que utiliza una tabla hash para almacenar los elementos. Es rápido para operaciones de búsqueda y eliminación, pero no garantiza el orden de los elementos.

TreeSet: Una implementación de Set que utiliza un árbol rojo-negro para almacenar los elementos. Es más lento que HashSet para operaciones de búsqueda y eliminación, pero mantiene los elementos ordenados.