El futuro digital es de todos

MinTIC

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Mision TIC 2022

SEMANA 7

INICIAMOS 8:05PM

UNIVERSIDAD DE ANTIOQUIA
1803
Facultad de Ingeniería

Luisa Fernanda Restrepo.

Agenda

- Patrones de arquitectura de software
- Patrón MVC

Patrones de Arquitectura de Software

FACULTAD DE INGENIERIA UdeA
Una facultad para la sociedad del aprendizaje

Luisa Fernanda Restrepo.

Patrones de arquitectura de software

- Es una solución a un problema recurrente que se comprende bien, en un contexto particular.
- Cada patrón consta de un contexto, un problema y una solución. El problema puede ser superar algún desafío, aprovechar alguna oportunidad o satisfacer uno o más atributos de calidad.
- Los patrones codifican el conocimiento y la experiencia en una solución que podemos reutilizar.
- El uso de patrones simplifica el diseño y nos permite obtener los beneficios de usar una solución que está probada para resolver un problema de diseño en particular.
- Un patrón de arquitectura de software proporciona una estructura y un comportamiento de alto nivel para los sistemas de software.
- Es una agrupación de decisiones de diseño que se han repetido y utilizado con éxito para un contexto determinado.
- Cada patrón tiene sus propias características, fortalezas y debilidades.
- Los patrones de arquitectura de software proporcionan la estructura y los componentes principales del sistema de software que se está construyendo.
- Introducen restricciones de diseño que reducen la complejidad y ayudan a prevenir decisiones incorrectas.

La clave es conocer los patrones disponibles y comprender los escenarios en los que deben aplicarse.

Un patrón de arquitectura de software solo debe usarse si es la mejor solución para un problema de diseño y un contexto determinados.

Patrón MVC

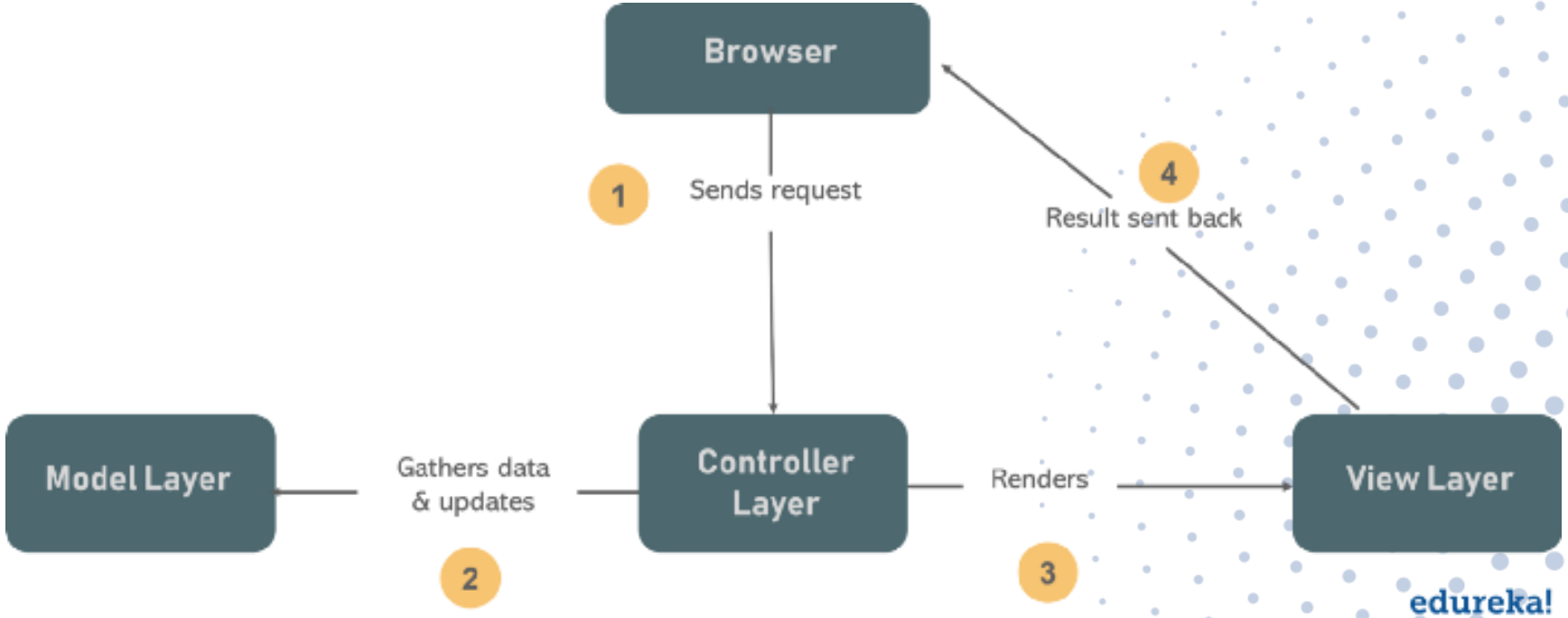
Luisa Fernanda Restrepo.

Patrón MVC

- El patrón Modelo-Vista-Controlador (MVC) es un patrón de arquitectura de software que se usa ampliamente para la interfaz de usuario de una aplicación. Es especialmente adecuado para aplicaciones web, aunque también se puede utilizar para otros tipos de aplicaciones, como aplicaciones de escritorio.

Patrón MVC

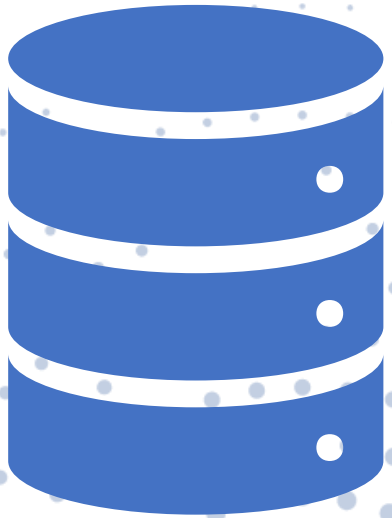
- El patrón proporciona una estructura para construir interfaces de usuario y **proporciona una separación de las diferentes responsabilidades involucradas.** Varios marcos de desarrollo web y de aplicaciones populares hacen uso de este patrón. Algunos ejemplos incluyen Ruby on Rails, ASP.NET MVC y Spring MVC.



MODELO

- El modelo gestiona los datos de la aplicación y el estado. Entre sus responsabilidades se encuentra el procesamiento de datos hacia y desde un almacén de datos, como una base de datos. Un modelo es independiente de los controladores y las vistas, lo que permite reutilizarlos con diferentes interfaces de usuario. Esto también les permite probarse de forma independiente.

- Los modelos reciben directivas de los controladores para recuperar y actualizar datos. Los modelos también proporcionan actualizaciones del estado de la aplicación. En algunas variaciones de MVC, el modelo es pasivo y debe recibir una solicitud para enviar una actualización del estado de la aplicación. En otras variaciones, una vista puede estar activa y enviar notificaciones de cambios de estado del modelo a una vista.

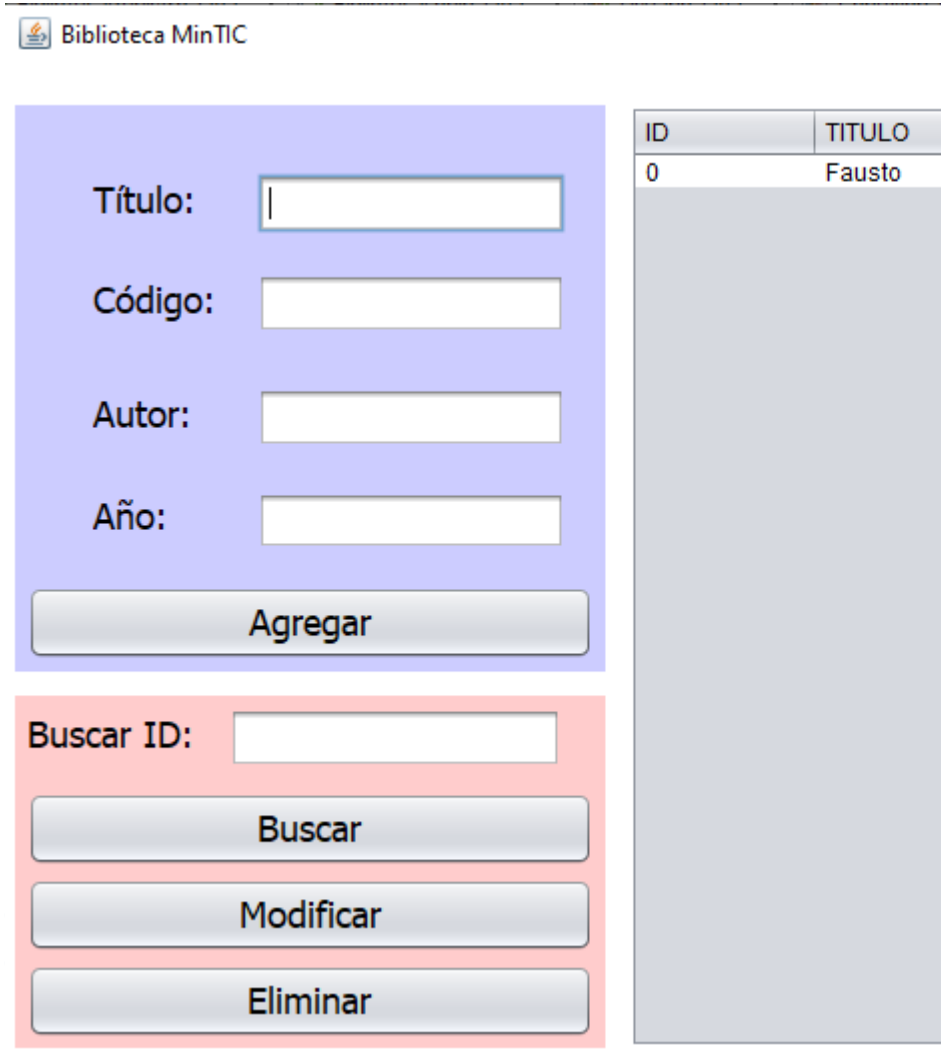


Base de Datos

VISTA

- La vista es responsable de la presentación de la aplicación. Es la parte de la aplicación que es visible para el usuario. La vista muestra datos al usuario en una interfaz apropiada basada en la información recibida del controlador. Si el modelo proporciona actualizaciones de estado de la aplicación directamente a las vistas, las vistas también pueden actualizarse en función de estas notificaciones.

- A medida que los usuarios manipulan una vista, como proporcionar información o realizar alguna acción del usuario, la vista enviará esta información a un controlador.



CONTROLADOR

- A medida que los usuarios navegan por una aplicación web, las solicitudes se enrutan al controlador apropiado según la configuración de enrutamiento. Un controlador actúa como intermediario entre el modelo y la vista.

- Un controlador ejecuta la lógica de la aplicación para seleccionar la vista adecuada y le envía la información que necesita para representar la interfaz de usuario. Las vistas notifican a los controladores sobre las acciones del usuario para que el controlador pueda responder a ellas. Los controladores actualizarán el modelo en función de las acciones del usuario.

Ventajas

- Separación de preocupaciones.
- Más fácil de probar.
- Reutilizar los componentes de la interfaz de usuario.
- Desarrolladores especializados en el desarrollo de frontend o backend.
- Algunas tareas pueden realizarse en paralelo.

Desventajas

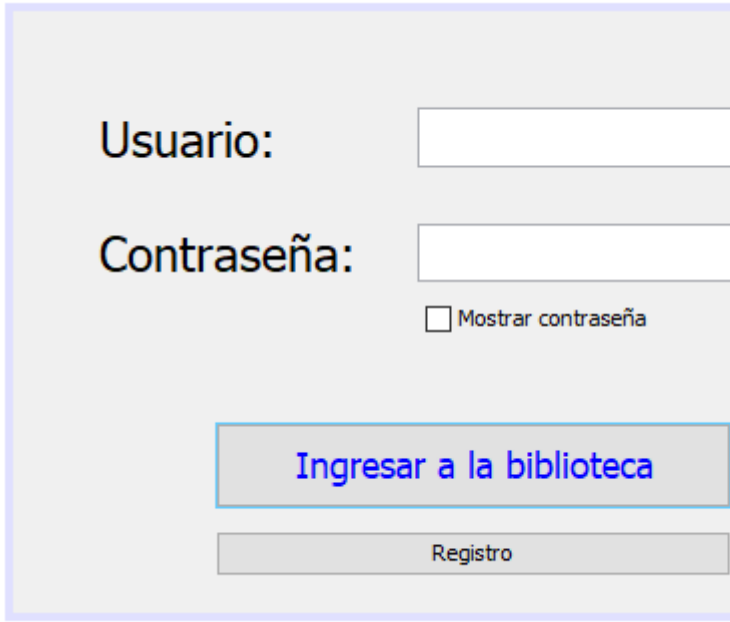
- Es difícil lograr una separación complete
- Requiere que los desarrolladores sean expertos multiples tecnologías.
- Cambios frecuentes en un modelo pueden resultar en actualizaciones excesivas de las vistas.

MODELO

CONTROLADOR

VISTA

```
public class bibliotecaE {

    private int id;
    private String nombre;

    public bibliotecaE(int id, String nombre) {
        this.id = id;
        this.nombre = nombre;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String toString(){
```



Referencias

Joseph Ingeno. 2018. Software Architect’s HandBook.