­

**10/05/2024**

**Xideral**

Luis Alberto García Reyes

**Modelo MVC**

# Introducción

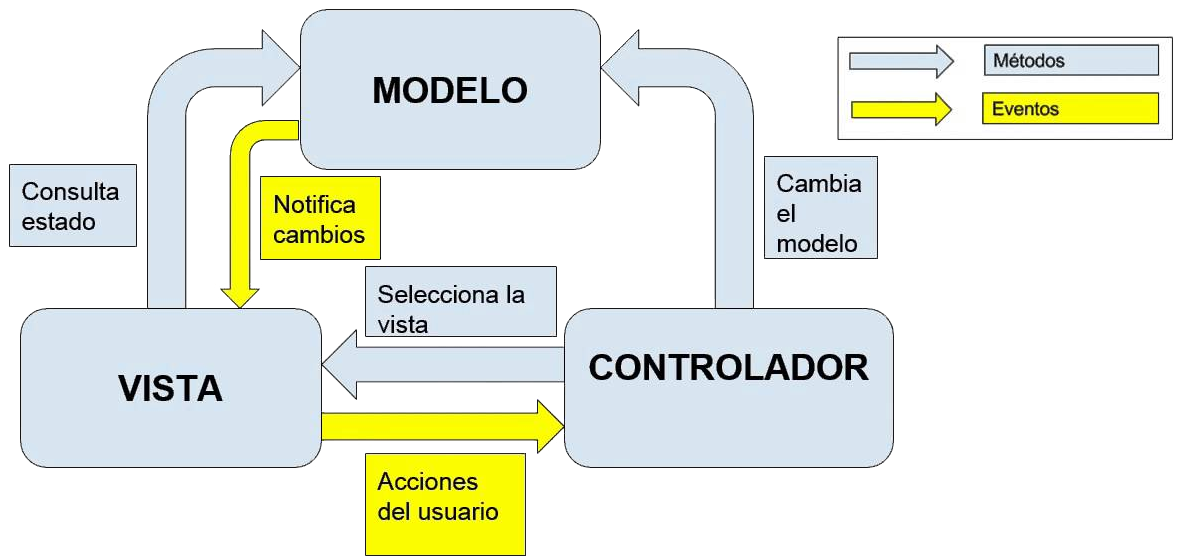
MVC (Modelo-Vista-Controlador) tiene sus raíces en la década de 1970, cuando fue desarrollado por primera vez en el laboratorio de informática de Xerox PARC como parte del proyecto Smalltalk. Es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control.

Este proyecto buscaba crear un entorno de desarrollo de software orientado a objetos y fácil de usar. MVC se diseñó para separar las preocupaciones de presentación y lógica de negocio en las aplicaciones de software, lo que permitía una mayor modularidad y facilidad de mantenimiento.

Las tres partes del patrón de diseño de software MVC se pueden describir de la siguiente manera:

* Modelo: Maneja datos y lógica de negocios.
* Vista: Se encarga del diseño y presentación.
* Controlador: Enruta comandos a los modelos y vistas

Con el tiempo, surgieron variantes del patrón MVC, como MVP (Modelo-Vista-Presentador) y MVVM (Modelo-Vista-Modelo de Vista), cada una con sus propias características y ventajas. Sin embargo, el patrón MVC original sigue siendo uno de los fundamentos del desarrollo de software moderno y continúa siendo utilizado en una amplia variedad de aplicaciones y tecnologías.



Modelo Simplificado

# MVC

Diagrama

Descripción generada automáticamente

10

9

8

7

6

5

4

3

2

1

**Componentes**

## 1.- Bases de datos

La función principal de las bases de datos en el MVC es **almacenar y gestionar los datos** utilizados por la aplicación. Esto incluye la creación, lectura, actualización y eliminación de registros, así como la gestión de la integridad de los datos y las relaciones entre ellos.

En la industria, se utilizan una variedad de productos de bases de datos para implementar el componente del Modelo en aplicaciones que siguen el patrón MVC. Algunos de los productos más populares incluyen:

* MySQL
* PostgreSQL
* Oracle Database
* SQL Server
* Mongo DB
* Cassandra
* Redis

Además de estas, existen otros productos de bases de datos relacionales y NoSQL que se utilizan en una variedad de aplicaciones según los requisitos específicos de cada proyecto.

En cuanto a los tipos de bases de datos, se pueden clasificar en varias categorías, incluyendo:

* **Bases de datos relacionales**: Utilizan tablas para almacenar datos y establecer relaciones entre ellos utilizando claves primarias y externas. Ejemplos incluyen MySQL, PostgreSQL, Oracle Database, SQL Server, etc.
* **Bases de datos NoSQL**: Ofrecen modelos de datos alternativos a las bases de datos relacionales, como documentos, clave-valor, columnares y gráficos. Ejemplos incluyen MongoDB, Cassandra, Redis, Neo4j, etc.
* **Bases de datos en memoria**: Almacenan datos en la memoria principal en lugar de en discos, lo que proporciona un acceso extremadamente rápido a los datos. Ejemplos incluyen Redis, Memcached, etc.

## 2.- Capa de Persistencia de datos

La capa de persistencia de datos dentro del Modelo en el patrón MVC es responsable de interactuar con la base de datos para realizar **operaciones CRUD** (Crear, Leer, Actualizar, Eliminar) en los datos de la aplicación. Esta capa maneja la comunicación entre la aplicación y la base de datos, abstrayendo los detalles de la implementación de la base de datos y proporcionando una interfaz para que el resto de la aplicación acceda y manipule los datos de manera eficiente y segura.

Algunas de las funciones clave de la capa de persistencia de datos incluyen:

* Establecer y manejar la conexión con la base de datos.
* Realizar consultas para recuperar datos según las necesidades de la aplicación.
* Insertar, actualizar y eliminar registros en la base de datos.
* Gestionar transacciones para garantizar la integridad de los datos.

En la industria, existen varios ejemplos de tecnologías y frameworks utilizados para implementar la capa de persistencia de datos en aplicaciones que siguen el patrón MVC. Algunos de ellos son:

1. **Hibernate**: Un framework de mapeo objeto-relacional (ORM) para Java que simplifica el acceso y la manipulación de datos en bases de datos relacionales.
2. **Entity Framework**: Un ORM para el ecosistema .NET de Microsoft que permite a los desarrolladores trabajar con datos utilizando objetos .NET en lugar de consultas SQL directas.
3. **Django ORM:** Parte del framework Django para Python, que proporciona una capa de abstracción sobre las bases de datos relacionales para facilitar el acceso y la manipulación de datos.
4. **Active Record (Rails)**: Parte del framework Ruby on Rails, que sigue el patrón Active Record y proporciona una forma simple y elegante de interactuar con la base de datos utilizando modelos de datos.

En el ecosistema de Spring Framework para Java, la capa de persistencia de datos se maneja típicamente a través de **Spring Data JPA**, que es una parte de Spring Data. Spring Data JPA simplifica el desarrollo de la capa de persistencia de datos al proporcionar un conjunto de abstracciones y herramientas para trabajar con bases de datos relacionales utilizando el enfoque de mapeo objeto-relacional (ORM). Además, Spring Framework ofrece soporte para otras tecnologías de persistencia de datos, como JDBC Template, que permite la interacción directa con la base de datos utilizando SQL nativo. En resumen, Spring facilita la implementación de la capa de persistencia de datos al proporcionar un conjunto completo de herramientas y abstracciones para trabajar con diferentes tecnologías de persistencia de datos en el contexto del patrón MVC.

## 3.- Capa de Servicio

La capa de servicio en el patrón MVC (Modelo-Vista-Controlador) actúa como una interfaz entre la capa de controlador y la capa de modelo. Su función principal es proporcionar una **lógica de negocio** y coordinar las operaciones complejas que involucran múltiples modelos o acciones en la aplicación. Algunas de las funciones que realiza la capa de servicio incluyen:

* **Orquestación de operaciones**: Coordina las operaciones que involucran múltiples modelos o componentes de la aplicación.
* **Validación de datos**: Verifica la validez y coherencia de los datos antes de realizar operaciones en la capa de modelo.
* **Aplicación de reglas de negocio**: Implementa la lógica de negocio de la aplicación, como cálculos, validaciones específicas del dominio, etc.
* **Transformación de datos**: Realiza cualquier transformación o conversión de datos necesaria antes de interactuar con la capa de modelo o la capa de vista.
* **Manejo de transacciones**: Coordina las transacciones que involucran operaciones en la capa de modelo para garantizar la integridad de los datos.

### Conexión con las capas adyacentes

La capa de servicio se conecta con otras capas del MVC de la siguiente manera:

**Conexión con la capa de controlador**: La capa de servicio es invocada por los controladores para realizar operaciones relacionadas con la lógica de negocio y coordinar las interacciones con la capa de modelo. Los controladores delegan las solicitudes recibidas desde la vista a los servicios correspondientes para realizar el procesamiento necesario.

**Conexión con la capa de modelo**: La capa de servicio interactúa directamente con la capa de modelo para realizar operaciones de lectura, escritura y manipulación de datos. Los servicios pueden acceder a los objetos de modelo directamente o a través de interfaces definidas en la capa de modelo.

**Conexión con la capa de vista**: Aunque la capa de servicio no interactúa directamente con la capa de vista, puede proporcionar datos o resultados procesados a los controladores, que luego los pasan a las vistas correspondientes para su presentación al usuario final.

## 4.- MVC Controlador

El controlador en el patrón MVC (Modelo-Vista-Controlador) desempeña un papel fundamental como el punto de entrada y coordinación en una aplicación. Su función principal es recibir las solicitudes del usuario desde la interfaz de usuario (la vista), interpretarlas y **coordinar las acciones necesarias** con el modelo y la vista correspondiente

* **Interpretación de solicitudes**: El controlador recibe las interacciones del usuario, como clics de botones, envíos de formularios, etc., desde la vista y las traduce en acciones o eventos específicos que la aplicación debe realizar.
* **Coordinación de acciones**: Una vez que el controlador interpreta la solicitud del usuario, coordina las acciones necesarias con el modelo y la vista correspondiente para realizar las operaciones requeridas, como leer o escribir datos, realizar cálculos, etc.
* **Actualización de la vista**: Después de realizar las operaciones necesarias en el modelo, el controlador actualiza la vista correspondiente para reflejar los cambios y mostrar la respuesta al usuario.
* **Manejo de navegación**: En algunos casos, el controlador también puede manejar la navegación entre diferentes vistas, redirigiendo al usuario a una página diferente en función de ciertas condiciones o acciones

## 5.- MVC Front Controller

El Front Controller es un patrón de diseño utilizado en el desarrollo de aplicaciones web para **centralizar la gestión de las solicitudes** del cliente. A diferencia de la Vista en el patrón MVC, el Front Controller no se ocupa de la presentación de datos al usuario, sino que se encarga de **direccionar y controlar** el flujo de todas las solicitudes entrantes en la aplicación web.

* **Gestión de rutas y solicitudes**: El Front Controller es responsable de recibir todas las solicitudes entrantes de los clientes y decidir qué controlador o componente de la aplicación debe manejar la solicitud en función de la URL solicitada.
* **Control de flujo**: Una vez que el Front Controller determina qué controlador debe manejar la solicitud, dirige la solicitud a ese controlador específico para su procesamiento. Esto permite centralizar la lógica de enrutamiento y control de flujo en un solo lugar.
* **Pre y Post-Procesamiento**: Puede realizar tareas de pre y post-procesamiento de las solicitudes, como autenticación de usuarios, registro de solicitudes, manipulación de errores, etc.
* **Gestión de excepciones**: El Front Controller puede manejar las excepciones que se producen durante el procesamiento de las solicitudes y tomar medidas apropiadas, como mostrar páginas de error personalizadas o redirigir a otras vistas.

### Diferencias con la Vista en el patrón MVC:

El Front Controller se centra en la gestión de solicitudes y el control de flujo en la aplicación web, mientras que la Vista en el MVC se centra en la presentación de datos al usuario.

El Front Controller es un componente fundamental en la arquitectura de una aplicación web, mientras que la Vista es solo una de las capas del patrón MVC, encargada específicamente de la presentación de datos.

## 6.- MVC Vista

La capa de vista en el patrón MVC (Modelo-Vista-Controlador) tiene como función principal la **presentación de la información** al usuario final en una interfaz gráfica o de usuario. Aquí hay un desglose de sus funciones y su importancia:

* **Presentación de datos**: La vista es responsable de mostrar los datos proporcionados por el modelo al usuario final de manera adecuada y comprensible. Esto puede incluir la presentación de información en forma de texto, imágenes, tablas, gráficos, etc.
* **Interacción con el usuario**: La vista maneja la interacción directa del usuario con la aplicación, como clics de botones, entradas de formularios, selección de elementos de menú, etc.
* Diseño de la interfaz de usuario: Define la apariencia visual de la aplicación, incluyendo el diseño, los estilos, los colores y otros aspectos relacionados con la experiencia del usuario.
* **Validación de entrada**: En algunos casos, la vista puede realizar una validación básica de la entrada del usuario antes de enviarla al controlador para su procesamiento.

### Conexión con las demás capas

**Conexión con el controlador**: La vista se comunica con el controlador para enviar solicitudes del usuario y recibir respuestas. Cuando el usuario interactúa con la interfaz de usuario, como hacer clic en un botón, la vista envía una solicitud al controlador correspondiente para que este la maneje.

**Conexión con el modelo**: Aunque la vista no interactúa directamente con el modelo, puede solicitar datos específicos al controlador, que a su vez puede obtenerlos del modelo y devolverlos a la vista para su presentación.

**Conexión con la lógica de presentación**: La vista puede contener cierta lógica de presentación para formatear y organizar los datos recibidos del controlador antes de mostrarlos al usuario. Sin embargo, esta lógica debe ser mínima para mantener la separación de preocupaciones.

## 7.- Request

Representa la **solicitud realizada** por el cliente al servidor web. Esta solicitud puede incluir información como la URL solicitada, los parámetros de la consulta, los encabezados HTTP y, en el caso de formularios, los datos del formulario. El objeto Request es procesado por el servidor web y se utiliza para determinar qué acción debe tomar el servidor en respuesta a la solicitud del cliente.

## 8.- Response

Representa la **respuesta enviada** por el servidor web al cliente en respuesta a la solicitud recibida. Esta respuesta puede incluir información como el código de estado HTTP, los encabezados de respuesta y el cuerpo de la respuesta (generalmente HTML, XML, JSON u otros datos. El objeto Response es generado por el servidor web en función de la lógica de la aplicación y se envía de vuelta al cliente para su procesamiento y visualización en el navegador.

## 9.- Browser

El navegador web, aunque se utiliza comúnmente para mostrar la interfaz de usuario de una aplicación web, tiene una función diferente a la capa de vista en el patrón MVC (Modelo-Vista-Controlador).

* **Navegador web**: El navegador web es una aplicación que se ejecuta en el dispositivo del usuario y se utiliza para acceder a contenido en la World Wide Web. Su función principal es **interpretar y renderizar** documentos HTML, CSS y JavaScript para que los usuarios puedan interactuar con ellos.
* **Capa de vista en MVC**: La capa de vista en el patrón MVC es una parte de la arquitectura de una aplicación web que se encarga de presentar datos al usuario final en una interfaz gráfica o de usuario. Mientras que el navegador web se encarga de la presentación final de la vista al usuario, la capa de vista en MVC se ocupa de generar la estructura y el contenido de la interfaz de usuario en función de los datos proporcionados por el controlador.
* **Interacción con la aplicación**: Aunque el navegador web puede mostrar la interfaz de usuario generada por la capa de vista, su función principal es permitir que los usuarios interactúen con la aplicación web a través de la entrada del teclado y el mouse, y mediante la interpretación de eventos como clics, desplazamientos y envíos de formularios. La capa de vista en MVC, por otro lado, no interactúa directamente con el usuario, sino que se comunica con el controlador para enviar y recibir datos.

## 10.- Contenedor Web

Un contenedor web (también conocido como servidor de aplicaciones web o contenedor de servlets) es un entorno de ejecución que **proporciona soporte** para la ejecución de aplicaciones web Java. Este entorno permite la ejecución de servlets, que son componentes Java diseñados para manejar solicitudes HTTP y generar respuestas dinámicas. Un contenedor web también puede admitir otros componentes de Java EE, como páginas JSP (JavaServer Pages) y componentes Enterprise JavaBeans (EJBs).Para el MVC un contenedor web se utiliza para implementar la capa de controlador de una aplicación web. Aquí hay una descripción de cómo se aplica el contenedor web con el patrón MVC: