2024

1-3-2024

MotoSave



Victor Lopez

Israel Ramirez

David Gonález Vilches

Índice

[1 Introducción a Motosave 1](#_Toc159494171)

[1.1 Descripción 1](#_Toc159494172)

[1.2 Funcionalidad 1](#_Toc159494173)

[1.3 Alcance 1](#_Toc159494174)

[2 ORM 1](#_Toc159494175)

[2.1 Definición 1](#_Toc159494176)

[2.2 Valoración de ventajas 1](#_Toc159494177)

[2.3 Herramientas ORM más utilizadas 2](#_Toc159494178)

[2.3.1 Doctrine (PHP) 3](#_Toc159494179)

[2.3.2 Hibernate (Java) 3](#_Toc159494180)

[2.3.3 Entity Framework (C#): 5](#_Toc159494181)

[3 Diferencias entre ORM y JPA 6](#_Toc159494182)

[4 Bases de Datos Documentales vs Bases de Datos Relacionales 6](#_Toc159494183)

[4.1 Ventajas 6](#_Toc159494184)

[4.2 Inconvenientes 7](#_Toc159494185)

[5 UML de Motosave 7](#_Toc159494186)

[6 Entidad-relación de la Base de Datos de Motosave 7](#_Toc159494187)

[7 Uso y funcionamiento de Motosave 8](#_Toc159494188)

[7.1 Uso general 8](#_Toc159494189)

[7.2 Estructura de clases 8](#_Toc159494190)

[7.2.1 Clases (por completar) 8](#_Toc159494191)

[7.3 Funcionamiento detallado 8](#_Toc159494192)

[7.3.1 Método no se que (que sea complejo) descripción detallada y explicación. 8](#_Toc159494193)

[8 Elementos añadidos a los contenidos mínimos 8](#_Toc159494194)

[8.1 Diagrama de flujo. 8](#_Toc159494195)

[8.2 Librerías, frameworks y plugins 8](#_Toc159494196)

[8.3 Otros (por si hubiera) 8](#_Toc159494197)

[8.4 Seguridad 8](#_Toc159494198)

[8.4.1 Seguridad de datos 8](#_Toc159494199)

# Introducción a Motosave

## Descripción

## **Motosave: Gestión Integral para Ventas de Motocicletas**

Motosave es una aplicación de escritorio innovadora diseñada para optimizar las operaciones de venta y gestión en el sector de motocicletas. La plataforma se bifurca en dos interfaces especializadas que atienden a las necesidades de los administradores y comerciales de una empresa de motocicletas, respectivamente.

**Para el administrador**, Motosave es una herramienta de gestión integral que ofrece:

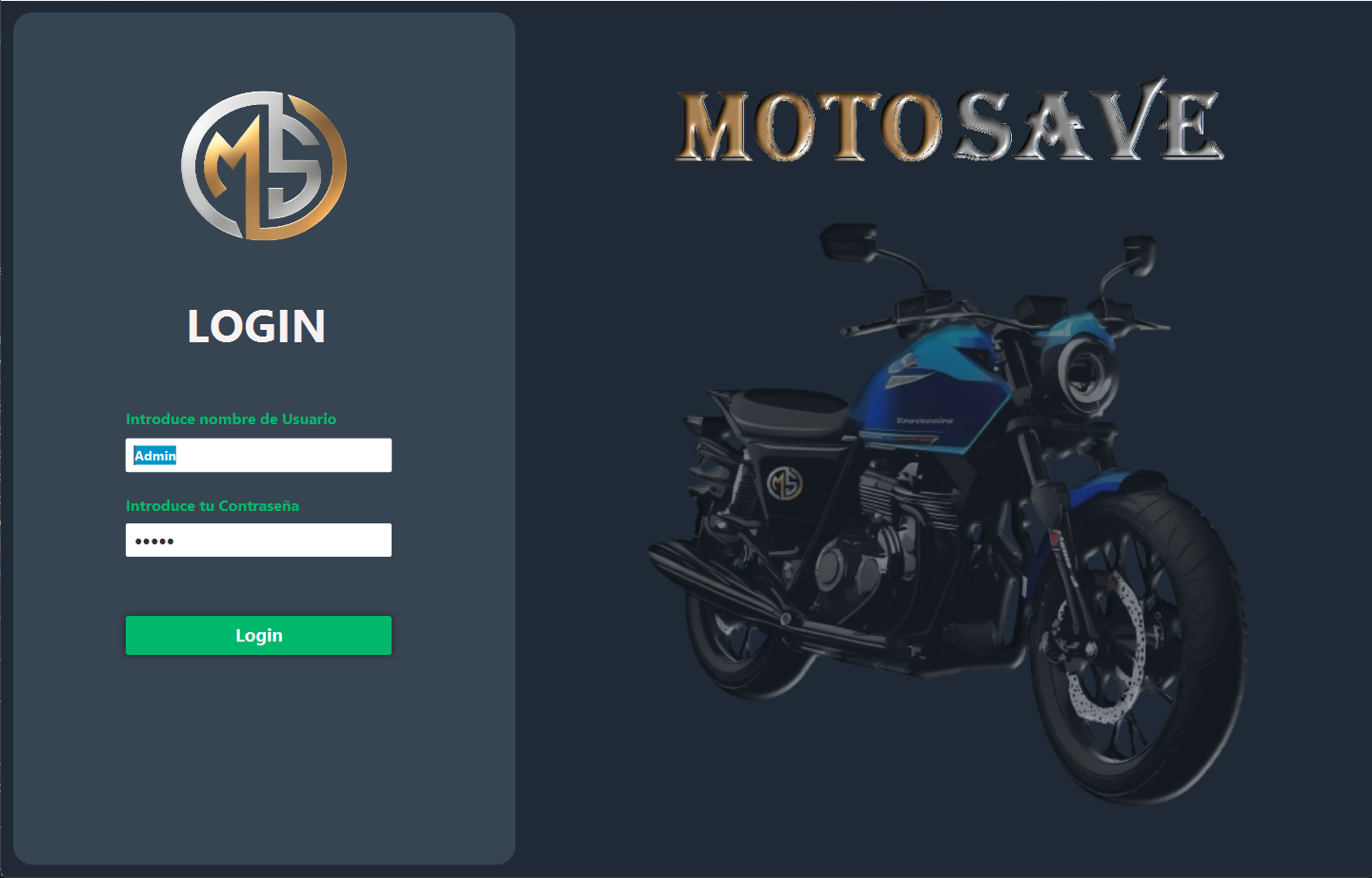
* **Gestión de Stock**: Un sistema intuitivo para el seguimiento y control del inventario, asegurando la disponibilidad de productos y la gestión eficiente de los recursos.
* **Gestión de Personal**: Facilidades para la administración de recursos humanos.
* **Estadísticas**: Información y análisis de las ventas realizadas por el personal.

**Para el comercial**, la aplicación facilita:

* **Venta de Motocicletas**: Un proceso de ventas ágil y simplificado que permite a los comerciales realizar ventas de manera intuitiva.
* **Estadísticas**: Acceso a datos sobre las ventas realizadas.

## Funcionalidad

Motosave presenta una ventana de inicio de sesión al arranque. En ella el usuario introduce sus credenciales y accede a la sección del aplicativo que le corresponda.



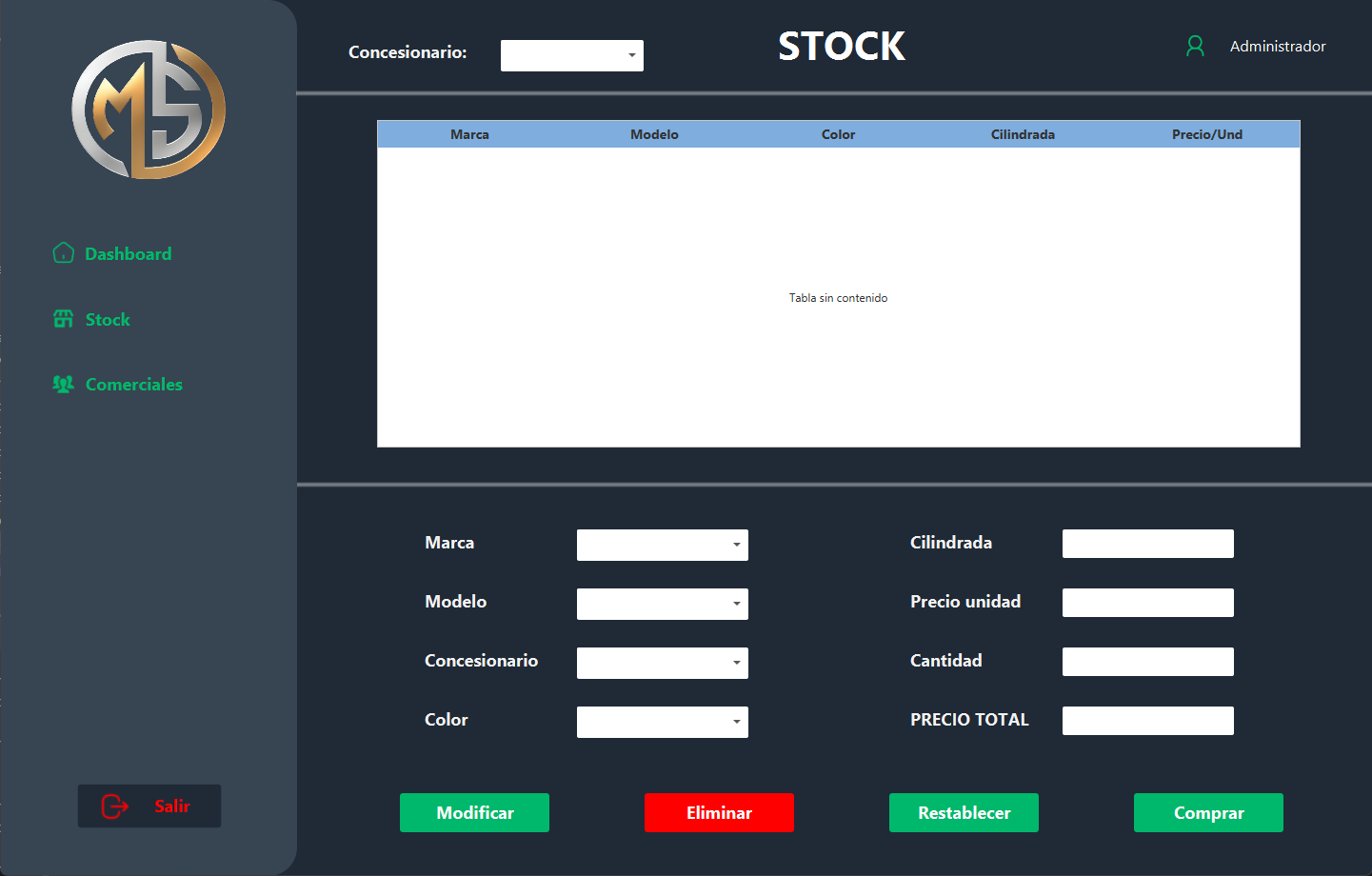
**Para el Administrador:**

* **Gestión de Stock**:

**Descripción**: Esta funcionalidad permite a los administradores monitorear y controlar el inventario de motocicletas de los diferentes concesionarios. Pueden agregar nuevos modelos, actualizar la ubicación o el color de un modelo, y eliminar motocicletas que ya no se venden.

**Detalles**: La parte superior de la interfaz muestra una tabla donde se visualiza todo el catálogo de motocicletas disponibles en cada concesionario. El administrador podrá seleccionar el concesionario del que quiere visualizar el stock mediante un desplegable.

La parte inferior contiene un formulario para realizar la compra de nuevos modelos. Este mismo formulario se autocompleta con los datos de un modelo al ser seleccionado en la tabla para que se puedan modificar ciertos datos.



* **Gestión de Personal**:

**Descripción**: Esta herramienta permite gestionar los datos del personal, añadir y eliminar comerciales.

**Detalles**: En la parte superior de la interfaz, se encuentra un formulario donde agregamos la información de los nuevos comerciales.

En la parte inferior, se puede visualizar y seleccionar los comerciales de una tabla para eliminarlos.

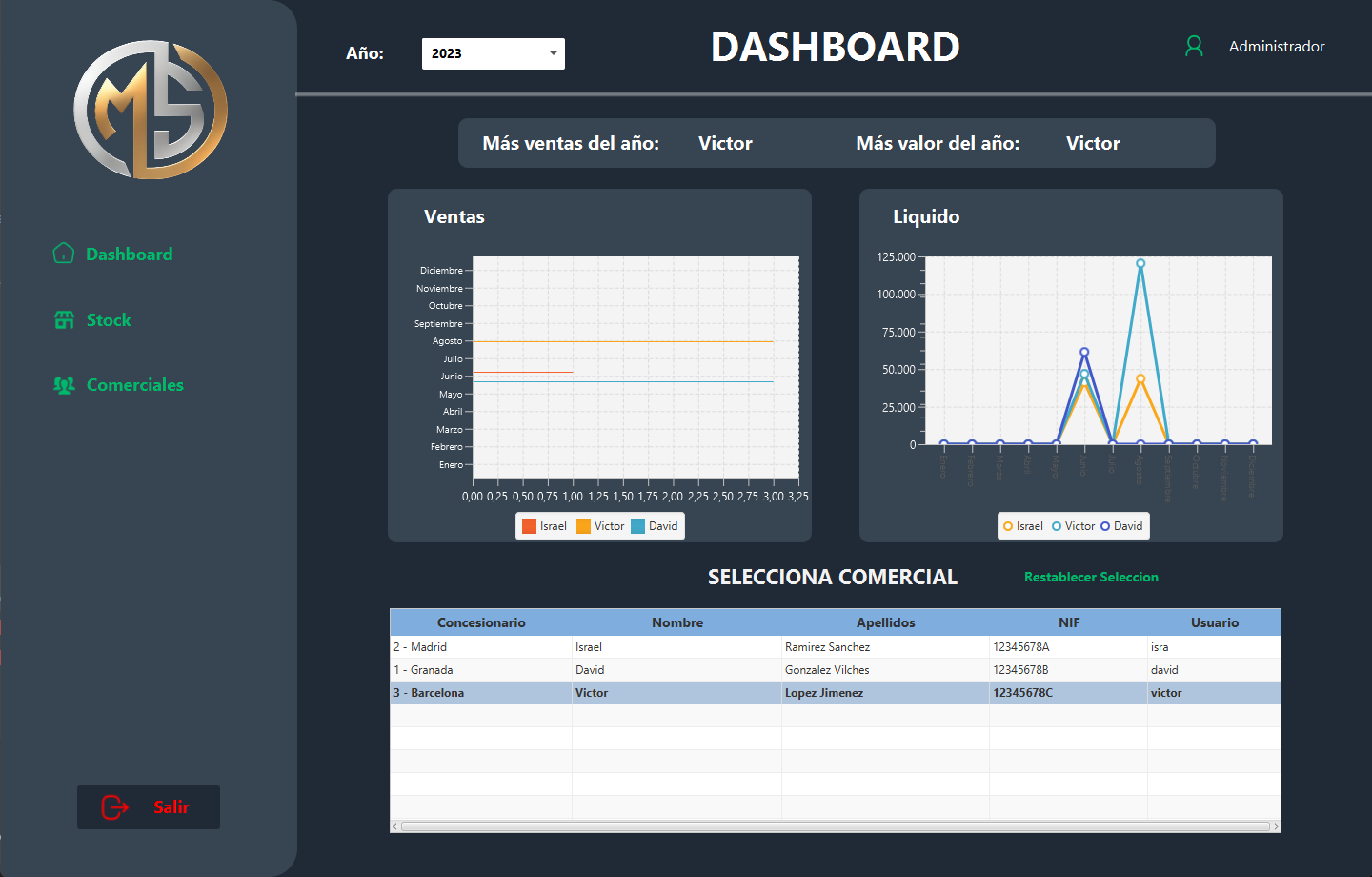


* **Estadísticas**:

**Descripción**: Ofrece informes y análisis estadísticos sobre diversos aspectos del negocio, como el número de ventas realizadas, el líquido total percibido de dichas ventas, el comercial que más ventas a realizado y el comercial que más líquido ha generado con sus ventas en un periodo de tiempo.

**Detalles**: La interfaz dispone de dos gráficas, una de barras que muestra el número de ventas que ha realizado cada comercial, y una de líneas que muestra el valor que ha generado cada comercial con sus ventas.

Además, en la parte inferior se podrá seleccionar un comercial en particular de una tabla para ver sus estadísticas individuales y un desplegable en la parte superior para seleccionar el año del cual se quiere ver las estadísticas.



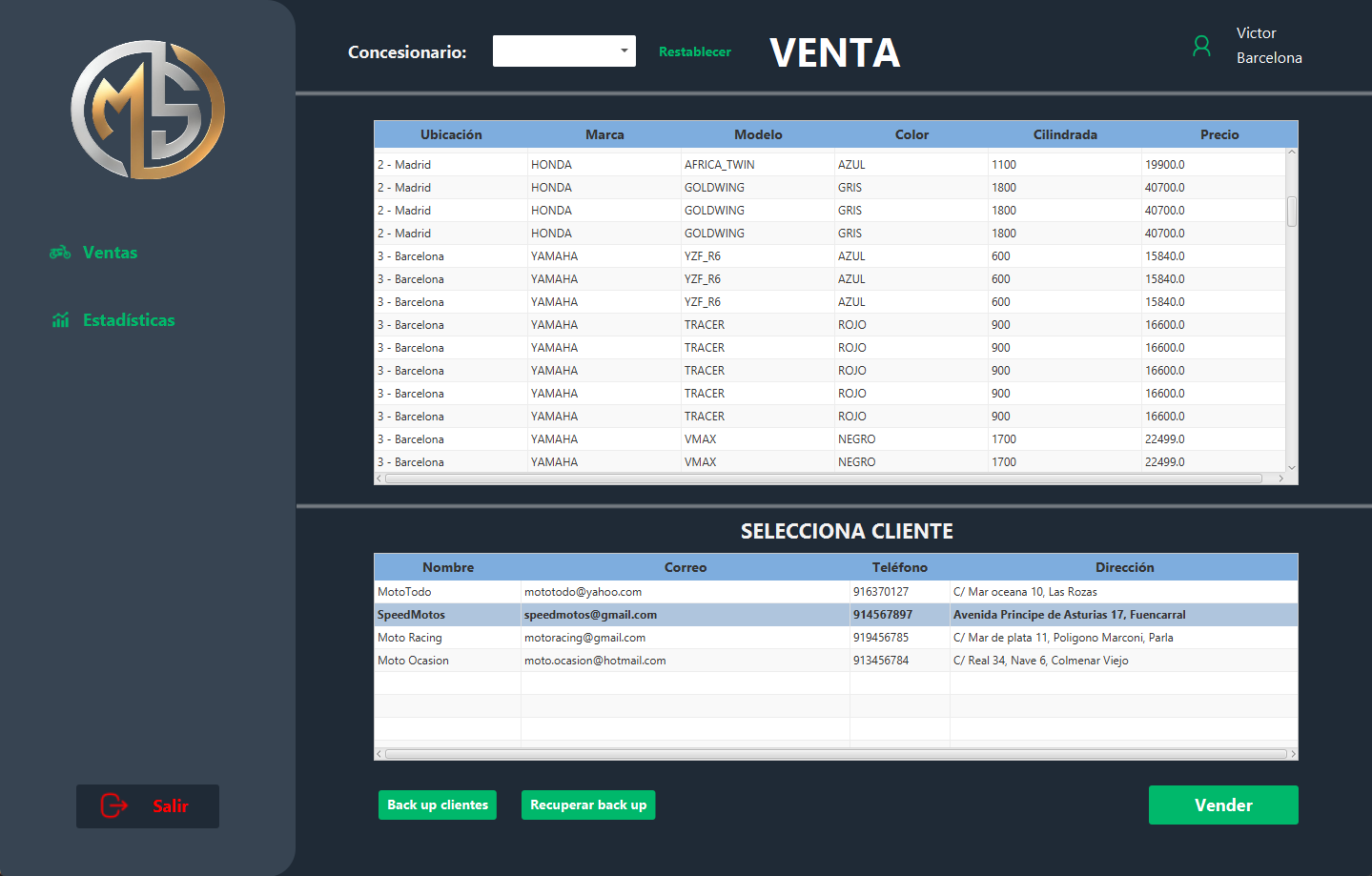
**Para el Comercial:**

* **Venta de Motocicletas**:

**Descripción**: Permite a los comerciales gestionar todo el proceso de venta, desde la demostración de productos hasta la finalización de la compra.

**Detalles**: La interfaz muestra una tabla que contiene todas las motos disponibles y otra con los clientes a los que se pueden vender las motocicletas. El desplegable de la parte superior permite filtrar las motocicletas por un concesionario en particular.

Además, se cuenta con dos botones en la parte inferior izquierda para realizar y recuperar una copia de seguridad de los clientes (Ver [7.3.2 Lectura y Escritura de XML](#_Lectura_y_Escritura)).



* **Estadísticas**:

**Descripción**: Proporciona a los comerciales acceso a sus propias estadísticas de ventas, permitiéndoles monitorear su rendimiento anual.

**Detalles**: La interfaz muestra una gráfica de barras con el número de ventas que ha realizado el comercial y otra gráfica que muestra el valor total de las ventas realizadas. El usuario comercial podrá seleccionar el año del que quiere ver sus estadísticas en un desplegable ubicado en la parte superior de la interfaz.



## Alcance

**Alcance para la Sección de Administrador**:

* **Monitoreo y Control de Inventario**: Habilidad para visualizar y administrar el stock de motocicletas de varios concesionarios.
* **Actualización de Datos**: Funcionalidad para modificar la información del inventario, como la ubicación o el color de las motocicletas.
* **Gestión de Catálogo**: Añadir nuevos modelos al catálogo y eliminar los que ya no están en venta.
* **Interfaz de Usuario**: Incluye una tabla interactiva con filtros y un formulario para la compra y modificación de motocicletas.
* **Administración de Datos de Empleados**: Posibilidad de añadir nuevos comerciales y eliminar los existentes.
* **Visualización y Gestión**: Interfaz con formularios y tablas para gestionar la información del personal de manera eficiente.
* **Análisis de Ventas**: Generación de informes y análisis estadísticos sobre ventas y rendimiento de los comerciales.
* **Visualización de Datos**: Incluye gráficas de barras y de líneas para una representación visual del rendimiento.
* **Filtrado por Período**: Opción para seleccionar el año de interés y visualizar las estadísticas correspondientes.

**Alcance para la Sección de Comercial**:

* **Gestión de Ventas**: Herramientas para manejar el proceso completo de ventas, incluyendo presupuestos y facturación.
* **Información de Productos**: Acceso a detalles de los productos para facilitar la asesoría a clientes.
* **Rendimiento Individual**: Acceso a estadísticas personales para monitorear el rendimiento en ventas.
* **Representación Gráfica**: Gráficas de barras para visualizar el número y valor de las ventas.
* **Selección de Periodo**: Funcionalidad para elegir el año para el cual se desean ver las estadísticas.

**Limitaciones y Restricciones**:

* **Especialización en Motocicletas**: La aplicación está diseñada específicamente para la gestión de motocicletas y no abarca otros productos o servicios.
* **Interfaz de Escritorio**: Presumiblemente, la aplicación está pensada para su uso en ordenadores de escritorio y no cuenta con versiones móviles o adaptaciones web.
* **Enfoque en Ventas y Gestión de Stock/Personal**: El alcance se centra en la gestión interna y las ventas, sin incluir módulos para clientes como seguimiento de servicios postventa o CRM.

**Proyecciones Futuras**:

* **Expansibilidad**: Motosave está preparado para ser ampliado a un número mayor de concesionarios, clientes y comerciales, además de poderse adaptar a cualquier otro tipo de negocio que precise una gestión de stock, ventas y recursos humanos.
* **Integración de Servicios**: Motosave es compatible con módulos adicionales como atención al cliente o marketing.
* **Migración de plataforma**: Nuestro aplicativo será llevado a diversas plataformas móviles y navegadores.

# ORM

## Definición

Las herramientas **ORM** (Object Relational Mapping) son modelos de programación que desempeñan un papel fundamental al **simplificar y automatizar** las tareas relacionadas con el manejo de datos en aplicaciones.

La función principal de un ORM es **transformar** las **estructuras de datos** de una base de datos relacional en **objetos** que pueden ser utilizados en el código de programación. En otras palabras, actúan como un **puente** entre el mundo de la **programación orientada a objetos** y las **bases de datos relacionales**.

Estas herramientas permiten a los desarrolladores trabajar con objetos en el código, abstrayéndolos de las complejidades de la manipulación directa de las tablas de la base de datos.

Al utilizar un ORM, las **operaciones** de acceso a datos, como la creación, lectura, actualización y eliminación (CRUD), **se simplifican**, lo que **ahorra tiempo y esfuerzo en el desarrollo de aplicaciones.**

## Valoración de ventajas

**Simplicidad en el Desarrollo:**

Al funcionar como una capa intermedia separada de la base de datos, las herramientas ORM permiten a los desarrolladores centrarse exclusivamente en el desarrollo de la aplicación, simplificando el proceso.

**Facilita Migraciones:**

Simplifica las migraciones, ya que cambiar la base de datos no requiere reconstruir todas las consultas. La aplicación puede migrarse a otra base de datos sin problemas.

Las herramientas ORM facilitan la portabilidad del código entre diferentes sistemas de gestión de bases de datos, ya que el código interactúa con objetos en lugar de consultas SQL específicas de una base de datos.

**Unificación del Lenguaje:**

Elimina la necesidad de que los programadores conozcan un lenguaje específico para cada base de datos, ya que el ORM unifica el lenguaje de acceso a los datos.

**Velocidad en Tareas Básicas:**

Proporciona mayor velocidad en tareas básicas relacionadas con el acceso a los datos, mejorando la eficiencia en operaciones comunes.

**Código Más Legible y Conciso:**

El código generado por las herramientas ORM suele ser más legible y requerir menos líneas en comparación con la escritura manual de consultas SQL.

**Mayor Seguridad:**

Al ser una capa independiente de los datos, las herramientas ORM contribuyen a la seguridad al protegerlos de ciberataques al operar en un nivel separado.

**Evita la Escritura Manual de Consultas SQL:**

Elimina la necesidad de escribir a mano las consultas SQL necesarias, simplificando el desarrollo y reduciendo posibles errores.

**Facilita el Trabajo con Operaciones Básicas (CRUD):**

Simplifica el manejo de operaciones CRUD (Create, Read, Update, Delete), facilitando la interacción con la base de datos.

**Guarda y Carga Información Automáticamente:**

Una de las ventajas más atractivas es que un ORM guarda y carga automáticamente toda la información de una base de datos relacional.

## Herramientas ORM más utilizadas

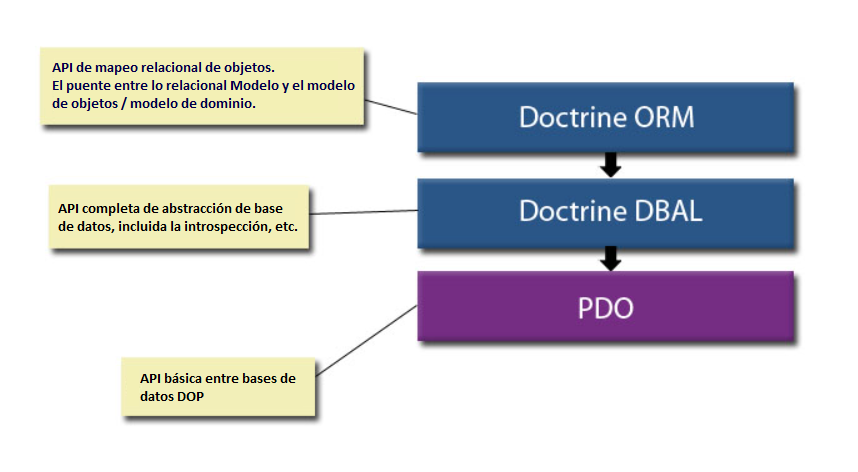
### Doctrine (PHP)



**Descripción**: Doctrine es una herramienta ORM para PHP que destaca por su código de alta calidad y estabilidad. Ofrece un conjunto de funciones poderosas y flexibles para la manipulación de datos en bases de datos relacionales.

**Características Clave**:

* Consultas flexibles y potentes.
* Capacidad para trabajar con modelos de datos complejos
* Facilita la gestión de esquemas de bases de datos.



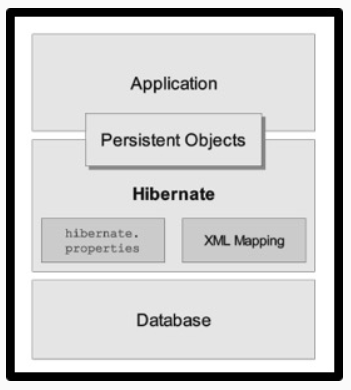
### Hibernate (Java)



**Descripción**: Hibernate es una herramienta ORM ampliamente utilizada en el entorno Java. Utiliza XML o anotaciones para mapear objetos a las tablas de la base de datos y se destaca por su capacidad para gestionar la persistencia de datos de manera eficiente.

**Características Clave:**

* Mapeo objeto-relacional mediante XML o anotaciones.
* Integración con Java Persistence API (JPA).
* Gestión eficiente de la persistencia de objetos en bases de datos relacionales.



### 

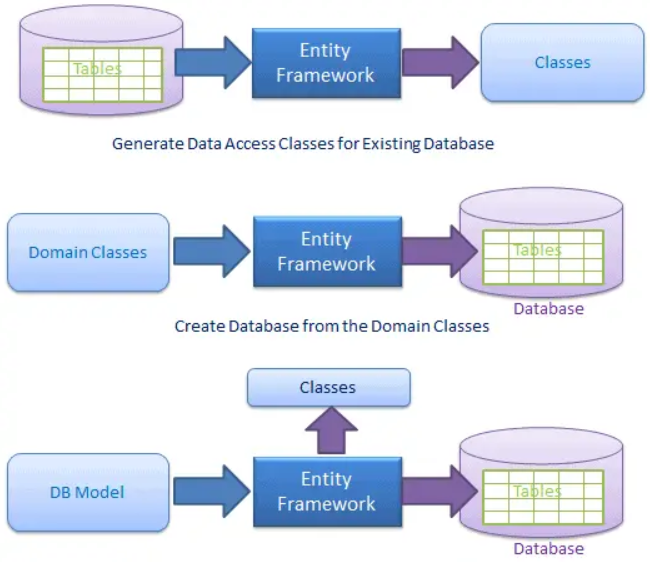
### Entity Framework (C#):



**Descripción**: Entity Framework es la herramienta ORM de Microsoft para el entorno .NET. Permite a los desarrolladores trabajar con datos relacionales utilizando objetos específicos de C#.

**Características Clave:**

* Modelado de datos a través de Entity Framework Designer.
* Admite la creación de nuevos modelos o la vinculación a bases de datos existentes.
* Facilita operaciones CRUD y manipulación de esquemas de bases de datos.



# Diferencias entre ORM y JPA

ORM es el concepto general de mapear objetos a tablas de base de datos, mientras que JPA es una especificación en Java que sigue el enfoque ORM. JPA proporciona estándares para el mapeo objeto-relacional en Java, y Hibernate es una de las implementaciones más populares de JPA. En otras palabras, JPA es un conjunto de especificaciones y estándares, mientras que Hibernate es una implementación concreta de esas especificaciones.

# Bases de Datos Documentales vs Bases de Datos Relacionales

**Las bases de datos documentales:** Son un tipo de sistema de gestión de bases de datos (DBMS) que almacenan, recuperan y gestionan datos en formato de documentos, generalmente en formato JSON o BSON. Cada documento puede contener datos semiestructurados o no estructurados.

**Las bases de datos relacionales**: Son sistemas de gestión de bases de datos que organizan los datos en tablas relacionadas entre sí. Utilizan un esquema predefinido y un lenguaje estructurado de consulta (SQL) para acceder y manipular los datos.

## Ventajas

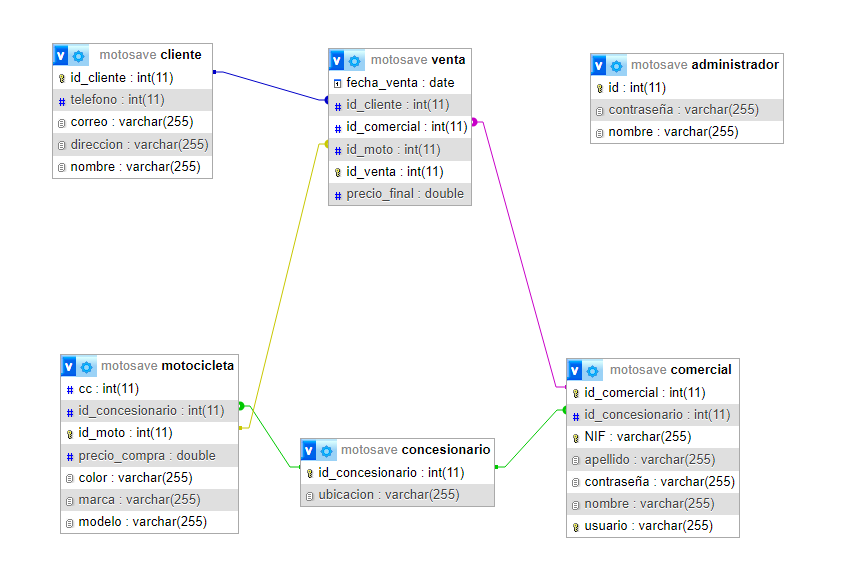
|  |  |
| --- | --- |
| **Bases de Datos Documentales** | **Bases de Datos Relacionales** |
| **Flexibilidad en la Estructura de Datos**:  Las bases de datos documentales permiten almacenar datos no estructurados, como documentos JSON o BSON, lo que brinda flexibilidad en la estructura de los datos. | **Estructura de Datos Rigurosa**:  Ofrecen una estructura de datos rigurosa con relaciones definidas entre tablas, lo que facilita la representación de datos complejos y relaciones entre entidades. |
| **Escalabilidad Horizonta**l:  Son más adecuadas para escalar horizontalmente, ya que permiten agregar nodos de forma eficiente, facilitando la gestión de grandes volúmenes de datos distribuidos | **Soporte para Transacciones ACID**:  Las bases de datos relacionales garantizan transacciones ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad), proporcionando coherencia y seguridad en operaciones críticas. |
| **Manejo de Datos No Estructurados**:  Eficientes para almacenar y recuperar datos no estructurados como documentos, imágenes y videos, comunes en entornos web y aplicaciones modernas. | **Optimización para Consultas Relacionales**:  Son altamente eficientes para consultas que involucran relaciones complejas, ya que están optimizadas para operaciones join y consultas relacionadas. |
| **Rendimiento en Operaciones de Lectura y Escritura**:  Suelen ser eficientes en operaciones de lectura y escritura, especialmente cuando se trata de documentos completos. | **Madurez y Experiencia**:  Las bases de datos relacionales tienen décadas de experiencia y una amplia adopción en la industria, con una abundancia de herramientas y recursos disponibles. |

## Inconvenientes

|  |  |
| --- | --- |
| **Bases de Datos Documentales** | **Bases de Datos Relacionales** |
| **Complejidad en Consultas Relacionadas**:  Pueden ser menos eficientes para consultas que involucran relaciones complejas entre diferentes tipos de documentos, ya que no están optimizadas para operaciones join. | **Menos Flexibilidad Estructural**:  Pueden resultar menos flexibles al tratar con datos no estructurados o con estructuras de datos que pueden cambiar con el tiempo. |
| **Consistencia y Transacciones**:  Algunas bases de datos documentales sacrifican la consistencia en favor de la disponibilidad y la tolerancia a la partición, lo que puede afectar la integridad de los datos en ciertos escenarios. | **Dificultades en Escalabilidad Horizontal**:  Escalar horizontalmente puede ser más desafiante, ya que la coherencia de los datos en múltiples nodos puede ser difícil de mantener. |
| **Mayor Overhead en Operaciones No Relacionales**:  Para operaciones que involucran datos no relacionales o estructuras más complejas, las bases de datos relacionales pueden tener un mayor overhead en comparación con las bases de datos documentales. |

# UML de Motosave

# Entidad-relación de la Base de Datos de Motosave



# Uso y funcionamiento de Motosave

## Uso general

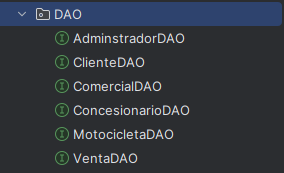
El diseño lógico

## Estructura de clases

### DAO

El paquete DAO esta dividido en 6 interfaces que corresponden a cada objeto de nuestro aplicativo. Las interfaces contienen los métodos relacionados con cada modelo para su correcta centralización y fácil acceso.

Estas interfaces son implementadas por las clases contenidas en el paquete ‘ImplementacionesDAO’ (Ver [7.2.5 Implementaciones DAO](#_Implementaciones_DAO)).



### DATA

El paquete DATA contiene 3 clases peculiares y que requieren **especial atención** para entender el correcto funcionamiento de Motosave.

#### ComercialLoggeado

**Explicación**: Puesto que en Motosave trabajamos con ORM para guardar nuestros datos, necesitamos tener una instancia del comercial que ha iniciado sesión.

El problema que encontramos es mantener esa misma instancia durante toda la ejecución ya que, al momento de realizar ventas es necesario añadir un comercial. Por lo tanto, para nosotros es crucial mantener esa instancia única.

Para ello, adaptamos el patrón ‘Singleton’ (Ver [8.3.1 Patron ‘Singleton’](#_Patrón_‘Singleton’)) en esta clase. A diferencia de una clase con patron ‘Singleton’ normal, ComercialLoggeado obtiene la instancia mediante un método setter cuando se inicia sesión correctamente en la ventana ‘Login’ (Ver [1.2 Funcionalidades](#_Funcionalidad))

**Atributos de clase:**

1. Instancia de ComercialLoggeado:

Para poder almacenar correctamente la instancia única, la instancia se guarda en forma de atributo privado estático para poder ser recuperado cuando se precise.



**Métodos**:

1. **getComercialLoggeado():**

Este método devuelve la instancia del comercial registrado. Es la **única** forma de acceder a la instancia.



1. **setComercialLoggeado():**

Este método define la instancia del ComercialLoggeado cada vez que se inicie sesión. Cabe destacar que el Administrador **no** se instancia aquí cuando introduzca sus credenciales.



#### Encriptador

**Explicación**: Encriptador es una clase auxiliar que utilizamos para encriptar las contraseñas antes de guardarlas en la base de datos.

**Métodos:**

1. **encriptarContraseña():**

Recibe un parámetro tipo String con la cadena de texto que se quiere encriptar y devuelve el String encriptado (Ver [7.3.1 ‘Metodo encriptarContraseña’](#_Método_‘encripitarContraseña’.)).

#### LOAD

**Explicación**: En la primera versión de Motosave, trabajábamos con una base de datos relacional, por lo que generábamos la base de datos mediante un fichero .sql. En esta versión, como trabajamos con Hibernate, necesitábamos una forma de cargar los datos de manera rápida.

Para ello creamos la clase LOAD. Esta contiene instancias de todo lo que necesita Motosave para funcionar y probarse sin ningún tipo de problema.

La clase Load también con contiene una variable tipo Double con el valor del beneficio que se quiera obtener por cada motocicleta vendida. Se utiliza para mostrar al comercial el valor total de la motocicleta cuando va a realizar la venta, y es el valor que se queda registrado cuando se realiza la venta.

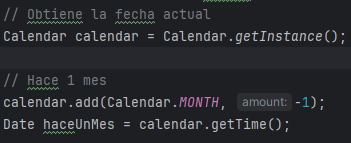


**Constructor:**

En el constructor de la clase Load llamamos a los métodos ‘crear’ de las clases del paquete ImplementacionesDAO (Ver [7.2.5 ‘Implementaciones DAO’](#_Implementaciones_DAO)) para generar todos los datos cuando se haga una instancia de Load.

**Otros:**

Para poder visualizar datos en las gráficas de las vistas ‘Estadísticas’, necesitamos ventas de fechas pasadas. Para poder ‘falsear’ las fechas utilizamos la clase ‘Calendar’ y creamos fechas pasadas para realizar diversas ventas.



### Enumerados

### FactoryMoto

### Implementaciones DAO

### Modelos

### MotosaveFX

### Persistencia

## Funcionamiento detallado

### Método ‘encriptarContraseña’.

### Lectura y Escritura de XML

### MiEntityManager

idea de cómo abordar la documentación:

Introducción: Explica el propósito de la clase miEntityManager, destacando su rol en la gestión de conexiones a bases de datos a través de JPA.

Descripción del Patrón Singleton: Detalla cómo se implementa este patrón para garantizar una única instancia de EntityManager en toda la aplicación.

Métodos y Funciones:

getEntityManager(): Describe cómo este método verifica si ya existe una instancia de EntityManager. Si no es así, crea una nueva utilizando Persistence.createEntityManagerFactory.

Ejemplo de Uso:

Incluye el fragmento de código proporcionado como ejemplo de cómo se utiliza miEntityManager para obtener una instancia de EntityManager y utilizarla en operaciones de base de datos.

### ComercialLoggeado

# Elementos añadidos a los contenidos mínimos

## Diagrama de flujo.

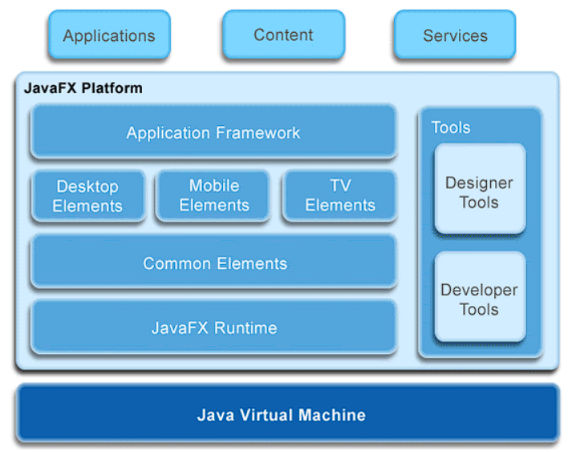


## Librerías, frameworks y plugins

### Java FX

JavaFX es una plataforma con un conjunto de paquetes de gráficos que permite a los desarrolladores diseñar, crear, probar, depurar e implementar aplicaciones de cliente enriquecido (integrando gráficos vectoriales, animación y sonido) que operen en diversas plataformas.

JavaFX amplía la potencia de Java permitiendo a los desarrolladores utilizar cualquier biblioteca de Java en aplicaciones JavaFX.



**Características de JavaFX**

* Acceso a la API de Java.
* Diseñador de interfaces con Scene Builder (FXML)
* Interoperabilidad con Java Swing.

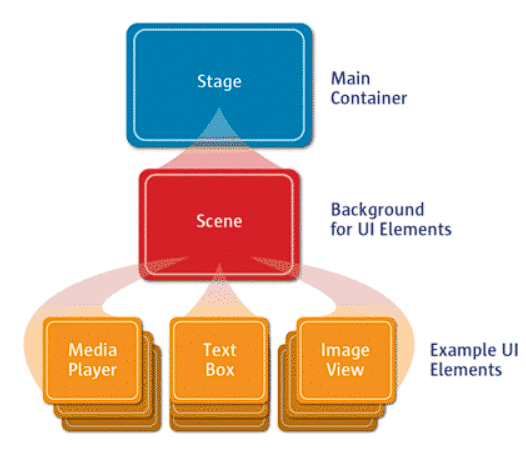
**Compatibilidad con CSS 3 (Controles de UI)**

* Aplicación del patrón MVC (Modelo Vista Controlador)
* Se puede integrar un módulo con compatibilidad web (webview)
* Proporciona un motor de Javascript.
* Incluido hasta la versión del Java JDK 8 (A partir de entonces, se debe integrar como una librería aparte)

**Estructura aplicación JavaFx**

Las aplicaciones JavaFX poseen tres componentes esenciales, los cuales son:

* Stage (escenario)
* Scene (escena)
* Node (nodo)



**Configuración del entorno de desarrollo:**

JavaFX es una biblioteca de software de código abierto y parte de la plataforma Java. Antes de comenzar a desarrollar aplicaciones con JavaFX, es necesario configurar el entorno de desarrollo adecuado. A continuación se detallan los pasos para configurar el entorno de desarrollo en diferentes entornos de desarrollo integrado (IDE) como IntelliJ IDEA, Eclipse y NetBeans:

* Configuración en IntelliJ IDEA:

1. Instalación de IntelliJ IDEA:

Descarga e instala IntelliJ IDEA desde el sitio web oficial (<https://www.jetbrains.com/idea/download/>) .

1. Configuración del JDK:

Abre IntelliJ IDEA y selecciona "Configure" en la pantalla de bienvenida.

Ve a "Project Defaults" > "Project Structure" > "Project".

Selecciona la versión del JDK que deseas usar para tu proyecto.

1. Creación de un proyecto JavaFX:

Selecciona "Create New Project" desde la pantalla de bienvenida.

Elige "JavaFX" en el menú de la izquierda y asegúrate de que esté seleccionado el SDK de Java correcto.

Completa la configuración del proyecto y haz clic en "Finish" para crear el proyecto.

* Configuración en Eclipse:

1. Instalación de Eclipse:

Descarga e instala Eclipse desde el sitio web oficial (<https://www.eclipse.org/downloads/>) .

1. Instalación del plugin e(fx)clipse:

Abre Eclipse y ve a "Help" > "Eclipse Marketplace".

Busca "e(fx)clipse" e instala el plugin.

Reinicia Eclipse después de la instalación.

1. Configuración del JDK:

Ve a "Window" > "Preferences" > "Java" > "Installed JREs".

Agrega el JDK que deseas utilizar si aún no está en la lista.

1. Creación de un proyecto JavaFX:

Selecciona "File" > "New" > "Other" > "JavaFX" > "JavaFX Project".

Completa la configuración del proyecto y haz clic en "Finish" para crear el proyecto.

* Configuración en NetBeans:

1. Instalación de NetBeans:

Descarga e instala NetBeans IDE desde el sitio web oficial (<https://netbeans.apache.org/download/index.html>) .

1. Configuración del JDK:

Abre NetBeans y ve a "Tools" > "Java Platforms".

Haz clic en "Add Platform" y selecciona el JDK que deseas utilizar para tu proyecto.

1. Creación de un proyecto JavaFX:

Selecciona "File" > "New Project" > "JavaFX" > "JavaFX Application".

Completa la configuración del proyecto y haz clic en "Finish" para crear el proyecto.

**Creacción de Interfaces:**

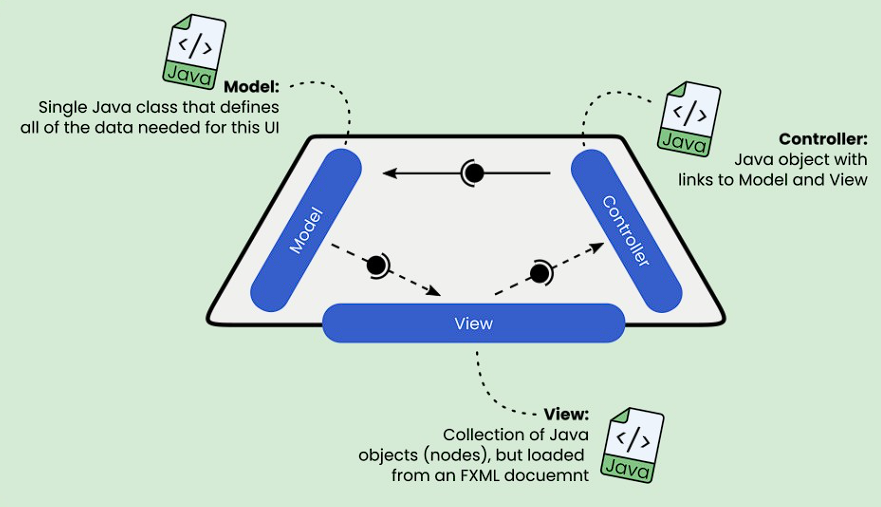
JavaFX ofrece una amplia gama de herramientas y componentes para crear interfaces de usuario (UI) modernas y atractivas. Aquí se destacan los aspectos más importantes para la creación de interfaces con JavaFX:

* FXML y Controladores: Utiliza archivos FXML para describir la estructura de la interfaz y separar la lógica de la aplicación en controladores.
* Controles y Componentes: Accede a una amplia variedad de controles y componentes personalizables, como botones, campos de texto y tablas, que pueden adaptarse fácilmente a las necesidades del proyecto.
* Escenas y Stage: Organiza la UI en escenas, que se muestran en ventanas independientes (Stage), permitiendo la gestión eficiente de la interfaz y la creación de múltiples ventanas según sea necesario.
* Diseño Responsivo: Diseña interfaces responsivas que se ajustan automáticamente a diferentes tamaños de pantalla y resoluciones, garantizando una experiencia de usuario consistente en diversas plataformas.
* Eventos y Manejadores: Implementa la interacción del usuario mediante la detección y el manejo de eventos, como clics de ratón y cambios de valor, facilitando una experiencia de usuario dinámica e intuitiva.
* Gestión de eventos: Manejo de eventos de usuario, como clics de ratón, pulsaciones de teclas, etc. Implementación de acciones y respuestas a eventos en la interfaz de usuario.
* Integración multimedia y gráficos: Uso de clases y APIs de JavaFX para integrar imágenes, audio, video y gráficos en la aplicación.
* Despliegue y distribución de la aplicación: Empaquetamiento y distribución de la aplicación JavaFX para diferentes plataformas (Windows, macOS, Linux). Opciones para distribuir la aplicación como un archivo JAR autónomo o mediante instaladores específicos del sistema operativo.

**Modelo Vista Controlador:**

Desde la perspectiva del Modelo Vista Controlador (MVC) con la que ya trabajamos:

* La lógica de vista debe incluirse en el **controlador** y define cómo se muestra la información y se interactúa con ella.
* La lógica empresarial debe incluirse en el **modelo** y define cómo se puede acceder, crear, almacenar y modificar los datos.
* Cada **Vista** debe tener una narrativa o propósito simple y consistente. Puede haber múltiples vistas dentro de una ventana.

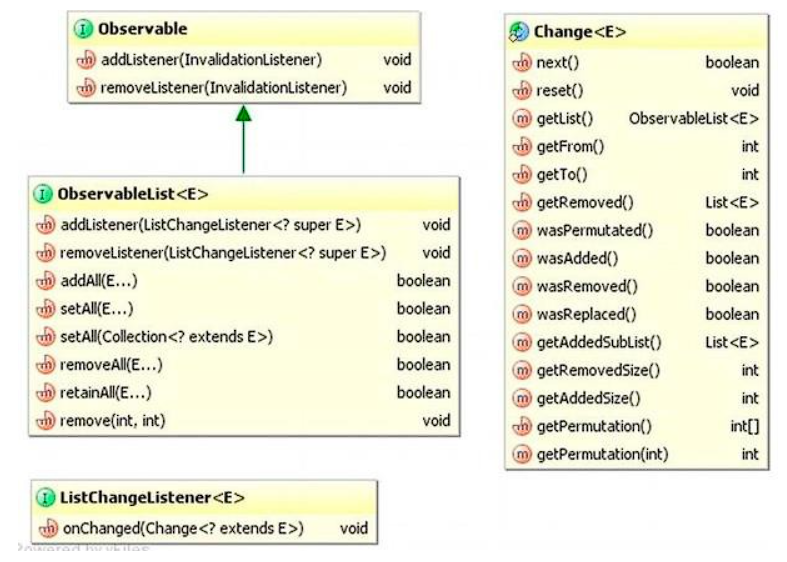


**JavaFX Observable Collections:**

El lenguaje de programación Java cuenta con colecciones como: List, Set, Map. La API de JavaFX extiende estas colecciones con las interfaces : ObservableList, ObservableSet, ObservableMap, con el objetivo de proporcionarle a las colecciones el soporte para la notificación de cambios e invalidación como lo hacen las Propiedades JavaFX.

Estas colecciones se encuentran en el paquete javafx.collections y para crearlas debemos usar la clase FXCollections que nos provee de distintos métodos estáticos para crear la colección que deseemos, veamos un par de ejemplos:

*// crear una colección vacia*  
**ObservableList<String>** textList **=** **FXCollections.**observableArrayList**();**  
  
*// crea la colección a partir de una lista de elementos*  
**ObservableList<Long>** numberList **=** **FXCollections.**observableArrayList**(**1L**,** 2L**,** 3L**)**



### Hibernate

Como método de persistencia hemos utilizado el framework de Hibernate (Ver [Hibernate (Java) 2.3.2](#_Hibernate_(Java))). Para poder implementarlo correctamente en nuestra aplicación hemos utiliza el plugin de IntelliJ IDEA, [JPA Buddy](#_JPA_Buddy).

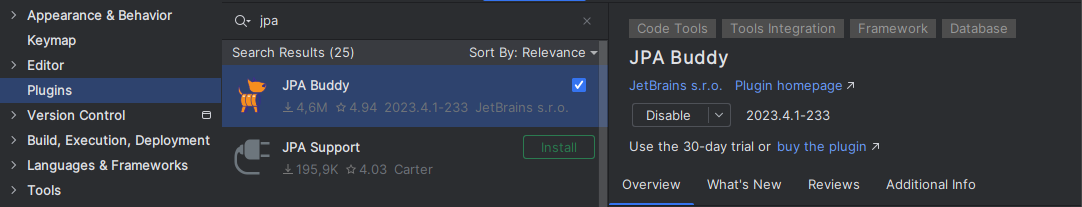
### JPA Buddy

JPA Buddy es un plugin de IntelliJ IDEA que da una solución gratuita a la gestión de ORMs desde el IDE. Este plugin trata de rebajar la barrera del desarrollo con JPA, mejora la productividad del desarrollador y asegura un flujo de código correcto.

JPA Buddy nos proporciona, entre otras ventajas:

* Wizards intuitivos para trabajar con JPA.
* Versionado de bases de datos automático.
* Generación de Entidades de JPA basándose en tablas (ingeniería inversa)

Para descargarlo, iremos a “Settings” > “Plugins” y en la sección ‘Marketplace’, buscaremos “JPA Buddy”:



Para mas información sobre JPA Buddy:

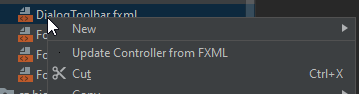
<https://www.jetbrains.com/help/idea/jpa-buddy.html#e9183a5d>

Ver Anexo X ‘Como configurar una base de datos ORM en IntelliJ con JPA Buddy’.

### FXML Manager

El plugin "FXML Manager" para IntelliJ IDEA facilita la sincronización entre archivos FXML y sus controladores Java en aplicaciones JavaFX.





Al seleccionar "Actualizar controlador desde FXML" en el menú contextual de un archivo .fxml, realiza automáticamente varias acciones:

Elimina campos @FXML que ya no están en el FXML del controlador Java.

Agrega nuevos campos @FXML faltantes al controlador.

Marca con @Deprecated los métodos de eventos de acción faltantes.

Crea métodos de eventos de acción necesarios que no estén presentes en el controlador, basándose en el archivo FXML.

Esto agiliza el desarrollo y mantiene la coherencia entre la interfaz de usuario y el código.

### IntelliJ Code with Me

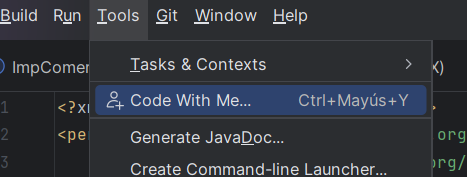


"Code With Me" es una herramienta de colaboración en tiempo real para IntelliJ IDEA que permite a múltiples desarrolladores trabajar juntos en el mismo proyecto a través de Internet. Facilita el trabajo en equipo a distancia ofreciendo:

* Configuración de permisos: Define el nivel de acceso de cada colaborador, desde solo lectura hasta acceso total.
* Edición simultánea: Permite a los usuarios editar código juntos, viendo los cambios en tiempo real.
* Seguimiento y navegación por el código: Sigue las acciones de los colaboradores o navega juntos por el código.
* Ejecución y depuración compartidas: Ejecuta y depura el código en sesiones compartidas como si estuvieras en tu entorno local.
* Herramientas de comunicación integradas: Soporte para llamadas de audio y vídeo, y compartir pantalla, directamente desde el IDE.
* Redirección de puertos y rica información de código: Accede a aplicaciones remotas y disfruta de funcionalidades completas del IDE como autocompletado.
* Seguridad para empresas: Posibilidad de desplegar sesiones en redes privadas para garantizar la seguridad y el cumplimiento.

Configuracion para su uso:

* Instalar Code With Me: Asegúrate de que tanto tú como tus colaboradores tengan instalado IntelliJ IDEA y el plugin "Code With Me".



* Iniciar una sesión: En IntelliJ IDEA, el anfitrión inicia una sesión de colaboración desde "Tools" > "Code With Me" > "Start Session".
* Invitar colaboradores: Una vez iniciada la sesión, el anfitrión obtiene un enlace de invitación que puede compartir con otros desarrolladores.
* Unirse a la sesión: Los colaboradores usan el enlace para unirse a la sesión de colaboración y comenzar a programar juntos.

Compatibilidad de IDE’s:



## Otros

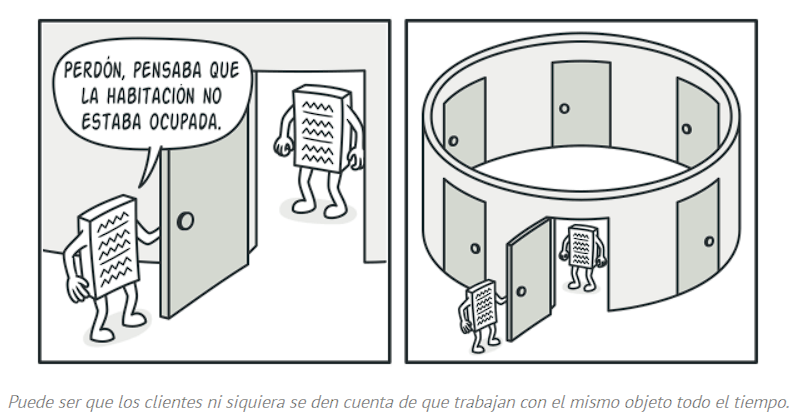
### Patrón ‘Singleton’

El Singleton es un patrón de diseño creacional que garantiza que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia. Esto resuelve dos problemas: controlar el acceso a un recurso compartido, como una base de datos o un archivo, y evitar que otros códigos sobreescriban la instancia.

**Problemas:**

* Garantizar una única instancia: El Singleton permite controlar cuántas instancias tiene una clase, útil para recursos compartidos. Esto se logra asegurando que la clase solo pueda ser instanciada una vez y devolviendo esa misma instancia en llamadas posteriores.
* Punto de acceso global: Similar a una variable global, el Singleton proporciona acceso a la instancia desde cualquier parte del programa. Sin embargo, evita que otros códigos puedan sobrescribir esa instancia, proporcionando seguridad.

Es importante tener en cuenta que el código que implementa el Singleton no debe estar disperso por todo el programa, sino encapsulado en la propia clase Singleton. Aunque el término "singleton" a menudo se usa para describir cualquier patrón que resuelva estos problemas, el Singleton específicamente aborda ambos problemas mencionados.



**Solución:**

* Hacer privado el constructor por defecto: Esto se hace para evitar que otros objetos utilicen el operador ‘new’ con la clase Singleton, lo que garantiza que no se puedan crear instancias adicionales de la clase desde fuera de la misma.
* Crear un método de creación estático: Se crea un método estático que actúa como constructor. Este método, generalmente llamado ‘getInstance()’ o similar, es responsable de crear una instancia única de la clase Singleton y almacenarla en un campo estático. En las siguientes llamadas a este método, se devuelve siempre la misma instancia almacenada en caché.

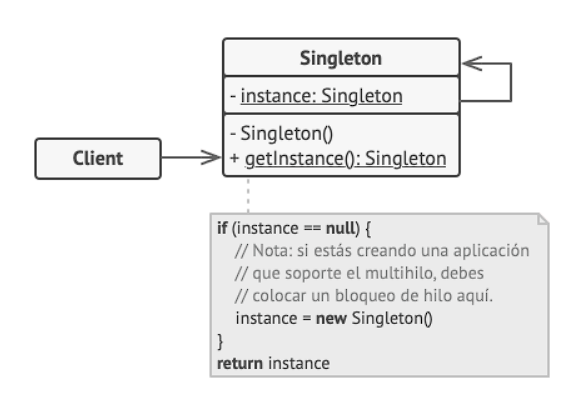
Este enfoque asegura que solo exista una instancia de la clase Singleton y que esta instancia sea accesible globalmente a través del método estático de creación.

Si el código tiene acceso a la clase Singleton, puede invocar el método estático para obtener siempre la misma instancia de la clase. Esto garantiza que no se creen múltiples instancias y que se mantenga el comportamiento de Singleton.

**Estructura:**

La clase Singleton declara el método estático ‘obtenerInstancia’ que devuelve la misma instancia de su propia clase.

El constructor del Singleton debe ocultarse del código cliente. La llamada al método ‘obtenerInstancia’ debe ser la única manera de obtener el objeto de Singleton.



**Aplicación:**

* Utilízalo cuando una clase en tu programa debe tener una única instancia disponible para todos los clientes, como un único objeto de base de datos compartido por distintas partes del programa.
* El patrón Singleton restringe la creación de objetos de la clase a través de métodos especiales de creación. Estos métodos pueden crear un nuevo objeto o devolver uno existente si ya ha sido creado.
* Emplea el patrón Singleton cuando necesites un control más estricto sobre las variables globales. A diferencia de las variables globales, el Singleton garantiza una única instancia de una clase, lo que previene la sustitución de la instancia en caché por otra.

Es importante destacar que puedes ajustar la limitación de una única instancia según sea necesario, permitiendo la creación de un número específico de instancias Singleton. Este ajuste solo requeriría modificaciones en el método ‘getInstance()’.

**Implementación:**

1. Añade un campo estático privado a la clase para almacenar la instancia Singleton.
2. Declara un método de creación estático público para obtener la instancia Singleton.
3. Implementa una inicialización diferida dentro del método estático. Debe crear un nuevo objeto en su primera llamada y colocarlo dentro del campo estático. El método deberá devolver siempre esa instancia en todas las llamadas siguientes.
4. Declara el constructor de clase como privado. El método estático de la clase seguirá siendo capaz de invocar al constructor, pero no a los otros objetos.
5. Repasa el código cliente y sustituye todas las llamadas directas al constructor de la instancia Singleton por llamadas a su método de creación estático.

**Pros y Contras:**

|  |  |
| --- | --- |
| PROS | CONTRAS |
| Puedes tener la certeza de que una clase tiene una única instancia. | Vulnera el Principio de responsabilidad única. El patrón resuelve dos problemas al mismo tiempo. |
| Obtienes un punto de acceso global a dicha instancia. | Puede enmascarar un mal diseño al fomentar dependencias entre componentes del programa. |
| El objeto Singleton solo se inicializa cuando se requiere por primera vez. | Requiere tratamiento especial en entornos con múltiples hilos para evitar problemas de concurrencia. |
|  | Realizar pruebas unitarias del código cliente del Singleton puede ser complicado debido a limitaciones en la creación de objetos simulados. |

### IntelliJ

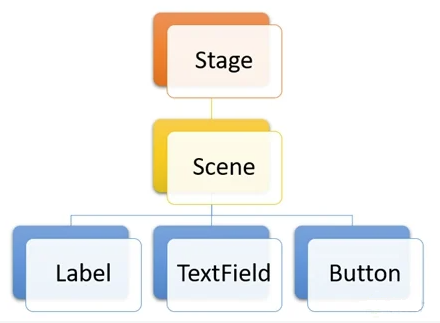
### SceneBuilder

JavaFX Scene Builder es una herramienta de diseño visual que permite a los usuarios diseñar rápidamente interfaces de usuario de aplicaciones JavaFX, sin necesidad de codificación. Los usuarios pueden arrastrar y soltar componentes de interfaz de usuario a un área de trabajo, modificar sus propiedades, aplicar hojas de estilo y el código FXML del diseño que están creando se genera automáticamente en segundo plano. El resultado es un archivo FXML que se puede combinar con un proyecto Java vinculando la interfaz de usuario a la lógica de la aplicación.

La aplicación Scene Builder permite diseñar, mediante un interfaz gráfico, las estructuras de las ventanas de las aplicaciones que queramos desarrollar usando JavaFX. En este artículo podrás conocer los fundamentos básicos para empezar a usar esta herramienta de manera integrada con el entorno de desarrollo NetBeans.

Desde una perspectiva de Model View Controller (MVC):

* El archivo FXML, que contiene la descripción de la interfaz de usuario, es la vista.
* El controlador es una clase Java, que implementa opcionalmente la clase Initializable, que se declara como el controlador para el archivo FXML.
* El modelo consta de objetos de dominio, definidos en el lado de Java, que se pueden conectar a la vista a través del controlador.



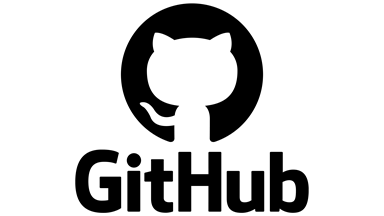
Ver Anexo X ‘Como configurar en Intellij IDEA el SceneBuilder’.

Para ampliar sobre SceneBuilder:

<https://code.makery.ch/es/library/javafx-tutorial/>

<https://www.jetbrains.com/help/idea/opening-fxml-files-in-javafx-scene-builder.html#open-in-scene-builder>

### GitHub



Hemos utilizado la herramienta de repositorios GitHub para compaginar el trabajo en equipo de manera correcta. Esta herramienta permite a los desarrolladores trabajar juntos en proyectos desde cualquier lugar.

Entre las ventajas principales de utilizar GitHub destaca:

* El control de versiones mediante Git.
* La facilidad de gestión de proyectos.
* Seguridad. Utiliza la verificación en dos pasos.

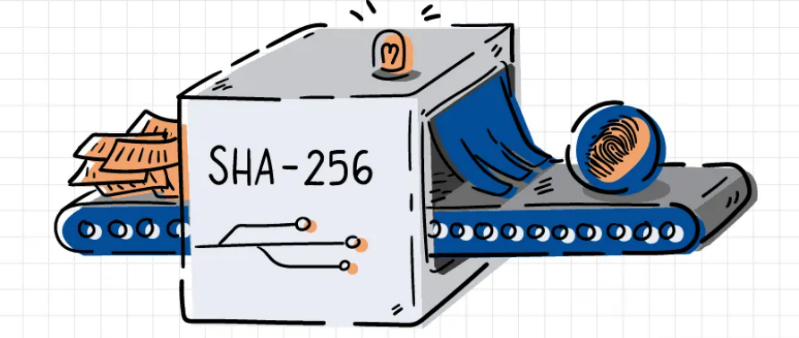
Ver Anexo X ‘Como preparar GitHub para un trabajo grupal’.

## Seguridad

### Seguridad de datos

**SHA256**

SHA256 se deriva de las palabras ***"Secure Hash Algorithm 256 bits"***, lo que significa que es un algoritmo de hash seguro de 256 bits.



Su tarea principal es tomar una cantidad arbitraria de información, ya sea un mensaje de texto, una contraseña o incluso un archivo completo, y convertirla en una secuencia de 256 bits, un conjunto de números y letras aparentemente aleatorios.

Esta secuencia conocida como ***"hash"***, es única para cada conjunto de datos que se le pasa al algoritmo, lo que significa que, si se cambia una sola letra en el mensaje original, el hash resultante será completamente diferente. Es como la ***"huella digital"*** de los datos, y es imposible que dos conjuntos de datos diferentes produzcan el mismo hash.

**¿Cómo funciona el algoritmo SHA256?**[**​**](https://cuevaneander.tech/blog/seguridad-informatica-algoritmo-sha256/#c%C3%B3mo-funciona-el-algoritmo-sha256)

Ahora que ya tenemos un poco más de contexto general sobre lo que es este algoritmo, es hora de profundizar un poco más en cómo funciona. Como se mencionó anteriormente, el algoritmo SHA256 toma una cantidad arbitraria de datos y los convierte en una secuencia de 256 bits. Pero, ¿cómo lo hace exactamente? ¿Cómo puede tomar cualquier cantidad de datos y producir un hash único para cada uno de ellos?

**Mensaje “M**[**​**](https://cuevaneander.tech/blog/seguridad-informatica-algoritmo-sha256/#mensaje-m)”

El mensaje "M" o "datos de entrada" es la información que la función SHA256 procesará para calcular el hash (también conocido como "digest"). M puede tener una longitud variable, lo que significa que puede consistir en un archivo de datos o una cadena de texto, como por ejemplo "Hola mundo", incluso puede ser una cadena de texto vacía.

El resultado de este proceso es una cadena ω de 256 bits de longitud, que se expresa en un formato de 64 caracteres hexadecimales(base16). Este proceso de transformación garantiza que, independientemente de la forma o contenido de la entrada, obtendremos un hash de 2556 bits que actúa como una huella digital única de los datos de entrada.

La función SHA256 se puede representar como SHA256(M) = ω, donde M es el mensaje de entrada y ω es el hash de salida.



Aplicando esa función a una cadena de texto como "Hola mundo" obtendríamos el siguiente resultado:

**"Hola mundo" | ca8f60b2cc7f05837d98b208b57fb6481553fc5f1219d59618fd025002a66f5c**

La cadena hexadecimal generada se obtiene mediante una serie de cálculos en los que se emplean entre otros datos los bits de entrada, es decir, para un mismo mensaje M siempre se obtendrá el mismo hash ω. Por el contrario, si modifica un solo bit, ya sea cambiando o agregando un caracter (espacios y saltos de línea incluidos), el hash resultante será completamente diferente.

echo-n"abc"|sha256sum  
ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad

echo -n "Abc" | sha256sum  
06d90109c8cce34ec0c776950465421e176f08b831a938b3c6e76cb7bee8790b

De este modo, se garantiza la integridad de los datos. Si alguien modifica el mensaje, el hash resultante será diferente, por lo que no coincidirá con el hash original. Esto permite detectar si los datos han sido modificados.

# Webgrafía

SceneBuilder

<https://platzi.com/tutoriales/1631-java-basico/9075-javafx-scene-builder/>

<https://code.makery.ch/es/library/javafx-tutorial/>

FXML manager

<https://plugins.jetbrains.com/plugin/14854-fxmlmanager>

JPA Buddy

<https://www.jetbrains.com/help/idea/jpa-buddy.html#e9183a5d>Code With me:

<https://www.jetbrains.com/es-es/code-with-me/>

# Anexos

## Anexo 1

## Anexo 2

## Anexo 3

## Anexo 4

## Anexo 5