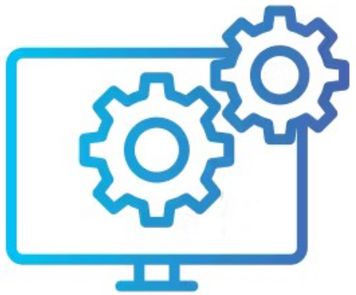
Persistencia de los datos

¿Que veremos hoy?

1. Software e información
2. Persistencia de datos
3. Almacenamiento de datos

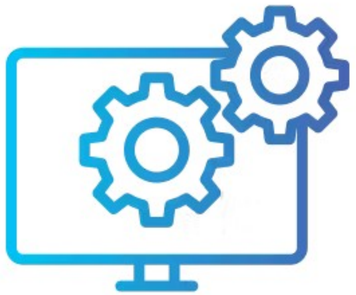


¿Cómo funciona el software o programa?

1. Software e información

> Un software carga información, la procesa y muestra un resultado.

- **¿Informática?** disciplina que estudia el tratamiento automático de la información mediante computadoras a través de software.
- **¿Ingeniera software?** una profesional espacializada en el diseño, desarrollo, mantenimiento y gestión de sistemas software.



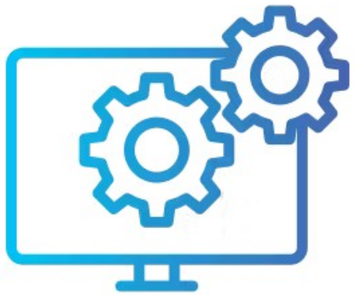
Ejemplo

La tabuladora

Una de las primeras máquinas de aplicación en informatica es la tabuladora

> Sistema de tarjetas perforadas eléctricas y basado en la lógica de Boole

Hollerith ideó un sistema mediante el cual la presencia de un agujero en una tarjeta implicaba el contacto entre dos cabezales de mercurio, de manera que había corriente eléctrica. Esto hacía que la **tabuladora** registrase la información, al impulsarse el contador.



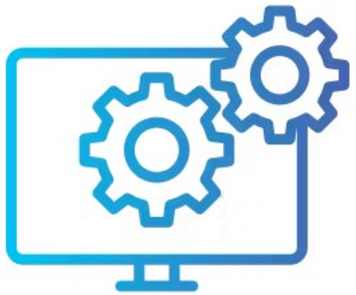
La tabuladora

Es decir ...

Tengo información externa al programa (tarjeta perforada) que cargo en el programa para procesar esa información.

En este caso:

- Almacenamiento primario: memoria en usa la tabuladora para procesar los datos.
- Almacenamiento secundario: tarjetas perforadas donde almaceno los datos cuando el programa no lo tiene en uso.



En la actualidad

- Almacenamiento primario: es la memoria principal o interna de un sistema informático, encargada de almacenar temporalmente los datos y las instrucciones que el procesador necesita para realizar sus operaciones.
  - RAM, Cache, ROM, etc
    - **Es volátil** y su contenido se pierde cuando se apaga el dispositivo
- Almacenamiento secundario: se refiere a los dispositivos y medios que almacenan datos de forma **permanente o a largo plazo.**
  - HDD, SSD, USB, CD, DVD, Nube, Cintas Magnéticas, etc
    - **Es persistente** y su contenido no se pierde cuando se apaga el dispostivo.

Pregunta ...

¿Por qué el almacenamiento primario es volátil y el almacenamiento secundario no?

¿Por qué el almacenamiento primario es volátil y el almacenamiento secundario no?

No hay forma física de hacer un almacenamiento secundario lo suficientemente rápido para ser a su vez almacenamiento primario.

Primario => Velocidad y Acceso temporal
Secundario => Mantener la información permamente

2. Persistencia de datos

Cómo diseñar software teniendo en cuenta dos restricciones:

- Los programas trabaja con datos de la memoria principal.
- Un programa puede crear datos en memoria principal y guardarlos en memoria secundaria

Este apartado resuelve la pregunta: "En la arquitectura software, ¿Dónde pongo la persistencia de los datos?"

Arquitectura monolítica

La **arquitectura monolítica** es un enfoque de diseño de software en el que todos los componentes y funcionalidades de una aplicación se agrupan en una sola unidad o código base.
Esto significa que la lógica de negocio, la interfaz de usuario y el acceso a datos están integrados en un único programa.

- Es fácil de desarrollar y desplegar en etapas iniciales.
- Complejo de mantener y escalar a medida que crece la aplicación.
- La falta de modularidad también dificulta la implementación de cambios o la adopción de nuevas tecnologías.

Arquitectura microservicios

La **arquitectura de microservicios** divide una aplicación en servicios pequeños e independientes que se ejecutan de forma autónoma, cada uno con su propia lógica de negocio y acceso a datos.
Cada microservicio se centra en una funcionalidad específica y se comunica con otros a través de API o mensajes.

- Esta arquitectura facilita la escalabilidad, la actualización y el despliegue de cada servicio de forma independiente.
- Permite utilizar diferentes tecnologías para distintos microservicios según sus necesidades.
- Introduce problemas de gestión y comunicación entre los servicios.

Arquitectura event-driven

La **arquitectura Event-Driven** (basada en eventos) se centra en la comunicación asíncrona entre componentes a través de eventos.
Los componentes se dividen en **emisores** que generan eventos y **receptores** que reaccionan a esos eventos, permitiendo un alto grado de independencia entre ellos.

- Esta arquitectura es ideal para aplicaciones en tiempo real, ya que mejora la capacidad de respuesta y la escalabilidad del sistema.
- Los eventos pueden ser manejados mediante colas de mensajes o sistemas de publicación-suscripción.
- Es más compleja de implementar y depurar debido a la naturaleza distribuida y asincrónica de los eventos.

3. Almacenamiento de datos

1. Ficheros
2. Bases de datos Relacionales
3. Base de datos No relacionales
4. Data Lakes
5. Flujos de datos

3.1 Ficheros

El **almacenamiento de datos en ficheros** es una técnica que implica guardar información en archivos de forma estructurada o no estructurada en un sistema de archivos.

Este método es uno de los más simples y directos para almacenar datos, y se utiliza comúnmente en diversas aplicaciones, desde simples programas hasta sistemas más complejos.

3.1 Ficheros

- Más lento en comparación con BD
- Ideales cuando no hay concurrencia
- Dificultad en grandes volúmenes
- Falta de integridad refencial

3.1.1 Ficheros no estructurados

Los **ficheros no estructurados** son aquellos que no tienen un formato predefinido, lo que dificulta su análisis y procesamiento.
Estos ficheros pueden ser documentos de texto, imágenes, videos, ... que contienen información valiosa pero carecen de un esquema claro.
Para acceder y extraer datos de ficheros no estructurados, se requieren herramientas y técnicas avanzadas, como el procesamiento de lenguaje natural (NLP) y el análisis de imágenes.

3.1.2 Ficheros estructurados

Los **ficheros estructurados** son aquellos que organizan datos en un formato predefinido y accesible, lo que facilita su almacenamiento, búsqueda y análisis.

- CSV
- JSON
- XML

3.1.2.1 CSV

El **CSV (Comma-Separated Values)** es un formato de archivo simple y ampliamente utilizado para el almacenamiento de datos tabulares.

En un archivo CSV, cada línea representa un registro, y los campos dentro de cada registro están separados por comas, aunque también se pueden usar otros delimitadores como punto y coma.

3.1.2.1 CSV

Este formato es fácil de crear y leer, lo que lo convierte en una opción popular para la importación y exportación de datos entre aplicaciones, especialmente en hojas de cálculo como **Microsoft Excel** y **Google Sheets**.

Sin embargo, no admite tipos de datos complejos ni estructuras jerárquicas, lo que puede dificultar la representación de información más elaborada.

También carece de metadatos que definan la estructura del contenido, lo que puede llevar a confusiones si no se documenta adecuadamente.

3.1.2.2 JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos ligero y fácil de leer, que se utiliza ampliamente en aplicaciones web y APIs.

Su estructura se basa en pares de clave-valor, organizando los datos en objetos que se delimitan con llaves `{}` y arrays que se indican con corchetes `[]`.

Esta simplicidad hace que JSON sea intuitivo y accesible tanto para humanos como para máquinas, facilitando la manipulación de datos.

3.1.2.2 JSON

Una de las ventajas de JSON es su capacidad para representar estructuras complejas y anidadas, lo que lo hace ideal para el almacenamiento de datos jerárquicos.

Sin embargo, a diferencia de XML, JSON no admite comentarios, lo que puede ser una limitación en ciertos contextos. A pesar de esto, su ligereza y facilidad de uso lo han llevado a convertirse en un estándar en el desarrollo de aplicaciones y servicios web.

3.1.2.3 XML

XML (eXtensible Markup Language) es un formato de marcado que permite la representación de datos de manera estructurada y jerárquica.

Utiliza etiquetas para encapsular datos, proporcionando un contexto claro para cada elemento. Esta flexibilidad permite que XML sea utilizado en una amplia variedad de aplicaciones, desde la configuración de software hasta el intercambio de datos entre sistemas y servicios web.

3.1.2.3 XML

Una de las características clave de XML es su capacidad para validar la estructura de los datos mediante esquemas como **DTD (Document Type Definition)** o **XML Schema**, lo que garantiza que los datos cumplan con ciertos requisitos antes de ser procesados.

Esto resulta especialmente útil en entornos empresariales donde la integridad de los datos es crucial. XML también es independiente del lenguaje, lo que significa que puede ser utilizado en diferentes plataformas y tecnologías.

Sin embargo, puede resultar en archivos más pesados y difíciles de leer en comparación a JSON.

zZzZzz

3.2 Bases de datos Relacionales

Las **bases de datos relacionales** son sistemas de gestión de datos que organizan la información en tablas, donde cada tabla consta de filas y columnas.

Cada fila representa un registro único, mientras que cada columna contiene un atributo específico de ese registro.

Este enfoque permite establecer relaciones entre diferentes tablas a través de **claves primarias** y **claves ajena**, lo que facilita la integridad y consistencia de los datos.

3.2 Bases de datos Relacionales

Utilizan **SQL (Structured Query Language)** para realizar operaciones de creación, lectura, actualización y eliminación (CRUD), lo que permite a los desarrolladores interactuar con los datos de manera efectiva.

3.2 Bases de datos Relacionales

Un **ORM (Object-Relational Mapping)** es una técnica de programación que permite convertir datos entre sistemas incompatibles (Objeto y BD) usando lenguajes de programación orientados a objetos.

Un ORM actúa como un intermediario que mapea las tablas de la base de datos a objetos en el código. Esto significa que los desarrolladores pueden interactuar con la base de datos utilizando objetos y métodos en lugar de escribir consultas SQL manualmente, lo que mejora la productividad y la legibilidad del código.

3.2 Bases de datos Relacionales

Eloquent es el **ORM** que se utiliza en Laravel y facilita la interacción con las bases de datos relacionales.

Por ejemplo:

- obtener todos los usuarios y sus publicaciones asociadas

Eloquent:

```
$usuarios = User::with('posts')->get();
```

SQL:

```
SELECT users.*, posts.* FROM users LEFT JOIN posts ON users.id = posts.user_id;
```

¿Cuál de las dos opciones anteriores es mejor?

"Cualquier tonto puede escribir código que una computadora entienda. Los buenos programadores escriben código que los humanos puedan entender."

– Martin Fowler

3.3 Bases de datos NO relacionales

Las **bases de datos no relacionales**, también conocidas como **NoSQL**, son sistemas de gestión de datos que no utilizan el modelo tabular de las bases de datos relacionales.

Se diseñaron para manejar grandes volúmenes de datos, alta velocidad de escritura y lectura, y la necesidad de escalabilidad horizontal.

- documentos (mongoDB)
- claves-valor (Redis)
- columnas (Apache Cassandra)
- grafos (Neo4j)

3.3 Bases de datos NO relacionales

Se pueden utilizar herramientas y enfoques similares a los ORM (Object-Relational Mapping) para trabajar con bases de datos NoSQL, aunque estos generalmente se denominan **ODM (Object-Document Mapping)** en el contexto de bases de datos orientadas a documentos.

3.3 Bases de datos NO Relacionales
Por ejemplo:

- obtener todos los usuarios y sus publicaciones asociadas

Eloquent ORM:

```
$usuarios = User::with('posts')->get();
```

jenssegers/mongodb ODM:

```
db.users.aggregate([{
  $lookup: {
   from: "posts",
   localField: "_id",
   foreignField: "user_id",
   as: "posts"
 }}]);
```

3.4 Data Lake

Un **data lake** es un repositorio de almacenamiento que permite guardar grandes volúmenes de datos en su formato nativo, tanto estructurados como no estructurados.

A diferencia de un data warehouse, que requiere un proceso de transformación y modelado de datos antes de su almacenamiento, un data lake permite la ingesta de datos en tiempo real, lo que facilita la recopilación de información desde diversas fuentes, como sensores, registros de eventos, redes sociales y bases de datos.

Apache Spark o Hadoop

3.5 Flujos de datos

El **flujo de datos** se refiere a la transmisión continua de información desde diversas fuentes hacia destinos en tiempo real, permitiendo el procesamiento y análisis instantáneo.

Este flujo se puede manejar a través de **eventos** y **colas**, donde los eventos representan cambios o acciones en el sistema (como una actualización de datos o una transacción) y las colas actúan como intermediarios que almacenan estos eventos temporalmente hasta que puedan ser procesados.

3.5 Flujos de datos

El **flujo de datos** se refiere a la transmisión continua de información desde diversas fuentes hacia destinos en tiempo real, permitiendo el procesamiento y análisis instantáneo.

Este flujo se puede manejar a través de **eventos** y **colas**, donde los eventos representan cambios o acciones en el sistema (como una actualización de datos o una transacción) y las colas actúan como intermediarios que almacenan estos eventos temporalmente hasta que puedan ser procesados.

- Laravel WebSockets
- Apache Kafka

Mentimeter

¿Dudas?