MODULO M09: PROGRAMACIÓN DE SERVICIOS Y PROCESOS

Profesor: Jorge Juan Delgado

ILERNA Online

UNIDAD FORMATIVA 2

PROCESOS E HILOS

ILERNA Online

¿Qué es un Programa?

¿Qué es un Proceso?

¿Qué es un Ejecutable?

ILERNA Online

¿Qué es un Programa?

Toda la información (**código + datos**) que están almacenados en memoria segundaria y que dan solución a una necesidad del usuario que lo ha ejecutado

ILERNA Online

¿Qué es un Proceso?

Es un trozo de programa en ejecución

No solo son datos y códigos, además todo lo necesario para la ejecución

Contador de programa (en qué paso está la ejecución) Es un registro que almacena la posición de memoria siguiendo a qué instrucción le toca el turno para que la CPU sepa en todo momento qué instrucción es la que tiene que ejecutar.

Imagen de memoria (espacio de memoria usada por proceso) Porción de memoria que está utilizando un proceso.

Estado del proceso (valor de registros del procesador en ejecución donde se queda cuando la CPU le quitó el control para luego poder continuar donde se quedó).

ILERNA Online

¿Qué es un Ejecutable?

Fichero que contiene toda la información necesaria para crear un proceso a partir de los datos almacenados de un programa, esto es, **fichero que permite poner el programa en ejecución como proceso**

El programa pasa a ser un proceso a través de un ejecutable.

ILERNA Online

¿Qué es el Sistema Operativo?

Programa que hace de intermediario entre el usuario y las aplicaciones que usa, así como con el hardware del ordenador

Ejecuta programas del usuario, hace de interfaz entre usuario y hardware y usa los recursos del ordenador de forma eficiente

ILERNA Online

¿Qué es un proceso Daemon (Demonio)?

Proceso no interactivo que se ejecuta continuamente en segundo plano

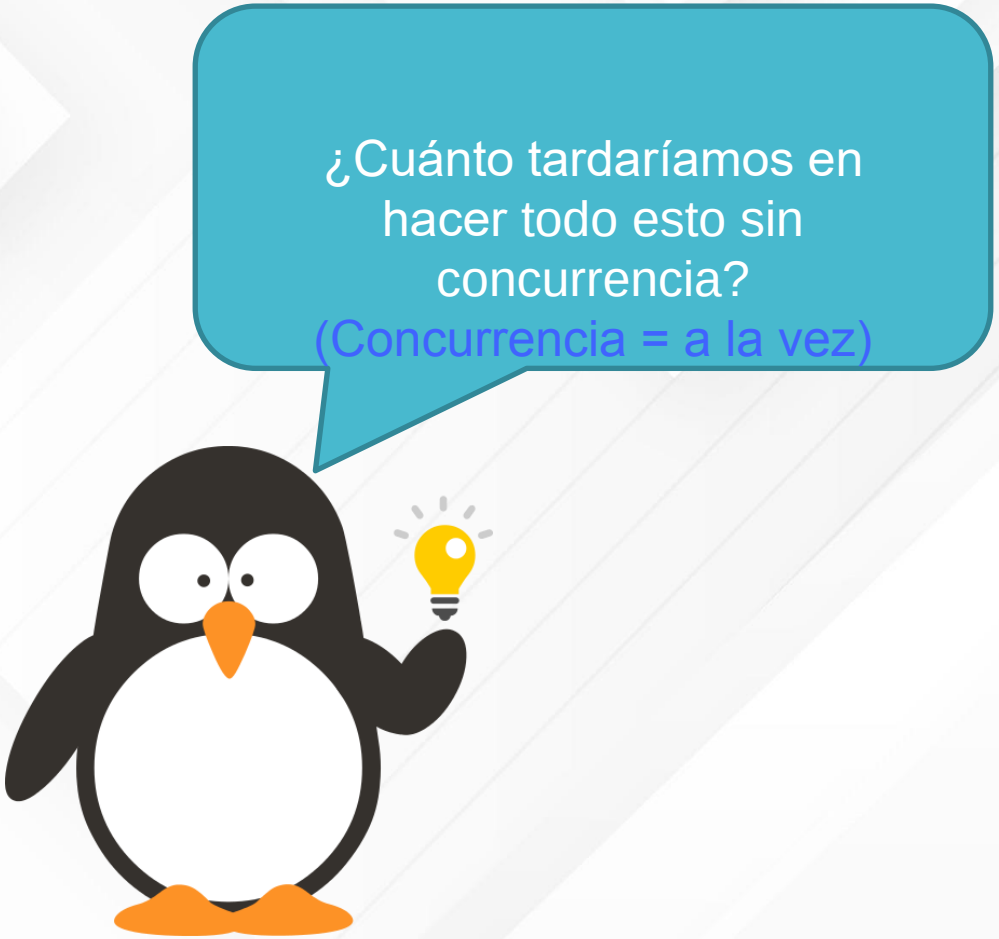
Son controlados por el Sistema Operativo y el usuario no puede intervenir

Proporcionan un servicio básico para el resto de procesos

Como el de una impresora wi-fi cuando recibe una petición para imprimir, se activa, recibe el documento e imprime. Está siempre a la espera de ser activada para imprimir.

ILERNA Online

- La programación concurrente permite tener en ejecución varias tareas interactivas al mismo tiempo
  - Escuchar música
  - Imprimir un documento
  - Ver un vídeo en Youtube
  - Descargar un fichero
  - Tener abierta une ventana de Chrome con 5 pestañas
  - Recibir notificaciones de actualizaciones



- Las tareas se pueden ejecutar de tres formas diferentes:

Único Procesador	Varios núcleos	Varios ordenadores en Red
Sólo un proceso en ejecución en un momento determinado, asignando orden y cambiando proceso cada poco tiempo. El usuario piensa que todos se ejecutan a la vez	Cada núcleo ejecuta un proceso al mismo tiempo. El SO planifica los trabajos por ejecutar en cada núcleo. Todos los núcleos comparten la memoria	Cada uno con sus propios procesadores y propia memoria Gestión de ordenadores en paralelo La memoria no se comparte, por lo que la comunicación requiere métodos más costosos a través de la red
Programación Concurrente	Programación Paralela	Programación Distribuída

Programación distribuida: Los terminales tontos son ordenadores sin disco duro para cargar un S.O, lo cargan del servidor. Ej: Universidades, puedes loguearte en cualquier pc, lo haces desde un servidor.

Procesador

Unidad central de proceso

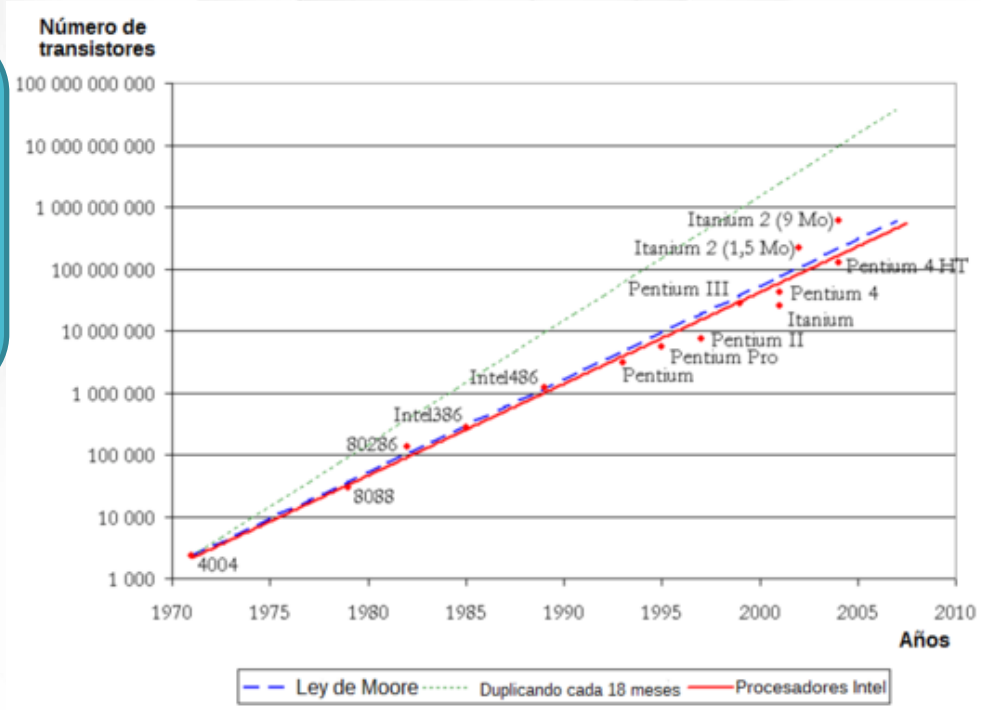
Interpreta instrucciones básicas:

- Aritméticas
- Lógicas
- Entrada/Salida

Procesador

Unidad central de proceso

Ley de Moore: la velocidad del procesador o el poder de procesamiento total de las computadoras se duplica cada doce meses



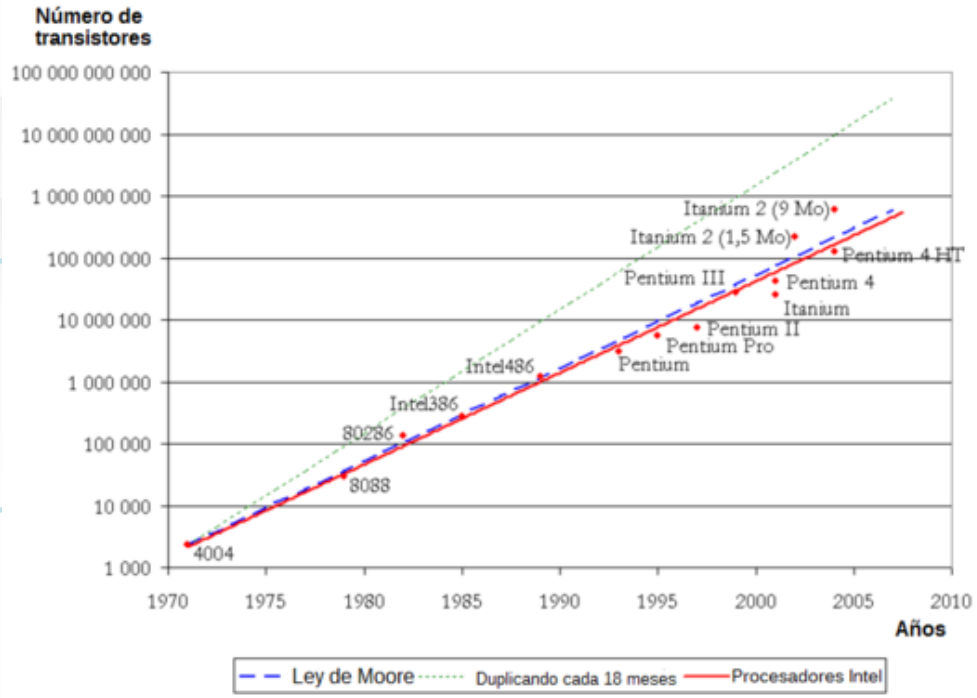
ILERNA Online

Procesador

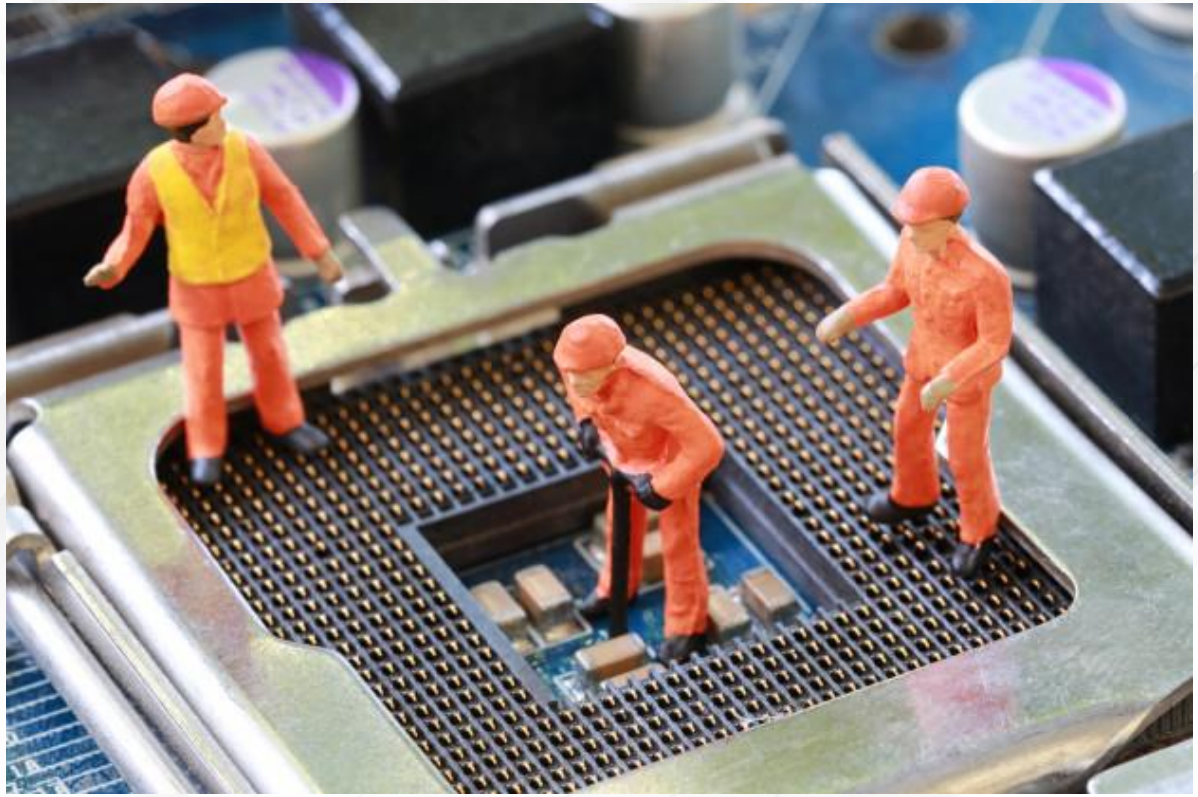
Unidad central de proceso

Interpreta instrucciones básicas:

Un programa: sin cambiar ninguna línea de código, cada año = el doble de rápido




Tiempo Ocioso: la CPU está esperando a que ocurra un evento para seguir trabajando

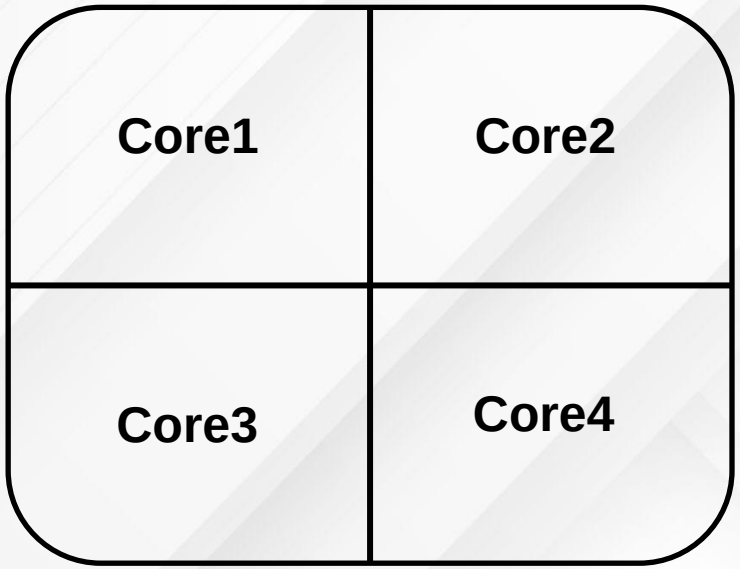


SOLUCIÓN: Repartir los procesos en hilos en segundo plano

> Cores de un procesador

Los cores (núcleos) son divisiones dentro de un procesador para poder repartir la carga que se le administra al procesador.

Cada uno de los cores funciona como una unidad independiente de proceso.



Ejemplo

Procesador Intel i7 6500U

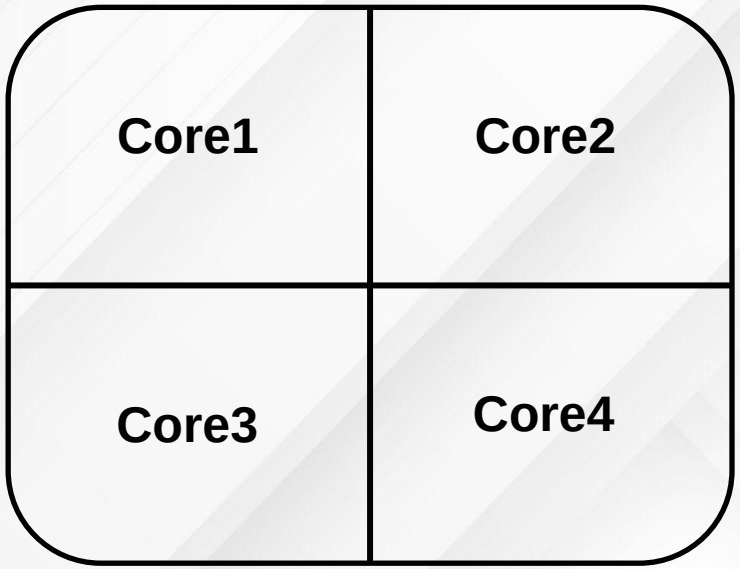
Cantidad de núcleos: 4

Cantidad de subprocesos: 4

Subproceso = hilo

Los cores (núcleos) son divisiones dentro de un procesador para poder repartir la carga que se le administra al procesador.

Cada uno de los cores funciona como una unidad independiente de proceso.

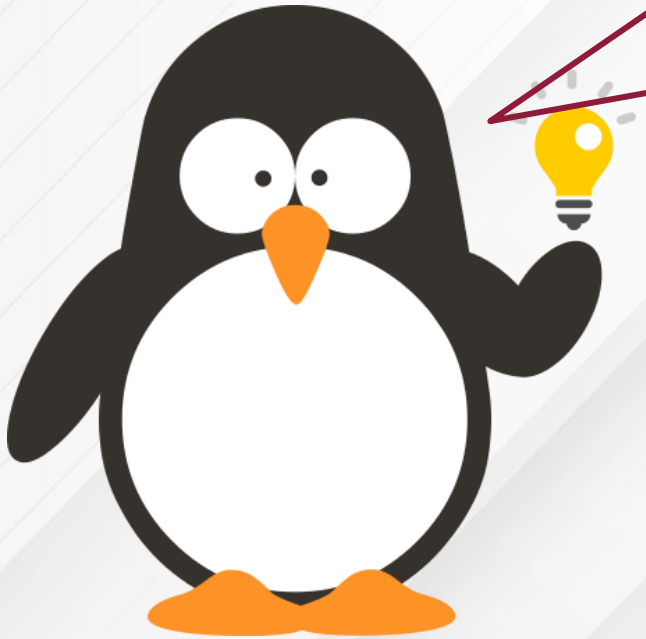


- Cada core = 4 procesos concurrentes
- Cuando un subproceso solicita E/S (más lentas)
  - Expulsa el proceso que se está ejecutando BLOQUEADO
  - Carga uno nuevo
  - Al recibir la respuesta el proceso que lo estaba esperando, vuelve a LISTO

> Ventajas de la programación concurrente

ILERNA Online

- ¿Cuánto tiempo puede tardar este algoritmo ?
- ¿Qué hace el procesador mientras?
- ¿Espera? ¿Ejecuta otro proceso mientras?
- ¡CONCURRENCIA!



```
1
2  public class Bucle_casi_infinito {
3      public static void main(String[] args) {
4
5          System.out.println("INICIO DEL PROGRAMA");
6
7          for (int i = 1; i< 999999999; i++)
8              System.out.println("Bienvenido Usuario #" + i);
9
10         System.out.println("FIN DEL PROGRAMA");
11     }
12
13 }
14
```

ILERNA Online

La concurrencia es la propiedad que permite a los sistemas ejecutar diferentes procesos en un mismo tiempo.

Una aplicación concurrente no garantiza menor tiempo

P1 → Proceso 1
P2 → Proceso 2
P3 → Proceso 3

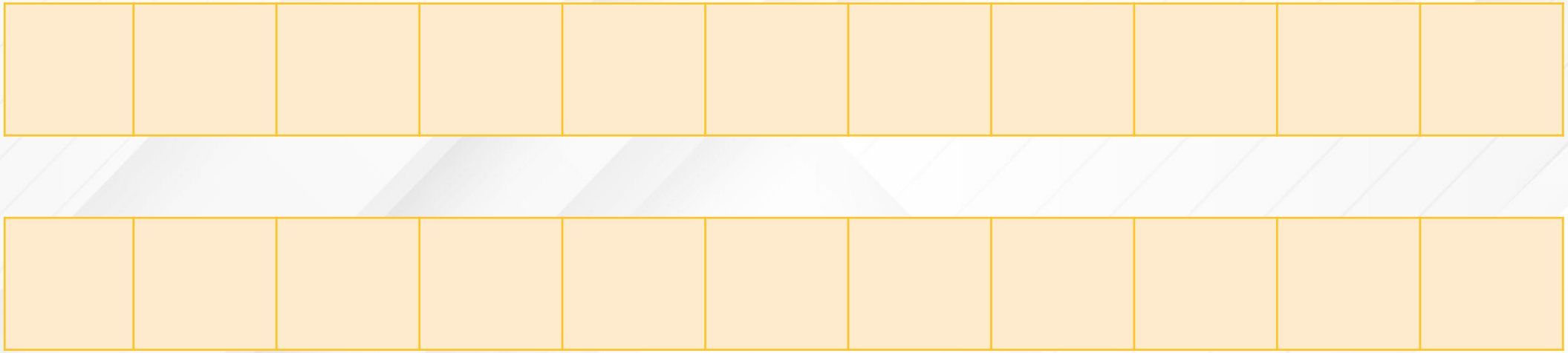
La programación concurrente es la que se encarga de ejecutar las tareas simultáneamente.

ILERNA Online

EJECUCIÓN NO CONCURRENTE

Vamos a ver con un ejemplo cómo el procesador gestiona los tiempos de CPU

P1 = 3 operaciones(1udt) + 1 operación E/S (3udt) + 2 operaciones (1udt)
P2 = 2 operaciones (1udt) + 1 operación E/S (3udt) + 2 operaciones (1udt)
P3 = 2 operaciones (1udt)



ILERNA Online

EJECUCIÓN CONCURRENTE

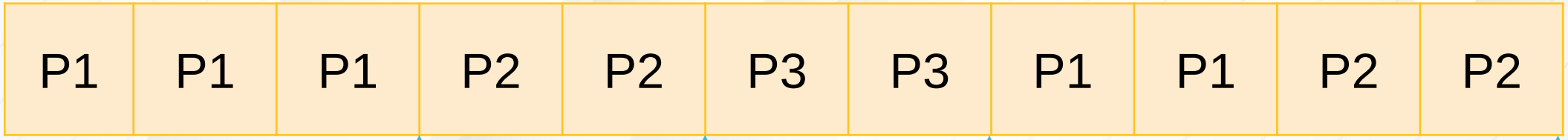
Vamos a ver con un ejemplo cómo el procesador gestiona los tiempos de CPU

P1 = 3 operaciones(1udt) + 1 operación E/S (3udt) + 2 operaciones (1udt)
P2 = 2 operaciones (1udt) + 1 operación E/S (3udt) + 2 operaciones (1udt)
P3 = 2 operaciones (1udt)

EJECUCIÓN CONCURRENTE

Vamos a ver con un ejemplo cómo el procesador gestiona los tiempos de CPU

P1 = 3 operaciones(1udt) + 1 operación E/S (3udt) + 2 operaciones (1udt)
P2 = 2 operaciones (1udt) + 1 operación E/S (3udt) + 2 operaciones (1udt)
P3 = 2 operaciones (1udt)



P1 sale
Entra P2

P2 sale
P1 esperando o bloqueado
Entra P3

Bloqueado porqué necesita 3 unidades de tiempo para estar listo.

P3 sale y FIN
P2 esperando
Entra P1

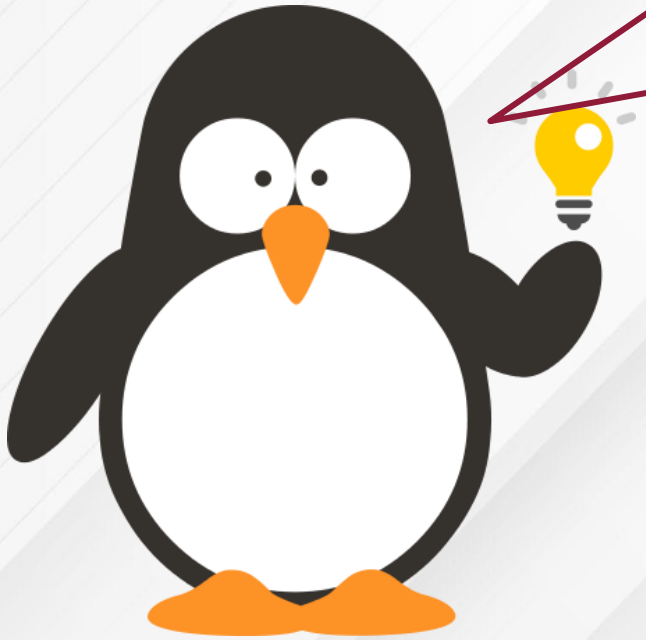
P1 sale y FIN
P2 entra

P2 sale y FIN

> Ventajas de la programación concurrente

ILERNA Online

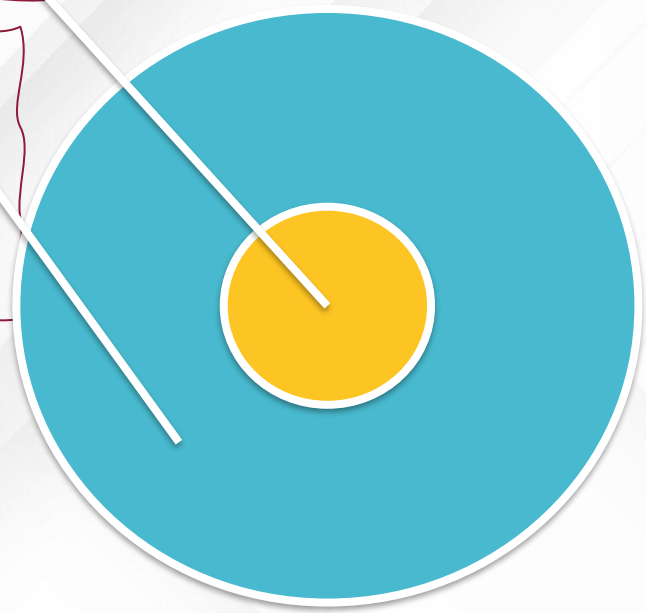
- Posibilita la compartición de recursos.
- Facilita la programación de aplicaciones en tiempo real.
- Puede reducir los tiempos de ejecución.



> Programa, Proceso e hilo

- **Un hilo** es un flujo de control independiente dentro de un programa
  - Secuencia de instrucciones que se ejecutan

- **Un proceso** es un programa en ejecución
  - Gestiona recursos necesarios para ejecutar el programa
  - Contiene uno o varios hilos de ejecución

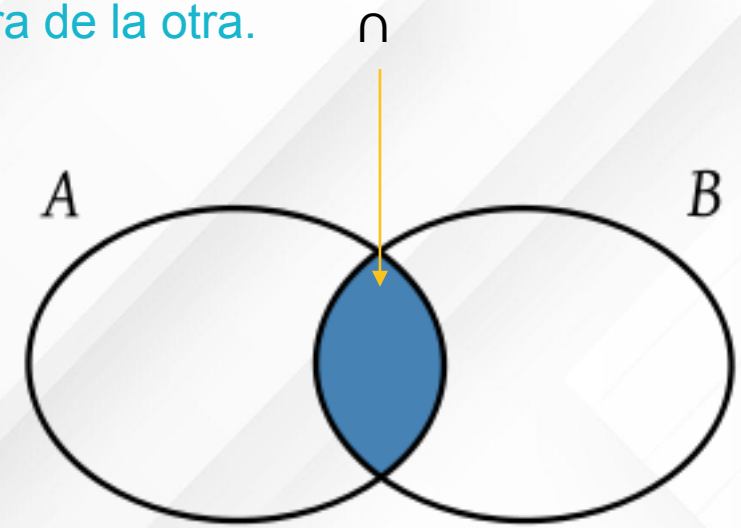


```java
public class Ejemplo1 {
    /**
     * Método principal de la clase Ejemplo1
     * @param args Argumentos de entrada
     */
    public static void main(String[] args){
        //Creación de nueva variable
        String txt;
        //Asignar valor a la variable
        txt = "Bienvenidos Onliners!!!! :)";
        //Sacar el resultado por consola.
        System.out.println(txt);
    }

}
```

Para poder ejecutar aplicaciones de forma concurrente se deben dar 3 condiciones:

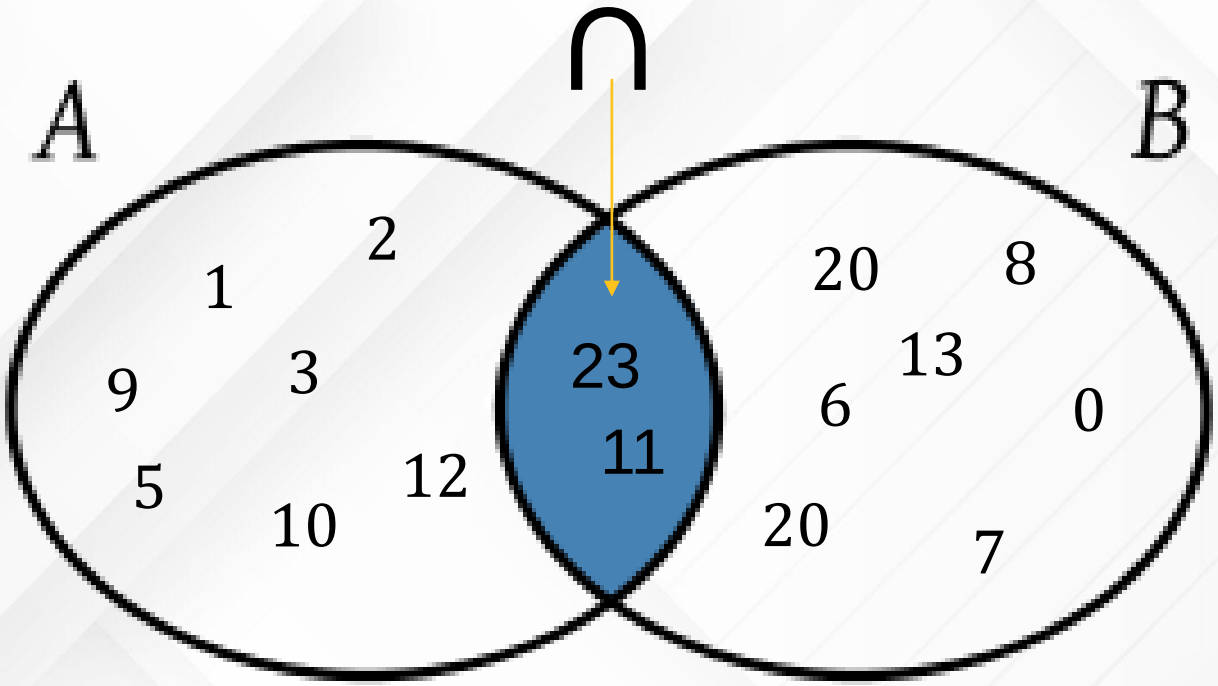
1. $L(S_i) \cap E(S_j) = \emptyset$
2. $E(S_i) \cap L(S_j) = \emptyset$
3. $E(S_i) \cap E(S_j) = \emptyset$

Que no haya elementos comunes entre la lectura o escritura de la una y la escritura o escritura de la otra.



Para poder ejecutar aplicaciones de forma concurrente se deben dar 3 condiciones:

1. $L(S_i) \cap E(S_j) = \emptyset$
2. $E(S_i) \cap L(S_j) = \emptyset$
3. $E(S_i) \cap E(S_j) = \emptyset$



ILERNA Online

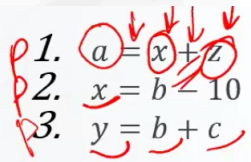
Ejercicio practico:

1. $a = x + z$
2. $x = b - 10$
3. $y = b + c$

Lecturas
1 – {x,z} - 2 lecturas
2 – {b} - 1 lectura
3 – {b, c} - 2 lecturas

Escrituras
1 – {a} – 1 escritura
2 – {x} - 1 escritura
3 – {y} - 1 escritura

P1 ∩ P2 = ∅ NO
P2 ∩ P3 = ∅ SI
P1 ∩ P3 = ∅ SI
L(P1) ∩ E(P3) = x,z ∩ y ok
L(P3) ∩ E(P1) = b,c ∩ a ok
E(P3) ∩ E(P1) = y ∩ a ok



Lecturas
1 – {x,z} - 2 lecturas
2 – {b} - 1 lectura
3 – {b, c} - 2 lecturas

Escrituras
1 – {a} – 1 escritura
2 – {x} - 1 escritura
3 – {y} - 1 escritura

Con p1 y p2 una de las 3 condiciones no se cumple, no se pueden ejecutar de forma concurrente. p2 con p3 si podrá, y p1 con p3 también.

$L(I_1) \cap E(I_2) \neq \emptyset$ No se cumplen, tienen en común la x
$L(I_1) \cap E(I_2) = \emptyset$ Se cumple, no tienen en común ninguna.
$E(P3) \cap L(P1)$ Se cumple.

$$L(S_i) \cap E(S_j) = \emptyset$$
$$E(S_i) \cap L(S_j) = \emptyset$$
$$E(S_i) \cap E(S_j) = \emptyset$$

Para poder ejecutar aplicaciones de forma concurrente se deben dar 3 condiciones:

1. $a = x + b$
2. $b = t + x$
3. $d = c + b$

Lecturas
1
2
3

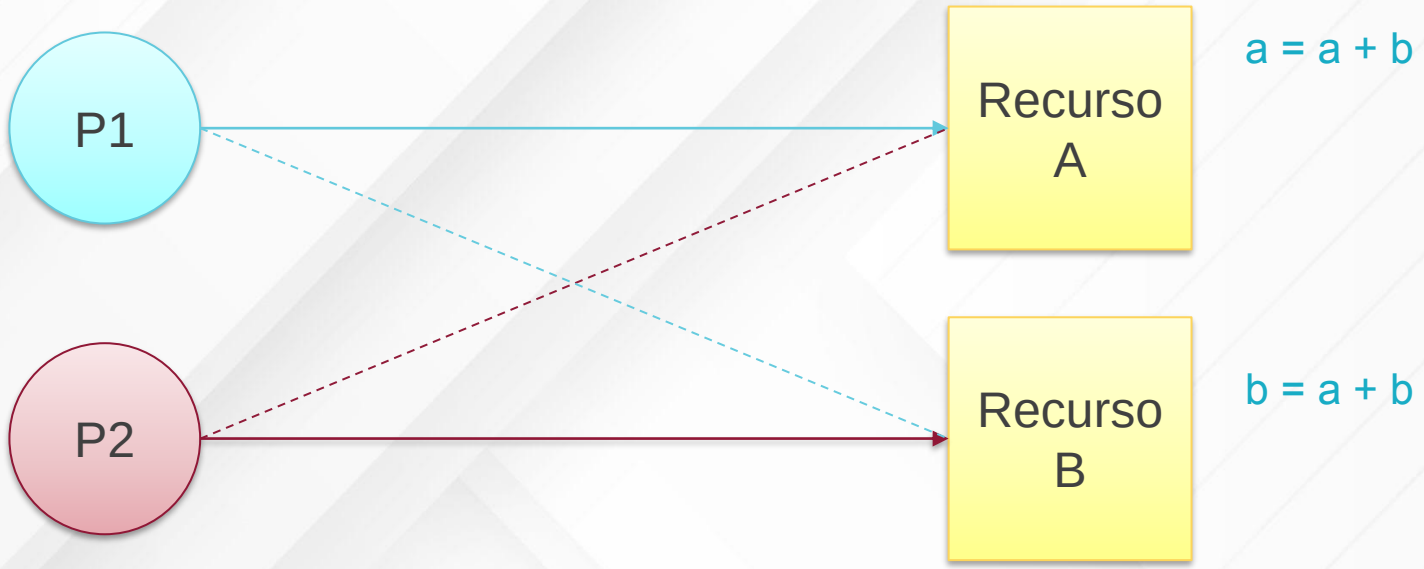
Escrituras
1
2
3

ILERNA Online

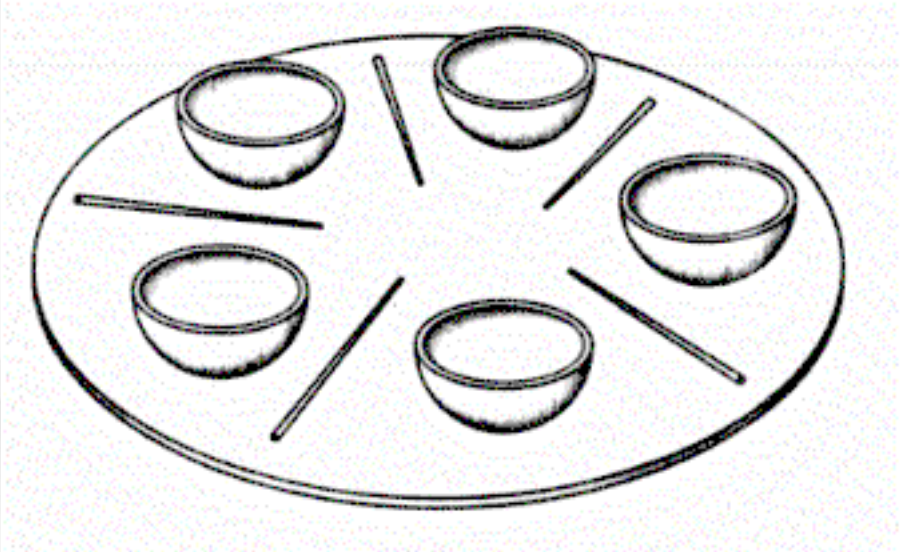
Exclusión mutua: 2 procesos intentar acceder al mismo recurso

Que dos procesos accedan al mismo recurso al mismo tiempo puede producir inconsistencia si uno lee y otro escribe.

Abrazo mortal: 2 procesos quedan bloqueados ya que ambos están esperando un recurso



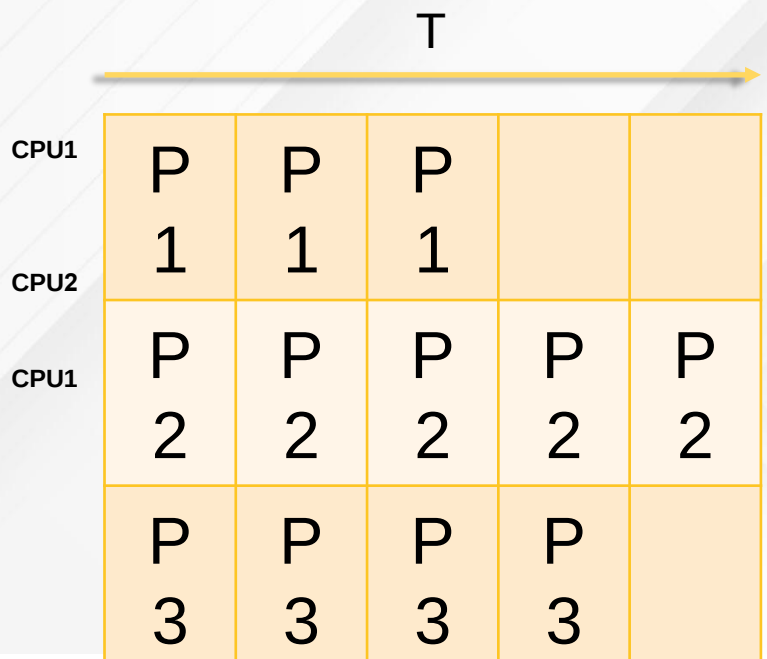
Inanición: un proceso se queda esperando un recurso compartido que siempre se le deniega.
Ejemplo cena china



Procesos

ILERNA Online

En este tipo de programación muchas instrucciones se ejecutan simultáneamente. Permite dar solución a problemas de grandes dimensiones.



Ejecución de tareas simultaneas	Dificultad de programación.
Resolver problemas complejos	Complejidad de acceso a datos
Disminuye tiempo de ejecución	

Tuberías: por donde pasar la información de forma segura. Por ejemplo un proceso padre pasa una variable a sus dos procesos hijos. Pasa la variable a través de la tubería de forma segura y luego los hijos se la devolverán al padre.

Memoria compartida

Zona de la RAM donde pueden acceder múltiples procesos

Mecanismos de control en las zonas críticas (lectura y escritura de datos)

Mecanismos de control
Semáforos
Tuberías
Monitores

Hasta que un hilo no termina otro no puede acceder

Paso de mensajes

Mecanismo más utilizado en la programación orientada a objetos.

Cada proceso tiene definidas sus funciones y métodos.

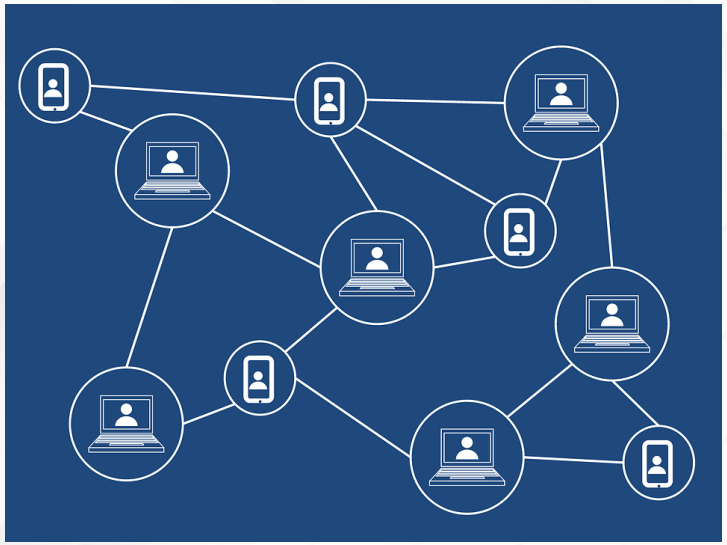
Si necesita datos de otro proceso los solicita.

Un monitor es un objeto en java que nos permite acceder o no acceder a una zona compartida. Declararemos métodos sincronizados. Cuando un método está sincronizado, necesita que no haya ningún otro utilizándolo. Un proceso cuando tiene que ejecutar este método si hay otro que lo está utilizando se espera hasta que el otro avise de que ha terminado. Tiene que haber una comunicación, el que termina avisa al que está esperando y el que está esperando se activa.

ILERNA Online

Conjunto de máquinas separadas físicamente, interconectadas por una red.

En el caso de haber un ordenador mucho más potente que los demás en la misma red, todos los demás usarian esta potencia de cálculo en lugar de hacer dichos calculos en el propio ordenador.

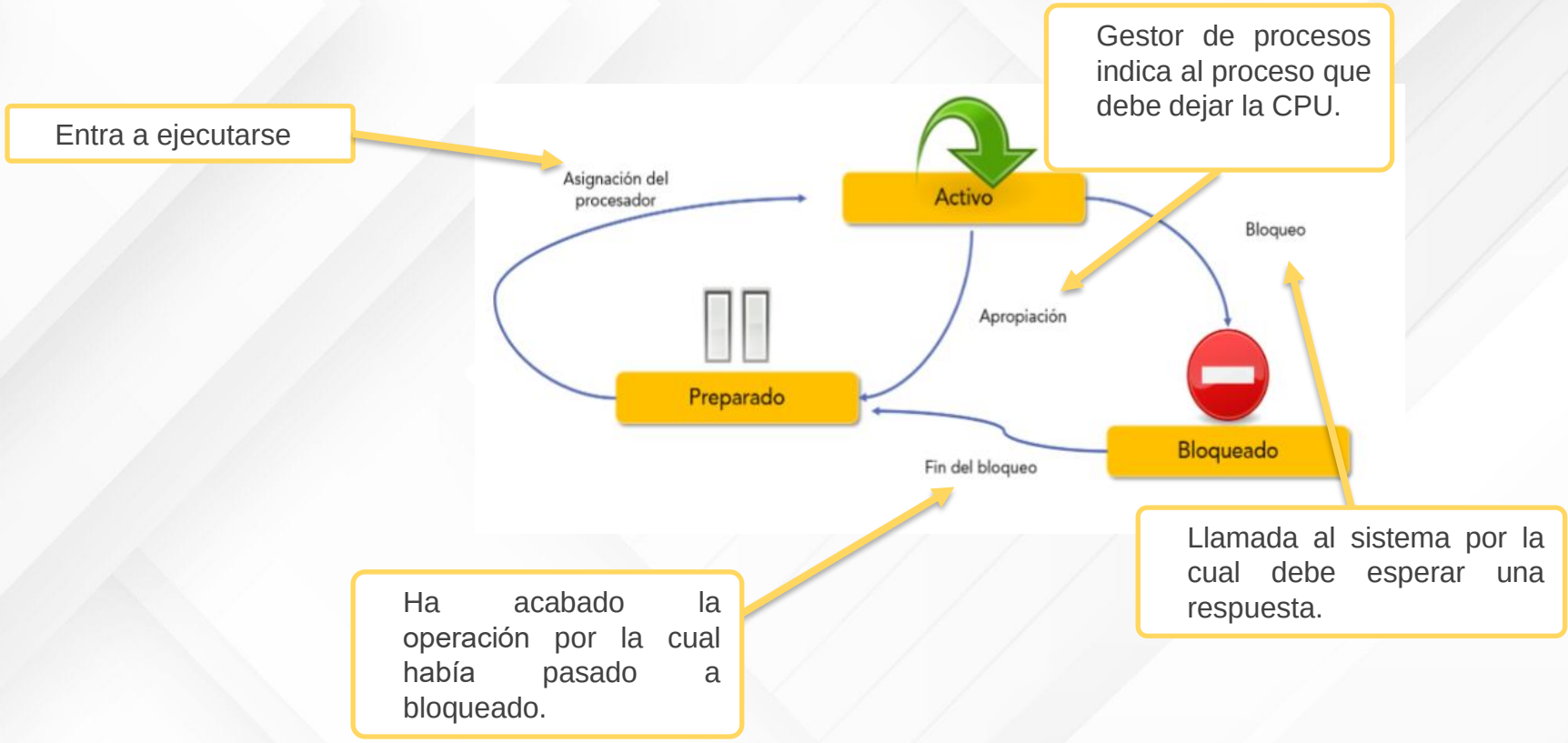


> Estados de un proceso

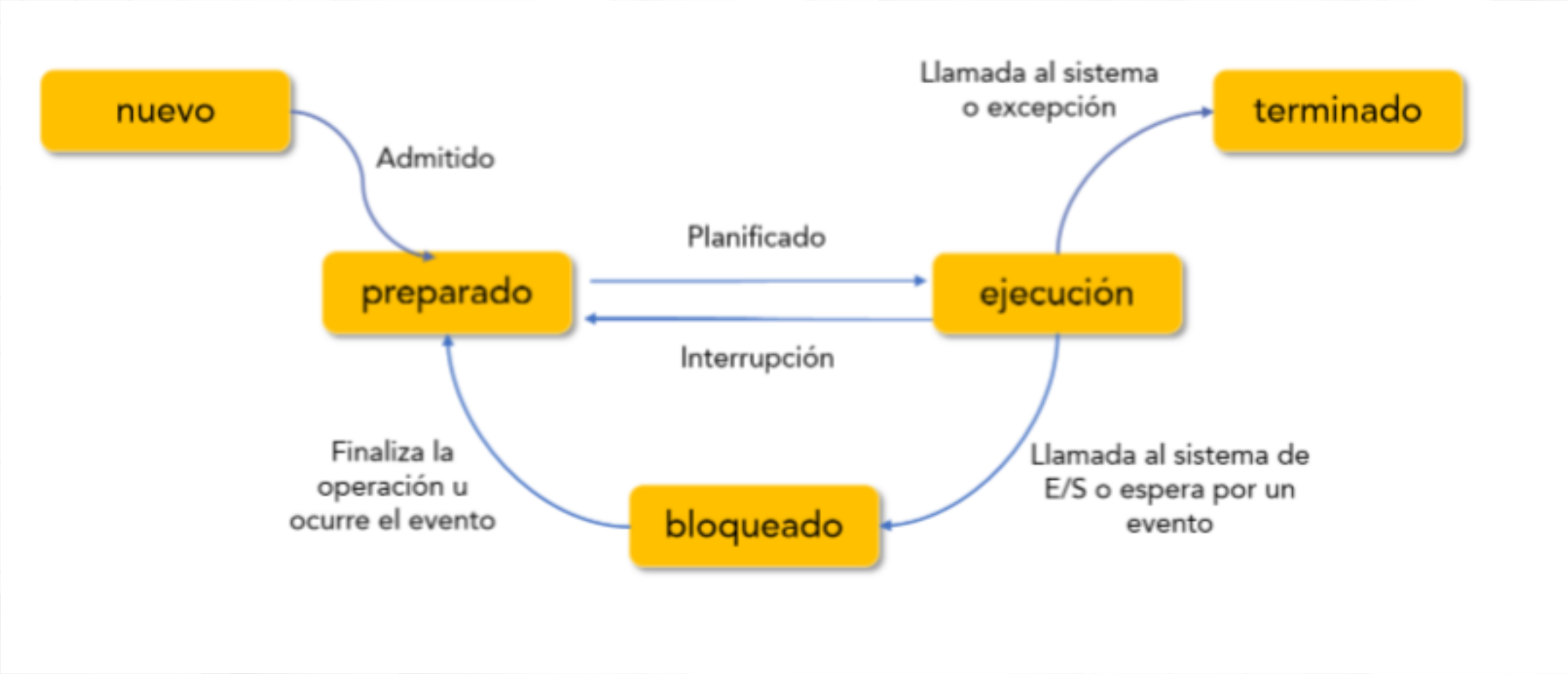
Activo/Ejecución: Proceso asignado para ejecutarse.

Bloqueado: Procesos que han interrumpido su ejecución.

Preparado: Proceso disponible para entrar en la CPU.

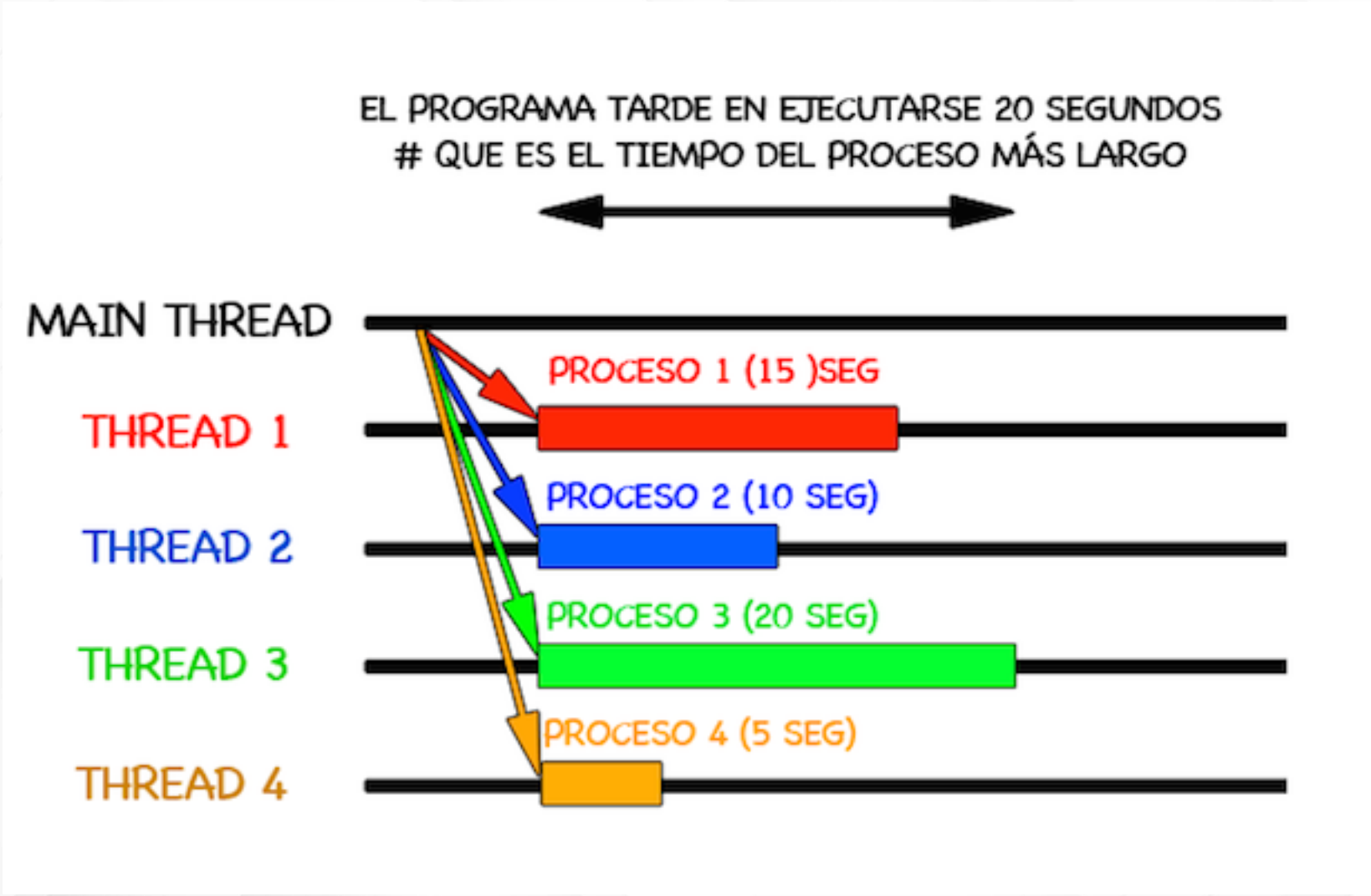


> Estados de un proceso



Programación de hilos en Java

Clase Thread


EL PROGRAMA TARDE EN EJECUTARSE 20 SEGUNDOS
QUE ES EL TIEMPO DEL PROCESO MÁS LARGO
MAIN THREAD
THREAD 1
THREAD 2
THREAD 3
THREAD 4
PROCESO 1 (15)SEG
PROCESO 2 (10 SEG)
PROCESO 3 (20 SEG)
PROCESO 4 (5 SEG)

> Clase Thread

- La clase Thread nos ofrece la posibilidad de crear nuevos hilos en Java.
  - NO existe ninguna forma de crear nuevos procesos sobre el mismo programa en Java

Funciones		
	Start()	Inicia un nuevo Thread de forma paralela
	Run()	Función que se ejecutará de forma concurrente.

```java
public class MiHilo extends Thread {

    @Override
    public void run() {
        for (int i = 0; i < 50; i++) {
            System.out.println("Nuevo hilo en Java");
        }
    }
}
```

```java
public class Main {
    public static void main(String[] args) {
        for (int i = 0; i < 5; i++) {
            new MiHilo().start();
        }
    }
}
```

> Clase Thread

- Veamos un ejemplo sencillo
  - Crea un programa que lance 10 hilos que muestren cada uno 10 saludos

```
1
2  public class Saludo extends Thread {
3
4      int numHilo = 0;
5
6      public Saludo (int numHilo) {
7          this.numHilo = numHilo;
8      }
9
10     @Override
11     public void run() {
12         System.out.println("--- ARRANCA EL HILO Nº " + numHilo);
13
14         for(int i = 0; i<10; i++) {
15             System.out.println("Hilo Nº " + numHilo + " te saluda (Vez: "
16                                + (i+1) + " )");
17         }
18         System.out.println("---- FIN DEL HILO Nº " + numHilo);
19     }
20 }
21
```

```
1
2  public class Main {
3      public static void main(String[] args) {
4
5          for (int i=0; i<10; i++) {
6              Saludo s = new Saludo(i+1);
7              s.start(); estaríamos arrancando un hilo
8          }
9      }
10 }
11
```

ILERNA Online

Join()

- Espera a que finalice el Thread.

Wait()

- Pausa la ejecución del Thread.

Notify()

- Despierta al Thread, lo libera del wait()

NotifyAll()

- Despierta a todos los nodos

Synchronized

- Evita que se pueda ejecutar concurrentemente una porción de código.

ILERNA Online

- Retomemos el ejemplo sencillo de antes
  - Crea un programa que lance 10 hilos que muestren cada uno 10 saludos
  - Añadimos una línea indicando Fin del Programa en la clase principal
  - ¿Cuándo mostrará esta línea?

El método Main es un hilo en si, por lo tanto si no se sincroniza, FIN DE LA EJECUCIÓN DEL PROGRAMA, aparecerá de forma aleatoria.

```
1
2
3  public class Main2 {
4      public static void main(String[] args) {
5
6          for (int i=0; i<10; i++) {
7              Saludo s = new Saludo(i+1);
8              s.start();
9          }
10         System.out.println("########  FIN DE LA EJECUCIÓN DEL PROGRAMA  #########");
11     }
12 }
13
```

- Retomemos el ejemplo sencillo de antes
  - ¿Cómo podríamos conseguir que la línea en cuestión se ejecutase justo antes de terminar el programa?
  - Queremos que primero terminen de ejecutarse todos los hilos

```
Hilo Nº 10 te saluda (Vez: 4 )
Hilo Nº 10 te saluda (Vez: 5 )
Hilo Nº 10 te saluda (Vez: 6 )
Hilo Nº 10 te saluda (Vez: 7 )
Hilo Nº 10 te saluda (Vez: 8 )
Hilo Nº 10 te saluda (Vez: 9 )
Hilo Nº 10 te saluda (Vez: 10 )
---- FIN DEL HILO Nº 10
########  FIN DE LA EJECUCIÓN DEL PROGRAMA  #########
```

```
2
3  public class Main2 {
4      public static void main(String[] args) throws Exception {
5
6          for (int i=0; i<10; i++) {
7              Saludo s = new Saludo(i+1);
8              s.start();
9              s.join();
10         }
11         System.out.println("########  FIN DE LA EJECUCIÓN DEL PROGRAMA  #########");
12     }
13 }
14
```

Hasta que no finalicen los hilos, no empieza el siguiente, por eso ahora FIN DE LA EJECUCIÓN DEL PROGRAMA aparece de forma ordenada al finalizar el programa.

FUNCIONES THREAD

MÉTODO	QUÉ HACE
int getPriority()	Devuelve prioridad del hilo
setPriority(int p)	Cambia prioridad del hilo al entero p
void interrupt()	Interrumpe ejecución del hilo
boolean interrupted()	Comprueba si hilo ha sido interrumpido
Thread currentThread()	Devuelve referencia al hilo que se está ejecutando
boolean isDaemon()	Comprueba si hilo es Daemon (prioridad baja ejecuta en segundo plano)
setDaemon(boolean on)	Establece hilo como daemon, con true o false (pasa a hijo de usuario)

FUNCIONES THREAD

MÉTODO	QUÉ HACE
start()	Comienza ejecución del hilo, la máquina virtual llama a run()
boolean isAlive()	Comprueba si hilo sigue vivo
sleep(long mils)	Hilo en ejecución pasa a bloqueado durante milisegundos indicados
run()	Construye hilo, llamado por start()Si devuelve el control, hilo se detieneÚnico método de interfaz Runnable
String toString	Devuelve hilo representado en cadena, con nombre, prioridad y grupo de hilos
long getId()	Devuelve identificador del hilo
void yield	Detiene hilo en ejecución temporalmente para dar paso a otros hilos
getName(String name)	Cambia nombre al hilo por el que se le pasa

> EJERCICIO PARA PRACTICAR

ILERNA Online

- Crea un programa en Java que lance 3 hilos, con un nombre y con una prioridad cada uno
- Cada uno de ellos mostrará un mensaje que avise que estamos dentro de él, su prioridad, su ID y el número de hilos que en ese momento hay activos.
- Además el programa principal, indicará que ha creado los 3 hilos y cuántos hilos están activos en el momento en que escribe

Ejercicios
Trabajando con Hilos

- Crea un programa en Java que simule una carrera de atletas
- Cada uno deberá recorrer 100 metros de forma que cada vez que se ejecuten se generará un número aleatorio de metros recorridos
- En caca iteración, cada atleta se identificará y dirá cuantos metros ha recorrido
- Una vez el atleta llegue a la línea de meta, avisará por consola y finalizará su ejecución
- El programa principal avisará cuando la carrera haya finalizado

Interfaz Runnable

- Java únicamente permite heredar de una clase.
- La interfaz Runnable nos permite tener Threads en una clase heredada

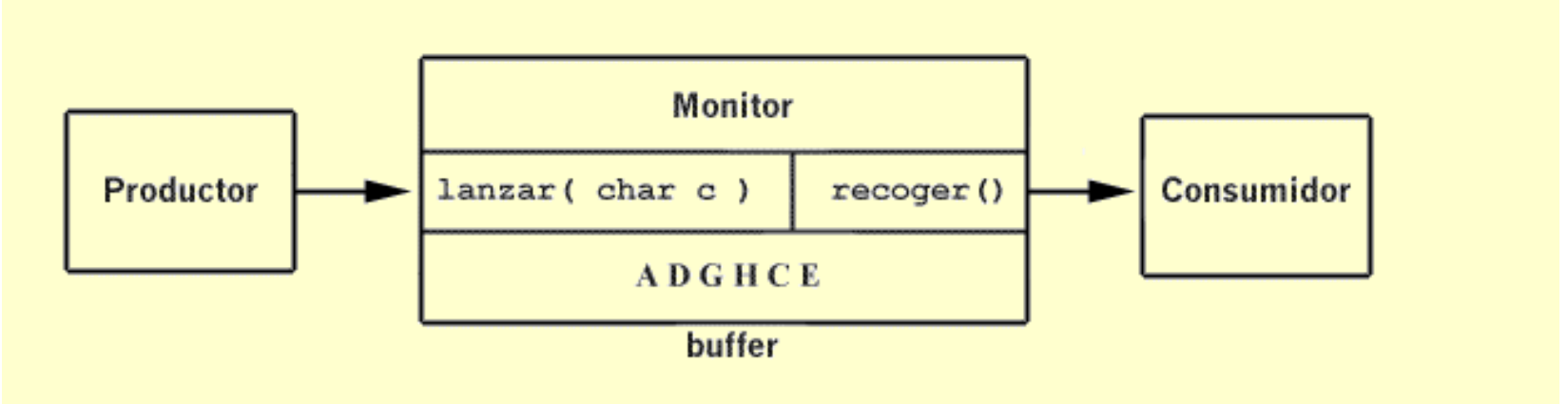
```
public class Main {
    public static void main(String[] args) throws
InterruptedException {
        for (int i = 0; i < 5; i++) {
            MiHilo hilo = new MiHilo(i);
           new Thread(hilo).start();
        }
    }
}
```

```
public class MiHilo implements Runnable {
    private int id;
    public MiHilo(int id) {
        this.id = id;
    }
    @Override
    public void run() {
        System.out.println("Runnable con id: " + id);
        for (int i = 0; i < 50; i++) {
            System.out.println("Nuevo hilo en Java con Runnable, " + i);
        }
    }
}
```

cl.get(i).start(); Para arrancar el hilo.

cl.get(i).join(); Para que el programa principal sepa cuando todos los hilos han terminado.

- EL MODELO PRODUCTOR-CONSUMIDOR
  - ¿Un problema típico de sincronización es el que representa el modelo Productor-Consumidor
  - Se da cuando uno o más hilos producen datos a procesar y otros hilos los consumen
  - El problema surge cuando el productor produce datos más rápidos que el consumidor los consuma, dando lugar a que el consumidor se salte algún dato
  - Igualmente el consumidor puede consumir más rápido que el productor produce, entonces el consumidor puede recoger varias veces el mismo dato o puede no tener datos que recoger o puede detenerse, etc



EJERCICIO PARA PRACTICAR

- Crea Crea dos clases (hilo) en Java que extiendan de la clase Thread. Uno de los hilos debe mostrar en la pantalla en un bucle infinito la palabra TIC y el otro hilo la palabra TAC.
- Dentro del bucle, usa el método sleep() para que dé tiempo a visualizar las palabras que se muestran al ejecutarlos.
- Debe mostrarse TIC TAC TIC TAC…

EJERCICIO PARA PRACTICAR

```java
public class Main {
    public static void main (String[] args){

        Tic tic = new Tic(4);
        Tac tac = new Tac(2);

        tic.start();
        tac.start();

    }
}
```

```java
public class Tic extends Thread{

    //Propiedades
    private int hilo;

    //Constructor
    public Tic(int hilo) {
        this.hilo = hilo;
    }
    //Métodos
    public void run() {
        while(true) {
            try {
                System.out.println("TIC");
                sleep(1000);
                this.yield(); //Para que el hilo
                    //se detenga unos instantes
            }catch(InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```java
public class Tac extends Thread {
    //Propiedades
        private int hilo;

        //Constructor
        public Tac(int hilo) {
            this.hilo = hilo;
        }

        //Métodos
        public void run() {
            while(true) {
                try {

                    System.out.println("TAC");
                    sleep(1000);

                }catch(InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
}
```

ejecutarlos.

- Debe mostrars

- EL MODELO PRODUCTOR-CONSUMIDOR
  - ¿Se definen tres clases:
  - **Clase Cola,** que será el objeto compartido entre el productor y el consumidor
  - **Clase Productor**
  - **Clase Consumidor**

- EL MODELO PRODUCTOR-CONSUMIDOR

  - CLASE PRODUCTOR

  - El productor produce números y los coloca en una cola, éstos serán consumidos por el consumidor.

  - El recurso a compartir será la cola con los números.

  - El productor genera números de 0 a 5 en un bucle for, y los pone en el objeto Cola mediante el método put(); después se visualiza y se hace una pausa con sleep().

```
class Productor extends Thread{
    private Cola cola;
    public Productor(Cola c){
        cola = c;

    }
    public void run(){
        for (int i =0; i<5;i++){
            cola.put(i); // pone el número en la cola
            System.out.println(i + "=>Productor produce: " + i);
            try {
                sleep(100);
             } catch (InterruptedException ex) {
             }
        }
    }
}
```

- EL MODELO PRODUCTOR-CONSUMIDOR

  - CLASE CONSUMIDOR

  - La Clase Consumidor es muy similar a la clase Productor, solo que en lugar de poner un número en el objeto Cola lo recoge llamando al método get().

  - En el ejemplo que se muestra a continuación no se ha puesto pausa, con esto hacemos que el consumidor sea más rápido que el productor

```
class Consumidor extends Thread{
    private Cola cola;
    public Consumidor(Cola c){
        cola = c;
    }

    public void run(){
        int valor=0;
        for (int i =0; i<5;i++){
            valor = cola.get();
            System.out.println(i + "=>Consumidor consume: " + valor);
        }
    }
}
```

- EL MODELO PRODUCTOR-CONSUMIDOR

  - CLASE COLA
  - La clase Cola define dos atributos y dos métodos
  - En el atributo numero se guarda el número entero y el atributo disponible se utiliza para indicar si hay disponible o no un número en la cola
  - El método put() guarda un entero en el atributo numero y hace que éste esté disponible en la cola para que pueda ser consumido el atributo disponible a true (cola llena)
  - El método get() devuelve el entero de la cola si está disponible (disponible=true) y antes pone la variable a false indicando cola vacía; si el número no está en la cola(disponible=false) devuelve -1

```
class Cola{
    int numero;
    private boolean disponible = false;

    public int get(){
        if (disponible){
            disponible = false;
            return numero;
        }
        return -1;
    }
    public void put(int valor){
        numero = valor;
        disponible = true;
    }
}
```

- EL MODELO PRODUCTOR-CONSUMIDOR

  - CLASE MAIN (productor-consumidor)

  - En el main() creamos tres objetos, un objeto de la clase Cola, un objeto de la clase Productor y otro objeto de la clase Consumidor

  - Al constructor de la clase Productor y Consumidor le pasamos el objeto compartido de la clase Cola:

```
public static void main(String[] args) {
    Cola cola = new Cola();
    Productor p = new Productor(cola);
    p.start();
    Consumidor c = new Consumidor(cola);
    c.start();
}
```

- EL MODELO PRODUCTOR-CONSUMIDOR
  - Si ejecutamos el código del ejemplo, obtendremos algo similar a esta salida:

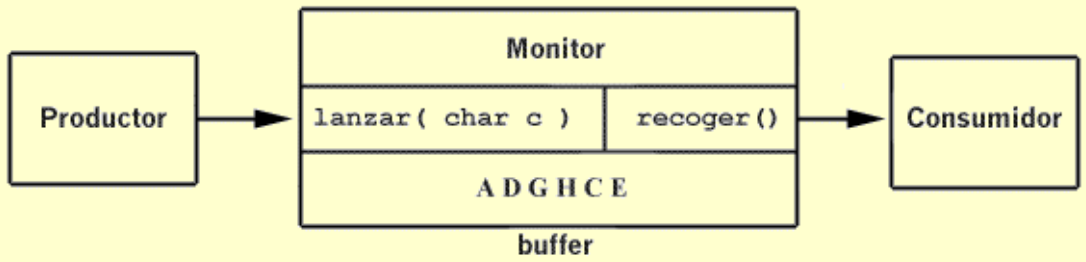
```
run:
0=>Productor produce: 0
0=>Consumidor consume: 0
1=>Consumidor consume: -1
2=>Consumidor consume: -1
3=>Consumidor consume: -1
4=>Consumidor consume: -1
1=>Productor produce: 1
2=>Productor produce: 2
3=>Productor produce: 3
4=>Productor produce: 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

- EL MODELO PRODUCTOR-CONSUMIDOR
  - Para que el resultado sea correcto:
  - PRIMERO: necesitamos que los métodos put y get sean SINCORNIZADOS, para evitar que no se meta nada en la cola si ya está llena, o para evitar que se coja de la coa si está vacía

```
public synchronized int get(){
// instrucciones
}
public synchronized void put(int valor){
//instrucciones
}
```

- EL MODELO PRODUCTOR-CONSUMIDOR
  - Para que el resultado sea correcto:
  - SEGUNDO: Para mantener esta coordinación usamos los métodos wait() y notifyAll()
  - IMPORTANTE: Los métodos wait() y notifyAll() pueden ser invocados solo desde dentro de un método sincronizado o dentro de un bloque sincronizado
  - El método get() tiene que esperar a que la cola se llene, esto se realiza en el bucle while: mientras la cola esté vacía, es decir disponible es false (while(!disponible)), espero(wait)
  - Se sale de un bucle cuando llega un valor, en este caso se vuelve a poner disponible a false (porque se devuelve el número quedando la cola vacía de nuevo), se notifica a todos los hilos que comparten el objeto este hecho y devuelve el valor

- EL MODELO PRODUCTOR-CONSUMIDOR
  - Solución usando MONITORES para la clase Cola




```
class Cola{
    int numero;
    private boolean disponible = false; // inicialmente cola vacía

    public synchronized int get(){
        while (!disponible){
            try {
                wait();
            } catch (InterruptedException ex) {

            }
        }
        disponible = false;
        notifyAll();
        return numero;


    }
    public synchronized void put(int valor){
        while (disponible){
            try {
                wait();
            } catch (InterruptedException ex) {

            }
        }
        numero = valor;
        disponible = true;
        notifyAll();
    }
}
```

EJERCICIO PARA PRACTICAR

- ¿Cómo podrías resolver el ejercicio de TIC TAC usando el modelo productor consumidor y los métodos sincronizados?

- PISTA: tendrás que modificar el Productor para reciba una cadena de texto que será el mensaje que el consumidor ponga el la Cola, y tras coger dicho mensaje, mostrarlo por la consola. El main tendrá dos hilos Productor, uno que escriba TIC y otro que escriba TAC

UF3

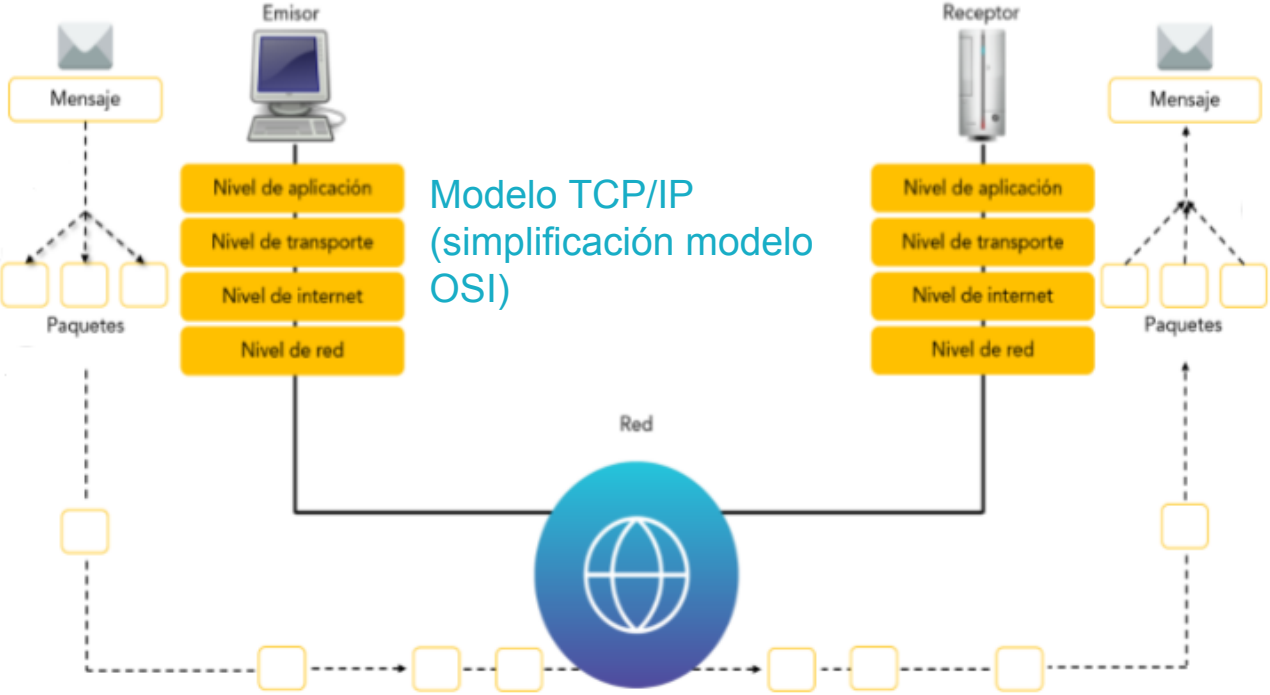
SOCKETS Y SERVICIOS

COMUNICACIONES

Las redes están formadas por un conjunto de dispositivos que se encuentran conectados entre ellos para poder intercambiar información.

Ejemplo: Internet

En estos momentos, yo estoy enviando señal de vídeo y audio y vosotros la estáis recibiendo. Vosotros enviáis mensajes de chat y yo los recibo



Modelo TCP/IP (simplificación modelo OSI)

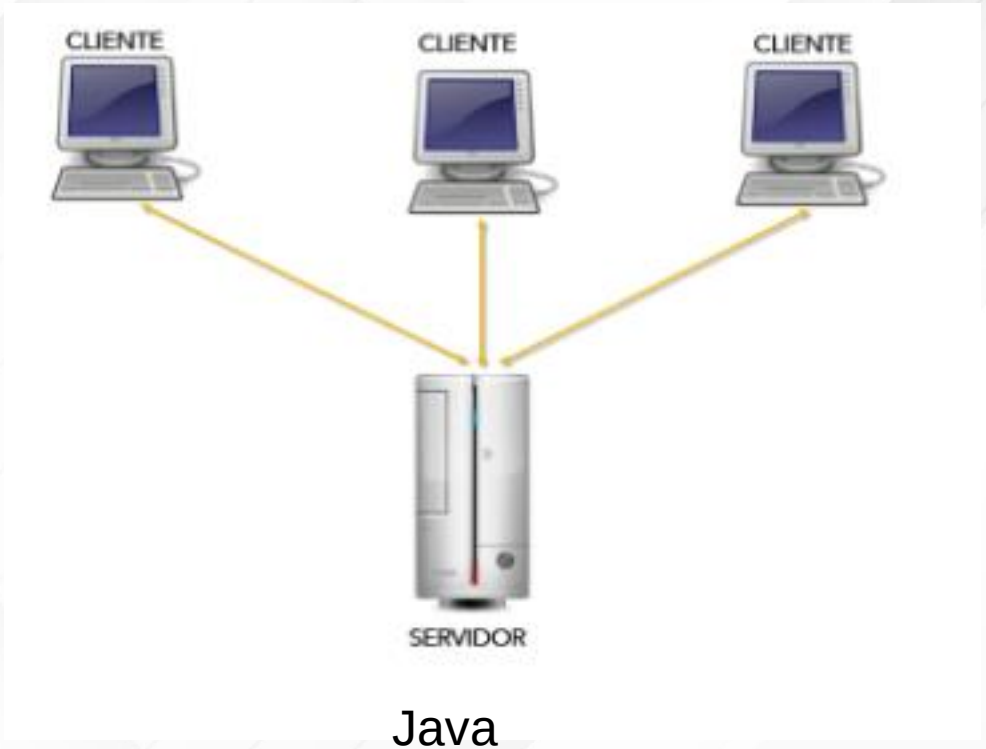
1. **Aplicación**: El emisor envía el mensaje.
2. **Transporte**: El mensaje se divide en paquetes para que puedan viajar por la red.
3. **Internet**: Se busca el camino y se envían los paquetes al destino.
4. Red se encarga de transmitir el paquete.
5. La tarjeta (de red) del emisor recibe los mensajes.
6. **Internet**: Comprueba si los paquetes son correctos.
7. **Transporte**: Reagrupa los paquetes y los descomprime.
8. **Aplicación**: Recibe el mensaje.

> Modelo cliente-servidor

En este modelo los clientes realizan las peticiones al servidor y este responde con la información necesaria.

Gracias a este modelo las aplicaciones cliente y servidor pueden estar programadas en diferentes lenguajes o ejecutarse en diferentes SO y entenderse.

JS / Java C# Python

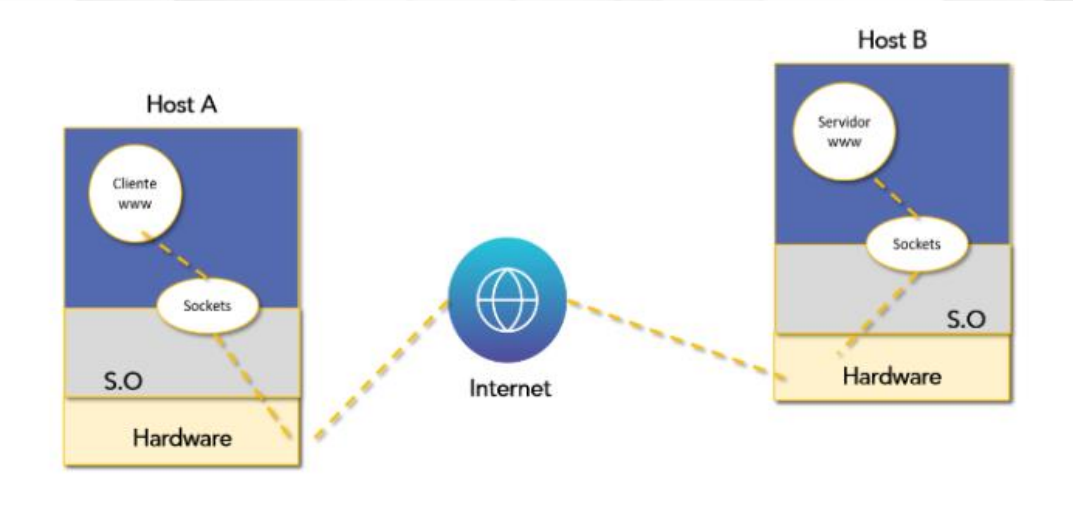


Java

> ¿Qué es un socket?

Es un mecanismo que permite la comunicación entre aplicaciones a través de la red. Medio mediante el cual vamos a establecer la comunicación. La información viajará entre la linea del socket 1 y el socket 2.
El socket va a ser el encargado de empaquetarlo y enviarlo, además del canal por el que se transmite.

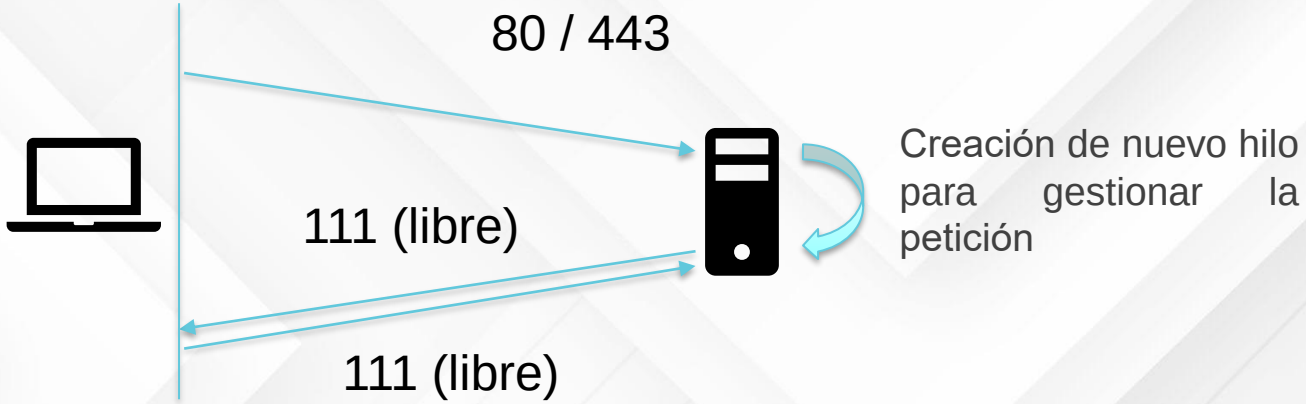
Abstrae al usuario del paso de la información entre diferentes capas.



Para que un cliente pueda conectarse con un servidor debe conocer su dirección IP y el puerto asignado.

Cuando un servidor recibe una petición, si la acepta, asigna un puerto para la comunicación y de esta forma deja libre el puerto correspondiente a las peticiones que reciba del exterior.

Gestor del DNS es el que se encarga de traducir la web a la IP que queremos acceder.



Tipos de SOCKETS

ILERNA Online

Sockets orientados a conexión - TCP

- Se utiliza en aquellas aplicaciones que quieran una alta fiabilidad en el envío de mensajes.
- Mediante este protocolo se asegura que todos los paquetes sean entregados.
- Ejemplo: Descarga de archivo

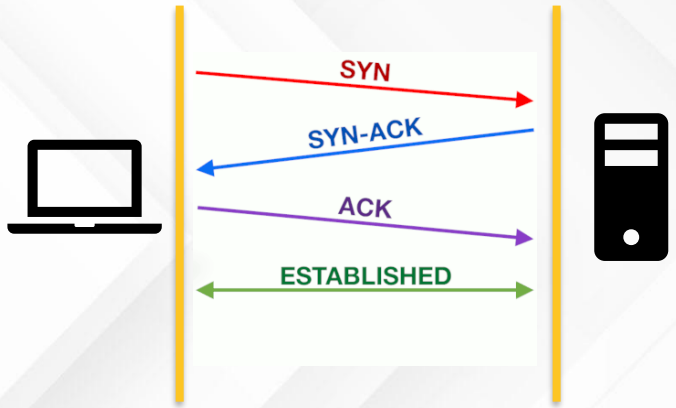
Sockets NO orientados a conexión - UDP

- Se utiliza en aquellas aplicaciones que deseen enviar los paquetes lo más rápido posible.
- No se garantiza ni la llegada ni el orden de los paquetes.
- Ejemplo: llamada VoIP o voz IP.

ILERNA Online

Establecer la conexión:
Three Way Handshake

Mantener la conexión:
Para cada petición se devuelve un ACK con la confirmación.
Si se pierde un paquete al llegar al timeout se vuelve a enviar.

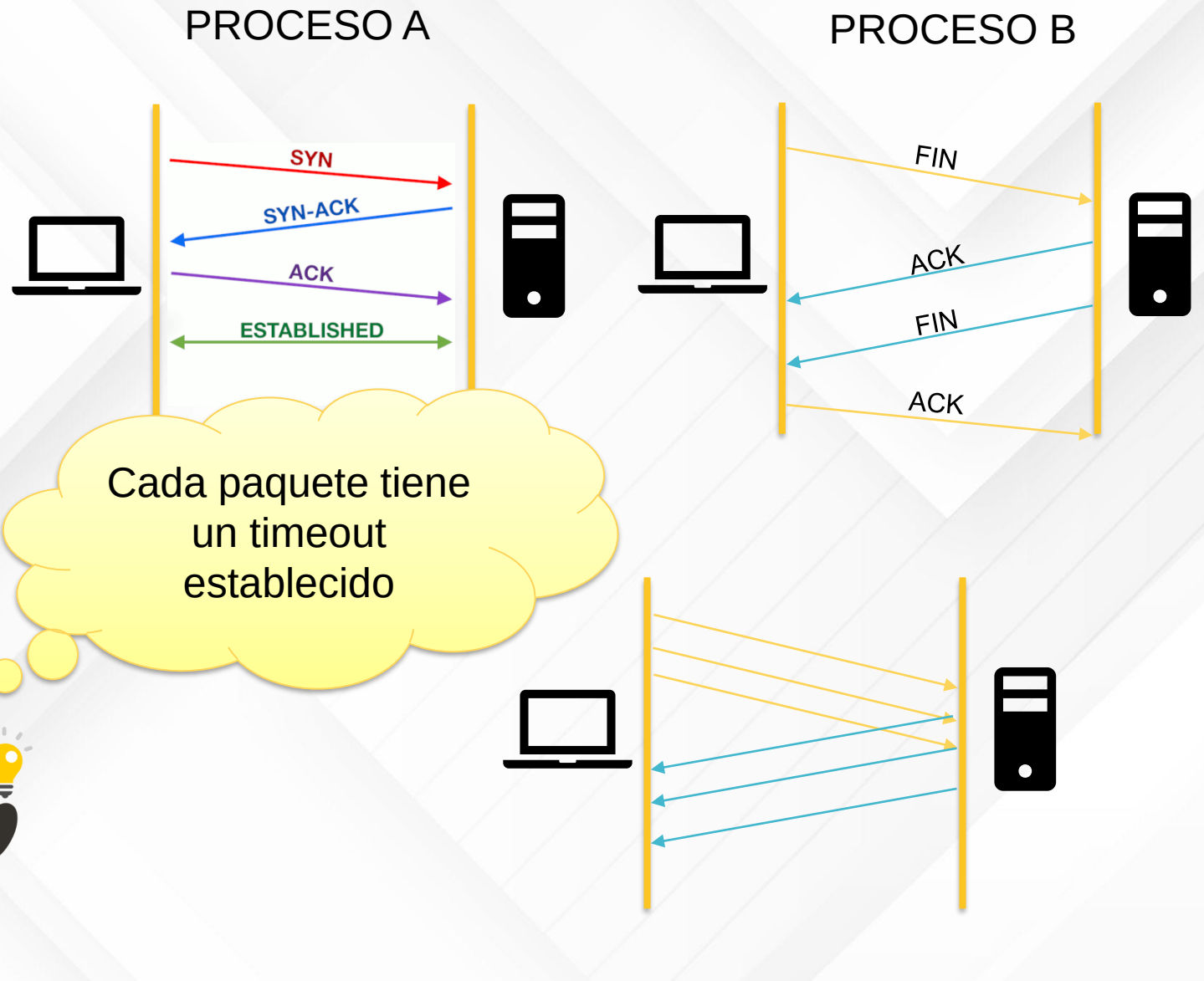


Así funciona la comunicación entre cliente y servidor

Mantener la conexión abierta enviando cada cierto tiempo señales ACK

Si se cortara la conexión, habría que repetir proceso A

Finalizar la conexión



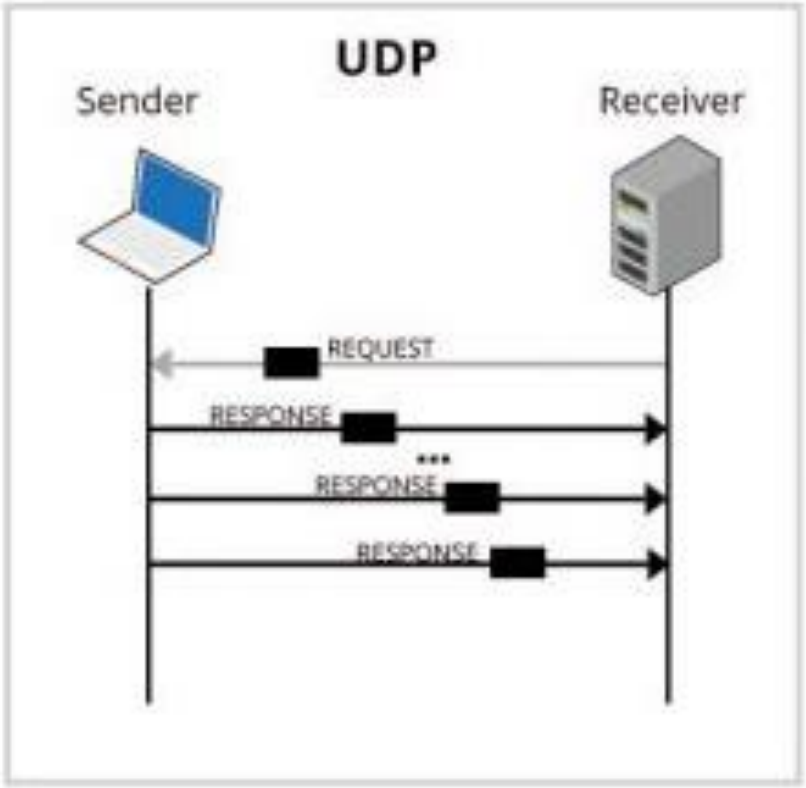
CLIENTE SERVIDOR

En la conexión, cliente envía paquetes al servidor.

Tanto si el servidor los recibe como si no, todo ok y sigue enviando

Si se cortara la conexión, habría que repetir proceso A

Finalizar la conexión

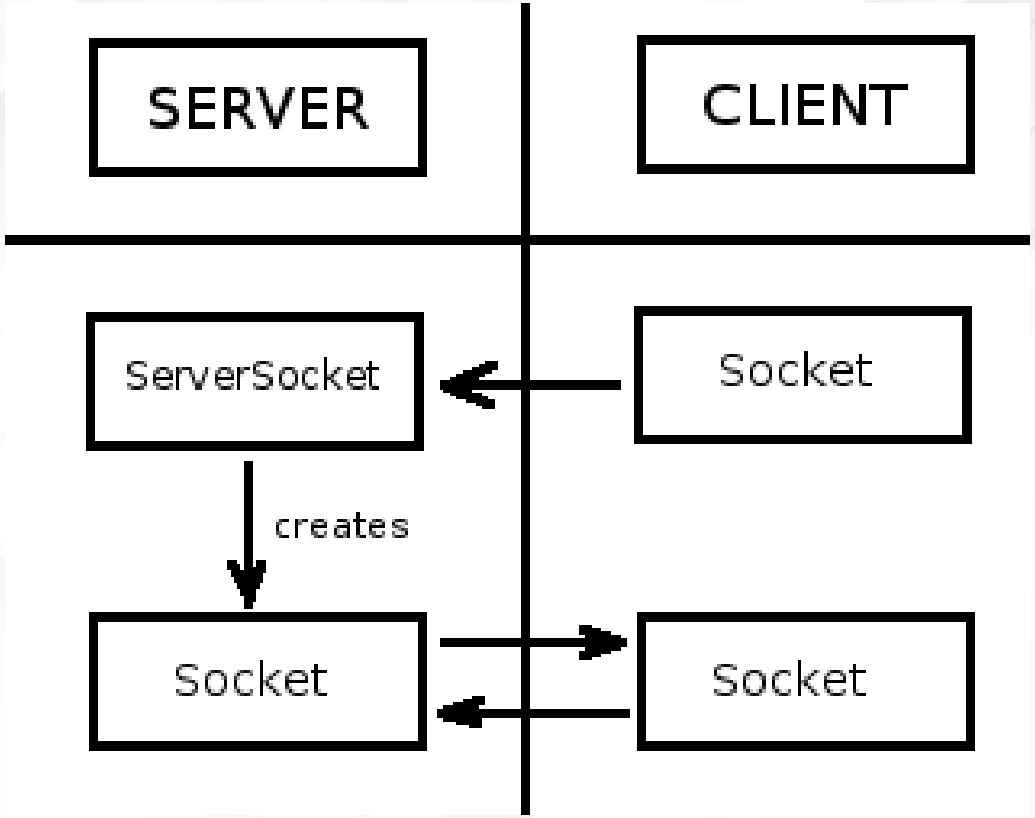


> Servidor – Crear connexiones

```
private final int PUERTO = 4321;
private ServerSocket serverSocket; //Socket correspondiente al servidor
private Socket socket; //Socket correspondiente al cliente

public Servidor() throws IOException {
    serverSocket = new ServerSocket(PUERTO);
    socket = new Socket();
}
```

> Aceptar conexiones

```
socket = serverSocket.accept();
```



ILERNA Online

PROGRAMACIÓN EN EL LADO DEL SERVIDOR

```
DataOutputStream mensajeCliente = new DataOutputStream(socket.getOutputStream());
mensajeCliente.writeUTF("Petición rebida");
DataInputStream entrada = new DataInputStream(socket.getInputStream());
String mensajeDeCliente;
try {
    while (!(mensajeDeCliente = entrada.readUTF()).isEmpty()) System.out.println(mensajeDeCliente);
}catch (EOFException ex){
    System.out.println("Fin de la comunicación");
}
```

ILERNA Online

PROGRAMACIÓN EN EL LADO DEL SERVIDOR

SERVIDOR	CLIENTE
DataInputStream	getInputStream()
DataOutputStream	getOutputStream()

```
DataOutputStream mensajeCliente = new DataOutputStream(socket.getOutputStream());
mensajeCliente.writeUTF("Petición rebida");
DataInputStream entrada = new DataInputStream(socket.getInputStream());
String mensajeDeCliente;
try {
    while (!(mensajeDeCliente = entrada.readUTF()).isEmpty()) System.out.println(mensajeDeCliente);
}catch (EOFException ex){
    System.out.println("Fin de la comunicación");
}
```

→ S envía datos a C

→ C envía datos a S

> Cliente – Crear connexión

```
Socket socket;
socket = new Socket("localhost", 4321);
```

A quién nos conectamos y
a través de qué puerto

> Enviar y recibir datos

```
DataInputStream entradaServidor = new DataInputStream(socket.getInputStream());
DataOutputStream salidaServidor = new DataOutputStream(socket.getOutputStream());
System.out.println(entradaServidor.readUTF());
for(int i =0;i<3;i++){
    salidaServidor.writeUTF("Este es el mensaje nº: "+i);
}
salidaServidor.close();
entradaServidor.close();
socket.close();
```

EJEMPLO DE COMUNICACIÓN CLIENTE - SERVIDOR

> PROYECTO COMUNICACIONES C - S

ILERNA Online

MainCliente.java

```java
package cliente;

import java.io.IOException;

public class MainCliente{

        public static void main(String[] args) throws IOException {
```

PROGRAMACIÓN EN EL LADO DEL CLIENTE

MainServer.java

```java
package servidor;

import java.io.IOException;

public class MainServer {

    public static void main(String[] args) throws IOException {
```

PROGRAMACIÓN EN EL LADO DEL SERVIDOR

ILERNA Online

PROGRAMACIÓN EN EL LADO DEL CLIENTE

Cliente.java

```java
package cliente;

import java.io.DataInputStream;

public class Cliente {

    private final String HOST = "localhost";
    private final int PUERTO = 4321;
    private Socket socket;

    public Cliente() throws IOException {
        socket = new Socket("localhost", 4321);
    }

    public void iniciarCliente() throws IOException {

        //Iniciamos la entrada de datos
        DataInputStream entradaServidor = new DataInputStream(socket.getInputStream());
        System.out.println(entradaServidor.readUTF()); //Mostramos el mensaje por pantalla

        //Enviar 3 mensajes
        DataOutputStream salidaServidor = new DataOutputStream(socket.getOutputStream());
        for(int i =0;i<3;i++){

            salidaServidor.writeUTF("Este es el mensaje num: "+i);

        }
        salidaServidor.close();
        entradaServidor.close();
        socket.close();
    }
}
```

Volvemos a la clase MainCliente para term

Servidor.java

```java
package servidor;

import java.io.*;

public class Servidor {
    private final int PUERTO = 4321; //no va a cambiar
    private ServerSocket serverSocket;
    private Socket socket;

    //Definimos el constructor
    public Servidor() throws IOException {
        serverSocket = new ServerSocket(PUERTO); //Definimos la conexión
        socket = new Socket(); //Iniciamos el cliente
    }

    //Función para iniciar la conexión
    public void iniciarServer() throws IOException {

        //Vamos a aceptar los datos que llegarán del cliente
        while (true) {
            System.out.println("Esperando la conexion del cliente");
            socket = serverSocket.accept(); //guardamos la petición que llegue al servidor en socket
            // El servidor se queda a la espera de recibir peticiones

            //Al recibir la petición, iniciamos la conexión
            DataOutputStream mensajeCliente = new DataOutputStream(socket.getOutputStream());
            //Enviamos mensaje al cliente
            mensajeCliente.writeUTF("Petición rebida");
```

Volvemos a la clase MainServidor para terminarla

> PROYECTO COMUNICACIONES C - S

MainServidor.java

PROGRAMACIÓN EN EL LADO DEL SERVIDOR

```java
package servidor;

import java.io.IOException;

public class MainServer {

    public static void main(String[] args) throws IOException {
        //Definimos objeto
        Servidor serv = new Servidor();
        System.out.println("Iniciando servidor . . .");

        //Iniciamos el servidor
        serv.iniciarServer();

        //finalizamos el servidor
        serv.finalizarServer();
    }
}
```

MainCliente.java

PROGRAMACIÓN EN EL LADO DEL CLIENTE

```java
package cliente;

import java.io.IOException;

public class MainCliente{

        public static void main(String[] args) throws IOException {

            //Creamos objeto de Cliente
            Cliente cli = new Cliente();
            System.out.println("Iniciando cliente...");

            //Iniciamos la conexión
            cli.iniciarCliente();
        }
}
```

IMPORTANTE

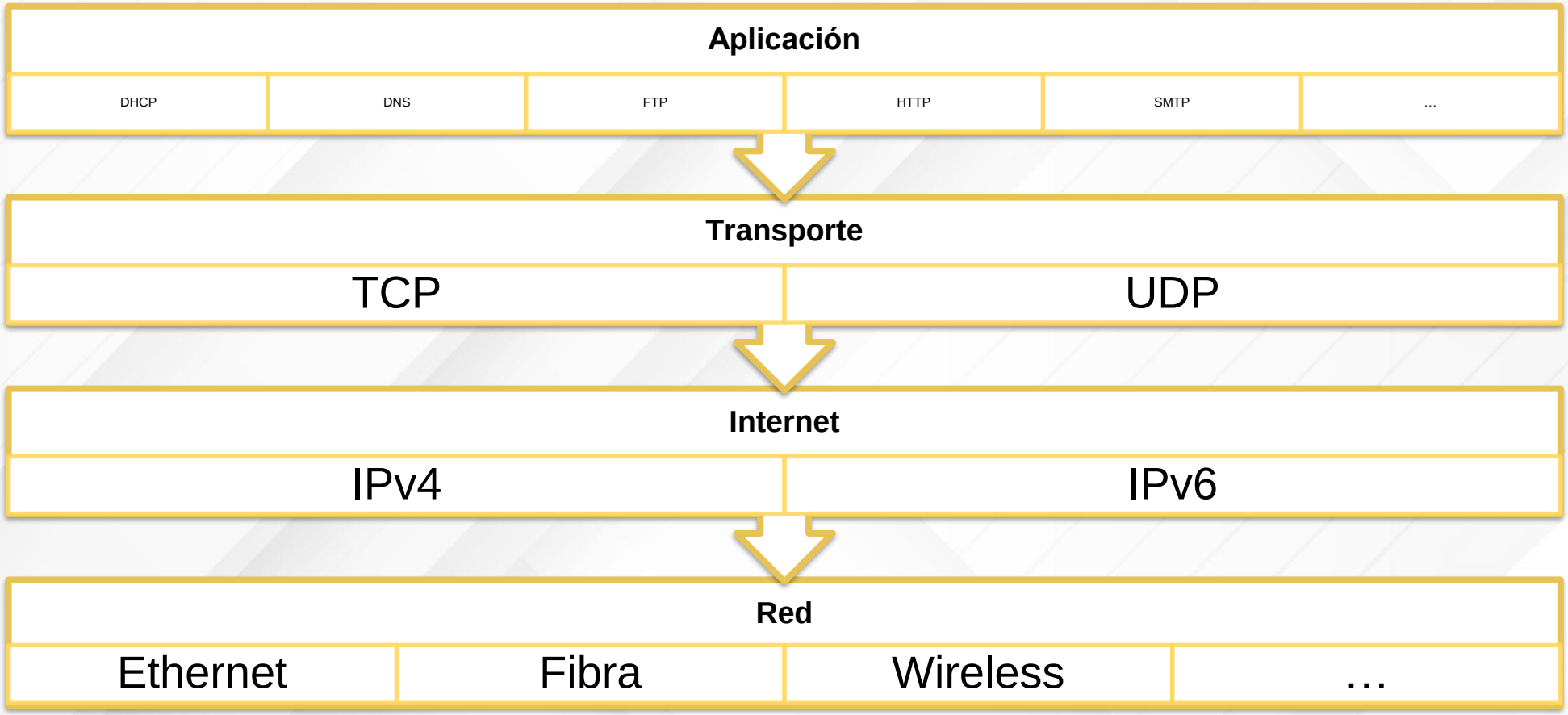
- Iniciar el servidor y después del cliente
- Si iniciamos primero al cliente, la conexión con localhost será rechazada porque no existe localhost
- Servidor queda a la espera de recibir cliente
- Ejecutamos el cliente
- Hay muchas formas diferentes de establecer las conexiones

Generación de servicios de red

Aplicaciones cliente de protocolos estándar.

Aplicaciones cliente:
Son aquellas que realizan peticiones a un servidor solicitando cualquier tipo de recurso o información que esté alojado en el servidor.

Modelo TCP/IP



ILERNA Online

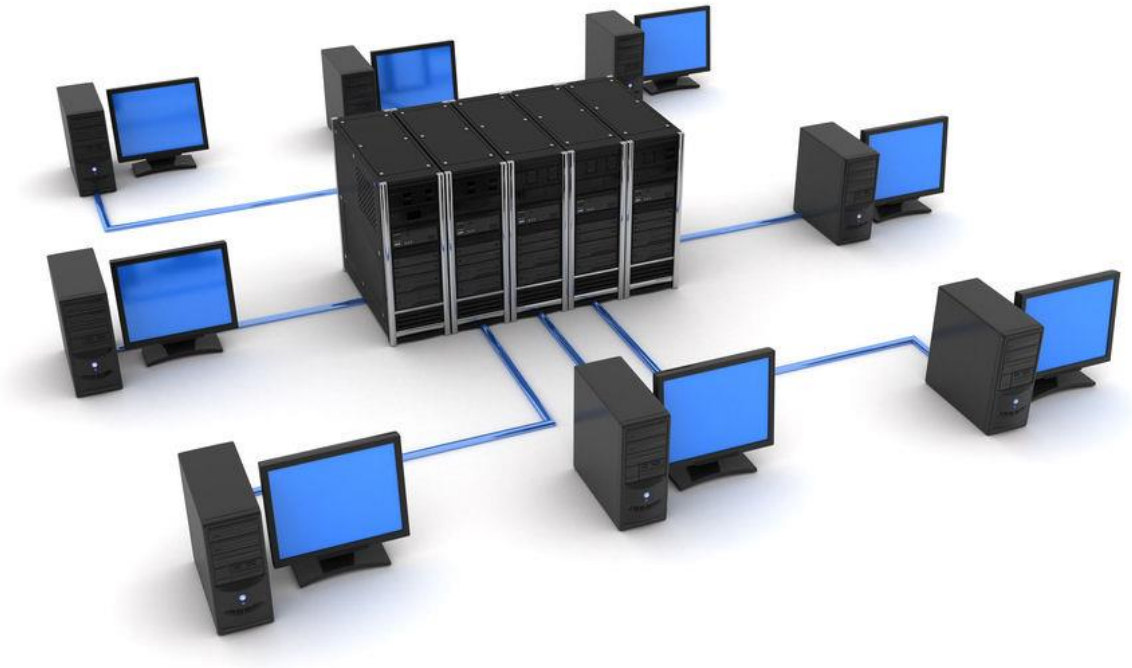
Servidor

ILERNA Online

Servidor

En esencia es un PC vitaminado.
Suelen tener unas características mucho más elevadas que un ordenador.

Un servidor a diferencia de un PC, está diseñado para trabajar de forma continua las 24h del día.



Servidor

Contienen un conjunto de elementos que rara vez se suelen ver en un ordenador como:

- Sensores de errores de los distintos elementos.
- Tarjetas RAID dedicadas (crear combinaciones de discos duros con una función determinada).
- Extracción de discos en caliente.
- SO especializado.
- Soportan gran carga de trabajo sin afectar a su rendimiento.

ILERNA Online

Tipos de servidores

Servidor compartido

- Prestaciones muy altas
- Compartidos entre muchos usuarios por su elevado coste

Servidor dedicado

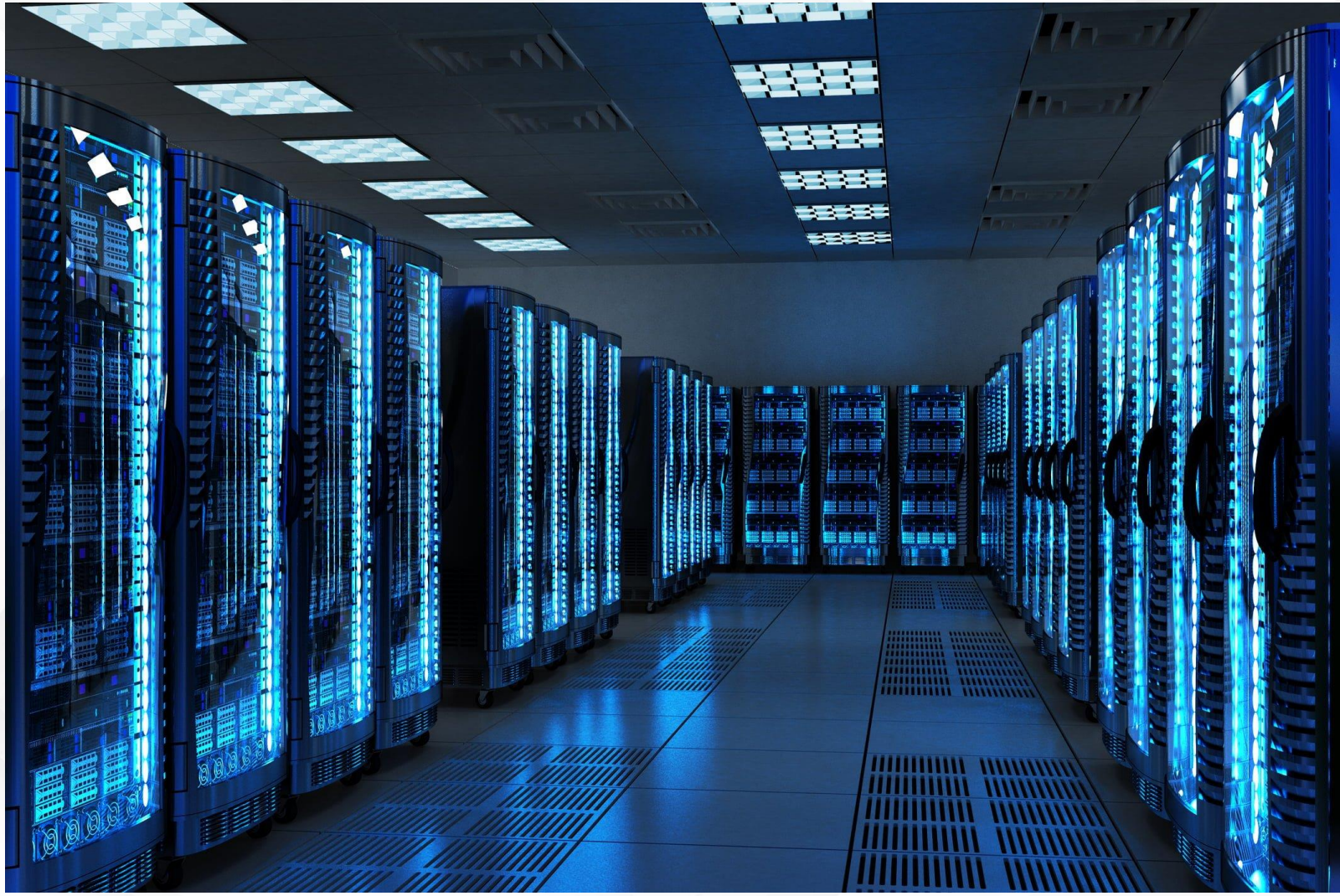
- Servidor único para cada empresa.

ILERNA Online

Tipos de servidores

SERVIDOR COMPARTIDO

- Alojados en CPDs
- Supervisión 24/7
- Acceso SSH



ILERNA Online

Tipos de servidores

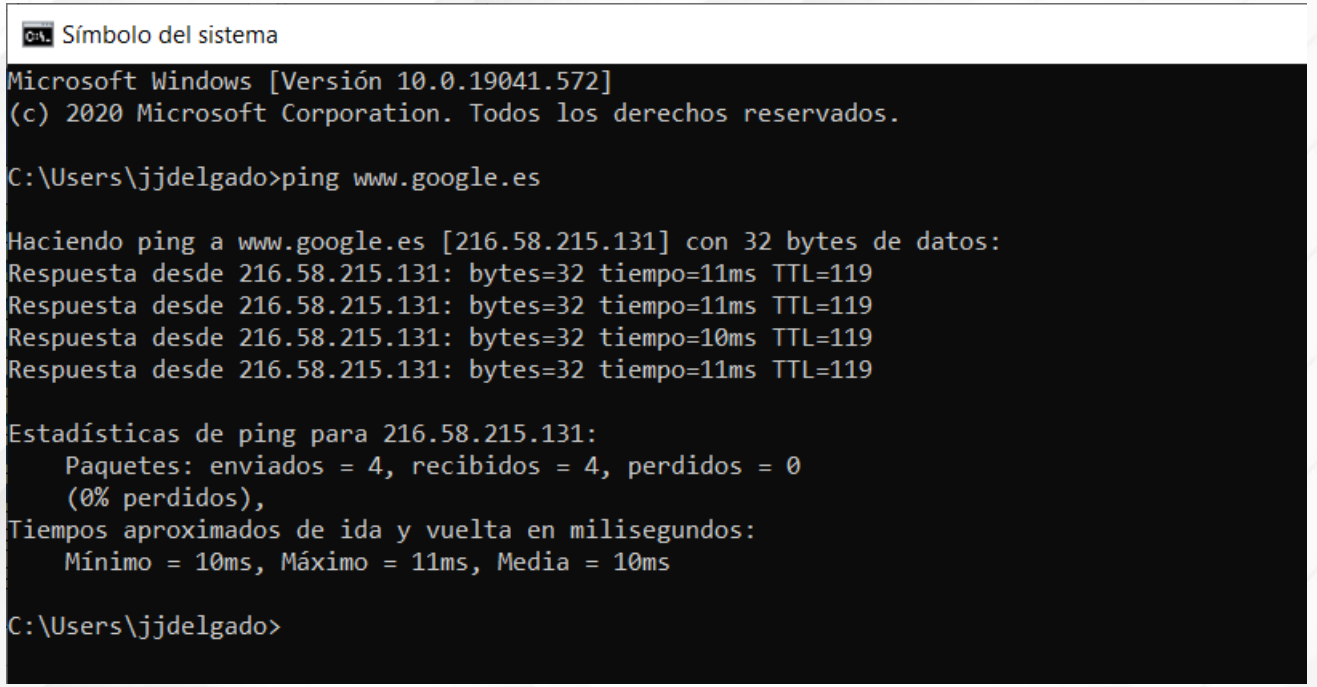
SERVIDOR DEDICADO

- Puede estar en la empresa o en CPD
- Mayor seguridad
  Porque si cae un servidor compartido por culpa de una página que tiene poca seguridad, caerán todas las demás que estén en el mismo servidor.
- No 24/7
- Poco mantenimiento

ILERNA Online

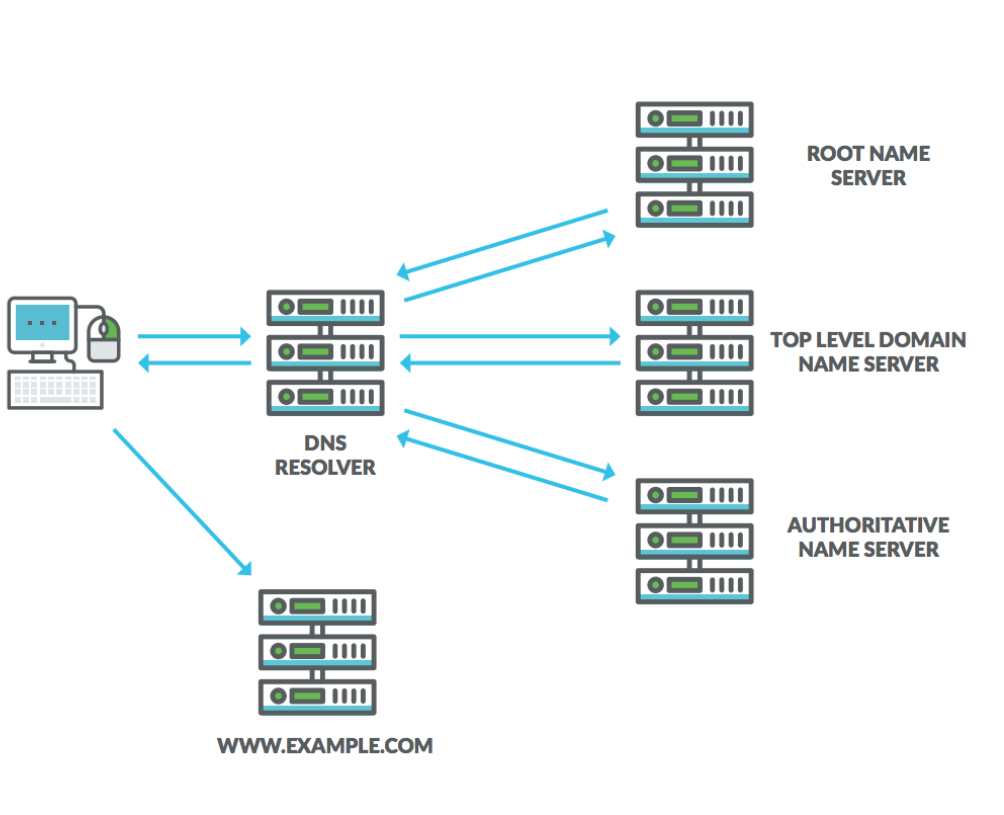
DNS (Domain name System)

- Traducción de nombres a IP, todo el contenido que viaja por la red utiliza IP.



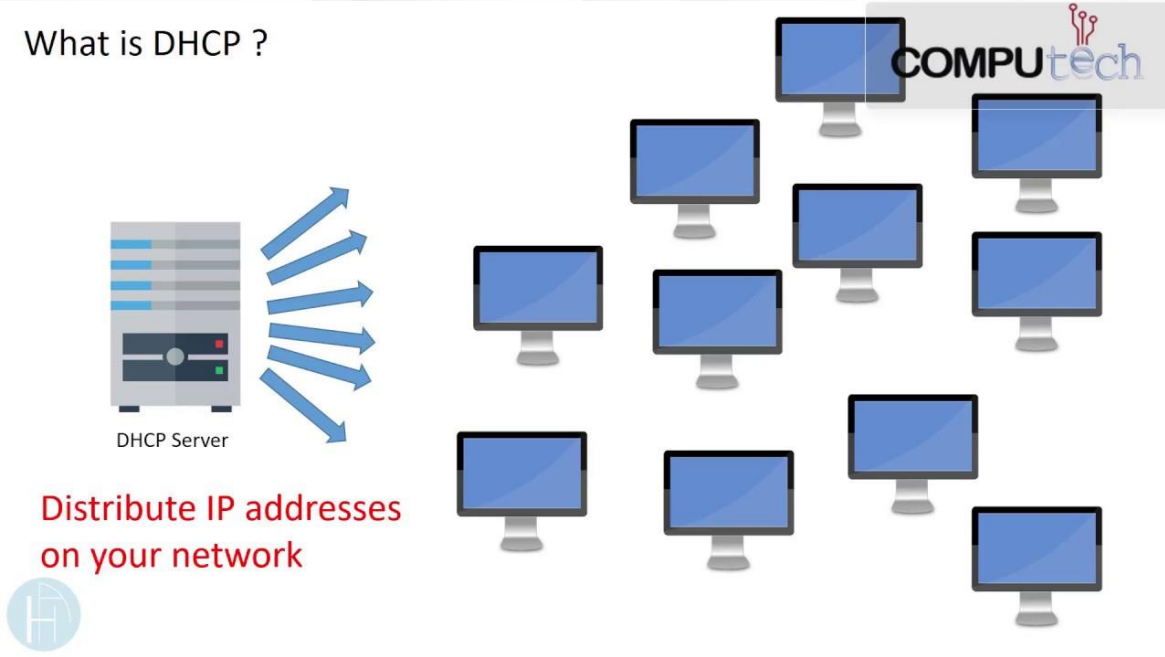
DNS (Domain name System)

- En todo el mundo existen unas DNS especiales que se encargan de distribuir el tráfico de red mundial
- Están repartidas siguiendo una estructura jerárquica
  - Dominios .com
  - Dominios .net
  - Dominios .org



DHCP (Dynamic Host Configuration Protocol)

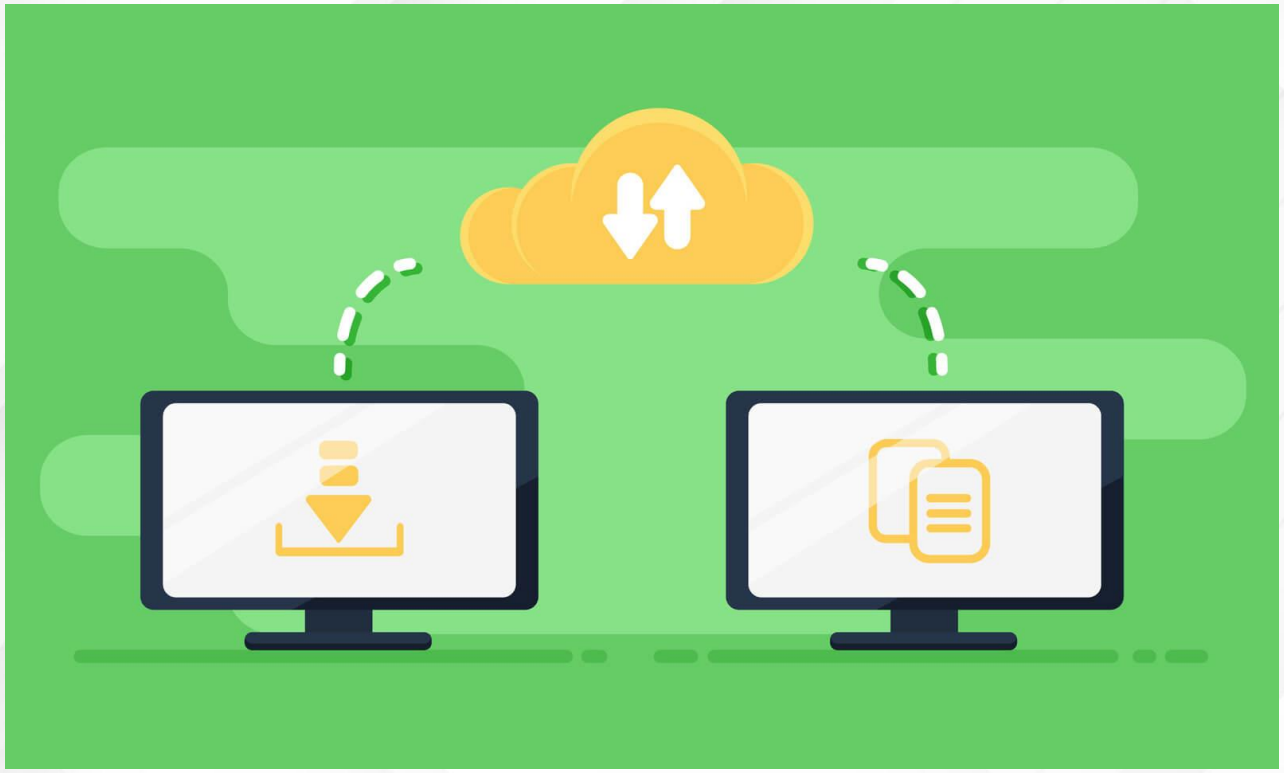
- Permite asignar una IP válida a aquellos equipos que lo soliciten
- Servidor DHC
- Peticiones Broadcast



ILERNA Online

FTP (File Transfer Protocol)

- Protocolo de Transferencia de datos, utilizado para enviar ficheros de un cliente a un servidor
- SFTP: Variante del FTP con servicio de transferencia encriptada, más segura



HTTP (Hyper Text Transfer Protocol)

- Protocolo más conocido y usado para el intercambio de información en internet (World Wide Web).

NFS (Nework File System)

- Protocolo que permite que distintos equipos que forman parte de una red puedan acceder a ficheros como si estuvieran almacenados de forma local en el equipo.

SMTP (Simple Mail Transfer Protocol)

- Protocolo simple de transferencia de correo electrónico.
  - POP e IMAP son protocolos para recibir correos electrónicos.

TELNET (Telecommunication Network)

- Protocolo que permite acceder a otro equipo remoto a través de la terminal. Actualmente se usa SSH que permite conexiones cifradas.

FTP

ILERNA Online

File Transfer Protocol

Protocolo de Transferencia de datos, utilizado para enviar ficheros de un cliente a un servidor.
SFTP: Variante del FTP con servicio de transferencia encriptada

Librerías necesarias:

- Apache Commons net
- Commons io 2.6

http://commons.apache.org/proper/commons-net/download_net.cgi

ILERNA Online

File Transfer Protocol

https://commons.apache.org/proper/commons-net/apidocs/org/apache/

org.apache.commons.net.ftp

Class FTPClient

java.lang.Object
  org.apache.commons.net.SocketClient
    org.apache.commons.net.ftp.FTP
      org.apache.commons.net.ftp.FTPClient

All Implemented Interfaces:

`Configurable`

Direct Known Subclasses:

`FTPHTTPClient, FTPSClient`

```
public class FTPClient
extends FTP
implements Configurable
```

ILERNA Online

File Transfer Protocol

Conexión al servidor:

```
private String server = “192.168.1.131";
private int port = 21;
private String user = “user";
private String password = "12345";
```

ILERNA Online

File Transfer Protocol

Conexión al servidor:

```
ftp = new FTPClient();
System.out.println("Iniciando conexión al servidor " + server);
ftp.connect(server, port);
int reply = ftp.getReplyCode();
if (!FTPReply.isPositiveCompletion(reply)) {
    ftp.disconnect();
    throw new IOException("Exception in connecting to FTP Server");
}
System.out.println("Servidor En línea");
System.out.println("Iniciando sesión en el servidor.");
ftp.enterLocalPassiveMode();
ftp.login(user, password);
```

En modo pasivo es siempre el programa cliente quien inicia la conexión con el servidor

ILERNA Online

File Transfer Protocol

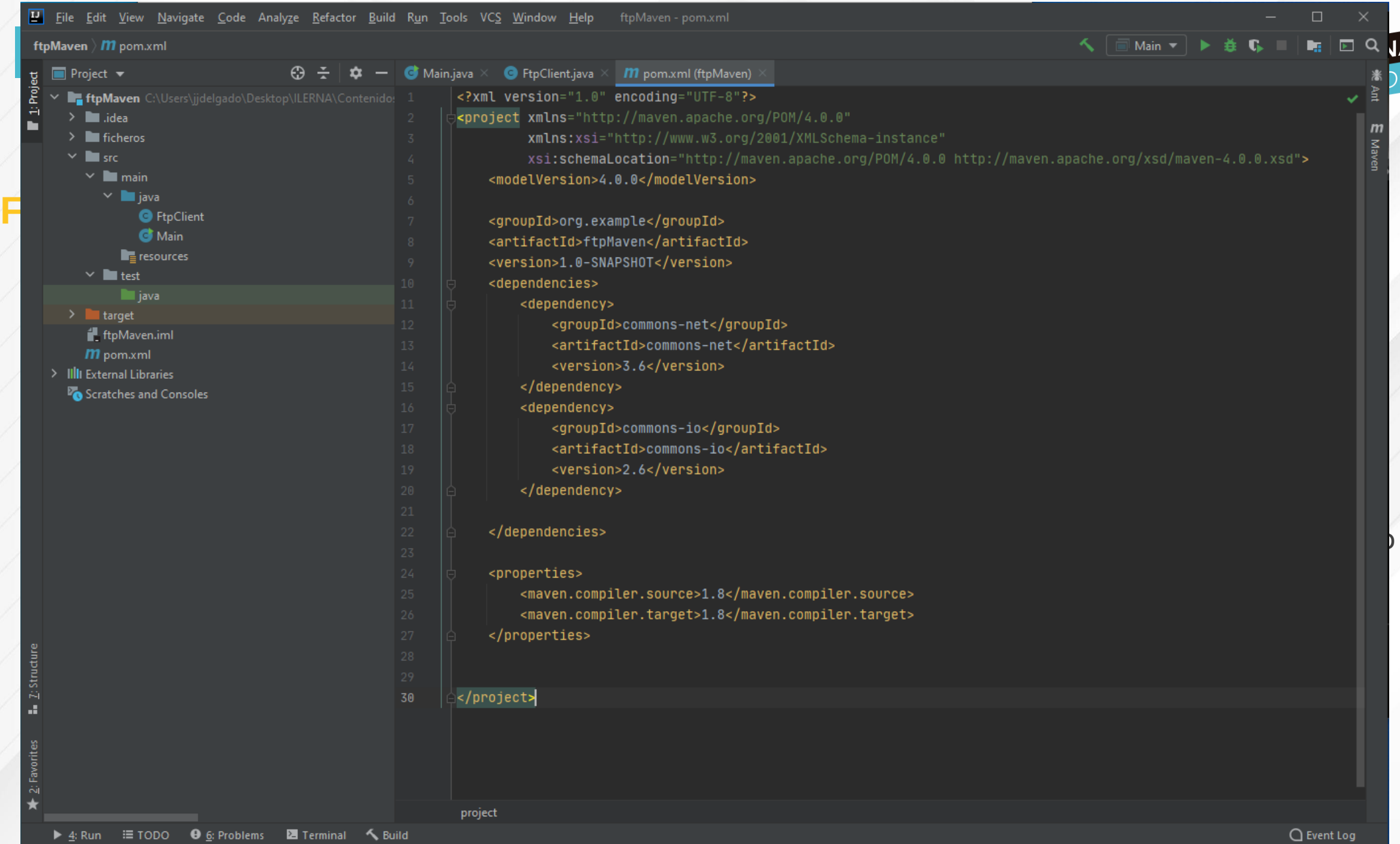
Recuperación de datos al servidor:

```
String remoteFile = "files.txt";
File downloadFile = new
File("C:\\Users\\jjdelgado\\Desktop\\Intellij\\M09\\ftp\\files\\files.txt");
OutputStream outputStream = new BufferedOutputStream(new FileOutputStream(downloadFile));
InputStream inputStream = ftp.retrieveFileStream(remoteFile);
byte[] bytesArray = new byte[4096];
int bytesRead = -1;
while ((bytesRead = inputStream.read(bytesArray)) != -1) {
    outputStream.write(bytesArray, 0, bytesRead);
}
```

ILERNA Online

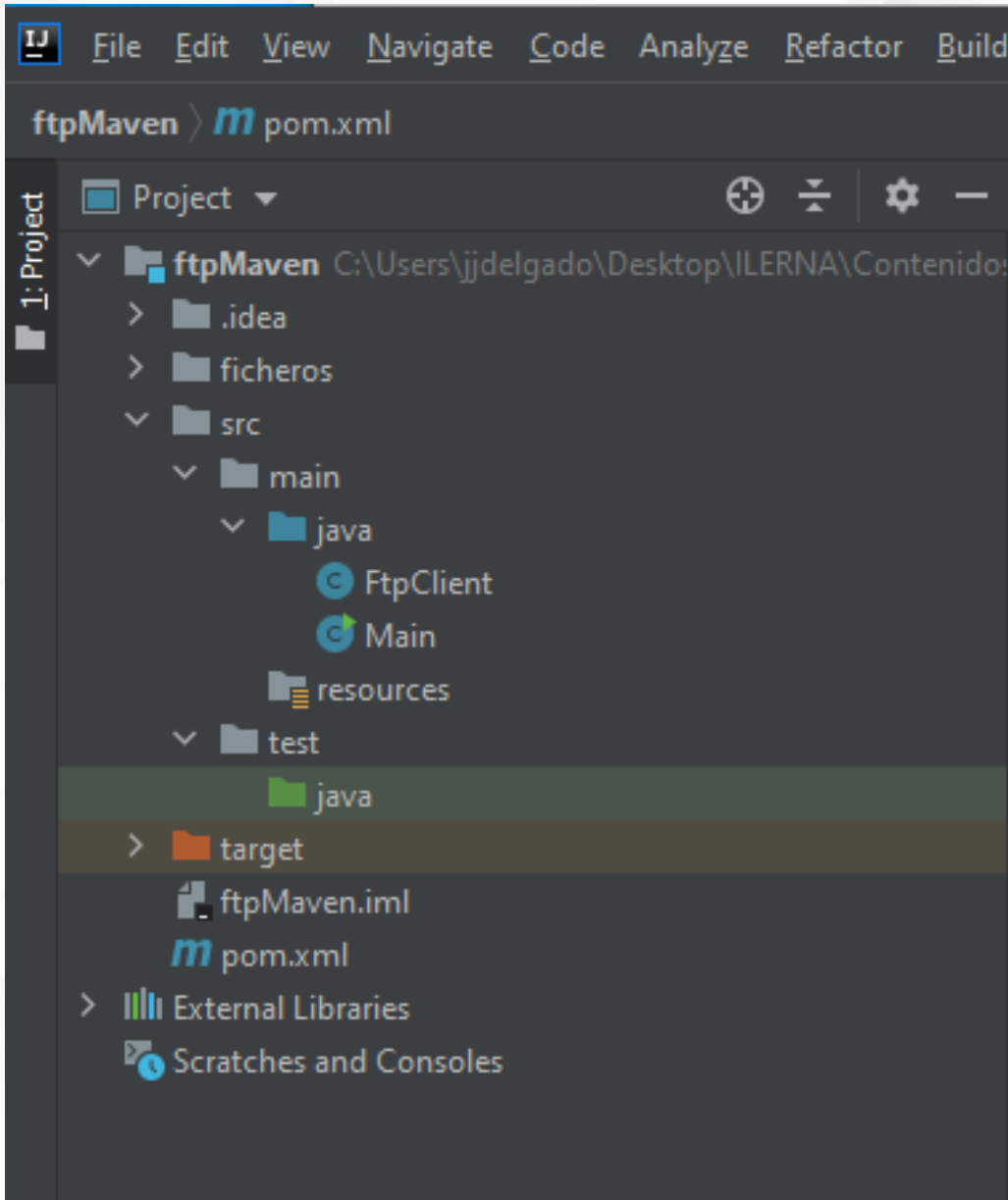
File Transfer Protocol

- Vamos a empezar el Proyecto con IntelliJ IDEA
- Proyecto Maven
- Maven es un repositorio de librerías que podemos descargar de internet
- La aplicación se conecta automáticamente al repositorio y se descarga las librerías que necesita
- Esto hace que la aplicación sea más portable y podamos compartirla sin necesidad de pasar junto a la aplicación, las librerías o .jar
- Al crear el proyecto, se crea la estructura de directorios y el fichero de configuración POM.XML


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help ftpMaven - pom.xml
ftpMaven pom.xml
Main
Project
ftpMaven C:\Users\jjdelgado\Desktop\ILERNA\Contenido
.idea
ficheros
src
main
java
FtpClient
Main
resources
test
java
target
ftpMaven.iml
pom.xml
External Libraries
Scratches and Consoles
Main.java
FtpClient.java
pom.xml (ftpMaven)
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.example</groupId>
<artifactId>ftpMaven</artifactId>
<version>1.0-SNAPSHOT</version>
<dependencies>
<dependency>
<groupId>commons-net</groupId>
<artifactId>commons-net</artifactId>
<version>3.6</version>
</dependency>
<dependency>
<groupId>commons-io</groupId>
<artifactId>commons-io</artifactId>
<version>2.6</version>
</dependency>
</dependencies>
<properties>
<maven.compiler.source>1.8</maven.compiler.source>
<maven.compiler.target>1.8</maven.compiler.target>
</properties>
</project>
project
4: Run
TODO
6: Problems
Terminal
Build
Event Log
1: Project
2: Favorites
7: Structure
Ant
Maven

File Transfer Protocol

- FICHEROS
- FtpClient. Java
- Instancia de FTPClient
- Importarlas de apache.commons
- Necesitamos servidor con una IP y un puerto
- Necesitaremos usuario y contraseña
- Función connect()



ILERNA Online

File Transfer Protocol

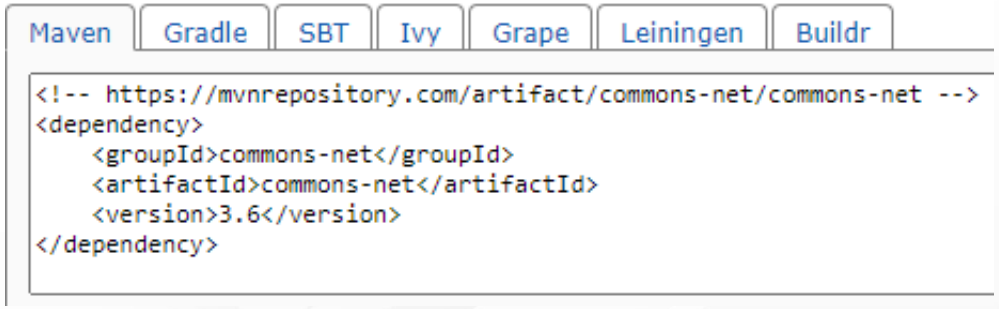
- FICHERO POM.XML
- Fichero XML con información de Maven y configuración del proyecto (nombre y versión)
- Aquí añadiremos las librerías <dependencias></dependencias>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>ftpMaven</artifactId>
    <version>1.0-SNAPSHOT</version>


</project>
```

ILERNA Online

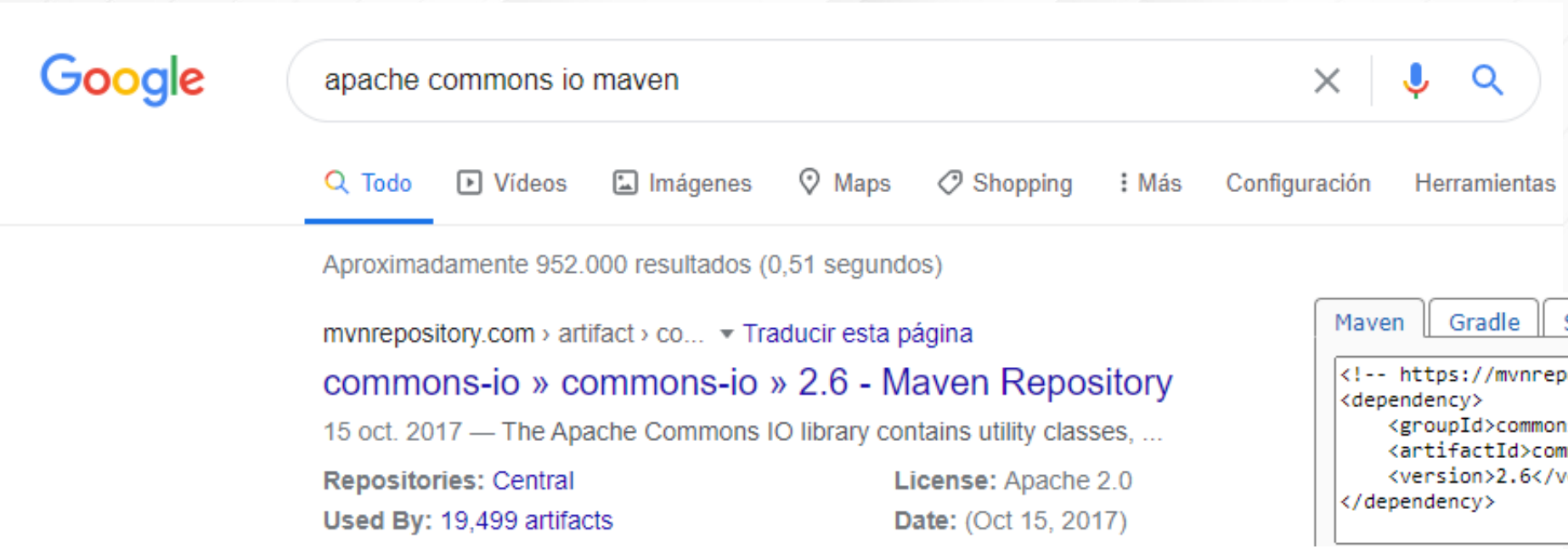
File Transfer Protocol

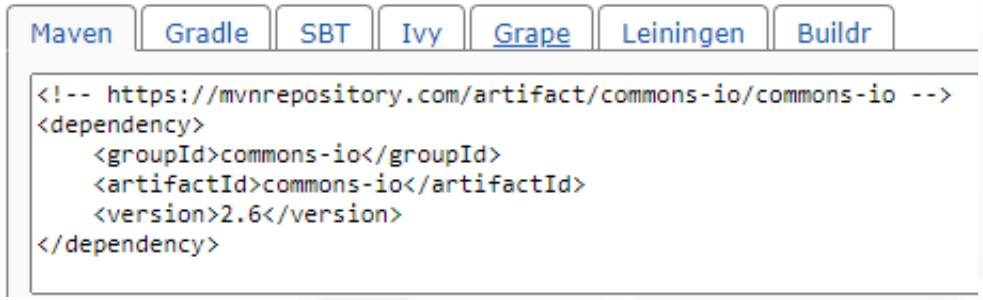
- ¿Qué librerías necesito para trabajar con FTPClient y Maven?
- Las buscamos en el repositorio de Maven: Apache Commons NET



File Transfer Protocol

- ¿Qué librerías necesito para trabajar con FTPClient y Maven?
- Ahora buscamos la segunda librería, la de Commons io. Repetimos el proceso





File Trans

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>ftpMaven</artifactId>
    <version>1.0-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>commons-net</groupId>
            <artifactId>commons-net</artifactId>
            <version>3.6</version>
        </dependency>
        <dependency>
            <groupId>commons-io</groupId>
            <artifactId>commons-io</artifactId>
            <version>2.6</version>
        </dependency>

    </dependencies>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>


</project>
```

ILERNA Online

File Transfer Protocol

- Podemos hacer el proyecto sin necesidad de usar Maven, esto es de la forma clásica en Java también desde IntelliJ IDEA
- Creamos un proyecto en Java
- Creamos carpeta lib y dentro incluiremos las librerías que descararemos de internet (copiar y pegar en este directorio)
    - commons-io-2.6.jar
    - commons-net-3.0.1.jar
- Una vez copiados: File /Project Structure / Libraries / tipo Java y buscamos las que hemos añadido al directorio

TELNET

ILERNA Online

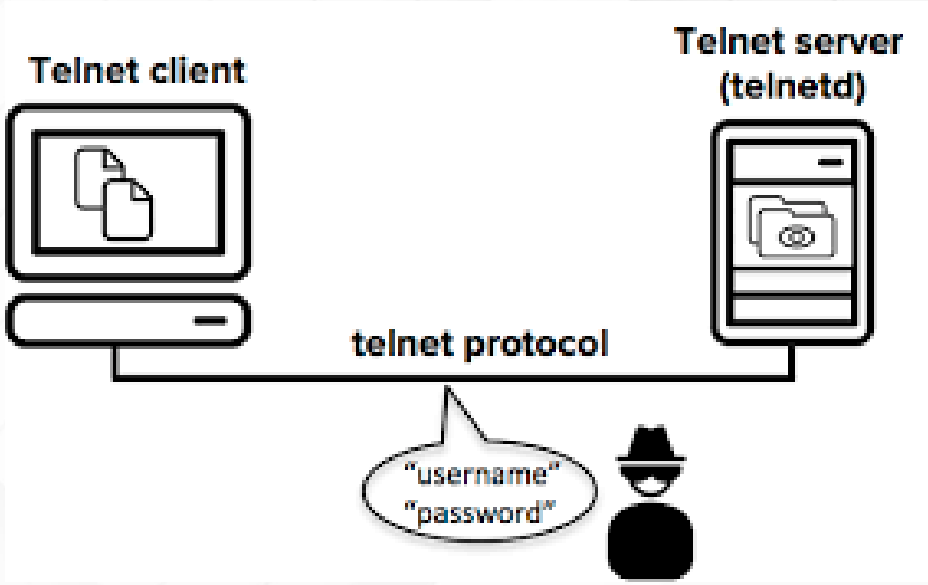
Telecommunication Network

Protocolo que permite acceder a otro equipo remoto a través de la terminal, de forma que conecta dos ordenadores entre sí usando la red.

Actualmente se usa SSH que permite conexiones cifradas

Librerías necesarias:

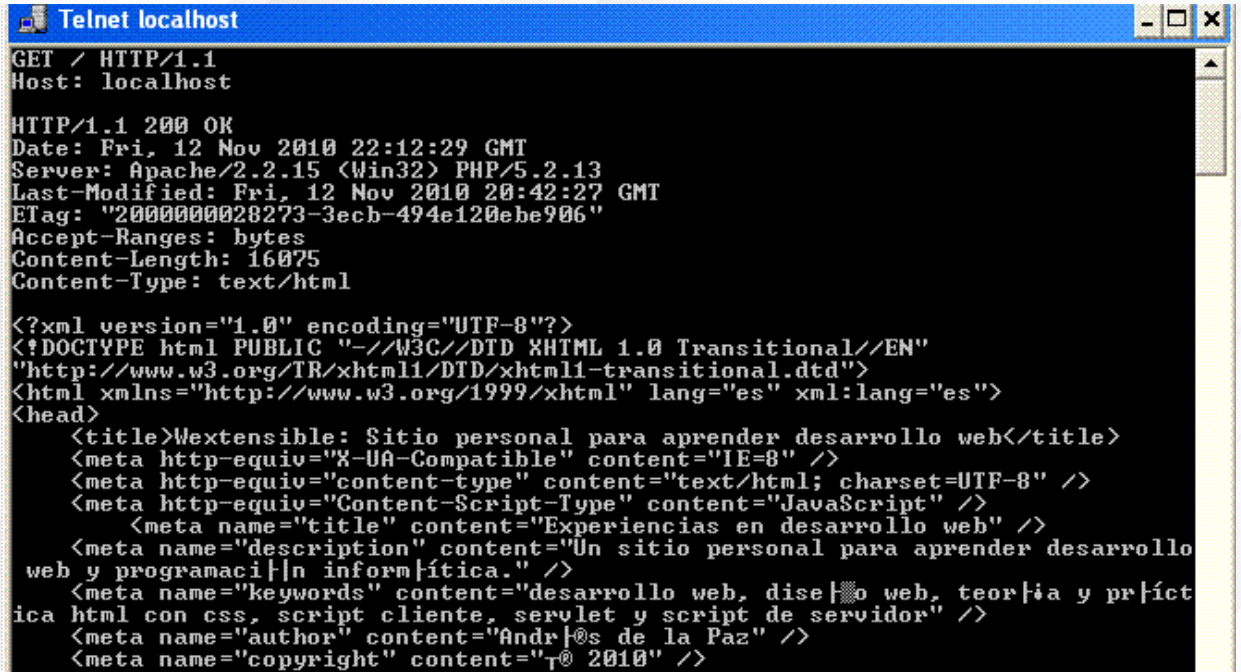
- Apache Commons net



ILERNA Online

Telecommunication Network

Telnet no cuenta con entorno gráfico de ventanas, sino que funciona en modo terminal y es una herramienta muy recomendada solucionar problemas en remoto ya que al conectarse al equipo, toma el control sobre él



ILERNA Online

Telecommunication Network

Características:

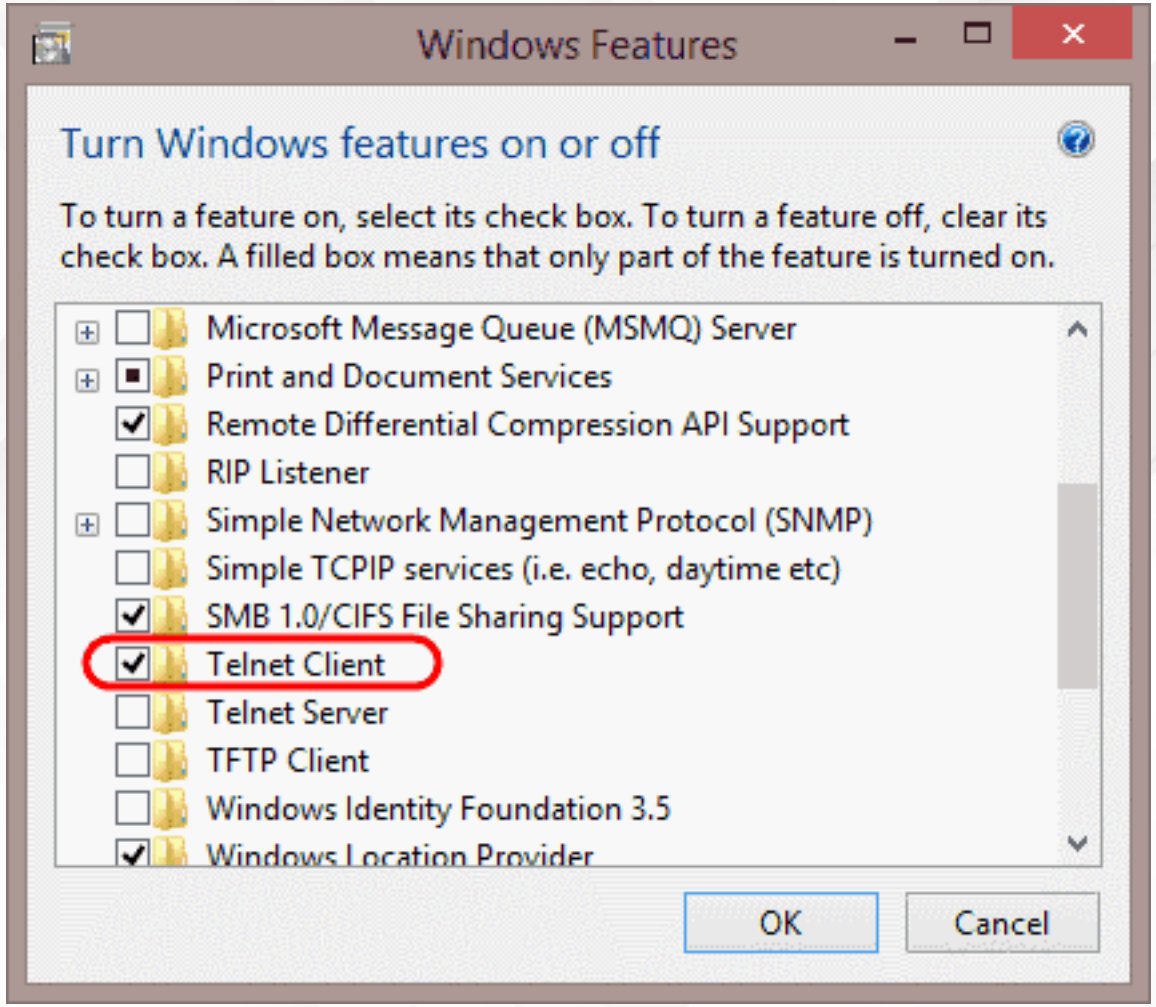
- Utiliza el puerto 23
- Protocolo de nivel de Aplicación
- Trabaja sobre TCP/IP
- Control remoto para solucionar problemas
- Consultar datos a distancia
- En consola: telnet 192.168.1.145 para conectarte al equipo en esta IP
- Puede requerir Usuario y Contraseña

ILERNA Online

Telecommunication Network

Habilitar telnet Windows 10

- Botón derecho encima del botón de Windows
- Programas y Características
- Activar o desactivar características de Windows
- Marcar la casilla del cliente de Telnet
- Aceptar



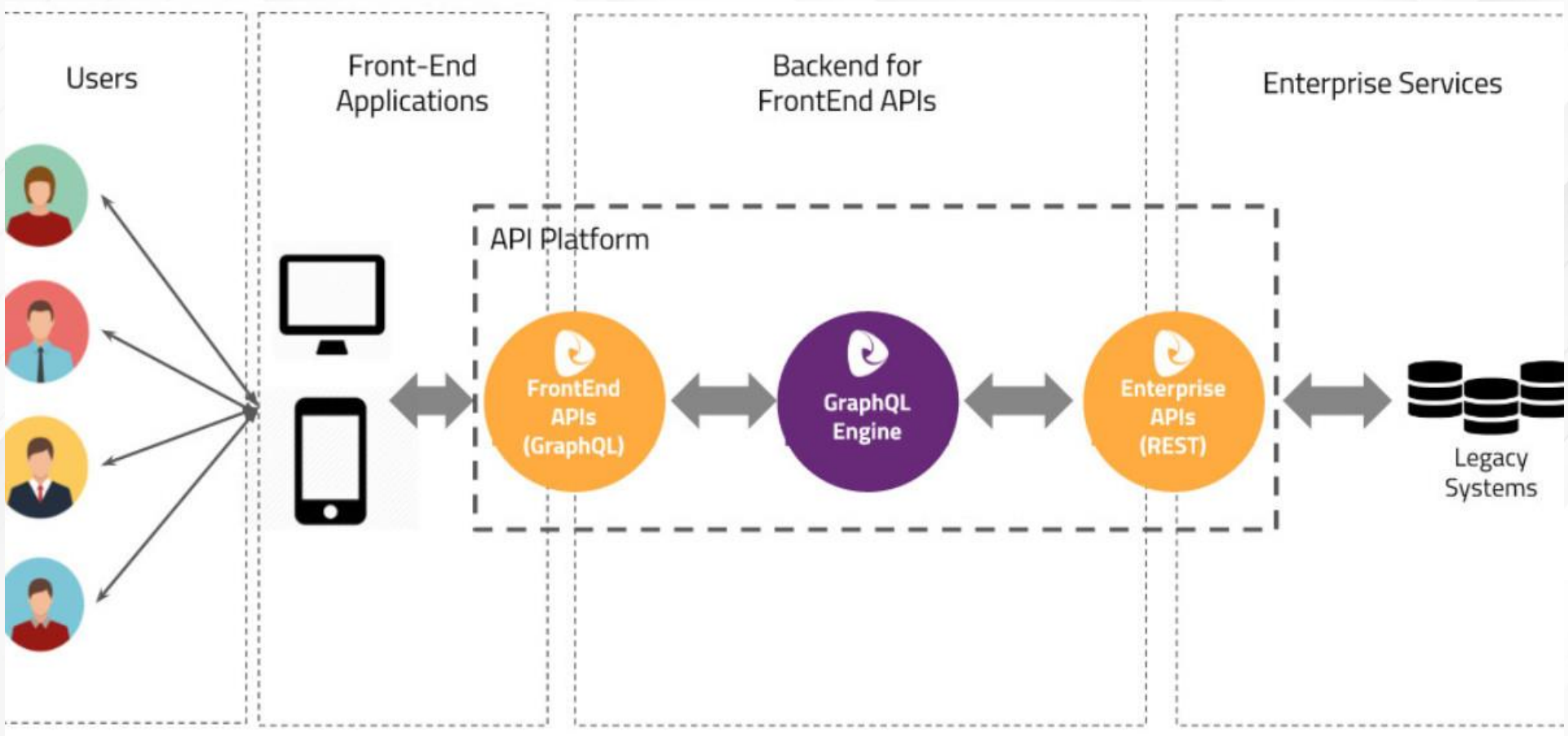
ILERNA Online

Telecommunication Network

```
public TelnetCliente(String host) {
    this.host = host;
    telnet = new TelnetClient();
}

public void connect() throws IOException {
    telnet.connect(host);
}

public void disconnect() {
    try {
        System.out.print("Cerrando telnet . . .");
        telnet.disconnect();
    } catch (IOException e) {
        System.out.println("Error occurred while telnet disconnection " + e);
    }
}
```

Servicios Web

CÓMO FUNCIONA LA CONEXIÓN



CÓMO FUNCIONA LA CONEXIÓN

Front End

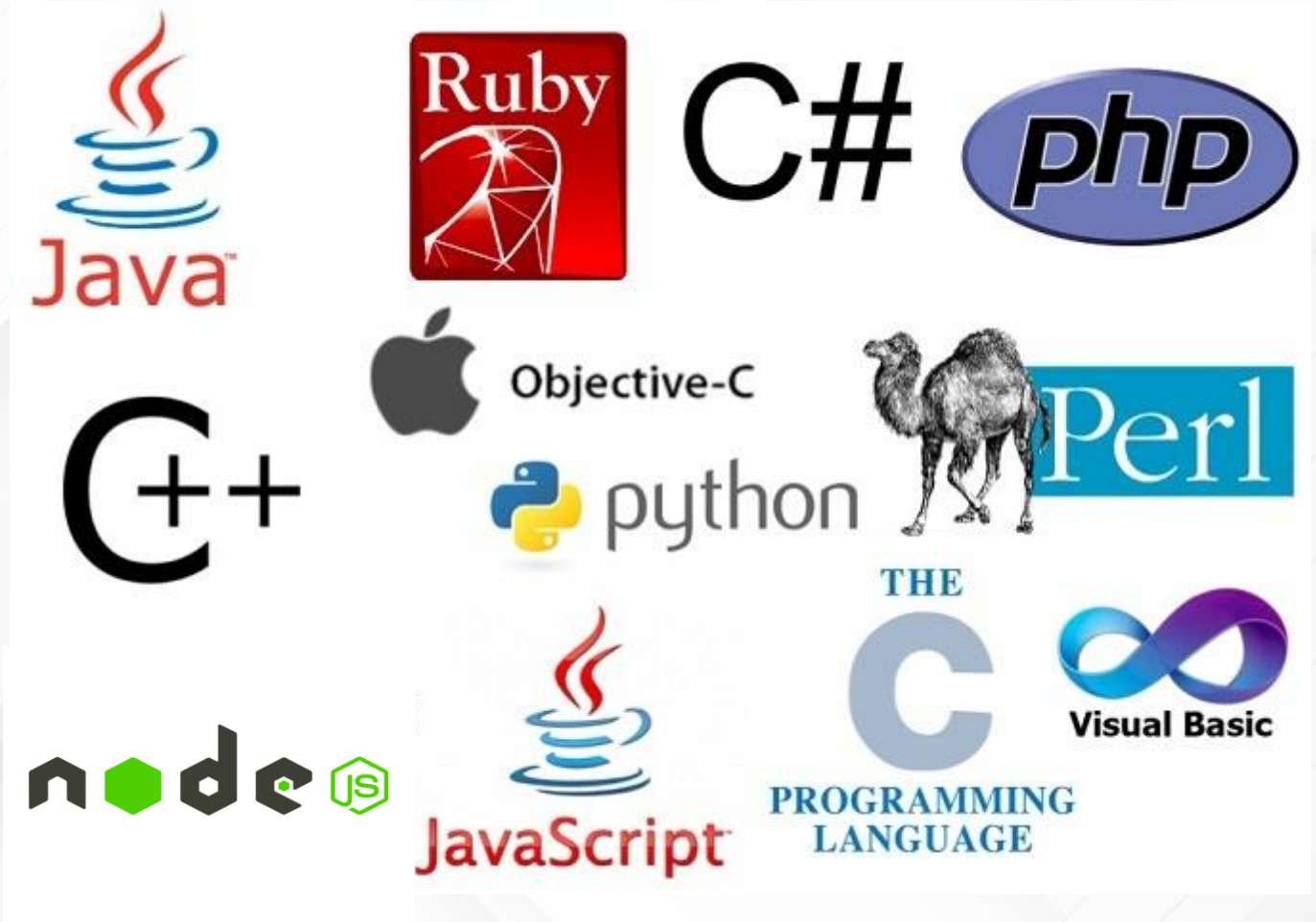
- Markup and web languages such as HTML, CSS and Javascript
- Asynchronous requests and Ajax
- Specialized web editing software
- Image editing
- Accessibility
- Cross-browser issues
- Search engine optimisation

Back End

- Programming and scripting such as Python, Ruby and/or Perl
- Server architecture
- Database administration
- Scalability
- Security
- Data transformation
- Backup

ILERNA Online

LENGUAJES DE PROGRAMACIÓN BACKEND



ILERNA Online

CÓMO SE LANZAN LAS PETICIONES

- Lanzaremos las peticiones desde el Frontend al Backend en formato URL, usando el protocolo HTTP
- Dicho protocolo permite usar un conjunto de operaciones llamadas: CRUD
    - INSERCIÓN – create
    - ELIMINACIÓN - remove
    - ACTUALIZACIÓN - update
    - CONSULTA DE DATOS – data query

Por tanto necesitaremos un servidor que nos permita realizar estas operaciones

ILERNA Online

CÓMO SE LANZAN LAS PETICIONES

- Vamos a necesitar un servidor que nos permita realizar estas operaciones
  - INSERCIÓN => POST
  - ELIMINACIÓN => DELETE
  - ACTUALIZACIÓN => PUT
  - CONSULTA DE DATOS => GET

- A partir de aquí ya podríamos transmitir datos

ILERNA Online

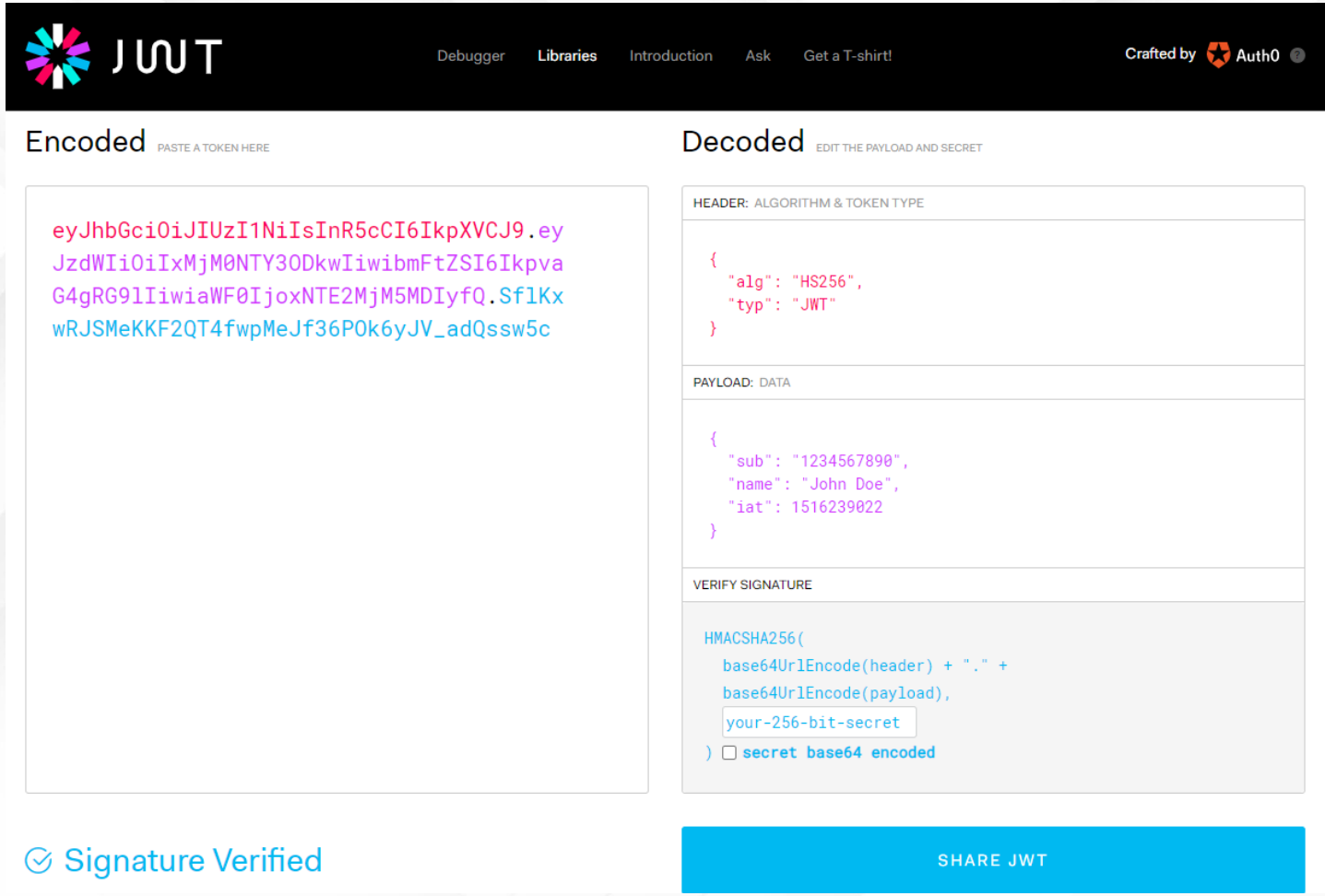
CÓMO SE LANZAN LAS PETICIONES

- Ventajas de usar un Backend
  - Oculta cómo se trabaja con la información a nivel de servidor
  - Oculta cómo es la conexión a la BD
  - Protege mucho más a los usuarios
- Cómo nos conectamos al Backend
  - Usuario y contraseña de administrador en el servidor
  - Se le otorga un TOKEN
  - Restringe el acceso al Backend
  - Proteger con CAPTCHA

ILERNA Online

CÓMO SE LANZAN LAS PETICIONES

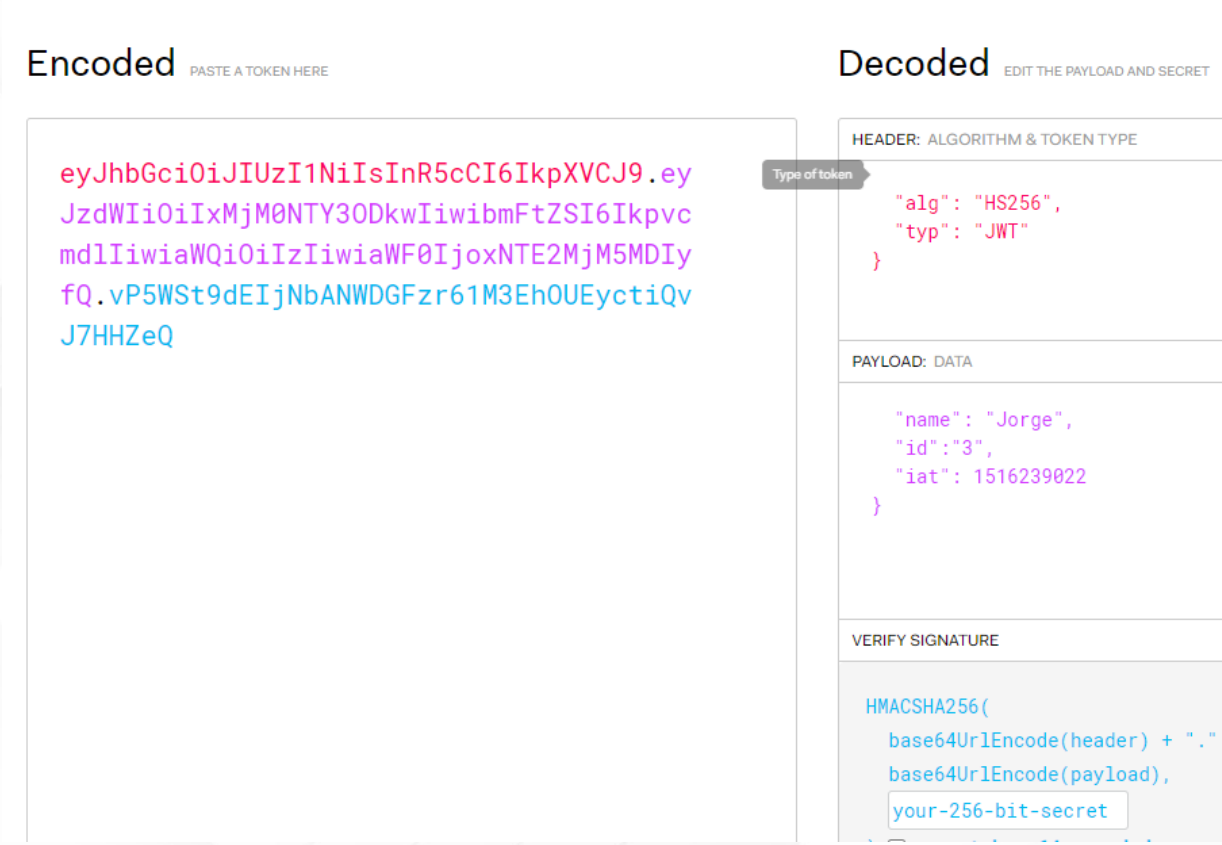
- JWT Web TOKEN
- https://jwt.io
- JSON web token
- Recibiremos un código como el que se muestra
- Formado por un conjunto de 3 tipos de datos:
  - Cabecera (HEADER)
  - Datos (PAYLOAD)
  - Firma (VERIFY SIGNATURE)



ILERNA Online

CÓMO SE LANZAN LAS PETICIONES

- Podemos confiugrar los datos que necesitemos en estas cajas de texto y se actualizará la información del token automáticamente
- En el servidor se recibe el login (usuario y contraseña)
- Una vez validados con la info del servidor, generamos un nuevo token con la información del usuario NO sensible y una fecha de expiración
- Le pasamos esta info al usuario que está conectado desde el Frontend

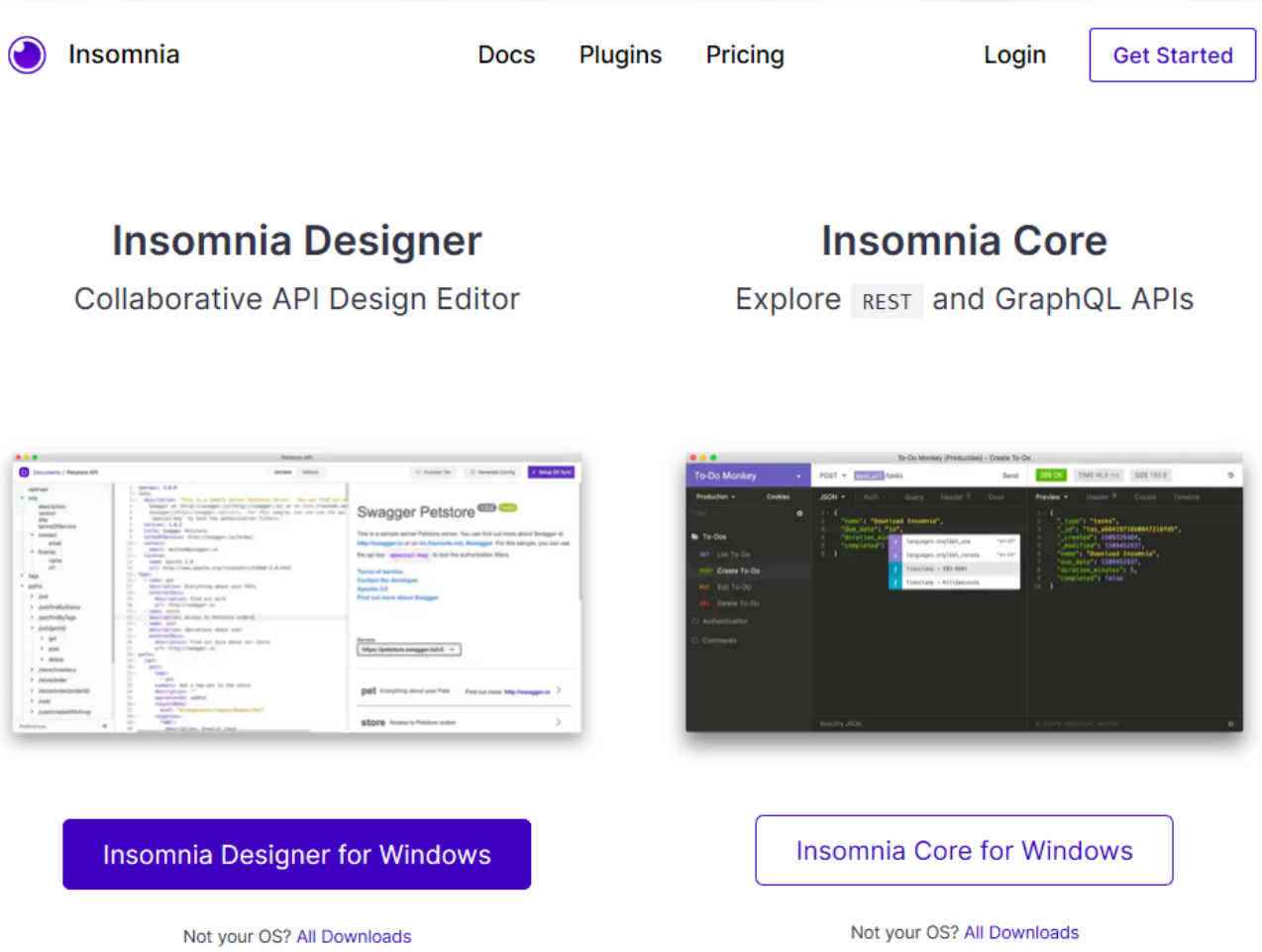


ILERNA Online

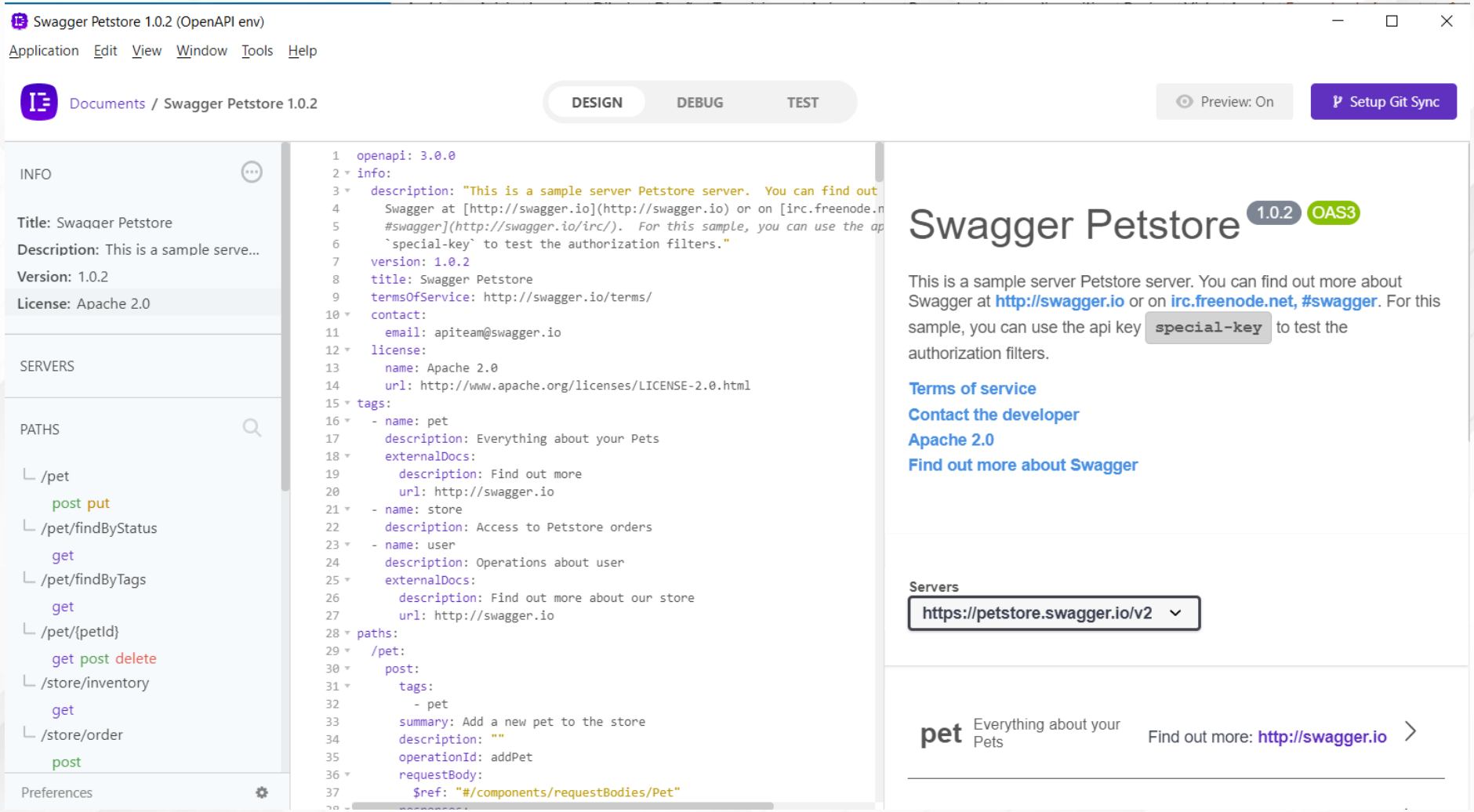
- Los servicios web son técnicas de comunicación estándares para cualquier tipo de acceo entre cliente y servidor
- Dos formas de comunicarse:
  - SOAP (Simple Object Access Protocol): Esta comunicación se hará usando el lenguaje XML. Es independiente de la plataforma y del lenguaje
  - REST (Representational State Transfer): Este tipo de comunicación permite otro tipo de mensajes como son JSON
- Las operaciones posibles son POST, GET, PUT y DELETE (CRUD)

ILERNA Online

- Programa Insomina (http://insomnia.rest/download)
- Descargar e instalar
- API de ejemplo



- API de ejemplo
- PetStore
- Ver información



ILERNA Online

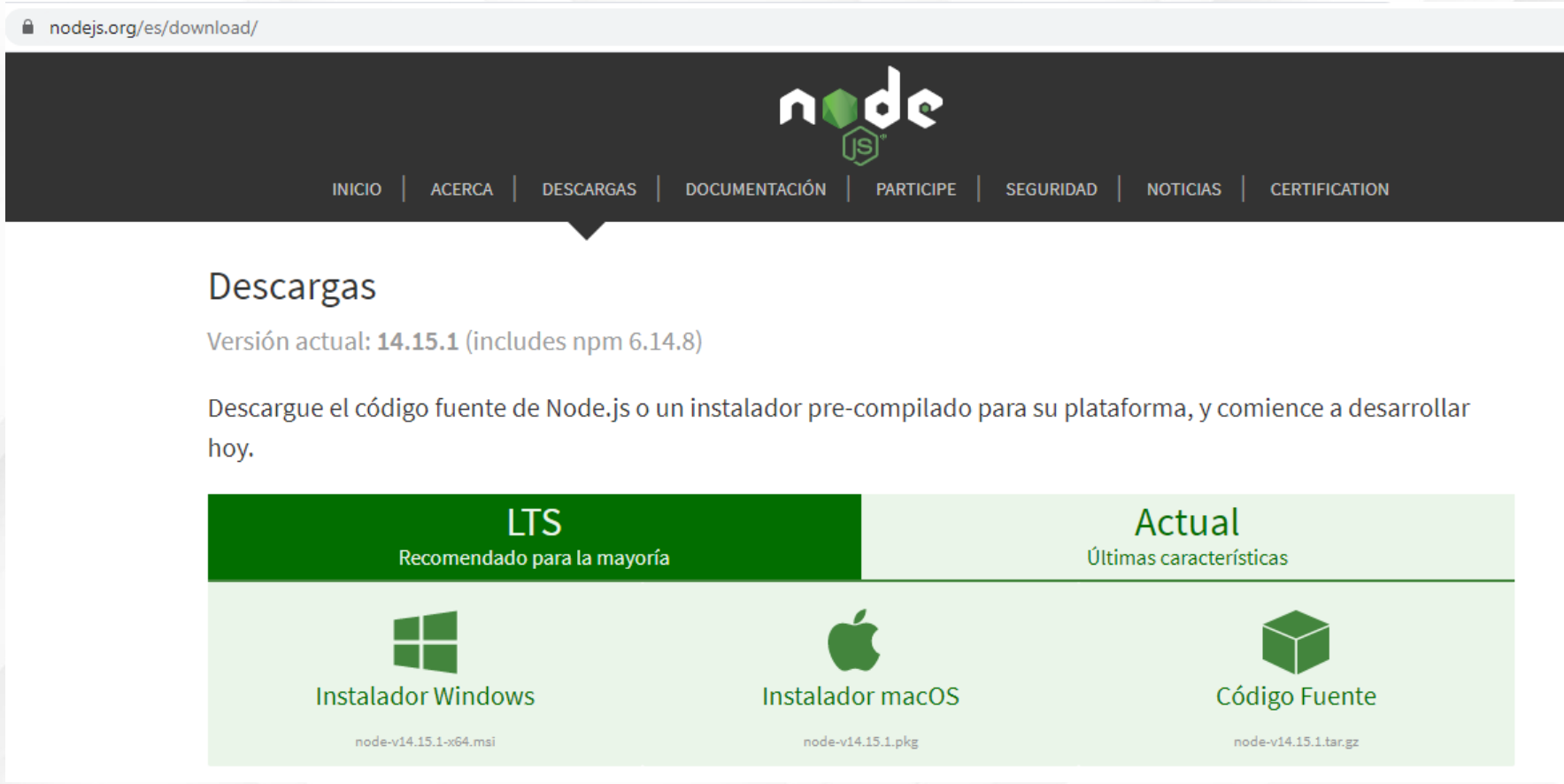
CREAR UN SERVIDOR NODE JS

- Vamos a crear un servidor muy sencillo en Node JS
- Va a responder a dos peticiones de tipo GET
- Usaremos una URL en la que pasamos datos y nos devolverá unos resultados

ILERNA Online

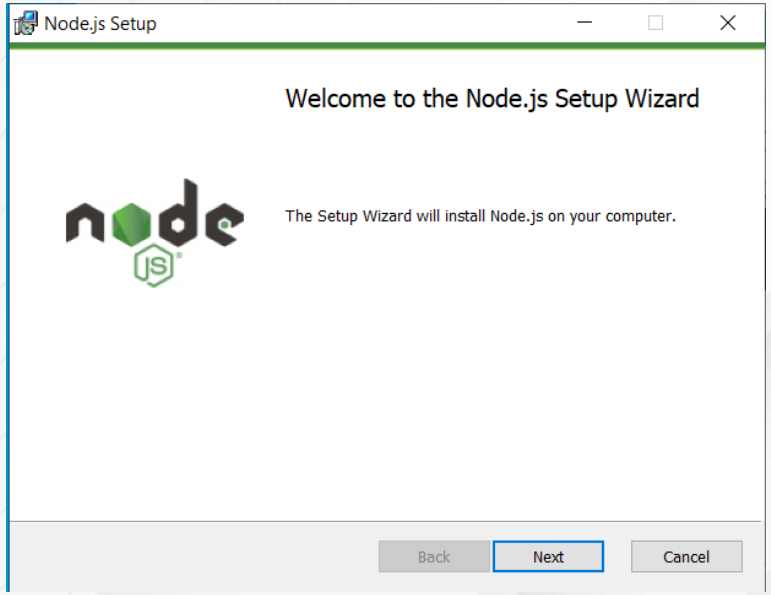
CREAR UN SERVIDOR NODE JS

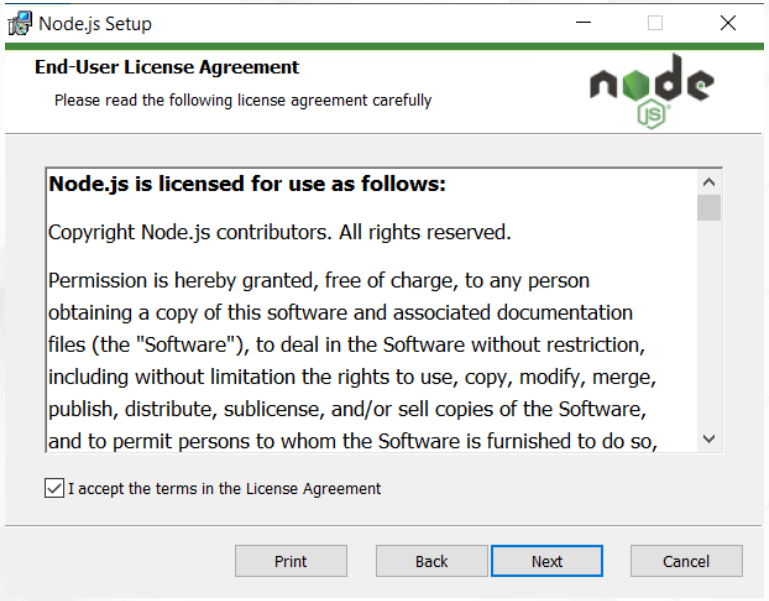
- Instalar NODE JS

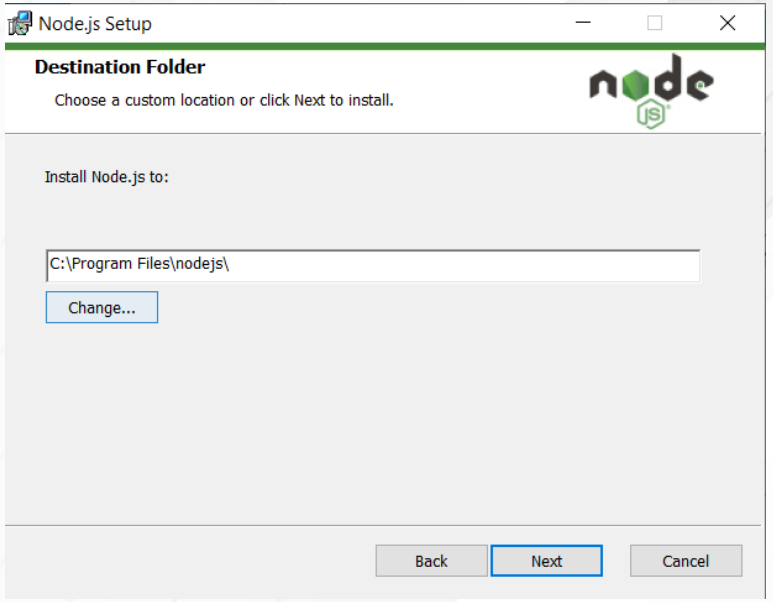


CREAR UN SERVIDOR NODE JS

- Instalar NODE JS



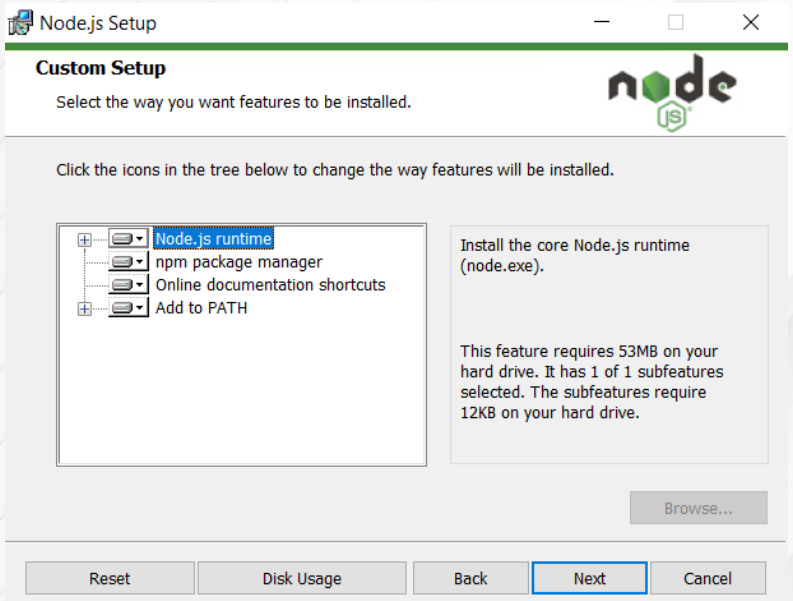


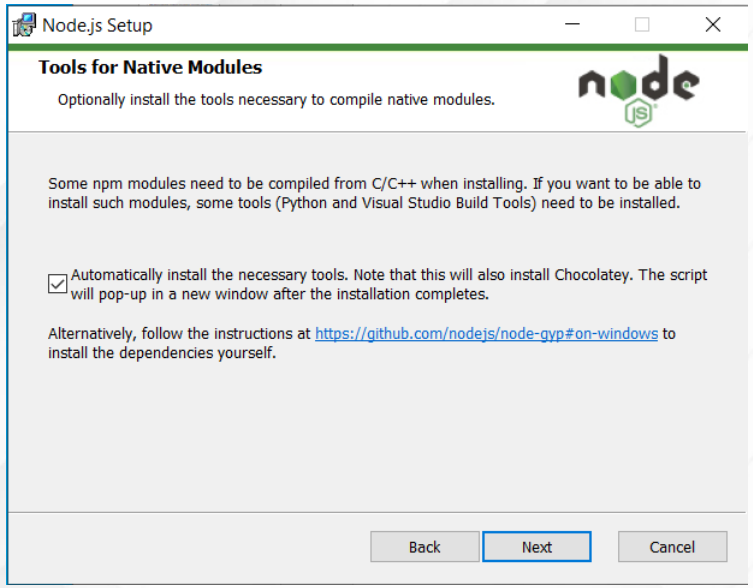


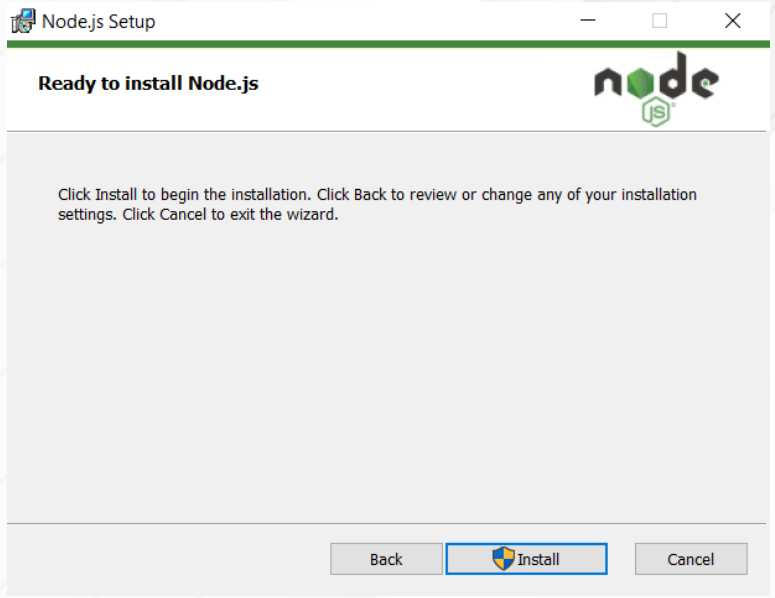
ILERNA Online

CREAR UN SERVIDOR NODE JS

- Instalar NODE JS



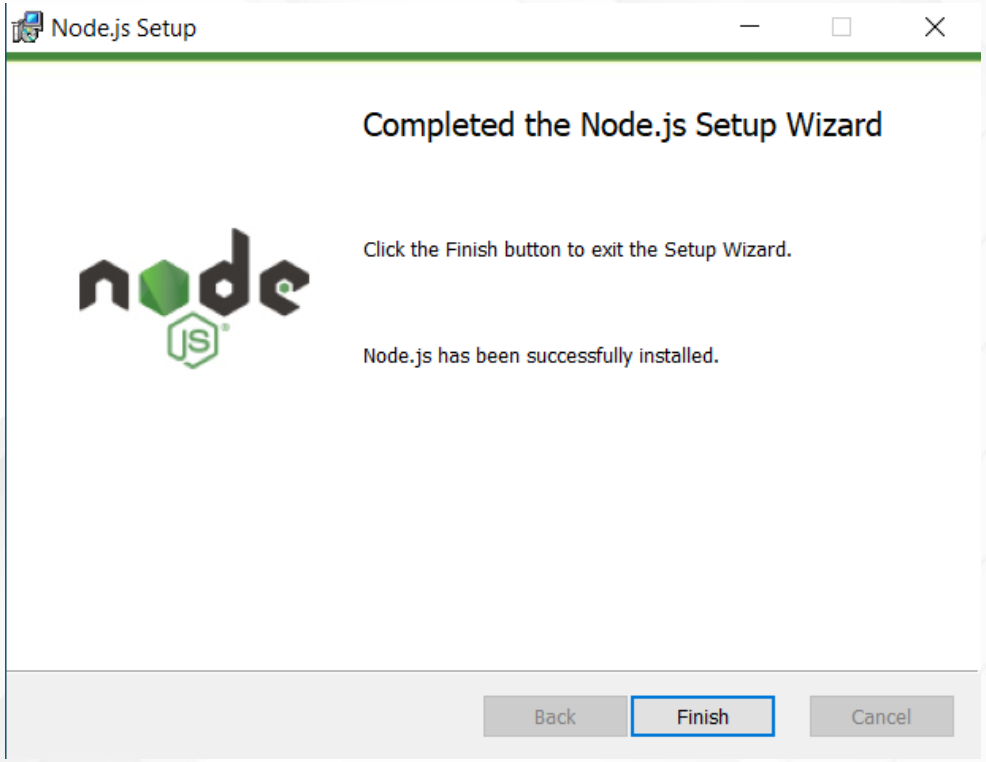




ILERNA Online

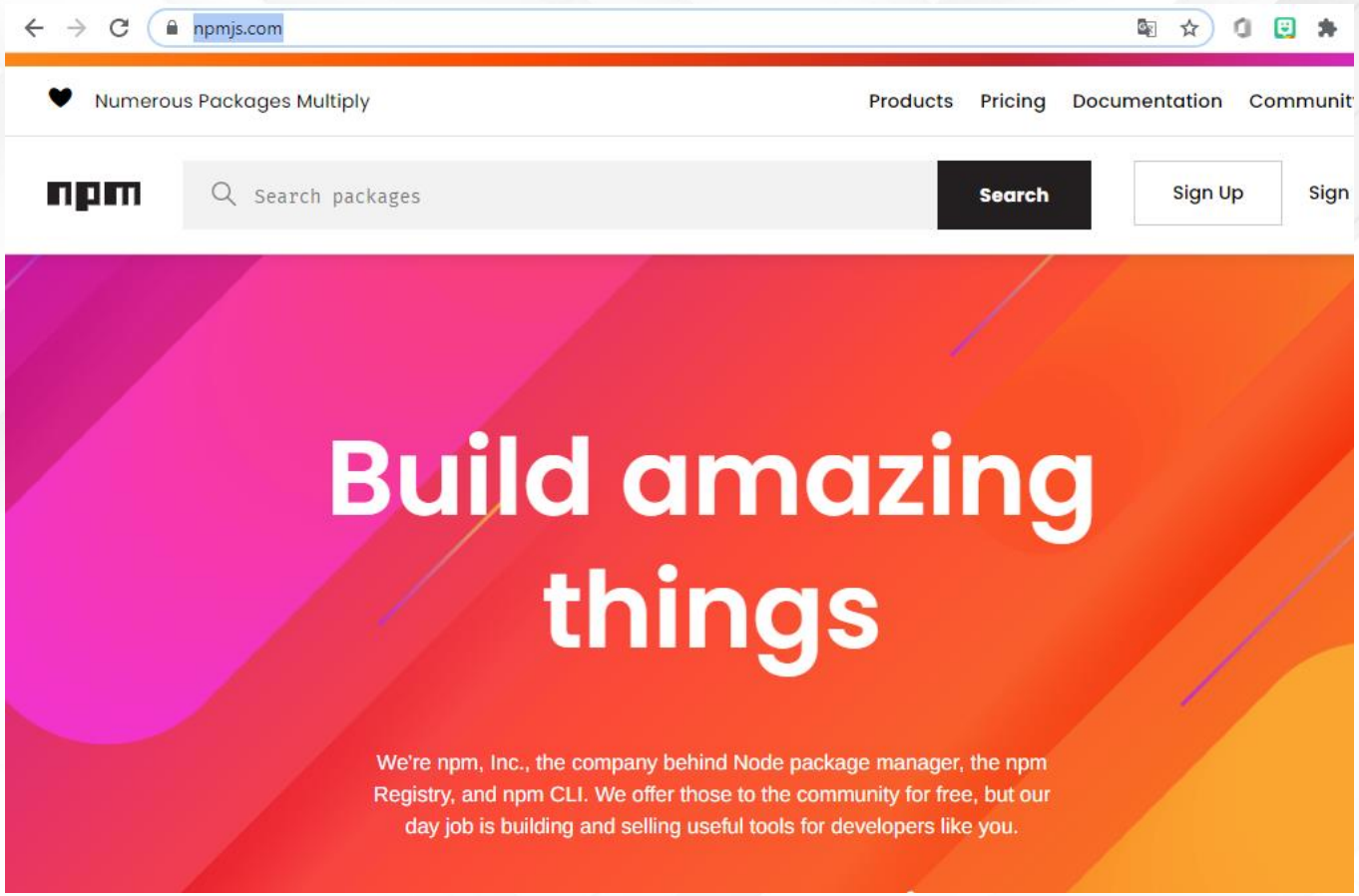
CREAR UN SERVIDOR NODE JS

- Instalar NODE JS



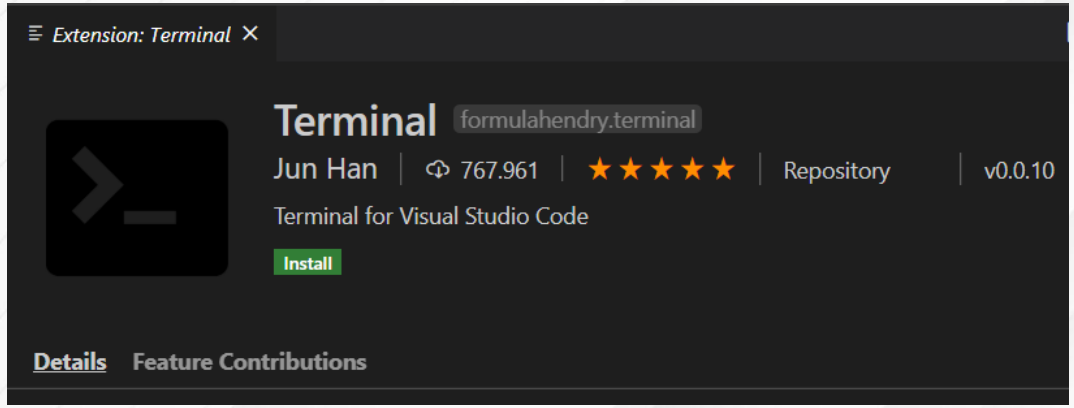
CREAR UN SERVIDOR NODE JS

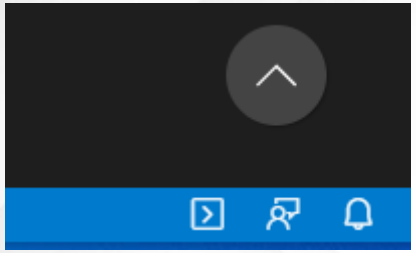
- Web oficial de librerías
- Npm (https://www.npmjs.com/)
- Al instalar Node, nos estamos instalando el repositorio NPM en nuestro ordenador
- Podemos consultar en la web todas las librerías que ya podremos usar en Node en nuestros proyectos

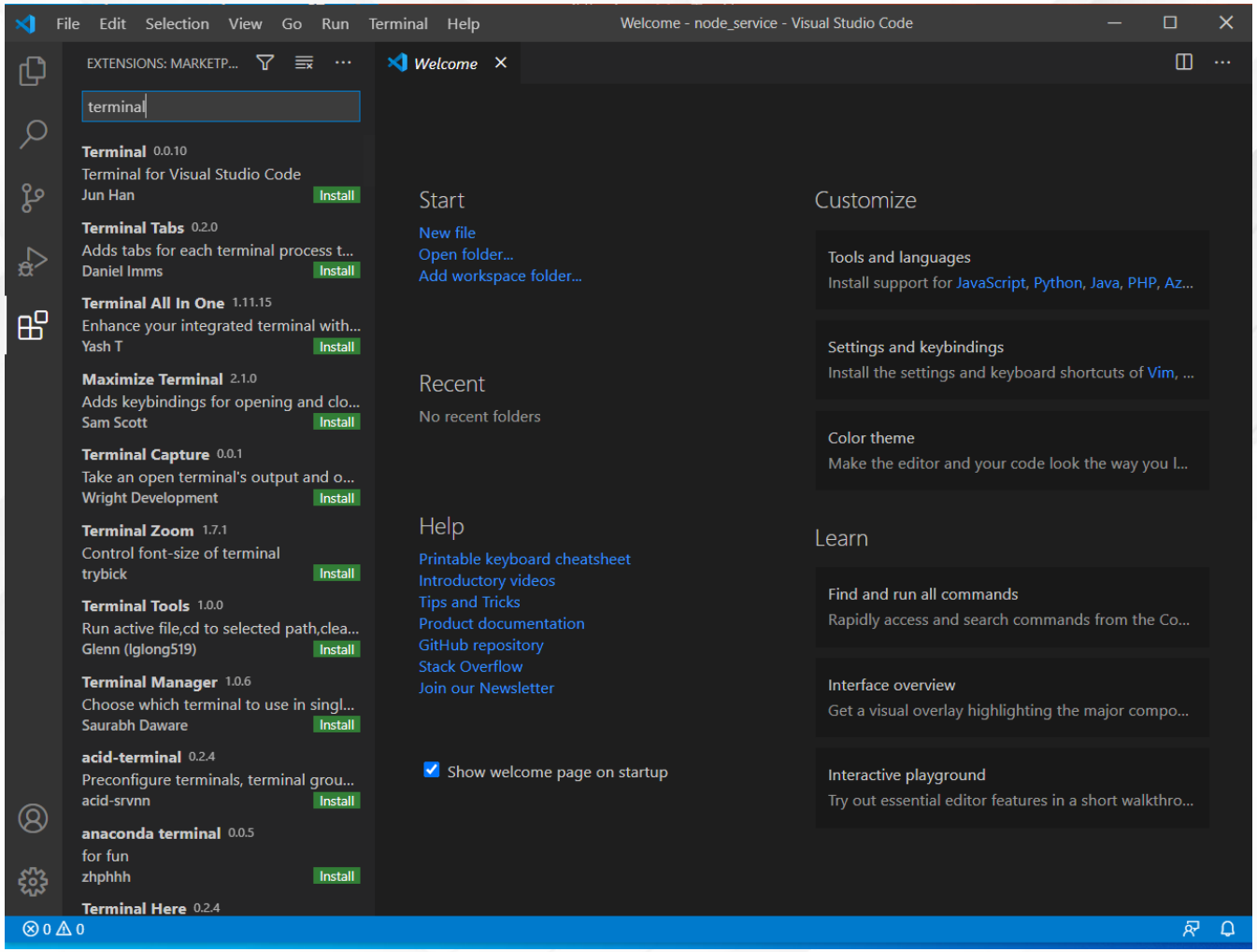


> SERVIDOR NODE JS

- Visual Studio Code
- Instalamos la terminal pulsando el icono Extensions

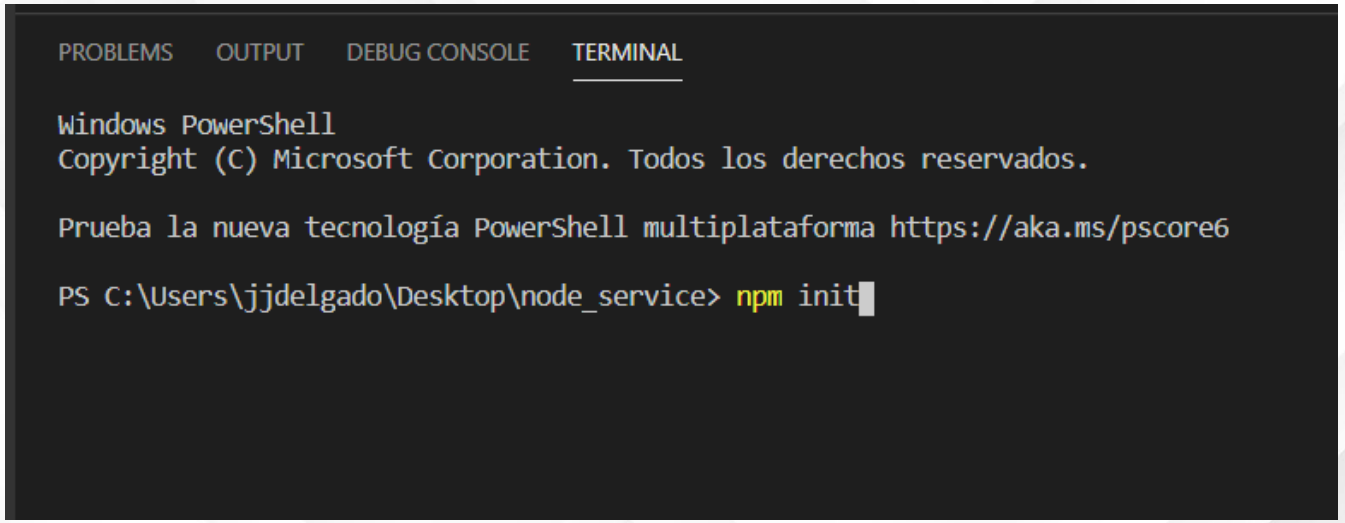


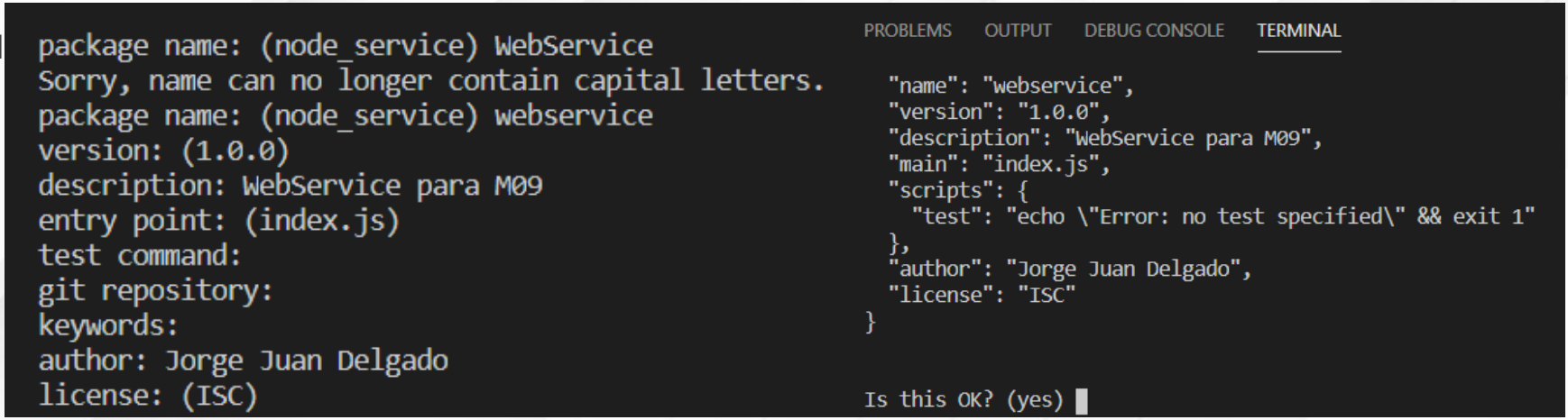


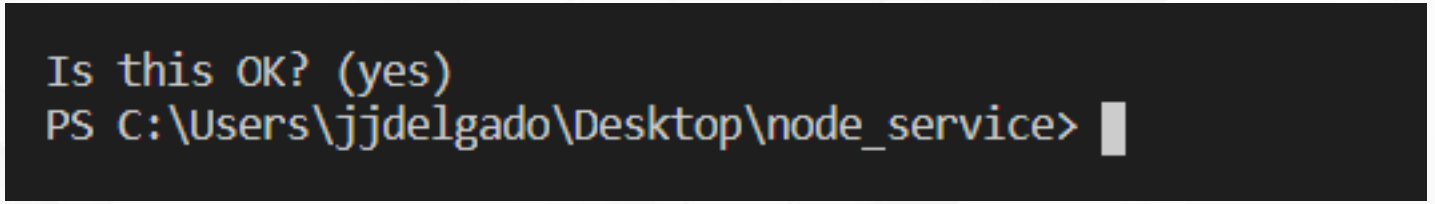


> SERVIDOR NODE JS

- En la terminal escribimos el primer comando:
  - npm init
- Nos pedirá un nombre para nuestro servidor, una versión, una descripción, u (dejamos el que sugiere)
- Test command: enter
- Git repository: enter
- Keywords: enter
- Author: Jorge Juan Delgado

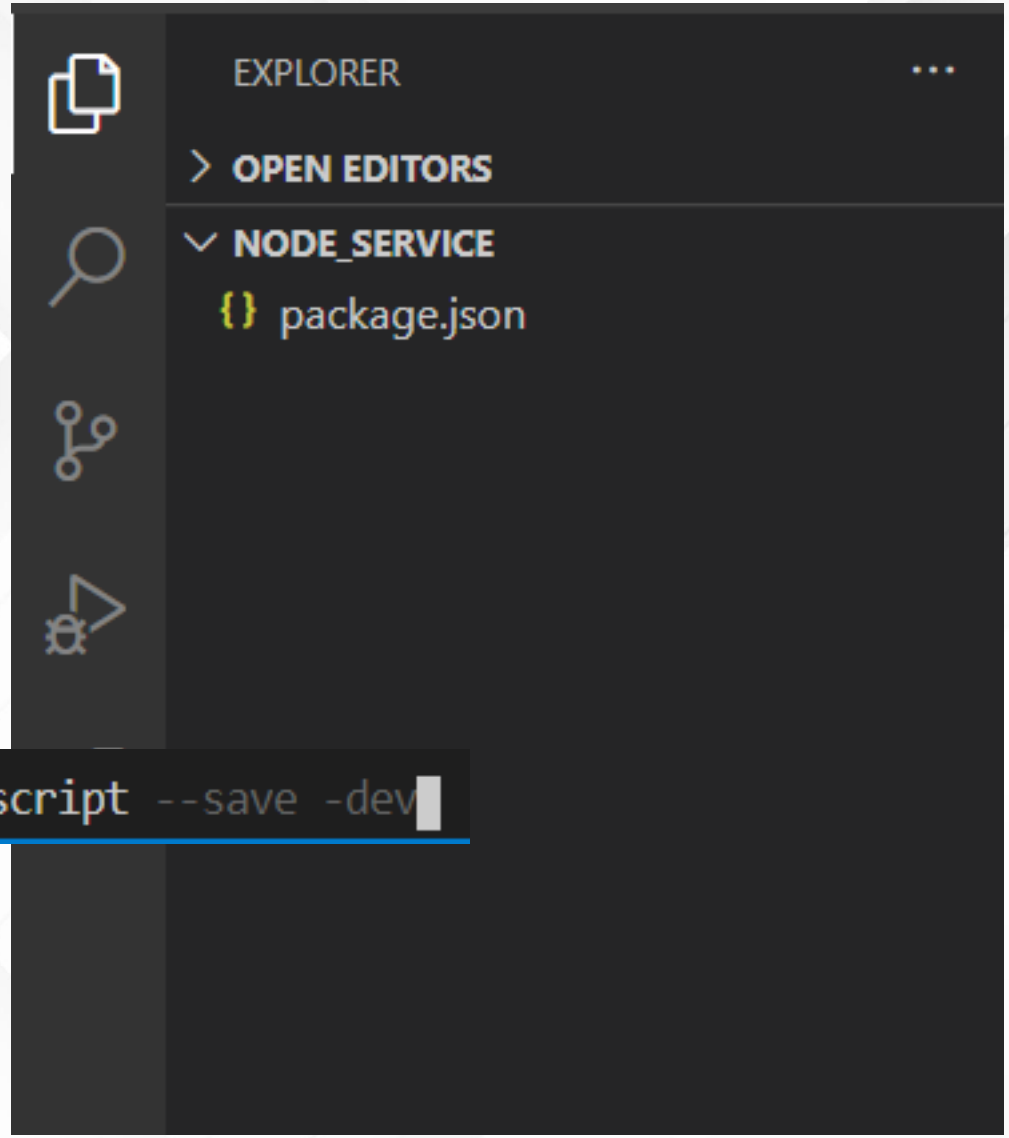






> SERVIDOR NODE JS

- Podemos ver que ha creado automáticamente un fichero llamado package.json
- Aquí vamos a incluir todas las librerías que vamos a necesitar a lo largo de nuestro proyecto
- Vamos a instalar la librería TypeScript




```
PS C:\Users\jjdelgado\Desktop\node_service> npm install typescript --save -dev
```

- Al añadir –save la estaremos incluyendo en el fichero package.json (opcional en las últimas versiones)

- Ahora vamos a instalar otra librería de desarrollo llamada ts-node nodemon

```
PS C:\Users\jjdelgado\Desktop\node_service> npm install --save-dev ts-node nodemon
```

- Por último, como estamos trabajando con TypeScript, instalaremos una librería más llamada @types/node

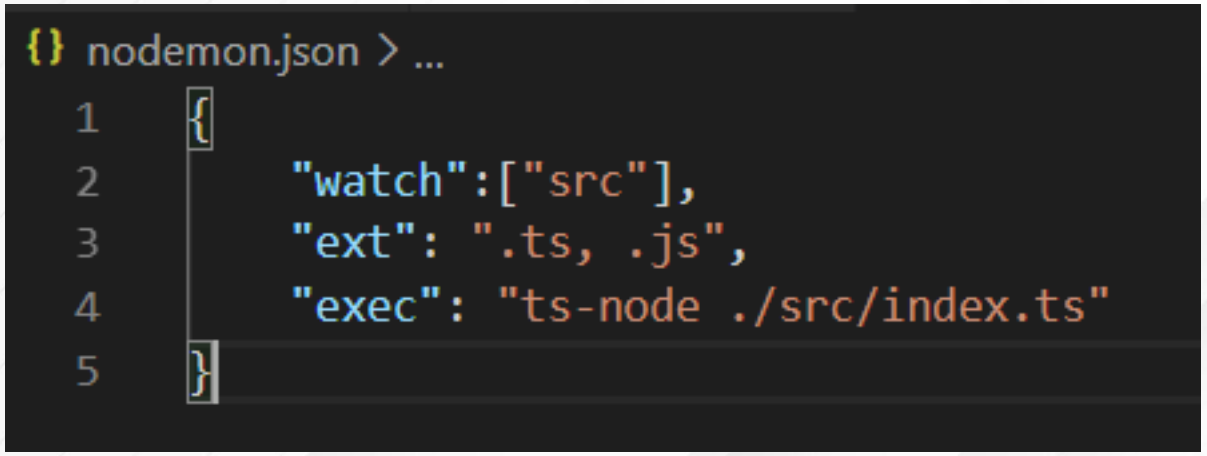
```
PS C:\Users\jjdelgado\Desktop\node_service> npm install @types/node --save-dev
```

> SERVIDOR NODE JS

- Al igual que al principio hicimos un npm init, ahora tendremos que hacer
  - npx tsc –init
- Con esto creamos otro fichero tsconfig
- Cambiamos el target a es6 y descomentamos outDir y le indicamos ./dist

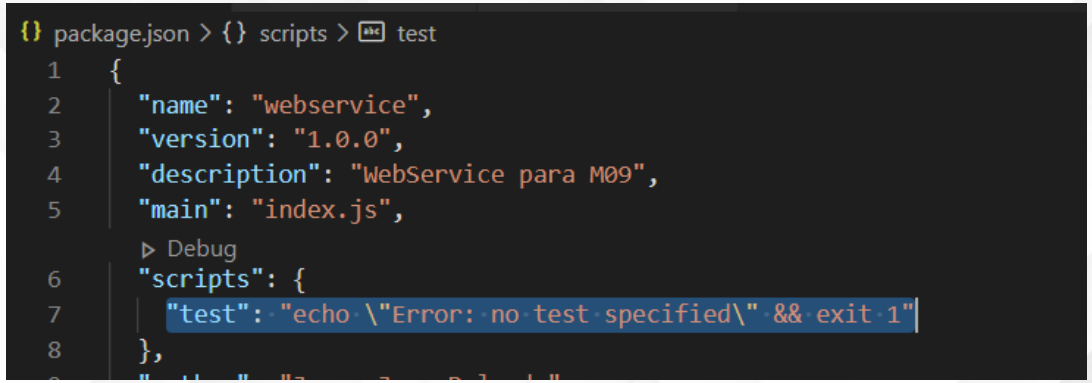
```
{
  "compilerOptions": {
    /* Visit https://aka.ms/tsconfig.j

    /* Basic Options */
    // "incremental": true,
    "target": "es6",
    "module": "commonjs",
    // "lib": [],
    // "allowJs": true,
    // "checkJs": true,
    // "jsx": "preserve",
    // "declaration": true,
    // "declarationMap": true,
    // "sourceMap": true,
    // "outFile": "./",
    "outDir": "./dist",
    // "rootDir": "./",
```

- En nodemon.json tenemos que configurarlo de esta forma:

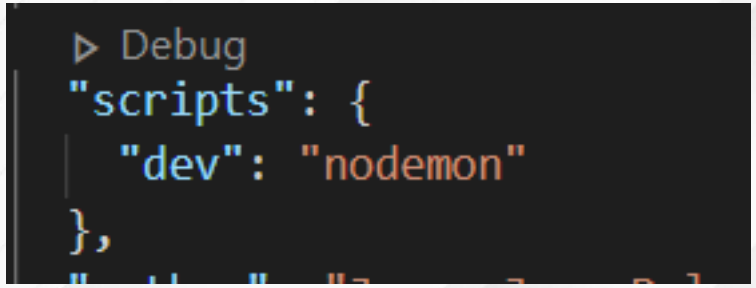


- Guardamos los cambios

- Ahora vamos a cambiar la configuración de nuestro proyecto de test a desarrollo
- Editamos en package.json

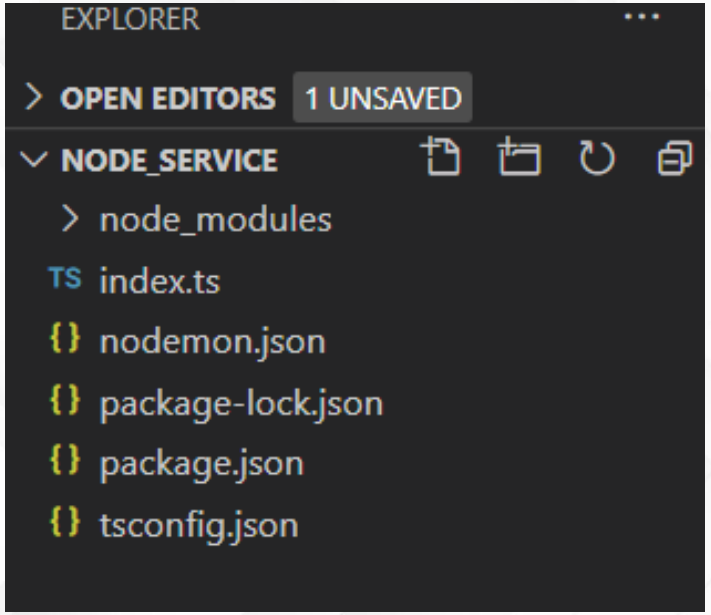


- Sustituimos test por dev y que use nodemon



- Guardamos los cambios

> SERVIDOR NODE JS



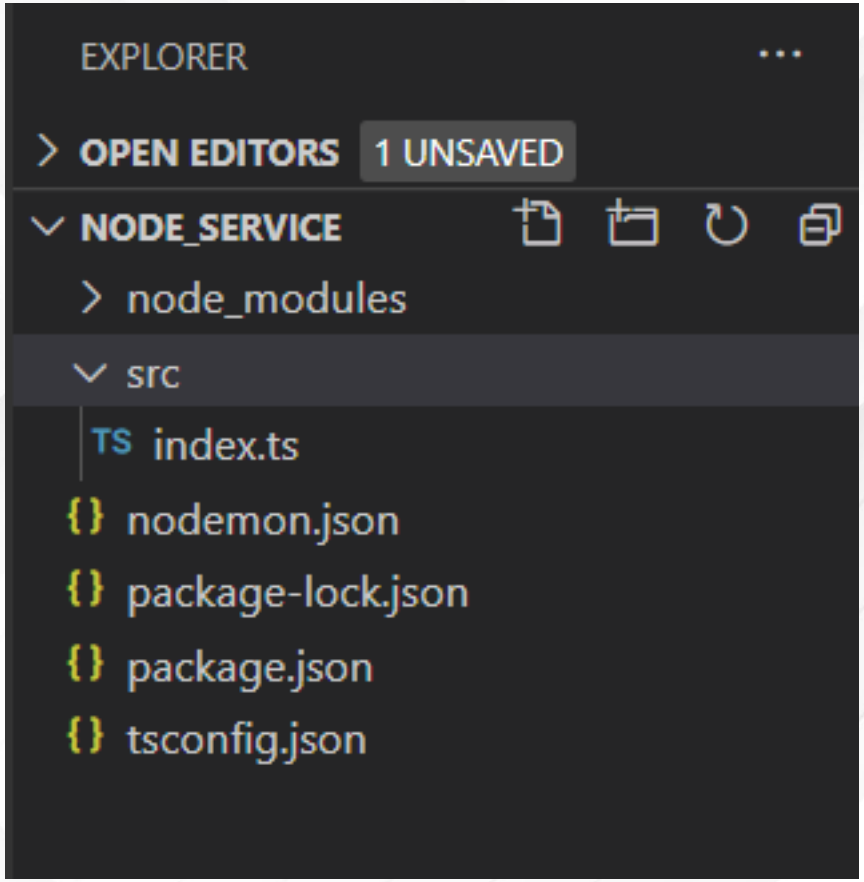
- Pasamos a crear los ficheros que hemos dicho antes:
  - index.ts: donde tendremos todo lo necesario para hacer funcionar al servidor
  - Ahora vamos a necesitar express, la librería que vimos al principio, por lo que tendremos que instalarla
    npm install express

```
PS C:\Users\jjdelgado\Desktop\node_service> npm install express
```

  - Copiamos el código que tenemos en la web de npm sobre express

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

- Pasamos a crear los ficheros que hemos dicho antes:
  - Creamos también el directorio src y movemos el fichero index.ts tal como habíamos indicado en la definición



> SERVIDOR NODE JS

- Cambiamos el mensaje de saludo por un status 200 en formato json indicando ok “Servidor funcionando”
- Si da error en status es porque no está importando bien la librería express
  - En este caso, deshabilitamos la restricción de tipado fuerte
    la ponemos a false en el fichero tsconfig.json

```
1   const {express} = require('express')
2
3   const app = express()
4
5   app.get('/', function (req, res) {
6     res.status(200).json({
7         ok:"Servidor funcionando"
8     })
9   })
10
11  app.listen(3000)
```

```
// "isolatedModules": true,                    /*

/* Strict Type-Checking Options */
"strict": false,                               /*
// "noImplicitAny": true                       /*
```

> SERVIDOR NODE JS

- Faltaría por instalar una librería para indicar que se usará express con TypeScript
  npm install express @types/express
- Solo la teníamos instalada para Java y ahora si la tendremos también para TypeScript

- Volvemos a poner en true la configuración fuerte de tipado

- Cambiamos la importación de express

src > TS index.ts > ...

```
1   import express = require('express');
2
3   const app = express()
4
5   app.get('/', function (req, res) {
6     res.status(200).json({
7         ok:"Servidor funcionando"
8     })
9   })
10
11  app.listen(3000)
```

> SERVIDOR NODE JS

- Lo ejecutamos:

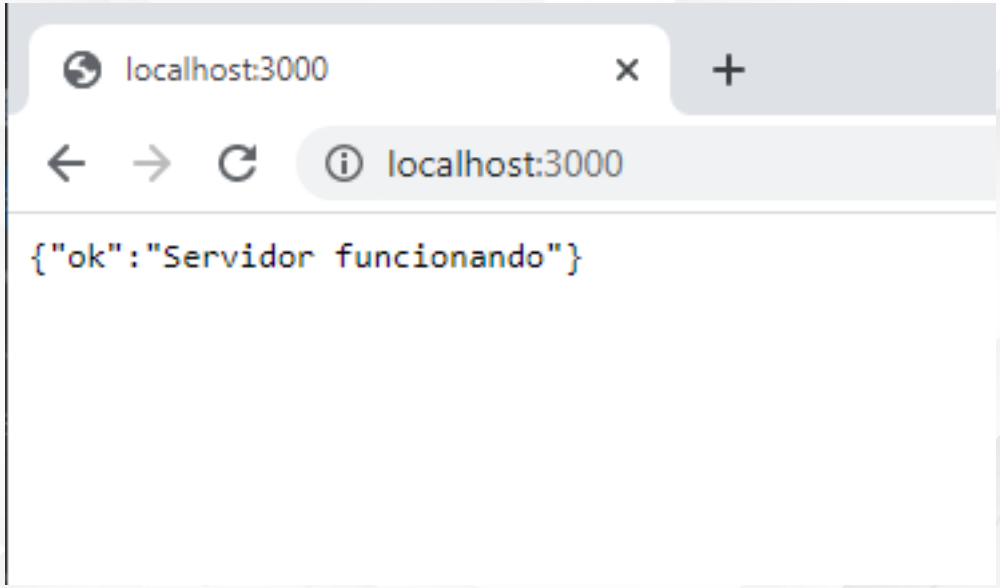
  npm run dev

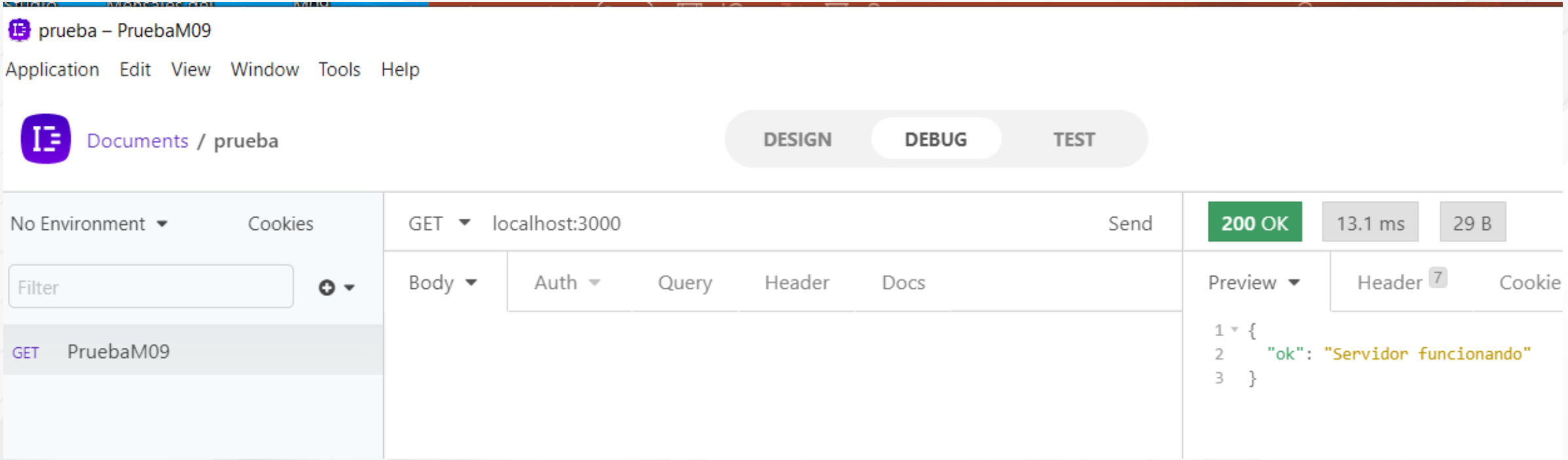
- Cuando nos conectemos a localhost en el puerto 3000 (por el que está escuchando) vamos a obtener una respuesta de tipo json

```
PS C:\Users\jjdelgado\Desktop\node_service> npm run dev

> webservice@1.0.0 dev C:\Users\jjdelgado\Desktop\node_service
> nodemon

[nodemon] 2.0.6
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): src\**\*
[nodemon] watching extensions: ts,js
[nodemon] starting `ts-node ./src/index.ts`
```

> SERVIDOR NODE JS

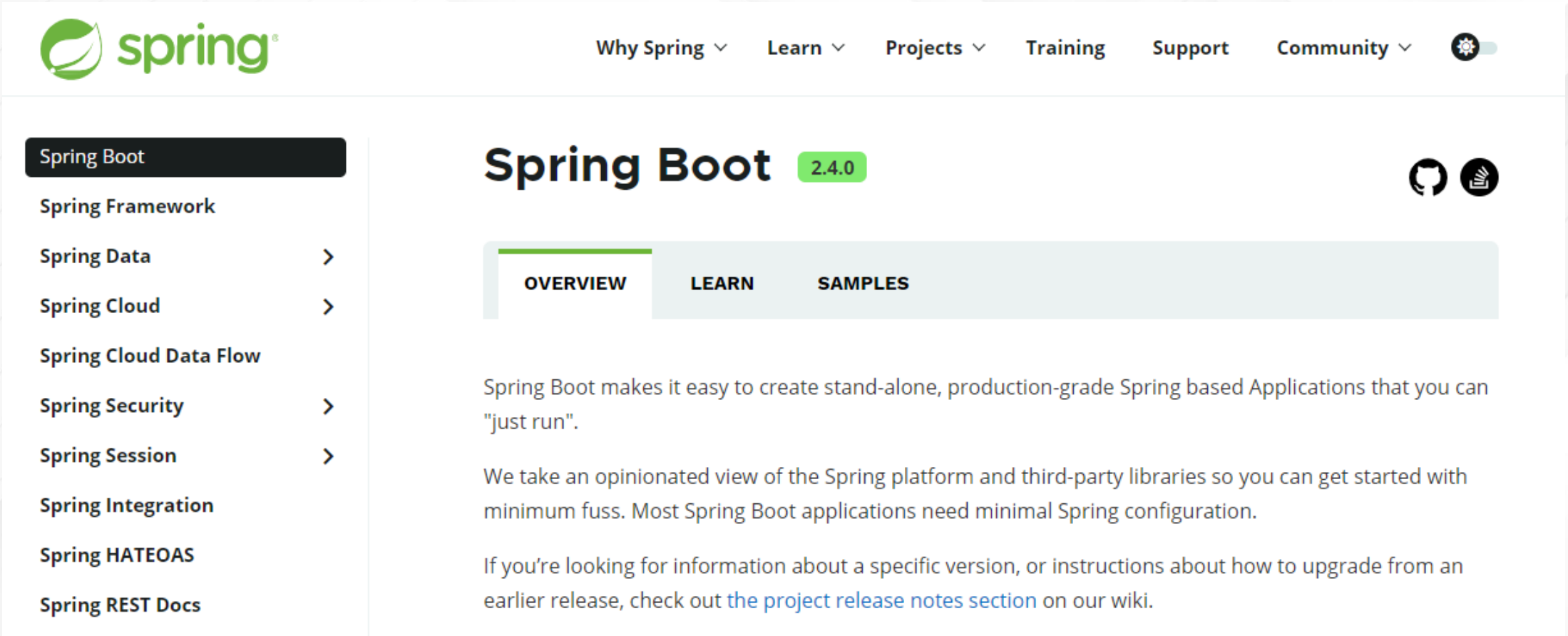
- Lo probamos desde Insomnia y desde el navegador haciendo un GET a localhost por el puerto 3000





CONFIGURAR SERVIDOR JAVA EN SPRING BOOT

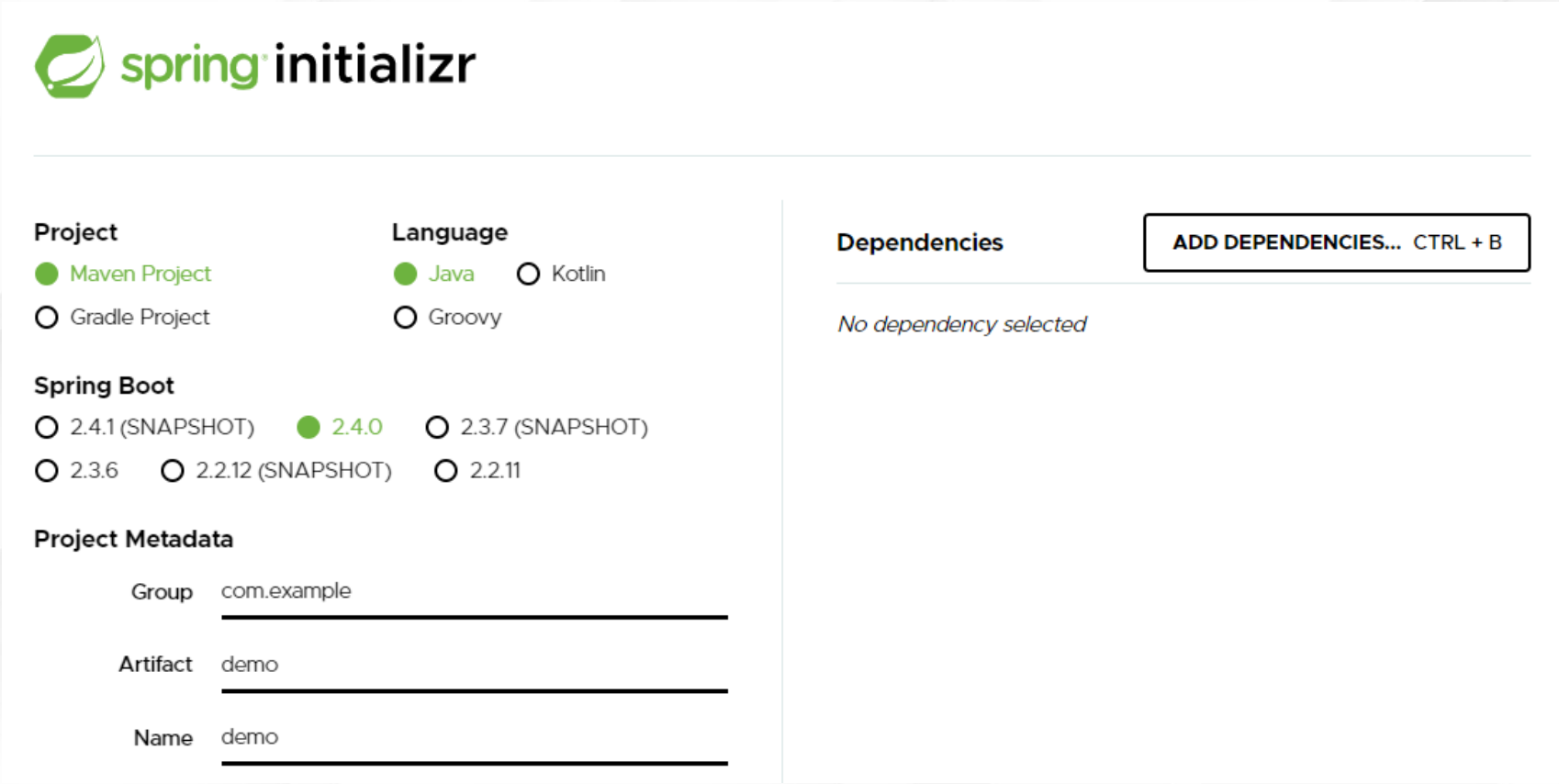
- Vamos a usar la aplicación Spring Initializr



> SERVIDOR JAVA SPRINGBOOT

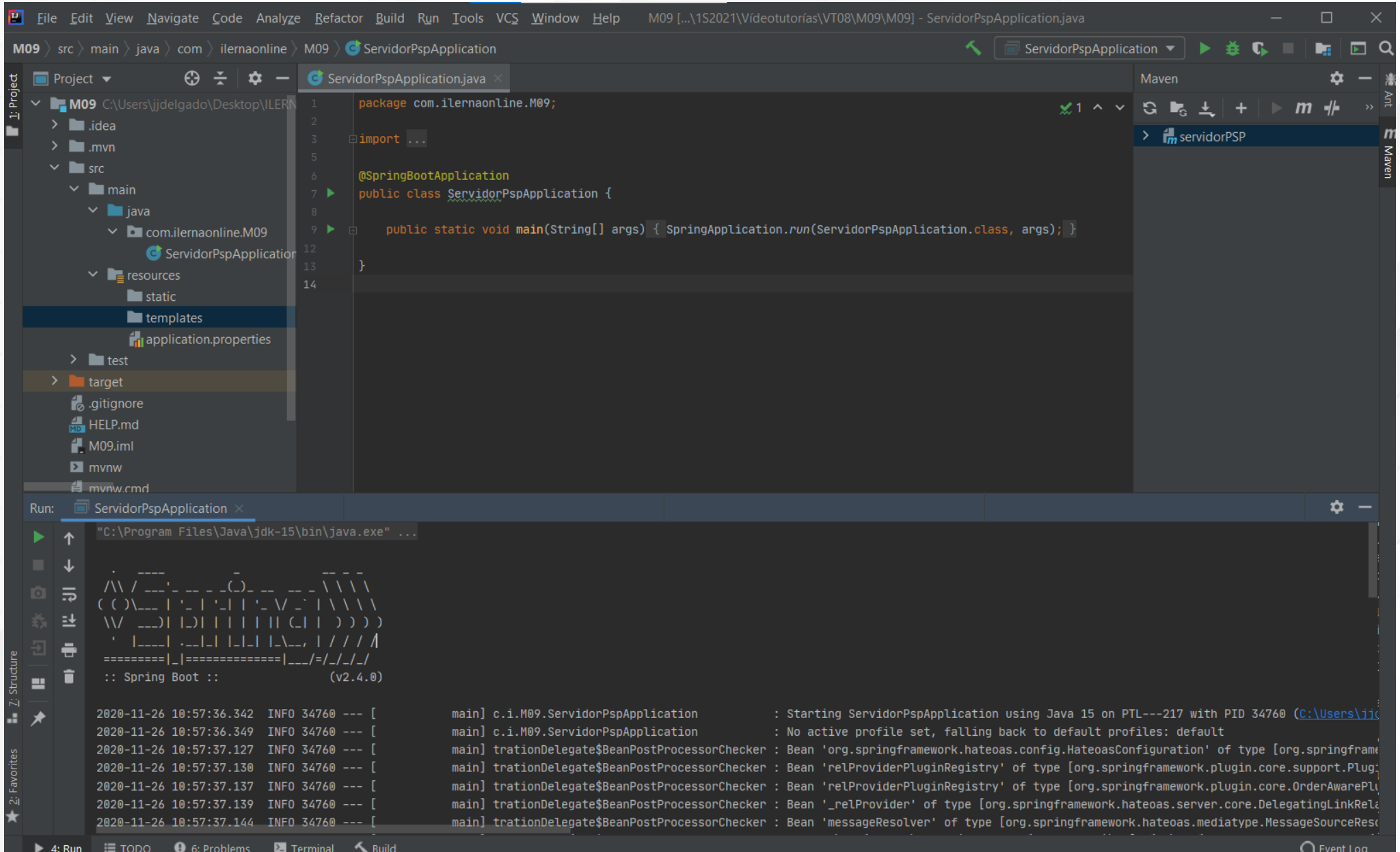
ILERNA Online

- Spring Initializr (https://start.spring.io/)
- Permite conifugrar:
- Tipo de proyecto, Maven o Gradle
    - Lenguaje (Java, Kotlin, Groovy)
    - Versión de Spring Boot
    - Datos del proyecto
    - Tipo de Packaging
    - Versión de Java



> SERVIDOR JAVA SPRINGBOOT

ILERNA Online



UF1

SEGURIDAD Y CRIPTOGRAFÍA

ILERNA Online

UF1 Seguridad y Criptografía.

- Al momento de programar, hay que tener en cuenta que un código limpio aportará seguridad a la aplicación.
- No dejar variables en blanco.
- Longitud del tamaño de las estructuras complejas.
- Vigilar acceso a memoria.

UF1 Seguridad y Criptografía.

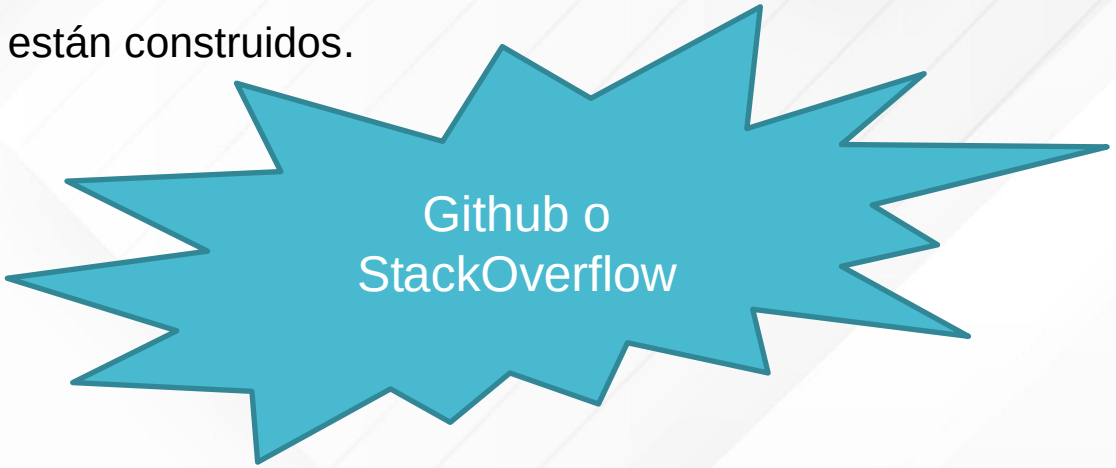
- Aspectos a tener en cuenta los siguientes aspectos:
  - Los usuarios intentarán vulnerar la aplicación.
  - Los archivos que se utilicen en la aplicación deben ser de solo lectura
  - Toda información sensible debe ir cifrada.
  - Comprobación de las llamadas al sistema.
  - Utilización de rutas absolutas.

ILERNA Online

UF1 Seguridad y Criptografía.

- Al desarrollar una aplicación hay que tener presente:

1. Estar siempre informado de los diferentes tipos de vulnerabilidades.
2. Encontrar un equilibrio entre medidas de seguridad y complejidad frente a la carga de nuestra aplicación o web
3. Explorar software libre, para poder observar como están construidos.



UF1 Seguridad y Criptografía.

- Recomendación: Libro Clean Code / Código limpio
- Autor: Robert C. Martin
- Muy recomendado para desarrolladores
- Técnicas para programar
- Normas de estilo, patrones de algoritmos
- Incluye paradigma de POO

UF1 Seguridad y Criptografía.

> Atención al desarrollar aplicaciones

Software

Microsoft recompensa detección de errores de código en Windows Vista

Javier Sanz | Publicado el 15 de mayo, 2006 - 13:45

Con el fin de solucionar el mayor número posible de errores de código antes de lanzar su próximo sistema operativo Windows Vista, Microsoft pone en práctica métodos poco tradicionales. Uno de los ingenieros jefe del equipo Windows Vista de Microsoft propuso antes del fin de semana una competencia poco tradicional entre los empleados de la compañía. Cada error que detectaran y solucionaran antes del fin de semana sería recompensado con 100 dólares. Aparte de ello, el empleado que corrigiera el mayor número de errores durante el fin de semana recibiría un premio especial de 500 dólares.

Microsoft pagará hasta 20.000 dólares si encontramos bugs y fallos en Xbox Live

La compañía de Redmond inicia un nuevo programa de recompensa para busca fallas y problemas en la red de internet y juegos de la consola.

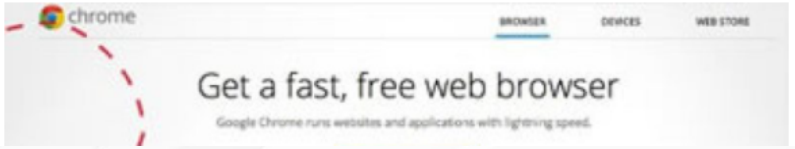
En agosto de 2016 Ivan Krstic, responsable de ingeniería y arquitectura y seguridad de Apple, anunciaba algo especial. Su empresa iniciaba su programa de cazarrecompensas de bugs informáticos. Más vale tarde que nunca, pensarían muchos: la celebérrima compañía con sede en Cupertino se había resistido durante años a algo así, pero al fin lo hacía, y con premios de lo más jugosos que **ascendían hasta los 200.000 dólares** para los descubrimientos más importantes.

TECNOLOGÍA

Google premia a 'hacker' con 60,000 dls

Un adolescente ganó por segunda vez el concurso al descubrir un error en el navegador Chrome; la tecnológica solucionó el problema con una actualización de software.

jue 11 octubre 2012 03:32 PM



El hacking ético se está convirtiendo en una necesidad ineludible para las empresas. Este tipo de práctica analiza la infraestructura tecnológica de las organizaciones simulando ataques "pirata" para así evaluar el nivel de ciberseguridad con el que cuentan

ILERNA Online

UF1 Seguridad y Criptografía.

- ¿Hacker o Cracker?
- De acuerdo con los principales diccionarios de inglés y algunos de español como el de María Moliner, se define como ***hacker*** **a una 'persona con sólidos conocimientos informáticos capaz de introducirse sin autorización en sistemas ajenos** para manipularlos, obtener información, etc., o simplemente por diversión'.

- La palabra ***cracker***, en cambio, se aplica a quien, además de ser capaz de entrar en sistemas ajenos, **lo hace con fines delictivos**, como señala el diccionario de Oxford.

ILERNA Online

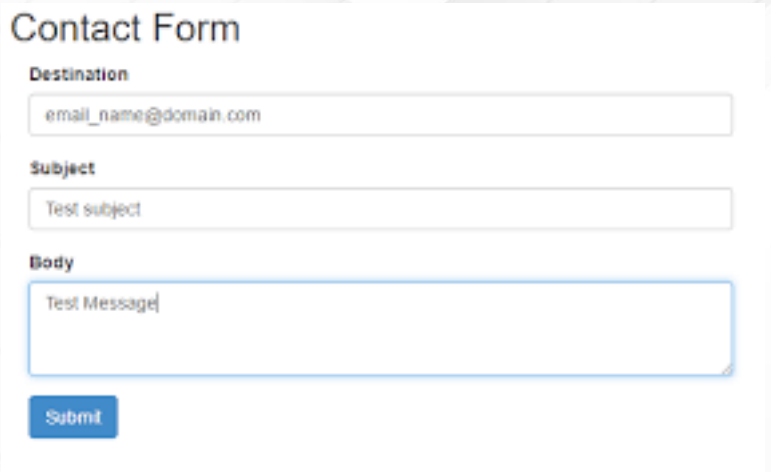
UF1 Seguridad y Criptografía.

- En casi todas las aplicaciones, los usuarios introducen y extraen información, por lo que hay que verificar en todo momento que dicha información sea válida

- En SQL tenemos por ejemplo el ataque SQL injection, con el fin de poder modificar las consultas y extraer información que no decería estar a nuestro alcance

- Una téncnica muy usada es la inyección de código en formularios donde admiten este tratamiento

SQL Injection



Result

```
SELECT * FROM Users WHERE Name ="John Doe" AND Pass ="myPass"
```

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the user name or password text box:

User Name:

" or ""="

Password:

" or ""="

The code at the server will create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

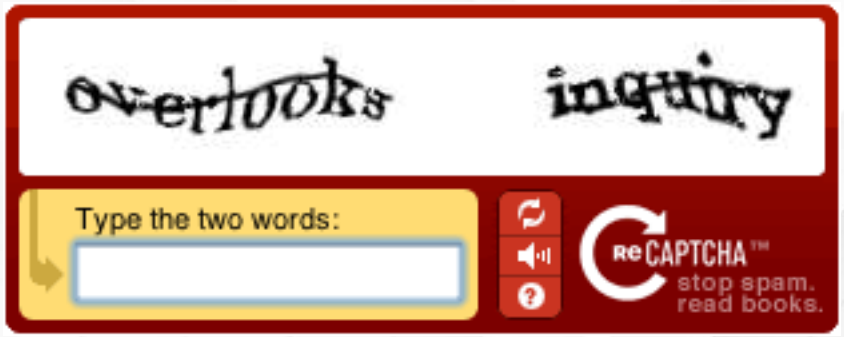
ILERNA Online

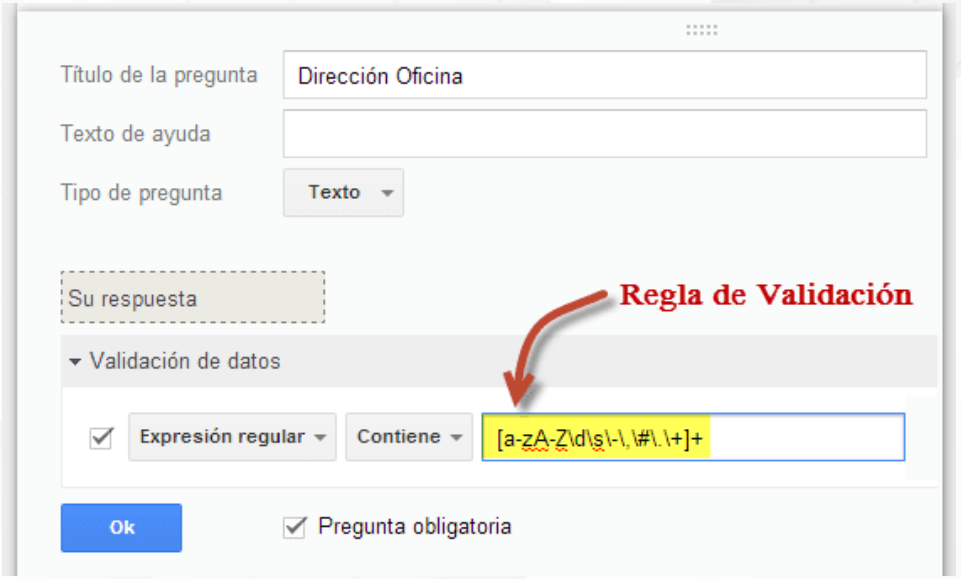
UF1 Seguridad y Criptografía.

- Solución

- Hacer un tratamiento previo de la información que se recibe antes de enviarla como consulta a la BD del servidor

  - Evitar que introduzcan comillas, comandos, etc
  - Usar CAPTCHAs
  - Limitaciones y reglas en formularios





UF1 Seguridad y Criptografía.

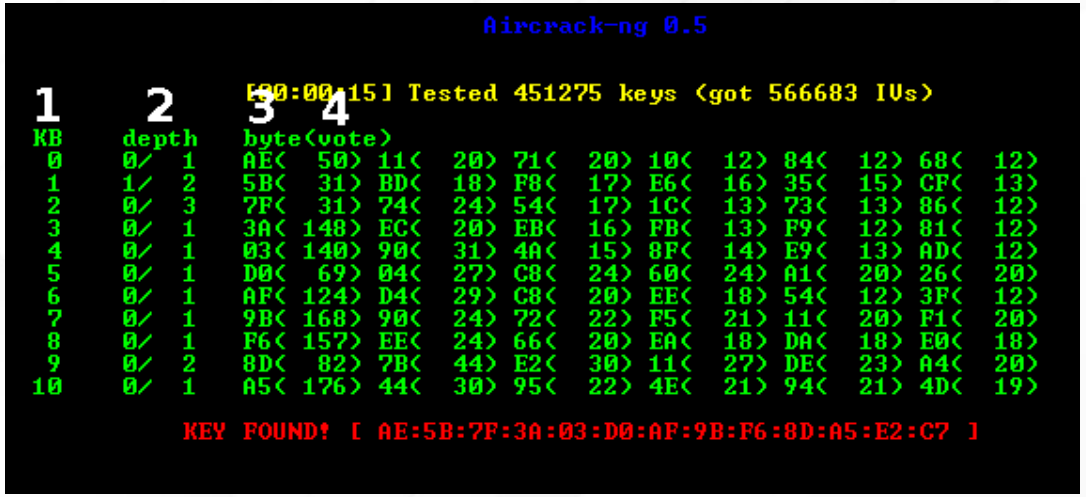
- ATAQUE FUERZA BRUTA

- El atacante emplea determinadas técnicas para probar combinaciones de contraseñas con el objetivo de descubrir las credenciales de una potencial víctima y así lograr acceso a una cuenta o sistema

  - Probando diferentes contraseñas

  - Usando software, como el programa John the Ripper, que permite romper varios algoritmos de cifrado o hash como DES SHA-1 y otros.

  - Usando software para averiguar contraseñas de redes Wi-Fi como Aircrack-ng



UF1 Seguridad y Criptografía.

- Aunque la criptografía es un tema que parezca muy actual, se remonta al S. V a.C. Donde ya se utilizaban técnicas para ocultar la información que se quería transmitir de un lugar a otro.
- Desde esa época han ido evolucionado los mecanismos de criptografía para hacerlos cada vez más seguros y difíciles de descifrar.

Criptografía: conjunto de técnicas que cifran información, con el objetivo que un usuario no pueda ver su contenido

UF1 Seguridad y Criptografía.

- Criptografía - Escítala

UF1 Seguridad y Criptografía.

- En el cifrado cesar se realiza una sustitución de cada letra por otra.
- Para conseguir que letra le corresponde se utiliza una función matemática, así de un texto podemos obtener otro diferente.
- Hola Onliners → Krñd Rpñlphuv

Mensaje original

Mensaje cifrado

Original	a	b	c	d	e	f	g	h	i	j	k	l	m
Cifrado	d	e	f	g	h	i	j	k	l	m	n	o	p

Original	n	o	p	q	r	s	t	u	v	w	x	y	z
Cifrado	q	r	s	t	u	v	w	x	y	z	a	b	c

UF1 Seguridad y Criptografía.

- Encriptar mediante la clave cesar el mensaje con una longitud de 7:
- Hoy es jueves y tenemos videotutoria

Ñvf lz pblclz f altlsvz coklvabavyoh

A B C D E F G H I
J K L M N Ñ O P Q
R S T U V W X Y Z
a b c d e f g h i j k l m n ñ
o p q r s t u v w x y z

UF1 Seguridad y Criptografía.

Una función hash es una función que, dada cualquier cadena de caracteres, los convierte en otra cadena de longitud fija.

Función hashCode()
de Java

UF1 Seguridad y Criptografía.

- **MD5**: Genera un hash de 128b, 32 caracteres hexadecimales.
  - Uso: Se suele utilizar para la comprobación de ejecutables.
- **SHA1**: Genera un hash de 160b, 40 caracteres hexadecimales,

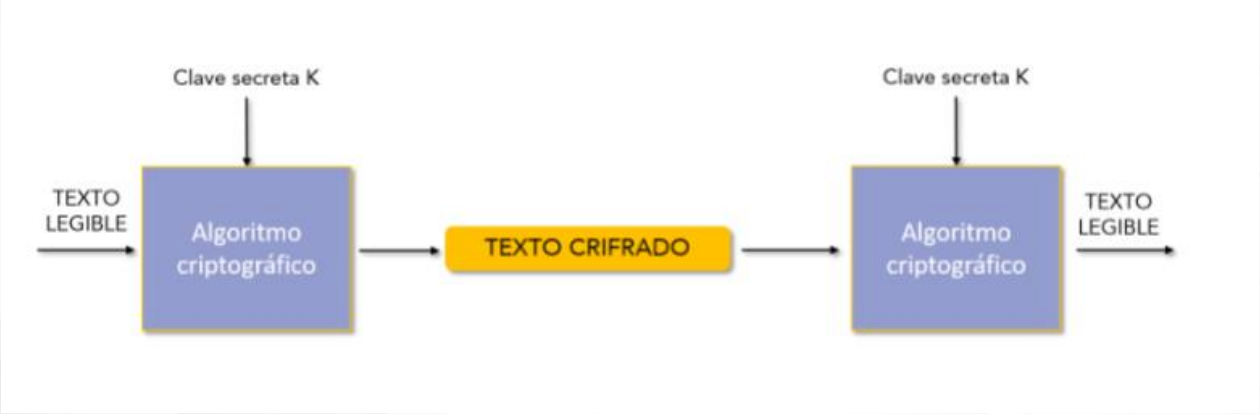
UF1 Seguridad y Criptografía.

- **URL: https://www.md5online.es**
- El MD5 es un algoritmo de codificación de 128 bits que genera un hash hexadecimal de 32 caracteres, independientemente de la longitud de la palabra de entrada
- Este algoritmo no es reversible, siendo normalmente imposible encontrar la palabra original a partir de un MD5
- Palabras en la base de datos: 1,154,869,499,174

UF1 Seguridad y Criptografía.

CRIPTOGRAFÍA SIMÉTRICA

- En este tipo de algoritmos el emisor y receptor comparten una única clave.
- El mensaje se cifra y se descifra con esa única clave.



UF1 Seguridad y Criptografía.

CRIPTOGRAFÍA SIMÉTRICA

- DES: Utiliza una clave de 56b, por lo que no se considera seguro.
- Triple DES: Igual a des pero pasó de 56b como clave a 112b.
- AES: Igual a DES y TDES pero la clave puede ser de 128,192 y 256b.

UF1 Seguridad y Criptografía.

- EJEMPLO de Criptografía Simétrica
- Servidor KERBEROS
- Desarrollado en los 80 en el MIT. La versión 5 (1993) se ha convertido en un estándar de Internet (RFC 1510)
- Se usa en Windows 2000, XP y Server 2003. Es el sistema de autenticación por defecto de DCE.
- Evita que las contraseñas viajen a través de la red (en claro o cifradas)

Servidor KERBEROS

- Elementos:
  - **Ticket**: Elemento que prueba ante un servicio determinado, que un cliente se ha autenticado recientemente con Kerberos Tienen tiempo de expiración
  - **Autenticación**: Elemento encriptado con la clave de sesión apropiada que contiene el nombre del cliente y una marca Universidad de Oviedo / Dpto. de Informática ATC-Distribuidas temporal. Demuestra la identidad del usuario
  - **Clave de sesión:** Clave secreta generada aleatoriamente por Kerberos y enviada a un cliente para su uso en una comunicación particular con algún servidor.

Servidor KERBEROS

- Un servidor Kerberos se conoce como Centro de Distribución de Claves (KDC, Key Distribution Center)
- Cada KDC ofrece dos servicios
  - Servicio de Autenticación (AS, Autentication Service), que se encarga de validar al usuario frente al sistema. Sustituye al login clásico
  - Servicio de Concesión de Tickets (TGS, Ticket Granting Service). Emite tickets que autorizan al usuario para acceder a un servicio determinado.
- Un ticket Kerberos tiene un período de validez fijo que comienza en t1 y acaba en t2.

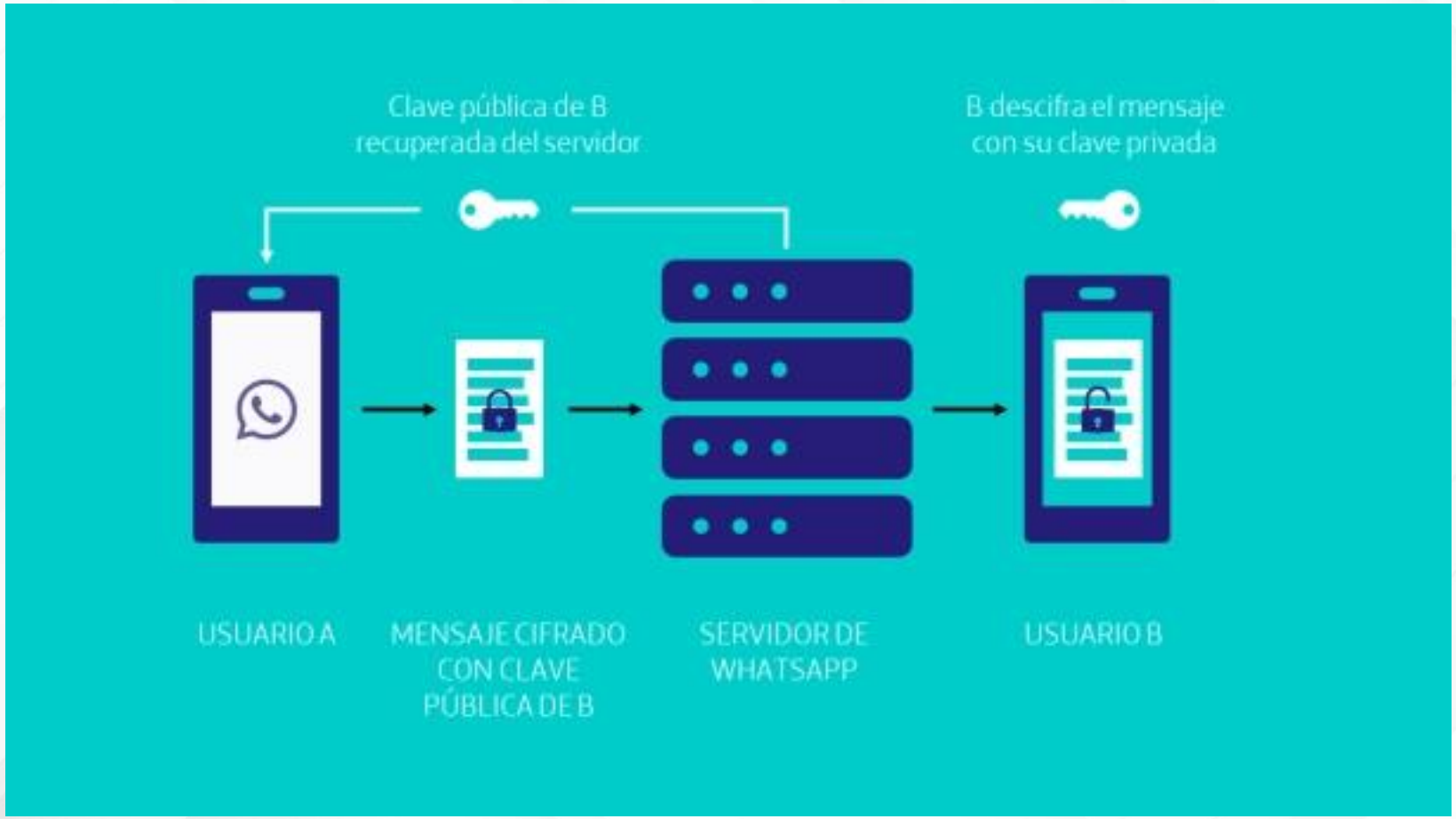
UF1 Seguridad y Criptografía.

CRIPTOGRAFÍA ASIMÉTRICA

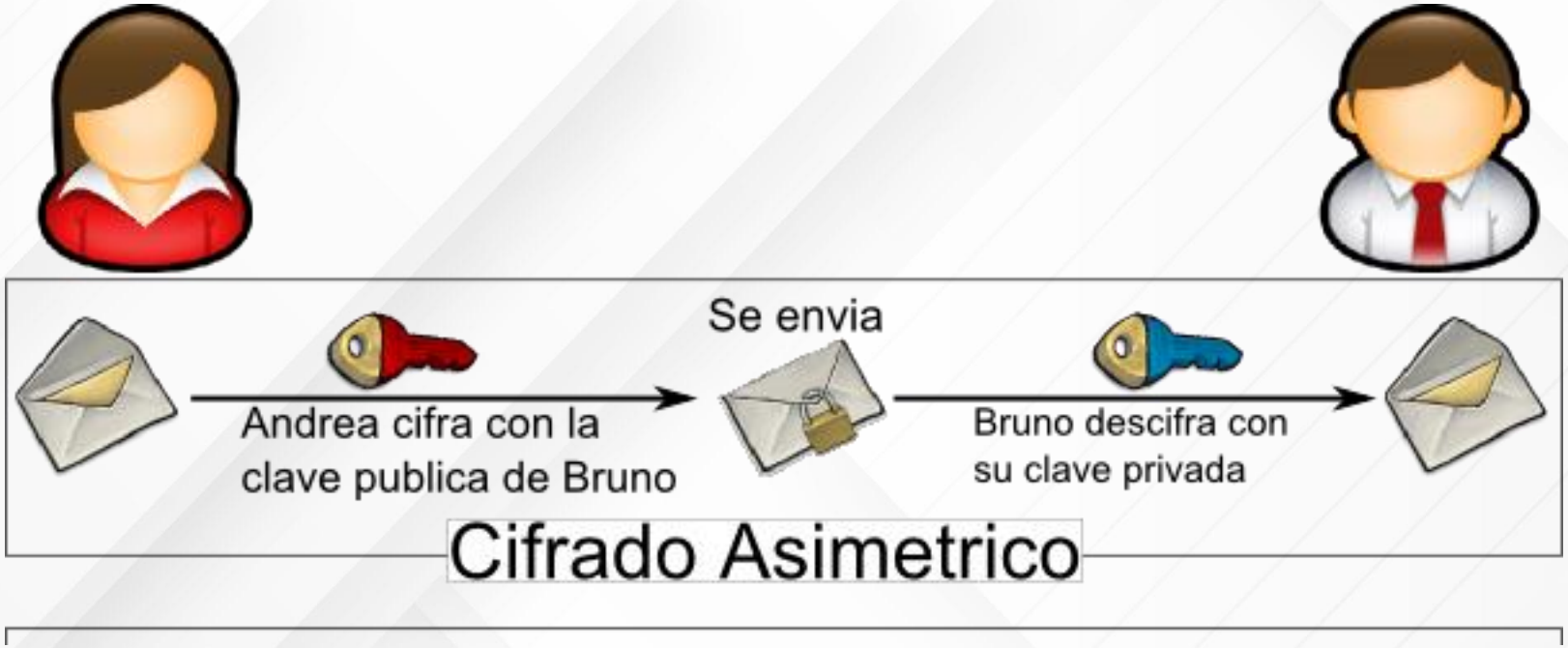
Este tipo de algoritmos genera 2 claves.

1. Clave pública (para cifrar)
2. Clave privada (para descifrar)

CRIPTOGRAFÍA ASIMÉTRICA



UF1 Seguridad y Criptografía.



UF1 Seguridad y Criptografía.

- La seguridad radica en el problema de la factorización de números enteros. Tanto la clave pública como la privada se componen de un par de números.

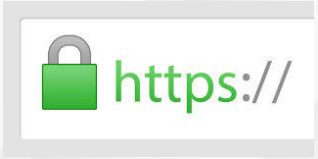
UF1 Seguridad y Criptografía.

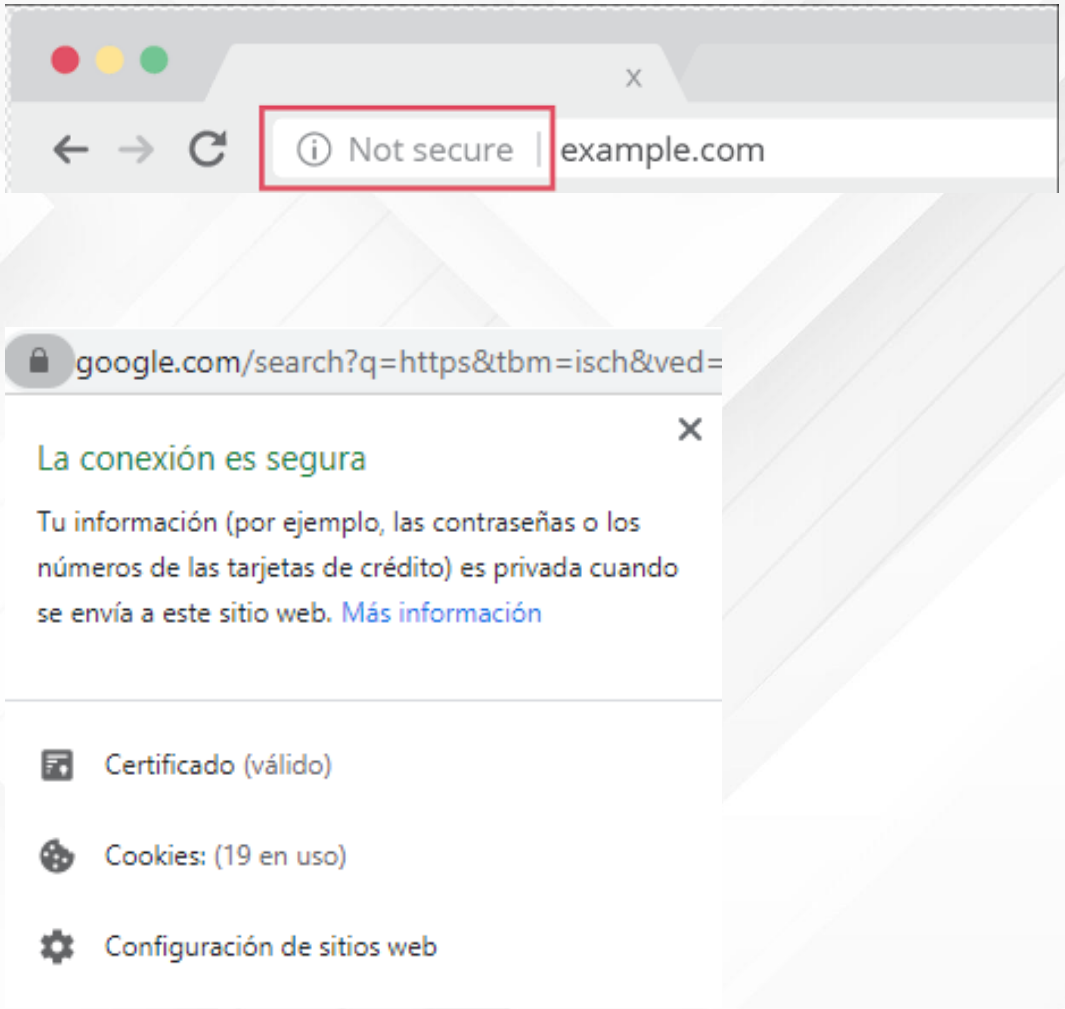
- **Seguridad en las comunicaciones** ➔ Principal aplicación, se encarga de que la información viaje de forma segura.
- **Identificación y autenticación en recursos y sistemas** ➔ Gracias a la criptografía es posible validar el acceso de los usuarios a la **firma digital**.

Mecanismo mediante el cual el receptor es capaz de identificar al emisor.

UF1 Seguridad y Criptografía.

- Al entrar en una web HTTPS estamos accediendo a la navegación segura
- Se certifica que la persona que está al otro lado es quién dice ser
- Posicionamiento SEO castiga a webs sin certificación digital





UF1 Seguridad y Criptografía.

- **Certificación** ➔ Certificado generado por una Autoridad Certificadora
- **Comercio electrónico** ➔ Realizar operaciones sensibles por Internet con la seguridad de que los datos no serán interceptados.

UF1 Seguridad y Criptografía.

- **Integridad de los datos** ➔ Se asegura que los datos no han sido modificados por usuarios sin permiso.
- **Disponibilidad** ➔ Se asegura que los datos estén disponibles cuando se solicitan.
- **Confidencialidad** ➔ Los datos solo podrán ser leídos por aquellas personas que están autorizadas.
- **Autenticidad** ➔ Característica por la cual el receptor sabe quien es el emisor.
- **No repudio** ➔ El emisor no puede negar haber enviado un mensaje.

UF1 Seguridad y Criptografía.

- **3 fases:**
  - Identificación ➔ El usuario dice quien es.
  - Autenticación ➔ El sistema verifica al usuario
    - Contraseña
    - Biometría
    - Tarjetas de identificación.
  - Autorización ➔ El sistema da permiso al usuario.

Sistemas mucho más seguro que la contraseña

ILERNA Online

UF1 Seguridad y Criptografía.

- Protocolo criptográfico ➔ Conjunto de reglas sobre la seguridad en la comunicación de los sistemas informáticos.
- Normalmente tratan los aspectos:
  - Establecimiento de las claves.
  - Autenticación de entidades.
  - Autenticación de mensajes.
  - Tipo de cifrado
  - Métodos de no repudio.

HTTPS Protocolo de la capa de aplicación que utiliza el cifrado basado en SSL/TLS.

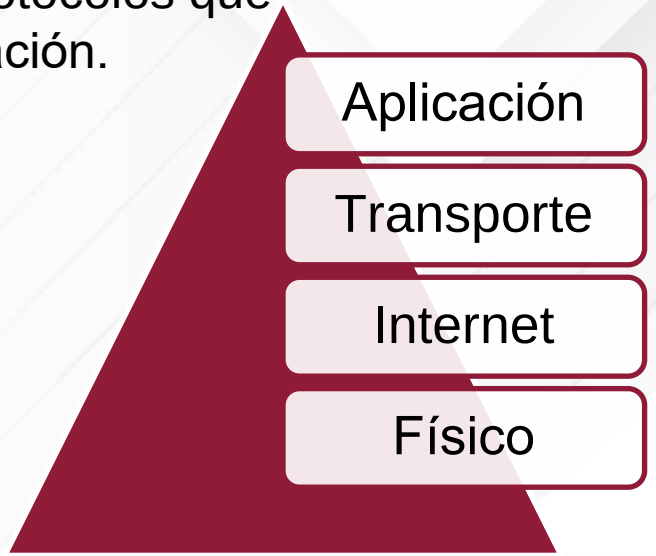
UF1 Seguridad y Criptografía.

- SSH ➔ Versión segura y mejorada del antiguo telnet, permite conexiones cifradas a interpretes de comandos.
- Utiliza tanto cifrado asimétrico como simétrico.
  - Cifrado asimétrico garantiza la autenticidad del cliente y del servidor.
  - Cifrado simétrico garantiza la confidencialidad e integridad de los mensajes.

UF1 Seguridad y Criptografía.

- SSL (**S**ocket **S**ecure **L**ayer) y TLS (**T**ransport **L**ayer **S**ecurity) protocolos que actúan, en la capa de transporte por debajo de la capa de aplicación.

- Ambos utilizan cifrado simétrico y asimétrico.
- Asimétrico ➔ Autenticación
- Simétrico ➔ Intercambio de mensajes
- Integridad de los mensajes ➔ Hash



UF1 Seguridad y Criptografía.

→ Identificar la información considerada sensible.

- Contraseñas
- Datos personales (DNI, CC, correo, etc).

Información privada del usuario.

Toda la información cifrada y el tipo de cifrado utilizado debe estar bien documentada.