2022

CONCEPTOS DE PROGRAMACIÓN

CURSO INICIAL DE PROGRAMACIÓN

PABLO GRADOLPH OLIVA

OPENBOOTCAMP

1. INTRODUCCIÓN AL ECOSISTEMA DE LA PROGRMACIÓN

1.1. INTRODUCCIÓN

¿QUÉ ES UN LENGUAJE DE PROGRAMACIÓN?

Es un lenguaje con reglas gramaticales bien definidas. Permite al programador escribir instrucciones en forma de algoritmos. Y controlar así el comportamiento de un sistema informático u otro tipo de sistemas como podría ser un autómata en una fábrica.

1.2. PARADIGMAS DE PROGRAMACIÓN

IMPERATIVOS VS DECLARATIVOS

Los imperativos son el tipo de paradigma más antiguo que tenemos. Consisten en una secuencia de instrucciones y se definen paso a paso.

Los declarativos se centran en el qué, en lugar del cómo. Se centran en el resultado final y el sistema se encarga de ir llegando a esa situación.

Ejemplo: Receta de cocina.

- Imperativo: Los pasos a seguir.
- Declarativo: Foto final de la receta.

```
const listaProgramadores = ["Gorka", "Martin", "Aris", "Leire"]

let nombres = []

// Programación imperativa
listaProgramadores.forEach((programador, posicion) => {
    nombres[posicion] = programador
})

// Programación declarativa
nombres = [...listaProgramadores]
```

Desde el punto de vista de la legibilidad es mejor la programación imperativa. Del mismo modo, es un mejor paradigma para el aprendizaje. Por último, la programación imperativa es menos escalable, y más difícil de mantener puesto que se necesitará mucho más código para obtener el mismo resultado.

FUNCIONALES VS PROCEDIMENTALES

Los procedimentales es similar al imperativo (línea por línea dando instrucciones de lo que tiene que hacer). Sin embargo, los funcionales se encargan de ir creando una serie de funciones (bloques de código) para ir creando nuestro programa en base a esos bloques de código que ya tenemos programados.

```
// Suma Procedimental
let suma = 0

for (let i = 1; i <= 10; i++){
    suma = suma + 1
}

////////////////

// Suma funcional
// 1 - Definimos la función
function sumar_los_diez_primeros_enteros(){
    let suma = 0

    for (let i = 1; i<=10; i++){
        suma = suma + i
    }

    return suma
}

// 2 - Utilizamos la fucnión
let suma = sumar_los_diez_primeros_enteros()
```

Un paradigma de programación procedimental llevará muchas más líneas (en general, no para este caso) que el funcional, a pesar de que el procedimental será más fácil de entender.

1.3. NIVELES DE LENGUAJES DE LA PROGRAMACIÓN

Los lenguajes de programación se clasifican también en función del nivel de especificidad que tenga cada uno de ellos. Cuanto más bajo sea el nivel, más características específicas podré tocar yo dentro del hardware, dentro del ordenador.

0. ***Lenguaje Máquina:*** lenguaje a base de 1 y 0 (en base a transistores, encendido o apagado). Nosotros no podemos programar a este nivel.
1. ***Lenguaje Ensamblador:*** Lenguaje intermedio entre la máquina y el ser humano. Lenguaje con características muy específicas que podemos aprender aunque sea muy muy complejo. Fue el primer lenguaje de programación que existió. Estas instrucciones específicas se convertirán a lenguaje máquina para que nuestro ordenador sea capaz de entenderlo.
2. ***Bajo Nivel:*** Relacionados directamente con el hardware y la arquitectura del sistema que estemos utilizando. Son el lenguaje ensamblador y el lenguaje máquina.
3. ***Medio-Bajo Nivel:*** Cuando todo se hizo más accesible al público surgieron este tipo de lenguajes como es el caso de C y de C++. Estos lenguajes tienen capacidades de alto nivel (legible para el ser humano) y también tienen capacidades de bajo nivel (acceso a registros de memoria).
4. ***Medio-Alto Nivel:*** Todo el resto de lenguajes de programación. Son más modernos, entre los que encontramos: PHP, Java, JavaScript, C# (versión moderna de C y C++), Python.
5. ***Alto nivel:*** Frameworks basados en lenguajes de programación de medio nivel.

1.4. PROCESO DE CONVERSIÓN

Nuestro ordenador solo entiende unos y ceros. Entonces, necesitamos un traductor de nuestras instrucciones en cualquier lenguaje de programación (código fuente) a unos y ceros (lenguaje máquina).

Existen dos formas de convertir una aplicación creada por nosotros. Existen lenguajes de programación compilados (C++) e interpretados (Python).

- ***Lenguajes compilados:*** Requerimos de un compilador. Es una especie de 'programa' que transforma nuestro código. Se traduce de golpe todo junto. En el caso de que exista algún error, no se ejecuta la salida (error compiling) y volvemos directamente a nuestro código fuente.
- ***Lenguajes interpretados:*** Requerimos de un intérprete que interpreta nuestro código fuente en tiempo real y no necesita que nuestro programa esté compilado. Se traduce línea a línea. En caso de error, es ya en la salida dónde nos aparecen los mensajes de error.

1.5. IDES

IDE = Entorno de Desarrollo Inegrado o integrated development environment.

Son necesarios a medida que tienes un código grande. Son programas con características y capacidades que nos ayudan a la hora de programar. Algunos ejemplos:

- Visual Studio Code.
- Atom.
- PyCharm.
- Sublime Text.
- Notepad++.
- CLion.
- InteliJ.
- Dev++.
- Visual Studio.

1.6. CONTROL DE VERSIONES

Si soy yo el único que está programando podríamos copiar y pegar nuestro trabajo en distintas carpetas aunque es medio incómodo. El problema viene cuando no soy el único que trabaja en este programa: Si yo hago un cambio y mi compañero está haciendo otro cambio de forma simultánea, tiene que haber alguna forma de que no nos entorpezcamos y de poder recuperar versiones anteriores del proyecto.

Versiones anteriores que se utilizaban como control de versiones fueron CVS (Concurrent Version System) o SVN (SubVersion) con algunas mejoras.

Nuestra solución hoy en día es Git: Es un sistema moderno para el control de versiones. Además existen dos plataformas que utilizan todo el sistema de Git por detrás, estas son: GitHub y GitLab. Son herramientas que nos ayudan a tener repositorios con distintas versiones de

nuestros proyectos. GitLab es muy similar a GitHub sólo que GitLab se centra más en repositorios privados mientras que GitHub en repositorios públicos.

Además de las 'versiones' o mejor dicho commits, otra utilidad es la de las ramas que nos permiten hacer estas herramientas. Podemos tener una rama principal y otra de desarrollo con un equipo que está desarrollando una nueva funcionalidad, lo que podemos hacer es incorporar esta nueva funcionalidad a la rama principal mediante un merge.

2. LENGUAJES DE PROGRAMACIÓN

Lenguajes de programación:

1. Python: Es el lenguaje de programación más utilizado actualmente. Es bastante generalista y se utiliza para todo.
2. Java: Es el más utilizado por las grandes corporaciones. Lenguaje de programación base para aplicaciones Android. Android Studio utiliza un lenguaje muy similar a día de hoy.
3. JavaScript: Lenguaje de programación para web más utilizado. Las grandes corporaciones cada vez lo utilizan más. Interesante el hecho de utilizar un solo lenguaje tanto del lado del servidor como del cliente.
4. TypeScript: Basado en JavaScript. Es lo mismo que JavaScript pero con una serie de funcionalidades y restricciones (tipado) extras. Desarrollado y mantenido por Microsoft.
5. C#: Derivado de C y C++. Es muy versátil. Se utiliza para backend, para videojuegos (unity) y para apps de Microsoft. Lenguaje algo más complejo.
6. ASP.NET: Para aplicaciones web, APIs, servicios a través de Azure y aplicaciones en tiempo real. Creado por Microsoft. Muy muy orientado a la web.
7. PHP: Utilizado sobretodo en web (backend / servidores), compite directamente con JavaScript y Python. Mucho apoyo de la comunidad por la cantidad de uso.
8. HTML: No es lenguaje de programación como tal. Es un lenguaje de etiquetado, regulado por W3C (World Wide Web Consortium). Es un lenguaje que deben interpretar todos los navegadores. Utilizado para crear sitios web y aplicaciones web estáticas. Muy sencillo de aprender.
9. CSS: Dedicado también a la web. Sirve para dar estilo a nuestro HTML. Mantenido también por W3C. Significa Cascading Style Sheets. Forma parte de la tríada web HTML – CSS – JS.
10. Ajax: También para la web. Sirve para complementar las características de JavaScript dentro de una página web. Significa Asynchronus JavaScript And Xml. Es una técnica para desarrollar aplicaciones web asíncronas (permite mantener el HTML en StandBy viéndose, esperando a recibir la información de la BDD sin tener que recargar la página).
11. Ruby: Se utiliza básicamente para servidores. Ruby on Rails es un framework, no es lo mismo, este servidor sirve para realizar aplicaciones web (Twitch, Twitter…).
12. Perl: Originalmente estaba desarrollado para la manipulación de texto. Hoy en día se utiliza para la administración de sistemas, desarrollo web y para el desarrollo de GUI o interfaces gráficas para el usuario.
13. Dart: Se utiliza para web, servidor y móvil, su sintaxis es del estilo C y es relativamente complejo de aprender y tiene una comunidad relativamente pequeña.

14. Kotlin: Para desarrollo de aplicaciones móviles, sobretodo de Android. Se desarrolló para hacerle la competencia a Java y tener las mismas funcionalidades que este pero siendo mucho más sencillo. Puede interactuar con Java.
15. Swift: Para desarrollo de aplicaciones exclusivas para iOS, mantenido por Apple. Tiene soporte solo a partir de iOS7.

Lenguajes más demandados/utilizados actualmente según StackOverflow:

1. JavaScript.
2. HTML/CSS.
3. Python.
4. SQL.
5. Java.
6. Node.js
7. TypeScript.
8. C#.
9. Bash/Shell.
10. C++
11. PHP.
12. C.
13. PowerShell.
14. Go.
15. Kotlin.
16. Rust.
17. Ruby.

3. PARADIGMAS DE PROGRAMACIÓN

3.1. PROGRAMACIÓN ESTRUCTURADA

Siempre se empieza a aprender por la programación estructurada. Consiste en una secuencia de instrucciones para que vaya en orden ejecutándose una detrás de otra (orden de arriba abajo).

Tiene problemas cuando queremos hacer proyectos más complejos en los que podemos ejecutar muchas cosas a la vez, tener un abanico amplio de opciones o filtrar en función de distintos parámetros.

3.2. PROGRAMACIÓN ORIENTADA A OBJETOS

Lo más importante en este paradigma de programación son los objetos. El objetivo es que estos objetos se interrelacionen entre ellos.

Estos objetos tienen características o atributos y funciones o métodos.

Las clases son un template creador de un mismo tipo de objetos. A partir de las clases, que son plantillas, se crean muchos objetos del mismo tipo (a esto se le llama instanciar, por eso los objetos también se conocen como instancias) y estos objetos se diferencian en los valores de los

atributos, aunque luego tengan todos las mismas funciones ya que se han creado a partir de la misma clase.

CONCEPTOS CLAVE DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

- Abstracción:
- Encapsulamiento:
- Herencia:
- Polimorfismo:

Definir mejor estos conceptos.

3.3 PROGRAMACIÓN FUNCIONAL

Se pude utilizar junto con tanto programación estructurada como con programación orientada a objetos. Consiste en agurpar bloques de código en funciones que cumplen ciertas funcionalidades que nos interesan en nuestro código. Los agrupamos en estos bloques para poder llamar a estas funciones cuando queramos y ejecutarla siempre que queramos sin tener que volver a escribir todo el código que hay dentro de ella.

Existen funciones puras (recomendables) siempre que se pasan los mismos atributos devuelven el mismo resultado y no cambian los valores de ninguna variable global. Justo lo contrario en las funciones no puras.

```
// Función no pura.
let total = 0
function suma(num1, num2) {
    total = total + num1 + num2
    return total
}

console.log(suma(3, 8)) // Devuelve 11
console.log(suma(3, 8)) // Ahora devuelve 22
console.log(suma(3, 8)) // Ahora devuelve 33
```

```
// Función pura
function suma(num1, num2) {
    return num1 + num2
}

console.log(suma(3, 8)) // Devuelve 11
console.log(suma(3, 8)) // Devuelve 11
console.log(suma(3, 8)) // Devuelve 11
```

Existen también las funciones sin estado, que son funciones que no tienen variables internas ni estado interno. Para esto utilizamos la recursividad, es decir, la propia función se llama así misma.

```
// Recursividad
// Cálculo factorial de un número entero
// Factorial de 5 = 5 * 4 * 3 * 2 * 1 = 120

// Función recursiva sin estado interno.
function factorial_rec(num) {
    if (num == 1) return 1
    return num * factorial_rec(num-1)
}

console.log(factorial_rec(5)) // Mira a ver si devuelve 120

// Función no recursiva con estado interno.
function factorial(num) {
    let fact = num
    for (let i=num-1; i>0; i--) {
        fact = fact * i
    }
    return fact
}

console.log(factorial(5)) // Devuelve 120
```

También podemos crear funciones que incorporen otras funciones dentro de la misma. Una función que hace uso exclusivamente de otras funciones puras, sigue siendo una función pura.

4. DESARROLLO WEB

4.1. FRONTEND Y BACKEND

El frontend es el diseño o la cara visible de un sistema, un negocio o de cualquier página web, mientras que el backend se encarga de dar toda la información necesaria al frontend. El backend es toda la parte que no vemos directamente. El frontend es exclusivamente la cara visible y la representación de todos los datos que nos proporciona el backend, en cambio, el backend se encarga de todos los procesos que hay por detrás (bases de datos, APIs públicas, cálculos, manejo de los datos, conversión de datos…).

Cuando nosotros buscamos algo, hacemos una llamada a un servidor. Éste hace todos los cálculos que tenga que hacer, como por ejemplo acceder a bases de datos y devolver toda la información a nuestro dispositivo o navegador.

Nosotros como proovedores de servicios queremos que la mayor parte de los cálculos se encuentren en el servidor (backend) para no depender de malas conexiones a internet o dispositivos de baja gama que puedan tener nuestros clientes y la información llegue muy servida al cliente.

Por último, el desarrollador Full-Stack se refiere a aquél que tiene conocimientos tanto en el frontend como en el backend.

4.2. LENGUAJES DE SERVIDOR Y DE CLIENTE

Lenuajes del lado del servidor:

- Java
- Ruby
- PHP
- Python
- C#
- ASP.Net
- Perl
- Node.js

Lenguajes del lado del cliente:

- HTML
- CSS
- JavaScript
- TypeScript
- Ajax

Estos lenguajes se conectan entres sí.

4.3. CMS (CONTENT MANAGEMENT SYSTEM)

Un CMS es un software que ya lo tiene todo, tiene su propio backend y su frontend. Son un montón de funcionalidades que ya vienen prefabricadas y que tú puedes adaptar a tu gusto. Sería como el no-coding de hacer páginas web. Este tipo de sistemas nos permiten muchísima agilidad a la hora de crear páginas web o sistemas bastante estándar. Algunos ejemplos:

- WordPress
- Joomla
- Drupal
- Magento
- PrestaShop
- Shopify
- Odoo

Los 5 primeros son los más parecidos entre sí y están programados en PHP y JavaScript. Shopify está creado en HTML, CSS y JavaScript y está muy enfocado en la creación de tiendas online. Y odoo está programado en Python y JavaScript y prácticamente está enfocado al CRM (Client Relationship Manager), es decir, un software de gestión de clientes aunque también tiene su parte de CMS.

4.4. FRAMEWORKS

Básicamente son un conjunto de librerías que nos facilitan y complementan nuestros lenguajes de programación con algún fin específico, en este caso, el desarrollo de páginas web.

Desde el punto de vista del cliente (para crear la interactividad con el usuario o el diseño), tenemos los siguientes frameworks:

- AngularJS (JavaScript)
- VueJS (JavaScript)
- ReactJS (JavaScript)
- Spring (Java)
- Laravel (PHP)
- Microsoft.Net (C#)
- Flask (Python)
- Django (Python)

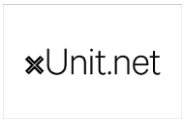



Desde el punto de vista del servidor:

- ExpressJS (JavaScript)
- AdonisJS (JavaScript)

Frameworks de pruebas unitarias (Test unitarios). Son utilizados para asegurarnos de que nuestra página queda bien, para hacer esto estamos obligados a realizar pruebas unitarias y asegurarnos de que nuestros cambios no afecten a las funciones ya existentes y que las nuevas funcionen. Para ello es imprescindible usar este tipo de frameworks en nuestras aplicaciones:

- Jest (JavaScript)
- JUnit5 (Java)
- PHPUnit (PHP)
- nUnit (.NET)
- xUnit.net (.NET)
- Pytest (Python)



4.5. LIBRERÍAS

Vamos a ver las librerías que se utilizan desde el punto de vista del cliente, en concreto desde el punto de vista de HTML:

- JQuery
- React
- Bootstrap

JQuery y React nos ayudan a trabajar con HTML con DOM (Document Object Model), es decir, a manipular la parte visible del HTML. React se puede utilizar como librería además de como framework. Bootstrap es la más utilizada que nos ayuda con los estilos (CSS).

4.6. SERVIDORES WEB

Un servidor no deja de ser un software que se utiliza para servir páginas web u otros sistemas. Servidores web más utilizados:

- Apache
- Nginx
- Tomcat
- OpenLiteSpeed
- NodeJS
- Microsoft-IIS (Internet Infromation Service)

Estos servidores nos sirven páginas web, para conexiones FTP (transferencia de archivos), también como servidor de correo electrónico o como servidor de base de datos.

4.7. NAVEGADORES

No podemos hablar de desarrollo web sin hablar de los navegadores. Los principales navegadores son:

- Microsoft Edge (Antigüo Internet Explorer)
- Google Chrome
- Safari
- Mozilla Firefox
- Opera
- Brave: Centrado en el mundo de las cryptos.

4.8. STACKS

Los stacks son conjuntos de tecnologías que nos permiten crear una página web en modo Full-Stack (cliente y servidor). En función de la tecnología que nosotros queramos utilizar tenemos los siguientes stacks:

- LAMP (Linux, Apache, MySQL, PHP/Perl/Python)
- WAMP (Windows, Apache, MySQL, PHP/Perl/Python)
- XAMPP (Multiplataforma, Apache, MySQL, PHP/Perl/Python)
- MERN (MongoDB, ExpressJS, React, NodeJS)
- MEAN (MongoDB, ExpressJS, Angular, NodeJS)
- MEVN (MongoDB, ExpressJS, Vue, NodeJS)
- PERN (PosgreSQL, ExpressJS, React, NodeJS)

5. DESARROLLO MÓVIL

5.1. PLATAFORMAS MÓVILES

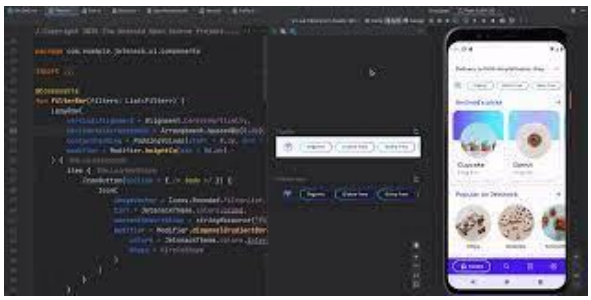
Las dos plataformas móviles más importantes y más utilizadas a día de hoy son Android e iOS. Androit tiene el 74.36% del mercado móvil mundial e iOS el 22.84%. Por eso nos vamos a centrar en estas dos plataformas móviles. La mayoría de personas tienen un móvil (más que

ordenadores y otros dispositivos) por eso es importante que nuestros desarrollos se vean bien en los smartphones.

5.2. IDES Y EMULADORES PARA DESARROLLO MÓVIL

- XCode para iOS.
- AndroidStudio para Android.

Cada uno tiene su propio emulador, es decir, podemos tener un emulador en tiempo real de un dispositivo Android/iOS. Ejemplo:



Para el desarrollo móvil, es mucho mejor utilizar este tipo de IDEs ya que vienen mucho más preparados que VSCode por ejemplo.

5.3. LENGUAJES DE DESARROLLO MÓVIL

Android Studio programa para Android y utiliza tanto Kotlin como Java como lenguajes de programación. Estos lenguajes son compatibles. En cambio, si queremos realizar aplicaciones excluivas para iOS utilizaremos Swift dentro de Xcode.

5.4. FRAMEWORKS Y LIBRERÍAS

Los cuatro frameworks más utilizados a día de hoy son:

- Xamarin
- Flutter
- React Native
- Ionic

Ionic es un framework muy enfocado en desarrollo móvil, desarrolla aplicaciones web basadas en el framework Angular mayoritariamente (también React y Vue) y además utiliza lo que se conoce como un WebView (Si yo quiero que una aplicación sea responsive tendré que hacer que se vea bien en dispositivos móviles). WebView es un simulador web mientras desarrollamos para poder ejecutar la web correctamente en un móvil. Además, utiliza un Bridge (Puente) –> Native, para hacer la aplicación y que tenga funcionalidades nativas como por ejemplo poder acceder a la cámara del móvil o a la ubicación. Para esto último utiliza Apache Cordoba.

Los otros tres frameworks se diferencian sobretodo en el lenguaje de programación utilizado: Xamarin utiliza C# con .Net, Flutter utiliza Dart y React Native utiliza JavaScript con React. Estos tres se diferencian de ionic es que cuando nosotros creamos aplicaciones en estos frameworks estamos compilando directamente nuestra aplicación nativa (creamos código nativo) tanto para Android como para iOS, mientras que ionic utiliza un emulador web a través del cuál nosotros podemos ejecutar la aplicación. A día de hoy los que más se utilizan son ReactNative y Flutter.

6. DESARROLLO MULTIPLATAFORMA

Consiste en crear un solo código y que luego sea la compilación la que cree diferentes tipos de archivos para distintos sistemas operativos.

6.1. PLATAFORMAS

Veamos las más importantes:

- iOS
- MacOs
- Windows
- Linux
- Java (JVM – Java Virtual Machine) (Minecraft)
- Android


6.2. LENGUAJES DE DESARROLLO MULTIPLATAFORMA

Al igual que para otras utilidades, los lenguajes más utilizados para el desarrollo multiplaforma son:

- JavaScript
- Java
- Python
- C#
- Ruby
- Dart



Estos lenguajes por sí mismos no quieren decir nada de que estemos desarrollando aplicaciones multiplataforma, sino que nosotros utilizamos diferentes frameworks o librerías para hacer este tipo de aplicaciones.

6.3. FRAMEWORKS DE DESARROLLO MULTIPLATAFORMA

Los más utilizados son:

- ApacheCordova (aplicación externa de soporte y ayuda)
- ReactNative (JavaScript)
- Appcelerator (JavaScript)
- NativeScript (JavaScript)
- Kivy (Python)
- BeeWare (Python)



- Flutter (Dart)
- CodeNameOne (Java)
- RubyMotion (Ruby)

Los más demandados siguen este orden: Flutter → Ionic → ReactNative → Xamarin → NativeScript → Appcelerator.

En conclusión, preferimos utilizar frameworks para un desarrollo multiplataforma y así utilizar un solo código para distintos sistemas operativos. A no ser que queramos entrar a una empresa grande en la que quieran utilizar aplicaciones nativas.

7. CODE REVIEW

Code Review o Revisión de código consiste en copiar el código de una versión y hacer un Pull Request que incorpore una nueva funcionalidad. Al hacer este Pull Request, entra en juego el Code Reviewer que revisa este código y lo compara con versiones anteriores del código. Esta persona hace una serie de comentarios y puede aceptar o rechazar el código. Cuando todo está correcto y el Code Reviewer y el desarrollador del mismo están de acuerdo, esta nueva copia o rama de código pasa a ser la nueva versión del código.

Ahora veamos las plataformas que de forma automática nos ayudan a revisar ese código:

- CodeFactor
- Collaborator
- Codestriker
- SonarQube
- Gerrit
- GitLab
- GitHub



La revisión se hace teniendo por un lado la versión nueva del código y por otro la vieja y hacer una comparación y ver las modificaciones.

8. BASES DE DATOS

8.1. BASES DE DATOS Y PROCESOS DE DESARROLLO

Una base de datos es un sistema que nos permite almacenar información desde un punto de vista lógico. La base de datos más típica es una base de datos en formato tabla (como un Excel).

Fases de diseño de una base de datos:

- ***Diseño Conceptual:*** Consiste en crear toda la idea de la base de datos, si va a tener relaciones y todo lo demás… Esta fase es totalmente acnóstica a la tecnología que vayamos a utilizar.
- ***Diseño Lógico:*** Cuando ya tenemos el diseño conceptual bien definido, elegimos el proveedor que más nos interese o el tipo de base de datos que utilizaremos y empezaríamos a convertir el diseño concepual en algo más adaptado a la tecnología que vamos a utilizar. Por ejemplo, establecer los tipos de variables.

- ***Diseño Físico:*** Por último, lo que hacemos en esta fase del desarrollo es crear realmente la base de datos en la plataforma que hayamos escogido. Creamos cada una de las tablas o modelos en la plataforma en la plataforma en la que hayamos creado la base de datos.

Pese a que muchas veces se empieza directamente por la fase 3, es muy recomendable enfocarse mucho en la primera y desarrollar bien la base de datos de forma coceptual para evitar posibles errores y dolores de cabeza a la hora de picar código.

8.2. BASES DE DATOS SQL Y NO SQL

Tenemos dos tipos de bases de datos que engloban todo:

- ***Bases de datos SQL:*** Este tipo de bases de datos consiste en tablas de datos. Se conocen también como bases de datos relacionales ya que podemos crear relaciones en base a claves de otras tablas. Los tipos de bases de datos SQL que más se utilizan en el mercado son: MySQL, MariaDB, PostgreSQL, Oracle Database y SQLite (sólo en Python).

- ***Bases de datos no SQL:*** También se conocen como bases de datos no relacionales. Existen muchos tipos de bases de datos no SQL. Podríamos tener ejemplos como las bases de datos del tipo Key-Value (no son muy grandes y son muy rápidas, suelen usarse en memoria RAM), bases de datos del tipo objetos o documentos, en este tipo de base de datos no existe ninguna relación sino que tenemos datos almacenados sin más (se suelen utilizar generalmente para prototipos o para pruebas de conceptos ya que tienen mucha más flexibilidad aunque con menos restricciones). El último ejemplo es el de las bases de datos del tipo GRAFOS donde se pueden relacionar nodos entre sí o no (lo más típico para una base de datos de una red social). Los proveedores de bases de datos no SQL más utilizados en el mercado son: mongoDB (documentos), Cloud Firestore (documentos), cassandra (Key-Value), redis (Key-Value) estas dos son muy rápidas, GraphQL (Grafos).

9. DEVOPS/CI/CD

Las DevOps son fundamentales en el mundo del desarrollo. Significa Development IT Operations. Las DevOps es una metodología de Desarrollo de software que consiste en ir desarrollando y publicando código de poco en poco. Tanto el equipo de desarrollo de software como el de IT deben trabajar codo con codo de forma casi diaria. Es decir, que todos los cambios se vayan implementando poco a poco. De esta manera, los dos equipos tienen la posibilidad de echar para atrás a una versión anterior y que todo siga funcionando correctamente, es mucho más fácil llegar a un entendimiento entre los dos equipos de esta forma.

Esta metodología necesita una serie de facilidades. Aquí entran los conceptos de CI y de CD:

- CI: Continuous Integration.
- CD: Continuous Deployment.

CI es una funcionalidad que lo que hace es publicar el código a través de un merge/pull/push. Mientras que el CD es desplegar la aplicación a producción para que ésta se pueda utilizar. No es lo mismo pero sí que uno va detrás del otro.

El CI/CD es una pequeña automatización que se hace antes de poder subir una nueva versión de nuestro código a producción. CI está entre la versión de desarrollo y la versión de preproducción y CD entre la versión de preprodcucción y la versión de producción.

Herramientas o servicios que ofrecen este tipo de soporte CI/CD:

- Ansible.
- Helm.
- Jenkins.
- GitLab CI.
- GitHub Actions.
- Bitbucket Pipelines.

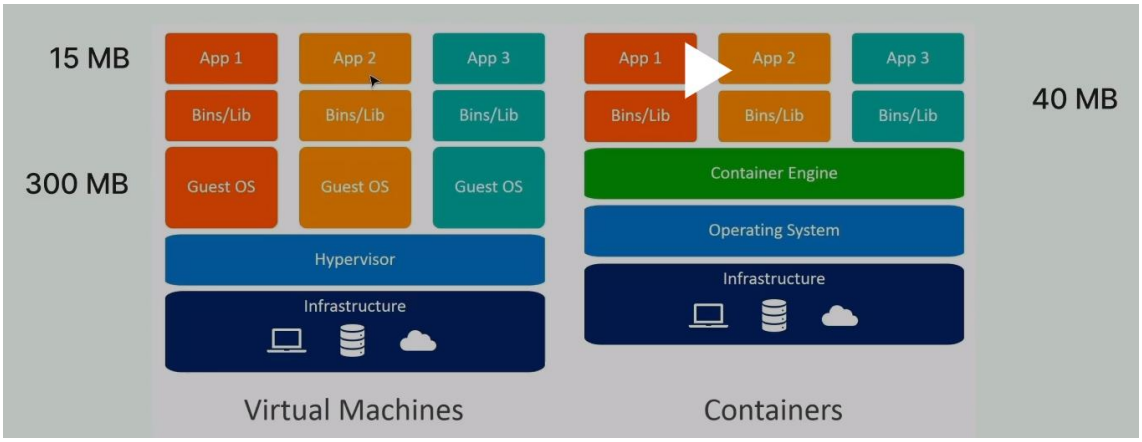
10. CONTENEDORES Y ORQUESTACIÓN

10.1. CONTENEDORES

Los contenedores son una especie de encapsulamiento donde nosotros introducimos la aplicación o el código fuente, también incluimos en este contenedor las librerías o dependencias que necesitamos para ejecutar la aplicación. Por último, introducimos las instrucciones del sistema operativo que tiene que utilizar (no el sistema operativo completo).

De esta forma, nosotros estaríamos generando un contenedor que podríamos ejecutar en cualquier máquina. Una vez tengo éste contenedor se lo podemos pasar a un amigo y él a través de un software que pueda leer e interpretar ese contenedor, se ejecute la aplicación como se ejecutaría desde cualquier máquina.

La forma de trabajar con contenedores es algo diferente. Mejora sustancialmente en el espacio requerido de infraestructura a la hora de desarrollar diferentes aplicaciones.



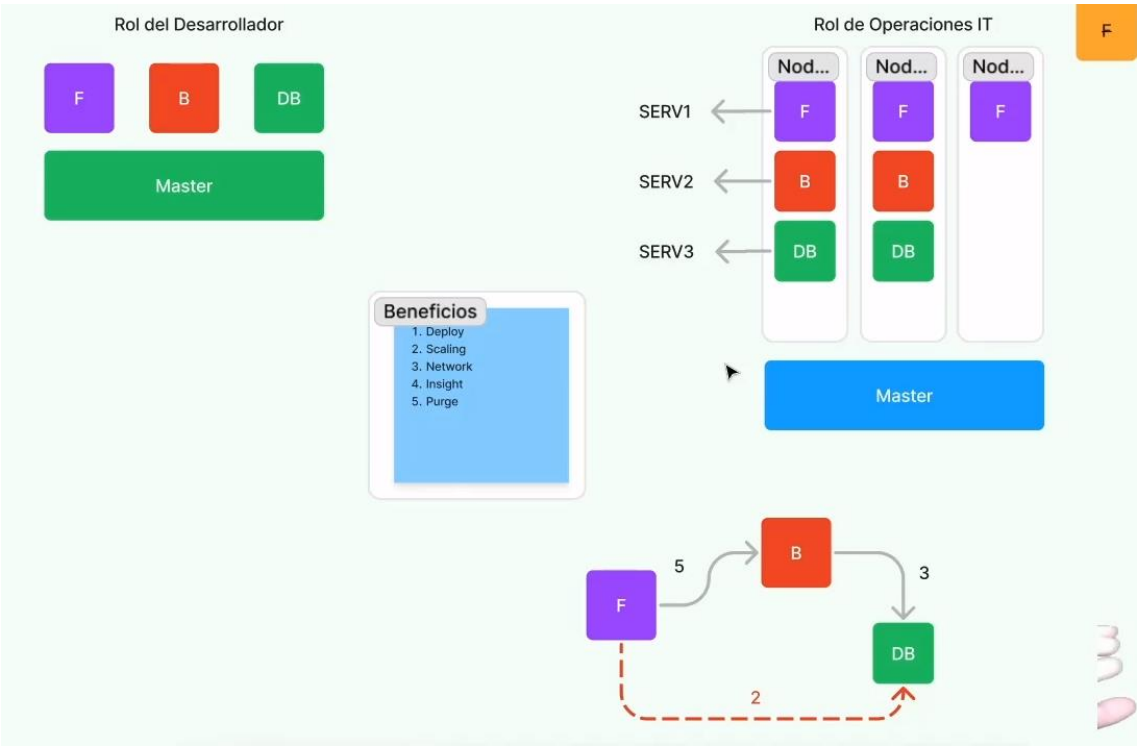
Los tipos de contenedores más conocidos y utilizados son los dos siguientes:

10.2. ORQUESTACIÓN

- Rol del Desarrollador: Consiste en desarrollar un contenedor con la aplicación del frontend, otro del backend y otro para la base de datos. Luego en producción se conectan. Necesitamos ahora que estos tres contenedores estén conectados, subidos y podamos acceder desde cualquier parte del mundo. Aquí entra en juego el Master que se encargará de coger los contenedores y llevarlos a un sistema. Ejemplo: El front está colapsando y debería duplicar los recursos (tarea un tanto ardua de realizar), por esta razón existen los orquestadores que ayudan con el despliegue, la escalabilidad, el Network y con las Insights o reportes internos de comportamiento del conjunto de contenedores.
- Rol de Operaciones IT (modelo de orquestación): En vez de tres contenedores, tendremos 3 nodos. En el primero encontraríamos todo lo que hemos incluido en los 3 contenedores del rol del desarrollador. Ahora el master nos da facilidad a la hora de desplegar los contenedores. Para el ejemplo de antes, el master nos ayuda también a crear otro nodo con un clon de la misma aplicación y de esta forma ESCALAMOS la aplicación de una forma mucho más sencilla con un servicio de orquestación. Lo siguiente que me permite es crear un nodo para cada una de mis aplicaciones o contenedores (de los del rol de desarrollador) y tener un servicio centrado para cada una de las partes de la aplicación independientemente del nodo en el que trabajemos. Por último, también nos permite, a través de los Insights, saber cuántas conexiones hay en tiempo real con cada uno de nuestros servicios (descritos anteriormente) y así podemos saber si hay conexiones que no se deberían dar y demás (por ejemplo del front a la base de datos) por temas de seguridad. Además, si algo no funciona, el sistema de

orquestación nos permite eliminar algo que no funcione y sustiuirla por otra que si lo haga.

MIRAR VÍDEO PARA MÁS CLARIDAD:



SERVICIOS COMERCIALES

11. CLOUD COMPUTING

Tradicionalmente cuando yo desarrollo una aplicación y quiero mostrarla al mundo tenemos dos opciones:

- Desplegar mi aplicación en mi propio servidor físico.
- Contratar algún servicio en la nube (servicio de cloud computing). De esta forma no necesito un ordenador en casa permanentemente conectado. Desplegamos la aplicación en la nube y está constantemente en marcha.

Mantenimiento: Es más costoso el mantenimiento de un ordenador en casa que ofrezca el servidor constantemente que en el caso del cloud computing ya que el servicio que contratas en la nube son los encargados del mantenimiento.

Coste conjunto: En el primer caso tengo un ordenador potente para ser utilizado en un 1% para mantener la aplicación. Sin embargo, en la nube tenemos la opción del pago por el uso. Si

necesito pocos recursos para alojar mi aplicación, yo solo pago por esos recursos que necesito y no mucho más.

Escalabilidad: Si necesito escalar la aplicación, puede ser que necesite otro servidor u otro ordenador y encargarnos de comprar otro ordenador y demás. Mientras que la escalabilidad a través del cloud computing es mucho más sencilla ya que nosotros pagaríamos un poco más para escalar nuestra aplicación.

Seguridad: Si nosotros tenemos un ordenador o un servidor, nosotros necesitamos hacer el mantenimiento o hacer los ajustes necesarios para evitar conexiones malignas y encargarnos de nuestra propia seguridad. Sin embargo, los servicios de cloud computing se encargan de su propia seguridad por lo que esta es otra tarea de la que nosotros nos desentenderíamos.

Los tres servicios comerciales que más se usan a día de hoy son:

12. TESTING

12.1. QUALITY ASSURANCE

Quality Assurance (QA) es el departamento de control de calidad de todo lo relacionado al software. Es el departamento que se encarga de testear el software desarrollado por la empresa o la organización y además, se asegura de que cumple los requisitos o estánderes establecidos antes de iniciar el proyecto. Ejemplos:

```
test("Prueba de suma", () => {
    const a = 1;
    const b = 2;
    const resultado = a + b;

    expect(resultado).toBe(3);
})

const suma = (a, b) => a + b;
```

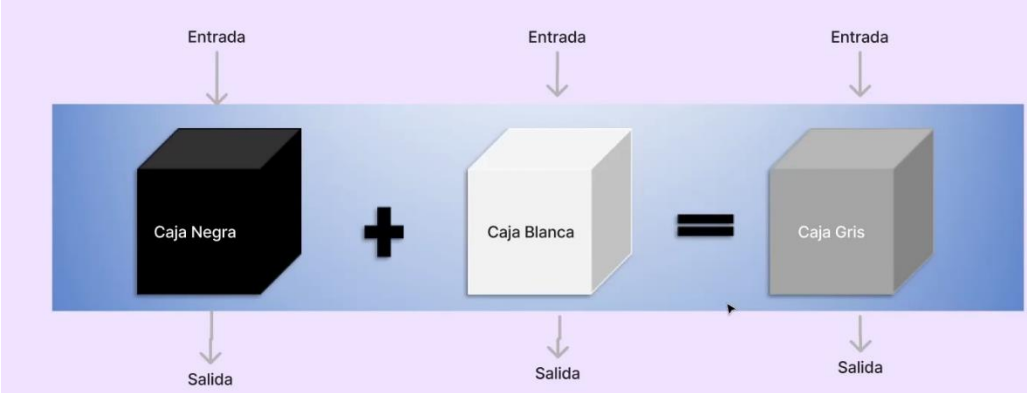
```
const suma = (a, b) => a + b;

test("Prueba de suma", () => {
    expect(suma(1, 2)).toBe(3);
})
```

Yo como responsable de control de calidad, testeo este programa y mi tarea va a ser realizar los casos de tes codificándolos de esta manera para que con un código me diga si todo funciona correctamente. Si creamos los test de manera perfecta, significa que si el programa pasa todos los test, significa que el programa o el software está bien.

12.2. TEST DE LAS CAJAS Y TIPOS DE TEST

Es una técnica que se utiliza mucho. Tenemos el test de la caja negra, el de la caja blanca y el de la caja gris. Son técnicas de testeo totalmente compatibles.

- Caja negra: Yo no se que es lo que hay dentro y realmente no es lo que estoy testeando, yo lo único que estoy testeando es la entrada y la salida. Yo quiero que para un tipo de entrada haya un tipo de salida. Por ejemplo, una interfaz. Si testeo una calculadora, lo que quiero es ver que si le doy al uno se pone un uno y que cuando hago una suma esta se ejecuta correctamente.
- Caja blanca: En este caso, nos interesa ver lo que está ocurriendo por dentro. Cuando le doy una entrada, me interesa ver cuáles son los pasos que sigue esa entrada (cálculos, llamadas…). Aunque también tendrá una salida, realmente nos enfocamos en el camino o el flujo de la aplicación por dentro.
- Caja gris: Este test es la combinación de los dos tipos de test anteriores. Nos fijamos en la entrada, el flujo y la salida de la aplicación y testeamos todo esto.



TIPOS DE TEST

Tenemos dos tipos de test:

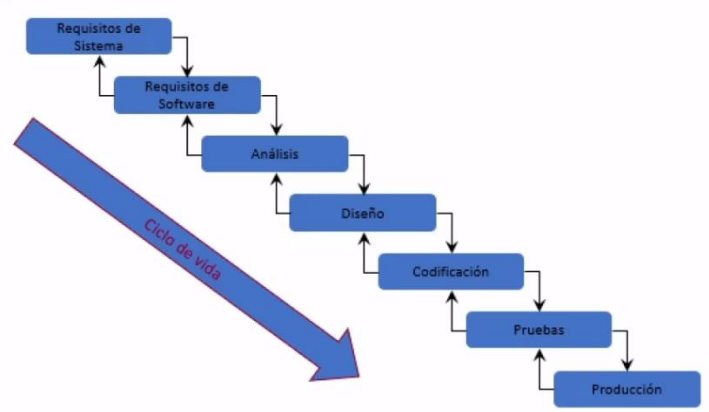
- Test funcionales: Lo que hace es testear todas las funcionalidades que se supone que nuestra aplicación debe cumplir. Aparte de la funcionalidad que hay que testear, también es necesario testear las situaciones negativas, es decir, testear los posibles errores que puedan surgir a lo largo de nuestra aplicación (desarrollar pantallas para errores que se pueden producir…).
- Test no funcionales: Hacen referencia a todos los test que hay alrededor de nuestra aplicación, como puede ser el rendimiento:
    - Tiempos de carga.
    - Stress test: maximizar los recursos que está consumiendo en ese momento (testear para 1000 usuarios en vez de 10).
    - Escalabilidad

  La accesibilidad, UX/UI (user experience y user interface), test de seguridad.

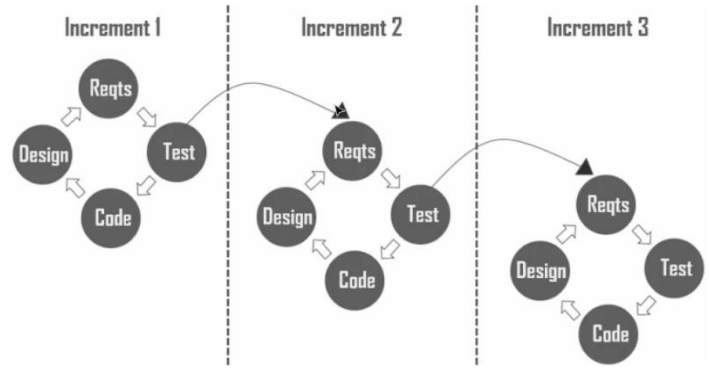
12.3. MODELOS DE TESTING

Los modelos de testing más utilizados a día de hoy son (vemos 4 aunque hay algunos tipos más):

- Modelo de testing en cascada o lineal: Lo que propone este modelo es ir testeando cada fase con respecto de la anterior. No vamos a pasar a la siguiente fase sin haber hecho una iteración o regresión de la fase anterior, vemos la imagen.

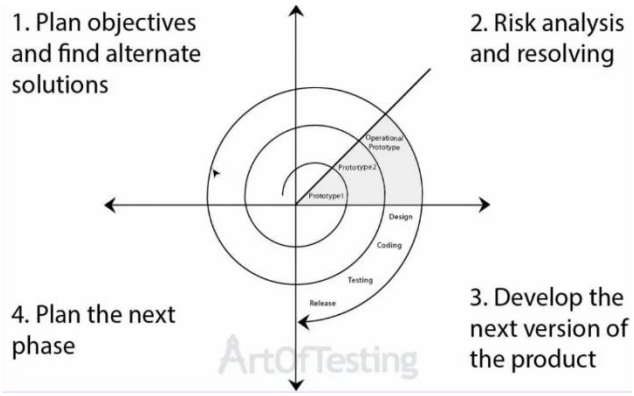


- Modelo de testing iterativo – repetitivo: Se trata cada fase del testing como un incremento, estos incrementos suelen estar definidos por sprints que duran alrededor de dos semanas. En la primera iteración se definen los requerimientos, se definen los casos de test, se codifica y se diseña. Este incremento se ejecuta durante un tiempo y se harán los test necesarios para pasar al siguiente incremento.



- Modelo de testing en V: Esto sería como una evolución del testing en cascada, donde en primer lugar tenemos la especificación de requerimientos, luego pasamos al diseño funcional, luego al técnico, luego la especificación detallada de cada uno de los componentes y luego la implementación del software. Después de esto seguimos con las pruebas de componentes que se comparan con la especificación y así sucesivamente (Mirar la foto). Vamos de fuera a dentro por la izquierda y viceversa en la derecha. El número de ciclos depende de lo que esté especificado de antemano.

- Modelo de testing en espiral: Primero definimos los objetivos y buscamos las soluciones que complementen a estos objetivos. Después realizamos un análisis de riesgo y redefinimos las soluciones. Por último, desarrollamos la siguiente versión del producto y ya viene el diseño la codificación el testeo y el despliegue del código. Una vez tenemos el código desplegado, pasamos a la planificación de la siguiente iteración (Mirar la foto).



13. COLA DE MENSAJES

Para entender el encolamiento de mensajes (Message Queueing) veamos un ejemplo:

Imaginemos que tenemos una aplicación, con una interfaz que se tiene que conectar con un backend. Entonces, el frontend se conecta con el backend y este último nos devuelve una respuesta. Pero, ¿Qué pasa si en vez de tener sólo una llamada tengo muchas a la vez? El backend devuelve respuestas de una forma normal, hasta que los recursos de este empiezan a colapsar y llega a su límite. En ese momento, la siguiente llamada que se produzca puede producir errores y esto no lo podemos permitir (preferimos un retraso que un error).

Aquí surge la cola de mensajes, que es básicamente una interfaz intermedia que nos permite almacenar en caché estos mensajes que van llegando y llevarlos de forma ordenada al backend para no colapsar el sistema. En lugar de llegar al backend, estaría llegando a una cola, y es la cola la que se conecta directamente con el backend. Es la cola la que gestiona el orden de los mensajes en función de prioridades u orden de llegada.

No todas las colas de mensaje funcionan de la misma manera, pero este es un ejemplo típico de colas de mensajes y de su funcionamiento.

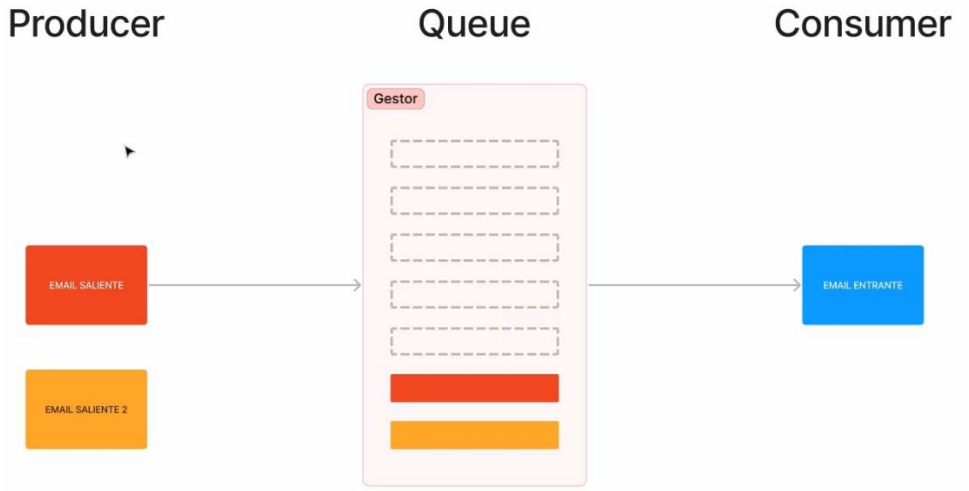
Lo que nos permite el encolamiento de mensajes es disociar las aplicaciones para que no este dependiendo continuamente el front del back sino que haya una interfaz intermedia que gestione las conexiones. También nos permite tener un sistema asíncrono (no lineal, hacemos la llamada y en el momento preciso llegará la respuesta).

¿QUÉ ES UN MENSAJE?

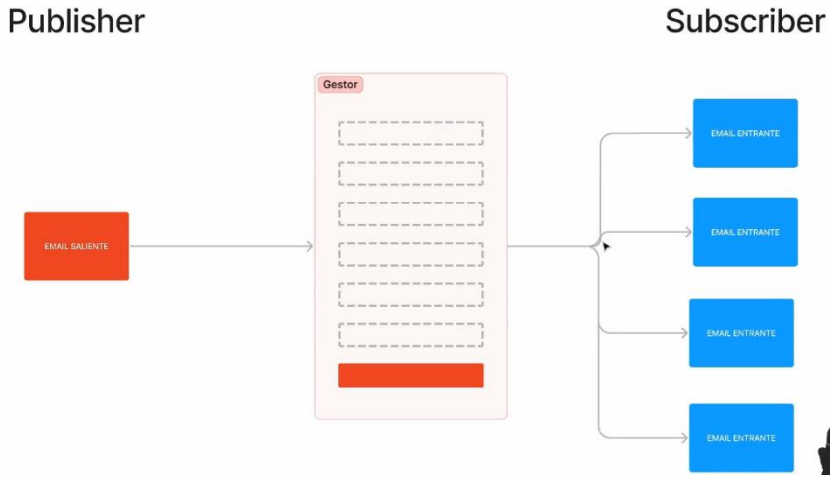
Un mensaje es una información o datos que debe ser transmitida desde un punto a otro. No tiene por qué ser un mensaje literalmente, aunque también. Algunos tipos de datos son: texto, todo tipo de archivos…

TIPOS DE ESTRUCTURAS

- Estructura punto a punto:



- Estructura publicador – suscriptor: La diferencia con la estructura anterior es una relación de 1 a n, donde n es el número de suscriptores que van a recibir el mensaje (la información).



BENEFICIOS

Lo que nos permite esencialmente es la disociación de las aplicaciones, con una interfaz intermedia que gestiona las conexiones.

Al poder disociar las aplicaciones, podemos darle una especialización a cada una de ellas, es decir, sino tuviésemos la cola de mensajes, tendríamos que crear una aplicación entera que gestione tanto el fron como el back, en cambio, disociando las aplicaciones podemos hacer aplicaciones específicas para una función.

También nos ayuda con la escalabilidad de las aplicaciones. Cuando se colapsan los recursos destinados a la aplicación, la cola de mensajes nos permite gestionar un poco mejor la aplicación, aunque esta también puede colapsar en algún momento.

Si desarrollo sin colas de mensajes, tendría que desarrollar toda la aplicación con el mismo lenguaje que no tiene porqué funcionar igual de bien tanto para el front como para el back. Con las colas no depende el back del front y puedo desarrollar ambas en distintos lenguajes.

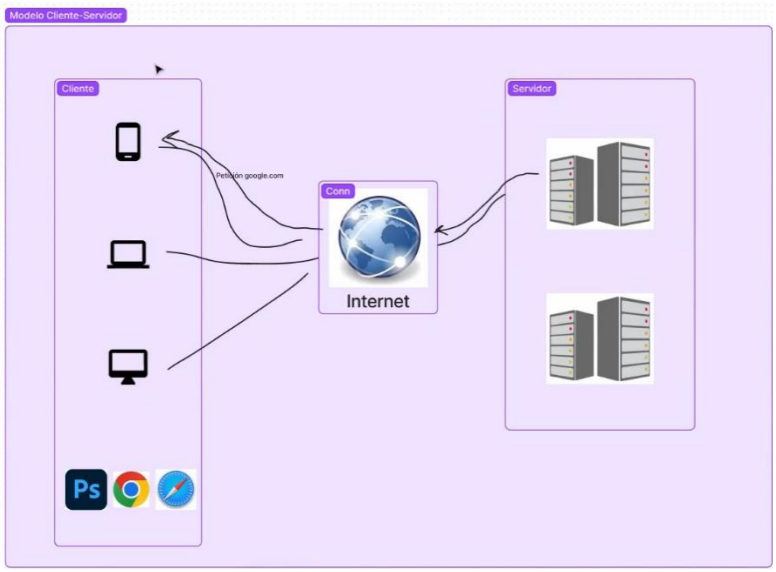
14. CLIENTE – SERVIDOR

14.1. DEFINICIONES

Este es el modelo que se utiliza en el desarrollo web.

El cliente es el dispositivo móvil, ordenador… (Esto es el dispositivo cliente). Es el dispositivo a través del cuál accedemos a internet. Nos conectamos a internet con el objetivo de obtener una cierta información. En este caso, el dispositivo le lanza una petición a internet. Este con la dirección de la petición, le lleva a un servidor, el cual trabaja la información que le está llegando y devolvérsela al cliente.

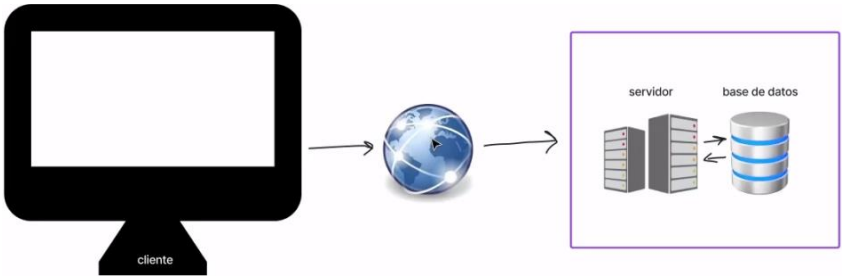
Los dispositivos clientes acceden a internet a través de diferentes aplicaciones, llamadas aplicaciones cliente, a través de las cuáles somos capaces de visitar diferentes páginas web y de interpretar el html, css y javascript que nos devuelven lo que está alojado en los servidores.



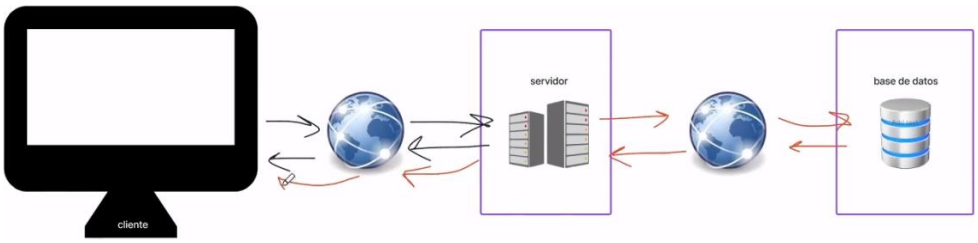
14.2. ESTRUCTURAS POR NIVELES

- **Estructura de un solo nivel:** Consiste en tenerlo todo en una sola máquina. En nuestro propio dispositivo tenemos el cliente, aparte del dispositivo, puede ser Google Chrome por ejemplo. A su vez, en nuestra máquina tenemos el servidor al que hacemos las peticiones y la base de datos.

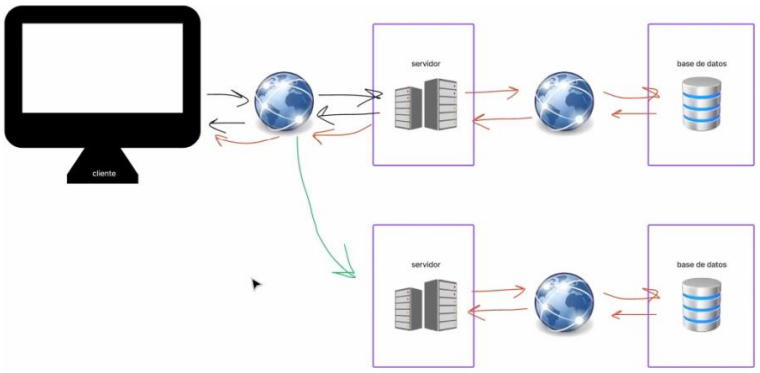
- **Estructura en dos niveles:** Tengo una máquina cliente, y este cliente a través de internet tiene acceso a otro dispositivo en el cuál están almacenados tanto el servidor que gestiona las peticiones que le llegan de internet como la base de datos. La conexión entre el servidor y la base de datos es muy rápida ya que no hay otros intermediarios.



- **Estructura en tres niveles:** En este caso tenemos una máquina cliente que se conecta a través de internet al servidor, y este puede devolver la información directamente o este se puede conectar a una base de datos a través también de internet y procesa la respuesta con la información obtenida de la base de datos.



- **Modelos de n niveles:** Son aquellos modelos en los que a través de internet, el cliente accede a diferentes servidores. Ya que puede ser que para obtener la información, necesita acceder a diferentes bases de datos las cuales se conectan a distintas bases de datos. Esto se suele usar a través de puntos Api. Api rest points.



14.3. TIPOS DE SERVIDORES Y MODELO P2P

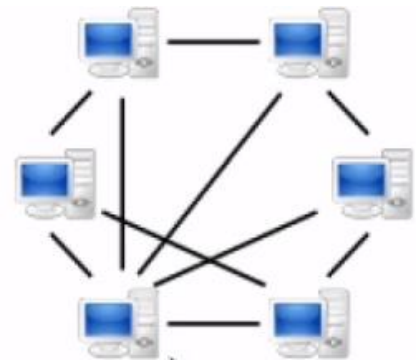
TIPOS DE SERVIDORES

Hablamos de servidores como aplicaciones. Los más típicos son:

- **Web Servers:** Aquellas aplicaciones de servidor que me sirven todos los archivos necesarios para que yo pueda renderizar una web a través de un navegador moderno.
- **File Servers:** Lo que vamos a mandar o a tener como intercambio van a ser archivos como un servidor de Mega, Google Drive o Dropbox.
- **Email Servers:** Aplicaciones que nos permiten tener un servicio de email.
- **VPN Servers:** Virtual Point of Network, nos permiten acceder a diferentes servicios que están alojados de forma local, pero remota, es decir, a través de un servidor VPN puedo acceder a aplicaciones y servicios locales de forma remota como los de la oficina desde casa.
- **Proxy Servers:** Es algo similar, solo que estás accediendo a ciertas diferencias a través de una especie de túnel y te haces pasar por otra dirección IP.

MODELO P2P

Modelo peer to peer. Es un modelo descentralizado donde el modelo cliente-servidor ya no existe. No sería tampoco cliente-cliente ya que cada dispositivo gestiona su propio servidor. Nosotros estamos conectados en red de cierta manera y tenemos acceso a los mismos datos y entonces yo si quiero acceder a cierta información, no lo hago a través de un servidor, sino que me conecto directamente a ti o a otro que tienen la misma base de datos. Es como funciona el mundo blockchain.



Verificamos la información ya que todos tenemos la misma base de datos.

15. MODELO VISTA-CONTROLADOR

Es el patrón de diseño de aplicaciones de arquitectura, más utilizado a día de hoy sobretodo en web. Todos los lenguajes y frameworks pueden adaptarse a este patrón de diseño. MVC hace referencia a Modelo Vista Controlador.

Cada una de las tres palabras hace referencia a una parte de nuestra aplicación que tiene una función muy concreta:

- **Modelo:** Tiene como función acceder y manipular los datos. Por ejemplo, el acceso a una base de datos.
- **Vista:** Mostrar los datos del modelo y proporcionar la interactividad al usuario.
- **Controlador:** Coordinar al modelo y a la vista. Es quién decide a qué partes del modelo llamar y que partes de la vista coger, es quén dirige el juego.

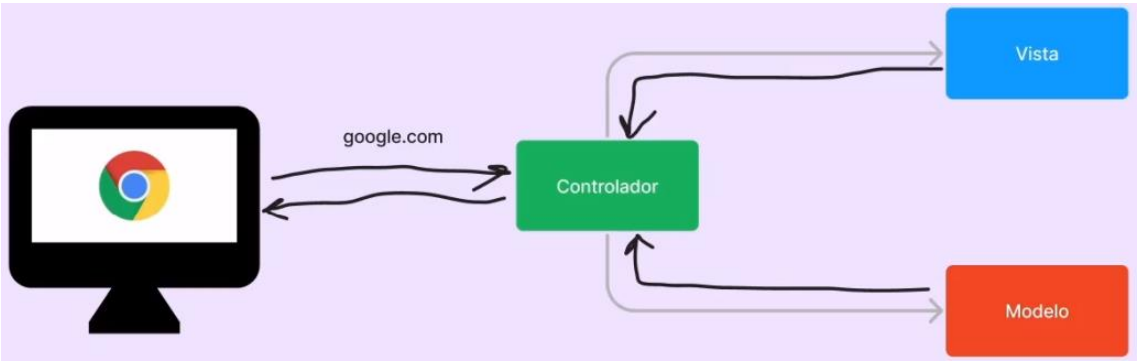
HISTOIRA DE MVC

Es un patrón de diseño que fue introducido en los años 70 pero que no fue popularizado hasta el año 88. Es el patrón de diseño más utilizado en aplicaciones web, y al ser tan popular, es normal que hayan surgido otros patrones derivados de aquí, como son:

- HMVC: Modelo Vista Controlador Jerárquico.
- MVA: Modelo Vista Adaptador.
- MVP: Modelo Vista Presentador.
- MV VM: Modelo Vista Vista Modelo.
- …

MVC

Nosotros tenemos un dispositivo, en este caso un ordenador. Enotnces enciendo y digo busca Google.com. El controlador me dice que he accedido a los servidores de Google y le dice al modelo todo lo que tiene que mostrar. El modelo sería el que accede a la base de datos con toda la información que tenemos que mostrar y le devuelve la información al controlador. El controlador ya tiene la información y hace una llamada a la vista para mostrar la información recibida del modelo de una forma ordenada. Cuando esté generada la vista se devuelve al controlador y el controlador con la vista se la muestra al cliente.



16. PROTOCOLO HTTP

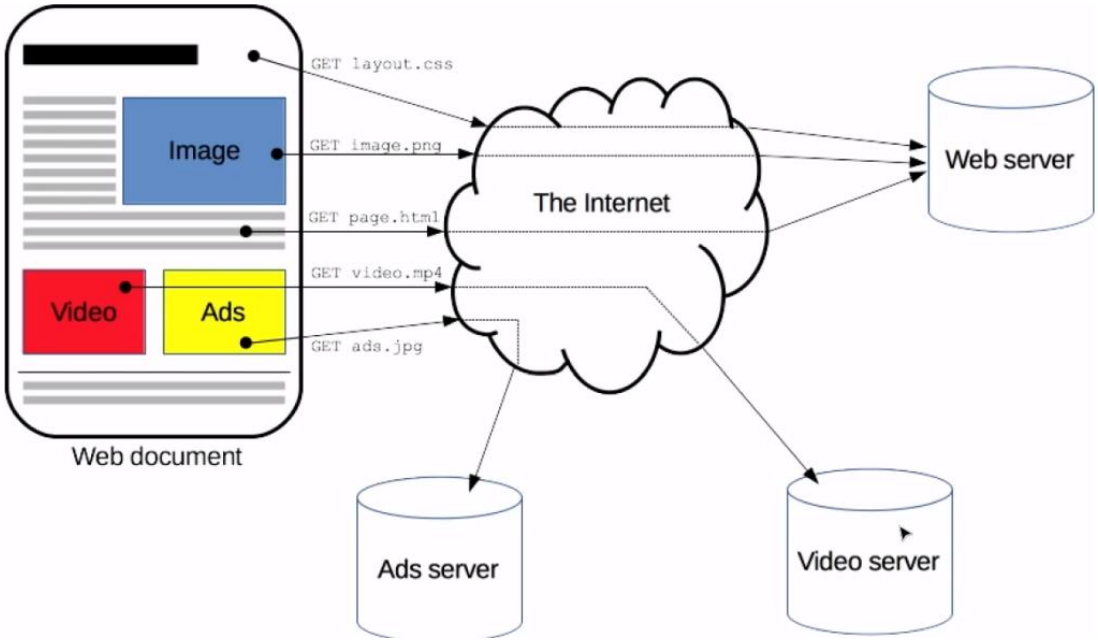
16.1. INTRODUCCIÓN A HTTP

Es el estándar de comunicaciones vía web. Lo primero que vamos a ver es como se relaciona el protocolo HTTP con el resto de cosas que hemos visto. Cuando hacemos a una conexión a internet a través de un dispositivo cliente seguimos el protocolo HTTP que hace referencia a la conexión a través de internet para obtener datos.

HTTP significa HyperText Transfer Protocol. Ya que al principio estaba destinado a la transferencia de documentos HTML. A lo largo del tiempo, el protocolo ha ido evolucionando y ha empezado a transferir CSS y JavaScript y también archivos, vídeo o música.

EJEMPLO

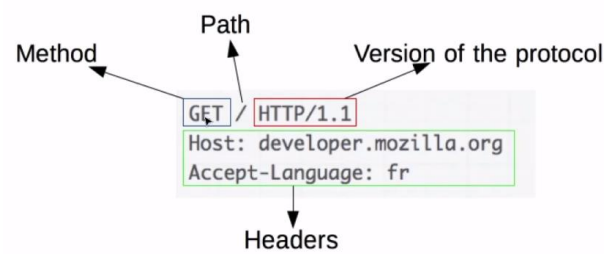
Cuando accedemos a internet, lo primero que hacemos es descargar el HTML y luego el CSS y el JavaScript. Entonces se hacen una serie de llamadas a internet a partir del método get obteniendo así diferentes partes que complementan nuestra página web. Luego, estas llamadas se hacen todas a internet y luego ya con la propia configuración del HTTP vamos a unos servidores para obtener una serie de datos.

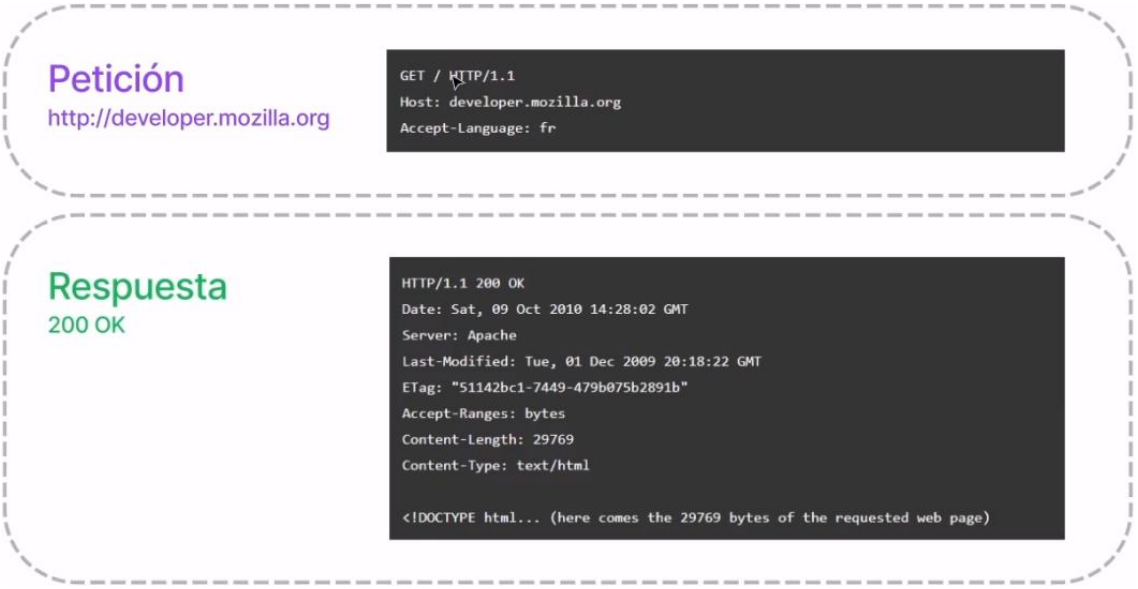


MENSAJE Y ESTRUCTURA

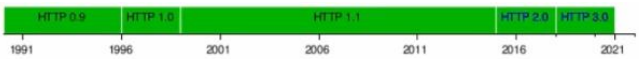
Un protocolo no es más que un conjunto de reglas estandarizadass con el fin de que diferentes dispositivos, lenguajes o tecnologías puedan conectarse entre sí. Lo ideal es tener un protocolo que permita que todos los lenguajes se puedan conectar entre sí y así hacer de internet una red global que no exluya ninguna tecnología.

La s de https hace referencia a que el tipo de conexión es cifrada.





16.2. HISTORIA Y VERSIONES DE HTTP



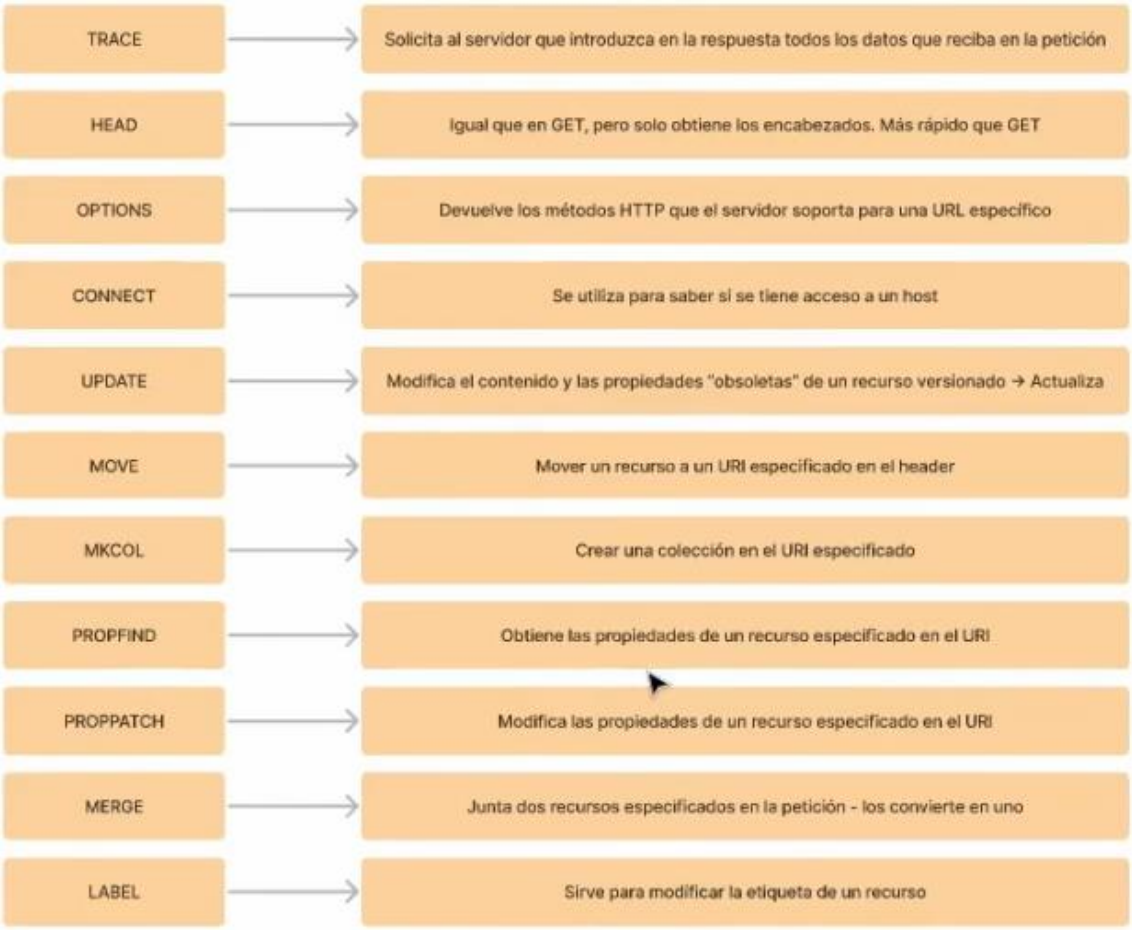
HTTP 0.9	Primera versión. Solo soporta GET. No soporta Headers
HTTP 1.0	Se usa sobre todo en servidores proxy. Métodos GET, HEAD y POST
HTTP 1.1	Más utilizado a día de hoy. Conexiones persistentes activadas por defecto. Pipelining
HTTP 2.0	Misma semántica que en la v1. Mejoras en el empaquetamiento de datos
HTTP 3.0	Última versión. Sustituye protocolo TCP por UDP

16.3. MÉTODOS EN HTTP

Existen muchos tipos de peticiones. Las peticiones son tratadas como sugerencias ya que cada servidor trabaja con cada petición como más le convenga. Las peticiones más utilizadas son:

GET	Obtener datos
POST	Enviar datos, crear nuevos recursos
PUT	Enviar datos, editar recursos existentes
DELETE	Eliminar recursos existentes
PATCH	Enviar datos, editar recursos existentes DE FORMA PARCIAL

Otras peticiones un poco menos utilizadas son:



16.4 RESPUESTAS HTTP

Una página web para ver el tipo de respuestas que podemos obtener de las estandarizadas en el protocolo http es: http.cat

El más típico es el 200 cuando está todo ok. Los códigos que empiezan por 200 son positivos, los 300 es que falta algo, los 400 son negativos y los 500 son errores internos del servidor.

18. API Y RESTFUL

18.1. API Y RESTFUL

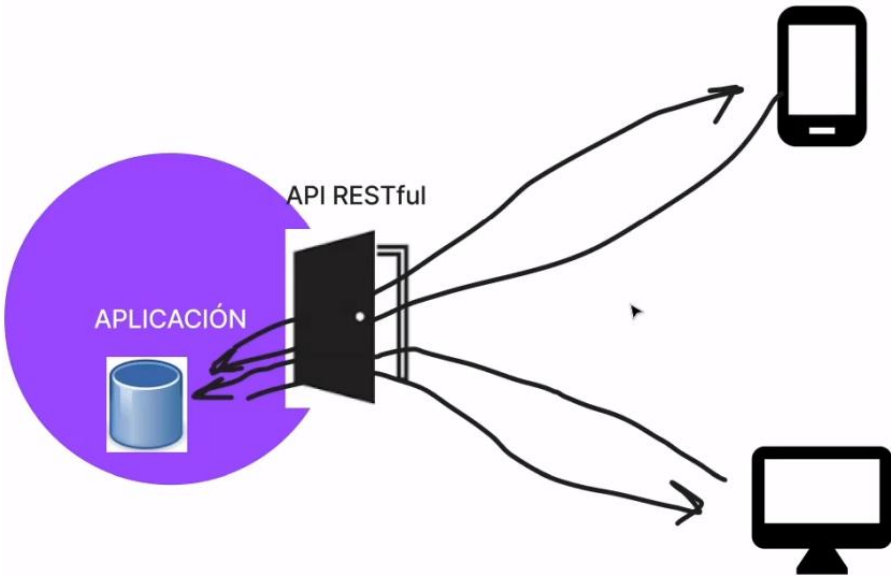
API son las siglas que hacen referencia a Application, Programming an Interface. Imaginémonos que tenemos creada una aplicación que tiene acceso a una base de datos. Entonces, la aplicación tiene una conexión directa con la base de datos. La cosa es que nosotros accedemos a la base de datos con una arquitectura interna, por eso, el control total del acceso CRUD (Create Read Update Delete) la tiene nuestra aplicación. Esto lo hacemos a través de funciones como addUser(), getUsers()… que tenemos que crear.

Entonces, lo que se conoce como API son este conjunto de funciones o de métodos que nos ayudan a realizar operaciones dentro de una base de datos sin necesidad de tocar ninguna lógica por dentro. Estas funciones son públicas y podemos acceder desde fuera.

REST, es una arquitectura más que un protocolo. Es de comunicación que significa REpresentational, State and Transfer. Ahora, cuando tenemos una API que trabaja bajo un protocolo REST, decimos que tenemos una RESTful API que es lo que verdaderamente se utiliza en el mundo real.

Si una aplicación decide que quiere compartir una parte o sus datos, entonces puede crear una RESTful Api, que es como si fuese una puerta al exterior, a través de ls cuales pueden entrar solicitudes del exterior para acceder a información de la base de datos. También puede ser que usuarios externos tengan acceso a escritura a través de una serie de permisos y a través de una serie de funciones específicas. Entonces, la aplicación deja una pequeña puerta al exterior (RESTful API) para que desde fuera, cualquier cliente pueda acceder a ellos.

Incluso si la API lo permite, podríamos realizar cambios en la base de datos.



18.2. MENSAJE Y EJEMPLO CON POSTMAN

Vamos a analizar ahora la estructura de un mensaje. Realmente un mensaje REST no es más que un mensaje HTTP. En la petición tenemos tres partes:

EJEMPLO CON POSTMAN

Aplicación: Postman.

Mirar vídeo, pero hemos utilizado la API pública de pokemon para hacer el ejemplo. Hacemos un get a https://pokeapi.co/api/v2/pokemon/charmander y nos devuelve toda la información del pokemon en formato json.

18. SOAP

Son las siglas de Simple Object Access Protocol. Se trata de un protocolo similar a REST pero con ciertas diferencias y que también se usa en el mercado. Las principales diferencias con REST son:

Que REST es una arquitectura más que un protocolo, mientras que SOAP si que lo es y además está mantenido por la Worl WIDE Web. Además, SOAP por defecto es statelees pero lo podemos configurar para que sea stateful. En SOAP solo podemos utilizar archivos XML, mientras que en REST podemos utilizar más tipos de datos. SOAP, al ser un protocolo más definido, hace que sea menos flexible y además, para poder, tanto enviar como recibir, necesitamos más recursos. Por último, SOAP, además de trabajar con HTTP también puede trabajar con SMTP o UDP y algún otro mientras que REST solo puede trabajar con HTTP.

SOAP también es una forma de acceso a las aplicaciones y a las API.

19. GRPC

19.1. INTRODUCCIÓN A GRPC

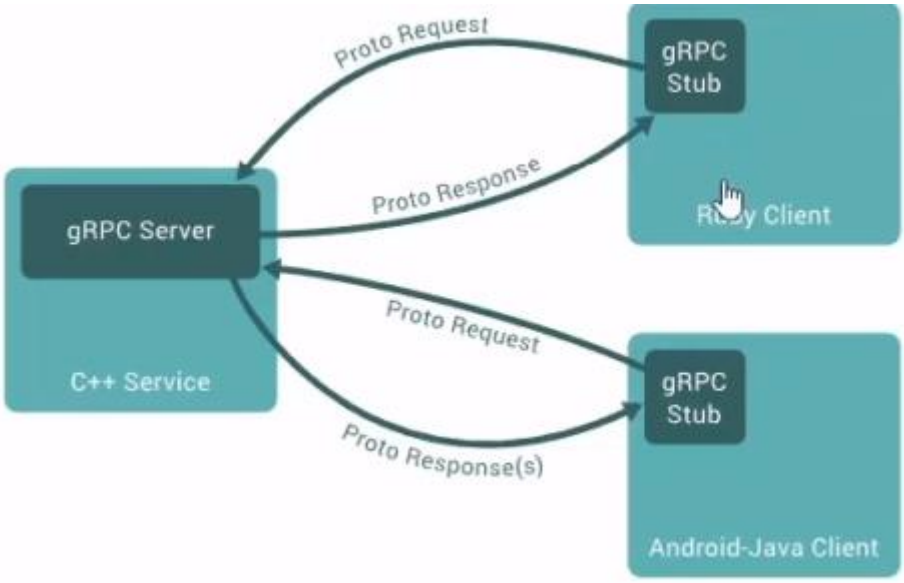
gRPC es una arquitectura o framework que se utiliza entre distintos servicios. Se trata de una arquitectura más moderna que HTTP y está hecha por Google.

Significa Remote Procedure Call:

- Se trata de una tecnología de transmisión de datos serializada.
- Creado por Google.
- Bi-directional streaming.
- Multilenguaje y multiplataforma.

La definición de los servicios es muy sencilla y se hace a través de los Protocol Buffers. Nos permite utilizar un montón de lenguajes de programación ya que existe soporte para la mayoría de ellos. Por debajo utiliza una tecnología HTTP 2.0 por lo que tenemos una mayor velocidad y enviar y recibir datos de forma bidireccional. Sería una alternativa a los RESTApis que hemos visto ya que sería mucho más rápido. Esta tecnología se está empezando a popularizar aunque no está tan extendida como las APIRest o las APISoap.

19.2. GRPC STUB Y PROTOBUFFER

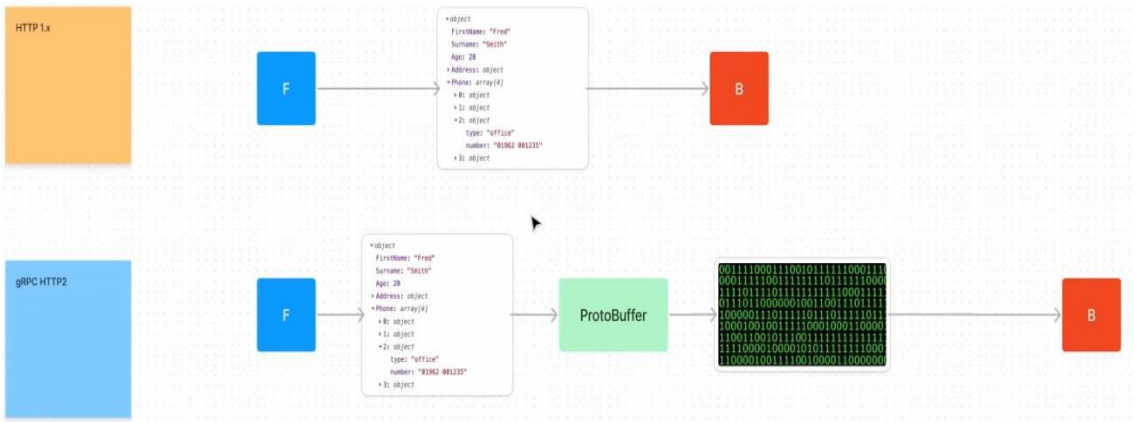


Estos clientes para poder conectarse con un servidor que utilice esta tecnología gRPC necesitan tener un Stub, que es una parte de nuestra aplicación que nos proporciona los mismos métodos que nos proporciona nuestro servidor gRPC. Es similar a lo que hemos visto pero se llaman proto por la tecnología utilizada en los intercambios en estos servidores.

PROTOCOL BUFFER

Es un mecanismo de serialización de datos. Lo que nos permite es que el peso de la comunicación sea mucho menor, y por tanto, mucho más rápida. Para más información https://developers.google.com/protocol-buffers

A través del protocolo gRPC lo que hacemos es usar el protoBuffer para serializar el mensaje y convertirlo en ceros y unos para que así pese menos. El servidor backend al que llega vuelve a transformar el mensaje o mejor dicho descodificarlo.



COMO SE DEFINE UN SERVICIO GRPC

Para definir un servicio gRPC lo primero que hacemos es definir un archivo de tipo .proto con el mensaje que vamos a transmitir. Este .proto tiene que estar tanto en el cliente como en el servidor porque sino no se van a poder comunicar. Por último, para definir un servicio:

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

20. LICENCIAS DE SOFTWARE

Estas licencias son términos legales que afectan a la utilización y la distribución de cierto software. Una licencia es:

Contrato a través del cual una persona que es el titular de los derechos de explotación de un determinado software establece las condiciones a través de los cuales otra u otras personas pueden acceder a usar ese software de acuerdo con las condiciones que él mismo estableció.

Licenciante es el desarrollador del software y Licenciatario es el que utiliza este software.

Existen diferentes licencias:

- Software Libre: Es totalmente permisiva. Podemos:
  - Ver el código fuente.
  - Estudiar el código fuente.
  - Modificar el código fuente.
  - Compartir el código fuente.
  - Compartir las modificaciones.
  - Uso comercial.
  - Utilizarlo con cualquier fin comercial y propósito.
- Open Source: Es similar pero con algunas diferencias. Podemos:
  - Ver el código fuente.
  - Estudiar el código fuente.
  - No siempre podemos modificar el código fuente.
  - No siempre podemos compartir el código fuente.
  - No siempre podemos compartir las modificaciones.
  - No podemos usarlo con fin comercial.
  - Y está limitado el uso con cualquier fin comercial o propósito.

- Propietario: No podemos hacer ninguna de las anteriores a excepción del uso con cualquier fin comercial o propósito que lo podremos hacer solo con permiso del fabricante. Existen dos tipos dentro de estas licencias:
    - Freeware: Programas gratuitos pero no podemos acceder al código.
    - Shareware: Distribuido gratuitamente solamente para su prueba, estrategia de marketing muy difundida en los últimos tiempos.

TIPOS DE LICENCIAS DE CÓDIGO ABIERTO QUE EXISTEN EN EL MERCADO

Licencias de código abierto Permisivas: Se puede crear una obra derivada sin que esta tenga obligación de protección alguna.

- Licencia Apache
- Licencia PHP
- BSD
- PSFL (Python Software Foundation License)
- MIT (Massachusetts Institute of Technology)

Licencias de Código abierto robustas fuertes: Contienen una cláusula que obliga a que las obras derivadas o modificaciones que se realicen al software original se deban licenciar bajo los mismos términos y condiciones de la licencia original:

- EPL (Eclipse Public License).
- GPL (GNU Pulic License).
- LGPL (GNU Lesser General Public License).
- AGPL (GNU Affero General Public License).

Licencias de Código abierto robustas débiles: Contienen una cláusula que obliga a que las modificaciones que se ralicen al software original se deban licenciar bajo los mismos términos y condiciones de la licencia original. Pero NO OBRAS DERIVADAS:

- MPL (Mozilla Public License).
- CDDL (Common Development and Distribution License).
- APSL (Apple Source License).

21. PROTECCIÓN DE DATOS

21.1. MD5 Y SHA

Son las dos metodologías más utilizadas a día de hoy para codificar los mensajes, pero no para descodificar.

MD5

Se creó en 1991 para reemplazar MD4 que ya fue vulnerada. Hoy en día es inseguro. Sólo se utiliza para asegurar la integridad, por ejemplo, al descargar algo comparamos los MD5 del fabricante con el nuestro. De esta manera, nos estamos asegurando de que el software es el correcto y que nadie entremedias lo ha manipulado. Mirar md5.cz

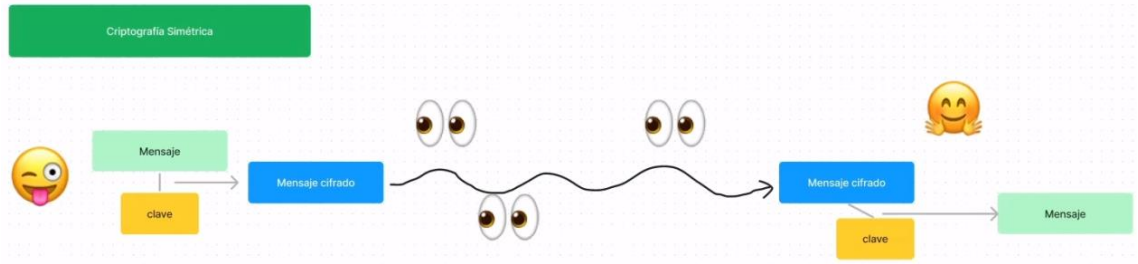
SHA

Significa Secure Hash Algotithm. Hoy en día el que se utiliza es el SHA-256. Mirar emn178.github.io.

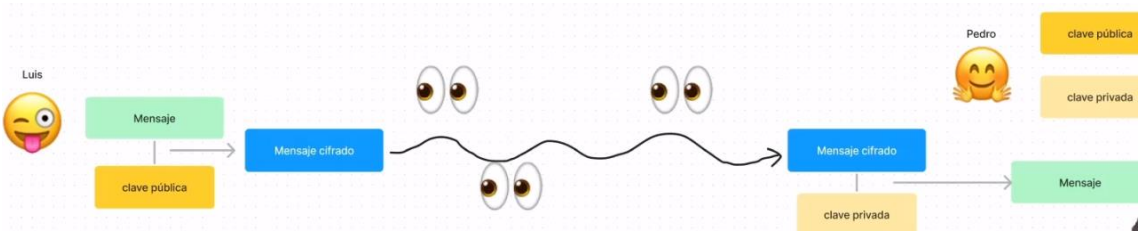
No se pueden descifrar. Esto nos lleva al funcionamiento de las contraseñas dentro de las aplicaciones. Si alguien consigue vulnerar mi base de datos podría descubrir la contraseña de todos los usuarios. Por eso, lo que se hace hoy en día es almacenar el usuario como tal y la contraseña, para almacenarse pasa por un mecanismo de HASH. Por ejemplo el SHA-256. De esta manera se guarda la contraseña cifrada. De esta manera, la próxima que el usuario quiera acceder, es introducir el usuario y la contraseña y antes de enviarlo lo que hacemos es comparar el cifrado de la contraseña pasada y si son iguales le dejamos entrar, pero si no coinciden las contraseñas cifradas, asumimos que son diferentes.

21.2. CRIPTOGRAFÍA SIMÉTRICA Y ASIMÉTRICA

Criptografía simétrica: Definimos una clave de encriptación tanto el que envía como el que recibe. Con el mensaje y la clave yo soy capaz de obtener un mensaje cifrado. Ya estoy cómodo para que este mensaje viaje a través de internet hasta el destinatario. Como el destinatario ya sabe la clave, va a ser capaz de descifrar el mensaje y obtener el mensaje original.



Criptografía asimétrica: Entran en juego dos claves. Clave pública y clave privada. El que envía el mensaje lo cifra con la clave pública y solo puede ser descifrado por la clave privada del destinatario. (Es lo que hicimos en una de las sesiones de laboratorio).



21.3. PROTOCOLOS DE SEGURIDAD EN INTERNET

Los protocolos de seguridad que existen a día de hoy en internet son: