

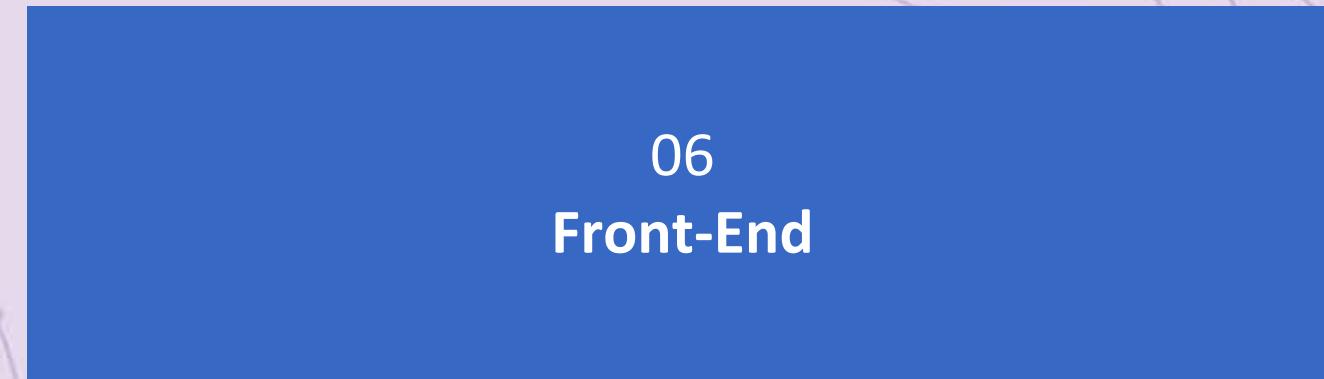


El futuro digital
es de todos

MinTIC



Ciclo 3: Desarrollo de Software



Hechos
QUE CONECTAN ✓



UNIVERSIDAD
NACIONAL
DE COLOMBIA

Misión
TIC 2022



El futuro digital
es de todos

MinTIC

Objetivo de Aprendizaje

Identificar los principales elementos asociados a la construcción de un componente de presentación (front-end), usando los lenguajes JavaScript, HTML y CSS, y el framework Vue.js.



El futuro digital
es de todos

MinTIC

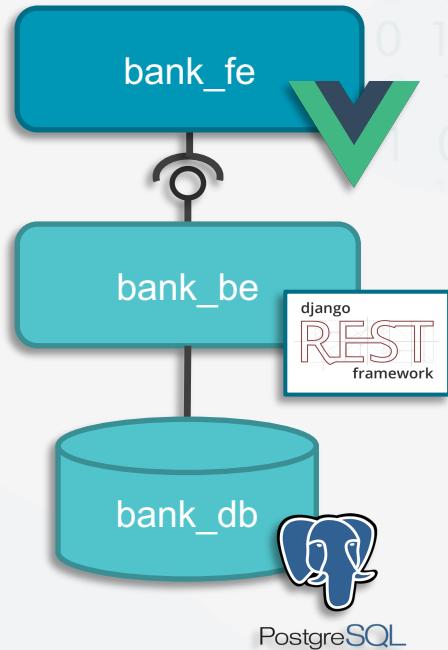
Parte 1



Capa de Presentación

En la **capa de presentación** se construirá un **componente** (*bank_fe*) cuya función será la de permitir la **interacción** con el **usuario final**, mediante una **interfaz gráfica** de tipo **web**.

En este caso, se hará uso de los lenguajes **HTML**, **CSS** y **JavaScript**, y del framework **Vue.js**.





HTML: Maquetación Web

HTML



HyperText Markup Language o HTML, es un **lenguaje de marcado** que se utiliza para **definir el contenido** y la **estructura** de una **página web**. Dicha estructura se refiere al uso de títulos, párrafos, imágenes, listas y tablas.

- HTML **puede** ser **asistido** por tecnologías como CSS y JavaScript.
- Se debe tener en cuenta que un **lenguaje de marcado** es **diferente** a un **lenguaje de programación**.



El futuro digital
es de todos

MinTIC

CSS: Estilos

Cascading Style Sheets o CSS, es un **lenguaje de hojas de estilo** que se utiliza para **aplicar estilos** de manera selectiva a elementos en **documentos HTML**. CSS permite la **estilización** de páginas web, y por ende, la creación de páginas con un **diseño llamativo** y elaborado.

CSS



[Imagen] CSS3 logo. (s. f.). [PNG]. Wikimedia. https://upload.wikimedia.org/wikipedia/commons/thumb/d/d5/CSS3_logo_and_wordmark.svg/1200px-CSS3_logo_and_wordmark.svg.png



El futuro digital
es de todos

MinTIC

Parte 2



El futuro digital
es de todos

MinTIC

JavaScript

JavaScript es el lenguaje de programación más utilizado en el desarrollo de componentes front-end de tipo web.





JavaScript: Definición

JavaScript es un **lenguaje de programación de alto nivel**, y al día de hoy es el **lenguaje de programación más utilizado** en la **WEB** (97% de los sitios de la web usan JavaScript para ejecutar scripts en el lado del cliente).

Las características de **JavaScript** están dadas por una **especificación** llamada **ECMAScript**, la cual se actualiza cada año.





JavaScript: Historia



JavaScript apareció en 1995 en el navegador **Netscape**, su propósito inicial era **agregar interacción** a los **sitios web** desde el lado del **cliente** y no únicamente desde el lado del servidor.

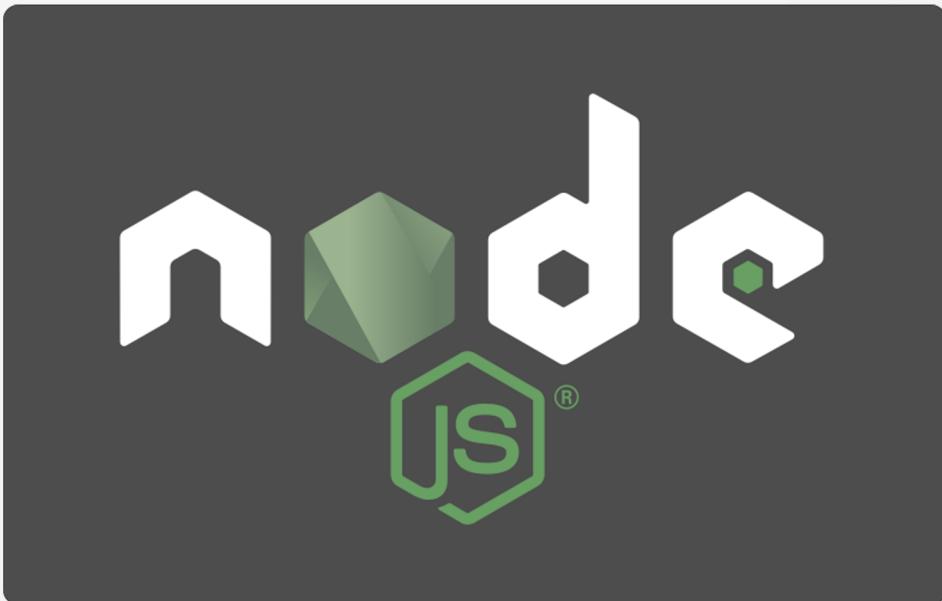
JavaScript tuvo gran éxito ya que al poder **realizar** validaciones y **cálculos** en el lado del **cliente** evitaba el **envío** de **datos** al **servidor** a través de la **red**. En los años 90 la **velocidad de internet** era muy **baja** y cualquier interacción entre el cliente y el servidor **tardaba** un **tiempo** considerable.



JavaScript: Usos Actuales

Inicialmente JavaScript fue pensado como un lenguaje de programación para ser usado en navegadores y poder ejecutar funcionalidades desde el lado del cliente, sin embargo a día de hoy, JavaScript es usado para muchos otros propósitos.

Un ejemplo de lo anterior es el uso de JavaScript como lenguaje de programación del lado de servidor, lo cual es posible gracias al entorno de ejecución NodeJS.





JavaScript: Sintaxis

```
1 <script>
2   (function () {
3     window._harvestPlatformConfig = {
4       "application": "LeCompany",
5       "permalink": "https://lecompany.com/item/%ITEM_ID%"
6     };
7     var s = document.createElement("script");
8     s.src = "https://harvestapp.com/assets/plat
9     s.async = true;
10    var ph = document.getElementsByTagName("script")[0];
11    ph.parentNode.insertBefore(s, ph);
12  })();
13 </script>
```



La **sintaxis** de **JavaScript** es muy **similar** a la sintaxis de lenguajes como **Java** o **C++**, pero **mucho** más **flexible** y **ligera**, de hecho en la mayoría de casos permite **omitar** el **punto y coma**, y a diferencia de otros lenguajes como Python, **no** existen **problemas** con la **indentación**.

A **continuación** se mostrará la **sintaxis** de algunas de las **instrucciones** más **comunes** usadas en JavaScript.



JavaScript: Variables

Actualmente **JavaScript** maneja el concepto de **variable** y **constante**, por lo cual establece la palabra reservada **LET** para definir variables y **CONST** para definir constantes.

Antiguamente JavaScript **no** tenía una **distinción** clara y hacía uso de la palabra reservada **VAR** para definir constantes y variables.

```
js index.js > ...
1 // Uso de Let y Const
2 let miVariable      = 0;
3 const miConstante  = 0;
4
5 // Uso de Var
6 var variableAntigua = 0;
7 var constanteAntigua = 0;
8
```



JavaScript: Tipos de Datos

JavaScript maneja algunos **tipos de datos** como **String**, **Number** y **Boolean**, estos no son definidos de manera explícita, ya que JavaScript es capaz de **determinar** el tipo de dato de **manera dinámica**.

Existen otros tipos de dato como **Null** que representa un valor nulo, y **Undefined** que indica que no se tiene valor alguno.

Para determinar el tipo de dato se puede hacer uso de la instrucción **typeof**.

```
js index.js > ...
1 // Number
2 let numero = 10;
3
4 //String
5 let cadena = "hola";
6
7 //Boolean
8 let booleanoTrue = true;
9 let booleanoFalse = false;
10
11 //Null
12 let nulo = null;
13
14 //Undefined
15 let sinValor;
16
```



JavaScript: Operadores

En **JavaScript** se tienen dos grupos principales de operadores, los **operadores lógicos** que incluyen la **negación**, la **conjunción** (AND) y la **disyunción** (OR), y los **operadores aritméticos** que incluyen la **suma**, la **resta**, la **multiplicación**, la **exponenciación**, la **división**, el **modulo**, el **incremento** y el **decremento**.

La mayoría de operadores son **binarios**, es decir necesitan dos valores para efectuar la operación, salvo la **negación**, el **incremento** y el **decremento** que son **unitarios**.

```
JS index.js > ...
1 //OPERADORES LOGICOS
2 let negacion      = !true;
3 let conjucion    = true && false;
4 let disyuncion   = true || false;
5
6 //OPERADORES ARIMETICOS
7 let suma          = 12 + 24;
8 let resta         = 12 - 24;
9 let multiplicacion = 12 * 24;
10 let exponenciacion = 12 ** 24;
11 let division     = 12 / 24;
12 let numero       = 12;
13 let incremento   = numero++;
14 let decremento   = numero--;
15
```



JavaScript: Operadores

También se tiene **otro grupo de operadores**, llamados **operadores de comparación**, estos son **útiles** a la hora de trabajar **con condicionales y ciclos**.

Dentro de estos operadores se tiene el igual (`==`), el diferente (`!=`), el mayor que (`>`), el mayor o igual que (`>=`), el menor que (`<`), el menor o igual que (`<=`) y el estrictamente igual (`===`) que no solo compara el valor si no también el tipo de dato.

```
JS index.js > ...
1 //OPERADORES DE COMPARACION
2 let igual = (12 == 24);
3 let diferente = (12 != 24);
4 let mayor_que = (12 > 24);
5 let mayor_igual_que = (12 >= 24);
6 let menor_que = (12 < 24);
7 let menor_igual_que = (12 <= 24);
8 let estrictamente_igual = (12 === 24);
9
```



JavaScript: Arrays

A diferencia de otros lenguajes **JavaScript** permite **definir arrays** (o **arreglos**) con **distintos tipos de datos**. Esto se debe a que JavaScript maneja un tipado dinámico.

Otra característica interesante de los **arrays** en **JavaScript** es que **pueden** tener un **tamaño dinámico**, es decir que su tamaño puede variar durante la ejecución del programa.

```
JS index.js > ...
1 //Definiendo un ARRAY
2 let carros = ["Saab", "Volvo", "BMW"];
3
4 //Seleccionando elementos
5 let x = carros[0];      // x = "Saab"
6
7 //Modificando elementos
8 carros[0] = "Opel";
9
10 //Agregar Elemento al Final
11 carros.push("Corsa");
12
13 //Eliminar Ultimo Elemento
14 carros.pop();
15
```



JavaScript: Objetos

En **muchos lenguajes** se tiene el **concepto** de **diccionario**, el cual es una **colección** de parejas de **claves y valores**, dentro de **JavaScript** se tiene este mismo concepto pero es conocido como **Objeto**.

En **JavaScript** los **objetos** suelen **utilizarse bastante**, ya que permiten **organizar** distintos **datos** en una **estructura** clara y concisa.

```
js index.js > ...
1 //Definiendo un Objeto
2 const persona = {
3   nombre: "John",
4   apellido: "Ramirez",
5   edad: 50,
6   colorOjos: "azul"
7 };
8
9 //Accediendo a los valores #1
10 let nombre = persona.nombre;
11
12 //Accediendo a los valores #2
13 let apellido = persona["apellido"];
14
```



JavaScript: Condicionales

JavaScript maneja una estructura **clásica** de **condicionales**, la cual incluye un **IF**, un **ELSE** y un **ELSE IF**. Su **funcionamiento** se bastante **similar** a la mayoría de los **lenguajes**.

Al igual que en otros lenguajes JavaScript ofrece la estructura **SWITCH**, la cual suele ser **útil** cuando se requiere una **gran** cantidad de **comparaciones** sobre un mismo **dato**.

```
JS index.js > ...
1 // Uso de IF, ELSE IF y ELSE
2 let edad = 30;
3 let resultado;
4
5 if (edad > 60) {
6     resultado = "Anciano";
7 }
8
9 else if (edad > 18) {
10    resultado = "Adulto";
11 }
12
13 else {
14     resultado = "Niño";
15 }
16
```



JavaScript: Ciclos

JavaScript ofrece tres tipos de **ciclos**, **FOR**, **WHILE** y **DO WHILE**, estos tres **funcionan** de manera **similar** a la **mayoría** de **lenguajes**.

Pero además de los ciclos tradicionales JavaScript ofrece algunas **variantes** con las instrucciones **IN** y **OF** para los ciclos **FOR**, estas variantes **facilitan** el proceso de **recorrer** los **elementos** de un **array** o de un **objeto**.

```
js index.js > ...
1  for (let i = 0; i < 10; i++) {
2    |  // CODIGO
3  }
4
5
6  while (condition) {
7    |  // CODIGO
8  }
9
10
11 do {
12   |  // CODIGO
13 }
14 while (condition);
15
```



JavaScript: Funciones

A diferencia de otros la lenguajes, la definición de **funciones en JavaScript** es muy **flexible**:

- En primer lugar, al **no** existir de **tipado** de datos, al momento de definir los **parámetros** de la función solamente bastará con definir sus **nombres**.
- En segundo lugar, el **retorno** de **valores** por parte de una **función no es obligatorio**, cuando se deseé **retornar un valor** simplemente se **usa** la instrucción **RETURN**.

```
js index.js > ...
1 // Definicion
2 function miFuncion(a, b) {
3 | return a * b;
4 }
5
6 // Uso
7 let x = miFuncion(4, 3);
8
```



JavaScript: Funciones Flecha

Además de la manera **tradicional** para **definir funciones**, **JavaScript** ofrece **otra manera** de definirlas, las funciones definidas de esta manera son conocidas como **funciones flecha**.

Las **diferencias** entre estos dos tipos de funciones son un poco **avanzadas**, pero en la **práctica** para **casos sencillos** de uso, **no existen diferencias** notorias.

```
js index.js > ...
1 // Definicion Tradicional
2 function hello () {
3     return "Hello World!";
4 }
5
6 // Definicion Funcion Flecha
7 const hello = () => {
8     return "Hello World!";
9 }
10
```



El futuro digital
es de todos

MinTIC

JavaScript: DOM

El DOM permite que JavaScript pueda manipular el contenido de documentos HTML.

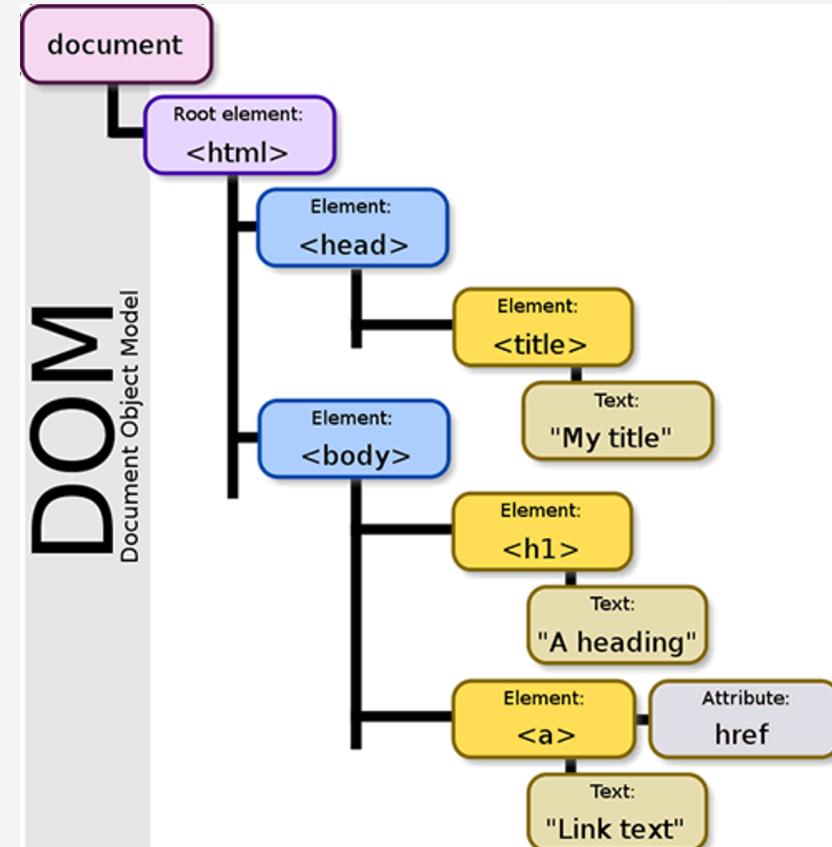




JavaScript: DOM

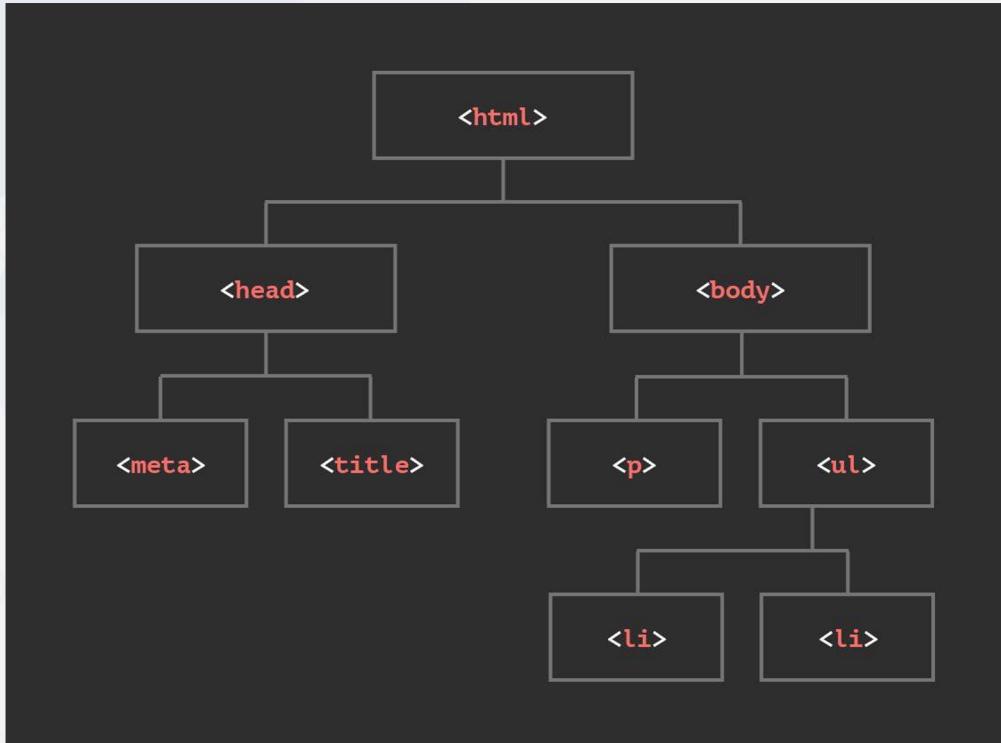
El **objetivo** de **JavaScript** en el cliente (navegador) es brindar **interactividad** a los sitios **WEB**, para ello **JavaScript** debe tener la capacidad de **manipular** y **editar** el **contenido** de los documentos **HTML** que definen la **estructura** de los sitios **WEB**.

Para llevar a cabo los **cambios** en el contenido de los sitios **WEB**, **JavaScript** hace uso del **DOM (Document Object Model)**, este **documento** define un **objeto** que **representa** la estructura del sitio **WEB**.





JavaScript: Document



Dentro de **JavaScript** se tiene de manera **predeterminada** un objeto conocido como **Document**, este objeto **recolecta** toda la información del **DOM**, por lo cual a través de este objeto se puede **acceder** a **cualquier elemento** del sitio WEB.

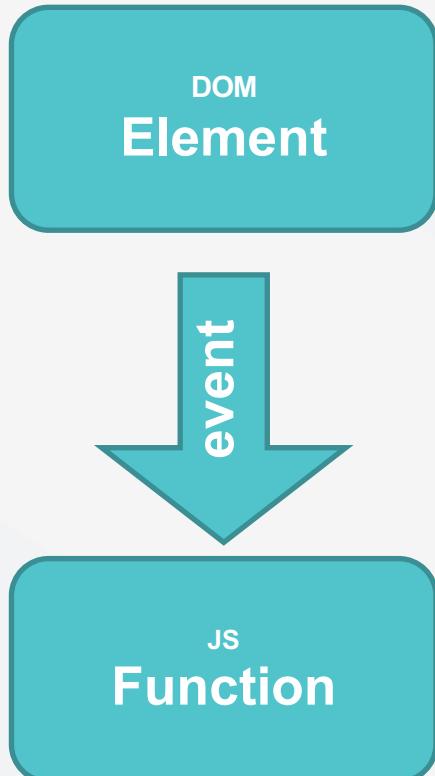
Además de **recolectar** toda la información del **DOM**, el objeto **Document** brinda una serie de **funcionalidades** que facilitan la **interacción** entre **JavaScript** y **HTML**.



JavaScript: Eventos

Una de las **interacciones** más importantes entre **JavaScript** y **HTML** es el manejo de **eventos**, un **evento** es un suceso **emitido** por **HTML** producto de la **interacción** del **usuario** con el **sitio WEB**.

Por si mismo **HTML no** es capaz de **controlar** estos **eventos**, sin embargo **JavaScript** es capaz de **capturar** dichos **eventos** y **ejecutar** ciertas **funcionalidades** cuando este suceda.





El futuro digital
es de todos

MinTIC

Parte 3



Petición HTTP: CORS

Cross-Origin Resource Sharing o **CORS** es un mecanismo que le permite a los **servidores** indicar qué **orígenes** (dominios, esquemas o puertos) tienen permitido **acceder** a sus recursos **desde un navegador web**. Por esta razón, cuando un **navegador web** va a realizar una **petición HTTP** a una API, este debe realizar antes una **petición Preflight** para **validar** si tiene **acceso** al **recurso** que va a solicitar.





Petición HTTP: Preflight

En una **petición Preflight** el navegador web envía a la API, el **origen** y el **método** de la petición HTTP que se va a realizar. Cuando el servidor de la API recibe esta petición, este responde al navegador con los **orígenes** y **métodos permitidos**. De esta forma, el navegador **únicamente** realiza la petición HTTP si en la **respuesta Preflight** se indica que su origen y método son **válidos**.

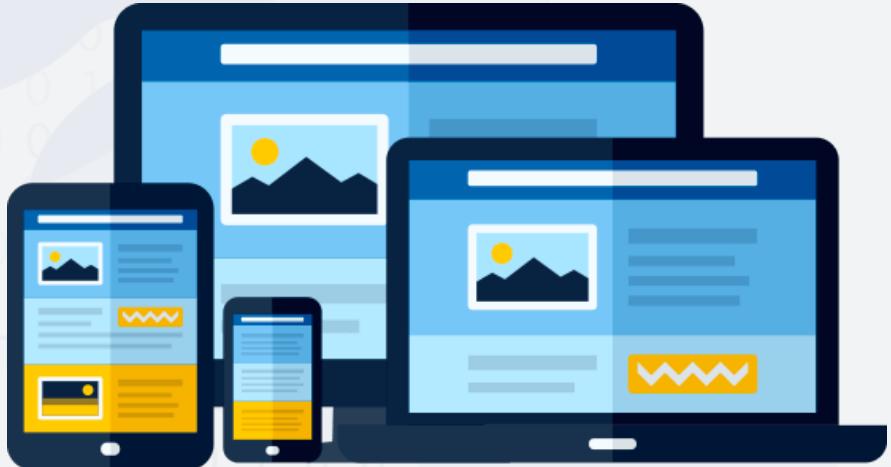




El futuro digital
es de todos

MinTIC

Django: CORS



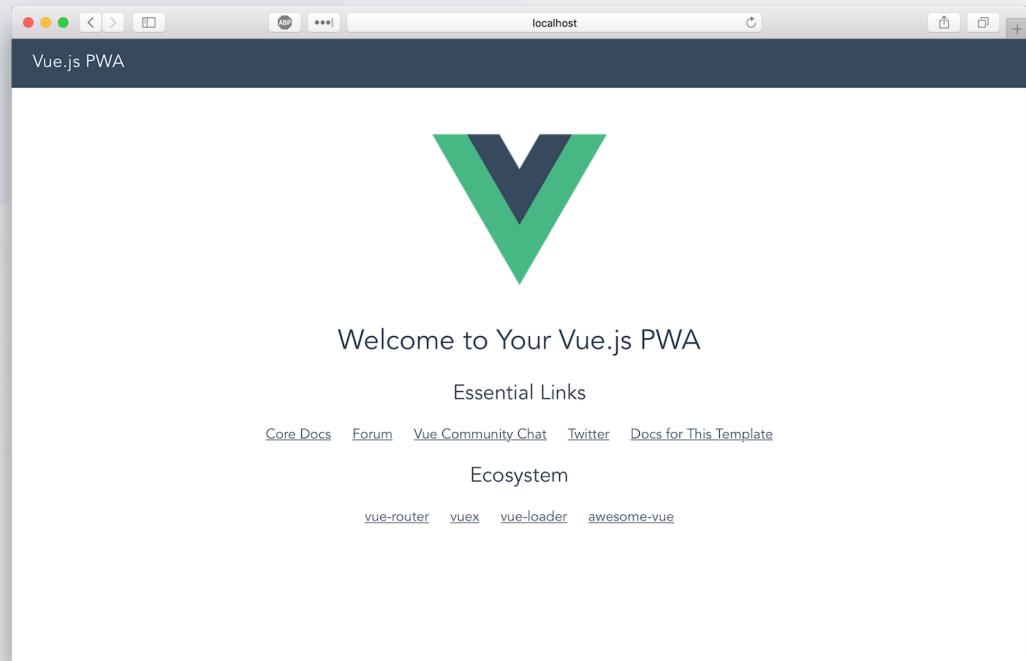
En el **componente de presentación** se desarrollará una **aplicación web** que realizará peticiones al componente lógico. Esto implica que el **navegador web** realizará **peticiones** a la API. Para evitar que el servidor rechace dichas peticiones, se **configurará** en el componente lógico la **respuesta Preflight** del mecanismo CORS. Para esto se utilizará el paquete **django-cors-headers**.



El futuro digital
es de todos

MinTIC

Vue.js: Definición



Vue.js es un **framework** que permite la **construcción** de **interfaces** de usuario, utilizando un **modelo** incremental basado en **componentes**.

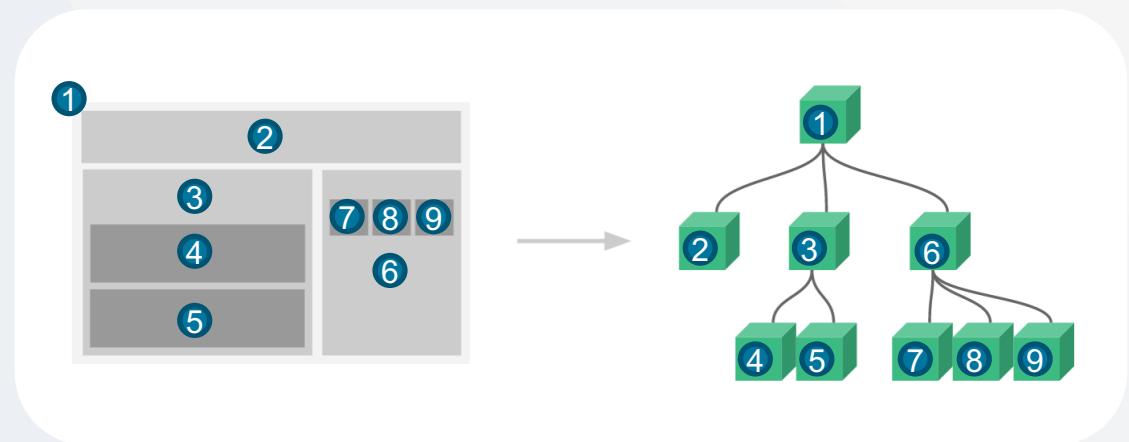
En su mayoría, Vue.js se utiliza para crear **aplicaciones web** que utilizan dicho modelo basado en componentes, pues este ofrece algunas **ventajas** para la **reutilización** de código, la **organización** del proyecto, y la **carga rápida** de la página web.

[Imagen] Welcome to Vue. (s. f.). [PNG]. Wubrfki. <https://d33wubrfki0l68.cloudfront.net/abfa76feabe7e5aa22999ba7e6088540d94ffb50/87f7b/assets-jeckvill/blog/vue-crud-node/welcome-to-vue-pwa-eea589caa41609323c8793291ebd75f19e9e9139b9165a077005118577def59e.png>



Vue.js: Componentes

En **Vue.js** una aplicación web se divide en **componentes**, es decir, en **fragmentos de código** que contienen el **código** HTML, CSS y JavaScript de una **sección** específica de la **página**. Estos componentes tienen una **jerarquía** similar a la que tiene un **árbol anidado**, pues **dentro** de cada componente se pueden **declarar** otros componentes, con los cuales se forma una **relación padre/hijos**.



[Imagen] Word Image. (s. f.). [PNG]. Linux Hint. <https://linuxhint.com/wp-content/uploads/2020/11/word-image-116.png>



Vue.js: Componentes

En Vue.js existen múltiples formas de **declarar** un componente; sin embargo, la forma más utilizada es por medio de **archivos .vue**, que integran el código **HTML, JS, y CSS** de cada componente.

```
<template lang="jade">
  div
    p {{ greeting }} World!
    other-component
  </template>

<script>
  import OtherComponent from './OtherComponent.vue'

  export default {
    data () {
      return {
        greeting: 'Hello'
      }
    },
    components: {
      OtherComponent
    }
  }
</script>

<style lang="stylus" scoped>
  p
    font-size 2em
    text-align center
</style>
```

Line 27, Column 1

Spaces: 2 Vue Component



Vue.js: HTML

En el contenido de la etiqueta **template** se escribe el **código HTML** del componente. Su sintaxis es la misma que se ha visto en sesiones anteriores, sin embargo, Vue.js **añade** algunas **funcionalidades** extras, cuya sintaxis se explicará en el desarrollo del componente de presentación.

HTML



```
<template lang="jade">
  div
    p {{ greeting }} World!
    other-component
</template>

<script>
import OtherComponent from './OtherComponent.vue'

export default {
  data () {
    return {
      greeting: 'Hello'
    }
  },
  components: {
    OtherComponent
  }
}
</script>

<style lang="stylus" scoped>
p
  font-size 2em
  text-align center
</style>
```

Line 27, Column 1 Spaces: 2 Vue Component



Vue.js: JavaScript

En el contenido de la etiqueta **script** se escribe el **código JavaScript** del componente. Allí, para que Vue.js **reconozca** el componente adecuadamente se debe **exportar un objeto** que contiene las **propiedades** y **funciones** del componente. La sintaxis definida por Vue.js para dicho objeto se explicará en el desarrollo del componente de presentación.

JavaScript ←

```
<template lang="jade">
  div
    p {{ greeting }} World!
    other-component
  </template>

<script>
  import OtherComponent from './OtherComponent.vue'

  export default {
    data () {
      return {
        greeting: 'Hello'
      }
    },
    components: {
      OtherComponent
    }
  }
</script>

<style lang="stylus" scoped>
  p
    font-size 2em
    text-align center
</style>
```

Line 27, Column 1 Spaces: 2 Vue Component



Vue.js: CSS

En el contenido de la etiqueta **style** se escribe el **código CSS** del componente. Su sintaxis es la misma que se ha visto en sesiones anteriores, sin modificación alguna.

CSS

```
<template lang="jade">
  div
    p {{ greeting }} World!
    other-component
  </template>

<script>
import OtherComponent from './OtherComponent.vue'

export default {
  data () {
    return {
      greeting: 'Hello'
    }
  },
  components: {
    OtherComponent
  }
}
</script>

<style lang="stylus" scoped>
  p
    font-size 2em
    text-align center
</style>
```

Line 27, Column 1

Spaces: 2 Vue Component



El futuro digital
es de todos

MinTIC

Vue.js: Servidor



Vue.js se encarga únicamente del **renderizado** de la aplicación web. Por esta razón, si se necesita **crear un servidor** que exponga en internet la página web creada con Vue.js, se debe utilizar un **entorno de ejecución** que lo permita. Uno de los más utilizados para ello es **Node.js**. El cual se encarga, entre otras cosas, de la **ejecución del servidor**, y de la **administración de dependencias** del proyecto.





El futuro digital
es de todos

MinTIC

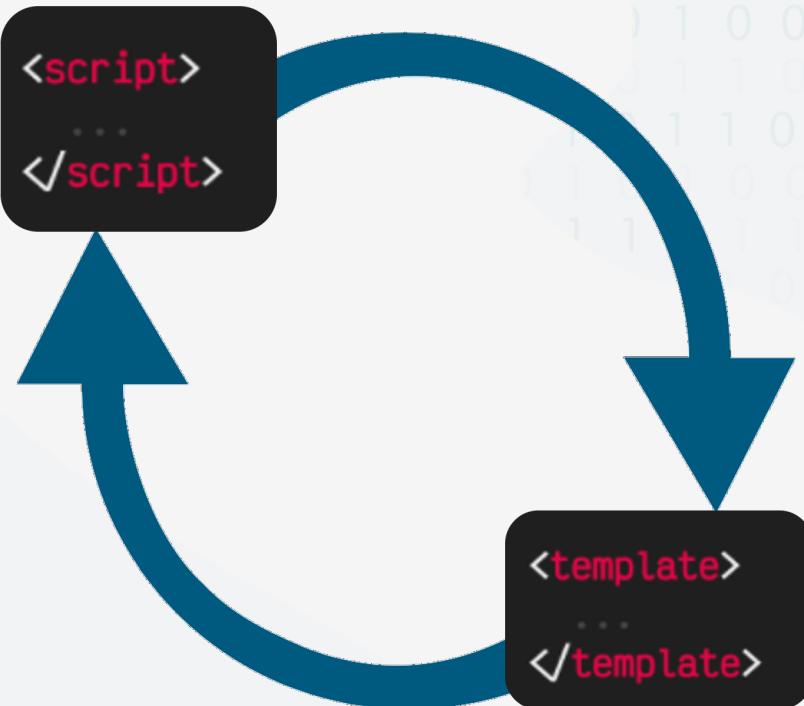
Parte 4



Componentes: Script y Template

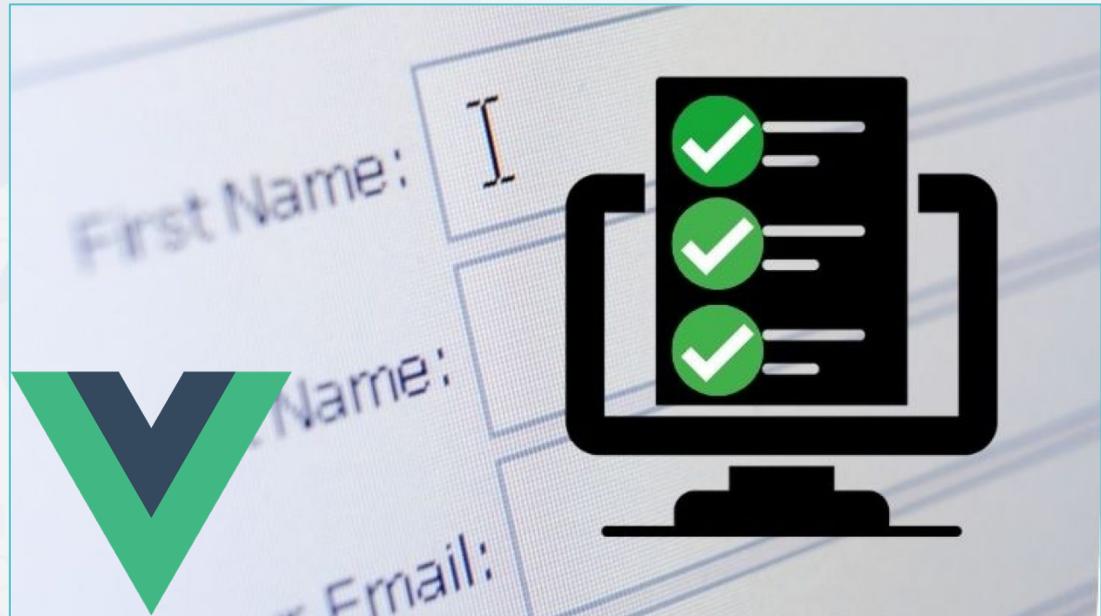
La **interactividad** de las aplicaciones desarrolladas en **Vue** se debe a la **comunicación** directa entre el **Script** y el **Template**. Vue ofrece herramientas que permiten que el **template** se cree y modifique dinámicamente a partir de valores e interacciones generadas en el **script**.

Un ejemplo es **v-if**, este es un atributo que se puede agregar a las **etiquetas del template**, y permite que estas se **muestren o no**, dependiendo del **valor** que reciban.





Componentes: Formularios

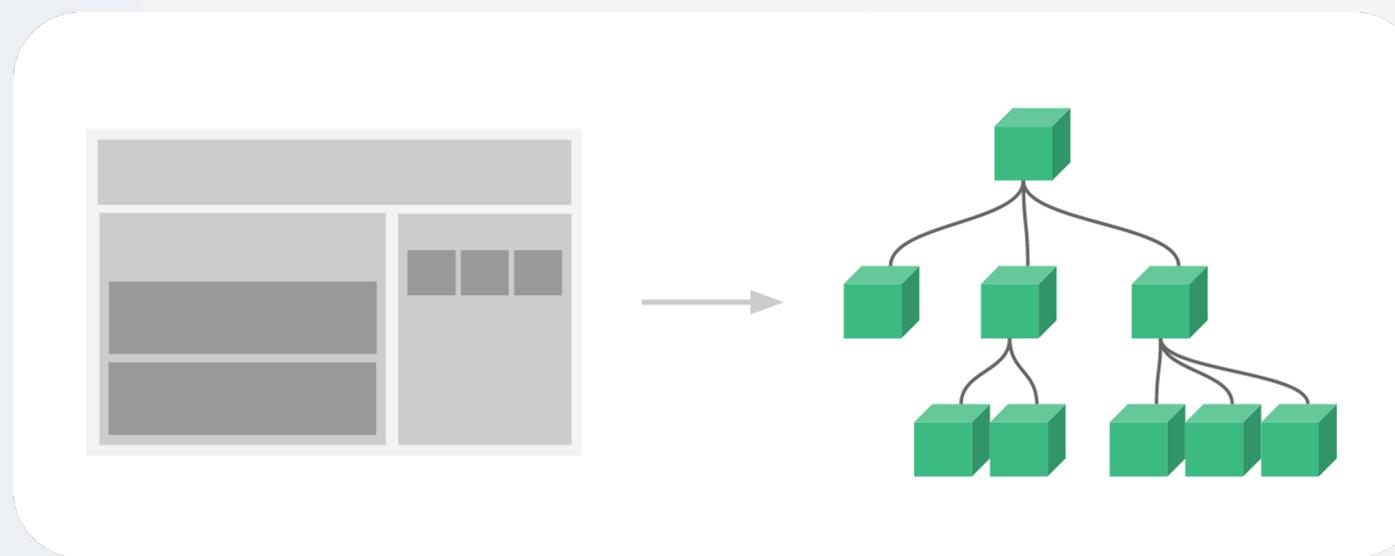


Uno de los escenarios donde más se suele aprovechar la **comunicación** directa entre el **Script** y el **Template** es en los **formularios**. Vue permite asociar de manera fácil y rápida los **datos capturados** en un **formulario** del template con un **objeto** del script. Usando solamente **JavaScript**, realizar tareas como estas sería un trabajo **muy tedioso**.



Componentes: Jerarquía

Vue estructura los **componentes** de sus **aplicaciones** en forma de un **árbol jerárquico**, se parte del componente **principal** llamado **APP**, que hace las veces **de** raíz y a partir de este se agregan los demás **componentes**, formando un **árbol** con relaciones de **padre e hijos**.



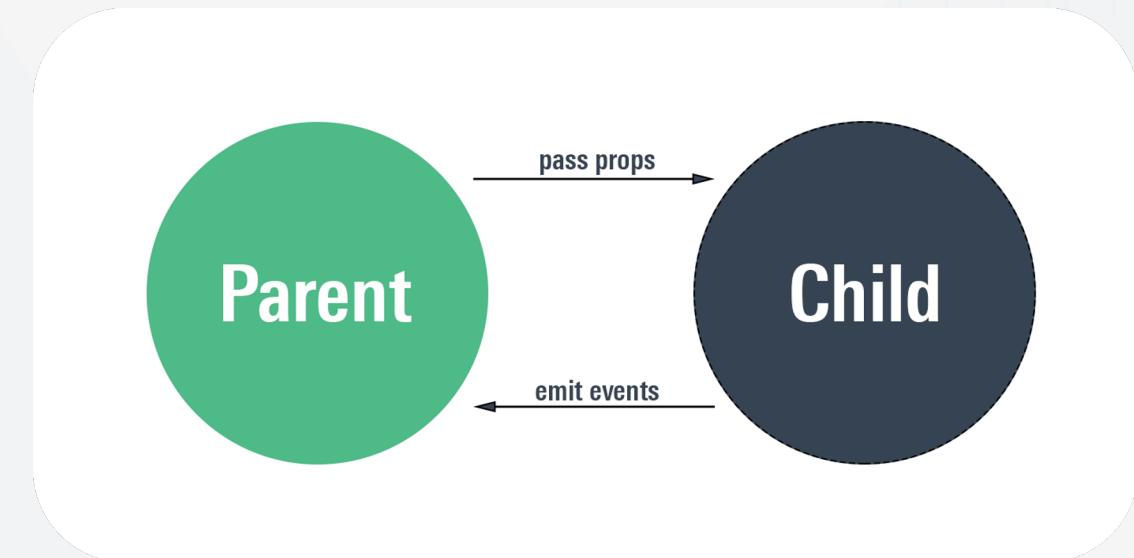
Vue Árbol de Componentes. (s. f.). [Gráfico]. <https://es.vuejs.org/images/components.png>



Componentes: Props y Emit

Debido a la **estructura jerárquica** de los componentes todas las relaciones serán del tipo **Padre-Hijo**, la **comunicación** entre estos se dará a través de **props** y **events**.

- Los **props**, son **parámetros** que el componente **padre** le debe **pasar** al componente **hijo** al momento de definirlo.
- Un componente **hijo** se **comunica** con su componente **padre** emitiendo o **ejecutando** **eventos** o **funciones** que este le provee.





El futuro digital
es de todos

MinTIC

Router: Navegación URL



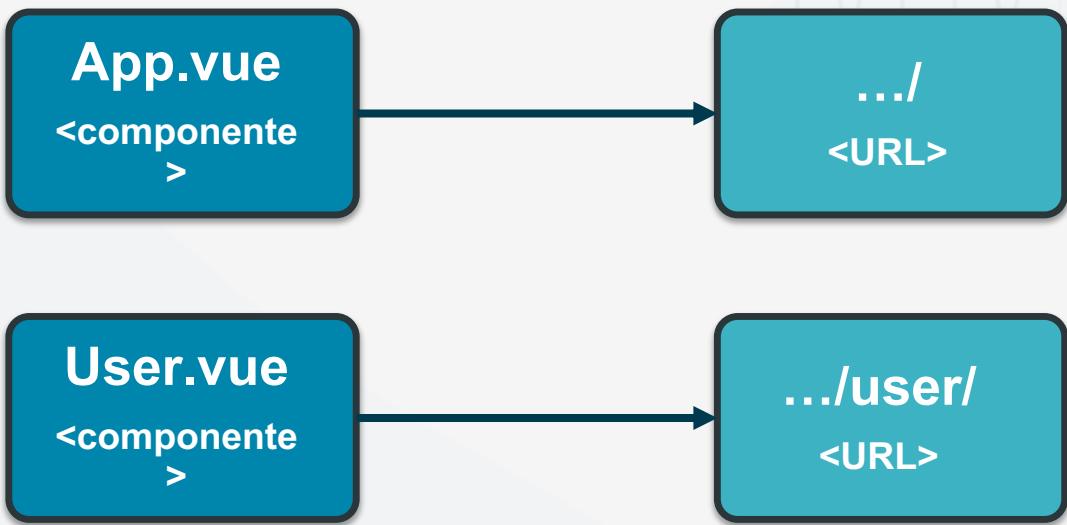
La **capa de presentación** genera un **componente WEB** que se **accede** a través de un **navegador** el cual ofrece el concepto de **navegación por URL**, es decir, que a través de **pequeñas** modificaciones de la **URL** del **sitio web** se puede acceder a **recursos o cambiar el estado de la aplicación**.



Router: Componentes

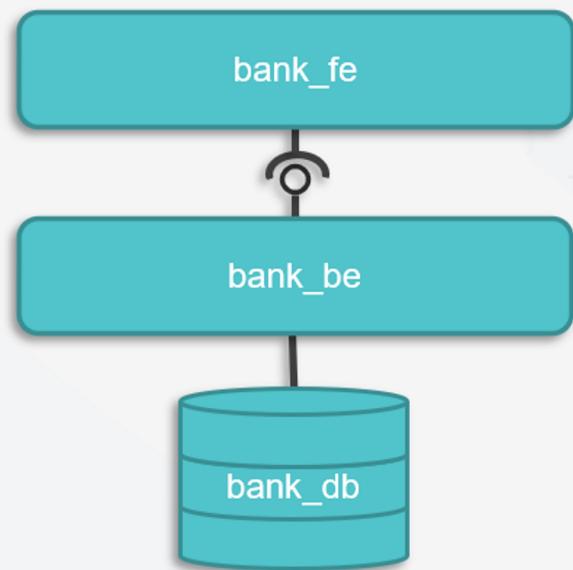
Con la idea de **controlar** la **navegación** a través de la **URL**, Vue ofrece el concepto de **Router**, el cual tiene como objetivo **manejar** las **URL** dentro la **aplicación**, para esto parte de la idea básica de asociar una **URL** a un **Componente** de **Vue**.

Lo anterior permite crear una serie de **URLs** con las cuales el usuario puede **acceder** de manera directa para obtener distintos **recursos** de la **aplicación**.





Capa Lógica: Servicios

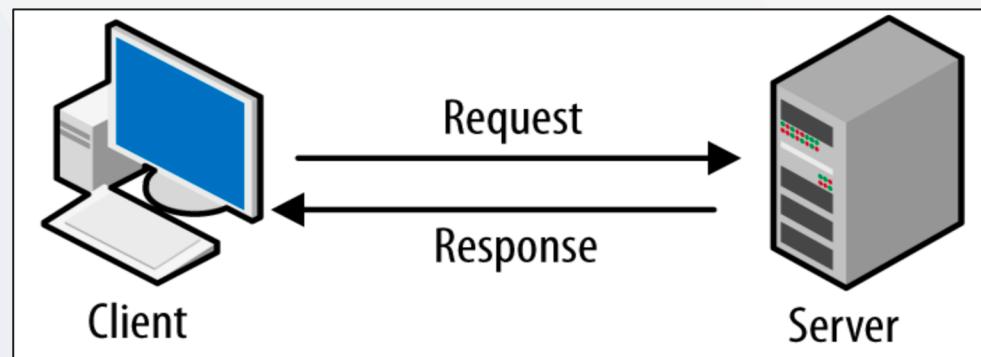


Teniendo en cuenta el **diseño** de **tres capas** del sistema que se está desarrollando, la **capa de presentación** representada a través del componente de **bank_fe** es la encargada de interactuar **directamente** con el usuario **final**, para hacer esto **posible** debe **comunicarse** con la capa de **lógica** representada a través del componente **bank_be** y **consumir** los **servicios** que esta ofrece.



Capa Lógica: Peticiones HTTP

La **comunicación** entre los **componentes bank_fe** y el **bank_be** con el fin de **consumir** los **servicios**, se realiza a través peticiones **HTTP**, donde el **front-end** asume el **rol** de **cliente** y envía una **petición** al **back-end**, quien asume el **rol** de **servidor**.





El futuro digital
es de todos

MinTIC

Parte 5



JavaScript: LocalStorage

```
localStorage.setItem("str", "XYZ");

const getStr = localStorage.getItem("str");

console.log("localStorage.getItem(): ", getStr);
```

El **localStorage** es un método para el **almacenamiento** de datos **locales** en el **navegador web**, sin necesidad de acceder a una base de datos remota.

Este almacena datos por medio de **ítems**, conformados por parejas de **llave-valor**. Estos ítems se asocian con una aplicación web específica y **no se eliminan** aunque se cierre el navegador web.



JavaScript: Async/Await

En **JavaScript** se pueden declarar **funciones async**. Sin entrar en detalles, estas funciones permiten utilizar la palabra reservada **await** justo antes del llamado de otra función, para **esperar** a que esta última termine su ejecución, antes de ejecutar el código que se encuentra debajo. Sin embargo, para ello, es necesario que la **función** llamada con **await**, **retorne una promesa**.

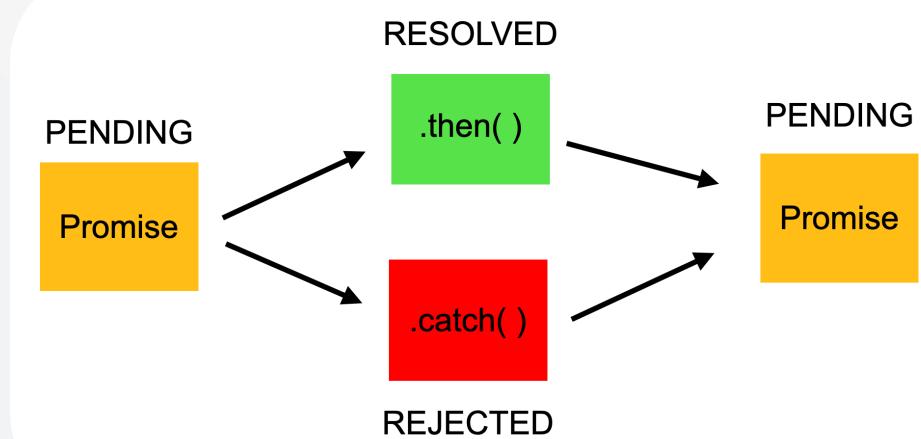
```
const fetchUser = async userId => {
  try {
    const user = await fetchData(userId);
    console.log("Here's your data:", user);
  } catch (err) {
    console.error(err);
  }
};

fetchUser(5);
```



JavaScript: Promesas

Una **promesa** en JavaScript es un **objeto** que **representa la finalización o el fracaso de una operación**. Su creación y usos fuera de las funciones `async/await` son un poco complejos. Por ello, solo es necesario saber que cuando se realiza una **petición HTTP** con cualquier cliente HTTP como `axios`, este **retorna una promesa**, la cual será utilizada en conjunto con una **función `async/await`**.





El futuro digital
es de todos

MinTIC

Parte 6



El futuro digital
es de todos

MinTIC

Despliegue: HTTPS



HTTPS (por sus siglas en inglés, **Hypertext Transfer Protocol Secure**) es un protocolo que permite establecer una **conexión segura** entre el **servidor** y el **cliente**, dicha conexión que no puede ser interceptada por personas no autorizadas.

Este protocolo se puede ver como la **versión segura** del protocolo **HTTP**.



Despliegue: Certificados SSL

Un **certificado SSL** es un **documento digital** que **autentica la identidad** de un sitio web y permite crear una **conexión segura** entre el **servidor** y sus **clientes** (conexión HTTPS).

Las empresas y las organizaciones deben **agregar** certificados **SSL** a sus aplicaciones web para **proteger** las **transacciones en línea** y la **información sensible** de sus usuarios.

Los **navegadores** suelen ser **rigurosos** con el manejo de **certificados**, ya que reducen el **acceso a lo sitios** que no los **poseen**.

