1. Fundamentos de Aplicaciones Web

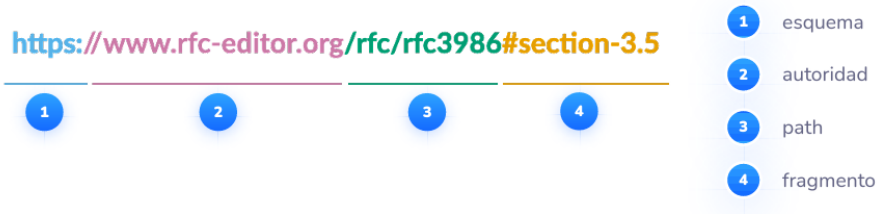
1. World Wide Web (WWW)

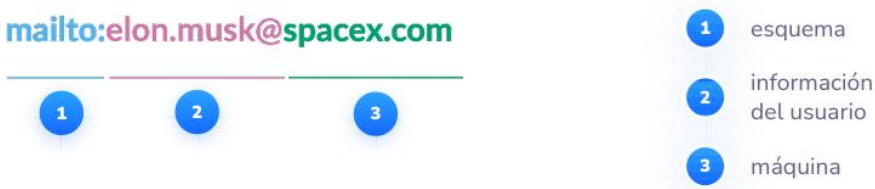
- **Sistema de información interconectada:** Accesible a través de Internet, basado en hipervínculos (elemento capaz de enlazar un recurso electrónico con partes de sí mismo o con otros recursos diferentes). Web = colección de recursos conectados.
- **Componentes esenciales:**
  - **HTTP:** Protocolo para transferir información.
  - **URI/URL:** Identificadores de recursos web específico.
  - **HTML:** Formato más utilizado para documentos web.

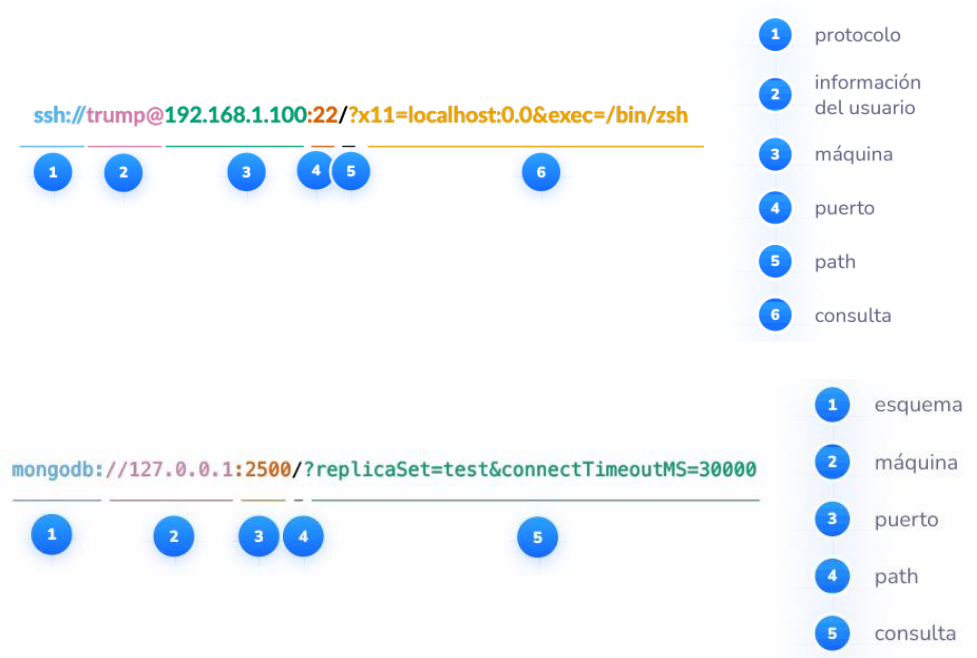
2. Recursos Web

- **Definición:** Cualquier contenido identificable en la Web, etiquetado con MIME (tipo/subtipo) y accesible mediante una URI.
- **Estructura de una URL:** Incluye el esquema, autoridad, y path (ruta)

1. El esquema identifica el protocolo para acceder al recurso: `esquema:`
2. La autoridad identifica el sistema donde está el recurso: `//[usuario:contraseña@]host[:puerto]`
3. El path identifica la referencia al recurso: `/ruta/a/recurso[?p1=val1&p2=val2][#fragmento]`







3. Funcionamiento de la Web

- **Modelo Cliente-Servidor:** Tipodesistema distribuido que involucra la comunicación de uno o más clientes que solicitan servicios o recursos a uno o más servidores a través de la red.
    - **Cliente:** Actúa en representación del usuario y origina el tráfico web
    - **Servidor**: Almacenan los recursos solicitados por un cliente
    - **Intermediarios:** Proxy: agente de redirección; Gateway: agente receptor; Túnel: agente retransmisor
- **Proceso de solicitud y respuesta:**
    - Cliente realiza una petición HTTP.
    - Servidor procesa la petición y envía una respuesta.
    -

4. Aplicaciones Web

- **Tipos de aplicaciones:**
    - **Página web estática:** Contenido fijo.
    - **Página web dinámica:** Contenido generado al vuelo.
- **Arquitecturas comunes:**
    - **Dos capas:** Presentación y lógica combinadas, datos separados.
    - **Tres capas:** Presentación, lógica, y datos separados.
    - **Patrones de diseño:**
        - **Modelo Vista Controlador (MVC):** Separa funciones y responsabilidades.
        - **Single-Page Application (SPA):** Contenido accesible a través de un único documento.

2. HTML

1. Introducción a HTML

- **HyperText Markup Language (HTML):** Lenguaje de marcado utilizado para definir y estructurar contenido web.

2. Elementos HTML

- **Estructura de un elemento:**
  - **Etiqueta de apertura:** `<p>`
  - **Contenido:** "Esto es un párrafo."
  - **Etiqueta de cierre:** `</p>`

3. Atributos

- **Definición y uso:** Proporcionan información adicional sobre los elementos.
- **Ejemplos de atributos:** <p Lang=”x”> ____ <p>
  - `lang`: Define el idioma.
  - `id`: Identifica unívocamente un elemento.
  - `class`: Categoriza elementos.

4. Caracteres especiales y comentarios

- **Caracteres especiales:** Como `<`, `>`, `&`, etc.
- **Comentarios:** `<!-- comentario -->`

5. Estructura básica de un documento HTML

- **Doctype:** `<!DOCTYPE html>`
- **Elemento raíz:** `<html>`
  - **Cabecera:** `<head>`
  - **Cuerpo:** `<body>`

6. Manipulación de texto

- **Encabezados:** `<h1>` a `<h6>`
- **Párrafos:** `<p>`
- **Énfasis:** `<em>` y `<strong>`
- **Listas:**
  - No ordenadas: `<ul>` y `<li>`
  - Ordenadas: `<ol>` y `<li>`
  - Anidadas: Listas dentro de listas.

```
<ul>
  <li>Milk</li>
  <li>
    Cheese
    <ul>
      <li>Blue cheese</li>
      <li>Feta</li>
    </ul>
  </li>
</ul>
```

7. Hipervínculos

- **Elementos y atributos:**
  - `<a>`: Representa un enlace.
  - `href`: URL del recurso.
  - `target`: Define dónde se abre el enlace.

```html
<p>You can reach Michael at:</p>
<ul>
  <li>
   <a href="https://example.com">Website</a>
  </li>
  <li>
   <a href="mailto:m.bluth@example.com">Email</a>
  </li>
  <li><a href="tel:+123456789">Phone</a></li>
</ul>
```

8. Imágenes

- **Elementos y atributos:**
  - `<img>`: Representa una imagen.
  - `src`: URL de la imagen.
  - `alt`: Texto alternativo.

```html
<img
  src="/images/grapefruit-slice-332-332.jpg"
  alt="Grapefruit slice atop a pile of other
  ↪  slices">
```

9. Tablas

- **Estructura básica:**
  - `<table>`: Define una tabla.
  - `<tr>`: Define una fila.
  - `<td>`: Define una celda.
  - `<th>`: Define un encabezado de celda.
  - `<caption>`: Define un título para la tabla.

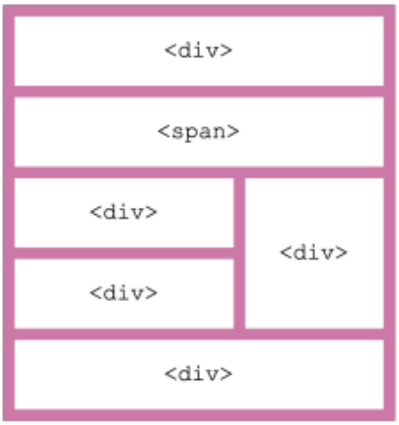
```html
<table>
  <caption>Alien football stars</caption>
  <tr>
    <th scope="col">Player</th>
    <th scope="col">Za'taak</th>
  </tr>
  <tr>
    <th scope="row">TR-7</th>
    <td>4,569</td>
  </tr>
  <tr>
    <th scope="row">Khiresh Odo</th>
    <td>7,223</td>
  </tr>
</table>
```

10. Formularios

- **Estructura y funcionamiento:**
  - `<form>`: Contenedor del formulario.
  - Atributos `action` y `method`.
- **Widgets y controles:**
  - `<label>`, `<input>`, `<button>`, etc.
- **Métodos de envío:** `get` y `post`

```html
<form action="/register" method="post">
  <label for="name">Name: </label>
  <input type="text" name="name" id="name">
  <label for="email">E-mail: </label>
  <input type="email" name="email" id="email">
  <button>Register</button>
</form>
```

11. Contenedores



Contenedores genéricos

Contenedores semánticos

3. CSS

Introducción a CSS

Cascade Style Sheet (CSS)

Evolución:

- CSS 1: 1996
- CSS 2: 1998
- CSS 2.1: 2011
- CSS 3: 2012

CSS es un lenguaje de marcado para describir la apariencia y el diseño del contenido de una página web. Sigue la filosofía del principio DRY (Don’t Repeat Yourself) para aplicar estilos entre diferentes elementos y páginas web.

Funcionamiento básico:

1. Declarar un conjunto de reglas para presentar el contenido.
2. Asociar cada regla con uno o varios elementos HTML.

Modelo de caja

La forma de representar un elemento HTML en CSS incluye cuatro capas:

1. Contenido
2. Espacio de relleno (transparente) entre el contenido y el borde.
3. Borde que rodea el contenido del elemento.
4. Espacio de margen (transparente) que separa el borde de otros elementos.

Capas:

- Margen
- Borde
- Relleno
- Contenido

Aplicar estilo a un documento

Usando <link>**:**

```
<!DOCTYPE html>

<html lang="en-US">

    <head>
```

```
        <link href="css/styles.css" rel="stylesheet" type="text/css">
    </head>
    <body>
        <p>Este texto será rojo según lo definido en la hoja de estilo externa.</p>
    </body>
</html>
```

Usando <style>:

```
<!DOCTYPE html>
<html lang="en-US">
    <head>
        <style>
          p { color: #26b72b; }
        </style>
    </head>
    <body>
        <p>Este texto será verde según lo definido dentro del documento.</p>
    </body>
</html>
```

Usando el atributo global `style`:

```
<p style="color: rgb(0 0 255)">Este texto será azul según lo definido.</p>
```

Selectores

Tipos de selectores:

Tipo o elemento:

```
h1, strong { color: darkcyan; }
```

Identificador:

```
#unique { font-family: Arial, sans-serif; }
```

Clase:

```
.alert { font-weight: bold; font-size: 18px; }
```

Pseudoclase:

```
a:hover { color: rebeccapurple; text-decoration: none; }
```

Pseudoelemento:

```
p::first-letter { color: black; text-transform: uppercase; }
```

Universal:

```
* { color: blue; font-size: 12px; }

div * { color: green; font-size: 11px; }
```

Atributo:

```
[value] { font-style: italic; }

a[href^="#"] { background-color: yellow; color: red; }
```

Combinaciones

Descendientes:

```
div span { border-color: black; }
```

Hijos directos:

```
article > p { text-align: justify; }
```

Parientes adyacentes:

```
section + h2 { text-align: left; }
```

Pariente sen general:

```
h1 ~ p { padding-top: 50px; }
```

Cascada y especificidad

Cascada: En caso de conflicto entre dos o más reglas, se aplica la última regla declarada.

Especificidad: Cálculo de un peso asignado a una declaración. En caso de conflicto de selección, se aplica la regla de mayor peso. Si el conflicto de peso persiste, se aplica el criterio de cascada.

Herencia

La herencia controla qué valor debe ser asignado a una propiedad que no haya sido especificada sobre un elemento. Es análogo al concepto de herencia en el Paradigma de Programación Orientada a Objetos.

Propiedades CSS

Color:

- `color`: Define el color del texto dentro de un elemento.
- `background-color`: Define el color del fondo de un elemento.

Fondo:

- `background-image`: Define una imagen de fondo para un elemento.
- `background-repeat`: Define un patrón de repetición para la imagen de fondo.

Tipografía:

- `font-family`: Define una lista de familias tipográficas.
- `font-size`: Define el tamaño de la tipografía.
- `font-style`: Define el estilo de la tipografía.
- `font-weight`: Define el peso o cantidad de negrita de la tipografía.

Texto:

- `text-decoration`: Define decoraciones sobre el texto.
- `text-align`: Define la alineación horizontal del texto.

Tamaño:

- `width`: Define la anchura del elemento.
- `height`: Define la altura del elemento.

Relleno:

- `padding`: Establece el espacio de relleno de un elemento.

Borde:

- `border`: Define el borde de un elemento.

Maquetación

Control de maquetación

Display

Determina cómo un elemento debe ser visualizado en términos de maquetación.

Visibilidad

Determina la visibilidad de un elemento pero sin modificar la maquetación del documento. Para eliminar el elemento de la maquetación hay que utilizar la declaración `display: none;`.

Flujo normal

Diseño por defecto que determina cómo posicionar un elemento en un documento antes de realizar cualquier tipo de maquetación. Todo elemento sigue el modelo de caja. Participantes primarios dentro del flujo normal son el elemento de bloque y el elemento de línea.

Flexbox

Diseño unidimensional para alinear y distribuir un elemento y su contenido. Cada elemento puede expandirse o encogerse.

Cuadrícula

Diseño bidimensional para alinear y distribuir un elemento y su contenido. Cada elemento de la cuadrícula puede ocupar las mismas celdas que otros elementos solapándose entre sí para crear capas.

Extensiones

Framework

Librería de estilos genéricos para facilitar y reducir el proceso de diseño del contenido de una página web. Promueve el diseño adaptable y generalmente utiliza algún tipo de preprocesador.

Syntactically Awesome Style Sheets (SASS)

Preprocesador de CSS que utiliza un lenguaje de script para generar código CSS. Soporta variables, anidamiento de selectores, estilos y propiedades, módulos y mixins.

Sintaxis

1. **SASS**: utiliza indentaciones, eliminando los símbolos de llaves `{}` y de punto y coma `;`.

2. **SCSS**: utiliza la sintaxis normal de CSS.

Características

Soporta el uso de variables, anidamiento de selectores, módulos con la directiva `@use` y mixins.

4. JavaScript

Introducción a JavaScript

- JavaScript es un lenguaje de scripting para crear páginas web interactivas, también conocido como ECMAScript.
- Es el único lenguaje de programación soportado de forma nativa por los navegadores.
- Posee un tipado dinámico y débil, y es multiparadigma (estructurado, orientado a objetos y funcional).

Funcionamiento básico

- El código JavaScript es compilado y ejecutado sobre la marcha por un navegador usando la compilación Just-In-Time (JIT).
- Al cargar una página web, el código se ejecuta automáticamente según su orden de declaración. También es posible ejecutar código manualmente a través de la consola interactiva del navegador.

Añadir scripts a un documento

- Se utiliza la etiqueta `<script>` para incrustar código JavaScript en un documento HTML.
- Es recomendable cargar siempre código JavaScript a través de ficheros externos para mantener una separación de funcionalidades y evitar incrustar código directamente en el elemento `<script>`.

Fundamentos del lenguaje

- La sintaxis de JavaScript es análoga a la de C/C++ y Java, pero también tiene elementos de Awk, Perl y Python.
- Soporta comentarios de línea `//` y de bloque `/* */`.
- Las sentencias están separadas por el símbolo de punto y coma `;`.

Declaración de variables

- `var` declara una variable con ámbito global o relativo a una función y es elevada a un ámbito superior (hoisting).
- `let` declara una variable local con ámbito de bloque y soluciona problemas de referencia que tiene `var`.
- `const` declara un nombre de constante de solo lectura con ámbito de bloque.

Tipado dinámico

- No es necesario declarar el tipo de datos. Las variables poseen el tipo del último valor asignado.

Tipos de datos

- `number`: Representa valores numéricos enteros o reales, sigue el formato IEEE 754 y tiene valores especiales como `+Infinity`, `-Infinity` y `NaN`.
- `string`: Representa una cadena de texto delimitada por comillas simples o dobles. Son inmutables y soportan el operador de concatenación `+`.
- `boolean`: Representa un valor lógico (`true` o `false`) y soporta la conversión automática de valores no booleanos.
- `null`: ausencia de un valor.
- `Undefined`: valor que no ha sido asignado.

Funciones

- Las funciones pueden ser tratadas como datos (first-class functions) y pueden ser declaradas mediante la palabra reservada `function` o usando la sintaxis de flecha (`arrow function`).
- La declaración de una función siempre es elevada a un ámbito superior (hoisting), permitiendo su uso antes de su declaración.

Objetos

- Un objeto es una colección desordenada de propiedades, cada una definida por un par nombre-valor.
- Se puede acceder a las propiedades de un objeto mediante la notación por punto `.` o por corchete `[]`.

Arrays

- Un array es un objeto especial utilizado para crear una lista de datos delimitados por corchetes `[]` y separados por comas `,`.
- Pueden contener elementos vacíos con valor `undefined`.

API del DOM

El Document Object Model (DOM) es una interfaz de plataforma cruzada que permite a los programas y scripts acceder y actualizar el contenido, la estructura y el estilo de los documentos HTML y XML.

Nodos del DOM

- Cada parte de un documento es un nodo del DOM.
- Los nodos pueden ser elementos, atributos, textos y más.

Acceso a elementos

- `document.getElementById(id)`: Obtiene un elemento por su ID.
- `document.getElementsByClassName(class)`: Obtiene elementos por su clase.
- `document.getElementsByTagName(tag)`: Obtiene elementos por su etiqueta.
- `document.querySelector(selector)`: Obtiene el primer elemento que coincida con el selector CSS.

- `document.querySelectorAll(selector)`: Obtiene todos los elementos que coincidan con el selector CSS.

Modificación de contenido

- Crear un nuevo elemento: const nuevoElemento = document.createElement('div');
- Agregar un nodo como hijo: document.body.append(nuevoElemento);
- Eliminar un nodo hijo: document.body.remove(nuevoElemento);
- Modificar contenido HTML de un elemento: nuevoElemento.innerHTML = ' Hola Mundo ';
- Modificar contenido textual de un elemento: nuevoElemento.textContent = 'Hola Mundo';
- Modificar le valor de un atributo: nuevoElemento.setAttribute('id', 'nuevoID');
- Modificar una propiedad CSS de un elemento: nuevoElemento.style.color = 'red';

Eventos

Concepto:

- Paradigma donde el flujo de ejecución es controlado por eventos.
- Basado en patrones Publicador/Suscriptor y Observer.
- Componentes principales: eventos y manejadores (handlers/listeners).

Funcionamiento en JavaScript

1. **Evento**: Señal de que algo ha sucedido (ej. clic en un botón).
2. **Manejador de Eventos**: Código que se ejecuta en respuesta a un evento.

Tipos de Manejadores de Eventos

En HTML: <button onclick="console.log('Click');">Click</button>

En JavaScript:

const btn = document.getElementById("btn");

btn.addEventListener("click", (event) => console.log("Click"));

Eliminar un manejador de Eventos

btn.removeEventListener("click", handleClick);

Objeto Event

- **Propiedades**:
  - `type`, `timeStamp`, `currentTarget`, `target`.
- **Método `preventDefault`**
  - event.preventDefault();

Propagación de eventos:

- **Fases**:

    1. **Captura**: Desde el más externo al más interno.
    2. **Objetivo**: El elemento que originó el evento.
    3. **Burbujeo**: Desde el más interno al más externo.

- **Método `stopPropagation`**

```
event.stopPropagation();
```

Temporización de Eventos:

- setTimeout: setTimeout(handler, 1000);
- setInterval: setInterval(handler, 1000);
- clearInterval: clearInterval(id);

5. Comunicación Cliente-Servidor

Protocolo HTTP

HyperText Transfer Protocol (HTTP)

- Protocolo de la capa de aplicación para transferir información entre cliente y servidor.
- Utiliza mensajes para intercambiar información y permite transferir cualquier tipo de recurso web.
- Es extensible y ha evolucionado con el tiempo.
- No tiene estado; cada comunicación es tratada como una operación independiente del resto.

Funcionamiento básico:

1. El cliente solicita un recurso a un servidor.
2. Se establece una conexión TCP entre cliente y servidor.
3. El cliente construye y envía un mensaje de petición.
4. El servidor procesa la petición, construye y envía un mensaje de respuesta.

Mensaje:

- Mecanismo para transferir información entre cliente y servidor.
- Tipos de mensajes: Request (petición) y Response (respuesta).

Método HTTP:

- Acciones más comunes: GET, HEAD, PUT, DELETE, POST.

Método	Significado	Característica
GET	Obtener la representación del recurso	Seguro
HEAD	Obtener información del recurso	Seguro
PUT	Crear/Reemplazar completamente la representación del recurso	Idempotente
DELETE	Eliminar completamente el recurso	Idempotente
POST	Enviar datos para provocar un cambio de estado al recurso	Cacheable

Same-Origin Policy (SOP)

- Mecanismo de seguridad utilizado por navegadores para restringir interacciones entre documentos o scripts cargados desde diferentes orígenes.
- Importante para evitar peticiones HTTP a orígenes distintos del inicial.

Cross-Origin Resource Sharing (CORS)

- Mecanismo basado en cabeceras HTTP para permitir que un servidor describa a un navegador cuáles son los orígenes permitidos para cargar otros recursos.
- Utilizado por muchas APIs y servicios web.

Servicio REST

REST (REpresentational State Transfer):

- Patrón de arquitectura para sistemas distribuidos.
- Premisa: todo recurso posee una representación de su estado transferible mediante comunicaciones estándar.
- Sistema que implementa este patrón es denominado RESTful.

Principios REST:

1. Definir una interfaz uniforme.
2. Seguir un modelo cliente-servidor.
3. Comunicaciones sin estado.
4. Comunicaciones cacheables o no-cacheables.
5. Jerarquía basada en capas.

Recurso:

- Entidad identificable y representable para ser transmitida.
- Modelos: Singleton, Colección, Subcolección.

API RESTful:

- Servicio web que implementa los principios REST.
- Utiliza el protocolo HTTP.
- Características: URIs para identificar recursos, formato de datos (principalmente JSON), métodos HTTP (GET, POST, PUT, DELETE).

Método HTTP	Operación	Descripción	Códigos de estado
`POST`	*Create*	Crear recurso	201 (👍); 400 (👎)
`GET`	*Read*	Obtener colección o recurso	200 (👍); 404 (👎)
`PUT`	*Update*	Actualizar recurso	200 (👍); 409 (👎)
`DELETE`	*Delete*	Eliminar recurso	204 (👍); 404 (👎)

JavaScript asíncrono

Bucle de eventos:

- Esquema para gestionar la ejecución de operaciones asíncronas.

Promise:

- Representa el posible resultado de una operación asíncrona.
- Métodos: then(), catch(), finally().
- Estados: Pending, Fulfilled, Rejected.

Encadenamiento de promesas:

- Permite encadenar operaciones asíncronas.
- Métodos devuelven otro objeto Promise.

async y await:

- async: función que devuelve una promesa.
- await: operador que resuelve una promesa y devuelve su resultado.
- Mejora la lectura y el mantenimiento del código asíncrono.

API Fetch

API Fetch:

- Colección de características para obtener un recurso local o remoto.
- Basado en promesas.
- Método: `fetch()`.

6. Servidor Web

1. Servidor web

- **Servidor web**:
  - Software ejecutado en el lado del servidor que procesa peticiones de clientes destinadas a obtener recursos web.
  - Utiliza el protocolo HTTP para la comunicación (servidor HTTP).
  - Sirve contenido estático y dinámico mediante la combinación con otros software del lado del servidor o servicios web.
- **Servidor web estático**:
  - Proporciona recursos cuyo contenido no ha sido modificado o procesado.
  - Generalmente utilizado en redes de distribución de contenido (CDN).
- **Servidor web dinámico**:
  - Proporciona recursos cuyo contenido ha sido modificado o procesado a petición del cliente o factores externos.
  - Utilizado en aplicaciones web.
- **Servidor HTTP**:
  - Socket abierto que escucha constantemente peticiones HTTP en un puerto de red específico para procesarlas y enviar respuestas HTTP.
  - Un socket es un mecanismo para intercambiar datos entre dos máquinas conectadas a una red.
  - Un puerto es una ubicación lógica de una conexión de red para identificar un protocolo o servicio ejecutado en una máquina.
- **Procesamiento de peticiones HTTP**:
  - Enfoque tradicional: buscar el recurso utilizando el path de la URL y realizar operaciones de lectura/escritura o ejecutar un programa/script asociado al recurso.
  - Enfoque actual (basado en el patrón MVC): utilizar un controlador para analizar la URL, obtener un modelo de datos del recurso y generar una vista aplicando una plantilla.

2. Node.js

- **Node.js**:
  - Entorno de ejecución de JavaScript basado en el motor V8 de Google.
  - Ejecuta código fuera del navegador.
  - Tiene una arquitectura dirigida por eventos basada en la librería libuv.
  - Ofrece una API estándar para trabajar sobre cualquier aspecto del sistema operativo subyacente (entrada/salida de ficheros y sockets, criptografía, compresión, etc).
- **Comando `node`**:
  - Permite ejecutar código JavaScript desde la línea de comandos.
  - Inicia un bucle REPL si es lanzado sin argumentos.
- **Diferencias entre JavaScript en un navegador y en Node.js**:
  - Distintos ecosistemas de APIs.
  - Navegador utiliza el objeto global `window` para interaccionar con la API del DOM.

    - Node.js no tiene objetos globales ni API del DOM, utiliza su API estándar para acceder a funcionalidades del sistema operativo.
- **Módulo**:
    - Cualquier archivo JavaScript utilizado por Node.js.
    - API estándar es una colección de módulos.
    - Tradicionalmente utiliza el sistema de módulos CommonJS pero también da soporte para el sistema estándar de ECMAScript.
    - *Exports, require(), module, __filename y __dirname.*
- **Paquete**:
    - Estructura de directorios descrita por un archivo `package.json` y un directorio de dependencias `node_modules`.
    - `package.json` contiene metadatos del paquete.
    - `npm` es el gestor de paquetes por defecto para Node.js.
- **Módulo `http`**:
    - Colección de características para trabajar con el protocolo HTTP.
    - Permite implementar un servidor web a muy bajo nivel.

3. Express.js

- **Express.js**:
    - Microframework web para Node.js utilizado para crear aplicaciones web.
    - Ofrece una capa sobre la API estándar de Node.js que implementa casi cualquier funcionalidad relativa a un servidor web.
    - Controla el ciclo de vida de petición-respuesta.
    - Gestiona el direccionamiento de métodos HTTP y URLs a través de manejadores.
    - Implementa el procesamiento de peticiones HTTP por capas utilizando middleware.
    - Permite generar contenido dinámico utilizando motores de plantillas.
- **Middleware**:
    - Función que acepta tres parámetros para procesar una petición HTTP: `req` (representa la petición HTTP), `res` (representa la respuesta HTTP), `next` (delegar en el siguiente middleware el procesamiento de la petición HTTP).
    - El procesamiento permite ejecutar un bloque de código específico y/o modificar los objetos Request y Response.
    - Tipos de middleware:
        - De aplicación: asociado a una instancia de Application.
        - De direccionamiento: asociado a una instancia de Router.
        - De gestión de errores: maneja errores surgidos durante el procesamiento de la petición.
        - Integrado: funciones integradas dentro del propio framework de Express.js (e.g., `express.static()`, `express.json()`).
        - De terceros: funciones de terceros que deben ser cargadas desde el middleware de aplicación o de direccionamiento.

7. Autenticación

Sesión

HTTP (HyperText Transfer Protocol):

- Es un protocolo de la capa de aplicación para transferir información entre cliente y servidor.
- No tiene estado, lo que significa que cada comunicación es independiente.
- Se requiere almacenar el estado de las comunicaciones en algún lugar, ya sea en el cliente o en el servidor, en forma de sesión.

Sesión:

- Información temporal que un servidor almacena para identificar a un cliente específico.
- Comienza cuando el cliente hace login y finaliza cuando hace logout.
- Cada sesión es única para cada cliente y no puede ser transferida a otros servidores.
- Generalmente es una variable que contiene un número aleatorio denominado UserID o SessionID.

Formas de almacenamiento de sesiones:

1. **Cookie:** Ofrece acceso simple pero obliga a encriptar su contenido.
2. **Servidor web (memoria):** Ofrece acceso rápido pero puede afectar la escalabilidad de la aplicación web.
3. **Sistema de almacenamiento general (base de datos):** Ofrece acceso distribuido y centralizado, pero agrega carga de trabajo.
4. **Sistema de almacenamiento dedicado (motor de caché de datos):** Ofrece acceso rápido.

Funcionamiento:

1. El cliente debe ser autenticado de forma correcta en el servidor para crear la sesión. Protocolos de autenticación (básica, bearer, digest, etc). Enviar la sesión al cliente (cookies, parámetros de URL, cabeceras HTTP, etc).
2. Posteriormente, el cliente siempre debe enviar la sesión para acceder a un recurso del servidor. El servidor verifica la validez y responde con el recurso solicitado o de forma apropiada.

Cookie

Funcionamiento:

- Las cookies almacenan información relativa a las comunicaciones entre un cliente y un servidor.
- Almacenadas por el cliente como un fichero o cadena de texto plano.

- Utilizadas para recordar el estado de la aplicación web, gestión de sesiones, personalización del frontend o seguimiento del comportamiento.

Atributos comunes de las cookies:

- **Domain:** Qué servidor y subdominios pueden recibir la cookie.
- **Expires:** Tiempo máximo de vida útil en el cliente.
- **HttpOnly:** Prohíbe su acceso desde código JavaScript.
- **SameSite:** Evita su envío en peticiones cruzadas.

Gestión de cookies en una aplicación web:

- Desde el lado del cliente mediante las APIs DOM y fetch.
- Desde el lado del servidor mediante middleware de Express.js.

JSON Web Token (JWT)

JWT:

- Mecanismo compacto y autocontenido para transmitir información de forma segura entre un cliente y un servidor.
- La información se almacena como un objeto JSON y se codifica en una cadena de texto denominada token.
- Cada token está firmado digitalmente utilizando un algoritmo HMAC o un par clave pública-privada.
- Utilizado tanto para la autenticación de usuarios como para la transmisión segura de información.

Estructura del JWT:

1. **Header (Cabecera):** Identifica el algoritmo de codificación y el tipo de token.
2. **Payload (Contenido):** Contiene los datos relativos al cliente.
3. **Signature (Firma):** Verifica la autenticidad del token.

Funcionamiento:

- El cliente se autentica enviando credenciales al servidor.
- El servidor crea un token JWT y lo envía al cliente.
- El cliente utiliza este token para acceder a recursos protegidos en el servidor.
- El servidor verifica la validez del token en cada solicitud.

Uso en JavaScript:

- En Node.js, se puede utilizar el paquete `jsonwebtoken` para crear y verificar tokens JWT.
- `jwt.sign()` crea un token a partir de un secreto o una clave privada.
- `jwt.verify()` verifica la autenticidad del token y devuelve el payload decodificado si es válido.

8. Persistencia

Terminología

1. **Base de datos y DBMS (DataBase Management System):**
   - Una base de datos es una colección organizada de datos.
   - Los DBMS son software que permiten almacenar, obtener y actualizar datos.
   - Tipos de bases de datos: Relacional (estructura fija) y No relacional (estructura flexible).
   - Ejemplos de DBMS: MongoDB, MariaDB, PostgreSQL, etc.
2. **Object Relational Mapping (ORM):**
   - Técnica para realizar consultas u operaciones CRUD a una base de datos relacional utilizando un lenguaje de programación.
   - Mapea los datos de la base de datos a tipos concretos del lenguaje de programación.
3. **Object Document Mapping (ODM):**
   - Técnica análoga al ORM pero aplicable a bases de datos no relacionales.
   - En el caso de MongoDB, los documentos se mapean a objetos JavaScript.
4. **Operaciones CRUD:**
   - Acrónimo de "Create, Read, Update and Delete", referente a las cuatro operaciones básicas sobre la capa de persistencia de un software.
   - También utilizado en APIs REST.

MongoDB

Sistema de gestión de base de datos de código Abierto, orientada a documentos que encapsulan y codifican los datos. Utilizan rutas o URLs para identificar los documentos.

1. **Características:**
   - No relacional, con esquema flexible.
   - No existen las operaciones de transacción ni los joins.
2. **Conceptos:**
   - **Base de datos:** Contenedor de colecciones.
   - **Colección:** Grupo de documentos, análogo a una tabla en una base de datos relacional.
   - **Documento:** Objeto binario tipo JSON (BSON), análogo a una fila en una base de datos relacional.
   - **Campo:** Clave que tiene asignada un valor específico, análogo a una columna en una base de datos relacional.
3. **Comandos de `mongo` (cliente de línea de comandos):**
   - `show dbs`: Muestra todas las bases de datos.
   - `use database_name`: Cambia la base de datos a utilizar.
   - `show collections`: Muestra todas las colecciones de la base de datos actual.
   - `db.collection.find(query)`: Devuelve los resultados de la colección que satisfacen la query.
   - `db.collection.insertOne(document)`: Agrega un documento a la colección.

- `db.collection.deleteOne(query)`: Elimina el primer resultado de la colección que satisfaga la query.
- `db.collection.drop()`: Elimina la colección de la base de datos.

Mongoose

1. **Características:**
   - Librería ODM para MongoDB y Node.js.
   - Traduce objetos JavaScript a su correspondiente representación en MongoDB.
   - Posee un sistema de validación y tipado.
2. **Conexión a MongoDB:**
   - `mongoose.connect(url, [options])`: Permite establecer una conexión con la base de datos.
3. **Esquema:**
   - Define la estructura de un documento junto a sus valores predeterminados, validadores, etc.
   - Representada con la clase `Schema`.
4. **Modelo:**
   - Contenedor de un esquema.
   - `mongoose.model(name, schema)`: Permite compilar un modelo a partir de un esquema.
5. **Operaciones Básicas:**
   - **Crear/Insertar documentos:**
     - `new Model(doc).save([options])`: Inserta o edita un documento en una colección.
     - `Model.create(docs, [options])`: Inserta uno o más documentos en una colección.
     - `Model.insertMany(docs, [options])`: Valida e inserta un array de documentos en la colección.
   - **Actualizar documentos:**
     - `Model.updateOne(filter, update, [options])`: Actualiza el primer documento que satisfaga un filtro.
     - `Model.updateMany(filter, update, [options])`: Actualiza todos los documentos que satisfagan un filtro.
   - **Eliminar documentos:**
     - `Model.deleteOne(conditions, [options])`: Elimina el primer documento que satisfaga unas condiciones.
     - `Model.deleteMany(conditions, [options])`: Elimina todos los documentos que satisfagan unas condiciones.
   - **Leer/Obtener documentos:**
     - `Model.find(filter, [projection, options])`: Obtiene un array de documentos que satisfacen un filtro dado.
     - `Model.findById(id, [projection, options])`: Obtiene un único documento buscando por su identificador.
     - `Model.findOne(conditions, [projection, options])`: Obtiene un documento que satisfaga un filtro dado.

6. **Operaciones Derivadas:**
    - **Combinar búsqueda con operaciones básicas:**
        - `Model.findOneAndUpdate(filter, doc, [options])`: Busca y actualiza un documento.
        - `Model.findByIdAndUpdate(id, doc, [options])`: Busca y actualiza un documento por su identificador.
        - `Model.findOneAndDelete(conditions, [options])`: Busca y elimina un documento.
        - `Model.findByIdAndDelete(id, [options])`: Busca y elimina un documento por su identificador.