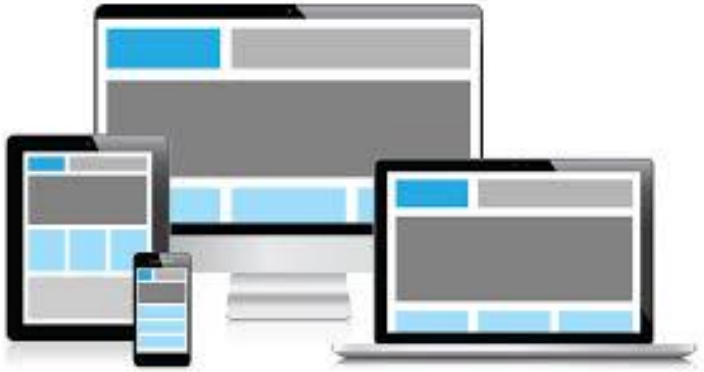
CENEC MALAGA

Introducción a tecnologías web



Formación Profesional DAW

Diego Martín
diego.martin@cenecmalaga.es

Índice

¿Qué se necesita?

1. Un navegador web
(i.e: Firefox, Chrome, Edge, etc.)

2. Un editor de texto ASCII
(e.g: Visual Studio Code, Notepad, Notepad++, etc.)

3. Otras herramientas útiles opcionales
(e.g: Postman, Beeceptor, etc.)

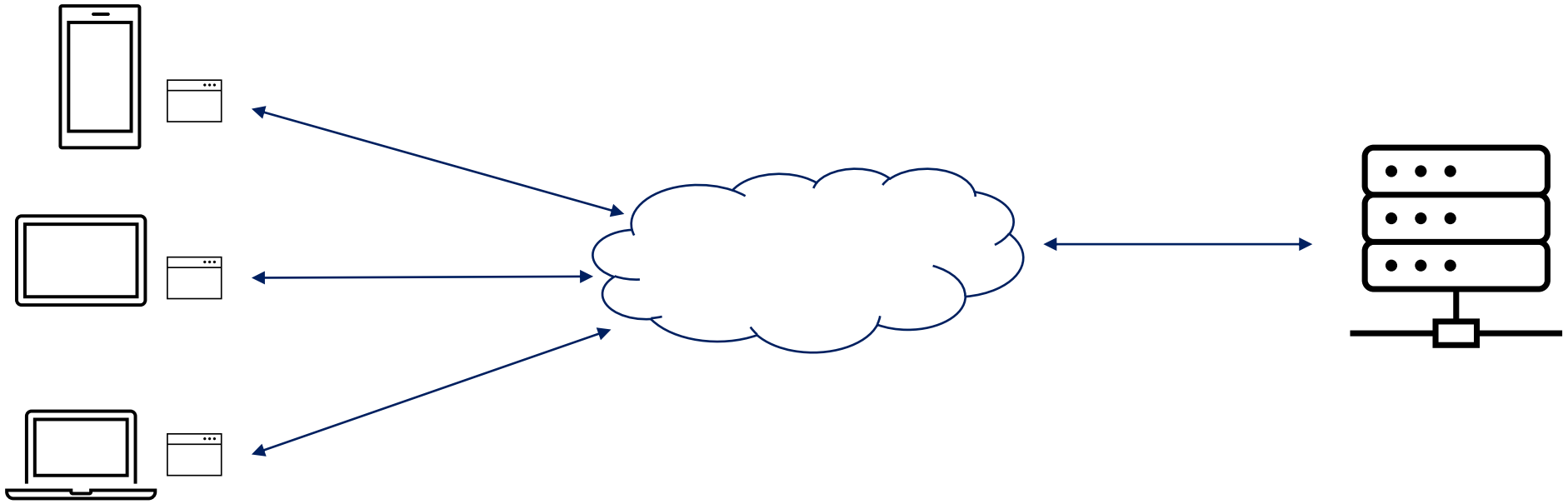
Los estándares web – HTML, CSS y ECMA Script

- **HTML** (*HyperText Markup Language*) para el contenido y la estructura de una web
- **CSS** (*Cascade Styling Sheets*) para el aspecto y estilo
- **ECMA Script** para el comportamiento
  - **JavaScript** es la implementación más conocida del estándar ECMAScript, aunque hay otras como JScript

Existe otro estándar web, WebAssembly o Wasm, que permite que a los navegadores ejecutar código binario

El modelo cliente-servidor en la web

- Los dispositivos clientes (PC, móvil, tablet, portátil, etc.) con los que un usuario interactúa, tienen un **navegador web** (e.g: firefox, chrome, edge, opera, safari, etc.) capaz de entender lenguajes HTML, CSS y JavaScript
- Para ejecutar código JavaScript los navegadores web tienen incorporado un motor de ECMAScript y de *WebAssembly*
  - El motor del navegador Chrome se llama V8, y se usa también en *NodeJs* para ejecutar javascript fuera de un navegador
- Los navegadores web envían *peticiones* **HTTP** a un servidor, a través de la red, y los servidores procesan la petición y devuelven una *respuesta*. A menudo esa respuesta consiste en archivos estáticos (e.g: archivos CSS, html, js, etc.) u otro tipo de contenido (e.g: XML, JSON, bytes, etc.)



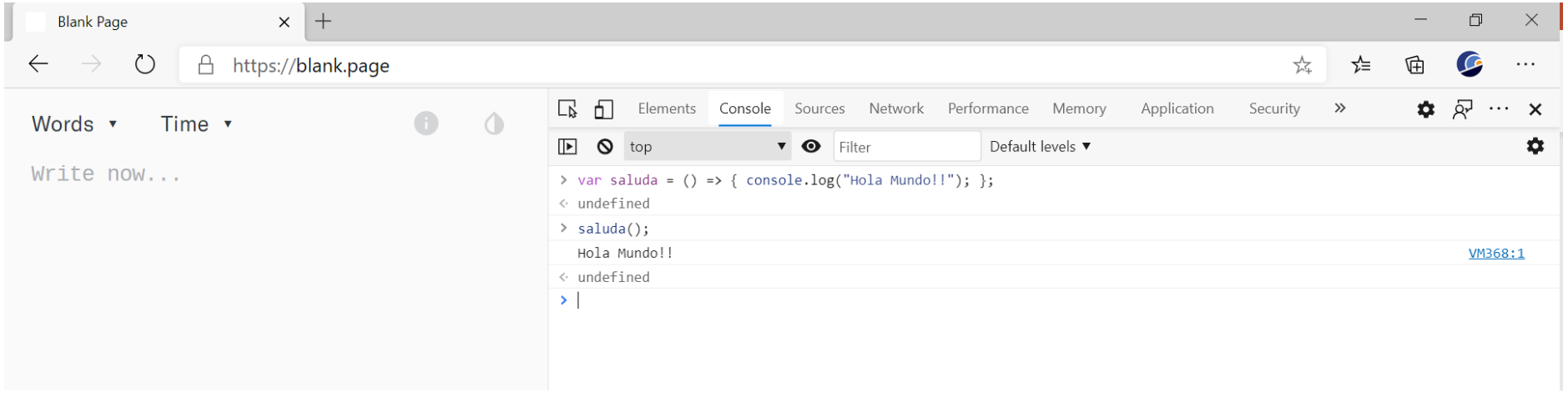
El modelo cliente-servidor en la web

- Para empezar a desarrollar con HTML basta con crear un documento html con las etiquetas básicas y abrirlo en el navegador *<!doctype html><html><head></head><body></body></html>*

- Para añadir estilo a un documento HTML basta con utilizar el atributo *style* en las etiquetas html, incrustar código CSS entre las etiquetas *<style>* y *</style>* o referenciar archivos css externos con *<link rel="stylesheet" href="miArchivo.css">*

- Para empezar a desarrollar con JavaScript, basta con acceder a la consola de cualquier navegador web y escribir código, aunque la práctica habitual es la de escribir código JavaScript en archivos con extensón *.js referenciados por algún documento HTML y dejar que el navegador web lo interprete al renderizar el documento, o incrustar el código entre *<script type="application/javascript">* y *</script>*

  - Si no se require visualización en un navegador, también se ejecutar código JavaScript con *headless browsers* o con motores de JavaScript autónomos como el V8 que está disponible por consola al instalar *NodeJs*



Ejercicios básicos con HTML, CSS y JavaScript

1. Crea una página web solamente con contenido donde se lea *Hola mundo* dentro de unas etiquetas de párrafo <*p*> y </*p*>

2. Añade estilo a ese texto para que se muestre de otro color, en negrita, a un mayor tamaño y para que el cursor se cambie a una mano cuando se ponga encima del párrafo. ¿Qué otras formas existen para alcanzar el mismo resultado en el estilo?

3. Ahora añade comportamiento con JavaScript para que cuando se haga click en el párrafo, el texto pase a ser *Hola mundo interactivo*. ¿Qué otras formas existen para alcanzar el mismo comportamiento?

WebAssembly

- Los navegadores soportan y soportarán JavaScript, pero no es la única forma de programar comportamiento en cliente (i.e: front-end)
- WebAssembly o **Wasm** es un estándar abierto que permite la ejecución de código binario, compilado con lenguajes como C, C++, Rust, Java o C# en un navegador o en una máquina virtual y ofrece un nivel de rendimiento muy alto
- Todos los navegadores modernos, tanto de escritorio como móviles, ya soportan WebAssembly
- Existen frameworks que permiten desarrollar código para WebAssembly, para que los navegadores lo ejecuten
  - Microsoft ha lanzado a finales de 2020 ***Blazor*** https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor, que permite programar aplicaciones web en cliente con C# o F# utilizando estándares web abiertos, sin plugins y sin transpilación de código
- ¿Es WebAssembly el futuro de la web? ¿Sustituirá a JavaScript o simplemente coexistirán?

jQuery

- **Librería de JavaScript** que facilita el acceso al DOM y la comunicación AJAX con algún servidor, entre otras cosas
  - Descarga la última version estable de https://jquery.com/download (e.g: *jquery-3.5.1.js*)
- En los últimos años **jQuery ha perdido mucha relevancia**. Algunas razones:
  - Es una librería "pesada" (88KB en su version minificada) y a menudo solamente se utiliza un par de funcionalidades
  - Todo lo que jQuery hace, se puede hacer con JavaScript o con TypeScript directamente y con mejor rendimiento
  - Los frameworks y librerías como *Angular*, *React* o *Vue* incluyen sus propias utilidade
  - Algunas librerías populares como Bootstrap han decidido sustituir jQuery por *vanilla* JavaScript (e.g: Bootstrap 5 ya no usa jQuery)
  - El mundo del desarrollo front-end sufre de una interminable *fatiga* (ver https://css-tricks.com/javascript-fatigue) y los frameworks y librerías aparecen y desaparecen continuamente a un ritmo vertiginoso y no siempre de forma racional

JSON

- JSON (JavaScript Object Notation) es el **formato de representación de datos** más usado en la web sustituyendo a menudo a XML
- También se usa mucho para configuraciones o incluso para persistir datos estructurados en muchas bases de datos
- Es más legible que XML
- El tipo MIME oficial es "**application/json**" (e.g: en cabecera http `Content-Type: application/json`)
- Un objeto JSON se define con `{}` o con `[]` y soporta los siguientes **tipos de datos**:
  - *Numbers*: enteros o decimales
  - *String*: cualquier texto o incluso fechas en formato ISO 8601 (e.g: "`2021-01-15T20:00:00.000Z`")
  - Boolean: valores `true` o `false`
  - *Array*: entre caracteres `[]` que pueden contener valores simples u objetos
  - *Object*: cualquier objeto entre caracteres { }
  - *null*: para indicar explícitamente que no hay valor

```json
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

HTTP

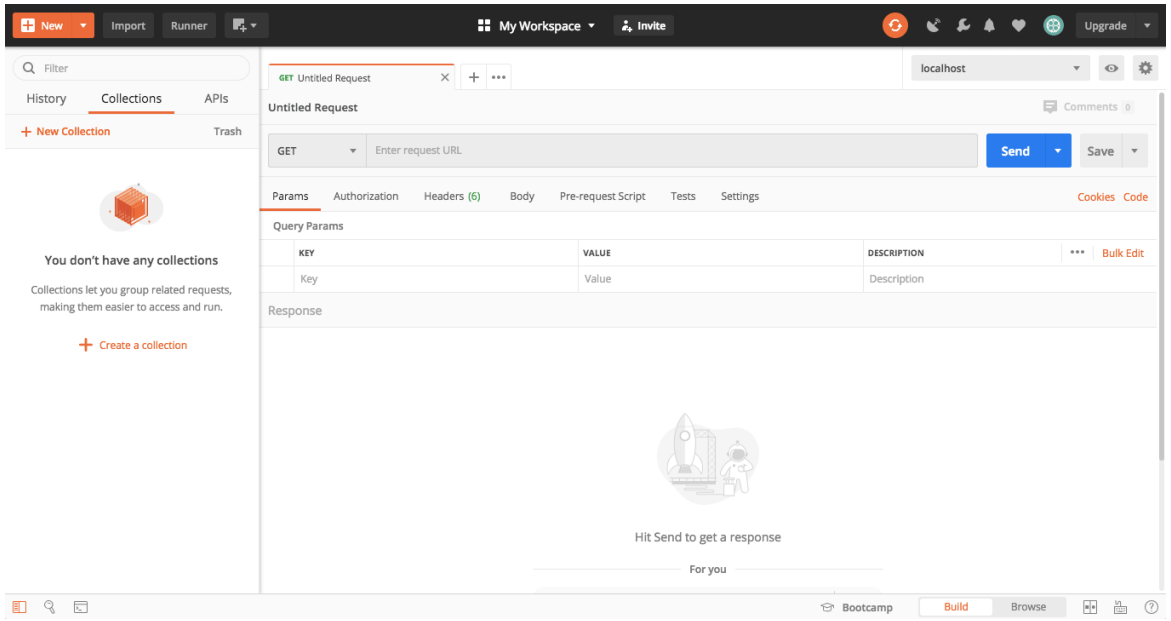
- **HTTP** (*Hypertext Transfer Protocol*) es un **protocolo** de capa de aplicación para comunicación en internet
- Es la base para el funcionamiento de la web. Cada vez que escribimos una *url* en un navegador, éste está enviando una petición http
- Hay varios métodos HTTP, también llamados **verbos HTTP**: GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD
- Cuando se diseña una API REST se deben seleccionar los verbos HTTP que mejor representen el tipo de operación
- Todos los verbos HTTP son idempotentes menos POST, que se utiliza para crear nuevos recursos

HTTP Verb	CRUD
POST	Create
GET	Read
PUT	Update/Replace
PATCH	Update/Modify
DELETE	Delete

Explora HTTP con Postman

1. Descarga la aplicación Postman https://www.postman.com/downloads o utiliza la versión web

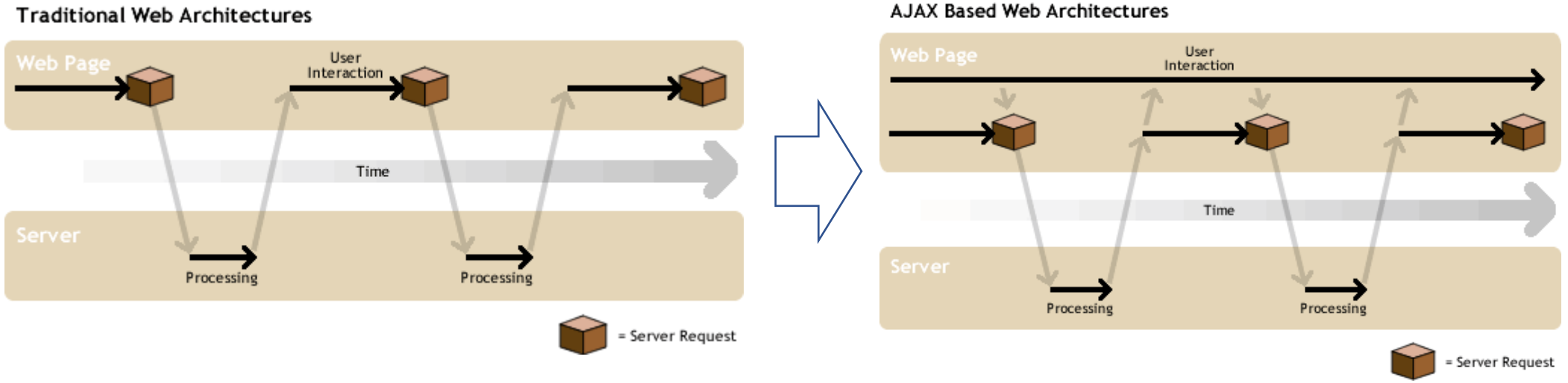
2. Explora su interfaz y la forma en la que se pueden crear colecciones y peticiones http utilizando todos los verbos



Postman es una herramienta fundamental para cualquier desarrollador web de front o back end y para testeadores. Existen otras herramientas similares como Insomnia o Fiddler.

AJAX

- AJAX (Asynchronous JavaScript And XML) es una técnica utilizada en aplicaciones cliente (i.e: aplicaciones en JavaScript que se ejecutan en un navegador) para enviar y recibir datos de un servidor de forma asíncrona
- Permite a las aplicaciones web cambiar contenido dinámicamente sin necesidad de recargar la página HTML completa
- Antiguamente se utilizaban objetos `XMLHttpRequest` (XHR) para interactuar con servidores
- Ahora `fetch()` es una alternativa más elegante que XHR que además utiliza promesas (e.g: las promesas evitan el *callback hell*). Para aprender más sobre promesas investiga https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises
- En la actualidad es mucho más común utilizar AJAX con JSON que con XML



Ejercicios básicos con AJAX

1. Crea una página web con una dependencia a la última versión de jQuery

2. Añade una etiqueta llamada *Url*, un caja para texto de entrada y un botón con el texto *Get*

3. Crea un servidor web HTTP capaz de recibir peticiones a una url específica y responder con un 200 OK. Se recomienda utilizar https://beeceptor.com para simiplificar la tarea

4. Añade funcionalidad a la web para que envíe una petición HTTP de tipo GET a la Url indicada en la caja de texto cuando se haga click en el botón. Haz que muestre el contenido de respuesta

5. Intenta hacer lo mismo pero esta vez utilizando JavaScript puro o bien con XHR o preferiblemente con `fetch()`

Para documentación sobre AJAX con jQuery visita https://api.jquery.com/jQuery.ajax

Ejercicios con JSON y AJAX

1. Explora los diferentes verbos HTTP como POST, PUT, PATCH, DELETE y GET configurando una API remota REST utilizando buenas practicas de diseño creando los siguientes *endpoints* en la herramienta de **Beeceptor** y pruébalos con **Postman**:

   GET api/personas
   Debería devolver varias personas con una respuesta 200 OK junto con un array JSON que contenga objetos que representen personas

   POST api/personas
   Debería devolver 201 Created cuando se envía JSON que represente a una persona (no hace falta que el servidor cree ningún recurso)
   También debería devolver una cabecera HTTP indicando la url del recurso creado (e.g: `Location: api/personas/4`)

   PUT api/personas/3
   Debería devolver 200 OK cuando se envía JSON que representa a la persona que sustituirá a una existente con Id 3

   DELETE api/personas/3
   Debería devolver 200 OK cuando se envía la petición http

2. Implementa una web que utilice JavaScript o jQuery a tu elección, y sea capaz de enviar peticiones HTTP al servidor *mock* anterior utilizando al menos dos verbos a elegir y mostrando por pantalla la respuesta.

3. Cambia el *mock* de algún endpoint de escritura para que devuelva un error 400 Bad Request y haz que la web lo muestre.

Una petición http que contiene datos en forma de JSON debería incluir una cabecera HTTP `Content-Type: application/json`