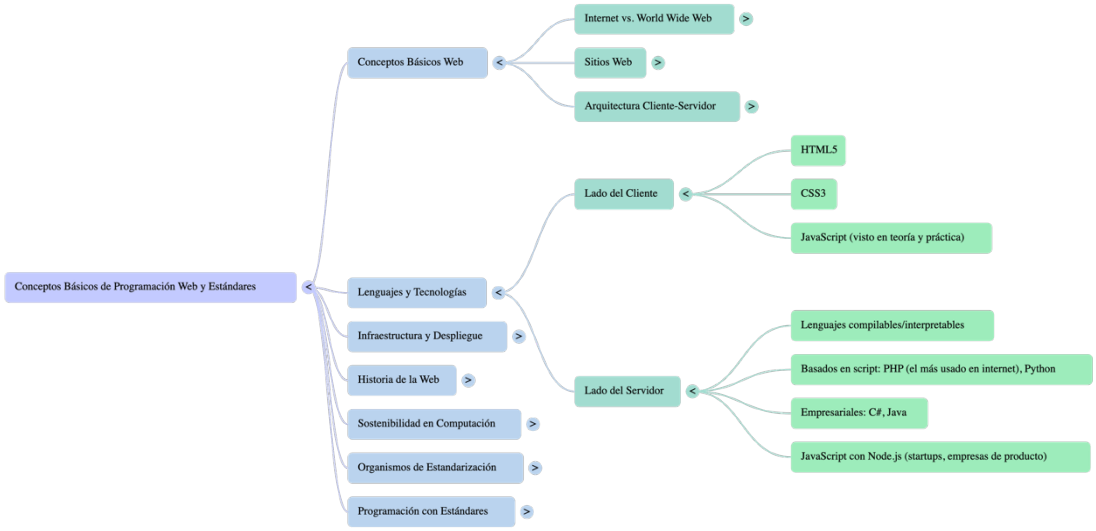
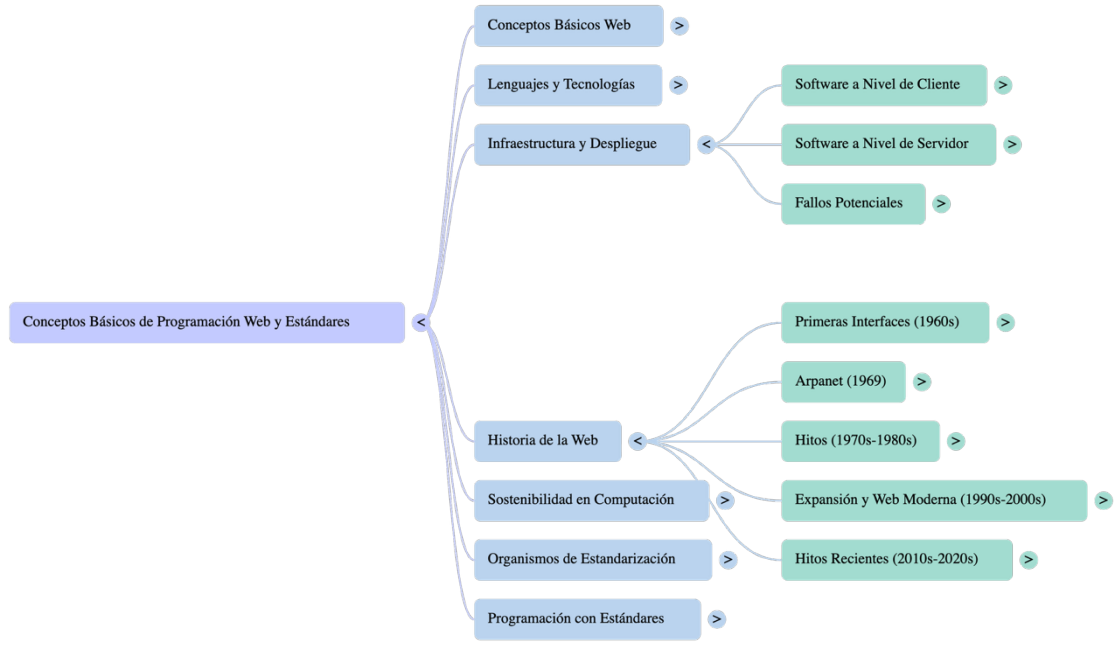
NOTAS DE SEMINARIO 1

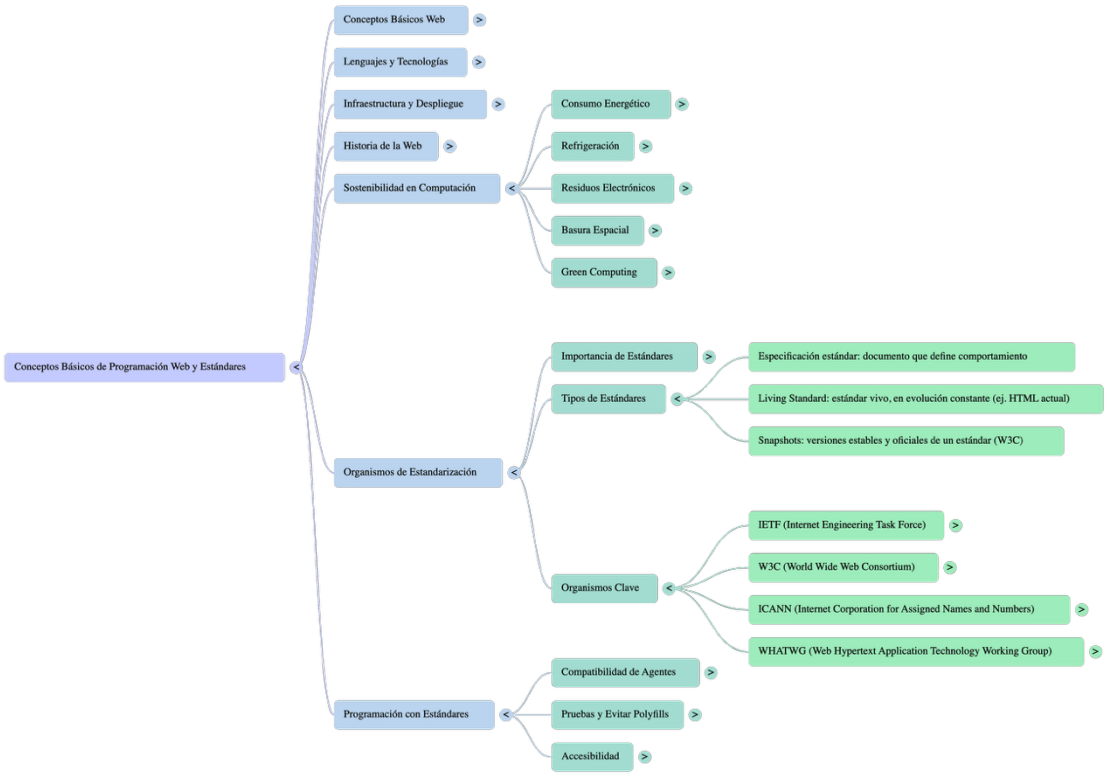
Importante: Este documento se trata de un resumen generado por IA generativa a partir de la grabación de la clase. Por tanto, no contiene todo lo visto en el aula. Solo se presenta como ayuda al estudio y complementar aspectos que podrían haberse olvidado. Es responsabilidad del estudiante dar un uso responsable a este material.

Este material pertenece a la asignatura Programación Web (Prof. José Raúl Romero) – curso 25/26.

MAPA MENTALES







RESUMEN GENERADO

1) INTERNET VS. WORLD WIDE WEB

- **Internet**
  - Red global de nodos computacionales interconectados sobre **TCP/IP**.

    - TCP proporciona transporte fiable; IP, direccionamiento/ruteo.
    - Actúa como **infraestructura física y lógica** sobre la que corren servicios.
- **World Wide Web (WWW)**
    - Origen: **conjunto de documentos** (HTML) enlazados mediante **hiperenlaces** ⇒ grafo de documentos por temáticas.
    - Evolución: hoy la Web funciona más como **infraestructura de servicios** accesibles vía **APIs**; el navegador descarga la app (HTML/CSS/JS) y esta **consume APIs** (generalmente HTTP) para datos/acciones.

2) SITIOS ESTÁTICOS VS. DINÁMICOS

- **Sitios estáticos**
    - Conjunto de **recursos**: HTML, CSS, JS, imágenes, PDF, etc.
    - No hay lógica de servidor: el navegador ("**agente**") **interpreta/ejecuta** los recursos.
    - Ejemplo académico: alojamiento personal en la UCO (carpeta tipo WWWDOCS con permisos públicos, accesible bajo URL con el login del estudiante).
    - Aunque un JS pueda animar mucho la página, **desde el servidor** sigue siendo un **recurso estático**.
- **Sitios dinámicos / aplicaciones web**
    - El **HTML se construye en tiempo de petición** (plantillas/servidor) a partir de **BD** y lógica.
    - Escenario típico: catálogo enorme (p. ej., Amazon) ⇒ no se codifica cada página; se renderiza dinámicamente según un **ID** (producto 757 ⇒ consulta BD ⇒ render).
    - Requiere **servidor de aplicaciones** (interpreta/compila, accede a BD y servicios externos) además del **servidor web**.

3) ROLES Y PIEZAS: CLIENTE, SERVIDOR WEB, SERVIDOR DE APLICACIONES

- **Cliente (agente/navegador)**
    - Ejecuta **HTML5, CSS3 y JavaScript**.
    - Variabilidad de agentes (versiones y *vendors*) complica las pruebas.
- **Servidor web**
    - Recibe peticiones **HTTP** (p. ej., GET /index.html) y **sirve recursos estáticos** desde un **sistema de ficheros público** (requiere permisos adecuados).
    - Si se pide un recurso dinámico (p. ej., .jsp, .php), **delegará**.
- **Servidor de aplicaciones**
    - Ejecuta código (Java/JSP, PHP, Python, C#, Node.js…), accede a otros servidores como bases de datos/APIs, **genera HTML** o JSON y lo devuelve al servidor web para que este lo entregue al cliente.
- **HTTP (protocolo textual)**

    - Define métodos para **solicitar recursos**, **enviar/actualizar** datos o **borrarlos** (se verán en detalle).
    - Base de **integración** (consumo de APIs) y de la navegación.

4) LENGUAJES Y ECOSISTEMA TECNOLÓGICO

- **Cliente**: HTML5, CSS3, JavaScript (ECMAScript).
- **Servidor**: Java, C# (.NET), PHP, Python, Node.js (JS en servidor), etc.
    - **Node.js**: muy usado en **empresas de producto/startups**; en sectores **corporativos** (banca, telco, retail) predominan **Java/C#** para sistemas *core*.
    - **PHP**: su estadística de uso está "inflada" por **CMS** (*Content Management Systems*) como WordPress y Joomla, y plataformas como **MediaWiki**; en proyectos corporativos nuevos su peso relativo es menor, pero el parque de instalaciones **es enorme**.

5) DESPLIEGUE, MODALIDADES Y FIABILIDAD

- **Despliegue**: publicar recursos en un servidor accesible (estáticos o artefactos de servidor).
- **Apps de escritorio "envoltorio"** de web (ej.: **Microsoft Teams**) comparten código/UX con su versión web.
- **Superficie de fallo** amplia:
    - Cliente (navegador/versión), servidor web, servidor de aplicaciones, **red**, **infraestructura** (incl. seguridad/ataques).
    - **Probar** sistemas web es **complejo** por la diversidad de agentes y rutas de ejecución.

6) HISTORIA Y HITOS (VISIÓN CRONOLÓGICA CON FOCO EN LA WEB)

- **Años 60**: interfaces de línea de comandos.
- **1968 – "The Mother of All Demos" (Douglas Engelbart)**
    - Introduce **ratón**, **editores de texto**, **edición colaborativa** ¡sin PC ni Internet aún!
- **1969**: **ARPANET** (64 nodos) como precursor de Internet.
- **1971**: primer **email**; necesidad de **FTP** para mover ficheros.
- **1972**: **Creeper**, primer "virus" que **se autopropaga en red** (hito por distribución).
- **1977**: criptografía de **clave pública**; en Europa surge una red homóloga que luego se interconecta.
- **1981–83**: FTP sobre TCP; formalización **TCP/IP (1982)**; Apple lanza interfaz **gráfica (1983)**.
- **Finales 80**: velocidades aún bajas; primeras conexiones "masivas" de investigación/academia.
- **1990s**
    - **Yahoo!** como **directorio** editado a mano (pre-buscadores).

    - **1994**: se funda **Amazon** (tarda años en ser rentable; inversión masiva).
    - **1995**: **Netscape** (rama **Mozilla**), primer navegador gráfico popular.
    - **1997**: nacen **JavaScript** y **PHP**.
    - **1998**: **Google** y **PageRank** profesionalizan la **búsqueda** (bots que indexan/relevancia por enlaces, etc.).
- **2000s**
    - Móviles empiezan a **renderizar HTML/JS**;
    - **2004**: redes sociales;
    - **2005**: **Ajax** trae **asincronía** granular (actualiza "trozos" de interfaz sin refrescar toda la página), **YouTube**,
    - **2008: Chrome**.
- **2010s**
    - **2010**: **Responsive Web Design** (layouts fluidos/adaptativos).
    - **2015**: **ES6/ES2015** consolida el lenguaje (módulos, clases, let/const, etc.), cataliza **Node** y **frameworks** modernos.
    - **2013–2016**: auge de **React**, **Angular**, **Bootstrap**, **Vue**.
    - **2018**: impulso a **accesibilidad** (WCAG).
- **2020s**
    - **SPA** (Google Docs, Instagram "scroll infinito") y **serverless** (cliente "grueso" + APIs gestionadas).
    - **HTTP/3** en despliegue progresivo; impulso de **IPv6**; **WebAssembly** se consolida.

7) SOSTENIBILIDAD Y "GREEN COMPUTING"

- **Centros de datos**: consumo estimado cercano al **2%** de la electricidad global; refrigeración (circuitos de agua), renovación de hardware ⇒ **residuos electrónicos**.
- **IA** (2022): estimaciones de consumo eléctrico **creciente**; su auge sumado al de centros de datos eleva la huella.
- Ubicación de **data centers** en entornos fríos/subterráneos o cercanos al mar ⇒ **eficiencia de refrigeración**.
- **Green computing**: algoritmos/sistemas que buscan **menor consumo** manteniendo funcionalidad/rendimiento.

8) ESTÁNDARES Y ORGANISMOS (QUÉ BUSCAR Y DÓNDE)

- **IETF (Internet Engineering Task Force)**
    - Protocolos de Internet y aspectos de **transporte/red**; publica **RFC** (REQUEST FOR COMMENTS).
    - Relevantes: **HTTP/1.1 – RFC 2616** (histórica) y **HTTP/3 – RFC 9114**.
    - **JSON** es una excepción interesante: también estandarizado bajo IETF.
- **W3C (World Wide Web Consortium)**
    - **Lenguajes y tecnologías web**: **CSS**, **XML**, (históricamente **HTML**).

    - Mantiene **versiones estables** (snapshot) y herramientas como el **validador de HTML/CSS**.
- **WHATWG (Web Hypertext Application Technology Working Group)**
    - Desde ~2014 los **vendors de navegadores** impulsan aquí los **living standards**: **HTML (living)**, **DOM**, **URL**, **fetch**, etc.
    - **2019**: acuerdo de **cooperación** W3C–WHATWG: WHATWG mantiene el **estándar vivo**; W3C publica **snapshots** estables.
- **ICANN**
    - Gestiona **nombres de dominio** y su delegación global (**DNS**).
    - Noción de **"transparencia de nombres"**: el nombre legible (e.g., www.uco.es) **oculta** la dirección **IP** real a la que resuelve.
- **IANA**
    - Autoridad para **asignación de números**: **direcciones IP**, **puertos** y otros identificadores de protocolos.

Nota histórica técnica: el control del direccionamiento/servidores raíz se hizo patente en eventos críticos (p. ej., **11-S, 2001**), afectando la **resolución y alcance** de Internet fuera de dominios locales.

9) VALIDACIÓN, ACCESIBILIDAD, SEO Y COMPATIBILIDAD ENTRE NAVEGADORES

- **Por qué validar y seguir estándares**
    - **Interoperabilidad** entre navegadores/agentes.
    - **Compatibilidad hacia atrás** y menor deuda técnica.
    - **Herramientas de validación** (W3C) reportan **warnings**, **errores** y **errores fatales** (p. ej., atributos deprecados, sintaxis inválida o recursos mal enlazados).
    - Impacto en **SEO** (mejor estructura semántica; aunque el **SEO clásico** pierde peso por respuestas **generativas** integradas en buscadores).
- **Accesibilidad**
    - Estándares **WCAG (p. ej., 2.2)**; requisitos **europeos** y **españoles** (BOE: **Real Decreto 2018** y **Ley 11/2023**).
    - Diseño para **personas con diversidad funcional** (visual, motora, cognitiva, daltonismo, etc.): contraste, semántica, navegación por teclado, etiquetas ARIA correctas, etc.
- **Compatibilidad de características (features)**
    - Cada navegador puede exponer **APIs y comportamientos distintos** (incluso funciones específicas como estado de batería).
    - Estrategias:
        - **Detección de características** (*feature detection*) con GUARDS en JS.
        - Consultar **living standards** y tablas de compatibilidad.
        - **Web Platform Tests** (wpt.fyi) para ver **qué navegador pasa qué tests** de cada API.
        - Evitar **polyfills** salvo necesidad: lo nativo suele ser más robusto.

        - Decidir **coste/beneficio**: si el **98%** de usuarios ya tiene `fetch`, quizá no compense polyfill para el **2%** restante.

10) SPA, SERVERLESS Y PATRONES MODERNOS DE FRONT-END

- **SPA (Single Page Application)**:
    - Interacción en **una sola página**; navegación/estado mediante **JS** (ej.: Google Docs, Instagram).
    - Ventaja: UX fluida; Implicaciones: **routing** en cliente, **gestión de estado**, **SEO** más complejo.
- **Arquitecturas serverless**:
    - Lógica en **APIs gestionadas**; cliente "grueso" en el navegador; menor gestión de servidores propios.
    - Alineado con **consumo de APIs HTTP** y despliegues en **cloud**.