0. Qué entra seguro (según todo lo que has apuntado)

- Desde **Tema 1** en adelante (Tema 0 no).
- **Diferencia frontend / backend** (muy importante).
- **HTTP** (versiones, métodos, códigos de estado...).
- **HTML, CSS, JavaScript** (JS es lo más importante del cliente).
- **Node.js + Express + Socket.io** (hasta sockets incluidos).
- **JSON, package.json, package-lock, event loop**.
- **Accesibilidad** (teoría).
- **Talleres**: los dos primeros

Cosas que **no son relevantes**:

- jQuery, TamperMonkey, Bootstrap, Sass/Less (solo saber que existen).
- A partir de la diapositiva ~140 de Node/Express "no entra".

1. HTTP + temas intro web

Tengo que saber

- **Diferencia frontend vs backend**
  - Qué hace cada uno, tipos de tareas típicas.
- **Aplicaciones para Internet / sitios web / servicios web**
  - Ventajas / desventajas.
  - Diferencias entre *aplicación web* y *servicio web*.
- **HTTP versiones**
  - HTTP/1.0 vs 1.1 vs 2 vs 3 → qué cambia a nivel general (no estadísticas de uso).
  - Saber que HTTP es **stateless**.
  - Saber qué es el **User-Agent** (por encima).
- **URL / URI** → qué son y para qué sirven.
- **Peticiones HTTP**
  - Qué es una petición HTTP.
  - **Métodos** (tabla de colores que menciona):
    - Saber para qué sirve GET, POST, PUT, DELETE, etc.
    - Qué métodos son **safe** y/o **idempotentes**.
    - Qué significa "safe" e "idempotente".
  - **GET vs POST** (muy clave):
    - Cuándo NO usamos GET → cuando hay datos sensibles (contraseñas, etc.) → usar POST.
- **Respuestas HTTP**
  - **Categorías de códigos**: 1xx, 2xx, 3xx, 4xx, 5xx.
  - Tener **al menos un par de ejemplos de cada** (ej: 200, 201, 301, 404, 500...).

No hace falta

- Postman, Bruno, Chrome DevTools para red → no los va a preguntar.

2. HTML

Tengo que saber

- HTML como **lenguaje de marcado**, no de programación.
- Qué es un **element** y cómo se estructura:
  - Etiqueta de apertura, contenido, etiqueta de cierre (o void element).
  - Estructura básica: `html`, `head`, `body`.
  - En el `head`: título, favicon, metadatos, etc.
- **Formularios** (súper importante):
  - Inputs y sus tipos (`text`, `password`, `email`, `number`, `checkbox`, `radio`, etc.).
  - Atributos importantes:
    - `action`
    - `method`
    - `name`
  - Cómo **enlazar `<label>` con `<input>`** → `for` y `id`.
  - Cuándo se hace GET y cuándo POST:
    - **Nunca GET con datos sensibles**.
- **Canvas**
  - Saber **para qué sirve** la etiqueta `<canvas>`.
  - No hace falta aprender funciones de dibujo de memoria.
- **HTML bien formado / validación**
  - Entender cuándo un HTML está bien o mal.
  - Void elements (`<img>`, etc.).
  - Rutas relativas vs absolutas.
  - Idea importante:
    - Si usamos un **validador**, no deberían salir **warnings** ni errores.

No hace falta

- Memorizar los métodos de canvas para dibujar formas.

3. CSS

Ideas clave

- Separar responsabilidades:
  - HTML → contenido
  - CSS → estilos

    - JS → lógica
- Poner CSS dentro del HTML o en estilos en línea es **mala práctica** (sería fallo en examen, aunque no cero).

Tengo que saber

- **Sintaxis básica**: `selector { propiedad: valor; }`
- **Selectores** (lo más importante de CSS):
    - **Simples**: por etiqueta, por clase (`.clase`), por id (`#id`).
    - **De combinación**: descendiente, hijo directo, etc.
    - Los muy complicados del final son mucho menos importantes.
- **Modelo de caja (Box model)**:
    - `margin`, `padding`, `border`, `width`, `height`.
- **Unidades**:
    - `px`, `%`, `em`, `rem`, etc. → lo básico.
- **Entender un CSS dado**:
    - Si te da código CSS, tienes que poder explicar qué hace.
- **Colocación de elementos**:
    - `position: static / relative / absolute / fixed / sticky`.
- **Specificity**:
    - Cómo se resuelven conflictos cuando varios estilos aplican.
    - Saber lo **básico** de qué selector tiene más prioridad.
- **Etiquetas semánticas > `div`**:
    - Evitar abusar de `div` para todo.
    - Usar etiquetas específicas: `header`, `nav`, `section`, `article`, `footer`, etc.
- **Responsive Web Design** (muy importante):
    - **Media queries**: idea de cómo se usan y por qué.

No hace falta

- Grid, flexbox (no lo enfatiza).
- Less, Sass → solo saber que existen.
- Frameworks tipo Bootstrap → **no entra**.

Nivel de examen

- Saber **escribir algo sencillo** y **entender código**. No te va a pedir una web superbonita ni nada loco.

4. JavaScript (la parte más importante del cliente)

Organización

- Mejor **separar JS del HTML** en archivos `.js`.

- Entender `defer` y `async` para la carga de scripts (bloque 1).

Base del lenguaje

- **`let` y `const`** vs `var`:
  - En la práctica: usar `let` y `const`, olvidarse de `var`.
- `NaN`, `undefined`, `null`:
  - Saber qué son y en qué situaciones aparecen.
- Operaciones básicas con strings y números.

Condiciones y operadores

- Diferencia entre == y === (**triple igualdad es importante**).
- Operadores lógicos:
  - `&&`, `||`, `!`
  - Y operadores tipo "si no está definido, uso esto otro" (operadores cortocircuito, `??`, etc.).
- `if`, `else`, `switch`.

Estructuras básicas

- Bucles: `for`, `while`, `for...of`, `for...in`.
- Funciones:
  - Declaración normal.
  - Funciones flecha (lambda).
  - Parámetros por defecto.

Arrays y objetos

- **Muy importante**:
  - **Cómo recorrer un array** (con índices, `for...of`, etc.).
  - **Cómo recorrer un objeto** (`for...in`, `Object.keys`, etc.).
  - NO recorrer un objeto como un array ni al revés (error típico de examen).

DOM y eventos

- Por qué usamos **`addEventListener`** y no `onclick=` en HTML o `element.on...`.
- Entender el **por qué** del API, no solo memorizar la sintaxis.

Canvas + JS

- Como antes:
  - Entender qué hace el código, pero no escribir todo desde cero.
  - **Importante**: `setInterval`, `setTimeout` y **`requestAnimationFrame`**.

Asíncrono: Ajax, Promesas, Fetch

- **jQuery NO es relevante**.
- **Strict mode** (`"use strict"`) sí.
- **Módulos** (import/export) conectan con Node.
- **Tampermonkey no es relevante**.
- **Ajax**:
  - Saber **qué es** y la idea general.
- **Promesas**:
  - Saber leer y escribir usos básicos (`.then`, `.catch`).
- **Fetch**:
  - **Muy importante** → saber trabajar con él para hacer peticiones.
- **async / await**:
  - Muy importante, va pegado a Fetch y Promesas.

5. Node.js

Conceptos

- Diferencia entre:
  - JavaScript en el navegador.
  - JavaScript en el servidor con Node.js.
- **Instalación**:
  - Dice que **va a caer** algo de esto.
  - Saber qué son:
    - `npm` (gestor de paquetes, muy importante).
    - REPL de Node (modo interactivo).
    - npx (menos importante).

Módulos y entorno

- Variables de entorno:
  - Qué son y para qué sirven.
- Argumentos de línea de comandos:
  - Módulo `minimist` no es importante, pero sí el concepto de argumentos.
- Módulos `process`, `os`:
  - Saber qué son, no aprender todas sus funciones.
  - Si hace falta, él te da la función en el enunciado.

JSON (SUPER IMPORTANTE)

- Qué es JSON y cómo se escribe bien.
- **7 tipos de valores válidos** en JSON.

- JSON que es solo `true` → **es válido** (es uno de los 7 valores posibles).
- Muy importante:
  - **El orden de las claves no importa** en un objeto JSON.
  - Si el orden importa → usar un **array**.
- Ejercicios de **corregir JSON** (tipo pág. 49): muy importantes.
- Cómo recorrer un **objeto JavaScript** (relacionado con JSON).

package.json / package-lock.json

- `package.json`:
  - Para qué sirve.
  - Estructura general.
  - Saber que en `scripts` se definen comandos tipo `npm test`, etc.
- `package-lock.json`:
  - Por qué es necesario: asegura que todos usan **las mismas versiones** de dependencias.

Event loop

- Entender el **concepto** de event loop (la diapositiva roja) → muy importante.

Otros módulos

- `path`, `fs`:
  - No memorizar todas las funciones.
  - Sí las principales que habéis usado en ejercicios.
- Peticiones HTTP en Node "a pelo":
  - Saber la idea general, pero **más importante en Express**.

6. Express

Más uso que teoría

- No quiere que recites definiciones, quiere que **sepas usarlo**.

Tienes que entender:

- Cómo se crea una app Express.
- Qué son `req`, `res` y **`next`** (súper importante).
- Qué es un **middleware** y para qué sirve:
  - Middleware de sesión.
  - Middleware de control de acceso (ej. `checkLogin`).
  - Middleware final de 404.
- **res.locals** y **app.locals**:
  - `app.locals` → datos globales a todas las vistas.

- `res.locals` → datos que pasas a una vista concreta con `res.render`.

7. Ejemplo LOGIN (la joya del examen)

Esto es lo **más importante** de la parte práctica:

Tienes que ser capaz de:

- Saber **qué hace cada fichero** (y cómo se interconectan):
  - Archivo `bin/www` (o equivalente) → **punto de entrada**.
  - `app.js` → configuración de Express:
    - Dónde se configuran las rutas.
    - Dónde se configura la carpeta de estáticos (`public`) → línea 25 en tu apunte.
    - Dónde se configuran las vistas (`views`) y el motor (EJS) → línea 18–19.
    - Dónde se configura la sesión.
    - Dónde se añaden los middlewares (incluido el de errores 404).
- Saber contestar cosas tipo:
  - "¿En qué línea se indica la carpeta de plantillas?"
  - "¿Dónde se configura qué archivos son accesibles desde el navegador?"
  - "¿Qué hace este middleware?"
  - "¿Qué pasa si llamo `next()` aquí?"
  - "¿Cómo harías un middleware `checkLogin` que mire `req.session.user`?".
- **Ejemplos típicos que te puede pedir**:
  - Añadir roles a los usuarios:
    - Tocar el modelo (añadir campo `role`).
    - Ajustar el registro/login para guardar el rol.
    - Crear un middleware que compruebe `req.session.user.role === 'admin'`.
  - Crear una vista de administrador que liste todos los usuarios:
    - Añadir ruta `/admin`.
    - Cargar usuarios desde la "base de datos" (`database.users.data`).
    - Pasarlos a la vista con `res.render('admin', { users: ... })`.
    - En la vista EJS, **recorrer el objeto** y pintar una tabla/lista.

8. Socket.io

- Entender la estructura básica:
  - En servidor: `io.on('connection', socket => {...})`.
  - **Rooms**:
    - `socket.join('room')` → meter usuario en una sala.
    - `io.to('room').emit('evento', datos)` → enviar a todos de esa sala.

- Diferencia:
  - `socket.emit` → servidor → ese socket.
  - `io.emit` o `io.to` → servidor → varios.
- Cliente y servidor:
  - `socket.on(...)` / `socket.emit(...)` en ambos lados, con los mismos nombres de evento.
- Entender el ejemplo tipo **Kahoot**:
  - Cómo se define la room.
  - Cómo se envían mensajes a todos los de una room.

9. Tipo de examen (cómo te van a preguntar)

Según todo lo que has apuntado:

- Te va a **dar código** (HTML/CSS/JS/Node/Express/Socket) y te va a pedir:
  - Explicar qué hace.
  - Modificar cosas (ej. añadir middleware, nueva ruta, nuevo rol, nueva vista).
  - Decir **qué fiches tocarías** para añadir una funcionalidad.
  - Encontrar fallos (un HTML mal formado, un JSON no válido, mala práctica de CSS/JS…).
- Menos “hazme todo desde cero” y más:
  - “Aquí tienes esto, **toca lo justo** para que haga X”.

⭐ **en lo que es MUY importante**

Introducción, web, frontend vs backend

- ❑ ⭐ Sé explicar qué es una **aplicación para Internet** (ventajas / desventajas).
- ❑ [ ] Distingo entre **sitio web**, **aplicación web** y **servicio web**.
- ❑ ⭐ Sé explicar la diferencia entre **frontend** y **backend** (tareas típicas de cada uno).
- ❑ [ ] Conozco ejemplos de trabajos/roles de frontend y backend.

HTTP

- ❑ ⭐ Sé que HTTP es un protocolo **stateless**.
- ❑ [ ] Sé lo que es aproximadamente el **User-Agent**.
- ❑ ⭐ Conozco las diferencias básicas entre:
  - HTTP/1.0
  - HTTP/1.1
  - HTTP/2
  - HTTP/3
- ❑ ⭐ Sé qué es una **URL** y qué es una **URI**.
- ❑ ⭐ Sé qué es una **petición HTTP** (línea de petición, cabeceras, cuerpo).
- ❑ ⭐ Conozco los **métodos HTTP** principales:
  - GET
  - POST
  - PUT
  - DELETE
  - Otros (HEAD, OPTIONS… aunque sea por encima).
- ❑ ⭐ Sé qué significa que un método sea **safe**.
- ❑ ⭐ Sé qué significa que un método sea **idempotente**.
- ❑ ⭐ Distingo correctamente entre **GET y POST** y sé:
  - Cuándo NO usar GET (datos sensibles → usar POST).
- ❑ ⭐ Conozco las **5 categorías de códigos de estado**: 1xx, 2xx, 3xx, 4xx, 5xx.
- ❑ ⭐ Sé al menos **2 ejemplos de códigos** por categoría (200, 201, 301, 404, 500, etc.).

HTML

- ❑ ⭐ Sé que HTML es un **lenguaje de marcado**, no de programación.
- ❑ [ ] Sé qué es un **elemento HTML** (etiqueta apertura, contenido, cierre / void).
- ❑ [ ] Conozco la **estructura básica** de un documento:
  - `<!DOCTYPE html>`
  - `<html>`, `<head>`, `<body>`
  - Qué va en el **head** (título, favicon, metadatos).
- ❑ ⭐ Entiendo cuándo un HTML está **bien formado** y cuándo no.

- ❑ [ ] Sé lo que es un **void element** (`<img>`, etc.).
- ❑ [ ] Entiendo la diferencia entre **rutas relativas y absolutas**.
- ❑ ⭐ Sé que, si uso un **validador**, el HTML correcto no debe tener **warnings** ni errores.

Formularios

- ❑ ⭐ Conozco los tipos de **`<input>`** más habituales (text, password, email, number, checkbox, radio...).
- ❑ ⭐ Sé para qué sirven los atributos:
  - `action`
  - `method`
  - `name`
- ❑ ⭐ Sé enlazar correctamente un **`<label>`** con su **`<input>`** (`for` / `id`).
- ❑ ⭐ Sé cuándo usar **GET** y cuándo **POST**, y que **nunca se usa GET con datos sensibles**.

Canvas

- ❑ [ ] Sé para qué sirve la etiqueta **`<canvas>`** (dibujar gráficos vía JS).
- ❑ [ ] Entiendo código sencillo que use canvas, aunque **no recuerde todas las funciones de memoria**.

CSS

- ❑ ⭐ Entiendo que se debe **separar**:
  - HTML → contenido
  - CSS → estilos
  - JS → comportamiento
- ❑ [ ] Sé que meter CSS dentro del HTML (inline o `<style>` gordo) es **mala práctica** (fallo en examen).

Sintaxis y selectores

- ❑ ⭐ Conozco la **sintaxis básica**: `selector { propiedad: valor; }`.
- ❑ ⭐ Domino los **selectores simples**:
  - Por etiqueta (`p`, `h1`...).
  - Por clase (`.clase`).
  - Por id (`#id`).
- ❑ ⭐ Conozco los **selectores de combinación** básicos:
  - Descendiente (`div p`)
  - Hijo (`div > p`)
  - Algún otro sencillo.
- ❑ [ ] Sé que los selectores muy raros del final del tema son **menos importantes**.

Box model y unidades

- ❑ ⭐ Entiendo el **modelo de caja**:
  - `margin`
  - `border`
  - `padding`
  - `width`, `height`
- ❑ [ ] Conozco las **unidades** básicas (`px`, `%`, `em`, `rem`, etc.).

Otros puntos

- ❑ ⭐ Si me dan un **CSS**, soy capaz de **entender qué hace**.
- ❑ [ ] Entiendo cómo se **colocan elementos** con:
  - `position: static`
  - `relative`
  - `absolute`
  - `fixed`
  - `sticky`
- ❑ ⭐ Entiendo la idea de **specificity**:
  - Puedo decidir qué regla gana si hay conflicto.
- ❑ [ ] Sé que no debo abusar de `div`; es mejor usar etiquetas **semánticas** (`header`, `nav`, `section`, `article`, `footer`).
- ❑ ⭐ Entiendo qué es el **Responsive Web Design** y cómo se usan **media queries**.

No prioritario

- ❑ [ ] Sé que Sass, Less, BEM existen, pero **no entran en detalle**.
- ❑ [ ] Sé que **Bootstrap** existe, pero **no entra en el examen**.

JavaScript (cliente)

Organización de scripts

- ❑ ⭐ Sé que es mejor **separar JS del HTML** en archivos `.js`.
- ❑ [ ] Entiendo qué hacen `defer` y `async` al cargar scripts.

Básicos del lenguaje

- ❑ ⭐ Entiendo la diferencia entre:
  - `var`
  - `let`
  - `const`

  y sé que en la práctica debo usar `let`/`const`.
- ❑ [ ] Sé lo que son:
  - `NaN`

- `undefined`
- `null`

- [ ] Puedo hacer operaciones básicas con strings y números.

Condiciones, igualdad y operadores

- [ ] ⭐ Entiendo la diferencia entre:
  - `==` y `===` (triple igualdad).
- [ ] Sé usar:
  - `if`, `else`, `switch`.
- [ ] Sé usar operadores lógicos:
  - `&&`, `||`, `!`
- [ ] Entiendo patrones tipo “si esto no está definido, uso este otro valor”.

Estructuras

- [ ] Sé usar bucles:
  - `for`
  - `while`
  - `for...of`
  - `for...in`
- [ ] Sé declarar **funciones** normales y **funciones flecha**.
- [ ] Entiendo los **parámetros por defecto**.

Arrays y objetos

- [ ] ⭐ Sé **recorrer correctamente un array** (índices, `for...of`, etc.).
- [ ] ⭐ Sé **recorrer correctamente un objeto** (`for...in`, `Object.keys`, etc.).
- [ ] ⭐ No confundo array y objeto cuando programo (error típico de examen).

DOM y eventos

- [ ] ⭐ Entiendo por qué usamos **`addEventListener`** en vez de `onclick` o cosas en línea.
- [ ] Sé manejar eventos básicos (click, submit, etc.).

Canvas + timers

- [ ] Entiendo a grandes rasgos cómo JS dibuja en **canvas**.
- [ ] ⭐ Conozco:
  - `setTimeout`
  - `setInterval`
  - `requestAnimationFrame`

  y para qué se usan.

Asíncrono: Ajax, Fetch, Promesas

- ❑ [ ] Sé **qué es Ajax** y que sirve para peticiones asíncronas (sin refrescar página).
- ❑ ⭐ Entiendo **qué son Promesas** y sé leer código con `.then` / `.catch`.
- ❑ ⭐ Sé usar **`fetch`** para hacer peticiones HTTP desde JS.
- ❑ ⭐ Sé usar **`async` / `await`** con `fetch`.

Otros

- ❑ [ ] Sé qué es **"use strict"** (modo estricto).
- ❑ [ ] Sé que existen los **módulos ES** (import/export) y que conectan con Node.
- ❑ [ ] Sé que **jQuery** y **Tampermonkey** NO son relevantes para el examen.

Accesibilidad y talleres

- ❑ [ ] He leído el tema de **accesibilidad** (solo teórico).
- ❑ [ ] Recuerdo los conceptos principales vistos en los **talleres** (especialmente los primeros):
  - Qué se hacía.
  - Qué problema resolvían.
  - Qué tecnologías/conceptos se aplicaban.

Node.js (conceptos básicos)

- ❑ ⭐ Sé la diferencia entre:
  - JS en el navegador.
  - JS en el servidor con Node.js.
- ❑ ⭐ Sé qué es **`npm`** y para qué sirve.
- ❑ [ ] Sé qué es el **REPL de Node**.
- ❑ [ ] Sé qué es npx (aunque es menos importante).
- ❑ ⭐ Sé que la **instalación y puesta en marcha de Node** puede caer en el examen.
- ❑ [ ] Entiendo el **ejemplo 0** de Node (el “hola mundo” simple).

Entorno y módulos

- ❑ [ ] Sé qué es una **variable de entorno** y por qué se usa.
- ❑ [ ] Sé que se pueden pasar **argumentos** a un script Node.
- ❑ [ ] Sé qué es el objeto global **`process`** (información del proceso).
- ❑ [ ] Sé qué hace, a grandes rasgos, el módulo **`os`**.
- ❑ [ ] Sé qué hace, a grandes rasgos, el módulo **`path`**.
- ❑ [ ] Sé qué hace, a grandes rasgos, el módulo **`fs`** (sistema de ficheros).

JSON y package.json:

JSON

- ❑ ⭐ Sé la sintaxis de **JSON** y cuándo es válido.
- ❑ ⭐ Conozco los **7 tipos de valores válidos** en JSON.
- ❑ ⭐ Sé que un JSON que solo sea `true` es **válido**.
- ❑ ⭐ Sé que el **orden de las claves** en un objeto JSON **no importa**.
- ❑ ⭐ Sé que si el orden importa, hay que usar un **array**.
- ❑ ⭐ Soy capaz de **corregir un JSON mal escrito** (ejercicio tipo pág. 49).
- ❑ ⭐ Sé cómo **recorrer un objeto JS** derivado de JSON.

package.json y package-lock.json

- ❑ ⭐ Sé para qué sirve **`package.json`**:
  - Describir el proyecto.
  - Declarar dependencias.
  - Definir scripts.
- ❑ ⭐ Sé para qué sirve **`package-lock.json`**:
  - Fijar versiones de dependencias para todo el equipo.
- ❑ ⭐ Soy capaz de leer un **`package.json`** y localizar:
  - `name`, `version`, `dependencies`, `scripts`, etc.

Event loop y módulos Node

- ❑ ⭐ Entiendo el concepto de **event loop** de Node (la famosa diapositiva roja).
- ❑ [ ] Entiendo que las operaciones bloqueantes pueden parar el event loop.
- ❑ [ ] Reconozco cuándo algo es síncrono/asíncrono en Node.

Express

- ❑ ⭐ Sé qué es Express (framework HTTP para Node).
- ❑ ⭐ Sé crear un servidor básico con Express.
- ❑ ⭐ Entiendo qué son:
  - `req` (objeto petición).
  - `res` (objeto respuesta).
  - `next` (función para pasar al siguiente middleware).
- ❑ ⭐ Sé qué es un **middleware** y para qué se usa:
  - Para loggear.
  - Para comprobar sesión / permisos.
  - Para manejar errores / 404, etc.
- ❑ ⭐ Entiendo la diferencia entre:
  - `app.locals` (datos globales).
  - `res.locals` (datos de una respuesta concreta).

- ❑ [ ] Sé definir rutas:
  - `app.get('/ruta', ...)`
  - `app.post('/ruta', ...)`
  - Separar rutas en routers (`router`).

Proyecto LOGIN (súper importante)

- ❑ ⭐ Sé cuál es el **punto de entrada** del proyecto (p. ej. `bin/www`).
- ❑ ⭐ Sé qué hace `app.js`:
  - Configura Express.
  - Configura rutas.
  - Configura los estáticos (`public`).
  - Configura las vistas (`views`) y el motor EJS.
  - Configura la sesión.
  - Añade middlewares.
- ❑ ⭐ Sé localizar en el código:
  - Dónde se indica que los ficheros estáticos están en `public`.
  - Dónde se indica que las plantillas están en `views`.
  - Dónde se indica que se usa el motor de plantillas EJS.
- ❑ ⭐ Entiendo el flujo de:
  - Petición → router → controlador → vista.
- ❑ ⭐ Soy capaz de:
  - Añadir una **nueva ruta** con GET y POST.
  - Añadir un **middleware** tipo `checkLogin`.
  - Usar `req.session` para comprobar si un usuario está logueado.
- ❑ ⭐ Sé cómo **pasar datos a una vista EJS** (`res.render('vista', { datos })`).
- ❑ ⭐ Sé **recorrer en la vista** (EJS) una lista u objeto para generar una tabla/lista (ej. panel de admin con usuarios).

Socket.io

- ❑ ⭐ Sé que Socket.io sirve para **comunicación en tiempo real** (bidireccional).
- ❑ ⭐ Entiendo el patrón:
  - En servidor: `io.on('connection', socket => { ... })`.
  - En cliente: conectar y escuchar eventos con `socket.on`.
- ❑ [ ] Sé qué es `socket.emit` (enviar evento a un cliente).
- ❑ [ ] Sé qué es `io.emit` (enviar a todos).
- ❑ ⭐ Entiendo qué es una **room**:
  - `socket.join('room')`.
  - `io.to('room').emit('evento', datos)`.
- ❑ [ ] Entiendo la estructura del ejemplo tipo **Kahoot**:
  - Uso de rooms.
  - Eventos entre cliente y servidor.