Bundlers

Sistemas y Tecnologías Web (CC3062) - 2026

Bundlers

Semestre 2, 2026

El problema del frontend moderno

Realidad del desarrollo actual

- Los proyectos usan decenas o cientos de archivos.
- Se usan dependencias externas (node_modules).
- El código moderno no siempre corre directamente en el navegador.
- Múltiples formatos: TypeScript, JSX, SCSS, etc.

Sin bundlers

- El navegador no entiende TypeScript ni JSX.
- Cargar 100 archivos JS por separado es lento.
- Las dependencias son difíciles de gestionar manualmente.
- El código no está optimizado para producción.

El flujo que queremos

```
Código fuente (muchos archivos, TS, JSX...)
          ↓
      Bundler
          ↓
Bundle optimizado (pocos archivos, JS puro)
```

Bundler

Definición

- Herramienta que procesa el código fuente de un proyecto.
- Une múltiples archivos en uno o pocos archivos.
- Resuelve dependencias automáticamente.
- Optimiza el resultado para producción.

Idea clave

Un bundler convierte muchos módulos en pocos archivos optimizados listos para el navegador.

Proceso

- Analiza el grafo de dependencias (`import` , `require`).
- Transforma código (TypeScript → JS, SCSS → CSS).
- Elimina código muerto (*tree-shaking*).
- Minimiza y comprime el resultado.

Bundler

ECMAScript Modules (ESM)

Definición

- Sistema estándar de módulos en JavaScript.
- Permite dividir el código en archivos reutilizables.
- Usa `import` y `export`.
- Soportado nativamente por los navegadores modernos.

Ejemplo ESM

```
// math.js
export const suma = (a, b) => a + b
export const resta = (a, b) => a - b

// app.js
import { suma } from './math.js'

console.log(suma(2, 3)) // 5
```

Funcionamiento

- El navegador descarga cada módulo por separado.
- Resuelve dependencias de forma recursiva.
- Ejecuta el código en el orden correcto.

Características

- Carga bajo demanda.
- Scope propio por archivo (no contamina el global).
- Módulos aislados y reutilizables.

El problema de ESM puro en producción

- Muchas peticiones HTTP si hay muchos archivos.
- Las dependencias de `node_modules` no están optimizadas.
- Algunos entornos no soportan ESM directamente.

Por eso existen los bundlers

- Agrupan los módulos en pocos archivos.
- Optimizan las dependencias.
- Garantizan compatibilidad con más entornos.

Herramientas populares

Las más usadas hoy

- **Webpack**: el más veterano, muy configurable.
- **Parcel**: zero-config, fácil de arrancar.
- **Vite**: moderno, rápido, orientado a ESM.

Comparación general

| Herramienta | Enfoque | Velocidad en dev | | ---------- | ---------- | ---------- | | Webpack | Configurable | Lenta | | Parcel | Zero-config | Media | | Vite | ESM moderno | Muy rápida |

Webpack

- El más usado durante años.
- Ecosistema enorme de plugins y loaders.
- Muy flexible pero con configuración compleja.
- Arranque lento en proyectos grandes.

Parcel

- Detecta automáticamente el tipo de archivo.
- No requiere configuración inicial.
- Bueno para proyectos pequeños o prototipos.
- Menos adoptado en equipos grandes que Webpack o Vite.

Vite

Definición

- Herramienta moderna de desarrollo frontend.
- Incluye servidor de desarrollo y build tool.
- Diseñada para velocidad desde el principio.
- Creada por Evan You (Vue).

El problema que resuelve

- Webpack bundlea todo antes de servir → arranque lento.
- En proyectos grandes, esperar segundos por cada cambio.
- Feedback lento frena la productividad.

La solución de Vite

- En desarrollo: no bundlea. Sirve los módulos directamente usando ESM.
- El navegador pide los archivos, Vite los transforma bajo demanda.
- Solo procesa lo que se necesita en el momento.

Flujo en desarrollo

```
Navegador solicita /src/app.ts
        ↓
Vite transforma el archivo (TS → JS)
        ↓
Entrega el módulo transformado
        ↓
Navegador solicita las dependencias del módulo
        ↓
Vite transforma cada una bajo demanda
```

Pre-bundling de dependencias

- Las dependencias de `node_modules` se procesan una sola vez.
- Usa esbuild para esto (escrito en Go, súper rápido).
- El resultado se guarda en caché.
- Reduce el número de peticiones HTTP.

Module Graph

- Vite mantiene un grafo de qué módulo depende de cuál.
- Cuando cambia un archivo, sabe exactamente qué actualizar.
- Base del sistema de Hot Module Replacement.

Hot Module Replacement (HMR)

Definición

- Actualiza solo el módulo que cambió.
- No recarga toda la página.
- El estado de la aplicación se conserva.

Sin HMR vs con HMR

| Sin HMR | Con HMR | | ---------- | ---------- | | Recarga la página | Actualiza el módulo | | Pierde el estado | Conserva el estado | | Tarda segundos | Casi instantáneo |

Resultado

- Feedback inmediato al guardar un archivo.
- Ciclo de desarrollo mucho más ágil.

Vite en producción

Cambios

- En producción, Vite sí bundlea el código.
- El objetivo es optimizar para el usuario final.
- Usa Rollup como bundler de producción.

Rollup

- Bundler moderno basado en ESM.
- Especializado en producir bundles pequeños y eficientes.
- Aplica tree-shaking agresivo.

Tree-shaking

- Elimina código que nunca se usa.
- Analiza los `import` y `export` estáticamente.
- Resultado: bundles más pequeños.

```
// utils.js
export const usado = () => "necesario"
export const noUsado = () => "nunca importado"

// app.js
import { usado } from './utils.js'
// noUsado() no llega al bundle final
```

Code splitting

- Divide el bundle en partes más pequeñas.
- Cada parte se carga solo cuando se necesita.
- Mejora el tiempo de carga inicial.

Rolldown

- Nueva implementación inspirada en Rollup.
- Escrita en Rust para mayor performance.
- Vite está migrando hacia Rolldown progresivamente.

| Herramienta | Lenguaje | Estado | | ---------- | -------- | ---------- | | Rollup | JS | Estable, actual | | Rolldown | Rust | En desarrollo |

Velocidad de Vite

Las razones

- esbuild para pre-bundling (Go es muy rápido).
- No bundlea en desarrollo: sirve ESM directo.
- Caché inteligente de dependencias.

- HMR basado en el Module Graph.

Crear un proyecto con Vite

Comando

```
npm create vite@latest
```

Opciones que ofrece

- Nombre del proyecto.
- Framework: Vanilla, Vue, React, Svelte, etc.
- Variante: JS o TypeScript.

Estructura generada

```
mi-proyecto/
├── index.html          ← punto de entrada
├── package.json
├── vite.config.js
└── src/
    ├── main.js
    └── style.css
```

Ejecutar en desarrollo

```
npm install
npm run dev
```

Generar build de producción

```
npm run build
```

Genera la carpeta `dist/` con los archivos optimizados listos para desplegar.