

Multiplatform szoftverfejlesztés

Teljesítmény, csomagolás és formátumok

Teljesítmény

Fogalmak és mérőszámok

Teljesítmény

- A szoftver
 - Letöltési/betöltési ideje
 - Parse-olás/JIT fordítási ideje
 - Futási ideje
 - Mérete
 - Ez erősen összefügg mindegyikkel
- Főleg a kiadott csomagnál fontos
 - De fejlesztés közben is szempont

Mérőszámok

- Első rajzolás (FP – first paint)
 - Amikor bármi változást látunk (pl. háttér)
- Első tartalom rajzolás (FCP – first contentful paint)
 - Amikor az első HTML-beli elemet látjuk
- DOMContentLoaded esemény
 - Minden globális függvény lefutott
 - A HTML betöltve, a DOM felépítve
- Load esemény
 - Minden betöltve (CSS is)

Mérőszámok

- Oldal használható (TTI – time to interactive)
 - Látható az oldal és reagál 50 ms alatt
 - Ez a legutolsó és legfontosabb mérőszám
- Sokat idézett kutatás (Google ad network mérése alapján): Ha az oldal 3 másodperc alatt nem töltődik be, akkor a felhasználók 53%-a elhagyja a oldalt
 - Ezt nehéz megoldani, ha nem készülünk rá

Optimalizáció

- Caching – volt
- Gyors/kicsi könyvtárak használata
- Csomagolás (bundling)
 - Tree shaking
- Lazy loading
- Média formátumok (pl. webp)
- Tömörítés
- HTTP/2, HTTP/3

Csomagolás

Fordítás

- Fordítás (compiling, transpiling)
 - Kód átalakítása JS-re
- Például
 - .ts – TypeScript
 - .vue – Vue SFC (Single File Component)
 - .tsx és .jsx (React, vagy HTML template-et tartalmazó fájl)
- Gyors művelet
 - Forrás és cél formátum nagyon hasonló
 - Nincs szükség strukturális átalakításra
 - De szintaktikai ellenőrzés kell

Source map: app.js.map

- A fordítás/átalakítás miatt szükséges eltárolni, hogy adott kódrészlet hol volt az eredeti kódban
 - Célja a breakpoint támogatás, illetve más debug eszközök (soronként léptetés, függvénybe lépés, átugrás)
- Többszörös átalakítást támogatni kell
 - Például TS => JS => bundle

Bundling

- Több fájl egymás után másolása
 - Majd modulokra bontása
- Célok
 - Egyesítés után kevesebb fájlt kell letölteni
 - Modulok kialakítása, hogy ne egyben töltődjön le a teljes csomag
 - Általában: a kiadott csomag függetlenné tétele a fejlesztés alatt kialakított struktúráról
- Modul rendszert át kell alakítani
 - Vagy egy kimeneti fájl esetén meg kell szüntetni

Tree Shaking

- Felesleges kód eltávolítása – DCE (Dead Code Elimination)
 - Meg kell találni a nem használt függvényeket
 - Problémás osztályok esetén
 - Általában is nehéz, ha nem erre van optimalizálva egy könyvtár
 - Nem talál meg mindent
- Célok
 - A saját kódunk kisebbé tétele
 - A használt könyvtárak kisebbé tétele
 - Vagy akár teljes kiszűrése

Tree Shaking

- Sajnos a cél elérése nem garantált
 - Saját kódunkban általában kevés nem használt függvény van
 - Külső könyvtárak esetén lehet hasznos, ha úgy vannak megírva
 - Számos könyvtár jön testre szabható (customize) formában – én választom ki, hogy mi kell belőle
- Példa: Pixi.js (2D render motor)
 - 370K a min.js verzió (1.2M az eredeti verzió)
 - Tree shaking szinte semmit sem tud kivágni
 - Az egyes modulok módosítják a belsejét (pl. Batch render)
 - 180K, ha kézzel válogatom be, ami kell

Minify: app.min.js

- Nevek lecserélése rövidre
- Célok
 - Kisebb fájl méret
 - Jelentős méretcsökkenés (1:3, 1:4)
 - Tömörítve nem annyira nagy a hatás, de még úgy is jelentős
 - Gyorsabb
 - Letöltés – cache esetén kicsi a hatása, de nem nulla
 - Betöltés – itt is számít a méret
 - Nehezebb olvasni (mellékhatás) – nehezíti a visszafejtést/megértést
 - Ha ez nem cél, akkor adunk mellé egy „fejlesztői” verziót, amiben jók a nevek

Csomagolás CSS – Minify

- CSS méretének csökkentése a felesleges whitespace-ek kivételével és akár átnevezéssel
- Célok – azonos a JS minifierrel
- Az átnevezés nem triviális, mert a HTML-t és JS-t is módosítani kell hozzá
 - Vannak eszközök rá
- Nehéz debuggolni
 - Nincs source map
- Kicsi a hatása, mert tömörítés után alig lesz kisebb és a CSS fájlok eleve kisebbek

Lazy loading

Modulok

- A legjobb módszer csomagokra bontani a kódot
 - Ez nem triviális feladat
 - Értelmes csomagokat kell kapjunk
 - Például hiába van szétszedve, ha a fő oldalhoz mindegyik kell
- Import és export használata
- Modul betöltő automatikusan megoldja
 - Fordító paramétere, hogy melyiket használjuk
 - Lehet használni a natív modulkezelőt
 - Ez már széleskörben támogatott

Modulok – kimeneti formátumok

- CommonJS – szerver oldal Node.js
 - AMD – require.js
 - UMD – mindkettő + globális változó
 - A fájl úgy kezdődik, hogy megnézi, hogy van-e require.js, és így megy tovább
 - Natív ES modul
- A kimeneti formátumon nem működik tree-shaking

Modulok – natív modulok

- ES6-tól van
 - A böngészőre bízunk a betöltést
 - Ez gyorsabb, mint a JS megoldások
 - Tree-shaking működik, ha további feldolgozásra van szükség
- Jelenleg csak Rollup tud ilyen kimenetet adni
- Hiába gyorsabb, több száz fájl esetén lassul
 - Nem publikálhatjuk az összes fájlt
 - Továbbra is kell csomagoló (Rollup)
 - Vagy nem használunk natív modulokat – ez a bevett módszer még

Külső könyvtárak

- Itt is működik a modul módszer
 - De lehetséges, hogy nincs így csomagolva
- Késleltetni lehet a betöltést
 - async és defer: aszinkron tölti a scriptet
 - Párhuzamosít, így javít a TTI-n
 - Manuálisan csak a gombnyomásra betölteni
 - Prediktív módszerek
 - Pl. elindítani a betöltést mouseover-re
 - Mobilon nem működik (touch)
 - Vagy amikor a gomb bejön a képernyőre
 - Ez jó mobilon

Képek és egyéb tartalom

- Ami nem látszik, azt lehet késleltetni
- Natív megoldás van, és már multiplatform (iOS 15.4-től)
 - `loading="lazy"`
- Fontos, hogy mit késleltetünk
 - Ha akkor döntjük el, amikor a JS már fut, az késő
 - Böngésző párhuzamosan tölt és futtat
 - Csak azt lehet késleltetni, amiről biztosan tudjuk, hogy nem kell a TTI-hez
 - Image carousel
 - Oldalon később jövő anyagok
 - ...

Media formátumok

webp, webm, ...

Mindenhol működő formátumok

- JPEG

- Veszteséges tömörítés, nincs átlátszóság

- PNG

- Veszteségmentes tömörítés, van átlátszóság

- SVG

- Vektorgrafikus ábra, vonal, spline, ...

- TTF, OTF, WOFF

- Font, vektorgrafikus, egyszínű, subpixel rendering támogatás
- Főleg kicsi képek esetén fontos

Újabb formátumok

- WOFF2

- Tudásban azonos, jobban tömörített (brotli)

- WebP

- Erős tömörítés, átlátszóság, veszteséges/mentes
 - JPEG-hez képest 30-70%-kal kisebb képek azonos minőség mellett (másfajta tömörítési hibákat ad)

- WebM/VP9 codec (iOS nincs)

- Videó

- AVIF (Edge nincs)

- Még jobb tömörítés, mint WebP

picture – összes böngésző támogatása

- picture támogatás nélkül is megy
 - img fog csak érvényesülni

```
<picture>
  <source type="image/webp"
    srcset="images/p-5-256.webp 256w, images/p-5-512.webp 512w,
images/p-5-1024.webp 1024w"
    sizes="320px">
  <source
    srcset="images/p-5-256.jpg 256w, images/p-5-512.jpg 512w,
images/p-5-1024.jpg 1024w"
    sizes="320px">
  
</picture>
```


Tömörítés

- XHR esetén deflate vagy brotli
- WebSocket esetén csak deflate
 - Vagy bináris és mi magunk írunk tömörítőt
 - Például wasm-brotli
- Média webp/webm/stb.
- JS/CSS minified
 - És tömörítve jön le (brotli/deflate)
- HTTP/2

HTTP/2 és HTTP/3

■ HTTP/2

- Push resource
- Header tömörítés
- Multiplexing – egy TCP csatorna több HTTP csomag (akár több kérés válasz előtt)

■ HTTP/3

- UDP alapú (UDP-QUIC-HTTP)
- TLS 1.3 beépítve
- Safari (macos és iOS) még nem támogatja általában

Betűtípusok

- Lehet saját betűtípust használni
 - Nem csak szöveg célból
 - Ikonokra is, pl. Font Awesome
- Formátumok
 - ttf, otf, woff (mindenhol), woff2
 - A különbség főleg méretben van, nem tudásban
 - Ezek mind spline-ból kirakott alakzatok, így tetszőleges méretben rajzolhatók
- Az ikonokat célszerű így tárolni a Subpixel rendering miatt
- Színes nem lehet, de adhatunk neki színt

Subpixel rendering

- Csak font rendereléskor támogatott, és csak asztali OS-eken

asztali OS

- Háromszorozza a vízszintes felbontást
 - Egy 100 dpi-s monitor 300 dpi-sként látszik
 - Függőleges felbontás marad, de arra nem vagyunk érzékenyek, főleg nem szövegnél
 - Vízszintesen is csak mozgatni lehet, nem lesz több pixel
- Nehéz megoldani, játékokban sajnos nincs
 - Canvas-on és bizonyos CSS animációk esetén kikapcsol

Subpixel rendering

- Ha eleve nagy a DPI, akkor nincs rá szükség
 - Például mobilok: 500+ DPI
 - 4K-s laptopok
- Attól is függ, hogy milyen távolról nézzük
 - PPD: Pixel per Degree
 - Mobil tipikusan 30 cm-re van
 - 600 DPI felett tökéletes kép
 - 300-600 DPI: egyesek kezdik látni
 - 300 DPI alatt mindenkinek feltűnik a probléma
 - Laptop 60 cm
 - PC 80-100 cm

Kérdések?