Multiplatform szoftverfejlesztés

Progressive Web Apps

Web alapú alkalmazás++

- Mint web alapú alkalmazás, plusz
  - Telepíthető
    - Indítható a bolt/web látogatása nélkül
    - Akár offline is működik – ha olyan
  - Képes elérni olyan OS szolgáltatásokat, amit weben nem lehet

Telepített kliens

- Electron
  - HTML+CSS+JS csomagolva
  - Node.js futtatja
  - Chrome motor renderel
  - API-t biztosít a fájlrendszer és egyéb OS szolgáltatások eléréséhez
  - Platform: Windows, Linux, macOS
- PWA – ezzel foglalkozunk csak
  - Telepíthető a webalkalmazás
  - Ablakban indul, böngésző UI nincs ott

PWA

PWA

- Web alapú alkalmazás
- App szerű (ez főleg UX kérdés)
  - Például nem görgethető az egész
- Telepíthető
  - Offline is működik
- Responsive – minden méretben jól működik
- OS integráció
  - Megosztás támogatás (forrás és cél)
  - Push notification
  - …

PWA – telepíthető

- Minimum követelmény
  - HTTPS
    - Ez nem gond, amúgy is így kommunikálunk
  - Manifest fájl
    - Az alkalmazás paramétereit írja le
  - Service Worker
    - Offline működést oldja meg
- Opcionális: prompt esemény
  - beforeinstallprompt
  - Ha nem kezeljük, akkor a felhasználó automatikusan kapja a telepíthető felszólítást (iOS-en nem)

HTML

- HTML head

```
<meta name="theme-color" content="white" />
<meta name="apple-mobile-web-app-title" content="My Chat" />
<meta name="apple-mobile-web-app-capable" content="yes" />
<meta name="apple-mobile-web-app-status-bar-style" content="white" />
<link rel="manifest" href="manifest.json" />
<link rel="apple-touch-icon" href="Images/Logo180w.png" />
```

- manifest kötelező
- A többi csak iOS miatt kell

manifest.json

```
{
  "short_name": "My Chat",
  "name": "My Chat – Group Chat",
  "start_url": "index.html?utm_source=homescreen",
  "display": "standalone",
  "theme_color": "white",
  "background_color": "white",
  "icons": [
    {
      "src": "Images/Logo48.png",
      "type": "image/png",
      "sizes": "48x48"
    },…
  ]
}
```

manifest.json beállításai

- Telepítés után érvényesek csak
  - name: ez lesz az alkalmazás neve
  - start_url: ezt indítja el az OS
  - display: böngésző ablakban, vagy anélkül
  - theme_color: a böngésző ablakát átszínezi, ha van
  - background_color: betöltés közbeni szín
  - icons: a telepített ikon
    - Több méretben kell, mert az OS azt használja mindig ami a legjobban passzol
    - Például Windows taskbaron vs start menüben
  - share_target: megjelenik a share listában

Böngészőnként eltér

- Sajnos a manifest.json támogatása böngészőnként eltér
  - Valahol kitesz splash screen-t – itt kell nagy kép
  - Felhasználja a színeket, vagy nem
  - Stb.
- Szerencsére az unió működik
  - Mindent beállítunk minden módon – ez megy
- Bizonyos funkcionalitások nem mindenhol támogatottak
  - iOS lemaradásban
  - Címsor desktop OS-eket testre szabható

Service Worker

Service Worker

- Egy külső JS fájl
  - Nincs benne a csomagban
  - Mi írjuk meg
- Az alkalmazásunktól függetlenül fut
  - Akkor is futhat
    - Ha nem futtatjuk az alkalmazást
    - Ha a böngésző nincs elindítva
  - Háttér szolgáltatás (OS service) futtatja
- Feladata
  - Offline működés – cache
  - Push Notification kezelés

Web Worker

- Web Worker != Service Worker
  - Többszálúságot biztosít
  - Minden szál (worker) az adott alkalmazásban fut
    - Mint normális többszálú programokban
    - Életciklusuk azonos az appéval
  - Felülethez nem férhetnek hozzá
    - Az továbbra is csak a fő szálból érhető el
  - Üzenetekkel kommunikál (mint SW)
  - Elér pár API-t, amit SW nem
    - Például indexedDB, WebSocket

Service Worker

- Telepítése

```
navigator.serviceWorker.register( 'sw.js' ).then(
  reg => …,
  err => console.log('ServiceWorker registration failed') );
```

- sw.js az SW kódja
- Sikeres register hívás esetén elkezd működni
  - Tudunk push notificationt fogadni
  - Cache elindul
- Csak a következő betöltés megy teljesen cache-ből

Service Worker

- fetch esemény
  - Ha nincs benne a cache-ben, akkor töltse le

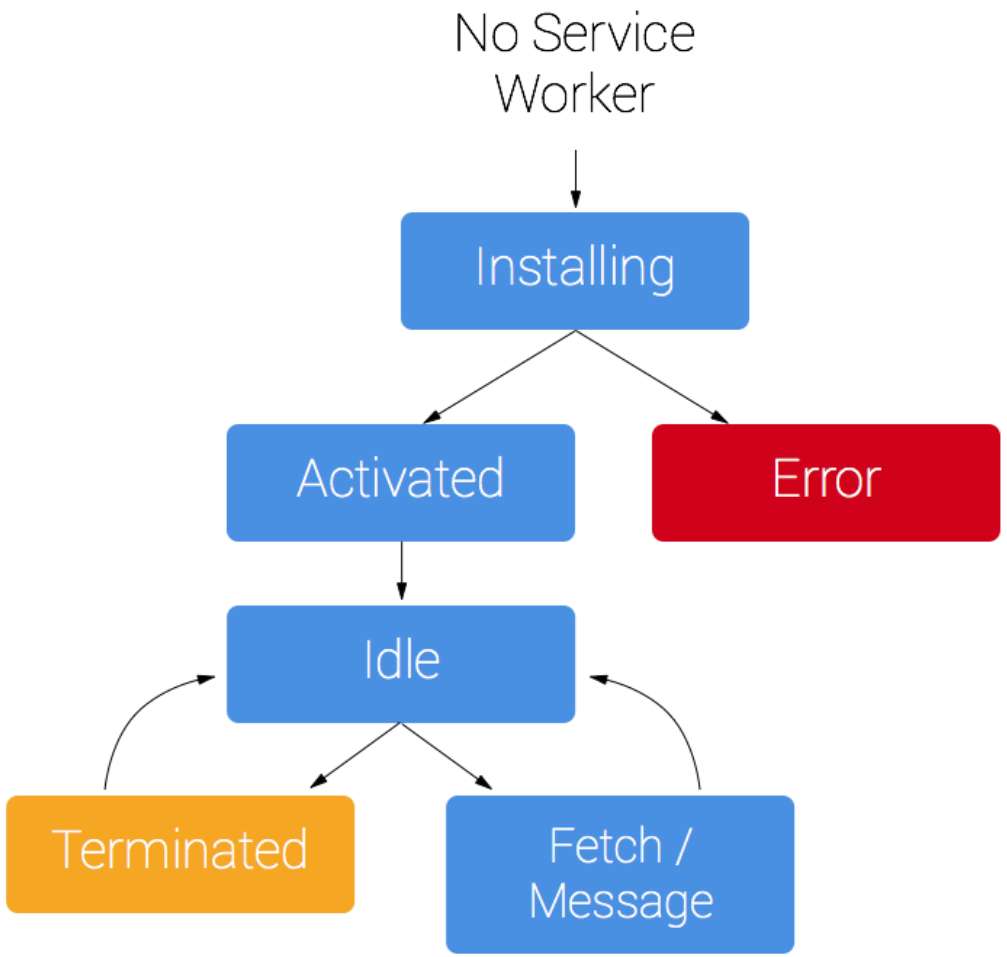
```
self.addEventListener('fetch', function (event) {
  event.respondWith(
    caches.match(event.request)
      .then(function (response) {
          if (response)
            return response;
          return fetch( event.request );
        })
      );
});
```

Service Worker életciklus

- Amikor először elindul az alkalmazásunk
  - Az eredeti címről letöltve
  - Még nincs telepítve SW
  - Böngésző elkezdi letölteni és telepíteni
    - El is indítja, amiről kapunk eseményt
      - Az SW-ben is (install esemény)
      - Az alkalmazásban is (registered promise)
- Ez a példány nem fog SW-t használni
- A következő indításkor viszont már igen

Service Worker életciklus

- A teljes életciklus így néz ki
  - Egy verzióra
- Amikor frissítjük
  - A régi verzió fut, amíg be nem záródik minden példánya az alkalmazásunknak
- Cache-re visszatérünk



Service Worker események

- Életciklus
  - install: telepítéskor és új verziónál
  - activate: működik, régi cache törölhető
- Cache
  - fetch: az oldal le akar tölteni valamit
- Kommunikáció
  - message: az alkalmazás küldött üzenetet
    - Ez az egyetlen mód a kommunikációra

Service Worker események

- Push notification
  - push: push notification jött
  - notificationclick: a felhasználó rányomott a notification-re
  - notificationclose: a felhasználó bezárta az üzenetet
  - pushsubscriptionchange: megszűnik a push notification feliratkozás
    - Bizonyos körülmények között újra feliratkozhatunk
- Egyelőre óvatosan kell használni
  - Mert iOS csak 16.4-től tudja

Cache

Alkalmazás oldal – Helyi tár

- Tárolhatunk adatot helyben
  - localStorage
  - IndexedDB
  - 10MB+ (akár GB feletti méretű is lehet)
- Nem feltétlen elég nagy képek és videók tárolására
  - De lehet, hogy a kezdő oldal tartalma belefér

Cache API

- Tipikusan SW-ben használjuk
- Minden függvény Promise alapú
- caches globális objektum
  - open függvénye megnyitja a megadott cache-t
  - Több cache is létezhet egyszerre
- Cache.addAll mindent letölt

```
self.addEventListener('install', function (event) {
  event.waitUntil(caches.open(CACHE_NAME)
    .then(function (cache) {
      return cache.addAll(urlsToCache);
    }));
});
```

Cache API

- Régi cache-t törölni kell, ha aktiválódott az új

```
self.addEventListener('activate', function (event) {
  event.waitUntil(
    caches.keys()
      .then(function (cacheNames) {
          return Promise.all(cacheNames
            .filter(function (cacheName) {
              return cacheName !== CACHE_NAME;})
            .map(function (cacheName) {
              return caches.delete(cacheName); }));
        })
    );
});
```

Cache stratégiák

- Nincsen mindig jó megoldás
- Minden letöltendő fájlról meg kell mondani
  - Benne legyen-e a cache-ben
  - Frissítsük-e
    - Ha igen, mikor
- A build tool összeállíthat egy fájllistát
- Dinamikusan előálló tartalom gond lehet
- Van pár tervezési minta – cache stratégiák
  - Lekérdezés típusonként kiválasztjuk a megfelelőt

Mindent cache-be

- Egyszerű, gyors

```
self.addEventListener( 'fetch', function ( event ){
  event.respondWith( caches.match( event.request ) );
} );
```

- Csak offline alkalmazásnál jó
- Minden fájlnak benne kell lennie
  - Dinamikus fájlok nem lehetnek
- Kezdő letöltés lassú, minden fájl kell
- Ha egy fájl nincs meg, akkor 404

Mindent hálózatról

- Egyszerű, lassú

```
self.addEventListener( 'fetch', function ( event ){
  event.respondWith( fetch( event.request ) );
} );
```

- Azonos működése van, mintha nem lenne
- Lassabb, mintha nem írtunk volna semmit
  - Át kell menni a hívásnak az SW-n
- Valóságban ezt sosem használjuk

Cache, majd hálózat

- Ha megvan, akkor nem töltjük le
- Az előző kettőnél több esetben használható
- Cache itt nem dinamikus
  - Ha valami nincs benne, akkor azt letöltjük, de nem adjuk hozzá
  - Ez bizonyos fájlok/kérések esetén fontos
    - Például állandóan változó hírlista

Hálózat, majd cache

- Ha nincs net, akkor a tárolt változatot adjuk vissza
- Cél
  - Offline is kéne működni
  - De ha van friss adat, akkor azt mutassuk
- Dinamikus cache
  - A cache-t frissíthetjük a letöltött adattal
- Hátrány: lassú hálózat belassít mindent

Hálózat és cache

- Elindítjuk mindkét lekérést
  - Cache előbb visszatér, azt visszaadjuk
  - Majd hálózat visszatér
    - Frissítjük cache-t
    - Értesítjük az appot, hogy van új adat
- Komplex megoldás
  - Az appnak is együtt kell működnie
- Egyszerűsítési lehetőség
  - Nem értesítjük az appot
  - Legközelebb már friss adatot adunk vissza

Kommunikáció

XHR

- Régi módszer
  - Az új verzió a fetch
- Http kérést indít egy szerver felé
- Aszinkron módon kap választ
- A kérés HTTP headerjeit részben írhatjuk
- A body-t teljesen egészében mi írjuk
  - Lehet JSON, XML, bináris, …
- A válasz headerjeit részben olvashatjuk
- A válasz body-t teljesen olvashatjuk

XHR

- Tömörítés
  - Támogatott: gzip, deflate, brotli (20%-kal jobb, mint gzip)
  - Ez a Content-Type headerben van jelezve és csak akkor jön, ha az Accept-Encoding headerben kérte a böngésző

```
let xhr = new XMLHttpRequest();
xhr.addEventListener( "load", () =>
  console.log( xhr.response ) );
xhr.open( "GET", "http://www.example.org/example.txt" );
xhr.send();
```

fetch

- XHR helyett – de nem tud mindent
  - Általában jó
- Promise-t ad vissza (lehet await-elni)

```
fetch( 'http://example.com/movies.json' )
    .then( ( response ) =>
    {
        return response.json();
    } )
    .then( ( data ) =>
    {
        console.log( data );
    } );
```

```
let res = await fetch( 'http://example.com/movies.json' );
let obj = await res.json();
```

WebSocket

- Üzenet alapú keretező protokoll
- Kétirányú csatorna jön létre
  - Nem szűnik meg csomagonként
  - Titkosítás azonos, mint HTTP esetén
- Kliens is és szerver is küldhet csomagot
  - Alacsony késleltetés
- Kicsi az overhead: max 8 bájt csomagonként
- Ezt használja számos klienst értesítő keretrendszer (pl. SignalR)

WebSocket

- Tömörítés van, de csak deflate (nincs brotli)
- Cache nincs
- A kapcsolódás egy HTTP kéréssel indul
  - A szerver visszaküldi, hogy OK, és nem zárja be a kapcsolatot
    - Az eredeti TCP kapcsolaton keresztül megy minden további kommunikáció
- Nem csak böngészőkben van implementálva
  - Két szerver is kommunikálhat WebSockettel
  - De botok nem támogatják

```
let ws = new WebSocket( "ws://echo.websocket.org/" );
ws.onmessage = e => console.log( e.data );
ws.onopen = () => this.webSocket.send( "Hello, World!" );
```

Share API

- navigator.share meghívja az OS beépített megosztás kezelőjét
  - Meg lehet adni: url, title, text
  - Csak mobilokon csinál valamit
  - Asztali OS-en copy-paste a megszokott mód
    - Esetleg valamilyen popupban listázhatjuk a szokásos dolgokat
- Feltehetjük az appunkat a megosztási listára
  - Az OS beépített megosztás kezelőjében ott lesz
  - manifest fájlba kell megadni, hogy mi történjen
  - El lehet kapni a SW-ben

Share API

- https://twitter.com/manifest.json részlet

```
"share_target": {
  "action": "compose/tweet",
  "enctype": "multipart/form-data",
  "method": "POST",
  "params": {
    "title": "title",
    "text": "text",
    "url": "url",
    "files": [
      {
        "name": "externalMedia",
        "accept": ["image/jpeg", "image/png", "image/gif", "video/quicktime", "video/mp4"]
      }
    ]
  }
}
```

Játékok

Piaci részesedés

- A legtöbb alkalmazás játék kategóriában van a piactereken
  - Mobilra igaz csak
  - Kevés a productivity alkalmazás
- Bevétel nagy része játék kategóriában van
  - Bizonyos piactereken 75% feletti a játék bevétel
  - PC-n árnyaltabb a helyzet
    - Itt is az egyik legnagyobb a játék költés
  - (Konzolon szinte csak az van)

Web alapú játékok

- Web alapú technológia elsősorban nem játékokra volt tervezve
  - HTML5 nagy előre lépés volt
  - Canvas és WebGL megjelent
- JavaScript sebesség problémák
  - Sokat javult a helyzet – közel C# sebesség
  - Köszönhető az optimalizáló fordítónak
  - Közel sem tartunk C++ sebességnél
- Eleinte egyszerűbb, ma már közepes grafikájú játékok is elfutnak

Web alapú játékok

- Megjelentek játékmotorok webes célplatformmal
  - Unity 3D
  - Unreal
  - WebGL-t és WebAssembly-t használnak
- Megjelentek 3D motorok JS-ben
  - Three.js
  - Babylon.js
- Nem csak grafika
  - Fizikai motorok, AI, video és audio támogatás

Canvas és 3D API

- <canvas> elem, amire renderelni lehet
- Működési módjai
  - 2d – lassabb, mint WebGL, de még így is sokkal gyorsabb, mint HTML elemekkel operálni
  - webgl – OpenGL ES 2.0 mód
  - webgl2 – OpenGL ES 3.0 mód
- navigator.gpu – WebGPU (Chrome/Edge 113+, Opera 99+)
  - Alacsony szintű 3D API
    - Mint Vulcan, vagy DirectX 12
  - Nincs köze OpenGL ES-hez

2d

- getContext-ben 2d-t adunk meg
- 2D rajzoló parancsok
  - rect, ellipse, drawImage, …

```
let context = canvas.getContext( "2d" );
context.rect( 0, 0, 100, 100 );
```

- Mindig az aktuális kitöltéssel és tollal rajzol
  - fillStyle és strokeStyle színekkel
- Nem törli automatikusan, nekünk kell
- A transzformációkat is megőrzi
  - Kényelmes

2d

- Általában képeket rajzolunk
  - Gond nélkül lehet több száz elemet rajzolni
  - De több ezer már problémás lehet 60 FPS-nél
- Felhasználói felületet (HUD) nem rajzolunk
  - HTML-ben oldjuk meg a canvas felett
  - Szöveget kiírhatunk canvasra is
- WebGL is alkalmas 2d-ben rajzolásra
  - Gyorsabb, kevésbé kényelmes
  - Vannak hozzá könyvtárak: PixiJS, TwoJS

requestAnimationFrame

- Bármikor lehet rajzolni
- Stabil, magas FPS-hez akkor rajzolunk, amikor böngésző amúgy is kiteszi a képet
- requestAnimationFrame pont ezt csinálja
- Általában 60 FPS
  - Mobilokon 30 FPS bizonyos esetekben
  - Erősebb mobilokon a képernyő frissítést tartja
    - 90, vagy akár 120 FPS
  - PC-ken is 60 fölé megy, ha a monitor engedi

WebAssembly

- Letölthető és futtatható .wasm bináris fájl
- Fordítók
  - Emscripten: C, C++
  - Blazor: C#, beleteszi a CLR-t is
- Nem tud hozzáférni a DOM-hoz
  - Egyenlőre, de talán soha…
- Jól elkülöníthető algoritmusokat lehet futtatni
- Nem feltétlen gyorsabb, mint JS
  - JS erősen optimalizált már
  - Gyorsabb, ha az adatformátum nem JS szerű

Kérdések?