Multiplatform szoftverfejlesztés

Web alapú alkalmazások

Web alapú alkalmazás

- Webes technológiákat használ
  - HTML, CSS, JS
- Asztali és/vagy mobil alkalmazások
  - Nem weboldalak, nem web alkalmazások
- Multiplatform
  - A lehető legtöbb eszközön menjen
  - 5% felett: Windows, macOS, iOS, Android
  - 5% alatt: Linux, ChromeOS, …

Web alapú alkalmazás

- Alkalmazásként működik
  - Új néz ki, mint egy alkalmazás – design
  - Új viselkedik, mint egy alkalmazás
    - Nem linkel ki, …
    - Együttműködik a többi alkalmazással
    - OS integráció (share, drag&drop, …)

Webes technológia

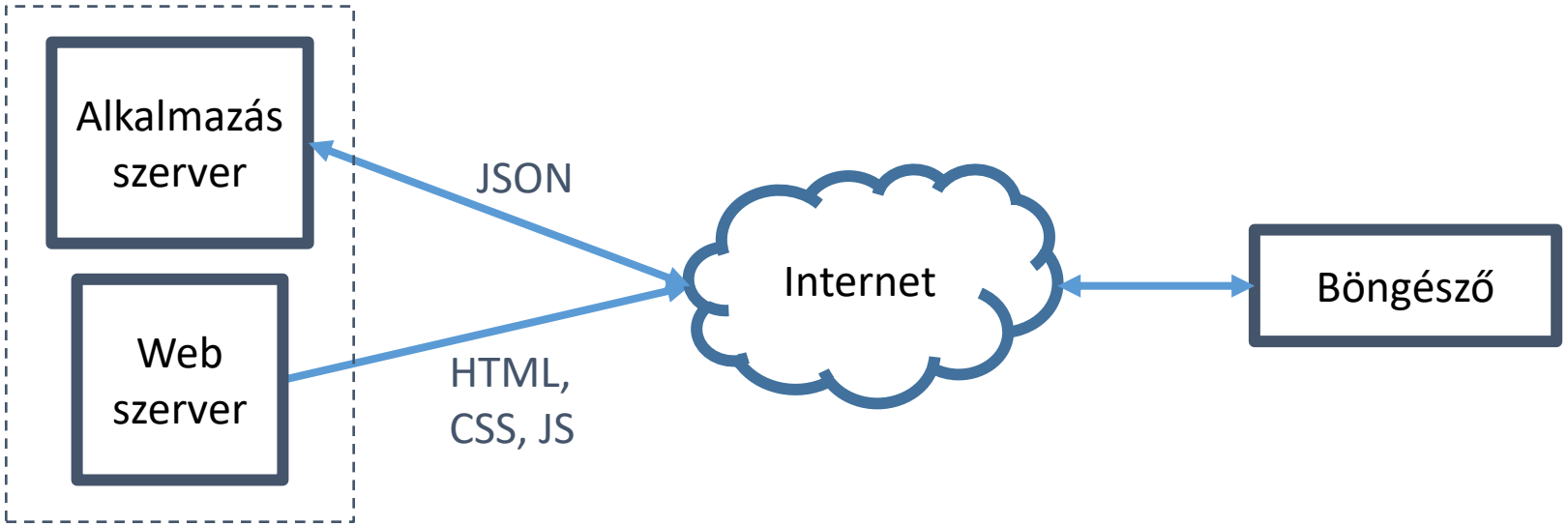
- Webes technológiák használata felhasználói felület készítésére
  - HTML elemek + CSS
    - Tartalomfogyasztó alkalmazások (Twitter, …)
      - Ezek lehetnek sima webalkalmazások is
    - Utility és productivity alkalmazások
    - Egyszerűbb játékok
  - Canvas
    - Tipikusan játékok (legnagyobb bevétel mobilon)
    - Esetleg multimédia alkalmazások
- Hogyan lehet hatékonyan fejleszteni HTML-en és CSS-en alapuló alkalmazást?
  - Canvas-ra visszatérünk

Kliens, vagy szerver oldali keretrendszer

Architektúra

- Szerver oldali keretrendszer

- Kliens oldali keretrendszer



Szerver oldal

- Példa: ASP.NET, PHP, JSP
- Minden kérést a webszerver kezel
- Bemenet: HTTP kérés
- Kimenet: HTTP válaszban HTML [+JS+CSS]
  - Teljes oldal, vagy csak része
  - Ha nem teljes oldal, akkor kliensen kell JS a feldolgozáshoz
- Működik JS nélkül is
  - Ha van JS, akkor lehet validálást, felhasználót segítő apróságokat végezni

Kliens oldali keretrendszer

- Példa: Angular, React, Vue
- Webszerver statikus oldalakat ad vissza
- Alkalmazás szerver JSON-t ad vissza
  - Kliens készíti el a HTML oldalt az adatokból
- Nem működik JS nélkül – ma már nem gond
- Szervert nem terheli a render

Kliens oldal – előnyök

- Jobban skálázódik
  - Szerver oldalról egy nagy tétel eltűnik (render)
- Gyorsabb
  - Kliens nem felejt navigálások között
    - Nem kell újra letölteni az oldal részeit
  - Maga a lekérés is kisebb
    - JSON kisebb, mint az abból készített HTML
- Gyorsabbnak tűnik
  - Kliens oldalon lehet animációkkal, stb. úgy tenni, mintha az adat már itt is lenne

Kliens oldal – előnyök

- Hozzáfér csak kliens oldalon lévő szolgáltatásokhoz (share target, notification)
- Aktív kapcsolatot tarthat fenn a szerverrel
  - Frissítheti magát
- Több szerverrel is kommunikálhat
  - Nem kell minden kommunikációnak átmenni saját szerveren
- PWA (Progressive Web Apps)
- A hibrid megoldások (szerver render, de van kliens oldali része is) ezek egy részét tudják

Botok (crawler) – kliens oldal

- Indexelő crawlerek
  - Google, Bing, DuckDuckGo, Yandex, …
  - Linkeket követik
  - Legtöbb nem hajt végre JS-t
  - Néhány igen
    - Google és Bing még XHR-t is végrehajt
    - Websocketet egyik sem tud
  - Komplex oldalon hibáznak
    - Lehet/kell segítséget adni, mindegyiknek van leírása
    - Sajnos több száz crawler van

Botok (share link) – kliens oldal

- Share link botok
  - Facebook, Twitter, Skype, Viber, Telegram, …
  - Egy link miatt elemzik az oldalt, keresnek
    - Képet
    - Címet
    - Leírást
  - Nem keresnek linkeket, nem követik őket
  - Kliens oldali kódot általában nem hajtanak végre
  - Bonyolult oldal esetén hibáznak
    - Segíteni kell nekik
    - meta tagek formájában (pl. Open Graph tagek)

Botok – szerver oldal

- Szerver oldalon renderelt oldal hátrányai nem problémák botok esetén
  - Nem kell validálni
  - Nem kell interakció: animáció, egyéb hatások
- Minden botnak más a fontos
  - Például képméret függő, hogy facebook hogyan teszi ki a linket
  - Több különböző oldalt/variációt kell gyártani
    - Akár 3-4
  - Szerver oldali render kell

Botok – amikor nem számít

- Ha az oldal nem indexelhető
  - Intranet site
    - Csak a site felhasználóinak érdekes adat
  - Védett tartalom
    - Bizonyos felhasználók férhetnek csak hozzá
  - Alkalmazás
    - Játékok
    - Utility appok
    - Egyéb alkalmazások
  - Vagy ezek kombinációja
- Akkor nem kell foglalkozni a botokkal

SSR – Server Side Rendering

- Szerver oldalra átvitt kliens kód
  - Tipikusan Node.js szerverrel
  - Nem megy minden – nyilván
- Minden általunk vizsgált technológia támogatja
- Hibrid megoldás: szerver és kliens render
  - Első körben a szerveren elkészül a HTML
  - Böngésző megjeleníti
  - Majd letöltődik a teljes app
  - Átveszi a kész HTML-t, működik minden

Kliens oldali render

A probléma

- Miért kell keretrendszer?
  - HTML-t előállítani nem nehéz

```
let name = "Leo";
div.innerHTML = `<span>Kedves, ${ name }!</span>`;
```

  - Eseményekre cserélni is tudjuk
  - Bonyolult HTML-t is tudunk készíteni
    - Bár az olvashatósága egy stringben nem jó
    - Nincs szintaktikai színezés és kódkiegészítés
    - De szét tudom bontani egyszerűbb komponensekre
      - Ez amúgy is jó gyakorlat

A probléma

- A teljes HTML cserélése nem jó
  - Villog
  - Lassú
    - 10-500 ms függően az elemek számától
    - Egyszer nem gond
    - Minden bemenetre nem életképes
  - Elveszti az állapotot
    - Fókusz
    - input típusú elemek

A probléma

- Megkereshetjük a kérdéses elemet

```
let span = container.querySelector( "div span.sum" );
span.textContent = "Összeg: " + total;
```

  - Sajnos a kód függ a HTML felépítésétől
    - Karbantarthatóság problémás
  - Ha nem akarunk használni semmit, akkor ez a megoldás

Struktúra szinkronizáció

- A HTML fa felépítésű, egy ágon sok levél lehet
- Fa/lista szinkronizációs probléma (set/tree reconciliation)
  - Van két listánk A és B, generáljunk módosító utasításokat (insert, remove)
    - A-ból B legyen
    - Legkevesebb utasítás generálódjon
  - Fára ez $O(n^3)$ - lassú
- Naiv módszer: oldjuk meg csak beszúrásra és törlésre
  - Átrendezésre előről kezdjük

Megoldás

- Készítsünk egy keretrendszert, ami
  - Az általunk megadott HTML-t legyártja
  - Képes frissíteni
    - Nem villog
    - Nem lassú
    - Nem veszti el az állapotot
  - Opcionálisan
    - Adatkötést támogat, akár kétirányú adatkötést
    - Jó a tooling
      - Segít a HTML szerkesztésében
      - Ellenőrzi a kódot szerkesztés/fordítás időben
      - Segít debuggolni, ha gond van

Vizsgált keretrendszerek

- React
  - Csak UI keretrendszer
  - Legnépszerűbb
  - Nagy ökoszisztéma
- Vue
  - Szintén csak UI keretrendszer
- Angular
  - Teljes keretrendszer
- Összehasonlítás a végén

Kompozíció (összetétel)

Strukturális tervezési minta általában (Composite)

A probléma

- Szeretnénk felépíteni egy struktúrát, ami a felhasználói felületet jól leírja és kezeli
- Lehetne típusonként eltérő interfész
  - Konténereket máshogyan kezelni, mint az elemeket
  - Ez kényelmetlen a fejlesztőnek
    - Minden típusra eltérő kódot kell írni
    - Tesztelni, újra felhasználni (pl. konténerben konténer)
- Lehetne egzotikus struktúra
  - Nem fa, hanem például gráf
  - Meg akarjuk oldani, hogy egy elemnek több szülője is legyen, stb.

A megoldás (kompozíció minta)

- Egy elemet (komponens) attól függetlenül akarunk kezelni, hogy az egy konténer, vagy csak egy pici rész/elem
  - Egymásba ágyazhatóság miatt legyen minden komponensnek egy olyan interfésze, ami támogatja a minimumot függetlenül attól, hogy levél, vagy ág
- Fa struktúrában szeretnénk tárolni
  - Egy szülő és 0, vagy több gyerek
  - Adjuk fel az egyéb struktúrákat, mert nehéz kezelni őket

A megoldás

- Meg lehet oldani objektum orientált módon
  - Örökléssel: Levél és Konténer származik komponensből
- Vagy simán minden komponens
  - Ebben az esetben futásidőben kell megoldani, hogy ha valaminek nem lehet gyereke
    - Ez nem feltétlen gond, mert a kivételeket amúgy is kezelni kell

Kivételek

- Készíthetünk olyan elemet, aminek
  - Nem lehet gyereke
  - Megadott számú gyereke lehet csak
    - Például 1
- Ezeket nehéz örökléssel kezelni
  - Főleg, hogy adatfüggő is lehet
- A fa felépítésénél kell hibát dobni
  - Ez lehet futásidőben
  - Vagy a szerkesztő eszköz által
    - Ez utóbbi gyakori

Felhasználása

- Nagyon gyakran használt minta felhasználói felület kialakítására
  - Adja magát
- Minden általunk tárgyalt keretrendszer ezt használja
- A komponensek célja
  - Dekompozíció: részekre bontani a bonyolult felületet
  - Felelősség: csak saját magán belül felelős, de ott mindenért
  - Újrafelhasználás
    - Jól körülhatárolt, ezért jó eséllyel működik máshol is

Komponens

- A komponensben van
  - Nézet (HTML) leírása: sablon, vagy kód
  - Nézettel való interakció: események, adatkötés
  - Állapotkezelés
- A komponensek egymásba ágyazhatók
  - Így épül fel a logikai fa
  - Vannak komponensek, amik csak levelek lehetnek

Eszközök

VS Code, webpack, ...

Történelem

- 2007: Steve Jobs vízionálja a webet, mint appot telefonon (iPhone bemutatása)
  - De nem ez valósult meg, 2008-ban kijött az AppStore
- 2010 körül váltak a böngészők futtatókörnyezetté
  - Előtte lassú volt minden, nem volt általános a web alapú alkalmazás
- 2010: NPM – Node Package Manager
- 2010: AngularJS, ez zsákutca lett
- 2013: React
  - 2019: Hooks

Történelem

- 2013: Electron
  - Böngésző elég gyors
- 2014-től jelennek meg a csomagolók
  - Webpack (2014), Rollup (2015), Parcel (2017) és társai
  - Fejlesztés közben is képesek csomagolni, akár hot reload képességekkel
  - A végső csomag optimalizált
    - Tree-shaking (DCE: Dead Code Elmination)
    - Lazy loading, modulokra bontott
    - Minimalizált
    - Verzionált, hogy pl. ne ragadjon be régi verzió a cache-ben

Történelem

- 2014: Vue
- 2015 PWA
  - Az ötlet, hogy átvegyük a vezérlést a cache felett
  - De ekkor még használhatatlan
- 2015 Visual Studio Code, Webstorm, …
  - Webstorm (Jetbrains) eredetileg 2010-es, de alapvető funkciókat később kap csak
  - Visual Studio nem jól használható ebben a modellben
- 2016: Angular
- 2020 ESM csomagolók (Vite, WMR, …)
  - Nem csomagolnak fejlesztés közben

Történelem

- Jelen
  - Lassan változnak szokások
  - Korábban elkezdett projektek nem kerülnek konvertálásra általában
  - Egyre nagyobbak az alkalmazások
    - Egy 5-10 évvel ezelőtti projekt webpackben 1-10 másodperc alatt frissül
    - Mai projekt több perc is lehet
  - Állandóan változó eszközpark a mai napig
    - Nincs integrált környezet, ellentétben sok más technológiával

Böngésző motor

- Jelenlegi böngészők a leggyorsabb UI-t biztosítják
- JS motorok lehetővé tették a szerver oldali programozást
- PWA-kat lehet csomagolni AppStore-ba és PlayStore-ba
- JS és TS alapú multiplatform fejlesztés egyre gyakoribb
  - JavaScript talán a leghasználtabb nyelv jelenleg

Kérdések?