Frontend programozás JavaScript és HTML5 segítségével: Webalkalmazások fejlesztése

...

Kiss Balázs Webváltó Kft. • 2019.05.09.

Magamról

- BME VIK Mérnökinformatikus MSc
- Frontend szoftverek tervezése és fejlesztése
- Webváltó Kft-nél 4 éve
- Vodafone Hungary, Magyar Államkincstár, Semmelweis Egyetem, Budapesti Vízművek, Magán megrendelések, Nemzetközi projektek
- Fő tech: Angular

Áttekintés

Alapfogalmak

JS történelem

JS 2019-ben

Kapcsolódó technológiák

1. Alapfogalmak

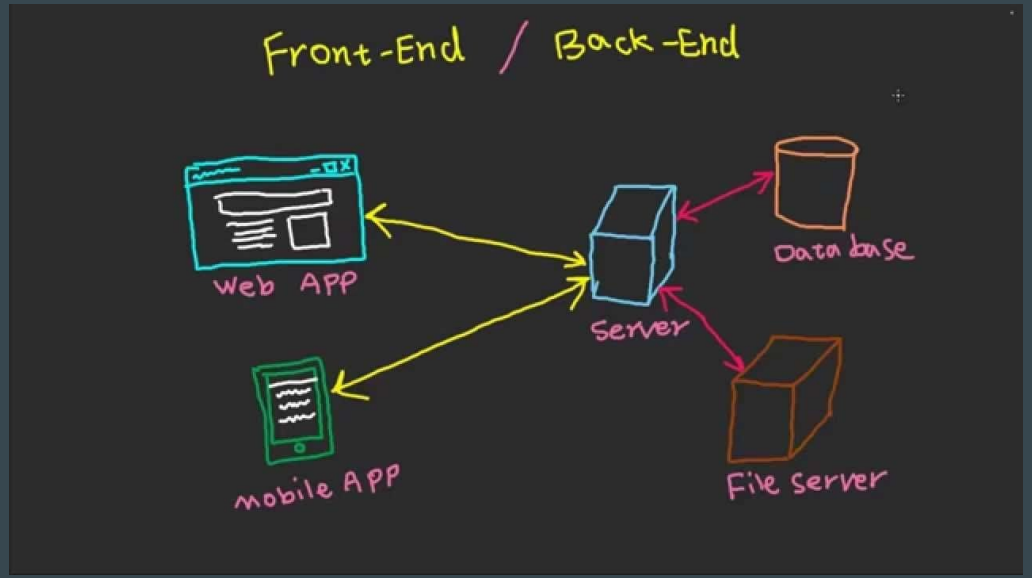
Frontend-backend
Webes frontend
HTML, CSS, JS

1. Alapfogalmak - Frontend vs. Backend

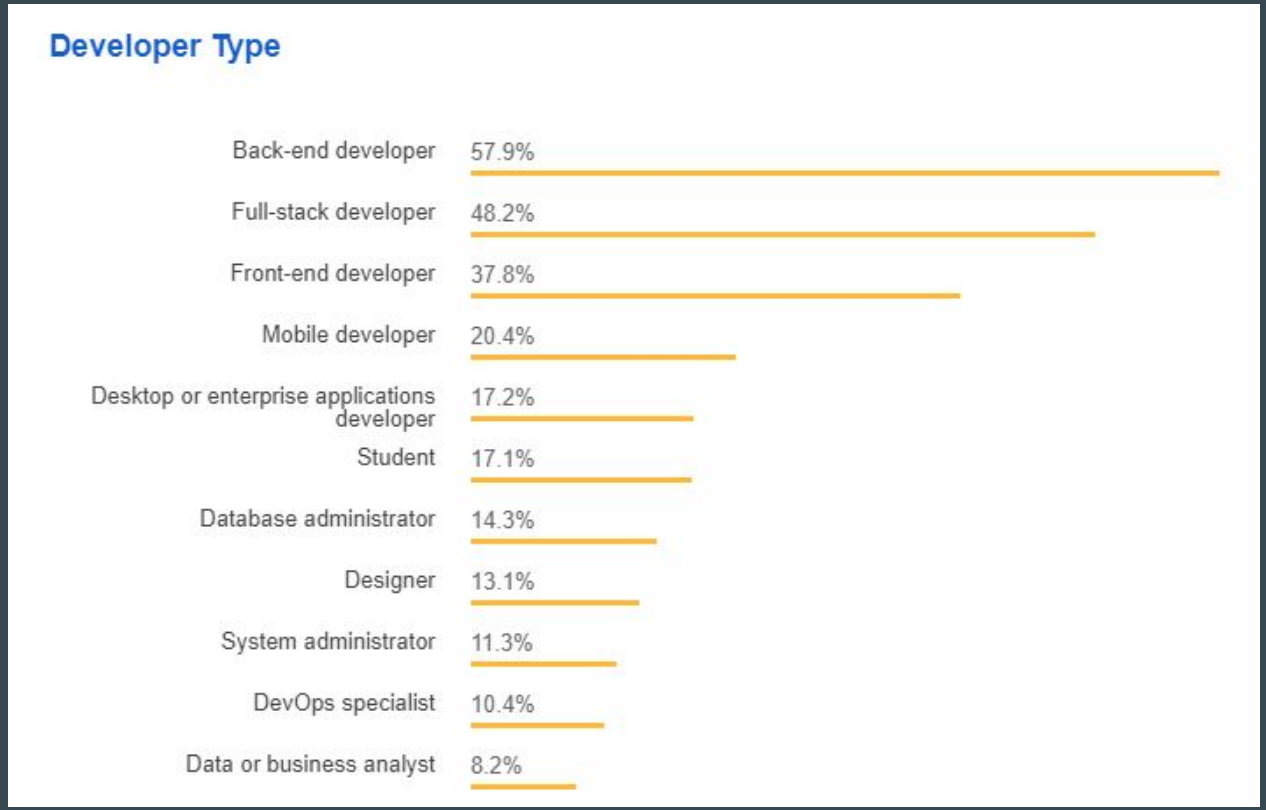
Front-end: azok a **felületek,** amellyel a **felhasználó közvetlenül kapcsolatba lép.**

Back-end: az alkalmazás-rendszer működéséért felelős réteg.

A határok elmosódnak...



StackOverflow 2018 Survey Results

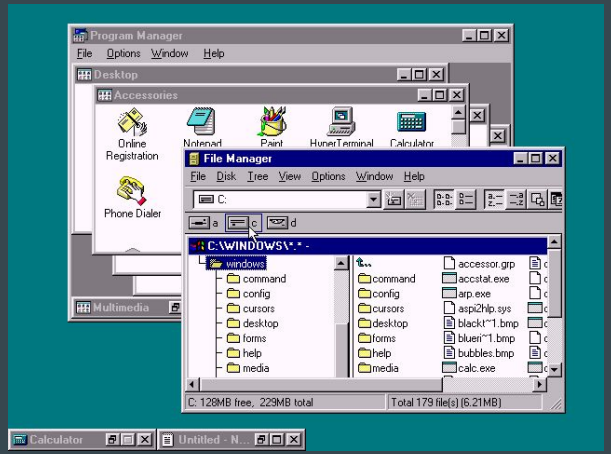


1. Alapfogalmak - Frontend fejlődése

	Fókusz	Interaktivitás
Mainframe	• Green-text, CLI • Egyszerű, funkcionális • Szerver ++	• Kezdetleges, CLI • Nincs testreszabhatóság • Funkcionális

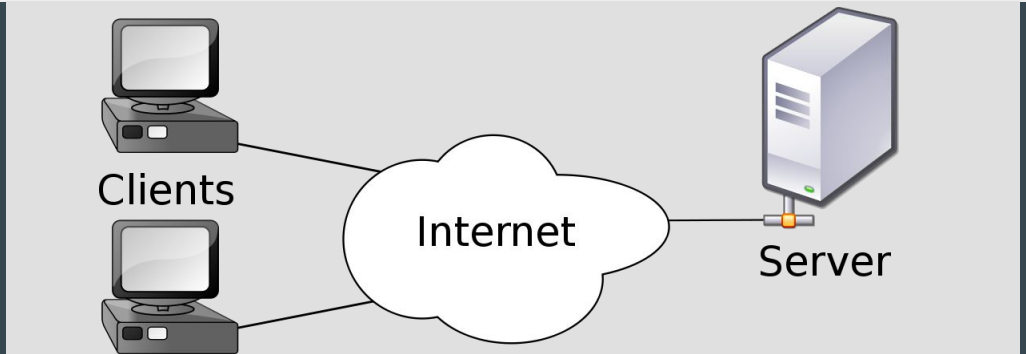
1. Alapfogalmak - Frontend fejlődése

	Fókusz	Interaktivitás
Mainframe	• Green-text, CLI • Egyszerű, funkcionális • Szerver ++	• Kezdetleges, CLI • Nincs testreszabhatóság • Funkcionális
Desktop	• Asztali alkalmazás a user gépén • Minden funkció Frontend-en • Frontend ++	• Egér + multimédia • Magas testreszabhatóság • Ezer féle platform

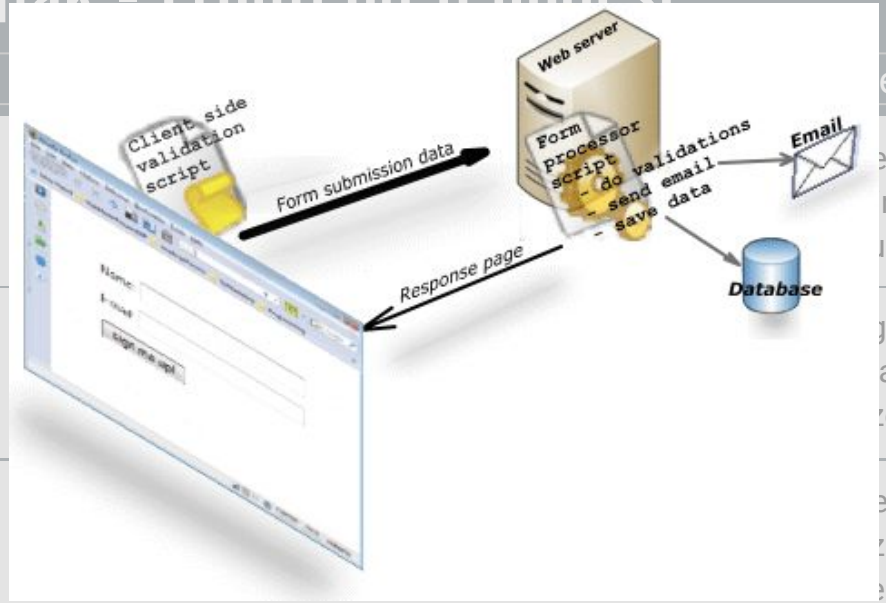


1. Alapfogalmak - Frontend fejlődése

	Fókusz	Interaktivitás
Mainframe	• Green-text, CLI • Egyszerű, funkcionális • Szerver ++	• Kezdetleges, CLI • Nincs testreszabhatóság • Funkcionális
Desktop	• Asztali alkalmazás a user gépén • Minden funkció Frontend-en • Frontend ++	• Egér + multimédia • Magas testreszabhatóság • Ezer féle platform
Kliens-szerver	• Szerver + Desktop • Adatok szerver oldalról, kliens gépén “utókezelés”	• Desktop-éval egyezik • Számításba kell venni a nem helyben történő folyamatokat

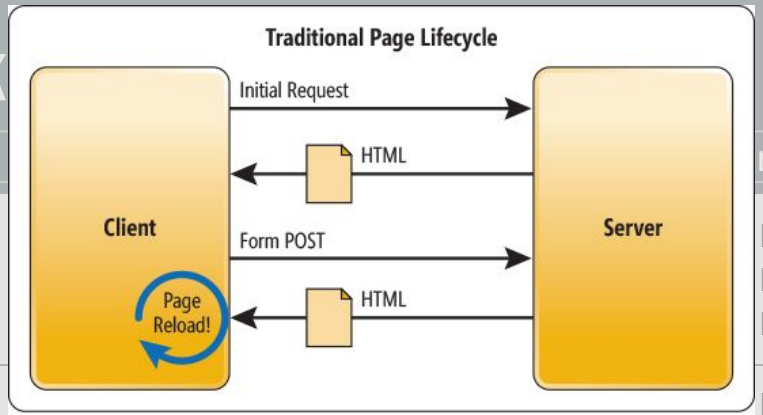


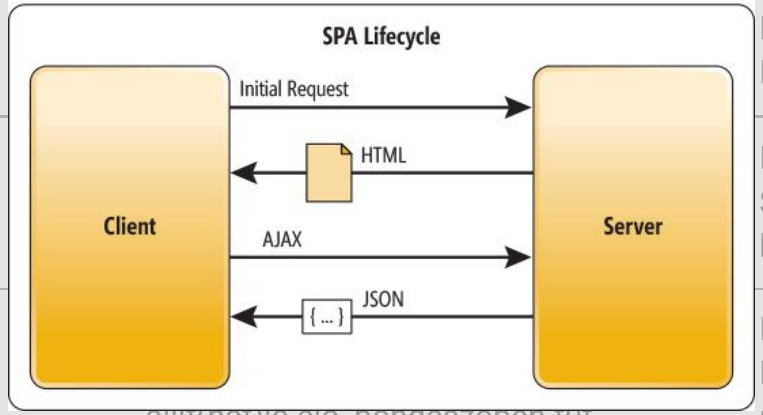
1. Alapfogalmak - Frontend fejlődése



		eraktivitás
Mainframe		ezdetleges, Green-screen, CLI ncs testreszabhatóság unkcionális
Desktop		gér + multimédia agas testreszabhatóság zer féle platform
Kliens-szerver		esktop-éval egyezik zámításba kell venni a nem elyben történő folyamatokat
Statikus web	• Kliens-szerver egy speciális esete • Kliens-t “HTML-t” is a szerver állít(hat)ja elő, böngészőben fut	• Kevés interaktivitás • Form-ok kitöltése, gombok, navigálás

1. Alapfogalmak

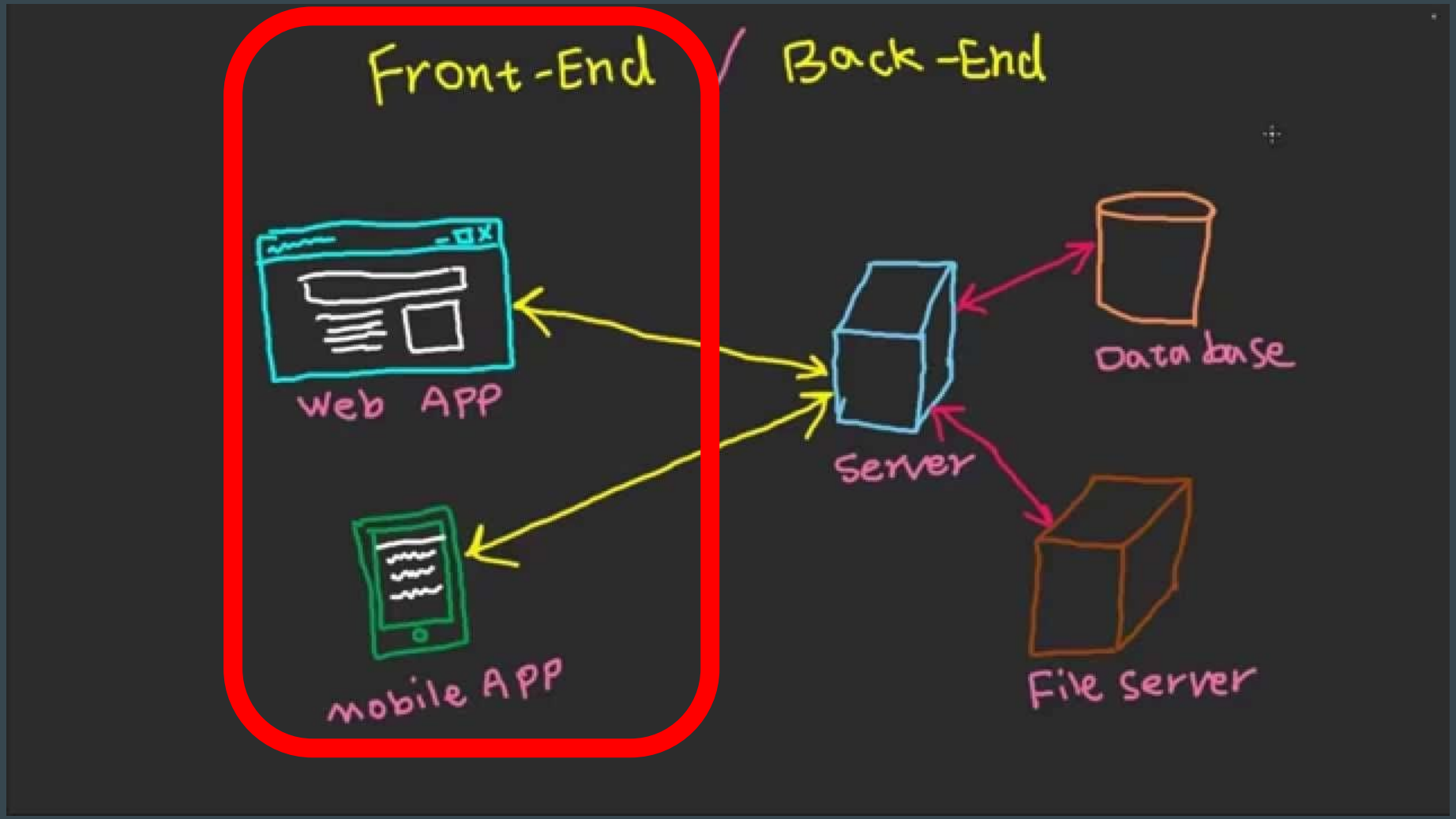




		interaktivitás
Mainframe		Kezdetleges, Green-screen, CLI Nincs testreszabhatóság Funkcionális
Desktop		Egér + multimédia Magas testreszabhatóság Ezer féle platform
Kliens-szerver		Desktop-éval egyezik Számításba kell venni a nem helyben történő folyamatokat
Statikus web	állít(hat)ja elő, böngészőben fut	Kevés interaktivitás Form-ok kitöltése, gombok, navigálás
Dinamikus web	• Klienst már részben a szerver, részben a felhasználónál futó szoftver (JS) állítja elő • Dinamikus kommunikáció	• Gazdag interakciós lehetőségek • Ötvözi a megoldásokat (SPA)

1. Alapfogalmak - Frontend fejlődése

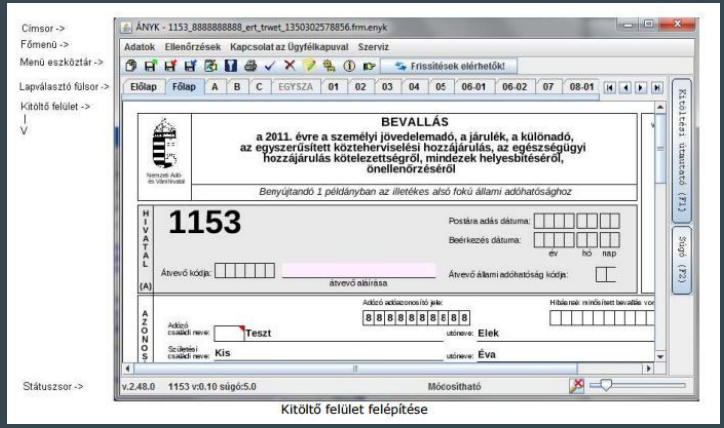
	Fókusz	Interaktivitás
Mainframe	• Green-text, CLI • Egyszerű, funkcionális • Szerver ++	• Kezdetleges, CLI • Nincs testreszabhatóság • Funkcionális
Desktop	• Asztali alkalmazás a user gépén • Minden funkció Frontend-en • Frontend ++	• Egér + multimédia • Magas testreszabhatóság • Ezer féle platform
Kliens-szerver	• Szerver + Desktop • Adatok szerver oldalról, kliens gépén “utókezelés”	• Desktop-éval egyezik • Számításba kell venni a nem helyben történő folyamatokat
Statikus web	• Kliens-szerver egy speciális esete • Kliens-t “HTML-t” is a szerver állít(hat)ja elő, böngészőben fut	• Kevés interaktivitás • Form-ok kitöltése, gombok, navigálás
Dinamikus web	• Klienst már részben a szerver, részben a felhasználónál futó szoftver (JS) állítja elő • Dinamikus kommunikáció	• Gazdag interakciós lehetőségek • Ötvözi a megoldásokat (SPA)


Front-End
Back-End
Web App
Mobile App
Server
Database
File Server

1. Miért jó frontend app-nak SPA?

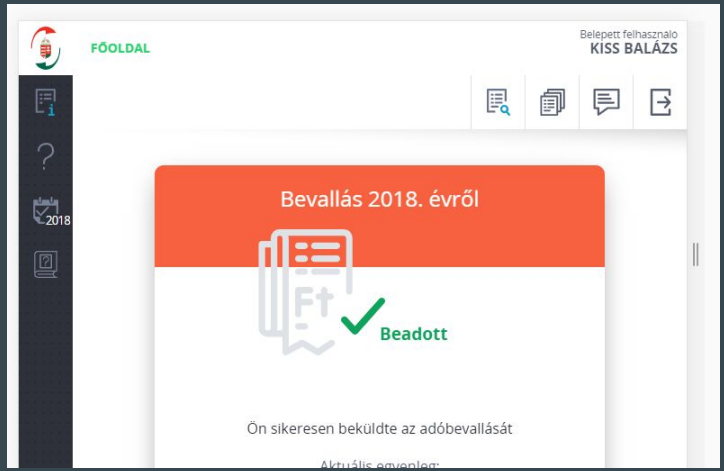
- MVC egyből adott
  - View rétegnek tökéletes váz a HTML
  - ViewControl esetén JS + HTML
  - Control esetén szintén backend + JS
- Nem kell telepíteni, karbantartani (auto update)
  - Kivéve, ha....
- Kompatibilis minden platformon
  - A böngészők az új JVM?
- Nem kell teljes alkalmazást elérhetővé tenni
  - Lazy-load
- Analitikai lehetőségek
  - Nehéz lekövetni egy egyszer eladott lemezt
- Felhasználók jobban szeretik:
  - Koherens, megszokott felület
  - UserExperience megoldások
  - Userek számára a legjobb: mobil app
    - Hiszen, mobilja mindenkinek van

NAV ÁNYK



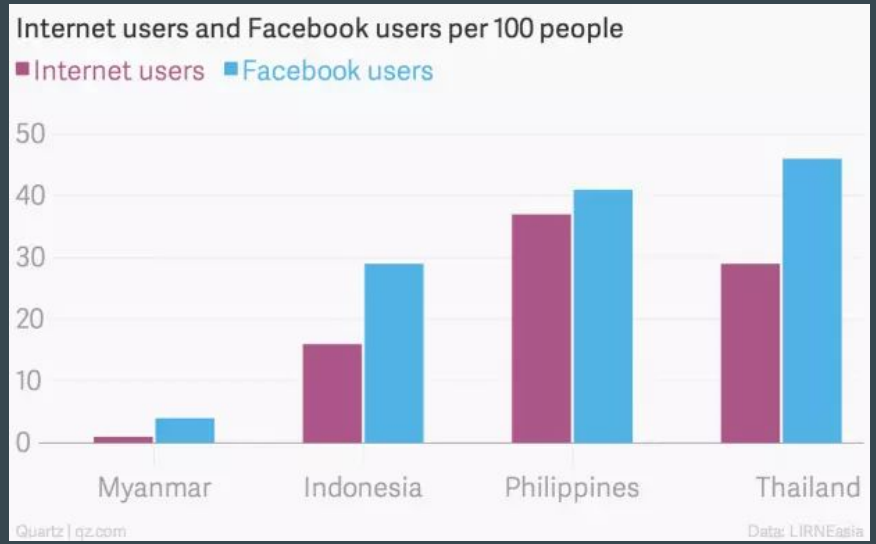
Kitöltő felület felépítése

Adóbevallás 2019



1. Miért jó frontend app-nak SPA? 2.rész

- 2010': a desktop appok "halála"
- A felhasználók az élményhez kötnek, nem a technológiához
- Webappok elterjedése desktop társakkal ellentétben
  - Gmail
  - Facebook
  - Spotify
  - OneDrive, Office 365
- Mobil/Desktop/Webes app
  - Közös nevezőre hozni sokat segíthet
- App store-ok
  - Google Play
  - Steam

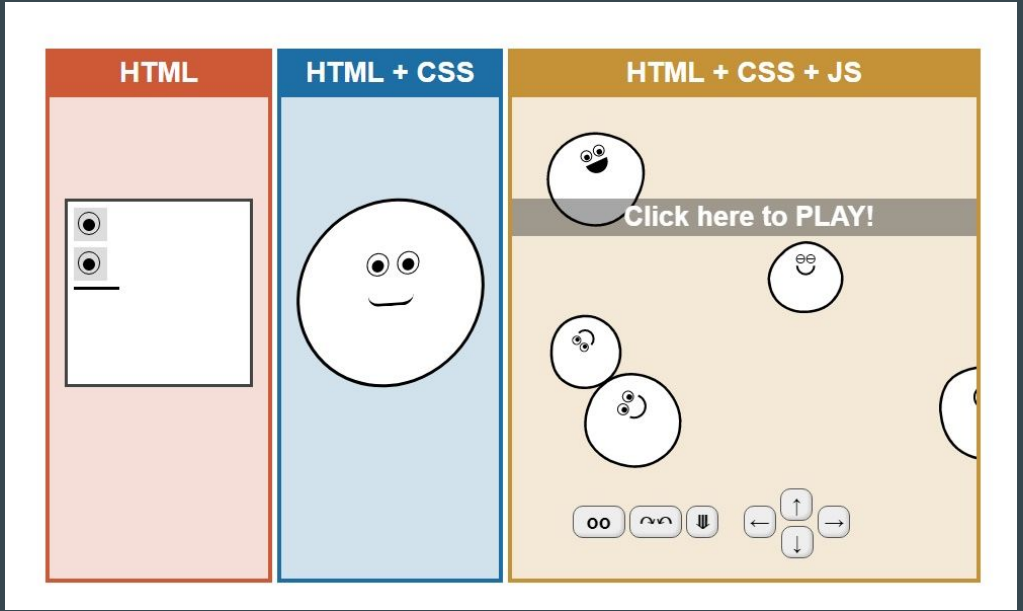


1. Miért NEM Jó frontend-re a webapp?

- Natív alkalmazások többet tudnak
  - Nem éri utol a fejlődés soha, van ahol direkt
- NPM “hell”, biztonsági kérdések
- Offline működés problematikus
  - PWA / Natív alkalmazások a jövő?
- Erősen limitált telepíthetőség, működés
  - Tabok? Ikonok? Nincs még erős fő csapásirány
- Hosting pénzbe kerül...
  - Ha mindenki letölti az appot, az drágább mintha mindenki telepíti
- Valószínűleg a Desktop és a Webapp világ egymás mellett fog még fejlődni jó sokáig

1. Alapfogalmak - HTML, CSS, JS

HTML, CSS, Javascript



- **World Wide Web Consortium**
- **HTML:** Hypertext Markup Language, egy dokumentum leíró nyelv, XML-szerű
- **CSS:** Cascading Style Sheets: stílus leíró nyelv, **"XML szerű"** fájlok megjelenítéséhez (mivel a
- **JavaScript:** Egy szkript nyelv, amivel interaktívvá tehető az egyébként statikus HTML+CSS tartalom

1. Alapfogalmak - DOM

DOM: Document Object Model

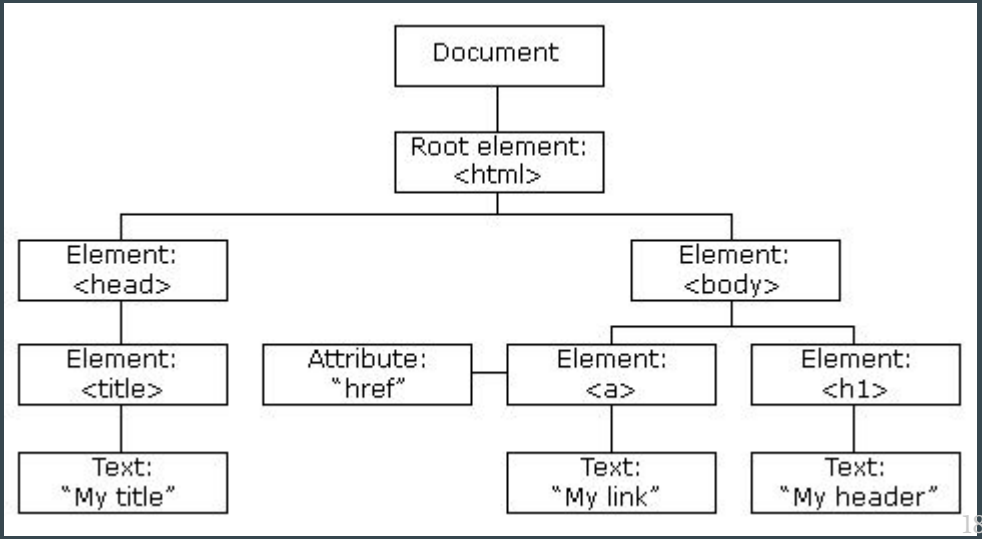
HTML: ennek a leírására szolgáló nyelv

(“programozási” nyelv-e? Elméletileg igen, gyakorlatilag nem)

HTML definiálja a DOM:

- Kiindulási állapotát
- Rajta értelmezett JS funkciókat

HTML != DOM!

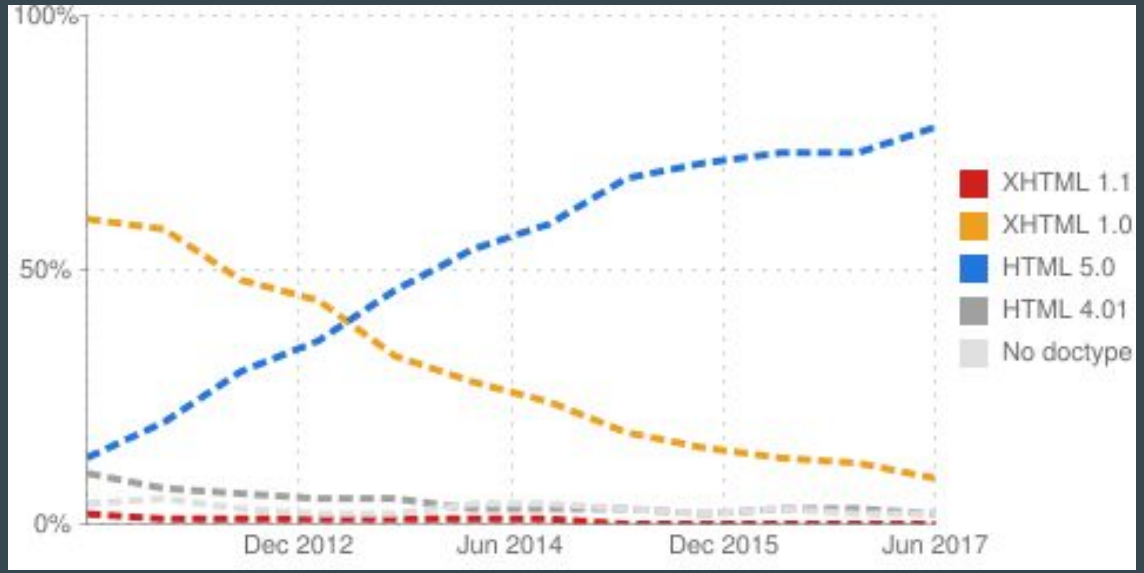


1. Alapfogalmak: DOM, és egy példa: HTML, CSS, JS együtt

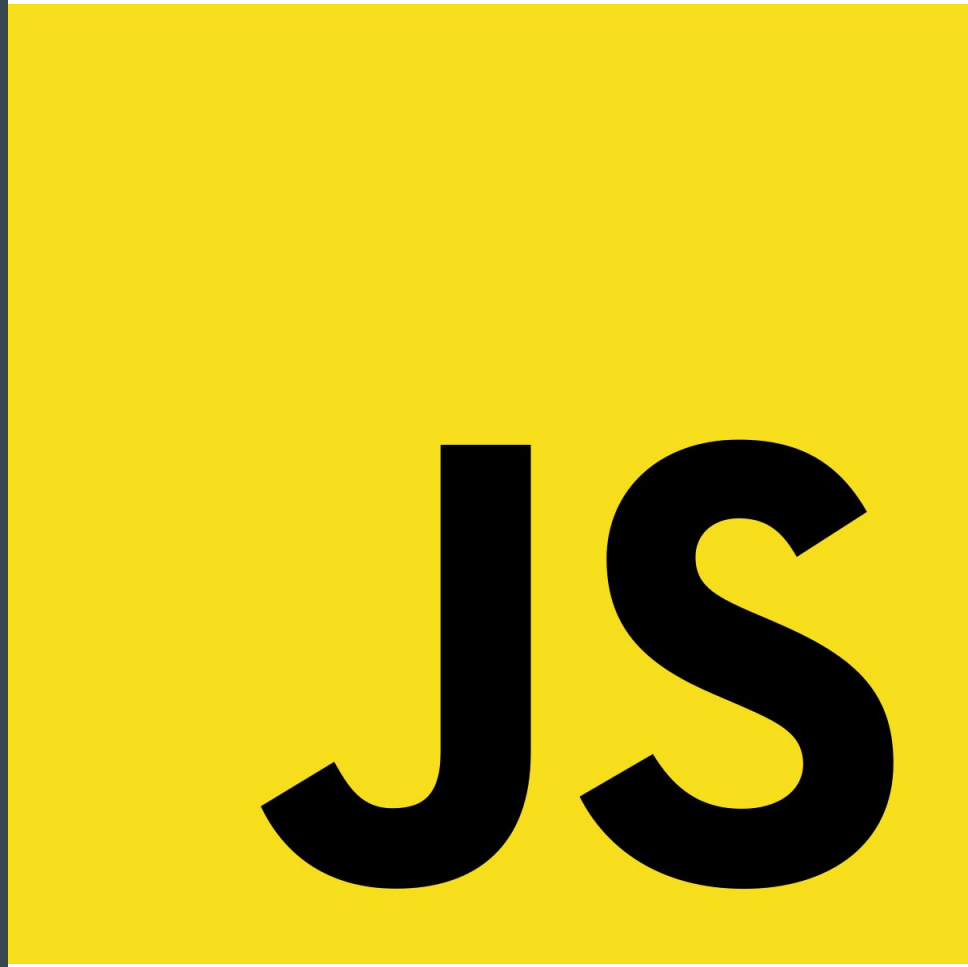
Példa: https://codesandbox.io/s/xomppjkvzo)

- HTML tag-ek
- Script Tag
- CSS

HTML fejlődés



2. JavaScript



2. JavaScript (ECMAScript)

“JavaScript ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. It was invented by Brendan Eich, co-founder of the Mozilla project, the Mozilla Foundation, and the Mozilla Corporation.”

ECMAScript: Szabány, Standard
JavaScript: “Dialektus”

Jelenleg: ES6

Brendan Eich

https://github.com/getify/You-Dont-Know-JS

2. JavaScript (ECMAScript) - fejlődés

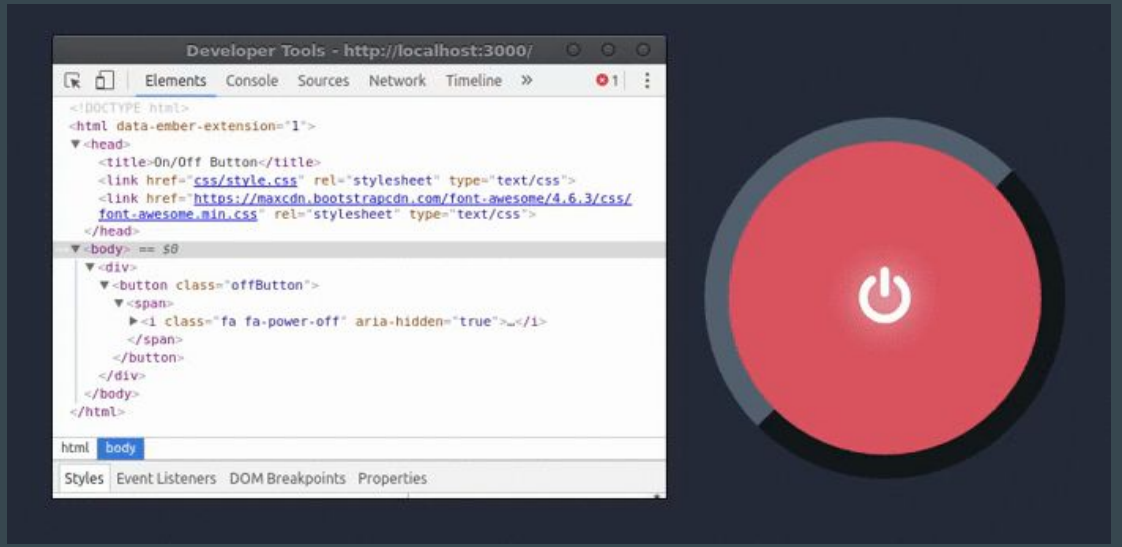
1	ECMAScript 1 (1997)	First Edition.
2	ECMAScript 2 (1998)	Editorial changes only.
3	ECMAScript 3 (1999)	Added Regular Expressions. Added try/catch.
4	ECMAScript 4	Never released.
5	ECMAScript 5 (2009)	Added "strict mode". Added JSON support. Added String.trim(). Added Array.isArray(). Added Array Iteration Methods.
5.1	ECMAScript 5.1 (2011)	Editorial changes.
6	ECMAScript 2015	Added let and const. Added default parameter values. Added Array.find(). Added Array.findIndex().
7	ECMAScript 2016	Added exponential operator (**). Added Array.prototype.includes.
8	ECMAScript 2017	Added string padding. Added new Object properties. Added Async functions. Added Shared Memory.
9	ECMAScript 2018	Added rest / spread properties. Added Asynchronous iteration. Added Promise.finally(). Additions to RegExp.

2. JavaScript működés

- Source kód -> Lexing -> parser -> translator -> böngésző natív kódja
- JustInTime compilerek: helyben fordul (hasonló, mint a Java ByteCode JIT)
  - Mozilla: SpiderMonkey / TraceMonkey
  - Google: V8 (NodeJS is ez)
  - Rendering Engine vs. Scripting Engine

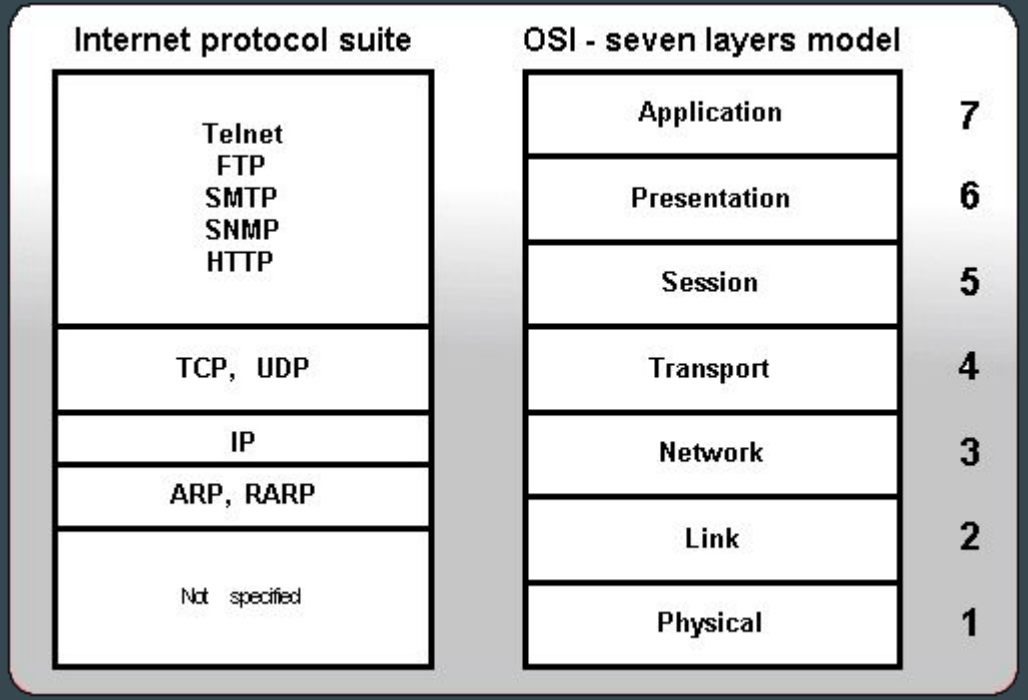
Mire jó?

- **Dinamikus DOM Manipuláció**
  - Nem kell mindent újra renderelni
- Egyeduralkodó és független.



2. JavaScript kommunikáció

- HTTP kommunikáció, böngésző a fő platform (Security réteg)
- Szöveg alapú
- Applikációs rétegben
- Websocketek
- XHR, Ajax

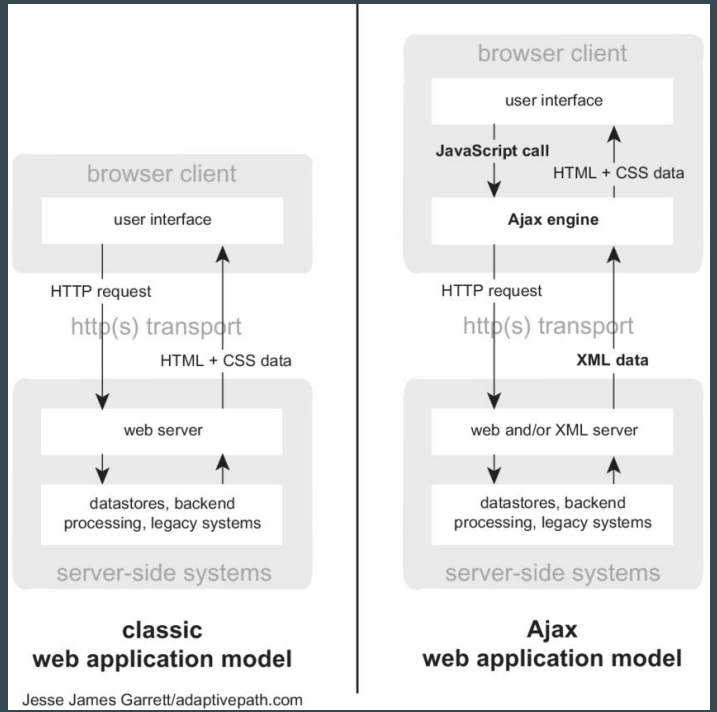


2. JavaScript - elmélet

HTTP request methods (GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATH)

AJAX

- Weboldal betöltés után aszinkron kommunikáció a háttérben
- **Javascript zömében aszinkron szkriptek futásából áll.**



2. XML vs. JSON

JavaScript Object Notation

Ordered name-value pairs

JSON:	XML:

{
“AGE”: 29,
“NAME”: “Balázs”
}
	<PERSON>
	<AGE>29**</AGE>**
	<NAME>Balázs**</NAME>**
	</PERSON>

```
myObj = JSON.parse(xxx);
myObj.age
```

```
myObj = someXMLParseFunction(xxx);
myObject.getChilrdren(“person”)[0].getChildnren(“age”)[0].value();
```

AJAX-hoz jobb a JSON, mert:

- gyorsabb a parse
- és kisebb mint az XML.

`JSON.parse(JSON.stringify(obj))`

2. JS Összefoglalás

- **Frontend:** Felhasználói interakciókra szolgáló réteg, hosszú fejlődés, jelenleg a böngészők a legnagyobb platform
- **Ami mindennek az alapja:** HTML, CSS, JavaScript
  - HTML -> DOM leíró
  - CSS: Eyecandy
  - JavaScript -> valódi program
- **JavaScript:** a DOM-ra specializált programozási nyelv
  - **Kommunikáció:** HTTP felett, metódusokkal, aszinkron, főleg JSON-ben

Áttekintés

Alapfogalmak

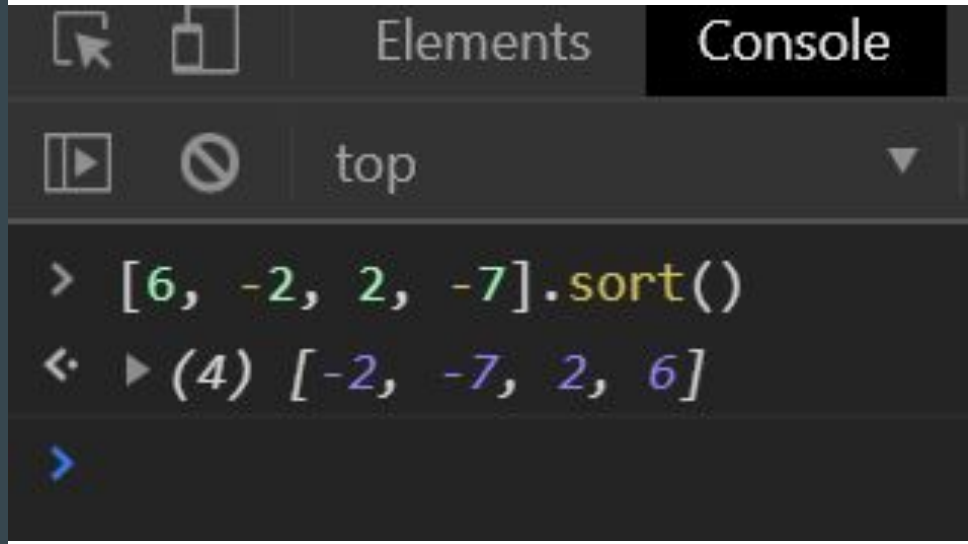
JS történelem

JS 2019-ben

Kapcsolódó technológiák

3. JS 2019-ben

TypeScript, Angular, React Vue



3. Keretrendszerek - de miért?

- Általános előnyök (mint minden más programozási nyelvben)
- Programozási architekturális pattern-ek integrálva
- Miért nem Static Site generation?
- Keretrendszerek = Evolúció

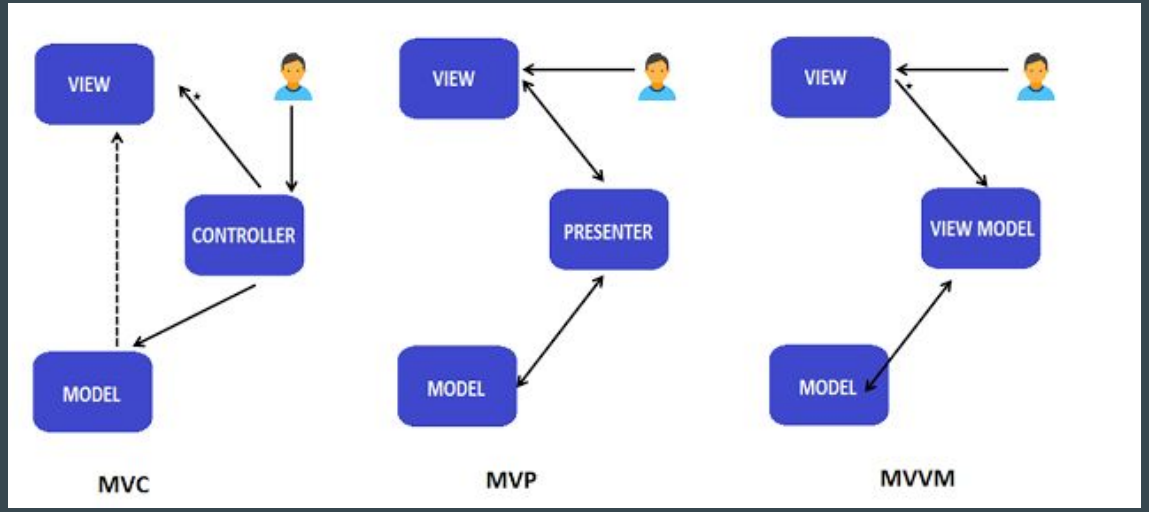
Fő ok:

- Kényelem
- Előre megírt eszközök (boilerplate csök
- **STATE kezelés:**
  - Keretrendszerek nélkül lehetetlen jól újrafelhasználhatóságú kódot írni
  - Web komponensek kevesek ehhez
  - Virtual DOM, DOM Frissítés

2. Keretrendszer típusok

3 nagy kategória:

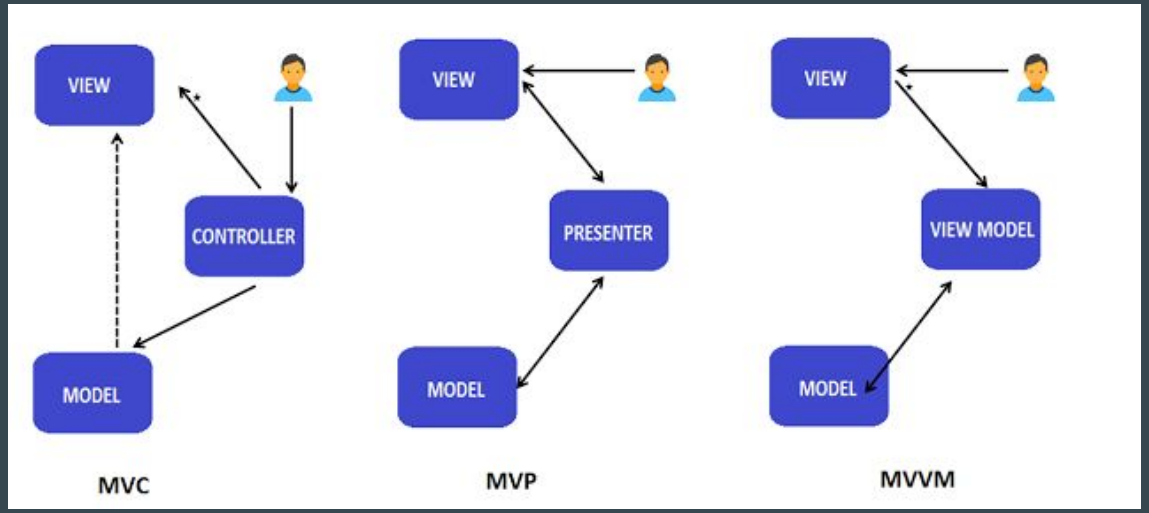
- MVC
  - Éles határok
- MVP
  - Presenter
- MVVM
  - Two-way bind



2. Keretrendszer típusok

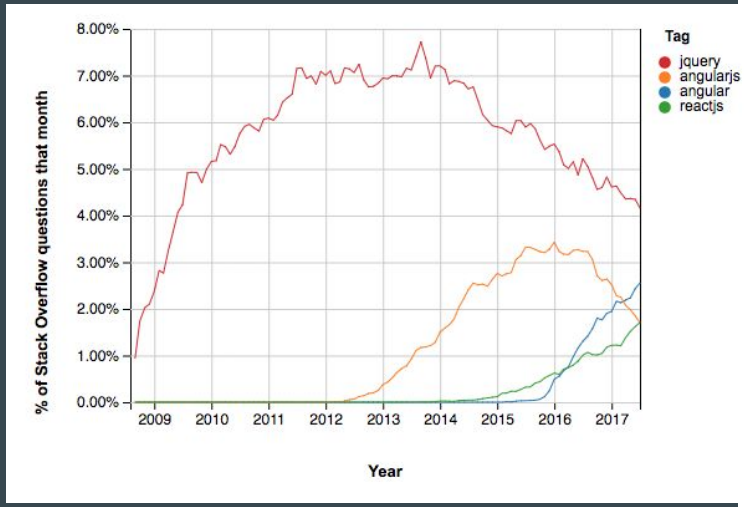
3 nagy kategória:

- MVC
  - Éles határok
- MVP
  - Presenter
- MVVM
  - Two-way bind

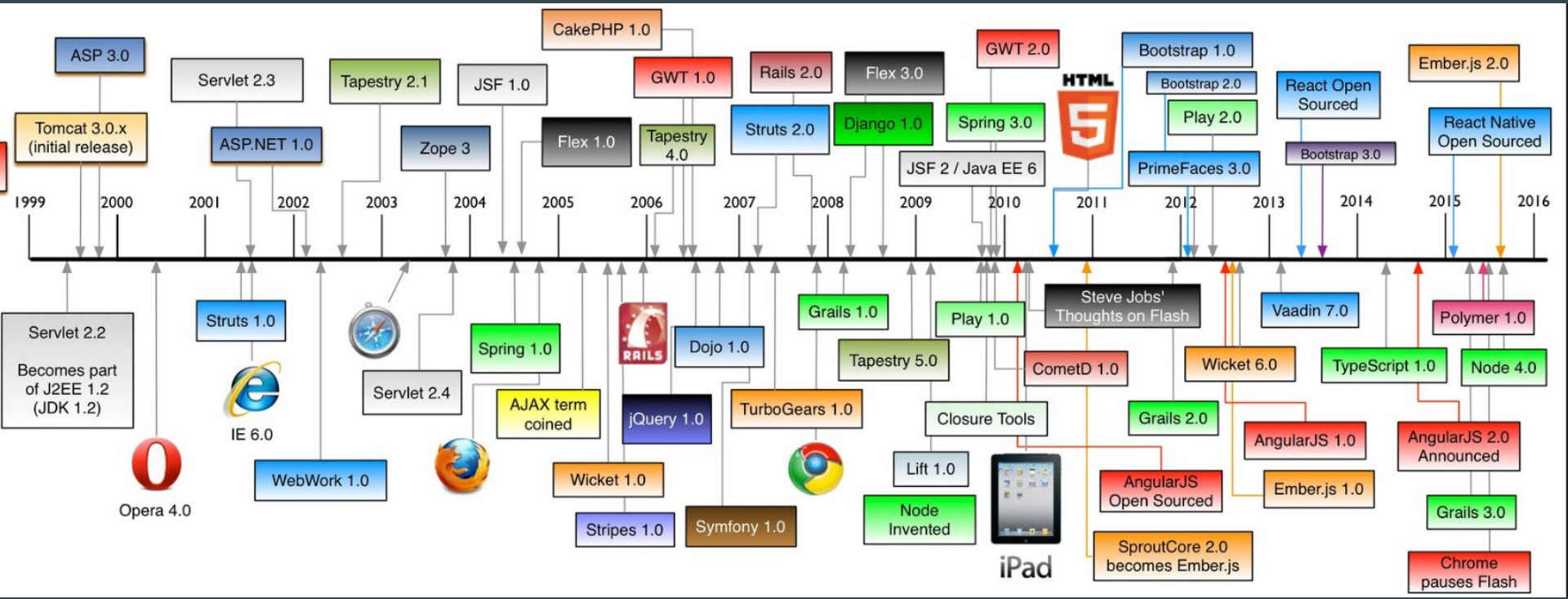


\+ MVWhatever

3. Mi volt eddig?



3. Mi volt eddig?



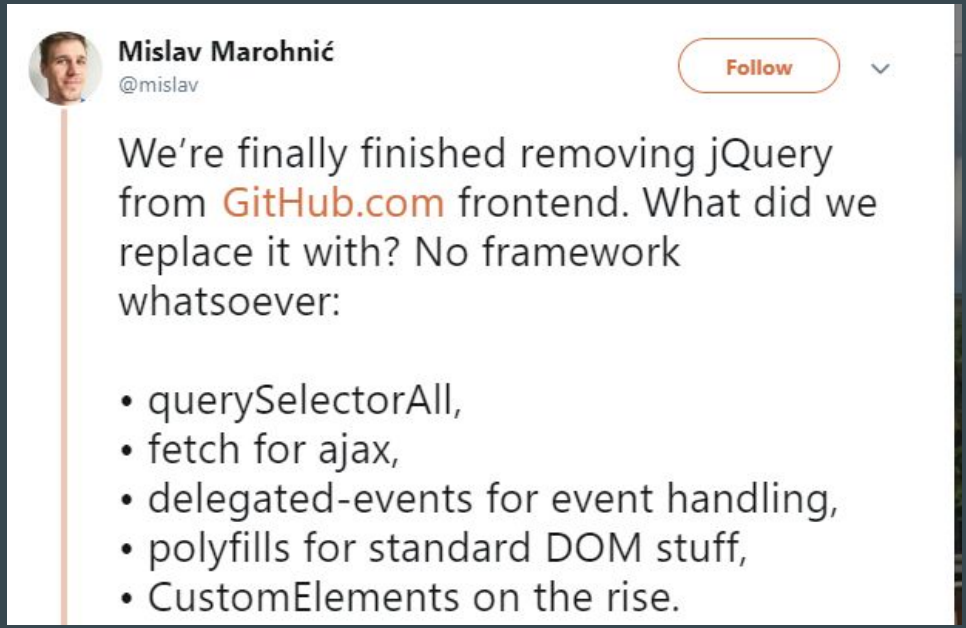
3. Mi volt eddig?

Fejlődést meghatározta:

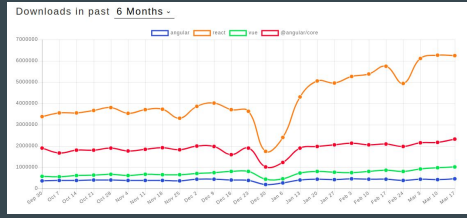
- HTML korlátai
- ECMAscript korlátai
- HTTP kommunikáció korlátai

Tipikus példa: jQuery

- Egy időben nélkülözhetetlen eszköztár
- Beolvasztották a szabványba
- Mostmár **felesleges**

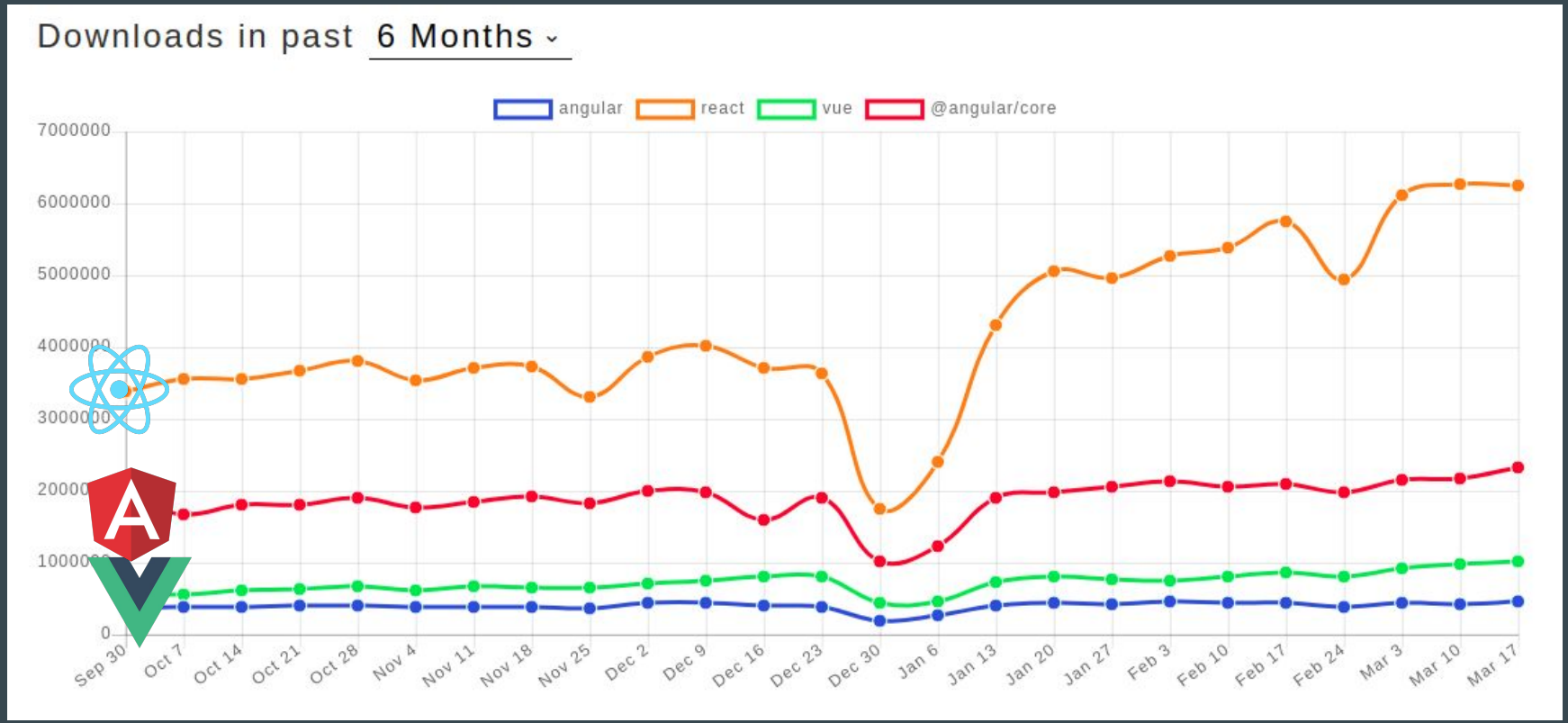


3. A nagy három



	Vue.js	React	Angular
Definíció	MVVM interface **library**	Interface builder **library**	MVW **framework**
Megjelenítés	Single file components	JSX	HTML
Szerepkör	Csak Library	Csak Library	Kompakt
State management & Data binding	VueX	Redux	Two-way binding One-way binding
“Sztár” felhasználó	GitLab	Facebook, Netflix, Reddit, Paypal	Google, Forbes, Wix

3. A nagy három



3. A nagy három

Angular

- Typescript
- Kompakt, heavy
- Modulok, Binding, DI
- Material
- **PWA-hoz**

Vue

- Kicsi, gyors, egyszerű
- Kompakt, könnyen kiegészíthető
- Kicsi, könnyű, hozzáadható egyszerűen bármihez
- **Lightweight SPA**

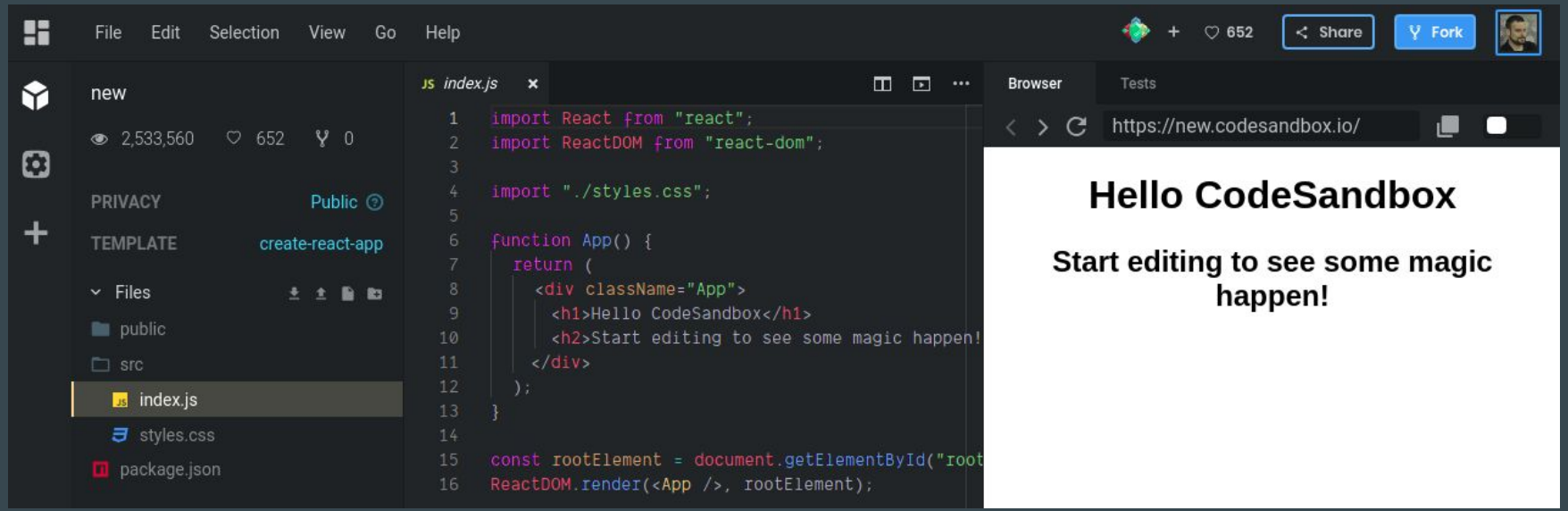
React

- View-Kód egyben: JSX
- Középen van bonyolultság terén
- App Store, Redux
- **PWA-hoz / State elemzéshez**

3. Fejlesztő környezet

a.) Lokális környezet telepítése

b.) Online szerkesztők pl. https://codesandbox.io/



Áttekintés

Alapfogalmak

JS történelem

JS 2019-ben

Kapcsolódó technológiák

4. Kapcsolódó technológiák

Web Components, Chrome, NPM, Aszinkronitás, RxJS, PWA ...

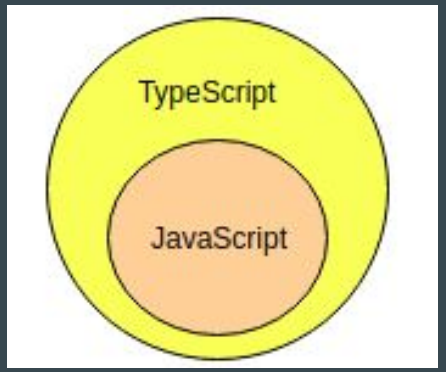
4. TypeScript

Érvényes JS szintaxis:

```
let result = 12;
result = “Típus módosító érték”;
```

Dependency Injection esetén valami hasonló a szintaxis:

```
class MyFirstClass {
  constructor(someService: SpecialServiceType){
    let x = this.someService.mySpecificValue
  }
}
```



4. SPA vs. MPA?

- Gyorsabb
- Nagyobb kezdeti payload
- Frontend
- SEO optimalizálás: trükközni kell
- JavaScript
- Frontend oldali rendering (kivételekkel)

- Lassabb
- Kevesebb kezdeti payload
- Backend + Frontend
- SEO optimalizálás sima ügy
- Nem kell JavaScript
- Szerver oldali rendering

Eredmény:

- Legtöbb hátrányt megoldotta az SPA világ
- Céloldal: SPA
- Weboldal-ökoszisztéma: MPA

4. NPM

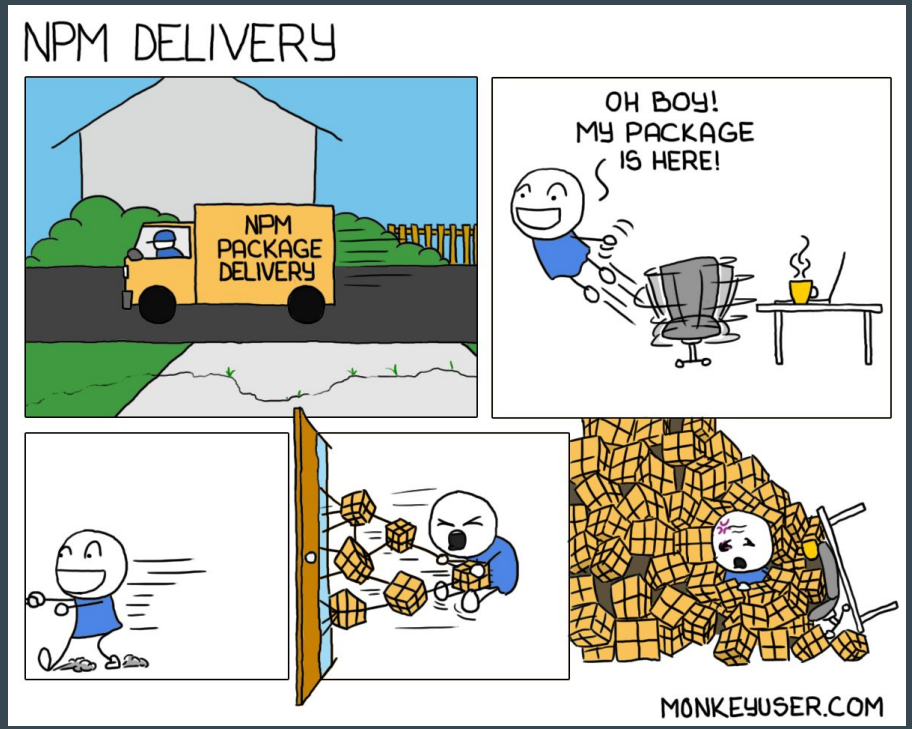
Node Package Manager: Build és Dependency manager tool

Mi ez? Három dolog:

1. Software Registry JavaScript package-ekhez
2. Egy weboldal
3. Egy Command Line Interface

Mire használjuk?

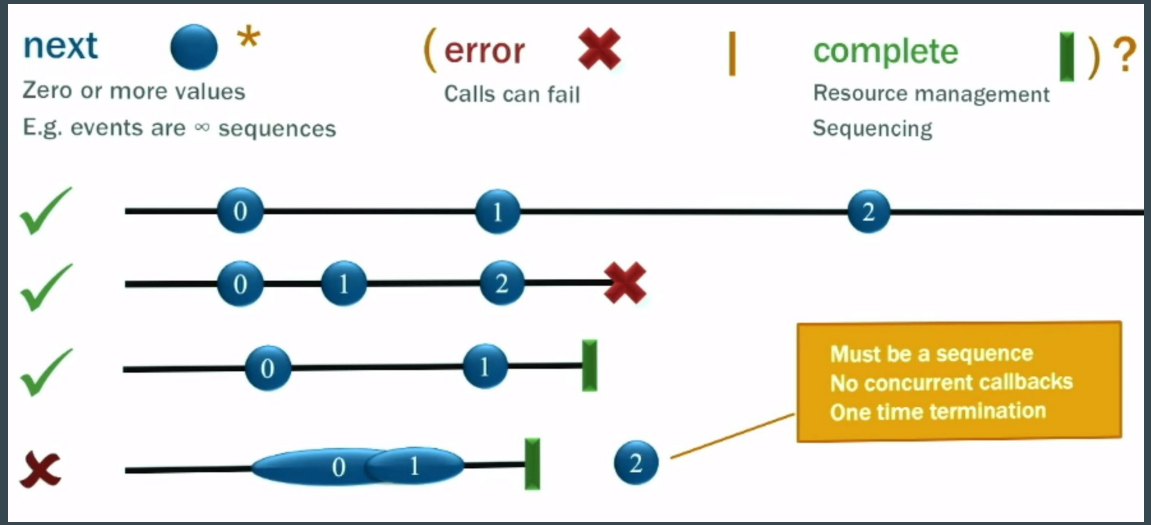
- Projekt kezelés: **package.json**
- Package kezelés: **npm install**
- (java-ban kb. a Maven)



4. Reactive Programming

Reactive Programming

- Browser események
- Backend kommunikáció
- Stream (filter, map, reduce, merge stb.)



Általános szabályok

- Stream = egy esemény-sor, aminek a rész-eseményei (időben) sorrendben történnek
- Események = értékek, hibák, indikátorok
- Promise és Observable

4. Reactive Programming - Promise

Promise object - valójában JavaScript aszinkron hívások

```
const getUser = function(name) {
  return new Promise(function (resolve, reject) {
    let response = http.get…();   // blokkoló, szinkron hívás
    if(response.status === 200){
      resolve(response)
    } else {
      reject('Nem található');
    }
  });
};
```

4. Promise

Promise object

```
const getUser = function(id) {
  return new Promise(function (resolve, reject) {
    let response = http.get…();
    if(response.status === 200){
      resolve(response)
    } else {
      reject('Nem található');
    }
  });
};
```

```
getUser(2)
  .then(function (user)  { console.log(user); },
        Function (error) { console.log(error);});
```

4. Promise

Promise object

```
const getUser = function(id) {
  return new Promise(function (resolve, reject) {
    let response = http.get…();
    if(response.status === 200){
      resolve(response)
    } else {
      reject('Nem található');
    }
  });
};
```

```
getUser(2)
  .then(function (user)  { console.log(user); },
        Function (error) { console.log(error);});
```

```
getUser(2)
  .then(user => console.log(user); error => console.log(error);
```

4. Promise

Promise object

```
const getUser = function(id) {
  return new Promise(function (resolve, reject) {
    let response = http.get…();
    if(response.status === 200){
      resolve(response)
    } else {
      reject('Nem található');
    }
  });
};
```

```
getUser(2)
  .then(function (user)  { console.log(user); },
        Function (error) { console.log(error);});
```

```
getUser(2)
  .then(user => console.log(user); error => console.log(error);
```

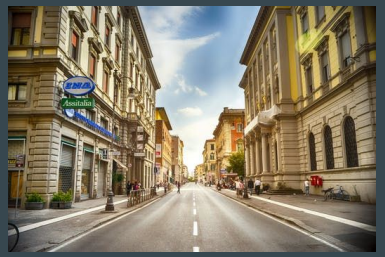
```
getUser(2)
  .then(user => findFavouriteDrink(user))
  .then(drink => serveDrink(drink))
```

4. RxJS

- **Funkcionális programozás**
  - Observer, Iterator, a korábbi egyedüli Promise + Callback hell helyett
- **Reaktív pattern**
  - Szekvenciális programfutás: A:=B+C értéke akkor értékelődik ki, mikor ez a parancs a soron következő.
    Reaktív programozás: A:=B+C értéke updatelődik automatikusan ha változik B vagy C
  - dataflow

4. Observables & RxJS

```
const myObservable = Observable.of(1, 2, 3);
```



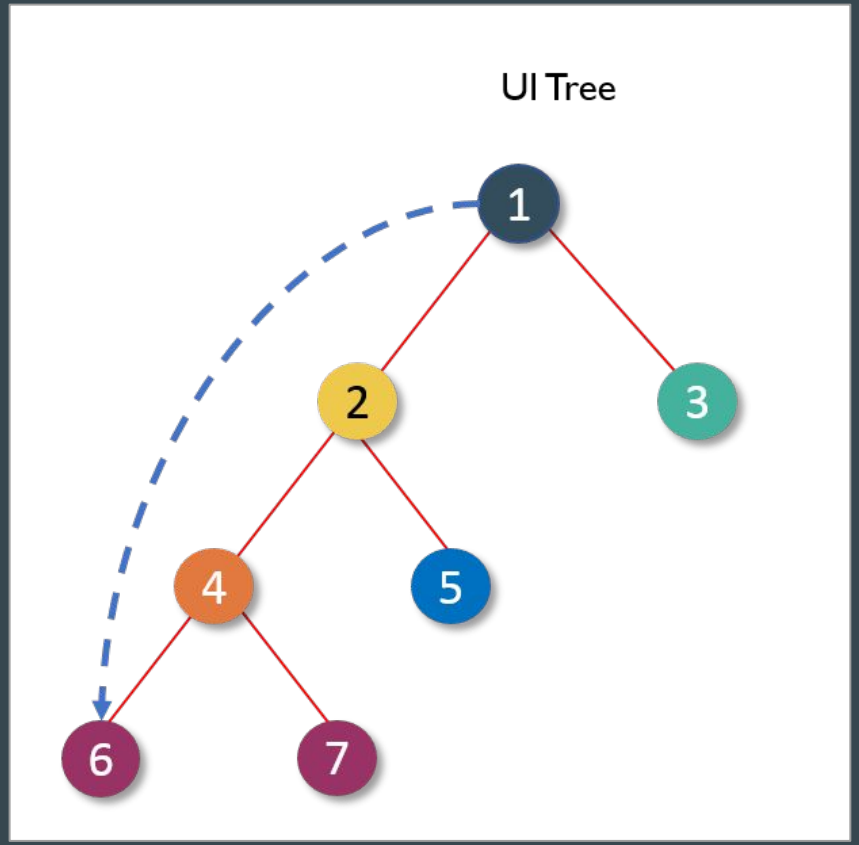
```
const myObserver = {
  next: x => console.log('got next value: ' + x),
  error: err => console.error(‘got an error: ' + err),
  complete: () => console.log('got a complete notification')
};
```

```
myObservable.subscribe(myObserver);
```

4. Redux

Redux

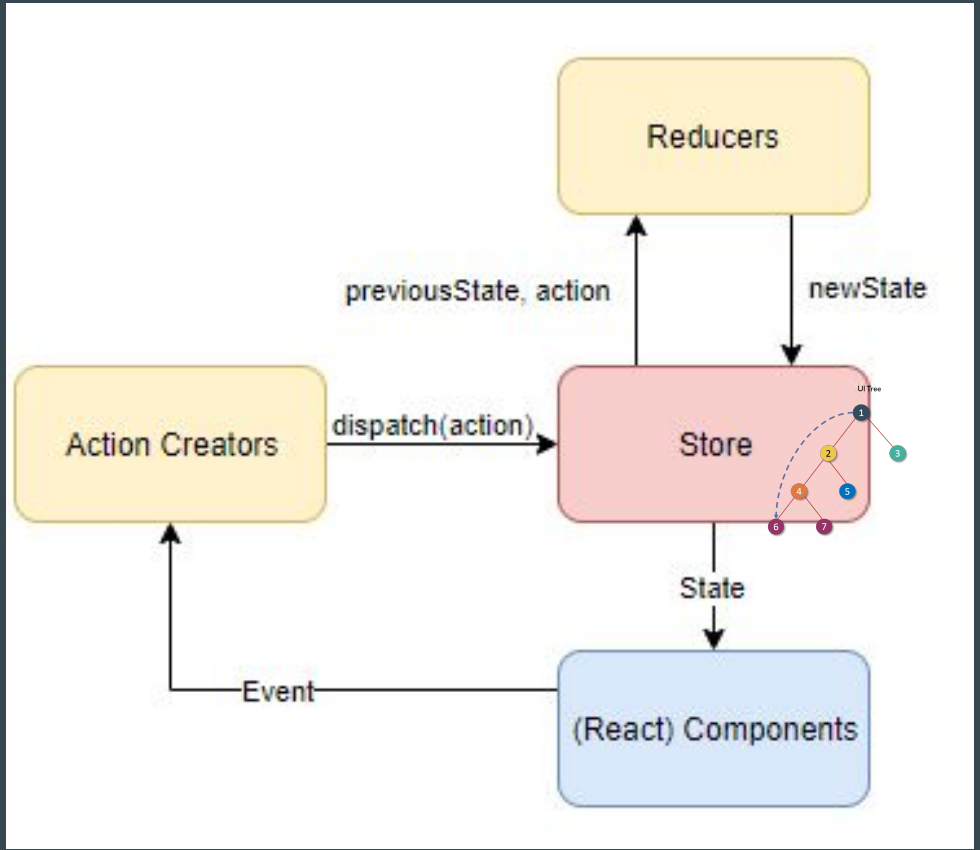
- Centralizált State kezelés
- Nem csak React-ra
- Store, event, reducer etc.
- Three principles:
  1. **Single source of truth**
  2. **State is read-only**
  3. **Changes are made with pure functions**



4. Redux

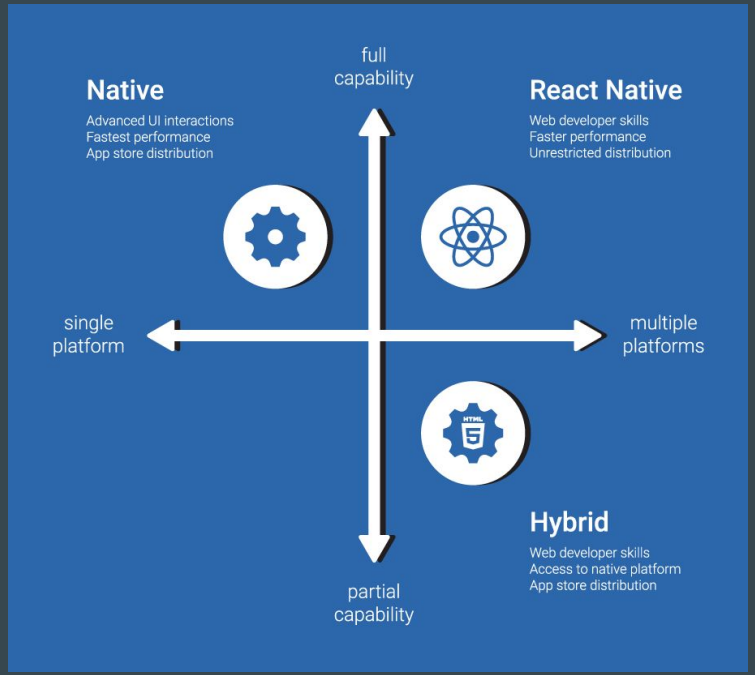
Redux

- Centralizált State kezelés
- Nem csak React-ra
- Store, event, reducer etc.
- Three principles:
  1. **Single source of truth**
  2. **State is read-only**
  3. **Changes are made with pure functions**



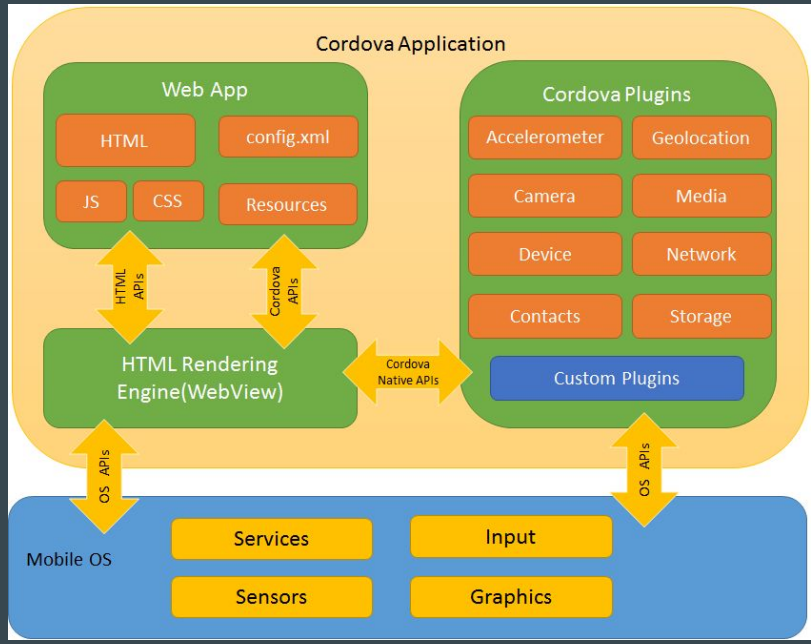
4. Mobil világ

- **Native app:** ami egyből a mobil platformra forduló kód; pl. React Native
  - Js -> android,

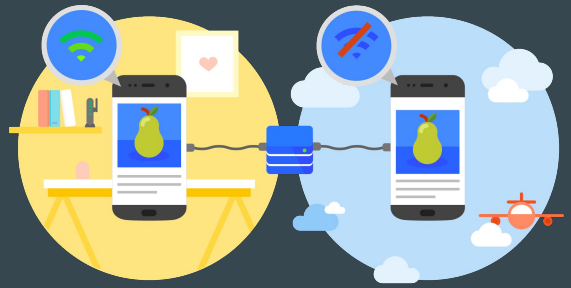


4. Mobil világ

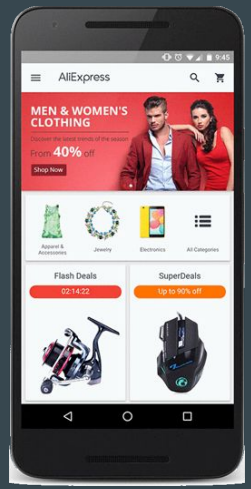
- **Native app:** ami egyből a mobil platformra forduló kód; pl. React Native
  - Js -> android,
- **Hibrid app:** Oprendszer -> applikáció ami hozzáfér a telefon funkcióihoz natívan -> belül egy böngésző -> benne egy speciális web app
    - “Majdnem” böngésző, de annál platform-orientáltabb
    - Így marad a JS meg HTML kód, de cserébe tudjuk használni a telefon egyéb funkcióit
    - Cordova
      - +1 réteg meg eszköztár: Ionic (ne kelljen platformspecifikus dolgokat írni ... túl sokat)



4. PWA



- **Progressive Web Apps (PWA)**
  - Natív app érzés
  - Mobile platform egy (jelentős) szeletének kiharapása
  - Tisztán JS, böngészőben fut, de mobil app-nak látszik
- **Firebase, Firestore (Google)**
  - Backend as a service | Service as a Service | ** as a Service
    - (noSQL)DB + common backend(!) + webstore + ezernyi hasznos támogatási funkció
  - Beépített analytics
  - Beépített projekt kezelés
  - Cloud szolgáltatások



4. Web components

Cél: egységbe zárt komponensek, komplex keretrendszerek nélkül

Generalizált kommunikáció és definíció. **w3c**

Három fő eszköz:

- JS Custom Elements API
- Shadow DOM encapsulation
- HTML templates

```
import { LitElement, html, property, customElement } from 'lit-element';

@customElement('simple-greeting')
export class SimpleGreeting extends LitElement {
  @property() name = 'World';

  render() {
    return html`<p>Hello, ${this.name}!</p>`;
  }
}
```

```
<simple-greeting name="Everyone"></simple-greeting>
```

http://dblsai.github.io/WebGL-Fluid/

4. WebGL

Kiegészíti a JS specifikációt úgy, hogy az használni tudja a böngészőn keresztül a rendszer grafikus kártyáját

(OpenGL|ES alapok)

Nem ECMAScript szabvány (még)

Erre is épülnek már keretrendszerek:

- PlayCanvas
- Unity

4. WebAssembly (wasm)

- Közelebbi nyelv a JavaScript motorhoz, mint a JS
  - Sebességkritikus
  - Assembly-szerű
  - Lehet rá fordítani C, C++-ról is többek között
  - Kvázi “bytecode”
- Verem alapú virtuális gép
  - Ugyanúgy sandbox-ban fut
  - Csak parancsfuttatás -> nincs teljes DOM manipulálás
- Gyorsabb, mint a JS
  - W3c
- 2019-ben várhatóan tovább nő az érdeklődés:
  - 2018 augusztus: Unity WebGL build erre fordul

https://www.crazygames.com/game/bullet-force-multiplayer

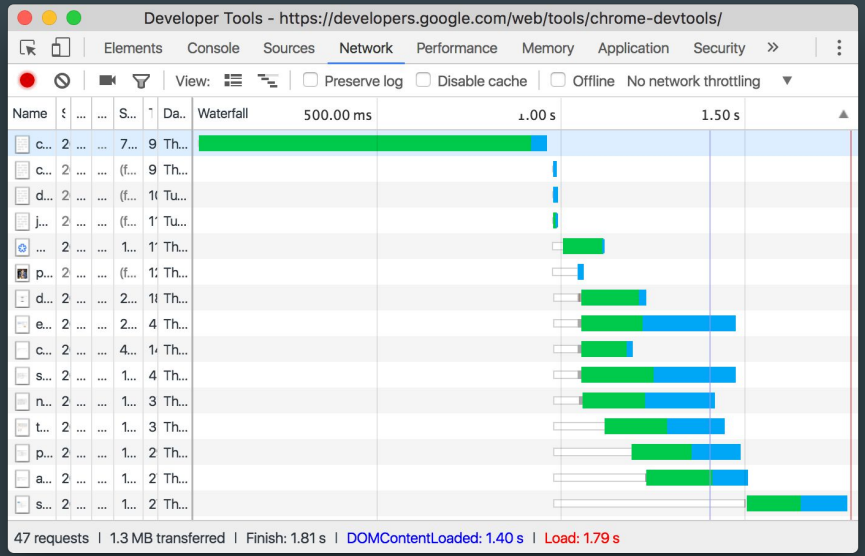
4. Chrome DevTools

Mi az a Google Chrome

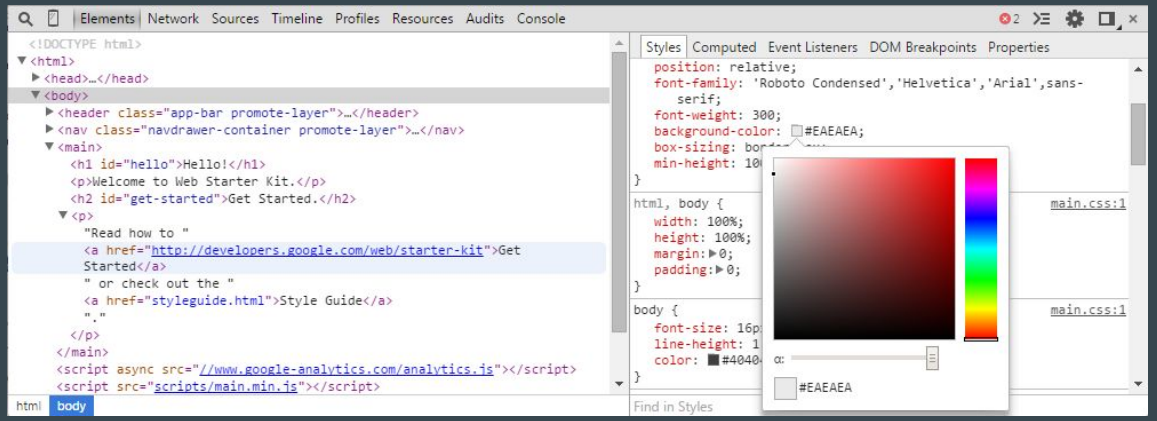
- Google böngészője
- Chromium open source
- WebKit layout engine
- 66% -a az asztali böngészőknek (2018)

Mi az a DevTools?

- Debugging kiegészítés



4. Chrome DevTools



- Elements: DOM
- Network: XMLHTTP (xhr)
- Sources: JavaScript
- Performance: benchmark
- Memory: dark magic
- Application: cookies
- Security: certificate, https
- Audits: 400.000 forint / nap (Lighthouse project,

https://developer.chrome.com/devtools

Köszönöm a figyelmet!