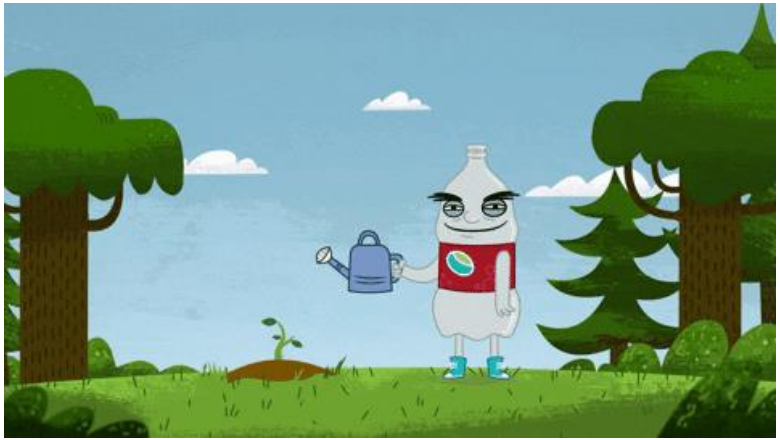

Gondolatok a **Microfrontendekről**...
egy **Angular**-os sapka alól

A probléma

A (JS alapú) Frontend alkalmazásunk:

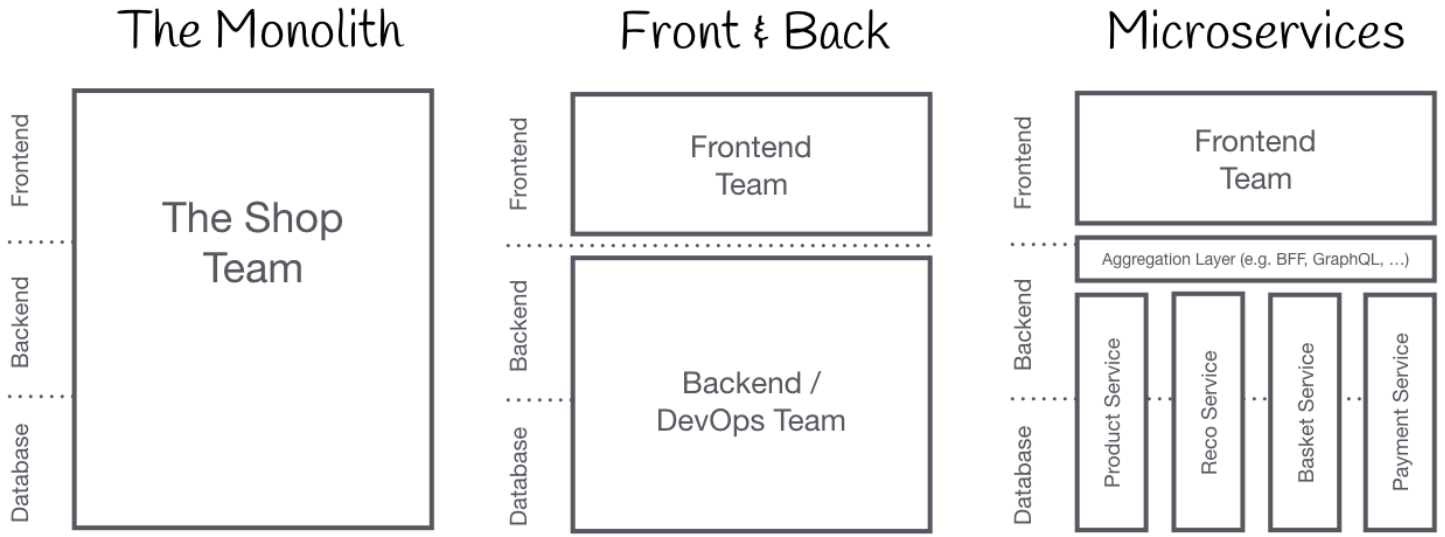
- **Komplex, sok funkciós**
- **Sokan dolgoznak rajta**
- **Több részegysége egymástól különállóan működik**



Hogyan lehetne kezelhető méreten tartani az **alkalmazást** és vele együtt a **kódbázist?**

“klasszikus” felosztások

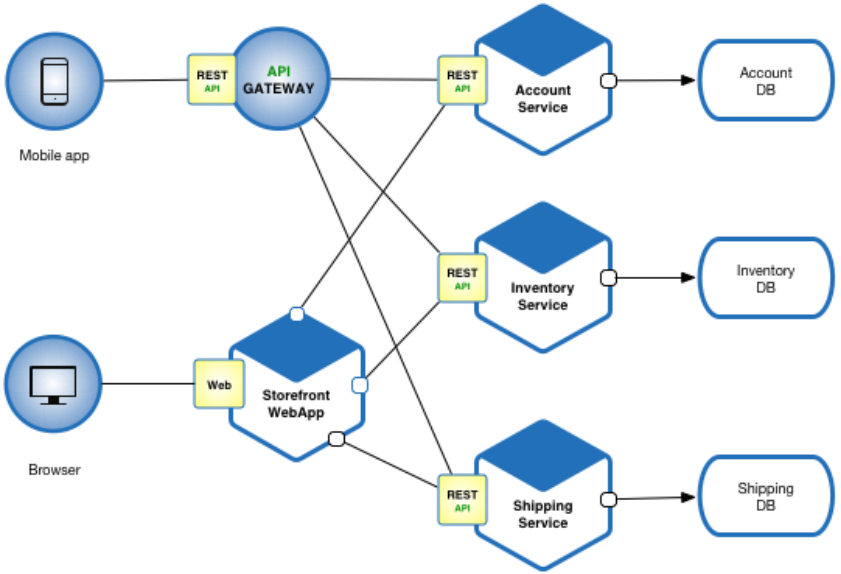
Figyeljük meg: a Frontend gyakran monolitikus marad…



Alap ötlet: Micro <bármi>

Microservices:

- Service-Oriented Architecture (SOA) egy alfaja
- Laza kapcsolatok, külön deploy-olható alkalmazások, egymástól leválasztott folyamatok, “aggregált” API mögött.



Miért jó?

- Könnyebben fejleszthető (és gyorsabban)
- Jobban skálázható
- Különállóan deploy-olható

Frontend szemszögből lehetőségek

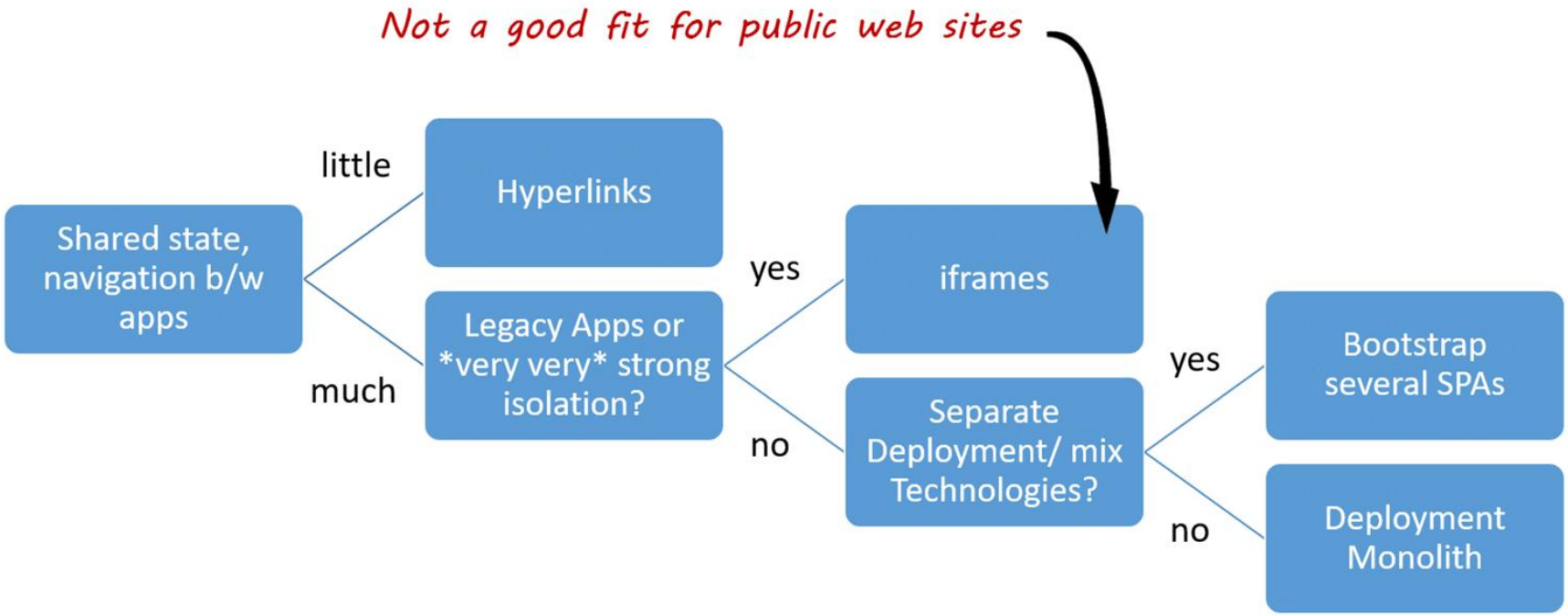
Több alkalmazást írunk.
Aztán őket valahogy összekötjük...

Nem túl összefüggő appok **Egyszerű site**

- **Csak átlinkelünk**
  url linkek a többi webapp-ra
- **iFrame beágyazás**
  minimális kommunikáció is lehetséges
- **Web komponensek**
  ~~specifikus interface alapú komm.~~
- **Library / Monorepo / Microfrontend**
  JS “térben” integrált megoldás

Nagyon összefüggő appok **Komplex site**

Mikor melyiket válasszuk?



Mi az a Micro Frontend?

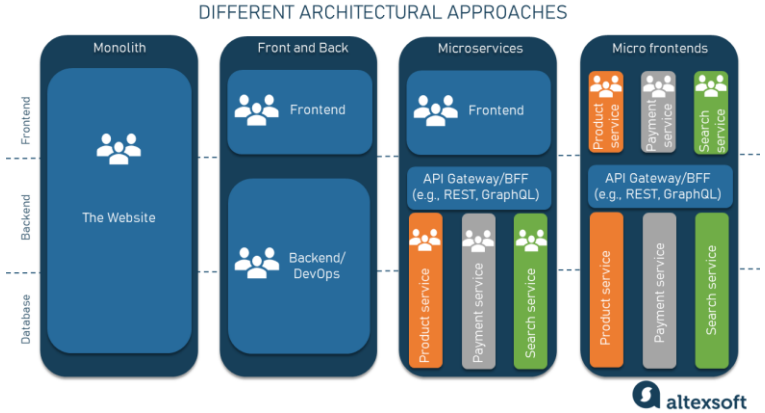
Microfrontend megoldások

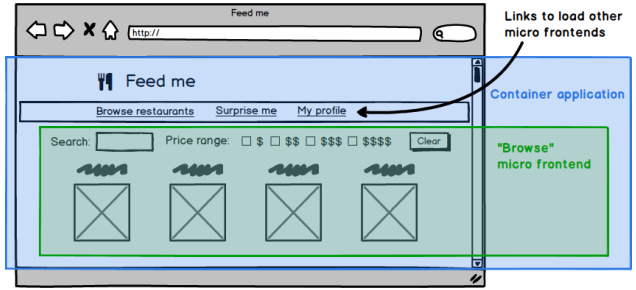
Micro frontend: csak egy architekturális minta. A konkrét implementációra többféle megközelítés létezik, **két fő aspektus** mellett:

1. **Program működése:**
- Natív JS közös kódtér
- Web Components
- Webpack Module federation
- Angular lazy loaded module

Fontos kísérő probléma:

1. **Kódszervezés**:
- Multi repo, Library vs. projekt kód
- Monorepository



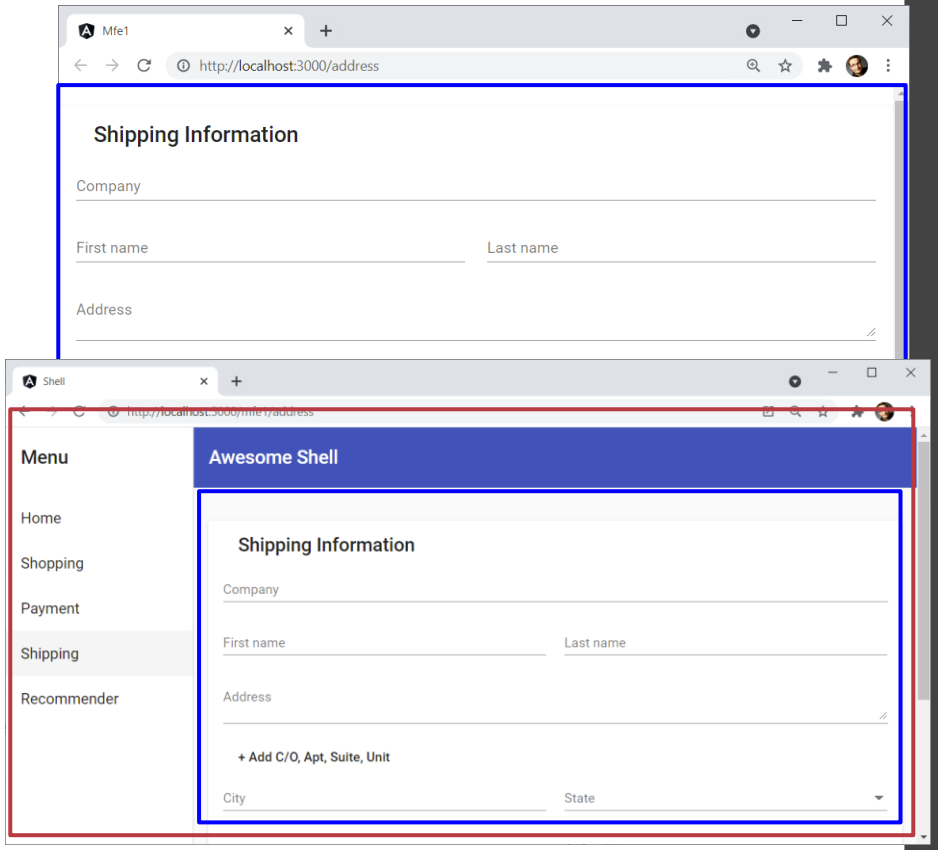


Module federation

Webpack által adott natív lehetőség microfrontend-re (framework függetlenül):

- Build time felkészítem az **külső** alkalmazást, hogy a **belsőt** dinamikusan fogja betölteni
- Működés közben URL hivatkozások segítségével töltöm be a remote appot

újra feltaláltuk az AJAX-ot? :)



Module federation

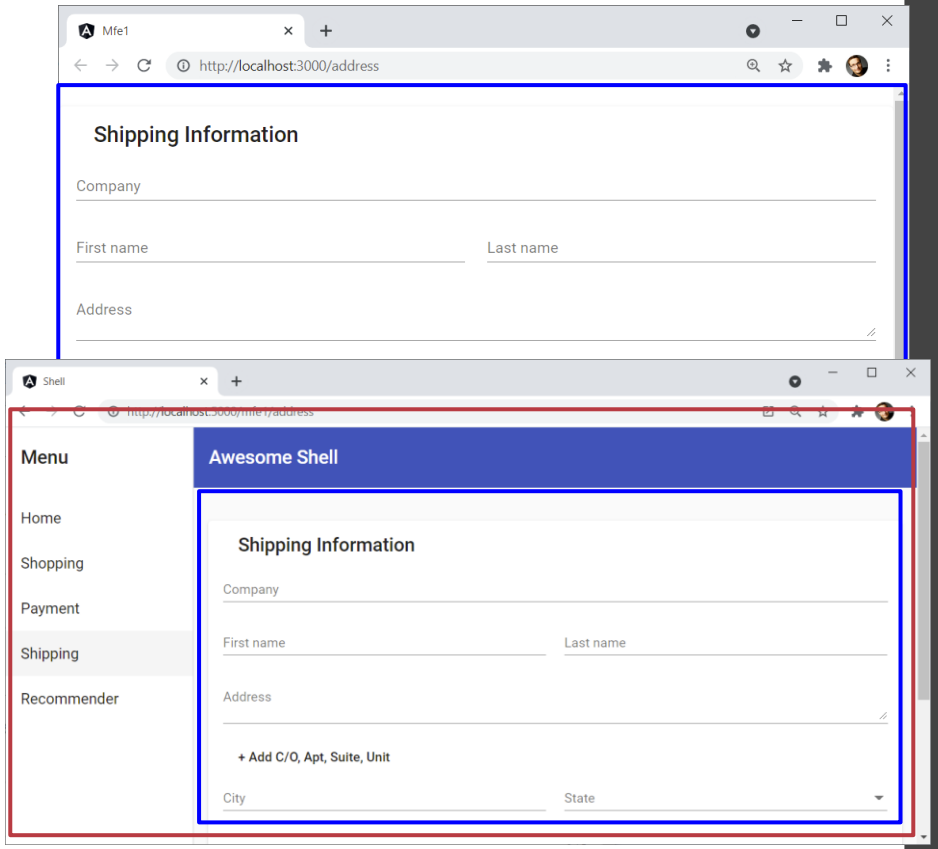
- Build time felkészítem az **külső** alkalmazást, hogy a **belsőt** dinamikusan fogja betölteni

Host: az aktuálisan futó app, általában shell-ként funkcionál

Remote: referencia a külső modul-ra

Module: EcmaScript Module (=== akár egy magányos JS fájl is)

Lényegében “lazy-load”-oljuk a remote modult.



Angular + Module Federation

Nagyon felületes bepillantás :)

Angular CLi generálja

- Mi konfigulhatjuk még

shared: megosztott npm module-ok (ez a host ugye)

- Csak egyszer loadol! (singleton, version)

```
// projects/mfe1/webpack.config.js

[...]
plugins: [

  new ModuleFederationPlugin({

      // For remotes (please adjust)
      name: "mfe1",
      filename: "remoteEntry.js",
      exposes: {
          './AddressModule': './projects/mfe1/src/app/address/address.module.ts',
      },

      shared: share({
        "@angular/core": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
        "@angular/common": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
        "@angular/common/http": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
        "@angular/router": { singleton: true, strictVersion: true, requiredVersion: 'auto' },
        "@angular/material": { singleton: true, strictVersion: true, requiredVersion: 'auto',
includeSecondaries: true },

        ...sharedMappings.getDescriptors()
      })

  }),
],
[...]
```

Angular + Module Federation

Angular module federation segéd lib

A microfrontend csak URL szinten van behivatkozva.

```
// projects/shell/src/app/app.module.ts

[...]
RouterModule.forRoot([
  {
    path: '',
    component: HomeComponent
  },
  {
    path: 'mfe1',
    loadChildren: () => loadRemoteModule({
      remoteEntry: 'http://localhost:3000/remoteEntry.js',
      remoteName: 'mfe1',
      exposedModule: './AddressModule',
    }).then(m => m.AddressModule)
  }
])
[...]
```

Microfrontend Pro - Cons:

Pro:

- Separation of concerns ++
- **Könnyebb tesztelés**
- **Könnyebb karbantarthatóság**
- **Gyorsabb / könnyebb deploy**
- **Ha nagy a team,** segít
- **Ha rendben van az architektúra** akkor valszeg jó választás :)

Cons:

- Könnyen **“visszabonyolódik”**
- **Extra komplexitás** egyébként is
- Nehezebb **kommunikáció**
- **Interface megosztás** (TS esetén)
- **Ha nagy a team,** hátrányok
- Csapatok között **komoly együttműködés** kell hozzá
- **Jelenleg is “tesztelik” hogy mennyire hatékony**

Monorepo

Mi az a “monorepo”?

Git stratégia

1. **Multi-repo**
   - Több projekt: több repository
1. **Mono-repo**
   - Csak egy repository
     - Könyvtárszerkezettel megoldjuk!
1. **(Monolithic)**
   - (amikor minden egyben van)

Monorepo pro - cons

Pro:

- **Könnyebb** elindítani a projektet
- **Jobb szinkronizáltság** a projektek között
- **Verziózás** egyértelműbb
- **Refaktorálás gyorsabb** projektek között
- Kód kultúra is **közös**

egyszerűsítés

Cons:

- **Nehezebb** deploy
- **Nagyobb méretű projekt**
- **Külön verziózni** nem nagyon lehet jól
- **Csapatok kényszerítve vannak** egymáshoz idomulásra
- Separation of concerns - -

nem elég rugalmas

Nx workspace

"a tool suite designed to architect, build and manage monorepos at any scale"

- Angular Core, NgRx team members
- **"Kiegészíti" a CLI-t, újabb eszközökre ad támogatást, könyvtárstruktúrát kínál**
- egyedi plugin-ek (Cross-platform, .NET, E2E testing etc.
- Dependencia gráf, Kódmegosztás, Dev szerver, Build optimalizálás, könyvtárstruktúra, kódgenerálás, CI/CD integráció... and many more

MicroFrontends vs. Nx Workspace

Nx Workspace:

- Monorepo kezelése
- Több projekt ugyanabban a repo-ban, és lehet köztük kapcsolat **kódbázis szinten**

Micro Frontends:

- Több Frontend alkalmazás egymásba ágyazva **futás közben**

Micro Frontend projekt lehet monorepo-ban, és poly-repoban is.

Az én véleményem

- **Ami kipróbált és hasznos: monorepo**
  - ökölszabályként kb. bárhol jó
- De Microfrontend még “tesztelés alatt áll”
  - komoly csapatok, alapos tervezés után
  - kisebb projekteknél overkill
- Jelenleg még **vendor lock-in**-t eredményez (pl. webpack)
- Kód szinten összekötni module federation segítségével JS programokat hasznos, de
  - **Angular** esetén ott a lazy load + Lib-ek készítése jó alternatíva (pláne most a lazy loaded component-ekkel)
- **Más framework-ök** esetén valszeg hamarabb, és több létjogosultsága van
  - Ez egy újabb mód, majd idővel kiderül, mennyire fog elterjedni

esbuild Vite

TURBOPACK

Köszönöm a figyelmet!

Kérdésed van? Keress bátran:

balazs.kiss@webvalto.hu

Források

Felhasznált források, inspirációt adó cikkek:

- Általános elemzés a microfrontendről: https://www.toptal.com/front-end/micro-frontends-strengths-benefits
- Egy nagyon alapos könyv a témában: https://livebook.manning.com/book/micro-frontends-in-action/chapter-1/
- Architecturális elemzés a microfrontend-ekről: https://martinfowler.com/articles/micro-frontends.html
- Egy nagyon jó Angular Microfrontendes könyv: https://www.angulararchitects.io/en/book
- Egy másik elemzés hasznos ábrákkal: https://www.altexsoft.com/blog/micro-frontend/
- Monorepo vs. Multi repo elemzés: https://kinsta.com/blog/monorepo-vs-multi-repo/
- Friss meetup a Module Federation jövőjéről: https://www.youtube.com/watch?v=3zOnJfTo8Gw
- Altenratíva az NxWorkspace-hez: https://single-spa.js.org/
- Angular Module Federation gyakori hibák https://www.angulararchitects.io/aktuelles/pitfalls-with-module-federation-and-angular/