- **TEORIE – Aplicații și servicii web**

O aplicație sau un serviciu de tip web reprezintă un program/o colecție de programe care utilizează un browser web pentru a putea realiza diverse acțiuni prin intermediul internetului. În strânsă legătură se află concepte precum: **protocolul HTTP, TCP/IP, client/server, request/response**; toate acestea fiind situate la baza site-urilor și aplicațiilor pe care le accesăm.

"Cele două părți ale web-ului":

- ***Server-Side/Back-End*** – acest segment are ca scop *manipularea datelor, obținerea, editarea și publicarea informațiilor noi într-o bază de date sau într-un sistem de colectare*. Putem folosi pentru construcția părții logice mai multe limbaje și framework-uri, printre care enumerăm: **C# (.NET, .NET Core), PHP (Laravel, Symfony), Ruby (Ruby on Rails), Python (Django, Flask), Node.js (Express, Meteor)**
- ***Client-Side/Front-End*** – cea de a doua componentă a aplicațiilor web este utilizată pentru a prezenta informațiile prelucrate utilizatorilor, în format crud/neformatat **(JSON/XML/text)** sau prin intermediul unei interfețe servite de browser **(HTML, CSS, JS).** În prezent sunt utilizate două tipuri majoritare de framework-uri/librării:
  - Design: **Bootstrap, Materialize, Foundation, etc.**
  - Funcționalități complexe: **React, Angular, Vue, etc.**

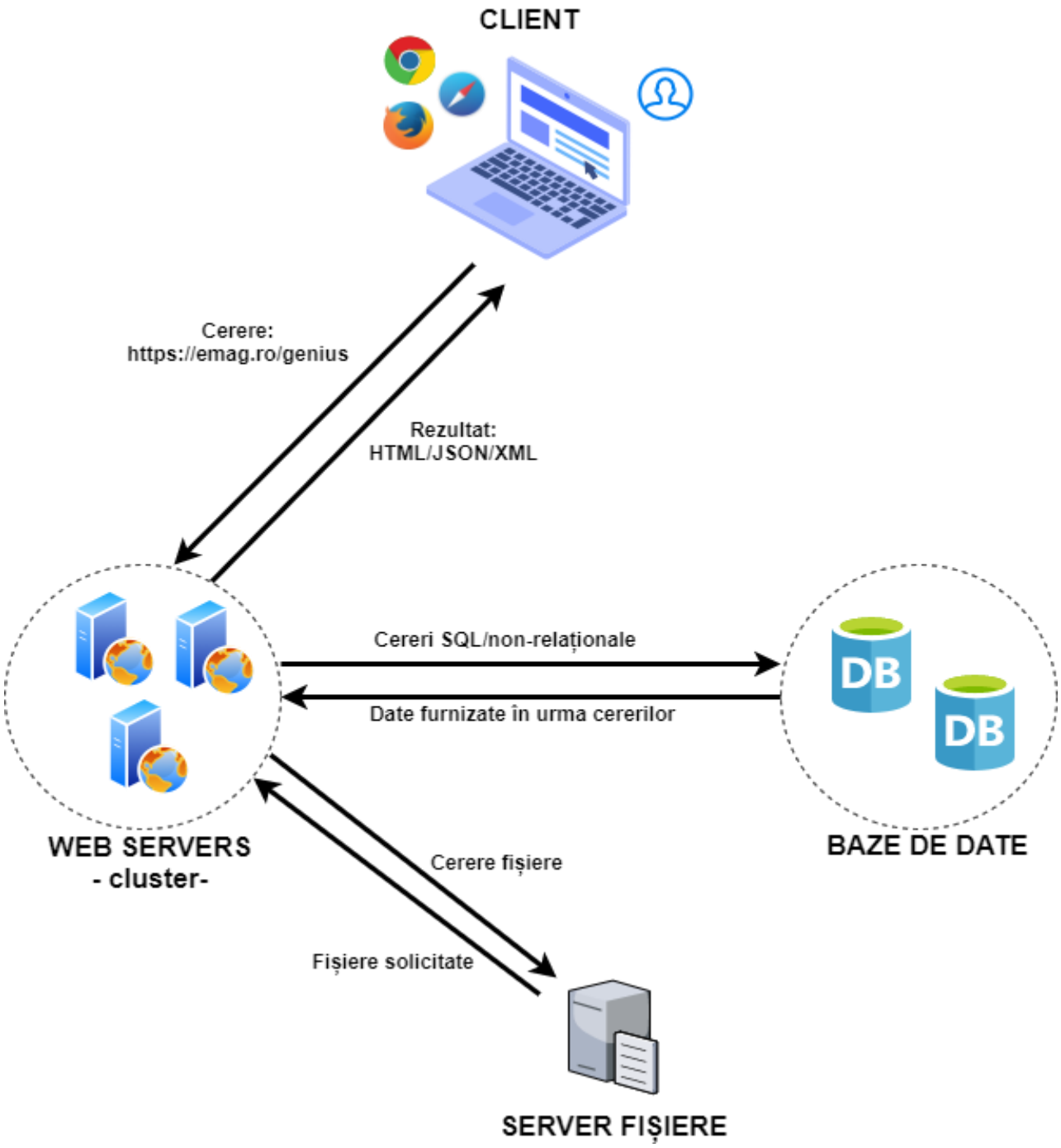
Modelul clasic al unei aplicații web:

1. Utilizatorii trimit **cereri (requests)** către un **server web**, prin intermediul internetului, folosind un **browser web**, o **interfață grafică** din cadrul unei aplicații sau un **API (Application Programming Interface).**

   [i] Vom reveni la conceptul de API într-un laborator ulterior.

2. **Server-ul web** execută cererea utilizatorului, în funcție de natura acesteia putând fi efectuate mai multe tipuri de acțiuni: interogarea unei baze de date, prelucrări simple sau complexe ale datelor introduse, etc.

3. Răspunsul este trimis înapoi utilizatorului sub formele enunțate în capitolul anterior: în mod grafic sau nu.



De ce utilizăm serviciile web? Care sunt avantajele acestora?

1. Cel mai benefic aspect este utilizarea oricărei aplicații web folosind un singur program: **browser-ul!** Astfel, ne este permisă accesarea internetului, conectarea cu alte persoane și efectuarea unor sarcini diverse fără a mai instala diverse produse software. Depindem exclusive de browser-ul utilizat pe sistemul propriu de operare.
2. Din punct de vedere al dezvoltării unui produs software, o aplicație web este mult mai ușor de distribuit consumatorilor, deoarece fiecare îmbunătățire sau modificare adusă la produs are loc pe server-ul web. Astfel, clienții noștri sunt la zi cu serviciile oferite, fără a le complica viața cu actualizări frecvente.
3. O aplicație servită prin intermediul unui browser ne scutește de problemele întâmpinate la dezvoltarea de programe specifice anumitor platforme: **telefoane mobile, tablete, etc.** Singurele modificări necesare se fac la nivel de design.

Există dezavantaje?

DA! 😢

- Minusul cel mai evident este dat de nevoia unei conexiuni la internet.

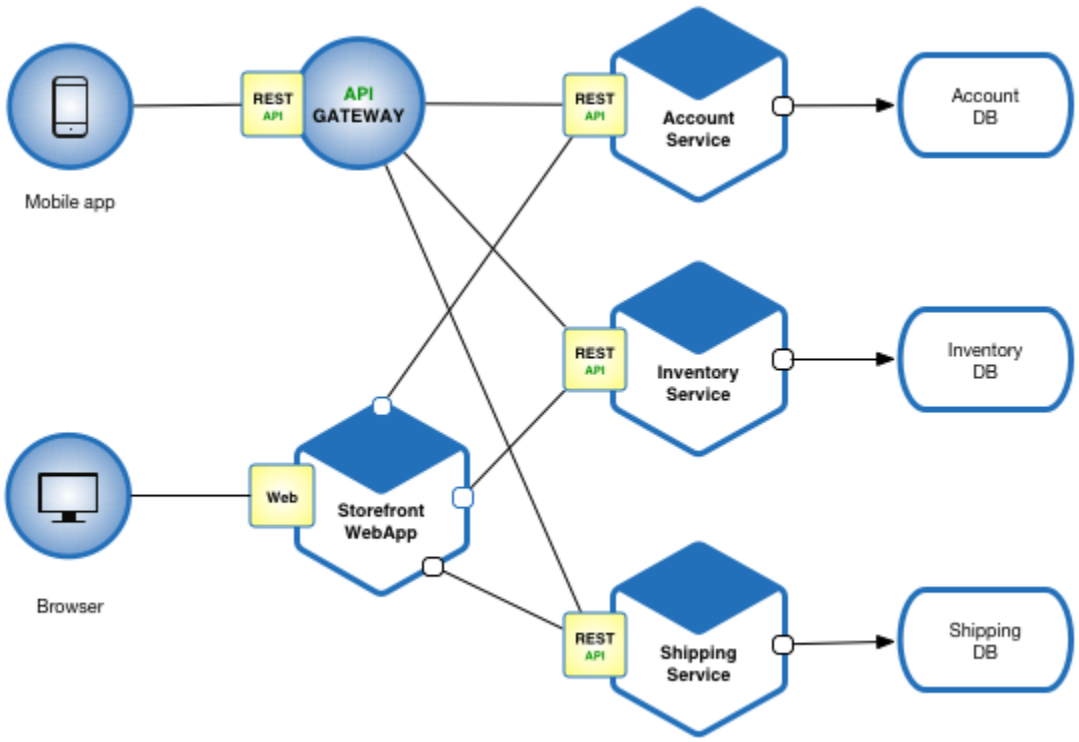
Tipuri de aplicații web:

1. **Aplicații clasice/tradiționale/statice**
    - Sunt folosite atunci când **cerințele sunt simple**, iar **fluxul de date este unul static** (nu avem nevoie de funcționalități complexe atunci când vine vorba de citirea sau scrierea informațiilor).
    - Site-ul sau serviciul realizat **nu** au nevoie de componente JavaScript complexe.
    - Echipa de dezvoltare **nu este familiară cu JavaScript sau TypeScript**, ci preferă un mod de lucru clasic (HTML + CSS).

2. **SPA (Single-Page Application)**
    - Utilizate atunci când produsul dezvoltat are o **funcționalități complexe** (servicii de plată, coșuri de cumpărături, aplicații de transport, etc.)
    - Expunem conexiuni complexe la diverse surse de date. Spre exemplu putem utiliza **emag.ro.** În cazul acesta vorbim despre produse expuse de multipli furnizori, avem integrări cu diverse companii de marketing și promovare, avem un serviciu dedicat pentru confirmarea și livrarea comenzilor, iar lista poate continua.
    - Scopul este oferirea unui **conținut dinamic** utilizatorilor (ex.: actualizarea în timp real a stocului). Dinamismul este obținut prin intermediul unor mecanisme complexe de comunicare între componente.
    - **Framework-uri** populare: **React, Vue, Angular, etc.**
    - Spre deosebire de aplicațiile tradiționale, acolo unde servim conținut static (HTML & CSS), în cazul acestor aplicații oferim **un singur cod**, scris în JavaScript sau TypeScript, care generează la rândul lui paginile din browser.

      🤔 Ce se întâmplă cu un site generat dinamic? Pot expuse ușor o astfel de aplicație? Dacă generez cod HTML din JavaScript nu îmi va fi afectată prezența online? **Citește mai multe despre *SPA & SEO!***

3. **Microservicii**
    - Arhitectura aceasta constă în **dezvoltarea mai multor unități de cod**, de **dimensiune mică**, numite **servicii.** Au scop unic și se îmbină între ele pentru a forma o aplicație mai mare.
    - **Avantaje:**
        i. **Nu este necesară dezvoltarea folosind același limbaj.**
        ii. **Nu sunt dependente în mod direct unele de altele**. Spre exemplu: atunci când încercăm să efectuăm o plată online microserviciul responsabil de plăți va solicita o verificare a datelor de autentificare pentru utilizator care solicită plata. Dacă serviciul de autentificare nu este functional, plata va fi refuzată și utilizatorul va primi un răspuns negative, un feedback, nu va rămâne blocat în pagină.
        iii. **Productivitate sporită**! 😁



Aplicație web folosind modelul de microservicii

4. **Serverless**
    - **Prezintă similarități cu arhitectura bazată pe microservicii, diferențele fiind la nivel superior.**
    - Dezvoltatorii lucrează cu o infrastructură situată în cloud, nefiind configurate mașini virtuale sau Docker containers.
    - **Avantajul principal:** orientare strictă asupra codului aplicației, fără a configura mediul de producție, respectiv fără a deține componente hardware.

Tips & Tricks:

- **Simplitatea** utilizării este cheia către un produs web de success!
- **Învățați arhitectură**, nu vă limitați la un limbaj de programare sau la un framework specific.
- Orice aplicație trebuie gândită în conformitate cu tendințele actuale, în cazul nostru: **mobile first!**

- **Consistența** codului scris este cheia către mai puține dureri de cap! Da, spațierea, culorile și felul în care scriem cod contează.
- **Evitați optimizările la început de proiect!** Orice fracțiune de secundă petrecută în minus de utilizatori aduce un plus de complexitate codului. Astfel, putem pierde săptămâni întregi cu optimizări lipsite de importanță, în detrimentului unui modul nou al aplicației.
- **K.Y.T. (Know your tools)!** Un singur IDE sau un singur framework **(bine învățate, evident)** vă vor asigura o productivitate sporită.
- **Nu scrieți cod până nu înțelegeți felul în care vreți să funcționeze aplicația!**
- Doi aliați importanți: **uneltele de debug** & **curățarea codului (refactor)**