Metode de Dezvoltare Software

Arhitectura modernă a sistemelor web de mari dimensiuni

De la Monolit la Microservicii & Micro Frontends

/thoughtworks

05.04.2022

Disclaimers

- Orice asemănare cu persoane, fapte, intamplari, sisteme software sau hardware reale este pur întâmplătoare.
- Ce urmeaza e doar un exercitiu de imaginatie din partea mea (si sper ca si a voastra), dar as dori sa il ancorez in realitate cumva...
- Imi cer scuze in avans pentru *romgleza* si o sa va rog sa ma intrerupeti cand nu intelegeti un termen si sa ma ajutati daca imi scapa varianta in limba romana.

Arhitectura software

"Architecture is about the important stuff. Whatever that is."

[Ralph Johnson]

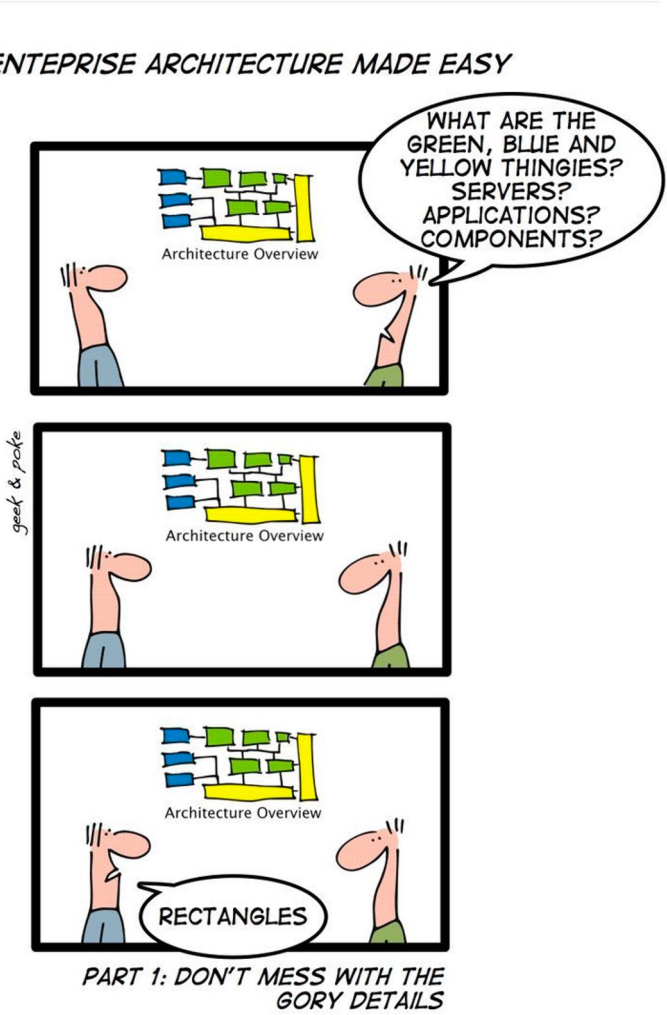
...are de-a face cu elementele fundamentale dintr-un sistem software si interactiunile dintre ele.

...are legatura cu luarea unor decizii structurale care odata implementate se pot schimba doar cu costuri semnificative.

...este intelegerea comuna ce o au dezvoltatorii experti dintr-un sistem.

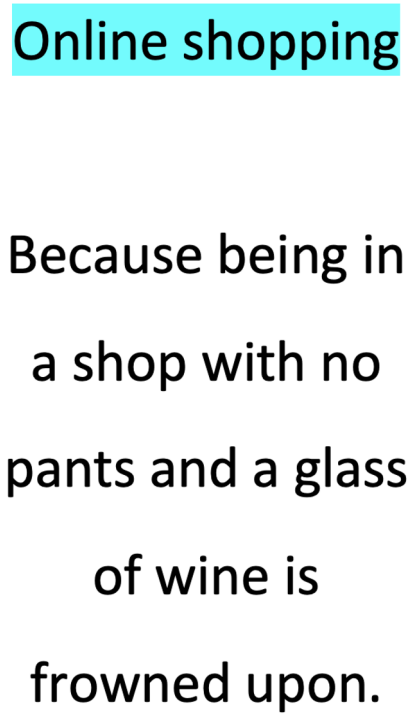
...este suma deciziilor ce ai dori sa fie bune la inceputul unui proiect.

Exemple de discutat: automotive, e-commerce, gaming



Introducere in poveste

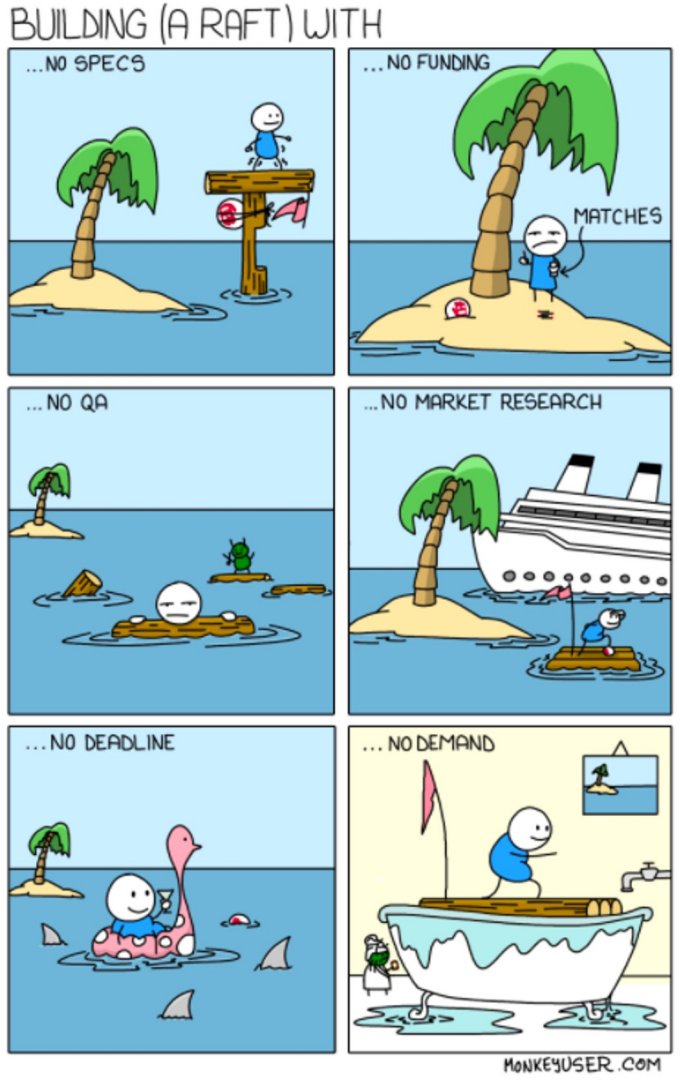
- Vreau sa construiesc un magazin online...



#0 Pregatiri

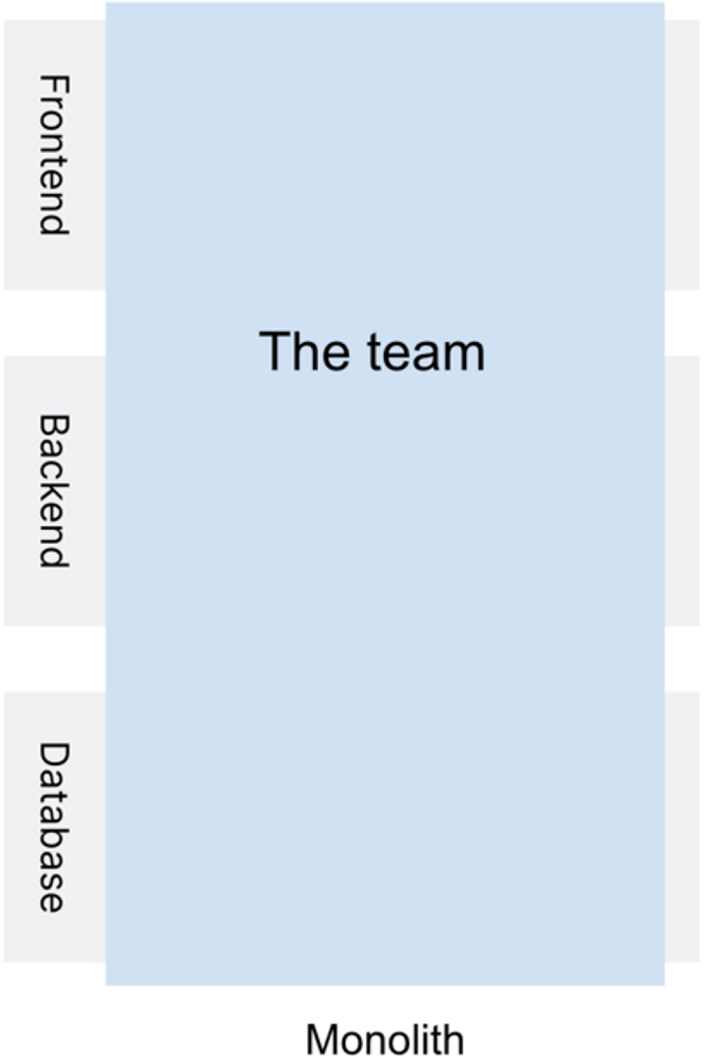
- Imi fac un plan de afaceri beton
- Adun un buget pentru o perioada suficient de indelungata
- Incep sa scriu specificatii (sau macar o viziune)
- Incropesc o echipa de 5-6 oameni care stiu sa faca de toate in ale software-ului

De ce 5-6 dezvoltatori si nu 10 sau 20? *(Agile discussion)*



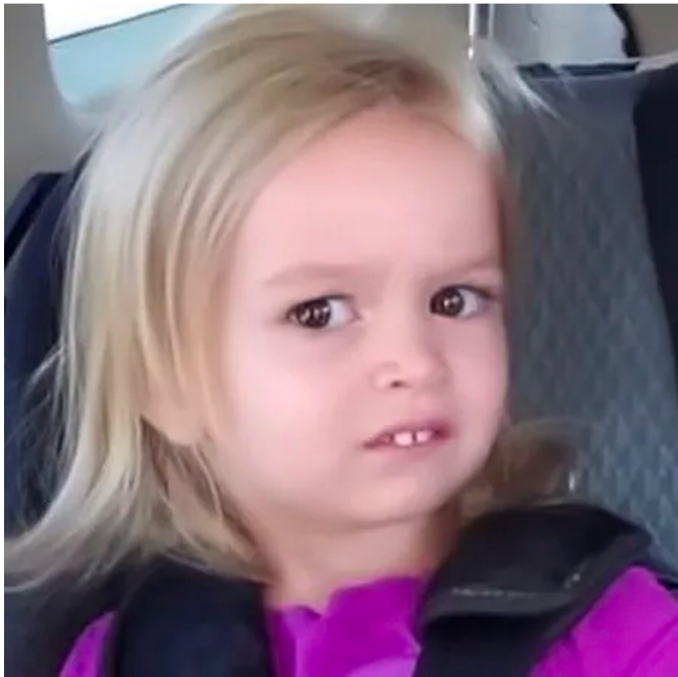
#1 Monolith

- Incepem dezvoltarea
- Construim o aplicatie web tipica, pe 3 straturi - frontend, backend, data - AKA *“three-tier architecture - presentation layer, logic/business layer, data layer”.*
- Toata lumea lucreaza peste tot, la ce se pricepe.
- Suntem in faza de *start-up* si produsul are succes
    - Echipa creste.
    - Se da mai putina importanta arhitecturii si mai multa feature-urilor.
    - Se lucreaza repede pentru a capitaliza pe succesul initial.
    - Testarea se face in mare masura manual.



#2 Front & Back

- Inca suntem in faza de start-up dar produsul are succes si echipa creste rapid.
- Incep insa sa apara primele probleme:
    - Numarul de clienti creste si ne permitem tot mai putine bug-uri si downtime-uri in productie.
    - Numarul de feature-uri creste si e tot mai greu sa ne asiguram ca nu stricam nimic din urma cand implementam ceva nou.
    - Echipa de dezvoltare creste si devine tot mai greu sa nu se calce pe picioare unii cu ceilalti.



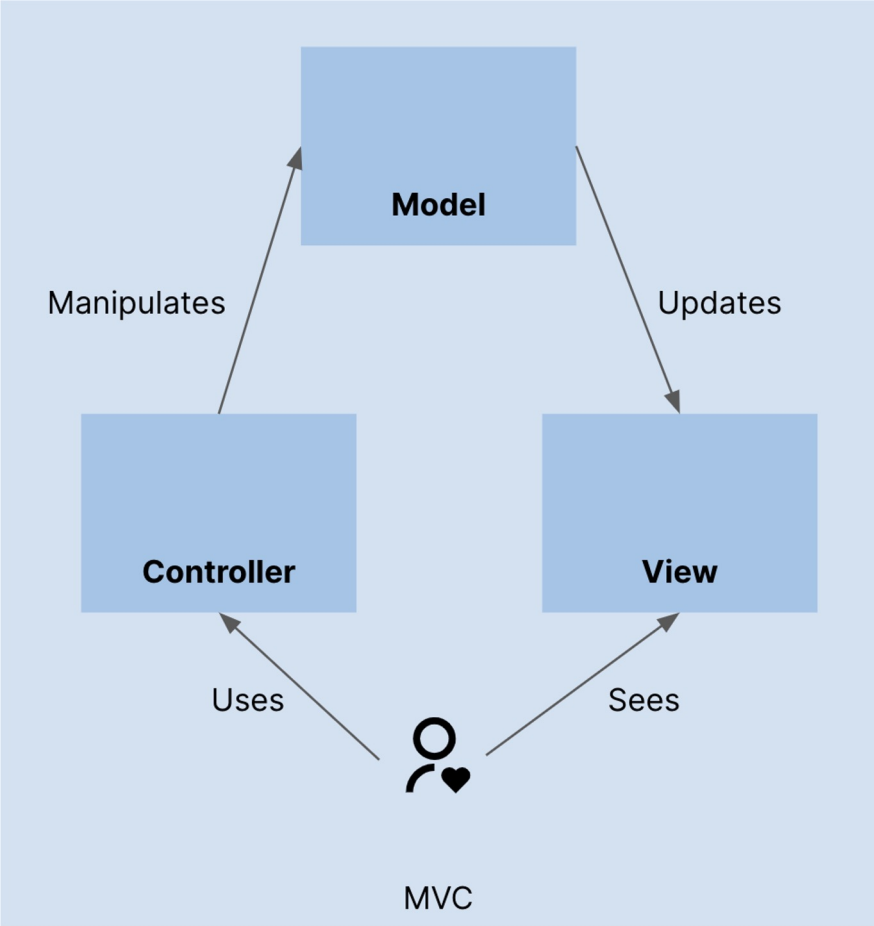
#2 Front & Back

- Incepem sa ne gandim tot mai mult la architectura si design pentru ca devine tot mai importanta pe masura ce produsul creste.
- Ne hotaram sa implementam una din arhitecturile web clasice:
    - MVC - Model View Controller
    - MVP - Model View Presenter
    - MVVM - Model View ViewModel

MVC Pattern

Model View Controller

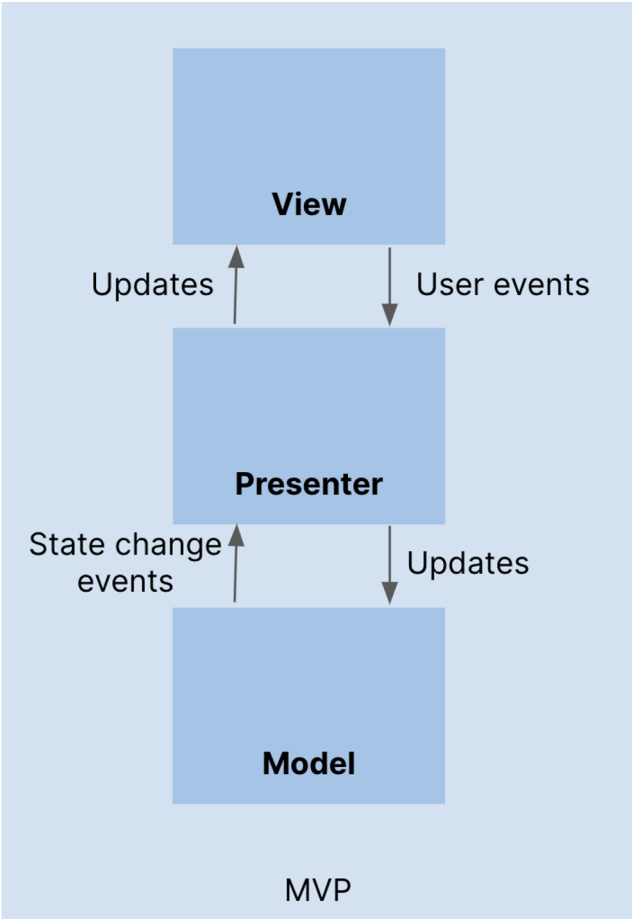
- Concept introdus in 1970
- Formata din trei parti:
    - *Model* - descrie logica de business si obiectele folosite (data)
    - *View* - componentele de UI
    - *Controller* - proceseaza inputurile si face legatura dintre *View* si *Model*. Coordoneaza procesul.
- Decupleaza responsabilitatile
    - Usureaza dezvoltarea in paralel, testarea, extinderea si mentenanta - creste viteza
    - View-uri multiple (web/mobile/etc)



MVP Pattern

Model View Presenter

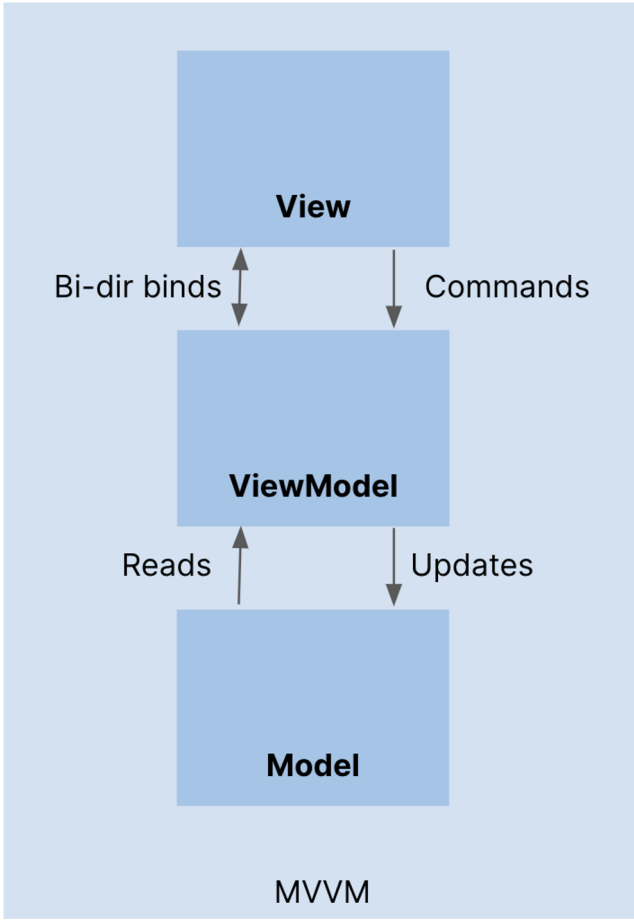
- Formata din trei parti:
    - *View* - componentele de UI. Coordoneaza procesul.
    - *Presenter* - proceseaza inputurile si face legatura dintre *View* si *Model*
    - *Model* - descrie logica de business si obiectele folosite (data)
- Asemanator cu MVC, dar *View*-centric
- Presenter-ul intermediaza legatura dintre *View* si *Model*
- Decupleaza responsabilitatile
    - *View*-ul e chiar mai bine deculpat de *Model* decat in MVC ceea ce face testarea si mai simpla



MVVM Pattern

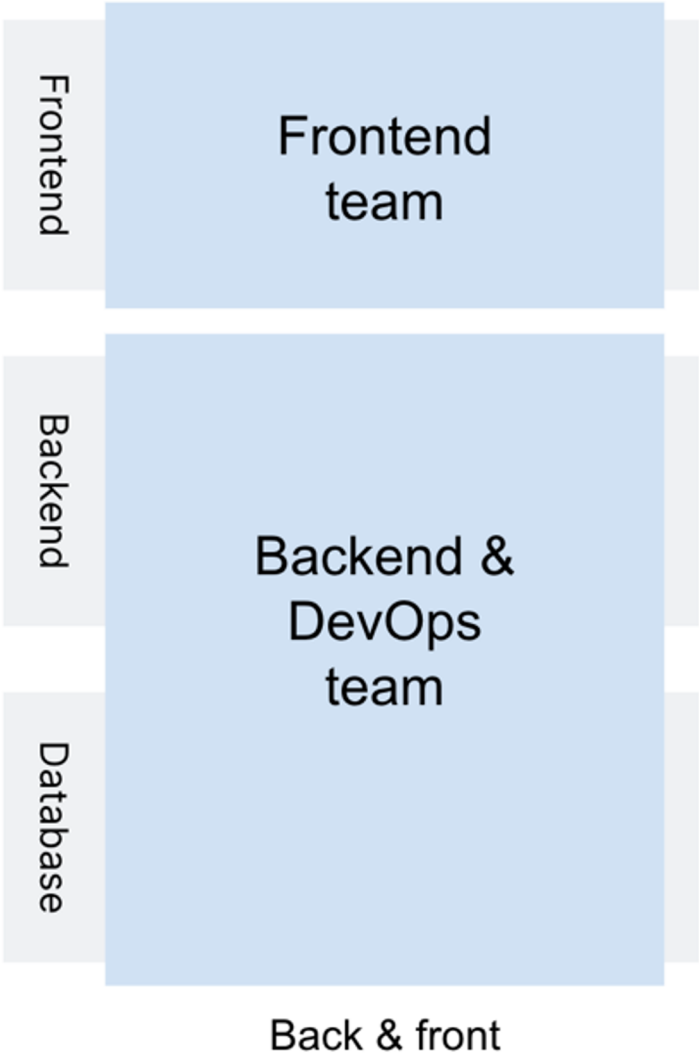
Model View ViewModel

- Formata din trei parti:
    - *View* - componentele de UI
    - *ViewModel* - abstractie a *View*-ului
    - *Model* - descrie logica de business si obiectele folosite (data)
- Legatura bidirectional (binding) intre *View* si *ViewModel*
- Decupleaza responsabilitatile
    - Usureaza dezvoltarea in paralel, testarea, extinderea si mentenanta



#2 Front & Back

- Incepem sa ne gandim tot mai mult la architectura pentru ca devine tot mai importanta pe masura ce produsul creste.
- Testarea devine tot mai importanta si incepem sa investim in testare automata si pipeline-uri de CI/CD (*Continuous Integration/Continuous Delivery*)
- Pe masura ce produsul si echipa creste, apare o segregare naturala intre dezvoltatori pe baza de afinitati si experienta. Unii migreaza spre frontend, altii spre backend. Pe termen lung, specializarea aduce viteza in dezvoltare.



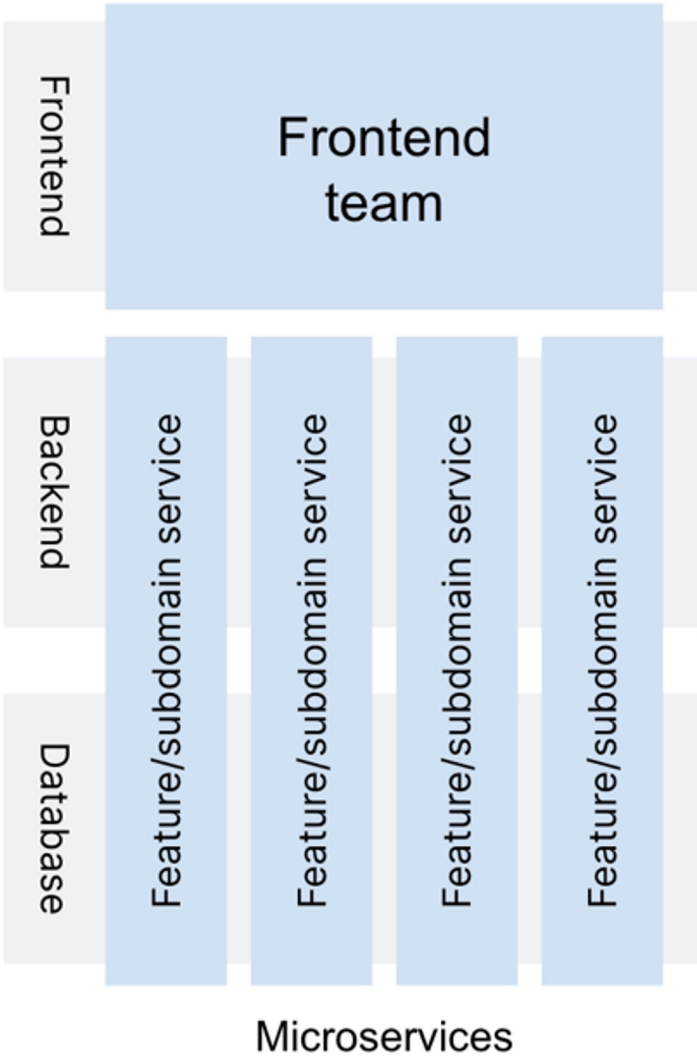
#3 Microservices

- Magazinul in continuare are mare succes, numarul si tipul de produse vandute si servicii creste si se diversifica, numarul de clienti si request-uri creste si el, echipa de dezvoltare trebuie sa creasca de asemenea.
- Apar noi challenge-uri:
    - Logica de business devine tot mai complexa si interactiunile dintre diferitele zone de cod tot mai complicate. Ca urmare, dezvoltatorii depind tot mai mult unii de altii, trebuie sa inteleaga un codebase extrem de mare si e tot mai greu sa adaugi oameni noi in echipa.
    - Devine tot mai greu sa modifici ceva fara sa strici altceva.
    - Testarea devine tot mai greoaie din cauza complexitatii.
    - Release-urile in productie se intampla tot mai rar pentru ca riscul de a strica ceva creste cu fiecare modificare. Reactivitatea la nevoile pietei scade si agilitatea de business se diminueaza.
    - Totul incetineste in afara de frustrare care creste si se accelereaza.

#3 Microservices

- Vrand-nevrand, echipele incep sa se specializeze - apare o echipa care se ocupa cu importul produselor in platforma, una care se ocupa de search, o alta care se ocupa de comenzi, etc.
- Aici e momentul cand arhitectul va veni si va introduce microserviciile.

“Separately deployed, fine grained, single purpose, encapsulating a business capability, exposing APIs, each containing its own data.”



#3 Microservices

Caracteristici

- Fiecare cu business-ul lui (cod + date)
- Evolueaza separat
    - Echipe autonome
    - Codebase separat
- Se pot pune in productie independent
- Tech agnostic (+discutie despre APIs)

#3 Microservices

Pros:

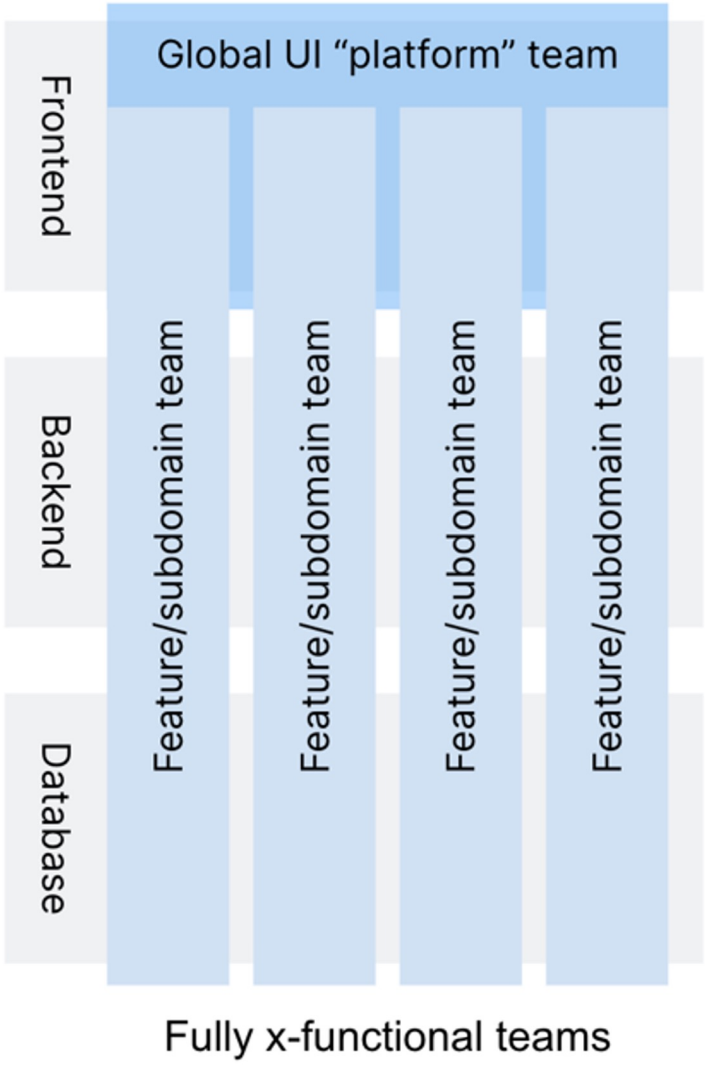
- Creste agilitatea - modular by design, echipe independente, incarcare cognitiva scazuta.
- Testare usoara si independenta.
- Punere in productie usoara si rapida - serviciile nu deping unele de altele.
- Diversitate tehnologica - fiecare serviciu poate fi implementat in alt limbaj (dar nu e indicat)
- Scalabilitate crescuta - atat ca numar de dezvoltatori cat si ca hardware necesar in productie.

Cons:

- Sistem distribuit - creste overhead-ul de comunicare intre servicii si sansele ca ceva sa nu functioneze (e.g. service down)
- *Eventual consistency* - modificari tranzactionale dificil de obtinut
- Complexitate operationala crescuta - numarul de entitati si *release*-ul lor in productie creste. E nevoie de o echipa de *DevOps* matura.

#4 Micro Frontends

- Tot mai mult din business logic se muta in browser, in frontend. Aplicatiile web devin tot mai responsive si interactive. Ca urmare, ceea ce nu de mult se intampla pe backend incepe sa se intample si pe frontend - dezvoltatorii se calca pe picioare intre ei.
- Pasul urmator este unul natural, de a aplica aceeasi tehnica care a fost aplicata la backend, dar la frontend prin impartirea UI-ului in componente care evolueaza independent, si alinierea acestora la microserviciile din backend.



#4 - Micro Frontends

- Conceptul e relativ nou - Thoughtworks l-a indentificat si consacrat in Nov 2016 in Thoughtworks Technology Radar
- *“An architectural style where independently deliverable frontend applications are composed into a greater whole.”*

Pros:

- Scalabilitate si viteza de dezvoltare
- Codebase mai mic
- Testare mai usoara
- Independent deployment
- Strategic vs Tactical focus
- Reutilizare
- Diversitate tehnologica
- Descentralizare

Cons:

- Lipsa de consistenta/consistenta trebuie intretinuta
- Complexitate crescuta
- Atentie la duplicare!
- Nestandardizat pentru moment (2022)

Q&A

Va multumesc!

doru.karacsonyi@thoughtworks.com

https://www.linkedin.com/in/doru-karacsonyi/