**UNIVERSITATEA BABEŞ-BOLYAI CLUJNAPOCA**

**FACULTATEA DE MATEMATICǍ ŞI**

**INFORMATICǍ**

**SPECIALIZAREA INFORMATICĂ – ROMANĂ**

**LUCRARE DE LICENŢĂ**

**Boardify**

**Conducător ştiinţific:**

**Dr. CZIBULA Istvan**

**Absolvent:**

**Antinie Radu**

**2020**

Cuprins

1. **Introducere**
   1. Motivatie
   2. Provocari
   3. Descrierea
2. **Implementari existente**
3. Microservicii

3.1 Introducere

3.2 Sabloane arhitecturale

3.3 Elemente definitorii ale unui microserviciu

3.4 De ce folosim microservicii?

1. **Deployment**

4.1 Service Discovery

4.2 Load balancing

4.3 Containere

4.4 Gateway service

4.5 Comunicare intre microservicii

1. Boardify

5.0 Specificatii

5.1 Analiza

* 1. Proiectare
  2. Implementare
  3. Testare
  4. Deployment
  5. Tehnologii folosite

1. Concluzie si dezvoltari viitoare
2. **Introducere**

Odata cu dezvoltarea tehnologiei interactiunea dintre oameni scade si astfel apare o nevoie a unor aplicatii care sa creeze legaturi in viata reala intre acestia.

Jocurile de masa (board games) au fost frecventate de fiecare din noi in copilarie dar multi oameni continua sa le joace pe tot parcursul vietii fiind o activitate pentru toate varstele.

**1.1 Motivatie**

Inainte ca tehnologia sa joace un rol atat de important in viata noastra aceste jocuri se aflau in casa fiecaruia dintre noi. Astazi exista in fiecare oras localuri speciale pentru pasionatii de aceste jocuri.

**1.2 Provocari**

Totusi cum relatiile dintre oameni au avut de suferit in ultimii ani, multi jucatori nu au cunostiinte interesate de aceste jocuri. Astfel, tehnologia ar putea juca un rol important in a pune in legatura aceste personae cu scopul de a se intalni intr-un local si a experimenta impreuna placerea jocurilor de masa.

**1.3 Descriere**

O solutie prin care tehnologia ar putea rezolva problema lipsei de interactiune intre persoare ar fi creearea unei platforme unde pasionatii de board games pot fi potriviti. Acesta ar fi un punct de plecare urmand ca acestia sa se intalneasca intr-un local.

Aceasta lucrare este organizata dupa cum urmeaza:

* Capitolul 1 serveste drept o scurta introducere.
* Capitolul 2 prezinta o sinteza teoretica a

arhitecturii bazate pe microservicii cu anumite proprietati ale acesteia dar si alternative.

* Capitolul 3 surprinde procesul de deployment.
* Capitolul 4 reprezinta partea practica a aplicatiei si explica modul in care aceasta a fost implementata.
* Capitolul 5 isi propune sa sumarizeze punctele importante surprinse in lucrare si posibilele dezvoltari viitoare ale aplicatiei.

**1.3 Implementari existente**

**Tabeltopia**

**Tabeltopia** este o platforma digitala cu acces instant la sute de jocuri de masa. Platforma permite sa creezi o camera de joc in care alti jucatori se pot alatura si odata ce numarul minim de jucatori a fost atins poti incepe jocul.

Tabeltopia a fost conceput pentru a juca jocurile de masa online si nu pentru a permite jucatorilor sa se intalneasca in viata reala.

Tabeltopia are o lista cu camerele de joc active:



Jucatorul poate alege o camera de joc si sa se alature acesteia apasand butonul “Join”:



Tabeltopia permite mai multe tipuri de conturi. Versiunea gratis permite unui juctor sa se alature simultan in doar 2 jocuri. De asemenea exista anumite jocuri premium in care doar jucatori ce au un cont Silver sau Gold pot intra.  
Pentru a avea un astfel de cont jucatorii trebuie sa plateasca o subscriptie lunara:

In menu-ul “All games” un jucator poate vedea toate jocurile si aplica diverse filtre pe acestea:



De asemenea poate alege un joc iar apoi sa creeze o camera de joc pentru acesta:



1. **Microservicii**

**3.1 Introducere**

Microservicile sunt un sablon architectural pentru software ce devine foarte popular in zilele noastre multumita multor avantaje precum flexibilitate si usurinta in scalare. Impartindu-le in echipe mici intr-o firma ofera multa eficienta in dezvoltare. Totusi, pentru a alege daca sa folosesti sau nu microservicii nu este   
suficient. Este nevoie sa stii ce provocari ai sa intampini si cum poti solutiona.

**3.2 Sabloane arhitecturale**

Inainte ca conceptul de microservicii sa evolueze majoritatea aplicatilor bazate pe web erau construite folosind un sablon arhitectural monolitic. Intr-o arhitectura monolitica o aplicatie este livrata ca un singur artefact software livrabil. UI-ul(interfata cu utilizatorul), business logic-ul si logica accessului la baza de date sunt impachetate intr-un singur artefact si livrate pe serverul aplicatiei.  
 Desi o aplicatie poate fi livrata ca o singura unitate de lucru de cele mai multe ori exista mai multe echipe de dezvoltare ce lucreaza la aplicatie. Fiecare echipa de dezvoltare are o anumita parte din aplicatie pentru care este responsabila. O problema este faptul ca pe parcursul dezvoltarii aplicatiei monolitice dimensiunea si complexitatea creste dar comunicarea si costurile de coordonare intre echipe nu scaleaza. Atunci cand o echipa are nevoie sa faca o modificare intreaga aplicatie are nevoie sa fie reconstruita, retestata si relivrata.  
 Conceptul de microservicii a aparut in vizorul comunitatii software in jurul anului 2014 si a fost un raspuns direct la provocarile de a scala atat din punct de vedere tehnic cat si organizatoric aplicatiile monolitice. Un microserviciu este un sistem distribuit slab cuplat. Microserviciile permit descompunerea aplicatilor mari in componente mai mici, usor de gestionat, cu responsabilitati restranse. Microservicile ajuta sa combatem problemele traditionale ale complexitatii unei baze de cod mari, descompunand baza mare de cod in bucati mai mici, bine definite. Conceptul cheie ce trebuie inteles despre microservicii este descompunerea functionalitatilor aplicatiei astfel incat acestea sunt complet independente una fata de celelalte.

**3.3 Elemente definitorii ale unui microserviciu** O arhitectura bazata pe microservicii are urmatoarele caracteristici:

* logica aplicatiei este descompusa in componente slab granulate, cu limite de responsabilitate bine definite ce sunt coordonate pentru a livra o solutile;
* fiecare componenta are un domeniu mic de responsabilitate si este deployed complet independent fata de celelalte; microservicile ar trebuie sa aibe responsabiliteatea pentru o singura parte din domeniu; de asemenea, un microserviciu trebuie sa fie reutilizabil in cadrul mai multor aplicatii
* comunicarea microservicilor se bazeaza pe niste principii de baza si adopta protocolare de comunicare usoare(lightweight) ca HTTP si JSON (JavaScript Object Notation) pentru a schimba date intre consumator si producator;
* implementarea tehniac de baza pentru serviciu este irelevanta pentru ca aplicatile mereu comunica folosind un protocol neutru de comunicare (JSON in cele mai multe cazuri); acest lucru inseamna ca o aplicatie dezvoltata folosind microservicii poate fi construita folosind multiple limbaje de programare si tehnologii;
* microservicile prin natura lor: mici, independente si distribuite – ofera organizatilor sa aibe echipe mici de dezvoltare cu domenii bine definite de responsabilitate; echipele lucreaza impreuna cu scopul de a livra aplicatia dar fiecare echipa este responsabila doar pentru microserviciul pe care lucreaza;

Folosirea microserviciilor ofera o flexibilitate mare si autonomie pentru echipele de dezvoltare dar echipele descopera faptul ca natura mica si independenta a microservicile le fac deployable to the cloud. Odata ce servicile sunt in cloud dimensiunea lor mica face usoara pornirea unui numar mare a aceluiasi serviciu si astfel aplicatia devine usor scalabila.

**3.4 De ce folosim microservicii** Comaniile ce aveau ca grup tinta comunitatea locala au inceput sa realizeze faptul ca pot ajunge la piata globla folosind internetul. Dar odata cu o piata mai mare competitia creste. Acest nivel crescut de competivitate face ca dezvoltatorii sa trebuiasca sa se gandesc la urmatoarele aspecte atunci cand construiresc o aplicatie:

* complexitatea mult crescuta: aplicatile ce folosesc o singura baza de date si nu integreaza alte aplicatii nu mai sunt norma; aplicatile din ziua de azi trebuie sa comunice cu multiple servici si sa foloseasca multiple baze de date nu doar in cadrul companiei dar si in cadrul internetului;
* clientii vor livrari mai rapide: clientii nu mai vor sa astepte pentru lansari anuale ale unui pachet software; acestia se asteapta ca functionalitatile sa fie descompune astfel incat noile lansari sa poata fi realizate in cateva saptamani fara nevoia unei lansari a intregului produs software;
* performanta si scalabilitate: aplicatiile globle fac extrem de dificil de prezis cat de mult volum trebuie gestionat de o aplicatie si cand acesta va aparea; aplicatile trebuie sa scaleze rapid peste multe servere cand nevoia apare sau dispare;
* clienti se asteapta ca aplicatile lor sa fie mereu disponibile: esuarea intr-o parte a aplicatiei nu ar trebui sa determine esuarea intregii aplicatii;

Pentru a indeplinii aceste astepari si a construi aplicatii ce pot scala mult, ca dezvoltatori trebuie sa descompunem aplicatia in servicii mai mici ce pot fi dezvoltate si lansate(deployed) indepenendent unul fata de celalalte. Daca descompunem aplicatia in servicii mai mici si le mutam dintr-un singur artefact monolitic putem construi sisteme ce sunt:

* flexibile – servicile decuplate pot fi compuse si rearanjate pentru a oferi noi functionalitati; cu cat unitatea de cod pe care o functionalitate este adaugata cu atat este mai putin complicat sa schimbam codul si dureaza mai putin sa testam si sa facem deploy la cod;
* de incredere – decuplarea servicilor unei aplicatii implica faptul ca atunci cand o degradare are loc intr-o parte a aplicatiei aceasta nu determina intreaga aplicatie sa esueze; esuarile pot fi descoperite si izolate intr-o parte aplicatiei fara a determina esuarea intregii aplicatii; acest lucru permite serviciului sa se degradeze dintr-o eroare irecuperabila;
* scalabilitate – servicile decuplate pot fi distribuite cu usurinta in cadrul mai multor servere, facand scalarea functionalitatilor ca atare; in cadrul unei arhitecturi monolitice unde toate logica intregii aplicatii este in acelasi modul, intreaga aplicatie trebuie sa scaleze chiar daca doar o mica parte aplicatiei o impune; scalare unor servici mai mici este mult mai eficienta din punct de vedere a costurilor;