UNIVERSITATEA POLITEHNICA BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL CALCULATOARE

|  |  |
| --- | --- |
| upb | cs |

PROIECT DE DIPLOMĂ

Get Together

O unealtă socială pentru interacțiuni reale

Radu-Ovidiu Catrangiu

**Coordonator științific:**

Conf. dr. ing. Alexandru Boicea

BUCUREŞTI

2019

UNIVERSITY POLITEHNICA OF BUCHAREST

FACULTY OF AUTOMATIC CONTROL AND COMPUTERS

COMPUTER SCIENCE DEPARTMENT

|  |  |
| --- | --- |
| upb | cs |

DIPLOMA PROJECT

Get Together

A social tool for real world interactions

Radu-Ovidiu Catrangiu

**Thesis advisor:**

Conf. dr. ing. Alexandru Boicea

BUCHAREST

2019

**CUPRINS**

[Sinopsis 2](#_Toc11881517)

[Abstract 2](#_Toc11881518)

[Introducere 3](#_Toc11881519)

[1.1 Context 3](#_Toc11881520)

[1.2 Problema 3](#_Toc11881521)

[1.3 Obiective 3](#_Toc11881522)

[1.4 Structura lucrării 4](#_Toc11881523)

[2 Analiza și specificarea cerințelor 5](#_Toc11881524)

[3 Studiu de piață / Abordări existente 6](#_Toc11881525)

[4 Soluția propusă 7](#_Toc11881526)

[4.1 Limbajul de programare 7](#_Toc11881527)

[4.2 Mediul de rulare și dezvoltare pentru server 8](#_Toc11881528)

[4.3 Baza de date 9](#_Toc11881529)

[4.4 Arhitectura sistemului de back-end 14](#_Toc11881530)

[4.5 Platforma de lansare (deploy) a microserviciilor 15](#_Toc11881531)

[4.6 Sistem de orchestrare a microserviciilor 17](#_Toc11881532)

[4.7 Tehnologie pentru dezvoltare front-end 20](#_Toc11881533)

[4.8 Tehnologie pentru stilizare front-end 21](#_Toc11881534)

[4.9 Alte tehnologii folosite în back-end 22](#_Toc11881535)

[4.9.1 Mosquitto / MQTT. 22](#_Toc11881536)

[4.9.2 Redis for tokens caching 22](#_Toc11881537)

[4.9.3 WebSockets. Socket.io 23](#_Toc11881538)

[5 Detalii de implementare 24](#_Toc11881539)

[6 Studiu de caz / Evaluarea rezultatelor 25](#_Toc11881540)

[7 Concluzii 26](#_Toc11881541)

[8 Bibliografie 27](#_Toc11881542)

[9 Anexe 28](#_Toc11881543)

# Sinopsis

TODO:

Sinopsisul proiectului are rol de introducere, conținând atât o descriere pe scurt a problemei abordate cât și o enumerare sumară a rezultatelor și a concluziilor. Se recomandă ca sinopsisul să fie redactat într-un limbaj accesibil unei persoane nefamiliarizate cu domeniul, dar în același timp destul de specific pentru a oferi rapid o vedere de ansamblu asupra proiectului prezentat.

Sinopsisul proiectului va fi redactat atât în română cât și în engleză. Ca dimensiunea recomandată aceasta secțiune va avea maxim 200 de cuvinte pentru fiecare variantă. Împreună, ambele variante se vor încadra într-o singură pagină.

# Abstract

TODO:

The abstract has an introductory role and should engulf both a brief description of the issue at hand, as well as an overview of the obtained results and conclusions. The abstract should be formulated such that even somebody that is unfamiliar with the projects’ domain can grasp the objectives of the thesis while, at the same time, retaining a specificity level offering a bird’s eye view of the project.

The projects’ abstract will be elaborated in both Romanian and English. The recommended size for this section is limited to 200 words for each version. Together, both versions will fit in one page.

# Introducere

## Context

TODO:

Get Together este simultan un produs menit să intre pe piața platformelor sociale, dar și o demonstrație de utilizare a tehnologiilor noi, în așa manieră încât să ușureze munca dezvoltatorilor atunci când vine vorba despre mentenanță și adăugarea de noi funcționalități.

## Problema

Get Together încearcă să rezolve două probleme. O problemă orientată către utilizatorii aplicației, respectiv o problemă orientată către dezvoltatorii acesteia.

Prima problemă, cea socială, constă în faptul că, după o vârstă mulți oameni nu mai au destul timp să planifice ieșiri în oraș cu prietenii. De multe ori apar probleme de sincronizare din cauza programului de zi cu zi al fiecăruia și din cauza locației la care se află fiecare într-un anumit moment al zilei, iar timpul de decizie asupra unui loc de întâlnire devine din ce în ce mai mare pe măsură ce se alătură grupului mai mulți oameni. Get Together își propune să ofere o soluție pentru a evita aceste probleme legate de planificare.

A doua problemă este una de inginerie, legată de implementare. Se dorește ca proiectul să capete dimensiuni mari, cu alte cuvinte să fie folosit de cât mai multă lume, deci este necesară o arhitectură ce poate susține un volum foarte mare de cereri. În timpul dezvoltării inițiale trebuie ținut în vedere faptul că un asemenea proiect se poate schimba după lansare atât în funcție de cererile utilizatorilor cât și în funcție de alți factori, atât interni cât și externi.

## Obiective

Se încearcă rezolvarea problemei sociale prin dezvoltarea unei aplicații web, cu un design atractiv, în care utilizatorii, după crearea contului, se pot alătura unui grup unde își pot raporta, pentru fiecare zi a săptămânii, intervalul orar la care ar putea fi disponibili să se întâlnească, împreună cu locația de unde vor pleca spre întâlnire. După raportarea locațiilor un server va compila toate datele legate de locațiile utilizatorilor din grupul respectiv și va sugera un număr de localuri la care s-ar putea întâlni, împreună cu un interval orar la care poate avea loc întâlnirea. Creatorul grupului poate seta tipul locațiilor ce pot fi sugerate, având la dispoziție trei variante: cafenele, baruri sau restaurante. De asemenea, aplicația trebuie să ofere posibilitatea utilizatorilor de a comunica rapid în cadrul platformei, deci va fi implementat și un sistem de comentarii în cadrul unui grup. Pentru a se putea distinge rapid de ceilalți utilizatori în secțiunea de comentarii, un utilizator are posibilitatea de a-și încărca propria imagine de profil. Un utilizator poate fi membru a mai multor grupuri, iar pentru a nu fi nevoit să verifice fiecare grup în parte pentru a afla ultimele noutăți va fi implementat un sistem de notificări în timp real.

Pentru rezolvarea problemei de inginerie trebuie luată în calcul o arhitectură modulară, care să poată scala ușor, dar care totodată să țină costurile cât mai joase. De asemenea, se dorește dezvoltarea cât mai ușoară a componentelor necesare funcționării aplicației, astfel încât adăugarea de noi funcționalități să nu genereze probleme. Pentru realizarea acestor obiective se optează pentru o arhitectură a back-end-ului bazată pe microservicii. O astfel de arhitectură permite împărțirea aplicației pe mai multe componente independente, ce pot rula pe mașini diferite. Principalele avantaje al acestei abordări sunt posibilitatea de scalare a resurselor prin lansarea aceleiași componente pe mai multe mașini și redundanța, posibilitatea ca o componentă să răspundă la cererile altei componente identice care a întâmpinat o eroare și care trebuie să treacă prin etapele unei recuperări. Un alt avantaj îl prezintă flexibilitatea în dezvoltare. Atunci când sunt adăugate noi funcționalități unei componente sau când este adăugată o componentă nouă, nu este afectată întreaga aplicație, ci doar părți care au o oarecare legătură între ele.

## Structura lucrării

TODO: Un paragraf în care fiecare dintre secțiunile următoare este prezentată în 1-2 fraze, punând accentul pe elementele cele mai semnificative din fiecare secțiune.

# Analiza și specificarea cerințelor

TODO:

# Studiu de piață / Abordări existente

În prezent, aplicațiile și platformele de socializare sunt centrate mai mult pe utilizator ca individ. Acestea oferă posibilitatea de a încărca și posta imagini, de a comenta la respectivele imagini și de a partaja informații precum articole sau alte postări de la alte persoane. Facebook, de exemplu, oferă și posibilitatea de a crea un grup în care să-ți inviți prietenii, însă în cadrul grupului utilizatorii sunt încurajați să facă aceleași lucruri ca în spațiul public al platformei, astfel încurajând utilizatorii să petreacă cât mai mult timp în cadrul platformei, singura diferență fiind faptul că în cadrul grupului conținutul este accesibil numai membrilor. Scopul final al aplicațiilor existente, în afară de cel expus de brand, acela de a aduce împreună utilizatorii din toate colțurile lumii, este de fapt acela de a vinde spațiu pentru publicitate. De aceea, platforme precum Facebook sau Instagram îndeamnă utilizatorii să petreacă cât mai mult timp în fața unui ecran de calculator sau de telefon, pentru că traficul suplimentar justifică un preț mai mare pentru spațiul publicitar digital.

Get Together țintește să întoarcă utilizatorul dinspre mediul virtual către mediul real, ușurând planificarea întâlnirilor, stimulând astfel activitățile reale de grup. În prezent nu există o soluție, o unealtă, unde utilizatorii să își introducă date despre programul lor de zi cu zi, împreună cu locațiile aferente programului, și să primească sugestii de locații și intervale orare în care s-ar putea întâlni.

TODO:

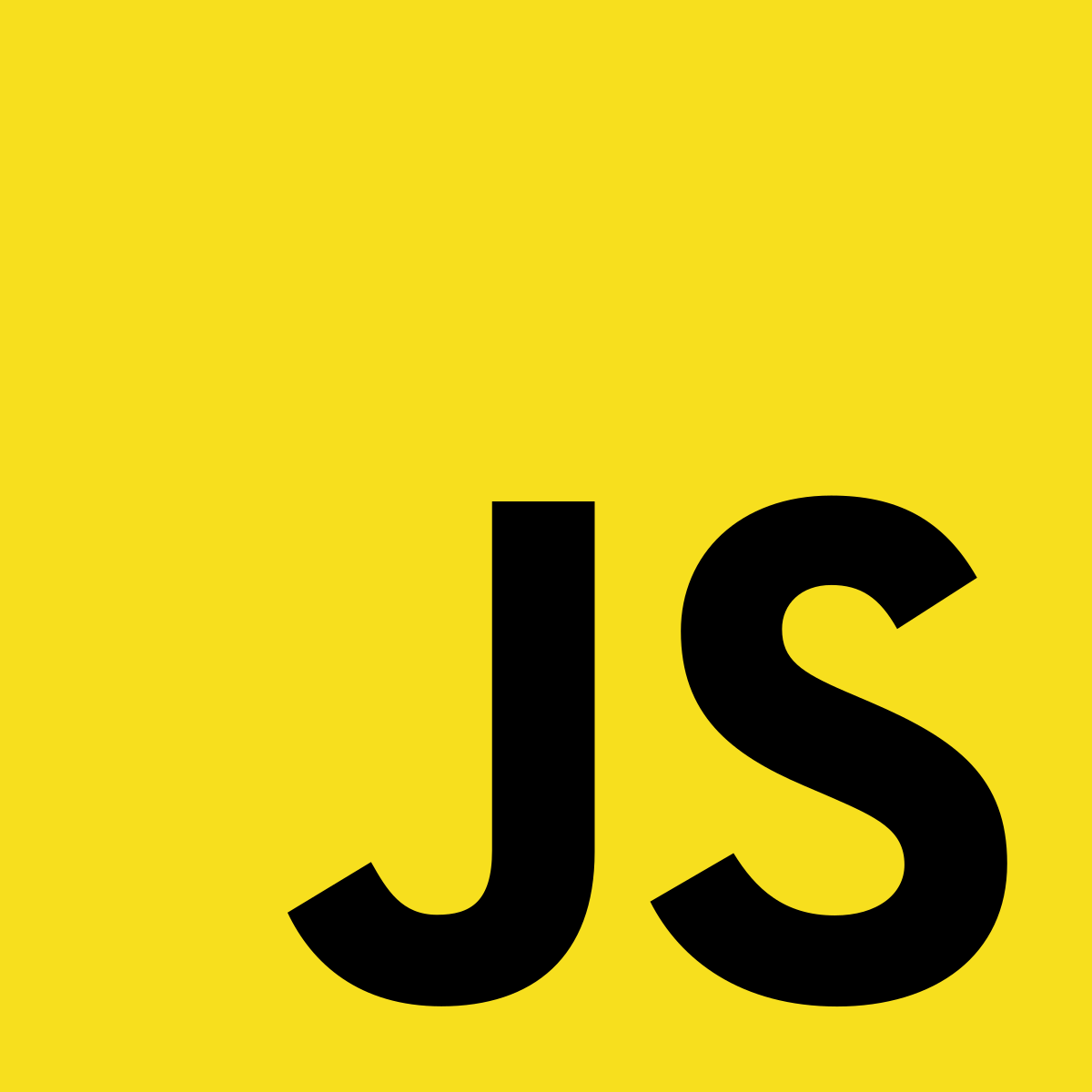
# Soluția propusă

Pentru a rezolva problemele prezentate anterior vom oferi o soluție modernă, care se folosește de cele mai noi tehnologii și trenduri din programarea web. Întreg sistemul se bazează pe microservicii scrise în întregime în JavaScript, folosind Vue.JS pentru front-end, respectiv Node.js pentru back-end. Fiecare serviciu rulează independent unul de celălalt, în propriul container Docker, astfel având acces la un mediu curat, independent de mașina gazdă, pentru a avea setări și dependențe într-un singur pachet ce va rula întotdeauna fără să fie nevoie de configurări adiționale.

## Limbajul de programare

JavaScript este un limbaj de programare multi-paradigmă ce suportă stilul clasic de programare imperativă (precum C, C++ sau Java), programarea funcțională (precum Haskell și mai nou Java sau Scala) și nu în ultimul rând programarea bazată pe evenimente. Acest ultim tip de programare menționat joacă un rol foarte important în tehnologia din zilele noastre. Datorită programării bazate pe evenimente astăzi ne putem bucura de aplicații web reactive, cu mult diferite de paginile statice scrise doar in HTML de la începuturile internetului. Cu acest tip de programare putem adăuga acțiuni specifice pentru fiecare click din pagină, putem asculta pentru evenimente precum schimbări în pagină, mișcarea mouse-ului sau utilizarea tastaturii. Astfel ne putem bucura de interfețe grafice mult mai complexe, mai intuitive, menite să ajute utilizatorul oferind o experiență mult mai plăcută în navigarea internetului. De asemenea, dinamismul oferit de programarea bazată pe evenimente este o principală caracteristică regăsită în tehnologiile de back-end. De aceea încă din 2009, de la lansarea Node.js, JavaScript, un limbaj inițial conceput pentru front-end, a migrat și către partea nevăzută a tehnologiei web, devenind și un limbaj de back-end, astfel oferind posibilitatea dezvoltatorilor să conceapă soluții „full-stack” fără a fi nevoiți să învețe și să folosească două limbaje de programare diferite.

Figură 1 Logo neoficial JavaScript

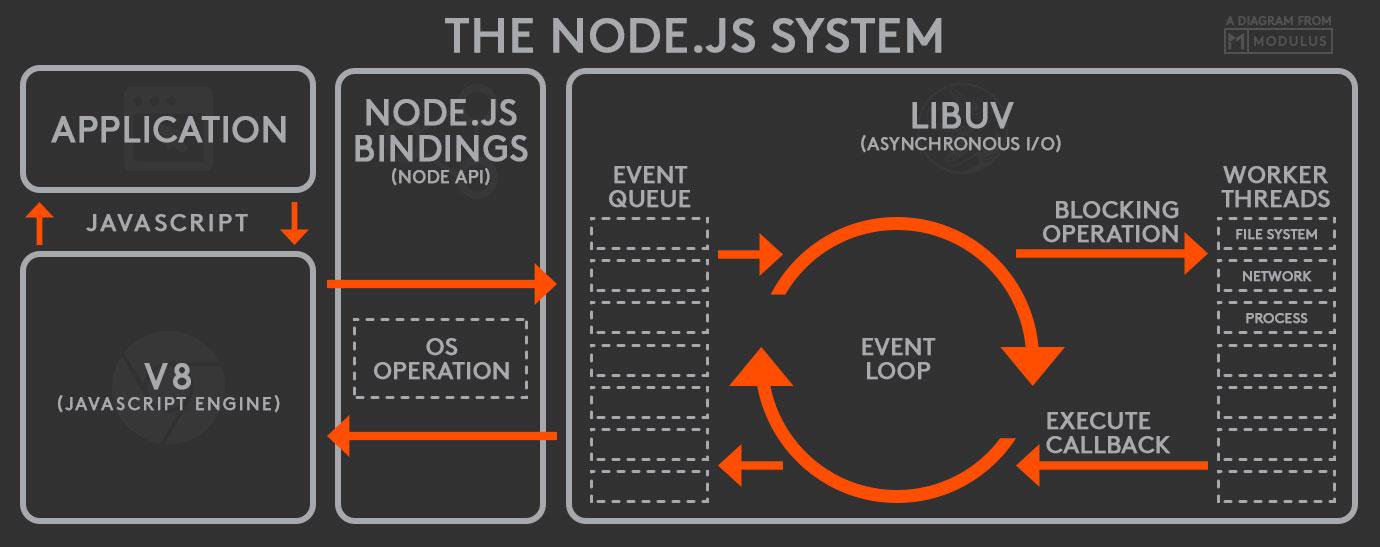


## Mediul de rulare și dezvoltare pentru server

Node.js este un mediu de rulare pentru JavaScript menit să aducă limbajul de programare din mediul browser-ului în mediul sistemului de operare. Fiecare browser dispune de un motor de rulare pentru JavaScript, o componentă care transformă codul din script-uri în cod binar, pe care îl poate înțelege și executa procesorul. Câteva exemple de asemenea motoare de rulare sunt „SpiderMonkey” pentru Mozilla Firefox, „Chakra” pentru Microsoft Edge și, nu în ultimul rând, „V8” pentru Google Chrome. Ultimul motor de rulare menționat, „V8”, stă la baza Node.js, însă nu reprezintă în totalitate mediul de rulare oferit de Node. Așa cum browser-ul pune la dispoziție dezvoltatorului documentul paginii, sau DOM – Document Object Model, pentru a putea prelucra informațiile din pagină, împreună cu evenimentele ce au loc pe aceasta, Node.js pune la dispoziție dezvoltatorului o suită de module ce mediază accesul la componentele sistemului de operare. Câteva dintre aceste module sunt „fs” (File System) pentru accesul la hard disk și lucrul cu sistemul de fișiere, „http” sau „https” pentru accesul la placa de rețea, deschiderea porturilor de rețea sau realizarea apelurilor în internet prin protocoalele standard TCP și UDP, modulul „crypto” care pune la dispoziție dezvoltatorilor câțiva algoritmi consacrați de criptare și decriptare și multe alte module utile în funcție de proiectul pentru care sunt folosite.

Figură 2 Logo oficial Node.js



Node.js strălucește atunci când proiectul impune multe scrieri și citiri atât din sistemul de fișiere, cât și din rețea, însă atunci când este nevoie de calcul intens, ocuparea pentru mult timp a procesorului cu o singură problemă, se va observa un impact major asupra performanței. Acest lucru se datorează modului de funcționare a limbajului de programare și a mediului de rulare. JavaScript este un limbaj de programare ce rulează pe un singur fir de execuție, spre deosebire de alte limbaje de programare existente care se pot folosi de toate nucleele și firele de execuție ale procesorului pe care rulează programul rezultat.

Figură 3 Arhitectura Node.js

Ce este special la Node.js este faptul că maximizează beneficiile oferite de natura asincronă a limbajului de programare, adică de paradigma de programare pe bază de evenimente. Codul scris de programator rulează pe un singur fir de execuție, însă fiecare funcție asincronă care accesează o componentă blocantă, de intrare-ieșire, este executată de Node.js pe un fir de execuție separat, urmând ca rezultatul să fie adăugat în coada de evenimente, coadă monitorizată în permanență de firul principal de execuție. Această arhitectură avantajează programatorul deoarece nu trebuie să aibă grijă să nu blocheze firul de execuție, deci reducând complexitatea codului, devenind mai accesibil pentru noi programatori.

Un alt aspect foarte important în favoarea acestei noi tehnologii este disponibilitatea unui manager de pachete, NPM – Node Package Manager, care oferă acces la o mulțime de module open-source. Apariția NPM este principalul motiv pentru creșterea în popularitate a Node.js, în mod special datorită modulelor „express” și „request”. Aceste două module au făcut mult mai accesibilă construirea unui server și comunicarea prin protocolul HTTP. Astfel, un simplu server poate fi scris în doar cinci linii de cod, iar un simplu apel de tip `GET` poate fi scris în doar două linii.

Figură 4 Logo oficial Node Package Manager



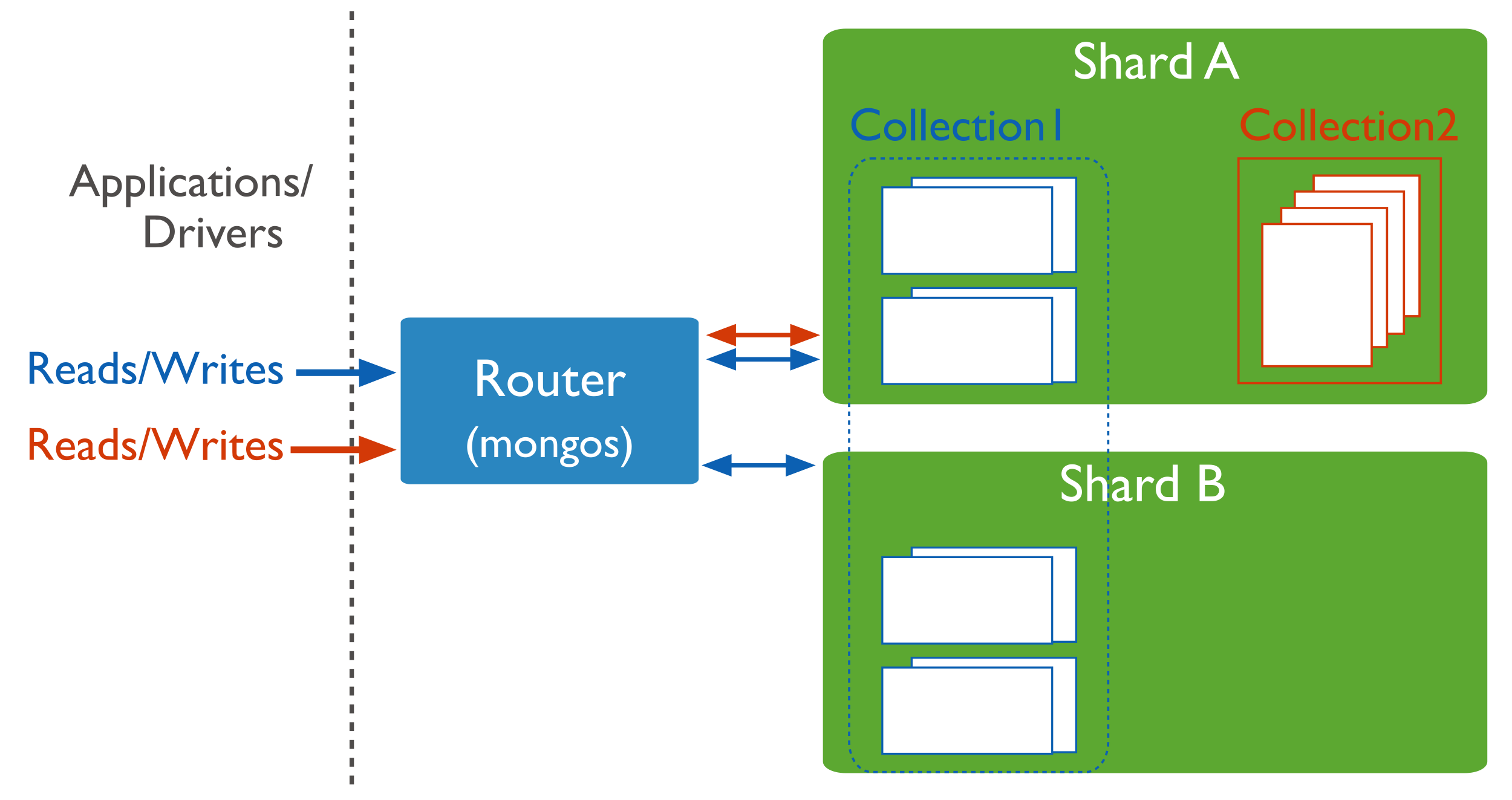
Odată cu popularizarea acestei noi tehnologii, multe dintre marile companii de IT ale lumii au început să își migreze serverele de la soluții tradiționale în domeniul business, precum Java Spring, la Node.js. Câteva companii mari care folosesc servere de producție scrise în Node.js sunt Netflix, PayPal, Uber, eBay, NASA și Yahoo. Paypal a raportat în urma dezvoltării aceluiași produs folosind Java Sprint și Node.js că folosind Node aplicația web a fost realizată aproape de două ori mai rapid de mai puțini dezvoltatori decât aplicația Java și a fost compusă din 40% mai puține fișiere și 33% mai puține linii de cod. Din punct de vedere al performanței, aplicația scrisă în Node.js a fost capabilă să răspundă la de două ori mai multe cereri decât aplicația scrisă în Java, cele două rulând pe mașini cu un fir de execuție pentru Node.js, respectiv cu cinci fire de execuție pentru Java Spring.

## Baza de date

Alt motiv pentru creșterea în popularitate a Node.js este MongoDB. MongoDB este o bază de date de tip NoSQL unde datele sunt reprezentate de documente (echivalentul rândurilor într-o tabelă) stocate în colecții (echivalentul tabelelor). MongoDB utilizează pentru documente un format de date numit JSON, un acronim pentru JavaScript Object Notation, însă aceste date sunt stocate în format binar, BSON – Binary JSON. Din nume reiese faptul că acest format de date este nativ pentru aplicațiile scrise în JavaScript, deci nu apare costul suplimentar necesar transformării datelor dintr-un tip în altul pentru prelucrare pe server. MongoDB este o soluție pentru aplicațiile în care au loc foarte multe citiri și relativ puține scrieri în colecții foarte mari de date. Chiar numele „mongo” provine din englezescul informal „humongous” care se poate traduce ca „foarte mare”, deci și din nume se poate înțelege că această bază de date suportă gestionarea unor cantități imense de date. Spre deosebire de bazele de date SQL, MongoDB, o bază de date de tipul NoSQL, nu impune o schemă, un format, pentru datele stocate, deci într-o colecție putem regăsi documente ce conțin date variate, un lucru ce poate fi foarte util în dezvoltarea pe termen lung al unui produs. De exemplu, după o perioadă îndelungată de timp în care s-au acumulat foarte multe date într-o colecție se dorește adăugarea unui nou parametru opțional. Adăugarea noului parametru se face prin modificarea codului care adaugă sau actualizează datele pe server, fără să necesite alte operațiuni care rulează în fundal în cadrul bazei de date, afectând performanțele pentru o perioadă de timp. Un alt punct forte pentru acest tip de baze de date este posibilitatea de replicare. MongoDB oferă o soluție pentru păstrarea redundanței prin posibilitatea configurării unui „Replica Set”, astfel dacă o mașină dintr-un „Replica Set” pățește ceva, datele nu sunt pierdute.

Figură 5 Logo oficial MongoDB



Alt mare avantaj pe care îl oferă această soluție pentru baza de date este posibilitatea de a scala orizontal. Multe dintre soluțiile SQL nu au posibilitatea de a scala orizontal, ci doar vertical, adică singura posibilitate de scalare este cea de a mai adăuga memorie și putere de procesare mașinii pe care rulează baza de date. Acest tip de scalare este limitată deoarece după un anumit prag nu se mai poate adăuga putere de procesare, iar scalarea orizontală este uneori imposibilă deoarece nu se pot împărți datele din tabele între mai multe mașini în cadrul unui cluster. Soluția NoSQL permite scalarea orizontală. În cazul MongoDB aceasta se materializează sub forma conceptului de „sharding”, posibilitatea de a avea mai multe „shard-uri”, adică mașini sau grupuri de mașini („replica set-uri”) care stochează datele în funcție de anumite criterii. Această soluție este una transparentă, utilizatorul bazei de date neavând posibilitatea de a ști că datele i-au fost livrate dintr-un „shard” sau dintr-un simplu „replica set”.

Figură 6 Exemplu de Sharding cu MongoDB

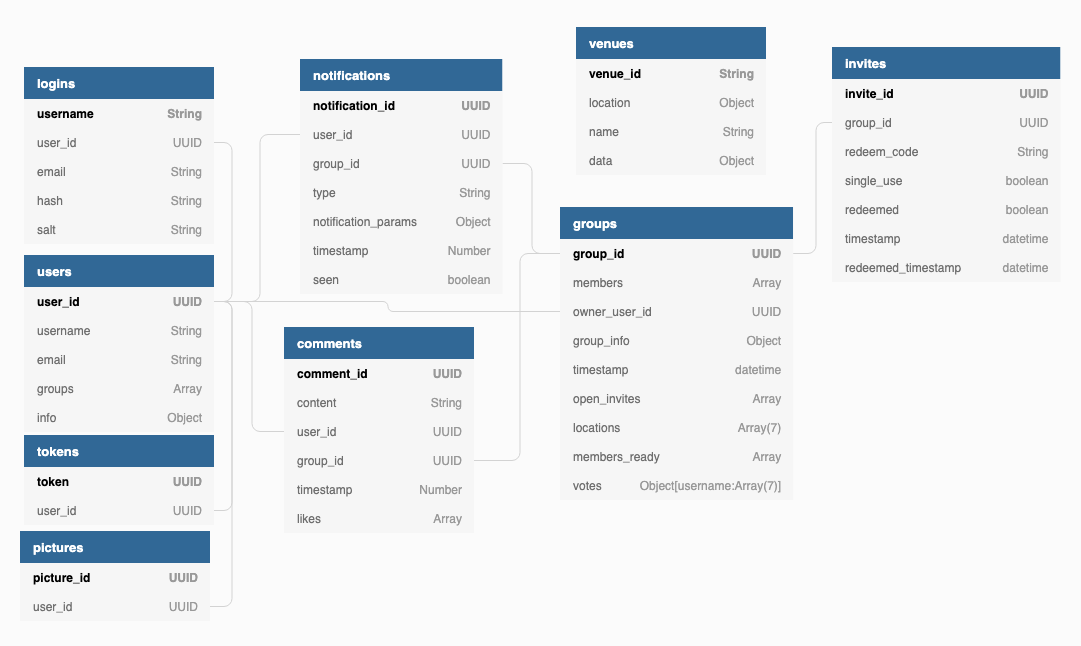


Figura 7 Structura bazei de date

În cadrul aplicației Get Together au fost necesare nouă colecții afișate în următoarea schemă (figura 1). Majoritatea ID-urilor folosite pentru indexarea și interogarea bazei de date sunt de forma „UUID” – „Universally Unique Identifier”, în mod normal un număr de 128 de biți reprezentat ca 32 de cifre hexazecimale („123e4567-e89b-12d3-a456-426655440000”), dar pe care îl stocăm ca pe un șir de caractere, un obiect de tip String.

Colecția „logins” conține datele adăugate în urma procesului de înregistrare pe platformă, utilizate pentru autentificare. Cheia principală, unică, este numele utilizatorului. Nu pot exista doi utilizatori cu același nume. De asemenea este adăugat și „user\_id”-ul utilizatorului nou creat pentru a stabili o relație între colecțiile „logins” și „users”. Cele mai importante date în această colecție, în afară de „username”, sunt „hash”-ul și „salt”-ul, date rezultate în urma criptării parolei utilizatorului folosind „PBKDF2” („Password-Based Key Derivation Function 2”).

Colecția „users” dispune de o cheie principală, unică, stocată sub numele „user\_id”. Această colecție conține date despre utilizatori precum „username”, „email”, o listă „groups” cu id-urile grupurilor din care face parte utilizatorul, respectiv un obiect „info” cu date opționale despre utilizator precum numele și prenumele real, data nașterii, sexul și id-ul pozei de profil. În această colecție și în colecția „logins” sunt adăugate date simultan, la înregistrarea unui nou utilizator.

Colecția „tokens” conține o asociere între „user\_id” și „token”-ul de autentificare pe baza căruia se permite accesul în back-end. Această colecție conține datele ce fac distincția între utilizatori într-o manieră opacă, fără să expună public date sensibile ce ar putea da de gol publicului structura bazei de date, precum „user\_id”-ul, deschizând o vulnerabilitate la atacuri. În colecția „tokens” sunt adăugate date în urma execuției procesului de autentificare pe bază de nume de utilizator și parolă.

Colecția „pictures” conține o asociere între id-ul pozei de profil al fiecărui utilizator și id-ul acestuia. În această colecție se adaugă date în urma procesului de încărcare a unei imagini. Principalul rol al acestei colecții este de a păstra o referință a fișierelor aflate în volumul de memorie a serviciului însărcinat cu servirea și stocarea pozelor de profil, „picture\_id”-ul fiind același cu numele fișierului stocat.

Colecția „notification” conține cheia unică „notification\_id” împreună cu id-urile pentru grupul unde a fost generată, respectiv al utilizatorului căruia i se adresează notificarea. De asemenea mai sunt stocate date precum tipul notificării, un „timestamp”, un număr care reprezintă milisecundele trecute de la 1 Ianuarie 1970, un parametru boolean, „seen”, care marchează dacă a fost văzută, sau nu, notificarea. Alături de datele obligatorii menționate este prezent și un obiect numit „notification\_params” în care sunt adăugate date opționale utilizate pentru generarea textului ce va fi afișat în front-end.

Colecția „comments” conține date despre comentariile adăugate într-un grup. Cheia unică este „comment\_id”, dar pentru o identificare rapidă sunt indexate și id-urile grupului și a utilizatorului care a adăugat comentariul. Alături de aceste date de identificare sunt prezente câmpurile „content”, pentru conținutul efectiv al comentariului, un timestamp pentru a marca data exactă când a fost adăugat comentariul și o listă cu id-urile utilizatorilor care au apăsat butonul de „like”.

Colecția „invites” conține date despre invitațiile generate. Această colecție conține două chei unice, „invite\_id”, respectiv „redeem\_code”. Id-ul invitației, un UUID, este folosit în operațiile de ștergere și de actualizare, iar codul de revendicare are o formă mai ușor de introdus manual. Acesta este format din 12 caractere alfanumerice, grupate câte trei și separate folosind caracterul cratima (Ex: „QBK3-BAPF-25HM”). Un astfel de document mai conține id-ul grupului aferent invitației, un parametru boolean „single\_use” care indică dacă invitația poate fi folosită de mai multe ori, respectiv două obiecte de tip „Date”, „timestamp” – pentru data la care a fost creată invitația, respectiv „redeem\_timestamp” – pentru data la care a fost revendicată invitația.

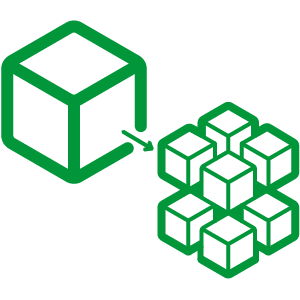
Colecția „venues” are un rol foarte important în menținerea costurilor la un nivel cât mai scăzut, împreună cu serviciul ce se ocupă de gestionarea acestei colecții. Aici se stochează date despre localurile obținute prin apeluri către serviciul „Google Places”, metoda „Nearby Search”. Acest serviciu este unul relativ scump ajungând să coste 32.00 de dolari americani pentru 1000 de cereri atunci când aplicația realizează într-o lună un număr de cereri aflat între 0 și 100 000, 25.69 de dolari americani pentru 1000 de cereri atunci când aplicația realizează între 100 001 și 500 000 de cereri pe lună. Pentru un volum mai mare de 500 000 de cereri HTTP pe lună se poate negocia prețul cu reprezentanții de vânzări ai companiei Google. Astfel, accesul la acest serviciu poate genera costuri foarte mari în timp. Pentru a combate acest fenomen toate cererile de listare a sugestiilor către back-end-ul Get Together, pentru o anumită zonă, se vor face doar după interogarea colecției „venues”. În cazul în care interogarea bazei de date nu a oferit destule rezultate se încearcă aceeași interogare, dar către serviciul „Google Places”, apoi rezultatul este stocat în această colecție. Câmpul „venue\_id” reprezintă un șir de caractere unic atribuit de serviciul oferit de Google. Acesta este folosit pentru a nu stoca de două ori aceleași date. Alte câmpuri ce se regăsesc într-un document din această colecție sunt un șir de caractere ce reprezintă numele localului, un obiect „data” ce conține răspunsul nealterat oferit de metoda „Nearby Search” oferită de „Google Places”, respectiv un obiect numit „location”. Acest ultim obiect este ceea ce face colecția „venues” să fie specială. Acesta este un index geospațial, mai precis un obiect „GeoJSON”, definit urmărind documentația MongoDB. Acesta conține un șir de caractere, „type”, care are întotdeauna valoarea „Point”, deoarece stocăm puncte pe hartă, respectiv o listă, „coordinates”, ce conține două valori numerice, longitudinea și latitudinea, în această ordine. Folosind acest index spațial putem interoga această colecție la fel cum am căuta date pe o hartă sau pe un glob pământesc. Putem seta un punct și o rază, urmând să obținem toate datele ce se intersectează cu sfera rezultată, sau putem căuta date ce se află în interiorul unui poligon. Faptul că putem realiza cu ușurință interogări geospațiale este un alt motiv pentru care am ales MongoDB pentru această aplicație.

Colecția „groups” este cea mai complexă dintre cele nouă utilizate în cadrul aplicației. Aceasta conține un index unic „group\_id”, o listă „members” cu toate id-urile utilizatorilor ce aparțin grupului, un „owner\_user\_id” pentru a reține cine a fost creatorul grupului, un obiect de tip dată numit „timestamp” menit să marcheze data la care a fost creat grupul și un obiect „group\_info” ce conține date generale despre grup, precum numele, descrierea sau tipul de locație ce se dorește a fi sugerată. De asemenea într-un obiect din această colecție mai sunt prezente trei liste, „open\_invites”, „locations”, „members\_ready”, respectiv un obiect „votes”, în care sunt marcate câteva dintre interacțiunile cu utilizatorul. În lista „open\_invites” se țin id-urile pentru invitațiile marcate ca fiind nelimitate sau care nu au fost revendicate încă. Lista „locations” conține obiecte cu trei câmpuri: „user\_id”, „username” și o listă cu fix șapte elemente numită „days”. În lista „days” sunt ținute pentru fiecare zi a săptămânii intervalul de timp la care este disponibil utilizatorul, respectiv coordonatele la care se va afla în ziua respectivă. Lista „members\_ready” conține numele utilizatorilor care au anunțat că au terminat de raportat locațiile și intervalul de timp la care vor fi disponibili. Când această listă conține toți membrii grupului are loc calculul final ce oferă sugestia locului de întâlnire, împreună cu intervalul orar la care poate avea loc întâlnirea. Obiectul „votes” conține date de forma cheie-valoare, unde cheia o reprezintă numele utilizatorului care a votat, iar valoarea este o listă cu șapte șiruri de caractere de tip „venue\_id” pentru locațiile pe care le preferă utilizatorul în funcție de ziua săptămânii. Acest obiect este utilizat în calculul final al sugestiei, dar și într-un caz special. În cazul în care un utilizator și-a votat preferința înainte ca toți ceilalți membri ai grupului să își raporteze locația, această preferință trebuie salvată și luată în calcul. Este posibil ca atunci când ceilalți utilizatori își introduc datele algoritmul să ofere noi sugestii care nu se află în vecinătatea opțiunii primului utilizator. În acest caz trebuie să permitem celorlalți utilizatori să poată vota opțiunea primului.

În urma descrierii fiecărei colecții utilizate se poate observa că MongoDB este o alegere favorabilă pentru acest proiect deoarece operațiile uzuale, cele care au loc în cadrul grupului, nu necesită scrierea în mai mult de o colecție, fapt pentru care colecția „groups” este cea mai complexă, dar care totuși nu trebuie compilată folosind date ce se pot schimba din alte colecții. Acest avantaj și posibilitatea de a avea redundanță prin „Replica Set”-uri, împreună cu faptul că MongoDB permite scalarea pe orizontală prin funcția de „sharding” face ca această soluție să se preteze perfect unei arhitecturi de microservicii.

## Arhitectura sistemului de back-end

Arhitectura de microservicii este un concept relativ nou ce aduce o multitudine de beneficii unei aplicații atât în din punct de vedere al operabilității cât și pentru dezvoltare.



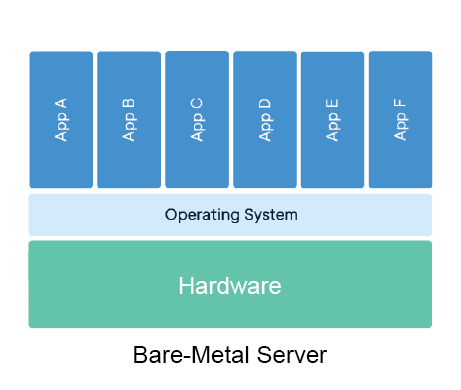
Majoritatea aplicațiilor care sunt destinate să ruleze pe dispozitivele utilizatorilor sunt dezvoltate într-o manieră similară modelului de microservicii. Este cunoscut și predat în toate cursurile de programare că o aplicație trebuie să fie modulară, să conțină componente slab cuplate care lucrează împreună pentru a realiza un întreg. Această practică ajută la scrierea unui cod mai ușor de înțeles, mai ușor de reutilizat, mai ușor de testat și, cel mai important, mai ușor de menținut în proiecte mari, pentru care este dorită o creștere pe un termen de lungă durată. În aplicațiile tradiționale, menite să ruleze pe dispozitivele utilizatorului nu poate fi adoptată o structură similară cu cea utilizată în dezvoltare. În urma procesului de construire, în care se leagă toate modulele din dezvoltare, rezultă un singur pachet ce reprezintă produsul final. Aceeași abordare a fost utilizată pentru foarte mult timp și în domeniul internetului, având unul sau mai multe servere, fiecare incluzând întreaga logică a aplicației. Drept rezultat al acestei abordări o aplicație putea fi scalată doar în întregime, astfel consumând resurse fizice, computaționale și economice pentru a crește performanțele anumitor componente care nu necesitau o creștere în performanțe. Alt dezavantaj al acestei abordări îl reprezintă dificultatea în a testa separat modulele, mai ales atunci când se dorește introducerea unor noi funcționalități. Abordarea descrisă până acum poartă numele de „Arhitectură Monolitică”, numele provenind din cuvântul „monolit”, care se traduce în expresia „dintr-un singur bloc” (conform DEX 2009). La fel poate fi privită și abordarea tradițională care produce un singur pachet, un „singur bloc”.

Figură 8 Exemplu vizual pentru modularizare

Figură 9 Arhitectura Monolitică. Arhitectura de Microservicii

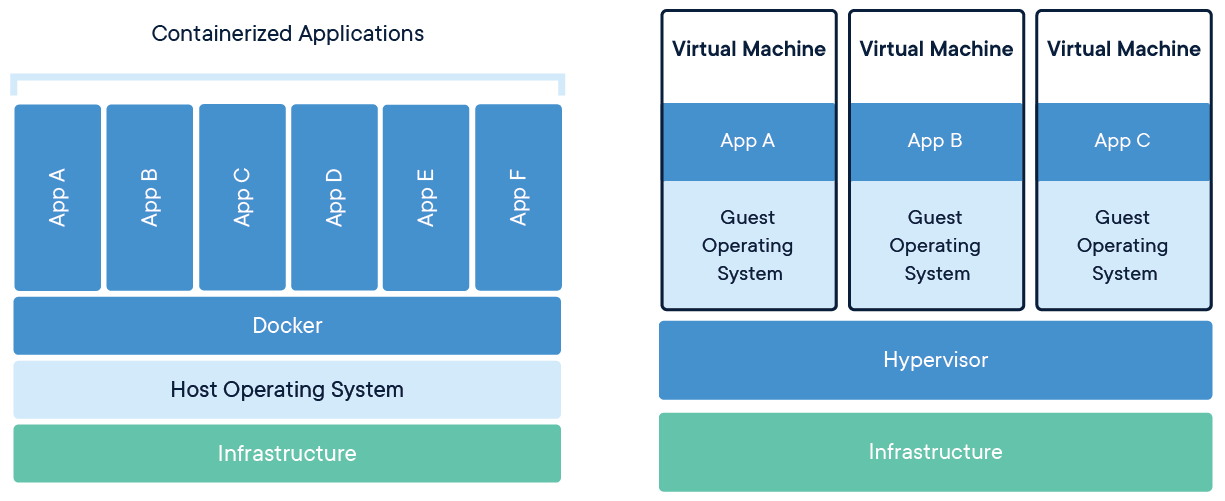
Odată cu popularizarea și dezvoltarea aplicațiilor web a început și dezvoltarea arhitecturii bazate pe microservicii. Spre deosebire de aplicațiile tradiționale, aplicațiile web nu necesită instalarea unui nou pachet pe dispozitivele utilizatorului deoarece acestea utilizează browser-ul pentru a afișa conținutul, acesta fiind livrat de către un server, sau serviciu, dedicat. De asemenea, orice altă prelucrare are loc pe o mașină independentă de cea a utilizatorului. Drept rezultat, dezvoltatorii acestui tip de arhitectură au realizat că „monolitul” nu mai este necesar. Ca urmare, fiecare modul a devenit un serviciu de sine stătător, comunicând cu celelalte module, sau servicii, folosind anumite protocoale precum HTTP, MQTT sau AMQP. Un avantaj oferit de acest model este dezvoltarea mai ușoară și independentă. Mai multe echipe pot dezvolta și testa în paralel anumite module fără să fie constrânse la a folosi un anumit limbaj de programare. Alt avantaj, pentru aplicațiile ce vor avea trafic intens, este scalarea individuală a componentelor cheie, astfel fiind posibilă maximizarea utilizării resurselor.

## Platforma de lansare (deploy) a microserviciilor

Pentru a maximiza utilizarea resurselor trebuie să avem în vedere și platforma de lansare al sistemului de microservicii. Prima opțiune, și cea mai rudimentară, este lansarea fiecărui serviciu pe propriul sistem „bare-metal”. Această opțiune practic atribuie fiecărui serviciu propriul său calculator fizic, propriul server într-un data center, având acces la întreaga memorie a sistemului, la tot procesorul, cu toate nucleele, însă un singur serviciu înseamnă o singură aplicație care rulează la orice moment de timp. Acest lucru poate avea atât avantaje cât și dezavantaje. Un principal avantaj este faptul că serviciul va avea întotdeauna acces la resurse fără să fie blocat de un alt proces care să ocupe timp pe procesor sau spațiu de memorie, dar acest lucru poate prezenta și un dezavantaj major din punct de vedere economic deoarece are crește costurile indiferent dacă toată puterea de procesare este utilizată sau nu. Un alt dezavantaj este configurarea serviciilor pe aceste sisteme. Trebuie ținut cont de toate dependențele fiecărui serviciu în parte, trebuie realizate, de cele mai multe ori manual, configurațiile necesare funcționării corecte a serviciului, deci procesul de lansare sau de actualizare a unui serviciu poate fi îngreunat pe un astfel de sistem.

Figură 10 Diagramă Sistem Bare-Metal

O alternativă mai eficientă din punct de vedere al resurselor, ce oferă mai multă flexibilitate, este utilizarea unor mașini virtuale. Utilizând un sistem bare-metal, adăugăm un hypervisor de tip 1 precum Hyper-V sau KVM, eliminând necesitatea unui sistem de operare. Fiecare serviciu rulează în mod independent, separat unul de celălalt, în propria mașină virtuală. Utilizând hypervisor-ul putem scala vertical fiecare mașină virtuală în parte, adăugând resurse în funcție de necesitatea serviciului. Dezavantajele acestei alternative apar atunci când este nevoie de actualizarea sau repornirea rapidă a unui serviciu. Mașinile virtuale, deși eficientizează utilizarea resurselor, au ca dezavantaj faptul că fiecare serviciu necesită propriul sistem de operare, cu propriile dependențe, iar în cazul unei reporniri se așteaptă repornirea sistemului de operare cu toate componentele sale.



Figură 11 Comparație intre Docker si Mașini Virtuale

Cea mai eficientă platformă pentru microservicii o reprezintă containerele Docker. Acestea rulează pe sistemul de operare al mașinii hardware, Docker ocupându-se de separarea, izolarea, fiecărei aplicații. Un mare avantaj pe care îl au imaginile de Docker este dimensiunea redusă. Spre deosebire de mașinile virtuale care conțin pe lângă aplicație un sistem de operare întreg, cu toate bibliotecile, driverele și dependențele sale, o imagine Docker conține strict aplicația ce urmează să fie rulată împreună cu dependențele acesteia. Ca rezultat direct al dimensiunii reduse ale containerelor se observă timpii necesari reduși pentru operațiile de pornire, respectiv repornire.

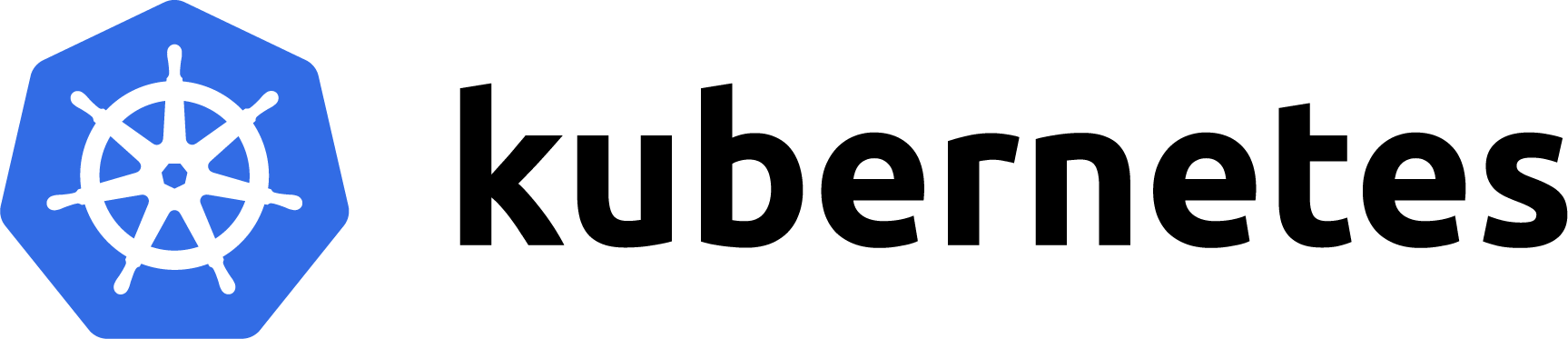
Figură 12 Logo oficial Docker



## Sistem de orchestrare a microserviciilor

Pentru a nu expune sistemul de microservicii la eroarea umană, mai ales într-un mediu de producție, este nevoie de un sistem de orchestrare a serviciilor. Acest sistem trebuie să fie capabil să construiască imagini de Docker, să ofere informații despre acestea, să lanseze containerele noi create într-un cluster și, cel mai important, să permită actualizarea containerelor fără să întrerupă funcționarea serviciilor.

Figură 13 Logo oficial Kubernetes



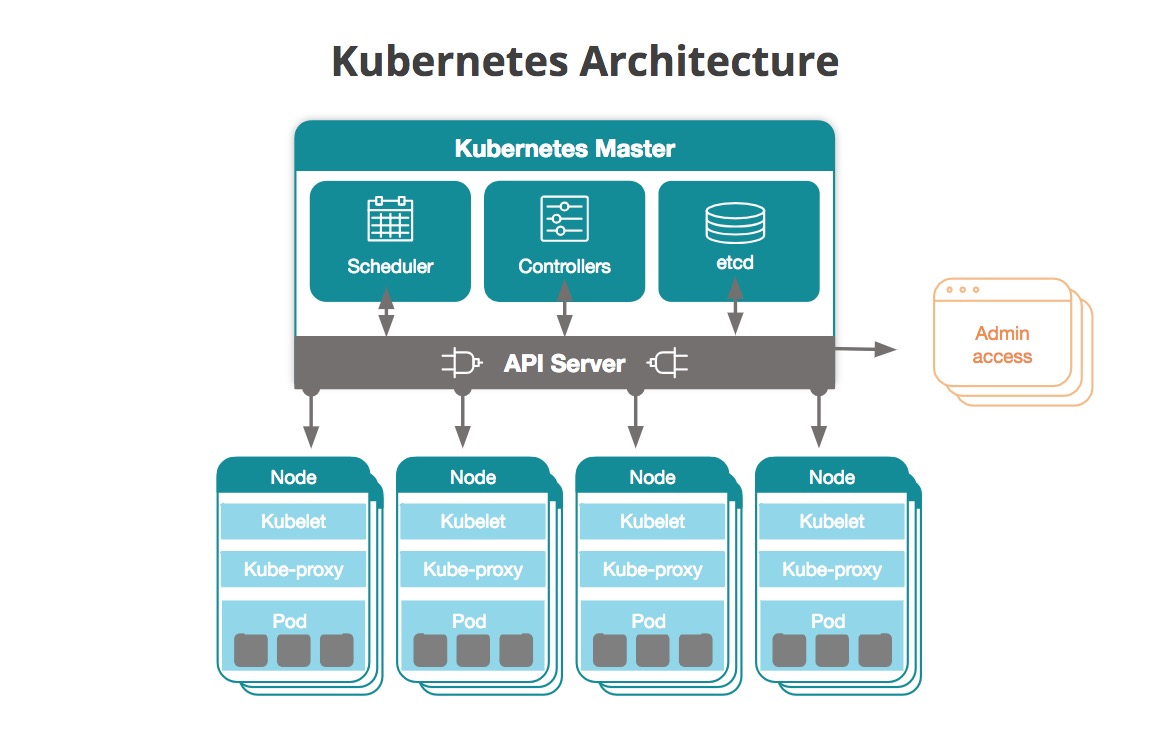
Pentru această problemă soluția este oferită de Kuberentes. Kuberentes este un sistem de orchestrare a serviciilor într-un cluster. Această soluție funcționează cu cei mai mari furnizori de infrastructură cloud precum Amazon AWS, Google Cloud sau Microsoft Azure.

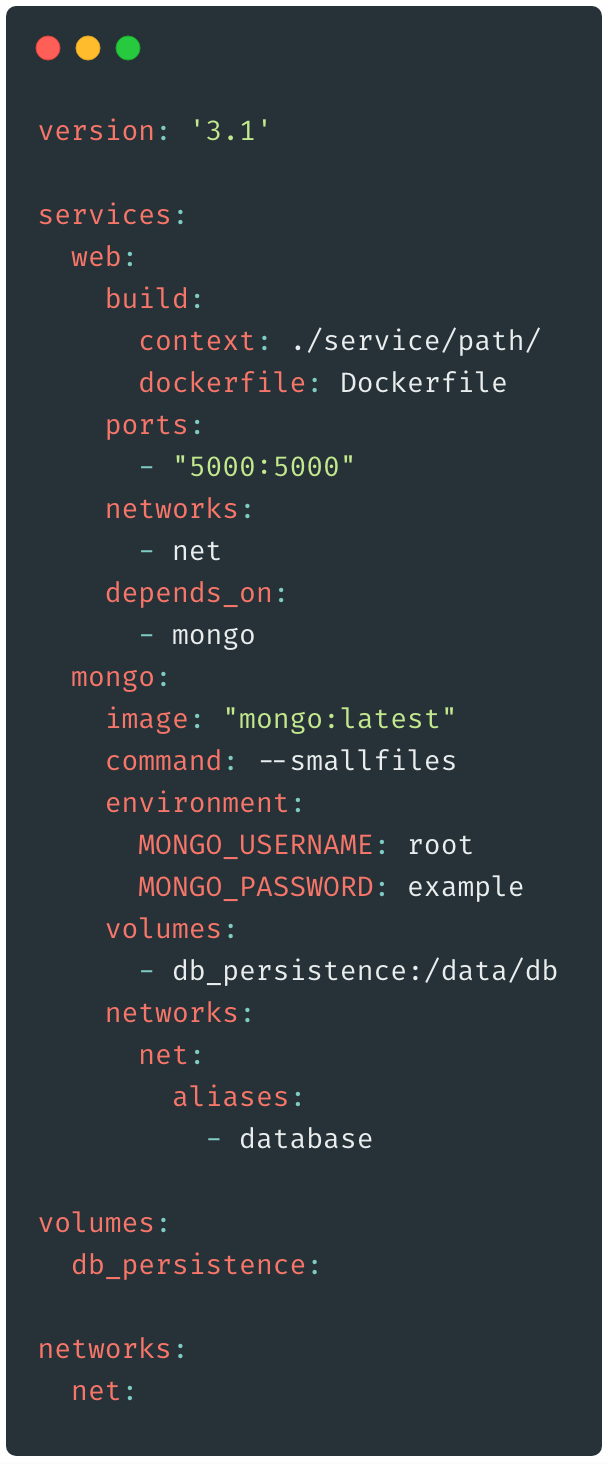
Kubernetes este capabil să îndeplinească mai multe funcții necesare pentru a rula un sistem de microservicii într-un cluster. Un prim lucru de care este capabil este lansarea microserviciilor și monitorizarea acestora pentru a se asigura că funcționează în parametri normali. De asemenea, într-un fișier de configurare, pentru servicii administrate de Kubernetes, pot fi specificate numărul de instanțe, deci orchestratorul se poate ocupa și de replicare, pot fi specificate limitele normale de resurse pe care le consumă serviciul, astfel dacă acesta depășește limitele mai pot fi pornite automat instanțe care să preia din volumul de muncă. Alte funcții utile într-un cluster, de care este capabil Kubernetes, sunt organizarea rețelei ce interconectează serviciile, astfel încât să poată exista comunicație între acestea, balansarea volumului de muncă între instanțe ale aceluiași serviciu și funcția de descoperire în rețea, astfel încât fiecare serviciu să fie informat despre celelalte servicii la care are acces.

Kubernetes folosește o arhitectură de tip „Master-Worker”. Server-ul Master expune un API (Application Programming Interface), utilizat de o eventuală interfață de administrare, care se poate utiliza pentru a încărca fișiere de configurare și pentru a realiza alte setări. Acesta conține o componentă „Scheduler”, responsabilă cu stabilirea nodului pe care vor rula containerele noi adăugate, un „store etcd”, o magazie distribuită în cluster pentru a stoca date sub forma cheie-valoare, respectiv o multitudine de controllere pentru alte subsisteme.

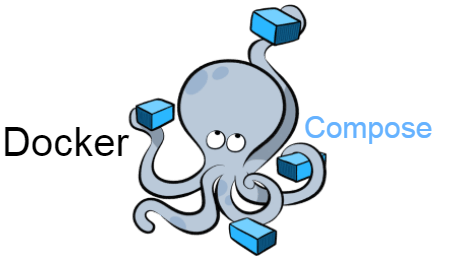
Entitățile de tip „Worker”, sau noduri, sunt de fapt mașinile pe care urmează să fie rulate containerele de Docker. Fiecare dintre acestea rulează un proces numit „Kubelet” care se ocupă de managementul containerelor, iar alături de acesta o altă componentă numită „Kube-proxy” care, așa cum indică și numele, se ocupă de comunicarea în rețea cu celelalte mașini de tip „worker”. De asemenea, pe figura 14 se poate observa prezența unor elemente numite „Pod”. Aceasta este unitatea principală de lucru utilizată de Kubernetes. Într-un „Pod” pot fi pornite unul sau mai multe containere ce împart aceleași resurse de rețea sau de stocare împreună cu date, setări sau configurări, despre rularea serviciului pornit în containerele respective.

Figură 14 Arhitectura Kubernetes



Pentru dezvoltarea sistemului de microservicii Get Together am folosit un sistem mai rudimentar de pornire a serviciilor în containere, deoarece dezvoltarea a avut loc pe o singură mașină, un laptop, însă pentru un mediu de producție este recomandată utilizarea unui sistem de orchestrare precum Kubernetes.

Figură 15 Logo neoficial Docker-Compose



Sistemul utilizat în dezvoltare se numește „Docker-Compose”. Acesta oferă o soluție rapidă pentru configurarea serviciilor ca într-un cluster folosind un fișier de tipul *.yaml* (Figura 16) și câteva comenzi în terminal. După cum se poate observa din figura 16, docker-compose permite declararea mai multor servicii. În exemplu sunt declarate două containere „web”, respectiv „mongo”. Folosind eticheta „build” se poate specifica locația, sub-etichetă „context”, și fișierul „dockerfile” necesare pentru construirea imaginii ce va fi folosită în container pentru rularea serviciului. De asemenea se poate observa eticheta „ports” folosită pentru a marca o asociere între portul ce va fi deschis pe mașina gazdă, respectiv pe container. Eticheta „networks”, prezentă în descrierea serviciului „web”, respectiv la finalul fișierului de configurare, marchează apartenența serviciului la o anumită rețea declarată la finalul fișierului. Aceeași etichetă din descrierea serviciului „mongo” conține o sub-etichetă „aliases”, folosită pentru a adăuga noi adrese la care poate fi accesat serviciul în cadrul rețelei.. Folosind eticheta „depends\_on” condiționăm serviciul să pornească doar după pornirea altui serviciu. În configurarea serviciului „mongo” putem observa o etichetă „command” folosită pentru a adăuga un parametru în linia de comandă la pornirea procesului, împreună cu eticheta „environment” utilizată pentru a adăuga variabile de mediu în container. Eticheta „volumes” este utilizată pentru a asocia un volum declarat la finalul fișierului de configurare cu un director din container, astfel asigurând persistența datelor.

Figură 16 Exemplu de fișier de configurare docker-compose.yaml

## Tehnologie pentru dezvoltare front-end

Pentru front-end există trei mari opțiuni pentru începerea unui nou proiect: ReactJS, dezvoltat de inginerii de la Facebook, AngularJS, dezvoltat de ingineri de la Google împreună cu alți dezvoltatori independenți și companii, respectiv Vue.js, dezvoltat de un fost inginer Google. Framework-ul cel din urmă este cel mai nou dintre cele enumerate, acesta fiind lansat inițial în 2014. În ciuda perioadei relativ mică de la lansare, Vue.js a căpătat foarte multă popularitate. Motivația din spatele dezvoltării acestei soluții a fost nevoia autorului, Evan You, de a avea la dispoziție o metodă rapidă pentru realizarea de prototipuri.

Figură 17 Logo oficial Vue.js



Vue.js este supranumit un „Framework progresiv” deoarece nu obligă dezvoltatorul să rescrie o aplicație de la zero pentru a-l folosi, fiind posibil să integrezi Vue.js doar într-o anumită componentă a unui site sau a unei aplicații deja existente, fiind un framework relativ mic.

Un principal motiv pentru care acest framework a câștigat atât de multă popularitate este documentația cu un aspect prietenos, cu multe exemple și explicații, scrisă atât pentru dezvoltatori cu experiență, cât și pentru noi programatori. Se poate observa în mod clar că documentația a fost scrisă evitând pe cât de mult cu putință fenomenul de „over-engineering”, tendința de a face lucruri simple să pară mult mai complicate decât sunt cu adevărat, astfel Vue.js obținând o audiență de noi programatori care doresc să învețe rapid și ușor un framework pentru dezvoltare a aplicațiilor web. Un alt motiv pentru creșterea rapidă în popularitate este faptul că a împrumutat idei și concepte de la celelalte framework-uri. De exemplu, directive precum „v-bind”, pentru asocierea unui câmp din interfața HTML cu o variabilă, sau „v-for”, pentru iterarea prin elementele unei liste, au fost inspirate de „ng-bind” și „ng-repeat” din AngularJS. Structura orientată pe componente modulare este un punct forte pentru ReactJS, motiv pentru care a fost implementată și în Vue.js.

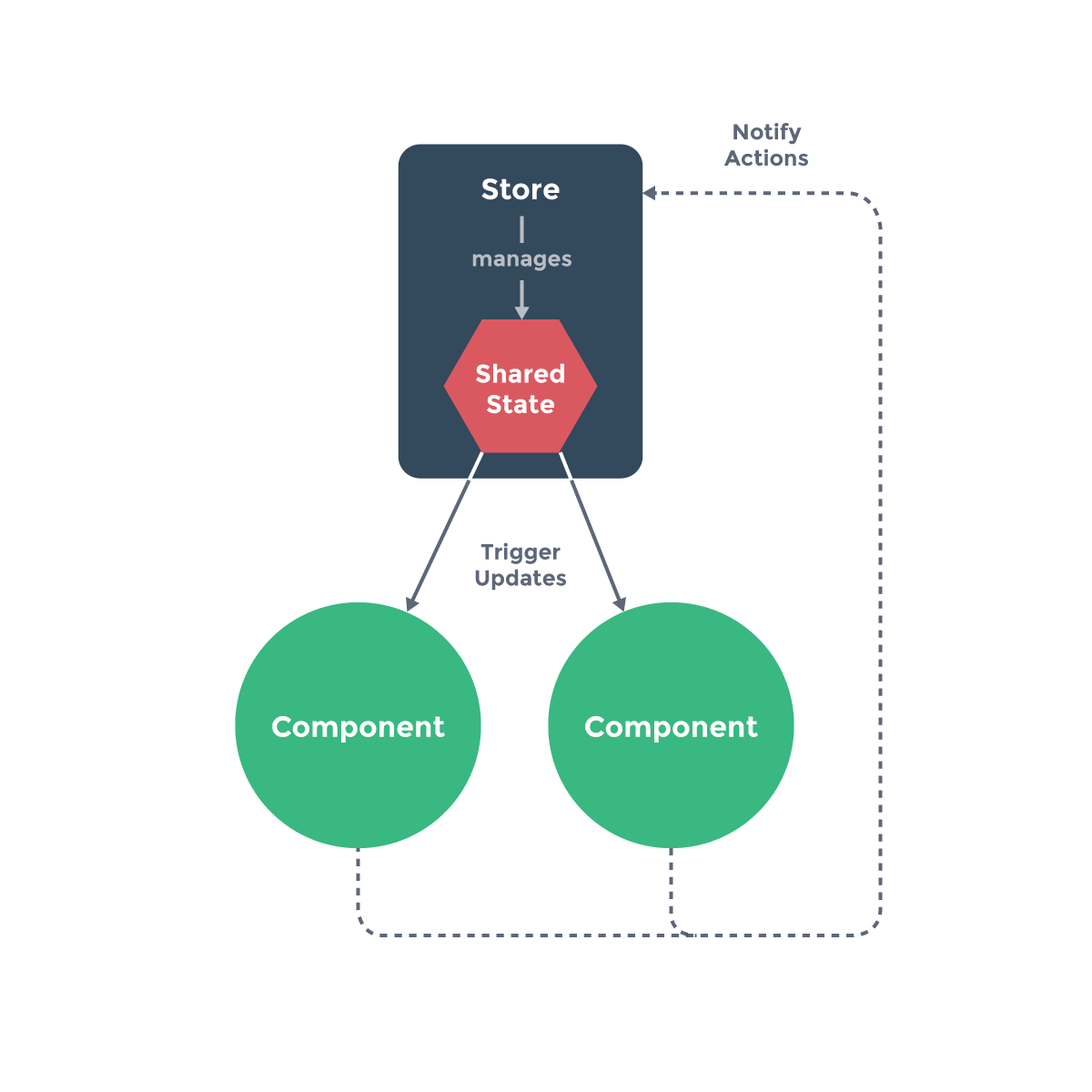
Vue.js pune la dispoziția dezvoltatorilor și metode de a dezvolta o aplicație de la 0, oferind o varietate de unelte utile în procesul de dezvoltare. Utilitarul „@vue/cli” (în traducere „Vue Command Line Interface” – „Interfață pentru Linia de Comandă”) permite generarea rapidă a unui șablon pentru începerea dezvoltării oricărui proiect, șablon ce conține toate dependențele necesare pentru dezvoltare precum „Babel”, un utilitar de convertire a codului JavaScript modern în cod JavaScript compatibil cu versiuni mai vechi ale unor aplicații de tip browser, sau „ESLint”, un utilitar menit să marcheze erorile de formatare a codului, astfel obligând programatorul să scrie cod lizibil. O altă funcție extrem de utilă în timpul dezvoltării este „vue-cli-service serve”, mai ușor pornită utilizând script-urile oferite de „npm” prin rularea comenzii „npm run serve”, care pornește un server de dezvoltare pe portul 8080. În urma pornirii serverului, orice schimbare din cod va fi procesată în timp real. Astfel în câteva milisecunde fiecare modificare făcută în fișierele sursă poate fi observată.

Un alt avantaj pe care îl prezintă Vue.js este accesul rapid la module adiționale folosind „npm” – Node Package Manager, același manager de pachete pe care îl folosim și pentru dezvoltarea serviciilor de back-end.

Printre modulele folosite în front-end-ul Get Together se numără:

* „axios”, un modul folosit pentru executarea cererilor de tip HTTP către back-end,
* „moment”, un modul folosit pentru a transforma obiectele de tip dată în șiruri de caractere reprezentând timpul trecut până în prezent într-o manieră umană,
* „vue-cookie”, un modul menit să asiste utilizarea cookie-urilor în browser,
* „vue-browser-geolocation”, un modul folosit pentru a simplifica obținerea locației oferite de browser
* „vue-croppie”, o adaptare pentru Vue.js a modulului „croppie”, modul utilizat pentru normalizarea imaginilor încărcate de utilizatori înainte de a fi încărcate pentru utilizarea ca poză de profil
* „bootstrap” și „popper.js”, module folosite pentru integrarea pachetului de componente de front-end.

Figură 18 „Design Pattern”-ul introdus de modulul "vuex"



* „vue-router”, modulul ce permite browser-ului să afișeze diferite căi, precum „/login” sau „/dashboard”, însă fără să necesite reîncărcarea paginii ci doar a componentelor din pagină
* „vue-socket.io”, o adaptare pentru Vue.js a modului „socket.io” ce realizează o integrare perfectă cu stilul de lucru impus de acest framework, modul utilizat pentru comunicarea folosind WebSockets
* „vuex”, un modul care integrează un „Design Pattern” de

## Tehnologie pentru stilizare front-end

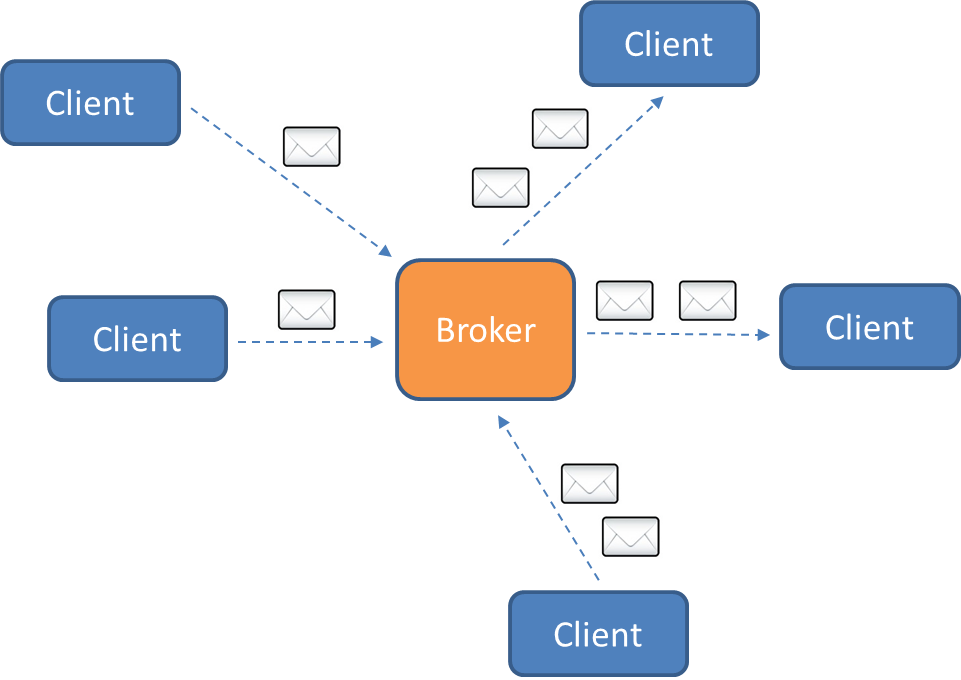
Bootstrap



## Alte tehnologii folosite în back-end

### Mosquitto / MQTT.



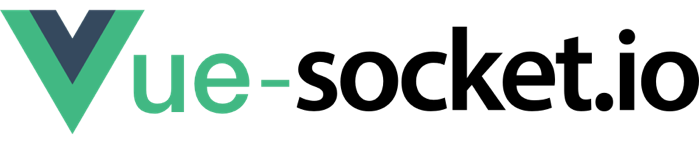


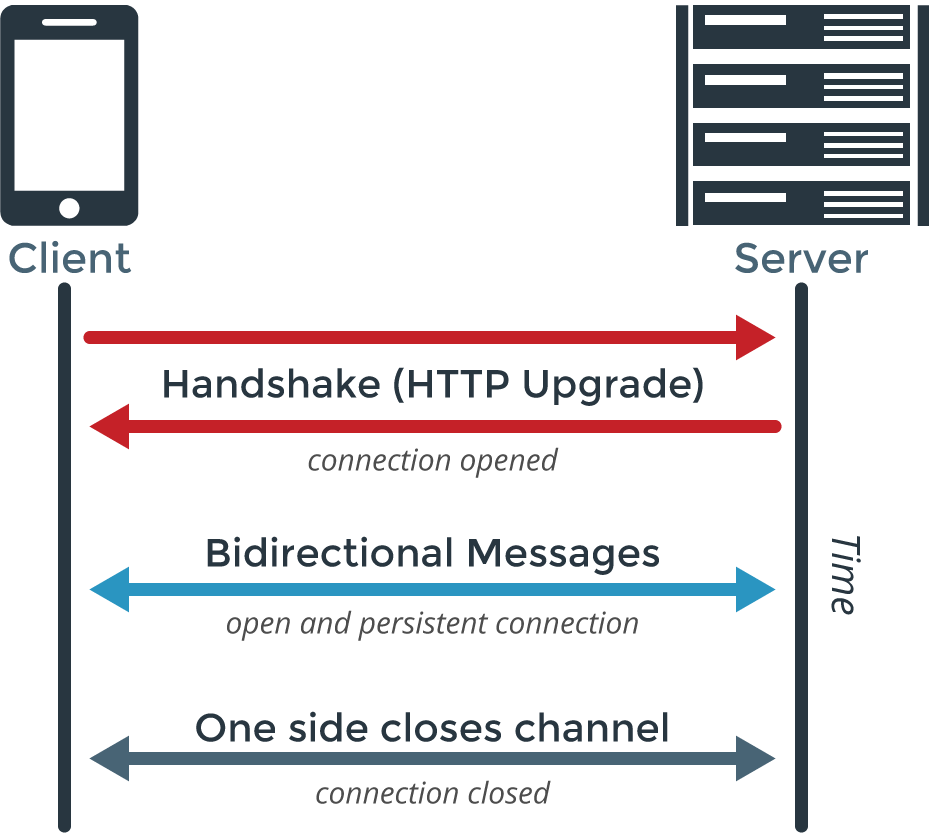
### Redis for tokens caching



### WebSockets. Socket.io







# Detalii de implementare

# Studiu de caz / Evaluarea rezultatelor

# Concluzii

Concluzia este că

# Bibliografie

https://medium.com/paypal-engineering/node-js-at-paypal-4e2d1d08ce4f

https://www.youtube.com/watch?v=zphcsoSJMvMN

https://twitter.com/rauschma/status/496213393146404864

https://www.netguru.com/blog/top-companies-used-nodejs-production

https://developers.google.com/places/web-service/usage-and-billing#nearby-search

<https://rancher.com/blog/2019/microservices-vs-monolithic-architectures/>

<https://blog.newrelic.com/engineering/what-is-kubernetes/>

<https://www.monterail.com/blog/reasons-why-vuejs-is-popular>

# Anexe