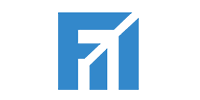
**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI**

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**“Doctorul Meu”**

**Arhitecturi de tip “serverless” în dezvoltarea aplicațiilor web bazate pe microservicii**

**propusă de**

***Andrei Codrin Harpa***

**Sesiunea:** *iulie, 2018*

**Coordonator științific**

**Conferenţiar, Dr., Anca Vitcu**

**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI**

**FACULTATEA DE INFORMATICĂ**

**„Doctorul Meu”**

**Arhitecturi de tip serverless în dezvoltarea aplicațiilor web bazate pe microservicii**

***Andrei Codrin Harpa***

**Sesiunea:** *iulie, 2018*

**Coordonator științific**

***Conferenţiar, Dr., Anca Vitcu***

Avizat,

Îndrumător Lucrare de Licență

Conferenţiar, Dr., Anca Vitcu\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Data \_\_\_\_\_\_\_\_\_\_\_\_ Semnătura \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DECLARAȚIE privind originalitatea conținutului lucrării de licență**

Subsemnatul Harpa Codrin-Andrei cu domiciliul în Str. Constantin Gane, Nr. 9, Iași, jud. Iași născut la data de 14.03.1997, identificat prin CNP 1970314226711, absolvent al Universității „Alexandru Ioan Cuza” din Iași, Facultatea de Informatică specializarea Informatică, promoția 2018, declar pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul: „Doctorul Meu - *Arhitecturi de tip serverless în dezvoltarea aplicațiilor web bazate pe microservicii”* elaborată sub îndrumarea d-nei Anca Vitcu, pe care urmează să o susțină în fața comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări științifice in vederea facilitării falsificării de către cumpărător a calității de autor al unei lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Dată azi, .............................. Semnătură student ..............................

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „Doctorul Meu - *Arhitecturi de tip serverless în dezvoltarea aplicațiilor web bazate pe microservicii”*, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent Harpa Codrin-Andrei

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

Table of Contents

[1 Introducere 6](#_Toc517922972)

[1.1 Motivație 6](#_Toc517922973)

[1.2 Context 6](#_Toc517922974)

[1.3 Obiective 7](#_Toc517922975)

[1.4 Descrierea soluției și a metodologiei folosite 8](#_Toc517922976)

[1.5 Structura lucrării 9](#_Toc517922977)

[2 Contribuții 10](#_Toc517922978)

[3 Ingineria Cerințelor 11](#_Toc517922979)

[3.1 Diagrama Use Case 12](#_Toc517922980)

[4 Cloud Computing în dezvoltarea aplicațiilor web bazate pe microservicii 13](#_Toc517922981)

[4.1 Introducere în Cloud Computing 13](#_Toc517922982)

[4.2 Dezvoltarea web clasica 14](#_Toc517922983)

[4.3 Microservicii 14](#_Toc517922984)

[4.4 Rolul Cloud în dezvoltarea web bazata pe Microservicii 15](#_Toc517922985)

[5 Aplicații serverless 16](#_Toc517922986)

[5.1 Dezvoltarea aplicațiilor web la nivel de client 16](#_Toc517922987)

[5.2 Angular 5 16](#_Toc517922988)

[5.3 Paradigma Serverless 18](#_Toc517922989)

[6 Detalii de implementare 19](#_Toc517922990)

[6.1 Limbaj și mediu de execuție 19](#_Toc517922991)

[6.2 Rutare API Gateway 20](#_Toc517922992)

[6.3 Autentificare și autorizare 21](#_Toc517922993)

[6.3.1 AWS Cognito 21](#_Toc517922994)

[6.4 Nivelul de stocare 23](#_Toc517922995)

[6.4.1 Diagrama de clase 24](#_Toc517922996)

[6.4.2 DynamoDB 25](#_Toc517922997)

# 1 Introducere

## 1.1 Motivație

Scopul acestei lucrari este de a prezenta avantajele folosirii tehnologiilor de tip „serverless” în dezvoltarea produselor software bazate pe microservicii. Paginile care urmează au rolul de a compara diversele modalități de construcție a aplicațiilor de acest tip din punct de vedere a usurinței dezvoltării, a performanței și a costurilor necesare. Se vor prezenta diferite arhitecturi folosite în prezent aducându-se argumente în favoarea sau în detrimentul fiecăreia dintre acestea cu rolul final de a prezenta o soluție viabilă dar încă neadoptată la scară largă prin intermediul construirii unei aplicații complete.

Subiectul tezei este aplicația „Doctorul Meu” care înglobează soluția menționată. Aceasta propune o modalitate de a rezolva neajunsurile procesului clasic de programări la specialiștii din domeniul medical. Sistemul gestioneaza calendarul clinicilor aducând avantaje notabile, atât acestora care pot beneficia de o mai bună organizare internă, cât și pacienților care au parte de o calitate a desfășurării serviciilor îmbunătățită.

## 1.2 Context

Dezvoltarea accentuată a tehnologiei informației a permis adoptarea soluțiilor software pretutindeni, astfel încât tot ce ne înconjoară depinde în proporții mai mici sau mai mari de produse informatice, unelte care vin în ajutorul nostru raspunzând unei nevoi specifice.

Curentul actual de a construi aplicații ușor de însușit de către utilizatorul final, este de a folosi cât mai mult web-ul datorită faptului că este accesibil aproape de oriunde, folosirea acestuia nu depinde de un anumit dispozitiv, iar tehnologiile din prezent sunt suficient de evoluate pentru a crea funcționalități complexe care altădată necesitau o implementare specifică de tip mobile sau desktop.

În contextul actual românesc, nu multe cabinete particulare dețin o soluție centralizată pentru organizarea informațiilor pacienților, pentru stocarea detaliilor importante relevante în diagnosticare și pentru programări la care doctorul să poată avea acces de pretutindeni. De multe ori pacienții uită sau notează eronat ora la care au fost programați. Astfel aceștia revin cu apeluri pentru detalii suplimentare, ceea ce conduce la risipa de timp valoros al secretarei sau al doctorului. O problemă și mai mare o constituie faptul că pacienții nu mai revin cu un apel ulterior și astfel nu se onorează programarea.

Există numeroase aplicații pentru gestionarea programărilor sau a informațiilor pacienților precum „PracticeSuite”, „Evolvo” sau „AmericanMedical”, dar multe dintre ele nu intră in categoria aplicațiilor web, iar cele care sunt de acest tip au o arhitectura care necesită costuri ridicate constând în mentenanță. Acest lucru reduce din calitatea produsului, deoarece creatorul nu are profit decât după ce încheie suficiente contracte.

„Doctorul Meu” este o astfel de aplicație ce permite doctorilor înregistrați să afle detalii despre pacienții acestora, să îi programeze sau reprogrameze cu ușurință fără a depinde de agenda clasică sau de sistemul de programări pe care multe clinici îl folosesc deja pe calculatoarele din cabinete și spitale, dar care nu sunt ușor accesibile când doctorul sau asistenta nu este în preajmă.

## 1.3 Obiective

„Doctorul Meu” are ca obiective construirea efectivă a unei aplicații care să vină în întâmpinarea problemelor expuse anterior. Aceasta trebuie să ruleze într-un mediu de producție și să poată fi folosită în contextul în care a fost gândită fără a mai fi nevoie de operațiuni suplimentare pentru instanțiere.

Aceasta trebuie să permită scalabilitatea sistemului prin arhitectura folosită și acomodarea la un număr în creștere de utilizatori, fără a fi nevoie de costuri substanțiale.

## 1.4 Descrierea soluției și a metodologiei folosite

Soluția prezentată este folosirea suitei de tehnologii „Cloud” în detrimentul mentenanței mai multor servere fizice de către echipa de programatori. Acest lucru conferă avantajul lipsei achiziționării de echipament hardware inițial, costurile calculându-se în funcție de folosirea echipamentului care este expus către utilizare sub forma unor interfețe sau a unor servicii web.

Arhitectural, microserviciile sunt folosite deoarece permit o independență a componentelor software crescută ceea ce va conferi scalabilitatea necesară unui astfel de sistem.

În lucrare se vor prezenta avantejele folosirii paradigmei „faas - function as a service” tocmai pentru a utiliza microserviciile într-un mod mai facil, cu resurse financiare scăzute și oferind o mentenabilitate crescută. Aceasta naște pseudotermenul de „Serverless” prin care programatorul nu mai are acces la platforma hardware pe care rulează programul, ci doar la anumite aspecte legate de performanța dorită.

Natura acestei arhitecturi de tip „Serverless” presupune folosirea de servicii web pentru a proviziona anumite funcționalități tocmai din dorința de a reduce nevoia de mentenanță. Astfel, aplicația „Doctorul Meu” folosește servicii pentru: rularea aplicației, găzduirea site-ului, rutare, baze de date, autentificare, management de drepturi către resurse .

Suita de servicii „Cloud” aleasă pentru implementarea aplicației este oferită de „Amazon AWS” care oferă o multitudine de soluții aparținând tuturor paradigmelor de cloud computing: „Service as a service”, „Platform as a service”, „Infrastructure as a service”, „Function as a service” fiind una dintre cele mai folosite pe langă „Microsoft Azure” și „Google Cloud ”.

## 1.5 Structura lucrării

Lucrarea este structurată pe mai multe capitole unde voi argumenta contextul dezvoltarii aplicațiilor web la nivel de client folosind paradigma serverless axată pe microservicii. În capitolele următoare voi prezenta implementarea produsului în sine facând referire la tehnologiile și principiile folosite argumentând folosirea lor și făcând referire și la alte modalități de implementare.

Capitolul 3 stabilește ingineria cerințelor produsului ce se vrea a fi dezvoltat iar capitolele 4 și 5 introduc succint noțiunile de serverless, microservicii, cloud computing având ca scop familiarizarea cititorului față de contextul de dezvoltare al produsului software.

Detaliile de implementare impreună cu tehnologiile folosite se găsesc în capitolul 6.

# 2 Contribuții

Tematica lucrării a fost propusă de mine și transmisă mai departe profesorului coordonator care mi-a oferit câteva sugestii în ceea ce privește funcționalitatea ei. Totodată am stabilit tehnologiile folosite in scenariile de utilizare ale aplicației web.

Elementele practice și teoretice folosite în dezvoltarea lucrării provin din experiența proiectelor personale desfașurate pe parcursul facultății, a stagiilor pe timpul verii obținute la companii prin intermediul programului „Stadii pe bune” și din participarea la cursurile și laboratoarele materiilor studiate de-a lungul celor 3 ani de facultate, în special următoarele:

-  Cloud Computing

-  Tehnologii Web

-  Dezvoltarea aplicațiilor web client (CLIW)

- Ingineria programării

-  Programare Orientată Obiect

În elaborarea lucrării m-am documentat și am experimentat diverse tehnologii de actualitate din întregul spectru „Full Stack” dobândind astfel atât abilități inginerești, necesare pentru a alege adecvat instrumentele folosite, cât și de programare pentru a pune în practică noțiunile studiate.

Atât în procesul documentației cât și în cel continuu al implementării, am luat în considerare și am ales între diverse limbaje de programare , „framework”- uri pe partea de server cât și pe cea de client, tehnici de autorizare și autentificare, platforme de dezvoltare și de livrare a codului sursă.

# 3 Ingineria Cerințelor

În contextul prezentat, se dorește a construi o aplicație web, care să rezolve neajunsurile expuse, să eficientizeze procesul programărilor la medici și care să aibă următoarele caracteristici:

- O interfață cu utilizatorul care este modernă, ușor de folosit, intuitivă și care să poată fi accesată atât de pe telefonul mobil cât și de pe un laptop

- Permite accesul unui număr ridicat de utilizatori

- Are funcționalități pentru creearea și căutarea rapidă a pacientului, care să fie mai eficientă decât modalitatea clasică

- Este posibil ca secretara să poată să creeze programări oricărui doctor din clinică asupra cărui are acces

- Conține calendarul detaliat al doctorilor și legăturile necesare între programări și pacienți

- Trimite mesaje automate pacienților când acestora li se confirmă sau modifică o programare

- Permite pacientului să anuleze ușor programarea, anunțând astfel din timp doctorul despre modificare pentru a reduce timpii morți și banii pierduți

## 3.1 Diagrama Use Case

Următoarea diagramă prezintă scenariile de utilizare din punctul de vedere a actorilor principali „doctor” și „clinică”, reprezentând tipul utilizatorilor care folosesc aplicația: medicul și respectiv secretara sau șeful de clinica, care are acces la toate operațiunile doctorilor asociați.

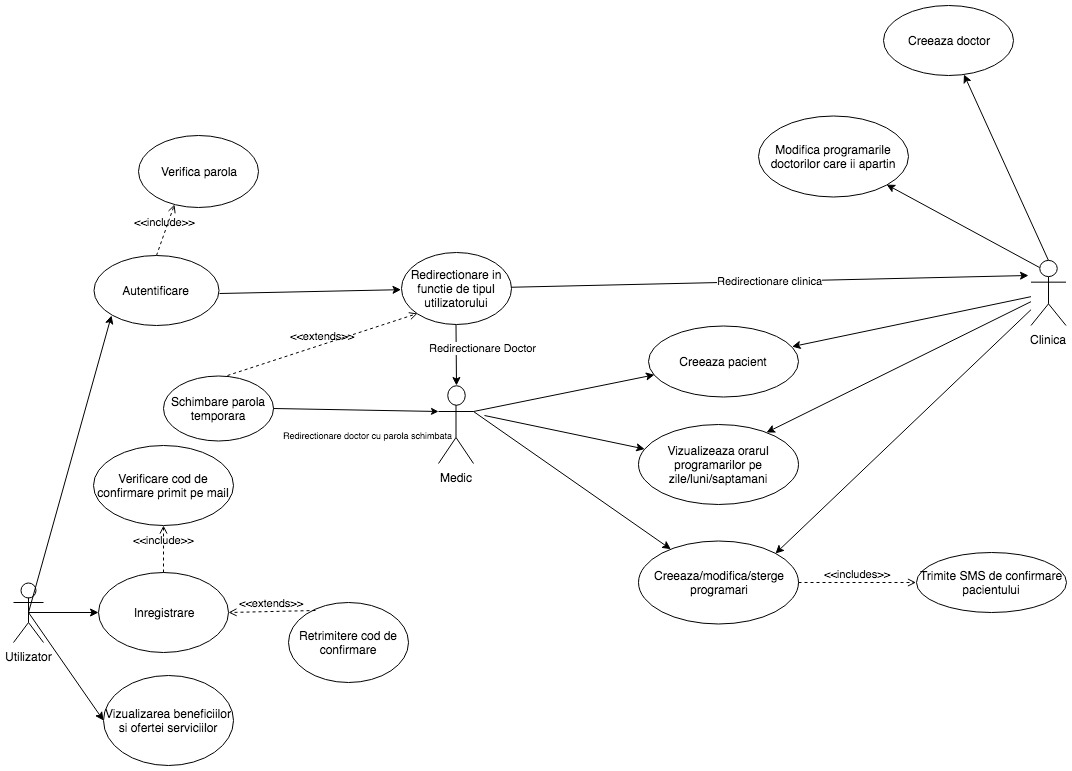
****

Fig. 1 „Diagrama use case”

# 4 Cloud Computing în dezvoltarea aplicațiilor web bazate pe microservicii

## 4.1 Introducere în Cloud Computing

Cloud computing este o paradigmă a teghnologiei informației fiind o generalizare asupra principiilor livrării de produse IT prin abordări arhitecturale care implică o infrastructură de tip grid computing pentru a oferi resurse de calcul sau de stocare într-o manieră cât mai usor de folosit. Folosirea facilă a resurselor se face prin intermediul serviciilor si a interfețelor programatice sau de tip consolă care permit accesul la o suită variată de functionalități fără a fi nevoie de o cunoaștere aprofundată asupra implementării lor. Acestea sunt oferite prin intermediul unui sistem de partajare contra-cost care asigura pe de-o parte managementul intern al resurselor folosite dar și de disponibilitatea sau scalarea  acestora in funcție de nevoile produsului care trebuie creat.

Odată cu dezvoltarea domeniului IT prin creșterea substanțială a puterii de procesare dar și de maturizarea a arhitecturilor de tip grid, domeniul cloud devine unul relevant pentru mediul business al dezvoltării software, ajungându-se la ideea de „careless computing”.

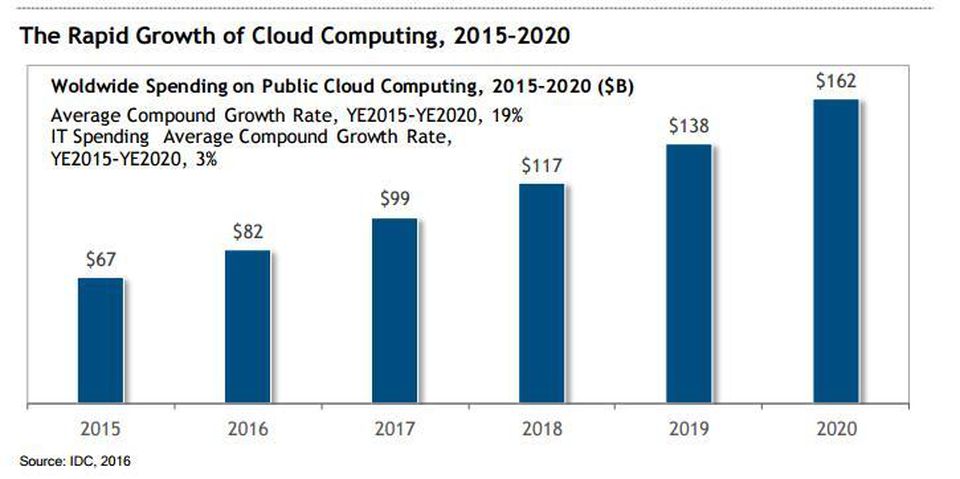
Dinamismul industriei a beneficiat intens de adopția treptată a acestor tehnologii, în ziua de astăzi mare parte din produse fiind construite parțial sau total folosind infrastructuri de tip cloud. Diagrama de mai jos arata întocmai creșterea preferințelor pentru aceste tehnologii.****

Fig. 2 “Ritmul adopției tehnologiilor cloud”

## 4.2 Dezvoltarea web clasica

Orice aplicație web clasică are nevoie de un server pentru a funționa. Acesta are rolul de a servi navigatorului web paginile HTML, foile de stiluri și fisierele javascript pentru ca acesta să le interpreteze și să ofere funcționalitatea interfeței cu utilizatorul. Când serverul primește o cerere de tip HTTP, acesta o proceseaza printr-o componentă numită controller și îi returnează navigatorului web o pagină HTML dinamică în funcție de natura cererii pe care o primește. În timpul procesării serverul se folosește de servicii pentru a face diverse operațiuni necesare precum: citiri sau scrieri din fișiere, inserări sau interogări în baza de date, apeluri către alte servicii externe. Acesta este modul de funcționare clasică al șablonului comportamental MVC ( Model View Controller) care separă accesul la date, vizualizarea acestora și orchestrarea operațiunilor necesare pentru ca acestea să fie procesate.

## 4.3 Microservicii

Termenul de Arhitectura bazată pe Microservicii a apărut în ultimii ani pentru a descrie un mod particular de construcție al aplicațiilor software ca și suite de servicii ce pot fi instanțiate independent unele de altele. Microserviciile au urmatoarele avantaje:

* Fiecare componentă fiind independentă poate fi scrisă intr-un limbaj diferit de programare permițând lucrul la produse de dimensiuni mari mai ușor împarțind munca dintre echipe
* Nu există punct de eșec unitar având posibilitatea de a fi modificate separat sau rescrise fară a compromite integritatea întregii aplicații
* Sunt ușor scalabile datorită independenței crescute și a principiului separării de responsabilitate a acestora

Modul de construire a unei aplicații bazate pe microservicii este urmatorul: se împarte aplicația în module cu responsabilități diferite și se construiește câte un serviciu de dimensiuni reduse, cât mai unitar, pentru fiecare tip de funcționalitate din cadrul acestora. Fiecare microserviciu are propria lui bază de date daca are nevoie să stocheze informații și are nevoie de propriul mediu de execuție. Comunicarea între servicii se face folosind mesaje REST sau SOAP.

## 4.4 Rolul Cloud în dezvoltarea web bazata pe Microservicii

Deoarece mediile de execuție sunt independente, microserviciile trebuie instanțiate diferit, fiecare dintre ele având propria lui suita de resurse necesare pentru a rula. În acest context a apărut virtualizarea, ceea ce permite replicarea independentă a unor resurse hardware sau software. Instanțierea acestor microservicii nu mai depinde de o singura mașina de calcul ci de mai multe sau de mai multe medii virtualizate capabile să ruleze pe aceeași masină dar izolat.

Serviciile de tip platformă în cloud permit instanțierea de componente virtualizate precum „docker” sau „kubernetes” oferind disponibilitate crescută.

Datorită inițiativei pentru mentenanță scăzută, o aplicație complexă ar putea fi construită folosind servicii de baze de date, de echilibrare a sarcinii sau de calcul. Toate acestea dar și altele fac parte din suita de servicii disponibile in cloud, servicii necesare pentru o posibila scalare a aplicației fără a cumpara fizic mașini hardware, a reconfigura manual setarile mediului și fără a angaja o persoană specializată pentru a menține disponibilitatea crescută a sistemului.

# 5 Aplicații serverless

## 5.1 Dezvoltarea aplicațiilor web la nivel de client

Contrar dezvoltării web clasice descrise în capitolul 4.2 în care construirea conținutului dinamic constând în fișiere HTML se face pe server, în dezvoltarea web la nivel de client paginile se construiesc direct în browser prin intermediul limbajului javascript. Browserul face cereri HTTP, dar de această dată serverul îi răspunde cu mesaje conținând diverse obiecte, proprietăți care trebuie să fie injectate în DOM.

Deoarece serverul nu mai trimite pagini în funcție de o ruta prestabilită, programul javascript trebuie să își defineasca propriile rute ca și resurse.

Unele aplicații nu folosesc un server propriu pentru a primi mesaje și se folosesc strict de alte servicii care sunt accesate direct din codul browserului clientului.

Pentru creearea de componente, reutilizarea lor, rutarea in browser și abilitatea de a face apeluri HTTP într-o manieră organizată, am ales să folosesc Angular 5 ca și framework de javascript, acesta fiind una din principalele tehnologii folosite actual pe langa React și VueJS.

## 5.2 Angular 5

Angular este un framework de javascript folosit pentru dezvoltarea aplicațiilor de tipul „single page app”, aplicații care dau senzația utilizatorului că există o singură pagină, aceasta neactualizându-se odată cu trimiterea formularelor sau cu redirecționarea către o altă rută. Apelurile către server se fac asincron folosind modulul HttpClient iar eventualele date primite ca răspuns sunt injectate în template-ul paginii HTML.

Typescript este supramulțimea limbajului javascript folosită de angular, fiind adus să imbunătățească sistemul de autocompletare a codului datorită faptului că este un limbaj restrictiv la tipul variabilelor.

Un concept important adus de această tehnologie este cel de controller la nivel de browser, care este strâns legat de view, acestea două împărțind variabile dacă acest lucru este dorit. Alt lucru care face Angular o opțiune bună este conferit de sistemul de injectare a dependențelor, prin care acestea sunt decuplate, aplicația având o coeziune sporită.

Ierarhia sistemului de fișiere este una bine structurată ceea ce ajută programatorul să se organizeze mai eficient. Spre exemplu, fiecare componentă are propria lui foaie de stiluri, care nu permite coliziuni nedorite de selectori între elemente diferite din componente diferite.

Rutarea în browser este facilitată de fișierul app.routing-module.ts unde se definește ierarhia rutelor și componentele care ar trebui activate pentru fiecare rută parțială. Mai jos se poate observa o componentă care include un meniu de navigație

și care prin marcatorul router-outlet va insera codul HTML al componentei care a fost activată în urma rutării.

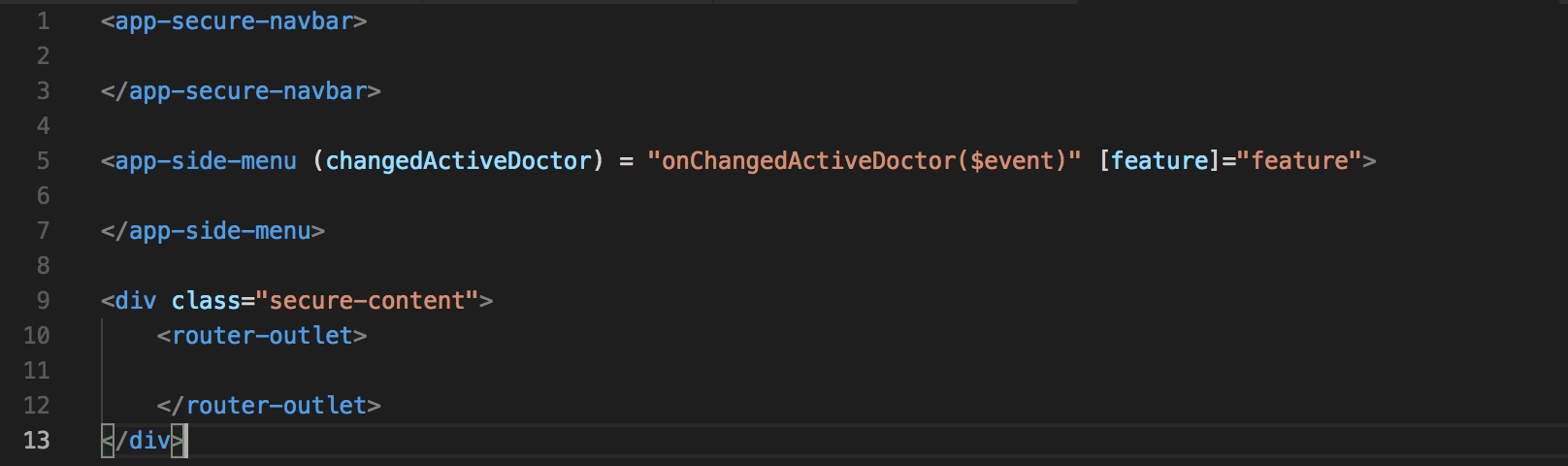


Fig. 3 “Rutarea în angular 5”

## 5.3 Paradigma Serverless

Arhitectura serverless se referă la aplicații care depind în proporție mare de servicii externe sau de cod care rulează sub paradigma „faas - function as a service”.

„Function as a service” reprezintă un mod de a rula un program fără a decide sau a avea grijă de mediul în care executat. Furnizorii de astfel de servicii expun o interfață prin care programul poate fi încărcat și una prin care acesta poate fi rulat, clientul fiind nevoit să plăteasca timpul efectiv de calcul și resursele pe care funcția le-a folosit.

Fiindă utilizatorul nu cunoaște detaliile stratului fizic ale serverului, acesta nu trebuie sa își pună problema scalabilității. Aplicația va fi alocată cu mai multe resurse pe parcurs pentru a face față nivelului cerut de capacitate, acesta fiind putere de procesare, memorie, spațiu etc. Aceasta este tolerantă la eroare deoarece platforma este construită astfel încât să reaprovizioneze serverul cu alte noduri dacă cele existente eșuează.

Alt avantaj major ar fi faptul că deținătorul aplicației nu platește pentru când aplicația nu este folosită, aducând beneficii financiare, acesta netrebuind să comande hardware mai performant cu o marjă pozitivă de eroare. Dacă aplicația nu rulează costurile sunt 0, iar dacă aceasta atrage trafic suplimentar, funcțiile vor primi mai multe resurse menținând ridicată disponibilitatea.

Deși conferă multe avantaje, adopția acestei paradigme ca și o soluție completă este încă mică, multe proiecte folosind doar parțial „function as a service” pentru anumite funcționalități bazate pe evenimente („event driven”).

# 6 Detalii de implementare

## 6.1 Limbaj și mediu de execuție

„Doctorul Meu” folosește ca și mediu de execuție „AWS Labmda” care este propunerea celor de la Amazon pentru „FaaS”, acesta fiind oferit la un preț accesibil de numai $0.0000002 pe cerere.

Modul clasic de lucru pentru a încărca o funcție lambda este ori prin intermediul consolei web ori prin interfața liniei de comandă AWS unde se pot urca programe Python, Java,NodeJs C# și Go.

Limbajul pe care am ales sa îl folosesc este extensia Node pentru Javascript. Acesta permite rularea pe server a unui limbaj inițial folosit numai în browser, adăugând accesul la sistemul de fișiere prin module scfise în C/C++ care comunică direct cu sistemul de operare. Javascript este un limbaj asincron, ceea ce înseamnă că ordinea scrierii instrucțiunilor nu este întotdeauna cea a execuției lor, făcând programarea în acest limbaj să fie bazată pe funcții de callback.

Asincronismul este unul din motivele principale ale alegerii mele, deoarece imi permite cu ușurință să fac mai multe apeluri la servicii în același timp și să aștept răspuns de la toate.Pentru a ușura lucrul am folosit pe cât s-a putut sistemul de promisiuni din versiunea ES6 a limbajului care permite codului sa fie mai citibil și îi oferă o ordine mai logică a scrierii sale.

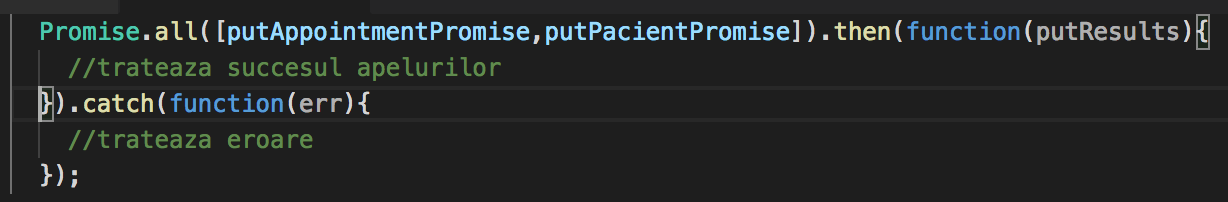


Fig. 4 „Promisiuni in ES6”

## 6.2 Rutare API Gateway

API Gateway este tot un serviciu AWS, care are rolul de a creea rute pentru diferite servicii care se vor a fi expuse. Poate fi folosit ca o abstractizare pentru accesul microserviciilor undei aplicații. În aplicația de față, acesta este folosit pentru a corela o anumită funcție lambda de un URL pentru a putea invoca programul dorit printr-o cerere HTTP.

API Gateway permite securizarea apelurilor prin autorizatori definibili din consola. Aceștia pot fi autorizatori incluși de alte servicii, cum ar fi AWS Cognito sau autorizatori lambda definibili de către utilizator, care preiau cererea și o validează în funcție de un token primit ca header HTTP, de natura requestului sau de parametrii acestuia.

În imaginea de mai jos se pot observa rutele care corespund unei acțiuni HTTP pentru fiecare dintre serviciile definite de aplicație, cât și autentificatorul care permite sau respinge cererile făcute către URL-ul corespunzător fiecărui serviciu.

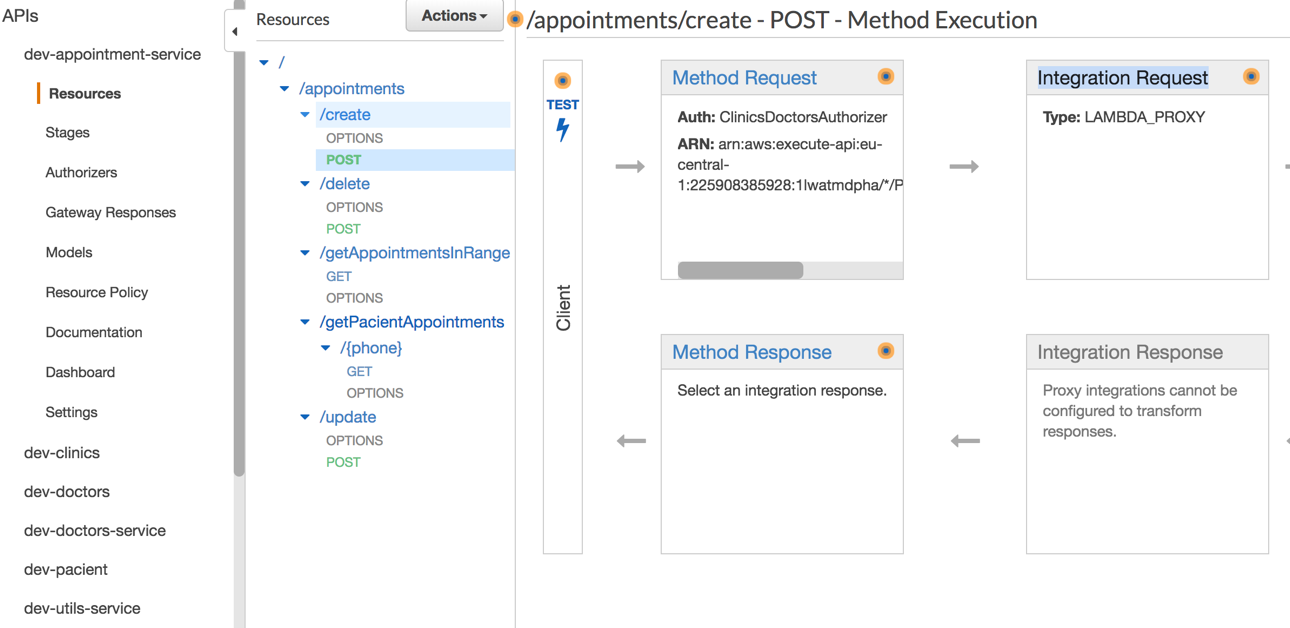


Fig. 5 “Rutarea în consola API Gateway”

## 6.3 Autentificare și autorizare

Există mai multe mecanisme de autentificare a utilizatorilor folosite în dezvoltarea aplicațiilor web. Cel mai răspândit este cel bazat pe sesiune în care serverul menține starea unui utilizator de la care primește un token care este ținut ca și un cookie. Acesta este trimis automat către server odată cu cererea HTTP iar serverul decide autorizarea cererii în funcție de validarea autentificării.

Deoarece aplicația construită este una serverless, nu putem avea sesiuni. Am putea implementa un mecanism asemănător folosind o bază de date la distantă, dar ar există soluții mai bune.

Autentificarea prin „Bearer token” este răspunsul pentru aplicațiile care sunt lipsite de stări. Clientul primește un token generat pe care il folosește in rânduri repetate impreună cu numele de utilizator pentru a obține un token de access specific acestuia. Tokenul de acces este trimis în cererile HTTP , de obicei ca și header, iar serverul il validează folosind un mecanism criptografic.

### 6.3.1 AWS Cognito

Deși există numeroase librarii javascript care se ocupă cu generarea și decodificarea token-urilor web, am decis sa folosesc încă un serviciu AWS numit Cognito User Pools. Acesta este un serviciu complet de înregistrare, autentificare și autorizare permițând crearea de mulțimi de utilizatori corelate cu una sau mai multe aplicații.

Înregistrarea utilizatorilor se poate face prin email sau prin numarul de telefon fiind necesară și o parolă. „Doctorul Meu” folosește autentificarea prin email. După ce utilizatorul se înregistrează, serviciul îi trimite automat prin AWS SES un email cu un mesaj de întâmpinare prestabilit în consola Cognito impreună cu un cod de verificare pe care utilizatorul trebuie să îl introducă în aplicație pentru a confirma înregistrarea.

Autentificarea se face similar, clientul primind tokenul de access dupa ce serviciul verifică identitatea tokenului bearer.

Apelurile către Cognito sunt făcute direct din browser, fără a mai trece prin AWS lambda. Aici este prezentat codul care autentifică un utilizator și trateaza autenificarea cu succes, caz în care primește un callback catre o metodă care setează sesiunea Cognito cu tokenurile primite ca răspuns.

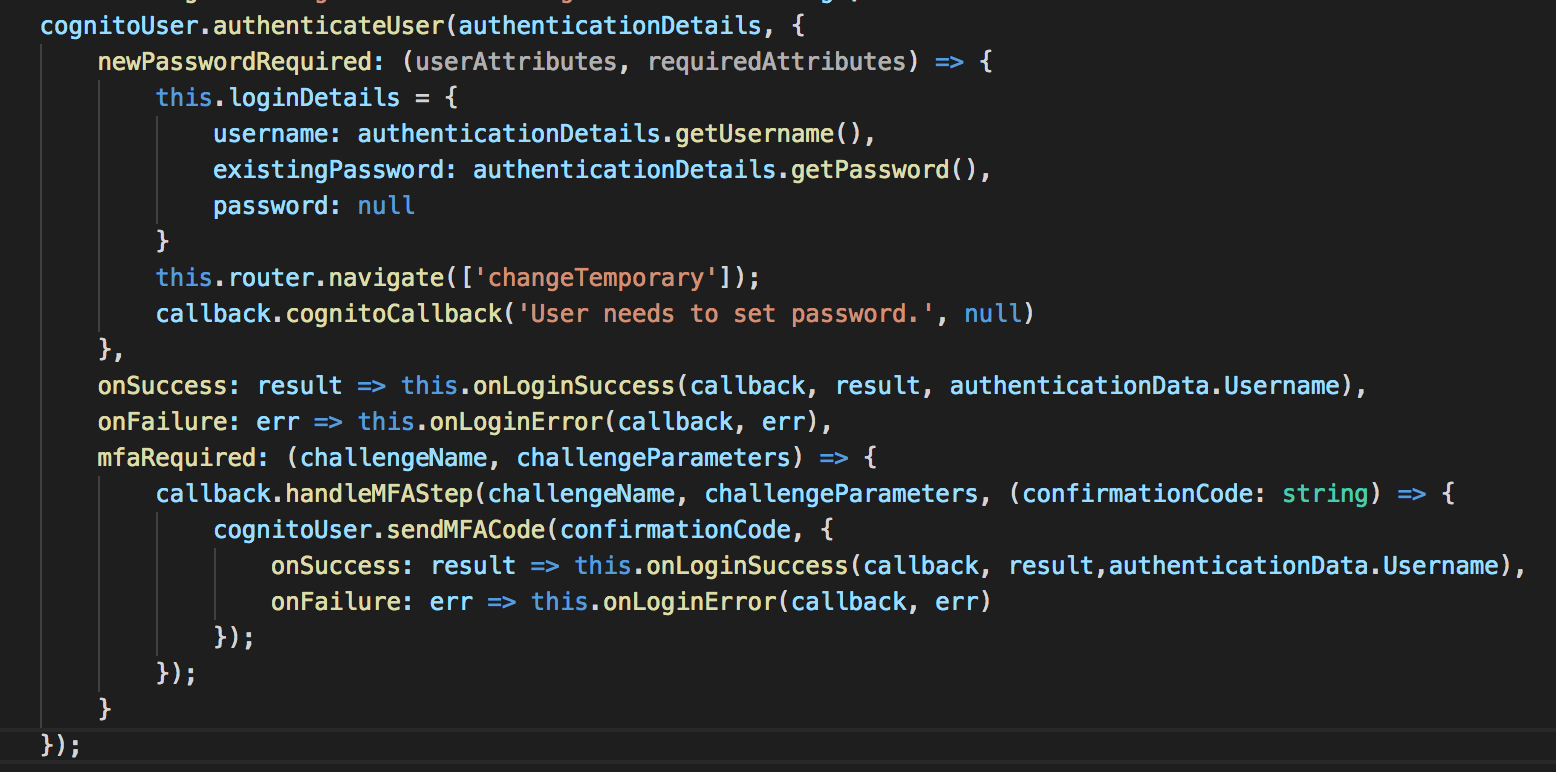


Fig. 6 “Autentificarea cu Cognito User Pools în Angular”

Metoda onLoginSuccess redirecționeaza utilizatorul în funcție de grupul cognito din care face parte verificând câmpul groups care a fost setat la înregistrare.



Fig. 7 “Redirectarea în funcție de grupul cognito în Angular”

## 

## 6.4 Arhitectura aplicației

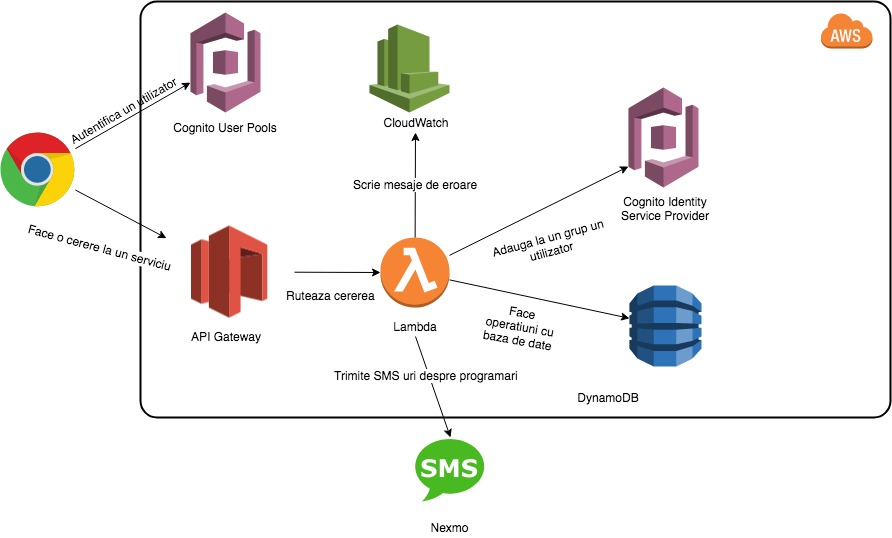


Fig. 8 “Diagramă arhitectură servicii”

### 6.4.1 Rolul serviciilor

Aplicația folosește o arhitectură de tip serverless, comunicând cu diverse servicii pentru a prelua și a stoca informația. Acestea sunt urmatoarele:

**Utilis Service** are rolul de a comunica cu baza de date pentru a aduce tipurile de clinici și de doctori.

**Clinics Service** are rolul de a creea și modifica utilizatorii de tip clinică. Când clientul deschide formularul de inregistrare a unei clinici, browserul cere prin **Utilis Service** tipul clinicii iar mai apoi, după ce utilizatorul a completat formularul se face cererea către API Gateway. API Gateway rutează cererea HTTP către funcția lambda corespunzătoare care lansează simultan două apeluri asincrone: Pe de-o parte creează utilizatorului și il adaugă pe acesta la grupul de clinici folosind Cognito, iar pe de altă parte adaugă detaliile clinicii în tabela din DynamoDB.

**Doctor Service** permite extragerea doctorilor pentru o anumită clinică. API Gateway citește tokenul de access inclus în câmpul Authorization ale cererilor HTTP și populează obiectul context dacă Cognito Authorizerul a validat accesul, altfel răspunde cu mesajul “Unauthorized”. Crearea unui doctor se face tot în acest serviciu. O clinică autentificată poate creea un doctor, acesta primind o parolă temporară pentru prima intrare în aplicație.

**Appointment Service** creează, sterge și actualizează programări. Procesul autentificării cu API Gateway este același, operațiunile modificând tabelele Clinics, Doctors și Apppointments lansând în execuție trei fire diferite.

## 6.5 Nivelul de stocare

Orice produs software are nevoie de o bază de date. Actual există două tipuri de baze de date care se pretează acestui tip de aplicație, cele relaționale și cele neraționale, deși există și baze de date bazate pe grafuri.

Am ales să folosesc bazele de date nerelaționale deoarece consider că avantajele aduse de acestea asupra sistemului clasic relațional se pretează bine contextului aplicației. Bazele de date nerelaționale numite și baze de date NoSql nu trebuie să scaleze vertical, însemnând că odată cu dezvoltarea popularității produsului dezvoltat, hardware-ul nu trebuie imbunătățit exponențial. Acest lucru face costurile să rămână scăzute în comparație cu bazele de date relaționale oricât de mult ar scala aplicația.

Alt avantaj important este faptul că îm NoSql se pot stoca date nestructurate, acest lucru făcându-se dupa sistemul cheie – valoare. Aces lucru permite ca pe viitor, spre exemplu, tabela care conține datele pacienților să aibă obiecte diferite ca și structură în funcție de cerințele fiecărui doctor. Acesta ar putea avea opțiunea de a defini niște atribute pe care fiecare pacient trebuie să le completeze pentru a interveni sau nu cu o operație. Acest lucru este compicat de realizat în sistemul relațional. Deși nu imposibil, acest scenariu presupune muncă suplimentară în proiectarea nivelului de stocare.

Nu în ultimul rând, mentenanța este aproape înexistentă când vine vorba de NoSql, pe când la sistemul relațional de multe ori este nevoie de un specialist în administrarea bazelor de date care să optimizeze căutările, să creeze indici, să înțeleagă bine nevoile programatorilor.

Alegerea unei baze de date nerelaționare are desigur și dezavantaje, dar care pot fi rezolvate prin proiectare, atâta timp cât scenariul de utilizare nu necesită explicit folosirea sistemului relațional. Aceste cazuri sunt cele în care căutările complexe după diverse atribute primează sau în care performanța poate lipsi, cum ar fi cazul unor tabele care stochează traseul unui utilizator pe un site pentru a lua decizii de bussiness în viitor.

### 6.5.1 Diagrama de clase

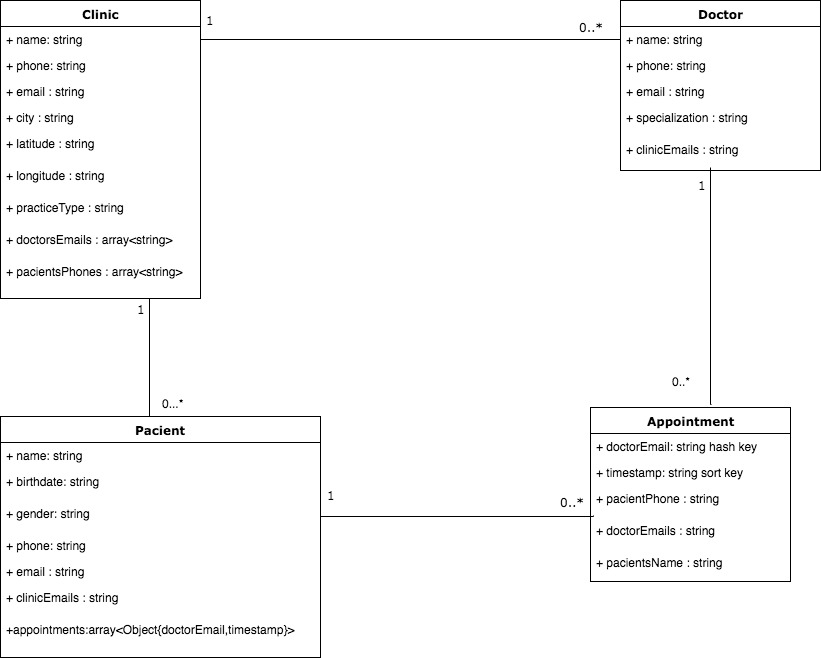


Fig. 9 “Diagrama de clase”

Fiecare inserare sau ștergere din tabela Programări urmează să actualizeze și intrarea din tabela Pacienți corespunzătoare cheii primare compusă din cheia de distribuție și cheia de sortare, astfel încât o căutare care returnează toate programările unui pacient va returna valoarea cheii appointments corespunzătoare acestuia. Aceeași abordare se folosește și pentru a extrage toți doctorii unei clinici.

Acesta este modul de lucru care permite căutări altele decât după cheia primară în tabele NoSql pentru scenarii de căutare restrânse care sunt gândite înaintea programării bazei de date, dar care pot fi extinse prin artificii asemănătoare pe parcurs datorită extensibilității sistemului. Acesta permite schimbarea structurii acestuia oricând, permițând stocări neuniforme.

Tabela de programări este construită având ca și cheie primară ( doctorEmail, timestamp) care permite extragerea tuturor programărilor ordonate ale unui doctor. Dacă un doctor vrea să interogheze sistemul pentru toate programările avute de un pacient anume care urmează să fie tratate de el, trebuie interogat atributul appointments a pacientului rezultatele urmând a fi filtrate după campul doctorEmail.

Pe lângă aceste tabele mai există alte două: PracticeTypes și Specializations care au rolul de a predefini opțiuni pentru înregistrarea clinicii și a doctorului. În funcție de tipul cabinetului sau a clinicii specificate la înregistrare, doctorul poate deține numai specializările conexe acelei arii.

### 6.5.2 DynamoDB

Serviciul care furnizeaza accesul la baza de date NoSql este DynamoDB fiind accesibil prin modulul node aws. Apelurile către serviciu sunt abstractizate folosind obiectul documentClient care oferă o interfață mai prietenoasă programatorului.

## 6.7 Găzduirea site-ului propriu-zis

Asseturile generate de compilarea aplicației angular pot fi stocate pe un serviciu de stocare statică. „Doctorul Meu” ce folosește S3 ( Amazon Simple Storage Service).

### 6.7.1 Cloud formation

Cloud formation este un limbaj comun (YAML) ce permite utilizatorului să își configureze resursele AWS din interiorul unei suite de aplicații setând diverși parametri: Cămpurile ARN ( Address Resourse Notation) setate împreună și cu operațiunea

### 6.7.2 Serverless framework

Pentru dezvoltare am folosit frameworkul serverless care este un utilitar livrat ca și un modul node ce ușurează munca în contextul serverless. Acesta permite modificarea unui fisier YAML, similar cu cel din cloudformation unde progr amatorul își dă drepturi la resurse și iși configurează funcțiile lambda, cât și rutarea lor din API Gateway.

Serverless framework urcă fișierul YAML în cloudformation și codul funcțiilor lambda în S3.

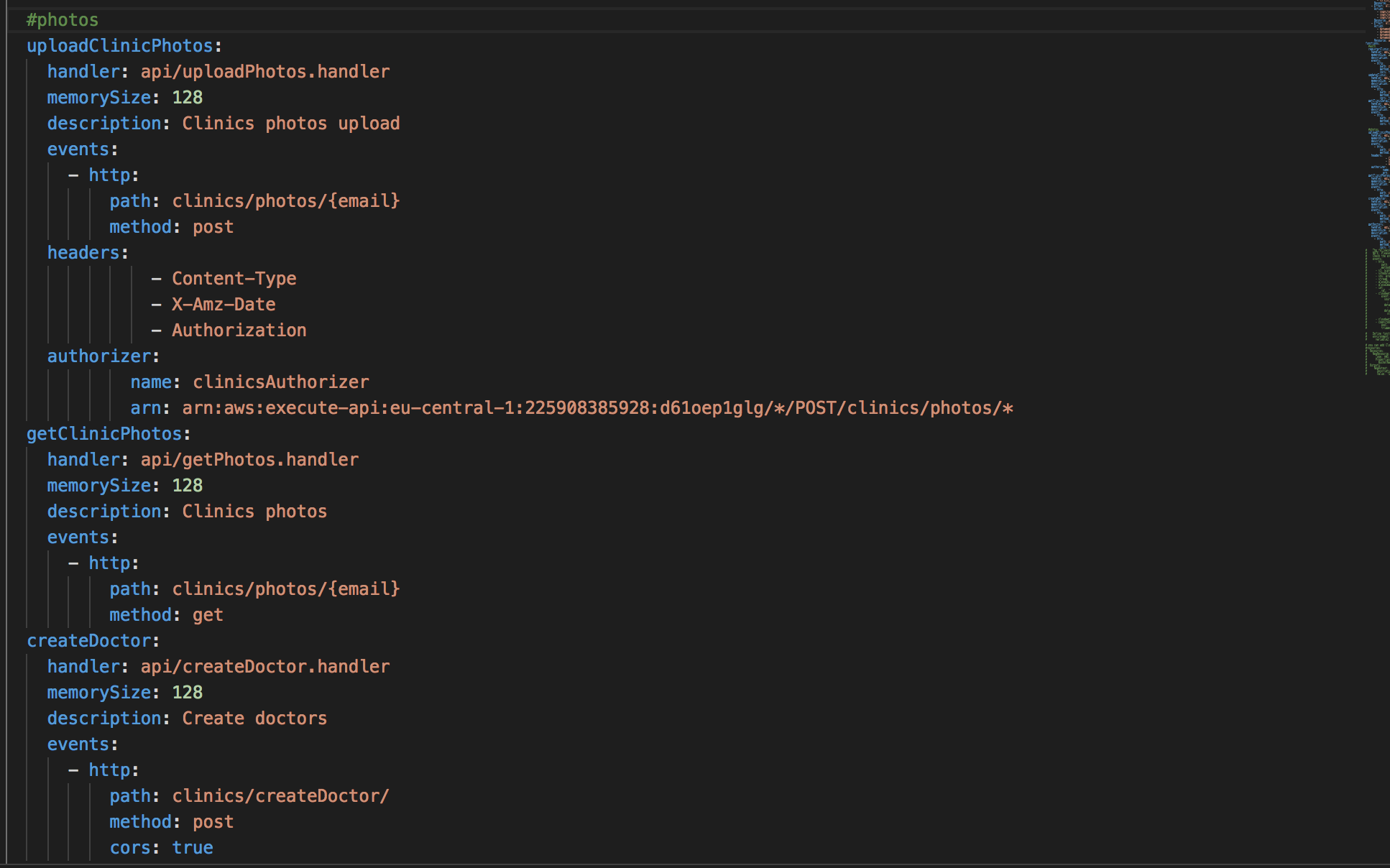


Fig. 10 “Exemplu fișier YAML Cloudformation”

# Concluzii și direcții de dezvoltare

Deși tehnologiile serverless sunt încă la început, iar adopția acestora ca și soluție completă la scară largă este rară, acestea au o performanță bună și pot fi folosite acum într-un mediu de dezvoltare prietenos, intuitiv și cu o comunitate de adepți în creștere.

Costurile reduse vor reprezenta principalul atuu al acestora care v-a conta în alegerea unei tehnologii serverless în detrimentul uneia clasice.

Consider că rata de descoperabilitate asupra serverless este încă mică, avându-și locul în special în operațiunile bazate pe evenimente, loc unde “function as a sercive” excelează.

Proiectul „Doctorul Meu” ar putea fi îmbunătățit adăugând funcționalitatea doctorilor și a pacienților să încarce documente, analize, rețete pe platformă, anexându-le detaliilor une programări. Altă direcție de dezvoltare pe care eu o văd de succes este implementarea sistemului de programări și pe telefonul pacientului, astfel el să poată să facă programarea online, fără a mai suna la telefon doctorul.

# Bibliografie

1. Documentație Angular 5 <https://angular.io/docs>
2. Curs Cloud Computing Lenuța Alboaie <https://profs.info.uaic.ro/~adria/teach/courses/CloudComputing/coursepractical-works.html>
3. Documtație aws <https://aws.amazon.com/documentation/>
4. Arhitectura Serverless <https://www.martinfowler.com/articles/serverless.html>
5. Microservicii <https://www.linkedin.com/pulse/onion-architecture-architects-perspective-tackling-hell-santikary/>
6. Documentație NodeJs <https://www.linkedin.com/pulse/onion-architecture-architects-perspective-tackling-hell-santikary/>