

UNIVERSITATEA DIN BUCUREŞTI



FACULTATEA DE
MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA TEHNOLOGIA INFORMAȚIEI

Proiect de diplomă

IMPLEMENTAREA DE SOLUȚII SERVERLESS CU KAAS ÎN APLICAȚII DIN INDUSTRIA TURISMULUI

Absolvent

Oprea Ana-Maria-Ruxandra

Coordonator științific

Conf. Univ. Dr. Kevorchian Cristian

București, iunie 2023

Rezumat

Având în vedere era tehnologică în care ne aflăm unde informația este la un clic distanță, tot mai mulți oameni își desfășoară activitatea profesională în mediul online, pornind de la educație, până la entertainment. Din aceste motive mediul cloud a devenit un instrument esențial și indispensabil. În lucrarea de față se dorește introducerea în procesul de dezvoltare al soluțiilor întâlnite în producție, care utilizează Azure Kubernetes și cloud pentru a garanta performanță și scalabilitate. Tehnologia aleasă ne permite să rulăm platforma într-un mediu virtualizat, asigurând astfel un timp de răspuns rapid și o experiență fără întreruperi pentru utilizatori. Platforma hostată în interiorul unui cluster Kuberntes a fost proiectată pentru a fi ghidul personal în călătoriile utilizatorilor. Indiferent că este în căutarea unui loc de interes istoric, a unei experiențe culinare autentice sau a unei aventuri în natură, platforma îl va ajuta să descopere cele mai captivante obiective turistice, într-un mod intuitiv și rapid. Prin intermediu aplicației, pot fi accesate ghiduri turistice și imagini impresionante pentru fiecare obiectiv, împreună cu recenzii și comentarii ale altor utilizatori.

Abstract

Given the technological era in which we find ourselves, where information is a click away, more and more people are conducting their professional activities online, starting from education to entertainment. For these reasons, the cloud environment has become an essential and indispensable tool. This paper aims to introduce the development process of solutions encountered in different environments, which use Azure Kubernetes Service and the cloud to guarantee performance and scalability. The chosen technology allows us to run the platform in a virtualized environment, thus ensuring a fast response time and an uninterrupted experience for users. The platform, hosted within a Kubernetes cluster, is designed to be the personal guide in users' journeys. Whether in search of a historical site, an authentic culinary experience, or an adventure in nature, the platform will help them discover the most captivating tourist attractions in a fast and intuitive manner. Through the application, tourists guides and stunning images for each objective can be accessed, along with reviews and comments from other users.

Cuprins

1 Introducere	5
1.1 Motivația alegerii temei	5
1.2 Obiective	5
1.3 Scurt istoric despre Cloud și Kubernetes	6
1.4 Structura lucrării	6
2 Arhitectura aplicației și tehnologiile utilizate	7
2.1 Arhitectura generală a soluției serverless	7
2.2 Integrarea serviciilor din Cloud	8
2.2.1 Azure Kubernetes Service	8
2.2.2 Azure Container Registry	12
2.3 Instrumente de CI/CD pentru AKS	14
2.3.1 GitActions	14
2.3.2 FluxCD	14
2.3.3 Posibilitatea integrării cu Azure DevOps	20
2.4 Arhitectura generală a platformei de turism	21
2.5 Un duo puternic pentru dezvoltarea aplicațiilor web	22
2.5.1 NodeJs	22
2.5.2 React	23
2.5.3 Redux	23
2.6 Persistența datelor cu MongoDB	24
3 Descrierea și implementarea aplicației	25
3.1 Autentificare și autorizare	25
3.2 Gestionează rolurile: admin și utilizator obișnuit	28
3.3 Eficientizarea preluării datelor folosite recurrent cu Redux store	28
3.4 Interfață din perspectiva utilizatorului obișnuit	29
3.5 Interfață din perspectiva adminului	35
3.6 Dockerfiles	37
4 Concluzii	40

Capitolul 1

Introducere

1.1 Motivația alegerii temei

Lucrarea redactată este inspirată din viața personală, călătoritul și tehnologia fiind două dintre pasiunile mele. Integrarea serviciilor cloud a fost o provocare pe care mi-am însușit-o, în ultimii ani tehnologia dezvoltându-se într-o legătură strânsă cu acestea. Companiile mari renunță la infrastructura costisitoare și mențenanța dificilă, apelând la providerii de cloud care se vor ocupa de aceste lucruri în locul lor.

Dintre toate aceste servicii, am fost rapid interesat de aprofundarea capacitațiilor pe care le oferă Azure Kubernetes Service [1]. Clusterele din Kubernetes [2] au renumele că sunt dificil de configurat, iar Azure vine în sprijinul programatorilor prin continua simplificare și eficientizare a tot acestui întreg proces de rulare al aplicațiilor și serviciilor în clustere.

Astfel, fără ezitare am decis să combin cele două pasiuni, prin dezvoltarea unei platforme web de turism prin care să încurajez oamenii să viziteze România și să fie „turiști la ei acasă”, și prin deployment-ul acesteia utilizând servicii ca Azure Kubernetes Service.

1.2 Obiective

Într-o notă generală, aplicația dezvoltată vine în sprijinul turiștilor să își planifice cu ușurință vacanțele și călătoriile online, iar această platformă web le va permite să se informeze în legătură cu acest lucru într-un mod intuitiv și simplificat. Aplicația poate oferi utilizatorilor acces rapid și ușor la informații despre locații, atracții turistice, activități și facilități în diverse zone din România, toată implementarea fiind trecută printr-un proces tehnologic care utilizează servicii din mediul cloud.

1.3 Scurt istoric despre Cloud și Kubernetes

Mediul Cloud reprezintă un instrument revoluționar pentru tehnologie, încurajând industriile să migreze de la dezvoltarea tradițională și locală de soluții, într-un mediu în continuă expansiune, bazat pe internet.

Un pasaj din articolului „Când tehnologia ajunge la nori. Revoluția Cloud Computing. O scurtă istorie a soluțiilor de stocare a informațiilor (VII)” spune că „În 1961, americanul John McCarthy, inventatorul termenului de inteligență artificială, a fost primul care a sugerat ideea unui sistem de schimb de informații între calculatoare, ceea ce a reprezentat primul pas în crearea Cloud Computing-ului. Douăzeci de ani mai târziu, la sfârșitul anilor '70, IBM a lansat un sistem de operare denumit VM care permitea administratorilor sistemului S/370 să aibă mai multe sisteme virtuale pe același sistem principal.” [3]

Cu trecerea timpului, mediul cloud a reușit performanța de a oferi utilizatorilor acces oriunde, oricând la datele sau aplicațiile pe care aceștia le-au urcat, de pe orice dispozitiv conectat la internet.

Kubernetes [2] este o platformă open-source, proiectată inițial de Google și apoi predată *Fundației Cloud Native Computing* în 2015. Scopul primar pe care Kubernetes l-a avut la început a fost de a automatiza deployment-urile, a scală și a gestionă aplicații care rulează în containere.

Pe parcursul anilor, Kubernetes a evoluat, mulți ingineri contribuind la dezvoltarea lui. Performanțele sale au crescut și includ mai multe servicii precum lansări și derulări automate, gestionarea secretelor și echilibrarea traficului.

Astăzi, diferiți provideri de cloud incorporează Kubernetes ca un serviciu care a devenit un instrument indispensabil în industria IT și în marile companii.

1.4 Structura lucrării

Lucrarea redactată este alcătuită din 4 capitole importante.

- Primul capitol face o scurtă introducere în tema abordată și a necesității incorporării mediului și serviciilor cloud în soluții software.
- Al doilea capitol dezvăluie arhitectura procesului de deployment și a platformei web dezvoltate, precum și scurte caracterizări ale tehnologiilor folosite. De asemenea, sunt detaliate serviciile cloud utilizate.
- Cel de al treilea capitol surprinde implementarea aplicației web de turism, cât și funcționalitățile disponibile atât din perspectiva utilizatorului obișnuit, cât și din cea de admin.
- Ultimul capitol prezintă concluzia elaborării acestui proiect, precum și potențialul dezvoltării platformei sau a cluster-ului în continuare.

Capitolul 2

Arhitectura aplicației și tehnologiile utilizate

Pentru un parcurs treptat și coerent, lucrarea de față va introduce în acest capitol arhitectura generală a soluției, atât din punct de vedere al furnizorului de cloud și al serviciilor integrate de la acesta, cât și arhitectura aplicației de turism implementate cu scop demonstrativ al utilizării tehnologiilor propuse.

2.1 Arhitectura generală a soluției serverless

O arhitectură serverless este preferată de dezvoltatori deoarece elimină din stiva atribuțiilor gestionarea infrastructurii, rezultând un proces de implementare și hostare agil al aplicațiilor prin orchestrarea de containere. Însă, chiar și utilizarea de vanilla Kubernetes [2] poate ridica dificultăți pe mai multe nivele, printre care se pot enumera instalarea, configurarea sau securizarea clusterului ce trebuie realizate manual.

Astfel, furnizorii de cloud vin în sprijinul programatorilor, oferind posibilitatea integrării Kubernetes-as-a-Service(KaaS) în care instalarea, crearea și securizarea clusterelor sunt realizate de către aceștia, iar dezvoltatorii se pot focusa pe implementarea programelor și acordând mai puțină atenție infrastructurii.

Suportul oferit de serviciile cloud se regăsește într-o proporție considerabilă în soluția propusă, aceasta bazându-se pe varianta KaaS oferită de Azure, adică Azure Kubernetes Service(AKS) [1]. Mai multe detalii vor fi oferite în secțiunile următoare.

În acest fel, tot fluxul procesului folosirii AKS este reprezentat prin figura 2.1. Pentru a ține o evidență clară a configurațiilor aplicate cluster-ului este folosit FluxCD [4] care va sincroniza la un interval de timp stabilit tot conținutul repository-ului controller cu acesta și va aplica modificările necesare. Un workflow de GitActions [5] este declanșat de fiecare dată când cod nou este adăugat în repository-ul aplicației pe ramura „main” și creează o imagine de Docker [6] care este stocată într-un Azure Container Registry(ACR) [7]. Aceasta va fi preluată de resursele

puse la dispoziție de Flux aflate în repository-ul controller și va actualiza pod-urile din cluster în care se află aplicația containerizată cu ultima versiune a imaginii. În acest mod, programul va fi întotdeauna *up-to-date* și accesibil utilizatorilor.

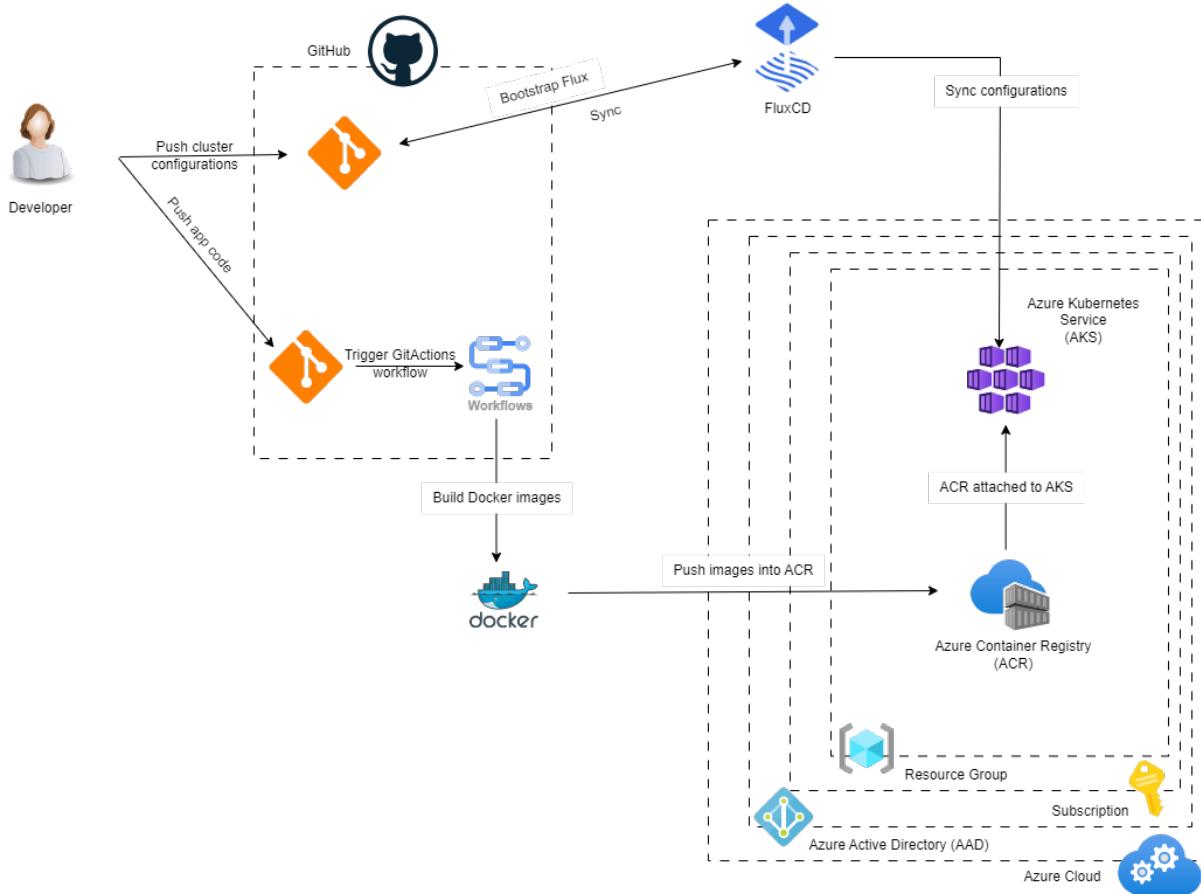


Figura 2.1: Arhitectura soluției utilizând Azure Kubernetes Service

2.2 Integrarea serviciilor din Cloud

2.2.1 Azure Kubernetes Service

Un cluster în Kubernetes este o grupare de mai multe noduri (mașini virtuale) controlate de un nod master, în care se regăsesc unul sau mai multe pod-uri, în care, la rândul său, rulează unul sau mai multe containere. Un container este cea mai mică structură din cluster care virtualizează resursele unui computer, transformându-le apoi în instanțe cu ajutorul cărora pot rula servicii software sau aplicații. Un pod încapsulează unul sau mai multe containere, acestea putând comunica între ele printr-un IP(Internet Protocol) [8] sau printr-un port și localhost. Unui nod îi este atribuit un pod și acesta va rula în incinta nodului atât timp cât execuția lui nu s-a încheiat, nu a fost șters, nodul nu a fost șters, a eşuat sau un controller din nodul master a ordonat întreruperea ciclului de viață al podului.

Câteva din capabilitățile unui cluster Kubernetes:

- **Service Discovery**: diferitele instanțe pot fi expuse și accesate prin adrese DNS (Domain Name System) [9] sau IP-uri
- **Load Balancing**: traficul îngreunat către un serviciu poate fi echilibrat și redirecționat către alte pod-uri replicate ce oferă același serviciu
- **Orchestarea spațiului de stocare**: se poate monta un spațiu de stocare unui pod atât local, cât și în cloud, pentru a facilita persistența datelor
- **Deployment-urile**: pe baza unui manifest, fișier YAML (Yet Another Markup Language) [10], care conține configurațiile specifice instanței care este dorită, Kubernetes se ocupă de adevărata implementare și o creează
- **Set de replici**: se creează un număr extra specificat în manifest de replici pentru a asigura disponibilitatea instanței în caz de fail sau trafic îngreunat
- **Self-healing**: instanțele se autorestartează dacă este nevoie
- **Secrete și ConfigMaps**: instanțe speciale pentru păstrarea datelor cu caracter sensibil, cum ar fi parole, tokeni, pentru buna funcționare a serviciilor dependente de acestea

AKS este un sistem de orchestrare al containerelor bazat pe instrumentul open-source Kubernetes, care este în special folosit datorită abilităților simplificate de gestiune, deployment și scalare ale aplicațiilor containerizate. Azure oferă o preconfigurare a clusterului, ocupându-se deja de câteva aspecte operaționale, cum ar fi monitorizarea sănătății și menținerea clusterului.

Pe lângă capabilitățile amintite mai sus, mai putem adăuga câteva din cele specifice unui cluster AKS:

- **Azure Active Directory(AAD)** [11]: utilizarea acestui serviciu oferit de Azure simplifică gestionarea rolurilor de acces în ecosistemul clusterului
- **Autoscalarea nodurilor**: la configurarea clusterului se poate seta un număr minim și un număr maxim de noduri existente în cluster, iar acestea vor fi puse la dispoziție în funcție de nivelul de procesare necesar, crescând sau scăzând, fiind o practică bună pentru eficientizarea costurilor
- **Monitorizare**: în portalul din Azure se pot vizualiza diferite metriki puse la dispoziție despre cluster
- **Suport pentru upgrade**: în funcție de actualizările aduse la Kubernetes API(Application Programming Interface) [12], nodurile vor fi automat actualizate și ele, fără ca administratorul sau developerul să intervină

- **ACR:** suport pentru atașarea unui ACR într-un cluster AKS, din care deployment-urile își pot trage automat versiunile imaginilor pe baza cărora funcționează
- **Disponibilitatea regiunilor:** aflându-se într-un mediu cloud, există mai multe regiuni (servere fizice pe glob) în care deployment-ul clusterului se poate realiza, alegerea acestuia rămânând la latitudinea administratorului/developerului. Alte servicii ce vor interacționa cu clusterul AKS trebuie deployate în aceeași regiune

Înainte de a putea crea un cluster AKS, trebuie îndeplinite un set de cerințe:

1. Crearea unui cont în Azure
2. Crearea unui AAD cu o configurație personalizată sau folosirea celui default care este creat automat și atribuit unui cont nou
3. Crearea unei subsecvenții care să permită crearea de resurse care pot costa
4. Crearea unui grup de resurse în care vom adăuga ulterior clusterul și ACR-ul

Crearea unui cluster AKS se poate realiza atât din portal, cât și din Azure CLI(Command Line Interface) [13]. În soluția prezentă s-a folosit o combinație între cele două. Crearea inițială s-a realizat în portal (figura 2.2), urmând ca alte permisiuni necesare să fie făcute din Azure CLI. Din figură se pot observa configurațiile disponibile prin furnizorul de Cloud, în soluția prezentată apărând la strictul necesar, majoritatea fiind generate automat.

Create Kubernetes cluster ...

✓ Validation passed

Networking	
Network configuration	Kubenet
DNS name prefix	akslicenta-dns
Load balancer	Standard
Private cluster	Disabled
Authorized IP ranges	Disabled
Network policy	None
Integrations	
Container registry	containerRegistryLicenta
Microsoft Defender for Cloud	Free
Container registry resource group	RG_Licenta_RO
Enable Container Logs	Disabled
Enable Prometheus metrics (preview)	Disabled
Enable Grafana	Disabled
Alerts	Enabled
Alert rules	2 rules
Azure Policy	Disabled
Advanced	
Infrastructure resource group	MC_RG_Licenta_RO_aks_licenta_eastus
Tags	
student	Ruxandra Oprea (Kubernetes service)
student	Ruxandra Oprea (Node pool: agentpool)

[Create](#)
[< Previous](#)
[Next >](#)
[Download a template for automation](#)

Figura 2.2: Crearea clusterului AKS din portal

Pentru a putea controla clusterul din incinta terminalului mașinii locale, o serie de module trebuie instalate:

1. **azure-cli**:

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

2. **kubectl**: instrumentul pentru linia de comandă care susține configurarea unui cluster Kubernetes

```
brew install kubectl
```

3. **kustomize** [14]: un instrument care permite personalizarea manifestelor yaml fără a fi nevoie de şablonane, ține mai bine evidența între resursele create cu scopul de a servi diferite medii de implementare(staging, dev, production, etc.)

```
brew install kustomize
```

4. adăugarea cluster-ului în contextul mașinii locale:

```
az aks get-credentials --resource-group RG_Licenta  
--name aks_licenta
```

După urmarea îndeaproape a acestor pași, putem verifica local din terminal starea clusterului AKS deployat (figura 2.3).

```
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta$ az aks get-credentials --resource-group RG_Licenta_R0 --name aks_licenta  
Merged "aks_licenta" as current context in /home/ruxi/.kube/config  
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta$ kubectl get namespaces  
NAME STATUS AGE  
default Active 15h  
kube-node-lease Active 15h  
kube-public Active 15h  
kube-system Active 15h  
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta$ █
```

Figura 2.3: Verificarea clusterului AKS în terminal

Din această figură se poate observa că sunt patru namespace-uri [15] (un fel de foldere), generate automat:

1. **default**: un namespace gol în care se vor duce toate resursele deployate cărora nu le-a fost specificat un anumit namespace
2. **kube-node-lease**: fiecare nod are un obiect de tip Lease corespondent care monitorizează „semnalul bătăilor inimii” [16] al nodului pentru a-i determina nivelul sănătății
3. **kube-public**: în acest namespace se regăsesc informații generale despre cluster care sunt de tipul read-only pentru toți utilizatorii autentificați, modificarea lor putând fi făcută doar de cei cu drepturile necesare
4. **kube-system**: aici vor exista toate resursele create de sistemul Kubernetes

2.2.2 Azure Container Registry

Aplicațiile containerizate funcționează pe baza logicii încapsulate într-o imagine, în cazul de față o imagine Docker. Astfel de imagini se pot găsi deja preconstruite și gata de a fi utilizate în software-urile dezvoltate. O colecție de imagini open-source poate fi găsită pe platforma [Docker Hub](#).

De asemenea, acestea pot fi folosite pentru a sta la baza unei imagini customizate, create special să servească o anumită soluție software. După ce acestea au fost construite, ele pot fi stocate într-un mediu cloud pentru a facilita accesul la ele și a nu rămâne doar pe mașina locală unde au fost build-uite. Un instrument care vine în ajutorul arhivării, gestionării și construirii

imaginilor este Azure Container Registry. Acesta este preferat în cazul unui cluster AKS datorită integrării ușoare pe care Azure o oferă între serviciile sale.

Crearea unui ACR este una simplă și poate fi făcută atât din portal, cât și din CLI. În cazul de față s-a optat pentru crearea din portal deoarece nu a fost nevoie de un set de configurații avansate (figura 2.4).

Figura 2.4: ACR deployat din portal

Pentru a ataşa un ACR la un cluster AKS, următoare comandă trebuie introdusă în terminal, schimbând denumirile cu cele aferente:

```
az aks update --name myAKSCluster --resource-group myResourceGroup --attach-acr acrLicenta
```

Dar nu este de ajuns doar să adăugăm un ACR la cluster, deoarece pot exista mai multe ACR-uri atașate la unul singur. Pentru a face diferență între acestea trebuie creat un secret (figura 2.5) baza căruia clusterul se va putea autentifica la ACR-ul dorit și va putea extrage imaginile containerizate.

```
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta/aks-infrastructure$ kubectl create secret docker-registry acr-auth --docker-server=acrlicenta.azurecr.io --docker-username=[REDACTED] --docker-password=[REDACTED] --docker-email=[REDACTED] -n licenta
secret/acr-auth created
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta/aks-infrastructure$ kubectl get secrets -n licenta
NAME          TYPE           DATA   AGE
acr-auth      kubernetes.io/dockerconfigjson  1      3s
```

Figura 2.5: Creare secret specific unui ACR

Fluxul automat de construire și push-uire al imaginilor în ACR, precum și extragerea ultimei versiuni de către deployment-urile din cluster, vor fi descrise în secțiunea următoare, fiind concepte de integrare și dezvoltare continuă.

2.3 Instrumente de CI/CD pentru AKS

După acest parcurs, clusterul AKS și ACR sunt deployate și comunică între ele, însă lipsesc resursele care vor hosta și vor face accesibilă aplicația utilizatorilor din online. Cum a fost amintit într-o secțiune anterioară, manifestele redactate în YAML stau la baza creării resurselor. Astfel, pentru a adopta o practică bună de organizare a manifestelor și de sincronizare a configurațiilor clusterului, se va folosi FluxCD.

Pentru a automatiza procesul de containerizare al aplicației și a adăuga întotdeauna ultima versiune a acesteia în colecția din ACR, se va declanșa un workflow de GitActions.

Apoi FluxCD și ACR vor interacționa pentru a extrage și updata automat manifestele deployment-urilor din cluster cu ultima versiune a imaginilor disponibilă găsită.

2.3.1 GitActions

Workflow-ul care face posibilă containerizarea aplicației în imagini Docker se află în repository-ul acesteia, implementare ce va fi detaliată mai târziu în redactarea acestei lucrări.

Într-o notă generală, la fiecare push pe branch-ul „main” se va declanșa fie workflow-ul pentru partea de frontend, fie cel de backend, fie amândouă, depinzând de foldere modificate. Astfel, este citită versiunea actuală a imaginii care este salvată într-un fișier .txt, se conectează cu credențialele aferente păstrate ca Actions Secrets la ACR, construiește imaginea pe baza Dockerfile-ului găsit în același director de lucru și o împinge cu eticheta noii versiuni. În ACR se vor crea două repository-uri, unul pentru imaginile de frontend și unul pentru cele de backend. Acestea vor fi populate cu versiunile incrementate ale aplicației containerizate (figura 2.6)

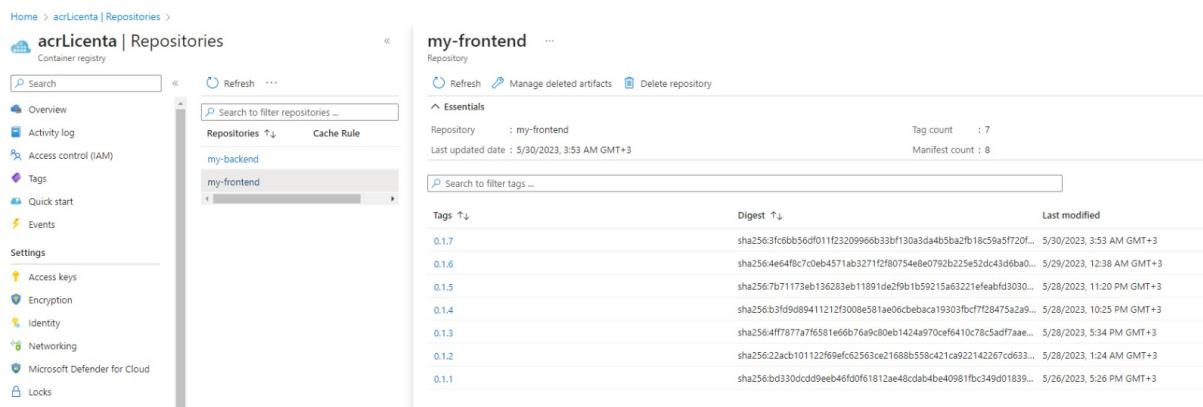


Figura 2.6: Versiunile încărcate în ACR pentru imaginea de frontend a aplicației

2.3.2 FluxCD

FluxCD sau Flux este un instrument puternic care vine în sprijinul utilizatorilor în automatizarea implementării, administrării și monitorizării containerelor din Kubernetes. Flux urmărește

abordarea GitOps [17], adică utilizează Git pentru a gestiona infrastructura și setările aplicațiilor. Printre dezvoltatorii acestui tool se enumeră și românul Ștefan Prodan.

Clusterul va avea un singur repository ca sursă de încredere din care își va lua configurațiile. Acestea nu se vor mai aplica prin comanda *kubectl apply*, ci prin push-uri în acest repository controller, iar la un interval de timp stabilit modificările vor fi sincronizate cu clusterul. Dacă este nevoie ca modificările să fie aplicate imediat, se poate forța sincronizarea prin reconciliere.

```
flux reconcile source git flux-system --verbose  
flux reconcile kustomization flux-system -n flux-system  
flux reconcile kustomization <namespace> -n flux-system
```

Câteva capabilități cheie pe care Flux le oferă sunt:

- **Continous delivery:** deployment-urile în cluster pot fi automatizate, la fiecare schimbare în Git, Flux va sincroniza și aplica noile configurații clusterului
- **Derulări automate (rollbacks):** se poate face rollback la o configurație anterioară în cazul în care sunt descoperite probleme
- **Suport pentru multi-tenancy și diferite roluri pe bază de acces:** se pot crea mai multe tipuri de „chiriași” care vor primi diferite nivele de acces în același cluster, astfel asigurându-se că utilizatorii neautorizați nu pot modifica configurații din afara ariei de acces
- **Observabilitate:** Flux ține evidența tuturor evenimentelor aplicate clusterului
- **Securizare:** doar codul din repository-ul de configurații poate fi aplicat clusterului

Pentru a utiliza Flux în clusterul AKS, acesta trebuie instalat apriori local în terminal. Apoi, se poate realiza comanda de *bootstrapping* care va genera automat legătura dintre repository-ul controller și cluster și va instala un nou namespace *flux-system* în acesta. În cazul în care repository-ul nu există, acesta va fi creat. Un exemplu de comandă pentru bootstrapping se regăsește în figura următoare.

```
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta/aks_infrastructure$ flux bootstrap github \
>   --owner=ixxur \
>   --repository=aks-infrastructure \
>   --branch=main \
>   --path=./clusters/aks-luenta \
>   --personal \
>   --components-extra=image-reflector-controller,image-automation-controller
Please enter your GitHub personal access token (PAT):
> connecting to github.com
✓ repository "https://github.com/ixxur/aks-infrastructure" created
> cloning branch "main" from Git repository "https://github.com/ixxur/aks-infrastructure.git"
✓ cloned repository
> generating component manifests
✓ generated component manifests
✓ committed sync manifests to "main" ("96dcabcbfe0c824fe76ffd74669803bb56db14c8")
> pushing component manifests to "https://github.com/ixxur/aks-infrastructure.git"
> installing components in "flux-system" namespace
✓ installed components
✓ reconciled components
> determining if source secret "flux-system/flux-system" exists
> generating source secret
✓ configured key "edsa-sha2-nistp384 AAAAE2VjZHNhLXNoYTItbmlzdHAzODQAAAIBmlzdHAzODQAAAABhBOATSVQoig6qTZFpaCPIzAUWZ02D4MNW3k7lbIwl66vFTB0dxw3ZdiuYVrrbXn7AvW67cfOJlFGZd5NtQ8ei5ZDPAg5e1wF7wA5zIMTwexWB4cg289+8lnEAbY5nTSQ=="
✓ configured deploy key "flux-system-main-flux-system-./clusters/aks-luenta" for "https://github.com/ixxur/aks-infrastructure"
> applying source secret "flux-system/flux-system"
✓ reconciled source secret
> generating sync manifests
✓ generated sync manifests
✓ committed sync manifests to "main" ("83dc46d9558250dbbb8562d1b102e3d3a88b8881")
> pushing sync manifests to "https://github.com/ixxur/aks-infrastructure.git"
> applying sync manifests
✓ reconciled sync configuration
@ waiting for Kustomization "flux-system/flux-system" to be reconciled
✓ Kustomization reconciled successfully
> confirming components are healthy
✓ helm-controller: deployment ready
✓ image-automation-controller: deployment ready
✓ kustomize-controller: deployment ready
✓ notification-controller: deployment ready
✓ source-controller: deployment ready
✓ all components are healthy
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta/aks_infrastructure$
```

Figura 2.7: Comanda de bootstrap finalizată cu succes

Putem verifica clusterul pentru a confirma faptul că namespace-ul flux-system s-a instalat conform aşteptărilor.

```
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta/aks_infrastructure$ kubectl get namespaces
NAME          STATUS    AGE
default       Active    17m
flux-system   Active    2m27s
kube-node-lease Active   17m
kube-public   Active    17m
kube-system   Active    17m
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta/aks_infrastructure$
```

Figura 2.8: Verificarea generării namespace-ului flux-system

Astfel, resursele pot începe să fie deplyate cu ajutorul lui Flux. În soluția prezentă s-a creat un nou namespace *licență* în care vor fi ulterior implementate deployment-urile, serviciile, ingress-ul, repository-ul pentru imagini și alte resurse necesare funcționării aplicației demonstrative de turism.

O bună practică adoptată este de a genera orice fel de resursă pornind de la un manifest YAML. În acest mod, în repository-ul controller sunt păstrare fișiere pentru namespace-ul licență, deployment și serviciu pentru partea de backend a aplicației, deployment și serviciu pentru partea de frontend, resurse de tipul image repository, image policy, image update automation, ingress și kustomizare pentru namespace. Într-o proporție mare, fișierele urmează aceeași structură, câmpurile fiind completate cu datele corespunzătoare fiecărui tip de resursă ce urmează a fi generat. Un exemplu de astfel de manifest se poate regăsi în figura următoare:

```

aks-infrastructure > infrastructure > licenta > deployment.yaml
 1  apiVersion: apps/v1
 2  kind: Deployment
 3  metadata:
 4    name: backend-deployment
 5    namespace: licenta
 6    labels:
 7      app: backend
 8  spec:
 9    replicas: 2
10    selector:
11      matchLabels:
12        app: backend
13    template:
14      metadata:
15        labels:
16          app: backend
17      spec:
18        imagePullSecrets:
19          - name: acr-auth
20        containers:
21          - name: backend
22            image: acrllicenta.azurecr.io/my-backend:0.1.7 # {"$imagepolicy": "licenta:backend-policy"}
23            envFrom:
24              - secretRef:
25                name: backend-secret
26            ports:
27              - containerPort: 5000
28            env:
29              - name: FRONTEND_API
30                value: "http://frontend-service.licenta.eec8dd5dad2b40b39b69.eastus.aksapp.io"
31

```

Figura 2.9: Fișier YAML pentru a genera deployment-ul backend-ului aplicației

Aplicațiile pot fi create, actualizate, patch-uite sau scalate prin deployment-uri. Datorită capacitatea sale de a controla ciclul de viață a aplicațiilor, este una dintre cele mai importante obiecte din Kubernetes.

Prin manifestul de mai sus, deployment-ul va rezulta într-un set de pod-uri cu numele backend-deployment situat în namespace-ul licență, cu două replici, adică se va mai crea încă un pod la fel pentru a asigura disponibilitatea acestuia. Sub identificatorul *spec* sunt adăugate secretele necesare pentru autentificarea și autorizarea de extragere a imaginilor din ACR, precum și numele imaginii folosite în acest deployment, adică backend-ul aplicației. În *backend-secret* este conținutul fișierului .env al backend-ului care conține credențiale transformate într-un secret. Pod-ul este expus prin portul 5000 și i se atribuie variabilele de environment pentru a facilita conexiunea cu deployment-ul de frontend care este configurat similar.

```
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/Facultate/Licenta/aks-infrastructure$ kubectl describe pod/backend-deployment-85b98b4dcf-8ww7s -n licenta
Name:           backend-deployment-85b98b4dcf-8ww7s
Namespace:      licenta
Priority:      0
Service Account: default
Node:          aks-agentpool-62641163-vmss000001/10.224.0.4
Start Time:    Fri, 26 May 2023 18:49:21 +0300
Labels:        app=backend
               pod-template-hash=85b98b4dcf
Annotations:   <none>
Status:        Running
IP:            10.224.0.24
IPs:
  IP:          10.224.0.24
Controlled By: ReplicaSet/backend-deployment-85b98b4dcf
Containers:
  backend:
    Container ID:  containerd://dc96ecf3d9514652dc35b1bd748ddfb727d91b9683b8090f69605b7a58cb5414
    Image:         acrllicenta.azurecr.io/my-backend:0.1.1
    Image ID:     sha256:47ef969085e6a2da1db9e2a59c6ef0ca029133d82c4becccd373c2e63e6ca2ef6
    Port:         5000/TCP
    Host Port:   0/TCP
    State:       Running
      Started:  Fri, 26 May 2023 18:49:54 +0300
    Ready:       True
    Restart Count: 0
    Environment Variables from:
      backend-secret Secret Optional: false
      Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-5462t (ro)
Conditions:
  Type        Status
  Initialized  True
  Ready       True
  ContainersReady  True
  PodScheduled  True
Volumes:
  kube-api-access-5462t:
    Type:       Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:  kube-root-ca.crt
    ConfigMapOptional: <nil>
    DownwardAPI:   true
    QoS Class:    BestEffort
    Node-Selectors: <none>
    Tolerations:   node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                   node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type  Reason  Age   From            Message
  ----  -----  --   --              --
  Normal Scheduled  99s  default-scheduler  Successfully assigned licenta/backend-deployment-85b98b4dcf-8ww7s to aks-agentpool-62641163-vmss000001
  Normal Pulling   98s  kubelet         Pulling image "acrllicenta.azurecr.io/my-backend:0.1.1"
  Normal Pulled    67s  kubelet         Successfully pulled image "acrllicenta.azurecr.io/my-backend:0.1.1" in 155.397003ms (31.243829001s including waiting)
  Normal Created   67s  kubelet         Created container backend
  Normal Started   67s  kubelet         Started container backend
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/Facultate/Licenta/aks-infrastructure$
```

Figura 2.10: Un pod din setul generat de manifestul deployment-ului backend-deployment

Flux vine în sprijinul automatizării actualizării imaginilor din deployment-uri prin resurse custom ca:

- **Image Repository**: colecție de imagini extrase din ACR
- **Image Policy**: set de reguli ce trebuie îndeplinite ca o imagine să fie validă pentru extragere, în cazul de față eticheta versiunii să fie mai mare
- **Image Update Automation**: este declanșat de fiecare dată când policy-ul găsește o nouă etichetă validă și modifică în manifestul deployment-ului versiunea imaginii folosite cu ultima

```
ruxi@DESKTOP-95DPTN1:~$ flux get image repository backend-repository -n licenta
NAME          LAST SCAN          SUSPENDED      READY      MESSAGE
backend-repository  2023-06-05T00:49:46+03:00  False        True      successful scan: found 7 tags
ruxi@DESKTOP-95DPTN1:~$ flux get image policy backend-policy -n licenta
NAME          LATEST IMAGE          READY      MESSAGE
backend-policy  acrllicenta.azurecr.io/my-backend:0.1.7  True      Latest image tag for 'acrllicenta.azurecr.io/my-backend'
updated from 0.1.6 to 0.1.7
```

Figura 2.11: Image Repository și Image Policy status

Pentru a expune un set de pod-uri cu aceleasi etichete sau selectori și a-l face disponibil utilizatorilor, este necesară crearea unui manifest pentru tipul de obiect abstract Service. O

adresă IP este atribuită service-ului și va rămâne neschimbată pe toată durata de viață a acestuia. Acum backend-ul aplicației poate fi accesat prin comanda de *port-forward* a service-ului, această manieră rămânând tot la nivel local, aşa cum poate fi observat în figurile 2.12 și 2.13:

```
ruxi@DESKTOP-95DPTN1:/mnt/d/Ruxi/facultate/Licenta/aks-infrastructure$ kubectl port-forward service/backend-service 8080:80 -n licenta
Forwarding from 127.0.0.1:8080 -> 5000
Forwarding from [::]:8080 -> 5000
Handling connection for 8080
Handling connection for 8080
Handling connection for 8080
```

Figura 2.12: Comanda care expune serviciul aplicației prin portul 8080

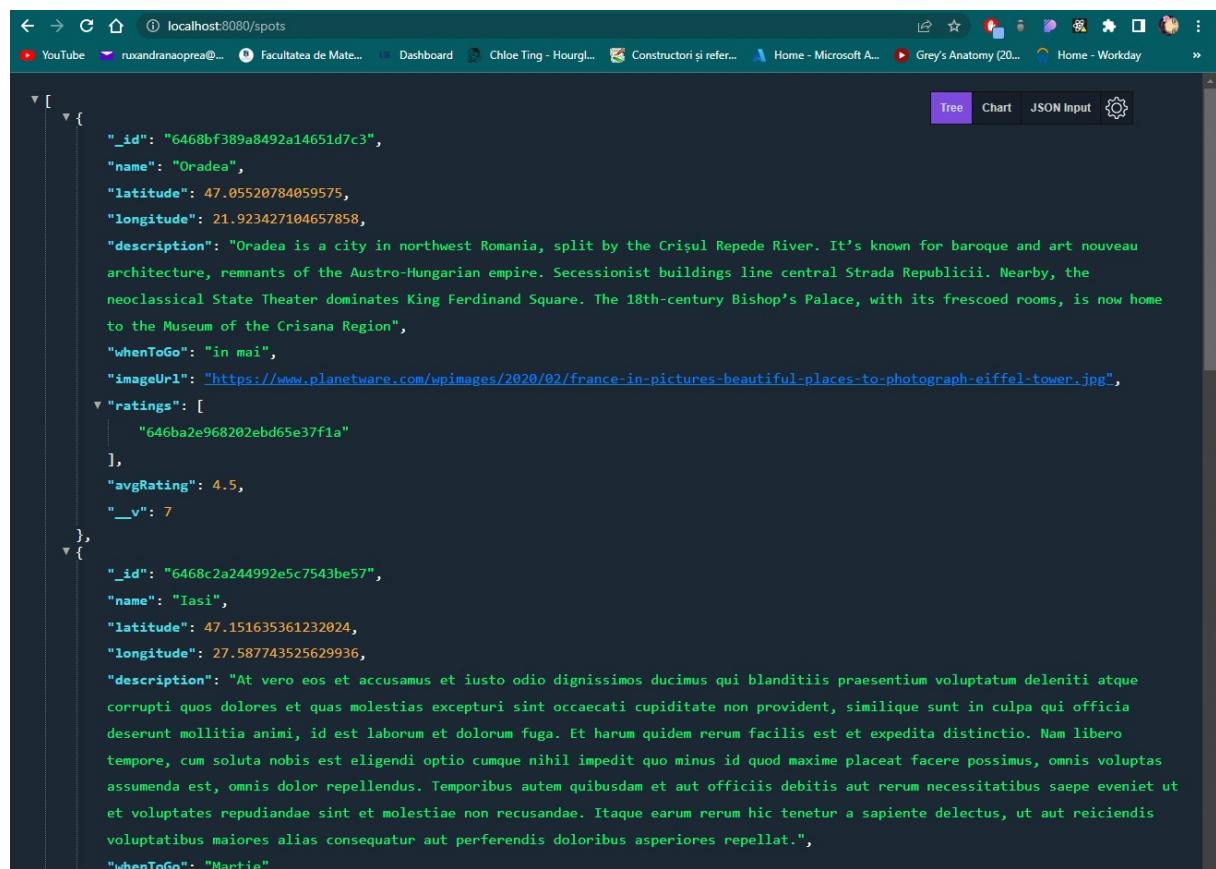


Figura 2.13: Accesarea backend-ului prin localhost:8080

Însă, scopul nu este de a rula aplicația local, ci de a face-o disponibilă utilizatorilor online. Astfel, AKS oferă suportul de activarea *HTTP application routing*. Această soluție generează DNS-uri publice pentru accesarea aplicațiilor deployate în cluster.

Pentru a ataşa DNS-ul generat de serviciile aplicației (backend și frontend), trebuie creat un manifest pentru Ingress. Un Ingress este un obiect API care controlează traficul ce utilizează în mod obișnuit HTTP sau HTTPS pentru a accesa serviciile dintr-un cluster din exterior. Ingress controllerul și Load Balancerul sunt generate automat de la activarea *HTTP application routing*

și nu mai este necesar un manifest și pentru acestea. Azure folosește Nginx [18], un server web des utilizat în livrarea soluțiilor software către clienți.

The service does not have a resource type: Ingress Backend					
NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
backend-ingress	<none>	backend-service.licenta.eec8dd5dad2b40b39b69.eastus.aksapp.io	4.157.65.183	80	7d12h
frontend-ingress	<none>	frontend-service.licenta.eec8dd5dad2b40b39b69.eastus.aksapp.io	4.157.65.183	80	7d12h

Figura 2.14: Ingress-urile serviciilor aplicației și adresele DNS generate

După cum poate fi observat în figura 2.14, serviciile aplicației pot fi acum accesate din browser la adresa aferentă fiecareia, astfel devenind live pentru utilizatori.

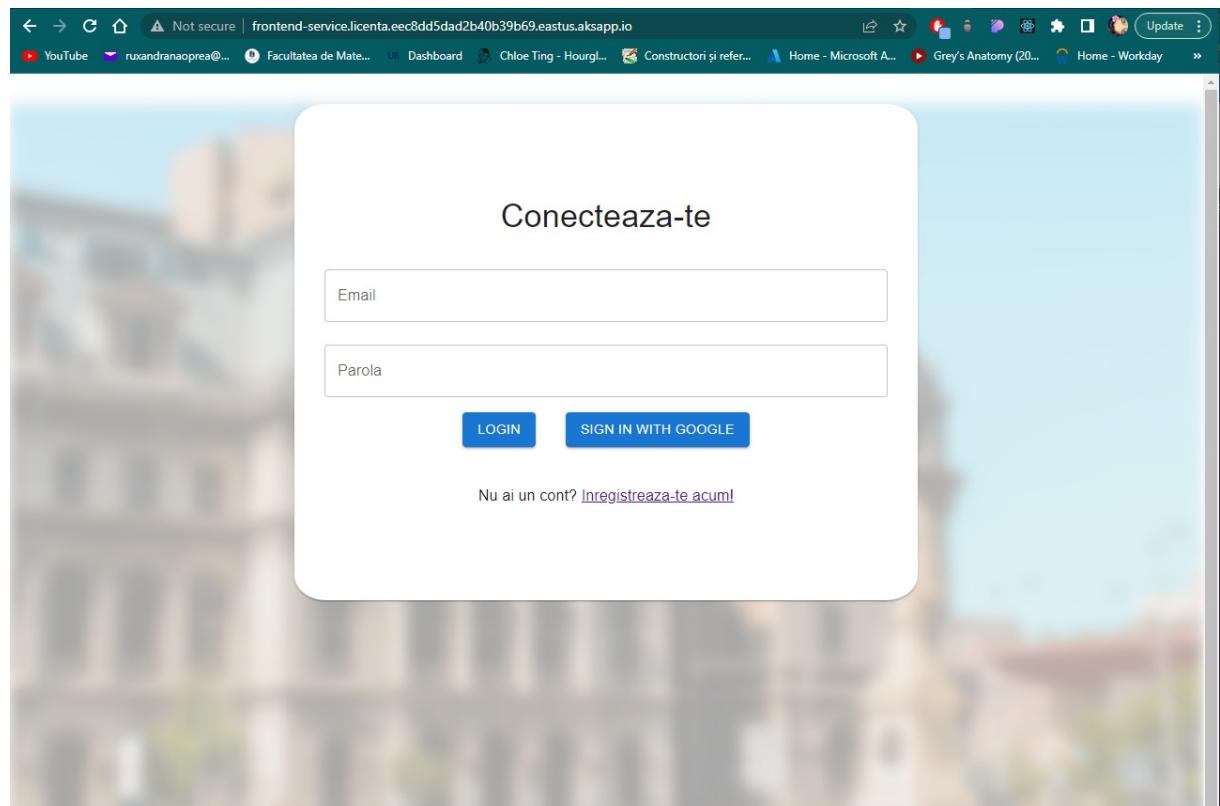


Figura 2.15: Aplicația este accesibilă online

2.3.3 Posibilitatea integrării cu Azure DevOps

În cazul în care este dorită păstrarea întregului flux CI/CD în mediul cloud oferit de Azure, se poate utiliza Azure DevOps [19]. Acest serviciu furnizează instrumente pentru dezvoltarea și menținerea soluțiilor software prin *pipeline-uri*, echivalentul workflow-urilor din GitActions. Azure DevOps poate fi folosit împreună cu FluxCD pentru a implementa un flux de tip GitOps pentru aplicații dezvoltate cu AKS.

O implementare teoretică a Azure DevOps împreună cu FluxCD poate fi:

- Se creează un Azure Repo [20]. La baza lui este un Git repository, cu diferență că va fi situat în mediul cloud oferit de Azure și are posibilități de integrare simplificate cu alte servicii oferite de acesta.
- Se va aplica comanda de *bootstrap* între AKS și Azure Repo pentru inițializarea configurărilor cluster-ului.
- Se va crea un Azure Pipeline [21]. Acesta este un instrument din Azure DevOps care oferă posibilitatea creării unui flux înlăncuit de acțiuni pentru construirea și push-uirea de imagini Docker în ACR. Pipeline-ul se declanșează ori de câte ori apar modificări în codul sursă al aplicației, urmând să containerizeze ultima versiune într-o nouă imagine gata de utilizat.
- După ce imaginea se regăsește în ACR, pipeline-ul actualizează versiunea imaginii folosite din fișierele manifest ale resurselor cluster-ului să o folosească pe cea mai nouă.
- Flux monitorizează schimbările din Azure Repo și va aplica configurațiile actualizate cluster-ului, făcându-l să extragă ultima imagine Docker din ACR și să actualizeze aplicația.

În acest workflow, Azure DevOps este responsabil pentru partea de CI, iar Flux de CD. Această variantă propune o soluție robustă și automatizată de GitOps, toate resursele fiind situate în același environment, însă nu s-a optat pentru adoptarea ei din cauza costurilor implicate integrând mai multe servicii Azure.

2.4 Arhitectura generală a platformei de turism

În această secțiune vor fi introduse noțiuni ale tehnologiilor care stau la baza implementării aplicației web de turism. Aceasta folosește o arhitectură obișnuită formată din două componente principale: componenta server și componenta client care interacționează prin cereri de tipul HTTP (GET, POST, DELETE, PUT), interacțiune ilustrată în figura 2.16. Acest tip de arhitectură este un model de aplicație în care serverul oferă servicii, iar utilizatorul le consumă prin intermediul clientului. Fiind două componente separate, se pot dezvolta independent una de celalătă oferind oportunități de scalabilitate aplicațiilor.

Astfel, atunci când un user accesează adresa atribuită serviciului de frontend și interacționează cu interfața, sunt trimise către componenta server cereri http potrivite la diferitele endpoint-uri implementate pentru acțiunea dorită. Acestea conțin în structura lor un obiect de tip JSON (JavaScript Object Notation) [22]. După procesarea unei cereri în server, va fi transmis un răspuns ce conține la rândul său tot un obiect JSON pe baza căruia componenta client va afișa rezultatul execuției acțiunii întreprinse de utilizator. Acest rezultat poate fi atât unul finalizat cu succes, cât și cu eroare, în funcție de parametrii inclusi în cererea inițială.

Server-ul trimite interogări către baza de date pentru a salva sau primi ca răspuns informațiile solicitate.

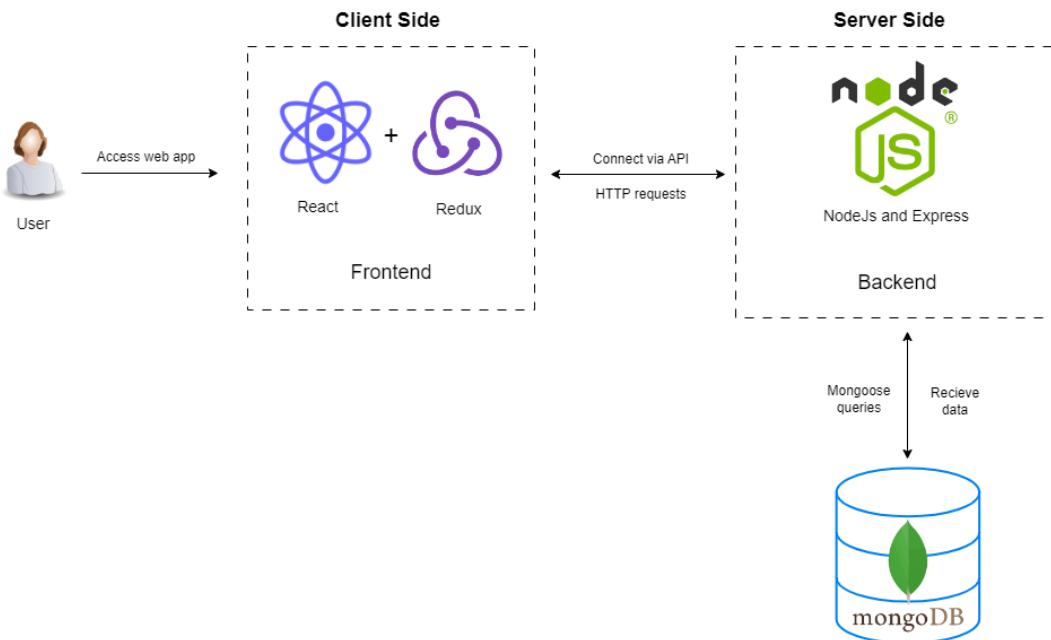


Figura 2.16: Arhitectura aplicației de turism

2.5 Un duo puternic pentru dezvoltarea aplicațiilor web

Există o multitudine de opțiuni atunci când vine momentul să fie alese tehnologiile prin intermediul cărora se vor implementa servicii, aplicații, API-uri, etc. Dintre acestea, conform unui sondaj realizat de *Stack Overflow* în 2022 [23] despre cele mai folosite limbaje de programare și *framework-uri* pentru web development, pe primele două locuri s-au evidențiat NodeJs [24] și React [25]. Integrarea celor două contribuie la un proces de dezvoltare eficient și flexibil deoarece ambele sunt construite pornind de la limbajul de programare JavaScript[**JavaScript**].

2.5.1 NodeJs

- NodeJs este un mediu de rulare (*runtime environment*) care folosește motorul JavaScript V8 al Google Chrome în afara acestuia, astfel devenind foarte performant.
- O aplicație construită folosind Node, rulează într-un singur fir (*single thread*), requesturile putând fi realizate într-o manieră asincronă într-un singur server, evitând concurența thread-urilor și bazându-se că acestea vor întoarce o promisiune că fluxul rulării va reveni după primirea răspunsului.
- Are un ecosistem larg, npm (Node Package Manager) [26] fiind capabil să completeze

funcționalitățile oferite de Node prin incorporarea de software open-source în logica aplicației.

- Este un bun suport pentru microservicii, aplicațiile dezvoltate în Node fiind ușor de containerizat.
- Pot fi create aplicații real-time, adică funcționalități pe baza declanșărilor de evenimente. Astfel de aplicații pot fi servere de jocuri sau servere de chat.

În aplicația prezentă, este folosit framework-ul Express[27] pentru crearea și funcționalitatea server-ului într-o manieră eficientă, Mongoose [28] pentru conexiunea cu baza de date și interogările realizate ulterior, Axios [29] pentru realizarea request-urilor http și multe alte pachete din npm care îmbunătățesc eficiența codului implementat.

2.5.2 React

- Componenta client este dezvoltată utilizând React. Aceasta este o librărie JavaScript folosită pentru construirea de *user interface*, care de cele mai multe ori este un SPA (single-page-application). Conținutul paginii se schimbă dinamic în funcție de interacțiunea utilizatorului cu interfața, fiind mult mai eficient decât încărcarea unei noi întregi pagini, devenind o soluție scalabilă și rapidă.
- Frontend-ul este alcătuit din mai multe componente de UI reutilizabile, fiecare deservind un scop bine stabilit.
- În implementarea componentelor se folosește o combinație de JavaScript și XML (Extensible Markup Language) [30], cunoscut drept JSX (JavaScript XML) [31]. JSX permite introducerea elementelor de HTML [32] în JavaScript și adăugarea lor în DOM (Document Object Model) [33] „fără a apela metode ca createElement() sau appendChild()” [31].
- Rapiditatea procesării este dată de operarea pe Virtual DOM, React realizând prima dată modificările în acesta, iar după actualizându-le și în DOM-ul adevărat.

2.5.3 Redux

- Redux [34] este o librărie folosită de obicei împreună cu React care furnizează un „magazin” (*store*) centralizat pentru state-ul aplicației, adică date care sunt necesare la un nivel global și pe baza cărora componenta client operează. Aceasta se inițializează prin metoda *createStore()*.
- State-ul este doar read-only, deci pentru modificarea acestuia este necesară adăugarea de acțiuni. Astfel, se creează funcții pure de JavaScript, numite *reducers*, care primesc state-ul anterior și acțiunea dorită și returnează state-ul următor.

- State-ul gestionat cu Redux poate fi ușor debugg-uit în Developer Console utilizând extensia Redux DevTools.

2.6 Persistența datelor cu MongoDB

Toate datele cu care aplicația operează vor fi salvate sau extrase din MongoDB [35]. Mongo este o bază de date non-relațională care în locul tabelelor tradiționale, adoptă documente populate cu obiecte JSON care definesc schemele. Toate câmpurile dintr-un document pot fi indexate, astfel interogările sunt mai rapide. Acestea vor fi efectuate cu ajutorul pachetului *mongoose*, care conține metode preimplementate de a filtra, căuta, modifica, șterge sau adăuga date noi în DB.

De asemenea, MongoDB dispune de Atlas, un mediu care elimină dificultățile deployment-ului și gestiunii bazei de date printr-un alt furnizor cloud. Toate aceste aspecte sunt îndeplinite automat, baza de date fiind stocată în cloud cu MongoDB Atlas, devenind disponibilă pentru aplicație indiferent pe ce mașină rulează aceasta.

Crearea schemelor este un procedeu flexibil, putând integra orice fel de date. Aplicația demonstrativă dispune de 4 scheme:

- **User** - conține datele utilizatorului: id generat automat, nume, adresă de email, hash-ul parolei, salt, identificator pentru poza de profil, rolul utilizatorului, array de referințe către locuri favorite, array de referințe către locurile vizitate, array de referințe a rating-urilor date;
- **Spot** - conține datele destinațiilor turistice: id generat automat, nume, descriere, longitudine, latitudine, perioada potrivită de vizitat, url imagine de prezentare, array de referințe către rating-urile acelui loc, media rating-urilor;
- **Rating** - conține datele rating-urilor: id generat automat, username-ul utilizatorului care a dat rating-ul, valoarea rating-ului, referință către id-ul destinației căreia i-a fost făcut rating-ul;
- **Comment** - conține datele comentariilor fiecărui loc: id generat automat, referință către id-ul destinației pentru care este comentariul, username-ul utilizatorului care a lăsat comentariul, textul comentariului, data creării comentariului și data actualizării comentariului în caz de editare al acestuia.

Capitolul 3

Descrierea și implementarea aplicației

În era digitală, tot mai mulți turiști caută să își planifice cu ușurință vacanțele și călătoriile online, iar această platformă web le va permite să se informeze în legătură cu acest lucru într-un mod intuitiv și simplificat. Aplicația poate oferi utilizatorilor acces rapid și ușor la informații despre locații, atracții turistice, activități și facilități în diverse zone din România.

3.1 Autentificare și autorizare

1. Cont normal

Pentru a beneficia de funcționalitățile aplicației, utilizatorii trebuie să aibă un cont cu care să se conecteze. În cazul în care este un utilizator nou, acesta își poate crea un cont din pagina de înregistrare completând câmpurile din formular cu datele necesare. După ce acesta apasă pe butonul „Înregistrare”, un POST request este trimis către server cu datele complete pentru a verifica dacă acestea îndeplinesc criteriile impuse de siguranță. În caz afirmativ se adaugă noul utilizator în baza de date, iar în caz negativ se va afișa în interfață un mesaj de eroare potrivit. Valorile cu care sunt completează câmpurile sunt verificate atât în logica din frontend, cât și în cea de backend pentru a maximiza securitatea conturilor. Email-ul trebuie să urmeze structura unui email adevărat, iar parola să conțină minim 8 caractere dintre care minim unul să fie literă mică, unul să fie majusculă, unul să fie cifră, iar unul să fie caracter special. Dacă aceste condiții nu sunt îndeplinite, logica înregistrării unui utilizator nou nu trece mai departe și afișează un mesaj aferent câmpurilor nevalidate. În caz contrar, pachetul *passport-local-mongoose* oferă suport pentru înregistrarea unui utilizator prin metoda *Model.register(new Model(username, name), password)*. Această metodă se va ocupa automat de adăugarea salt-ului și de hash-uirea parolei. Pentru acest proces, este folosit algoritmul oferit de *bcrypt* care la rândul său are la bază ciprul *Blowfish*[36]. Salt-ul este o bucătă de informație aleatoare. Aceasta este folosită pentru cazul în care doi utilizatori au aceeași parolă, varianta hash-uită va fi diferită datorită salt-ului. În baza de date se vor stoca parola hash-uită și salt-ul. Dacă totul este în regulă, utilizatorul

va fi redirecționat către pagina de Conectare pentru a se conecta cu noul cont creat. Automat toate conturile care se creează din interfață vor primi rolul *regular*, adică utilizator obișnuit.

După introducerea email-ului și a parolei în formularul de pe pagina de Conectare și apăsarea butonului „Conectează-te” fără ca erori să fie afișate pe ecran, un POST request cu aceste date va fi trimis către server. Acolo, se va căuta un user în baza de date care are același email. În caz afirmativ, cu ajutorul metodei *authenticate()* se va verifica parola introdusă în formular hash-uită cu varianta deja stocată în baza de date. Dacă sunt la fel, se continuă procesul de login. Pachetul *Passport* oferă posibilitatea creării unei sesiuni de logare a userului găsit în baza de date prin metoda *login()*. Mai departe, după ce acesta este autentificat cu email și parolă, este creat un token JWT (JSON Web Token) [37] care include id-ul userului, email-ul, rolul, are o dată de expirare de 4 ore și care este semnat cu un secret stabilit de developer. Apoi token-ul este trimis către client ca un cookie care nu poate fi accesat din partea de client și previne atacurile XSS și CSRF. Are o durată de tot 4 ore. Dacă nu a apărut nicio eroare până în acest moment, este trimis un status cu codul 200 către client și detaliile utilizatorului sunt salvate în sesiunea creată. Acum utilizatorul poate naviga prin paginile aplicației și interacționa cu acestea. În partea de frontend, un Redux store [34] este creat pentru a păstra într-un context global datele despre utilizatorul conectat pentru a nu fetch-ui inutil mereu endpoint-uri în componentele unde este nevoie de acestea.

În cazul în care utilizatorul închide browser-ul și apoi după un interval de timp dorește să acceseze din nou aplicația, se verifică dacă în cookie-uri există token-ul creat. În cazul în care este găsit, acesta se va decoda și se va verifica dacă secretul folosit la creare este același cu cel găsit. Dacă cele două coincid, utilizatorul este redirecționat către pagina principală a aplicației. Altfel, dacă secretele nu coincid sau token-ul a expirat și a dispărut, utilizatorul va fi redirecționat către pagina de Conectare și va fi trimis clientului un status cu codul 401 și mesajul „Unauthorized”.

Mai există cazul în care userul și-a uitat parola. Pe pagina destinată schimbării parolei, utilizatorul va introduce datele necesare, precum și noua parolă care trebuie să treacă toate condițiile de integritate. La apăsarea butonului, se trimitе un POST request către server cu noile informații și dacă este găsit în baza de date, se va salva userul cu noua parola și va fi redirecționat către pagina de Conectare reluându-se tot procesul de logare detaliat anterior.

The figure consists of three separate screenshots of web forms, each with a blurred background of green hills.

- Top Left:** A login form titled "Conectează-te". It contains fields for "Email" and "Parolă" (Password). Below the fields are two buttons: "LOGIN" and "CONECTEAZĂ-TE CU GOOGLE". A link "Mi-am uitat parola." (I forgot my password) is located above the "LOGIN" button. At the bottom, a link "Nu ai un cont? [Înregistrează-te acum!](#)" (Don't you have an account? [Register now!](#)) is visible.
- Top Right:** A password reset form titled "Resetează parola". It contains fields for "Email", "Parolă nouă" (New password), and "Confirmă parola" (Confirm password). Below the fields is a single button "RESETEAZĂ PAROLA". At the bottom, a link "Iată aduc aminte parola? [Conectează-te acum!](#)" (Did you remember your password? [Register now!](#)) is visible.
- Bottom Center:** An account creation form titled "Vreau cont". It contains fields for "Nume" (Name), "Email", "Parolă" (Password), and "Confirmă parola" (Confirm password). Below the fields is a single button "ÎNREGISTRARE" (REGISTER). At the bottom, a link "Deja ai un cont? [Conectează-te acum!](#)" (Already have an account? [Register now!](#)) is visible.

Figura 3.1: Formularele de conectare, înregistrare și resetează parola

2. Autentificare cu Google

Pentru autentificarea cu Google se va folosi OAuth2.0 din Google Console. Aici vom înregistra aplicația și adăuga adresele către care trebuie utilizatorul să fie redirectionat, precum și adresa endpoint-ului din server pentru realizarea operației de login. După ce acest set-up a fost realizat, vor fi generate cheile secrete care trebuie integrate în implementarea autentificării pentru a integra *Conectează-te cu Google*. Trebuie declarat *GoogleStrategy* din pachetul *passport-google-oauth.Strategy* pentru a-l putea folosi. Acesta primește două argumente: options - conține credențialele generate în Google Console și adresa de întoarcere, iar cel de al doilea argument este o funcție de verificare. În această funcție sunt primite token de acces, token de refresh și profilul contului de Google cu care se dorește realizată conectarea. Dacă utilizatorul s-a mai conectat în trecut cu contul de Google, datele acestuia i-au fost salvate în bază împreună cu un *googleId*, altfel un nou utilizator este creat. Procesul continuă printr-un GET request prin care folosindu-se aceeași metodă *authenticate()* din pachetul *passport*, se extrag profilul și email-ul user-ului și se trece prin aceeași logică de a genera un token și o sesiune de logare. În cazul în care apar probleme pe parcurs, server-ul va trimite statusuri și mesaje potrivite către client.

3.2 Gestiunea rolurilor: admin și utilizator obișnuit

Aplicația dispune de două tipuri de utilizatori: admin și regular. La înregistrarea noilor useri, acestora li se atribuie automat rolul de regular și este salvat în baza de date, astfel primind acces restrâns la funcționalitățile disponibile. Ca un user regular să devină admin trebuie schimbat manual în baza de date tipul rolului.

Un admin are o libertate mai mare asupra aplicației, putând opta pentru operații CRUD asupra tuturor tipurilor de date. Utilizatorii normali pot doar să filtreze anumite date și au un acces limitat în editarea conținutului.

În server este folosită metoda *authorizeAdmin()* pentru a verifica dacă cel care dorește navigarea către anumite endpoint-uri destinate adminului este autorizat. În aceasta se extrage token-ul din cookie, se decodează și dacă în componență sa rolul utilizatorului este cel de admin, atunci acesta va primi autorizație către acele endpoint-uri. În caz contrar se va trimite către componenta client statusul 401 și mesajul „Unauthorized”, iar userul va fi redirecționat către pagina principală la care are acces.

3.3 Eficientizarea preluării datelor folosite recurrent cu Redux store

Componența client dispune de un state global, inițial gol, gestionat printr-un Redux store. În acest „magazin” vor fi stocate informațiile utilizatorului care s-a conectat, extrase din baza de date după operația de login. Astfel, datele pot fi preluate din orice componentă de UI, oricără de îmbricată ar fi în arborele componentelor din React care alcătuiesc aplicația, fiind o metodă eficientă prin care nu se mai realizează call-uri la endpoint-urile din server ori de câte ori ar fi fost nevoie de acestea.

De asemenea, în partea de logică a Redux store, sunt implementate acțiunile funcționalităților care modifică acest state și care sunt întâlnite în mai multe componente. Acestea sunt:

- *loginSuccess()*: după primirea statusului 200 din partea server-ului, este inițializat state-ul global cu datele primite ca răspuns
- *logout()*: state-ul global este golit, token-ul din cookie este șters și utilizatorul este redirecționat către pagina de Conectare
- *addToFavorites()*: user-ul poate interacționa cu destinațiile turistice, adăugându-le în lista de favorite
- *removeFromFavorites()*: user-ul poate interacționa cu destinațiile turistice, ștergându-le din lista de favorite

- *addToVisited()*: user-ul poate interacționa cu destinațiile turistice, adăugându-le în lista de vizitate
- *removeFromVisited()*: user-ul poate interacționa cu destinațiile turistice, ștergându-le din lista de vizitate
- *updateUserData()*: permite modificarea datelor personale ale user-ului, precum și alegerea unei noi poze de profil
- *postUserRating()*: user-ul poate interacționa cu destinațiile turistice, adăugand sau modificând un rating

Redux oferă posibilitatea de a importa direct funcțiile în componente care au nevoie de logica acestora, eficientizând procesul implementării.

Se poate verifica state-ul global din developer console folosind *Redux DevTools*:

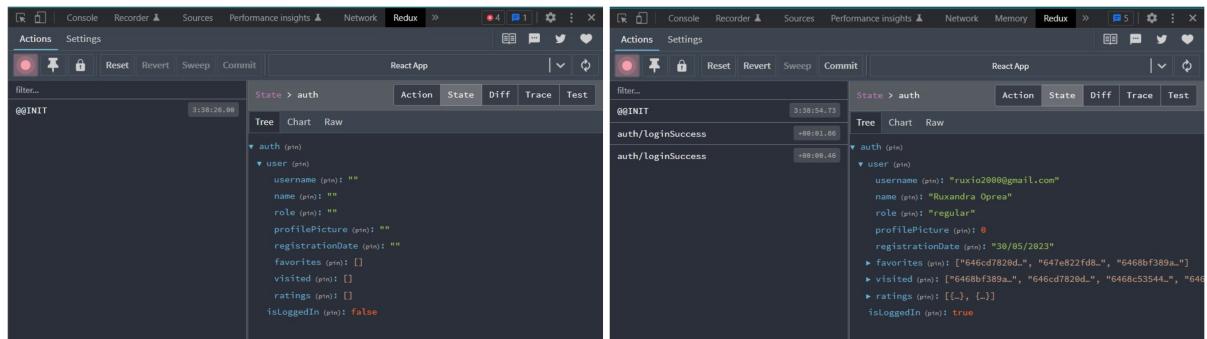


Figura 3.2: State-ul înainte de conectare și după realizarea login-ului

3.4 Interfața din perspectiva utilizatorului obișnuit

1. Pagina principală

După ce tot procesul de logare s-a încheiat cu succes și state-ul global a fost populat cu informațiile acestuia, utilizatorul va fi redirecționat către pagina principală a aplicației unde sunt listate toate destinațiile turistice extrase din baza de date (figura 3.3)

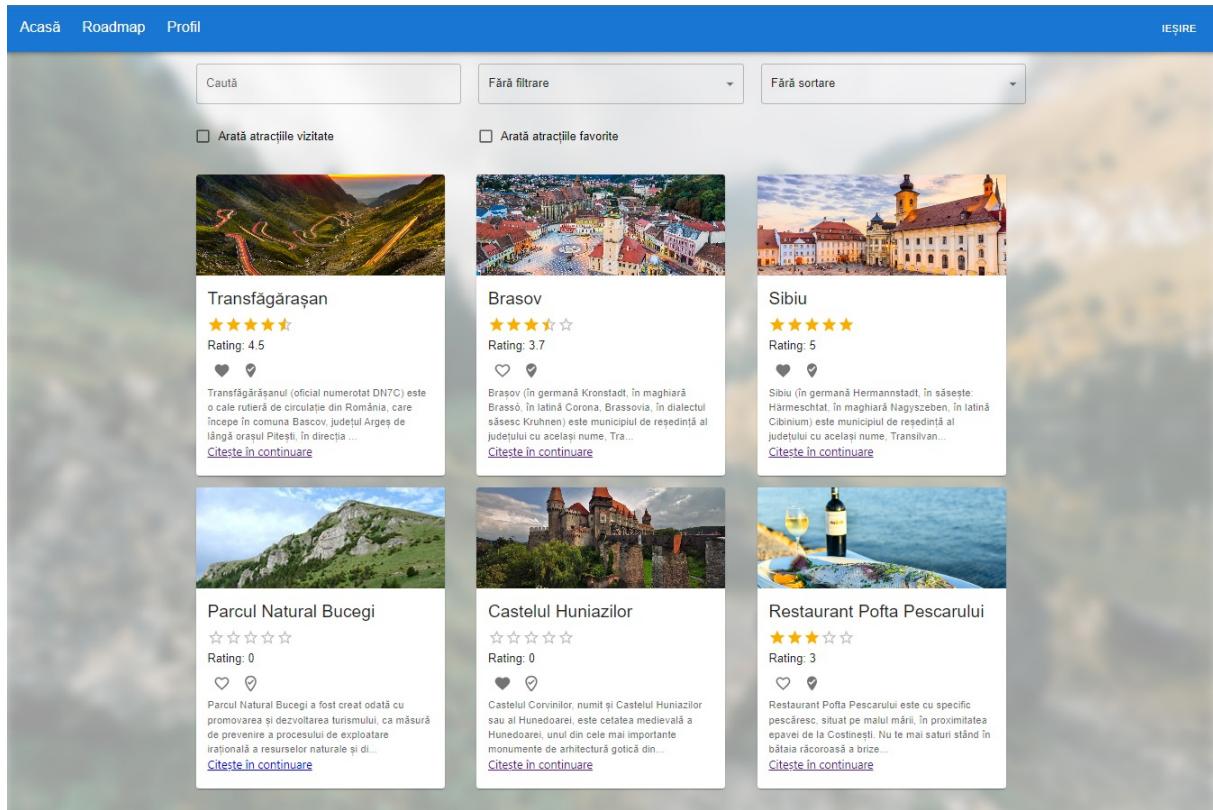


Figura 3.3: Pagina principală

După cum se poate observa în figura de mai sus, interfața dispune de mai multe componente dezvoltate în React care împreună alcătuiesc pagina principală. Printre elementele de UI folosite se numără și componentele deja dezvoltate de Material UI [38].

Utilizatorul poate interacționa cu card-urile punctelor turistice prin funcțiile din Redux store descrise anterior. Modificarea icon-urilor conduce spre actualizarea state-ului global al user-ului și a trimiterii de cereri POST spre server pentru a nu exista o discrepanță între ceea ce se află în baza de date și ceea ce este afișat în UI.

Pentru a personaliza experiența fiecărui utilizator, el poate filtra punctele turistice în funcție de un string introdus în bara de căutare pe care îl caută în nume, după numărul de steluțe acordate, dacă fac parte din array-ul de favorite sau vizitate sau le poate sorta în ordine crescătoare sau descrescătoare al rating-ului afișat.

În cazul în care o locație trezește interes, user-ul poate să o deschidă într-o pagină nouă în care vor fi dezvăluite toate detaliile despre aceasta.

2. Pagina detaliată a unui punct turistic

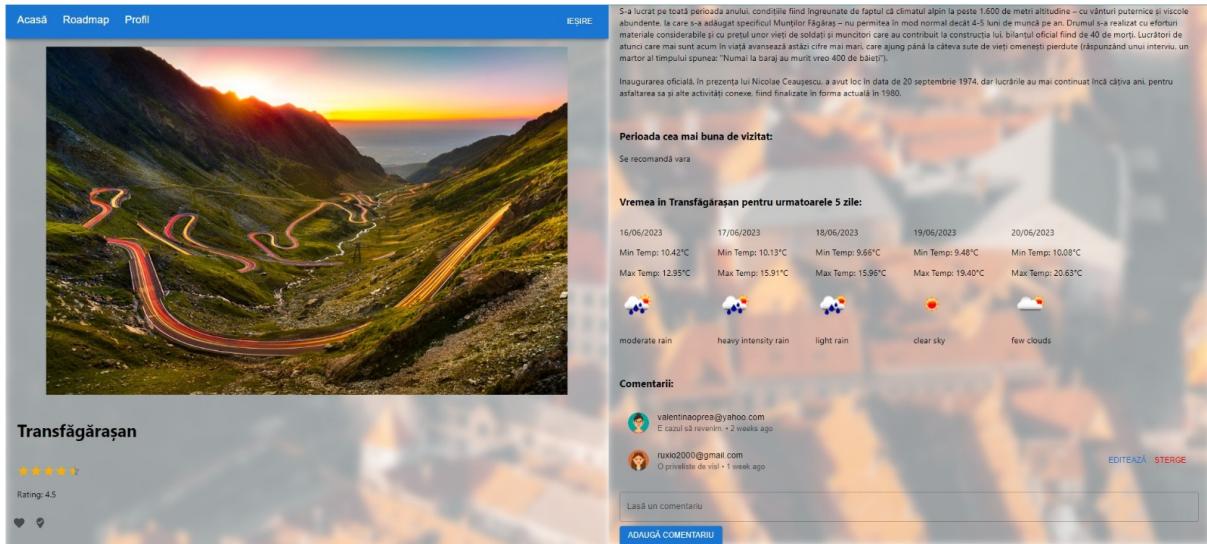


Figura 3.4: Pagina care detaliază fiecare punct turistic

În figura de mai sus se poate observa pagina generată pentru fiecare locație turistică în parte care cuprinde toate detaliile atribuite. Informațiile sunt generate dinamic, folosindu-se metoda *map* prin toate locațiile găsite în baza de date. Valoarea medie a rating-ului este calculată pe loc de fiecare dată când este lăsat un nou input de către utilizator. El își poate modifica rating-ul glisând pe zona stelușelor. De asemenea, se regăsesc iconurile pentru acțiunea de *toggle* pentru locațiile favorite sau vizitate.

Avansând pe pagină, sunt afișate informații despre locul ales și despre perioada cea mai potrivită de a fi vizitat. Mai departe, se regăsește prognoza meteo pe următoarele 5 zile. În această secțiune s-a utilizat un apel către API-ul extern *openWeather* [39]. Pentru a beneficia de funcționalitatea oferită de acest API a fost generată o cheie de către furnizor care va trebui integrată în apelarea lui, precum și valorile latitudinii, respectiv longitudinii, extrase din baza de date pentru a obține prognoza în locația potrivită.

Ultima componentă de pe pagină o reprezintă secțiunea de comentarii, în care utilizatorii își pot lăsa opinii și impresii și se poate porni o discuție despre atracția curentă. Userul are dreptul de a modifica sau șterge doar propriile comentarii. Aceste acțiuni trimit către server un PUT request sau un DELETE request. Dacă nu există comentarii se va afișa textul „Încă nu sunt comentarii”.

3. Pagina hărții

Selectează	Nume locație	Latitudine	Longitude
<input type="checkbox"/>	Transfagarasan	45.53997308187838	24.353033560302766
<input type="checkbox"/>	Brasov	45.65202220031746	25.597852632711778
<input checked="" type="checkbox"/>	Sibiu	45.797208604917074	24.1433257346314
<input checked="" type="checkbox"/>	Parcul Natural Bucegi	45.38848986480097	25.453325277836107
<input type="checkbox"/>	Castelul Huniazilor	45.74948876491034	22.88904991045504
<input type="checkbox"/>	Restaurant Pofta Pescarului	43.96472229204311	28.650574245587123

[GENEREAZĂ RUTA](#)

Figura 3.5: Pagina hărții

Pagina prezintă integrează harta oferită de Google Maps. Pentru a beneficia de hartă, trebuie realizat setup-ul din Google Console în urma căruia va fi generată o cheie API pe baza căreia harta va fi funcțională în componenta implementată. Paginile sau componente care folosesc harta trebuie împrejmuite de structura `<LoadScript googleMapsApiKey=googleMapsApiKey>` disponibilă din pachetul `@react-google-maps/api` care poate fi instalat din npm.

În Google Console există o întreagă colecție de API-uri create special pentru diferite funcționalități din Google Maps. În aplicația prezentă s-a folosit *Maps JavaScript API* care afișează harta, *Directions API* pentru generarea unei rute între minim două locații de pe

hartă și *Geolocation API* pentru locația curentă a utilizatorului. Aceasta va fi disponibilă doar dacă utilizatorul își va da acordul de a-i fi folosită locația curentă.

După cum se poate observa, pinurile hărții sunt colorate diferit.

- **roșu** - pentru locațiile adăugate la favorite
- **galben** - pentru locațiile adăugate la vizitate
- **albastru** - pentru locațiile fără nicio interacțiune
- **negru** - pentru locația curentă a utilizatorului

Când userul face click pe unul din pin-uri, o fereastră de tip modal va fi afișată în care sunt prezente numele, icon-urile în formă de inimă și pin, imaginea, perioada propice de vizitat și link către pagina cu mai multe detalii. Pentru a închide acest modal, user-ul trebuie să facă click oriunde pe ecran.

În cazul în care utilizatorul este de acord cu folosirea locației sale curente, generarea rutei se va face cu originea în aceasta și se vor adăuga locațiile de pe hartă în ordinea selectării lor. Cel mai eficient traseu va apărea pe hartă împreună cu o fereastră de informații în care vor fi prezente distanța și durata drumului. De asemenea, pașii rutei ce trebuie urmăriți vor fi afișați sub hartă într-o secțiune ce se poate minimiza sau maximiza la nevoie. Instrucțiunile sunt afișate în limba engleză. Pentru traducerea acestora este nevoie de integrarea API-ului de Google Translate.

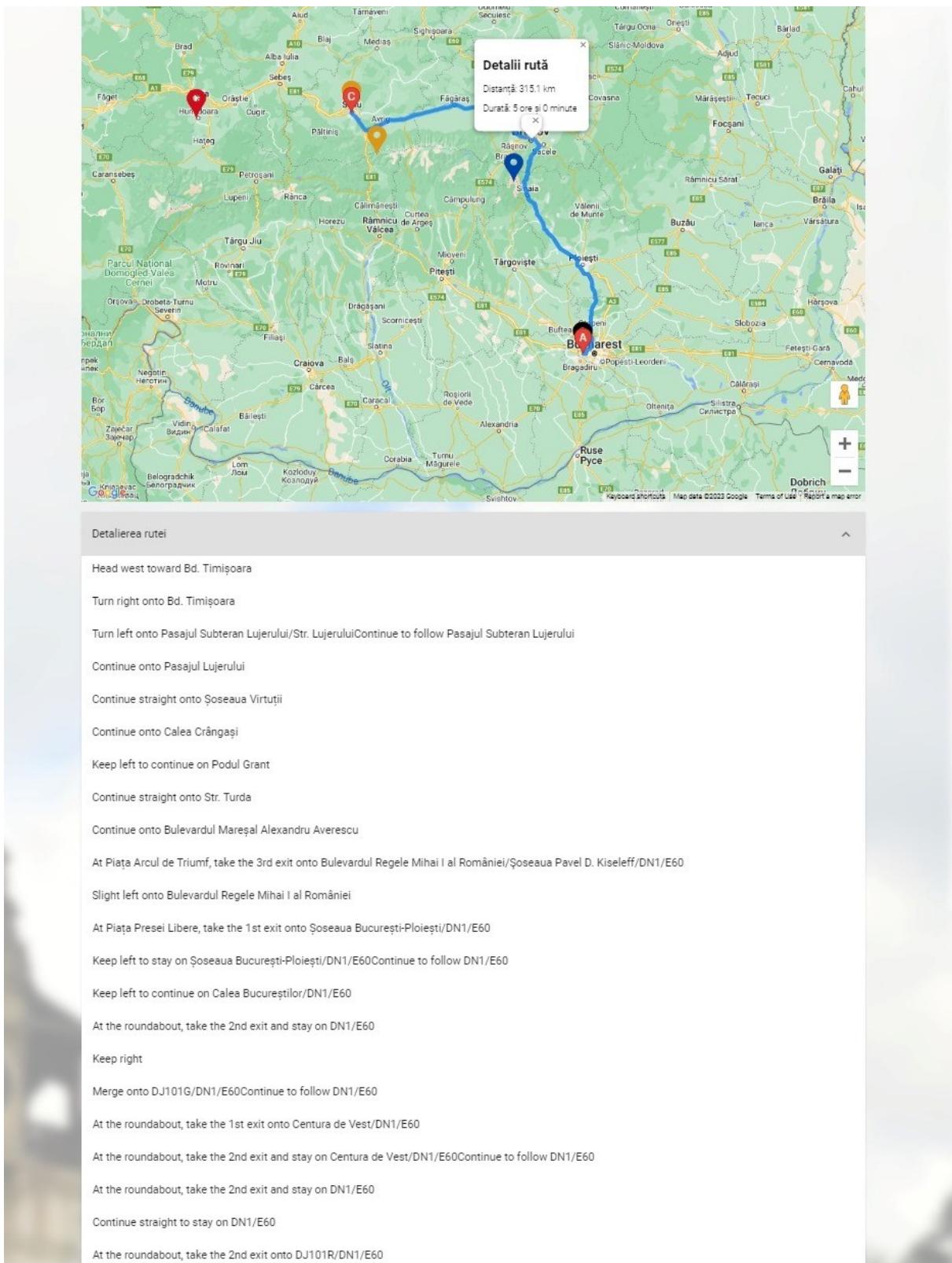


Figura 3.6: Ruta generată între locația curentă a utilizatorului și locațiile selectate

4. Pagina de profil

Pe pagina profilului sunt afișate informațiile generale ale unui utilizator: poza de profil,

nume, adresă de email și data de când acesta este membru al blogului. Inițial toate câmpurile sunt disabled, dar dacă acesta alege să își editeze profilul, doar numele și poza de profil devin active pentru editare. Pentru poza de profil, utilizatorul poate alege dintr-o selecție de 9 imagini prestabilite [40].



Figura 3.7: Formularele paginii de profil în funcție de acțiunea dorită

3.5 Interfața din perspectiva adminului

Aplicația va reține în state-ul controlat de Redux rolul utilizatorului. Dacă acesta este egal cu „admin” noi funcționalități devin active. Față de un utilizator normal, pe lângă componentele deja descrise anterior, adminul va avea acces la o serie de încă trei pagini prin intermediul cărora poate interveni în interacțiunea utilizatorilor cu blogul. De asemenea, acesta este responsabil de mențenanța locurilor ce vor apărea pe pagina principală adăugând noi destinații sau editându-le pe cele vechi.

- **Pagina pentru adăugarea unei destinații noi**

Figura 3.8: Formularul de adăugare a unei locații noi

După cum poate fi observat în figura 3.7, adminul trebuie să completeze o serie de câmpuri cu informațiile necesare noii destinații de adăugat. La apăsarea butonului „Adaugă locație” o cerere de tip POST este trimisă către server cu datele ce trebuie salvate în baza de date. În endpoint este verificat, de asemenea, dacă utilizatorul este autorizat să realizeze această acțiune. Dacă nu a fost întâmpinată nicio eroare, o notificare va apărea în interfață comunicând că locația a fost adăugată cu succes. În caz contrar, o eroare aferentă va fi afișată.

- **Pagina locațiilor turistice**

În această pagină sunt afișate sub formă de tabel toate locațiile găsite în baza de date. Adminul are posibilitatea de a viziona separat pagina cu detalii, a edita un loc sau a-l șterge de tot din baza de date. Pentru fiecare din aceste acțiuni, la trimiterea request-ului către server se verifică dacă persoana care dorește să producă aceste modificări este autorizată.

Acasă	Adaugă	Atractii	Utilizatori	Profil	IEȘIRE	
Imagine	Nume ↑	Descriere	Rating	Acțiuni		
	Brasov	Brașov (în germană Kronstadt, în maghiară Brassó, în latină Corona, Brassovia, în dialectul săsesc K...)	3.7	DETALII	EDITEAZĂ	ȘTERGE
	Castelul Huniazilor	Castelul Corvinilor, numit și Castelul Huniazilor sau al Hunedoarei, este cetatea medievală a Hunedo...	0	DETALII	EDITEAZĂ	ȘTERGE
	Parcul Natural Bucegi	Parcul Natural Bucegi a fost creat odată cu promovarea și dezvoltarea turismului, ca măsură de preve...	0	DETALII	EDITEAZĂ	ȘTERGE
	Restaurant Pofta Pescarului	Restaurant Pofta Pescarului este cu specific pescăresc, situat pe malul mării, în proximitatea epave...	3	DETALII	EDITEAZĂ	ȘTERGE
	Sibiu	Sibiu (în germană Hermannstadt, în săsește: Härmschтat, în maghiară Nagyszeben, în latină Cibinium)...	5	DETALII	EDITEAZĂ	ȘTERGE
	Transfăgărășan	Transfăgărășanul (oficial numerotat DN7C) este o cale rutieră de circulație din România, care începe...	4.5	DETALII	EDITEAZĂ	ȘTERGE

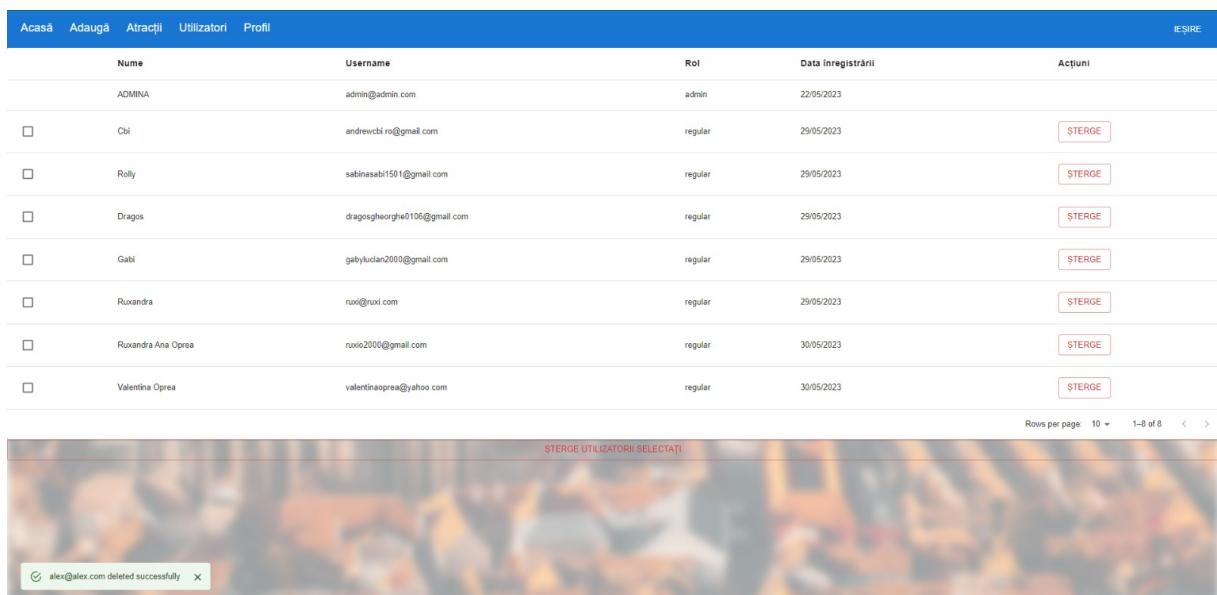
Figura 3.9: Pagina tuturor destinațiilor

Paginarea a fost implementată în această componentă utilizând *TablePagination* din pachetul *@mui/material* instalat din npm.

O altă funcționalitate pe care adminul o primește în pagina detaliată a unei locații este de a șterge orice comentariu dorește în cazul în care sunt încălcate reguli de conduită generală, însă nu poate adăuga unul deoarece rolul lui este de a administra blogul și nu poate fi subiectiv în legătură cu inputul utilizatorilor, decât în cazuri extreme.

- **Pagina utilizatorilor**

În această pagină, adminul poate vizualiza sub formă de tabel paginat toți membrii aplicației alături de informațiile lor generale. Aceasta poate să șteargă utilizatorii unul sau poate selecta mai mulți și elibera pe toți deodată. La apăsarea butonului „Șterge” un modal va apărea în care adminul este rugat să confirme dacă dorește să continue cu această acțiune. În cazul în care răspunsul este afirmativ, un DELETE request este trimis către server, este verificat dacă utilizatorul conectat este autorizat. Dacă da, procesul de stergere este continuat, iar la finalizarea acestuia clientul primește ca răspuns statusul 200 și afișează pe ecran o notificare în care este anunțat că utilizatorul a fost șters cu succes ca în figura 3.10. În cazul întâmpinării unei erori, se va afișa o eroare cu un mesaj aferent în UI.



The screenshot shows a table of users with columns: Nume, Username, Rol, Data înregistrării, and Acțiuni. The 'Acțiuni' column contains a 'STERGE' button for each user. A modal window titled 'ȘTERGE UTILIZATORII SELECȚAȚI' is open at the bottom, showing a confirmation message: 'Sunt sigur că doriți să ștergeți utilizatorii selectați?'. A green success message 'alex@alex.com deleted successfully' is visible at the bottom left of the page.

Utilizatori					IEȘIRE
	Nume	Username	Rol	Data înregistrării	Acțiuni
	ADMINA	admin@admin.com	admin	22/05/2023	
<input type="checkbox"/>	Cobi	andrewcobi.ro@gmail.com	regular	29/05/2023	<button>STERGE</button>
<input type="checkbox"/>	Rolly	sabinasebit501@gmail.com	regular	29/05/2023	<button>STERGE</button>
<input type="checkbox"/>	Dragos	dragosgeorghe0106@gmail.com	regular	29/05/2023	<button>STERGE</button>
<input type="checkbox"/>	Gabi	gabyluclan2000@gmail.com	regular	29/05/2023	<button>STERGE</button>
<input type="checkbox"/>	Ruxandra	ruxi@ruxi.com	regular	29/05/2023	<button>STERGE</button>
<input type="checkbox"/>	Ruxandra Ana Oprea	ruxi2000@gmail.com	regular	30/05/2023	<button>STERGE</button>
<input type="checkbox"/>	Valentina Oprea	valentinaoprea@yahoo.com	regular	30/05/2023	<button>STERGE</button>

Rows per page: 10 ▾ 1–8 of 8 < >

Figura 3.10: Pagina tuturor utilizatorilor

3.6 Dockerfiles

Toată logica aplicației este stocată într-un singur GitHub repository. Aceasta este împărțită în două foldere principale:

- **client** - conține toată implementarea și modulele folosite pentru partea de frontend
- **server** - conține toată implementarea și modulele folosite pentru partea de backend

Așa cum a fost specificat în capitolul anterior, la fiecare push pe Git, este declanșat fluxul GitHub Actions correspondent folderului în care au fost adăugate modificări pentru a construi o imagine containerizată a ultimei versiuni a platformei. Dar ca acest workflow să reușească să construiască imaginea, are nevoie de un Dockerfile pentru a-și lua instrucțiunile și comenzi necesare realizării acestei operații.

În acest mod, au fost create două Dockerfiles, unul pentru imaginea de frontend și unul pentru cea de backend. În continuare cele două fișiere vor fi analizate.

- **Imaginea de backend**

```
FROM node:14
WORKDIR / app
COPY package *.json ./
RUN npm install
COPY . .
ENV FRONTEND_API=http://frontend-service.licenta.eec8dd5da
d2b40b39b69.eastus.aksapp.io
EXPOSE 8080
CMD [ "node", "server.js" ]
```

Se pornește de la imaginea de bază node:14. Se setează ca folder de lucru /app și se copiază conținutul găsit în package.json și package-lock.json, se instalează dependențele aplicației și se împachetează sursa aplicației în interiorul containerului. Se setează adresa la care aplicația va fi disponibilă ca *environment variable* deoarece va fi nevoie de valoarea ei în workflow. Se exportă portul 8080 pentru a facilita comunicarea dintre container cu exteriorul și se definește comanda care va porni aplicația.

- **Imaginea de frontend**

```
FROM node:14 as build
WORKDIR / app
ARG REACT_APP_API_URL
ARG REACT_APP_GOOGLE_MAPS_API_KEY
ENV REACT_APP_API_URL=$REACT_APP_API_URL
ENV REACT_APP_GOOGLE_MAPS_API_KEY=$REACT_APP_GOOGLE_MAPS_API_KEY
COPY package *.json ./
RUN npm install
COPY . .
RUN npm run build

FROM nginx:1.19.0-alpine
COPY --from=build / app / build / usr / share / nginx / html
COPY default.conf / etc / nginx / conf.d / default.conf
EXPOSE 3000
CMD [ "nginx", "-g", "daemon off;" ]
```

Se pornește de la imaginea de bază node:14 și i se atribuie alias-ul build. Se setează ca folder de lucru /app, se adaugă cheia pentru Google Maps Api și dns-ul serviciului de

backend ca argumente de build deoarece React nu poate accesa variabilele env din fișierul .env la momentul build-urii. De abia apoi putem seta variabilele env cu aceleași de la argumente pentru a le folosi. Se copiază conținutul găsit în package.json și package-lock.json și se instalează dependențele aplicației în container. Apoi, se integrează a doua imagine de bază nginx:1.19.0-alpine, se copiază toate fișierele statice rezultate de la primul build în /usr/share/nginx/html. Docker copiază fișierul care conține configurațiile de bază pentru Nginx în /etc/nginx/conf.d/default.conf. În continuare, portul 3000 este expus pentru conectivitatea containerului cu exteriorul, apoi pornește server-ul nginx și îi comunică să ruleze în prim-plan ca să nu se opreasă containerul imediat după ce a pornit.

Capitolul 4

Concluzii

Prin redactarea acestei lucrări mi-am aprofundat cunoștiințele despre serviciile Kuberne-tes oferite de Azure, despre procesul automatizat de deployment al unei aplicații cu FluxCD și GitActions, precum și construirea unei platforme web ce presupune stăpânirea limbajului de programare JavaScript și a librăriilor și framework-urilor sale.

Industria turismului este într-o continuă dezvoltare și adaptarea la noile tehnologii este imperativă. Azure Kubernetes Service împreună cu FluxCD oferă automatizare pentru livrarea și gestionarea versiunilor aplicației, asigurându-se că utilizatorii sunt la zi cu ultimele actualizări și nu vor avea parte de întreruperi.

Platforma creată are potențial de dezvoltare, putând fi adăugate diferite funcționalități, precum crearea unui itinerariu și împărtășirea acestuia cu ceilalți utilizatori sau adăugarea unui rol nou de furnizor, în care administratorii diferitelor restaurante sau atracții turistice își pot gestiona singuri pagina produsului său. Clusterul AKS în continuare va oferi suport, scalându-și resursele și oferind cea mai fluidă experiență utilizatorilor.

Bibliografie

- [1] *What is Azure Kubernetes Service?*, <https://learn.microsoft.com/en-us/azure/aks/intro-kubernetes/>, (accesat în 3.6.2023).
- [2] *Learn Kubernetes Basics*, <https://kubernetes.io/docs/tutorials/kubernetes-basics/>, (accesat în 3.6.2023).
- [3] InCont.ro, *Când tehnologia ajunge la nori. Revoluția Cloud Computing. O scurtă istorie a soluțiilor de stocare a informațiilor (VII)*, 2013, URL: <https://ibani.stirileprotv.ro/best-of-it/cand-tehnologia-ajunge-la-nori-revolutia-cloud-computing-o-scurta-istorie-a-solutiilor-de-stocare-a-informatiilor.html/> (accesat în 3.6.2023).
- [4] *Flux Documentation*, <https://fluxcd.io/flux/>, (accesat în 3.6.2023).
- [5] *Understanding GitHub Actions*, <https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions/>, (accesat în 3.6.2023).
- [6] *Docker Reference documentation*, <https://docs.docker.com/reference/>, (accesat în 3.6.2023).
- [7] *Azure Container Registry Documentation*, <https://learn.microsoft.com/en-us/azure/container-registry/>, (accesat în 3.6.2023).
- [8] *What is an IP Address: Definition, Types Usage*, <https://whatismyipaddress.com/ip-address/>, (accesat în 3.6.2023).
- [9] John Burke Ben Lutkevich, *Domain name system (DNS)*, 2021, URL: <https://www.techtarget.com/searchnetworking/definition/domain-name-system> (accesat în 3.6.2023).
- [10] Dionysia Lemonaki, *What is YAML? The YML File Format*, 2022, URL: <https://www.freecodecamp.org/news/what-is-yaml-the-yml-file-format/> (accesat în 3.6.2023).
- [11] *What is Azure Active Directory?*, <https://learn.microsoft.com/en-us/azure/active-directory/fundamentals/active-directory-whatis/>, (accesat în 3.6.2023).

- [12] Tom Nolle Ben Lutkevich, *Application programming interface (API)*, 2022, URL: <https://www.techtarget.com/searchapparchitecture/definition/application-program-interface-API/> (accesat în 3.6.2023).
- [13] *What is the Azure CLI?*, <https://learn.microsoft.com/en-us/cli/azure/what-is-azure-cli/>, (accesat în 3.6.2023).
- [14] *Introduction to Kustomize*, <https://kubectl.docs.kubernetes.io/guides/introduction/kustomize/>, (accesat în 3.6.2023).
- [15] *Namespaces*, <https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/>, (accesat în 3.6.2023).
- [16] *Leases*, <https://kubernetes.io/docs/concepts/architecture/leases/>, (accesat în 3.6.2023).
- [17] *What is GitOps?*, <https://about.gitlab.com/topics/gitops/>, (accesat în 3.6.2023).
- [18] *NGINX Ingress Controller*, <https://docs.nginx.com/nginx-ingress-controller/intro/overview/>, (accesat în 3.6.2023).
- [19] *What is Azure DevOps?*, <https://learn.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops/>, (accesat în 16.6.2023).
- [20] *Start using Azure Repos*, <https://learn.microsoft.com/en-us/azure/devops/repos/get-started/?view=azure-devops/>, (accesat în 16.6.2023).
- [21] *Use Azure Pipelines*, <https://learn.microsoft.com/en-us/azure/devops/pipelines/get-started/pipelines-get-started?view=azure-devops/>, (accesat în 16.6.2023).
- [22] *Introducing JSON*, [https://www.json.org/json-en.html/](https://www.json.org/json-en.html), (accesat în 3.6.2023).
- [23] *Web frameworks and technologies*, <https://survey.stackoverflow.co/2022/#section-most-popular-technologies-web-frameworks-and-technologies>, (accesat în 3.6.2023).
- [24] *Introduction to Node.js*, <https://nodejs.dev/en/learn/>, (accesat în 3.6.2023).
- [25] *Describing the UI*, <https://react.dev/learn/describing-the-ui/>, (accesat în 3.6.2023).
- [26] *About npm*, <https://docs.npmjs.com/about-npm/>, (accesat în 3.6.2023).
- [27] *Express web framework (Node.js/JavaScript)*, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/, (accesat în 3.6.2023).
- [28] *Mongoose Documentation*, [https://mongoosejs.com/docs/index.html/](https://mongoosejs.com/docs/index.html), (accesat în 3.6.2023).

- [29] *Promise based HTTP client for the browser and node.js*, <https://axios-http.com/docs/intro/>, (accesat în 3.6.2023).
- [30] *Introduction to XML*, [https://www.w3schools.com/xml/xml_whatis.asp/](https://www.w3schools.com/xml/xml_whatis.asp), (accesat în 3.6.2023).
- [31] *React JSX*, [https://www.w3schools.com/react/react_jsx.asp/](https://www.w3schools.com/react/react_jsx.asp), (accesat în 3.6.2023).
- [32] *HTML basics*, [https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics/](https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics), (accesat în 3.6.2023).
- [33] *Introduction to the DOM*, [https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction/](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction), (accesat în 3.6.2023).
- [34] *Getting Started with Redux*, [https://redux.js.org/introduction/getting-started/](https://redux.js.org/introduction/getting-started), (accesat în 3.6.2023).
- [35] *Introduction to MongoDB*, <https://www.mongodb.com/docs/manual/introduction/>, (accesat în 3.6.2023).
- [36] AbhayBhat, *Blowfish Algorithm with Examples*, 2021, URL: [https://auth0.com/blog/hashing-in-action-understanding-bcrypt/](https://auth0.com/blog/hashing-in-action-understanding-bcrypt) (accesat în 3.6.2023).
- [37] *Introduction to JSON Web Tokens*, <https://jwt.io/introduction/>, (accesat în 3.6.2023).
- [38] *Material UI - Overview*, <https://mui.com/material-ui/getting-started/overview/>, (accesat în 16.6.2023).
- [39] *Weather data in a fast and easy-to-use way*, <https://openweathermap.org/guide/>, (accesat în 3.6.2023).
- [40] *Poze de profil*, [https://www.vecteezy.com/vector-art/1261016-business-people-round-avatars/](https://www.vecteezy.com/vector-art/1261016-business-people-round-avatars), (accesat în 3.6.2023).