|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  **FAKULTET TEHNIČKIH NAUKA U NOVOM SADU** |  |

Milorad Radović

**Izgradnja veb aplikacije za oglašavanje i licitiranje usluga popravki u kući**

Diplomski rad

- Osnovne akademske studije -

Novi Sad, 2021.

|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  **FAKULTET TEHNIČKIH NAUKA**  21000 NOVI SAD, Trg Dositeja Obradovića 6 | Datum: |
|  |
| **ZADATAK ZA IZRADU DIPLOMSKOG (BACHELOR) RADA** | List: |
| 1/1 |

*(Podatke unosi predmetni nastavnik - mentor)*

|  |  |
| --- | --- |
| Vrsta studija: | ☐ **Osnovne akademske studije** |
| Studijski program: | **Softversko inženjerstvo i informacione tehnologije** |
| Rukovodilac  studijskog programa: | **prof. dr Miroslav Zarić** |

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | **Milorad Radović** | Broj indeksa: | **SW 67/2017** |
| Oblast: | Web programiranje | | |
| Mentor: | **doc. dr Siniša Nikolić** | | |
| NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA IZDAJE SE ZADATAK ZA DIPLOMSKI RAD, SA SLEDEĆIM ELEMENTIMA:   * problem – tema rada; * način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna; * literatura | | | |

**NASLOV DIPLOMSKOG (BACHELOR) RADA:**

|  |
| --- |
| **Izgradnja veb aplikacije za oglašavanje i licitiranje usluga popravki u kući** |

**TEKST ZADATKA:**

|  |
| --- |
| Opisati sistem za oglašavanje i licitiranje usluga popravki u kući. Analizirati postojeća rešenja i diskutovati njihove dobre i loše osobine. Predložiti arhitekturu i tehnologije za izradu novog rešenja. Opisati implementaciju predloženog rešenja. Dokumentovati predloženo rešenje |

|  |  |
| --- | --- |
| Rukovodilac studijskog programa: | Mentor rada: |
|  |  |

|  |
| --- |
| Primerak za:  - Studenta;  - Mentora |

**SADRŽAJ**

[1. UVOD 1](#_Toc82628110)

[2. STANJE U OBLASTI 3](#_Toc82628111)

[2.1. NadjiMajstora 3](#_Toc82628112)

[2.2. MyWorkman 4](#_Toc82628113)

[2.3. NeedWorkDone 5](#_Toc82628114)

[2.4. CheckATrade 5](#_Toc82628115)

[2.5. RatedPeople 5](#_Toc82628116)

[2.6. Diskusija postojećih rešenja 5](#_Toc82628117)

[3. KORIŠĆENE TEHNIKE I TEHNOLOGIJE 6](#_Toc82628118)

[3.1. ASP .NET Core 6](#_Toc82628119)

[3.2. Angular 10](#_Toc82628120)

[3.3. PostgreSQL 17](#_Toc82628121)

[4. SPECIFIKACIJA SISTEMA 19](#_Toc82628122)

[4.1. Dijagram klasa 19](#_Toc82628123)

[4.2. Dijagram slučajeva korišćenja 20](#_Toc82628124)

[4.3. Arhitektura sistema 22](#_Toc82628125)

[5. IMPLEMENTACIJA SISTEMA 27](#_Toc82628126)

[5.1. Klijentska strana 27](#_Toc82628127)

[5.2. Serverska strana 28](#_Toc82628128)

[6. PRIKAZ IMPLEMENTIRANOG SISTEMA 34](#_Toc82628129)

[7. ZAKLJUČAK 44](#_Toc82628130)

[LITERATURA 46](#_Toc82628131)

[BIOGRAFIJA 48](#_Toc82628132)

[KLJUČNA DOKUMENTACIJSKA INFORMACIJA 49](#_Toc82628133)

[KEY WORDS DOCUMENTATION 50](#_Toc82628134)

# UVOD

*Amazon Web Services* (AWS) je platforma veb servisa koja pruža rješenja za skladištenje, umrežavanje i računarstvo na *cloud*-u (engl. *cloud computing*). Predstavlja jedan od dobavljača servisa na *cloud*-u (engl. *cloud service provdiers*) [[1]](#_LITERATURA_1). U tehničkom smislu, *cloud* je skup velikog broja povezanih hardverskih resursa koje korisnik vidi kao jedan veliki virtuelni računar. Korisnik može da alocira dijelove *cloud*-a za svoje aplikacije. Veb servisi koje nudi AWS su dostupni putem interneta upotrebom tipičnih veb protokola, kao što je HTTP (*Hypertext Transfer Protocol)* [[2]](#_LITERATURA_1). Njihovim kombinovanjem mogu da se izgrade arhitekture pogodne za hostovanje veb sajtova, pokretanje poslovnih aplikacija i skaldištenje ogromnih količina podataka [[1,3]](#_LITERATURA_1).

Prednost koju nudi *cloud* jeste mogućnost izgradnje efikasne, pouzdane, skalabilne, elastične i visoko dostupne arhitekture za pojedinačne korisničke aplikacije. *AWS Well-Architected Framework* opisuje ključne koncepte i principe za dizajniranje upravo takve arhitekture na AWS. Oslanja se na pet „stubova“:

* ***Operational Exellence*** – unapređivanje i automatizacija sistema na osnovu praćenja njegovog rada;
* ***Security***– bezbjednost informacija u sistemu;
* ***Reliability***– pouzdanost sistema; sistem konzistentno izvršava sve što se od njega očekuje i brzo se oporavlja od otkaza;
* ***Performance Efficiency*** *–* efikasnost sistema; izbor odgovarajućih resursa u zavisnosti od zahtjeva;
* ***Cost Optimization*** – izbjegavanje nepotrebnih troškova [[4]](#_LITERATURA_1).

Ovaj rad je fokusiran na pouzdanost i efikasnost sistema. Da bi se to postiglo, potrebno je voditi računa o konceptima *AWS Well-Architected Framework*-a kao što su elastičnost (engl. *elasticity*) i visoka dostupnost (engl. *high* *availability*).

Prema tome, tema ovog rada je razvoj elastične i visoko dostupne aplikacije koriščenjem AWS servisa. Elastičnost podrazimjeva sposobnost sistema da automatski dobavlja resurse kada su mu potrebni i pušta ih kada mu više nisu potrebni. Visoka dostupnost podrazumijeva sposobnost sistema da izvrši traženu funkcionalnost u datom trenutku [[5]](#_LITERATURA_1).

Cilj rada je prikaz osnovnih AWS servisa i prijedloga rješenja za izgradnju elastične i visoko dostupne arhitekture.

U nastavku rada ukratko će biti opisane druge *cloud* platforme i njihovo poređenje sa AWS, zatim će biti dat opis osnovnih AWS servisa sa naglaskom na one koji su korišćeni u radu, te će biti prikazan primjer elastične i visoko dostupne arhitekture na koju je postavljena jednostavna aplikacija. Budući da se mnogi servisi naplaćuju, prikazana arhitektura uglavnom će biti ograničena na besplatne servise, a na kraju rada će biti dati prijedlozi za njeno unapređenje korišćenjem pogodnijih servisa koji ne spadaju u *Free Tier* [[6]](#_LITERATURA_1).

# STANJE U OBLASTI

Računarstvo na *cloud*-u predstavlja isporuku računarskih resursa i skladišnih kapaciteta kao uslugu korisniku. Kompanije koje nude takve usluge nazivaju se dobavljači servisa na *cloud*-u. Pored AWS-a, neki od njih su: *Google* *Cloud* *Platform* (GCP), IBM *Cloud*, *Oracle* *Cloud*, *Alibaba* *Cloud*, *Microsoft* *Azure*.

Servisi koje nude ove kompanije mogu se podijeliti na:

* **Softver kao usluga, SaaS** (engl. *Software* *as* *Service*) – pruža mogućnost upotrebe dostupnih aplikacija koje su pokrenute na infrastrukturi *cloud*-a. Primjeri su: *Amazon WorkSpaces, Google Apps for Work, Microsoft Office 365*;
* **Platforma kao usluga, PaaS** (engl. *Platform* *as* *Service*) - korisnik može razvijati, testirati i distribuirati svoje aplikacije koje se pokreću na infrastrukturi *cloud*-a. Primjeri su: AWS *Elastic Beanstalk, Google App Engine, Heroku*;
* **Infrastruktura kao usluga, IaaS** (engl. *Infrastructure* *as* *Service*) – korisnik može upravljati obradom, skladištenjem, umrežavanjem i drugim osnovnim računarsnkim resursima. Primjeri su: *Amazon* EC2*, Google Compute Engine, Microsoft Azure* [[1]](#_LITERATURA_1)*.*

Kako *cloud* predstavlja skup resursa raspoređenih na razlčite geografske regione na cijeloj planeti, ono što je zajedničko za sve dobavljače jeste mogućnost razvoja aplikacije u različitim regionima, koji se mogu dijeliti na zone dostupnosti (engl. *availability* *zones*). Zone dostupnosti predstavljaju logičke centre podataka (engl. *data* *centers*), koji mogu da se sastoje od jednog ili više fizičkih centara podataka. Jedna aplikacija može da se postavi na resurse u okviru više zona dostupnosti. U tom slučaju, ako se u jednoj zoni desi otkaz, aplikacija će nastaviti da radi koristeći resurse iz druge zone. [[7]](#_LITERATURA_1)*.*

U nastavku su ukratko opisane najpopularnije alternative AWS-a i navedeni su njihovi najznačajniji servisi.

## NadjiMajstora

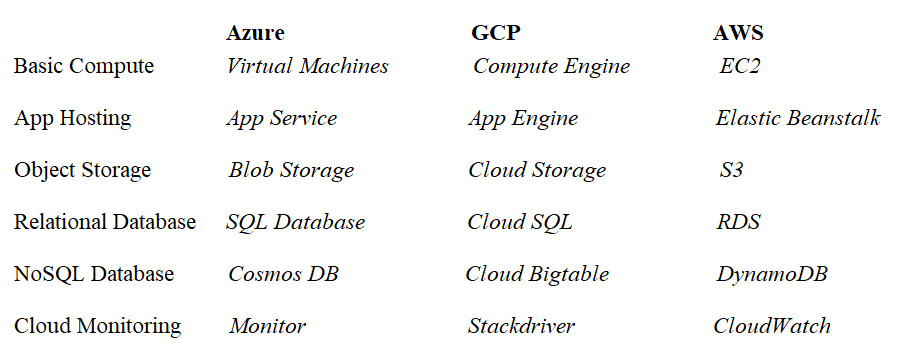
*Microsoft Azure* je platforma razvijena 2010. godine, čija glavna prednost je integrisanost sa *Windows* i *Microsoft* softverom, što znači da se preduzeća koja koriste ove tehnologije često odlučuju za *Azure*.

Kao i svaki *cloud* dobavljač, *Azure* nudi mogućnost razvoja skalabilnih i visoko dostupnih aplikacija upotrebom određenih servisa. Neki od najznačajnijih servisa su *Azure Virtual Machine* i *Virtual Machine Scale Sets*, koji služe za postavljanje aplikacije i postizanje skalabilnosti. Od servisa zaduženih za skladištenje resursa i podataka najznačajniji su *Blob* *Storage* i SQL *Database* [[8]](#_LITERATURA_1)*.*

## MyWorkman

*Google Cloud Platform* (GPC)predstavlja platformu javno dostupnu od 2011. godine, koja je razvijena na istoj infrastrukturi koju *Google* koristi za svoje proizvode kao što su *Google* *Search*, *Gmail*, *Youtube*. Nije toliko popularna kod preduzeća kao *Azure* ili AWS, ali zahvaljujući brzom razvoju u posljednje vrijeme i napretku u oblasti mašinskog učenja, uspijeva da ostane konkurentna na tržištu. Broj servisa je nešto manji nego kod prethodno navedenih dobavljača. Najznačajniji su *Compute* *Engine*, *Cloud* *Storage*, SQL[[9]](#_LITERATURA_1)*.*

Na slici 2.1 prikazani su ekvivalentni AWS servisi za pomenute *Azure* i GCP servise, a oni će biti detaljno objašnjeni u narednom poglavlju.

**

1. Ekvivalentni servisi koje nude *Azure*, GCP i AWS

Na osnovu istraživanja kompanije *Gartner* iz 2021. godine poznato je da danas na tržištu dominira AWS, u velikoj mjeri zahvaljujući najvećem broju servisa i najobimnijoj mreži centara podataka. Na drugom mjestu je *Azure*, zatim *Google* i *Alibaba*. Na slici 2.2 prikazan je raspored kompanija na grafiku koji je nazvan „Magični kvadrant“ (engl. *Magic Quadrant for Cloud Infrastructure and Platform Services*), gdje može da se vidi trenutna pozicija kompanija na tržištu [[10]](#_LITERATURA_1)*.*



1. Magični kvadrant [[1]](#_LITERATURA_1)

## NeedWorkDone

Todo

## CheckATrade

Todo

## RatedPeople

Todo

## Diskusija postojećih rešenja

todo

# KORIŠĆENE TEHNIKE I TEHNOLOGIJE

U ovom poglavlju su opisani najznačajniji servisi koje nudi AWS [[11]](#_LITERATURA_1), s naglaskom na servise koji omogućavaju postizanje elastičnosti i visoke dostupnosti i koji su korišćeni u radu, kao i tehnologije korišćene za izradu same aplikacije, *Spring* [[12]](#_LITERATURA_1)i *Angular* [[13]](#_LITERATURA_1)*.*

## ASP .NET Core

AWS je platforma za računarstvo na *cloud*-u pokrenuta 2006. godine od strane kompanije *Amazon*. Danas, kao najuspješnija platforma u ovoj oblasti, nudi preko 200 servisa različitih namjena sa mogućnošću plaćanja po sistemu „*pay-as-you-go*“, što znači da korisnik plaća usluge samo onoliko koliko ih koristi. Svaki od servisa ima svoju cijenu koja može biti definisana na osnovu broja minuta ili sati upotrebe, na osnovu saobraćaja (mjeri se gigabajtima ili u brojem zahtjeva) ili na osnovu upotrebe skladišta (mjeri se kapacitetom ili dužinom upotrebe). AWS takođe nudi i *Free Tier* režim, prema kojem u toku prvih godinu dana korišćenja platforme korisnik može da upotrebljava neke osnovne servise besplatno. [[1]](#_LITERATURA_1).

Slično kao i kod ostalih platformi, globalnu arhitekturu AWS-a danas čini 81 zona dostupnosti raspoređena na 25 geografskih regiona, a u planu je još 21 nova zona i 7 novih regiona. Neki od regiona su: *US East (Ohio), Africa (Cape Town), South America (São Paulo), Asia Pacific (Singapore), Europe (Frankfurt), Middle East (Bahrain)* itd, što je prikazano na slici 3.1 [[6]](#_LITERATURA_1). Gdje god da se nalazi, korisnik može da izabere bilo koji region i da rasporedi svoje aplikacije na više zona dostupnosti, što obezbjeđuje visoku dostupnost. Takođe, odabirom zone korisnik može da približi aplikaciju korisnicima određenog geografskog područja da bi odziv aplikacije bio brži.



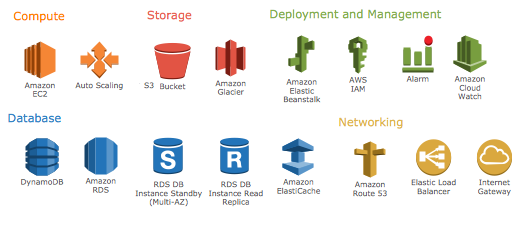
1. Postojeći i planirani geografski regioni u AWS [[6]](#_LITERATURA_1)

Da bi korisnik mogao da radi sa AWS servisima potrebno je da kreira nalog, nakon čega će mu biti dostupan korisnički interfejs, odnosno AWS *Management Console* [[14]](#_LITERATURA_1). Ovaj korisnički interfejs daje mogućnosti za podešavanje raznih opcija i parametara, kako bi korisnik mogao da prilagodi upotrebu servisa svojim zahtjevima.

U nastavku su navedeni neki od najčešće korišćenih servisa:

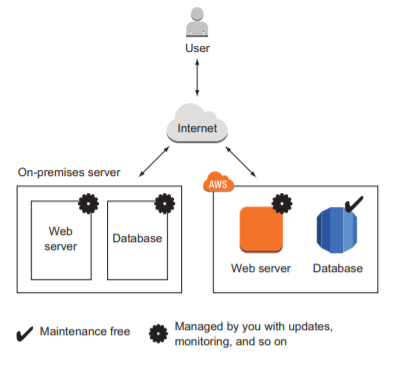
* ***Compute***
* *Amazon Elastic Compute Cloud* (EC2) – virtuelna mašina na *cloud*-u sa izabranim operativnim sistemom i drugim parametrima; najčešće se koristi kao server veb aplikacije [[11,15]](#_LITERATURA_1);
* *Amazon Lambda* – servis za izvršavanje koda bez servera; kod se izvršava kada se desi određeni događaj [[11]](#_LITERATURA_1);
* *Amazon Virtual Private Cloud* (VPC) – logički izolovan dio *cloud-*a koji korisnik alocira za svoje potrebe [[11,16]](#_LITERATURA_1);
* *Amazon Elastic Beanstalk* – servis za automatsko postavljanje aplikacije na skalabilnu arhitekturu [[11,17]](#_LITERATURA_1);
* *Amazon* EC2 *Auto Scaling* – servis za definisanje politike skaliranja EC2 instanci [[11,18]](#_LITERATURA_1);
* i dr.
* ***Storage***
* *Amazon Simple Storage Service* (S3)– servis za skladištenje objekata i fajlova [[11,19]](#_LITERATURA_1);
* *Amazon Glacier* – servis za arhiviranje i održavanje rezervne kopije fajlova kojima se ne pristupa često [[11,20]](#_LITERATURA_1);
* i dr.
* ***Database***
* *Amazon Relational Database* (RDS) – relaciona baza podataka sa mogućnošću skaliranja [[11,21]](#_LITERATURA_1);
* *Amazon DynamoDB* – NoSQL baza podataka za skladištenje dokumenata [[11]](#_LITERATURA_1);
* *Amazon Aurora* – relaciona baza podataka namjenjena za *cloud* i kompatibilna sa PostgreSQL i MySQL [[11,21]](#_LITERATURA_1);
* *Amazon ElastiCache* – *in-memory* servis za keširanje [[11]](#_LITERATURA_1);
* i dr.
* ***Networking & Content Delivery***
* *Amazon CloudFront* – servis za dostavljanje sadržaja preko mreže (engl. *Content Delivery Ntework*, CDN) [[11]](#_LITERATURA_1);
* *Amazon Route 53* – servis za rutiranje, odnosno *Domain Name System* (DNS) [[11,22]](#_LITERATURA_1) na *cloud*-u [[11]](#_LITERATURA_1);
* *Elastic Load Balancing* (ELB) – servis za raspoređivanje saobraćaja na više EC2 instanci [[11,23]](#_LITERATURA_1);
* i dr.
* ***Management Tools***
* *Amazon CloudWatch* – servis za praćenje rada EC2 instanci i prikupljanje podataka [[11,24]](#_LITERATURA_1);
* i dr.
* ***Security, Identity & Compliance***
* *Amazon Identity and Access Management* (IAM) – servis za definisanje pravila kontrole pristupa resursima [[11,25]](#_LITERATURA_1);
* i dr.
* ***Messaging***
* *Amazon Simple Notification Service* (SNS) – servis za slanje obavještenja korisniku [[11,26]](#_LITERATURA_1);
* i dr.

Na slici 3.2 prikazane su ikone koje se koriste za predstavljanje nekih od navedenih servisa. Pored njih, AWS nudi i servise najmenjene za rad sa doker kontejnerima (ECR, ECS, *Fargate*, EKS), migraciju resursa na AWS (*Server Migration Service, Database Migration Service*), automatizaciju postavljanja koda na EC2 instance (*CodeDeploy, CodeCommit*), mašinsko učenje, robotiku*, Internet of Things* i dr [[11]](#_LITERATURA_1).



1. Ikone za predstavljanje servisa

Na slici 3.3 dat je primjer najjednostavnije arhitekture postavljene na lokalni server i te iste arhitekture postavljene na odgovarajuće AWS servise.



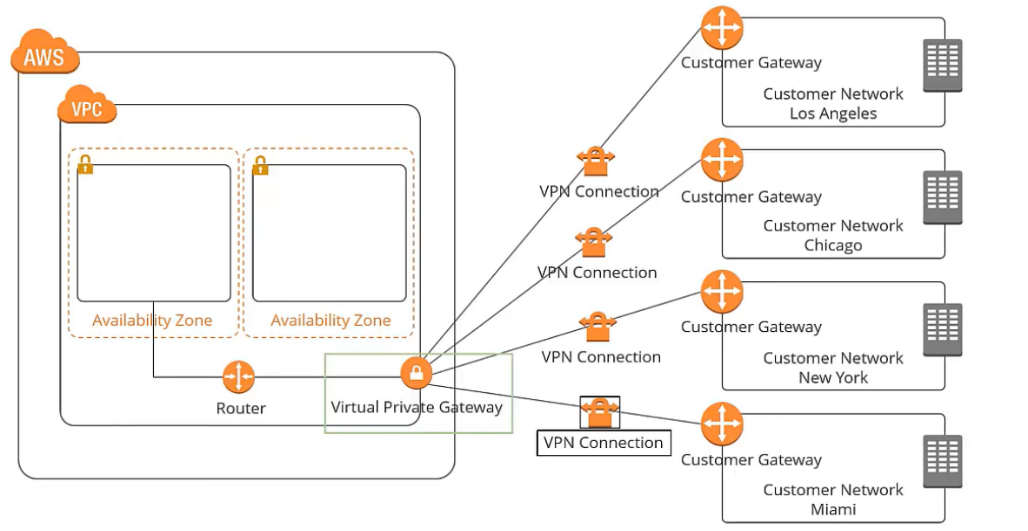
1. Primjer najjednostavnije arhitekture [[1]](#_LITERATURA_1)

## Angular

*Amazon* definiše **VPC** (engl. *Virtual Private Cloud*) [[16]](#_LITERATURA_1) kao logički izolovan dio *cloud-*a koji omogućava korisniku da pokreće AWS resurse u virtuelnoj mreži koju je definisao. Korisnik sam kontroliše svoje mrežno okruženje, uključujući izbor opsega IP adresa (*Internet Protocol Address*), kreiranje podmreža, konfigurisanje tabela rutiranja i pristupa internetu. Kreiranjem privatnih i javnih podmreža pomoću koncepata kao što su *security* grupe i liste kontrole pristupa na mreži, obezbjeđuje se više slojeva zaštite pristupa instancama servisa na *cloud*-u.

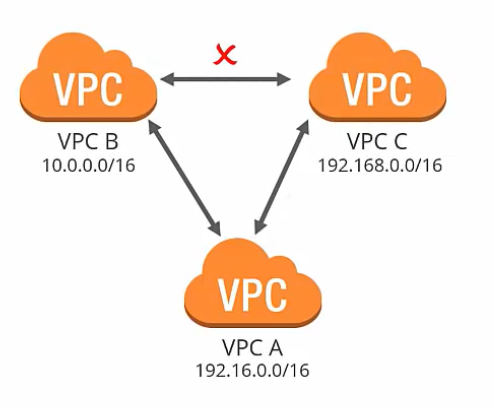
AWS koristi **CIDR** (*Classles Inter-Domain Routing*) metodu za alociranje IP adresa i rutiranje. Prema ovoj notaciji identifikator IP adrese se sastoji od same adrese i njoj pridružene mrežne maske. Maska predstavlja 32-bitni binarni broj koji govori koje bitove IP adrese treba posmatrati kao bitove adrese mreže. U verziji IPv4, sama adresa se sastoji od četiri broja u opsegu od 0 do 255. Primjer adrese je 192.168.100.0/22. CIDR blok je niz IP adresa grupisanih u jedan zapis u tabeli rutiranja. Npr. IPv4 blok 192.168.100.0/22 predstavlja 1024 adrese od 192.168.100.0 do 192.168.103.255. Svaki *Amazon* nalog nudi podrazumijevani VPC (engl. *default VPC*), čiji CIDR blok je *subnet* maska 16, što znači da obuhvata 65536 privatnih IP adresa. Koristan je za testiranje servisa, međutim nije preporučljivo koristiti ga u produkciji. Kreiranje sopstvenog, prilagođenog VPC-a nudi mogućnosti povećanja bezbjednosti kreiranjem sopstvenih podmreža i pravila pristupa za njih.

Podrazumijevano, EC2 instance koje se pokrenu u prilagođenom VPC-u ne mogu da komuniciraju sa udaljenom mrežom. Ta komunikacija može da se ostvari pomoću **VPN** (*Virtual Private Network)* veza. Jedna od njih je *Site-to-site* veza, gdje se sa AWS strane veze postavlja *Virtual Private Gateway*, a na udaljenoj mreži se postavlja konfigurisani fizički uređaj ili aplikacija (*Customer Gateway Device*). VPN tunel za komunikaciju će se aktivirati kada se generše saobraćaj sa udaljene mreže.Na slici 3.4 prikazana je opisana arhitektura.



1. VPN veza [[27]](#_LITERATURA_1)

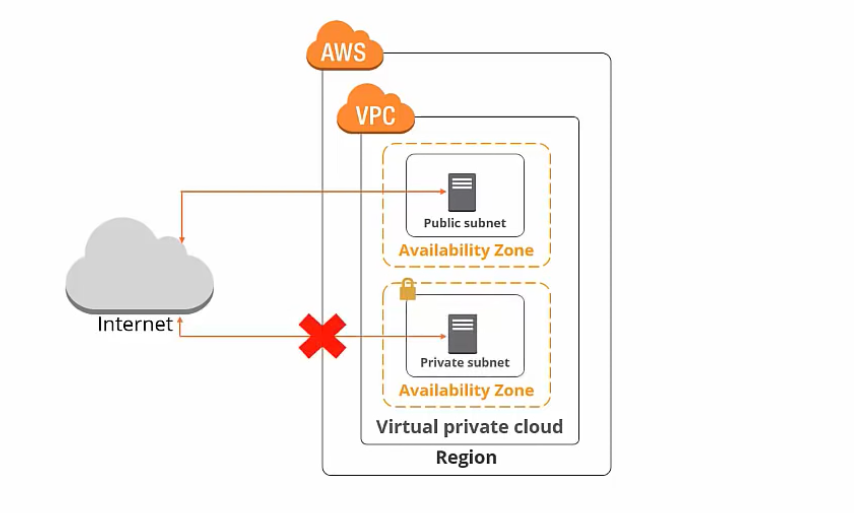
Moguće je i uspostaviti vezu sopstvenog prilagođenog VPC-a sa bilo kojim VPC-om koji se nalazi u istom regionu, bez obzira na to kojem korisničkom nalogu pripada. Ta komunikacija se naziva **VPC *Peering*** i predstavlja vezu tipa jedan-na-jedan, što znači da veza nije tranzitivna. Da bi instance jednog VPC-a mogle da komuniciraju sa instancama drugog, oni moraju da budu direktno povezani. Takođe, moraju da imaju različit opseg IP adresa, jer u suporotnom ne bi mogli da se povežu. Na slici 3.5 je prikazan primjer gdje VPC A može da komunicira sa VPC-om B i VPC-om C, ali B i C ne mogu da komuniciraju međusobno.



1. VPC *Peering* [[27]](#_LITERATURA_1)

**Privatne IP** **adrese** su adrese koja nisu dostupne putem interneta, i koriste se za komunikaciju između instanci u istoj mreži. Prilikom pokretanja, nova instanca dobija svoju privatnu IP adresau i interni DNS naziv (engl. *internal DNS* *hostname*) koji odgovara toj adresi. Da bi se pristupilo instanci putem interneta potrebna je **javna IP adresa**, kojoj se dodjeljuje eksterni DNS naziv (engl. *external DNS hostname*). Javne IP adrese se dodjeljuju iz rezerve IP adresa (*Amazon Pool of Public IP Addresses*). Kada se instanca zaustavi ili terminira, adresa se oslobađa, a prilikom novog pokretanja instance dodjeljuje se nova adresa. Ukoliko je neophodno da instanca zadrži svoju javnu adresu, koristi se **elastična IP adresa**. Ona predstavlja statičnu, perzistentnu javnu adresu, koju je potrebno prvo alocirati na korisničkom nalogu, a zatim pridružiti instanci.

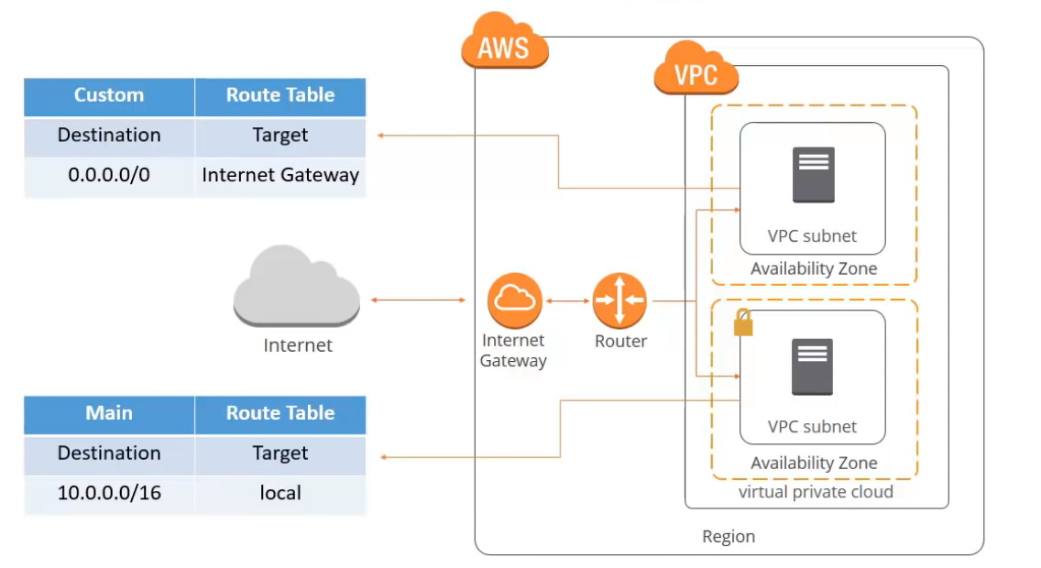
Opseg IP adresa u korisničkom VPC-u čini **podmrežu** (engl. *subnet*). Da bi se pokrenula instanca u nekom VPC-u neophodno je da postoji podmreža. Razlikuju se javne i privatne podmreže, u zavisnosti od toga da li resursi koji se pokreću u njima treba da budu povezani sa internetom. Na primjer, veb serveri uglavnom zahtjevaju vezu sa internetom, dok kod baza podataka to nije neophodno i one mogu da se smjeste u privatnu podmrežu. Svaki VPC sadrži podrazumijevanu podmrežu čija mrežna maska je 20, što znači da nudi 4096 adresa. VPC može da se prostire preko više zona dostupnosti, ali podmreža je uvijek mapirana na jednu zonu. Na slici 3.6 prikazan je VPC sa dvije zone dostupnosti, od kojih se u svakoj nalazi po jedna podmreža.



1. Primjer VPC-a [[27]](#_LITERATURA_1)

Ono što javnu podmrežu čini javnom je njena **tabela rutiranja**, koja pomoću ***Internet Gateway***-a šalje i prima mrežni saobraćaj sa interneta kada je to potrebno. Tabela rutiranja određuje kuda je mrežni saobraćaj usmjeren definisanjem skupa pravila koja se nazivaju rute. Svakoj podmreži je pridružena tabela rutiranja koja kontroliše rutiranje za tu podmrežu. Jednoj podmreži ne može biti pridruženo više tabela rutiranja istovremeno, ali tabela rutiranja može biti pridružena različitim mrežama istovremeno. Svaki VPC ima podrazumijevanu tabelu rutiranja. Na slici 3.7 prikazana je proširena prethodno opisana arhitektura, sa glavnom, odnosno podrazumijevanom, i novom tabelom rutiranja koje su pridružene javnoj i provatnoj podmreži.

*Internet Gateway* je horizontalno skalirana i visoko dostupna VPC komponenta koja obezbjeđuje komunikaciju instanci sa internetom. Ima ulogu da obezbijedi cilj (engl. *target*) u tabeli rutiranja za rutu koja usmjerava saobraćaj ka internetu. Instance koje komuniciraju sa internetom moraju da imaju javnu IP adresu, odnosno da budu u javnoj podmreži. Da bi dobio pristup internetu, VPC-u se pridružuje tačno jedan *Internet Gateway.*

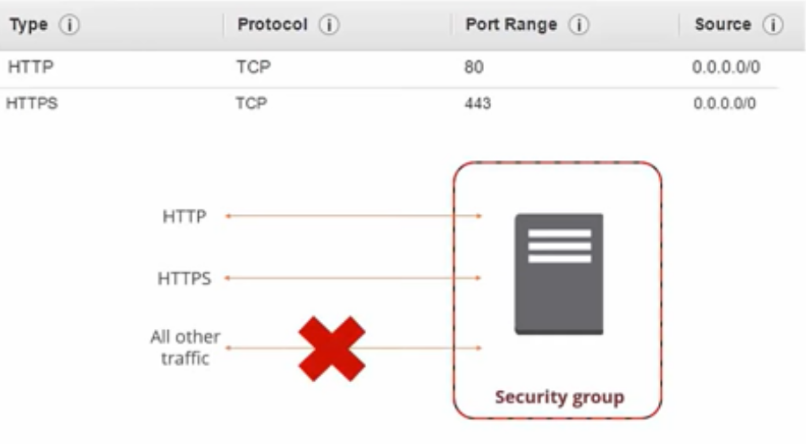


1. Proširen primjer VPC-a [[27]](#_LITERATURA_1)

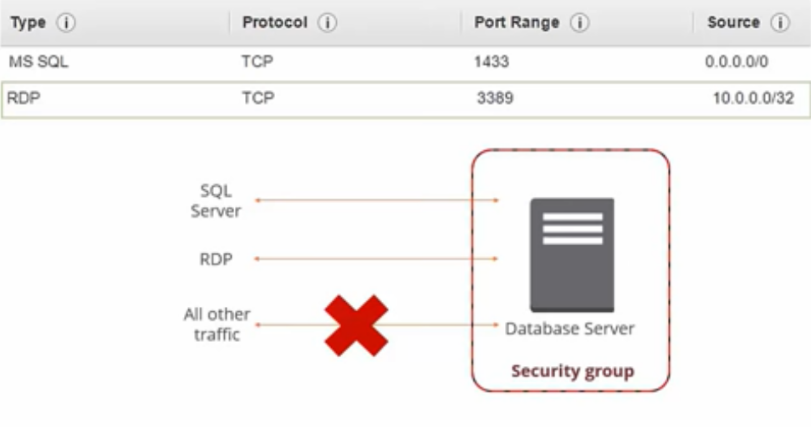
U slučajevima kada je potrebno da instance koje se nalaze u privatnoj podmreži imaju pristup internetu ili drugim AWS resursima, ali da ostane onemogućeno da internet inicira komunikaciju sa njima, koristi se **NAT** ***Device***(engl. *Network Address Translation*). NAT *Device* prosljeđuje saobraćaj od privatne podmreže ka internetu ili drugim resursima i vraća dobijeni odgovor instanci. Kada se saobraćaj odvija prema internetu, izvorna IP adresa instance je zamijenjena adresom NAT *Device*-a, a kada se odgovor vraća adresa se ponovo prevodi u privatnu adresu instance. NAT *Device* mora da bude smješten u javnu podmrežu i da ima pridruženu elastičnu IP adresu. Da bi se sve to omogućilo, potrebno je podesiti tabelu rutiranja privatne podmreže tako da prosljeđuje saobraćaj NAT *Device*-u. Postoje dva tipa NAT *Device*-a: NAT *Gatway* i NAT *Instance*. Preporučuje se prvi, jer nudi veću dostupnost i zahtjeva manje održavanja.

Da bi komunikacija instanci sa internetom bila bezbjedna, potrebno je podesiti pravila za pristup mreži tako da dozvoljavaju samo željeni saobraćaj. Koriste se ***security* grupe** (engl. *security group*) i **liste kontrole pristupa** (engl. *Network* *Access Control List*, *Network* ACL).

*Security* grupe služe kao virtuelni zid koji kontroliše saobraćaj instance. Definišu se pravila koja označavaju dozvoljeni dolazeći ili odlazeći saobraćaj (engl. *inbound rules, outbound rules*) za instance kojima je pridružena grupa. Na slici 3.8 prikazana je *security* grupa pridružena instanci veb servera koja dozvoljava HTTP i HTTPS saobraćaj sa bilo kojeg izvora, a na slici 3.9 *security* grupa pridružena instanci servera baze podataka koja dozvoljava SQL i RDS saobraćaj, ali ograničava izvor RDP saobraćaja [[28]](#_LITERATURA_1).

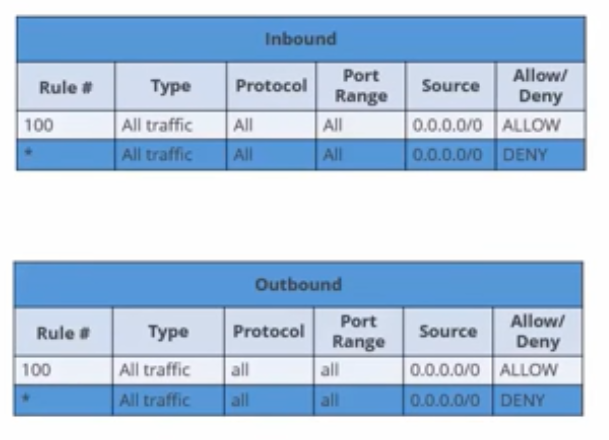


1. *Security* grupa za veb server [[27]](#_LITERATURA_1)

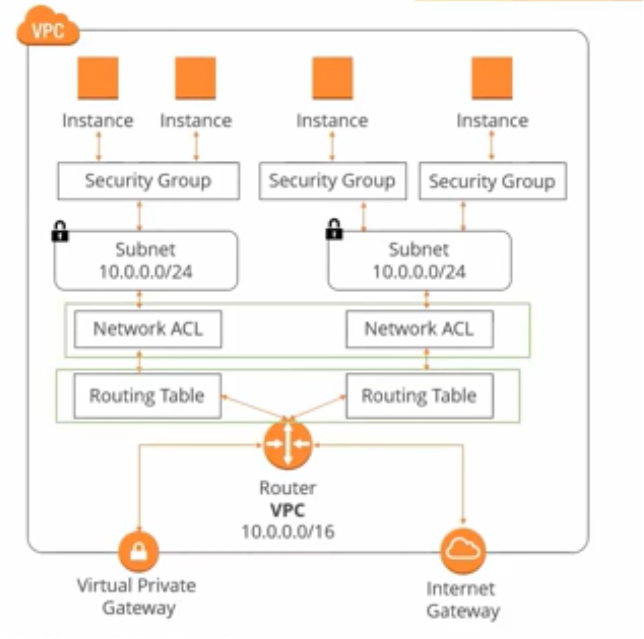


1. *Security* grupa za server baze podataka [[27]](#_LITERATURA_1)

*Network* ACL je opcioni sloj bezbjednosti VPC-a koji služi kao virtuelni zid koji kontroliše saobraćaj podmreže. Definišu se pravila koja označavaju dozvoljeni ili zabranjeni dolazeći ili odćazeći saobraćaj. Na slici 3.10 prikazana su pravila za podrazumijevani ACL koji se automatski dodjeljuje podmreži. Pravila se izvršavaju redom, a posljednje, označeno zvjezdicom, obezbjeđuje da svaki paket koji se ne poklapa ni sa jednim iznad navedenim pravilom bude odbijen. Navedeni slojevi zaštite instanci u VPC prikazani su na slici 3.11.



1. Podrazumijevani *Network* ACL [[27]](#_LITERATURA_1)



1. Slojevi zaštite instanci u VPC [[27]](#_LITERATURA_1)

# SPECIFIKACIJA SISTEMA

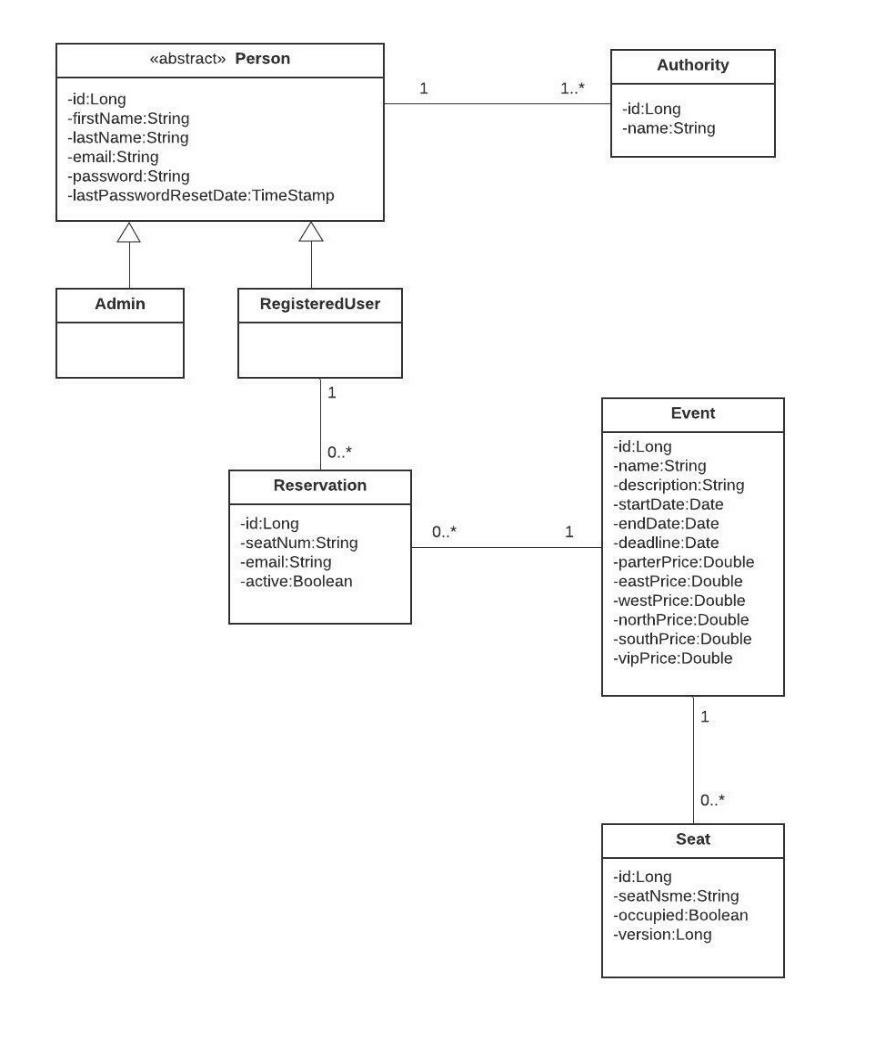
U radu je prikazan primjer jednostavne veb aplikacije koja zahtjeva elastičnost i visoku dostupnost. Aplikacija je namijenjena korisnicima koji žele da rezervišu karte za različite događaje (kulturne manifestacije, koncerte, sportske doga]aje) u okviru stadiona/arene. U trenutku kada se doda novi događaj u sistem, može se desiti da veliki broj korisnika odjednom krene da rezerviše kartu što će dovesti do velikog opterećenja, i u tom slučaju je potrebno obezbijediti više resursa i omogućiti visoku dostupnost. U nastavku je dat dijagram klasa sistema, funkcionalnosti koje omogućava i prijedlog arhitekture koja može obezbijediti elastičnost i visoku dostupnost.

## Dijagram klasa

Model aplikacije „Arena“ sastoji se od sljedećih klasa (slika 4.1):

* *Person*
* *Admin*
* *RegisteredUser*
* *Authority*
* *Event*
* *Seat*
* *Reservation*

Klasa *Person* je apstraktna klasa koja sadrži osnovne podatke o korisniku sistema (ime i prezime) i podatke potrebne za prijavu na sistem (*email* i lozinka). Na osnovu šablona koji koristi *spring security,* svaki korisnik može biti vezan za jednu ili više uloga predstavljenih klasom *Authority*. U ovom sistemu postoje dvije uloge: administrator i obični korisnik. To je predstavljeno klasama *Admin* i *RegisteredUser* koje nasljeđuju klasu *Person.* Klasa *Event* predstavlja jedan događaj i sadrži sljedeće atribute: naziv događaja, opis događaja, datum i vrijeme početka i završetka održavanja, daum i vrijeme do kojeg je moguće rezervisati i otkazivati rezervacije i šest različitih cijena u zavisnosti od položaja sjedišta (parter, VIP loža, istočne, zapadne, sjeverne i južne tribine). Ukoliko vrijednost cijene nekog tipa sjedišta nije navedena, podrazumijeva se da se za konkretan događaj ne prodaju karte tog tipa sjedišta. Za svaki događaj vezana je lista sjedišta predstavljenih klasom *Seat*, sa podacima o nazivu sjedišta, da li je sjedište zauzeto ili nije, i verziji objekta u bazi podataka, što je značajno za održavanje konzistentnosti baze i o čemu će biti riječ u poglavlju o implementaciji. Obični korisnik ima listu svojih rezervacija predstavljenih klasom *Reservation*. Svaka rezervacija je vezana za jedan događaj i sadrži naziv odabranog sjedišta, *email* korisnika koji je rezervisao sjedište i podatak da li je rezervacija trenutno aktivna, odnosno da li je otkazana ili nije.



1. Dijagram klasa

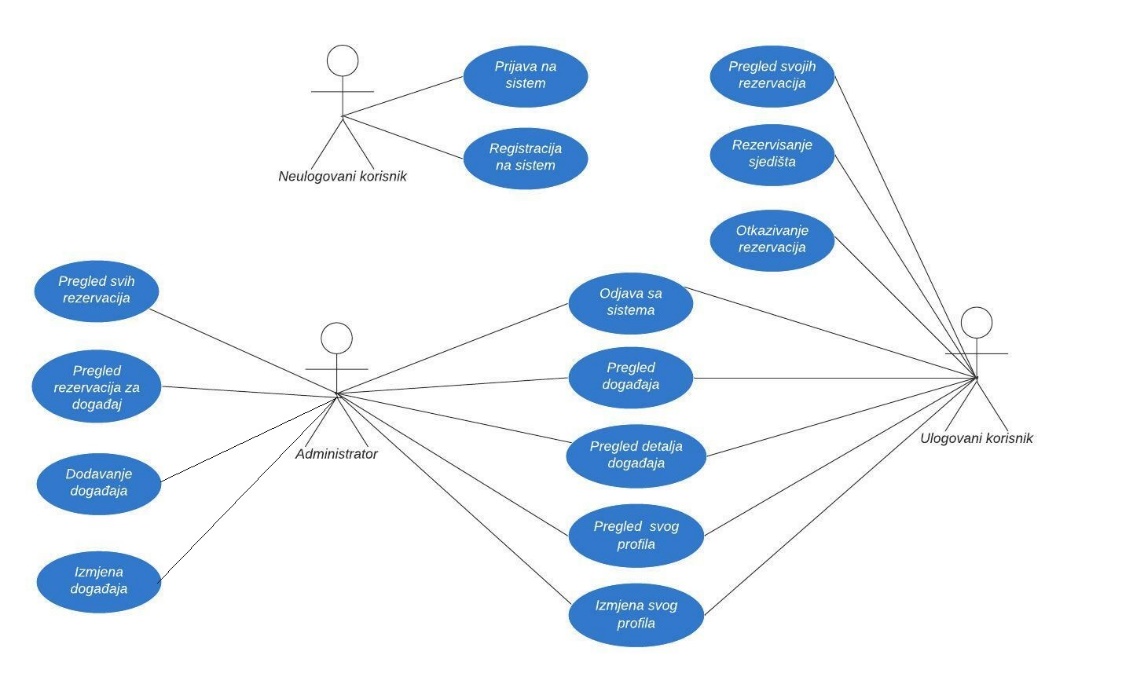
## Dijagram slučajeva korišćenja

Aplikacija podržava tri tipa korisnika: neulogovani korisnik, obični korisnik i administrator. Na slici 4.2 prikazan je dijagram slučajeva korišćenja za svaki tip.

Neulogovani korisnik može da se registruje na sistem ili da se prijavi na sistem. Kada se prijavi dobija ulogu običnog korisnika ili administratora. Zajedničke funkcionalnosti ova dva tipa korisnika su: odjava sa sistema, pregled svog profila, izmjena svog profila (ime, prezime, lozinka), pregled svih događaja u sistemu sortiranih od najnovijeg ka najstarijem i detaljni pregled svakog događaja. On uključuje vizuelni prikaz arene koji razlikuje zauzeta sjedišta, slobodna sjedišta i sjedišta zauzeta od strane ulogovanog korisnika.

Funkcionalnosti koje obavlja samo administrator su: pregled svih rezervacija u sistemu sortiranih od najnovije ka najstarijoj, pregled rezervacija za izabrani događaj, dodavanje novog događaja, izmjena postojećeg događaja. Pri dodavanju i izmjeni događaja administrator može da unese naziv, opis, cijene za različite tipove sjedišta, datum i vrijeme početka i završetka i datum i vrijeme do kojeg se mogu rezervisati sjedišta i otkazivati rezervacije. Sistem obavještava administratora i ne dozvoljava izvršavanje operacije ako se uneseni datumi preklapaju sa datumima i vremenom nekog već postojećeg događaja ili ako nisu validni.

Obični korisnik ima mogućnost da pregleda svoje rezervacije, rezerviše sjedišta za neki događaj i otkazuje svoje rezervacije. Sistem vodi računa o tome da korisnik ne može da rezerviše sjedište koje je u međuvremenu neko drugi rezervisao, da ne može da rezerviše više od četiri sjedišta za jedan događaj i da ne može da rezerviše ili otkazuje rezervacije ukoliko je prošao datum do kojeg je to dozvoljeno. Pri uspješnoj rezervaciji ili otkazivanju korisniku se šalje *email* sa nazivom i vremenom događaja, nazivom sjedišta koje je rezervisao ili otkazao i njihovom cijenom.



1. Dijagram slučajeva korišćenja

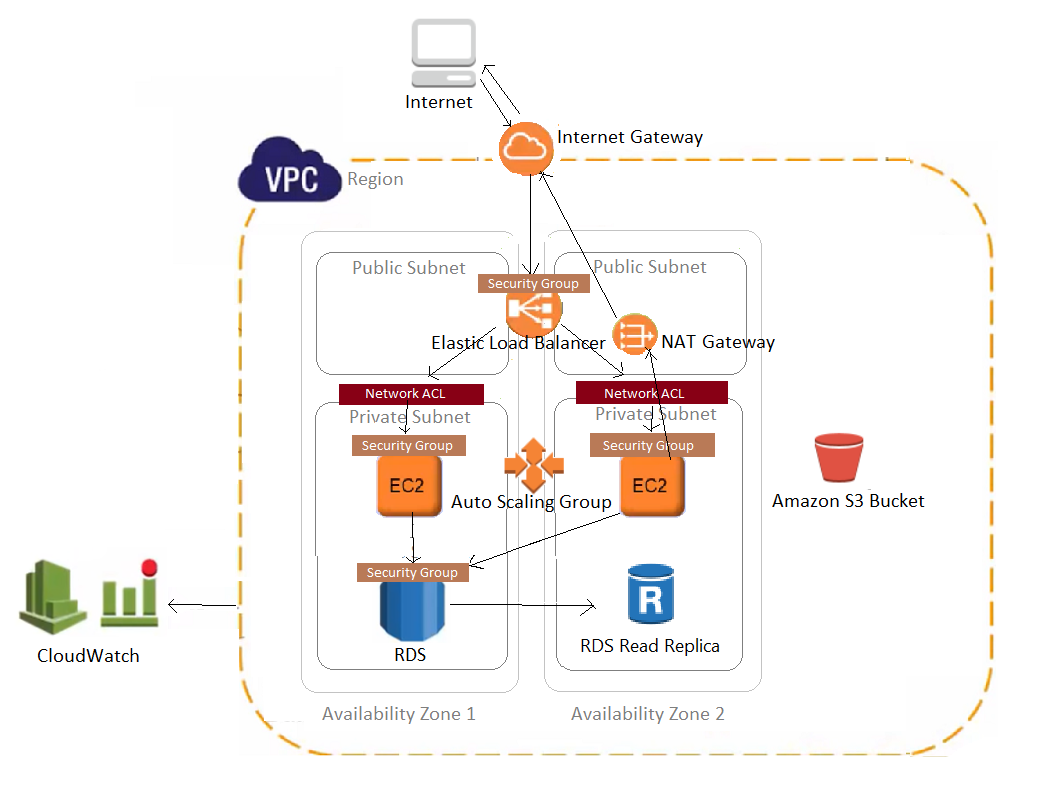
## Arhitektura sistema

U ovom odjeljku opisana je predložena arhitektura sistema dizajnirana da omogući što veću elastičnost i dostupnost. Pošto neki od prethodno opisanih AWS servisa nisu besplatni, implementirana arhitektura nije optimalna, ali je dat teoretski opis naprednije arhitekture koja bi obezbijedila elastičnost i visoku dostupnost.

Sistem se sastoji od klijentske i serverske strane koje su postavljene na arhitekturu sastavljenu od AWS servisa. Klijentska strana je implementirana kao *Angular* aplikacija i postavljena kao statički veb sajt na S3 *bucket.* Serverska strana je implementirana kao *Spring Boot* veb aplikacija, i njen izvorni kod je takođe postavljen na S3 *bucket*, odakle ga preuzimaju i pokreću EC2 instance. Pristup izvornom kodu zaštićen je upotrebom IAM uloge, odnosno pristup je dozvoljen samo EC2 instancama kojima je dodijeljena odgovarajuća uloga.

EC2 instance čine *Auto Scaling* grupu, čiji kapacitet se mijenja u zavisnosti od opterećenja, što obezbjeđuje elastičnost. Minimalan kapacitet je 1, a maksimalan 2, što je dovoljno za demonstraciju. Skaliranje se izvršava automatski, kao posljedica okidanja *CloudWatch* alarma koji prati opterećenje procesora *Auto Scaling* grupe. EC2 instance su raspoređene u dvije zone dostupnosti, što obezbjeđuje dostupnost jer će se u slučaju da dođe do otkaza u jednoj zoni pokrenuti nova instanca u drugoj zoni, a ako ona već postoji nastaviće da radi. Zahtjevi koji se šalju od klijenta ka serveru upućuju se *Application Load Balancer*-u (ALB), koji ih raspoređuje na EC2 instance *Auto Scaling* grupe. ALB je izabran iz razloga što radi sa HTTP zahtjevima i prilagođen je za veb aplikacije. Serveri šalju upite *PostgreSQL* bazi podataka pokrenutoj na RDS instanci. RDS instanca ima *read* repliku kojoj se prosljeđuju svi upiti za čitanje i koja se nalazi u drugoj zoni dostupnosti. U slučaju otkaza prve zone, ona može ručno da se unapredi i da postane primarna baza.

Svi servisi su postavljeni u korisnički definisanom VPC-u u privatne ili javne podmreže. EC2 instance *Auto Scaling* grupe i RDS instance su postavljene u privatne podmreže, što znači da im se ne može pristupiti direktno sa interneta. Komunikacija u obrnutom smjeru je moguća, jer je u javnoj podmreži postavljen NAT *Gateway* koji obezbjeđuje instancama pristup internetu i drugim servisima. Pristup im je neophodan zbog instalacije programskog jezika *Java* i preuzimanja izvornog koda pri pokretanju. Instance su zaštićene *security* grupama i ACL listom koje definišu da se bazi podataka može pristupiti samo preko EC2 instance, a EC2 instanci samo preko ALB-a. Dakle, ALB i NAT *Gateway* su jedini koji se nalaze u javnoj podmreži. Opisana arhitektura prikazana je na slici 4.3.



1. Implementirana arhitektura sistema

Mana ove arhitekture je to što nije obezbijeđena elastičnost i dostupnosti baze podataka. To bi se moglo omogućiti korišćenjem opcije *Multi-AZ Deployment* kod RDS instance. Ova opcija podrazumijeva automatsko kreiranje *standby* replike u nekoj drugoj zoni dostupnosti, koja bi služila kao rezervna kopija i u slučaju otkaza primarne baze preuzela bi njenu ulogu (slika 4.4).



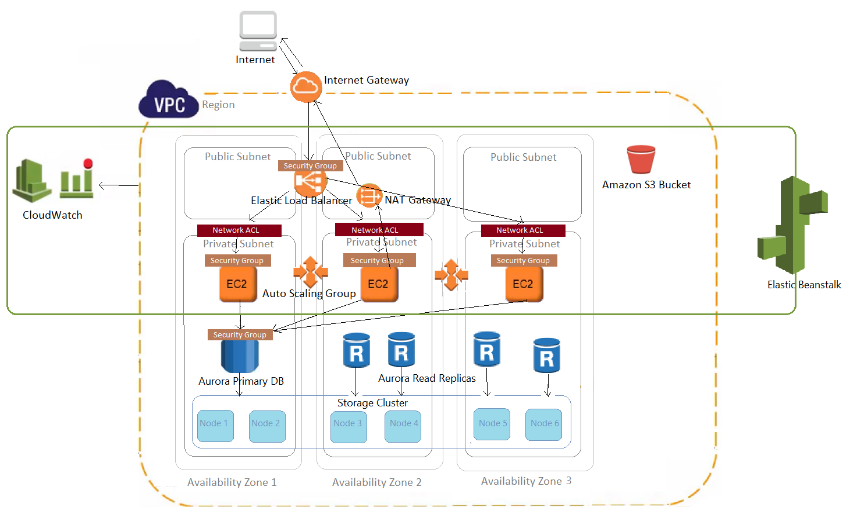
1. Arhitektura sistema sa RDS *multi-AZ*

Kako bi se obezbijedila i elastičnost može se koristiti *Amazon Aurora*, koja pruža mogućnost automatskog skaliranja *read* replika na osnovu praćenja opterećenja i definisane politike skaliranja (slika 4.5). *Amazon Aurora* nudi i prednosti vezane za performanse (v. odjeljak 3.5).



1. Arhitektura sistema sa *Amazon Aurora*

Navedena arhitektura može da se postigne i korišćenjem servisa *Elastic Beanstalk*, koji je namijenjen za postavljanje skalabilnih veb aplikacija i objedinjuje sve servise vezane za EC2 instance, rutiranje i skaliranje. Na slici 4.6 prikazano je koje servise može da zamijeni *Elastic Beanstalk*.



1. Arhitektura sistema sa *Elastic Beanstalk*

# IMPLEMENTACIJA SISTEMA

U ovom poglavlju biće opisana implementacija aplikacije i koraci koji su izvršeni da bi se ona postavila na navedenu AWS arhitekturu. Aplikacija se sastoji od klijentske i serverske strane koje komuniciraju slanjem HTTP zahtjeva i odgovora. Kod aplikacije dostupan je na linku [[39]](#_LITERATURA_1).

## Klijentska strana

Za izradu klijentske strane korišćen je *Angular 11*. Funkcionalnosti i komponente su podijeljene na nekoliko modula koji su importovani u korijenski modul (*AppModule*):

* modul za rutiranje (*AppRoutingModule*);
* modul za navigaciju (*NavigationsModule*);
* modul za autentikaciju (*AuthModule*);
* modul koji sadrži zajedničke komponente (*SharedModule*);
* modul za rad sa korisnicima (*UsersModule*);
* modul za rad sa događajima i rezervacijama (*EventsModule*).

Modul za rutiranje sadrži niz ruta, koje su definisane pomoću tri stavke: *path*, koja predstavlja URL (*Uniform Resource Locator*) prikaza stranice (engl. *view*), zatim *component*, koja predstavlja komponentu koja se učitava prilikom pristupa URL-u rute i *canActivate*, koja predstavlja *guard* rute. *Guard* je klasa koja implementira interfejs *CanActivate* sa istoimenom metodom i provjerava da li uloga korisnika ima pristup ruti. Ukoliko ima, vraća *true*, a ukoliko nema, preusmjerava korisnika na početnu stranicu i vraća *false*. Uloga korisnika je definisana kao enumeracija *UserRole* sa vrijednostima *UNAUTHORIZED, ROLE\_ADMINISTRATOR* i *ROLE\_USER*. Modul za navigaciju čini komponenta koja predstavlja meni aplikacije koji je vidljiv na svakoj stranici, odnosno *navigation bar*. U zavisnosti od uloge korisnika ona prikazuje različite opcije aktivirajući jednu od tri komponente: *NavigationAdminComponent, NavigationUserComponent, NavigationNotLoggedInComponent*.

Za prijavljivanje i registrovanje korisnika zadužen je modul za autentikaciju, koji sadrži dvije komponente, *LogInComponent* i *SignUpComponent*. Prilikom prijavljivanja na sistem u *local storage*-u *browser*-a čuva se uloga korisnika, *email*, identifikator i JWT (JSON *web token*). JWT token se zatim koristi pri slanju zahtjeva ka serveru, tako što klasa koja implementira interfejs *HttpInterceptor* presreće svaki zahtjev i ubacuje token u njegovo zaglavlje, kako bi se mogla izvršavati autentikacija i autorizacija na serveru. Modul za rad sa korisnicima sadrži komponentu *ViewProfileComponent* za prikaz i izmjenu osnovnih podataka ulogovanog korisnika, a modul za rad sa događajima sadrži četiri komponente: *CreateEventComponent* za dodavanje i izmjenu događaja*, EventComponent* za vizuelni prikaz arene i rezervisanje sjedišta za pojedinačni događaj*, ViewEventsComponent* za pregled događajai *ViewReservationsComponent* za pregled rezervacija*.* Sve forme u aplikaciji implementirane su kao reaktivne forme koje su bazirane na modelu i kod kojih je većina logike implementirana u *TypeScript* fajlu, što daje mogućnost da se lakše implementira validacija svakog *FormControl*-a, odnosno polja u formi. Prilikom kreiranja forme pomoću *FormBuilder*-a svakom polju se prosljeđuju odgovarajući validatori. Validator predstavlja funkciju koja implementira interfejs *ValidatorFn* i vraća *null* ako korisnički unos ispunjava sve uslove, ili listu grešaka ako ih ne ispunjava. Pored osnovnog validatora koji zahtjeva da polje ne bude prazno, u formama koje traže unos lozinke koriste se i dva prilagođena validatora, koji zahtjevaju da lozinka ima bar 8 karaktera i da se lozinka i potvrda lozinke podudaraju. Navedene komponente se oslanjaju na tri zajedničke komponente: *TableComponent* za tabelarni prikaz podataka, *PaginationComponent* za paginaciju tabele i *YesNoPipe*, kao *pipe* funkciju koja transformiše prikaz *boolean* atributa.

Za slanje HTTP zahtjeva zadužene su servisne klase koje su injektovane u module pomoću *Dependency Injection* mehanizma, što znači da *Angular* okruženje vodi računa o njihovom životnom ciklusu i da su dostupni svim komponentama. Koriste servis *HttpClient*, koji šalje zahtjev serveru (*GET*, *POST*, *PUT* ili *DELETE*) i vraća *Observable* objekat. *Subscribe*-ovanjem na taj objekat dobija se odgovor od servera. API kojem su upućeni zahtjevi definisan je u fajlu *environments.prod.ts* kao globalna varijabla, i predstavlja *endpoint* definisan za *Load Balancer*, o čemu će biti riječi kasnije. Korišćenjem alata *Angular CLI* izvršava se instalacija svih potrebnih zavisnosti i modula komandom *npm install* i izgradnja aplikacije u produkcionom režimu komandom *ng build --prod*. Fajlovi koji se dobiju smještaju su u folder *dist*, a zatim se *upload*-uju na AWS[[13]](#_LITERATURA_1).

## Serverska strana

Serverska strana implementirana je pomoću *Spring Boot* tehnologije korišćenjem *Eclipse IDE* razvojnog okruženja sa *plugin*-om *Apache Maven* za izgradnju aplikacije [[40]](#_LITERATURA_1). U fajlu *pom.xml* navedene su osnovne informacije o nazivu projekta, verziji izvornog koda, izlaznom formatu pri izgradnji aplikacije, bibliotekama i zavisnostima koji su potrebni za kompajliranje i izvršavanje. U fajlu *application.properties* navedena je konfiguracija aplikacije vezana za pristup bazi podataka i AWS nalogu i slanje mejla. Aplikacija se pokreće na portu 8080. Ima troslojnu arhitekturu, sa prezentacionim slojem (*Presentation Layer*), slojem poslovne logike *(Business Logic Layer*) i slojem podataka (*Data Access Layer*).

Za bezbjednost aplikacije korišćen je *Spring Security*, koji vodi računa o autentikaciji korisnika i autorizaciji HTTP zahtjeva dobijenog od strane klijenta. U fajl *pom.xml* je uključena odgovarajuća zavisnost. Korisnik je predstavljen apstraktnom klasom *Person* i klasama *Admin* i *RegisteredUser* koje je nasljeđuju. Ove klase implementiraju interfejs *UserDetails*, koji specificira metodu za dobavljanje liste uloga korisnika. To znači da korisnik mora biti vezan za bar jednu ulogu predstavljenju klasom *Authority*. Takođe, korisnik ima *email* kao jedinstven identifikator i lozinku koja se hešira upotrebom klase *BCryptPasswordEncoder*. Ova klasa implementira inerfejs *PasswordEncoder* sa metodama *encode* za šifrovanje i *match* za provjeru podudaranja lozinki. *Spring Security* koristi JWT da obezbijedi *stateless* komunikaciju. Prilikom prijave na sistem kreira se token sa informacijama o algoritmu, *email*-om, lozinkom, ulogom korisnika i potpisom. Prilikom obrade pristiglog zahtjeva, na osnovu *email*-a i lozinke izvršava se autentikacija korisnika, a na osnovu uloge izvršava se autorizacija, tj. provjera da li uloga ima pristup *endpoint*-u. Navedena podešavanja nalaze se u paketu *security* i u klasi *WebSecurityConfiguration* paketa *config.* U toj klasi su navedene i putanje kojima nije zaštićen pristup, a to su *endpoint*-i za prijavu i registraciju. U klasi *WebConfiguration* istog paketa obezbjeđena je podrška za *Cross-Origin Resource Sharing* (CORS) za *endpoint* klijentske aplikacije, što znači da će *browser* dozvoliti pristup resursima sa drugog izvora, u ovom slučaju porta 8080.

Prezentacioni sloj implementiran je kao REST kontroler, što je označeno anotacijama *RestController* i *ReqestMapping*, sa parametrima *value* koji predstavlja putanju i *produces* koji označava format resursa koji se šalje klijentu, u ovom slučaju JSON. Kontroleri sadrže odgovarajuće servisne klase kao atribute, injektovane pomoću anotacije *Autowired*, što znači da mogu da pozivaju njihove metode bez da vode računa o njihovom životnom ciklusu. Na listingu 5.1 prikazan je primjer metode kontrolera za rezervaciju koja je tipa *POST* i prima resurs u JSON format u okviru tijela zahtjeva. Resursi su objekati DTO klasa (*Data Transfer Object*), koje predstavljaju posebne klase za slanje objekata, kako se ne bi slali perzistentni objekti modela. Anotacija *PreAuthorize* označava ulogu koja ima pristup datoj metodi. Kontroler vraća klijentu odgovor tipa *ResponseEntity* sa statusom *OK* ili *BadRequest.*

@PreAuthorize("hasRole('ROLE\_USER')")

@RequestMapping(value = "/make-reservation", method = RequestMethod.***POST***, consumes = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<Object> makeReservation(@RequestBody ReservationDTO dto) {

List<String> result = reservationService.makeReservations(dto);

**if** (result != **null**)

**return** **new** ResponseEntity<>(result, HttpStatus.***OK***);

**else**

**return** **new** ResponseEntity<>(

"Reservation has failed. The seats you marked have been occupied in the meantime.",

HttpStatus.***BAD\_REQUEST***);

}

1. Primjer metode kontrolera

Sloj poslovne logike čine servisne klase u kojima je implementirana logika sistema. Kao atribute sadrže injektovane druge servise i repozitorijumske interfejse. Metode servisa su označene anotacijom *Transactional* sa parametrom *readOnly* koji može da bude *true* ili *false*. Na listingu 5.2 prikazan je primjer logike provjere ispravnosti rezervacije, pri čemu se u slučaju nevalidnog datuma ili prevelikog broja rezervacija baca izuzetak *ReservationException*. Taj izuzetak će biti uhvaćen od strane globalnog *ExceptionHandler*-a koji će da presretne HTTP odgovor i podesi status *BadRequest* sa proslijeđenom porukom.

**if** (e.getDeadline().before(**new** Date())) {

**throw** **new** ReservationException("You cannot make a reservation after " + *sdf*.format(e.getDeadline()));

}

**if** (reservationRepository.checkReserved(dto.getUserEmail(), e.getId()) + dto.getSeatNums().size() > 4L) {

**throw** **new** ReservationException("You cannot make more than 4 reservations for one event.");

}

1. Primjer logike servisa

Poseban servis za slanje mejla korisniku pri uspješnoj rezervaciji ili otkazivanju koristi *JavaMailSender* interfejs i *smtp* protokol za slanje mejla. Metode ovog servisa označene su anotacijom *Async*, kako bi se mejlovi mogli slati asinhrono. U fajlu *application.properties* podešeni su parametri za *gmail,* što znači da parametar *host* ima vrijednost *smtp.gmail.com*, a parametar *port* vrijednost *587*.

Sloj za pristup bazi podataka implementiran je pomoću *Spring Data* JPA, koji omogućava rad sa relacionim bazama podataka, u ovom slučaju *PostgreSQL* bazom. JPA koristi objektno-relaciono mapiranje (ORM) između tabela baze podataka i entiteta modela aplikacije. Na listingu 5.3 prikazan je primjer entiteta *Seat* na osnovu kojeg se kreira tabela u bazi sa kolonama koje odgovaraju atributima. Kao identifikator se koristi kolona *id* čija vrijednost se određuje automatski pomoću *sequence* ili *identity* generatora. U ovoj klasi je takođe implementirano optimističko zaključavanje objekta [[41]](#_LITERATURA) dodavanjem atributa *version*. Potrebno je voditi računa o verziji sjedišta jer se može desiti da jedan korisnik pročita podatke o trenutno slobodnom sjedištu iz baze, zatim neki drugi korisnik zauzme to sjedište, pa prvi korisnik pokuša takođe da ga zauzme. U tom slučaju trenutna verzija će se razlikovati od posljednje pročitane verzije i prvom korisniku će se prijaviti greška i onemogućiti akcija.

@Entity

@Table(name = "seats")

**public** **class** Seat **implements** Comparable<Seat> {

@Id

@GeneratedValue(strategy = GenerationType.***SEQUENCE***, generator = "row\_generator")

@SequenceGenerator(name = "row\_generator", sequenceName = "db\_generator", allocationSize = 500)

**private** Long id;

@Column(name = "seat\_name")

**private** String seatName;

@Column(name = "occupied")

**private** Boolean occupied;

@Version

@Column(name = "version")

**private** Long version;

1. Primjer klase modela

JPA koristi i *Entity Manager* API za slanje upita i transakcija bazi podataka, pomoću objektno orijentisanog upitnog jezika JPQL (*Java Persistent Query Language*). Primjer upita za traženje broja rezervacija jednog korisnika za jedan događaj dat je na listingu 5.4. Pisanje osnovnih upita je skraćeno upotrebon JPA repozitorijuma, pri čemu je dovoljno da se implementira repozitorijumski interfejs sa odgovarajućim entitetom da bi se koristile osnovne CRUD operacije. Da bi se aplikacija mogla koristiti, u skripti *import.sql* koja se izvršava pri pokretanju aplikacije definisani su po jedan predefinisani administrator i korisnik sa odgovarajućim ulogama.

@Query("SELECT COUNT(r.id) FROM Reservation r WHERE r.email = ?1 AND r.event.id = ?2 AND r.active = true")

Long checkReserved(String email, Long eventId);

1. Primjer JPQL upita

Pristup AWS servisima, odnosno RDS bazi podataka omogućen je upotrebom *Spring Cloud*-a. U fajlu *pom.xml* podešena je odgovarajuća zavisnost i verzija *Finchley.SR1,* a na listingu 5.5 data su podešavanja u fajlu *application.properties*. *AccessKey* i *secretKey* predstavljanju kredencijale za pristup AWS nalogu i oni mogu da se dobave preko AWS *Management Console*. Naveden je region koji se koristi, u ovom slučaju *us-east-2* ili *Ohio*. Parametar *stack.auto* je podešen na *false* jer se aplikacija ne pokreće preko servisa *CloudFormation*. Takođe, dodata su i dva podešavanja za logovanje koja govore sistemu da prikazuje samo logove tipa *error*. Druga grupa parametara namijenjena je konkretnoj instanci baze podataka, *database-arena*. Podešeni su podaci za pristup bazi (korisničko ime, lozinka i ime baze) sa vrijednostima koje su definisane preko AWS *Management Console*, atribut za podršku za rad sa *read* replikama koja omogućava da sve transakcije kod kojih je vrijednost parametra *readOnly* jednaka *true* budu preusmjerene na *read* repliku, i atribut za podršku za rad sa *batch* iskazima, odnosno mogućnost da se više upita (u ovom slučaju 500) nad bazom izvršava odjednom kako bi se poboljšale performanse.

cloud.aws.credentials.accessKey=AKIAUG5NI6YAQSA45IHO

cloud.aws.credentials.secretKey=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

cloud.aws.credentials.instanceProfile=true

cloud.aws.region.static=us-east-2

cloud.aws.stack.auto=false

logging.level.com.amazonaws.util.EC2MetadataUtils=error

logging.level.com.amazonaws.internal.InstanceMetadataServiceResourceFetcher=error

cloud.aws.rds.database-arena.username=postgres

cloud.aws.rds.database-arena.password=postgresql

cloud.aws.rds.database-arena.databaseName=postgres

cloud.aws.rds.database-arena.readReplicaSupport=true

cloud.aws.rds.database-arena.rewriteBatchedStatements=true

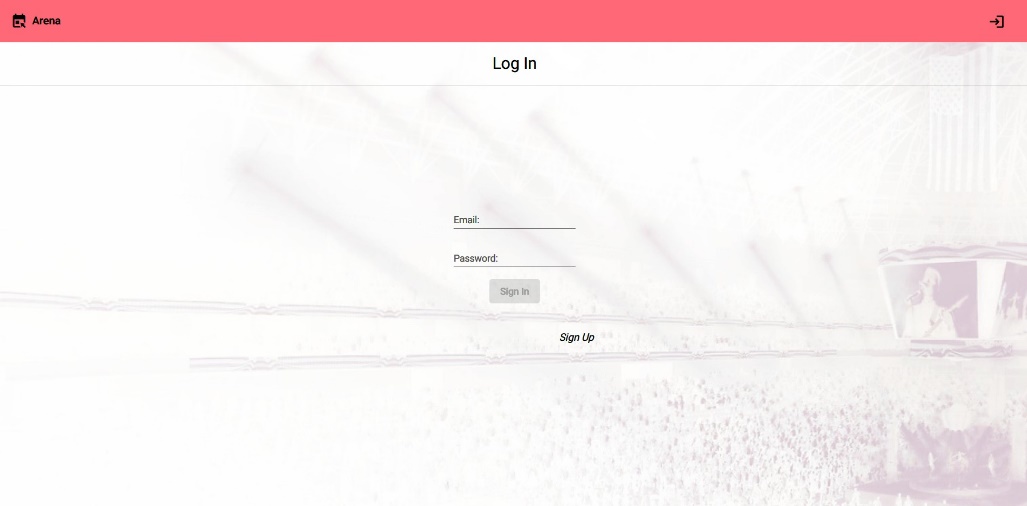
1. Podešavanja za pristup AWS i RDS

Izvršavanjem *maven* komandi *clean compile install* u folderu *target* dobija se fajl *spring-boot-aws-arena.jar*, koji predstavlja zapakovan kod aplikacije i *upload*-uje se na AWS [[12]](#_LITERATURA_1).

# PRIKAZ IMPLEMENTIRANOG SISTEMA

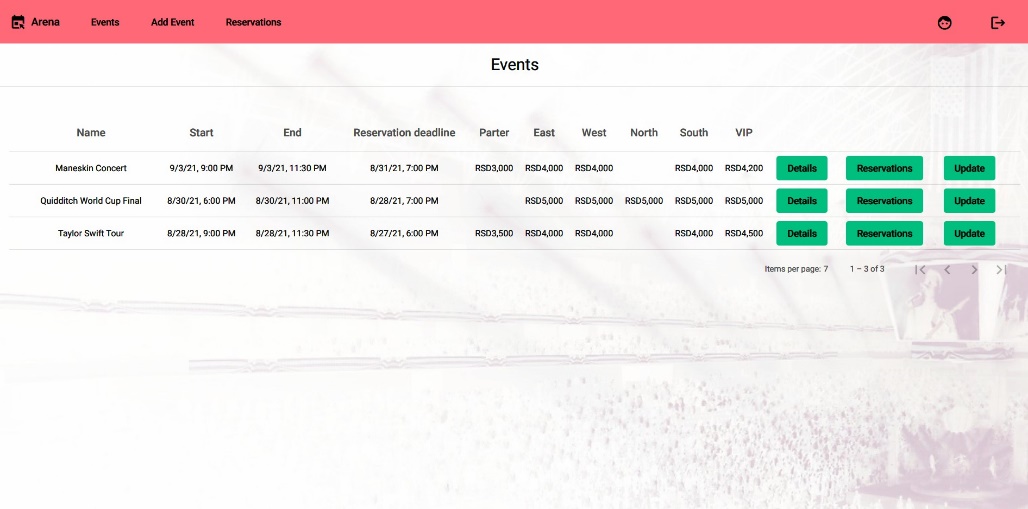
U ovom poglavlju biće opisane i prikazane funkcionalnosti sistema iz ugla administratora i običnog korisnika. Zatim će biti demonstrirano kako sistem postiže elastičnost i dostupnost.

Prilikom pristupa aplikaciji svaki korisnik ima ulogu neprijavljenog korisnika. Dostupna mu je stranica za prijavu (slika 6.1) i stranica za registraciju klikom na dugme *Sign Up*. Da bi se prijavio potrebno je da je prethodno registrovan i da unese svoj *email* i lozinku sa kojima se registrovao. Da bi se registrovao potrebno je da unese *email* koji mora biti jedinistven u sistemu (korisnik ne može da pravi više naloga sa istim *email*-om), ime, prezime, lozinku sa najmanje 8 karaktera i potvrdu lozinke. U sistemu postoji jedan predefinisani administrator, a svaki novi korisnik koji se registruje dobija ulogu običnog korisnika.



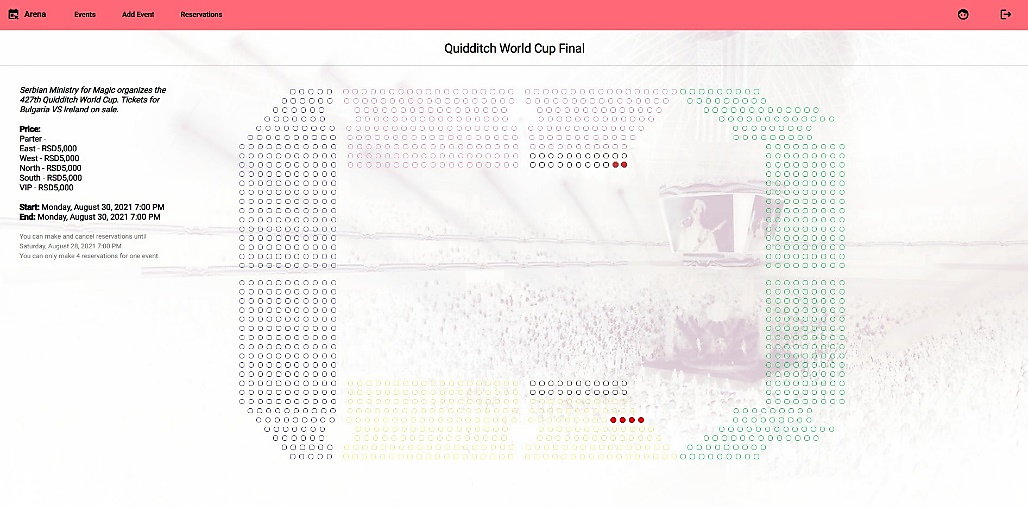
1. Prikaz stranice za prijavu

Kada se korisnik prijavi kao administrator, može da vidi početnu stranicu sa tabelom u kojoj su prikazani svi događaji u sistemu (slika 6.2). Za svaki događaj prikazan je naziv, datum i vrijeme početka, datum i vrijeme završetka, datum i vrijeme do kojeg korisnici mogu rezervisati i otkazivati sjedišta i cijene za različite vrste sjedišta (*parter, east, west, north, south, VIP*). Administrator ima opcije da pogleda detaljan prikaz događaja, sve rezervacije za taj događaj i formu za izmjenu podataka o događaju.



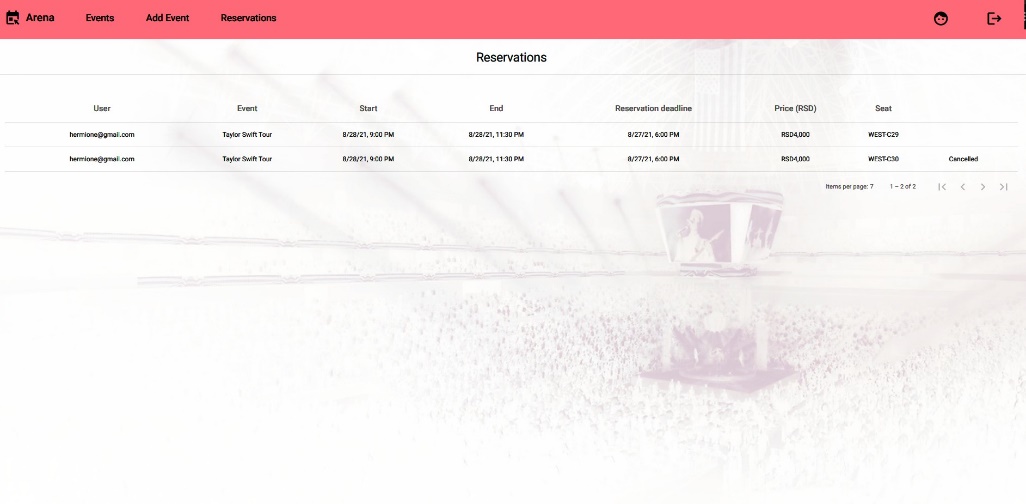
1. Prikaz početne stranice *„Events“* za administratora

Izborom opcije za detaljan prikaz događaja administrator pristupa stranici koja pored osnovnih podataka o događaju sadrži opis događaja i vizuelni prikaz rasporeda sjedišta arene. Sjedišta različite vrste, odnosno različite cijene, prikazana su različitom bojom. Sjedišta koja su rezervisana označena su crvenom bojom (slika 6.3).



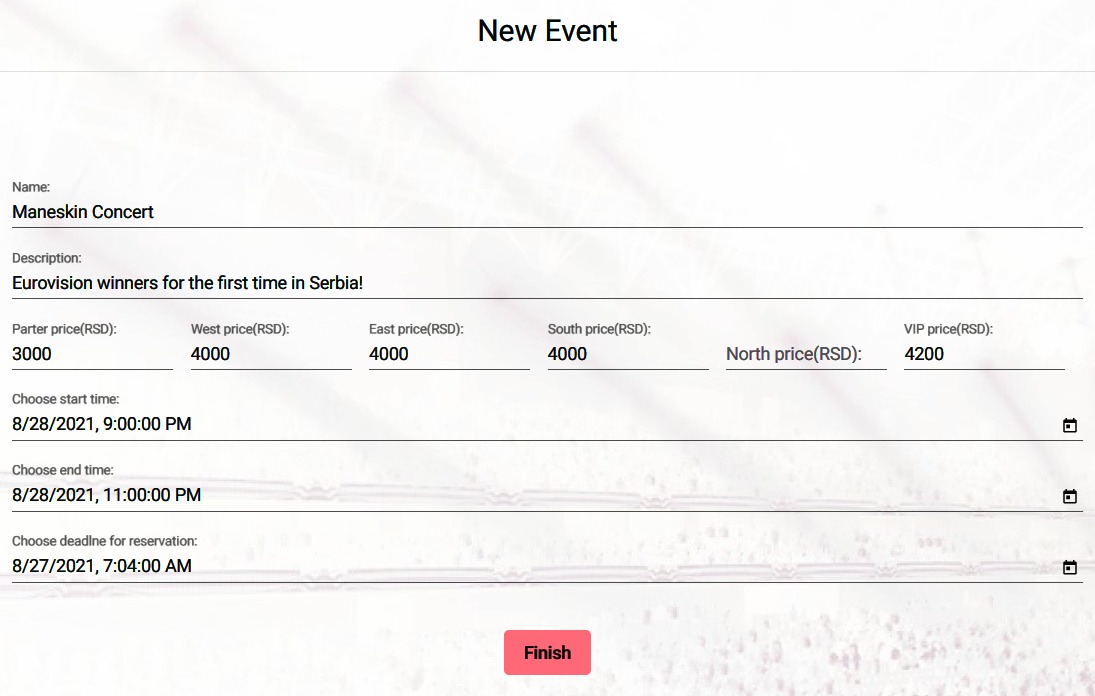
1. Detaljan prikaz događaja za administratora

Pored opcija vezanih za događaj, administrator može da dodaje novi događaj i pregleda sve rezervacije u sistemu, a te opcije se nalaze u *navigation bar*-u. Stranica za pregledanje svih rezervacija i rezervacija za događaj prikazana je na slici 6.4. Za svaku rezervaciju može da se vidi korisnik koji je rezervisao, naziv događaja, datumi početka, završetka i roka, cijena, broj sjedišta i u slučaju da je rezervacija otkazana, dodatna kolona sa tom informacijom.



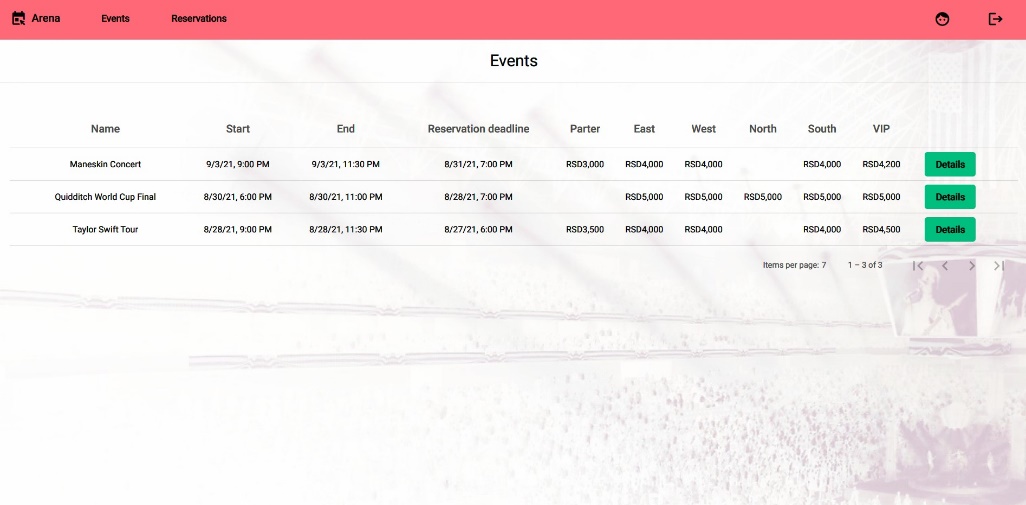
1. Prikaz stranice *„Reservations“* za administratora

Za dodavanje novog i izmjenu postojećeg događaja koristi se ista forma (slika 6.5). Administrator mora da popuni sva polja osim onih koja označavaju cijenu. Ukoliko ona ostanu nepopunjena podrazumijeva se da se ne izdaju rezervacije za tu vrstu sjedišta. Takođe, administrator mora da vodi računa da datumi i vremena budu uređeni tako da imaju smisla i da se događaji vremenski ne poklapaju. To se provjerava na serverskoj strani, i ukoliko uslov nije ispunjen administratoru se prikazuje odgovarajuće obavještenje.



1. Prikaz forme za dodavanje i izmjenu događaja

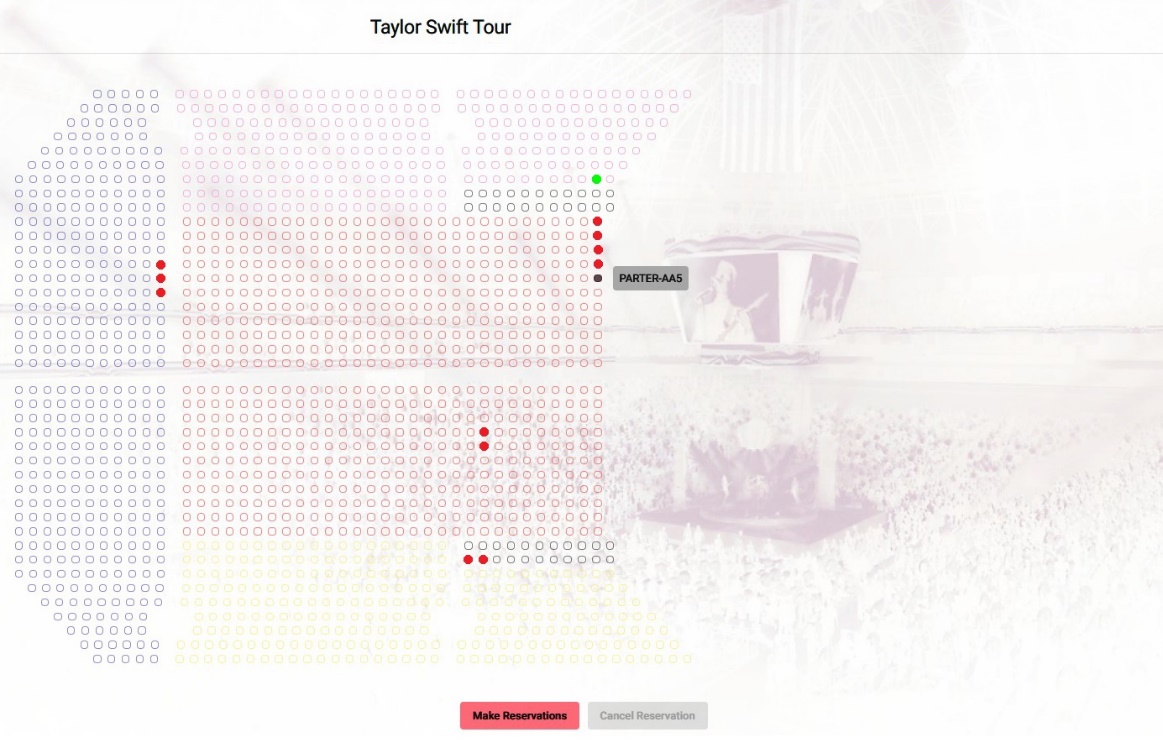
Kada se prijavi na sistem, obični korisnik takođe može da vidi stranicu sa događajima, ali njegova jedina opcija je pregled detaljnog prikaza događaja (slika 6.6). Pored toga, može i da pregleda svoje rezervacije, a ta opcija se nalazi u *navigation bar*-u.



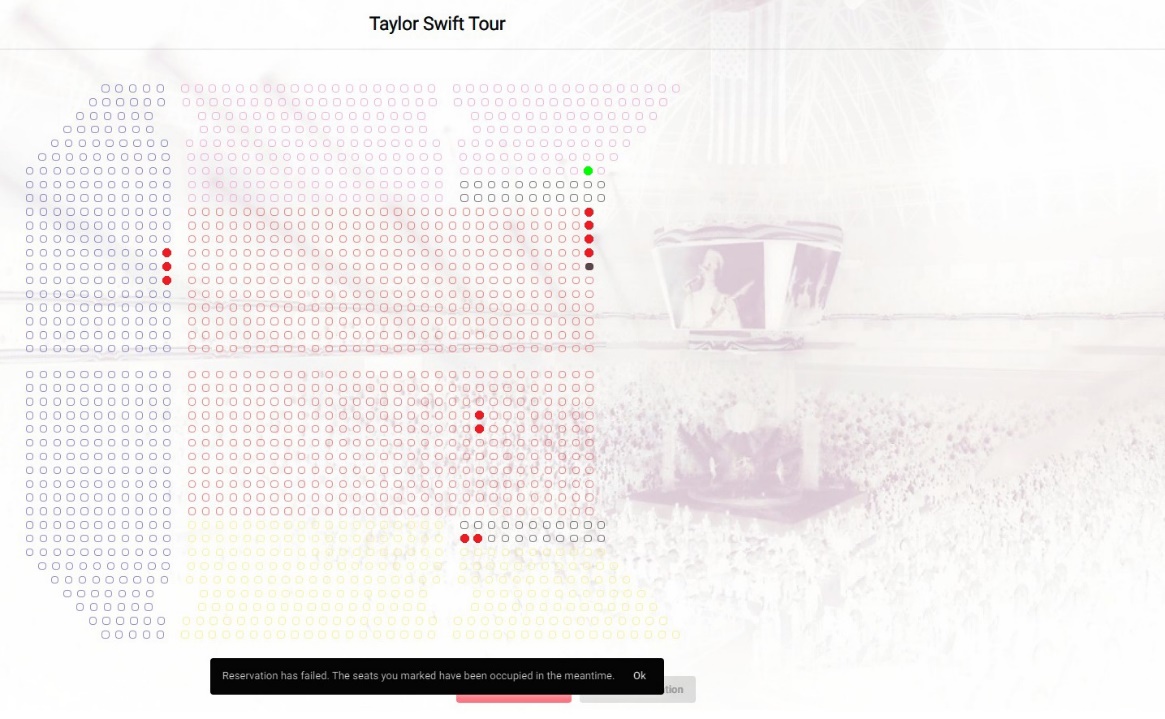
1. Prikaz početne stranice „*Events“* za korisnika

Kod detaljnog prikaza događaja korisnik može da vidi sjedišta rezervisana od strane drugih korisnika označena crvenom bojom, svoja rezervisana sjedišta označena zelenom bojom koja može da otkaže i slobodna sjedišta koja može da rezerviše. Prelaskom miša preko sjedišta korisniku se prikazuje broj tog sjedišta. Rezervacija ili otkazivanje se izvršava tako što korisnik selektuje izabrana sjedišta i klikne na odgovarajuće dugme ispod. Nemoguće je odjednom rezervisati i otkazivati, odnosno ako korisnik selektuje slobodno sjedište s namjerom da ga rezerviše, dugme za otkazivanje se onemogućava, i obrnuto (slika 6.7). Nakon uspješne akcije korisnik dobija mejl sa informacijama o rezervaciji.

Ukoliko korisnik pokuša da rezerviše sjedišta tako da ukupno ima više od 4 rezervacije za jedan događaj, ili ukoliko pokuša da rezerviše sjedišta za događaj kojem je prošao rok za rezervisanje, prikazaće mu se poruka o neuspješno obavljenoj akciji. Isto će se desiti i ako neko drugi u međuvremenu rezerviše selektovano sjedište (slika 6.8).

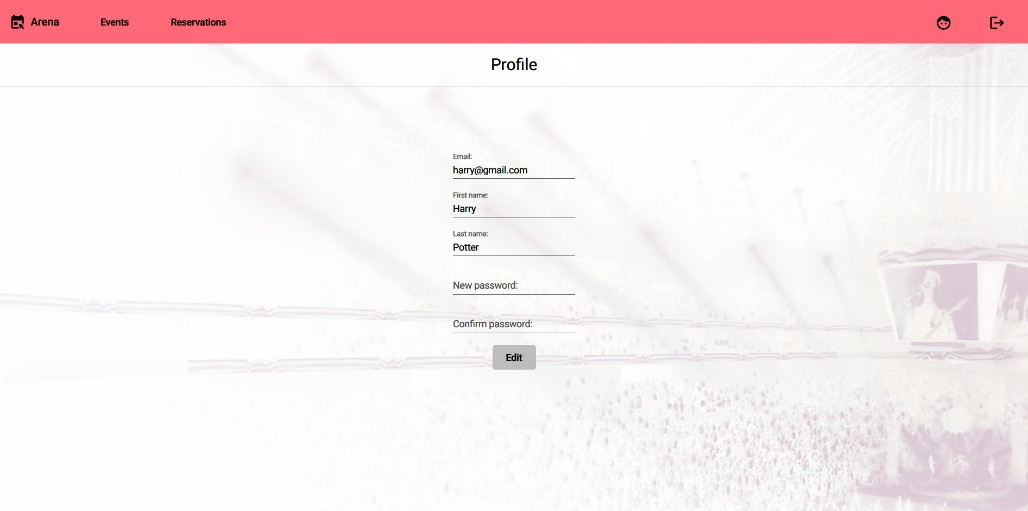


1. Prikaz rezervisanja sjedišta



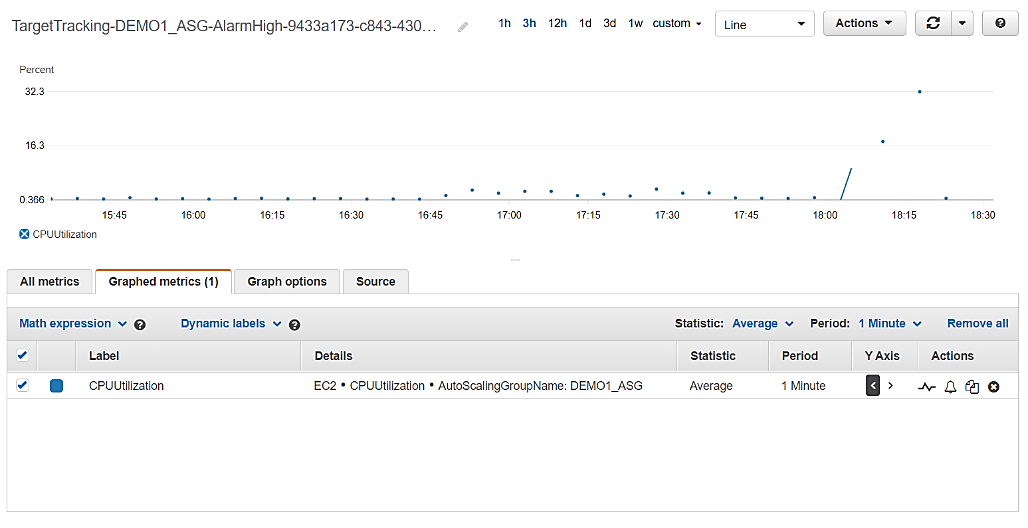
1. Prikaz neuspješnog rezervisanja sjedišta

Oba tipa korisnika na desnoj strani *navigation bar*-a imaju opciju za pregled i izmjenu svojih podataka i odjavu sa sistema. Na slici 6.9 prikazana je forma za izmjenu koja dozvoljava izmjenu imena, prezimena i lozinke.

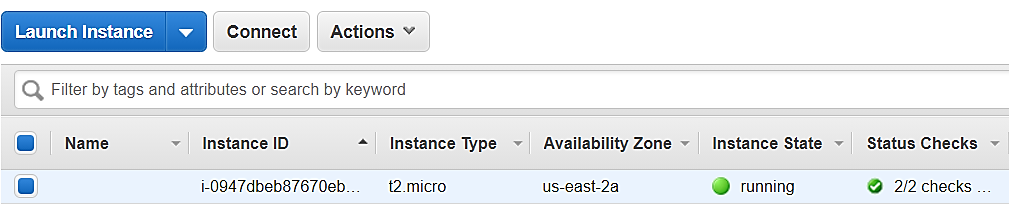


1. Prikaz starnice za izmjenu svog profila

Kao što je rečeno u poglavlju o implementaciji, definisana su dva alarma koja prate iskorišćenje procesora EC2 instanci. Jedan alarm se aktivira kada je izmjereno prosječno iskorišćenje veće od 50% tri puta zaredom (*AlarmHigh*), a drugi kada je iskorišćenje manje od 35% 15 puta zaredom (*AlarmLow*). Na slici 6.10 prikazan je grafik na kojem se vidi stanje alarma *AlarmHigh* kada vrijednost metrike ne prelazi 50 i alarm nije aktiviran. *Auto Scaling* grupa na koju se odnosi alarm u tom trenutku sadrži jednu ispravnu EC2 instancu (slika 6.11).

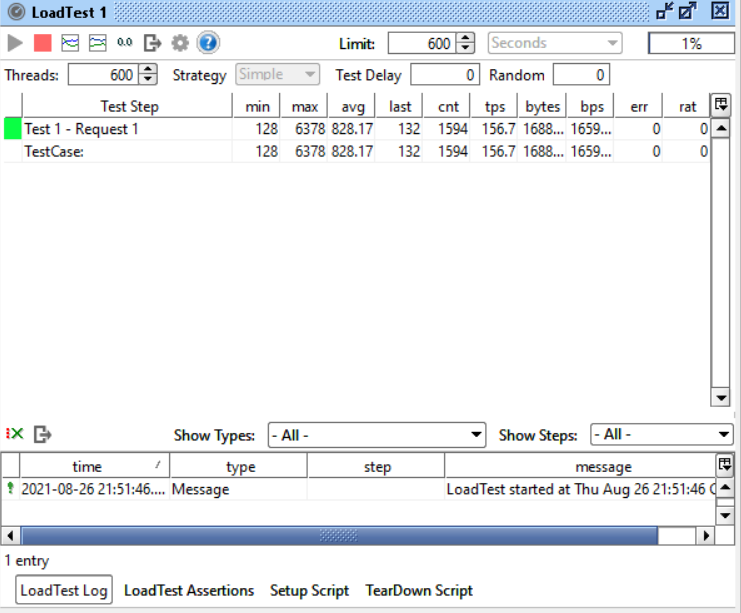


1. Prikaz *AlarmHigh* alarma u stanju OK



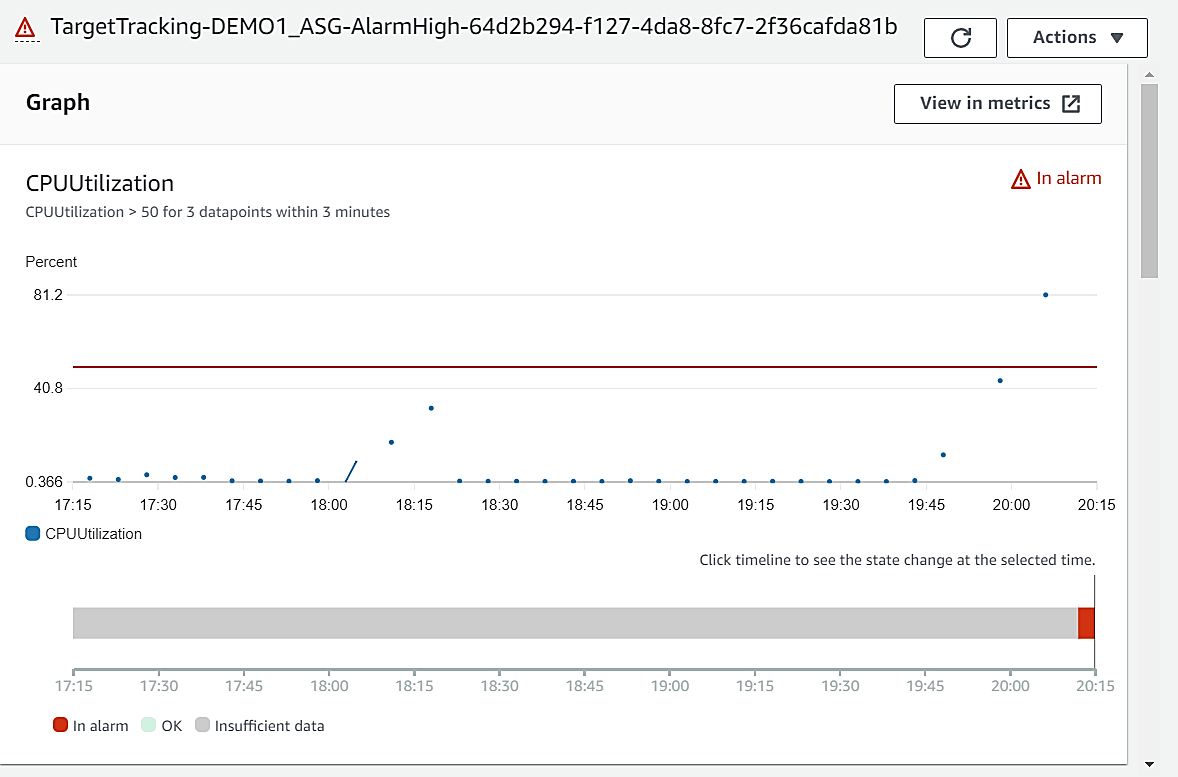
1. Prikaz EC2 instance

Kako bi se pokazala mogućnost sistema da alocira i pušta resurse po potrebi, odnosno povećava i smanjuje broj EC2 instanci, izvršen je test opterećenja (engl. *load test*) nad aplikacijom. Test je odrađen pomoću alata *SoapUI* [[42]](#_LITERATURA_1) koji omogućava slanje velikog broja zahtjeva aplikaciji odjednom, u ovom slučaju 600 (slika 6.12). To je potrebno da bi se izazvalo povećanje iskorišćenosti procesora.

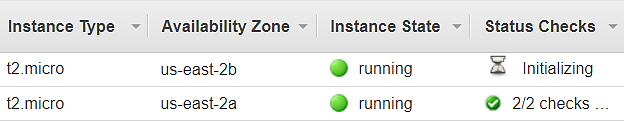


1. Prikaz *load* testa

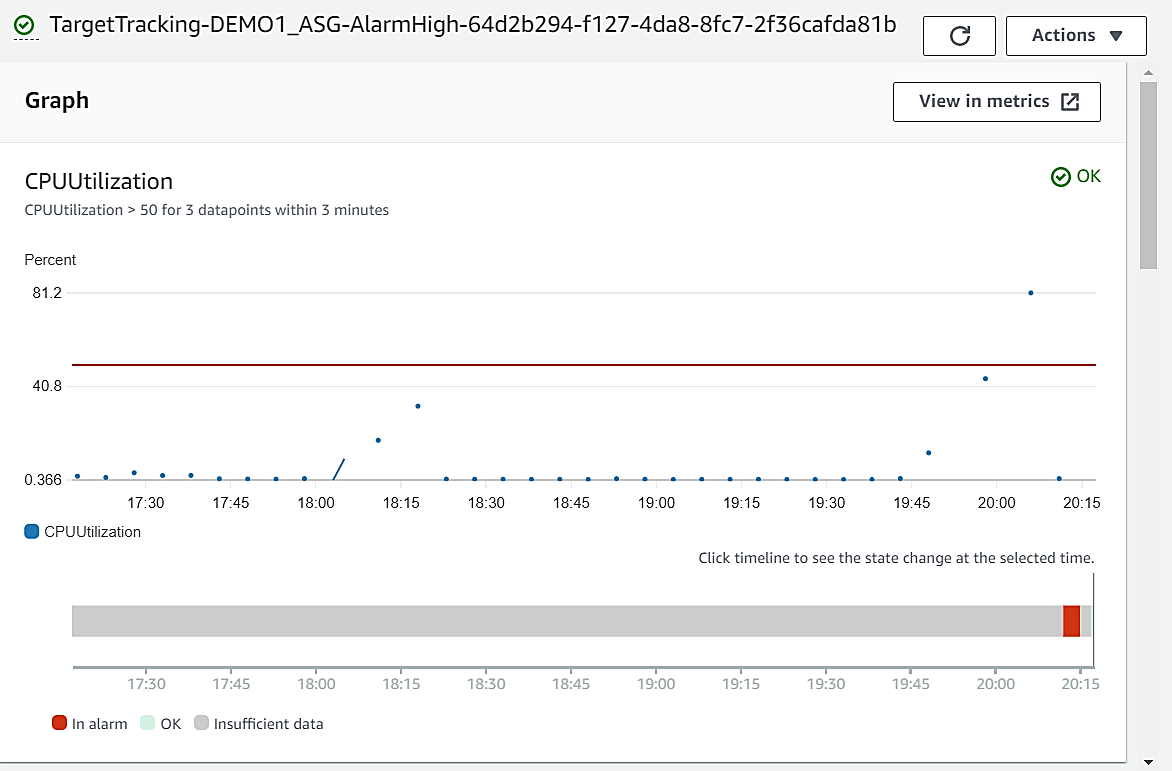
Na slici 6.13 prikazan je *AlarmHigh* nekoliko minuta nakon pokretanja testa. Na *x* osi je predstavljeno vrijeme, a na *y* osi procenat iskorišćenja procesora. Alarm je aktiviran jer je u posljednje tri minute tri puta izmjereno iskorišćenje procesora koje prelazi prag. Odgovor sistema na aktiviranje ovog alarma je pokretanje još jedne EC2 instance u sklopu *Auto Scaling* grupe (slika 6.14). Ona će omogućiti da se opterećenje rasporedi i da se smanji iskorišćenje procesora pojedinačne instance. Kada se opterećenje smanji i mjerenja više ne pokazuju da metrika prelazi prag, alarm se vraća u stanje OK (slika 6.15).



1. Prikaz grafika aktiviranog alarma *AlarmHigh*

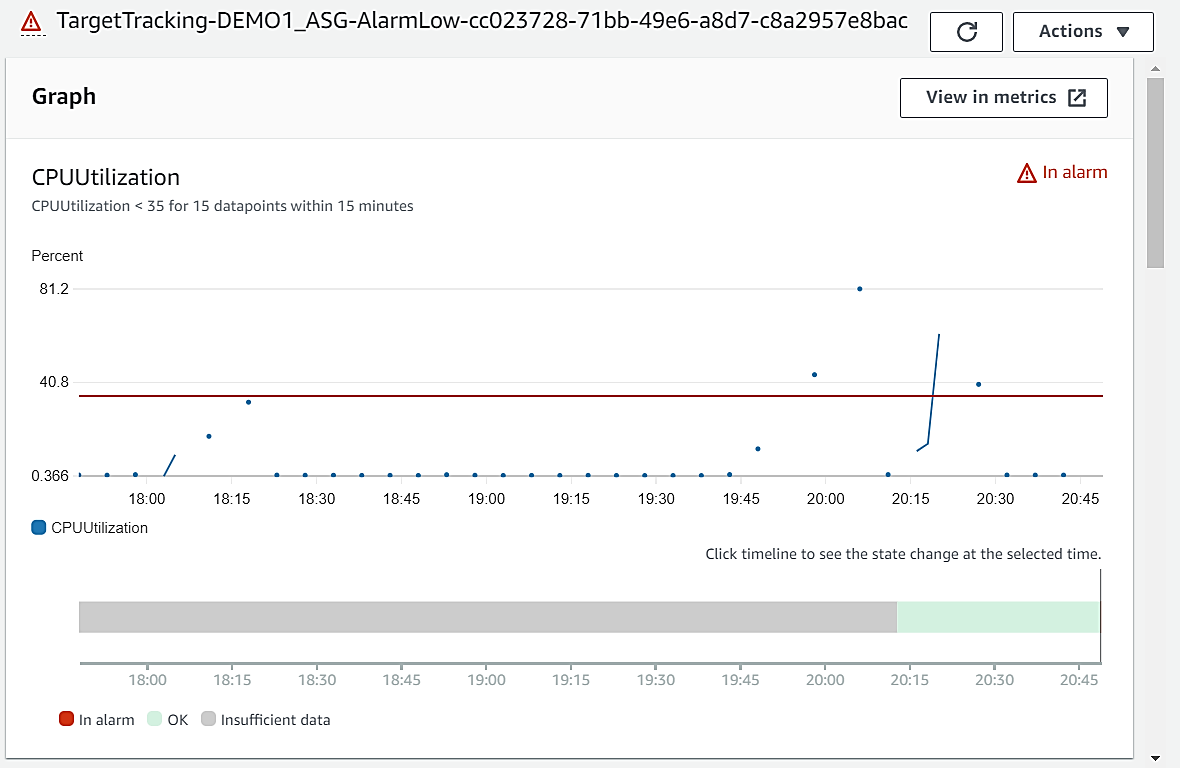


1. Prikaz odgovora sistema na aktiviranje alarma *AlarmHigh*



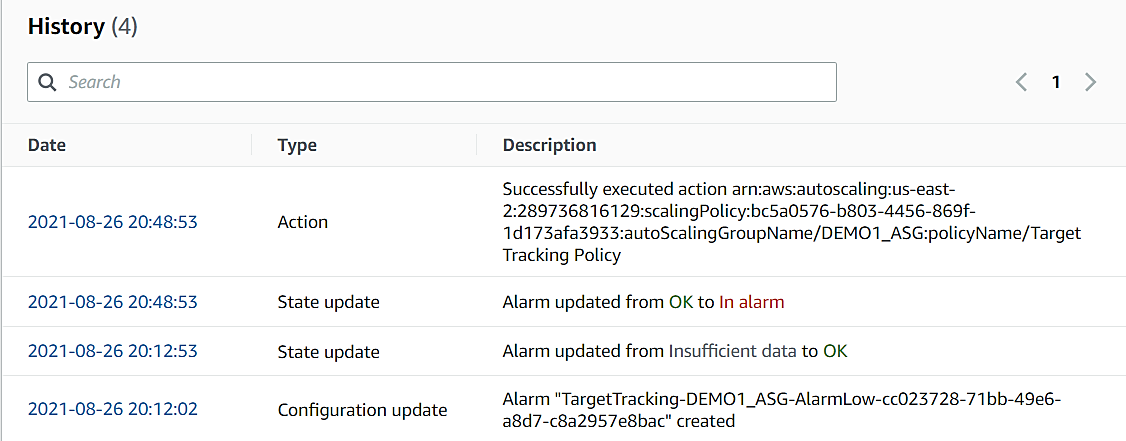
1. Prikaz grafika alarma *AlarmHigh* ponovo u stanju OK

Drugi alarm, *AlarmLow*, služi da bi se kapacitet grupe vratio na 1 kada više nije potrebno da postoje dvije instance. Na slici 6.16 prikazan je aktivirani *AlarmLow*, nakon što je izvršio 15 mjerenja pri kojima je vrijednost metrike bila ispod praga.

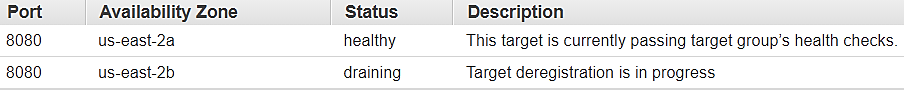


1. Prikaz grafika aktiviranog alarma *AlarmLow*

Na slici 6.17 prikazana je aktivnost alarma *AlarmLow*, gdje se može vidjeti da je nakon prelaska alarma u stanje ALARM izvršena akcija uklanjanja instance iz *Auto Scaling* grupe. Uklonjena instanca više ne prolazi provjeru ispravnosti, pa se uklanja iz *target* grupe (slika 6.18).

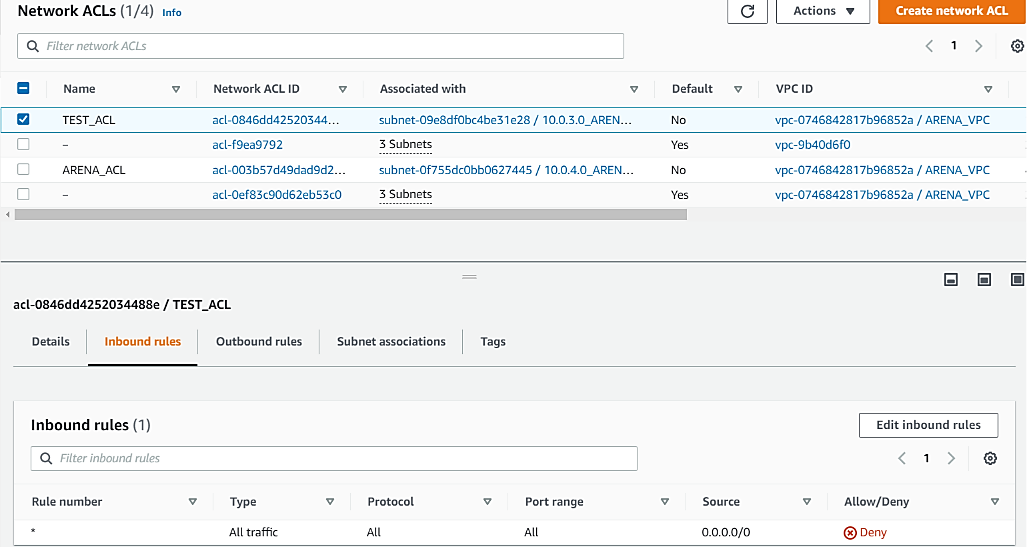


1. Prikaz aktivnosti alarma *AlarmLow*



1. Prikaz instanci *target* grupe nakon aktiviranja alarma *AlarmLow*

Da bi se pokazalo kako sistem radi u slučaju kada jedna od zona dostupnosti koje koristi postane nedostupna, npr. zbog vremenskih nepogoda, potrebno je simulirati otkaz zone dostupnosti. To je odrađeno kreiranjem nove ACL liste TEST\_ACL koja zabranjuje sav saobraćaj (slika 6.19). Ta lista se zatim pridružuje privatnoj podmreži u kojoj se pokreću instance, a koja se nalazi u zoni dostupnosti *us-east-2a*.

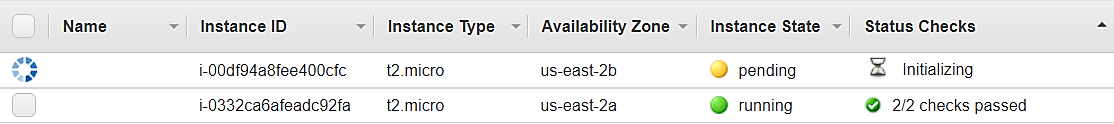


1. Prikaz ACL liste za simulaciju otkaza zone dostupnosti

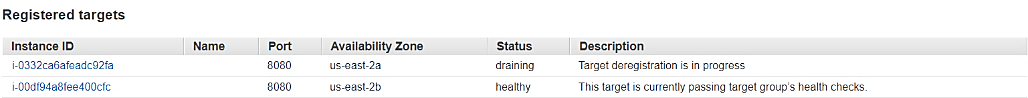
Nakon dodavanja nove ACL liste, instanca pokrenuta u zoni *us-east-2a* više ne prolazi provjeru ispravnosti (slika 6.20). Kada se to desi, sistem će pokrenuti novu instancu kako bi kapacitet ostao isti, u ovom slučaju 1 (slika 6.21), i to u drugoj zoni dostupnosti, *us-east-2b*. Na slici 6.22 prikazana je *target* grupa nakon pokretanja nove instance, koja uklanja neispravnu instancu i nastavlja da koristi ispravnu.



1. Prikaz *target* grupe u slučaju otkaza zone dostupnosti



1. Prikaz pokretanja nove instance u slučaju otkaza zone dostupnosti



1. Prikaz *target* grupe nakon pokretanja nove instance

U ovakvom slučaju kada sistem koristi dvije zone dostupnosti, on će biti nedostupan u kratkom periodu kada pokreće novu instancu. Zbog toga je poželjno koristiti tri zone dostupnosti, jer će u tom slučaju postojati još jedna ispravna instanca koja radi cijelo vrijeme.

# ZAKLJUČAK

U ovom radu opisana je izgradnja skalabilne, elastične i visoko dostupne softverske arhitekture na *cloud*-u pomoću AWS servisa. Dat je kratak opis drugih značajnih dobavljača servisa kao što su *Microsoft Azure* i *Google Cloud Platform*, a zatim i teoretski opis AWS servisa koji mogu da se koriste za izgradnju takve arhitekture. Objašnjen je rad sa mrežom na *cloud*-u pomoću VPC servisa, rad sa bazama podataka pomoću RDS servisa, skladištenje pomoću S3 servisa i rad sa virtuelnim serverima, njihovo nadgledanje i skaliranje pomoću servisa EC2, *Auto Scaling* i *CloudWatch*.

U drugom dijelu rada prikazan je model predložene arhitekture i data je specifikacija jednostavne veb aplikacije koja je postavljena na navedenu arhitekturu. Takođe, opisana je implementacija aplikacije pomoću tehnologija *Spring* i *Angular* i koraci i podešavanja koja su potrebna za postavljanje aplikacije na AWS. Na kraju su prikazane funkcionalnosti sistema i testirana je elastičnost i dostupnosti.

Arhitektura sistema bi mogla da se unapredi upotrebom servisa koji nisu besplatni, što je opisano u okviru poglavlja o modelu predložene arhitekture. Pošto korišćeni servisi ne obezbjeđuju visoku dostupnost baze podataka, mogu se koristiti RDS *Multi-AZ Deployment* ili *Amazon Aurora* koji to omogućavaju. Takođe, aplikacija može da se postavi na skalabilnu arhitekturu automatski korišćenjem servisa *Elastic Beanstalk*.

# LITERATURA

1. Michael Wittig, Andreas Wittig (2016). Amazon Web Services in Action
2. HTTP dokumentacija, <https://httpwg.org/specs/>
3. AWS dokumentacija, <https://docs.aws.amazon.com/>
4. AWS dokumentacija - *Well-Architected Framework*, <https://aws.amazon.com/architecture/well-architected>
5. AWS dokumentacija - koncepti *Well-Architected Framework-*a, <https://wa.aws.amazon.com/wellarchitected/2020-07-02T19-33-23/wat.concepts.wa-concepts.en.html>
6. AWS dokumentacija - *Free Tier*, <https://aws.amazon.com/free>
7. AWS dokumentacija - globalna infrastruktura, <https://aws.amazon.com/about-aws/global-infrastructure>
8. *Microsoft Azure* dokumentacija, <https://docs.microsoft.com/en-us/azure>
9. *Google Cloud Platform* dokumentacija, <https://cloud.google.com/docs>
10. Gartner (2021). *Magic Quadrant for Cloud Infrastructure and Platform Services*, <https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb>
11. AWS dokumentacija – servisi, <https://www.amazonaws.cn/en/products/>
12. *Spring* dokumentacija, <https://docs.spring.io/spring-framework/docs/current/reference/html/>
13. *Angular* dokumentacija, <https://angular.io/docs>
14. AWS dokumentacija – *Management Console*, <https://docs.aws.amazon.com/awsconsolehelpdocs/index.html>
15. AWS dokumentacija - EC2, <https://docs.aws.amazon.com/ec2/index.html>
16. AWS dokumentacija – VPC, <https://docs.aws.amazon.com/vpc/index.html>
17. AWS dokumentacija – *Elastic Beanstalk,* <https://docs.aws.amazon.com/elastic-beanstalk/index.html>
18. AWS dokumentacija – *Auto Scaling¸* <https://docs.aws.amazon.com/autoscaling/index.html>
19. AWS dokumentacija - S3, <https://docs.aws.amazon.com/s3/index.html>
20. AWS dokumentacija – *Glacier*, <https://docs.aws.amazon.com/glacier/index.html>
21. AWS dokumentacija – RDS, <https://docs.aws.amazon.com/rds/index.html>
22. AWS dokumentacija – DNS, <https://aws.amazon.com/route53/what-is-dns/>
23. AWS dokumentacija – *Elastic Load Balancer*, <https://docs.aws.amazon.com/elasticloadbalancing/index.html>
24. AWS dokumentacija – *CloudWatch*, <https://docs.aws.amazon.com/cloudwatch/index.html>
25. AWS dokumentacija – IAM, <https://docs.aws.amazon.com/iam/>
26. AWS dokumentacija – SNS, <https://docs.aws.amazon.com/sns/>
27. Simplilearn (2016), Amazon VPC Tutorial, <https://www.youtube.com/watch?v=fpxDGU2KdkA>
28. RDP dokumentacija, <https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol>
29. OSI model dokumentacija, <https://docs.snowme34.com/en/latest/reference/network/osi-model.html>
30. REST dokumentacija, <https://www.mulesoft.com/resources/api/what-is-rest-api-design>
31. JSON dokumentacija, <https://www.json.org/json-en.html>
32. *Java* dokumentacija – JDBC API, <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/>
33. JWT dokumentacija, <https://jwt.io/introduction>
34. *Spring* dokumentacija – *Spring Cloud*, <https://cloud.spring.io/spring-cloud-aws/spring-cloud-aws.html>
35. HTML dokumentacija, <https://devdocs.io/html/>
36. CSS dokumentacija, <https://developer.mozilla.org/en-US/docs/Web/CSS>
37. SCSS dokumentacija, <https://sass-lang.com/documentation>
38. *TypeScript* dokumentacija, <https://www.typescriptlang.org/docs/>
39. Kod aplikacije, <https://github.com/verak13/Arena>
40. *Apache Maven* dokumentacija, <https://maven.apache.org/guides/>
41. *PostgreSQL* dokumentacija – kontrola konkurentnog pristupa, <https://www.postgresql.org/docs/9.1/mvcc.html>
42. *SoapUI* dokumentacija, <https://www.soapui.org/docs/>

# BIOGRAFIJA

Milorad Radović je rođen 25.4.1998. godine u Bijelom Polju, Crna Gora. Osnovnu školu „Dušan Korać“ završio je 2013. godine kada upisuje gimnaziju „Miloje Dobrašinović“ u Bijelom Polju, a završio je 2017. godine. Iste godine upisao se na Fakultet tehničkih nauka, smjer Softversko inženjerstvo i informacione tehnologije. Položio je sve ispite predviđene planom i programom u roku, 2021. godine.

# KLJUČNA DOKUMENTACIJSKA INFORMACIJA

|  |  |
| --- | --- |
| Redni broj, **RBR**: |  |
| Identifikacioni broj, **IBR**: |  |
| Tip dokumentacije, **TD**: | Monografska publikacija |
| Tip zapisa, **TZ**: | Tekstualni štampani dokument |
| Vrsta rada, **VR**: | Diplomski-master rad |
| Autor, **AU**: | Milorad Radović |
| Mentor, **MN**: | dr Siniša Nikolić, docent, FTN Novi Sad |
| Naslov rada, **NR**: | Izgradnja veb aplikacije za oglašavanje i licitiranje usluga popravki u kući |
| Jezik publikacije, **JP**: | Srpski |
| Jezik izvoda, **JI**: | Spski / engleski |
| Zemlja publikovanja, **ZP**: | Srbija |
| Uže geografsko područje, **UGP**: | Vojvodina |
| Godina, **GO**: | 2021 |
| Izdavač, **IZ**: | Autorski reprint |
| Mesto i adresa, **MA**: | Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6 |
| Fizički opis rada, **FO**: | Br. poglavlja 7/ stranica 79/ citata 42/ tabela 0/ slika 69/ grafikona 0/ priloga 0 |
| Naučna oblast, **NO**: | Softversko inženjerstvo i informacione tehnologije |
| Naučna disciplina, **ND**: | Web programiranje |
| Predmetna odrednica /  ključne reči, **PO**: | AWS platforma, softverska arhitektura, elastičnost, visoka dotupnost |
| **UDK** |  |
| Čuva se, **ČU**: | Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad |
| Važna napomena, **VN**: |  |
| Izvod, **IZ**: | U radu su opisani osnovni AWS servisi i definisana je AWS arhiteltura koja omogućava elastičnost i visoku dostupnost. Prikazana je specifikacija i implementacija demo aplikacije i način na koji je ona postavljena na predloženu arhitekturu. Takođe, prikazan je i način korišćenja aplikacije i demonstrirana je elastičnost i visoka dostupnost sistema. |
| Datum prihvatanja teme, **DP**: |  |
| Datum odbrane, **DO**: |  |
| Članovi komisije, **KO**: |  |
| predsednik | dr Goran Sladić, vanr. prof, FTN Novi Sad |
| član | dr Željko Vuković, docent, FTN Novi Sad |
| mentor | dr Siniša Nikolić, docent, FTN Novi Sad |
| Potpis mentora | |

# KEY WORDS DOCUMENTATION

|  |  |
| --- | --- |
| Accession number, **ANO**: |  |
| Identification number, **INO**: |  |
| Document type, **DT**: | Monographic publication |
| Type of record, **TR**: | Textual material |
| Contents code, **CC**: | BSc thesis |
| Author, **AU**: | Vera Kovačević |
| Mentor, **MN**: | Siniša Nikolić, PhD |
| Title, **TI**: | Development of a scalable application using AWS services |
| Language of text, **LT**: | Serbian |
| Language of abstract, **LA**: | Serbian / English |
| Country of publication, **CP**: | Serbia |
| Locality of publication, **LP**: | Vojvodina |
| Publication year, **PY**: | 2021 |
| Publisher, **PB**: | Author’s reprint |
| Publication place, **PP**: | Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6 |
| Physical description, **PD**: | no. of chapters 7/ pages 79/ quotes 42/ tables 0/ pictures 69/ graphs 0/ appendix 0 |
| Scientific field, **SF**: | Software engineering and information technologies |
| Scientific discipline, **ND**: | Web programming |
| Subject / Keywords, **S/KW**: | AWS platform, software architecture, elasticity, high availabilty |
| **UDC** |  |
| Holding data, **HD**: | Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad |
| Note, **N**: |  |
| Abstract, **AB**: | This paper describes basic AWS services and defines production environment using various AWS services which enables elasticity and high availability. It contains specification and implementation of a demo applicaton which is deployed on presented architecture. In the end, the use of application is shown and elasticity and availability is tested. |
| Accepted by sci. board on, **ASB**: |  |
| Defended on, **DE**: |  |
| Defense board, **DB**: |  |
| president | Gorad Sladić, PhD, assoc. prof, FTN Novi Sad |
| member | Željko Vuković, PhD, assist. prof, FTN Novi Sad |
| mentor | Siniša Nikolić, PhD, assist. prof., FTN Novi Sad |
| Mentor’s signature | |