|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ** |  |

Лука Матић

**Аутентификација и ауторизација у оквиру серверлес архитектуре**

Дипломски рад

- Основне академске студије -

Нови Сад, 2022.

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ(BACHELOR) РАДА** | Лист: |
| 1/1 |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | **Основне академске студије** |
| --- | --- |
| Студијски програм: | **Рачунарство и аутоматика** |
| Руководилац студијског програма: | **проф. др. Милан Рапаић** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Лука Матић** | Број индекса: | **RA 16/2018** |
| Област: | **Електротехничко и рачунарско инжењерство** | | |
| Ментор: | **проф. др. Горан Сладић** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ РАД, САСЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:**

|  |
| --- |
| **Аутентификација и ауторизација у оквиру серверлес архитектуре** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| 1. Анализирати стање у области.  2. Израдити спецификацију захтева софтверског решења.  3. Израдити спецификацију дизајна софтверског решења.  4. Имплементирати софтверско решење према израђеној спецификацији.  5. Тестирати имплементирано софтверско решење.  6. Документовати (1), (2), (3), (4) и (5). |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
|  |  |

|  |
| --- |
| Примерак за:- Студента;  - Ментора |

# КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

|  |  |
| --- | --- |
| Редни број, **РБР**: |  |
| Идентификациони број, **ИБР**: |  |
| Тип документације, **ТД**: | монографска публикација |
| Тип записа, **ТЗ**: | текстуални штампани документ |
| Врста рада, **ВР**: | дипломски рад |
| Аутор, **АУ**: | Лука Матић |
| Ментор, **МН**: | др Горан Сладић, редовни професор |
| Наслов рада, **НР**: | Аутентификација и ауторизација у оквиру серверлес архитектуре |
| Језик публикације, **ЈП**: | српски |
| Језик извода, **ЈИ**: | српски / енглески |
| Земља публиковања, **ЗП**: | Србија |
| Уже географско подручје, **УГП**: | Војводина |
| Година, **ГО**: | 2022 |
| Издавач, **ИЗ**: | ауторски репринт |
| Место и адреса, **МА**: | Нови Сад, Факултет техничких наука, Трг Доситеја Обрадовића 6 |
| Физички опис рада, **ФО**: | бр. поглавља / страница / цитата / табела / слика / графикона / прилога |
| Научна област, **НО**: | Рачунарство и аутоматика |
| Научна дисциплина, **НД**: | Рачунарске науке |
| Предметна одредница /  кључне речи, **ПО**: | Аутентификација, ауторизација, серверлес |
| **УДК** |  |
| Чува се, **ЧУ**: | Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад |
| Важна напомена, **ВН**: |  |
| Извод, **ИЗ**: | апстракт – један пасус који добро описује суштину рада – проблем, м |
| Датум прихватања теме, **ДП**: |  |
| Датум одбране, **ДО**: |  |
| Чланови комисије, **КО**: |  |
| председник |  |
| члан |  |
| ментор |  |
| Потпис ментора | |

# KEY WORDS DOCUMENTATION

|  |  |
| --- | --- |
| Accession number, **ANO**: |  |
| Identification number, **INO**: |  |
| Document type, **DT**: | monographic publication |
| Type of record, **TR**: | textual material |
| Contents code, **CC**: | bachelor thesis |
| Author, **AU**: | Luka Matič |
| Mentor, **MN**: | Goran Sladić, full professor, PhD |
| Title, **TI**: | Authentication and authorization in serverless architecture |
| Language of text, **LT**: | Serbian |
| Language of abstract, **LA**: | Serbian / English |
| Country of publication, **CP**: | Serbia |
| Locality of publication, **LP**: | Vojvodina |
| Publication year, **PY**: | 2022 |
| Publisher, **PB**: | author’s reprint |
| Publication place, **PP**: | Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6 |
| Physical description, **PD**: | br. poglavlja / stranica / citata / tabela / slika / grafikona / priloga |
| Scientific field, **SF**: | Electrical Engineering |
| Scientific discipline, **SD**: | Computer Science |
| Subject / Keywords, **S/KW**: | Authentication, authorization, serverless |
| **UDC** |  |
| Holding data, **HD**: | Library of the Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad |
| Note, **N**: |  |
| Abstract, **AB**: | Prevod apstrakta na engleski |
| Accepted by sci. Board on, **ASB**: |  |
| Defended on, **DE**: |  |
| Defense board, **DB**: |  |
| president |  |
| member |  |
| mentor |  |
| Mentor's signature | |

**Садржај**

[КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА 5](#_Toc98832020)

[KEY WORDS DOCUMENTATION 6](#_Toc98832021)

[1. UVOD 9](#_Toc98832022)

[2. PREGLED SLIČNIH SISTEMA 11](#_Toc98832023)

[3. KORIŠĆENE SOFTVERSKE TEHNOLOGIJE 13](#_Toc98832024)

[3.1 Spring 13](#_Toc98832025)

[3.2 Opis tehnologije na koji se vaš rad oslanja 13](#_Toc98832026)

[4. SPECIFIKACIJA 15](#_Toc98832027)

[4.1 Specifikacija zahteva 15](#_Toc98832028)

[4.1.1 Funkcionalni zahtevi 15](#_Toc98832029)

[4.1.2 Nefunkcionalni zahtevi 15](#_Toc98832030)

[4.2 Specifikacija sistema 15](#_Toc98832031)

[4.2.1 Model podataka 16](#_Toc98832032)

[4.2.2 Arhitektura sistema 16](#_Toc98832033)

[5. IMPLEMENTACIJA 17](#_Toc98832034)

[6. DEMONSTRACIJA 19](#_Toc98832035)

[7. ZAKLJUČAK 21](#_Toc98832036)

[8. LITERATURA 23](#_Toc98832037)

[9. BIOGRAFIJA 25](#_Toc98832038)

# Увод

Да би софтверски инжењери били ефикаснији у креирању производа, потребно је да се што више фокусирају на сам развој софтвера и избегну превелику потрошњу времена и ресурса на изградњу и одржавање инфраструктуре. Желимо да бринемо о томе како функционише код у оквиру наше апликације, а не о томе да ли треба да зашрафимо нову плочицу радне меморије унутар сервера.

Ту нам помаже рачунарство у облаку, које омогућава доступност рачунарских ресурса на основу корисничких захтева (енгл. *on-demand availability*). Компаније које пружају услуге рачунарства у облаку називају се клауд провајдери (енгл. *Cloud providers*). Постоје три основна типа рачунарства у облаку:

* Софтвер као услуга (енгл. *Software as a Service - SaaS*)
* Платформа као услуга (енгл*. Platform as a Service - PaaS*)
* Инфраструктура као услуга (енгл. *Infrastructure as a Service - IaaS*)

Иницијално, инфраструктура као услуга је била реализована тако што бисмо путем мреже подешавали наш удаљени сервер. Ово јесте много боље од тога да зашрафљујемо сервер у нашем подруму, али ипак морамо донекле да водимо рачуна о том удаљеном серверу (нпр. да подешавамо оперативни систем на њему). Временом се појавила идеја о серверлес архитектури, која подразумева да бринемо искључиво о коду који ће се извршавати у одабраном окружењу, док ће све остало бити конфигурисано од стране провајдера.

Две ствари које су готово неизбежне у оквиру сваког информационог система, па и оног изграђеног кроз серверлес архитектуру, јесу аутентификација и ауторизација. Аутентификацијом утврђујемо који ентитет приступа систему, док ауторизацијом утврђујемо којим деловима система ентитет може да приступи.

У оквиру овог рада посматраћемо имплементацију аутентификације и ауторизације корисника унутар апликације Адверто. Адверто представља апликацију за постављање и читање огласа, коју ћемо изградити у серверлес архитектури, коришћењем сервиса које нам нуди један од водећих клауд провајдера данашњице. У питању је компанија Амазон, односно њен огранак који се зове *AWS (Amazon Web Services). AWS* је клауд провајдер који нуди преко 200 различитих сервиса, као што су *EC2, S3, RDS, API Gateway, Lambda, Cognito*, итд.

# 

# Коришћене софтверске технологије

У овом поглављу позабавићемо се технологијама које су коришћене за израду Адверто апликације. Прво ћемо се осврнути на алат за подизање инфраструктуре кроз код, а потом ћемо описати сервисе које смо користили у оквиру инфраструктуре заједно са библиотекама помоћу којих смо вршили интеракцију са сервисима у оквиру апликације.

## Алати за инфраструктуру кроз код

Инфраструктура коју посматрамо састоји се из више различитих сервиса на *AWS*-у. Могуће је креирати сервисе на 3 начина:

* Коришћењем AWS-ове веб апликације познате као конзола (*AWS Console)*
* Коришћењем интерфејса командне линије (*AWS CLI – AWS Command Line Interface*)
* Коришћењем неког од алата за подизање инфраструктуре кроз код (*Infrastructure as Code – IaC*)

Први начин је погодан за упознавање са сервисима, али у продукцијском окружењу није препоручљив јер морамо некако памтити шта смо све и којим редом искликтали у конзоли, док је други начин превише напоран за инжењере. Најбољи од претходно наведених начина јесте трећи начин, који нам омогућава да уредно кроз код бележимо све кораке које смо направили да би креирали и подесили одређени сервис. Основни алат, а уједно и сервис за подизање инфраструктуре кроз код, у виду *JSON* или *YAML* синтаксе, јесте *Cloud Formation*.



Слика 2.1 – Cloud Formation

Временом се испоставило да та синтакса није претерано погодна за програмере, па је AWS *креирао AWS CDK (Cloud Development Kit)* који представља радни оквир за подизање инфраструктуре кроз код. Он подржава више програмских језика међу којима су *Javascript*, *Typescript*, *Python*, *Java*, *C#* и *Go*, а најчешће се користе *Typescript* и *Python*. Код написан у оквиру *AWS CDK* се преводи у *Cloud Formation* и тако се испоручује на *AWS*.

Постоје различити радни оквири који се даље надограђују на *Cloud Formation* или *CDK*, а један од њих јесте и *SST (Serverless Stack Toolkit)*. *SST*, који ћемо користити за израду наше Адверто апликације, додатно олакшава коришћење *CDK* тако што додаје могућност развоја ламбда функција уживо (енгл. *Live Lambda Development*). Ова функционалност нам омогућава да током самог развоја уживо тестирамо код у окружењу једнаком оном које ћемо имати у продукцијској верзији.



Слика 2.2 – AWS CDK и SST

## Коришћени сервиси и библиотеке

Сада ћемо погледати који су то све сервиси који се појављују у оквиру апликације Адверто и како они функционишу у теорији.

### Фронтенд (*Cloudfront, Amplify, AWS SDK)*

У оквиру фронтенда имамо веб апликацију изграђену помоћу *React* библиотеке, при чему је за стилизовање коришћен *CSS* радни оквир по имену *Tailwind CSS*, а хостинг је одрађен помоћу *AWS* сервиса Cloudfront који обезбеђује дистрибуцију статичког и динамичког веб садржаја.

*AWS SDK* је библиотека која служи за интеракцију са *AWS* сервисима, док је *AWS Amplify* библиотека која је коришћена специфично за интеракцију са *Cognito* и *API Gateway* сервисима у оквиру фронтенда јер је за те сервисе једноставнија за коришћење у поређењу са *AWS SDK*.



Слика 2.3 – React и Tailwind CSS



Слика 2.4 – Cloudfront

### Бекенд (*API Gateway, Lambda)*

Бекенд наше апликације реализован је помоћу *AWS Lambda* функција које су увезане у оквиру *AWS API Gateway-a*.

*AWS Lambda* представља сервис који нуди рачунарске ресурсе специфичне за извршавање функције којој је тај сервис намењен. Када користимо *Lambda* сервис, не инстанцирамо никакав сервер и не размишљамо о администрацији, већ се бринемо само за код који се извршава, а све остало одрађује *AWS*. *Lambda* се аутоматски скалира, а наплаћује се на основу времена извршавања функције. Док се *Lambda* не извршава, она нас не кошта ништа. Први пут када се *Lambda* позове, одговор је нешто спорији због хладног подизања (енгл. *cold start*) који подразумева буђење наше функције у оквиру *AWS*-a, јер се *Lambda* која се не користи одређено време аутоматски успава, па ново резервисање ресурса захтева одређено време.

*AWS* *API* *Gateway* сервис задужен је за креирање, објављивање, одржавање, надзор и обезбеђивање *REST*, *HTTP* и *WebSocket* *API*-ја на било којој скали. У оквиру Адверто апликације, користи се као *REST* *API* *Gateway* који рутира захтеве ка одговарајућим *Lambda* функцијама.

Бекенд, попут фронтенда за интеракцију са другим сервисима користи библиотеку *AWS SDK*.



Слика 2.5 – API Gateway и Lambda

### База података (*DynamoDB*)

База података коју ћемо користити јесте *DynamoDB*. *Dynamo* је нерелациона база креирана од стране људи из *АWS*-а која подржава кључ-вредност и документ базиране структуре. Специјално је оптимизована за претрагу, при чему је могуће вршити претрагу по кључу или по глобалном индексу (*Query*), као и претрагау скенирањем читаве базе (*Scan*). У оквиру претраге могуће је дефинисати и филтере. *DynamoDB* се наплаћује по читању при чему цена зависи од комплексности претраге. Скенирање са филтерима се не препоручује јер је прилично скупо. Примарни кључ се може састојати од једног или два дела. Први (обавезни) део јесте партициони кључ (*Partition Key/Hash Key*), а други (опциони) је сортни кључ (*Sort Key/Range Key*). Такође, глобални индекси (*Global Indexes*) се налик кључу могу састојати из једног или два дела.



Слика 2.6 – DynamoDB

### Складиштење датотека (*S*3)

*Amazon Simple Storage Service*, познатији као *S3*, јесте сервис за складиштење објеката који нуди велику скалабилност, доступност, безбедност и перформансе. Објекат (енгл. *object*) обухвата датотеку и све метаподатке који ту датотеку описују. Објекти се чувају у оквиру складишта које са назива корпa (енгл. *bucket*) при чему сваки објекат има кључ (енгл. *key*) који га јединствено идентификује у оквиру корпе. Кључ обухвата назив, путању на којој се он налази у корпи и његову верзију (уколико је верзионисање укључено унутар корпе). На пример, ако је кључ *photos/image234.jpeg* онда тај објекат има назив *image234.jpg*, а налази у директоријуму *photos* који се налази у корену. Поред самог складиштења, овај сервис нуди и разне друге функционалности које олакшавају рад са фајловима и доприносе перформансама система уколико се искористе на правилан начин. Једна од тих фунцкионалности је *Presigned Post Data* функционалност о којој ћемо више говорити када дођемо до имплементације.



Слика 2.7 – Simple Storage Service

### Аутентификација и ауторизација (*Cognito, Security Token Service, Identity and Access Management*)

Пре него што ускочимо у сам процес аутентификације и ауторизације, потребно је да разумемо основне принципе на основу којих функционише контрола права приступа ресурсима сервиса унутар AWS-а.

Пермисије за акције над ресурсима у оквиру *AWS*-а се реализују кроз полисе (*Policies*) које се представљају помоћу *JSON* синтаксе. Свака полиса садржи идентификатор (*Id*), верзију (*Version*) и листу исказа (*Statement*), при чему сваки исказ чине следећи елементи:

* *Principal* – листа ентитета на који се односи исказ
* *NotPrincipal* – листа ентитета на које не треба да делује исказ
* *Action* – листа акција које на које се односи исказ
* *NotAction* – листа акција на које не треба да делује исказ
* *Effect* – ефекат са вредностима Allow (дозвола) или Deny (забрана)
* *Resource* – листа ресурса на које се односи исказ
* *NotResource* – листа ресурса на које не треба да делује исказ
* *Sid* – идентификатор исказа
* *Conditions* – листа услова

Веома је важно напоменути то да унутар полиса, забране увек имају предност у односу на дозволе.

Постоје различите врсте полиса, но за потребе наше апликације нама су потребне само две врсте:

* *Identity-based policies* – полисе које се каче на кориснике, групе или роле (у оквиру Адверто апликације се каче за роле)
* *Resource-based policies* – полисе које се каче на ресурсе (нпр. за акције над одређеним директоријумом у оквиру S3 Bucket-a)

У оквиру *AWS*-а, сервиси примају захтеве послате путем *HTTP* протокола при чему се пре слања, због аутентификације и ауторизације, захтеви потписују приступним кључем помоћу *Signature Version 4 (SigV4)* алгоритма. Приступни кључ чине идентификатор приступног кључа (*Access Key Id*) и тајна приступног кључа (*Secret Access Key*). При коришћењу темпорарних креденцијала о којима ћемо говорити нешто касније, потребно је приложити и токен сесије (*Session Token*) који служи за верификацију темпорарних креденцијала. Када захтев стигне до сервиса, сервис контактира *IAM* *(Identity And Access Management)* сервис који је задужен за проверу креденцијала и права приступа ресурсима на основу полиса. Уколико је приступ омогућен, сервис даље обрађује захтев, а у супротном га одбија.

*Cognito* је сервис који служи за аутентификацију и ауторизацију корисника. Састоји се из две компоненте:

* *User Pool* – компонента задужена за аутентификацију
* *Identity Pool* – компонента задужена за ауторизацију

У оквиру *User Pool*-а могуће је подесити разне функционалности везане за ток регистрације и пријављивања као што су типови креденцијала (корисничко име, мејл, број телефона), критеријум лозинке, обавезни атрибути корисника, двофакторска аутентификација (путем *SMS* поруке или *Time-based One-time Password*-a), *SSO* помоћу провајдера као што су *Google*, *Facebook*, *Amazon* и *Apple* или протокола као што су *OpenID* и *SAML*. Такође, унутар *User Pool*-a, потребно је подесити и *User Pool Client* апликацију (или више њих) којима се може доделити приступ одређеним информацијама везаним за кориснике, а са којима ће комуницирати клијентске апликације ван *Cognito* сервиса.

Једна од значајних функционалности *User Pool*-а су окидачи. Окидачи представљају функције које се позивају као одговор на одређени догађај. Постоји 12 различитих окидача, али нас, за потребе наше апликације, занима само један који се зове *Post confirmation Lambda trigger*, а који се позива након што је корисник потврдио свој налог помоћу верификационог кода који му је стигао на имејл или број телефона у виду *SMS* поруке.

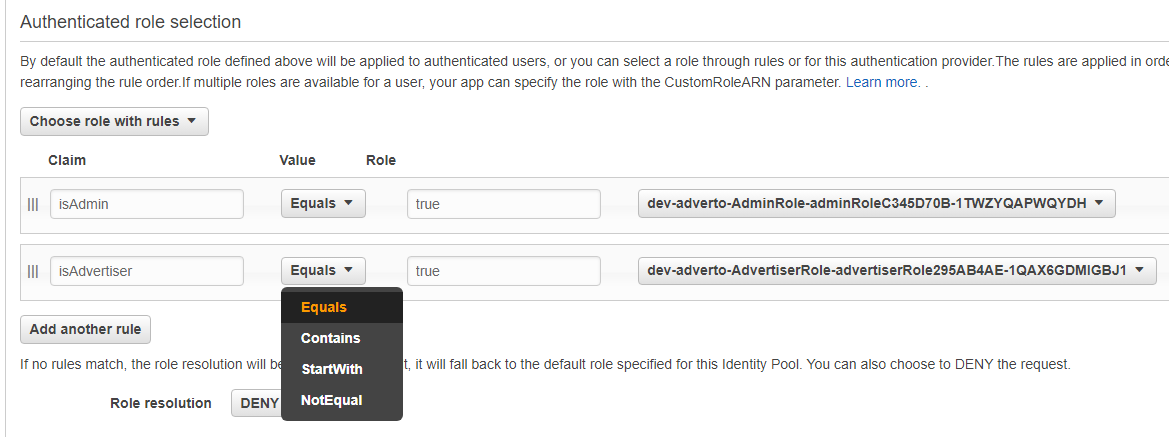
При успешној пријави, корисник као одговор добија три токена:

* *Access Token* – токен који служи за аутентификацију корисника у оквиру *API*–ja *User Pool*-а (нпр. када корисник жели да измени своје податке)
* *Identity Token* – токен који садржи детаљније информације о кориснику и служи за аутентификацију и ауторизацију у оквиру целог *AWS*-а, односно помоћу њега се добијају темпорарни креденцијали помоћу којих се приступа другим сервисима
* *Refresh Token* – токен који служи за добијање нових *Access* и *Identity* токена

*Identity Pool* решава проблем ауторизације тако што валидира *Identity* токен и потом, у случају валидног токена, комуницира са *STS* сервисом. *STS (Security Token Service)* генерише темпорарне *AWS* креденцијале који се састоје од *Access Key Id*-а, *Secret Access Key*-а и *Session Token*-а.

Додељивање рола се подешава у оквиру *Identity Pool*-а и могуће га је реализовати на 3 начина:

* Додељивањем подразумеване аутентификоване роле
* *Choose role from token* – на основу групе којој корисник припада и првенства (енгл. precedence) које та група има, при чему првенство представља број из скупа N+0 и уколико се корисник налази у више група, одабрана рола је она која има најниже првенство
* *Choose role with rules* (слика 2.8) – На основу правила која посматрају вредности атрибута које корисник поседује



Слика 2.8 – Подешавање одабира роле помоћу методе *Choose role with rules*

Ако се одаберу други или трећи начин, могуће је да корисник не задовољава критерујме ниједне роле (не припада групи, припада више група које имају исто првенство или нема одговарајуће атрибуте да би задовољио неку ролу), па је потребно одлучити да ли ће се у том случају кориснику доделити подразумевана улога (уколико је дефинисана).

Наша апликација ће користити други начин, с обзиром на то да кроз конзолу можемо мењати групе корисника док атрибуте морамо мењати кроз *AWS CLI* или некакву админску апликацију која користи *AWS SDK*.

Унутар *Identity Pool*-а се такође може креирати и посебна рола за неаутентификоване кориснике, која ће им омогућити да приступе деловима система који су делимично јавни, односно подлежу аутентификацији ради сложеније обраде захтева за аутентификоване кориснике, али аутентификованост није услов за приступ ресурсу.

Дијаграм тока аутентификације и ауторизације приказан је на слици 2.9.



Слика 2.9 – Дијаграм тока аутентификације и ауторизације

# СПЕЦИФИКАЦИЈА

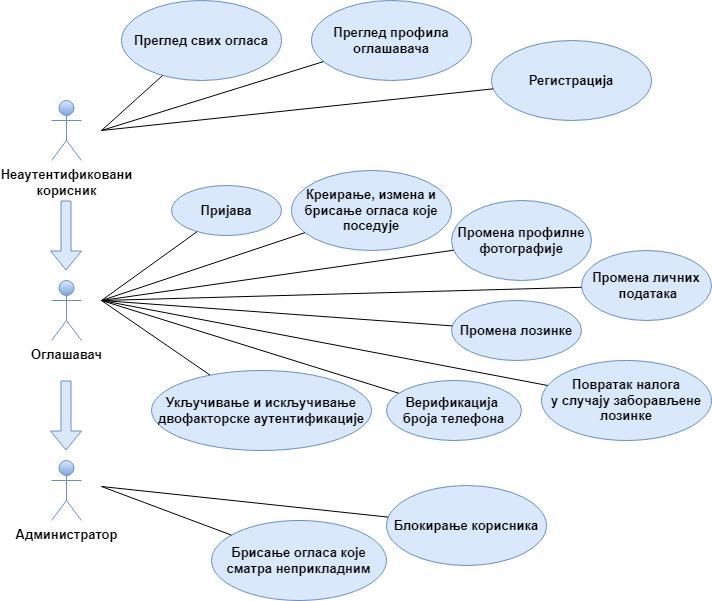
## Спецификација захтева

У оквиру овог поглавља размотрићемо опис функционалних и нефункционалних захтева које је потребно да систем омогући.

### Функционални захтеви

Функционалне захтеве ћемо представити помоћу дијаграма случајева коришћења, а потом ћемо дати дијаграм продискутовати.

.



Слика 3.1 – Дијаграм случајева коришћења

Апликација разликује 3 роле. Неаутентификовани корсиник, оглашавач и администратор.

У оквиру почетне странице, сви корисници могу да виде све огласе, сортиране по датуму креирања тако да се при врху налазе најновији огласи. У оквиру сваког огласа се приказују информације о оглашавачу (профилна слика, имејл адреса, име и презиме), као и информације о самом огласу (наслов, опис, цена и евентуална слика ако постоји). Кликом на име и презиме или на профилну слику оглашавача, корисник бива преусмерен на профил оглашавача.

У оквиру профила оглашавача приказују се његове информације (профилна слика, имејл адреса, име и презиме), као и сви његови огласи, сортирани по датуму креирања тако да се при врху налазе најновији огласи. У оквиру огласа приказани су сви подаци као и на почетној страници.

Корисници се региструју попуњавањем регистрационе форме или помоћу *Google* налога (*Single Sign-On*). Када се региструју путем форме, корисници уносе имејл, име, презиме и лозинку, након чега на имејл добијају верификациони код.

Пријава на систем подразумева уношење имејла и лозинке, а потом и се, у случају укључене двофакторске аутентификације уноси и сигурносни код који корисник добија путем SMS поруке. У оквиру странице за пријављивање, корисници имају опцију за повратак налога у случају заборављене лозинке. Повратак налога фунцкионише тако што кориснику на унету имејл адресу стиже сигурносни код који он потом шаље заједно са новом лозинком како би верификовао промену.

Оглашавач има могућности да креира своје огласе уносећи наслов, опис, цену и опциону слику. Поред тога, доступно му је и да мења и брише своје огласе.

У оквиру странице за подешавања профила, оглашавчи могу да мењају профилну слику и личне податке (име, презиме и број телефона). Такође, имају опцију за верификовање броја телефона како би могли да га користе за двофакторску аутентификацију путем *SMS* поруке. Број телефона се верификује тако што корисник уноси сигурносни код који му је на тај број послат путем *SMS* поруке. Двофакторска аутентификација укључује се и искључује у оквиру странице за подешавања профила. У оквиру исте странице, корисник има и опцију за промену лозинке, при чему на мејл добија сигурносни код који потом шаље заједно са новом лозинком како би верификовао промену.

Путем конзоле оглашавачи могу да постану администратори тако што се додају у администраторску групу и тиме добијају додатне могућности да блокирају и одблокирају кориснике, као и да бришу огласе које сматрају неприкладним.

### Нефункционални захтеви

Потребно је да систем испуњава следеће нефункционалне захтеве:

* Фронтенд мора да подржава респонзиван дизајн
* Омогућити једноставну навигацију у оквиру фронтенда
* Рад са сликама не сме да оптерећује бекенд:
  + при промени профилне слике директно комуницирати са *Cognito* и *S3* сервисима (са *Cognito* сервисом за измену атрибута корисника везаног за локацију слике, а са *S3* за чување слике)
  + при креирању или измени огласа који имају слику, потребно је прво сачувати информације о огласу (заједно са предефинисаном путањом до слике), а чување слике на *S3* одрадити у оквиру фронтенда помоћу *Pre-Signed Post Data* функционалности коју нуди *S3*
* Лозинка корисника мора да садржи:
  + најмање 8 карактера
  + бар једно велико слово
  + бар једно мало слово
  + бар један број
  + бар један специјални карактер

## Спецификација система

У оквиру овог поглавља ћемо прво кроз класни дијаграм представити модел система, а потом ћемо кроз други дијаграм представити и архитектуру система описану кроз повезаност сервиса.

### Модел података

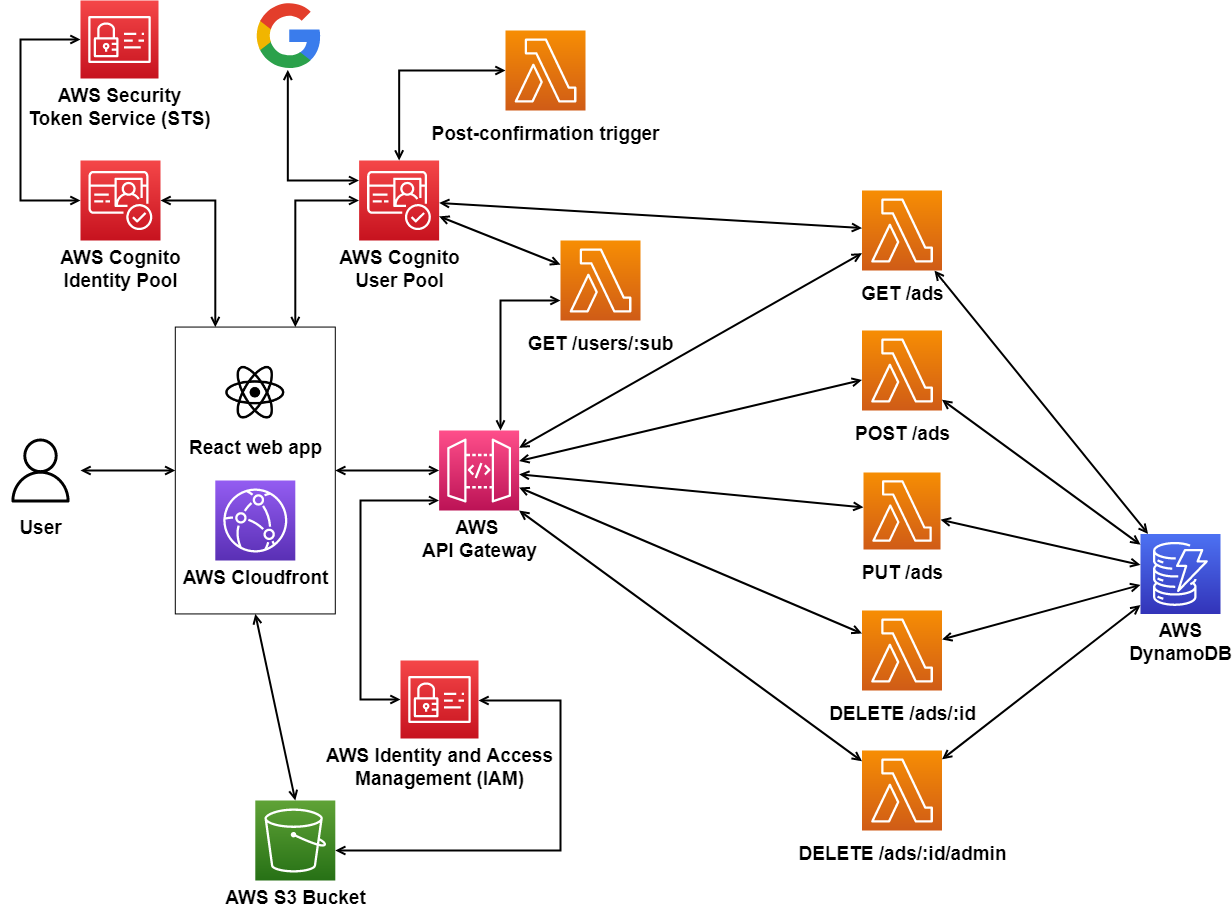
### class.png

Слика 3.2 – Класни дијаграм

Модел је поприлично једноставан, с тим што ће се подаци везани за кориснике (*User*) чувати у оквиру *User Pool*-a, док ће се подаци везани за оглас (*Ad*) чувати у Dynamo бази. Постојаће још података везаних за кориснике који подразмевано постоје у *User Pool*-у, али овде су наведени само они који нас занимају у оквиру наше апликације. *PreferredMFA* представља набројиви тип који описује одабрану методу двофакторске аутентификације корисника и долази нам из *User Pool*-а са својим предефинисаним вредностима. *TOTP* представља метод двофакторске аутентифкације који нећемо користити у оквиру Адверто апликације.

### Архитектура система

На слици 3.3 налази се дијаграм који описује повезаност свих сервиса које ћемо користити у реализацији.

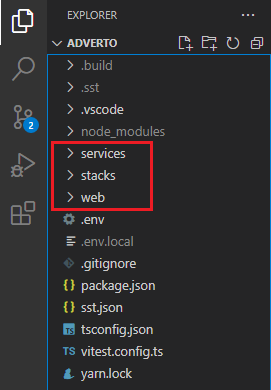


Слика 3.3 – Дијаграм архитектуре система

# ИМПЛЕМЕНТАЦИЈА

Како би описали имплементацију, за почетак ћемо размотрити структуру самог пројекта, а потом ћемо проћи кроз код за конфигурисање свих ресурса које креирамо помоћу *SST*-а, као и кроз код који се тиче имплементације конкретних функционалности.

## Структура пројекта



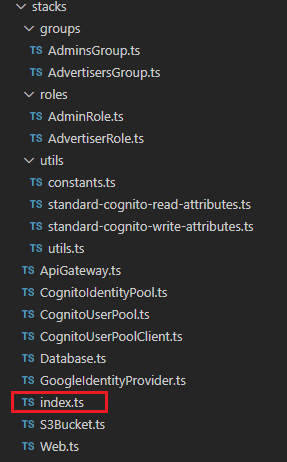
Слика 4.1 – Структура пројекта

Три најзначајнија директоријума којима ћемо се бавити су:

* *stacks* – садржи кодове везане за подизање инфраструктуре
* *services* – садржи кодове који се извршавају унутар Lambda функција, односно бекенд
* *web* – садржи кодове везане за веб апликацију, односно фронтенд

## Креирање ресурса

У оквиру директоријума *stacks* имамо посебну датотеку за конфигурисање сваког ресурса, а датотека *index.ts* представља полазну тачку *deployment* процеса.



Слика 4.2 – Структура *stacks* директоријума

### База података

У оквиру базе података ћемо чувати огласе тако да буду партиционирани по идентификатору креатора, а сортирани по идентификатору огласа. Поред тога, потребан нам је и глобални индекс за добављање огласа сортираних по времену креирања.

export function Database({ stack }: StackContext) {

  const table = new Table(stack, 'table', {

    fields: {

      pk: 'string',

      sk: 'string',

      gsi1pk: 'string',

      gsi1sk: 'number',

    },

    primaryIndex: {

      partitionKey: 'pk',

      sortKey: 'sk',

    },

    globalIndexes: {

      gsi1: {

        partitionKey: 'gsi1pk',

        sortKey: 'gsi1sk',

      },

    },

  });

  return table;

}

Листинг 4.1 – Конфигурација базе података за огласе

### Складиштење датотека

У оквиру *S3* корпе морамо да подесимо *CORS*, као и да омогућимо приступ прегледању датотека свим корисницима додавањем полисе.

export function S3Bucket({ stack }: StackContext) {

  const bucket = new Bucket(stack, 'bucket', {

    cors: [

      {

        allowedMethods: [HttpMethods.PUT, HttpMethods.POST],

        allowedOrigins: ['\*'],

        allowedHeaders: ['\*'],

      },

    ],

  });

  bucket.addToResourcePolicy(

    new PolicyStatement({

      effect: Effect.ALLOW,

      actions: ['s3:GetObject'],

      resources: [`${bucket.bucketArn}/\*`],

      principals: [new AnyPrincipal()],

    })

  );

  return bucket;

}

Листинг 4.2 – Конфигурација складишта датотека

### Аутентификација корисника

Како би подесили аутентификацију и ауторизацију корисника, потребно је прво да креирамо *User Pool* и исконфигуришемо га у складу са раније прописаним захтевима.

export function CognitoUserPool({ stack }: StackContext) {

  const userPool = new UserPool(stack, 'userPool', {

    userPoolName: `${stack.stage}-${constants.APP\_NAME}-userPool`,

    selfSignUpEnabled: true,

    signInAliases: {

      email: true,

    },

    signInCaseSensitive: false,

    standardAttributes: {

      givenName: {

        required: true,

      },

      familyName: {

        required: true,

      },

    },

    userVerification: {

      emailStyle: VerificationEmailStyle.CODE,

      emailSubject: 'Adverto verification code',

      emailBody: 'Your Adverto verification code is {####}.',

    },

    passwordPolicy: {

      minLength: 8,

      requireLowercase: true,

      requireDigits: true,

      requireUppercase: true,

      requireSymbols: true,

    },

    mfa: Mfa.OPTIONAL,

    mfaSecondFactor: {

      sms: true,

      otp: false,

    },

    mfaMessage: 'Your Adverto code is {####}.',

  });

  userPool.addDomain('userPoolDomain', {

    cognitoDomain: {

      domainPrefix: `${stack.stage}-${constants.APP\_NAME}`,

    },

  });

  addTriggers(stack, userPool);

  return userPool;

}

Листинг 4.3 – Конфигурација *User Pool*-a

Функцијом *addTriggers* додајемо окидаче, тј. у нашем случају конкретно само окидач који се догађа након што корисник верификује свој имејл. Овај окидач ће додати корисника у групу за оглашаваче, чиме ће кориснику бити додељена рола оглашавача. Како би *Lambda* функција која представља окидач била ауторизована за то, морамо јој додати пермисију у виду полисе за додавање корисника у групу.

const addTriggers = (stack: Stack, userPool: UserPool) => {

  // post confirmation

  const postConfirmationTrigger = new NodejsFunction(

    stack,

    'postConfirmationTrigger',

    {

      runtime: Runtime.NODEJS\_16\_X,

      entry: 'services/functions/triggers/post-confirmation-trigger.ts',

    }

  );

  postConfirmationTrigger.role!.attachInlinePolicy(

    new Policy(stack, 'post-confirmation-trigger-policy', {

      statements: [

        new PolicyStatement({

          actions: ['cognito-idp:AdminAddUserToGroup'],

          effect: Effect.ALLOW,

          resources: [userPool.userPoolArn],

        }),

      ],

    })

  );

  userPool.addTrigger(

    UserPoolOperation.POST\_CONFIRMATION,

    postConfirmationTrigger

  );

};

Листинг 4.4 – *Post Confirmation Trigger*

Додавање корисника се унутар *Lambda* функције ради помоћу *CognitoIdentityServiceProvider* класе из библиотеке *AWS* *SDK*, односно њене методе *adminAddUserToGroup*.

const provider = new aws.CognitoIdentityServiceProvider({

region: event.region,

});

const params = {

  GroupName: 'Advertisers',

  Username: event.userName,

  UserPoolId: event.userPoolId,

};

await provider.adminAddUserToGroup(params).promise();

Листинг 4.5 – Додавање корисника у групу за оглашаваче

Након што смо креирали сам *User Pool*, потребно је да подесимо и *User Pool Client* апликацију која ће служити за комуникацију фронтенд и бекенд апликација са *User Pool*-ом, као и за комуникацију са екстерним провајдерима као што је *Google*.

export function CognitoUserPoolClient({ stack }: StackContext) {

  const userPool = use(CognitoUserPool);

  const userPoolClient = new UserPoolClient(stack, 'userPoolClient', {

    userPoolClientName: `${stack.stage}-${constants.APP\_NAME}-userPoolClient`,

    userPool,

    supportedIdentityProviders: [

      UserPoolClientIdentityProvider.COGNITO,

      UserPoolClientIdentityProvider.GOOGLE,

    ],

    readAttributes: generateReadAttributes(),

    writeAttributes: generateWriteAttributes(),

    oAuth: {

      callbackUrls: [

        stack.stage.startsWith('local')

          ? 'http://localhost:3000'

          : process.env.SITE\_URL!,

      ],

      logoutUrls: [

        stack.stage.startsWith('local')

          ? 'http://localhost:3000/login'

          : `${process.env.SITE\_URL!}/login`,

      ],

    },

  });

  return userPoolClient;

}

Листинг 4.6 – Конфигурација *User Pool Client*-a

Последњи корак у подашавању аутентификације јесте повезивање *Google* апликације преко креденцијала које смо добили након креирања апликације унутар *Google* *Developer* конзоле, као и да подесимо мапирање података из *Google* налога на налог унутар нашег *User* *Pool*-a. Процес креирања апликације унутар *Google* *Developer* конзоле је изостављен јер је једноставан и своди се на праћење упутстава која нам *Google* даје.

export function GoogleIdentityProvider({ stack }: StackContext) {

  const userPool = use(CognitoUserPool);

  const googleIDP = new UserPoolIdentityProviderGoogle(stack, 'googleIDP', {

    userPool,

    clientId: constants.GOOGLE\_OAUTH\_CLIENT\_ID,

    clientSecret: constants.GOOGLE\_OAUTH\_CLIENT\_SECRET,

    // Email scope is required, because the default is 'profile' and that doesn't allow Cognito

    // to fetch the user's email from his Google account after the user does an SSO with Google

    scopes: ['email', 'profile', 'openid'],

    // Map fields from the user's Google profile to Cognito user fields, when the user is auto-provisioned

    attributeMapping: {

      givenName: ProviderAttribute.GOOGLE\_GIVEN\_NAME,

      familyName: ProviderAttribute.GOOGLE\_FAMILY\_NAME,

      email: ProviderAttribute.GOOGLE\_EMAIL,

      profilePicture: ProviderAttribute.GOOGLE\_PICTURE,

    },

  });

  return googleIDP;

}

Листинг 4.7 – Конфигурација *Google Identity Provider*-a

### Бекенд

При конфигурацији *API Gateway*-a и свих *Lambda* функција у оквиру њега, намапираћемо сваку руту на функцију која јој одговара и подесити тип ауторизације који желимо да применимо на тој рути. Поставићемо да је подразумевана ауторизација *IAM* ауторизација, а онда ћемо у оквиру јавних рута експлицитно нагласити да ауторизација на њима није потребна. Такође, потребно је да *Lambda* функцијама које за потребе приказа добављају информације о корисницима омогућимо приступ *User Pool*-у, као и *Dynamo* бази података и *S3* складишту.

export function ApiGateway({ stack }: StackContext) {

  const db = use(Database);

  const bucket = use(S3Bucket);

  const userPool = use(CognitoUserPool);

  const api = new Api(stack, 'api', {

    defaults: {

      authorizer: 'iam',

      function: {

        permissions: [db, bucket],

        environment: {

          REGION: stack.region,

          TABLE\_NAME: db.tableName,

          BUCKET\_NAME: bucket.bucketName,

          USER\_POOL\_ID: userPool.userPoolId,

        },

      },

    },

    routes: {

      // create ad

      'POST /commercials': 'functions/ads/create-ad.handler',

      // get all ads

      'GET /commercials': {

        function: 'functions/ads/get-ads.handler',

        authorizer: 'none',

      },

      // create ad

      'PUT /commercials': 'functions/ads/update-ad.handler',

      // delete ad

      'DELETE /commercials/{id}': 'functions/ads/delete-ad.handler',

      // admin delete ad

      'DELETE /commercials/{id}/admin': 'functions/ads/delete-ad-admin.handler',

      // get user

      'GET /users/{sub}': {

        function: 'functions/users/get-user.handler',

        authorizer: 'none',

      },

    },

  });

  const listUsersStatement = new PolicyStatement({

    actions: ['cognito-idp:ListUsers'],

    effect: Effect.ALLOW,

    resources: [userPool.userPoolArn],

  });

  api.getFunction('GET /commercials')?.addToRolePolicy(listUsersStatement);

  api.getFunction('GET /users/{sub}')?.addToRolePolicy(listUsersStatement);

  return api;

}

Листинг 4.8 – Конфигурација складишта датотека

### Ауторизација корисника

Да би ауторизовали кориснике потребно је да прво подесимо *Identity Pool* и одаберемо метод евалуације роле, а потом креирамо групе и роле помоћу којих ћемо разграничити ко чему има приступ како у оквиру *API*–ја, тако и у оквиру других претходно конфигурисаних ресурса.

export function CognitoIdentityPool({ stack }: StackContext) {

  const userPool = use(CognitoUserPool);

  const userPoolClient = use(CognitoUserPoolClient);

  const identityPool = new CfnIdentityPool(stack, 'identityPool', {

    identityPoolName: `${stack.stage}-${constants.APP\_NAME}-identityPool`,

    allowUnauthenticatedIdentities: true,

    cognitoIdentityProviders: [

      {

        providerName: userPool.userPoolProviderName,

        clientId: userPoolClient.userPoolClientId,

      },

    ],

  });

  new CfnIdentityPoolRoleAttachment(stack, 'identityPoolRoleAttachment', {

    identityPoolId: identityPool.ref,

    roles: {},

    roleMappings: {

      mapping: {

        type: 'Token',

        ambiguousRoleResolution: 'Deny',

        identityProvider: `cognito-idp.${stack.region}.amazonaws.com/${userPool.userPoolId}:${userPoolClient.userPoolClientId}`,

      },

    },

  });

  return identityPool;

}

Листинг 4.9 – Конфигурација *Identity Pool*-a

Што се тиче група и рола, потребно је да прво креирамо роле, а потом их проследимо групама чији ће корисници добити те роле.

Оглашавач кроз полисе добија приступ одговарајућим рутама бекенда, као и спцифичном делу *S3* складишта који се односи на његову профилну слику.

export function AdvertiserRole({ stack }: StackContext) {

  const identityPool = use(CognitoIdentityPool);

  const advertiserRole = new Role(stack, 'advertiserRole', {

    assumedBy: new WebIdentityPrincipal('cognito-identity.amazonaws.com', {

      StringEquals: {

        'cognito-identity.amazonaws.com:aud': identityPool.ref,

      },

      'ForAnyValue:StringLike': {

        'cognito-identity.amazonaws.com:amr': 'authenticated',

      },

    }),

  });

  addPolicyStatements(stack, advertiserRole);

  return advertiserRole;

}

function addPolicyStatements(stack: Stack, advertiserRole: Role) {

  const api = use(ApiGateway);

  const executeApiRoot = generateExecuteApiRoot(stack, api.httpApiId);

  advertiserRole.addToPolicy(

    new PolicyStatement({

      effect: Effect.ALLOW,

      actions: ['execute-api:Invoke'],

      resources: [

        `${executeApiRoot}/POST/commercials`,

        `${executeApiRoot}/PUT/commercials`,

        `${executeApiRoot}/DELETE/commercials/\*`,

      ],

    })

  );

  const bucket = use(S3Bucket);

  const principalsProfilePictureWithoutExtension = `${bucket.bucketArn}/profile-pictures/\${cognito-identity.amazonaws.com:sub}`;

  advertiserRole.addToPolicy(

    new PolicyStatement({

      effect: Effect.ALLOW,

      actions: ['s3:PutObject'],

      resources: [

        `${principalsProfilePictureWithoutExtension}.png`,

        `${principalsProfilePictureWithoutExtension}.jpg`,

        `${principalsProfilePictureWithoutExtension}.jpeg`,

      ],

    })

  );

}

Листинг 4.10 – Конфигурација роле оглашавача

Администраторска рола добија додатне могућности у виду права на брисање огласа других корисника, излиставање корисника и блокирање/одблокирање корисника.

export function AdminRole({ stack }: StackContext) {

  const identityPool = use(CognitoIdentityPool);

  const adminRole = new Role(stack, 'adminRole', {

    assumedBy: new WebIdentityPrincipal('cognito-identity.amazonaws.com', {

      StringEquals: {

        'cognito-identity.amazonaws.com:aud': identityPool.ref,

      },

      'ForAnyValue:StringLike': {

        'cognito-identity.amazonaws.com:amr': 'authenticated',

      },

    }),

  });

  addPolicyStatements(stack, adminRole);

  return adminRole;

}

function addPolicyStatements(stack: Stack, adminRole: Role) {

  const api = use(ApiGateway);

  const executeApiRoot = generateExecuteApiRoot(stack, api.httpApiId);

  adminRole.addToPolicy(

    new PolicyStatement({

      effect: Effect.ALLOW,

      actions: ['execute-api:Invoke'],

      resources: [

        `${executeApiRoot}/POST/commercials`,

        `${executeApiRoot}/PUT/commercials`,

        `${executeApiRoot}/DELETE/commercials/\*`,

        `${executeApiRoot}/DELETE/commercials/\*/admin`,

      ],

    })

  );

  const userPool = use(CognitoUserPool);

  adminRole.addToPolicy(

    new PolicyStatement({

      actions: [

        'cognito-idp:ListUsers',

        'cognito-idp:AdminDisableUser',

        'cognito-idp:AdminEnableUser',

      ],

      effect: Effect.ALLOW,

      resources: [userPool.userPoolArn],

    })

  );

}

Листинг 4.11 – Конфигурација роле администратора

Након што смо подесили роле, креирамо групе и придружујемо им одговарајуће роле. Администраторима ћемо доделити првенство 0, а оглашавачима 1 како би се приликом евалуације роле корисницима који се налазе у обе групе доделила рола администратора.

export function AdvertiserGroup({ stack }: StackContext) {

  const userPool = use(CognitoUserPool);

  const role = use(AdvertiserRole);

  const advertisersGroup = new CfnUserPoolGroup(stack, 'advertisersGroup', {

    userPoolId: userPool.userPoolId,

    groupName: 'Advertisers',

    precedence: 1,

    roleArn: role.roleArn,

  });

  return advertisersGroup;

}

export function AdminsGroup({ stack }: StackContext) {

  const userPool = use(CognitoUserPool);

  const role = use(AdminRole);

  const adminsGroup = new CfnUserPoolGroup(stack, 'adminsGroup', {

    userPoolId: userPool.userPoolId,

    groupName: 'Admins',

    precedence: 0,

    roleArn: role.roleArn,

  });

  return adminsGroup;

}

Листинг 4.12 – Конфигурација корисничких група

### Фронтенд

На крају нам остаје још само конфигурација фронтенда.

export function Web({ stack }: StackContext) {

  const apiGateway = use(ApiGateway);

  const userPool = use(CognitoUserPool);

  const userPoolClient = use(CognitoUserPoolClient);

  const identityPool = use(CognitoIdentityPool);

  const bucket = use(S3Bucket);

  const site = new ReactStaticSite(stack, 'site', {

    path: 'web',

    buildCommand: 'npm run build',

    environment: {

      REACT\_APP\_API\_URL: apiGateway.url,

      REACT\_APP\_AWS\_REGION: stack.region,

      REACT\_APP\_USER\_POOL\_ID: userPool.userPoolId,

      REACT\_APP\_USER\_POOL\_DOMAIN: `${stack.stage}-${constants.APP\_NAME}.auth.${stack.region}.amazoncognito.com`,

      REACT\_APP\_USER\_POOL\_CLIENT\_ID: userPoolClient.userPoolClientId,

      REACT\_APP\_IDENTITY\_POOL\_ID: identityPool.ref,

      REACT\_APP\_S3\_BUCKET\_NAME: bucket.bucketName,

    },

  });

  return site;

}

Листинг 4.13 – Конфигурација фронтенда

## Имплементација функционалности

Како би у оквиру фронтенда једноставно комуницирали са Cognito и *API Gateway* сервисима морамо да подесимо параметре на основу којих ће функционисати *AWS Amplify* библиотека. У оквиру ње, користићемо класу *Auth* за комуникацију са *Cognito* сервисом, док ћемо помоћу класе *API* интераговати са *API Gateway*-ом. Важно је напоменути да, с обзиром на нашу конфигурацију *User Pool*-а, параметар *username* који ће се често јављати заправо представља имејл. У оквиру овог поглавља, класе које буду спомињане односе се на библиотеку *AWS Amplify* уколико не буде наглашено другачије.

const awsAmplifyConfig = {

  Auth: {

    mandatorySignIn: true,

    region: process.env.REACT\_APP\_AWS\_REGION,

    userPoolId: process.env.REACT\_APP\_USER\_POOL\_ID,

    identityPoolId: process.env.REACT\_APP\_IDENTITY\_POOL\_ID,

    userPoolWebClientId: process.env.REACT\_APP\_USER\_POOL\_CLIENT\_ID,

  },

  API: {

    endpoints: [

      {

        name: 'api',

        endpoint: process.env.REACT\_APP\_API\_URL,

        region: process.env.REACT\_APP\_AWS\_REGION,

      },

    ],

  },

  oauth: {

    domain: process.env.REACT\_APP\_USER\_POOL\_DOMAIN,

    redirectSignIn: generateRedirectSignIn(),

    redirectSignOut: generateRedirectSignOut(),

    responseType: 'token',

    scope: ['email', 'profile', 'openid'],

  },

};

function generateRedirectSignIn() {

  return window.location.origin;

}

function generateRedirectSignOut() {

  return window.location.origin + '/login';

}

Amplify.configure(awsAmplifyConfig);

Листинг 4.14 – Конфигурација за AWS Amplify

### Регистрација и верификација

Регистрацију имплементирамо помоћу методе *signUp* класе *Auth*, након чега је потребно помоћу методе *confimSignUp* исте класе и потврдити налог уз приложени верификациони код.

const params = {

      username: data.email,

      password: data.password,

      attributes: {

        given\_name: data.givenName,

        family\_name: data.familyName,

      },

    };

await Auth.signUp(params);

await Auth.confirmSignUp(username, code);

Листинг 4.14 – Регистрација и верификација

Уколико је потребно, кориснику се може поново послати верификациони код методом *resendSignUp* класе *Auth*.

await Auth.resendSignUp(username);

Листинг 4.15 – Поновно слање верификационог кода

### Пријава

Пријаву имплементирамо помоћу методе *signIn* класе *Auth*, при чему је потребно да проследимо имејл и лозинку коју је корисник унео.

const response = await Auth.signIn(username, password);

Листинг 4.16 – Пријава

У случају укључене двофакторске аутентификације потребно је и проследити верифкациони код уз корисника. Корисник представља објекат класе *CognitoUser*, а добија се повратна вредност методе *signIn*.

const response = await Auth.confirmSignIn(mfaUser,code);

Листинг 4.17 – Двофакторска аутентификација

Пријава путем *Google* налога се имплементира помоћу *federatedSignIn* методе класе *Auth.*

await Auth.federatedSignIn({

      provider: CognitoHostedUIIdentityProvider.Google,

    });

Листинг 4.17 – Пријава путем *Google* налога

### Комуникација фронтенда са бекендом

За слање захтева ка бекенду коришћене су методе *get*, *post*, *put* и *delete* класе *API*, при чему као параметре прослеђујемо који *API* гађамо, на којој рути и који су додаци ток захтева (заглавља, тело и сл.).

const res = await API.get('api', '/commercials', {});

Листинг 4.17 – Пример комуникације са бекендом

### Добављање, креирање, измена и брисање огласa

Ове функционалности се пре свега заснивају на комуникацији са базом података помоћу класе *DocumentClient* из библиотеке *AWS SDK* при чему је битно да корисник може да манипулише само својим огласима што имплементирамо тако што из самог захтева који стиже на *Lambda* функцију као њен параметар екстрактујемо идентификатор корисника и потом вршимо проверу да ли је он власник тог огласа.

*Lambda* функције растерећујемо рада са сликама помоћу методе *createPresignedPost* класе из библиотеке *AWS* *SDK*. Тиме добијамо путању коју можемо да сачувамо у базу, а на фронтенд шаљемо темпорарни линк и креденцијале помоћу којих се може одрадити складиштење слике.

### Измена личних података и профилне фотографије корисникa

Атрибуте корисника мењамо помоћу методе *updateUserAttributes* класе *Auth*, при чему је потребно да проследимо тренутног аутентификованог корисника ког добијамо позивом методе *currentAuthenticatedUser* класе *Auth* и објекат са атрибутима и вредностима које желимо да изменимо.

const user = await Auth.currentAuthenticatedUser();

await Auth.updateUserAttributes(user, {

        given\_name: data.givenName,

        family\_name: data.familyName,

        phone\_number: data.phoneNumber,

      });

Листинг 4.14 – Промена лозинке и повратак налога

Промену профилне слике имплементирамо помоћу библиотеке *AWS* *SDK* тако што прво добавимо темпорарне креденцијале помоћу методе *getCredentials* класе *config*, потом те креденцијале искористимо за инстацирање објекта класе *s3* помоћу које позивом методе *upload* извршавамо слање слике.

### Укључивање и искључивање двофакторсе аутентификације

Укључивање двофакторске аутентификације имплементирамо помоћу методе *setPreferredMFA* класе *Auth*, при чему је потребно да проследимо тренутног аутентификованог корисника ког добијамо позивом методе *currentAuthenticatedUser* класе *Auth* и одабрани метод двофакторске аутентификације (“*SMS\_MFA*” или “*NO\_MFA*”).

const user = await Auth.currentAuthenticatedUser()

await Auth.setPreferredMFA(user, 'SMS\_MFA');

Листинг 4.14 – Подешавање двофакторске аутентификације

### Промена лозинке и повратак налога у случају заборављене лозинке

Обе фунцкионалности имплементирамо помоћу мeтода *forgotPassword* (помоћу које се на имејл шаље верификациони код за измену) и *submitForgotPassword* (помоћу које се потврђује измена) класе *Auth*.

await Auth.forgotPassword(username);

await Auth.forgotPasswordSubmit(

        username,

        data.code,

        data.password

      );

Листинг 4.14 – Промена лозинке и повратак налога

### Блокирање корисника

Блокирање корисника имплементирамо позивом методе *adminEnableUser* класе *CognitoIdentityServiceProvider* из библиотеке *AWS* *SDK*.

const provider = new CognitoIdentityServiceProvider({

      region: process.env.REACT\_APP\_AWS\_REGION,

      credentials: credentials,

    });

const params: AdminDisableUserRequest = {

      UserPoolId: process.env.REACT\_APP\_USER\_POOL\_ID!,

      Username: username,

    };

const response = await provider.adminEnableUser(params).promise();

# ДЕМОНСТРАЦИЈА

* Pokažite bitne elemente korišćenja aplikacije.
* Ovaj odeljak može biti poput uputstvo za korišćenje sistema.
* Opišite jedan scenario (ili više) pri korišćenjuvaše aplikacije. Korak po korak prikažite kako korisnici stupaju u interakciju s vašom aplikacijom. Ubacite slike sa izgledom ekrana, koji će ilustrovati važne faze u njegovom korišćenju. Ovaj scenario (ili više njih) koji je ovde predstavljen bi bilo poželjno da bude pokriven i dinamičkim dijagramima u Poglavlju 4, i delimično ili potpuno pokriven listinzima u Poglavlju 5.

# ЗАКЉУЧАК

Успешно смо имплементирали све функционалне и нефункционалне захтеве прописане спецификацијом, а при томе смо изградили веома скалабилан и безбедан систем који је лак за одржавање и представља прилично повољно решење са финансијског аспекта у поређењу са неким традиционалнијим приступима.

У будућности би погодно било додати пријављивање помоћу друштвене мреже *Facebook*, с обзиром да *AWS* то подржава. Поред тога, увођење *time-based one-time password* метода двофакторске аутентификације би добро дошло jer тај метод постаје све популарнији међу корисницима, а и такво функционисање двофакторске аутентификације би додатно срезало трошкове система.

# ЛИТЕРАТУРА

1. *Amazon Web Services*

<https://docs.aws.amazon.com/>

1. *AWS CDK*

<https://docs.aws.amazon.com/cdk/api/v2/docs/aws-construct-library.html>

1. *Serverless Stack Toolkit*

<https://docs.sst.dev/>

1. *AWS Amplify*

<https://docs.amplify.aws/lib/q/platform/js/>

1. *AWS SDK*

<https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html>

1. *React*

<https://reactjs.org/docs/getting-started.html>

# БИОГРАФИЈА

Лука Матић је рођен 18.04.1999. године у Сремској Митровици. Завршио је Гимназију "Стеван Пузић" у Руми. Факултет техничких наука у Новом Саду, смер Рачунарство и аутоматика, усмерење Примењене рачунарске науке и информатика, уписао је школске 2018/2019. године. Положио је све испите прописане планом и програмом и стекао услов за одбрану завршног рада.