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

Лука Матић

**Контрола приступа у серверлес апликацијама**

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:**

|  |
| --- |
| **Контрола приступа у серверлес апликацијама** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| Упознати се са AWS архитектуром за серверлес апликације као и са AWS сервисима, са посебним освртом на сервисе за контролу приступа. Специфицирати и имплементирати веб апликацију за постављање и преглед огласа, користећи наведене сервисе. Клијентски део апликације имплементирати коришћењем React библиотеке. Документовати решење |

|  |  |
| --- | --- |
| 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**: | Контрола приступа у серверлес апликацијама |
| 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**: | 6 / 61 / 0 / 0 / 27 / 0 / 0 |
| 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**: | Задатак рада представља веб апликација за читање и постављање огласа реализована кроз серверлес архитектуру са нагласком на функционалности везане за аутентификацију и ауторизацију. |
| 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**: | Access control in serverless applications |
| 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**: | 6 / 61 / 0 / 0 / 27 / 0 / 0 |
| 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**: | Subject of this thesis is a web application for reading and posting advertisements built using serverless architecture with accent on features regarding authentication and authorization. |
| Accepted by sci. Board on, **ASB**: |  |
| Defended on, **DE**: |  |
| Defense board, **DB**: |  |
| president |  |
| member |  |
| mentor |  |
| Mentor's signature | |

**Садржај**

[КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА 5](#_Toc115056435)

[KEY WORDS DOCUMENTATION 6](#_Toc115056436)

[1. УВОД 9](#_Toc115056437)

[2. КОРИШЋЕНЕ СОФТВЕРСКЕ ТЕХНОЛОГИЈЕ 11](#_Toc115056438)

[2.1 Алати за инфраструктуру кроз код 11](#_Toc115056439)

[2.2 Коришћени сервиси и библиотеке 12](#_Toc115056440)

[*2.2.1* Ф 12](#_Toc115056441)

[*2.2.2* Фронтенд (*Cloudfront, Amplify, AWS SDK)* 12](#_Toc115056442)

[*2.2.3* Бекенд (*API Gateway, Lambda)* 13](#_Toc115056443)

[2.2.4 База података (*DynamoDB*) 13](#_Toc115056444)

[2.2.5 Складиштење датотека (*S*3) 14](#_Toc115056445)

[2.2.6 Аутентификација и ауторизација 14](#_Toc115056446)

[3. СПЕЦИФИКАЦИЈА СИСТЕМА 19](#_Toc115056447)

[3.1 Спецификација захтева 19](#_Toc115056448)

[3.1.1 Функционални захтеви 19](#_Toc115056449)

[3.1.2 Нефункционални захтеви 21](#_Toc115056450)

[3.2 Спецификација система 22](#_Toc115056451)

[3.2.1 Модел података 22](#_Toc115056452)

[3.2.2 Архитектура система 23](#_Toc115056453)

[4. ИМПЛЕМЕНТАЦИЈА 24](#_Toc115056454)

[4.1 Структура пројекта 24](#_Toc115056455)

[4.2 Креирање ресурса 25](#_Toc115056456)

[4.2.1 База података 26](#_Toc115056457)

[4.2.2 Складиштење датотека 26](#_Toc115056458)

[4.2.3 Аутентификација корисника 27](#_Toc115056459)

[4.2.4 Бекенд 31](#_Toc115056460)

[4.2.5 Ауторизација корисника 33](#_Toc115056461)

[4.2.6 Фронтенд 38](#_Toc115056462)

[4.3 Имплементација функционалности 39](#_Toc115056463)

[4.3.1 Регистрација и верификација 40](#_Toc115056464)

[4.3.2 Пријава 41](#_Toc115056465)

[4.3.3 Комуникација фронтенда са бекендом 41](#_Toc115056466)

[4.3.4 Добављање, креирање, измена и брисање огласa 42](#_Toc115056467)

[4.3.5 Измена личних података и профилне фотографије корисникa 42](#_Toc115056468)

[4.3.6 Укључивање и искључивање двофакторсе аутентификације 43](#_Toc115056469)

[4.3.7 Промена лозинке и повратак налога у случају заборављене лозинке 43](#_Toc115056470)

[4.3.8 Блокирање корисника 43](#_Toc115056471)

[5. ДЕМОНСТРАЦИЈА 45](#_Toc115056472)

[6. ЗАКЉУЧАК 57](#_Toc115056473)

[ЛИТЕРАТУРА 59](#_Toc115056474)

[БИОГРАФИЈА 60](#_Toc115056475)

# 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. Програмери желе да брину о томе како функционише код у оквиру апликације, а не о томе да ли треба да зашрафе нову плочицу радне меморије унутар сервера.

Са циљем решавања тог проблема, појављује се рачунарство у облаку [1], 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 [1]:

* 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 би се путем мреже подешавао удаљени сервер. Ово јесте много боље од тога да програмер зашрафљује сервер у свом подруму, али ипак мора донекле да води рачуна о том удаљеном серверу (нпр. да подешава оперативни систем на њему). Временом се појавила идеја о серверлес архитектури [1], која подразумева да програмер брине 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 се утврђује који ентитет приступа систему, док се ауторизацијом утврђује којим деловима система ентитет може да приступи.

У оквиру овог рада посматраће се имплементација аутентификације и ауторизације корисника унутар апликације Адверто. Адверто представља апликацију за постављање и читање огласа, која је изграђена у серверлес архитектури, коришћењем сервиса које нуди један од водећих клауд провајдера данашњице. У питању је компанија Амазон [2], односно њен огранак који се зове *AWS (Amazon Web Services)* [3]*. 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 ће се посматрати технологије које су коришћене за израду Адверто апликације. Прво ће бити размотрен алат за подизање инфраструктуре кроз код, а потом бити описани сервиси који су коришћени у оквиру инфраструктуре заједно са библиотекама помоћу којих се врши интеракција са сервисима у оквиру апликације.

## Алати за инфраструктуру кроз код

Сервисе на *AWS*-у је могуће је креирати на 3 начина:

* Коришћењем AWS-ове веб апликације познате као конзола (*AWS Console)* [4]
* Коришћењем интерфејса командне линије (*AWS CLI – AWS Command Line Interface*) [5]
* Коришћењем неког од алата за подизање инфраструктуре кроз код (*Infrastructure as Code – IaC*) [1]

Први начин је погодан за упознавање са сервисима, али у продукцијском окружењу није препоручљив јер је потребно некако памтити шта је све и којим редом искликтано у конзоли, док је други начин превише напоран за инжењере. Најбољи од претходно наведених начина јесте трећи начин, који омогућава да се уредно кроз код бележе сви кораци који су направљени да би се креирали и подесили одређени сервиси. Основни алат, а уједно и сервис за подизање инфраструктуре кроз код, у виду *JSON* или *YAML* синтаксе, јесте *Cloud Formation* [6].

Временом се испоставило да та синтакса није претерано погодна за програмере, па је *AWS* креирао *AWS CDK (Cloud Development Kit)* [7] 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)* [8]. *SST*, који ће се користити за израду Адверто апликације, додатно олакшава коришћење *CDK* тако што додаје могућност развоја *Lambda* функција уживо (енгл. *Live Lambda Development*) [9]. Ова функционалност омогућава да се током самог развоја уживо тестира код у окружењу једнаком оном које ће се налазити у продукцијској верзији.

## Коришћени сервиси и библиотеке

Сада ће бити размотрено који су то све сервиси који се појављују у оквиру апликације Адверто и како они функционишу у теорији. Прво ће бити описана архитектура на глобалном нивоу, а потом ће засебно бити описан сваки од сервиса.

### Ф

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

У оквиру фронтенда налази се веб апликација изграђена помоћу *React* [10] библиотеке, при чему је за стилизовање коришћен *CSS* радни оквир по имену *Tailwind CSS* [11], а хостинг је одрађен помоћу *AWS* сервиса *Cloudfront* [12] који обезбеђује дистрибуцију статичког и динамичког веб садржаја.

*AWS SDK* [13] је библиотека која служи за интеракцију са *AWS* сервисима, док је *AWS Amplify* [14] 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*.

### Бекенд (*API Gateway, Lambda)*

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

*AWS Lambda* [15] представља сервис који нуди рачунарске ресурсе специфичне за извршавање функције којој је тај сервис намењен. Када се користи *Lambda* сервис, не инстанцира се никакав сервер и не размишља се о администрацији, већ се брине само о коду 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 не кошта ништа. Први пут када се *Lambda* позове, одговор је нешто спорији због хладног подизања (енгл. *cold* *start*) који подразумева буђење 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* [16] 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*.

### База података (*DynamoDB*)

База података коју систем користити јесте *DynamoDB* [17]. *Dynamo* је нерелациона база креирана од стране *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 [1] 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.

### Складиштење датотека (*S*3)

*Amazon Simple Storage Service* [18], познатији као *S3*, јесте сервис за складиштење објеката који нуди велику скалабилност, доступност, безбедност и перформансе. Објекат (енгл. *object*) [1] 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 ће се више говорити у оквиру поглавља везаног за имплементацију.

### Аутентификација и ауторизација

Пре него што се ускочи у сам процес аутентификације и ауторизације, потребно је да разумети 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*) [19] 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.

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)* [20] алгоритма. Приступни кључ чине идентификатор приступног кључа (*Access Key Id*) и тајна приступног кључа (*Secret Access Key*). При коришћењу темпорарних креденцијала о којима ће бити речи 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)* [21] 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* [22] 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 [23]. 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.

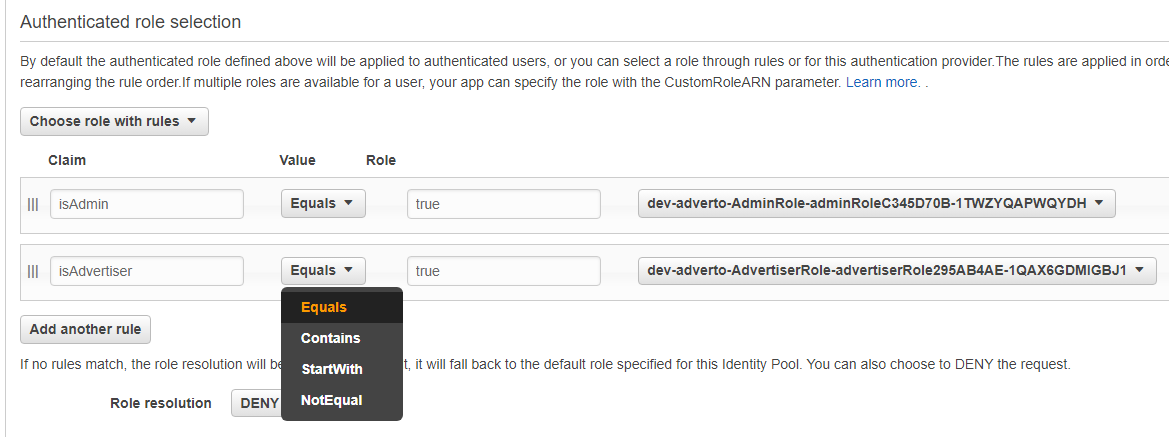
При успешној пријави, корисник као одговор добија три токена:

* *Access Token* [24] – токен који служи за аутентификацију корисника у оквиру *API*–ja *User Pool*-а (нпр. када корисник жели да измени своје податке)
* *Identity Token* [25] – 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* [26] – 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)* [27] 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 2.1) – На основу правила која посматрају вредности атрибута које корисник поседује



Слика 2.1 – 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 се кроз конзолу могу мењати групе корисника док се атрибути морају 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.



Слика 2.2 – Дијаграм тока аутентификације и ауторизације

На слици 2.2 приказан је дијаграм тока аутентификације и ауторизације у оквиру *AWS*-a на примеру приступања корисника ресурсу *S3* сервиса. Први корак је само пријављивање на систем при чему се контактира *Cognito User Pool*, који као одговор враћа токене. Затим се *Identity* токен шаље *Cognito Identity Pool*-у који проверава валидност токена и потом добавља темпорарне креденцијале за приступање ресурсима од *STS* сервиса и испоручује их клијентској апликацији. Клијентска апликација потписује захтев темпорарним креденцијалима помоћу *SigV4* алгоритма и шаље га ка *S3* сервису. *IAM* сервис врши проверу права приступа на основу креденцијала послатих у оквиру захтева и у случају валидних креденцијала сервис обрађује захтев и шаље одговор кориснику.

# СПЕЦИФИКАЦИЈА СИСТЕМА

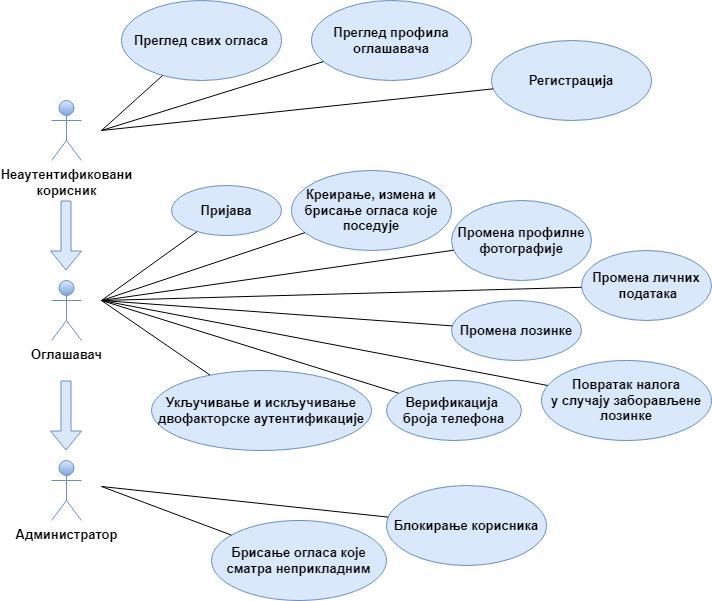
## Спецификација захтева

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

### Функционални захтеви

Функционални захтеви ће бити представљени помоћу дијаграма случајева коришћења, а потом ће дати дијаграм бити продискутован.

.



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 ће се прво кроз класни дијаграм представити модел система, а потом ће се кроз други дијаграм представити и архитектура система описана кроз повезаност сервиса.

### Модел података

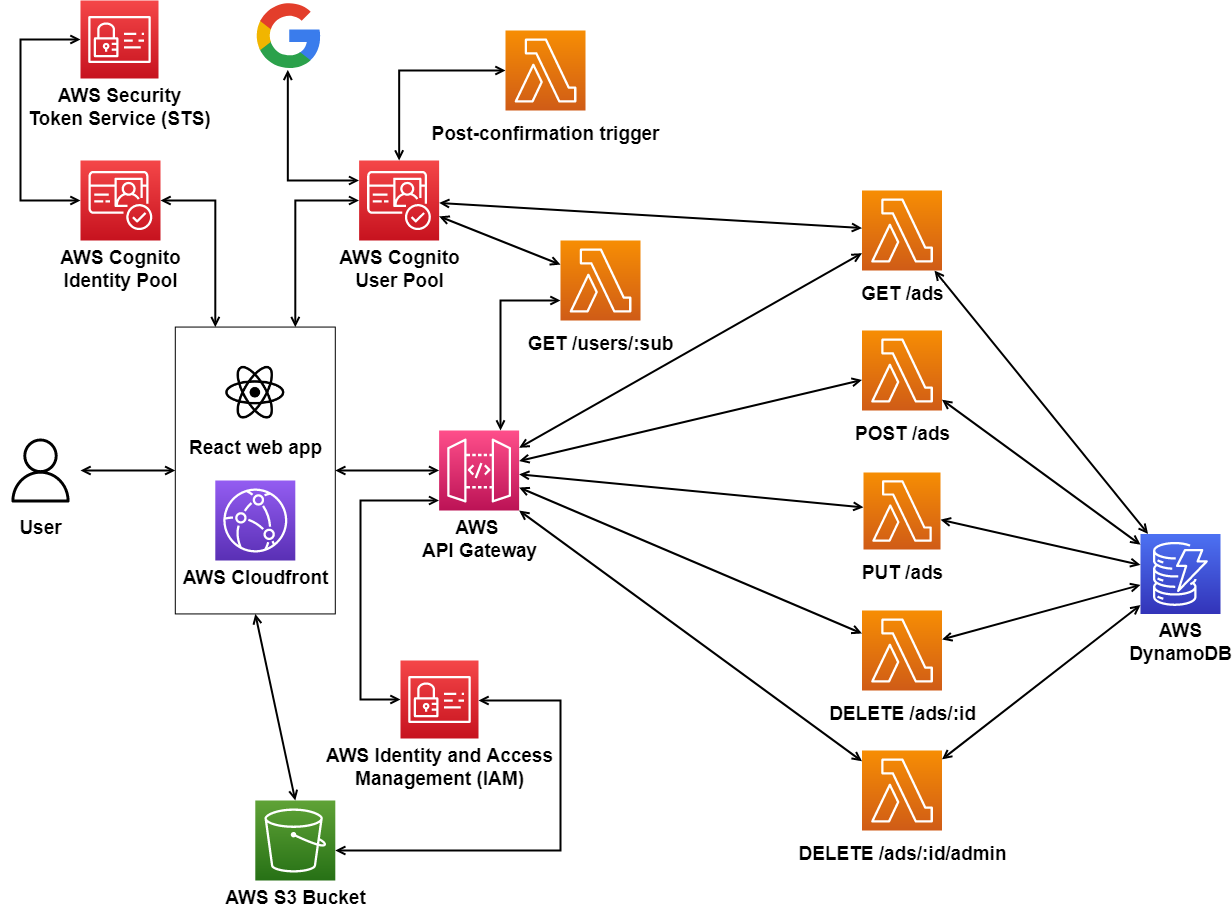
### class.png

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 неће бити коришћен у оквиру Адверто апликације.

### Архитектура система

На слици 3.3 налази се дијаграм који описује повезаност свих сервиса које систем користи.

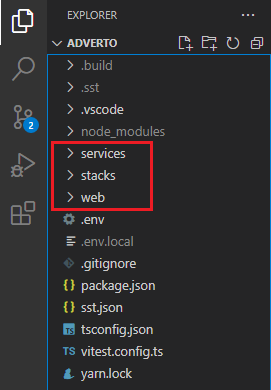


Слика 3.3 – Дијаграм архитектуре система

# ИМПЛЕМЕНТАЦИЈА

Како би се описала имплементација, за почетак ће бити размотрена структура самог пројекта, а потом ће се посматрати код за конфигурисање свих ресурса који се креирају помоћу *SST*-а, као и кроз код који се тиче имплементације конкретних функционалности.

## Структура пројекта



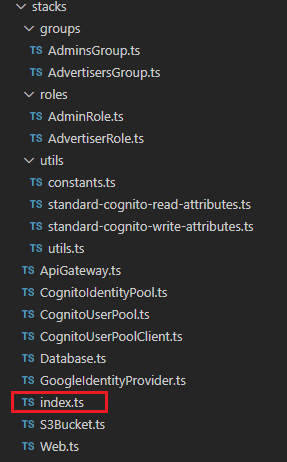
Слика 4.1 – Структура пројекта

Три најзначајнија директоријума која ће бити посматрана 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* налази се посебна датотека 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 се чувају огласи тако да буду партиционирани по идентификатору креатора, а сортирани по идентификатору огласа. Поред тога, потребан 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 мора се подесити *CORS (Cross-Origin Resource Sharing)* [28] тако што ће се назначити који извори, којим методама и са којим заглављима у оквиру захтева могу да приступају ресурсима. Такође, потребно је и омогућити 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 се подесила аутентификација и ауторизација корисника, потребно је прво креирати *User Pool* и исконфигурисати га у складу са раније прописаним захтевима који се односе на начин пријављивања, начин верификације, јачину лозинке и двофакторску аутентификацију.

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* додајеу се окидачи, 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, мора јој се додати пермисија 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 је креиран сам *User Pool*, потребно је подесити 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 који се добијају након креирања апликације унутар *Google* *Developer* конзоле, као и да подешавање мапирања података из *Google* налога на налог унутар *User* *Pool*-a. Процес креирања апликације унутар *Google* *Developer* конзоле је изостављен јер је једноставан и своди се на праћење упутстава која *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, намапираће се свака рута на функцију која јој одговара и подесити тип ауторизације који се примењује на тој рути. Поставиће се да је подразумевана ауторизација *IAM* ауторизација, а онда ће се у оквиру јавних рута експлицитно нагласити да ауторизација на њима није потребна. Такође, потребно је да се *Lambda* функцијама које за потребе приказа добављају информације о корисницима омогући 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

Ради ауторизације корисника потребно је да прво подесити *Identity Pool* и одаберати метод евалуације роле, а потом креирати групе и роле помоћу којих ће се 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 прво креирати роле, а потом их проследити 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 су подешене роле, креирају се групе и придружују им се одговарајуће роле. Администраторима ће бити додељено 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 остаје још само конфигурација фронтенда. Креира се *React* апликација и прослеђују јој се параметри на основу којих ће комуницирати са осталим сервисима..

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 фронтенд апликација једноставно комуницирала са Cognito и *API Gateway* сервисима морају се подесити параметри на основу којих ће функционисати *AWS Amplify* библиотека. У оквиру ње, користиће се класа *Auth* за комуникацију са *Cognito* сервисом, док ће се помоћу класе *API* интераговати са *API Gateway*-ом. Важно је напоменути да, с обзиром на 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

Регистрација се имплементира 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 4.15 – Регистрација и верификација

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

await Auth.resendSignUp(username);

Листинг 4.16 – Поновно слање верификационог кода

### Пријава

Пријава се имплементира помоћу методе *signIn* класе *Auth*, при чему је потребно проследити имејл и лозинку коју је корисник унео.

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

Листинг 4.17 – 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 4.18 – Двофакторска аутентификација

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

await Auth.federatedSignIn({

      provider: CognitoHostedUIIdentityProvider.Google,

    });

Листинг 4.19 – 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 се као прослеђују *API* који се гађа, рута која се гађа и додаци везани за захтев (заглавља, тело и сл.).

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

Листинг 4.20 – 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 се имплементира тако што се из самог захтева који стиже на *Lambda* функцију као њен параметар екстрактује идентификатор корисника и потом се врши провера да ли је он власник тог огласа.

*Lambda* функције се растерећују рада са сликама помоћу методе *createPresignedPost* класе из библиотеке *AWS* *SDK*. Тиме се добија путања коју се може да сачувати у базу, а на фронтенд се шаљу темпорарни линк и креденцијали помоћу којих се може одрадити складиштење слике.

### Измена личних података и профилне фотографије корисникa

Атрибуте корисника мењају се помоћу методе updateUserAttributes класе Auth, при чему је потребно прослеђивање тренутног аутентификованог корисника који се добија позивом методе currentAuthenticatedUser класе Auth и објекат са атрибутима и вредностима који се мењају.

const user = await Auth.currentAuthenticatedUser();

await Auth.updateUserAttributes(user, {

        given\_name: data.givenName,

        family\_name: data.familyName,

        phone\_number: data.phoneNumber,

      });

Листинг 4.21 – Промена лозинке и повратак налога

Промена профилне слике се имплементира помоћу библиотеке *AWS* *SDK* тако што се прво добавр темпорарни креденцијали помоћу методе *getCredentials* класе *config*, потом се ти креденцијали искористе за инстацирање објекта класе *s3* помоћу ког се позивом методе *upload* извршава слање слике.

### Укључивање и искључивање двофакторсе аутентификације

Укључивање двофакторске аутентификације имплементира се помоћу методе *setPreferredMFA* класе *Auth*, при чему је потребно проследити тренутног аутентификованог корисника који се добија 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 4.22 – Подешавање двофакторске аутентификације

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

Обе фунцкионалности имплементирају се 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 4.23 – Промена лозинке и повратак налога

### Блокирање корисника

Блокирање корисника имплементира се 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();

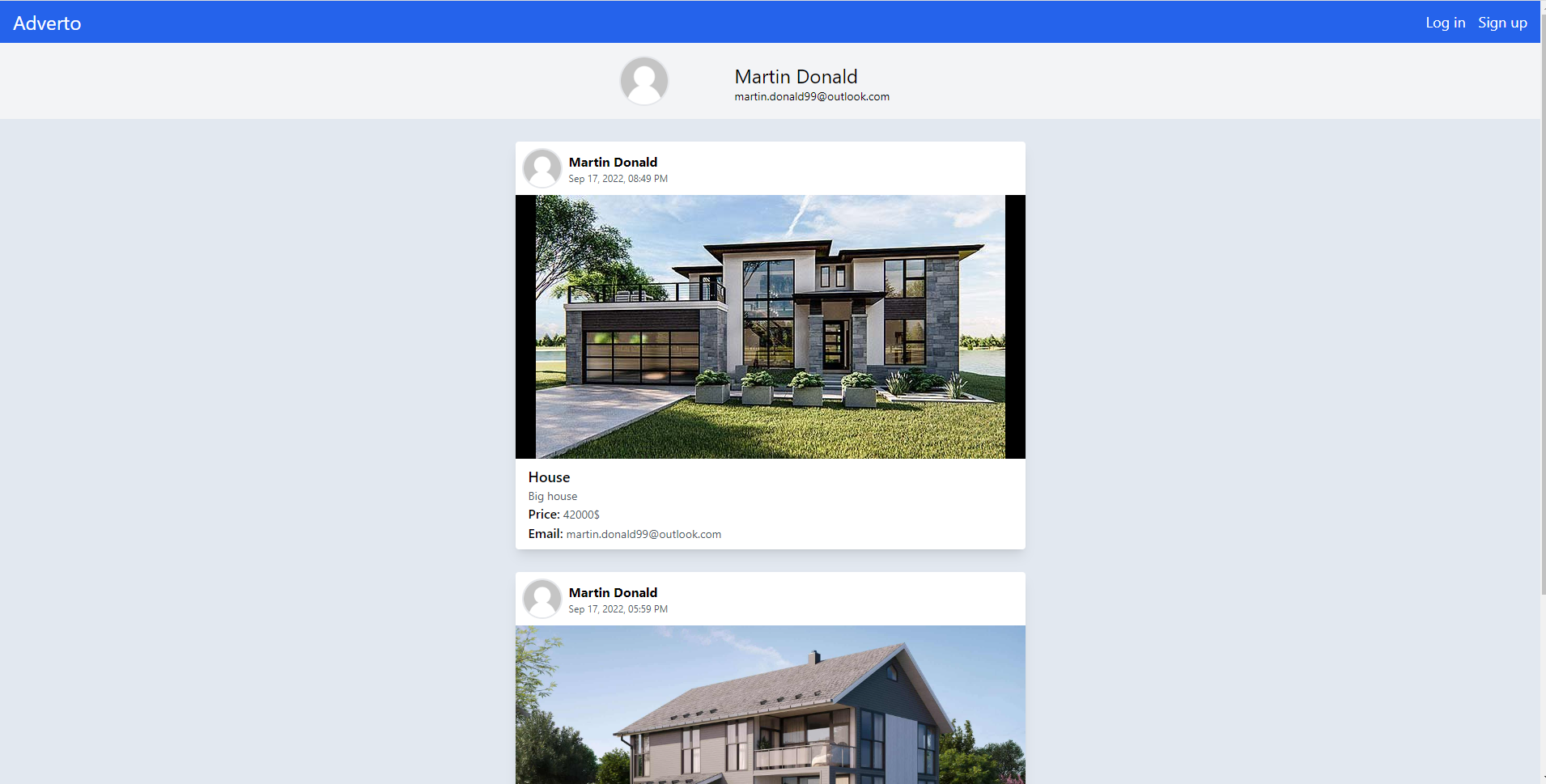
Листинг 4.24 – Блокирање корисника

# ДЕМОНСТРАЦИЈА

Кориснику се након приступа апликацији приказују огласи сортирани по датуму тако да се на врху појављују најновији огласи. Одавде има могућност да прегледа профиле корисника који су постављали огласе и њихове објаве такође соритране по времену тако да се прво приказују најновији огласи.

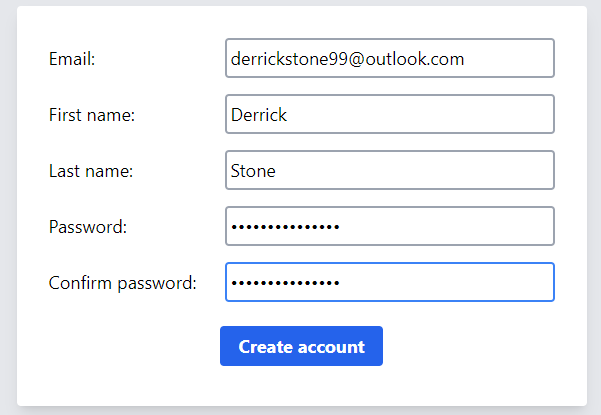


Слика 5.1 – Почетна страница

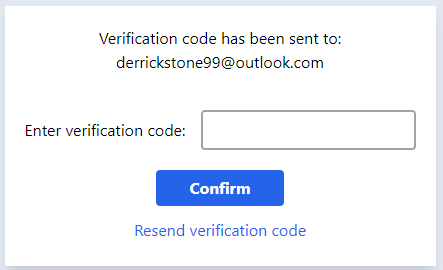


Слика 5.2 – Преглед профила

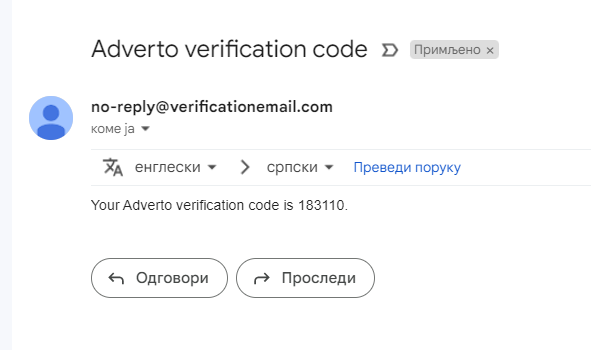
За даљи приступ апликацији, неаутентификовани корисник треба да се пријави на систем или да се региструје уколико нема креиран профил. При регистрацији се попуњавају неопходна поља и притиском на дугме *Create account* се креира профил, који се мора и потврдити верификационим кодом послатим на имејл адресу.



Слика 5.3 – Регистрација

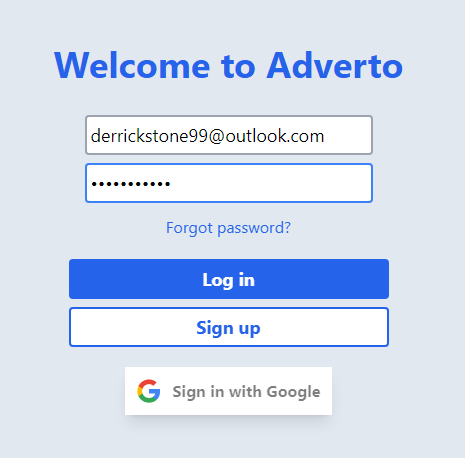


Слика 5.4 – Верификација профила



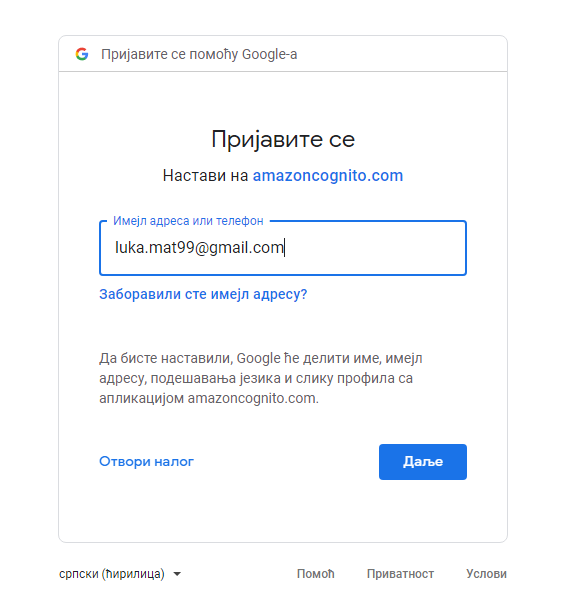
Слика 5.5 – Верификациони мејл

Уколико већ има креиран профил, неаутентификовани корисник прелази на страницу за пријаву на систем и попуњавањем оба поља и притиском на дугме *Login* се пријављује.

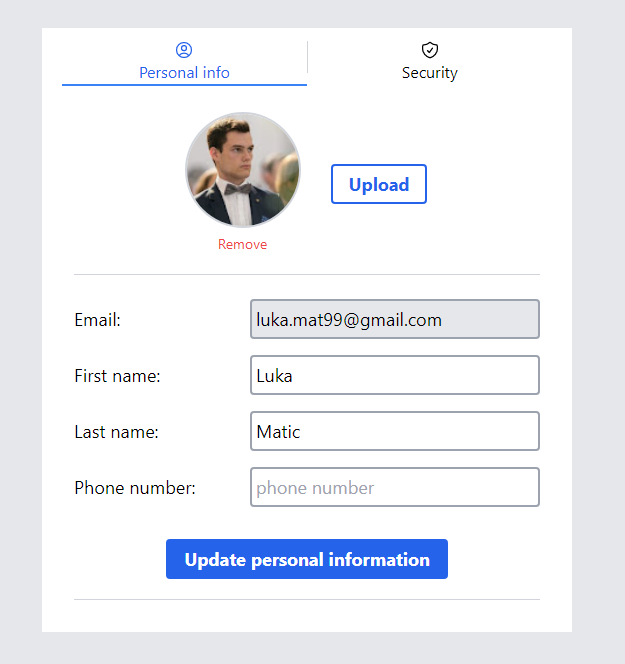


Слика 5.6 – Верификациони мејл

Пријава је такође могуће и помоћу *Google* налога. Одабиром ове опције се корисник редиректује на страницу избора *Google* налога. Након уноса *Google* налог креденцијала, корисник се редиректује на почетну страницу, а подаци са његовог Google налога као што су име, презиме, имејл и профилна слика се мапирају на податке у апликацији.

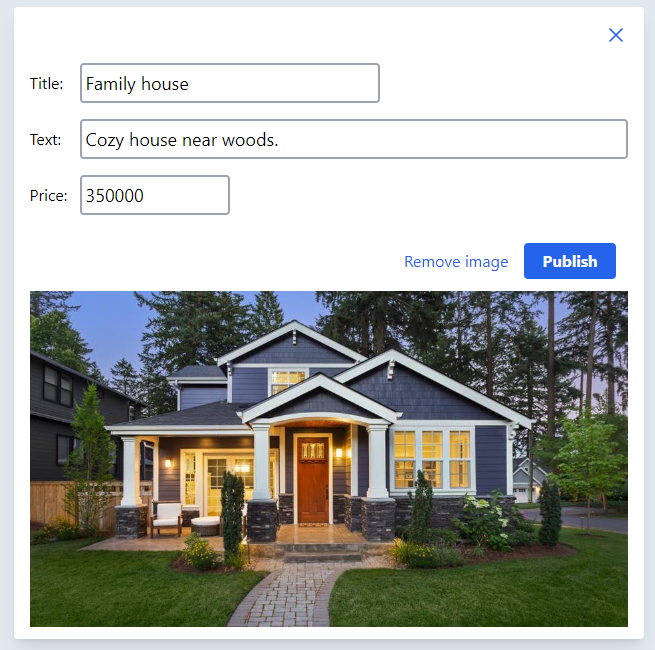


Слика 5.7 – Пријва путем *Google* налога



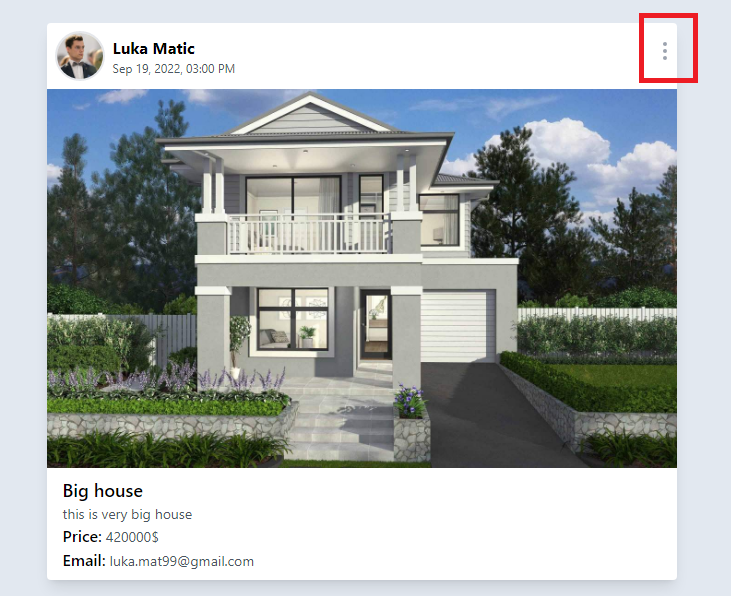
Слика 5.8 – Намапирани подаци са *Google* налога

Након пријаве на систем, корисник има могућност прављењa нових огласа попуњавањем одређених поља и одабиром слике.



Слика 5.9 – Креирање новог огласа

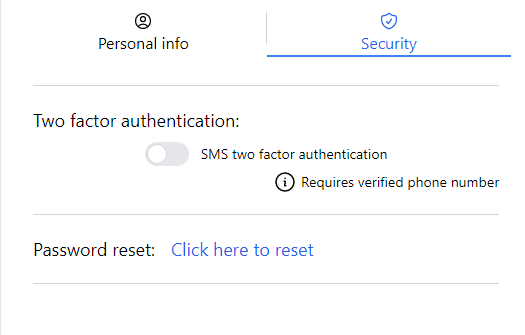
Такође корисник у горњем десном углу има дугме за отварање менија са опцијама за измену и брисање.



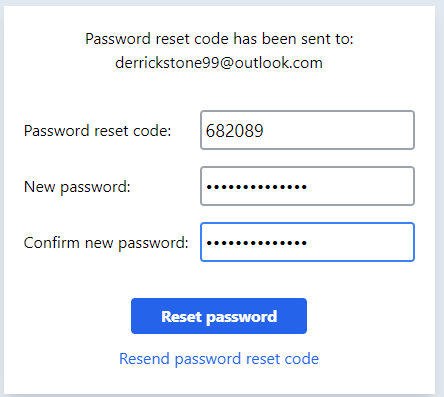
Слика 5.10 – Дугме за отварање менија са опцијама за измену и брисање огласа

На слици 5.8 такође види се и то да корисник има увид у своје личне податке, где може да промени профилну слику или измени име, презиме и број телефона.

Одабиром *Security* таба отварају се могућности за укључивање и искључивање двофакторске аутентификације, као и за ресетовање шифре. Одабиром опције за промену лозинке корисник добија имејл са кодом, који се шаље заједно са новом лозинком како би промена била верификована. На сличан начин функционише и повратак налога у случају заборављене лозинке, само што је у том случају потребно да корисник унесе имејл адресу на коју ће код бити послат.

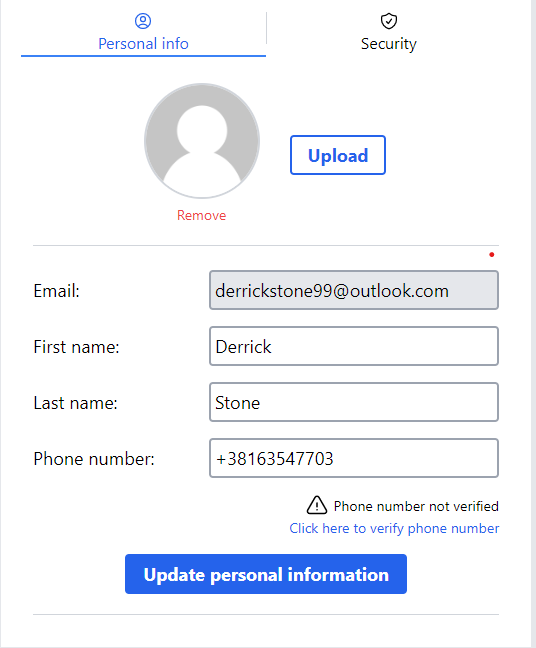


Слика 5.11 – *Security* таб

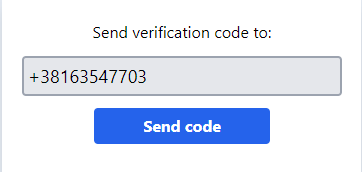


Слика 5.12 – Промена лозинке

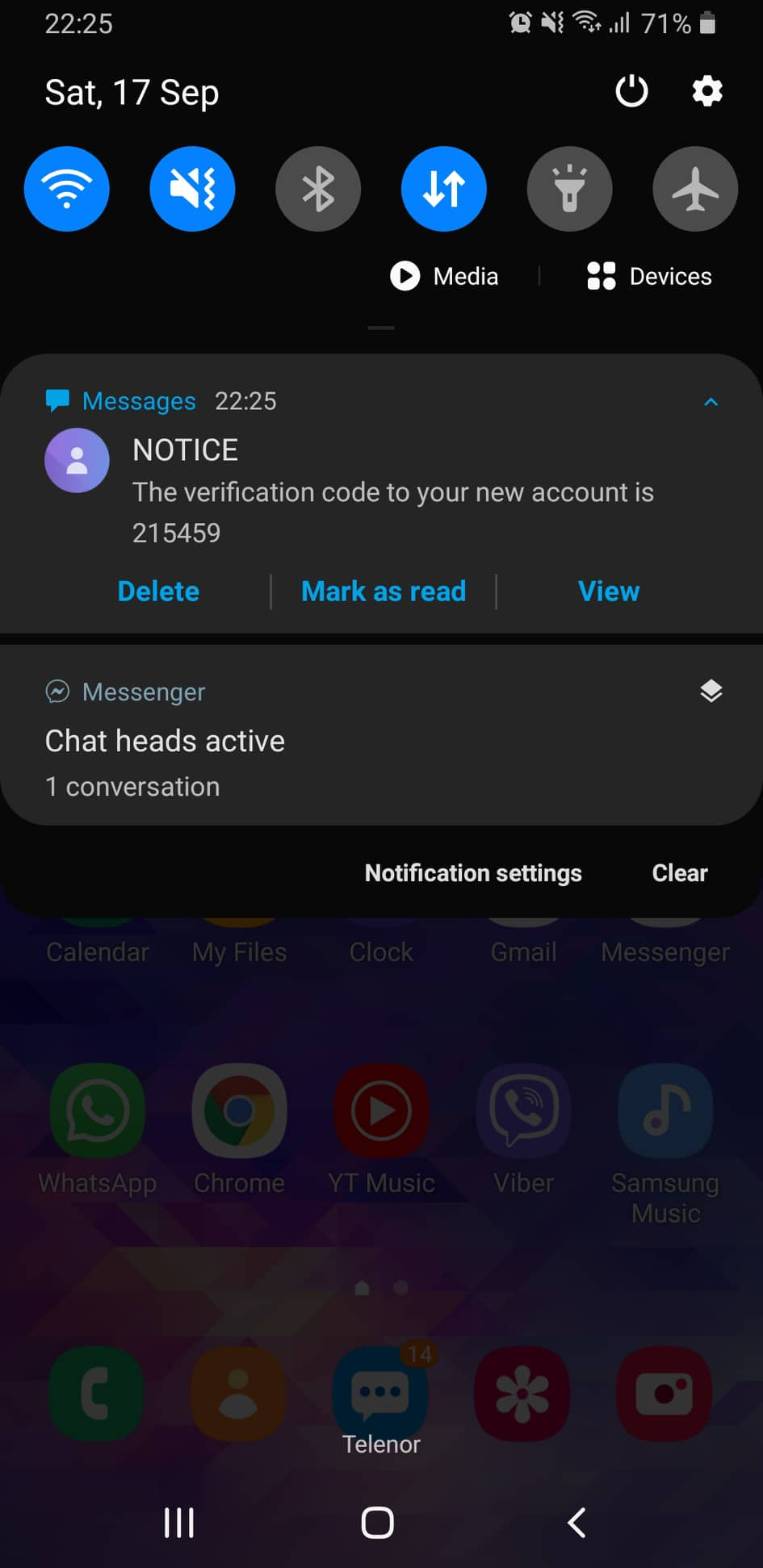
Како би се укључила двофакторска аутентификација потребно је прво верификовати број телефона. Након што је сачувао нови број телефон, корисник се кликом на линк за верификацију редиректује на страницу где се потврђује број тако уноси шестоцифрени број који је добио путем *SMS* поруке.



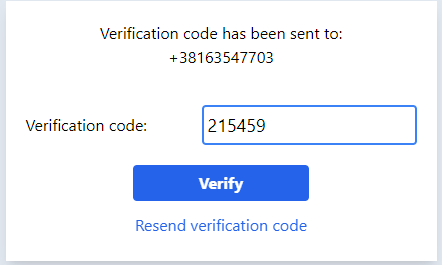
Слика 5.13 - Додавање броја телефона



Слика 5.14 – Слање верификационог кода

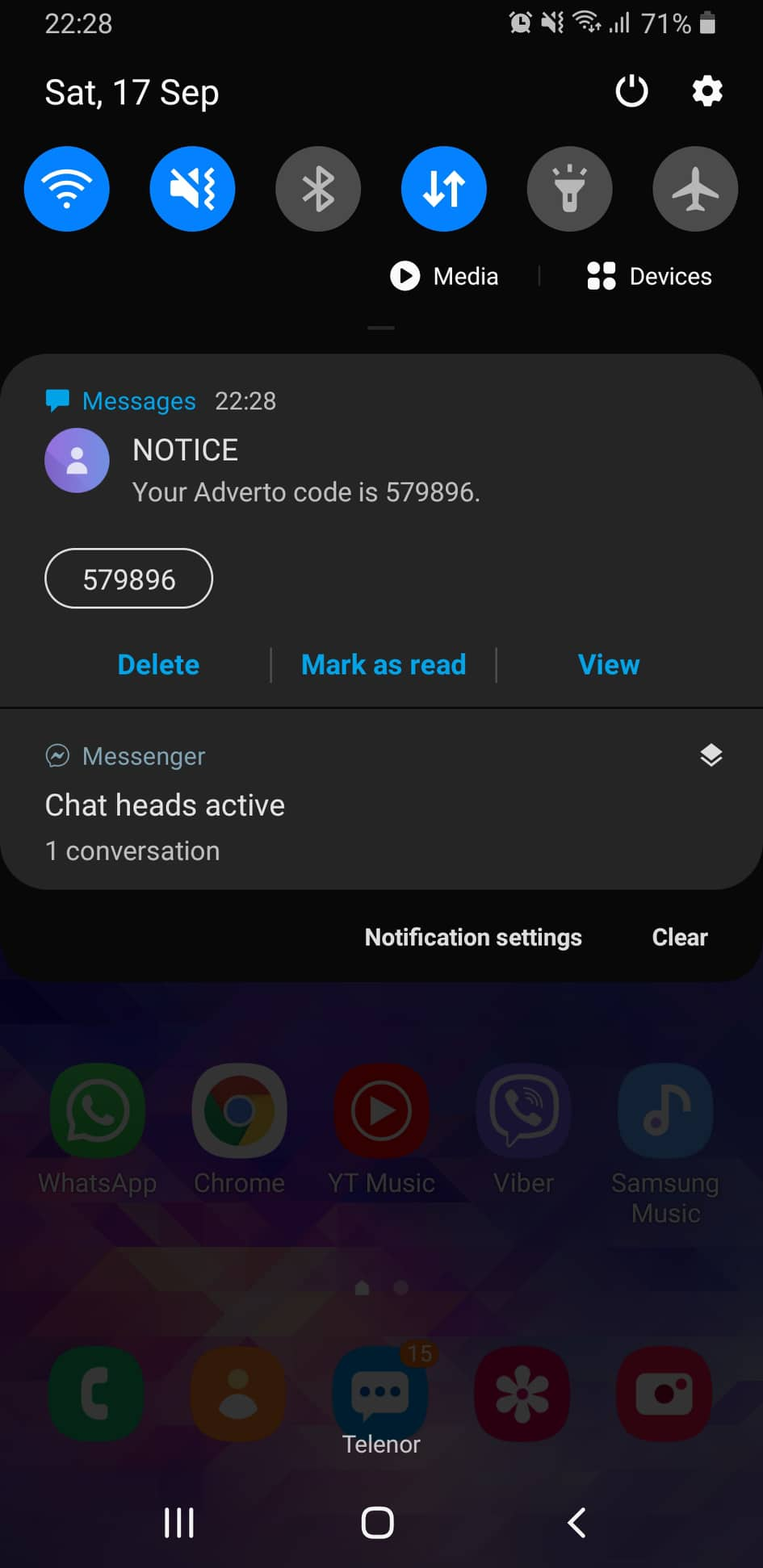


Слика 5.15 – Добијени верификациони код

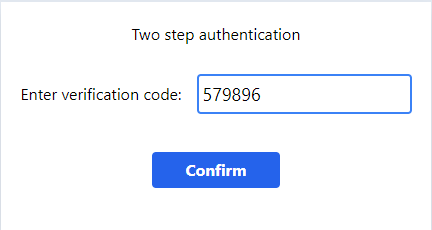


Слика 5.16 – Унос верификационог кода

Након што је корисник верификовао број телефона и укључивања двофакторске аутентификације, пријављивање корисника на систем ће захтевати поред мејла и лозинке и унoс шестоцифреног кода који се добија путем SMS-a.

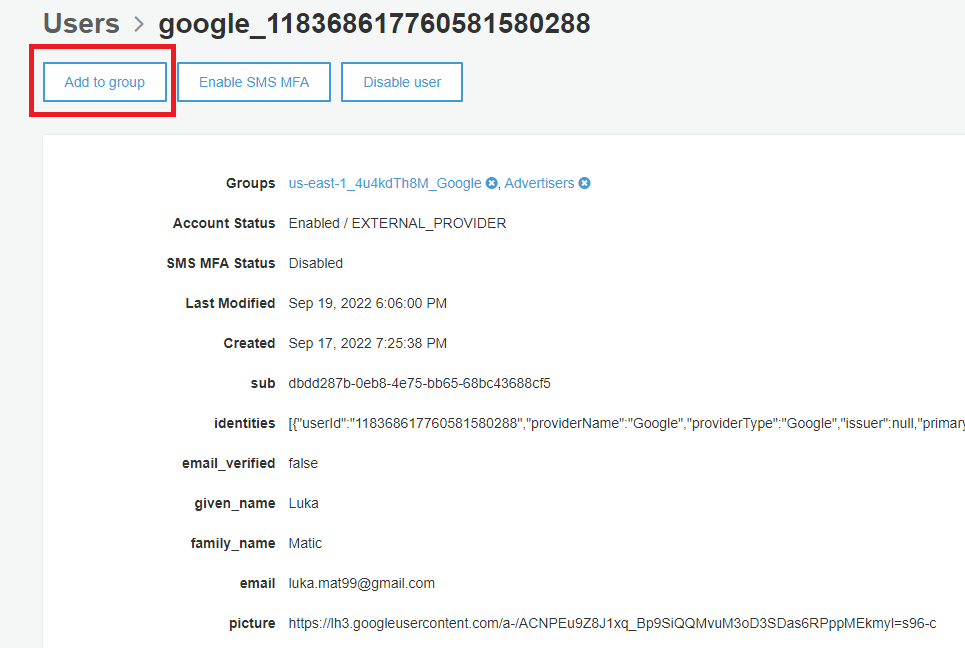


Слика 5.17 – Добијени код за двофакторску аутентификацију

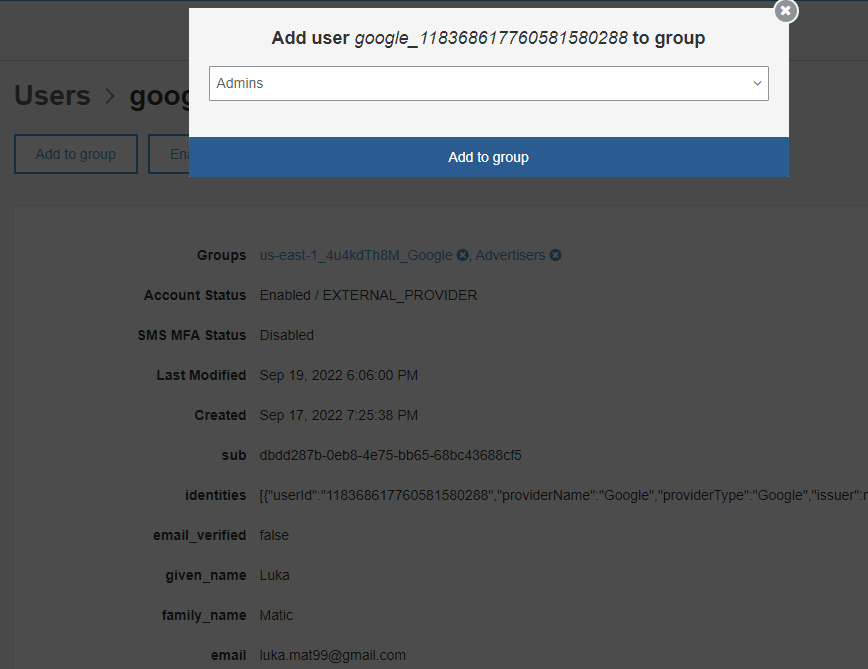


Слика 5.18 – Унос кода за двофакторску аутентификацију

Уколико је неком кориснику потребно доделити администраторску ролу, то се чини тако што се додаје у администраторску групу путем *АWS* конзоле.

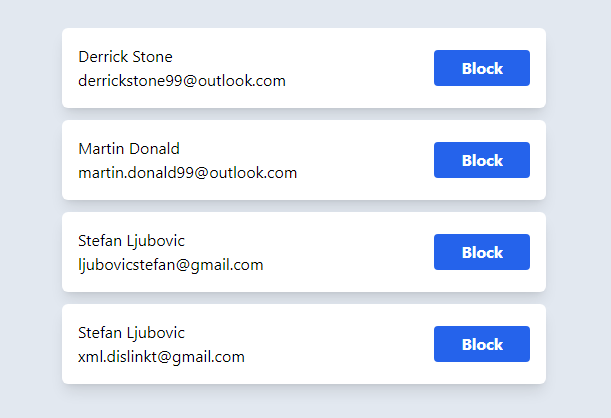


Слика 5.19 – Додавање корисника у администраторску групу (први корак)



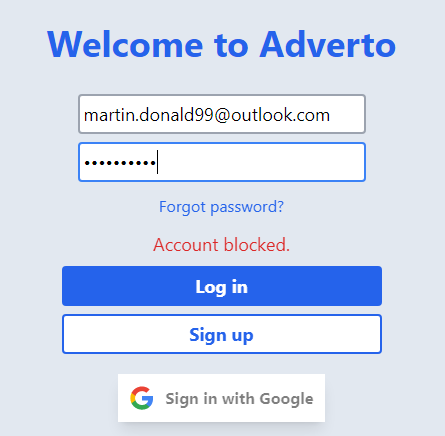
Слика 5.20 – Додавање корисника у администраторску групу (други корак)

Сада наш корисник има могућност да брише огласе других корисника, да види све кориснике система и да блокира или одблокира неког од корисника.



Слика 5.21 – Aдминистраторски преглед свих корисника

Након што је корисник блокиран, није више у могућности да се пријави на систем.



Слика 5.22 – Неуспешна пријава блокираног корисника

# ЗАКЉУЧАК

У уводном делу рада описан је смисао и значај рачунарства у облаку и серверлес архитектуре, а потом су описане и неке од технологија које се користе за развој серверлес апликација кроз разматрање теоријских аспеката њиховог функционисања. У оквиру спецификације система представљени су функционални и нефукнционални захтеви, као и архитектура читавог система кроз повезаност сервиса. Потом је описан и процес имплементације самог система који је, с обзиром на то да је имплементиран у виду инфраструктуре кроз код, лак за одржавање и представља прилично повољно решење са финансијског аспекта у поређењу са неким традиционалнијим приступима.

У будућности би било погодно 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 јер тај метод постаје све популарнији међу корисницима, а и такво функционисање двофакторске аутентификације би додатно срезало трошкове система.

# ЛИТЕРАТУРА

1. Michael Witting, Andreas Witting: *Amazon Web Services in Action 2nd Edition*, 2018
2. *Amazon*

<https://www.amazon.com/>

1. *Amazon Web Services*

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

1. *AWS Console*

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

1. *AWS CLI*

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

1. *AWS* *Cloud Formation*

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

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. *Live Lambda Development*

<https://docs.sst.dev/live-lambda-development>

1. *React*

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

1. *Tailwind CSS*

<https://tailwindcss.com/>

1. *AWS* *Cloudfront*

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

1. *AWS SDK*

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

1. *AWS Amplify*

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

1. *AWS* *Lambda*

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

1. *AWS* *API* *Gateway*

<https://aws.amazon.com/api-gateway/>

1. *AWS* *DynamoDB*

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

1. *AWS* *Simple* *Storage* *Service* *(S3)*

<https://aws.amazon.com/s3/>

1. *AWS* *Policies*

<https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html>

1. *Signature* *Version* *4 Algorithm*

<https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html>

1. *AWS Identity And Access Management*

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

1. *AWS* *Cognito*

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

1. *Customizing* *user* *pool* *workflows* *with* *Lambda* *triggers*

<https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-identity-pools-working-with-aws-lambda-triggers.html>

1. *Using* *the* *access* *Token*

<https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-using-the-access-token.html>

1. *Using* *the* *ID* *Token*

<https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-using-the-id-token.html>

1. *Using* *the* *refresh* *token*

<https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-using-the-refresh-token.html>

1. *Temporary* *security* *credentials* *in* *IAM*

<https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_temp.html>

1. *Cross-Origin* *Resource* *Sharing*

<https://docs.aws.amazon.com/AmazonS3/latest/userguide/cors.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.