



UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET



Miljan Denić

**IMPLEMENTACIJA SISTEMA ZA UPRAVLJANJE DOGAĐAJIMA U
MIKROSERVISNOJ ARHITEKTURI**

Master rad

Studijski program: Računarstvo i informatika

Modul: Softversko inženjerstvo

Student:

Miljan Denić, br. ind. 1517

Mentor:

doc. dr. Petar Rajković

Niš, april 2024. godina

Univerzitet u Nišu

Elektronski Fakultet

IMPLEMENTACIJA SISTEMA ZA UPRAVLJANJE DOGAĐAJIMA U MIKROSERVISNOJ ARHITEKTURI

IMPLEMENTATION OF THE EVENT MANAGEMENT SYSTEM IN MICROSERVICE ARCHITECTURE

Master rad

Studijski program: Informatika i računarstvo

Modul: Softversko inženjerstvo

Student: Miljan Denić, br. ind. 1517

Mentor: doc. dr. Petar Rajković

Zadatak:

Teoretska osnova rada obuhvata istraživanje Eventella platforme za upravljanje događajima, sa posebnim fokusom na analizu arhitekture sistema i ključnih komponenti.

Napravljena je analiza funkcionalnosti API Gateway-a u mikroservisnim sistemima kao i integracija Keycloak servisa za upravljanje identitetom.

U praktičnom delu će biti implementirana Eventella platforma. Biće odrađena analiza i demonstracija funkcionalnosti API Gateway-a, Keycloak-a kao i servisa u pozadini, pružajući uvid u proces implementacije i ulogu svake komponente u sistemu.

Datum prijave rada:

Datum predaje rada:

Datum odbrane rada:

Komisija za ocenu i odbranu:

1. <Prof. / Doc dr Ime Prezime>, Predsednik komisije

2. <Prof. / Doc dr Ime Prezime>, Član

3. <Prof. / Doc dr Ime Prezime>, Član

IMPLEMENTACIJA SISTEMA ZA UPRAVLJANJE DOGAĐAJIMA U MIKROSERVISNOJ ARHITEKTURI

SAŽETAK

Ovaj rad istražuje Eventella platformu za upravljanje događajima, fokusirajući se na arhitekturu sistema i ključne komponente. Poseban fokus je usmeren ka funkcionalnostima API Gateway-a u mikroservisnoj arhitekturi. Pored detaljne analize mogućnosti koje pruža API Gateway prikazana je i praktična implementacija najvažnijih funkcionalnosti kao što su autentifikacija, ograničavanje broja zahteva, rutiranje saobraćaja i logovanje informacija. Implementacija je odrađena u FastAPI razvojnom okviru. Pored implementacije najvažnijih funkcionalnosti prikazan je i pregled trenutno popularnih cloud i on-premise rešenja.

Detaljno je opisana integracija sa Keycloak servisom, sa naglaskom na osnovne koncepte, funkcionalnosti i korišćenje OpenID Connect protokola. Prikazane su osnovne komponente Keycloak servisa. Obradeni su različiti načini za integraciju sa primerima korišćenja.

Prikazana je implementacija i funkcionalnosti servisa u pozadini. Svaki od njih je objašnjen pojedinačno radi boljeg razumevanja njihove uloge u sistemu. Kroz analizu ovih komponenti, rad pruža uvid u korišćene tehnologije i aktuelne prakse za kreiranje sigurnog i skalabilnog sistema za upravljanje događajima.

Ključne reči: mikroservisi, API Gateway, Keycloak, Eventella, FastAPI

IMPLEMENTATION OF THE EVENT MANAGEMENT SYSTEM IN MICROSERVICE ARCHITECTURE

ABSTRACT

This paper is an exploration of the Eventella event management platform, with a focus on its system architecture and key components. The API Gateway within the microservices architecture is given special attention, with a detailed analysis of its capabilities and practical implementation of key functionalities such as authentication, request rate limiting, traffic routing, and information logging, using the FastAPI framework. The paper also provides an overview of popular cloud and on-premise solutions.

Integration with the Keycloak service is extensively described, with a focus on fundamental concepts, functionalities, and the utilization of the OpenID Connect protocol. Basic components of the Keycloak service are presented, and various integration methods with usage examples are discussed.

The paper also presents the implementation and demonstration of service functionalities in the background, explaining the role of each service to comprehend their contribution to the system. Through the analysis of these components, the paper provides insight into the technologies employed and current practices for building a secure and scalable event management system.

Keywords: microservices, API Gateway, Keycloak, Eventella, FastAPI

SADRŽAJ

1. Uvod.....	9
2. Eventella - platforma za upravljanje događajima.....	11
3. Arhitektura sistema.....	11
4. API Gateway.....	13
4.1. Autentifikacija.....	13
4.1.1. Osnovna autentifikacija API-ja.....	13
4.1.2. Autentifikacija putem ključa.....	13
4.1.3. LDAP autentifikacija.....	14
4.1.4. OAuth autentifikacija.....	14
4.1.5. OIDC autentifikacija.....	14
4.1.6. Implementacija autentifikacije na Eventella API Gateway-u.....	14
4.2. Ograničavanje broja zahteva.....	16
4.2.1. Token bucket.....	17
4.2.2. Leaky bucket.....	18
4.2.3. Algoritam fiksnog prozora.....	18
4.2.4. Algoritam kliznog prozora.....	18
4.2.5. Algoritam brojača kliznih prozora.....	19
4.2.6. Implementacija ograničavanja broja zahteva.....	20
4.3. Rutiranje zahteva.....	23
4.4. Keširanje.....	25
4.5. Analitika.....	26
4.6. Transformacija podataka.....	27
4.7. Upravljanje verzijama.....	28
4.8. Logovanje.....	29
4.9. Gotova rešenja.....	31
4.9.1. Cloud rešenja.....	31
AWS API Gateway.....	31
Azure API Gateway.....	31
Oracle API Gateway.....	32
4.9.2. On-premise rešenja.....	32
Kong Gateway.....	32
Tyk API Gateway.....	32
Express Gateway.....	33
5. Keycloak.....	33
5.1. Osnovi koncepti i terminologija.....	33
Korisnici.....	34
Autentifikacija.....	34
Autorizacija.....	34
Kredencijali.....	34
Uloge.....	34
Mapiranje uloge korisnika.....	34
Kompozitne uloge.....	34

Grupe.....	35
Realm.....	35
Klijenti.....	35
Klijentski adapteri.....	35
Klijentski šabloni.....	35
Saglasnost (Consent).....	35
Klijentska uloga.....	35
Identifikacioni token.....	35
Pristupni token.....	36
Tvrđnja.....	36
Servisni nalog.....	36
Direktno dodeljivanje.....	36
Protokolni maperi.....	36
Sesija.....	36
Provajder za udruživanje korisnika.....	36
Provajder identiteta.....	36
Udruživanje provajdera identiteta.....	36
Maperi provajdera identiteta.....	37
Obavezne radnje.....	37
Autentifikacioni tokovi.....	37
Događaji.....	37
Teme.....	37
5.2. Funkcionalnosti Keycloak servisa.....	37
5.2.1. Single-Sign On (SSO).....	37
5.2.2. Upravljanje identitetom i prijavljivanje preko društvenih mreža.....	38
5.2.3. Federacija korisnika.....	39
5.2.4. Administratorska konzola.....	39
5.2.5. Konzola za upravljanje nalogom.....	40
5.3. Korišćenje OpenID Connect-a za obezbeđivanje aplikacija i usluga.....	41
5.3.1. OpenID Connect.....	41
5.3.2. Dostupni endpointi.....	41
Well-known endpoint.....	41
Endpoint za autorizaciju.....	42
Token endpoint.....	42
Endpoint o korisničkim informacijama.....	42
Endpoint za odjavljivanje.....	43
Endpoint za sertifikate.....	43
Endpoint za introspekciju.....	43
Endpoint za dinamičku registraciju klijenta.....	43
Endpoint za poništavanje tokena.....	43
Endpoint za odobrenje uređaja.....	44
5.3.3. Podržani protokoli (Grant type).....	44
Authorization code.....	44
Implicit.....	46

Resource Owner Password Credentials.....	46
Client credentials.....	47
5.3.3. Primer integracije Keycloak servisa.....	48
5.3.4. Prosleđivanje informacija iz dekoriranog tokena ili prosleđivanje tokena.....	51
5.3.5. Personalizovanje teme login strane.....	52
5.3.6. Poređenje sa sličnim rešenjima na tržištu.....	53
6. Uloga ostalih mikroservisa.....	54
Zaključak.....	56
Literatura.....	57

1. Uvod

Poslednjih godina, evolucija softverskih arhitektura dovela je do sve učestalijeg korišćenja mikroservisnih sistema kao modernog pristupa razvoju i održavanju softvera. Mikroservisna arhitektura omogućava dekompoziciju kompleksnih aplikacija na manje, nezavisne servise, čime se povećava fleksibilnost, skalabilnost i agilnost u razvoju i održavanju softvera.

U srcu mikroservisne arhitekture nalaze se mikroservisi, nezavisne komponente koje obavljaju specifične zadatke i razmenjuju poruke putem mreže. Ova distribuirana priroda mikroservisnih sistema postavlja izazove u upravljanju komunikacijom među servisima, bezbednosti, skaliranju i upravljanju saobraćajem. Upravo u rešavanju ovih izazova, bitnu ulogu ima API Gateway.

API Gateway predstavlja centralizovanu tačku za upravljanje i kontrolu komunikacije između klijenata i mikroservisa u okviru mikroservisne arhitekture. Ova komponenta pruža niz ključnih funkcionalnosti, uključujući rutiranje zahteva, autentifikaciju, autorizaciju, ograničavanje broja zahteva, praćenje performansi i bezbednost. Kroz svoju ulogu, API Gateway olakšava integraciju, poboljšava bezbednost, optimizuje performanse i pruža jednostavan interfejs za klijente mikroservisa.

U ovakvim sistemima je jako bitna sigurnost korisnika tako da je sigurno upravljanje identitetom i pristupom postalo imperativ. Sistem često može da komunicira i integriše se sa više raznih servisa i resursa i sve to je potrebno osigurati na pravi način. Tradicionalne metode autentifikacije i autorizacije više nisu dovoljne da obezbede visok nivo bezbednosti i upravljanja u kompleksnom okruženju modernih aplikacija. Upravo ovde dolaze do izražaja servisi kao što je Keycloak - moćan alat za upravljanje identitetom i pristupom, čija integracija sa web servisima obezbeđuje kvalitetno rešenje za ove izazove.

Ovaj rad će istraziti, analizirati i prikazati praktičnu implementaciju API Gateway-a u mikroservisnim sistemima na primeru Eventella mikroservisne platforme za upravljanje događajima. Biće objašnjene njegove glavne funkcionalnosti sa teorijskog i tehničkog gledišta. Takođe biće reči i o rešenjima koje se nude na tržištu sa kratkim osvrtom na pojedinačne prednosti i mane tih rešenja. Takođe će posebna pažnja biti posvećena integraciji API Gateway-a i mikroservisa sa Keycloak sistemom.

Keycloak kao open-source i skalabilna platforma za upravljanje identitetom se može integrisati u mikroservisnoj arhitekturi kako bi se ostvarila sigurna autentifikacija i autorizacija korisnika. Kroz detaljan pregled Keycloak arhitekture, mehanizama autentifikacije i autorizacije, kao i praktičnih primera integracije, ovaj rad će pružiti uvid u različite načine na koje Keycloak može poslužiti kao centralna jedinica za upravljanje identitetom u složenim aplikacionim ekosistemima. Keycloak, sa svojim mogućnostima prilagođavanja, fleksibilnošću i podrškom za standarde kao što su OAuth 2.0 i OpenID Connect, predstavlja praktično rešenje za ove probleme. Mogućnost da centralizuje i standardizuje autentifikaciju i autorizaciju pruža mogućnost programerima da se usredsrede na razvoj funkcionalnosti aplikacija umesto na tehničke detalje sigurnosti.

Implementacija Eventella sistema sa API Gateway-om i Keycloak-om omogućava generalno rešenje koje prikazuje komunikacione i bezbednosne aspekte mikroservisne arhitekture. Kao svoj poseban doprinos ovom radu bih izdvojio implementaciju sopstvenog rešenja API Gateway-a u sve popularnijem FastAPI razvojnom okviru u kombinaciji sa jednim od najpopularnijih open-source servisa sa autentifikaciju kao što je Keycloak. Sve ukupno ovakva arhitektura sistema može biti primenjena na veliki broj softverskih rešenja danas.

2. EVENTELLA - PLATFORMA ZA UPRAVLJANJE DOGAĐAJIMA

Za demonstraciju tehničkog rešenja je razvijena platforma pod imenom Eventella. Sama platforma je razvijena u mikroservisnoj arhitekturi i ideja je da pruža podršku upravljanju događajima. Pored toga ideja je da sistem može da pruži usluge prodaje karata, pregled analitičkih podataka o upotrebi funkcionalnosti platforme kao i API koji bi mogli da koriste eksterni klijenti za kasniju integraciju sa platformom. Ideja je platformu koristiti za organizovanje sportskih događaja, muzičkih koncerata, predstava i raznih drugih događaja.

Zamišljeno je da platformi može da se pristupi putem veb aplikacije kao i preko mobilne aplikacije.

3. ARHITEKTURA SISTEMA

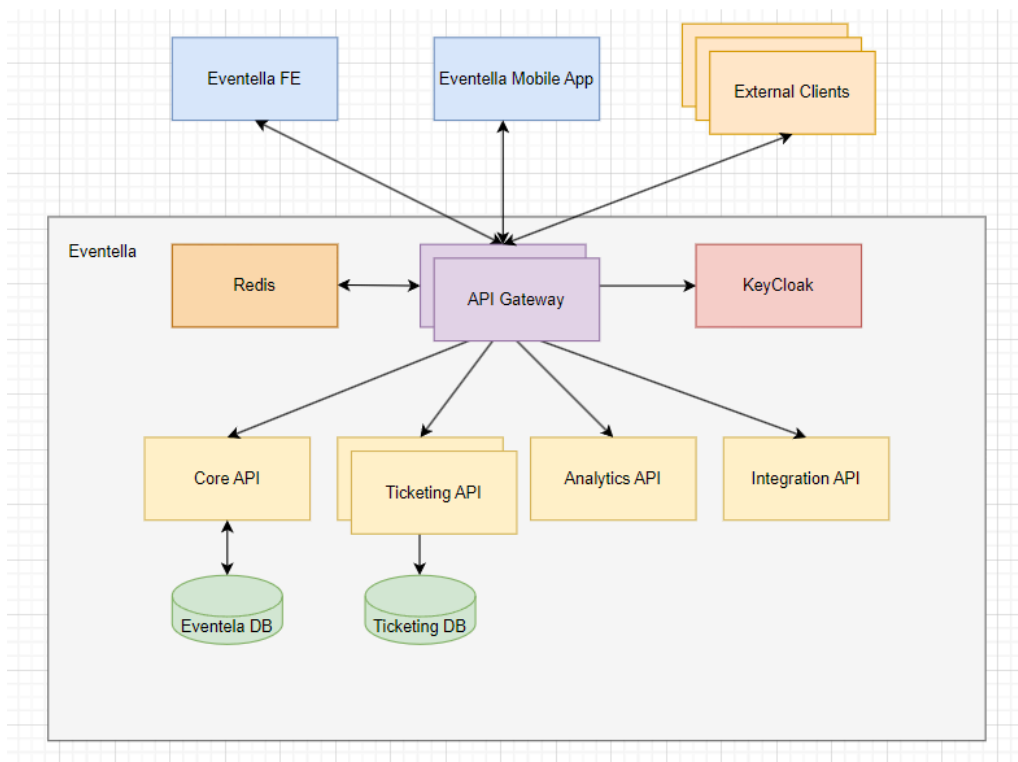
Na slici 1. je prikazana arhitektura Eventella platforme. Sastoji se iz četiri mikroservisa koja implementiraju biznis logiku platforme (Core API, Ticketing API, Analytics API i Integration API). Mikroservisi su razvijani u FastAPI frejmvorku u Python programskom jeziku. Osnovni i API za prodaju karata su povezani sa PostgreSQL bazama podataka koje se koriste za skladištenje podataka. Tim bazama podataka imaju pristup i servisi za obradu podataka i API koji mogu da koriste eksterni klijenti za integraciju sa Eventella sistemom.

Pored mikroservisa koji su zaduženi za obezbeđivanje glavnih funkcionalnosti sistema implementiran je API Gateway koji služi kao ulazna tačka ka platformi. API Gateway obezbeđuje jedinstvenu adresu preko koje se pristupa servisima na strani servera i vrši rutiranje u odnosu na segment URL-a koji se prosleđuje. API Gateway ima i sigurnosnu ulogu tako što implementira ograničavanje broja zahteva, a povezan je sa Keycloak servisom i vrši autentifikaciju zahteva. Na API Gateway-u se takođe beleže svi dolazni zahtevi.

Keycloak servis se koristi za autentifikaciju i autorizaciju i povezan je sa API Gateway-em i sa ostalim mikroservisima. Na API Gateway-u se vrši autentifikacija, a na ostalim mikroservisima autorizacija. Svaki od mikroservisa nakon prolaska zahteva kroz API Gateway dodatno validira i dekodira zahtev kako bi dobio informacije o korisniku koji pristupa servisu.

Redis je povezan takođe sa API Gateway-em i koristi se za čuvanje informacija prilikom implementiranja funkcionalnosti ograničavanja broja zahteva. U budućnosti proširenjem funkcionalnosti moguće je proširiti njegovu ulogu i koristiti ga i u drugim mikroservisima. Takođe, moguće je iskoristiti ga za keširanje odgovora koji se mogu keširati.

Eventella FE je zamišljen kao frontend deo aplikacije koji će koristiti sve mogućnosti koje Eventella platforma pruža. Pored frontend dela aplikacije zamišljeno je da postoji i mobilna aplikacija. Eventella kao SaaS rešenje ima mogućnost integracije i sa zainteresovanim partnerima preko integracionog API-a tako da je podržana mogućnost povezivanja eksternih klijenata.



Slika 1. Arhitektura Eventella sistema

Svi ovi servisi su u formi doker kontejnera i za pokretanje sistema je napravljen docker-compose.yml fajl gde je definisano koji image koristi svaki servis pojedinačno, promenljive okruženja(environment variables), portovi, komande za pokretanje i ostale specifičnosti.

```
services:
  eventella-keycloak:
    image: quay.io/keycloak/keycloak:20.0.3
    container_name: eventella-keycloak
    ports:
      - "8080:8080"
    environment:
      - KEYCLOAK_ADMIN=admin
      - KEYCLOAK_ADMIN_PASSWORD=admin
    command: "start-dev"

  eventella-postgres:
    image: postgres:15.2
    container_name: eventella-postgres
    mem_limit: 512m
    ports:
      - "5437:5432"
    environment:
      <<: *postgres-environment
    volumes:
      - ./data/db:/var/lib/postgresql/data

  eventella-app:
    mem_limit: 512m
    build:
      context: .
      dockerfile: Dockerfile
    depends_on:
      - "eventella-postgres"
    env_file:
      - ./app/.env.dev
    container_name: eventella-app
    ports:
      - "5679:5678"
      - "5007:80"
    command: "app/bin/runserver_vsc.sh"
    volumes:
      - ./app:/app

  eventella-gateway:
    mem_limit: 512m
    build:
      context: .
      dockerfile: Dockerfile.gateway
    env_file:
      - ./gateway/.env.dev
    container_name: eventella-gateway
    ports:
      - "5680:5678"
      - "5008:80"
    command: "gateway/bin/runserver_vsc.sh"
    volumes:
      - ./gateway:/gateway

  eventella-ticketing:
    mem_limit: 512m
    build:
      context: .
      dockerfile: Dockerfile.ticketing
    env_file:
      - ./ticketing/.env.dev
    container_name: eventella-ticketing
    ports:
      - "5681:5678"
      - "5009:80"
    command: "ticketing/bin/runserver_vsc.sh"
    volumes:
      - ./ticketing:/ticketing

  redis:
    mem_limit: 512m
    image: redis
    container_name: redis
    ports:
      - "6379:6379"
    volumes:
      - ./data:/data
```

Slika 2. Izgled docker-compose.yml fajla

4. API GATEWAY

Ukoliko se radi na aplikaciji koja uključuje više različitih vrsta klijenata i sastoji se od većeg broja mikroservisa, potrebno je uključiti API gateway unutar arhitekture. API gateway se pozicionira kao jedina ulazna tačka i nalazi se između strane klijenta i mikroservisa i funkcionira kao vrsta obrnutog proksija. Prosleđuje dolazne zahteve mikroservisima koji se nalaze iza njega. U zavisnosti od veličine aplikacije i količine dolaznog saobraćaja moguće je replicirati API Gateway u više instanci.

4.1. Autentifikacija

API predstavlja mehanizam za zahtevanje ili primanje podataka sa endpointa i vraćanje istih obično u bazu podataka. (1)

Endpointi mogu imati nekoliko različitih slučajeva korišćenja:

- Čovek koji kao krajnji korisnik pristupa API-ju putem veb aplikacije ili mobilne aplikacije.
- Hardverski uređaj koji šalje podatke putem API-ja vezane za razne uređaje povezane na internet (IoT).
- Zaposleni ili partner koji koristi interni API za slanje ili obradu podataka.

U svim ovim slučajevima, autentifikacija ima značajnu ulogu. Za spoljne API-je, uključujući one koje su orijentisane ka ljudima i IoT, ima smisla autentifikovati endpoint pre nego mu se dozvoli prenos podataka putem API-ja. Autentifikacija štiti od zlonamernih pristupa podacima, pomaže u upravljanju ukupnom količinom prenesenih podataka i pruža neophodan sloj sigurnosti za implementaciju kontrole pristupa, uključujući opciju za prekid pristupa kada pretplata istekne. Za interne API-je, autentifikacija možda deluje manje značajno, ali i dalje je važan korak u potvrđivanju da li su podaci koji se prenose ovlašćeni.

Autentifikacija takođe pruža mogućnost utvrđivanja kako se postupa s neuspehim zahtevom. To može značiti jednostavno blokiranje zahteva i vraćanje greške, ali u nekim okolnostima je moguće pružiti ograničen pristup.

4.1.1. Osnovna autentifikacija API-ja

Osnovna autentifikacija predstavlja jednostavan autentifikacioni mehanizam ugrađen u HTTP protokol. Pri osnovnoj autentifikaciji, klijent šalje HTTP zahtev sa korisničkim imenom i lozinkom enkodiranim u base64 formatu. U tom slučaju API gateway validira korisničko ime i lozinku i dozvoljava pristup ukoliko su prosleđeni kredencijali validni.

4.1.2. Autentifikacija putem ključa

Pri autentifikaciji putem API ključa, klijent dodaje jedinstveni ključ u zaglavlje zahteva ili kao URL parametar, a API gateway proverava da li je ključ validan. API ključevi mogu biti generisani na strani API-a ili od strane nekog eksternog servisa poput servisa za upravljanje tokenima. Ovaj pristup je koristan za HTTP API-je.

4.1.3. LDAP autentifikacija

LDAP (Lightweight Directory Access Protocol) je široko korišćeni protokol za čuvanje i upitivanje informacija o autentifikaciji. Sa LDAP autentifikacijom, API gateway može da validira klijentske kredencijale proverom prema LDAP serveru, koji deluje kao centralni repozitorijum informacija o korisnicima. Ovo može biti korisno u situacijama gde želite autentifikaciju klijenata prema postojećem korporativnom direktorijumu korisnika.

4.1.4. OAuth autentifikacija

OAuth 2.0 je široko korišćeni standard za delegiranje pristupa resursima. Sa OAuth 2.0, klijent dobija pristupni token od autorizacionog servera, a zatim uključuje taj token u svaki naredni zahtev ka API gateway-u. API gateway može zatim validirati token i odrediti nivo pristupa klijenta. Ovo može biti korisno u situacijama gde želite dati trećim aplikacijama ili uslugama ograničen pristup vašem API-ju.

4.1.5. OIDC autentifikacija

OpenID Connect (OIDC) je široko korišćeni standard izgrađen na osnovu OAuth 2.0. Pruža način autentifikacije klijenata i dobijanja informacija o korisniku u jednom zahtevu. Sa OpenID Connect, klijent dobija ID token od autorizacionog servera, pored pristupnog tokena, koji se može koristiti za autentifikaciju klijenta i pristup informacijama o korisniku. OpenID Connect može biti koristan u situacijama gde želite dobiti informacije o korisniku pored autentifikacije klijenata.

4.1.6. Implementacija autentifikacije na Eventella API Gateway-u

Za integraciju autentifikacija na Eventella platformi je iskorišćen python-keycloak paket kako bi se Keycloak servis integrisao sa FastAPI aplikacijom. To je paket koji pruža pristup Keycloak API-u. Za analizu načina funkcionisanja ove integracije koristiće se endpoint za rutiranje saobraćaja.

Na slici 3. se može videti metoda *gateway* koja implementira endpoint za rutiranje saobraćaja. U toj metodi se preko *dependency injection*-a poziva metoda *get_auth* koja proverava da li je korisnik koji želi pristupiti ovoj metodi ima pristup. Takođe, ova metoda kroz varijablu *current_user* dobija informacije o korisniku iz tokena.

```

@router.get("/{type}/{path:path}")
@router.post("/{type}/{path:path}")
@router.put("/{type}/{path:path}")
@router.delete("/{type}/{path:path}")
async def gateway(
    type: str,
    path: str,
    request: Request,
    current_user = Depends(get_auth),
):
    service_url = _get_service_url(type=type)
    url = f"{service_url}/{path}"
    logger.info(f"Request URL: {url}")
    headers = {key: value for key, value in request.headers.items()}
    try:
        async with httpx.AsyncClient() as client:
            if request.method == "GET":
                response = await client.get(url, headers=headers, params=request.query_params)
            elif request.method == "POST":
                response = await client.post(url, headers=headers, data=await request.body())
            elif request.method == "PUT":
                response = await client.put(url, headers=headers, data=await request.body())
            elif request.method == "DELETE":
                response = await client.delete(url, headers=headers)
            else:
                raise HTTPException(status_code=405, detail="Method not allowed")

        return Response(content=response.content, status_code=response.status_code, media_type="application/json")
    except httpx.HTTPError as e:
        raise HTTPException(status_code=e.response.status_code, detail=str(e))

def _get_service_url(type: str):
    if type == "core":
        return os.getenv("CORE_API_URL")
    elif type == "ticketing":
        return os.getenv("TICKETING_API_URL")
    else:
        raise HTTPException(
            status_code=400,
            detail='Provided service type does not exist!'
        )

```

Slika 3. Implementacija endpointa za rutiranje zahteva

U `auth.py` fajlu sa slike 5 nalazi se metoda `get_auth` i u nastavku je data analiza koda tog fajla. Metoda `get_auth` zahteva token kao paramtar, i u slučaju da nije prosleđen dobija se preko OAuth2 šeme za autentifikaciju. `oauth2_schema` se koristi za dobijanje OAuth2 tokena iz zahteva. Koristi se `OAuth2AuthorizationCodeBearer` sigurnosna šema iz FastAPI razvojnog okruženja za *Authorization Code Flow*. Parametri `authorizationUrl` i `tokenUrl` se dobijaju iz environment varijabli. `keycloak_openid` varijabla inicijalizuje `KeycloakOpenID` objekat za autentifikaciju. Parametri za server URL, ID klijenta, ime realma, tajni ključ klijenta i provera konfigurišu se takođe iz environment varijabli. Sama metoda `get_auth` prvo dobija javni ključ preko asinhronne metode `get_idp_public_key`, a zatim koristeći taj ključ vrši dekodiranje tokena. Tu se definišu i dodatni opcioni parametri za potpis tokena i vremena isteka. U slučaju bilo kakve greške prilikom autentifikacije, kreira se *HTTP* greška sa kodom *401 UNAUTHORIZED* i detaljnim opisom greške.

```

#KEYCLOAK ENV VARIABLES
AUTHORIZATION_URL=http://localhost:8080/realms/eventella/protocol/openid-connect/auth
TOKEN_URL=http://localhost:8080/realms/eventella/protocol/openid-connect/token
SERVER_URL=http://eventella-keycloak:8080
CLIENT_ID=eventella
REALM_NAME=eventella
CLIENT_SECRET_KEY=2WojwWUhbqpCE2XfCCIDxVQRRqGJFxxu

```

Slika 4. Prikaz promenljivih okruženja(environment variables)

```

# ./auth.py
from fastapi.security import OAuth2AuthorizationCodeBearer
from keycloak import KeycloakOpenID
from fastapi import Security, HTTPException, status, Depends
from pydantic import Json
from app.core.custom_logger import EventellaLogger
import os

logger = EventellaLogger(__name__)

# This is just for fastapi docs
oauth2_scheme = OAuth2AuthorizationCodeBearer(
    authorizationUrl=os.getenv('AUTHORIZATION_URL'),
    tokenUrl=os.getenv('TOKEN_URL')
)

# This actually does the auth checks
keycloak_openid = KeycloakOpenID(
    server_url=os.getenv('SERVER_URL'),
    client_id=os.getenv('CLIENT_ID'),
    realm_name=os.getenv('REALM_NAME'),
    client_secret_key=os.getenv('CLIENT_SECRET_KEY'),
    verify=True
)

async def get_idp_public_key():
    return (
        "-----BEGIN PUBLIC KEY-----\n"
        f"{keycloak_openid.public_key()}"
        "\n-----END PUBLIC KEY-----"
    )

async def get_auth(token: str = Security(oauth2_scheme)) -> J
    try:
        public_key = await get_idp_public_key()
        return keycloak_openid.decode_token(
            token,
            key= public_key,
            options={
                "verify_signature": True,
                "verify_aud": False,
                "exp": True
            }
        )
    except Exception as e:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail=str(e),
            headers={"WWW-Authenticate": "Bearer"},
        )

```

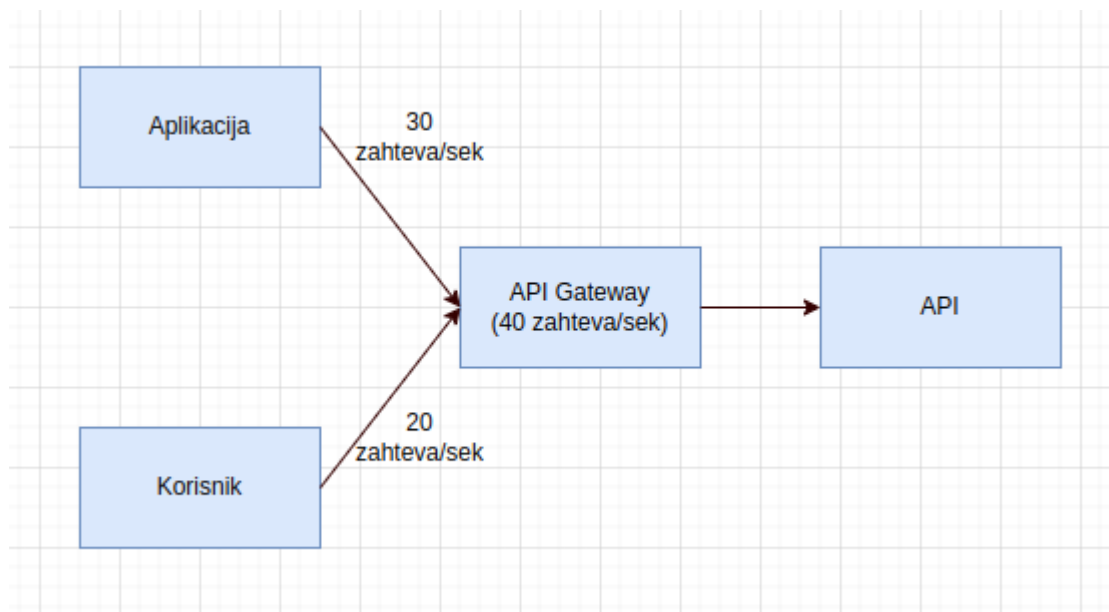
Slika 5. get_auth metoda

4.2. Ograničavanje broja zahteva

Ograničavanje broja zahteva (Rate limiting) predstavlja tehniku za kontrolu mrežnog saobraćaja sa ciljem sprečavanja korisnika da iscrpe sistemske resurse. Ograničavanje broja zahteva otežava zlonamerne subjekte da optereće sistem i izazovu napade poput DoS napada. Ovo uključuje napade u kojima napadači preplavljaju ciljni sistem zahtevima, čime zauzimaju previše mrežnog kapaciteta, skladišnog prostora i memorije.

API-ji koji koriste ograničavanje broja zahteva mogu usporavati ili privremeno blokirati svakog klijenta koji pokušava izvršiti previše API poziva. To može usporiti zahteve korisnika koji su pod ograničenjem u određenom vremenskom periodu ili ih potpuno odbiti. Ograničavanje broja zahteva osigurava da legitimni zahtevi mogu stići do sistema i pristupiti informacijama bez uticaja na performanse celokupne aplikacije.(2)

U nastavku će biti prikazani algoritmi koji se mogu iskoristiti za implementaciju ograničavanja broja zahteva.



Slika 6: Primer ograničenja broja zahteva

4.2.1. Token bucket

Token bucket algoritam je algoritam sa najboljim performansama. On je jednostavan, lako razumljiv i predstavlja dobar izbor za rate limiter-e koji se koriste u distribuiranim sistemima. Algoritam se može opisati na sledeći način:

- Token se dodaje u grupu(bucket) sa fiksnim korakom, recimo 1 token po sekundi.
- Grupa ima kapacitet od X tokena.
- Kada se zahtev napravi, jedan token se uklanja iz grupe
- Ako nema dostupnih tokena u grupi, vraća se statusni kod 429.

Prednosti ovog algoritma mogu biti to što je jednostavan za implementaciju i efikasan u pogledu upotrebe memorije. Nedostaci ovog algoritma su to što je potrebno postaviti dva parametra: stopa obnavljanja (refill rate) i veličinu grupe(bucket size).

4.2.2. Leaky bucket

Leaky bucket algoritam je sličan token bucket algoritmu, ali umesto dodavanja tokena u grupu, on uklanja tokene iz grupe. Algoritam se može opisati na sledeći način:

- Grupa ima kapacitet od x tokena, recimo 10 tokena.
- Kada zahtev stigne, dodajemo 1 token u grupu.
- Ako je grupa puna, odbijamo zahtev i vraćamo statusni kod 429.
- Ako grupa nije puna, dozvoljavamo zahtev i dodajemo 1 token iz grupe.
- Tokeni se uklanjaju sa fiksnom stopom od r tokena po sekundi. Recimo, 1 token po sekundi.

Prednosti ovog algoritma su to što je efikasan u pogledu upotrebe memorije, jednostavan je za implementaciju i zahtevi se obrađuju sa fiksnom stopom. Nedostaci ovog algoritma su to što je potrebno postaviti dva parametra: stopa odliva (outflow rate) i veličinu grupe(bucket size).

4.2.3. Algoritam fiksnog prozora

Algoritam fiksnog prozora (Fixed window counter) je najjednostavniji algoritam. Problem sa ovim algoritmom nastaje na granicama vremenskih prozora. Na primer, pretpostavimo da je brojač prazan u 10:59, i pristigne 10 zahteva. Ti zahtevi će biti obrađeni. U 11:00, stiže još 10 zahteva, koji će takođe biti prihvaćeni jer je brojač prazan. Ovaj scenario dovodi do kršenja limita broja zahteva i ograničenje od 10 zahteva po satu koje nije održano. Algoritam se može opisati na sledeći način:

- Vremenska linija se deli na fiksne vremenske prozore.
- Svaki vremenski prozor ima svoj brojač.
- Kada zahtev stigne, brojač za trenutni vremenski prozor se povećava.
- Ako je brojač veći od limita brzine, zahtev se odbacuje.
- Ako je brojač manji od limita brzine, zahtev se prihvata.

Prednosti su to što je efikasan u pogledu upotrebe memorije, jednostavan za implementaciju i novi zahtevi nisu uskraćeni od strane starih. Nedostaci su to što niz zahteva blizu kraja vremenskog prozora može premašiti limit broja zahteva.

4.2.4. Algoritam kliznog prozora

Algoritam fiksnog prozora (Fixed window counter) je dobar, ali ima problem sa dozvoljavanjem grupe zahteva da prođe oko kraja prozora. Algoritam kliznog prozora (Sliding window log) rešava ovaj problem pomeranjem prozora unapred prilikom svakog novog zahteva. Na ovaj način možemo postići preciznije ograničavanje brzine. Algoritam se može opisati na sledeći način:

- Kreirati keš za vremenske oznake, na primer, u Redis-u.

- Kada stigne nov zahtev, uklanjaju se sve zastarele vremenske oznake iz keša. Pod zastarelim se podrazumevaju vremenske oznake koje su starije od veličine prozora.
- Dodaje se nova vremenska oznaka u keš.
- Ako je broj vremenskih oznaka u kešu veći od limita, odbacuje se zahtev i vraća se statusni kod 429.
- Ako je manji od limita, prihvata se zahtev i vraća se statusni kod 200.

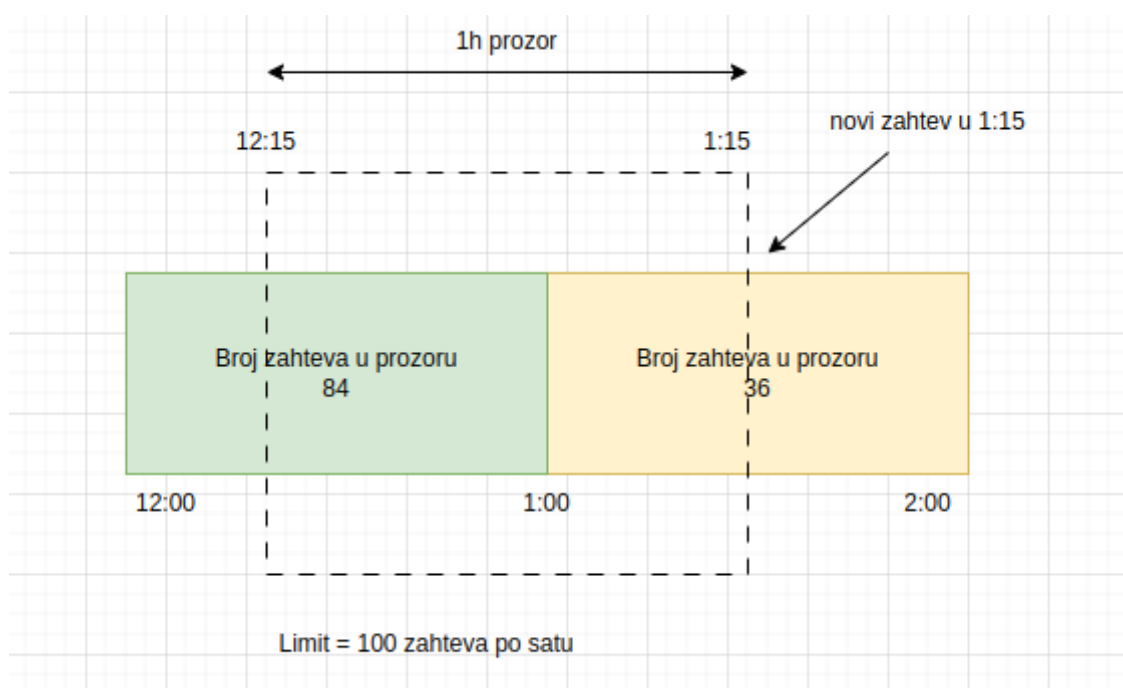
Prednosti ovog algoritma su to što je veoma precizno ograničavanje broja zahteva. Nedostaci su to što je kompleksnija implementacija i upotreba memorije linearno raste sa vremenom.

4.2.5. Algoritam brojača kliznih prozora

Brojač kliznih prozora je algoritam koji predstavlja kombinaciju fiksnog brojača prozora i algoritma kliznog prozora. Pomoću ovog algoritma je moguće ublažiti pikove u broju zahteva i istovremeno pratiti saobraćaj u poslednjih n minuta. Algoritam se može opisati na sledeći način:

- Imamo prozor veličine N minuta (npr. N=1).
- U prvih 10 sekundi stiže 5 zahteva. [0:00–0:10].
- U vremenu [01:15] stiže još 10 zahteva.
- Maksimalan broj zahteva se izračunava pomoću sledeće formule:

$$\text{Broj zahteva u trenutnom prozoru} + \text{Broj zahteva u prethodnom prozoru} * \text{Procenat preklapanja između kliznog i prethodnog prozora}$$
- Rezultat se zaokružuje na najbliži ceo broj.



Slika 7: Prikaz brojača kliznih prozora^[OBJ]

Prednosti ovog algoritma su to što je efikasan u pogledu upotrebe memorije i ublažava pikove u saobraćaju. Nedostaci ovog algoritma su to što je metoda efikasna samo kada prozor posmatranja nije previše strog. Pruža procenu stvarnog broja zahteva, pretpostavljajući da su zahtevi unutar prethodnog prozora ravnomerno raspoređeni.

4.2.6. Implementacija ograničavanja broja zahteva

Na API Gateway-u je odabran način za implementiranje ograničenja broja zahteva preko algoritma kliznog prozora. Implementira se klasa `RateLimiter` koja prilikom inicijalizacije kreira vezu ka Redis bazi podataka. Stvara se `redis_pool` objekat koristeći `ConnectionPool` iz redis modula, što omogućava efikasno upravljanje konekcijom prema Redis bazi podataka.

Metoda `get_redis` vraća instancu `Redis` objekta iz `redis_pool` objekta i omogućava da više puta možemo iskoristiti postojeću konekciju ka Redis bazi.

```
class RateLimiter:
    def __init__(self, redis_host: str, redis_port: int):
        self.redis_pool = redis.ConnectionPool(host=redis_host, port=redis_port, db=0, decode_responses=True)

    def get_redis(self):
        return redis.Redis(connection_pool=self.redis_pool)

    def is_rate_limited(self, key: str, max_requests: int, window: int) -> bool:
        current = int(time.time())
        window_start = current - window
        redis_conn = self.get_redis()
        with redis_conn.pipeline() as pipe:
            try:
                pipe.zremrangebyscore(key, 0, window_start)
                pipe.zcard(key)
                pipe.zadd(key, {current: current})
                pipe.expire(key, window)
                results = pipe.execute()
            except redis.RedisError as e:
                raise HTTPException(
                    status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
                    detail=f"Redis error: {str(e)}"
                ) from e
        return results[1] >= max_requests
```

Slika 8: Prikaz brojača kliznih prozora

Glavna logika je sadržana u `is_rate_limited` metodi. Ova metoda proverava da li je ograničenje broja zahteva postignuto za određeni ključ. U ovom slučaju je ključ kombinacija IP adrese i rute kojoj se pristupa. Metoda definiše navođenje tri parametara. Prvi je ključ, drugi je maksimalni broj zahteva koji je dopušten u određenom vremenskom prozoru i treći predstavlja vremenski prozor izražen u sekundama. Algoritam koristi klizni prozor kako bi izbrisao stare zahteve i proverio broj zahteva unutar trenutnog prozora.

Unutar metode, trenutno vreme se dobija pozivom `time()` metode, a zatim se izračunava početak prozora oduzimanjem vrednosti prozora od trenutnog vremena. Uz pomoć `pipeline` objekta pristupa se Redis bazi i izvršavaju se sledeće operacije:

- `zremrangebyscore`: uklanja elemente iz sortirane set strukture (zset) s vremenskim oznakama, čije su vremenske oznake manje od početka prozora.
- `zcard`: vraća broj elemenata u sortiranoj set strukturi.
- `zadd`: dodaje novi element u sortiranu set strukturu s trenutnim vremenom.
- `expire`: postavlja vreme nakon kojeg će ključ biti automatski uklonjen iz baze podataka.

Ako dođe do greške prilikom izvršavanja operacija vraća se HTTP odgovor sa kodom greške 500 (interna serverska greška) i odgovarajućim detaljima o grešci.

Na kraju se proverava da li je broj zahteva u prozoru veći ili jednak *max_requests*. Ako jeste, metoda vraća *True*, što znači da je ograničenje broja zahteva dostignuto. Inače, vraća se *False*, što znači da ima još dostupnih zahteva.

```
class RateLimiterMiddleware(BaseHTTPMiddleware):
    def __init__(self, app: FastAPI):
        super().__init__(app)
        self.rate_limiter = RateLimiter("redis", 6379)

    async def dispatch(self, request: Request, call_next):
        max_requests = os.getenv("RATE_LIMITING_MAX_REQUESTS")
        window = os.getenv("RATE_LIMITING_WINDOW")

        key = f"rate_limit:{request.client.host}:{request.url.path}"
        if self.rate_limiter.is_rate_limited(key, int(max_requests), int(window)):
            return JSONResponse({"message": "Too many requests"}, status_code=429)

        # Continue handling the request
        response = await call_next(request)

        return response
```

Slika 9: Prikaz middleware klase za ograničavanje broja zahteva

Za ograničavanje broja zahteva implementirana je middleware klasa *RateLimiterMiddleware*. Klasa prilikom pokretanja servisa instancira instancu *RateLimiter* klase. Pre obrade bilo kog zahteva na API Gateway-u formira se ključ koji se koristi za proveravanje da li je postignuto ograničenje broja zahteva pod tim ključem. U ovom slučaju je implementirano ograničenje broja zahteva po kombinaciji IP adrese i putanje HTTP zahteva. To znači da svaki korisnik ima pravo pristupiti *max_requests* puta svakoj putanji u *window* periodu. *max_requests* i *window* vrednosti se čitaju iz environment promenljivih. Ukoliko je postignut postavljeni limit broja zahteva API Gateway će vratiti klijentu HTTP odgovor sa porukom da je poslao previše zahteva i status kod 429.

Middleware klasa se aktivira korišćenjem metode *add_middleware* iz FastAPI frejmworka prilikom konfiguracije same aplikacije i pre njenog pokretanja.

Bitno napomenuti jeste da će funkcionalnost ograničavanja broja zahteva raditi ukoliko imamo i više instanci API Gateway-a zato što se za memorisanje informacija o zahtevima koristi Redis. U tom slučaju će nebitno na kojoj instanci dođe zahtev biti ispravno validiran i procesiran. To ne bi bio slučaj da je umesto Redisa korišćen in-memory pristup za skladištenje informacija na API Gateway-u. U tom slučaju bi svaka instanca vodila nezavisnu evidenciju o broju zahteva u vremenskom prozoru.

```
def configure():
    app.include_router(api_router)

    app.add_middleware(
        CORSMiddleware,
        allow_origins=origins,
        allow_methods=["*"],
        allow_headers=["*"],
    )
    app.add_middleware(RateLimiterMiddleware)

configure()

if __name__ == "__main__":
    uvicorn.run(app)
```

Slika 10: Konfiguracija middleware klase za ograničavanje broja zahteva

Name	Status	Type	Initiator	Size	Time	Waterfall	
resource%2Fall	200	fetch	actions.js:470	333 B	71 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	46 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	53 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	39 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	47 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	48 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	84 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	51 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	47 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	47 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	81 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	42 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	49 ms		
resource%2Fall	200	fetch	actions.js:470	333 B	54 ms		
resource%2Fall	429	fetch	actions.js:470	171 B	9 ms		
resource%2Fall	429	fetch	actions.js:470	171 B	6 ms		
resource%2Fall	429	fetch	actions.js:470	171 B	10 ms		
resource%2Fall	429	fetch	actions.js:470	171 B	7 ms		

40 requests | 12.7 kB transferred | 7.6 kB resources

Slika 11: Primer vraćanja statusa 429 nakon dostizanja limita

4.3. Rutiranje zahteva

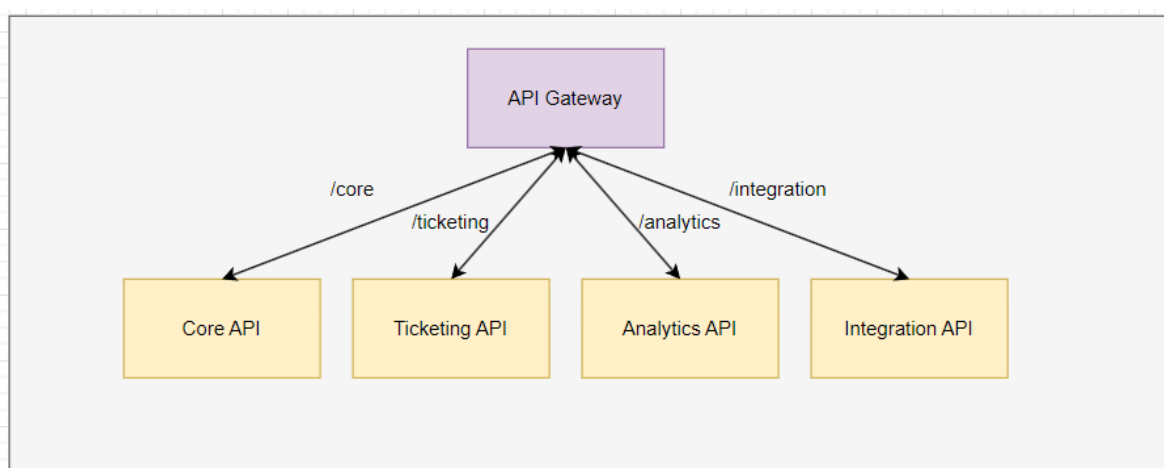
Tipičan slučaj korišćenja ove funkcionalnosti je kada određeni tim želi jednostavnu arhitekturu i gde je dovoljno da programer zapamti samo jedan URL, npr. `api.example.com`, kako bi interagovao sa određenim API servisom. Dokumentacija API-ja je često lakša za razumevanje jer se čuva na jednom mestu umesto da bude razdeljena po različitim portalima ili PDF-ovima.

Rutiranje putem putanja smatra se jednostavnim mehanizmom za deljenje HTTP API-ja. Međutim, uključuje dodatni posao poput konfiguracije, autorizacije, integracija i dodatne latencije usled više skokova. Takođe zahteva definisane procese upravljanja promenama kako bi se obezbedilo da netačna konfiguracija ne poremeti sve usluge.

Takođe, čest zahtev je rutiranje zahteva poslatih istom API gateway-u ka različitim pozadinskim sistemima, na osnovu elemenata u zahtevu. To može biti na osnovu: (3)

- Hosta, domena, poddomena ili segmenta URL-a kome je zahtev poslat.
Primer: usmeravanje dolaznih zahteva sa `/core` i `/ticketing` delovima URL-a koji su poslani istom API gateway-u ka potpuno različitim backend sistemima.
- Paketu usluge prema pretplati korisnika.
Primer: usmeravanje dolaznih zahteva ili ka standardnom hostu za API klijente pretplaćene na besplatan paket usluge ili ka hostu sa boljim performansama za API klijente pretplaćene na premium paket usluga.
- Zaglavlja i vrednostima zaglavlja u zahtevu.
Primer: rutiranje zahteva koji sadrži `Accept` zaglavlje sa vrednošću `application/xml` ka odgovarajućem backend sistemu koji vraća odgovor tog sadržaja.

Dinamički izbor pozadinskog sistema omogućava da se predstavi jedinstvena fasada potrošačima API-ja i zaštitite ih od kompleksnosti višestrukih backend sistema.



Slika 12: Primer rutiranja preko API Gateway-a

Na primeru API Gateway-a u Eventella sistemu je rutiranje izvršeno na osnovu segmenta URL-a.

API Gateway implementira jedan generički endpoint koji podržava sve tipove zahteva. Njegova uloga je da iz URL putanje izvuče informaciju o tome kom servisu treba da prosledi zahtev i na kojoj putanji. Na osnovu imena servisa preko metode `_get_service_url` dobija se adresa servisa kome zahtev treba da se prosledi. Formira se kompletan URL kombinovanjem adrese servisa i ostatka putanje dobijenog zahteva. Kopiraju se svi hederi koji su došli sa zahtevom. Na osnovu tipa zahteva formira se novi zahtev i preko `httpx` biblioteke šalje na odgovarajući servis u pozadini. Ukoliko je reč o GET zahtevu dodaju se i parametri putanje, a ukoliko je reč o POST ili PUT tipu zahteva dodaje se i telo zahteva koje je bilo u inicijalnom zahtevu.

Nakon dobijanja odgovora od servisa iz pozadine vraća se klijentu telo odgovora sa status kodom koji je došao.

```
@router.get("/{type}/{path:path}")
@router.post("/{type}/{path:path}")
@router.put("/{type}/{path:path}")
@router.delete("/{type}/{path:path}")
async def gateway(
    type: str,
    path: str,
    request: Request,
    current_user = Depends(get_auth),
):
    service_url = _get_service_url(type=type)
    url = f"{service_url}/{path}"
    logger.info(f"Request URL: {url}")
    headers = {key: value for key, value in request.headers.items()}
    try:
        async with httpx.AsyncClient() as client:
            if request.method == "GET":
                response = await client.get(url, headers=headers, params=request.query_params)
            elif request.method == "POST":
                response = await client.post(url, headers=headers, data=await request.body())
            elif request.method == "PUT":
                response = await client.put(url, headers=headers, data=await request.body())
            elif request.method == "DELETE":
                response = await client.delete(url, headers=headers)
            else:
                raise HTTPException(status_code=405, detail="Method not allowed")

        return Response(content=response.content, status_code=response.status_code, media_type="application/json")
    except httpx.HTTPError as e:
        raise HTTPException(status_code=e.response.status_code, detail=str(e))

def _get_service_url(type: str):
    if type == "core":
        return os.getenv("CORE_API_URL")
    elif type == "ticketing":
        return os.getenv("TICKETING_API_URL")
    else:
        raise HTTPException(
            status_code=400,
            detail='Provided service type does not exist!'
        )
```

Slika 13. Implementacija endpointa za rutiranje zahteva

API Gateway 0.1.0 OAS3

/openapi.json

API Gateway with example of authentication and routing

Authorize



base



GET

/ {type} / {path} Gateway



PUT

/ {type} / {path} Gateway



POST

/ {type} / {path} Gateway



DELETE

/ {type} / {path} Gateway



default



GET

/ Root



Slika 14: Prikaz strukture putanja na API Gateway-u

4.4. Keširanje

Cilj keširanja je obično smanjiti nepotrebno opterećenje servisa u pozadini kako bi se poboljšale performanse i smanjili troškovi. Jedan od načina za smanjenje tog opterećenja je keširanje odgovora zahteva, ukoliko se odgovori mogu ponovo koristiti u budućnosti. Ukoliko dođu takvi zahtevi, oni mogu biti zadovoljeni dobijanjem podataka iz keširanog odgovora umesto slanja zahteva servisu u pozadini.

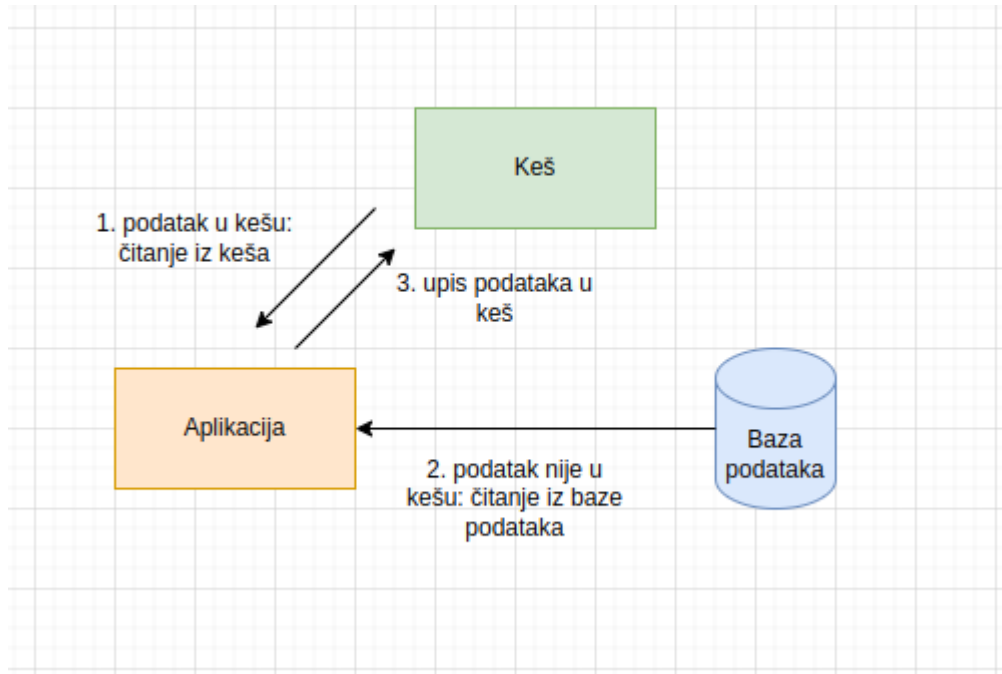
API Gateway se može integrisati sa eksternim keš serverom do kog je potrebno omogućiti pristup. Primer takvog servisa može biti Redis. Tako konfigurisani API Gateway može:

- Čuvati podatke koji su vraćani od strane pozadinskog servisa u keš serveru i ti podaci mogu služiti kao odgovor na originalni zahtev.
- Dobiti prethodno sačuvane podatke iz keš servera kao odgovor na zahtev koji je sličan originalnom zahtevu, bez slanja zahteva pozadinskom servisu.

Prednosti kešinga:

- Poboljšanje performansi aplikacije: korišćenje keširanja može poslužiti kao mehanizam za poboljšanje performansi sistema i smanjenje latencije API-ja.
- Smanjenje troškova baze podataka: keširanje može preusmeriti određeni saobraćaj ka keš serveru i na taj način smanjiti broj upita ka bazi podataka, što u krajnjem slučaju dovodi do smanjenja troškova baze podataka.

- Smanjenje opterećenja servisa u pozadini: umesto opterećenja pozadinskog servisa ukoliko se odgovor nalazi u kešu biće vraćan klijentu direktno iz keša bez aktiviranja servisa u pozadini.
- Povećanje protoka čitanja: keš server mnogo brže odgovara na upite što povećava protok čitanja.



Slika 15: Način rada mehanizma za keširanje

4.5. Analitika

Jedna od funkcionalnosti API Gateway-a je i prikupljanje informacija o zahtevima i odgovorima. Neke od najčešćih metrika koje je moguće pratiti su:

- 4XX greške:
Broj grešaka sa klijentske strane zabeležene u određenom periodu. API Gateway može brojati status kodove odgovora gateway-a koje su oblika 4XX. Moguće je predstaviti kao sumu tih grešaka u određenom periodu ili kao prosek u odnosu na ukupan broj zahteva.(4)
- 5XX greške
Broj grešaka sa serverske strane zabeležene u određenom periodu. API Gateway može brojati status kodove odgovora gateway-a koje su oblika 5XX. Moguće je predstaviti kao sumu tih grešaka u određenom periodu ili kao prosek u odnosu na ukupan broj zahteva.(4)
- Broj odgovora dobijenih iz keša
Broj odgovora dobijenih iz keša API Gateway-a. Takođe je moguće dobiti sumu ili prosek broja odgovora u određenom intervalu.

- Broj odgovora dobijenih iz pozadinskih servisa dok je keš aktiviran
Moguće je dobiti sumu ovakvih odgovora ili prosek odgovora u određenom intervalu.
- Broj ukupnih zahteva
- Integraciono vreme
Vreme između trenutka kada API Gateway prosledi zahtev servisu u pozadini i trenutka kada primi odgovor od tog sistema. Jedinica mere može biti milisekunda.
- Ukupno vreme
Vreme između trenutka kada API Gateway primi zahtev od klijenta i trenutka kada vrati odgovor klijentu. Ukupno vreme uključuje vreme integracije i ostale dodatno vreme koje se izgubi tokom obrađivanja zahteva na API Gateway-u. Merna jedinica ove metrike je takođe milisekunda.

4.6. Transformacija podataka

Transformacija podataka kod API Gateway-a se odnosi na proces modifikacije ili preoblikovanja podataka između klijenta i pozadinskih servisa kako bi se prilagodili ili optimizovali za specifične zahteve ili potrebe. Ova transformacija može uključivati promenu formata, strukture ili vrednosti podataka.

Primeri transformacije podataka kod API Gateway-a uključuju:

- Promena formata podataka:
Originalni zahtev klijenta može sadržavati podatke u jednom formatu (na primer, JSON), dok backend sistemi očekuju drugi format (na primer, XML). API Gateway može izvršiti transformaciju formata podataka između ova dva oblika kako bi se obezbedila kompatibilnost.

{	<request>
"user": {	<user>
"username": "john_doe",	<username>john_doe</username>
"password": "securepass123"	<password>securepass123</password>
},	</user>
"data": {	<data>
"info": "Some additional information"	<information>Some additional
}	information</information>
}	</data>
	</request>

slika 16: Primer tela zahteva pre i posle prolaska kroz API Gateway

- Dodavanje ili uklanjanje podataka:
API Gateway može dodati dodatne informacije u zahtevu pre nego što ga prosledi pozadinskom sistemu. Na primer, dodavanje HTTP zaglavlja sa informacijama o

autentifikaciji ili informacijama o korisniku. Takođe, može ukloniti ili očistiti određene informacije iz odgovora pre nego što ih vrati klijentu.

- Spajanje podataka sa različitih servisa:
Ako se podaci nalaze na različitim servisima API Gateway može agregirati ove podatke i vratiti ih klijentu u jednom, zajedničkom odgovoru. Ovo smanjuje broj zahteva koje klijent mora da napravi kako bi dobio sve potrebne informacije.
- Enkripcija i dekripcija podataka:
U slučaju komunikacije preko sigurnih protokola, API Gateway može enkriptovati podatke prilikom slanja zahteva klijenta ili dekriptovati podatke prilikom prijema odgovora od backend sistema.
- Promena putanje i parametara:
Ako backend sistem očekuje specifičnu putanju ili parametre u zahtevu, a klijent šalje podatke na drugačiji način, API Gateway može transformisati putanju ili parametre kako bi odgovarali očekivanjima sistema u pozadini.

Transformacija podataka kod API Gateway-a omogućava fleksibilnost u komunikaciji između klijenta i backend sistema, obezbeđujući da podaci budu u odgovarajućem obliku i strukturi.

4.7. Upravljanje verzijama

U mikroservisnoj arhitekturi, gde više servisa međusobno komunicira putem API-ja, verzionisanje API-ja je jako značajno kako bi se obezbedila stabilnost i izbegle promene koje mogu uticati na celokupni sistem.

Razlog za postojanje verzija API-ja u mikroservisnoj arhitekturi je mogućnost ažuriranja API-ja tokom vremena bez izazivanja problema za postojeće korisnike. Kako se dodaju novi servisi ili modifikuju postojeći, API može zahtevati izmene. Bez verzionisanja, ove promene bi potencijalno mogle izazvati probleme za postojeće korisnike, dovodeći do vremenskih prekida i drugih problema.

Kreiranje više verzija API-ja omogućava programerima da vrše ažuriranja, istovremeno osiguravajući da postojeći korisnici i dalje mogu koristiti staru verziju. Ova fleksibilnost koristi kako programerima tako i korisnicima, omogućavajući razvoj novih verzija API-a tokom vremena bez narušavanja trenutne funkcionalnosti.

Načini navođenja verzije API-a(5):

- Navođenje verzije kroz URL putanju:
Ovaj način navođenja verzije podrazumeva navođenje verzije kao deo URL putanje. API Gateway će izvući verziju iz putanje i izvršiti rutiranje na zahtevanu verziju API-a. Primer: <http://www.example.com/api/1/product>
- Navođenje verzije putem parametra upita
Jednostavan način implementacije koji rešava problem verzionisanja API-ja. Tehnički je jednostavniji od navođenja verzije kroz URL putanju.

Primer: <http://www.example.com/api/products?version=1>

- Verzionisanje putem zaglavlja zahteva
U ovom slučaju, verzija se navodi dodavanjem zaglavlja sa brojem verzije kao atributom. Na ovaj način nije potrebno menjati oblik URL-a.
Primer: curl -H "Accepts-version: 1.0" <http://www.example.com/api/products>

Zaduženja API Gateway-a:

- Transformacija zahteva:
Kada postoje različite verzije API-ja, zadatak API Gateway-a je da rutira zahteve navedenoj verziji servisa.
- Kompatibilnost prethodnih verzija:
Moguće je određene klijente zadržati na prethodnu verziju. API Gateway će prepoznati kom klijentu je potrebno servirati koju verziju servisa.
- Mogućnost povratka na prethodnu verziju:
Ukoliko postoje problemi sa novom verzijom servisa, API Gateway može brzo prerutirati zahteve na prethodnu stabilnu verziju.

4.8. Logovanje

API Gateway implementacije često pružaju mogućnost logovanja različitih vrsta podataka kako bi omogućile praćenje i analizu rada sistema, detekciju problema, bezbednosno praćenje i poboljšanje performansi. Uobičajene vrste podataka koje se obično loguju su:

- Zahtevi i odgovori:
Logovanje zahteva i odgovora podrazumeva logovanje podataka o ulaznim zahtevima, uključujući tip HTTP metode, putanju, zaglavlja, parametre i telo zahteva. Mogu se logovati i informacije o izlaznim odgovorima, uključujući status kod, zaglavlja i telo odgovora.
- Vremena odgovora:
Logovanje vremena odgovora podrazumeva logovanje vremena obično izraženog u milisekundama koje se potroši na obradu zahteva i generisanje odgovora. Može biti korisno za praćenje performansi sistema i za identifikaciju problematičnih zahteva u sistemu.
- Logovanje klijentskih podataka:
Logovanje klijentskih podataka može uključivati klijentsku IP adresu ili druge klijentski vezane informacije koje se mogu iskoristiti za klijentsku identifikaciju.
- Greške i izuzeci:
Mogu se logovati detalji o greškama, izuzecima ili problemima koji se jave prilikom ograde zahteva. Mogu se logovati vrlo detaljno navodeći vrstu greške, stek poziva i dodatne informacije koje bi mogle pomoći u dijagnostici problema.

- Podaci o bezbednosti:
Podrazumeva neuspešne pokušaje autentifikacije ili pokušaje neovlašćenog pristupa.
- Metrike i analitika:
Podrazumeva logovanje podataka o performansama, poput broja zahteva, vremena odgovora, i slično, koji se koriste za analizu i praćenje performansi i poslovanja.

Eventella API Gateway ima implementiran logger koji vrši logiranje osnovnih informacija o svakom zahtevu. Svrha ovog logera je prikupljanje informacija o dolaznim zahtevima, prikrivanje osetljivih podataka pre beleženja, i beleženje grešaka ako se dese tokom obrade zahteva. Ovi podaci mogu biti korisni tokom debugovanja, praćenja saobraćaja i drugih metrika prilikom rada aplikacije.

```
class LogRoute(APIRoute):
    def get_route_handler(self):
        handler = super().get_route_handler()

        async def log_route_handler(request):
            params = {**request.path_params, **request.query_params}
            self._hide_sensitive_data(params)
            logger.info(f'{self.endpoint.__name__} - '
                       f'start - '
                       f'{params}')

            try:
                response = await handler(request)
            except Exception as e:
                logger.error(f'{self.endpoint.__name__} - '
                           f'failed ({repr(e)}) - '
                           f'{params}')
                raise
            else:
                logger.info(f'{self.endpoint.__name__} - '
                           f'finish - '
                           f'{params}')
                return response

        return log_route_handler

    def _hide_sensitive_data(self, data):
        sensitive_keys = ('username', 'password', 'refresh_token', 'access_token', 'token')
        for key, value in data.items():
            if isinstance(value, abc.Mapping):
                self._hide_sensitive_data(value)
            else:
                if key in sensitive_keys:
                    data[key] = 'obfuscated'
```

slika 17: implementacija LogRoute klase

Ovaj logger je implementiran na način da nasleđuje klasu *APIRoute*. Vršiti override metode *get_route_handler* iz roditeljske klase. U *get_route_handler* metodi se definiše nova asinhrona metoda *log_route_handler* koja obuhvata originalni handler rute. Pre pozivanja originalnog hendlera rute loguje informacije o dolazećem zahtevu, uključujući ime endpointa i parametre. Nakon izvršenja hendlera loguju se informacije o izvršenom zahtevu ili grešci ukoliko je došlo do greške prilikom izvršenja zahteva. U sklopu ove klase se nalazi i pomoćna metoda *_hide_sensitive_data* koja proverava rečnik sa parametrima i vrši sakrivanje osetljivih podataka ukoliko postoje. Klasa se kasnije uključuje u definisani ruter aplikacije.

```

eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 200 OK
eventella-gateway | 2024-04-04 20:53:11,911 - gateway.routers.gateway - INFO - Request URL: http://eventella-app:80/resource/all
eventella-app | INFO: 172.18.0.5:37944 - "GET /resource/all HTTP/1.1" 200 OK
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 200 OK
eventella-gateway | 2024-04-04 20:53:12,115 - gateway.routers.gateway - INFO - Request URL: http://eventella-app:80/resource/all
eventella-app | INFO: 172.18.0.5:37946 - "GET /resource/all HTTP/1.1" 200 OK
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 200 OK
eventella-gateway | 2024-04-04 20:53:12,334 - gateway.routers.gateway - INFO - Request URL: http://eventella-app:80/resource/all
eventella-app | INFO: 172.18.0.5:37960 - "GET /resource/all HTTP/1.1" 200 OK
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 200 OK
eventella-gateway | 2024-04-04 20:53:12,568 - gateway.routers.gateway - INFO - Request URL: http://eventella-app:80/resource/all
eventella-app | INFO: 172.18.0.5:37964 - "GET /resource/all HTTP/1.1" 200 OK
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 200 OK
eventella-gateway | 2024-04-04 20:53:12,807 - gateway.routers.gateway - INFO - Request URL: http://eventella-app:80/resource/all
eventella-app | INFO: 172.18.0.5:37978 - "GET /resource/all HTTP/1.1" 200 OK
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 200 OK
eventella-gateway | 2024-04-04 20:53:13,015 - gateway.routers.gateway - INFO - Request URL: http://eventella-app:80/resource/all
eventella-app | INFO: 172.18.0.5:37988 - "GET /resource/all HTTP/1.1" 200 OK
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 200 OK
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 429 Too Many Requests
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 429 Too Many Requests
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 429 Too Many Requests
eventella-gateway | INFO: 172.18.0.1:38528 - "GET /core/resource/all HTTP/1.1" 429 Too Many Requests

```

slika 18: izgled log poruka

4.9. Gotova rešenja

Prilikom odabira API Gateway-a za svoje rešenje je potrebno razmotriti više tehničkih aspekta. Potrebno je razmisliti da li će rešenje biti hostovano u cloud okruženju ili će biti on premise rešenje. U zavisnosti od toga postoje već gotove implementacije API Gateway-a i obično nije potrebno razvijati rešenje od nule. Neka od poznatih cloud rešenja koje se nude na tržištu su AWS API Gateway, Google Cloud API Gateway i Azure API Management dok on premise rešenja mogu biti Kong Gateway, Tyk API Gateway i Express Gateway.

Obično najčešće funkcionalnosti koje se koriste su već implemntirane kod ovih popularnih opcija ali je naravno moguće implementirati i svoje rešenje ukoliko postojeća ne zadovoljavaju sve potrebe ili je možda cena neodgovarajuća.

4.9.1. Cloud rešenja

AWS API Gateway

Prednosti:

- Laka integracija sa ostalim AWS servisima
- Automatsko skaliranje
- Fleksibilnost protokola: REST, HTTP i WebSocket-i
- Lako podešavanje autorizacije i autentifikacije
- Lako se integriše sa serverless arhitekturom (posebno sa Lambda funkcijama)

Mane:

- Cena može postati visoka sa povećanim obimom saobraćaja
- Za neke od napredne funkcionalnosti konfiguracija može biti kompleksna

Azure API Gateway

Prednosti:

- Integracija sa ostalim Azure servisima je glatka
- Nudi različite cenovne nivoe prilagođene potrebama korisnika
- Omogućava kreiranje serverless API-ja
- Pruža niz bezbednosnih funkcija, uključujući i integraciju sa Active Directory-em.

Mane:

- Može biti manje fleksibilan u određenim aspektima u poređenju sa AWS-om
- Cena može postati visoka sa povećanim obimom saobraćaja

Oracle API Gateway

Prednosti:

- Dobra integracija sa Oracle Cloud ekosistemom
- Pruža napredne opcije bezbednosti, uključujući integraciju sa Oracle Identity Cloud uslugom
- Omogućava visok stepen prilagođavanja i konfiguracije

Mane:

- Može biti preopširan i složen za male timove ili manje projekte
- Može imati ograničenu integraciju sa uslugama drugih cloud provider-a
- Može zahtevati specifično znanje Oracle tehnologija

4.9.2. On-premise rešenja

Kong Gateway

Prednosti:

- Podržava različite protokole poput HTTP, HTTPS, WebSocket, TCP i UDP
- Fleksibilan i omogućava laku integraciju sa različitim ekstenzijama
- Open-source projekt, što znači da je dostupan za modifikovanje i prilagođavanje prema potrebama

Mane:

- Zahteva određenu ekspertizu za konfiguraciju, posebno u složenijim scenarijima
- U poređenju sa drugim gateway-ima, Kong može imati manju zajednicu korisnika

Tyk API Gateway

Prednosti:

- Jednostavna konfiguracija i upravljanje
- Pruža detaljnu analitiku i praćenje saobraćaja
- Ima ugrađenu podršku za GraphQL API-je
- Ima aktivnu zajednicu korisnika i tehničku podršku

Mane:

- Može biti manje fleksibilan u određenim naprednim scenarijima u poređenju sa nekim drugim gateway-ima.
- Besplatan plan ima ograničenja u odnosu na broj zahteva

Express Gateway

Prednosti:

- Namenski dizajniran da bude jednostavan za korišćenje i konfiguraciju
- Pruža podršku za serverless arhitekturu
- Omogućava automatsko skaliranje sa povećanim opterećenjem
- Ima razvijen ekosistem koja pomaže korisnicima tokom razvoja i implementacije

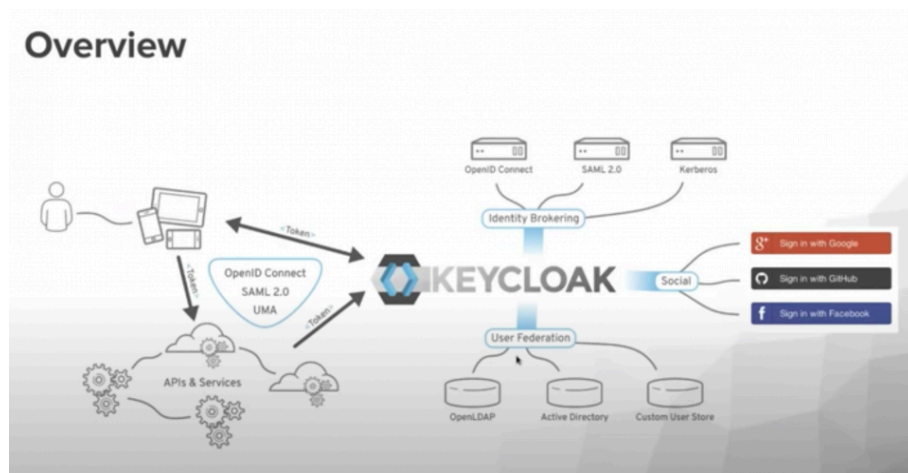
Mane:

- Može imati manje opcija u određenim naprednim funkcionalnostima u poređenju sa drugim API gateway-ima

5. KEYCLOAK

5.1. Osnovi koncepti i terminologija

Keycloak funkcioniše kao server kojim se upravlja u mreži. Aplikacije su konfigurisane da budu osigurane od strane ovog servera. Keycloak koristi otvorene protokol standarde kao što su Open ID Connect ili SAML 2.0 kako bi obezbedio sigurnost aplikacija. Prilikom pristupa veb aplikaciji koja koristi Keycloak servis kao zaštitni mehanizam korisnik se preusmerava sa na Keycloak server za autentifikaciju gde korisnici unose svoje pristupne podatke. Ovo je važno jer su korisnici potpuno izolovani od aplikacija i aplikacije nikada ne vide korisničke pristupne podatke. Umesto toga, aplikacijama se dodeljuje identifikacioni token.



Slika 19. Pregled arhitekture Keycloak servisa

Tokeni mogu sadržati informacije o identitetu kao što su korisničko ime, adresa, e-mail adresa i druge podatke profila. Takođe, mogu sadržati podatke o dozvolama kako bi aplikacije mogle donositi odluke o autorizaciji. Tokeni se takođe mogu koristiti za bezbedne pozive ka servisima zasnovanim na REST arhitekturi.

Korisnici

Korisnici su entiteti koji mogu da se prijave na sistem. Mogu imati atribute koji su povezani sa njima kao što su e-mail adresa, korisničko ime, adresa, broj telefona i datum rođenja. Mogu im biti dodeljena članstva u grupama i mogu imati specifične uloge dodeljene njima.

Autentifikacija

Autentifikacija predstavlja proces identifikovanja i validiranja korisnika. Autorizacija

Autorizacija

Autorizacija predstavlja proces dodeljivanja pristupa korisniku.

Kredencijali

Kredencijali su delovi podataka koje Keycloak koristi kako bi verifikovao identitet korisnika. Neki primeri uključuju lozinke, jednokratne lozinke, digitalne sertifikate ili čak otiske prstiju.

Uloge

Uloge identifikuju tip ili kategoriju korisnika. Administrator, korisnik, menadžer i zaposleni su tipične uloge koje mogu postojati u organizaciji. Aplikacije često dodeljuju pristup i dozvole određenim ulogama umesto pojedinačnim korisnicima, budući da se upravljanje korisnicima može činiti previše detaljnim i teško za upravljanje.

Mapiranje uloge korisnika

Mapiranje uloge korisnika definiše vezu između uloge i korisnika. Korisnik može biti povezan sa nula ili više uloga. Ove informacije o mapiranju uloge se mogu enkapsulirati u tokene i tvrdnje kako bi aplikacije mogle donositi odluke o pristupu resursima koje upravljaju.

Kompozitne uloge

Kompozitna uloga je uloga koja može biti povezana sa drugim ulogama. Na primer, kompozitna uloga superkorisnika može biti povezana sa ulogama administratora prodaje i administratora unosa narudžbina. Ako je korisnik mapiran na kompozitnu ulogu superkorisnika, takođe nasleđuje uloge administratora prodaje i administratora unosa narudžbina.

Grupe

Grupe upravljaju grupama korisnika. Atributi se mogu definisati za grupu. Takođe, uloge se mogu mapirati na grupu. Korisnici koji postanu članovi grupe nasleđuju attribute i mapiranja uloga koja grupa definiše.

Realm

Realm upravlja skupom korisnika, kredencijala, uloga i grupa. Korisnik pripada i prijavljuje se u realm. Realm-ovi su izolovani jedni od drugih i mogu upravljati i autentifikovati samo korisnike koje kontrolišu.

Klijenti

Klijenti su entiteti koji mogu zahtevati od Keycloak-a da autentifikuje korisnika. Najčešće, klijenti su aplikacije i servisi koji žele koristiti Keycloak kako bi se osigurali i obezbedili rešenje za jednostruko prijavljivanje. Klijenti takođe mogu biti entiteti koji žele samo da zahtevaju informacije o identitetu ili pristupni token kako bi mogli bezbedno pozivati druge servise na mreži koji su osigurani putem Keycloak-a.

Klijentski adapteri

Klijentski adapteri su dodaci koji se mogu instalirati u okruženje aplikacije kako bi se komuniciralo na siguran način sa Keycloak-om. Keycloak ima niz adaptera za različite platforme koji se mogu preuzeti. Takođe postoje i third-party adapteri koji se mogu koristiti za okruženja koja nisu obuhvaćena standardnim adapterima.

Klijentski šabloni

Kada se registruje klijent, potrebno je uneti konfiguracione informacije o tom klijentu. Često je korisno sačuvati šablon kako bi se olakšalo kreiranje novih klijenata. Keycloak pruža koncept klijentskog šablona za ovu svrhu.

Saglasnost (Consent)

Saglasnost (Consent) predstavlja situaciju kada administrator želi da korisnik da dozvolu određenom klijentu pre nego što taj klijent može učestvovati u procesu autentifikacije. Nakon što korisnik unese svoje pristupne podatke, Keycloak će prikazati ekran koji identifikuje klijenta koji zahteva prijavljivanje i koje informacije o identitetu su od korisnika zatražene. Korisnik može da odluči da li želi ili ne da odobri zahtev.

Klijentska uloga

Klijenti mogu definisati uloge koje su specifične za njih. Ovo je suštinski prostor za uloge posvećene klijentu.

Identifikacioni token

Token koji pruža informacije o identitetu korisnika. Deo je OpenID Connect specifikacije.

Pristupni token

Token koji može biti deo HTTP zahteva koji daje pristup usluzi koja se poziva. Ovo je deo OpenID Connect i OAuth 2.0 specifikacije.

Tvrdnja

Informacija o korisniku. Ovo se obično odnosi na XML blok koji je uključen u SAML autentifikacioni odgovor i pruža metapodatke o identitetu autentifikovanog korisnika.

Servisni nalog

Svaki klijent ima ugrađeni servisni nalog koji mu omogućava dobijanje pristupnog tokena.

Direktno dodeljivanje

Način za klijenta da dobije pristupni token u ime korisnika putem REST poziva.

Protokolni maperi

Za svakog klijenta se mogu prilagoditi tvrdnje koje se čuvaju u OIDC tokenima ili SAML tvrdnjama. Prilagođavanje se vrši za svakog klijenta tako što se kreira i konfiguriše protokolni maper.

Sesija

Kada se korisnik prijavi, kreira se sesija za upravljanje prijavljivanjem. Sesija sadrži informacije kao što su vreme kada je korisnik prijavljen i koje aplikacije su učestvovala u jednostranom prijavljivanju tokom te sesije. I administratori i korisnici mogu pregledati informacije o sesiji.

Provajder za udruživanje korisnika

Keycloak može skladištiti i upravljati korisnicima. Često, kompanije već imaju LDAP ili Active Directory servise koji čuvaju informacije o korisnicima i njihove kredencijale. Moguće je preusmeriti Keycloak da validira kredencijale iz tih spoljnih skladišta i postaviti mogućnost da preuzme informacije o identitetu.

Provajder identiteta

Provajder identiteta (IDP) predstavlja uslugu koja može autentifikovati korisnika. Keycloak je IDP rešenje.

Udruživanje provajdera identiteta

Keycloak se može konfigurisati da delegira autentifikaciju jednom ili više IDP-ova. Društvena prijava putem Facebook-a ili Google-a je primer udruživanja provajdera identiteta. Takođe moguće je povezati Keycloak da delegira autentifikaciju bilo kom drugom Open ID Connect ili SAML 2.0 IDP-u.

Maperi provajdera identiteta

Kada se vrši udruživanje sa IDP-om, moguće je mapirati dolazne tokene i tvrdnje na atribute korisnika i sesije. Ovo omogućava da se prenese informacija o identitetu iz eksternog IDP-a do klijenta koji traži autentifikaciju.

Obavezne radnje

Obavezne radnje su radnje koje korisnik mora da izvrši tokom procesa autentifikacije. Korisnik neće moći da završi proces autentifikacije dok ove radnje nisu izvršene. Na primer, administrator može podesiti da korisnici resetuju svoje lozinke svakog meseca. Radnja za obavezno ažuriranje lozinke biće postavljena za sve ove korisnike.

Autentifikacioni tokovi

Autentifikacioni tokovi su tokovi koje korisnik mora da izvrši prilikom interakcije sa određenim aspektima sistema. Tok prijavljivanja može definisati koje vrste kredencijala su potrebne. Tok registracije definiše koje informacije o profilu korisnika se moraju uneti i da li se koristi nešto poput reCAPTCHA-e da bi se izfiltrirali botovi. Tok za resetovanje akreditiva definiše koje akcije korisnik mora da izvrši pre nego što može da resetuje svoju lozinku.

Događaji

Događaji su nizovi revizije koje administratori mogu da pregledaju i povežu se sa njima.

Teme

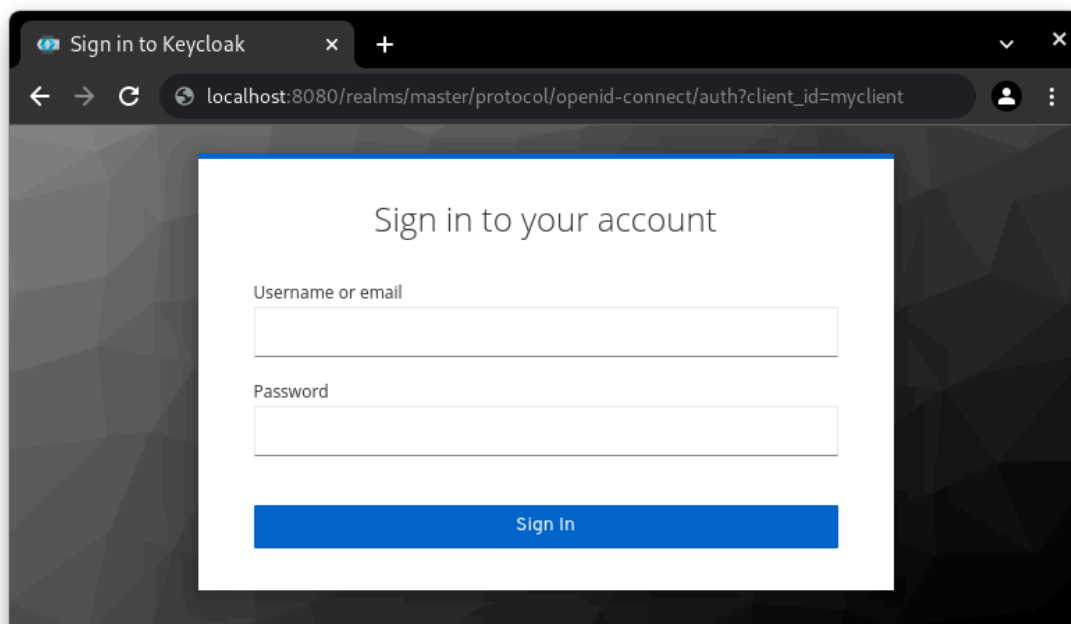
Svaka stranica koju pruža Keycloak ima podršku za različite teme. Teme se definišu preko HTML, CSS i JS fajlova koji se mogu dodati po potrebi.

5.2. Funkcionalnosti Keycloak servisa

5.2.1. Single-Sign On (SSO)

Jednostruko prijavljivanje (Single-Sign On, SSO) omogućava korisnicima da se autentifikuju putem Keycloak-a umesto individualnih aplikacija. Ova funkcionalnost eliminiše potrebu da aplikacije koriste forme za prijavljivanje, autentifikacijom korisnika i skladištenjem korisničkih podataka. Nakon što se korisnik prijavi u Keycloak, nije potrebno ponovno prijavljivanje kako bi se pristupilo različitim aplikacijama.

Ovaj princip takođe važi i za proces odjavljivanja. Keycloak omogućava funkciju Single-Sign Out, što znači da korisniku treba samo jedno odjavljivanje da bi bio izlogovan sa svih aplikacija koje koriste Keycloak.

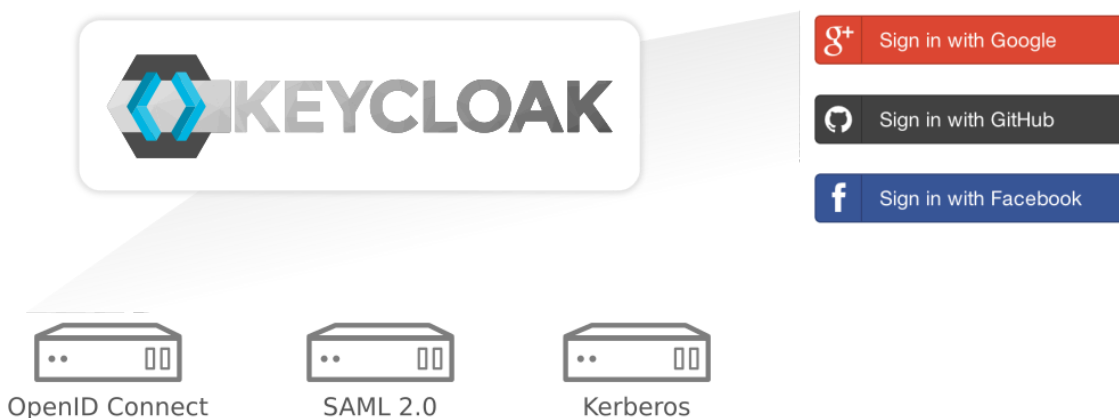


Slika 20. Izgled ugrade stranice za prijavljivanje korisnika

5.2.2. Upravljanje identitetom i prijavljivanje preko društvenih mreža

Omogućavanje prijavljivanja korisnika putem društvenih mreža predstavlja postupak koji se jednostavno izvršava putem administratorske konzole. Ovaj proces obuhvata selektovanje željene društvene mreže unutar administratorskog interfejsa i postavljanja podešavanja. Za podešavanje prijavljivanja preko društvenih mreža nisu potrebna programerska znanja niti modifikacije unutar aplikacija.

Takođe, Keycloak pruža mogućnost autentifikacije korisnika putem već postojećih provajdera identiteta kao što su OpenID Connect ili SAML 2.0. Ova funkcionalnost se takođe konfiguriše putem administratorske konzole, omogućavajući lakše integrisanje sa već postojećim provajderima identiteta.



Slika 21. Prikaz mogućnosti prijavljivanja na Keycloak servis

5.2.3. Federacija korisnika

Keycloak ima ugrađenu podršku za povezivanje sa postojećim LDAP ili Active Directory serverima. Takođe je moguće implementirati svoj sopstveni provajder ukoliko na sistemu postoje korisnici u drugim skladištima, kao što su relacijske baze podataka.



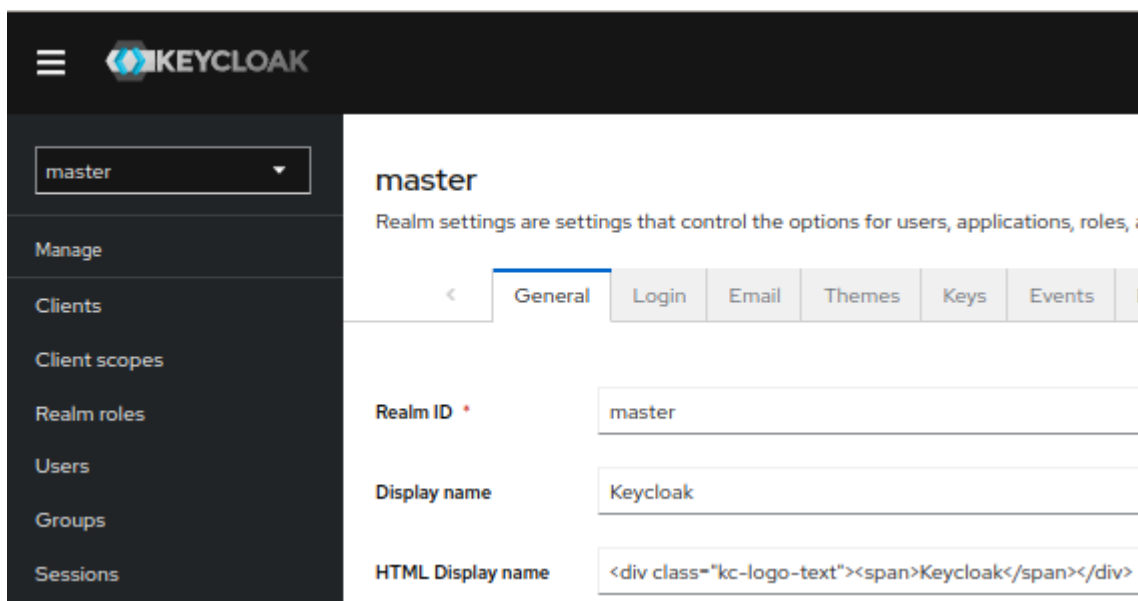
Slika 22. Prikaz mogućnosti povezivanja Keycloak servisa

5.2.4. Administratorska konzola

Administratorska konzola kod Keycloak servisa predstavlja glavnu upravljačku komponentu koja omogućava administratorima centralizovano upravljanje svim mogućnostima Keycloak servera u okviru organizacione infrastrukture. Konzola pruža administratorima ovlašćenje da efikasno obavljaju zadatke povezane sa sigurnošću i identitetom korisnika, kao i konfiguracijom servisa.

Kroz administratorsku konzolu, administratori mogu omogućavati ili onemogućavati raznovrsne funkcionalnosti i opcije sistema. Ovo uključuje podešavanje i konfiguraciju sistema za povezivanje identiteta sa različitim servisima, kao i federaciju korisnika iz različitih izvora. Pored toga, administratorska konzola omogućava kreiranje i upravljanje aplikacijama i uslugama unutar sistema, omogućavajući detaljno definisanje politika autorizacije i pristupa.

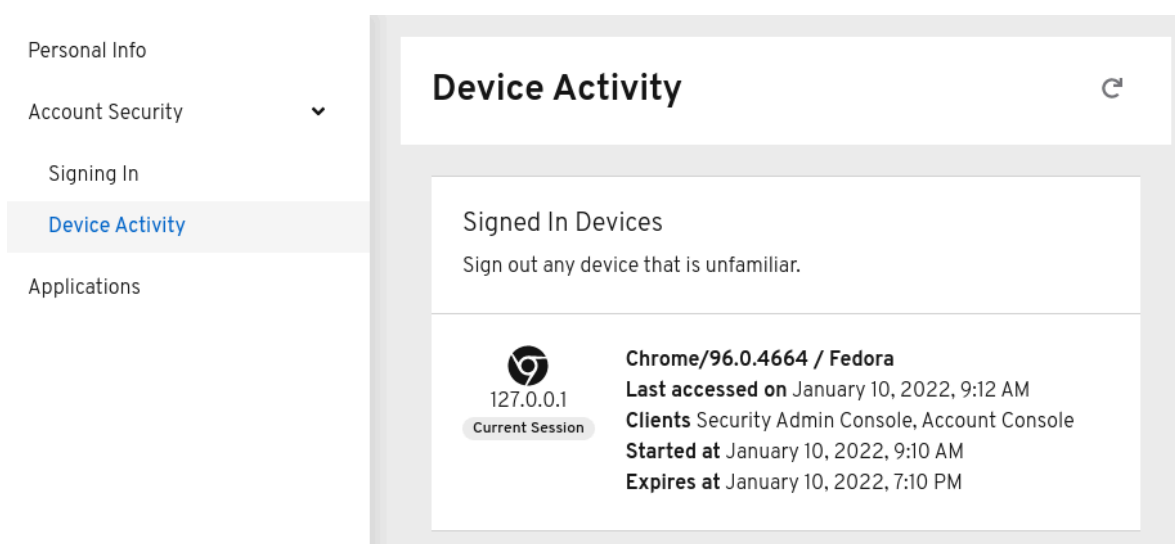
Ključna uloga administratorske konzole leži u sposobnosti administratora da upravljaju korisnicima, uključujući njihove privilegije i sesije. Ovime se osigurava adekvatna kontrola nad pravima korisnika, kao i pravilna upotreba sesija radi osiguranja sigurnosti pristupa i transakcija. Kao rezultat, administratorska konzola kod Keycloak servisa predstavlja glavni alat za održavanje visokog nivoa bezbednosti i upravljanja identitetima u okviru organizacione IT infrastrukture.



Slika 23. Izgled administratorske konzole

5.2.5. Konzola za upravljanje nalogom

Putem konzole za upravljanje nalogima, korisnici imaju mogućnost da samostalno upravljaju svojim nalogima. Oni mogu ažurirati svoj profil, promeniti lozinke i konfigurirati dvofaktornu autentifikaciju.



Slika 24. Izgled korisničke konzole

Korisnici takođe imaju sposobnost da upravljaju svojim sesijama, kao i da pregledaju istoriju svojih naloga.

Ukoliko je omogućena prijava putem društvenih mreža ili povezivanje identiteta, korisnici takođe mogu povezati svoje naloge sa dodatnim provajderima kako bi im se

omogućilo da se autentifikuju na isti nalog putem različitih provajdera identiteta. Ovo doprinosi fleksibilnosti korisničke autentifikacije i pristupa, pružajući korisnicima veći stepen kontrole nad svojim nalogima.

5.3. Korišćenje OpenID Connect-a za obezbeđivanje aplikacija i usluga

5.3.1. OpenID Connect

OpenID Connect (OIDC) je otvoreni protokol za autentifikaciju koji funkcioniše na osnovu OAuth 2.0. Orjentisan je ka korisnicima i OIDC omogućava pojedincima da koriste Single-Sign On kako bi pristupili sajtovima koji se oslanjaju na OpenID provajdere (OP), kao što su servisi elektronske pošte ili društvene mreže, radi autentifikacije njihovih identiteta. Pruža aplikaciji ili usluzi informacije o korisniku, kontekstu njihove autentifikacije i pristup njihovim profilnim informacijama.

Svrha OIDC-a je da omogući korisnicima da pruže jedan skup kredencijala za pristup više sajtova. Svaki put kada se korisnici prijave na aplikaciju putem OIDC-a, preusmeravaju se na svoj OP, gde se autentifikuju, nakon čega se ponovo preusmeravaju na aplikaciju ili uslugu.

OIDC je jedan od najnovijih sigurnosnih protokola i osmišljen je da zaštiti aplikacije veb aplikacije, API-je i mobilne aplikacije. Delegira autentifikaciju korisnika servisnom pružaocu koji hostuje korisnički nalog i omogućava trećim stranama da pristupe korisničkom nalogu.

Na primer, postoje dva načina za kreiranje naloga neke aplikacije. Moguće je registrovati se putem forme te aplikacije ili je moguće prijaviti se putem Google-a. Google šalje ime i email adresu toj aplikaciji, koja koristi te informacije za autentifikaciju.

5.3.2. Dostupni endpointi

Kao potpuno usaglašena implementacija OpenID Connect provajdera, Keycloak podržava skup endpointa koje aplikacije i usluge mogu koristiti za autentifikaciju i autorizaciju svojih korisnika.

Well-known endpoint

Endpoint URL: **`/realms/{ime-realma}/.well-known/openid-configuration`**

Najvažnija endpoint koji treba razumeti je endpoint za konfiguraciju informacija. Ovaj endpoint vraća listu endpointa i drugih relevantnih konfiguracionih opcija za OpenID Connect implementaciju u Keycloak-u.

Za punu URL adresu potrebno je dodati osnovni URL za Keycloak i zameniti {ime-realma} imenom konkretnog realma.

Primer: **`http://localhost:8080/realms/{ime-realma}/.well-known/openid-configuration`**

Endpoint za autorizaciju

Endpoint URL: `/realms/{ime-realma}/protocol/openid-connect/auth`

Endpoint za autorizaciju vrši autentifikaciju krajnjeg korisnika. Autentifikacija se vrši tako što se korisnički agent preusmerava na ovaj endpoint.

```
GET /authorize?
  response_type=code
  &scope=openid%20profile%20email
  &client_id=s6BhdRkqt3
  &state=af0ifjsldkj
  &redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb HTTP/1.1
Host: server.example.com
```

Slika 25. Primer auth HTTP zahteva

Token endpoint

Endpoint URL: `/realms/{ime-realma}/protocol/openid-connect/token`

Ovaj endpoint se koristi za dobijanje tokena. Tokeni se mogu dobiti razmenom autorizacionog koda ili direktnim unosom kredencijala, zavisno o tome koji tok se koristi. Ovaj endpoint se takođe koristi za dobijanje novih pristupnih tokena kada stari isteknu.

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=authorization_code&code=Splxl0BeZQQYbYS6WxSbIA
&redirect_uri=https%3A%2F%2Fclient.example.org%2Fcb
```

Slika 26. Primer token HTTP zahteva

Endpoint o korisničkim informacijama

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "sub": "248289761001",
  "name": "Jane Doe",
  "given_name": "Jane",
  "family_name": "Doe",
  "preferred_username": "j.doe",
  "email": "janedoe@example.com",
  "picture": "http://example.com/janedoe/me.jpg"
}
```

Slika 27. Primer odgovora userinfo endpointa

Endpoint URL: **/realms/{ime-realma}/protocol/openid-connect/userinfo**

Endpoint userinfo vraća standardne informacije o autentifikovanom korisniku i zaštićen je bearer tokenom.

Endpoint za odjavljivanje

Endpoint URL: **/realms/{ime-realma}/protocol/openid-connect/logout**

Endpoint za odjavljivanje (logout) omogućava odjavljivanje autentifikovanog korisnika. Korisnički agent može biti preusmeren na ovaj endpoint, što uzrokuje da aktivna korisnička sesija bude odjavljena. Zatim, korisnički agent je ponovo preusmeren nazad ka aplikaciji. Takođe, endpoint može biti direktno pozvan od strane aplikacije. Da bi se ovaj endpoint direktno pozvao, osvežavajući token (refresh token) treba biti uključen, kao i kredencijali koji su potrebni za autentifikaciju klijenta.

Endpoint za sertifikate

Endpoint URL: **/realms/{ime-realma}/protocol/openid-connect/certs**

Endpoint za sertifikate vraća javne ključeve kreirane u okviru realma, kodirane kao JSON Web Key (JWK). Zavisno od podešavanja realma, jedan ili više ključeva može biti omogućeno za verifikaciju tokena.

Endpoint za introspekciju

Endpoint URL: **/realms/{ime-realma}/protocol/openid-connect/token/introspect**

Endpoint za introspekciju se koristi za dobijanje aktivnog statusa tokena, odnosno može se koristiti za validaciju pristupnog ili osvežavajućeg tokena. Ovaj endpoint mogu pozivati samo poverljivi klijenti.

Endpoint za dinamičku registraciju klijenta

Endpoint URL: **/realms/{ime-realma}/clients-registrations/<provider>**

Da bi aplikacija ili servis mogli koristiti Keycloak, moraju registrovati klijenta u Keycloak. Administrator to može uraditi putem administratorske konzole (ili administratorskih REST endpointa), ali se klijenti takođe mogu registrovati putem Keycloak servisa za registraciju klijenata.

Podržani provajderi su: default, install, openid-connect i saml2-entity-descriptor

Endpoint za poništavanje tokena

Endpoint URL: **/realms/{ime-realma}/protocol/openid-connect/revoke**

Ovaj endpoint se koristi za poništavanje tokena. Ovaj endpoint podržava pristupne i tokene osvežavanja. Kada se poništi osvežavajući token, takođe se poništava i saglasnost korisnika za odgovarajućeg klijenta.

Endpoint za odobrenje uređaja

Endpoint URL: `/realms/{ime-realma}/protocol/openid-connect/auth/device`

Endpoint za odobrenje uređaja koristi se za dobijanje koda uređaja i koda korisnika. Može je koristiti poverljivi ili javni klijent.

5.3.3. Podržani protokoli (Grant type)

Protokoli određuju niz koraka koji su uključeni u OAuth proces. Vrsta dodele takođe utiče na to kako klijentska aplikacija komunicira sa OAuth servisom u svakoj fazi, uključujući i način slanja pristupnog tokena.

OAuth servis mora biti konfigurisan da podržava određeni protokol pre nego klijentska aplikacija pokrene odgovarajući tok. Klijentska aplikacija specificira koji protokol želi da koristi u inicijalnom zahtevu za autorizaciju koji šalje OAuth servisu.

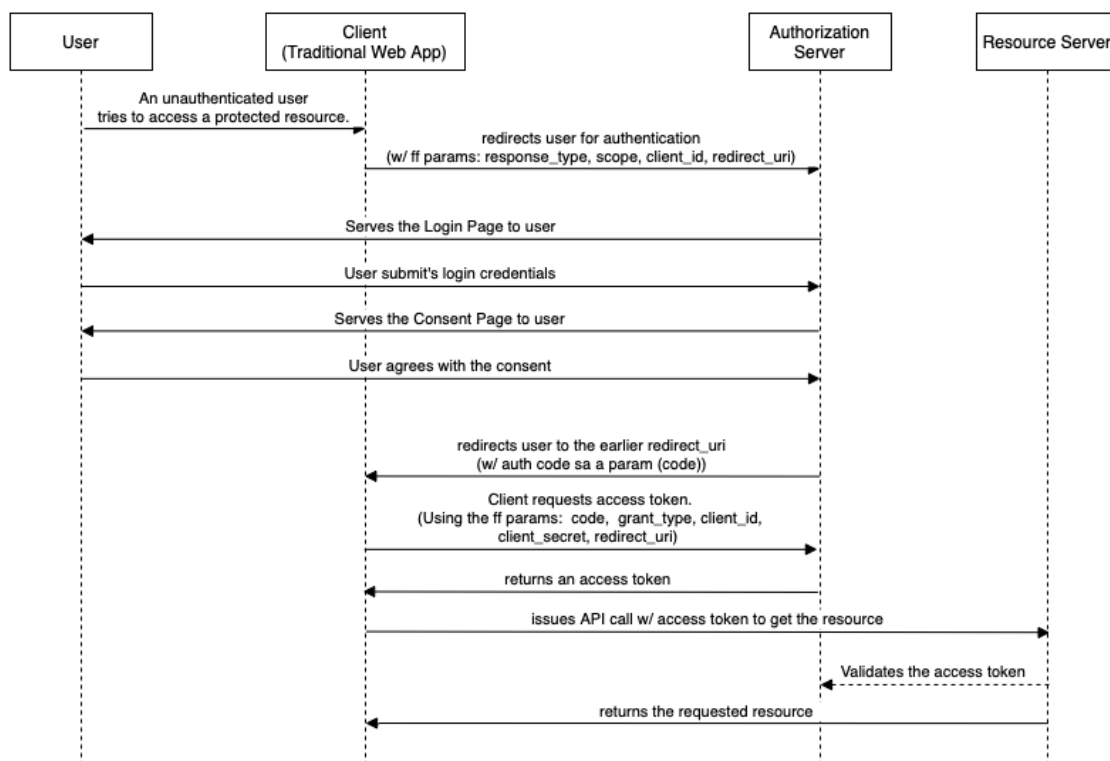
Authorization code

Authorization Code protokol je uobičajeni OAuth 2.0 protokol koji se koristi za autentifikaciju korisnika i dobijanje pristupnih tokena, tokena za obnavljanje i ID tokena.

1. **Korisnik pokreće proces prijave:** Proces započinje kada korisnik pokrene proces prijave ili autentifikacije unutar aplikacije. Na primer, kliknuće na dugme "Prijava".
2. **Preusmeravanje ka Keycloak-u:** Aplikacija primeti da korisnik nije autentifikovan, pa preusmerava korisnikov veb pregledač na Keycloak autorizacioni server putem `/authorize` endpoint-a. Ovo preusmeravanje uključuje različite parametre, kao što su identifikator klijenta (client ID), obim (scopes) i URL za povratne informacije (callback URL).
3. **Autentifikacija korisnika:** Keycloak autorizacioni server je odgovoran za autentifikaciju korisnika. Može zatražiti od korisnika da unese svoje pristupne podatke, kao što su korisničko ime i lozinka. Takođe, korisnik može videti prozor za dodeljivanje pristanka koji specificira dozvole koje aplikacija zahteva.
4. **Autorizacioni kod:** Nakon što korisnik uspešno prođe autentifikaciju i, ako je potrebno, odobri pristup, Keycloak autorizacioni server generiše autorizacioni kod. Ovaj kod je jednokratna oznaka koja je kratkotrajna i može se koristiti samo jednom za dobijanje tokena.
5. **Povratak na aplikaciju:** Zatim Keycloak preusmerava korisnikov pregledač nazad ka aplikaciji koristeći prethodno navedeni callback URL. Autorizacioni kod se uključuje kao upitni parametar u URL-u.
6. **Zahtev za token:** Aplikacija izdvaja autorizacioni kod iz URL-a i šalje zahtev ka Keycloak token endpoint-u (`/oauth/token`). Ovaj zahtev uključuje autorizacioni kod, pristupne podatke klijenta (kao što su client ID i client secret) i informacije o callback URL-u.

7. **Dobijanje tokena:** Keycloak proverava autorizacioni kod, zajedno sa pristupnim podacima klijenta, i ako je sve u redu, odgovara sa ID tokenom, pristupnim tokenom i opcionalno tokenom za obnavljanje.
8. **Upotreba tokena:** Aplikacija sada može koristiti ove tokene:
 - ID token: Sadrži informacije o korisniku, kao što je identitet.
 - Pristupni token: Koristi se za pristup zaštićenim resursima u ime korisnika.
 - Token za obnavljanje: Koristi se za dobijanje novog pristupnog tokena kada istekne trenutni, bez potrebe za ponovnom prijavom korisnika.
9. **Pristup resursima:** Aplikacija može koristiti pristupni token da bi sigurno pozvala API-je ili pristupila zaštićenim resursima u ime korisnika.
10. **Obnavljanje tokena (ako je potrebno):** Kada pristupni token istekne (obično je kratkotrajan), aplikacija može koristiti token za obnavljanje da dobije novi pristupni token, bez potrebe da korisnik ponovno obavlja prijavu.

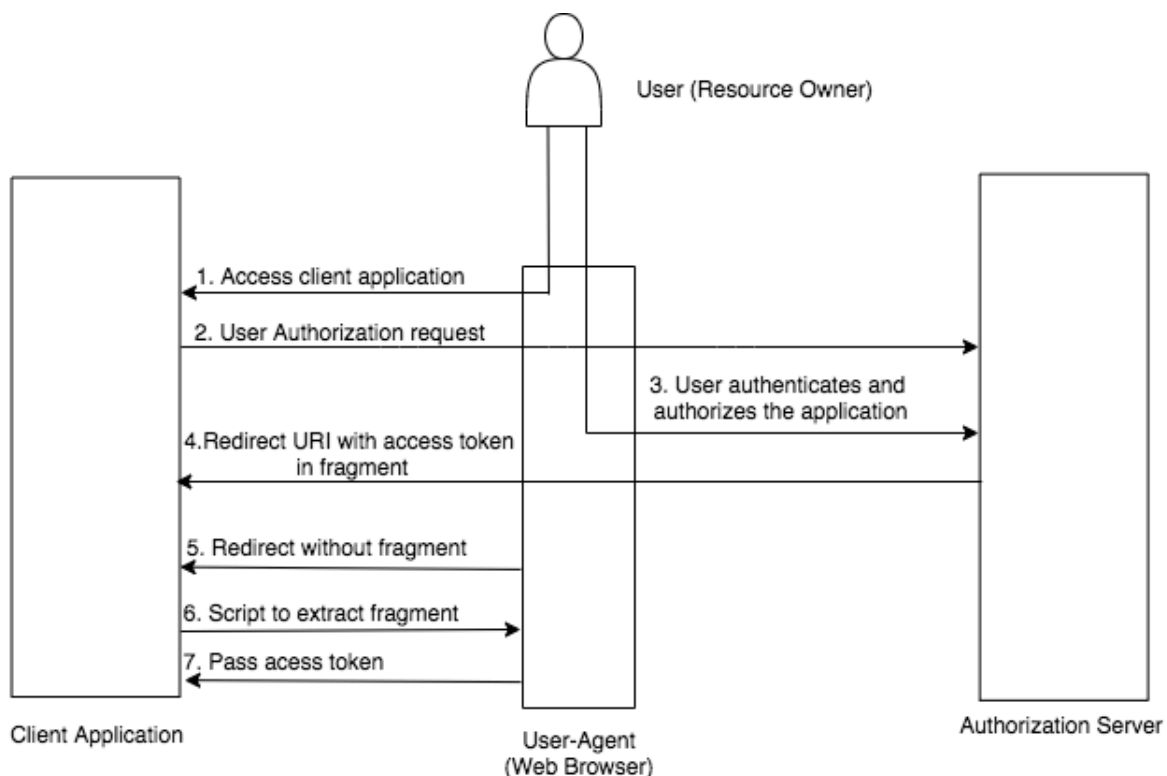
Ovaj protokol je pogodan za veb aplikacije, ali se takode preporučuje i za native aplikacije, uključujući mobilne aplikacije, gde je moguće ugraditi pregledač korisničkog agenta za obradu redirekta i interakciju sa korisnicima. Pruža siguran način dobijanja i upravljanja tokenima.



Slika 28. Authrization code protokol

Implicit

Implicitni protokol radi na sličan način kao i Authorization Code, ali umesto da vrati autorizacioni kod, vraća pristupni token i id token. Ovaj pristup smanjuje potrebu za dodatnim pozivom za razmenu autorizacionog koda za pristupni token. Međutim, ne uključuje token za obnavljanje. To dovodi do toga da pristupni tokeni imaju dugoročnu važnost. Međutim, ovo nije praktično rešenje jer je vrlo teško poništiti ove tokene. Alternativno, može se zahtevati nova redirekcija kako bi se dobio novi pristupni token nakon što je prvobitni istekao. Implicitni protokol je koristan ako aplikacija želi samo da autentifikuje korisnika i samostalno upravlja odjavom.



Slika 29. Implicit code protokol

Resource Owner Password Credentials

Protokol za dobijanje dozvola od vlasnika resursa, poznat kao Direct Grant u Keycloak-u, omogućava razmenu korisničkih kredencijala za tokene. Korišćenje ovog protoka nije preporučljivo osim ako nije apsolutno neophodno. Primeri gde ovaj protokol može biti koristan uključuju legacy aplikacije i korisničke interfejse komandne linije.

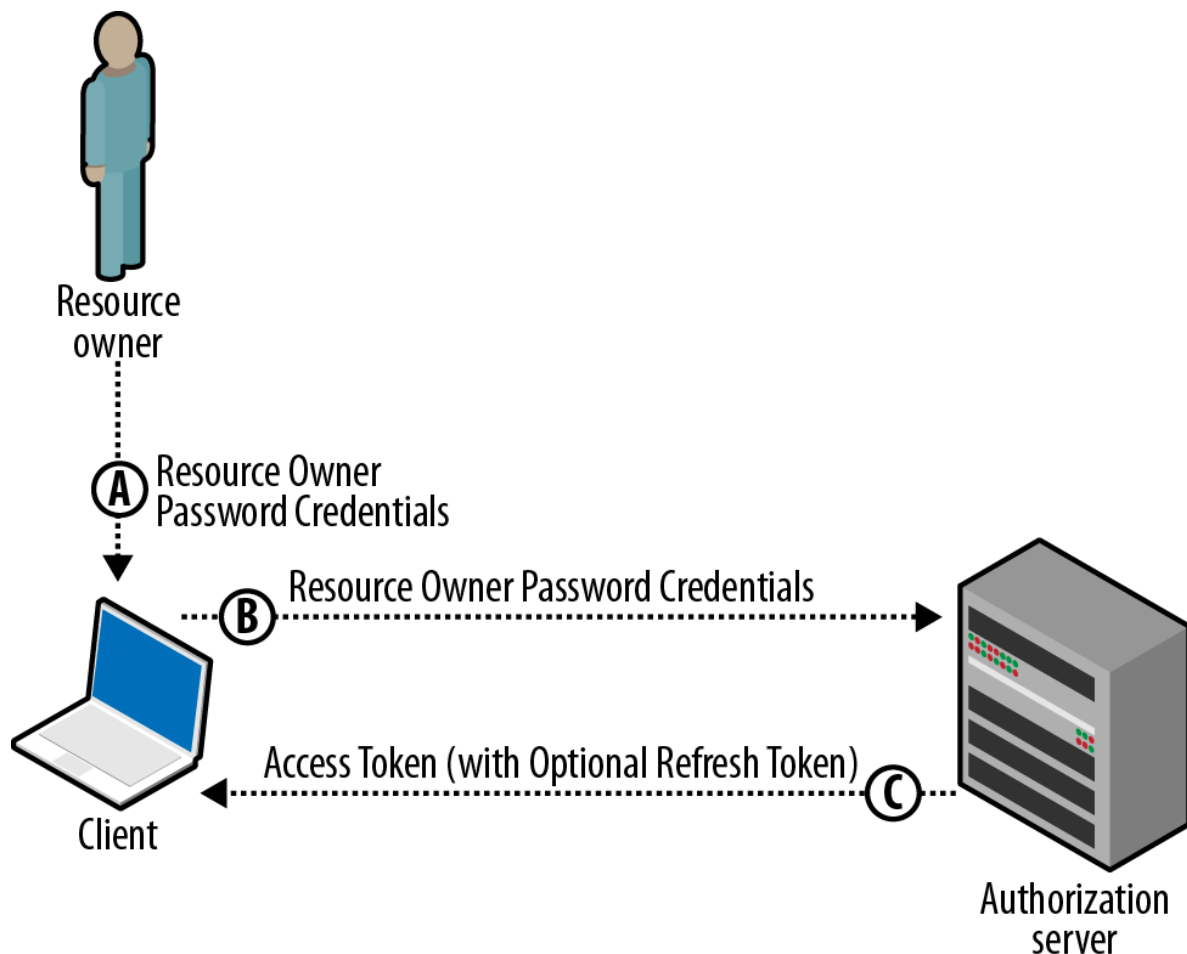
Ograničenja korišćenja ovog protoka uključuju:

- Korisnički kredencijali su izloženi aplikaciji.
- Aplikacije moraju imati stranicu za prijavu.

- Aplikacija mora biti svesna šeme autentifikacije.
- Promene u načinu autentifikacije zahtevaju promene u aplikaciji.
- Nema podrške za integrisanje identiteta ili prijavu preko društvenih mreža.
- Nema podrške za dodatne funkcionalnosti (korisnička samoregistracija, obavezne radnje i slično).

Da bi klijentu bilo dozvoljeno da koristi protokol za dobijanje dozvola od vlasnika resursa, klijent mora imati omogućenu opciju "Direct Access Grants Enabled".

Važno je napomenuti da ovaj protokol nije deo OpenID Connect specifikacije, već je deo OAuth 2.0 specifikacije.



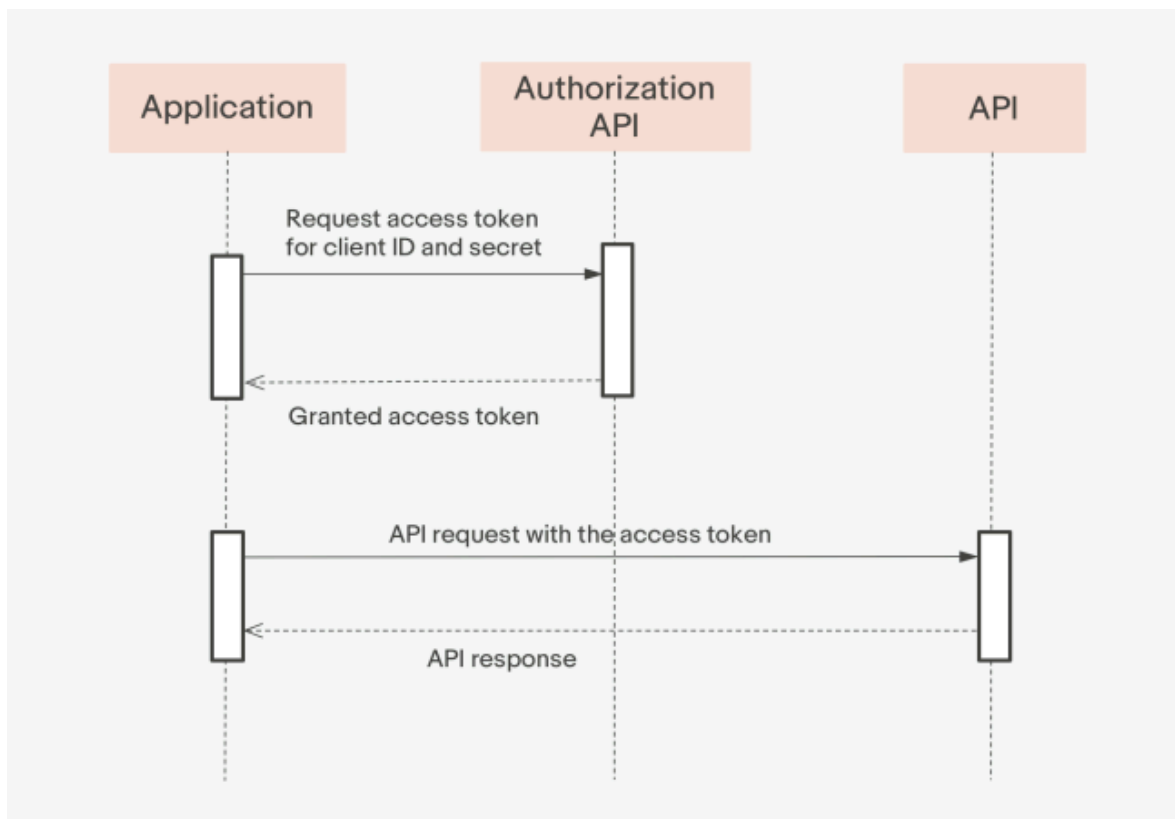
Slika 30. Resource Owner Password Credentials

Client credentials

Korisnički kredencijali klijenta koriste se kada klijenti (aplikacije i servisi) žele da dobiju pristup u svoje ime, umesto u ime korisnika. Na primer, ovi kredencijali mogu biti korisni za servise koji rade u pozadini i primenjuju promene u sistemu uopšte, umesto za određenog korisnika.

Keycloak pruža podršku za autentifikaciju klijenata putem tajnog ključa ili putem javnih/privatnih ključeva.

Važno je napomenuti da ovaj protokol nije deo OpenID Connect specifikacije, već je deo OAuth 2.0 specifikacije.



Slika 31. Client Credentials

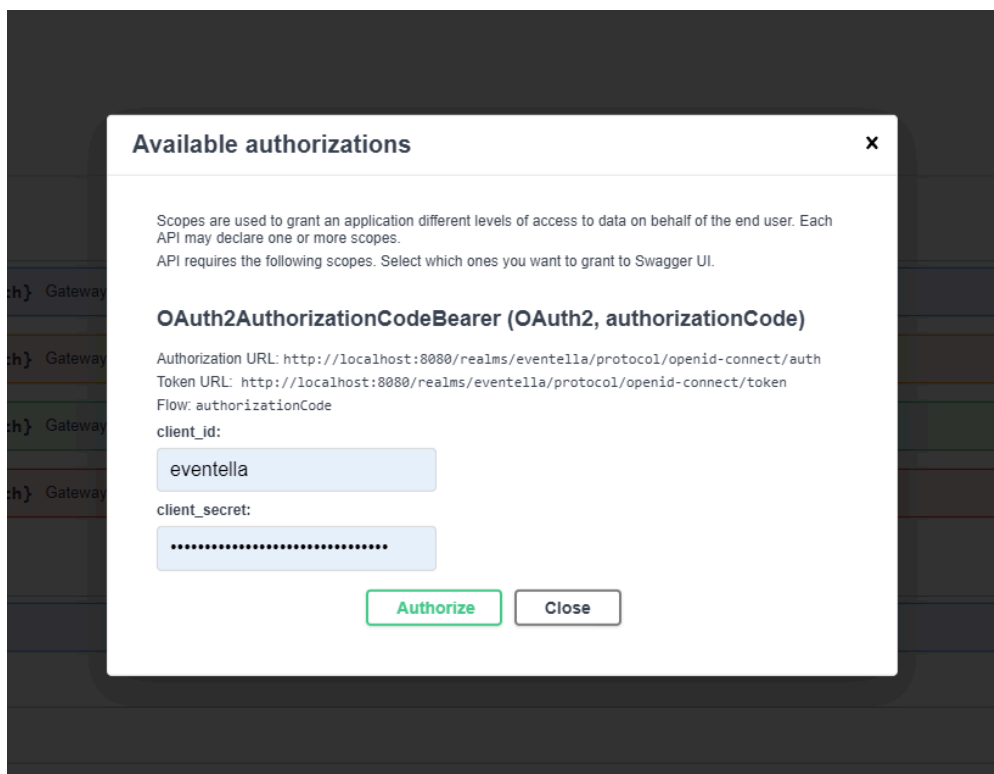
5.3.3. Primer integracije Keycloak servisa

Implementacija samog koda za integraciju mikroservisa sa Keycloak servisom je detaljno opisana u poglavlju 4.1.6. U ovom poglavlju će pažnja biti posvećena ostalim aspektima integracije sa Keycloak servisom.

FastAPI frejmwork pruža podršku za implementaciju autentifikacije i autorizacije ali je u ovom slučaju za tu svrhu iskorišćen Keycloak servis. Razlog je plan za proširenje sistema dodavanjem novih mikroservisa, frontend aplikacija i mobilne aplikacije. Is tih razloga je dosta praktičnije imati centralizovani servis koji će upravljati korisnicima i sigurnošću. Keycloak omogućava članovima tima da vreme koje bi utrošili na razvoj funkcionalnosti koje Keycloak servis već podržava posvete rešavanju domenskih problema što dodaje vrednost proizvodu.

U nedostatku pravog frontend klijenta za demonstraciju implementacije će biti iskorišćena Swagger stranica API Gateway-a. Korišćen je protokol autorizacionog koda. Prvi

korak je otvaranje dijalog forme gde je potrebno da korisnik unese *client_id* i *client_secret* kodove koje je moguće pronaći na Keycloak administracijskoj konzoli u sekciji za upravljanje klijentom. U ovom primeru implementacije mikroservisa je korišćen jedan oauth klijent za sve mikroservise, a ideja je praviti posebne klijente za svakog eksternog klijenta koji bude hteo da se integriše sa Eventella platformom.



Slika 32. Unos *client_id* i *client_secret* kodova

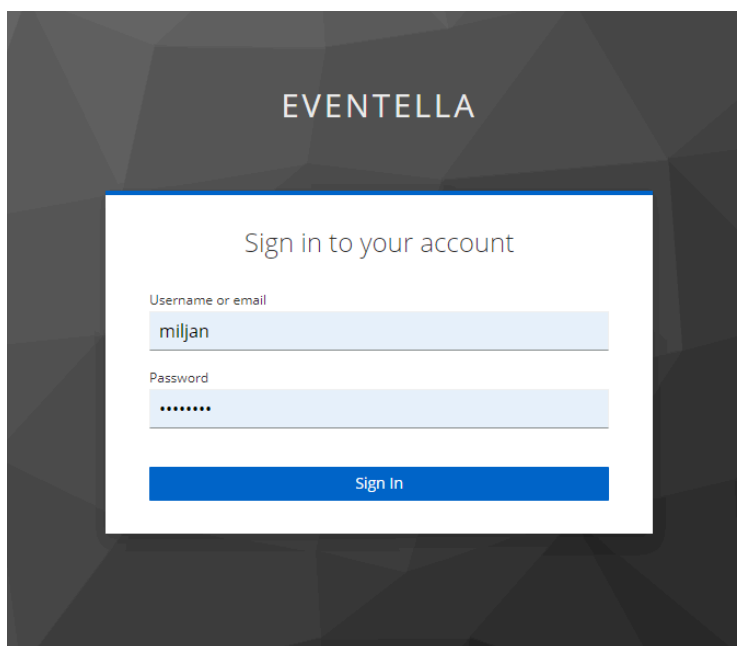
Opcija centralizovanog korišćenja jednog OAuth 2.0 klijenta za sve interne mikroservise unutar arhitekture, uključujući i API gateway, predstavlja dobru praksu iz više razloga:

- Ovaj pristup olakšava upravljanje i održavanje sistema centralizacijom postavki autentifikacije i autorizacije, kao što su opcije za kredencijale korisnika, pristupne politike i slično.
- Korišćenje jedne konfiguracije klijenta osigurava konzistentnost u svim internim servisima, što pojednostavljuje razvoj i smanjuje mogućnost grešaka u konfiguraciji ili bezbednosnim postavkama.
- Upravljanje višestrukim OAuth 2.0 klijentima za svaki mikroservis može dovesti do povećanog opterećenja u pogledu konfiguracije, održavanja i praćenja. Korišćenje jednog klijenta smanjuje ovo opterećenje.

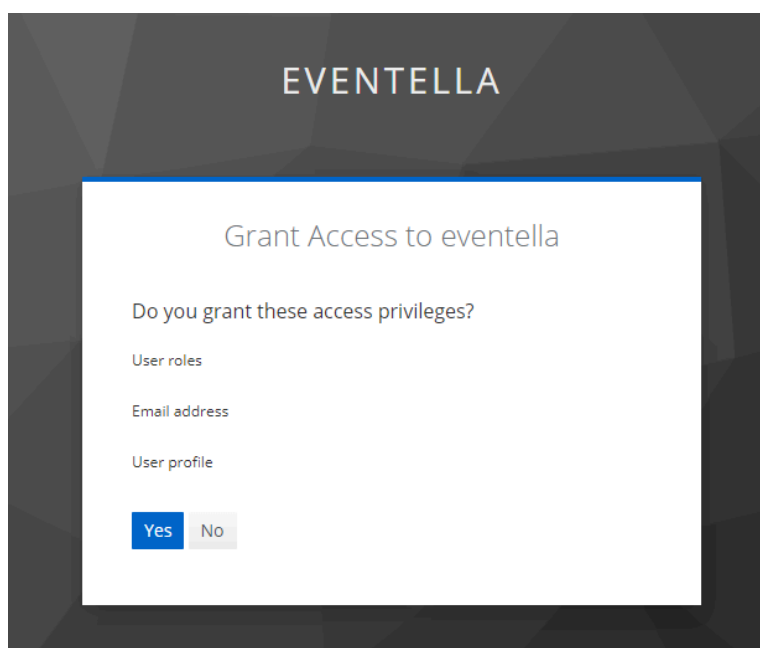
Međutim, za spoljni pristup, kao što su partneri, integracije ili aplikacije trećih strana, često je dobra ideja kreirati odvojene OAuth 2.0 klijente. Ovo omogućava primenu različitih

bezbednosnih politika, ograničenja pristupa i vremena trajanja tokena prilagođenih spoljnim slučajevima upotrebe.

Nakon unosa *client_id* i *client_secret* kodova koji se koriste za autentifikaciju klijenta korisnik je redirektovan na Keycloak servis gde je potrebno uneti svoje kredencijale. Nakon validacije kredencijala potrebno je dati odobrenje za pristup određenim resursima ili podacima, u ovom konkretnom slučaju podacima o korisniku. Nakon toga klijent je dobio token za pristup i može pristupati Eventella endpointima.



Slika 33. Unos korisničkih podataka



Slika 34. Dodela saglasnosti za pristup podacima

5.3.4. Prosleđivanje informacija iz dekoriranog tokena ili prosleđivanje tokena

Nakon što je token dobijen na API Gateway-u on je i dekodiran i dostupne su informacije iz tokena. U ovom trenutku postoje mogućnost prosleđivanja tih podataka mikroservisima u pozadini dodavanjem tih podataka u zahtev ili prosleđivanja samog tokena gde bi klijenti sami vršili ponovno dekodiranje tokena i pristupali informacijama. Oba načina imaju svojih prednosti i mana koje će u nastavku biti objašnjene.

Prosleđivanje dekodiranih podataka:

- Prednosti:
 - manji obim posla na mikroservisima jer nije potrebno ponovno dekodiranje tokena.
 - bolje performanse jer mikroservisi mogu direktno pristupiti informacijama o korisniku bez dodatne obrade.
 - konzistentnost informacija o korisniku u svim mikroservisima.
- Mane:
 - povećana kompleksnost API Gateway-a jer je potrebno bezbedno proslediti podatke tokena.
 - implementacija dodatnih sigurnosnih mera kako bi se osiguralo da dekodirani podaci sigurno stignu do mikroservisa.
 - mogućnost krađe osetljivih informacije o korisniku.

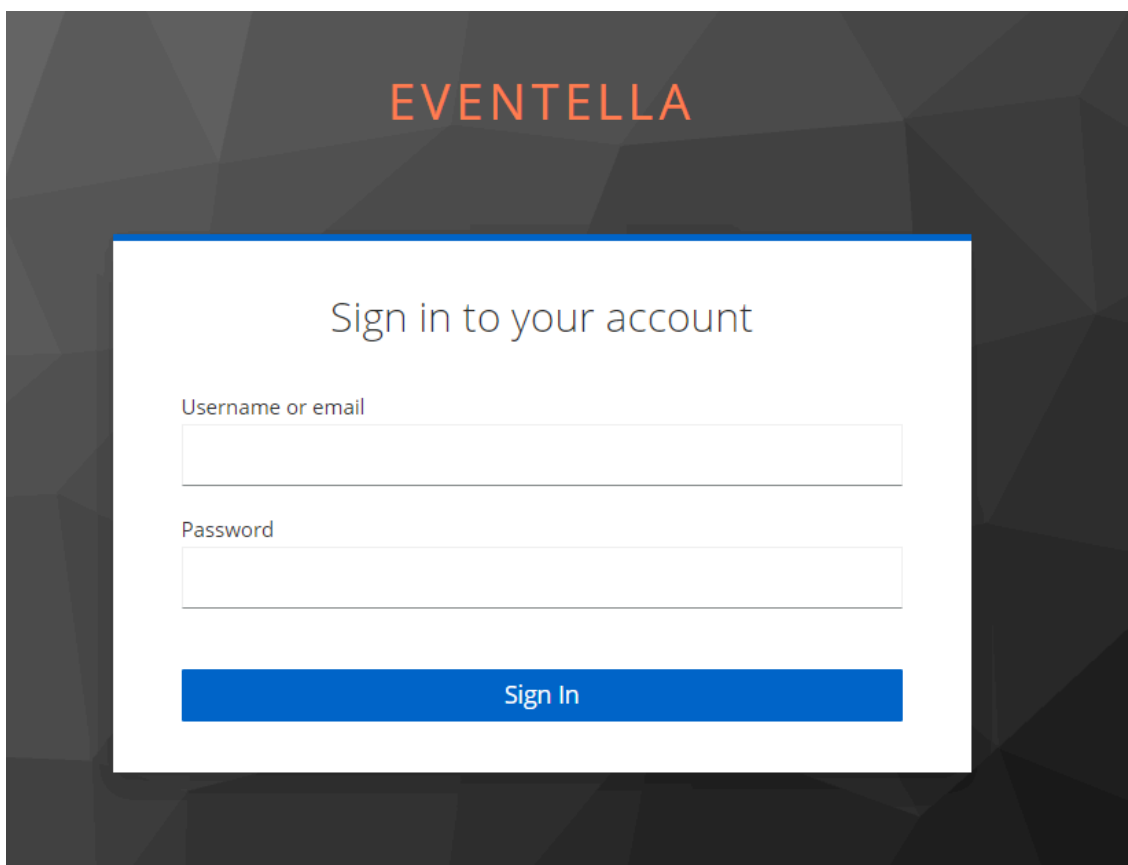
Prosleđivanje tokena:

- Prednosti:
 - bolja enkapsulacija jer je svaki mikroservis odgovoran za sopstvenu autentifikacionu logiku.
 - povećana fleksibilnost omogućavajući mikroservisima nezavisnu verifikaciju tokena i pristup informacijama o korisniku.
 - API gateway samo prosleđuje token bez obrade sadržaja.
- Mane:
 - dodatno opterećenje na mikroservisima jer moraju dekodirati token za svaki zahtev.
 - može dovesti do dupliciranja autentifikacione logike širom mikroservisa ako se ne upravlja ispravno.

Na primeru Eventella sistema odabran je pristup prosleđivanja tokena. Razlozi su veća sigurnost i smanjenje kompleksnosti na API Gateway-u.

5.3.5. Personalizovanje teme login strane

Keycloak nudi mogućnost personalizovanja korisničkog interfejsa. Na primeru Eventella sistema je odrađen primer personalizacije login strane gde je zamenjen izgled naziva aplikacije, pored toga je naravno moguće personalizovati i ostale elemente.



Slika 35. Izgled personalizovane login strane

Potrebno je kreirati direktorijum sa imenom teme u *themes* direktorijumu, zatim u tom direktorijumu kreirati direktorijum sa imenom tipa teme koji se personalizuje, u ovom slučaju je to *login* tip teme. Nakon toga se kreira *theme.properties* fajl gde se dodaju podešavanja teme. Moguće je i proširiti već postojeću temu što je slučaj u ovom primeru. Nakon toga je moguće kroz user interfejs na samom realmu izabrati novokreiranu temu.



Slika 36. Struktura fajlova novokreirane teme

U ovom primeru je preko *parent* podešavanja izmenjena *keycloak* tema i importovani su resursi preko *import* podešavanja iz osnovne teme. *styles* podešavanjem su uključeni trenutni stilovi te teme i dodan novokreirani fajl *styles.css*. Moguće je dodati i fajlove za lokalizaciju, JS fajlove, slike kao i HTML templejte.

```
opt > keycloak > themes > eventella_theme > login > ⚙ theme.properties
1 parent=keycloak
2 import=common/keycloak
3 styles=web_modules/@fontawesome/fontawesome-free/css/icons/all.css web_modules/@patternfly/react-core/dist/styles/base.css
4 web_modules/@patternfly/react-core/dist/styles/app.css node_modules/patternfly/dist/css/patternfly.min.css node_modules/patternfly/dis
5 styles.css

opt > keycloak > themes > eventella_theme > login > resources > # styles.css > ↵ #kc-header-wrapper
1 #kc-header-wrapper {
2 |   color: ■ coral;
3 }
```

Slika 37. theme.properties i styles.css fajlovi

5.3.6. Poređenje sa sličnim rešenjima na tržištu

Pored Keycloak-a na tržištu postoji dosta drugih rešenja koja se mogu upotrebiti za istu svrhu. Svako od ovih rešenja ima prednosti i mane i u zavisnosti od potreba neće uvek isto rešenje biti najbolji izbor. Prilikom izbora treba sagledati prednosti i mane svakog rešenja i nakon toga doneti odluku o izboru rešenja. U nastavku su prikazane najosnovnije informacije i mogućnosti nekih od popularnih rešenja na tržištu.

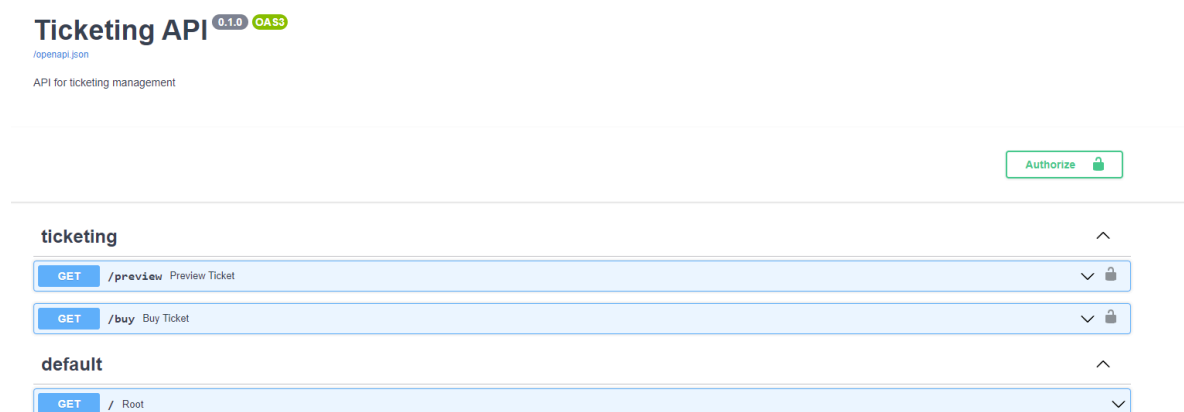
- **Okta Workforce Identity:**
 - Okta je servis za upravljanje identitetom i pristupom veb aplikacijama. Može se koristiti kao cloud rešenje i kao on-premise rešenje.
 - Podržava Single Sign-On.
 - Skuplje je rešenje za razliku od Keycloak-a.
 - Ima bolju korisničku podršku
- **Auth0:**
 - Auth0 je cloud servis koji pruža skup API-a i alata koji omogućuju Single Sign-On i upravljanje korisnicima za bilo koji tip aplikacije. To može biti API, IoT uređaj i pruža povezivanje sa bilo kojim provajderom identiteta (društvene mreže, enterprajs rešenja kao i specifično napravljena rešenja)
 - Sporiji je za vraćanje uloženog novca, skuplji je
 - Ima bolju korisničku podršku
- **OneLogin:**
 - OneLogin pojednostavljuje upravljanje identitetom putem sigurnog pristupa za zaposlene, korisnike i partnere putem svih vrsta uređaja, kako bi pristupili svim servisima u oblaku i on-premises.
 - Podržava Single Sign-On
 - Sporiji je za vraćanje uloženog novca, skuplji je

- Ima bolju korisničku podršku
- **FusionAuth:**
 - Pruža brzo rešenje za implementiranje autentifikacije, autorizacije, upravljanje korisnicima, kreiranje izvrštaja, analizu i još dosta funkcionalnosti
 - Pruža podršku za SSO
 - Lakši je za podešavanje i postavljanje
 - Lakši je za poslovanje

6. ULOGA OSTALIH MIKROSERVISA

Core API je mikroservis koji sadrži osnovnu biznis logiku Eventella sistema. Implementira CRUD operacije nad tri glavna entiteta. To su događaji, lokacije i resursi. Podaci se čuvaju u PostgreSQL bazi.

Ticketing API je mikroservis koji je zamišljen kao servis preko kog bi korisnici platforme bili u mogućnosti da kupe karte za određene događaje. Za ovu svrhu su implementirani primeri nekoliko endpointa kako bi se pokazale mogućnosti rutiranja zahteva i autentifikacije na primeru više mikroservisa.



Slika 38. Ticketing API

Pored ova dva mikroservisa zamišljeno je da Eventella platforma implementira i mikroservis za analitiku. U nastavku su navedene neke od potencijalnih uloga ovog mikroservisa:

















- prikupljanje podataka o interakcijama korisnika sa platformom, kao što su klikovi, pregledi stranica, kupovine karata i slično.
- nakon prikupljanja podataka mogućnost analiziranja podataka kako bi se generisale korisne informacije poput trendova u korišćenju platforme, navikama korisnika ili efikasnosti marketinških kampanja.
- generisanje različite vrste izveštaja i dijagrama koji omogućavaju uvid u analitičke metrike, performanse ili druge relevantne informacije za menadžment platforme.

Integracioni mikroservis bi u perspektivi implementirao dodatne neophodne funkcionalnosti prilikom integracije sistema sa zainteresovanim strankama.

Eventella event management system 0.1.0 OAS3

/openapi.json

Rest API service which demonstrate in practice FastAPI possibilities

			Authorize 
event			^
POST	/event	Create the new event	▼ 
GET	/event/all	Get all events	▼ 
GET	/event/{event_id}	Get the new event	▼ 
PUT	/event/{event_id}	Update the existing event	▼ 
DELETE	/event/{event_id}	Delete the event	▼ 
location			^
POST	/location	Create the new location	▼ 
GET	/location/all	Get all locations	▼ 
GET	/location/{location_id}	Get the location	▼ 
PUT	/location/{location_id}	Update the existing location	▼ 
DELETE	/location/{location_id}	Delete the location	▼ 
resource			^
POST	/resource	Create the new resource	▼ 
GET	/resource/all	Get all resources	▼ 
GET	/resource/{resource_id}	Get the resource	▼ 
PUT	/resource/{resource_id}	Update the existing resource	▼ 
DELETE	/resource/{resource_id}	Delete the resource	▼ 
default			^
GET	/	Root	▼

Slika 39. CoreAPI

ZAKLJUČAK

U ovom radu analizirane su najvažnije komponente i funkcionalnosti Eventella platforme za upravljanje događajima, kao i arhitektura sistema koja je omogućila implementaciju ove platforme. Fokus je bio na API Gateway-u kao delu sistema koji spaja sve mikroservise, koji omogućava efikasno upravljanje i kontrolu pristupa resursima putem različitih autentifikacionih mehanizama, ograničavanja broja zahteva, rutiranja, keširanja, analitike, transformacije podataka, upravljanja verzijama i logiranja.

Pored toga, detaljno je istražena i integracija sa Keycloak servisom, koji predstavlja komponentu za upravljanje identitetom i autentifikaciju korisnika. Kroz analizu osnovnih koncepata, terminologije i funkcionalnosti Keycloak-a, objašnjene su mogućnosti single-sign on (SSO), upravljanja identitetom preko društvenih mreža, federacije korisnika, administratorske i korisničke konzole. Poseban fokus bio je na korišćenju OpenID Connect-a za obezbeđivanje aplikacija i usluga, sa pregledom dostupnih endpointa i podržanih protokola.

Pažnja je posvećena i ostalim mikroservisima u sistemu, obzirom na njihovu važnu ulogu u podršci Eventella platforme i pružanju dodatnih funkcionalnosti. Objašnjena je uloga tih mikroservisa i ideja za njihovo proširenje u budućnosti.

U zaključku, može se istaći da su analizirane komponente i funkcionalnosti Eventella platforme i pripadajućih servisa vrlo važne za dobijanje efikasnog, skalabilnog i sigurnog sistema za upravljanje događajima. Rad je pružio dobru osnovu za dalju nadogradnju sistema. Smer u kome je moguće dalje unaprediti Eventella platformu jeste dalji razvoj mikroservisa i omogućavanje dodatnih funkcionalnosti u skladu sa potrebama klijenata.

Najznačajnije obrađene celine su:

- primer organizacije mikroservisne arhitekture na primeru Eventella sistema
- uloga i mogućnosti API Gateway-a u mikroservisnoj arhitekturi
- načini integracije sa Keycloak servisom za upravljanje identitetom

LITERATURA

1. *Solo*, <https://www.solo.io/topics/api-gateway/api-gateway-authentication/>
2. *System Design*, <https://systemsdesign.cloud/SystemDesign/RateLimiter>
3. *Oracle*,
https://docs.oracle.com/en-us/iaas/Content/APIGateway/Tasks/apigatewaydynamicroutingbasedonrequest_topic.htm
4. *Amazon*,
<https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-metrics-and-dimensions.html>
5. *Medium*,
<https://medium.com/@sumanthreddy01/best-practices-for-api-versioning-and-the-role-of-the-api-gateway-53b4340a71f4>
6. *Dzone*, <https://dzone.com/articles/what-is-keycloak-and-when-it-may-help-you>
7. *Keycloak*, <https://www.keycloak.org/>
8. *wjw465150*,
https://wjw465150.gitbooks.io/keycloak-documentation/content/server_admin/topics/overview/concepts.html
9. *Pingidentity*.
<https://www.pingidentity.com/en/resources/identity-fundamentals/authentication-authorization-standards/openid-connect.html>
10. *Keycloak*, https://www.keycloak.org/docs/latest/securing_apps/index.html#_oidc
11. *Keycloak*, https://www.keycloak.org/docs/latest/server_development/#_default-themes
12. *G2*, <https://www.g2.com/products/keycloak/competitors/alternatives>
13. *Zluri*, <https://www.zluri.com/blog/keycloak-alternatives/>