

Bezbedan razvoj web aplikacije koristeći Python Flask i OWASP Top 10 smernice

Uvod

Razvoj veb aplikacija, pored funkcionalnosti i korisničkog iskustva, zahteva i visok stepen sigurnosti kako bi se zaštitili korisnici i njihovi podaci. U digitalnom svetu, veb aplikacije predstavljaju čestu metu za hakere, zbog čega je ključno da programeri prilikom razvoja aplikacije posebnu pažnju posvete aspektu sigurnosti.

Razvoj veb aplikacije prikazan je koristeći *Python Flask framework*, primenu *OWASP Top 10* smernica. *OWASP (Open Web Application Security Project)* je globalna zajednica koja definiše standarde za sigurnost veb aplikacija. Njihov *Top 10* projekat predstavlja listu najkritičnijih sigurnosnih rizika sa kojima se web aplikacije suočavaju.

U aplikaciji su implementirane osnovne sigurnosne funkcionalnosti, uključujući autentifikaciju, autorizaciju, ograničavanje broja zahteva (*rate limiting*), verifikaciju korisnika putem imejla, *hashovanje* lozinki, zahtev za novom lozinkom, *logovanje*, zaštita sesije, i druge. Kroz ove funkcionalnosti, aplikacija pruža primer kako se *OWASP Top 10* smernice mogu primeniti u praksi za zaštitu od najčešćih sigurnosnih pretnji.

Zajedno sa *Python Flask framework-om*, koriste se i sledeće tehnologije i alati:

- *SQLite* baza podataka - *SQLite* je lagana SQL baza podataka koja ne zahteva server i sve podatke smešta u jedan fajl na disku.
- *Redis* baza podataka - *in-memory* baza podataka koja se koristi za skladištenje podataka u ključ-vrednost formatu.
- *SQLAlchemy* - *ORM (Object-Relational Mapping)* biblioteka za rad sa bazama podataka.
- *Jinja2* - Šablonski jezik za renderovanje HTML-a.
- *WTForms* - Biblioteka za kreiranje i validaciju obrazaca.
- *Flask-Limiter* - Ekstenzija za ograničavanje zahteva prema aplikaciji (*rate limiting*).
- *Flask-Mail* - Ekstenzija za slanje imejl poruka iz *Flask* aplikacije.
- *Flask-Login* - ekstenzija za *Flask* koja omogućava jednostavno upravljanje autentifikacijom korisnika i sesijama

Cilj ovog dokumenta je da programerima i stručnjacima za sigurnost pruži jasan i konkretan vodič za izgradnju bezbednih veb aplikacija koristeći *Flask* i najbolje prakse definisane od strane *OWASP-a*.

Sadržaj

1. Autentifikacija.....	4
1.1. Šta je autentifikacija?	4
1.2. Flask-Login	4
1.3. User model klasa	5
1.4. Upravljanje sesijama	5
1.5. Dvofaktorska autentifikacija (2FA).....	9
1.6. Implementacija 2FA.....	9
2. Sigurno skladištenje podataka.....	12
2.1. Šta je hešovanje?	12
2.2. Hešovanje lozinki	12
2.3. Sigurno skladištenje identifikatora sa UUID-om	13
3. Autorizacija.....	14
3.1. Šta je autorizacija?	14
3.2. Implementacija RBAC	15
4. Bezbednost Lozinki: Metode i alati za proveru.....	17
4.1. Procena jačine lozinke (Strength meter i zxcvbn)	17
4.2. Provera kompromitovanosti lozinke (Have I Been Pwned).....	20
5. Ograničavanje broja zahteva (Rate Limiting).....	20
5.1. Flask-Limiter.....	21
5.2. Zaključavanje korisničkog naloga	24
6. Verifikacija korisničkog naloga.....	26
6.1. Implementacija verifikacije putem e-pošte	27
6.2. Biblioteka itsdangerous	28
7. Zaboravljena lozinka	30
7.1. Implementacije resetovanja lozinke.....	30
8. CSRF (Cross-Site Request Forgery).....	34
8.1. Scenario CSRF napada	35
8.2. Zaštita formi od CSRF napada	35
9. Eskalacija privilegija	36
9.1. Horizontalna eskalacija privilegija	36

1. Autentifikacija

1.1. Šta je autentifikacija?

Autentifikacija je proces provere identiteta korisnika, sistema ili uređaja kako bi se osiguralo da su oni zaista ono za šta se predstavljaju. Ovaj proces obično uključuje verifikaciju kredencijala kao što su korisničko ime i lozinka, biometrijski podaci (npr. otisak prsta), ili jedinstveni kodovi dobijeni putem dvofaktorske autentifikacije (2FA).

- **Kredencijali:** Informacije koje korisnik daje kako bi dokazao svoj identitet. Najčešći primeri su korisničko ime i lozinka, ali mogu uključivati i biometrijske podatke (npr. otisak prsta), ili dvofaktorsku autentifikaciju (2FA), gde korisnik unosi dodatni kod koji dobije putem SMS-a ili imejla.
- **Proces:** Kada korisnik unese svoje kredencijale, sistem ih proverava upoređujući ih sa onima koji su pohranjeni u bazi podataka. Ako se poklapaju, korisnik se uspešno autentifikuje i može pristupiti resursima za koje ima ovlašćenje.

Autentifikacija je prvi korak za postizanje sigurnosti veb aplikacije jer osigurava da je korisnik ono za šta se predstavlja.

1.2. Flask-Login

Flask-Login je ekstenzija koja se koristi za implementaciju autentifikacije u *Flask* veb aplikaciji. Pruža funkcionalnosti za **upravljanje korisničkim sesijama**, **zaštitu ruta** i **praćenje stanja** prijavljenog korisnika.

- **Upravljanje korisničkim sesijama:** *Flask-Login* omogućava jednostavno upravljanje sesijama korisnika. Kada se korisnik prijavi, ekstenzija kreira sesiju koja se koristi za praćenje njegovog stanja dok je prijavljen na svoj korisnički nalog.
- **Zaštita ruta:** Omogućava jednostavnu zaštitu ruta koristeći dekoratore kao što je `@login_required`, koji osigurava da samo autentifikovani korisnici mogu pristupiti određenim delovima aplikacije.
- Ekstenzija omogućava lako pristupanje informacijama o trenutnom korisniku putem `current_user` objekta, koji sadrži podatke o prijavljenom korisniku.

Pored osnovne autentifikacije pomoću korisničkog imena i lozinke, ekstenzija omogućava lako proširenje za različite metode autentifikacije, kao što su *OAuth*, autentifikacija bazirana na tokenima, i druge.

1.3. User model klasa

Prilikom modelovanja klase koja predstavlja korisnika veb aplikacije, potrebno je da klasa *User* nasledi klasu *UserMixin*. Nasleđena klasa dodaje metode koje su neophodne za rad sa *Flask-Login* ekstenzijom, kao što su *get_id()*, *is_authenticated()*, *is_active()* i *is_anonymous()*. Atributi klase koji se koriste za autentifikaciju su *email* i *password*.

- Imejl je modelovan kao *string* sa maksimalnom dužinom od 32 karaktera, mora biti jedinstven u bazi podataka i omogućena je pretraga putem indeksa.
- Lozinka je modelovan kao *string* od 128 karaktera i predstavlja *hešovanu* vrednost, o čemu će dalje biti više reči.

```
class User(db.Model, UserMixin):
    id = db.Column(db.String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    first_name = db.Column(db.String(24))
    last_name = db.Column(db.String(24))
    username = db.Column(db.String(24), index=True, unique=True)
    email = db.Column(db.String(32), index=True, unique=True)
    password = db.Column(db.String(128))
    birth_date = db.Column(db.Date)
    posts = db.relationship('Post', backref='author', lazy='dynamic', cascade="all, delete-orphan")
    role = db.Column(db.String(20), default='Reader')
    is_verified = db.Column(db.Boolean, default=False)
```

1.3.1. Izgled User model klase

1.4. Upravljanje sesijama

Flask koristi kolačić (cookie) za identifikaciju sesije korisnika. Kada se korisnik prijavi, *Flask* postavlja kolačić u pregledaču korisnika koji sadrži jedinstveni identifikator sesije. Ovaj kolačić omogućava *Flask* aplikaciji da prepozna korisnika u budućim zahtevima i održi stanje sesije tokom interakcije sa aplikacijom. *Flask*-ova sesija je sigurno sačuvana pomoću serijalizacije i enkripcije. Podaci sesije se serijalizuju u JSON format pre nego što se pohrane u kolačić. Da bi se obezbedila sigurnost podataka i sprečilo njihovo neovlašćeno menjanje, *Flask* koristi enkripciju. Enkripcija se vrši pomoću tajnog ključa aplikacije (*app.secret_key*), koji se mora postaviti na sigurno i nasumično generisanu vrednost u konfiguraciji aplikacije. Ovaj tajni ključ osigurava da podaci u kolačiću nisu samo zaštićeni od neovlašćenog pristupa, već i da se spreči njihovo neovlašćeno menjanje.

The image shows a login form titled "Sign In" centered within a white rounded rectangle, which is itself set against a teal background. The form contains two input fields: "Email" and "Password". The "Password" field includes a toggle icon (an eye) on its right side. Below the input fields is a teal "Login" button. At the bottom of the form, there are two blue links: "Don't have an account yet? Sign Up" and "Reset Password".

1.4.1. Forma za prijavu

Proces autentifikacije počinje sa zahtevom za prijavljivanje (logovanje). Prvo se proverava da li su prosleđeni imejl adresa i lozinka u ispravnom formatu. Ako podaci nisu u odgovarajućem formatu, podiže se *InvalidParameterException* izuzetak kako bi se obavestio korisnik o grešci prilikom unosa. Nakon provere formata, sistem pristupa bazi podataka kako bi proverio da li postoji korisnik sa datim imejlom. Ako korisnik nije pronađen, podiže se *EntityNotFoundException* izuzetak, što ukazuje na to da ne postoji korisnik sa tim imejlom. Ako korisnik postoji, prosleđena lozinka se hešuje koristeći isti algoritam koji je korišćen za hešovanje lozinke prilikom njenog skladištenja u bazi podataka. Hešovana lozinka se zatim upoređuje sa hešovanom lozinkom koja je pohranjena u bazi podataka. Ako se hešovane vrednosti poklapaju, korisnik je uspešno autentifikovan. Ako ne, prijava se odbacuje i korisniku se daje informacija o neispravnim kredencijalima.

Proces autentifikacije je detaljno prikazan na slici 1.4.2, koja ilustruje sve korake i odluke koje se donose tokom prijavljivanja korisnika, uključujući eventualne izuzetke i odgovore sistema.

```

if not email or not isinstance(email, str):
    raise InvalidParameterException("email", "Invalid or missing parameter")

if not password or not isinstance(password, str):
    raise InvalidParameterException("password", "Invalid or missing parameter")

try:
    user = self.user_service.get_user_by_email(email)
    if not user.is_verified:
        raise AccountNotVerifiedError()

    user_id = user.id
    logout_key = f"logout:{user_id}"
    failed_attempts_key = f"failed_attempts:{user_id}"

    if self.redis_client.get(logout_key):
        raise AccountLockedException()

    if password_utils.check_password(password, user.password):
        self.redis_client.delete(failed_attempts_key)
        return user
    else:
        failed_attempts = self.redis_client.incr(failed_attempts_key)
        if failed_attempts >= 5:
            self.redis_client.set(logout_key, "locked", ex=15*60)
            raise AccountLockedException()
        raise InvalidPasswordException('Password does not match')

except (InvalidParameterException, EntityNotFoundError, DatabaseServiceError) as e:
    raise e
except Exception as e:
    raise e

```

1.4.2. Koraci autentifikacije korisnika

```

def check_password(password: str, hashed_password: str) -> bool:
    return bcrypt.checkpw(password.encode('utf-8'), hashed_password.encode('utf-8'))

```

1.4.3. Provera podudaranja lozinki

Nakon uspešne provere unetih kredencijala, korisnik se prijavljuje pozivanjem metode *login_user(user)* koja kreira sesiju za korisnika (slika 1.4.4.). Identifikator korisnika se pohranjuje u sesiju korisnika i sesija se smešta u kolačić (*cookie*) pregledača korisnika. Pri svakom sledećem zahtevu ka veb aplikaciji, *Flask-Login* proverava da li postoji aktivna sesija i da li u njoj postoji identifikator korisnika. Ukoliko postoji aktivna sesija, pomoću *load_user metode* (slika 1.4.7.) iz baze podataka se dobavlja korisnik sa datim identifikatorom i smešta se u *current_user* objekat.

```

user = auth_service.authenticate(email, password)
if user:
    if user.role == 'Admin':
        otp_token, generated_time = email_service.send_otp(user.email)
        session['otp_token'] = otp_token
        session['user_id'] = user.id
        session['otp_generated_time'] = generated_time
        return redirect(url_for('auth.verify_otp'))
    login_user(user)
    return redirect(url_for('main.index'))
else:
    flash('Wrong email or password', 'error')

```

1.4.4. Kreiranje sesije za autentifikovanog korisnika

Name	Value
session	.eljFz81qFUEQBeB3ma32pbq6uqtqdoG4SPBCxjCjoZuifanMTTGSmg5GQd8-lius6H-fU87T01babaR7ro72dlIOb5qIDZE0IFjDi1A21gHCtksRH9U2yFpWkAfdLrAljtV4zFw6cVDioRGvGCSijejbMGrh47Zy1Zbxj...

1.4.5. Sesija ulogovanog korisnika

Da bi omogućili da određene funkcionalnosti budu dostupne samo autentifikovanim korisnicima, neophodno je zaštititi rute. Zaštita ruta se postiže dodavanjem dekoratora `@login_required` iznad definisanja rute (slika 1.4.6.). Rute koje omogućavaju korisniku da se prijavi (uloguje) ili da kreira svoj korisnički nalog ne treba da budu zaštićene, jer treba da omoguće pristup anonimnim korisnicima, odnosno korisnicima koji nisu autentifikovani (slika 1.4.8.).

```

@main_bp.route('/')
@login_required
def index():

```

1.4.6. Primer zaštićenih ruta

```

@auth_bp.route('/logout')
@login_required
def logout():

```

```

@login_manager.user_loader
def load_user(user_id):
    user_service = current_app.user_service
    return user_service.get_user(user_id)

```

1.4.7. User loader metoda

```

@auth_bp.route('/login', methods=['GET', 'POST'])
@current_app.limiter.limit("10 per minute")
def login():

```

```

@auth_bp.route('/register', methods=['GET', 'POST'])
@current_app.limiter.limit("5 per hour")
def register():

```

1.4.8. Primer anonimnih ruta

Sesija autentifikovanog korisnika se zatvara kada se korisnik odjavi sa svog korisničkog naloga. Pozivom metode `logout_user()` (slika 1.4.9.) uklanjaju se podaci korisnika iz sesije, a sesija se briše iz kolačića (*cookies*). Nakon odjavljivanja, korisnik se ponovo preusmerava na stranicu za prijavu.

```
@auth_bp.route('/logout')
@login_required
def logout():
    logout_user()
    return redirect(url_for('auth.login'))
```

1.4.9. Ruta za odjavu korisnika

1.5. Dvofaktorska autentifikacija (2FA)

Dvofaktorska autentifikacija (2FA) je sigurnosni proces koji zahteva dva različita faktora za verifikaciju identiteta korisnika prilikom prijave na neki sistem. Cilj dvofaktorske autentifikacije je da poboljša sigurnost tako što dodaje dodatni sloj zaštite, čime se smanjuje rizik od neovlašćenog pristupa, čak i ako napadač zna korisničko ime i lozinku.

Ključni faktori u dvofaktorskoj autentifikaciji:

- **Nešto što korisnik zna:** Ovo je prva linija odbrane i obično podrazumeva korisničko ime i lozinku ili PIN kod. Ovaj faktor je najčešće korišćen u tradicionalnoj autentifikaciji.
- **Nešto što korisnik ima:** Drugi faktor autentifikacije uključuje posedovanje fizičkog uređaja ili sredstva kao što su mobilni telefon, token ili pametna kartica. Na primer, korisnik može dobiti jednokratni kod (OTP) putem SMS-a, aplikacije za autentifikaciju ili imejl koji mora uneti prilikom prijave.
- **Nešto što korisnik jeste:** U nekim slučajevima, treći faktor može biti biometrijska verifikacija kao što su otisak prsta, prepoznavanje lica, ili skeniranje dužice oka. Ovaj faktor je dodatno sredstvo koje se koristi u naprednijim sigurnosnim sistemima.

1.6. Implementacija 2FA

Dvofaktorska autentifikacija je implementirana korišćenjem jednokratne lozinke (*OTP*) koja se šalje putem imejla. Kada se korisnik sa ulogom administratora uspešno prijavi unošenjem svoje imejl adrese i lozinke, dobija šestocifreni broj na imejl, koji mora uneti da bi potvrdio svoj identitet. Na osnovu jednokratne lozinke generiše se **potpisani token** koji se čuva u kolačićima (*cookies*) korisnikovog pregledača (slika 1.6.3.), zajedno sa identifikatorom korisnika i vremenom generisanja lozinke. Potpisani token se kreira iz

jednokratne lozinke (slika 1.6.1.) korišćenjem serijalizatora URLSafeTimedSerializer iz biblioteke itsdangerous. Potpisani token obezbeđuje integritet i potvrdu identiteta.

```
def generate_otp_token(otp, s):  
    return s.dumps(otp, salt='otp-salt')
```

1.6.1. Generisanje OTP tokena

```
def verify_otp_token(token, s, expiration=60):  
    try:  
        otp = s.loads(token, salt='otp-salt', max_age=expiration)  
        return otp  
    except SignatureExpired:  
        raise SignatureExpired("The OTP token has expired.")  
    except BadSignature:  
        raise BadSignature("The OTP token is invalid.")  
    except Exception as e:  
        raise Exception(f"An error occurred: {str(e)}")
```

1.6.2. Verifikacija OTP tokena

Korisnik unosi jednokratnu lozinku (OTP) koja mu je stigla na imejl (slika 1.6.5.). Ova lozinka se poredi sa vrednošću koja se dobija dekripcijom OTP tokena sačuvanog u kolačićima korisnikovog pregledača. Trajanje OTP-a je ograničeno na šezdeset sekundi. Ako vreme isteka protekne, korisnik mora ponovo zatražiti slanje novog OTP-a putem imejla kako bi mogao da prođe drugi sloj zaštite. Kada korisnik unese ispravnu OTP lozinku, svi podaci uskladišteni u kolačićima (uključujući OTP token, identifikator korisnika i vreme generisanja) se brišu, a korisnik se uspešno prijavljuje na svoj nalog.

```
user = auth_service.authenticate(email, password)  
if user:  
    if user.role == 'Admin':  
        otp_token, generated_time = email_service.send_otp(user.email)  
        session['otp_token'] = otp_token  
        session['user_id'] = user.id  
        session['otp_generated_time'] = generated_time  
        return redirect(url_for('auth.verify_otp'))  
    login_user(user)  
    return redirect(url_for('main.index'))  
else:  
    flash('Wrong email or password', 'error')
```

1.6.3. Kod za slanje OTP-a preko imejla

```

if saved_otp is None:
    flash('The OTP is either invalid or expired.', 'error')
    return redirect(url_for('main.info'))

if otp == saved_otp:
    user_id = session.get('user_id')
    user = user_service.get_user(user_id)
    if user:
        login_user(user)
        session.pop('otp_token', None)
        session.pop('user_id', None)
        return redirect(url_for('main.index'))
    else:
        flash('Invalid OTP. Try again.', 'error')
else:
    flash('Invalid OTP. Try again.', 'error')

```

1.6.4. Deo koda zaproveru OTP-a

1.6.5. Korisnički interfejs za verifikaciju OTP-a

2. Sigurno skladištenje podataka

2.1. Šta je hešovanje?

Hešovanje je kriptografska tehnika koja se koristi za pretvaranje ulaznih podataka (poput lozinki, poruka, datoteka) u fiksni niz karaktera, koji je poznat kao "heš" ili "sažetak." Heš funkcije su matematičke funkcije koje uzimaju proizvoljno veliku količinu podataka i proizvode sažetak fiksne veličine. Ključne karakteristike heš funkcija su:

- **Determinističnost:** Za isti ulaz, heš funkcija će uvek generisati isti izlaz (*hash*). To znači da će, ako unesete istu lozinku, ona uvek biti heš na isti način.
- **Jednosmernost:** Heš funkcije su jednosmerne, što znači da je praktično nemoguće obrnuti proces hešovanja i dobiti originalne podatke iz heša. Ovo svojstvo je ključna komponenta sigurnosti.
- **Brzina:** Heš funkcije su dizajnirane da budu brze, tako da se mogu koristiti u realnom vremenu bez značajnog uticaja na performanse sistema.
- **Različitost izlaza:** Čak i najmanja promena ulaznih podataka trebalo bi da rezultira potpuno drugačijim hešom, što se naziva "**efekat lavine**".

2.2. Hešovanje lozinki

Prilikom skladištenja lozinki u bazama podataka, neophodno je obratiti pažnju na format u kojem se one čuvaju. Skladištenje lozinki u običnom tekstu (*plain text*) nije preporučljivo, jer bi napadač koji dobije pristup toj bazi podataka mogao zloupotребiti lozinke za pristup nalogima korisnika na drugim platformama. Pošto čuvanje lozinki u formatu običnog teksta predstavlja ozbiljan bezbednosni rizik, koristi se postupak hešovanja lozinki.

Hešovanja osigurava da su lozinke u bazi podataka sačuvane u formatu koji zlonamerni napadač ne može pročitati, a jednosmernost heš funkcija onemogućava primenu obrnutog postupka za dobijanje originalne lozinke. Dodatno, upotreba soli (*salt*) – nasumično generisanog niza karaktera koji se dodaje lozinci pre hešovanja – dodatno pojačava bezbednost. Nasumične *salt* vrednosti sprečavaju napade pomoću predefinisanih heš vrednosti (poznate kao *rainbow tables*) i čine svaki heš jedinstven, čak i ako se koriste iste lozinke.

Takođe, preporučuje se korišćenje modernih, sigurnih algoritama za hešovanje kao što su *bcrypt*, *Argon2*, ili *PBKDF2*. Ovi algoritmi su dizajnirani da budu otporni na *brute-force* napade i imaju ugrađene mehanizme za dodavanje nasumičnih *salt* vrednosti i prilagođavanje složenosti hešovanja.

Za potrebe hešovanja lozinki upotrebljena je bcrypt biblioteka. Definisane su dve pomoćne metode:

- *hash_password(password)* – Metoda koja kao parametar prima lozinku koju je korisnik uneo i vraća hešovanu vrednost te lozinke. Tokom procesa hešovanja, dodaje se nasumična *salt* vrednost kako bi se povećala sigurnost.
- *check_password(password, hashed_password)* – Metoda koja prima dva parametra: prvi je lozinka koju je korisnik uneo, a drugi je hešovana lozinka preuzeta iz baze podataka. Ova metoda hešuje unetu lozinku i poredi je sa heš vrednošću iz baze. Funkcija vraća rezultat poređenja, koji pokazuje da li uneta lozinka odgovara hešovanoj vrednosti.

```
import bcrypt

def hash_password(password):
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
    return hashed_password.decode('utf-8')

def check_password(password: str, hashed_password: str) -> bool:
    return bcrypt.checkpw(password.encode('utf-8'), hashed_password.encode('utf-8'))
```

2.2.1. Kod za hešovanje lozinke

461b3c69-11a1-4bcb-b79a-f4...	Mirko	Mirkovic	mirko123	mirko@gmail.com	\$2b\$12\$TamnuQt7/lr6GWWfR...	1989-11-10	Author	1
-------------------------------	-------	----------	----------	-----------------	--------------------------------	------------	--------	---

2.2.2. Hešovana lozinka u bazi podataka

2.3. Sigurno skladištenje identifikatora sa UUID-om

U implementaciji modela baze podataka, entiteti se identifikuju putem jedinstvenih identifikatora, koji su često numeričke vrednosti koje se automatski povećavaju sa svakim novim zapisom. Ovakva implementacija je problematična iz više razloga:

- **Predvidljivost:** Ako je napadač u mogućnosti da nasumično pogodi identifikatore (npr. 1, 2, 3, itd.), može pokušati da pristupi različitim resursima samo promenom broja u URL-u (/users/profile/1), što može dovesti do neželjenog pristupa.
- **Kolizije:** Ako aplikacija koristi više baza podataka ili distribuciju podataka, može doći do konflikta u generisanju identifikatora ako više instanci aplikacije pokušava da kreira zapise istovremeno.
- **Sigurnost i privatnost:** Numerički identifikatori mogu otkriti informacije o redosledu kreiranja zapisa ili broj korisnika, što može biti korisno napadaču za dalje planiranje napada.

Da bi se izbegli ovi problemi, preporučuje se upotreba UUID (*Universally Unique Identifier*). UUID je standard za identifikaciju koji pruža jedinstvene vrednosti koje se koriste za označavanje zapisa u bazi podataka. U implementaciji baze podataka za ovu veb aplikaciju, identifikatori se generišu i skladište koristeći UUID (slika 2.3.1.), gde svaki identifikator predstavlja *string* dužine trideset i šest (36) karaktera i generiše se sa svakim novim zapisom u bazi podataka. Upotreba UUID-a omogućava da svaki identifikator bude globalno jedinstven, smanjujući mogućnost podudaranja identifikatora i povećava sigurnost aplikacije.

```
id = db.Column(db.String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
```

2.3.1. Generisanje identifikatora

461b3c69-11a1-4bcb-b79a-f4...	Mirko	Mirkovic	mirko123	mirko@gmail.com	\$2b\$12\$TamnuQt7/lr6GWWfR...	1989-11-10	Author	1
-------------------------------	-------	----------	----------	-----------------	--------------------------------	------------	--------	---

2.3.2. Identifikator u bazi podataka

3. Autorizacija

3.1. Šta je autorizacija?

Autorizacija je proces kojim se određuje nivo pristupa i privilegija korisnika ili sistema u okviru određenog resursa ili aplikacije. Za razliku od autentifikacije, koja potvrđuje identitet korisnika, autorizacija određuje šta taj korisnik sme da radi unutar sistema, na osnovu njegovih dozvola i uloga. Na primer, u veb aplikaciji, autorizacija može odrediti da li korisnik može da pristupi određenim stranicama, izvršava određene akcije, ili upravlja podacima, na osnovu njegovih prava unutar aplikacije.

Postoji nekoliko vrsta autorizacije koje se koriste u različitim sistemima:

- **Role-Based Access Control (RBAC)** - Korisnicima se dodeljuju uloge, a svaka uloga ima unapred definisane privilegije. Ova vrsta autorizacije se često koristi u preduzećima i aplikacijama gde su prava pristupa standardizovana prema poslovnim funkcijama.

	Uloga	Pristup
Politika 1	<i>Administrator</i>	<i>Može pristupiti svim administrativnim funkcijama (npr. kreiranje i brisanje korisnika)</i>
Politika 2	<i>Autor, Administrator</i>	<i>Korisnici sa ulogama administrator i autor mogu kreirati i uređivati sadržaj</i>

3.1.1. Primer RBAC-a

- **Attribute-Based Access Control (ABAC)** - Pristup se kontroliše na osnovu skupa atributa povezanih sa korisnicima ili resursima. Atributi mogu uključivati vreme, lokaciju, ulogu korisnika, ili druge kontekstualne informacije. ABAC omogućava visoku granularnost u kontroli pristupa.

	Atribut	Pravilo
Politika 1	<i>Uloga (Menadžer), Vreme (09:00 - 17:00)</i>	<i>Ako je korisnik 'Menadžer' i trenutno vreme je unutar radnog vremena, pristup je odobren.</i>
Politika 2	<i>Odeljenje (Finansije), Tip dokumenta (budžet), Lokacija (kancelarija)</i>	<i>Ako je korisnik iz odeljenja 'Finansije', dokument je vezan za budžet, i korisnik je u kancelariji, pristup je odobren.</i>

3.1.2. Primer politika u ABAC-u

- **Policy-Based Access Control (PBAC)** - Koristi politike kao osnovu za donošenje odluka o pristupu. Politike su pravila ili uslovi koji određuju pristup resursima na temelju različitih faktora. PBAC koristi detaljnije i fleksibilnije politike koje mogu uključivati različite aspekte kao što su atributi korisnika, resursa ili okruženja. Ove politike su obično izrađene i upravljane od strane administratora sistema i mogu se primenjivati na različite scenarije i zahteve.

Politika 1	<i>Ako je korisnik 'Menadžer' i trenutno vreme je između 09:00 i 17:00, korisnik može pristupiti svim dokumentima</i>
Politika 2	<i>Ako je korisnik iz odeljenja 'Finansije', dokument je 'Finansijski izveštaj', i korisnik se nalazi u kancelariji, korisnik može pristupiti dokumentu</i>

3.1.3. Primer PBAC-a

3.2. Implementacija RBAC

Implementacija RBAC (*Role-Based Access Control*) u ovoj veb aplikaciji omogućava jasnu separaciju funkcionalnosti na osnovu uloga korisnika. Ovo pomaže u očuvanju sigurnosti aplikacije tako što osigurava da samo korisnici sa odgovarajućim ulogama mogu pristupiti specifičnim delovima aplikacije i obavljati određene zadatke.

Korisnici veb aplikacije mogu imati jednu od tri uloge: **Admin**, **Author** ili **Reader**. Na osnovu dodeljene uloge, korisnicima se omogućava različit nivo pristupa i funkcionalnosti u aplikaciji:

- **Admin:** Korisnici sa ulogom *Admin* imaju najviši nivo pristupa i kontrole unutar aplikacije. Mogu pristupiti administratorskom panelu gde mogu upravljati korisničkim nalogima, pregledati logove i obavljati druge administrativne zadatke. Takođe, *Admin* može upravljati svim sadržajem unutar aplikacije i dodeljivati ili menjati uloge drugih korisnika.

- **Author:** Korisnici sa ulogom *Author* imaju pristup funkcionalnostima koje im omogućavaju da kreiraju i objavljuju sadržaj. Mogu pisati, uređivati i objavljevati članke ili postove, ali nemaju pristup administratorskim funkcijama. Njihov pristup je ograničen na upravljanje sadržajem koji sami kreiraju.
- **Reader:** Korisnici sa ulogom *Reader* imaju najniži nivo pristupa. Mogu pregledati sadržaj koji je objavljen od strane *Author*-a ili *Admin*-a, mogu poslati zahtev da im se dodeli uloga *Author*-a ali nemaju mogućnost kreiranja ili uređivanja sadržaja. Njihov pristup je ograničen na čitanje i pregled informacija dostupnih u aplikaciji.

Prvi korak je dodavanje atributa *role* unutar klase *User* (slika 3.2.1). Ovaj atribut predstavlja ulogu korisnika i koristi se za kontrolu pristupa različitim delovima aplikacije. Kako bi zaštitili rute od pristupa korisnika koji nemaju odgovarajuća ovlašćenja, potrebno je definisati dekorator za proveru uloge korisnika (slika 3.2.2).

Dekorator *requires_roles(*roles)* kao parametar prima jednu ili više uloga koje korisnik mora imati da bi pristupio određenoj ruti. Kada se ruta pozove, dekorator vrši proveru da li uloga trenutno prijavljenog korisnika (*current_user.role*) odgovara nekoj od uloga koje su prosleđene dekoratoru. Ako korisnik nema odgovarajuću ulogu, biće preusmeren na početnu stranicu (*rutu main.index*). U suprotnom, pristup ruti će biti dozvoljen i originalna funkcija će se izvršiti.

```
class User(db.Model, UserMixin):
    id = db.Column(db.String(36), primary_key=True, default=lambda: str(uuid.uuid4()))
    first_name = db.Column(db.String(24))
    last_name = db.Column(db.String(24))
    username = db.Column(db.String(24), index=True, unique=True)
    email = db.Column(db.String(32), index=True, unique=True)
    password = db.Column(db.String(128))
    birth_date = db.Column(db.Date)
    posts = db.relationship('Post', backref='author', lazy='dynamic', cascade="all, delete-orphan")
    role = db.Column(db.String(20), default='Reader')
    is_verified = db.Column(db.Boolean, default=False)
```

3.2.1. User klasa

```
def requires_roles(*roles):
    def wrapper(f):
        @wraps(f)
        def wrapped(*args, **kwargs):
            if current_user.role not in roles:
                return redirect(url_for('main.index'))
            return f(*args, **kwargs)
        return wrapped
    return wrapper
```

3.2.2. Dekorator *require_roles*

<pre>@main_bp.route('/dashboard') @login_required @requires_roles('Admin') def dashboard():</pre>	<pre>@post_bp.route('/add_post', methods=['GET', 'POST']) @login_required @requires_roles('Admin', 'Author') def add_post():</pre>
---	--

3.2.3. Zaštita ruta od neautorizovanog pristupa

4. Bezbednost Lozinke: Metode i alati za proveru

4.1. Procena jačine lozinke (Strength meter i zxcvbn)

Jačina lozinke je ključni faktor u zaštiti korisničkih naloga od neovlašćenog pristupa. U današnjem digitalnom svetu, gde su sajber napadi sve učestaliji, značaj korišćenja snažnih i teško pogađanih lozinki nikada nije bio veći. Snažna lozinka ne samo da poboljšava sigurnost naloga, već i doprinosi ukupnoj bezbednosti sistema i podataka.

Da bi se osiguralo da lozinke korisnika zadovoljavaju visoke standarde sigurnosti, koristi se različite tehnike i alati za procenu njihove jačine. Jedan od najsavremenijih alata za ovu svrhu je *zxcvbn*, biblioteka koja nudi detaljnu analizu i ocenu lozinke na osnovu različitih kriterijuma.

Biblioteka *zxcvbn* je razvijena od strane *Dropbox*-a i pruža naprednu procenu jačine lozinke. Umesto da se oslanja na jednostavne kriterijume poput dužine lozinke ili broja različitih karaktera, *zxcvbn* koristi kompleksne algoritme i heuristike kako bi pružila preciznu analizu lozinke. Evo kako *zxcvbn* funkcioniše:

- **Procena Jačine Lozinke** - *zxcvbn* analizira različite aspekte lozinke, uključujući dužinu, složenost i prisustvo predvidivih obrazaca. Na osnovu tih faktora, biblioteka dodeljuje ocenu jačine lozinke u rasponu od 0 do 4.
- **Vrste Provera:**
 - **Dužina lozinke:** Duže lozinke pružaju veću sigurnost.
 - **Kombinacije karaktera:** Analizira raznovrsnost karaktera.
 - **Uobičajeni obrasci:** Proverava prisustvo predvidivih obrazaca.
 - **Leksikonske reči:** Proverava prisustvo reči iz rečnika.
 - **Povezanost sa ličnim podacima:** Proverava prisustvo ličnih podataka.
 - **Brzina napadača:** Simulira brzinu različitih vrsta napadača.
- **Ocena Jačine**
 - Ocene su dodeljene na osnovu procene: 0 (izuzetno predvidiva) do 4 (vrlo teško pogađana). Ova ocena pomaže korisnicima da shvate koliko je njihova lozinka zaista jaka i da li je potrebna dodatna poboljšanja.
- **Povratne Informacije**
 - *zxcvbn* pruža korisnicima povratne informacije o jačini lozinke, uključujući preporuke za poboljšanje lozinke kako bi bila sigurnija.

Korišćenje *zxcvbn* u aplikacijama omogućava razvijanje sigurnijih lozinki i smanjuje rizik od napada koji koriste loše lozinke. Ova biblioteka pruža sveobuhvatnu analizu lozinki, što je čini vrednim alatom za povećanje bezbednosti korisničkih naloga.

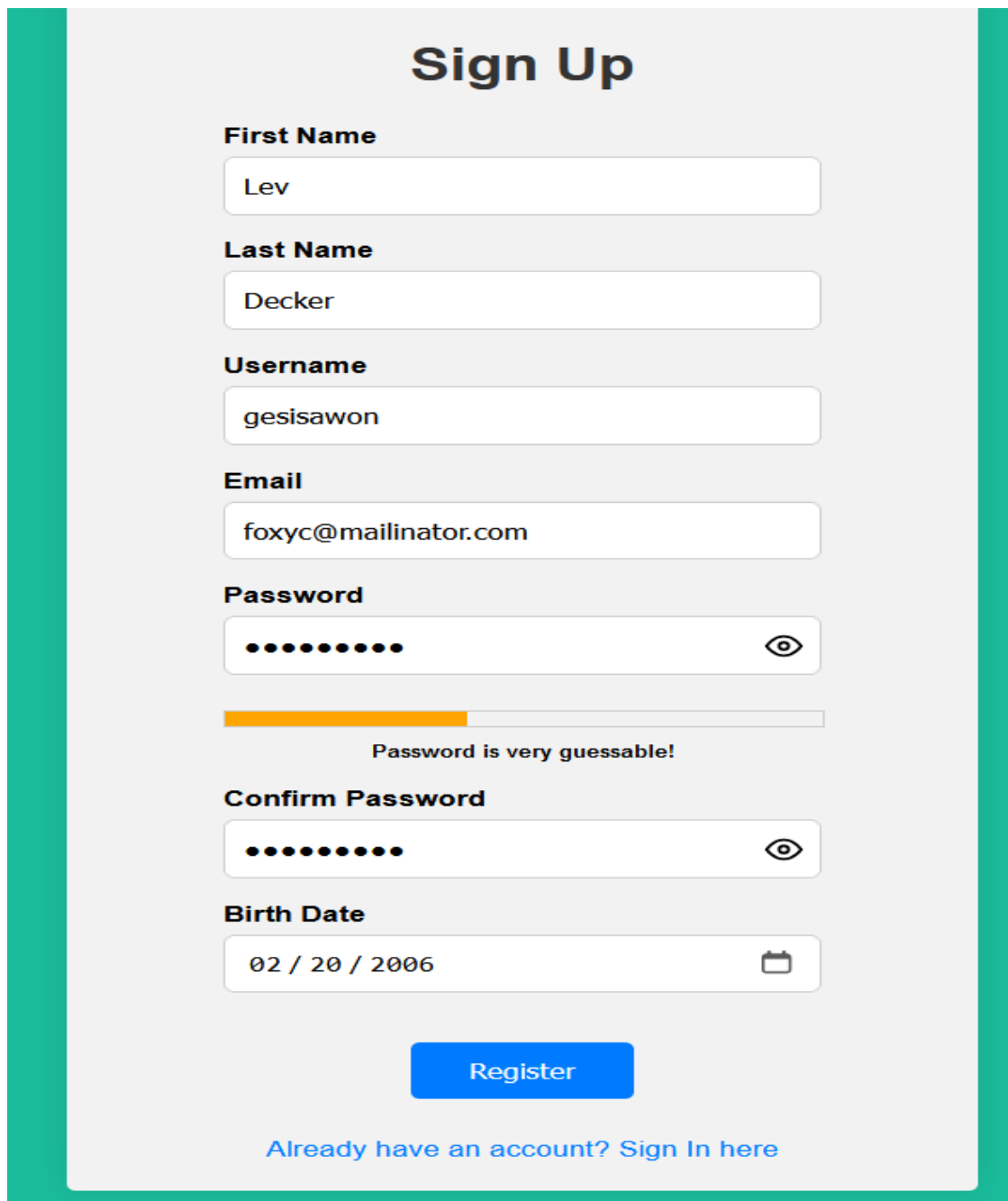
Sign Up


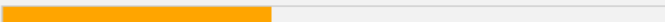
First Name

Last Name


Username


Email

Password
 

Password is very guessable!

Confirm Password
 

Birth Date
 

[Register](#)

[Already have an account? Sign In here](#)

4.1.1. Merač jačine lozinke prilikom registracije

```

function updatePasswordStrength(password) {
  const result = zxcvbn(password);
  const guessesLog10 = result.guesses_log10;
  const score = result.score;

  const strengthMeter = document.getElementById('password-strength-meter');
  const strengthBar = strengthMeter.querySelector('.strength-bar');
  const strengthText = strengthMeter.querySelector('.strength-text');

  let color = '';
  let text = '';

  switch (score) {
    case 0:
      color = 'red';
      text = 'Password is too guessable!';
      break;
    case 1:
      color = 'orange';
      text = 'Password is very guessable!';
      break;
    case 2:
      color = 'yellow';
      text = 'Password is somewhat guessable!';
      break;
    case 3:
      color = 'greenyellow';
      text = 'Password is safely unguessable!';
      break;
    case 4:
      color = 'green';
      text = 'Password is very unguessable!';
      strengthBar.style.width = '100%';
      break;
    default:
      color = 'gray';
      text = 'Password Strength';
  }

  if (score < 4) {
    strengthBar.style.width = `${(guessesLog10 / 10) * 100}%`;
  }
  strengthBar.style.backgroundColor = color;
  strengthText.textContent = text;
}

```

4.1.2. Kod za prikaz rezultata jačine lozinke

4.2. Provera kompromitovanosti lozinke (Have I Been Pwned)

Jedan od ključnih aspekata bezbednosti pri autentifikaciji korisnika jeste provera da se lozinka nije kompromitovala u ranijim incidentima curenja podataka. Kompromitovana lozinka može predstavljati ozbiljan bezbednosni rizik, jer zlonamerni napadači često koriste zbirke ukradenih lozinki za *brute-force* napade ili napade prepoznavanja obrazaca.

Provera kompromitovanosti lozinke se može realizovati korišćenjem javno dostupnih servisa kao što je "Have I Been Pwned?". Ovaj servis omogućava proveru lozinke u okviru baze podataka kompromitovanih lozinki. U tom procesu, lozinka se najpre hešuje koristeći SHA-1 algoritam, a zatim se od prvih pet karaktera heš vrednosti obrazuje prefix na osnovu kog se vrši pretraga u bazi podataka. Ovaj pristup je poznat kao *K-Anonymity*, jer smanjuje verovatnoću otkrivanja kompletne lozinke tokom pretrage. API vraća listu heš vrednosti lozinke koje počinju sa tim prefiksom zajedno sa brojem koliko puta su se pojavili u kompromitovanim bazama podataka. Proces provere kompromitovanosti je prikazan na slici 4.2.1.

U slučaju da hash lozinke odgovara nekom unosu u bazi kompromitovanih lozinki, smatra se da je lozinka kompromitovana i korisniku se preporučuje da odabere drugu, sigurniju lozinku (slika 4.2.2.). Na taj način, sistem pomaže u sprečavanju korišćenja lozinke koje su već ranije otkrivene u nekim od sigurnosnih incidenata, značajno povećavajući bezbednost korisničkih naloga.

```
import hashlib
import requests

class PwnedService:
    def is_password_compromised(self, password):
        hashed_password = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()
        prefix, suffix = hashed_password[:5], hashed_password[5:]

        response = requests.get(f'https://api.pwnedpasswords.com/range/{prefix}')
        if response.status_code == 200:
            hashes = (line.split(':') for line in response.text.splitlines())
            for h, count in hashes:
                if suffix == h:
                    return True
        return False
```

4.2.1. Provera kompromitovanosti lozinke

5. Ograničavanje broja zahteva (Rate Limiting)

Ograničavanje broja zahteva (*Rate Limiting*) predstavlja tehniku koja se koristi u veb aplikacijama kako bi se kontrolisala učestalost kojom klijenti mogu slati zahteve serveru. Ova tehnika je od suštinskog značaja

za zaštitu aplikacije od različitih oblika zloupotreba, kao što su *brute-force* napadi na lozinke, *DDoS* napadi ili druge vrste preopterećenja servera.

Implementacija ograničavanja broja zahteva omogućava definisanje pravila koja određuju koliko zahteva jedan korisnik može poslati u određenom vremenskom periodu. Na primer, može se odrediti da jedan korisnik može poslati maksimalno 10 zahteva u minuti, a svaki dodatni zahtev će biti blokiran ili usporen.

Jedan od popularnih alata za implementaciju ove tehnike u Flask aplikacijama je *Flask-Limiter*. Ovaj alat omogućava jednostavnu konfiguraciju i primenu ograničavanja broja zahteva na globalnom nivou aplikacije, po određenim rutama ili čak prema individualnim korisnicima. Korišćenje *Flask-Limiter*-a doprinosi povećanju bezbednosti aplikacije i očuvanju stabilnosti sistema, posebno u situacijama kada se aplikacija suočava sa velikim brojem zahteva ili potencijalnim zlonamernim aktivnostima.

Integracijom ograničavanja broja zahteva u aplikaciju, sistem se proaktivno štiti od preopterećenja i napada, osiguravajući da svi korisnici mogu imati pouzdan i siguran pristup resursima

5.1. Flask-Limiter

Za korišćenje *Flask-Limiter*-a neophodno je definisati objekat *Limiter* (slika 5.1.1.) iz biblioteke *Flask-Limiter*:

- **get_remote_address**: Funkcija koja određuje na koji način će se identifikovati korisnik koji šalje zahtev. U ovom slučaju, koristi se IP adresa klijenta (tj. korisnika) kako bi se razlikovali zahtevi od različitih korisnika.
- **app=app**: Ovo povezuje limiter sa Flask aplikacijom, omogućavajući da se ograničenja primenjuju na sve rute u aplikaciji.
- **default_limits=["200 per day", "50 per hour"]**: Ova linija definiše podrazumevana ograničenja za broj zahteva:
 - "200 per day": Svaki korisnik može poslati maksimalno 200 zahteva dnevno.
 - "50 per hour": Svaki korisnik može poslati maksimalno 50 zahteva na sat.
- **storage_uri=app.config['REDIS_URL']**: Ovo određuje gde će se podaci o broju zahteva čuvati. U ovom slučaju, koristi se *Redis* kao skladište, čija se adresa nalazi u konfiguracionom parametru *REDIS_URL*. O *Redis*-u će kasnije biti više reči.

Napomena:

Identifikovanje zahteva korisnika se ne mora striktno raditi po IP adresi, moguće je identifikovanje korisnika odraditi na osnovu identifikatora korisnika (user_id), identifikatora sesije (session_id) ili kombinacijom npr. IP adrese i informacije o pretraživaču (User-Agent).

```

app.limiter = Limiter(
    get_remote_address,
    app=app,
    default_limits=["200 per day", "50 per hour"],
    storage_uri=app.config['REDIS_URL']
)

```

5.1.1. Kreiranje objekta klase Limiter

Podrazumevane vrednosti za ograničavanje broja zahteva se mogu menjati postavljanjem dekoratora iznad određene rute (slika 5.1.2.). Npr. ruta za prijavljivanje korisnika ima ograničenje od deset zahteva po minutu po IP adresi korisnika, dok je zahtev za resetovanje lozinke ograničena na tri zahteva po minutu, 10 zahteva na sat vremena i 50 zahteva na dan po IP adresi korisnika.

```

@auth_bp.route('/login', methods=['GET', 'POST'])
@current_app.limiter.limit("10 per minute")
def login():

@auth_bp.route('/reset_request', methods=['GET', 'POST'])
@current_app.limiter.limit("3 per minute; 10 per hour; 50 per day")
def reset_request():

```

5.1.2. Ograničavanje broja zahteva rutama

Broj zahteva za rute se čuva u Redis bazi podataka u formatu **ključ-vrednost**. Svaka ruta koja je podložna ograničenju broja zahteva ima jedinstveni ključ u Redis bazi, a vrednost tog ključa predstavlja broj zahteva koje je korisnik poslao.

Redis se koristi kao skladište zbog svoje brzine i efikasnosti u rukovanju velikim brojem operacija čitanja i pisanja, što je idealno za potrebe *rate limiting*-a. Redis ključevi se generišu na osnovu kombinacije identifikatora korisnika (IP adresa) i naziva rute. Ovaj format sadrži ključ koji se sastoji od prefiksa "*LIMITS*", IP adrese korisnika, specifične rute, maksimalnog broja dozvoljenih zahteva, vremenskog perioda i jedinice vremena. Na osnovu ovih vrednosti, *Flask-Limiter* upravlja ograničavanjem broja zahteva, čuvajući stanje za svakog korisnika u Redis bazi podataka. Korisnik koja je poslao više zahteva od ograničenja koje je postavljeno se preusmerava na info rutu i obaveštava da pokuša pristup željenoj ruti kasnije (slika 5.1.4.)

Na slici 5.1.3. prikazano je kako izgledaju zapisi u *Redis* bazi podataka. Komandom **keys *** izlistani su svi ključevi u bazi, a zatim je pozivanjem komande **get "LIMITS:LIMITER/127.0.0.1/auth.login/10/1/minute"** dobijen broj koji pokazuje koliko je puta korisnik sa IP adrese 127.0.0.1 pristupio ruti za prijavu u prethodnom minutu. Ponovnim pozivanjem iste komande nakon jednog minuta vraća se (*nil*), što znači da je od trenutka prvog pristupa prošao jedan minut, i da je broj zahteva počeo da se meri za novu sekvencu

vremena. Ovaj proces pokazuje kako *Flask-Limiter* koristi *Redis* za praćenje i kontrolu broja zahteva po određenoj ruti, kao i kako se resetuje brojač nakon isteka vremenskog perioda.

```
C:\Program Files\Redis\redis-cli.exe
127.0.0.1:6379> keys *
1) "LIMITS:LIMITER/127.0.0.1/auth.login/10/1/minute"
2) "LIMITS:LIMITER/127.0.0.1/auth.logout/200/1/day"
3) "LIMITS:LIMITER/127.0.0.1/main.index/200/1/day"
4) "LIMITS:LIMITER/127.0.0.1/post.add_post/200/1/day"
5) "LIMITS:LIMITER/127.0.0.1/main.index/50/1/hour"
6) "LIMITS:LIMITER/127.0.0.1/post.post_details/200/1/day"
7) "LIMITS:LIMITER/127.0.0.1/user.profile/200/1/day"
8) "LIMITS:LIMITER/127.0.0.1/auth.logout/50/1/hour"
9) "LIMITS:LIMITER/127.0.0.1/main.dashboard/200/1/day"
127.0.0.1:6379> get "LIMITS:LIMITER/127.0.0.1/auth.login/10/1/minute"
"2"
127.0.0.1:6379> get "LIMITS:LIMITER/127.0.0.1/auth.login/10/1/minute"
(nil)
127.0.0.1:6379>
```

5.1.3. Redis-CLI

Information

You have exceeded the request limit. Please try again later.

[Already have an account? Sign In here](#)

5.1.4. Poruka o prekoračenju broja zahteva

5.2. Zaključavanje korisničkog naloga

Zaključavanje naloga nakon određenog broja neuspješnih pokušaja prijave je ključna mera bezbednosti koja pomaže u zaštiti korisničkih naloga od *brute-force* napada. U kodu koji je prikazan na slici 5.2.1., postupak zaključavanja funkcioniše tako što se broj neuspješnih pokušaja prijave prati korišćenjem *Redis* ključa ***failed_attempts:{user_id}***. Kada korisnik unese pogrešnu lozinku, broj neuspješnih pokušaja se povećava. Ako broj dostigne ili pređe pet pokušaja, korisnički nalog se zaključava na petnaest minuta korišćenjem *Redis* ključa ***lockout:{user_id}***. Ukoliko je korisniku zaključan nalog, podiže se izuzetak *AccountLockedException()* i obaveštava ga da pokuša sa ponovnom prijavom kasnije (slika 5.2.2.).

```
try:
    user = self.user_service.get_user_by_email(email)
    if not user.is_verified:
        raise AccountNotVerifiedError()

    user_id = user.id
    lockout_key = f"lockout:{user_id}"
    failed_attempts_key = f"failed_attempts:{user_id}"

    if self.redis_client.get(lockout_key):
        raise AccountLockedException()

    if password_utils.check_password(password, user.password):
        self.redis_client.delete(failed_attempts_key)
        return user
    else:
        failed_attempts = self.redis_client.incr(failed_attempts_key)
        if failed_attempts >= 5:
            self.redis_client.set(lockout_key, "locked", ex=15*60)
            raise AccountLockedException()
        raise InvalidPasswordException('Password does not match')

except (InvalidParameterException, EntityNotFoundError, DatabaseServiceError) as e:
    raise e
except Exception as e:
    raise e
```

5.2.1. Kod za zaključavanje naloga

Sign In

Email

ana@gmail.com

Password

.....

Account is locked. Try again later.

Login

[Don't have an account yet? Sign Up](#)

[Reset Password](#)

5.2.2. Pokušaj prijave na zaključan nalog

Na slici 5.2.3. prikazan je izgled zapisa *Redis* baze podataka. Pozivanjem komande **get "logout:ea0adee0-8d3f-4839-ba1f-363bae119761"**, gde je *logout* prefiks, a *ea0adee0-8d3f-4839-ba1f-363bae119761* identifikator korisnika, dobijamo vrednost *"locked"*. To znači da je nalog korisnika sa datim identifikatorom trenutno zaključan. Takođe, komanda **get "failed_attempts:ea0adee0-8d3f-4839-ba1f-363bae119761"** vraća vrednost *"5"*, što ukazuje da je korisnik sa ovim identifikatorom pet puta pogrešno uneo lozinku.

Nakon isteka perioda od petnaest minuta, zapis sa ključem **logout:ea0adee0-8d3f-4839-ba1f-363bae119761** automatski se briše iz baze, a vrednost za ključ **failed_attempts:ea0adee0-8d3f-4839-ba1f-363bae119761** se vraća na nulu. Ovaj mehanizam omogućava korisniku da ponovo pokuša prijavu nakon što istekne vreme zaključavanja, čime se obezbeđuje zaštita od *brute-force* napada, ali i omogućava legitiman povratak korisniku nakon perioda greške.

```
C:\Program Files\Redis\redis-cli.exe
127.0.0.1:6379> keys *
1) "LIMITS:LIMITER/127.0.0.1/auth.logout/200/1/day"
2) "LIMITS:LIMITER/127.0.0.1/main.index/200/1/day"
3) "LIMITS:LIMITER/127.0.0.1/main.info/200/1/day"
4) "lockout:ea0adee0-8d3f-4839-ba1f-363bae119761"
5) "failed_attempts:ea0adee0-8d3f-4839-ba1f-363bae119761"
127.0.0.1:6379> get "lockout:ea0adee0-8d3f-4839-ba1f-363bae119761"
"locked"
127.0.0.1:6379> get "failed_attempts:ea0adee0-8d3f-4839-ba1f-363bae119761"
"5"
127.0.0.1:6379>
```

5.2.3. Redis CLI prikaz zaključanog naloga

Implementacija mehanizma zaključavanja naloga je važna prema OWASP Top 10 preporukama, posebno zbog toga što onemogućava napadače da koriste *brute-force* metode za pogađanje lozinke. Zaključavanjem naloga se efikasno sprečava kontinuirani pokušaj pogađanja lozinke, čime se povećava sigurnost aplikacije i zaštita korisničkih podataka. OWASP Top 10 preporučuje implementaciju ovakvih mera kako bi se smanjila površina za napad i zaštitili korisnički nalozi od automatizovanih i ručnih napada.

6. Verifikacija korisničkog naloga

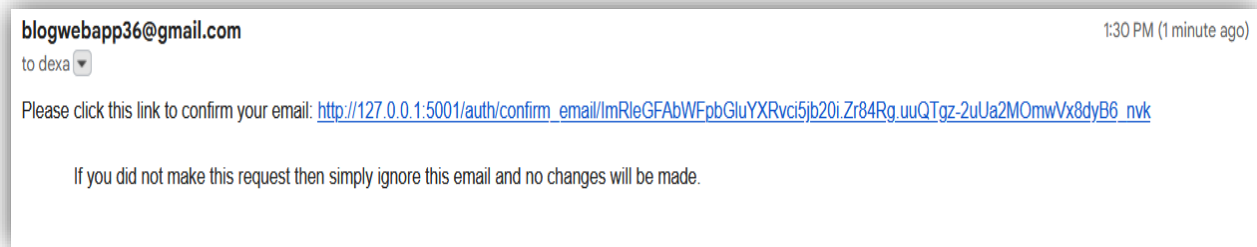
Verifikacija naloga je proces potvrđivanja identiteta korisnika kako bi se osiguralo da je osoba koja je unela podatke za registraciju stvarno ta osoba. Korišćenje verifikacije sprečava neovlašćene korisnike da koriste nepostojeće ili tuđe naloze. Takođe, onemogućava automatizovano kreiranje naloga.

Postoji više metoda za implementaciju verifikacije naloga:

- **Verifikacija putem e-pošte:** Najčešći metod verifikacije, gde korisnik dobija e-poruku sa jedinstvenim linkom ili kodom koji mora uneti ili kliknuti kako bi potvrdio svoj nalog.
- **Verifikacija putem SMS-a:** Korisnik dobija jednokratni kod (OTP) putem SMS-a koji mora uneti u aplikaciju.
- **Druge metode:** Metode kao što su telefonski poziv ili upotreba aplikacija za autentifikaciju.

6.1. Implementacija verifikacije putem e-pošte

Nakon što korisnik uspešno kreira svoj nalog, neophodno je da proveri sanduče imejl-a koji je naveo prilikom registracije. Na imejl stiže poruka sa putanjom koju korisnik treba da isprati kako bi verifikovao svoj nalog. Putanja je sastavljen od putanje do rute za verifikaciju naloga i od generisanog potpisanog tokena.



6.1.1. Imejl za potvrdu naloga

Generisanje potpisanog tokena je prikazano slici 6.1.2. Ugradnja potpisanog tokena u URL za potvrdu korisničkog naloga osigurava kasniju proveru integriteta i provere identiteta korisnika. To je suštinski korak u zaštiti naloga od neovlašćenog pristupa i u obezbeđivanju integriteta procesa registracije i verifikacije. Generisani tokeni se čuvaju u bazi podataka. Svaki token uskladišten u bazi podataka sadrži identifikator korisnika, sam token i vreme generisanja tokena (slika 6.1.4.).

Token prosleđen korisniku putem imejla šalje se na verifikaciju, gde se proverava njegova validnost. Proverava se da li takav token postoji u bazi podataka, da li je token istekao, da li je došlo do izmene tokena, kao i da li je došlo do neke nepredviđene greške. Token je vremenski ograničen na 3600 sekundi (jedan sat), čime se smanjuje rizik od njegove zloupotrebe u slučaju presretanja. Ako token uspešno prođe sve ove provere, vraća se imejl adresa korisnika kojem treba da se verifikuje nalog. Proces verifikacije se završava ažuriranjem korisničkog naloga u bazi, pri čemu se polje *is_verified* postavlja na *True*, čime se potvrđuje validnost naloga korisnika.

```
def generate_email_token(email, s):  
    return s.dumps(email, salt='email-confirm-salt')
```

6.1.2. Kod za generisanje tokena

```
def verify_email_token(token, s, expiration=3600):  
    try:  
        email = s.loads(token, salt='email-confirm-salt', max_age=expiration)  
        return email  
    except SignatureExpired:  
        raise SignatureExpired("The email token has expired.")  
    except BadSignature:  
        raise BadSignature("The email token is invalid.")  
    except Exception as e:  
        raise Exception(f"An error occurred: {str(e)}")
```

6.1.3. Kod za verifikaciju tokena

Metoda `verify_email_token` može da podigne sledeće izuzetke:

- **SignatureExpired:** Ako je token istekao (tj. prošlo je više od `max_age` vremena), baca se ovaj izuzetak sa porukom da je token istekao.
- **BadSignature:** Ako je token nevalidan (tj. ako je izmenjen ili je došlo do greške prilikom verifikacije), baca se ovaj izuzetak sa porukom da je token nevalidan.
- **Opšti izuzetak:** Ako se dogodi bilo koja druga greška prilikom obrade tokena, generiše se opšti izuzetak sa odgovarajućom porukom.

id	token	user_id	timestamp
b4bb5093-4877-459e-9c56-0a8354c0cd1a	ImRleGFabWFpbGluYXRvci5jb20i.Zr84Rg...	cbf6ae6f-34ac-4cc5-b7e4-969b42897615	2024-08-16 11:30:14.333441

6.1.4. Prikaz token za verifikaciju u bazi podataka

```
def send_confirmation_email(self, email: str):
    if not email:
        raise InvalidParameterException("email", "Invalid or missing parameter")
    try:
        user = self.user_service.get_user_by_email(email)

        token = token_utils.generate_email_token(email, self.s)
        self.token_service.add_confirm_token(token, user.id)

        confirm_url = url_for('auth.confirm_email', token=token, _external=True)
        msg = Message('Confirm Your Email', recipients=[email])
        msg.body = f'''Please click this link to confirm your email: {confirm_url}

        If you did not make this request then simply ignore this email and no changes will be made.
        ...

        self.mail.send(msg)

    except (EntityNotFoundError, DatabaseServiceError) as e:
        raise e
```

6.1.5. Kod za slanje verifikacionog imejla

6.2. Biblioteka `itsdangerous`

Za potrebe generisanja i verifikacije tokena u prethodnim primerima korišćena je biblioteka *itsdangerous*. Biblioteka *itsdangerous* koristi HMAC (Hash-based Message Authentication Code) za digitalno potpisivanje tokena, obezbeđujući da su podaci zaštićeni i da samo oni koji poseduju tajni ključ mogu da potvrde svoj

identitet. HMAC koristi isti tajni ključ za generisanje i verifikaciju tokena. Upotrebljava se za proveru integriteta i za autentifikaciju.

Primena HMAC-a:

- Prvo se ulazni podaci (npr. imejl adresa) serijalizuju u JSON format.
- Ovi podaci se zatim koriste u HMAC algoritmu sa SHA-256 heš funkcijom. Ova funkcija pretvara podatke u heš vrednost fiksne dužine (256 bitova).
- Biblioteka *itsdangerous* koristi HMAC u kombinaciji sa tajnim ključem (*app.secret_key*) kako bi generisao HMAC potpis.

Generisanje tokena:

- *itsdangerous* automatski serijalizuje podatke u JSON format.
- Enkodovanje serijalizovanih ulaznih podataka, timestampa i HMAC potpisa se obavlja takođe automatski, koristeći Base64 URL Safe enkodovanje.
- Generisani token uključuje enkodovane podatke, *timestamp* (ako je postavljen), i HMAC potpis. Sve ove komponente se kombinuju u jedan token koji se može koristiti za autentifikaciju i verifikaciju.

<i>lmR1c2VyQGV4YW1wbGUuY29tIlg.YHq5HQ.IAfpXsFZaxQdtEMF-fK2IHtjLkw</i>	
<i>lmR1c2VyQGV4YW1wbGUuY29tIlg</i>	Enkodovana imejl adresa
<i>YHq5HQ</i>	Enkodovan timestamp
<i>IAfpXsFZaxQdtEMF-fK2IHtjLkw</i>	HMAC potpis

6.2.1 Primer potpisanog tokena

Proces verifikacije tokena predstavlja obrnuti proces:

- **Dekodiranje i izdvajanje potpisa:**
 - Token se dekodira kako bi se izdvojili originalni podaci i HMAC potpis.
- **Regenerisanje potpisa:**
 - Zatim se koriste isti ulazni podaci (imejl), tajni ključ, i HMAC algoritam da se ponovo generiše potpis.
- **Poređenje potpisa:**
 - Ako su novo generisana HMAC vrednost i HMAC vrednost koja je poslata identične, smatra se da je token validan i to znači da podaci nisu izmenjeni.
 - Ako se vrednosti razlikuju, token je ili izmenjen ili falsifikovan. U tom slučaju baca se izuzetak *BadSignature*.

- **Provera vremenskog ograničenja:**
 - Ako je postavljeno vremensko ograničenje (npr. 15 minuta), provera se da li je token istekao
 - Ako je vreme isteklo, baca se izuzetak *SignatureExpired*.

7. Zaboravljena lozinka

Funkcionalnost za „Zaboravljenu lozinku“ je esencijalni deo svake aplikacije koja zahteva autentifikaciju korisnika. Ova funkcionalnost omogućava korisnicima da bezbedno i efikasno obnove pristup svojim nalogima u slučaju da zaborave svoje lozinke. Bez obzira na složenost lozinke, može se desiti da korisnici zaborave svoje pristupne podatke. Zbog toga je važno imati jasno definisan i siguran proces za resetovanje lozinki kako bi se očuvala sigurnost naloga i korisničko iskustvo.

Implementacija ove funkcionalnosti obuhvata nekoliko ključnih koraka, uključujući iniciranje zahteva za resetovanje lozinke, generisanje i slanje verifikacionih tokena, verifikaciju tokena i postavljanje nove lozinke. Svaki od ovih koraka mora biti pažljivo dizajniran i implementiran kako bi se obezbedila sigurnost i zaštita od potencijalnih napada.

U nastavku sledi primer implementacije poštujući najbolje smernice i prakse.

7.1. Implementacije resetovanja lozinke

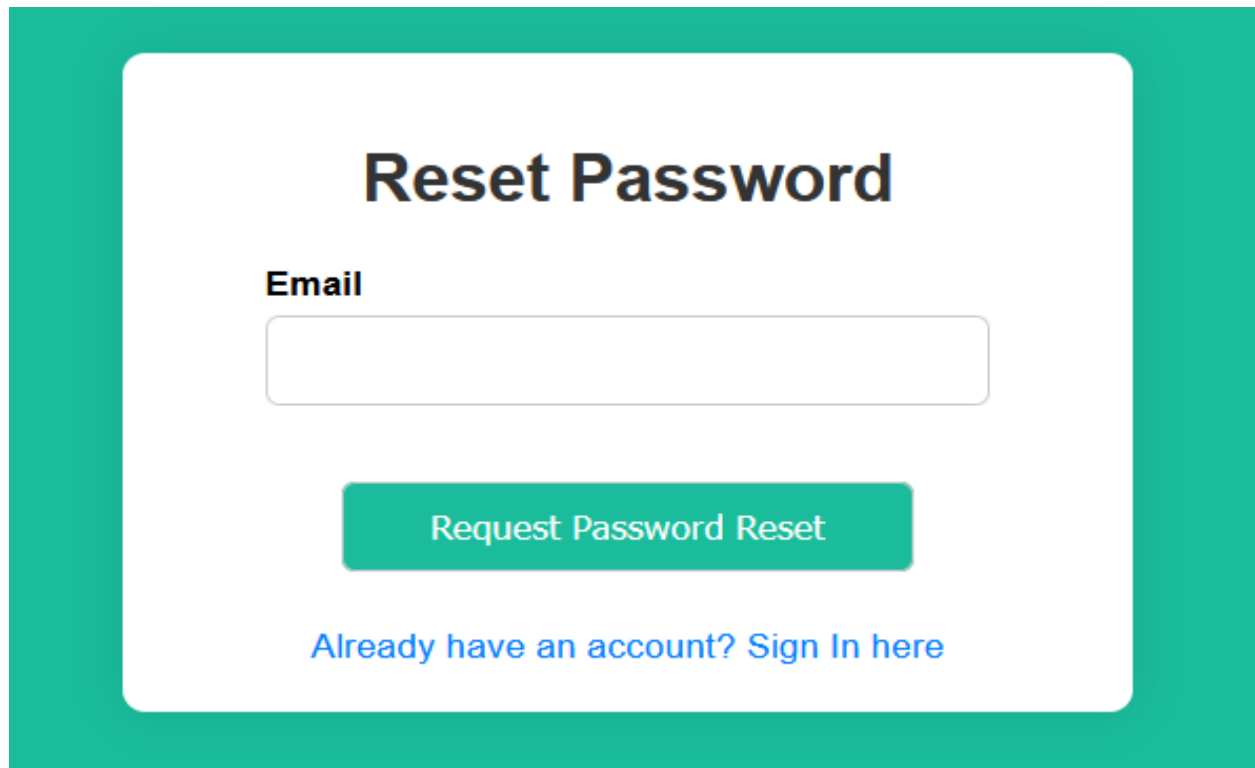
Zahtev za resetovanje lozinke se šalje putem forme za „Zaboravljenu lozinku“ (slika 7.1.1.), tako što se unose imejl adresu na koju treba da se pošalje uputstvo za resetovanje lozinke. Korisnik je preusmeren na stranicu (slika 7.1.2.) koja mu prikazuje poruku da ukoliko postoji korisnički nalog sa unetom imejl adresom, uputstvo za resetovanje lozinke je poslato. Neophodno je obratiti pažnju na oblikovanje poruke koja se korisniku ispisuje. Sledе primeri loših i dobrih praksi:

Loša praksa oblikovanja poruka:

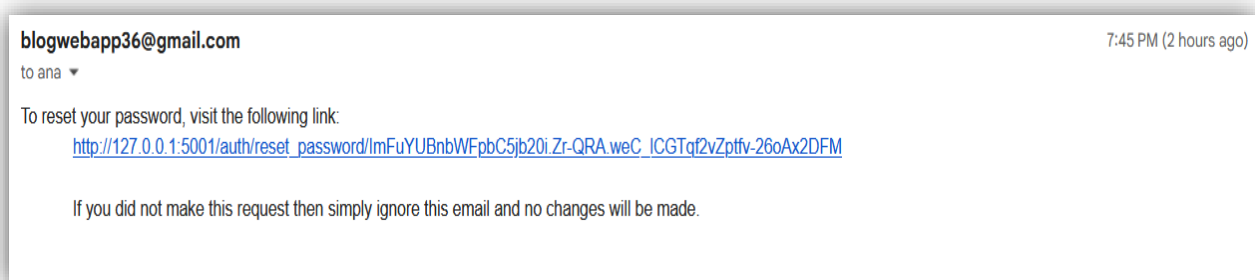
- *"An email has been sent to your email address with a link to reset your password."*
- Iz ove poruke zlonamerni napadač može zaključiti da u sistemu postoji registrovan nalog sa tom imejl adresom i može npr. pokrenuti *Credential Stuffing* napad.
- *"You will receive an email with a link containing a token for password reset. The token is generated using SHA-256 encryption."*
- Ova poruka može zbuniti korisnike jer daje previše tehničkih detalja o procesu generisanja tokena. Korisnici obično ne trebaju da znaju tehničke detalje, već samo koji su koraci do resetovanja lozinke.

Dobra praksa oblikovanja poruka:

- "If an account with that email address exists, you will receive an email with instructions to reset your password."
- Poruka ne otkriva da li korisnik sa unetim imejl adresom postoji u sistemu ili ne. Ovo sprečava curenje informacija o postojanju naloga, što može biti korisno za zaštitu od napadača koji bi mogli pokušati da otkriju validne imejl adrese.

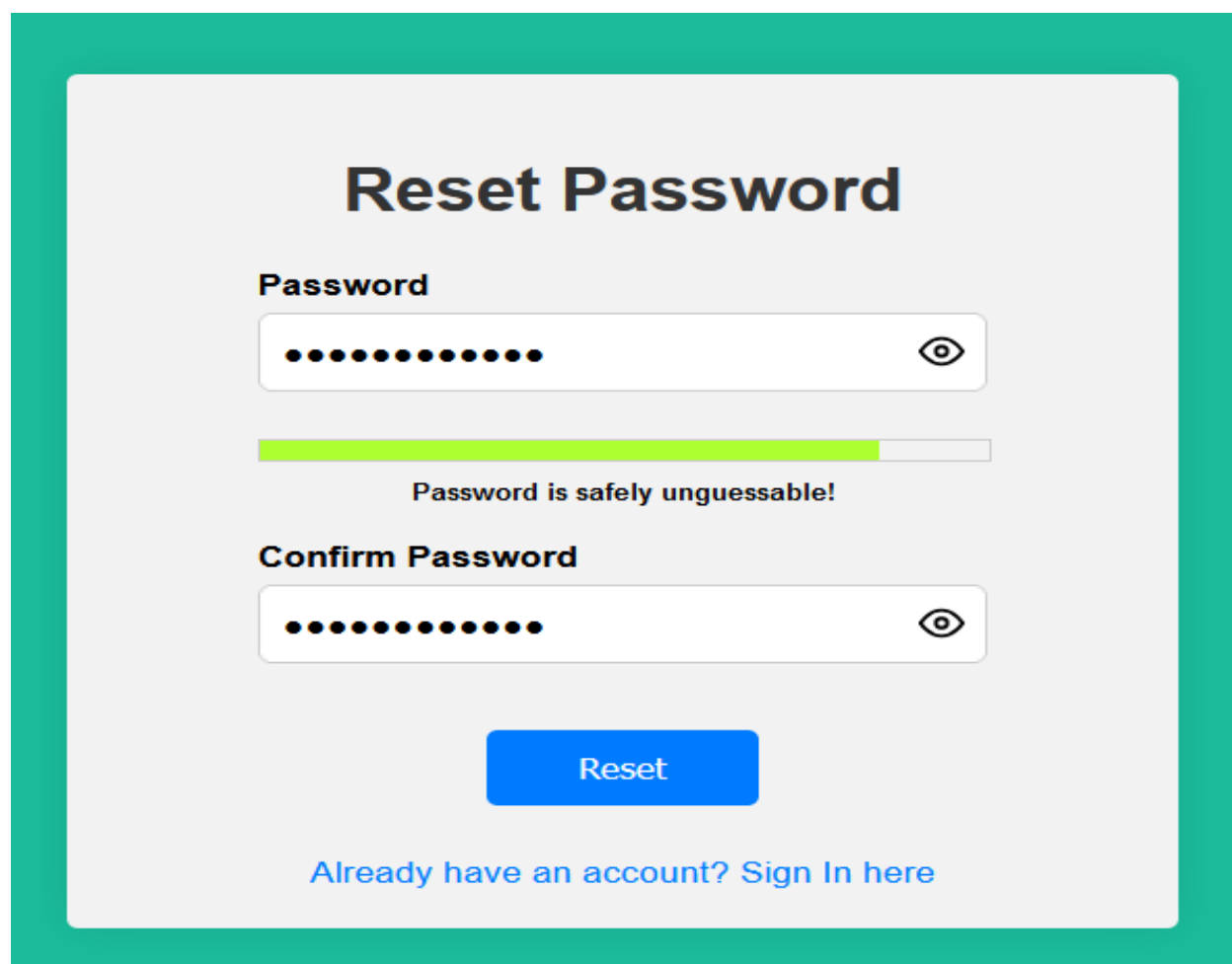
A screenshot of a web form titled "Reset Password" set against a teal background. The form itself is a white rounded rectangle. At the top, the title "Reset Password" is in a large, bold, black font. Below it, the label "Email" is in a bold black font, followed by a white rectangular input field with a thin grey border. Underneath the input field is a teal button with rounded corners and the text "Request Password Reset" in white. At the bottom of the form, there is a blue link that says "Already have an account? Sign In here".

7.1.1. Forma za resetovanje lozinke



7.1.2. Instrukcije za resetovanje lozinke

Korisniku koji isprati instrukcije poslate na mejl, preusmerava se na formu za unos nove lozinke (slika 7.1.3.).

The image shows a 'Reset Password' form with a teal border. The title 'Reset Password' is at the top. Below it is a 'Password' label and a text input field with 10 dots and an eye icon. A green progress bar is below the input field, followed by the text 'Password is safely unguessable!'. Then is a 'Confirm Password' label and another text input field with 10 dots and an eye icon. At the bottom is a blue 'Reset' button and a link 'Already have an account? Sign In here'.

7.1.3. Forma za unos nove lozinke

Nakon unosa nove lozinke i potvrde iste, šalje se zahtev sa promenom lozinke zajedno sa potpisanim tokenom kako bi se utvrdilo da li je narušen integritet i potvrdio integritet korisnika. Integritet i identitet utvrđuju se proverom na isti način kako je to ranije opisano u delu *Verifikacija korisničkih naloga*. Treba napomenuti da se potpisani tokeni za resetovanje lozinke takođe čuvaju u bazi podataka. Detaljan prikaz procesa resetovanja lozinke je prikazan na slici 7.1.4.


```

if any(value is None for value in vars(reset_password_dto).values()):
    raise InvalidParameterException("reset password dto values", "Invalid or missing parameter")

token_str = reset_password_dto.token
password = reset_password_dto.password

try:
    token, email = self.token_service.verify_reset_token(token_str)
    user = self.user_service.get_user_by_email(email)
    updated_user = self.user_service.update_password(user.id, password)
    self.token_service.set_reset_used(token)
    return updated_user
except (EntityNotFoundError, DatabaseServiceError) as e:
    raise e
except Exception as e:
    raise e

```

7.1.4. Kod za verifikaciju i resetovanje lozinke

```

@auth_bp.route('/reset_request', methods=['GET', 'POST'])
@current_app.limiter.limit("3 per minute; 10 per hour; 50 per day")
def reset_request():
    email_service = current_app.email_service
    form = RequestResetForm()

    if form.validate_on_submit():
        email = form.email.data

        try:
            email_service.send_reset_email(email)
        except EntityNotFoundError as e:
            current_app.logger.error('User not found: %s', (str(e),))
        except DatabaseServiceError as e:
            current_app.logger.error('Database: %s', (str(e),))
        except Exception as e:
            current_app.logger.error('Unhandled: %s', (str(e),))

        flash('If an account with that email address exists, you will receive an email with instructions to reset your password.', 'info')
        return redirect(url_for('main.info'))

    return render_template('reset_request.html', form=form)

```

7.1.5. Ruta koja prima zahteve za resetovanje lozinke

```

@auth_bp.route('/reset_password/<token>', methods=['GET', 'POST'])
def reset_password(token):
    auth_service = current_app.auth_service
    pwncd_service = current_app.pwncd_service
    token_service = current_app.token_service

    form = ResetPasswordForm(pwncd_service)

    try:
        token_service.verify_reset_token(token)

        if form.validate_on_submit():
            reset_password_dto = ResetPasswordDTO(password=form.password.data, token=token)

            updated_user = auth_service.reset_password(reset_password_dto)

            if updated_user:
                return redirect(url_for('auth.login'))
            else:
                return redirect(url_for('auth.reset_request'))
    except TokenException as e:
        current_app.logger.error('Reset token: %s', (str(e),))
        return redirect(url_for('auth.reset_request'))
    except EntityNotFoundError as e:
        current_app.logger.error('Reset Token not found: %s', (str(e),))
        return redirect(url_for('auth.reset_request'))
    except DatabaseServiceError as e:
        current_app.logger.error('Database: %s', (str(e),))
        return redirect(url_for('auth.reset_request'))
    except Exception as e:
        current_app.logger.error('Unhandled: %s', (str(e),))

    return render_template('reset_password.html', form=form)

```

7.1.6. Ruta za unos nove lozinke

8. CSRF (Cross-Site Request Forgery)

CSRF (*Cross-Site Request Forgery*) je ozbiljna sigurnosna ranjivost koja omogućava napadačima da izvrše neovlašćene akcije u ime korisnika bez njihovog znanja. Ova vrsta napada se oslanja na činjenicu da web pretraživači automatski šalju kolačiće (*cookies*) sa svakim HTTP zahtevom, bez obzira na to odakle je zahtev potekao. CSRF napad koristi ovu ranjivost kako bi naterao korisnika da nesvesno izvrši radnje kao što su promene lozinke, brisanje korisničkog naloga, transferi sredstava ili druge potencijalno štetne operacije.

Zbog toga je neophodno implementirati odgovarajuće mere zaštite u veb aplikacijama kako bi se sprečilo zloupotrebljavanje poverljivih podataka i neovlašćen pristup sistemu.

8.1. Scenario CSRF napada

Ne korišćenje CSRF tokena može dovesti do sledećeg scenarija:

- **Korisnik se autentifikuje:** Korisnik se prijavi na veb aplikaciju i dobije validnu sesiju ili autentifikacioni kolačić (cookie).
- **Napadač priprema zlonamernu stranicu:** Napadač kreira zlonamernu web stranicu koja sadrži skriveni obrazac ili skriptu koja šalje zahtev ka ranjivoj aplikaciji, koristeći korisnikov kolačić za autentifikaciju.
- **Korisnik posećuje zlonamernu stranicu:** Korisnik (koji je već prijavljen na ciljnu aplikaciju) posećuje zlonamernu stranicu, nesvesno pokrećući skriveni obrazac ili skriptu.
- **Nepoznati zahtev se šalje:** Zlonamerna stranica automatski šalje zahtev ka ciljnoj aplikaciji koristeći korisnikovu sesiju ili kolačić, imitirajući legitimni zahtev korisnika.
- **Aplikacija izvršava zahtev:** Ciljna aplikacija prima zahtev kao da ga je poslao legitimni korisnik i izvršava akciju bez dodatne provere. To može rezultirati neovlašćenim radnjama poput promene korisnikovih podataka, transfera novca, promene lozinke ili čak brisanja naloga.

Iz gore prikazanog primera, vidi se koliko je bitno da u kontekstu veb aplikacija sve forme za unos koje mogu promeniti stanje budu i adekvatno zaštićene.

8.2. Zaštita formi od CSRF napada

CSRF zaštita u Flask aplikacijama se obično implementira korišćenjem ekstenzije *Flask-WTF*, koja omogućava lako dodavanje CSRF zaštite. Ova ekstenzija automatski generiše i validira CSRF token za svaku formu u aplikaciji. Unutar *Flask* aplikacije, potrebno je konfigurisati tajni ključ (*app.secret_key*) koji će se koristiti za generisanje CSRF tokena, kao i objekat klase *CSRFProtect*. Poslednji korak je dodavanje koda `{{ form.hidden_tag() }}` u okviru svake forme za unos (slika 8.2.1.).

Validacija CSRF tokena se vrši na strani servera i u nedostatku ovog tokena, server odbija zahtev, čime se efektivno sprečava CSRF napad. Ključno je da se CSRF token osvežava pri svakom zahtevu, što dodatno smanjuje mogućnost uspešnog napada. Važno je napomenuti da se CSRF zaštita primenjuje na sve POST, PUT, DELETE i druge zahteve koji mogu promeniti stanje servera. Ovim pristupom osigurava se da čak i ako napadač uspe da prevari korisnika da izvrši neželjeni zahtev, server neće prihvatiti zahtev bez validnog CSRF tokena, pružajući dodatni sloj sigurnosti protiv zlonamernih napada koji koriste ranjivosti u autentifikaciji sesija.

```

<h1>Sign In</h1>
<form method="post">
  {{ form.hidden_tag() }}

  <div class="form-group">
    {{ form.email.label }}<br />
    {{ form.email(size=32) }}<br />
    {% if form.email.errors %}
    <ul class="error-message">
      {% for error in form.email.errors %}
      <li>{{ error }}</li>
      {% endfor %}
    </ul>
    {% endif %}
  </div>

```

8.2.1. Umetanje CSRF tokena

9. Eskalacija privilegija

Eskalacija privilegija je sigurnosni problem koji nastaje kada korisnik stekne viši nivo pristupa sistemu nego što mu je prvobitno dodeljen. To može uključivati dobijanje pristupa funkcijama, podacima ili resursima koje korisnik inače ne bi smeo da koristi. Eskalacija privilegija može biti:

- Horizontalna, gde korisnik pristupa resursima drugih korisnika sa istim nivoom privilegija.
- Vertikalna, gde korisnik pristupa resursima ili funkcijama namenjenim korisnicima sa višim privilegijama, poput administratorskih prava.

Ova vrsta napada može ozbiljno ugroziti sigurnost sistema, što čini adekvatnu zaštitu ključnom za sigurnost aplikacije.

9.1. Horizontalna eskalacija privilegija

Horizontalna eskalacija privilegija nastaje kada korisnik uspe da pristupi resursima ili podacima drugog korisnika koji ima isti nivo privilegija, što predstavlja zloupotrebu sistema. Na primer, ako korisnik A može

da izmeni profil korisnika B, a oba korisnika imaju isti nivo pristupa, to je primer horizontalne eskalacije privilegija. Ovaj napad iskorišćava nedostatke u horizontalnoj zaštiti ruta.

Na primer imamo putanju `http://127.0.0.1:5001/user/profile/eeb2bee5-1423-4169-af69-febc22b53ec2`, rute koje nisu zaštićene od horizontalne eskalacije privilegija dozvoljavaju izmenom poslednjeg parametra (u ovom slučaju je to identifikator korisnika), pristup informacijama za korisnika čiji smo identifikator naveli. Ovo predstavlja ozbiljan sigurnosni propust.

Horizontalna zaštita ruta je mehanizam dizajniran da spreči horizontalnu eskalaciju privilegija. Njena uloga je da osigura da korisnici mogu pristupiti samo svojim resursima i podacima, bez mogućnosti da pristupe resursima drugih korisnika sa istim nivoom privilegija. To se postiže proverom da li je korisnik koji pristupa resursu i njegov vlasnik i korišćenjem jedinstvenih identifikatora (*UUID*), kako bi se napadaču otežalo pogađanje identifikatora.

```
@user_bp.route('/request_author_role/<string:user_id>', methods=['GET'])
@login_required
@requires_roles('Reader')
def request_author_role(user_id):

    if user_id != current_user.id:
        return redirect(url_for('main.index'))

    author_requests_service = current_app.author_requests_service
    try:
        if author_requests_service.check_existence(user_id):
            author_requests_service.create_author_request(user_id)
            flash('You successfully send a request for an author role!', 'info')
        else:
            flash('Your request is still in progress.', 'info')
    except InvalidParameterException as e:
        current_app.logger.error('Parameter: %s', (str(e),))
    except DatabaseServiceError as e:
        current_app.logger.error('Database: %s', (str(e),))
    except Exception as e:
        current_app.logger.error('Unhandled: %s', (str(e),))

    return redirect(url_for('main.index'))
```

9.1.1. Primer zaštite rute