

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.....	5
1.1. Šta je autentifikacija?	5
1.2. Flask-Login	5
1.3. User model klasa	6
1.4. Upravljanje sesijama	6
1.5. Dvofaktorska autentifikacija (2FA).....	10
1.6. Implementacija 2FA.....	10
2. Sigurno skladištenje podataka.....	13
2.1. Šta je hešovanje?	13
2.2. Hešovanje lozinki	13
2.3. Sigurno skladištenje identifikatora sa UUID-om	14
3. Autorizacija.....	15
3.1. Šta je autorizacija?	15
3.2. Implementacija RBAC	16
4. Bezbednost Lozinki: Metode i alati za proveru.....	18
4.1. Procena jačine lozinke (Strength meter i zxcvbn)	18
4.2. Provera kompromitovanosti lozinke (Have I Been Pwned).....	21
5. Ograničavanje broja zahteva (Rate Limiting).....	21
5.1. Flask-Limiter.....	22
5.2. Zaključavanje korisničkog naloga	25
6. Verifikacija korisničkog naloga.....	27
6.1. Implementacija verifikacije putem e-pošte	28
6.2. Biblioteka itsdangerous	29
7. Zaboravljena lozinka	31
7.1. Implementacije resetovanja lozinke.....	31
8. CSRF (Cross-Site Request Forgery).....	35
8.1. Scenario CSRF napada	36
8.2. Zaštita formi od CSRF napada	36
9. Eskalacija privilegija	37
9.1. Horizontalna eskalacija privilegija	37
10. HTTPS.....	39

10.1.	Podešavanje HTTPS-a u Flask aplikaciji.....	39
10.2.	HTTPS i SSL/TLS: Kako obezbeđuju sigurnost na web-u	40
11.	ReCAPTCHA	41
11.1.	ReCAPTCHA v2: Integracija u Flask Veb Aplikaciji.....	42
11.2.	ReCAPTCHA v3: Integracija u Flask Veb Aplikaciji.....	44
12.	Čuvanje osetljivih podataka u varijablama okruženja.....	46
12.1	Konfiguracija u Flask veb aplikaciji	46
13.	Sanatizacija i validacija unosa.....	47
13.1.	Content Security Policy (CSP)	48
13.2.	WTForms: Rukovanje unosom	49
14.	Zaštita sesija	51
14.1.	Flask sesija	51
15.	HTTP zaglavlja.....	53
15.1.	Konfiguracija bezbednosnih zaglavlja koristeći Talisman	53
16.	Rukovanje izuzecima i logovanje	55
16.1.	Globalno rukovanje izuzecima	55
16.2.	Logovanje	57

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. Below the title, there are two input fields: "Email" and "Password". The "Email" field is a simple white rectangle with a thin border. The "Password" field is similar but includes a small eye icon on the right side, indicating a toggle for password visibility. Below these fields is a teal "Login" button with white text. 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	.eljFz81qFUEQBeB3ma32pbq6uqtqdoG4SPBCxJjoZuifanMTTGSmg5GQd8-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

```
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

lozinku, svi podaci uskladišteni u kolačićima (uključujući OTP token, identifikator korisnika i vreme generisanja) se brišu, a korisnik se uspešno prijavljuje na svoj nalog.

```

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\$TamnuJQt7/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)

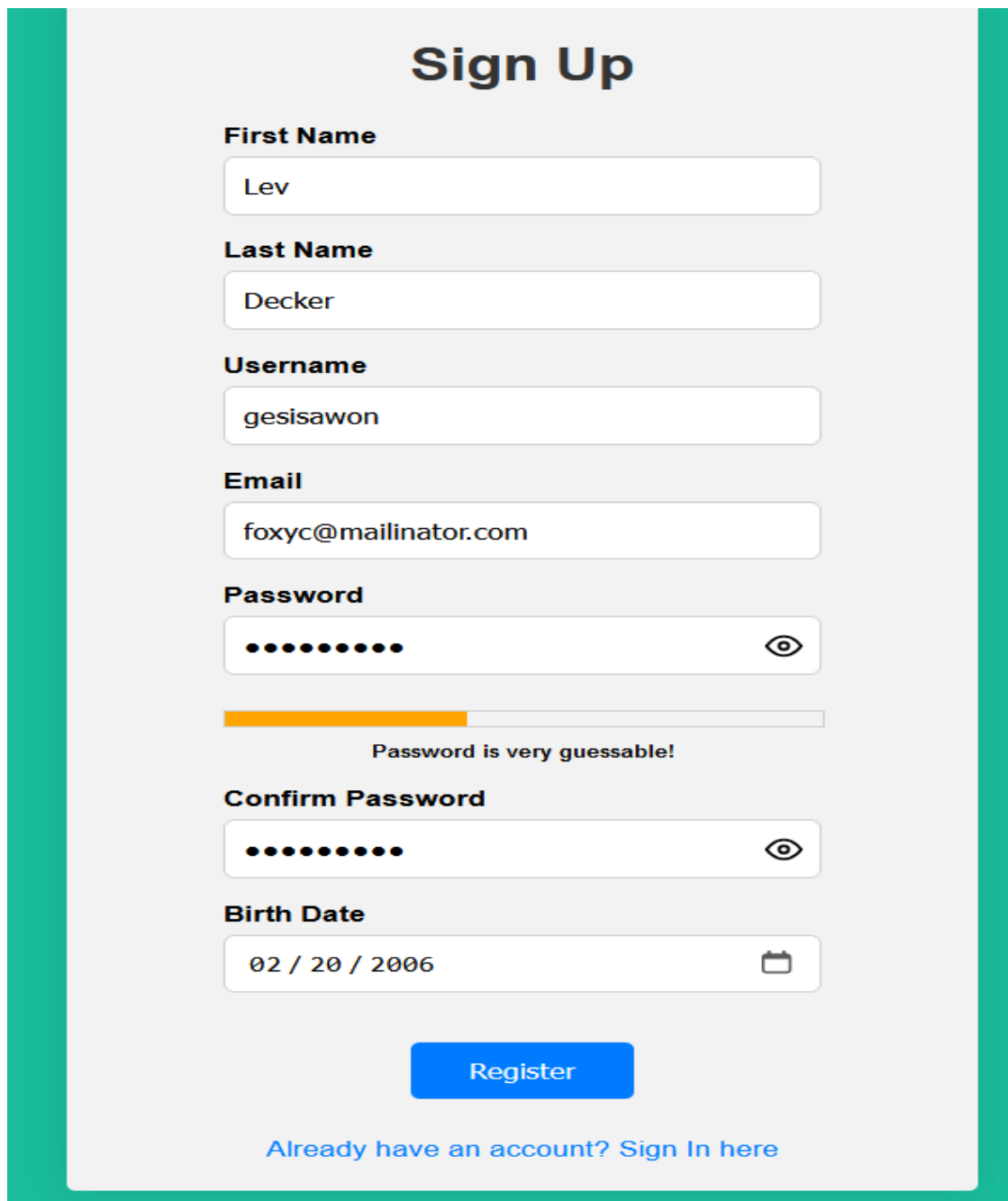
Jačina lozinke je ključni faktor u zaštiti korisničkih naloga od neovlašćenog pristupa. U današnjem digitalnom svetu, gde su sajber napadi sve učestaliji, značaj korišćenja snažnih i teško pogađanih lozinki nikada nije bio veći. Snažna lozinka ne samo da poboljšava sigurnost naloga, već i doprinosi ukupnoj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

Confirm Password

Birth Date

[Register](#)

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

Strength indicator: Password is very guessable!

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

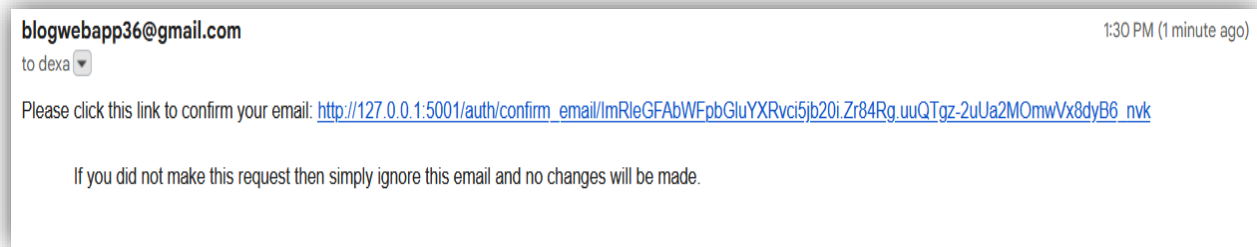
Verifikacija naloga je proces potvrđivanja identiteta korisnika kako bi se osiguralo da je osoba koja je unela podatke za registraciju stvarno ta osoba. Korišćenje verifikacije sprečava neovlašćene korisnike da koriste nepostojeće ili tuđe naloze. Takođe, onemogućava automatizovano kreiranje naloga.

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	lmRleGFabWFpbGluYXRvci5jb20i.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

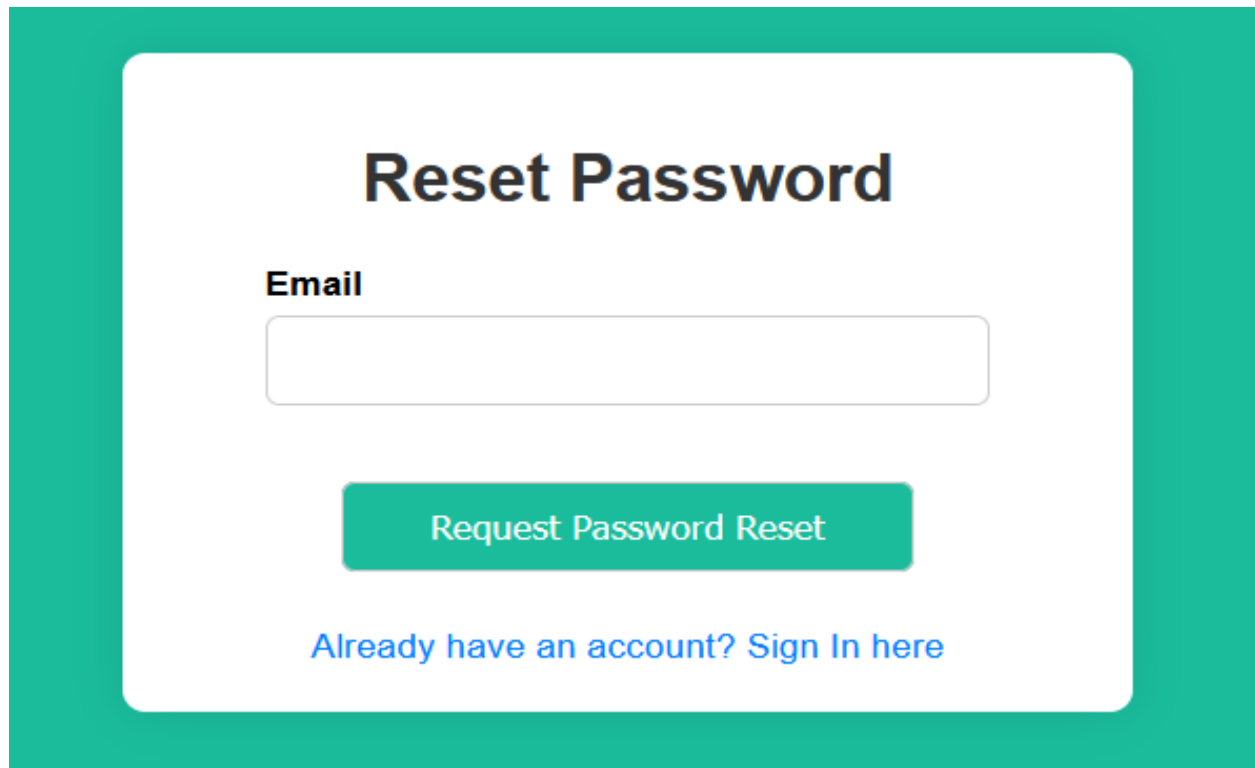
Zahtev za resetovanje lozinke se šalje putem forme za „Zaboravljenu lozinku“ (slika 7.1.1.), tako što se unose imejl adresu na koju treba da se pošalje uputstvo za resetovanje lozinke. Korisnik je preusmeren na stranicu (slika 7.1.2.) koja mu prikazuje poruku da ukoliko postoji korisnički nalog sa unetom 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 is a white rounded rectangle containing the title, an "Email" label above a text input field, a teal "Request Password Reset" button, and a blue link "Already have an account? Sign In here".

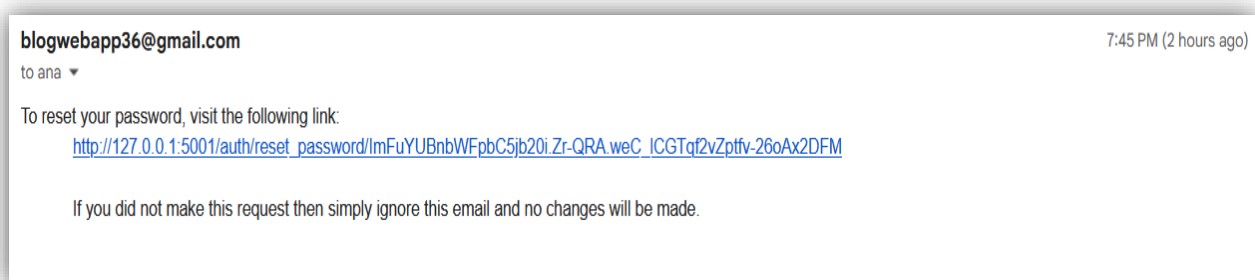
Reset Password

Email

Request Password Reset

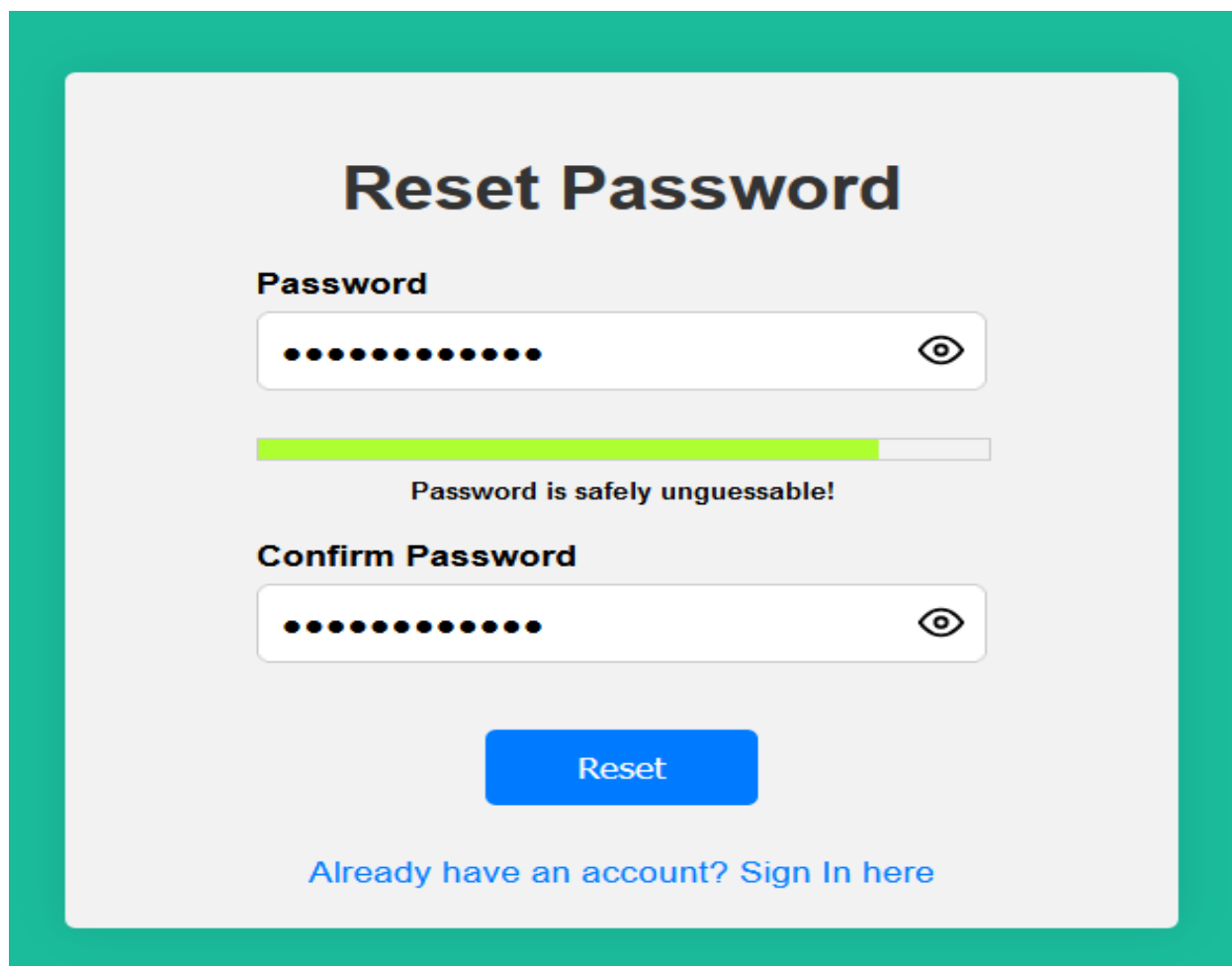
[Already have an account? Sign In here](#)

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 centered at the top. Below it is a 'Password' label followed by a text input field containing ten black dots. To the right of the input field is an eye icon. Below the input field is a green progress bar that is approximately 75% full. Under the progress bar, the text 'Password is safely unguessable!' is displayed. Below this is a 'Confirm Password' label followed by another text input field with ten black dots and an eye icon to its right. At the bottom of the form is a blue button labeled 'Reset'. Below the button is a link that says '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

10. HTTPS

HTTPS (*Hypertext Transfer Protocol Secure*) je proširenje HTTP protokola koje dodaje sloj sigurnosti enkripcijom podataka razmenjenih između klijenta (kao što je web pregledač) i servera. Ova enkripcija osigurava da se osetljive informacije, poput lozinki ili ličnih podatak, prenose na siguran način, štiteći ih od prisluškivanja, manipulacije i napada presretanjem komunikacije. Implementacija HTTPS-a u web aplikacijama je ključna za održavanje privatnosti i integriteta korisničkih podataka i smatra se standardnom praksom u zaštiti savremenih web aplikacija.

10.1. Podešavanje HTTPS-a u Flask aplikaciji

Implementacija HTTPS-a u Flask aplikaciji omogućava sigurno šifrovanje podataka između klijenta i servera, čime se obezbeđuje zaštita poverljivih informacija kao što su korisnički podaci, lozinke, i drugi osetljivi podaci koji se prenose putem interneta. HTTPS koristi SSL/TLS protokol za uspostavljanje sigurne veze, čime se sprečava presretanje i manipulacija podacima.

Da bi omogućili HTTPS u Flask aplikaciji u fazi razvoja i testiranja, prvo je potrebno generisati *self-signed* sertifikate. Ovo se može uraditi korišćenjem OpenSSL alata, unošenjem sledeće komande u CMD:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -nodes
```

- **-x509** - Sertifikat u formatu X.509 je standardni format za sertifikate koji se koristi u SSL/TLS protokolima.
- **newkey rsa:4096** - Kreira novi RSA privatni ključ sa dužinom od 4096 bita.
- **keyout key.pem** - Određuje naziv fajla za privatni ključ.
- **out cert.pem** - Određuje naziv fajla za sertifikat.
- **days 365** - Postavlja period validnosti sertifikata na 365 dana.
- **-nodes** - Privatni ključ neće biti šifrovan sa lozinkom

Generisani privatni ključ je tajni deo koji ostaje na serveru i koristi se za dešifrovanje podataka koje je šifrovao odgovarajući javni ključ. Javni ključ je deo samog sertifikata i služi sa šifrovanje podataka koji mogu biti dešifrovani jedino putem privatnog ključa.

```
if __name__ == '__main__':  
    app.run(ssl_context=('cert.pem', 'key.pem'), port=9001)
```

10.1. Podešavanje flask aplikacije

10.2. HTTPS i SSL/TLS: Kako obezbeđuju sigurnost na web-u

HTTPS koristi SSL/TLS protokole kako bi osigurao da komunikacija između klijenta i servera bude zaštićena od presretanja i manipulacije. Proces uključuje razmenu kriptografskih ključeva, verifikaciju identiteta, šifrovanje i dešifrovanje podataka, čime se obezbeđuje da su svi podaci zaštićeni tokom prenosa. Ovaj sloj sigurnosti je ključan za zaštitu poverljivih informacija i očuvanje privatnosti korisnika na internetu.

Inicijalna veza

Kada korisnik pokuša da pristupi web stranici koristeći HTTPS, web pregledač i server započinju proces poznat kao "SSL/TLS handshake" (*handshake* proces). Tokom ovog procesa:

- **Veb pregledač šalje zahtev:** Pregledač inicira vezu tako što šalje zahtev serveru za uspostavljanje sigurne veze. Ovaj zahtev uključuje informacije o verzijama TLS/SSL protokola i kriptografskim algoritmima koje pregledač podržava.
- **Server odgovara:** Server odgovara tako što šalje svoj SSL/TLS sertifikat pregledaču. Sertifikat uključuje javni ključ servera, podatke o sertifikacionoj vlasti (CA) i informacije o identitetu servera (domen, organizacija, i tako dalje).

Provera identiteta

Pregledač proverava sertifikat koji je poslat od servera kako bi osigurao da komunicira sa pravim serverom:

Verifikacija sertifikata:

- **Autentičnost sertifikata:** Pregledač proverava da li je sertifikat izdat od pouzdane sertifikacione vlasti (CA) koja je u njegovom skladištu sertifikata. Ako sertifikat nije izdat od pouzdane CA, pregledač će prikazati upozorenje korisniku (ako je sertifikat *self-signed*).
- **Validnost sertifikata:** Proverava se da li sertifikat još uvek važi i nije istekao.
- **Podudaranje sa domenom:** Proverava se da li se domen koji se pristupa poklapa sa domenom navedenim u sertifikatu.

Ako je sertifikat validan pregledač prelazi na sledeći korak u procesu.

Razmena ključeva i šifrovanje

Nakon što je sertifikat validiran, pregledač i server se usaglašavaju o kriptografskim ključevima koje će koristiti za enkripciju podataka.

Razmena ključeva:

- **Pregledač generiše sesijski ključ:** Pregledač koristi javni ključ servera za šifrovanje sesijskog ključa koji će se koristiti za sve buduće komunikacije. Ovaj sesijski ključ omogućava brzu i efikasnu enkripciju podataka (simetrična kriptografija).

Uspostavljanje sigurne sesije:

- **Pregledač i server koriste sesijski ključ** za šifrovanje i dešifrovanje svih budućih komunikacija tokom sesije. Sesijski ključ je tajni ključ koji se koristi za šifrovanje podataka tokom trajanja veze.

Šifrovanje podataka

Svi podaci koji se šalju između veb pregledača i servera su šifrovani kako bi se osigurala privatnost i integritet komunikacije:

1. Šifrovanje:

- **Podaci se šifruju** pomoću sesijskog ključa, što znači da su zaštićeni od neovlašćenog pristupa tokom prenosa. To uključuje sve informacije kao što su korisnički zahtevi, podaci o obrascima i druge poverljive informacije.

2. Dešifrovanje:

- **Pregledač i server koriste isti sesijski ključ** za dešifrovanje podataka. Kada server primi šifrovane podatke od pregledača, koristi sesijski ključ za njihovo dešifrovanje i obrnuto.

Kraj sesije

Kada se sesija završi, HTTPS veza može biti zatvorena. Kada korisnik ponovo pristupi stranici, nova veza se uspostavlja koristeći isti proces verifikacije i enkripcije.

11. ReCAPTCHA

ReCAPTCHA je tehnologija razvijena od strane Google-a, koja omogućava veb aplikacijama da prave razliku između stvarnih korisnika i automatizovanih botova. U današnje vreme, online servisi su sve više izloženi različitim oblicima zloupotreba, kao što su spam registracije, brute-force napadi, i slične aktivnosti koje mogu ugroziti sigurnost i funkcionalnost sistema. ReCAPTCHA pruža efikasno rešenje za ove probleme, nudeći sloj zaštite koji onemogućava botovima da automatski popune i pošalju formulare.

Integracija reCAPTCHA u veb aplikaciju ne samo da povećava sigurnost, već i doprinosi boljem korisničkom iskustvu, jer je dizajnirana da bude što manje invazivna za stvarne korisnike. Na primer, reCAPTCHA v2 koristi vizualne i tekstualne izazove koje su ljudi lako sposobni da reše, dok su botovi u tome neuspešni. reCAPTCHA v3 ide korak dalje, analizirajući ponašanje korisnika na stranici kako bi procenila da li je korisnik stvarna osoba ili ne, bez potrebe za dodatnim interakcijama.

11.1. ReCAPTCHA v2: Integracija u Flask Veb Aplikaciji

ReCAPTCHA v2 je popularna verzija Google-ove reCAPTCHA tehnologije, koja pruža dodatni sloj sigurnosti za veb aplikacije. Funkcioniše tako što prikazuje izazov korisniku, koji može biti jednostavan zadatak kao što je selektovanje određenih slika ili jednostavno potvrda da korisnik nije robot putem checkboxa ("I'm not a robot").

Načini Implementacije reCAPTCHA v2

ReCAPTCHA v2 nudi nekoliko načina implementacije:

- **Checkbox reCAPTCHA:** Najpoznatiji oblik reCAPTCHA v2 je "I'm not a robot" checkbox. Korisnik mora da klikne na checkbox, a reCAPTCHA analizira ponašanje korisnika da bi procenila da li je stvarna osoba. Ako se proceni da je potrebna dodatna verifikacija, korisniku se prikazuje vizuelni zadatak (slike) koji mora da reši.
- **Invisible reCAPTCHA:** Ova varijanta se aktivira bez potrebe da korisnik klikne na checkbox. Aktivira se u pozadini, što omogućava neprekidno korisničko iskustvo, dok i dalje štiti vašu aplikaciju od botova. Invisible reCAPTCHA može biti povezana sa dugmetom za submit, i aktivira se kada korisnik pokuša da pošalje formu.
- **reCAPTCHA sa slikama:** Ovaj tip reCAPTCHA zahteva od korisnika da selektuje određene slike iz grupe, kao što su svi automobili, saobraćajni znaci, itd. Ova metoda je efikasna u razlikovanju ljudi od botova, jer je botovima znatno teže da uspešno reše takve zadatke.

Registracija na reCAPTCHA servisu

Da biste koristili reCAPTCHA, prvo morate registrovati svoju aplikaciju na reCAPTCHA servisu putem Google reCAPTCHA sajta. Tokom registracije, dobićete *site key* i *secret key*, koji su potrebni za integraciju. Oba ključa treba dodati u konfiguraciju aplikacije (slika 11.1.1).

```
RECAPTCHA_PUBLIC_KEY = os.getenv('SITE_KEY_RECAPTCHA') SITE_KEY_RECAPTCHA = '6Ld7oCoqAAAAExiIQ8dkbwA6AcLH72DS9FMMwmK'  
RECAPTCHA_PRIVATE_KEY = os.getenv('SECRET_KEY_RECAPTCHA') SECRET_KEY_RECAPTCHA = '6Ld7oCoqAAAAACDBwmqB4bacQqAVy4JPJ35LQ5py'
```

11.1.1. Konfiguracija ključeva

Dodavanje reCAPTCHA u Flask-WTF formu

Sledeći korak je kreiranje forme koja uključuje reCAPTCHA polje (slika 11.1.2.). Koristićemo *RecaptchaField* koji automatski generiše reCAPTCHA vidžet u formi.

```

from flask_wtf import FlaskForm, RecaptchaField
from wtforms import StringField, PasswordField, SubmitField
from wtforms.validators import DataRequired, Email, Length

class LoginForm(FlaskForm):
    recaptcha = RecaptchaField()
    email = StringField('Email', validators=[DataRequired(), Email(), Length(max=32)])
    password = PasswordField('Password', validators=[DataRequired()])
    submit = SubmitField('Login')

```

11.1.2. Kod forme za prijavu

```

<div>
    {{ form.recaptcha }} {% if form.recaptcha.errors %}
    <span style="color: red">{{ form.recaptcha.errors[0] }}</span>
    {% endif %}
</div>

```

11.1.3. Dodavanje reCAPTCHA vidžeta

11.1.4. Korisnička forma za prijavu

11.2. reCAPTCHA v3: Integracija u Flask Veb Aplikaciji

reCAPTCHA v3 je najnovija verzija Google-ove reCAPTCHA tehnologije, koja pomaže u zaštiti web aplikacija od zlonamernih botova i automatizovanih napada, a da pritom ne ometa korisničko iskustvo. Za razliku od prethodnih verzija, gde je korisnik morao da rešava vizuelne izazove, nova verzija funkcioniše u pozadini i analizira korisničke interakcije kako bi procenila da li je u pitanju stvarni korisnik ili bot. Na osnovu te analize, reCAPTCHA v3 generiše rezultat (*score*) koji određuje nivo sumnje u vezi sa datim korisničkim ponašanjem. Ova tehnologija omogućava web aplikacijama da prilagode način odgovora na različite vrste zahteva, pružajući dodatnu zaštitu bez ugrožavanja iskustva korisnika.

Integracija sa HTML šablonom

U `<head>` tag HTML šablona za registraciju korisnika, dodaje se reCAPTCHA v3 skripta (slika 11.2.1.), koja će učitati reCAPTCHA biblioteku i omogućiti korišćenje funkcionalnosti reCAPTCHA v3. Unutar forme dodaje se skriveno polje koje će sadržati učitani token (slika 11.2.2.). Ovo polje se automatski popunjava sa generisanim tokenom. Još je neophodno dodati skriptu za generisanje i dodavanje tokena unutar skrivenog polja (slika 11.2.3.). Kod koristi *grecaptcha.ready* da se uveri da je reCAPTCHA skripta učitana, a zatim *grecaptcha.execute* za generisanje tokena na osnovu rezultata korisničkih akcija.

```
<script src="https://www.google.com/recaptcha/api.js?render=6LdxnioqAAAAFb8J7sg27ck3ZZv_JT3k0GnqIiN"></script>
```

11.2.1. ReCAPTCHA skripte

```
<input
  type="hidden"
  name="g-recaptcha-response"
  id="g-recaptcha-response"
/>
```

11.2.2. Skriveno polje za token

```
<script>
grecaptcha.ready(function () {
  grecaptcha
    .execute('6LdxnioqAAAAFb8J7sg27ck3ZZv_JT3k0GnqIiN', {
      action: 'submit',
    })
    .then(function (token) {
      document.getElementById('g-recaptcha-response').value = token;
    });
});
</script>
```

11.2.3. Kod za generisanje tokena

Verifikacija reCAPTCHA tokena na strani servera

Token koji se pošalje prilikom klika na submit dugme, na strani servera treba da se verifikuje. Unutar rute za registraciju dobavlja se tajni ključ koji je dobijen prilikom registracije na zvaničnom sajtu Google reCAPTCHA i zajedno sa tokenom prosleđuje servisu zaduženom za verifikaciju (slika 11.2.4.). Proces verifikacije obuhvata slanje zahteva ka Google API-ju kako bi se verifikovao dobijeni token (slika 11.2.5.). Na osnovu dobijenog odgovora, donosi se zaključak da li je zahtev za registraciju poslala automatizovana skripta ili stvarni korisnik.

```

if form.validate_on_submit():

    recaptcha_token = request.form.get('g-recaptcha-response')
    recaptcha_secret = current_app.config['RECAPTCHA_V3_PRIVATE_KEY']

    if not recaptcha_service.verify_token(recaptcha_token, recaptcha_secret):
        render_template('register.html', form=form)

```

11.2.4. Deo koda rute za registraciju

```

import requests

class RecaptchaService():
    def __init__(self):
        self.recaptcha_url = 'https://www.google.com/recaptcha/api/siteverify'

    def verify_token(self, recaptcha_token: str, recaptcha_secret: str) -> bool:
        response = requests.post(self.recaptcha_url, data={
            'secret': recaptcha_secret,
            'response': recaptcha_token
        })

        result = response.json()
        return False if not result.get('success') else True

```

11.2.5. Servis za verifikaciju reCAPTCHA tokena

```

INFO:werkzeug:127.0.0.1 - - [20/Aug/2024 22:18:22] "GET /statics/img/eye-password-hide.svg HTTP/1.1" 304 -
{'success': True, 'challenge_ts': '2024-08-20T20:18:22Z', 'hostname': '127.0.0.1', 'score': 0.9, 'action': 'submit'}
INFO:werkzeug:127.0.0.1 - - [20/Aug/2024 22:18:36] "POST /auth/register HTTP/1.1" 302 -

```

11.2.6. Rezultat provere reCAPTCHA tokena

Integracija reCAPTCHA v3 sa formom za registraciju korisničkog naloga je relevantna u zaštiti od lažnih registracija putem automatizovanih skripti, čiji je cilj preopterećenje veb aplikacije. Za razliku od prethodne verzije, reCAPTCHA v3 ne utiče na korisničko iskustvo jer ne zahteva interakciju korisnika. Takođe, može se primeniti i u procesu prijavljivanja korisnika, gde bi se na osnovu postignutog rezultata (*score*) donosile odluke. Na primer, korisnik koji ostvari nizak rezultat (što sugeriše da možda nije čovek) može biti preusmeren na 2FA autentifikaciju.

12. Čuvanje osjetljivih podataka u varijablama okruženja

Korišćene osjetljivih podataka direktno u kodu smatra se ozbiljnim sigurnosnim propustom. Pod osjetljivim podacima misli se na tajne ključeve, ključeva baza podataka, API ključeve i tako dalje. Njihova zaštita se može izvršiti njihovim čuvanjem u varijablama okruženja. Za potrebe skladištenja neophodno je definisati `.env` fajl i uključiti `dotenv` biblioteku.

12.1 Konfiguracija u Flask veb aplikaciji

Prvi korak u zaštiti osjetljivih podataka je definisanje `.env` fajla (slika 12.1.2.). Ovaj fajl sadrži varijable okruženja koje čuvaju poverljive informacije kao što su URL baze podataka, tajni ključevi, API ključevi i druge osjetljive podatke potrebne za rad aplikacije.

Kada je `.env` fajl pravilno postavljen, on se učitava prilikom pokretanja veb aplikacije, koristeći biblioteku kao što je `dotenv`. Na ovaj način, varijable okruženja iz `.env` fajla postaju dostupne aplikaciji preko `os.getenv` funkcije, što omogućava siguran pristup ovim podacima bez njihovog direktnog hardkodovanja u izvorni kod. Takođe, dobra praksa je definisanje `Config` klase koja sadrži sve učitane varijable okruženja kao konfiguracione parametre aplikacije (slika 12.1.2.).

Bitno je da `.env` fajl nikada ne bude uključen u verzionisanje koda, jer to može dovesti do curenja osjetljivih podataka ukoliko fajl postane dostupan na javnim repozitorijumima ili neovlašćenim osobama. Kako bi se ovo sprečilo, potrebno je dodati `.env` fajl u `.gitignore` datoteku, što osigurava da fajl ne bude verzionisan i deljen kroz sisteme za kontrolu verzija kao što je Git.

```
FLASK_SECRET_KEY=  
DEBUG=True  
MAIL_DEBUG=0  
UNSECURE_APP_PORT=5001  
MAIL_SERVER = 'smtp.gmail.com'  
MAIL_PORT = 587  
MAIL_USE_TLS = True  
MAIL_USERNAME = 'blogwebapp36@gmail.com'  
MAIL_PASSWORD =   
MAIL_DEFAULT_SENDER = ('blogwebapp36@gmail.com')  
REDIS_URL = 'redis://localhost:6379/0'  
SITE_KEY_RECAPTCHA = '6Ld7oCoqAAAAExilQ8dkbwA6AcLH72DS9FMMwmK'  
SECRET_KEY_RECAPTCHA =   
SECRET_KEY_RECAPTCHA_V3 = 
```

12.1.1. Primer `.env` fajla

```
class Config:
    SECRET_KEY = os.getenv('FLASK_SECRET_KEY')
    BASE_DIR = os.path.abspath(os.path.dirname(__file__))
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'sqlite:/// ' + os.path.join(BASE_DIR, 'instance/secure-app.sqlite')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
    REDIS_URL = os.getenv('REDIS_URL')
    DEBUG = os.getenv('DEBUG')
    MAIL_DEBUG = os.getenv('MAIL_DEBUG')
    MAIL_SERVER = os.getenv('MAIL_SERVER')
    MAIL_PORT = os.getenv('MAIL_PORT')
    MAIL_USE_TLS = os.getenv('MAIL_USE_TLS')
    MAIL_USERNAME = os.getenv('MAIL_USERNAME')
    MAIL_PASSWORD = os.getenv('MAIL_PASSWORD')
    MAIL_DEFAULT_SENDER = os.getenv('MAIL_DEFAULT_SENDER')
    RECAPTCHA_PUBLIC_KEY = os.getenv('SITE_KEY_RECAPTCHA')
    RECAPTCHA_PRIVATE_KEY = os.getenv('SECRET_KEY_RECAPTCHA')
    RECAPTCHA_V3_PRIVATE_KEY = os.getenv('SECRET_KEY_RECAPTCHA_V3')
```

12.1.2. Primer Config klase

```
app.config.from_object('config.Config')
```

12.1.3. Učitavanje Config klase

13. Sanatizacija i validacija unosa

Sanatizacija i validacija unosa predstavljaju bitne procese u zaštiti veb aplikacija od napada tipa SQL Injection, XSS, pogrešan format podataka i drugih sličnih napada koji mogu ugroziti sigurnost korisničkih podataka.

Validacija ulaza odnosi se na proces provere podataka koje korisnici unose kako bi se osiguralo da su u skladu sa očekivanim formatom i pravilima. Validacija se obično vrši na nekoliko nivoa:

- **Na Klijentskoj Strani:** U ovoj fazi, validacija se vrši pre nego što se podaci pošalju serveru. Klijentska validacija često koristi JavaScript da proveri da li su podaci u skladu sa pravilima, kao što su obavezna polja ili format imejl adrese. Validacija na strani klijenta nije dovoljna jer se može zaobići.
- **Na Serverskoj Strani:** Ova vrsta validacije je presudna za bezbednost, jer se podaci proveravaju na serveru pre nego što se obrade ili sačuvaju. Ovo osigurava da samo validni podaci dođu do baze podataka ili drugih delova aplikacije.

Sanatizacija ulaza odnosi se na proces čišćenja podataka kako bi se uklonili ili kodirali potencijalno opasni elementi (kao što su <, >, &, ", itd.). Sanatizacija pomaže u sprečavanju napada kao što su SQL Injection i XSS (*Cross-Site Scripting*).

U kontekstu razvoja *Flask* veb aplikacije, i dalje je važno obratiti pažnju na zaštitu od SQL Injection i XSS napada, iako neki mehanizmi već pružaju osnovnu zaštitu:

1. **SQL Injection:** SQLAlchemy, popularni ORM (*Object Relational Mapper*) koji se koristi u *Flask* aplikacijama, pruža zaštitu od SQL Injection napada kroz upotrebu parametarskih upita. Ovo znači da se korisnički unos automatski obrađuje i ne mogu direktno uticati na SQL upite, čime se smanjuje rizik od SQL Injection napada.
2. **XSS (Cross-Site Scripting):** *Flask* koristi *Jinja2* kao svoj template mehanizam, koji automatski kodira HTML specijalne karaktere u prikazu. Ovo pomaže u sprečavanju XSS napada tako što se sprečava izvršenje zlonamernog *JavaScript* koda koji bi mogao biti ubačen u korisnički unos.

13.1. Content Security Policy (CSP)

Content Security Policy (CSP) je sigurnosna mera koja pomaže u zaštiti veb stranica od napada kao što su XSS (*Cross-Site Scripting*) i druge vrste napada koji uključuju umetanje zlonamernog sadržaja. CSP omogućava vlasnicima veb stranica da definišu koje izvore sadržaja su dozvoljeni, čime se smanjuje mogućnost da napadači ubace i izvrše zlonamerni kod.

Koristi HTTP zaglavlje *Content-Security-Policy* za definisanje sigurnosnih pravila koja govore pretraživaču šta je dozvoljeno da se učita i izvrši na stranici. Ovo zaglavlje se šalje iz servera ka pretraživaču i sadrži pravila za različite vrste resursa kao što su skripte, stilovi, slike itd.

Na slici 13.1.1. prikazana su CSP pravila za veb aplikaciju definisana koristeći *Python* rečnik. Neka od pravila su:

- **img-src:**
 - Definiše izvore sa kojih se mogu učitavati slike.
 - Dozvoljena slike sa istog domena (self) i sa lokalnog servera (127.0.0.1:9001).
- **script-src:**
 - Definiše izvore sa kojih se mogu učitavati JavaScript skripte.
 - Dozvoljava skripte sa istog domena, sa lokalnog servera i dodatne domena kao što je npr. *Google*-ov za potrebe integracije reCAPTCHA.
- **style-src:**
 - Definiše izvore sa kojih se mogu učitavati CSS stilovi.
- **frame-src:**
 - Definiše izvore sa kojih se mogu učitavati okviri (<iframe>).
- **default-src:**
 - Definiše podrazumevane izvore sa sve tipove sadržaja koji nisu eksplicitno definisani


```

local_host = 'https://127.0.0.1:9001'

csp = {
    'default-src': ['"self"', local_host],
    'img-src': ['"self"', local_host],
    'script-src': ['"self"', local_host, "https://www.google.com", "https://www.gstatic.com", "'unsafe-inline'"],
    'style-src': ['"self"', local_host, 'data', "'unsafe-inline'"],
    'frame-src': [
        local_host,
        'https://www.google.com',
        'https://www.gstatic.com',
    ],
},
}

```

13.1.1. CSP pravila

Definisana pravila se prosleđuju instanci klase *Talisman* iz biblioteke *Flask-Talisman* koja omogućava upravljanje sigurnosnim zaglavlјima, što je u našem slučaju *Content Security Policy*.

```

talisman = Talisman(app, content_security_policy=csp)

```

13.1.1. Instanca klase *Talisman*

13.2. WTForms: Rukovanje unosom

WTForms je popularna biblioteka u *Pythonu* koja se koristi za kreiranje i validaciju formi u *Flask* veb aplikacijama. Ova biblioteka omogućava lako kreiranje HTML formi kroz Python kod, zajedno sa automatskom validacijom unetih podataka.

Ključne karakteristike WTForms-a:

- **Kreiranje formi kroz Python kod:** Umesto ručnog pisanja HTML formi, WTForms omogućava definisanje formi kao Python klasu. Svako polje forme je predstavljeno kao atribut klase.
- **Validacija:** WTForms dolazi sa ugrađenim validatorima koji omogućavaju validaciju unetih podataka. Neki od validatora su: *DataRequired*, *Email*, *Length*. Validatori treba da osiguraju da su podaci ispravni pre njihove obrade i unosa u bazu podataka.
- **Custom validatori:** WTForms omogućava kreiranje sopstvenih validatora, što znači da se mogu definisati specifični uslovi koje uneti podaci moraju da zadovolje. Ovo je korisno kada se radi o specifičnim pravilima koja ne pokrivaju ugrađeni validatori.
- **Integracija sa Flask-om:** *WTForms* se lako integriše sa *Flask* aplikacijama preko *Flask-WTF* ekstenzije. Ovo omogućava jednostavno rukovanje CSRF zaštitom, validacijom formi i generisanjem HTML-a direktno iz Python koda.

- **Podrška za različite tipove polja:** *WTForms* podržava različite tipove polja, uključujući tekstualna polja, imejl polja, lozinke, *checkbox*-ove, radio dugmad, i mnoge druge. Ovo omogućava fleksibilnost pri kreiranju različitih tipova formi.

```
class RegistrationForm(FlaskForm):
    current_username = None
    current_email = None

    first_name = StringField('First Name', validators=[DataRequired(), Length(max=24)])
    last_name = StringField('Last Name', validators=[DataRequired(), Length(max=24)])
    username = StringField('Username', validators=[DataRequired(), Length(max=24), validate_username])
    email = StringField('Email', validators=[DataRequired(), Email(), Length(max=32), validate_email])
    password = PasswordField('Password', validators=[DataRequired(), Length(min=8, max=64, message='Password must be between 8 and 64 characters long.')])
    confirm_password = PasswordField('Confirm Password', validators=[DataRequired(), EqualTo('password')])
    birth_date = DateField('Birth Date', validators=[DataRequired()], format='%Y-%m-%d')
    submit = SubmitField('Register')

    def __init__(self, pwned_service: PwnedService, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.pwned_service = pwned_service

    def validate_password(self, password_field):
        password = password_field.data

        if self.pwned_service.is_password_compromised(password):
            raise ValidationError('Password is compromised and cannot be used.')
```

13.2.1. Forma za registraciju korisnika

```
def validate_username(form, field):
    if form.current_username != None and form.current_username == field.data:
        return
    if not user_repo.is_username_unique(field.data):
        raise ValidationError('Username is already in use.')

def validate_email(form, field):
    if form.current_email != None and form.current_email == field.data:
        return
    if not user_repo.is_email_unique(field.data):
        raise ValidationError('Email is already in use.')
```

13.2.2. Custom validatori

14. Zaštita sesija

Sesija u veb aplikacijama predstavlja privremenu komunikaciju između korisnika i servera koja traje tokom jednog korišćenja aplikacije. HTTP protokol je stateless, što znači da svaki zahtev koji korisnik šalje serveru ne sadrži nikakvu informaciju o prethodnim zahtevima. Sesije omogućavaju serveru da „zapamti“ korisnika između različitih HTTP zahteva.

Kada korisnik pristupi aplikaciji, server kreira jedinstveni identifikator sesije, obično u obliku kolačića (cookie), koji se šalje korisniku. Ovaj kolačić se šalje zajedno sa svakim narednim zahtevom, omogućavajući aplikaciji da prepozna korisnika i pristupi njegovim podacima, kao što su korisnički profili, prethodni izbori, ili druga personalizovana podešavanja.

Elementi sesije uključuju

- **Identifikator Sesije (Session ID):** Jedinstveni token koji identifikuje sesiju korisnika i obično se čuva u kolačiću na strani klijenta.
- **Podaci Sesije:** Informacije o korisniku koje server koristi za personalizaciju i autentifikaciju. Ovi podaci se mogu čuvati u kolačićima klijenta (serijalizovani i enkriptovani), na serveru ili u bazi podataka i povezani su sa identifikatorom sesije.

Ključne Karakteristike Sesija

- **Jedinstvenost:** Svaka sesija ima jedinstveni identifikator koji je vezan za određenog korisnika.
- **Privremeno skladištenje:** Sesijski podaci se čuvaju dok korisnik ne zatvori pregledač ili dok sesija ne istekne.
- **Sigurnost:** Pošto sesije čuvaju informacije o korisnicima, važno je da budu adekvatno zaštićene kako bi se sprečili napadi poput krađe sesije (*session hijacking*).

14.1. Flask sesija

U *Flask* aplikacijama, podrazumevano čuvanje sesija je putem kolačića na klijentskoj strani, gde se podaci sesije automatski serijalizuju i enkriptuju od strane servera pre nego što se pošalju klijentu. Sesija se kreira prilikom prve posete ruti za prijavu i sadrži samo oznaku da li je sesija „sveže“ kreirana i CSRF token (slika 14.1.1.).

```
{  
  "_fresh": false,  
  "csrf_token": "f78adbba42fdf9c12716344dbbf068c9b0fc1edb"  
}
```

14.1.1. Primer dekodovane sesije neprijavljenog korisnika

Nakon što se korisnik uspešno prijavi, kreira se nova sesija koja pored polja `_fresh` i `csrf_token` sadrži (slika 14.1.2.):

- `_id`: Jedinstveni identifikator sesije. Ovaj ID se koristi za praćenje i upravljanje sesijama na strani servera.
- `_user_id`: Jedinstveni identifikator korisnika koji je prijavljen. Povezuje sesiju sa specifičnim korisnikom i koristi se za pristup korisničkim podacima.

```
{
  "_fresh": true,
  "_id": "f05796b5b0e476fe29b087cc8681591d8a9b98693229b5c625cefca7b73769873985ede7604a2917e2a937b19f7a9ca2737d9c2db914275c4dd79f961468ef52",
  "_user_id": "fef392db-e2ad-43f9-9fb1-c627a76dade3",
  "csrf_token": "f78adbb42fd9c12716344dbbf068c9b0fc1edb"
}
```

14.1.2. Primer dekovodane sesije prijavljenog korisnika

Prvi korak u zaštiti *Flask* sesije je generisanje jakog tajnog ključa na nivou aplikacije (`SECRET_KEY`). Preporuka je da se koristi kriptografski siguran generator slučajnih brojeva za generisanje ovog ključa. Jedan od preporučenih načina je korišćenje `os.urandom()` i `binascii.hexlify()` funkcija u *Python*-u, koji osiguravaju dovoljno dugačak i nepredvidiv ključ.

```
import os
import binascii

def generate_secret_key():
    return binascii.hexlify(os.urandom(24)).decode()

secret_key = generate_secret_key()
print(secret_key)
```

14.1.3. Kod za generisanje tajnog ključa

Tajni ključ se generiše samo jednom i čuva se na sigurnom mestu, kao što je *environment* promenljiva ili posebna konfiguraciona datoteka. Korišćenjem ovog ključa, osigurava se jaka zaštita za *Flask* sesiju.

Pored generisanja jakog tajnog ključa, treba razmotriti i sledeće mehanizme zaštite sesije:

- **Secure Cookies:** Postavljanjem opcije `SESSION_COOKIE_SECURE = True`, osigurava se da se sesijski kolačići prenose isključivo preko HTTPS veze. Ovo sprečava da kolačići budu presretnuti od strane napadača na nezaštićenim HTTP vezama. Bez ove postavke, sesijski kolačići mogu biti poslati putem nesigurnih veza, što omogućava napadaču da ih presretne, ukrade kolačić i potencijalno preuzme korisničku sesiju (*session hijacking*).

- **HttpOnly Cookies:** Opcija `SESSION_COOKIE_HTTPONLY = True` zabranjuje pristup sesijskim kolačićima putem *JavaScript*-a, čime se smanjuje mogućnost napada putem skripti, kao što su *Cross-Site Scripting* napadi. Ovaj atribut osigurava da, čak i ako napadač uspe da izvrši maliciozni kod na korisnikovom pretraživaču, neće moći da ukrade sesijske kolačiće.
- **Ograničeno trajanje sesije:** Postavljanjem `PERMANENT_SESSION_LIFETIME`, može se kontrolisati koliko dugo korisnička sesija ostaje aktivna pre nego što istekne. Na primer, može se koristiti `timedelta(minutes=30)` da se ograniči trajanje sesije na 30 minuta neaktivnosti. Ova mera osigurava da sesije ne ostanu aktivne predugo, što smanjuje rizik od zloupotrebe ako korisnik zaboravi da se odjavi. Štiti od *session fixation* i *session hijacking* napada.
- **SameSite:** Postavljanjem `SESSION_COOKIE_SAMESITE = 'Lax' ili 'Strict'`, ograničava se način na koji se sesijski kolačići šalju u različitim kontekstima. *'Lax'* omogućava slanje kolačića sa istog domena, ali ih blokira u većini situacija kada se zahtevi šalju sa drugih sajtova, čime se smanjuje rizik od CSRF napada. *'Strict'* pruža još viši nivo zaštite, potpuno blokirajući slanje kolačića u kontekstu *cross-site* zahteva.
- **Session protection:** Postavljanjem `SESSION_PROTECTION = 'strong'`, Flask će dodatno zaštititi sesije od neovlašćenih promena. Ova opcija detektuje promene u ključnim podacima, kao što su korisnička IP adresa ili *user-agent* i ako se promena detektuje, sesija će biti poništena. Ovo pomaže u sprečavanju otmice sesija (*session hijacking*), gde napadač pokušava da preuzme korisničku sesiju sa druge lokacije.

15. HTTP zaglavlja

HTTP zaglavlja igraju ključnu ulogu u sigurnosti web aplikacija, jer omogućavaju klijentu i serveru da razmenjuju informacije o načinu obrade zahteva i odgovora. Neadekvatna ili neispravna konfiguracija HTTP zaglavlja može dovesti do ozbiljnih bezbednosnih propusta, kao što su *Cross-Site Scripting* (XSS), *Clickjacking*, *MIME sniffing*, i drugi napadi. Zbog toga je važno implementirati odgovarajuće zaštitne mere za HTTP zaglavlja kako bi se smanjio rizik od ovih napada.

15.1. Konfiguracija bezbednosnih zaglavlja koristeći Talisman

Flask-Talisman je ekstenzija za Flask koja služi za unapređenje sigurnosti veb aplikacija kroz jednostavno postavljanje sigurnosnih HTTP zaglavlja.

Strict-Transport-Security (HSTS)

- Ovo zaglavlje osigurava da pretraživači komuniciraju sa serverom isključivo putem HTTPS-a i to ne samo za glavni domen, već i za sve poddomene. Takođe štiti od napada kao što su *man-in-the-middle* napadi.

- Primer: `strict_transport_security=True` - Ova opcija automatski postavlja `max-age=31536000` (jedna godina) i `includeSubDomains=True`, što znači da HSTS važi za sve poddomene.

X-Content-Type-Options

- Ova opcija sprečava da pretraživač preskoči MIME vrste koje su specificirane u HTTP zaglavlju i koristi ih za interpretaciju sadržaja. To smanjuje rizik od napada koji koriste MIME *type sniffing*.
- Primer: `x_content_type_options=True` - Ovo postavlja zaglavlje na nosniff, koje sprečava *MIME type sniffing*.

X-Frame-Options

- Ova opcija sprečava da se stranica učitava unutar iframe-a na drugom sajtu, što može sprečiti napade kao što su *clickjacking*.
- Primer: `frame_options="DENY"` - Ova postavka sprečava da se stranica učitava u iframe-u sa bilo koje domene.

Referrer-Policy

- Ova opcija kontroliše koje informacije o izvoru (*referrer*) se šalju sa svakim zahtevom, pružajući kontrolu nad privatnošću i sigurnošću. Na primer, može se sprečiti slanje izvornog URL-a prilikom prelaska sa jedne stranice na drugu.
- Primer: `referrer_policy="strict-origin-when-cross-origin"` - Ovo postavlja politiku tako da se referer šalje samo ako je iz iste domene ili preko HTTPS-a.

Permissions-Policy

- Ova opcija omogućava kontrolu nad pristupom različitim funkcionalnostima pretraživača, kao što su geolokacija, kamera, mikrofoni, itd. Pomaže u minimiziranju potencijalnih sigurnosnih rizika.

Session Cookie Secure

- Postavka `SESSION_COOKIE_SECURE=True` osigurava da se sesijski kolačići šalju samo preko HTTPS veze. To sprečava da kolačići budu presretnuti ili ukradeni putem nesigurnih HTTP veza.

16. Rukovanje izuzecima i logovanje

Rukovanje Izuzecima i Logovanje

Efikasno rukovanje izuzecima i pravilno logovanje predstavljaju ključne komponente za očuvanje integriteta, bezbednosti i performansi savremenih veb aplikacija. Ove prakse ne samo da omogućavaju pravilan odgovor na greške i probleme tokom rada aplikacije, već i pomažu u održavanju visokog nivoa korisničkog iskustva i funkcionalnosti sistema.

Rukovanje Izuzecima

Rukovanje izuzecima je proces koji omogućava aplikaciji da se nosi sa neočekivanim i nepredviđenim situacijama koje mogu nastati tokom njenog rada. To uključuje, ali nije ograničeno na, nevalidne ulaze od strane korisnika, greške u komunikaciji sa bazom podataka, neuspehe u komunikaciji sa spoljnim servisima, i druge vrste runtime grešaka. Pravilno rukovanje izuzecima osigurava da aplikacija može da odgovori na ove situacije na način koji minimizira negativan uticaj na korisničko iskustvo i funkcionalnost aplikacije. Na primer, umesto da aplikacija iznenada prekine rad ili prikaže korisniku tehničke detalje, ona može pružiti korisniku korisne poruke o grešci, beležiti detalje o problemu za dalju analizu.

Logovanje

Logovanje je proces beleženja informacija o događajima i radnjama koje se dešavaju tokom rada aplikacije. Kvalitetno logovanje omogućava razvojnom timu da prati rad aplikacije u realnom vremenu, identifikuje i analizira greške i prati performanse sistema. Logovi mogu sadržati razne informacije, uključujući, ali ne ograničavajući se na, vreme nastanka događaja, vrstu događaja (greška, upozorenje, informacija), korisničke podatke, IP adrese, informacije o zahtevima i odgovorima, kao i specifične greške i izuzetke.

Pravilno formatiranje i obuhvat logova igra ključnu ulogu u analizi grešaka i problema, omogućavajući brzu dijagnostiku i rešavanje problema. Logovi takođe pomažu u identifikaciji obrazaca koji mogu ukazivati na potencijalne probleme u aplikaciji ili infrastrukturi, kao što su neobični obrasci korišćenja ili sigurnosni incidenti.

16.1. Globalno rukovanje izuzecima

Globalno rukovanje izuzecima predstavlja pristup u kojem se izuzetci i greške u aplikaciji obrađuju centralizovano, na jednom mestu, kako bi se obezbedila doslednost u upravljanju greškama i pojednostavila dijagnostika problema. Ova strategija omogućava aplikaciji da se na sistematičan način nosi sa greškama, pružajući korisnicima jasne i korisne poruke o greškama i olakšavajući analizu problema.

Prednosti Globalnog Rukovanja Izuzecima

- **Doslednost u obrađivanju grešaka:** Globalno rukovanje izuzecima omogućava dosledno obrađivanje svih izuzetaka u aplikaciji. Umesto da se greške rešavaju na mestu gde se javljaju, globalni *handler* obezbeđuje centralizovano mesto za obradu grešaka, što pomaže u održavanju doslednosti u ponašanju aplikacije i porukama koje se prikazuju korisnicima.
- **Smanjenje dupliranja koda:** Korišćenje globalnog rukovanja izuzecima smanjuje potrebu za dupliranjem logike obrade grešaka u različitim delovima aplikacije. Umesto da svaki deo aplikacije individualno rukuje greškama, globalni *handler* može obraditi sve greške na centralizovan način, što olakšava održavanje i unapređenje aplikacije.
- **Olakšana analiza problema:** Globalno rukovanje izuzecima omogućava centralizovano logovanje grešaka, što olakšava analizu i praćenje problema u aplikaciji. Sve greške se beleže na jednom mestu, što omogućava razvojnom timu da lakše identifikuje obrasce i potencijalne probleme.
- **Unapređenje korisničkog iskustva:** Pružanje poruka o greškama poboljšava korisničko iskustvo. Globalni *handler* može obezbediti prikaz razumljivih i korisnih poruka korisnicima, čime se smanjuje konfuzija i omogućava korisnicima da se lakše nose sa problemima koji nastaju tokom korišćenja aplikacije.

U Flask aplikacijama, globalno rukovanje izuzecima se najčešće implementira korišćenjem `@app.errorhandler` dekoratora, koji pruža mogućnost centralizovane obrade izuzetaka. Ovaj dekorator omogućava definisanje specifičnih funkcija koje će se aktivirati svaki put kada se dogodi određeni tip izuzetka. Na primer, kada aplikacija naiđe na grešku poput *404 Not Found*, funkcija označena odgovarajućim handlerom će se automatski pozvati, omogućavajući personalizovanu obradu greške.

Korišćenje `@app.errorhandler` dekoratora je naročito korisno jer omogućava da se na jednom mestu definišu odgovori na najčešće izuzetke, poput *500 Internal Server Error*, koji označava neočekivanu grešku na strani servera, ili *403 Forbidden*, koja označava da korisnik nema dozvolu za pristup određenom resursu. Ovakva centralizacija omogućava dosledno rukovanje greškama kroz celu aplikaciju, što olakšava održavanje i unapređenje koda.


```

def register_error_handlers(app):
    @app.errorhandler(403)
    def forbidden_error(error):
        if current_user.is_authenticated:
            if hasattr(app, 'security_logger'):
                app.security_logger.log_access_denied(current_user)
        return render_template('403.html'), 403

    @app.errorhandler(Exception)
    def handle_unhandled_exception(e):
        current_app.security_logger.log_unhandled_exception(e)
        return render_template('500.html', error=str(e)), 500

    @app.errorhandler(404)
    def handle_unhandled_exception(e):
        current_app.security_logger.log_unhandled_exception(e)
        return render_template('404.html', error=str(e)), 404

    @app.errorhandler(AccountNotVerifiedError)
    def handle_account_not_verified_exception(e):
        current_app.security_logger.log_failed_login(e.email, str(e))
        flash(str(e), 'info')
        return redirect(url_for('main.info'))

    @app.errorhandler(SignatureExpired)
    def handle_signature_expired_exception(e):
        current_app.security_logger.log_signature_expired(str(e))
        return redirect_successfully('main.info', 'The OTP has expired.', 'error')

```

16.1.1. Deo modula za globalno rukovanje izuzecima

16.2. Logovanje

Logovanje je ključni aspekt upravljanja aplikacijom, posebno u kontekstu sigurnosti i dijagnostike. Ono omogućava praćenje toka izvršavanja aplikacije, identifikovanje potencijalnih problema i pruža dragocene informacije za rešavanje incidenata. Pravilno logovanje ne samo da poboljšava stabilnost aplikacije, već i olakšava pronalaženje i ispravljanje grešaka, kao i analiziranje sigurnosnih događaja.

U kontekstu sigurnosnih preporuka, logovanje mora biti pažljivo planirano kako bi se osigurala privatnost korisnika i zaštita osetljivih podataka. Logovanje ne bi trebalo da uključuje informacije kao što su lozinke, kompletni brojevi kreditnih kartica ili bilo koji drugi podaci koji mogu narušiti privatnost korisnika. Umesto toga, trebalo bi da se loguju ključni događaji kao što su pokušaji prijave, promene u korisničkim privilegijama, greške u pristupu resursima, ili neuspešni pokušaji autentifikacije.

Jedan od najvažnijih aspekata logovanja je odabir odgovarajućeg nivoa logovanja za različite tipove događaja. Na primer, INFO nivo može se koristiti za logovanje uspešnih operacija, kao što su uspešna prijava korisnika ili promena lozinke. S druge strane, WARNING nivo može se koristiti za događaje koji

ukazuju na potencijalne probleme, kao što su neuspešni pokušaji prijave ili pokušaji pristupa resursima bez odgovarajućih privilegija. ERROR nivo je rezervisan za ozbiljnije probleme koji utiču na funkcionalnost aplikacije, dok se CRITICAL nivo koristi za situacije koje zahtevaju hitnu pažnju, kao što su kritične greške ili sigurnosni incidenti.

Pored toga, logove treba čuvati na siguran način, uz obezbeđivanje pristupa samo ovlašćenim osobama. Logovi bi trebalo da budu zaštićeni od neovlašćene izmene ili brisanja, a u slučaju bezbednosnih incidenata, ovi zapisi mogu biti ključni za analizu i reakciju na pretnje.

Konačno, logovanje ne sme da utiče na performanse aplikacije, te je važno implementirati logovanje na način koji je efikasan i koji minimizuje uticaj na vreme odziva aplikacije. Korišćenje alata za centralizovano logovanje i analizu logova može značajno poboljšati efikasnost praćenja i reagovanja na događaje unutar aplikacije.

Funkcija *register_loggers* (slika 16.2.1) ima zadatak da postavi i registruje dva posebna *logger*-a unutar Flask aplikacije: *security_logger* i *entity_logger*. Oba logger-a su konfigurisana za različite svrhe logovanja, koristeći klase *SecurityLogger* i *EntityLogger*.

SecurityLogger(app, log_file, max_bytes, backup_count)

Ova klasa se koristi za kreiranje *logger*-a koji će pratiti i beležiti sigurnosne događaje, kao što su pokušaji prijave, promena privilegija, neautorizovani pristup, itd.

- **log_file='app/logs/security.log'**: Log fajl u kojem će se čuvati zapisi. U ovom slučaju, zapisi se čuvaju u fajlu *security.log* unutar direktorijuma *app/logs*.
- **max_bytes=1024*1024**: Ova vrednost definiše maksimalnu veličinu log fajla u bajtovima pre nego što se započne rotacija log fajla. U ovom slučaju, veličina je ograničena na 1 MB.
- **backup_count=5**: Ova vrednost određuje koliko starih verzija log fajla će se čuvati. Kada trenutni log fajl dostigne maksimalnu veličinu, rotira se i do 5 starih verzija fajla se čuva kao backup.

```
def register_loggers(app):
    app.security_logger = SecurityLogger(app, log_file='app/logs/security.log', max_bytes=1024*1024, backup_count=5)
    app.entity_logger = EntityLogger(app, log_file='app/logs/entity.log', max_bytes=1024*1024, backup_count=5)
```

16.2.1. Funkcija za registrovanje logera

```

class SecurityLogger:
    def __init__(self, app=None, log_file='security.log', max_bytes=1024*1024, backup_count=5):
        self.logger = logging.getLogger('security')
        self.log_file = log_file
        self.max_bytes = max_bytes
        self.backup_count = backup_count
        if app:
            self.init_app(app)

    def init_app(self, app):
        self.logger.setLevel(logging.INFO)
        handler = RotatingFileHandler(
            self.log_file, maxBytes=self.max_bytes, backupCount=self.backup_count
        )
        handler.setFormatter(logging.Formatter(
            '%(asctime)s - %(levelname)s - %(message)s'
        ))
        self.logger.addHandler(handler)

    def log_successful_login(self, user):
        self.logger.info(
            f"User {user.email} (ID: {user.id}) successfully logged in at {datetime.now().isoformat()} "
            f"from IP {request.remote_addr} using {request.headers.get('User-Agent')}. Session ID: {request.cookies.get('session')}."
        )

    def log_failed_login(self, user_email, error_message):
        self.logger.warning(
            f"Failed login attempt for user {user_email} at {datetime.now().isoformat()} "
            f"from IP {request.remote_addr} using {request.headers.get('User-Agent')}. Reason: {error_message}"
        )

```

16.2.2. Deo klase za logovanje

```

1  2024-08-30 20:27:51,271 - WARNING - Unauthenticated request from IP 127.0.0.1 attempted to access main.index using Mozilla/5.0 (Windows NT 10.0;
2  2024-08-30 20:28:09,843 - INFO - OTP sent to admin nemanja@gmail.com via email at 2024-08-30T20:28:09.843698. IP Address: 127.0.0.1. User-Agent:
3  2024-08-30 20:29:48,186 - INFO - OTP sent to admin nemanja@gmail.com via email at 2024-08-30T20:29:48.186342. IP Address: 127.0.0.1. User-Agent:
4  2024-08-30 20:30:04,264 - INFO - User nemanja@gmail.com (ID: 74ece098-63ff-424b-87a2-005b7a13f8d4) successfully logged in with OTP at 2024-08-30
5  2024-08-30 20:30:24,738 - INFO - User account with ID ab7868c9-fe5b-4c55-bb56-83613eaf8fc4 has been deleted by admin with 74ece098-63ff-424b-87a
6  2024-08-30 20:38:34,118 - INFO - User ana@gmail.com (ID: 83283082-201a-4c20-9ea2-bcd769175a79) successfully logged in at 2024-08-30T20:38:34.118
7  2024-08-30 20:38:35,270 - INFO - User with ID 83283082-201a-4c20-9ea2-bcd769175a79 has initiated a request for the AUTHOR role at 2024-08-30T20:
8  2024-08-30 21:24:55,211 - WARNING - Invalid input detected for field 'recaptcha' at 2024-08-30T21:24:55.211908 from IP 127.0.0.1 using Mozilla/5
9  2024-08-30 21:25:09,126 - INFO - User ana@gmail.com (ID: d759bc84-fdb2-4a3b-ba0b-fd6152993157) successfully logged in at 2024-08-30T21:25:09.126
10 2024-08-30 21:25:11,968 - WARNING - Unauthenticated request from IP 127.0.0.1 attempted to access main.index using Mozilla/5.0 (Windows NT 10.0;

```

16.2.3. Primer SecurityLog fajla