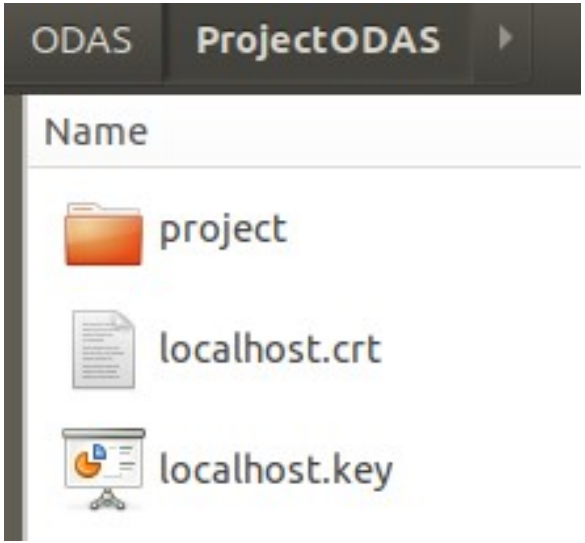
Password Manager

Certyfikat SSL



- flask run --cert localhost.crt --key localhost.key

Certificate

Szumanski

Subject Name

Country	PL
State/Province	Lublin
Locality	Lublin
Organization	Warsaw University of Technology
Common Name	Szumanski

Issuer Name

Country	PL
State/Province	Lublin
Locality	Lublin
Organization	Warsaw University of Technology
Common Name	Szumanski

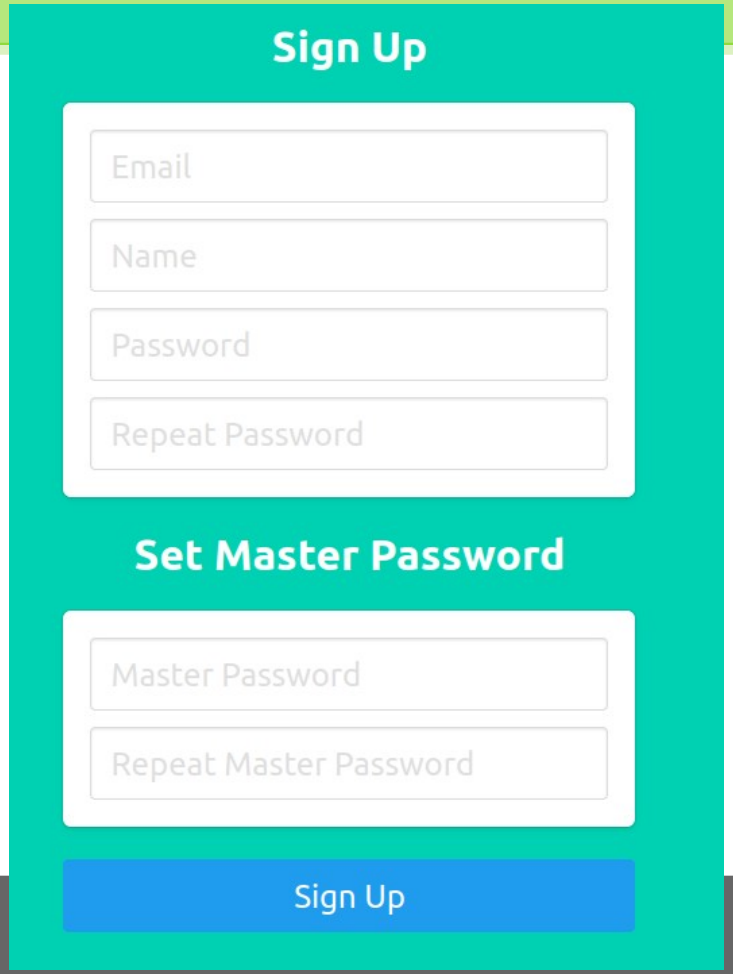
Validity

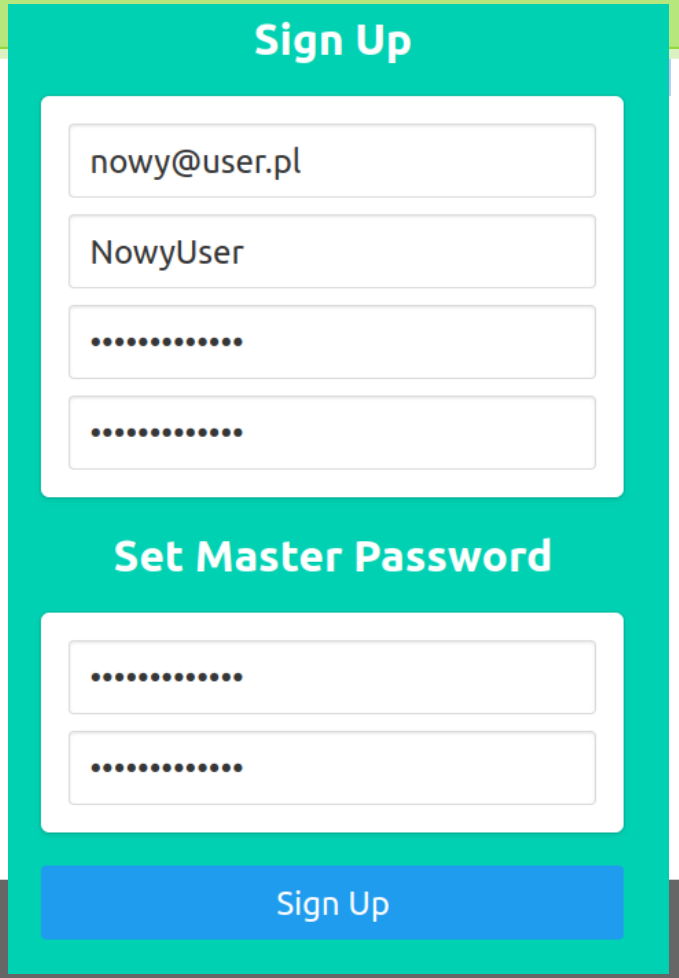
Not Before	Tue, 01 Feb 2022 12:03:12 GMT
Not After	Wed, 01 Feb 2023 12:03:12 GMT

Public Key Info

Algorithm	RSA
Key Size	4096
Exponent	65537
Modulus	D1:BC:FA:A2:05:11:D7:47:C3:2F:BE:B9:7B:72:8F:CD:38:37:1B:7A:B0:1A:0A:B…

/signup

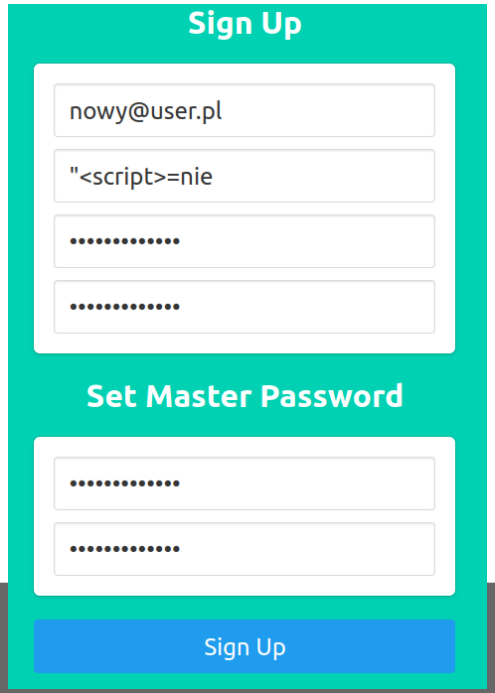

Sign Up
Email
Name
Password
Repeat Password
Set Master Password
Master Password
Repeat Master Password
Sign Up

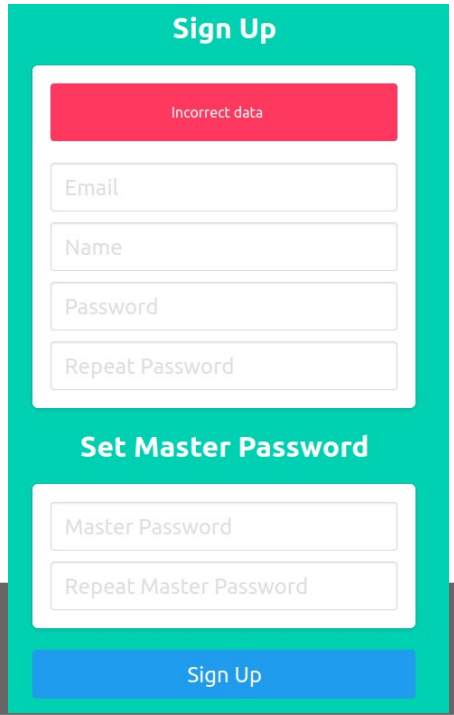

Sign Up
nowy@user.pl
NowyUser
Set Master Password
Sign Up

Weryfikacja wprowadzonych danych /signup

```python
@register.route('/signup', methods=['POST'])
def signup_post():
    if validate_email(request.form.get('email')) or validate_password(request.form.get('password')) \
        or validate_username(request.form.get('name')) or validate_password(request.form.get('repeat')) \
            or validate_password(request.form.get('master')) or validate_password(request.form.get('repeat_master')):
        flash('Incorrect data')
        return redirect(url_for('register.signup'))
```

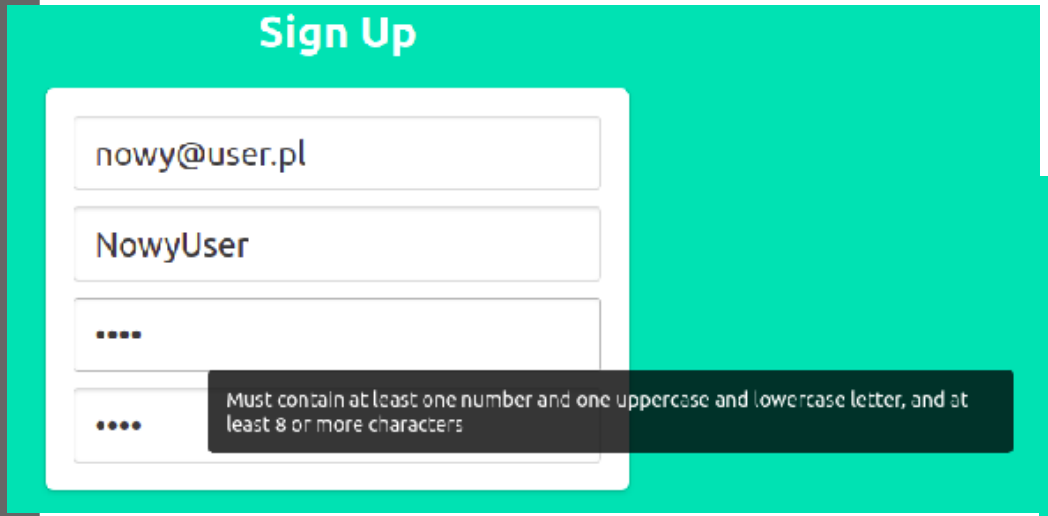
```python
def validate_email(email):
    url_pattern =  '^[a-zA-Z0-9 _\\.@]{1,}$'
    url_regex = re.compile(url_pattern)
    if re.match(url_regex, email):
        return False
    return True

def validate_username(username):
    if re.match('^[A-Za-z0-9]{1,}$', username):
        return False
    return True

def validate_password(password):
    if re.match('^.{8,30}$', password):
        return False
    return True
```

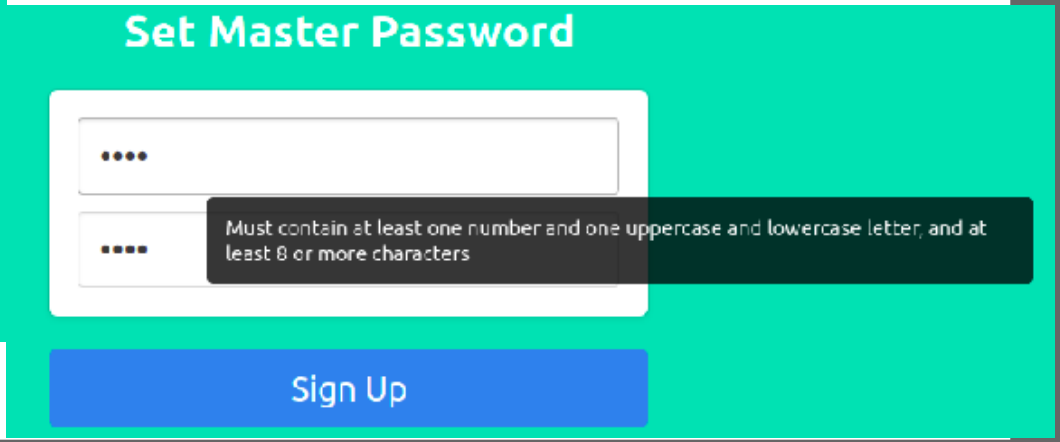




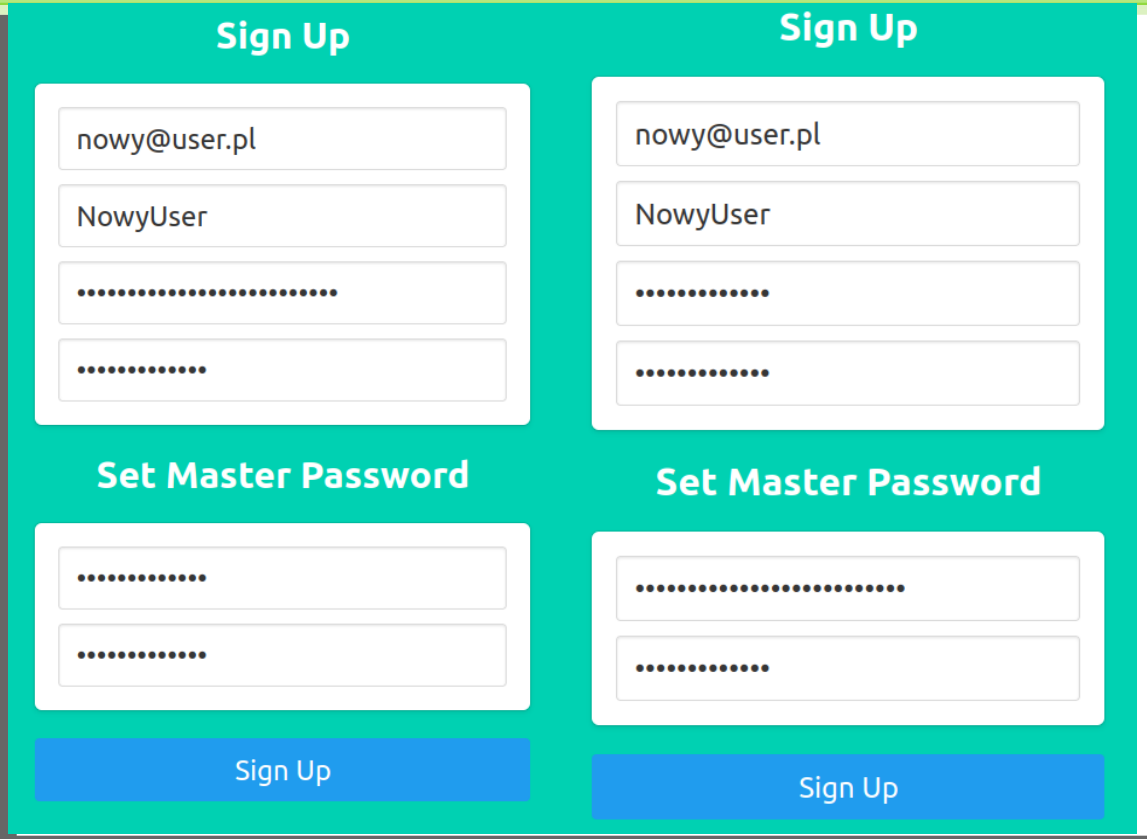
Wymuszanie silnego hasła

```
<input class="input is-large" type="password" name="password" placeholder="Password"
required  pattern="(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}"
title="Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters">
```

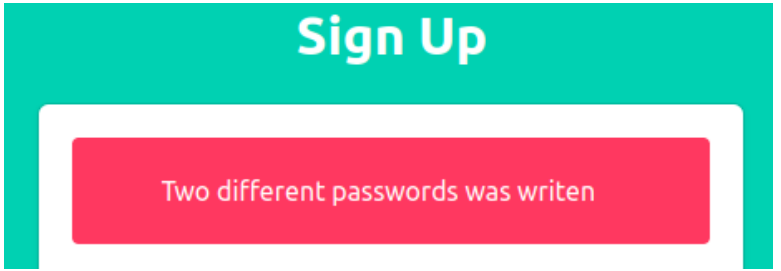


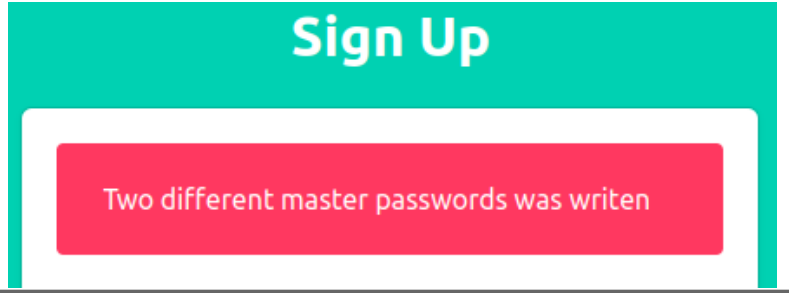


Powtórzenie hasła



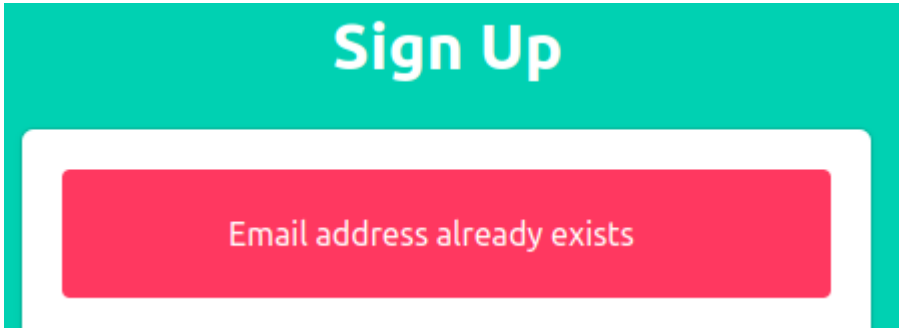

```
if password != reapeated:
    flash('Two different passwords was writen')
    return redirect(url_for('register.signup'))

if master != reapeated_master:
    flash('Two different master passwords was writen')
    return redirect(url_for('register.signup'))
```





Unikalny adres email

```
user = User.query.filter_by(email=email).first()

if user:
    flash('Email address already exists')
    return redirect(url_for('register.signup'))
```



Przechowywanie danych użytkownika

- auth – hashowane (email + password), służy do logowania
- masterpassword – hashowane (email + master password), służy do weryfikacji przy dodawaniu i odszyfrowywaniu haseł

```python
class User(UserMixin, db.Model):
    __tablename__ = 'user'
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(100), unique=True)
    auth = db.Column(db.String(200))
    masterpassword = db.Column(db.String(200))
    name = db.Column(db.String(100))
    attempts = db.Column(db.Integer)
    logedNow = db.Column(db.DateTime)
    lastlogedAt = db.Column(db.DateTime)
    lastFailedAttempt = db.Column(db.DateTime)
    passwords = db.relationship("Passwords", backref = 'user')
```

```
id INTEGER NOT NULL,
email VARCHAR(100),
auth VARCHAR(100),
masterpassword VARCHAR(100),
name VARCHAR(100),
attempts INTEGER,
"logedNow" DATETIME,
"lastlogedAt" DATETIME,
"lastFailedAttempt" DATETIME,
PRIMARY KEY (id),
UNIQUE (email)
```

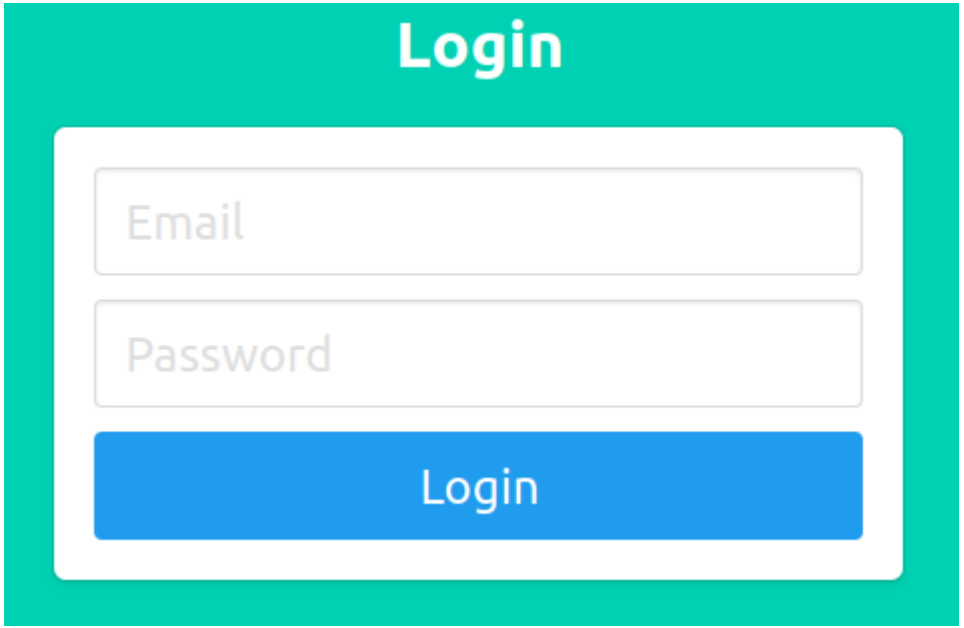
Hashowanie hasła konta i hasła głównego

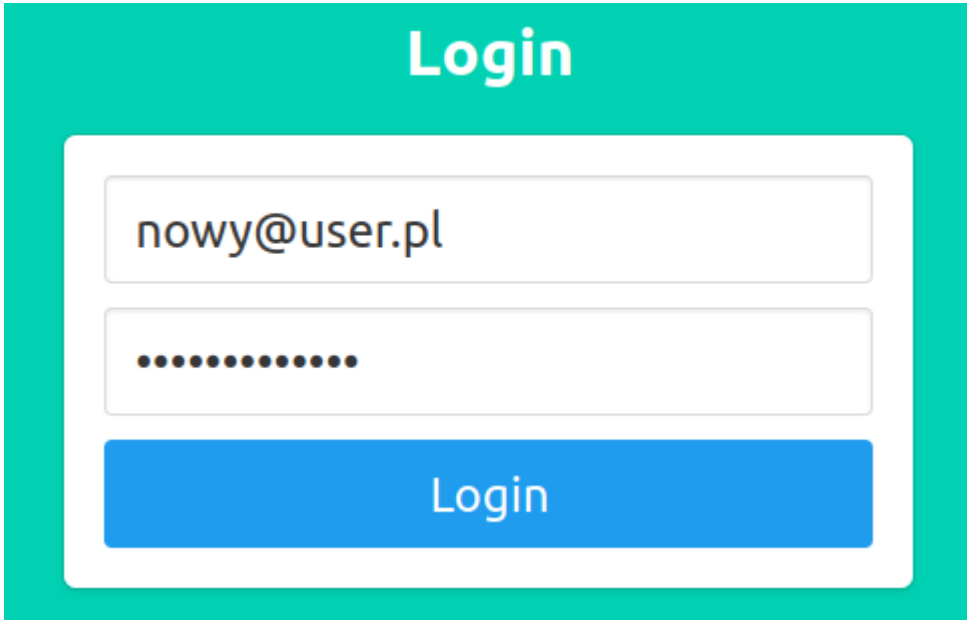
```
salt = os.urandom(16)
auth = passlib.hash.pbkdf2_sha512.using(rounds=100000, salt=salt).hash(email + password)
masterpassword = passlib.hash.pbkdf2_sha512.using(rounds=120000, salt=salt).hash(email + master)
new_user = User(email=email, name=name, attempts=0, auth=auth, masterpassword=masterpassword)
```

Hashowane za pomocą PBKDF2_SHA512 (im dłuższy hash tym mniejsza szansa na zdublowanie się hashy)

Powtarzanie tego hashowania jest różne dla auth i masterpassword, w celu różnych hashy przy wspianiu tego samego hasła do konta i hasła głównego

/login

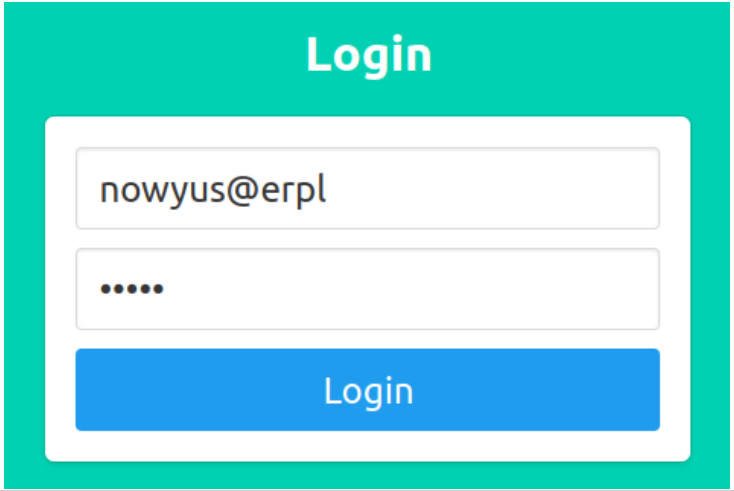


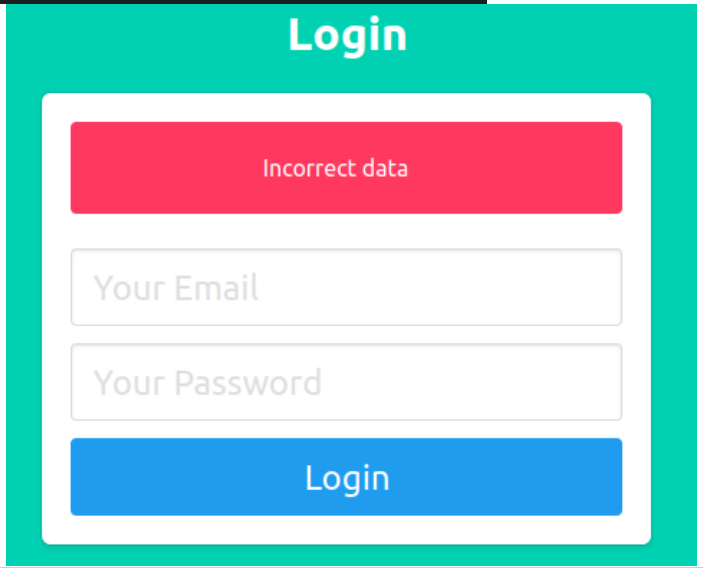


Weryfiakcja wprowadzanych danych /login

```python
@auth.route('/login', methods=['POST'])
def login_post():
    time.sleep(SLEEP_TIME)
    if validate_email(request.form.get('email')) or validate_password(request.form.get('password')):
        flash('Incorrect data')
        return redirect(url_for('auth.login'))
```

```python
SLEEP_TIME = 5
```

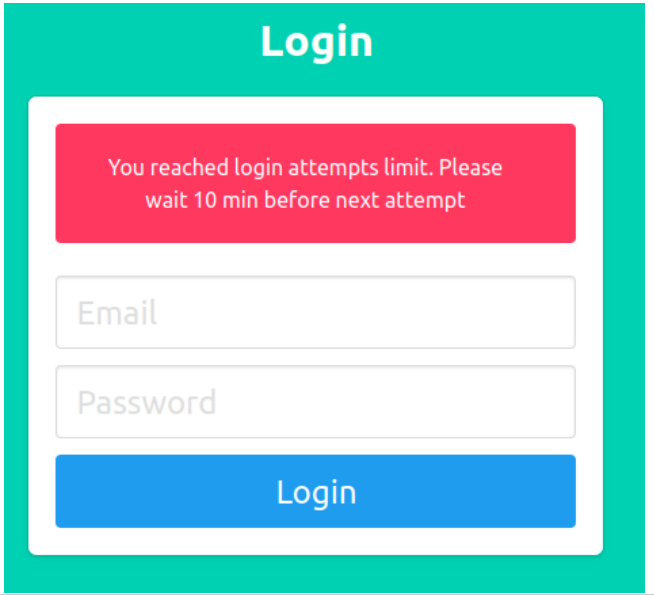




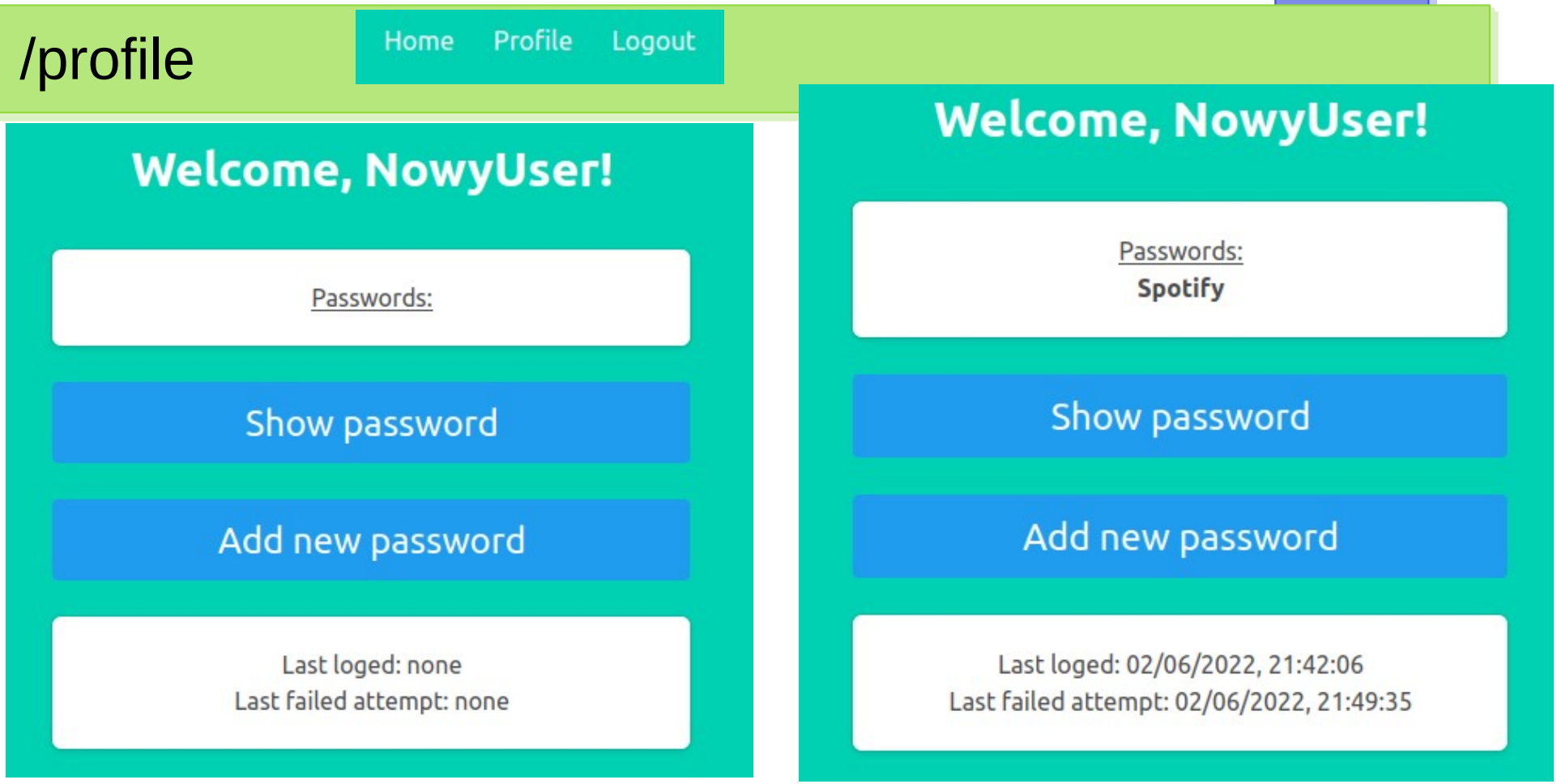
Limit prób logowania

```python
user = User.query.filter_by(email=email).first()

lastlogin = user.lastFailedAttempt
if lastlogin:
    timenow = datetime.now()
    diff = timenow - user.lastFailedAttempt
    if diff.seconds < TIMEOUT_PERIOD:
        flash('You reached your attempts limit. Please wait 10 min before next attempt')
        return redirect(url_for('auth.login'))
    if user.attempts >= MAX_ATTEMPTS:
        user.attempts = 0
        db.session.commit()

auth = email + password

if not user or not passlib.hash.pbkdf2_sha512.verify(auth, user.auth):
    att = user.attempts
    user.attempts = att + 1
    if user.attempts == MAX_ATTEMPTS:
        user.lastFailedAttempt = datetime.now()
        flash('You reached login attempts limit. Please wait 10 min before next attempt')
        db.session.commit()
        return redirect(url_for('auth.login'))
    flash('Please check your login details and try again.')
    db.session.commit()
    return redirect(url_for('auth.login'))

login_user(user)
user.logedNow = datetime.now()
user.attempts = 0
db.session.commit()
return redirect(url_for('main.profile'))
```

```python
MAX_ATTEMPTS = 5
TIMEOUT_PERIOD = 600
```



/profile


Home
Profile
Logout
Welcome, NowyUser!
Passwords:
Show password
Add new password
Last loged: none
Last failed attempt: none
Welcome, NowyUser!
Passwords:
Spotify
Show password
Add new password
Last loged: 02/06/2022, 21:42:06
Last failed attempt: 02/06/2022, 21:49:35

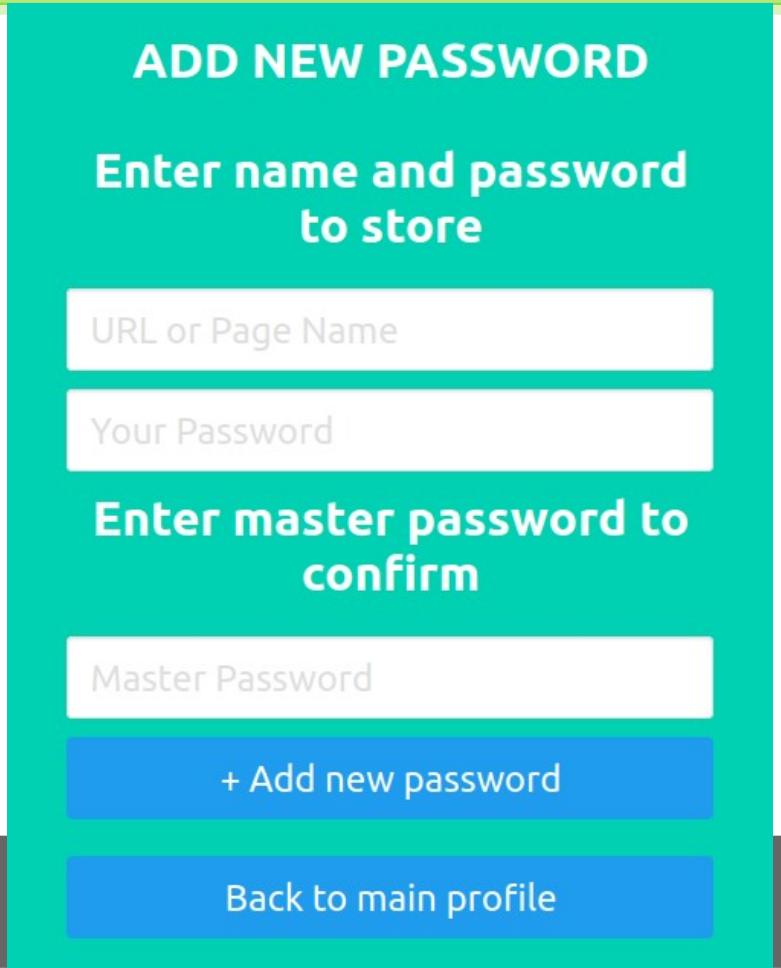
Informowanie o ostatnich działaniach

- Last loged – informuję zalogowanego użytkownika, kiedy ostatnio udało mu się zalogować
- Last failed attempt – informuję użytkownika o tym kiedy ostatni raz przekroczył limit prób autentykacji (w tym wpisywania hasła głównego)

```
login_user(user)
user.logedNow = datetime.now()
user.attempts = 0
db.session.commit()
return redirect(url_for('main.profile'))
```

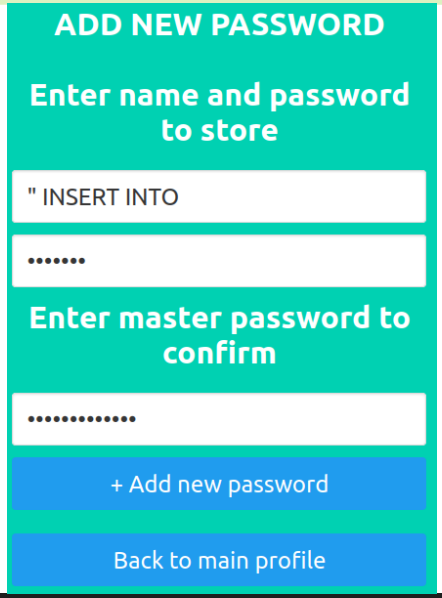

```
@auth.route('/logout')
@login_required
def logout():
    user = User.query.filter_by(email=current_user.email).first()
    loged = user.logedNow
    user.lastlogedAt = loged
    logout_user()
    db.session.commit()
    return redirect(url_for('auth.login'))
```

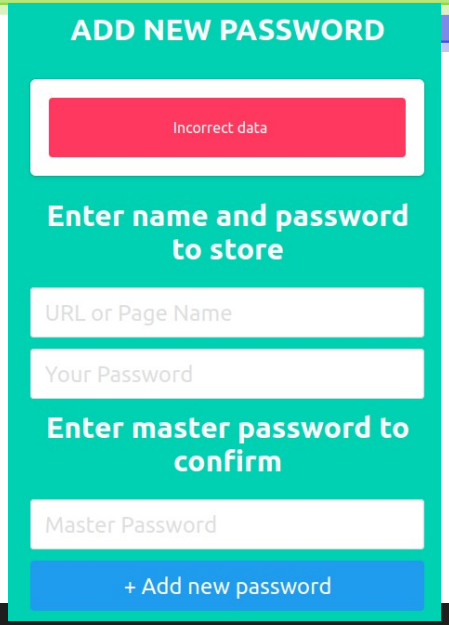
Dodawanie hasła - /addpassword



Weryfikacja wprowadzanych danych /addpassword

```python
def validate_url(url):
    url_pattern =  '^[a-zA-Z0-9 _\\/-:()!.?,]{1,}$'
    url_regex = re.compile(url_pattern)
    if (re.match(url_regex, url)):
        return False
    return True

def validate_password(password):
    if re.match('^.{1,30}$', password):
        return False
    return True

def validate_masterpassword(password):
    if re.match('^.{8,30}$', password):
        return False
    return True
```

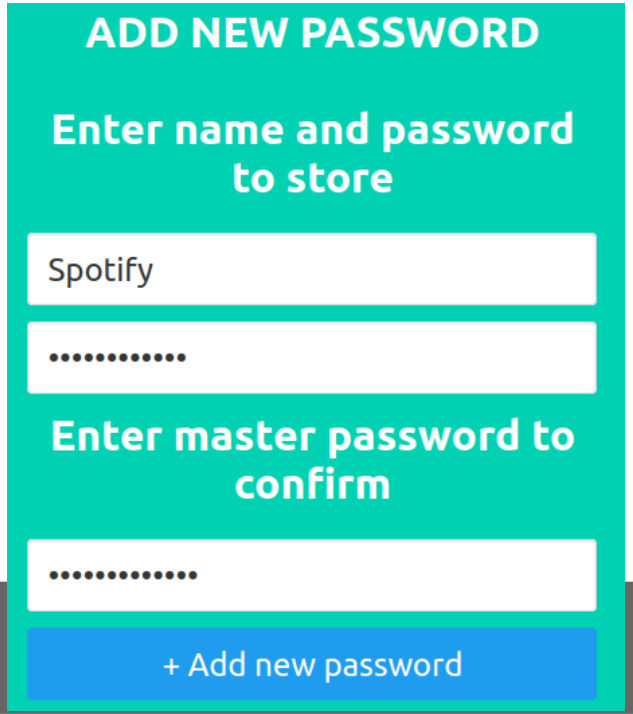


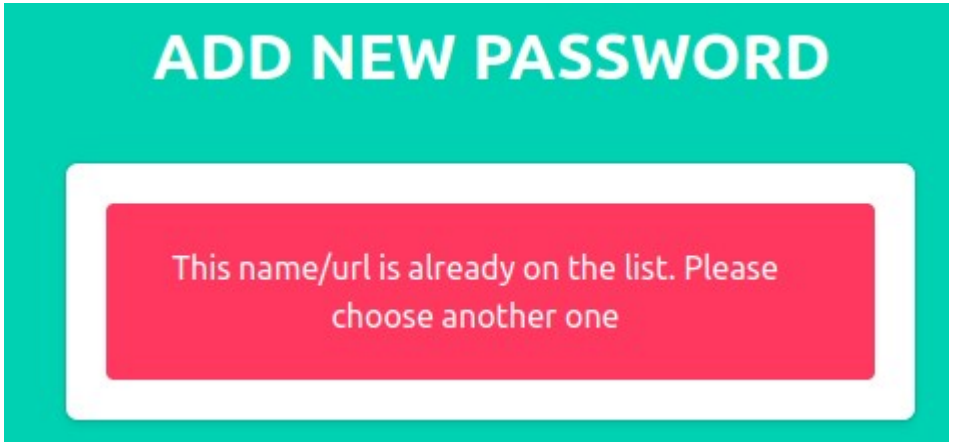



```python
@main.route('/addpassword', methods=["POST"])
@login_required
def addpassword_post():
    if validate_url(request.form.get('url')) or validate_password(request.form.get('new_password')) \
        or validate_masterpassword(request.form.get('masterp')):
        flash('Incorrect data')
        return redirect(url_for('main.addpassword'))
```

Nazwa/url unikalna dla UŻYTKOWNIKA

```
thepassword = Passwords.query.filter_by(url=url, user_id=current_user.id).first()

if thepassword:
    flash('This name/url is already on the list. Please choose another one')
    return redirect(url_for('main.addpassword'))
```





Weryfikacja hasła głównego

```python
masterp = request.form.get('masterp')
result = check_masterpassword(masterp)

if result == 1:
    flash('Wrong Master Password')
    return redirect(url_for('main.addpassword'))
elif result == 2:
    flash('You reached your attempts limit. Please wait 10 min before next attempt')
    return redirect(url_for('auth.logout'))
```

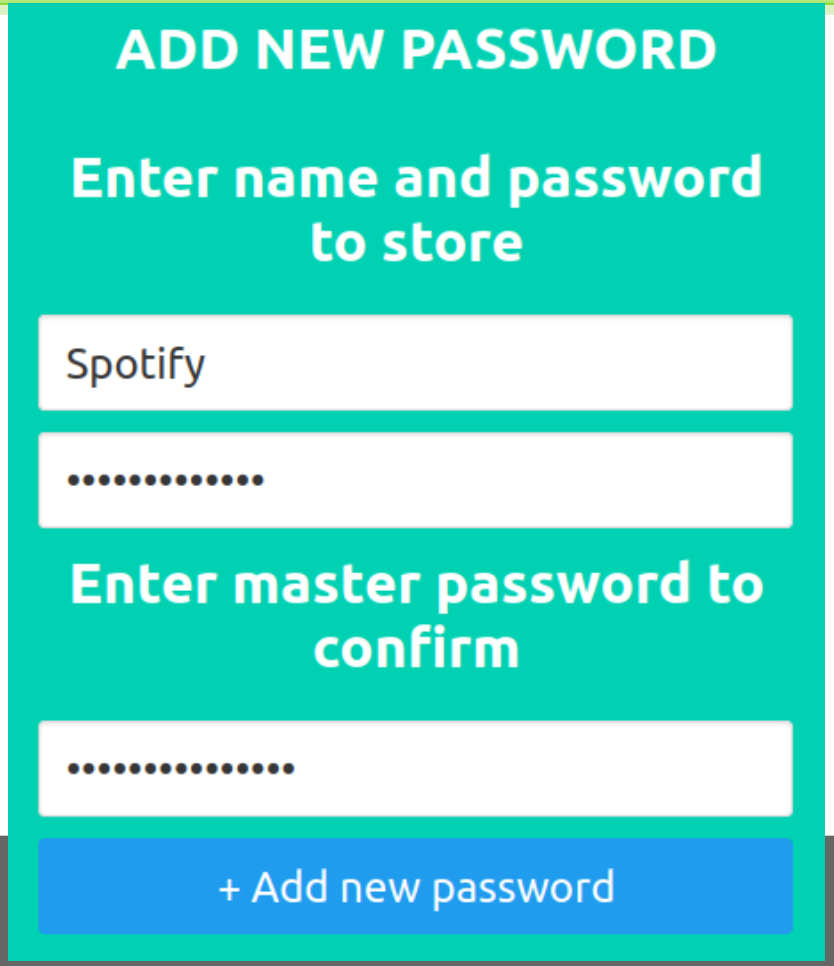
```python
SLEEP_TIME = 5
MAX_ATTEMPTS = 5
```

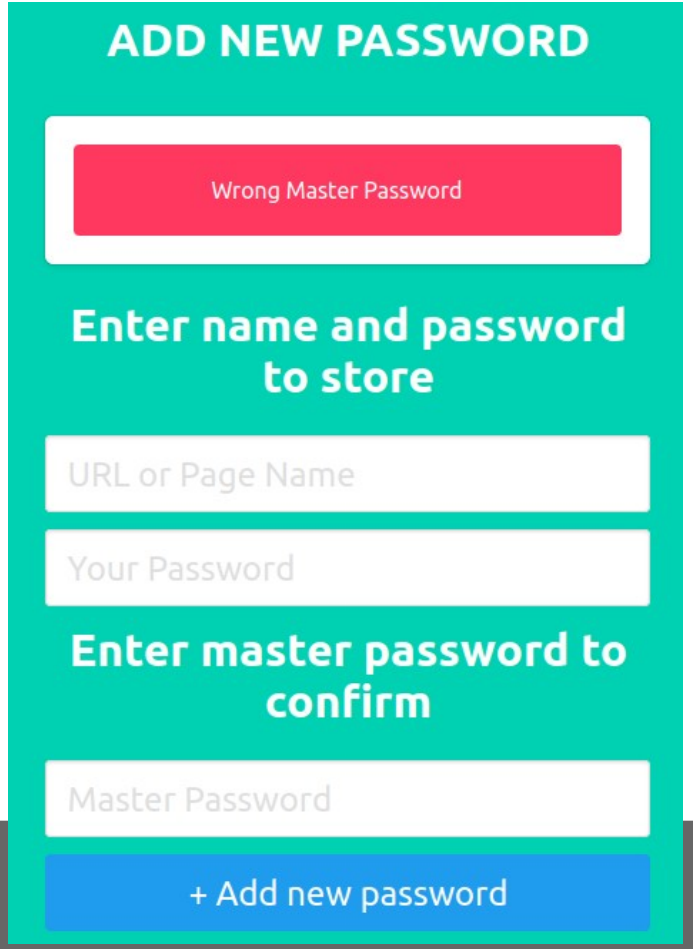
W tej weryfikacji mam również doczynienia z limitami prób, tak jak podczas logowanie

Przekroczenie tego limitu skutkuje wylogowaniem użytkownik i blokadę na określony czas

```python
def check_masterpassword(masterp):
    time.sleep(SLEEP_TIME)
    user = User.query.filter_by(id=current_user.id).first()
    masterhash = user.email + masterp

    if not passlib.hash.pbkdf2_sha512.verify(masterhash, user.masterpassword):
        att = user.attempts
        user.attempts = att + 1
        if user.attempts == MAX_ATTEMPTS:
            user.lastFailedAttempt = datetime.now()
            db.session.commit()
            return 2
        db.session.commit()
        return 1

    user.attempts = 0
    db.session.commit()
    return 0
```

Weryfikacja hasła głównego





Szyfrowanie hasła

- hasło jest szyfrowane schematem blokowym typu CBC za pomocą szyfrowania AES 256
- klucz do zaszyfrowanego hasła jest wytwarzany za pomocą PBKDF2, który hashuje 10000 razy (hasło głowne + email)

```
def encrypt(plain_text, password):
    salt = os.urandom(AES.block_size)
    iv = Random.new().read(AES.block_size)

    private_key = PBKDF2(password, salt, 32, 10000)

    padded_text = pad(plain_text)

    cipher_config = AES.new(private_key, AES.MODE_CBC, iv)
    encryption = cipher_config.encrypt(padded_text)
```

```
# pad with spaces at the end of the text
# beacuse AES needs 16 byte blocks
def pad(s):
    block_size = 16
    remainder = len(s) % block_size
    padding_needed = block_size - remainder
    return bytes(s + padding_needed * ' ', 'utf-8')
```

Przechowywanie hasła

- password – zaszyfrowane hasło wraz z wartościami potrzebmymi do jego odszyfrowania (sól + iv + zaszyfrowane hasło)

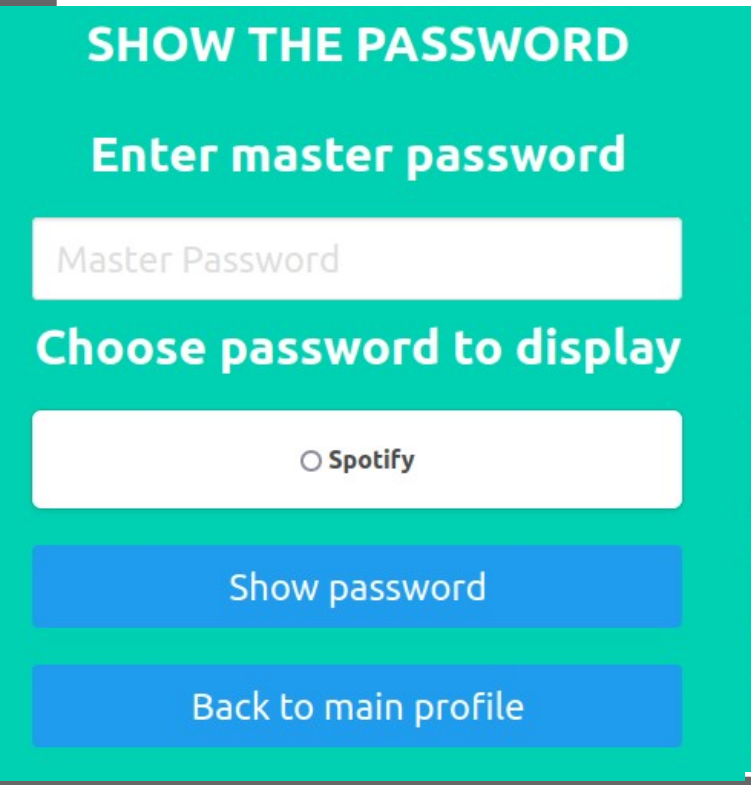
```
cipher_text = bytes.decode(base64.b64encode(encryption))
salt_str = bytes.decode(base64.b64encode(salt))
iv_str = bytes.decode(base64.b64encode(iv))

return salt_str + iv_str + cipher_text
```

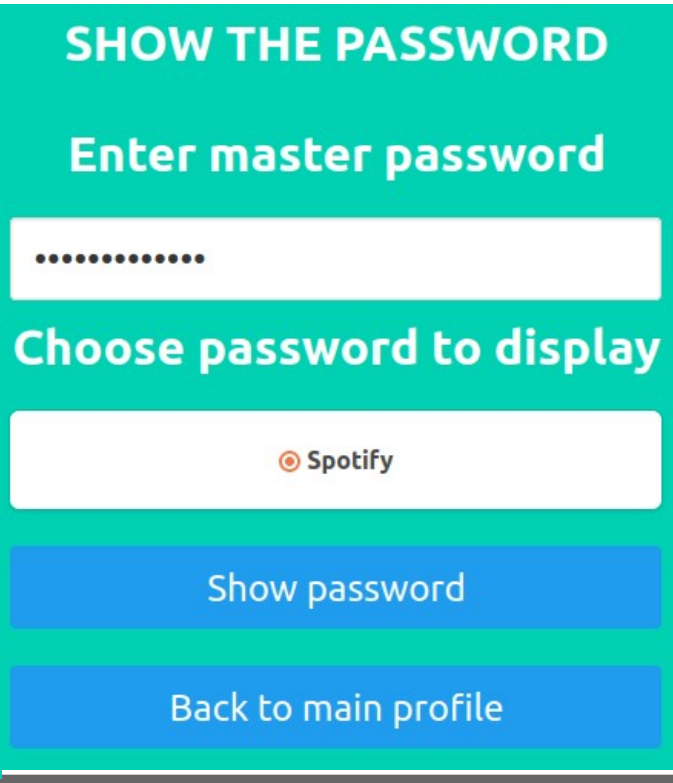
```
masterhash = masterp + current_user.email

password = request.form.get('new_password')
e_password = encrypt(password, masterhash)

new_password = Passwords(url = url, password = e_password, user_id = current_user.id)

db.session.add(new_password)
db.session.commit()

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

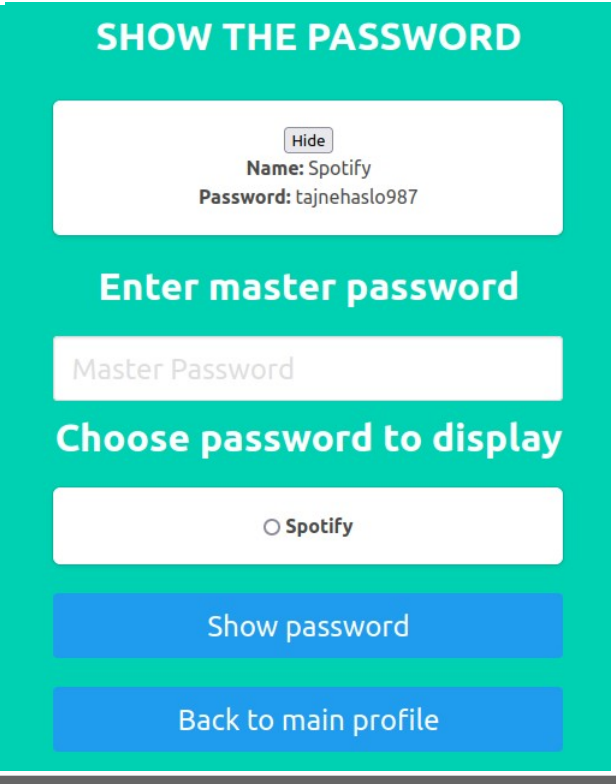
```
class Passwords(db.Model):
    __tablename__ = 'passwords'
    id = db.Column(db.Integer, primary_key=True)
    url = db.Column(db.String(1000), nullable=False)
    password = db.Column(db.String(1000))
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

```
id INTEGER NOT NULL,
url VARCHAR(1000) NOT NULL,
password VARCHAR(1000),
user_id INTEGER NOT NULL,
PRIMARY KEY (id),
FOREIGN KEY(user_id) REFERENCES user (id)
```

Pokazywanie hasła - /showpassword

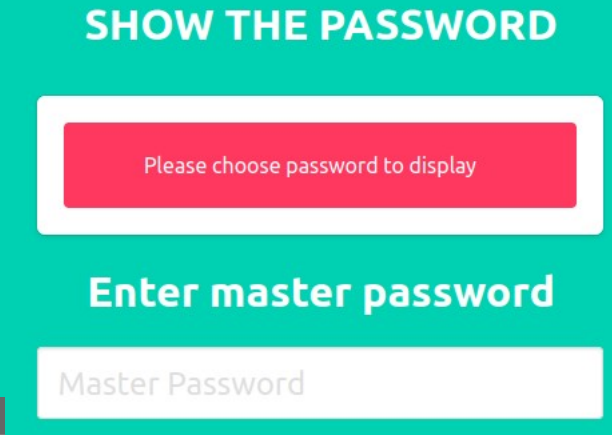


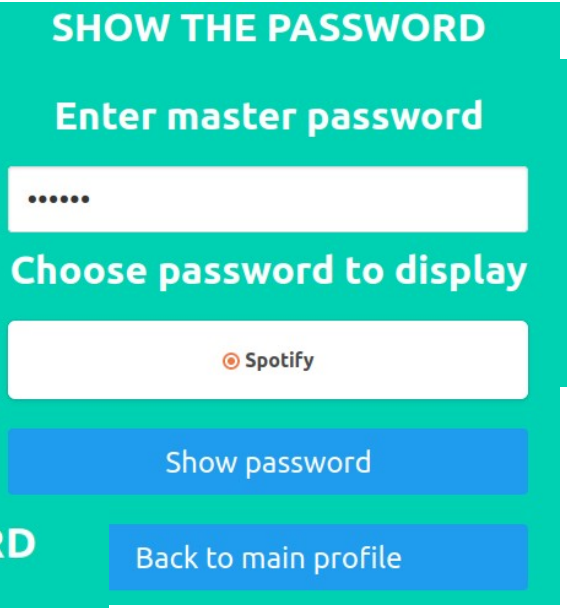


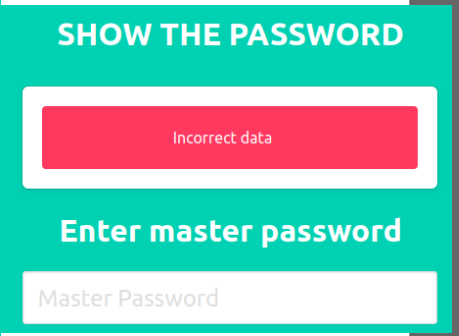


Weryfikacja wprowadanego hasła i nie wybranie opcji

```python
@main.route('/displaypassword', methods=['POST'])
@login_required
def display_post():
    if validate_masterpassword(request.form.get('masterp')):
        flash('Incorrect data')
        return redirect(url_for('main.display'))

    pname = request.form.get('options')
    if not pname:
        flash('Please choose password to display')
        return redirect(url_for('main.display'))
```



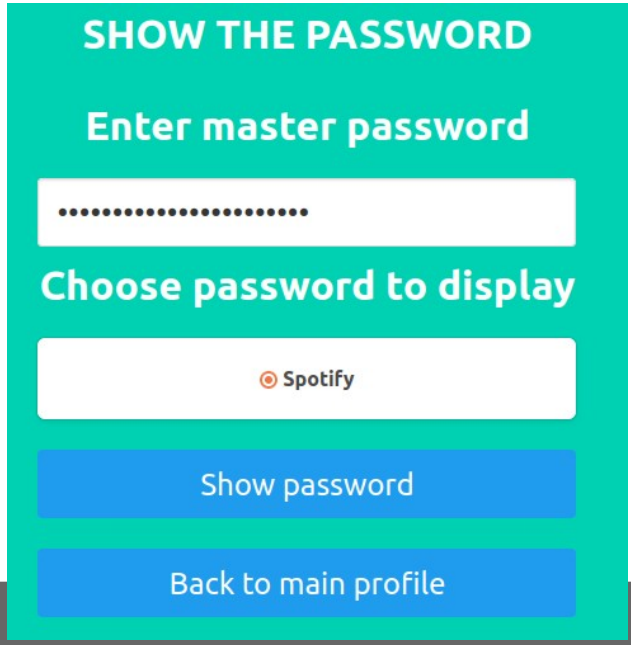




Weryfikacja hasła głównego dla /showpassword

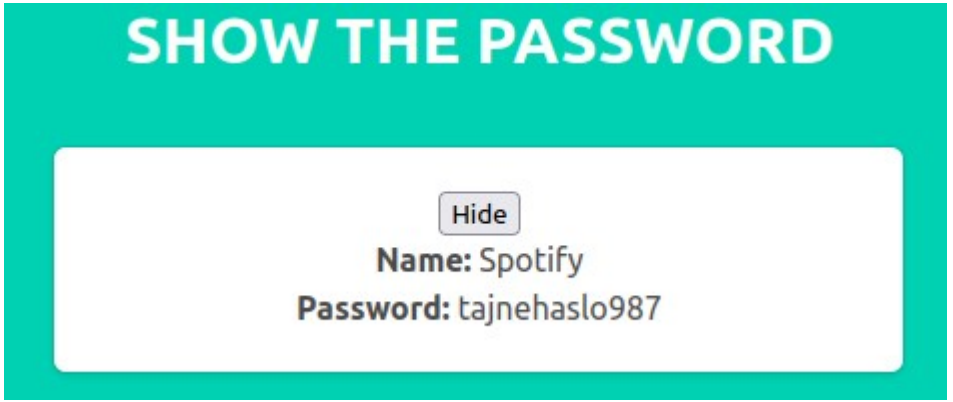
- Sposób weryfikacji hasła głównego dla /showpassword jest taki sam jak dla /addpassword, wliczając to limity prób oraz wylogowanie i blokowanie użytkownika po wykorzystaniu limitu

```
masterp = request.form.get('masterp')
result = check_masterpassword(masterp)

if result == 1:
    flash('Wrong Master Password')
    return redirect(url_for('main.display'))
elif result == 2:
    flash('You reached your attempts limit. Please wait 10 min before next attempt')
    return redirect(url_for('auth.logout'))
```



Odszyfrowywanie hasła

```python
def decrypt(encr, password):
    ssalt = bytes(encr[:24], 'utf-8')
    salt = base64.b64decode(ssalt)

    siv = bytes(encr[24:48], 'utf-8')
    iv = base64.b64decode(siv)

    senc = bytes(encr[48:], 'utf-8')
    enc = base64.b64decode(senc)

    private_key = PBKDF2(password, salt, 32, 10000)

    cipher = AES.new(private_key, AES.MODE_CBC, iv)

    decrypted = cipher.decrypt(enc)

    original = unpad(decrypted)

    return original
```




```python
# remove the extra spaces at the end
def unpad(s):
    return s.rstrip()
```

```python
masterhash = masterp + current_user.email
thepassword = Passwords.query.filter_by(url=pname, user_id=current_user.id).first()
d_password = decrypt(thepassword.password, masterhash)
password_str = bytes.decode(d_password)
passw_list = Passwords.query.filter_by(user_id = current_user.id).all()
return render_template('showpassword.html', passw_list=passw_list, url=pname, thepassword=password_str)
```

Wykorzystane biblioteki

- Flask
- Flask-Login
- Flask-Password
- Flask-SqlAlchemy
- Passlib
- Pycrypto