Sprawozdanie

Skonteneryzowanie przy pomocą Docker, aplikacja działa na porcie 8080.

```
(venv) vladislavkvinto@MacBook-Air-Vladislav NotesApp % docker-compose up
Starting notesapp_web_1 ... done
Attaching to notesapp_web_1
web_1  |  * Serving Flask app 'main.py' (lazy loading)
web_1  |  * Environment: production
web_1  |    WARNING: This is a development server. Do not use it in a production deployment.
web_1  |    Use a production WSGI server instead.
web_1  |  * Debug mode: off
web_1  |  * Running on all addresses.
web_1  |    WARNING: This is a development server. Do not use it in a production deployment.
web_1  |  * Running on http://172.18.0.2:5000/ (Press CTRL+C to quit)
```

Poniżej możemy zobaczyć konfigurację aplikacji:

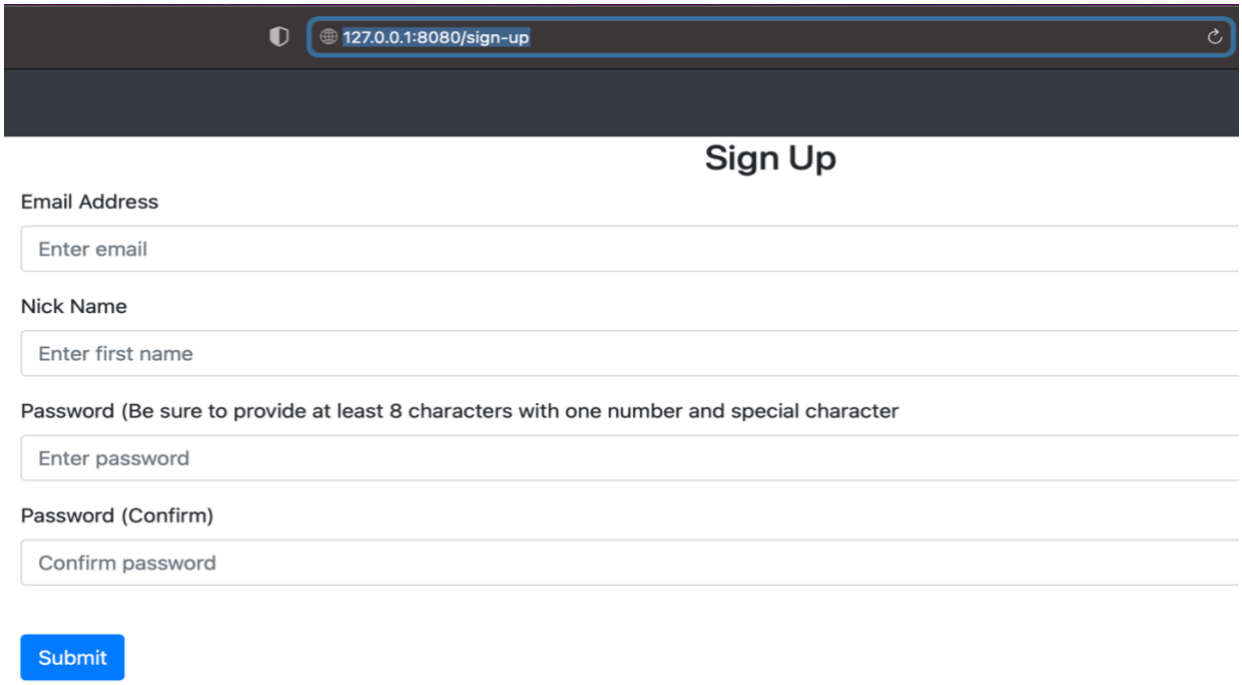
```
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}'
app.config["SIMPLEMDE_JS_IIFE"] = True
app.config["SIMPLEMDE_USE_CDN"] = True
app.config["SESSION_COOKIE_SECURE"] = True
app.config["REMEMBER_COOKIE_SECURE"] = True
app.config["REMEMBER_COOKIE_HTTPONLY"] = True
```

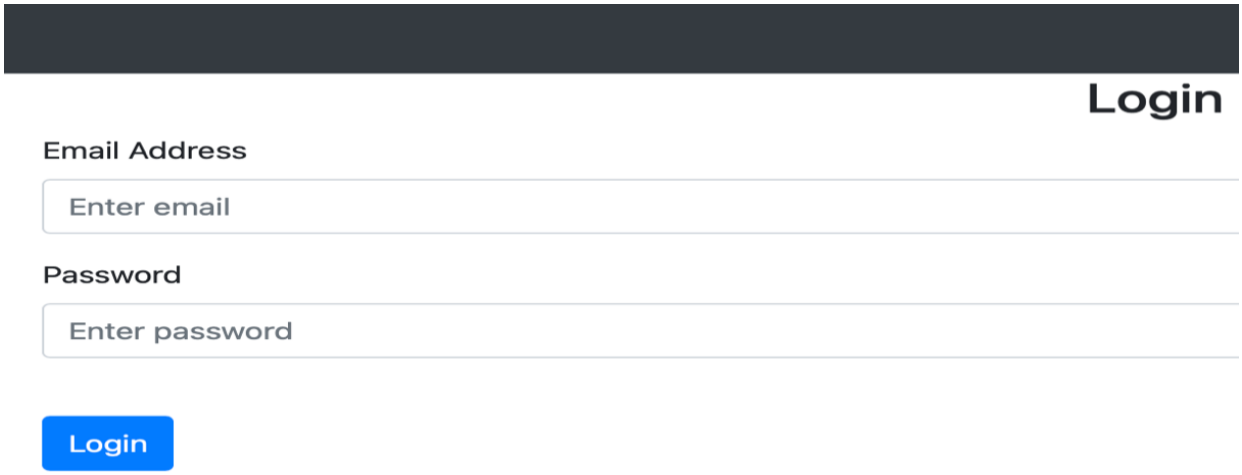
Oraz model bazy danych(sqlite):

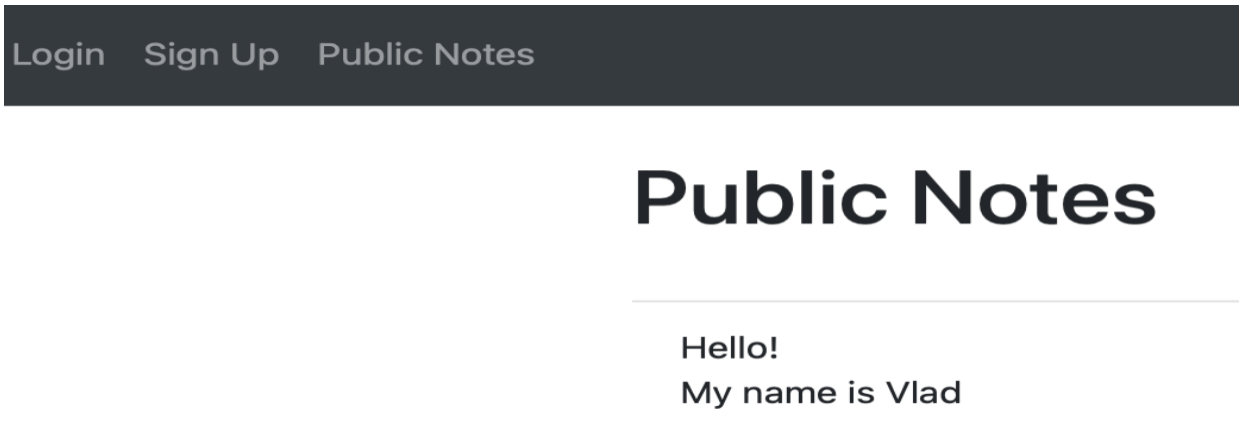
```
class Note(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    data = db.Column(db.String(10000))
    public = db.Column(db.Boolean, default=False)
    date = db.Column(db.DateTime(timezone=True), default=func.now())
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))


Vladislove228
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(150), unique=True)
    password = db.Column(db.String(150))
    ip = db.Column(db.String(100))
    new_ip = db.Column(db.String(100))
    first_name = db.Column(db.String(150))
    notes = db.relationship('Note')
```

Poniżej możemy zobaczyć wszystkie strony aplikacji:







Podstawowe ostylowanie jest realizowane za pomocą SimpleMDE Markdown Editor

Możemy zrobić notatkę jak publiczną tak i prywatną oraz usunąć ją.
Na przykładzie metody deleteNote pokażę, jak to działa

```
<button type="button" class="close" onClick="deleteNote({{ note.id }})">
```

Po kliknięciu na przycisk wywołuję się metoda deleteNote, która przekieruję dane do metody delete_note

```
function deleteNote(noteId) {
  fetch( input: "/delete-note",  init: {
    method: "POST",
    body: JSON.stringify( value: { noteId: noteId }),
  }).then((_res : Response ) => {
    window.location.href = "/";
  });
}
```

```
Vladislove228
@views.route('/delete-note', methods=['POST'])
@login_required
def delete_note():
    note = json.loads(request.data)
    noteId = note['noteId']
    note = Note.query.get(noteId)
    if note:
        if note.user_id == current_user.id:
            db.session.delete(note)
            db.session.commit()

    return jsonify({})
```

Żeby stworzyć konto trzeba podać email adres, imię oraz dwa razy hasło.

Oraz jest pobierany adres IP.

```
ip = request.environ.get('HTTP_X_FORWARDED_FOR', request.remote_addr)
```

Dla stworzenia konta trzeba spełniać podane niżej warunki:

```
if user:
    flash('Email already exists.', category='error')
elif len(email) < 5:
    flash('Email must be greater than 4 characters.', category='error')
elif '@' not in email:
    flash('Email should contain @ character', category='error')
elif len(first_name) < 3:
    flash('Nick name must be greater than 2 characters.', category='error')
elif password1 != password2:
    flash('Passwords don\'t match.', category='error')
elif len(password1) < 8:
    flash('Password must be at least 8 characters.', category='error')
elif len(password1) > 24:
    flash('Password must not be longer than 24 characters.', category='error')
elif 'admin123' in password1:
    flash('Password is too obvious', category='error')
elif not any(not c.isalnum() for c in password1):
    flash('Password should have at least one special character: !@#$%^&*()-+/', category='error')
elif not any(c.isdigit() for c in password1):
    flash('Password should have at least one number: 1234567890', category='error')
```

Jeżeli wszystko jest poprawne – tworzymy konto szyfrując hasło za pomocą pbkdf2 i sha256

```
new_user = User(email=email, first_name=first_name, ip=ip, password=generate_password_hash(
    password1, method='pbkdf2:sha256:100', salt_length=16))
db.session.add(new_user)
db.session.commit()
await take_login(new_user)
flash('Account created!', category='success')
return redirect(url_for('views.home'))
```

Po stworzeniu konta jest wywołana metoda take_login, która potrzebna do stworzenia opóźnienia

```
async def take_login(user, delay=1):
    await asyncio.sleep(delay)
    login_user(user, remember=True)
```

Przy logowaniu też jest pobierany jest adres IP

```
ip = request.environ.get('HTTP_X_FORWARDED_FOR', request.remote_addr)

if ip in ips:
    ips[ip] = {'count': ips[ip]['count'] + 1, 'time': datetime.now()}
else:
    ips[ip] = {'count': 1, 'time': datetime.now()}

if (ips[ip]['time'] - datetime.now()).total_seconds() > 30:
    ips[ip]['count'] = 1
```

Poniżej jest przedstawiony proces logowania:

```
    user = User.query.filter_by(email=email).first()
    if user:
        if check_password_hash(user.password, password):
            if ip == user.ip and user.new_ip is not None:
                flash('Logged in successfully. New device accessing your account found: ' + str(user.new_ip),
                      category='success')
            elif ip == user.ip and user.new_ip is None:
                user.new_ip = ip
                db.session.commit()
                flash('Logged in successfully!', category='success')
            elif ip != user.ip:
                user.new_ip = ip
                flash('Logged in successfully!', category='success')
            await take_login(user)
            login_user(user, remember=True)
            return redirect(url_for('views.home'))
        else:
            flash('Incorrect password, try again.', category='error')
            if ips[ip]['count'] > 5:
                abort(403)
    else:
        flash('Email does not exist.', category='error')
        if ips[ip]['count'] > 5:
            abort(403)

return render_template("login.html", user=current_user)
```

Przy poprawnym podaniu email i hasła jest wywołane opóźnienie i wtedy logujemy się.

Jeżeli hasło zostało niepoprane podane 5 razy w ciągu 30 sekund to wywołamy błąd 403(zabronienie wstępu), to samo jest i przy niepoprawnym podaniu email.