

DOCUMENTAȚIA DE IMPLEMENTARE

MODUL DE SECURITATE – SKY SECURITY

NOKIA

- DIANA ANTON -

- OVIDIU PROȚIUC -

- ALEX BLAJ -

01.06.2021

CUPRINS

1. GENERALITĂȚI	3
1.1. DENUMIRE	3
1.2. CONTEXTUL PROIECTULUI	3
1.3. COD	3
1.3.1 Funcția check_phone()	3
1.3.2. Funcția admin_settings_update()	3
1.3.1. Funcția user_home_run()	5
1.3.2. Funcția sign_up()	6
1.3.1. Funcția update_permissions()	7
1.4. SCURTĂ DESCRIERE	8
1.5. STRUCTURA (ARHITECTURA) SISTEMULUI - STILUL MVC	8
1.6. FUNCȚIILE SISTEMULUI	9
2. MIJLOACE DE VERIFICARE	10
3. CONDIȚII DE FUNCȚIONARE	12
3.1. TEST 1 - Implementarea modulului într-o aplicație externă	12
3.2. TEST 2 - Actualizarea datelor utilizatorului curent	12
3.3. TEST 3 - Înregistrarea unui utilizator	12
4. CONDIȚII DE VERIFICARE	12
4.1. TEST 1 - Implementarea modulului într-o aplicație externă	12
4.2. TEST 2 - Actualizarea datelor utilizatorului curent	14
4.3. TEST 3 - Înregistrarea unui utilizator	14

1. GENERALITĂȚI

1.1. DENUMIRE

Sky Security - Modul de Securitate

1.2. CONTEXTUL PROIECTULUI

În momentul în care avem de-a face cu mai multe conturi de utilizatori care sunt folosite de către angajații unei firme pentru a accesa diverse resurse sau aplicații avem nevoie de o soluție pentru a gestiona și limita accesul acestora strict la ceea ce au nevoie pentru a-și desfășura activitatea. Pentru a simplifica acest proces, noi punem la dispoziție o bază de date unde este implementată o logică ce ajută la gestionarea eficientă a accesului, cât și o interfață web care simplifică interacțiunea cu datele încât orice persoană să își poată asuma rolul de administrator asupra sistemul, chiar dacă nu deține anumite cunoștințe tehnice.

Un exemplu foarte bun la un nivel mult mai mare și mai complex a unui modul asemănător este un Identity and Access Module utilizat cu predominanță de către furnizorii de servicii Cloud pentru a oferi clienților o soluție de gestionare a resurselor.

1.3. COD

1.3.1 Funcția check_phone()

```
def check_phone(phone_number):  
    if re.search(r"^\d{3}\d{3}\d{4})$", phone_number):  
        return True  
    else:  
        return False
```

În aceasta funcție se va verifica formatul numărului de telefon (trebuie să aibă 10 cifre) cu ajutorul unui regex. Dacă numărul este valid va returna True, în caz contrar False.

1.3.2. Funcția admin_settings_update()

```
@app.route('/admin_settings_update', methods = ['POST'])  
def admin_settings_update():  
    # if the user is not logged in, redirect him/her to the login page  
    is_logged_in()  
  
    #empty dictionary to store information about the user  
    data_user = {}
```

```

username = str(user_name[0])
id = str(user_id[0])
counter = 0
# site method for update form is POST
if request.method == 'POST':
    if request.form.get('full_name'):
        #if user completed full_name field it will be added to dictionary
        date_user['full_name'] = request.form.get('full_name')
        counter +=1
    if request.form.get('username'):
        date_user['username'] = request.form.get('username')
        counter +=1
    else:
        date_user['username'] = 0
        if request.form.get('phone_number') and
check_phone(request.form.get('phone_number')):
        date_user['phone_number'] = request.form.get('phone_number')
        counter +=1
    elif not request.form.get('phone_number'):
        date_user['phone_number'] = 0
    else:
        flash("Invalid phone number")
    if request.form.get('email') and check_email(request.form.get('email')) :
        date_user['email'] = request.form.get('email')
        counter +=1
    elif not request.form.get('email'):
        date_user['email'] = 0
    else:
        flash("Not a valid email. Try again.")
    if request.form.get('password') and request.form.get('new_password'):
        if request.form.get('password') == request.form.get('new_password'):
            date_user['password'] = sha256_crypt.hash(request.form.get('password'))
            counter +=1
        else:
            flash("Password doesn't match")
    elif not request.form.get('password') and not request.form.get('new_password'):
        date_user['password'] = 0
sql = "UPDATE users SET "
# looping through dictionary and create the query to find which dates must be updated
for key, value in date_user.items():
    if value != 0:
        sql += key + "= " + "'" + value + "'"
        sql += ","
# remove the last character ", "
sql = sql[:-1]
#update from id user
sql += " WHERE id = " + id + ";"
# database connection
conn = mariadb.connect(host=DB_HOST, user=DB_USER, password=DB_PASSWORD,
                        database=DB_DATABASE)
try:
    cur = conn.cursor(buffered = True)
    if counter!=0:
        cur.execute(sql)
        flash("Successfully updated!")
    conn.commit()

```

```

        cur.close()
        conn.close()
    except mariadb.Error as error:
        print("Failed to read data from table", error)
    finally:
        if conn:
            conn.close()
            print('Connection to db was closed!')
    return redirect("/admin_settings")

```

Această funcție reprezintă acțiunea pe care o va face formularul în momentul apăsării butonului UPDATE. Utilizatorul va putea alege câmpurile pe care dorește să le schimbe (ex: doar username, username + email). După ce câmpurile care trebuie modificate au fost completate, se va forma o interogare sql cu datele completate în câmpuri. Interogarea sql se va executa, iar după vom puteam observa cum în pagina html datele s-au schimbat.

1.3.1. Funcția user_home_run()

Această funcție selectează toate permisiunile asociate grupurilor din care face parte utilizatorul curent, iar în baza acestora vor fi grupate în funcție de aplicația asupra căreia acționează și afișate în cadrul paginii “Apps” care reprezintă prima pagină pe care o va vedea utilizatorul după ce se conectează.

```

@app.route('/user_home')
def user_home_run():
    # if the user is not logged in, redirect him/her to the login page
    is_logged_in()
    # list of queries
    queries = []
    app_perms_list = {}
    # create query to get the permissions of the user based on the groups that
    # the user is a part of
    queries.append(
        "SELECT * FROM permissions p "
        "INNER JOIN groups_perm_relation gp ON gp.perm_id = p.id "
        "WHERE gp.group_id IN ( "
        "SELECT g.id FROM groups g "
        "INNER JOIN user_groups_relation ug ON ug.group_id = g.id "
        "WHERE ug.user_id = " + str(user_id[0]) + ");")
    # database connection
    conn = mariadb.connect(host=DB_HOST, user=DB_USER, password=DB_PASSWORD,
        database=DB_DATABASE)
    try:
        cur = conn.cursor(buffered=True)
        # get all the permissions for this user based on the groups that
        # the user is a part of
        cur.execute(queries[0])
        permissions = cur.fetchall()
        # store the distinct app ids
        app_ids = []
        # create the list of unique apps that the user has access to
        for p in permissions:
            if p[3] not in app_ids:
                app_ids.append(p[3])
        # select the app names and ids based on the list created
        queries.append("SELECT * FROM apps WHERE id IN")

```

```

        + form_delete_id_string(app_ids, True))

# fetch the apps
cur.execute(queries[1])
apps = cur.fetchall()
# store the id and name in a {id : name} format
app_id_name = {}
for app in apps:
    app_id_name[app[0]] = app[1]
# initialize the dictionary with the app name as a key and an empty list that
# will be filled later with the permissions
for name in app_id_name.values():
    app_perms_list[ name ] = []
# for each permission get the app_id and search it in the app_id_name
# to get the name of the app and append the permission to the list
# that has the name as a key
for p in permissions:
    if p[3] in app_id_name.keys():
        is_in_list = False
        for perm in app_perms_list[ app_id_name[p[3]] ]:
            if p[1] == perm[1]:
                is_in_list = True
                break
        if not is_in_list:
            app_perms_list[ app_id_name[p[3]] ].append(p)
# close the connection
cur.close()
conn.close()
except mariadb.Error as error:
    print("Failed to read data from table", error)
finally:
    if conn:
        conn.close()
        print('Connection to db was closed!')
# return the page with all the data stored in the app_perms_list variable
return render_template('user_files/user_home.html', app_perms_list = app_perms_list)

```

1.3.2. Funcția sign_up()

```

@app.route('/sign_up_run', methods=['POST', 'GET'])
def do_admin_sign_up():
    request_sign = request.form
    # if the sign up button was pressed
    if request_sign.get('login'):
        return redirect("/")
    full_name = str(request_sign.get('full_name', False))
    user_name = str(request_sign.get('user_name', False))
    email = str(request_sign.get('email', False))
    phone = str(request_sign.get('phone', False))
    password = str(request_sign.get('password', False))
    pass_conf = str(request_sign.get('confirm_password', False))
    sign_up_pers1 = sign_up_pers(full_name, user_name, email, phone, password, pass_conf)
    conn = mariadb.connect(host=DB_HOST, user=DB_USER, password=DB_PASSWORD,
        database=DB_DATABASE)
    try:
        cur = conn.cursor(buffered=True)
        cur.execute(sign_up_pers1.select())
        users_rows = cur.fetchall()
        for row in users_rows:
            if user_name == row[0]:

```

```

        flash('Invalid User Name.')
        return redirect("/sign_up")
    if not (sign_up_pers1.check_pass(pass_conf) and sign_up_pers1.check_email()):
        flash('Please check your sign up details and try again.')
        return redirect("/sign_up")
    pass_hash = sha256_crypt.hash(password)
    cur.execute(sign_up_pers1.insert(pass_hash))
    conn.commit()
    cur.close()
    conn.close()
except mariadb.Error as error:
    print("Failed to read data from table", error)
finally:
    if conn:
        conn.close()
    print('Connection to db was closed!')
return redirect("/")

```

Această funcție înregistrează un user, astfel preia toate datele din input-ul html, le verifică, criptează parola cu ajutorul lui sha256 și abia apoi introduce datele în baza de date.

1.3.1. Funcția update_permissions()

```

def update_permissions(modify):
    update_query = ""
    if (modify.get('id_perm') and modify.get('name_perm')) or \
        (modify.get('id_perm') and modify.get('desc_perm')) or \
        (modify.get('id_perm') and modify.get('app')):
        update_query = "UPDATE permissions"
        if modify.get('name_perm') != '' and modify.get('desc_perm') != '' and
modify.get('app'):
            update_query += " SET name = '\"' + modify['name_perm'] + '\', description = '\"' \
                + modify['desc_perm'] + '\', app_id = '\"' + modify['app'] + '\"'"
        if modify.get('desc_perm') == '':
            update_query += " SET name = '\"' + modify['name_perm'] + '\"'"
        if modify.get('name_perm') == '':
            update_query += " SET description = '\"' + modify['desc_perm'] + '\"'"
        update_query += " WHERE id = " + modify['id_perm'] + ';'
    return update_query

```

Aceasta permite să facem update pe una dintre permisiuni, putând să modificăm pe baza de id, unul dintre câmpurile tabelului permissions, sau de asemenea putem să le modificăm pe toate fără să existe probleme.

1.4. SCURTĂ DESCRIERE

Proiectul nostru începe prin înregistrarea unui utilizator (sign-up), iar datele colectate sunt introduse în baza de date Mysql, aici criptând parola cu ajutorul lui sha256. După ce un user este înregistrat acesta poate să se conecteze pe orice aplicație pe care are dreptul, acesta putând să își vadă permisiunile și grupurile din care acesta face parte. Dacă utilizatorul dorește să intre pe altă aplicație trebuie să ceară acordul administratorului, care dispune de o interfață web în care acesta poate să modifice, să adauge, să șteargă diferite grupuri, permisiunii, sau aplicații.

1.5. STRUCTURA (ARHITECTURA) SISTEMULUI - STILUL MVC

Stilul MVC(Model-View-Controller)

Acesta este un model de proiectare software utilizat în mod obișnuit pentru dezvoltarea interfețelor utilizator care împarte logica programului aferent în trei elemente interconectate. Acest lucru se face pentru a separa reprezentările interne ale informațiilor de modurile în care informațiile sunt prezentate și acceptate de către utilizator.

Motivarea alegerii

Am ales acest stil arhitectural deoarece se pliază perfect pe necesitățile aplicației pe care o dezvoltăm, după cum e exemplificat mai jos:

Problema pe care o rezolvă

Interacțiunea dintre elementele care compun o aplicație poate fi greu de gestionat și manipulat pentru a crea un produs final stabil. Separarea acestora, cât și împărțirea în funcționalităților în metode specifice rezolvă multe din problemele ce pot apărea dacă informațiile nu sunt organizate într-un mod coerent.

Contextul

Prin separarea în module specializate pentru anumite funcții, codul devine mult mai ușor de înțeles, testat, depanat și este simplu să fie refolosit salvând foarte mult timp. Toate aceste lucruri pot ajuta și pe partea de aplicație web fiind ușor de folosit cât și implementat noi funcționalități.

Soluția

Cele trei elemente interconectate între ele ne ajută să modularizăm aplicația separând-o în mai multe componente specializate pe un singur aspect:

- *Model* - baza de date unde sunt stocate informațiile despre conturile existente, grupurile, permisiunile cât și despre relația dintre acestea.
- *View* - aplicația web accesibilă dintr-un browser care permite utilizatorului să

interacționeze într-un mod mai plăcut și simplificat cu codul care stă la baza programului.

- *Controller* - reprezintă partea de backend (codul de Python) care leagă restul componentelor și organizează informațiile conform situației date.

Variante

Acest stil arhitectural este foarte utilizat în special în aplicațiile web care au implementat un MVC Framework, dar poate fi folosit și pentru dezvoltarea aplicațiilor pentru mobil sau programelor pentru desktop. Responsabilitatea arhitecturii MVC este de a media interacțiunea dintre client și server.

1.6. FUNCȚIILE SISTEMULUI

1. Cerințele sistemului

- Stocarea datelor într-o bază de date;
- Actualizarea bazei de date prin intermediul aplicației web;
- Design responsive pentru ca aplicația să poată fi ușor folosită de pe mobil.

2. Cerințe specifice utilizatorului normal

- Vizualizarea listei proprii de aplicații și permisiunile asociate;

3. Cerințe specifice administratorului

- Afișarea listei de utilizatori și detaliile asociate;
- Afișarea listei de grupuri și detaliile asociate;
- Afișarea listei de permisiuni și detaliile asociate;
- Afișarea listei de aplicații și detaliile asociate;
- Crearea de conturi de utilizatori;
- Crearea de noi permisiuni;
- Crearea de noi grupuri;
- Crearea de noi aplicații;
- Ștergerea unui cont;
- Ștergerea unui grup;
- Ștergerea unei permisiuni;
- Ștergerea unei aplicații;
- Modificarea detaliilor unui cont de utilizator;
- Modificarea detaliilor unui grup;
- Modificarea detaliilor unei permisiuni;
- Modificarea detaliilor unei aplicații;

4. Cerințe comune între cele două tipuri de utilizatori

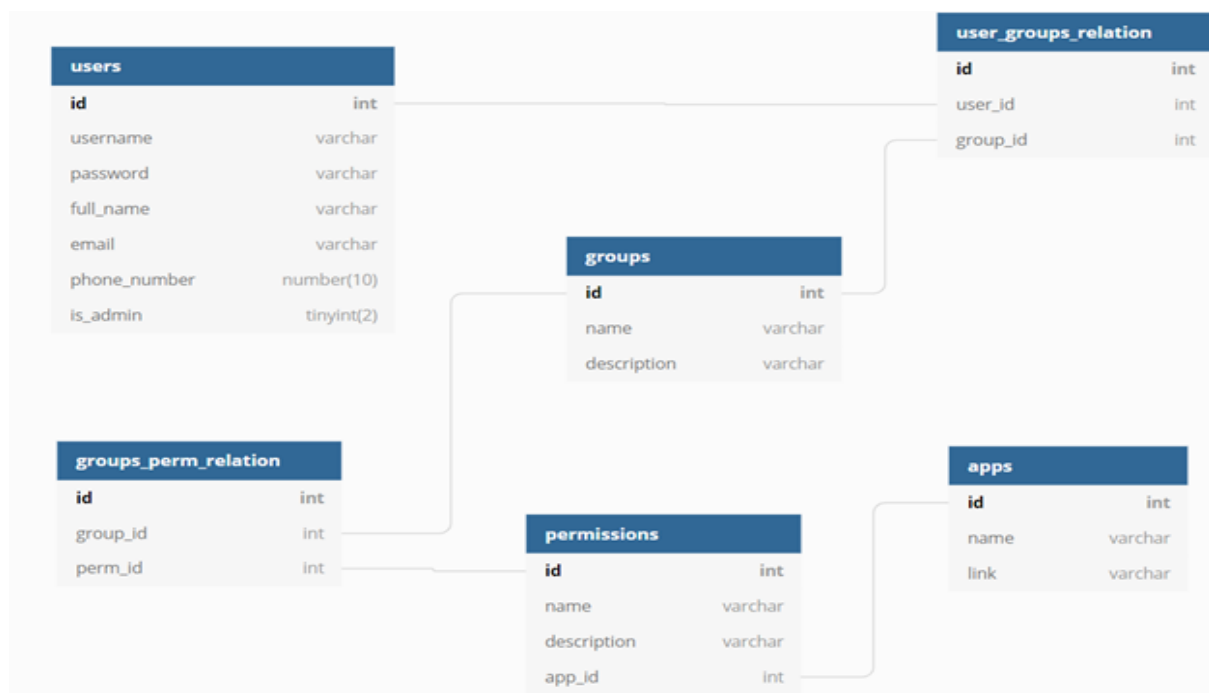
- Conectarea la cont;

- Crearea unui cont nou din prima pagină;
- Modificarea parolei și a altor detalii legate de contul personal;
- Vizualizarea tuturor grupurilor;
- Accesul la datele de contact ale persoanelor responsabile (administrator, serviciului de hosting etc.).

2. MIJLOACE DE VERIFICARE

Pentru configurarea și rularea eficientă a acestui program vom avea nevoie de următoarele:

1. Interpretor Python 3 pentru WSL(WSL- Windows Subsystem for Linux):
 - `sudo apt install python3 python3-pip ipython3`
2. Pachetul pip pentru a avea acces la toate librăriile din python (WSL- Windows Subsystem for Linux):
 - `sudo apt update && upgrade`
 - `sudo apt install python3-pip`
3. Librăriile pentru modulul flask și sha256_crypt:
 - `python -m pip install flask`
 - `from flask import Flask`
 - `from passlib.hash import sha256_crypt`
4. Baza de date adaptată pentru cerințele clientului:
 - putem observa diagrama noastră a bazei de date conform cerințelor clientului și codul necesar pentru crearea acesteia mai jos:



```

CREATE TABLE `users` (
  `id` int PRIMARY KEY AUTO_INCREMENT,
  `username` varchar(255),
  `password` varchar(255),
  `full_name` varchar(255),
  `email` varchar(255),
  `phone_number` number(10),
  `is_admin` tinyint(2)
);
CREATE TABLE `groups` (
  `id` int PRIMARY KEY AUTO_INCREMENT,
  `name` varchar(255),
  `description` varchar(255)
);
CREATE TABLE `permissions` (
  `id` int PRIMARY KEY AUTO_INCREMENT,
  `name` varchar(255),
  `description` varchar(255),
  `app_id` int
);
CREATE TABLE `apps` (
  `id` int PRIMARY KEY AUTO_INCREMENT,
  `name` varchar(255),
  `link` varchar(255)
);
CREATE TABLE `user_groups_relation` (
  `id` int PRIMARY KEY AUTO_INCREMENT,
  `user_id` int,
  `group_id` int
);
CREATE TABLE `groups_perm_relation` (
  `id` int PRIMARY KEY AUTO_INCREMENT,
  `group_id` int,
  `perm_id` int
);
ALTER TABLE `user_groups_relation` ADD FOREIGN KEY (`user_id`) REFERENCES `users` (`id`);
ALTER TABLE `groups_perm_relation` ADD FOREIGN KEY (`group_id`) REFERENCES `groups` (`id`);
ALTER TABLE `user_groups_relation` ADD FOREIGN KEY (`group_id`) REFERENCES `groups` (`id`);
ALTER TABLE `groups_perm_relation` ADD FOREIGN KEY (`perm_id`) REFERENCES `permissions` (`id`);
ALTER TABLE `permissions` ADD FOREIGN KEY (`app_id`) REFERENCES `apps` (`id`);

```

5. Rularea programului din terminal:

```
- python3 main.py
```

3. CONDIȚII DE FUNCȚIONARE

Aici, vor fi descrise, punct cu punct, toate funcționalitățile care vor fi testate.

3.1. TEST 1 - Implementarea modului într-o aplicație externă

Pentru a putea testa implementarea modului într-o aplicație externă am pus la dispoziție o mică aplicație de tip client-server care se folosește de biblioteca care conține anumite funcții specifice ce au rolul de a verifica în momentul conectării dacă utilizatorul are acces la aplicație și ce poate să facă.

În momentul conectării la această aplicație demo, dacă există un cont definit în baza de date pentru acel utilizator rezultatul așteptat este să îi returneze o listă cu ce permisiuni are sau un mesaj de eroare dacă pe parcurs întâmpină vreo problemă.

3.2. TEST 2 - Actualizarea datelor utilizatorului curent

Pentru această funcționalitate, utilizatorul fiind conectat va putea accesa pagina de Settings. Îi vor fi afișate datele curente ale profilului și câmpuri text pentru schimbarea acestor date. După ce câmpurile care trebuie schimbate au fost completate și s-a apăsat butonul UPDATE, pagina se va încărca cu noile date schimbate.

3.3. TEST 3 - Înregistrarea unui utilizator

Pentru această funcționalitate, utilizatorul nefiind încă înregistrat, va putea să își creeze cont fără mari probleme, dacă acesta respectă cerințele din field-urile pe care acesta trebuie să le completeze pentru a se înregistra un cont.

4. CONDIȚII DE VERIFICARE

4.1. TEST 1 - Implementarea modului într-o aplicație externă

Pentru a putea verifica această funcționalitate, vom avea nevoie de un set de date anume care va trebui inserat în baza de date cu interogările de mai jos:

```
---- Create the app
insert into apps(id, name, link) values(3,"Small server", "127.0.0.1:1337");

---- Create the permissions
insert into permissions(id, name, description, app_id) values(4, "CREATE data", "Permission to create new data", 3);
insert into permissions(id, name, description, app_id) values(5, "READ data", "Permission to read data", 3);
insert into permissions(id, name, description, app_id) values(6, "UPDATE data", "Permission to update data", 3);
insert into permissions(id, name, description, app_id) values(7, "DELETE data", "Permission to delete data", 3);

---- Create the groups
insert into groups(id, name, description) values(3, "Owners of Small servers", "Users are the owners of the Small server");
insert into groups(id, name, description) values(4, "Editors of Small servers", "Users are the editors of the Small server");
insert into groups(id, name, description) values(5, "Viewers of Small servers", "Users can only view the Small server");

---- Link the permissions to the groups
```

```

INSERT INTO groups_perm_relation(group_id, perm_id) values(3,4);
INSERT INTO groups_perm_relation(group_id, perm_id) values(3,5);
INSERT INTO groups_perm_relation(group_id, perm_id) values(3,6);
INSERT INTO groups_perm_relation(group_id, perm_id) values(3,7);
INSERT INTO groups_perm_relation(group_id, perm_id) values(4,5);
INSERT INTO groups_perm_relation(group_id, perm_id) values(4,6);
INSERT INTO groups_perm_relation(group_id, perm_id) values(5,5);

---- Create the users

INSERT INTO users VALUES
(8,'jdoe','$5$rounds=535000$pIIZjh00BbM9jvGs$h3aeOUR2H6m.hDzJzlSolSc5uKrFA9EsKN0Nd9.1u/4','John
Doe','jdoe@yahoo.com','0747553214',NULL),(9,'jadoe','$5$rounds=535000$ifC3P50afQtJ77dC$r0KO
b7J786FAxqxyVzrEMiAPK6tmOyYWJjBKCV/TjE4','Jane
Doe','jadoe@yahoo.com','0758314679',NULL),(10,'asmith','$5$rounds=535000$.lGpw41WUTC8igSe$L
ypTB6Yu1EdM4SNffFu0ST/nIS22Iez/ogldzlCagKB','Adam
Smith','asmith@yahoo.com','0734658214',NULL);

---- Link the users to the groups

insert into user_groups_relation(user_id, group_id) values(8, 3);
insert into user_groups_relation(user_id, group_id) values(9, 4);
insert into user_groups_relation(user_id, group_id) values(10, 5);

```

După inserare, vom avea 3 utilizatori pe care îi putem folosi:

- jdoe cu parola 123;
- jadoe cu parola 123;
- asmith cu parola 123;

Pentru a porni serverul va trebui să ne aflăm în folderul *demo_app* din cadrul proiectului și se va rula comanda:

- python3 server.py

Pentru a porni clientul va trebui să ne aflăm în folderul *demo_app* din cadrul proiectului și se va rula comanda:

- python3 client.py

În momentul conectării clientului la server, serverul va cere numele de utilizator sau adresa de email și parola pentru autentificare. Clientul le va transmite la server, iar acesta va verifica dacă este un utilizator valid și datele de conectare corespund. Dacă conectarea a fost făcută cu succes, serverul va returna toate permisiunile aplicabile pentru client dacă există sau un mesaj de eroare dacă întâmpină vreo problemă.

4.2. TEST 2 - Actualizarea datelor utilizatorului curent

În acest caz, pentru accesarea meniului de setări, utilizatorul va selecta butonul de Profile din iar mai apoi se va deschide un meniu dropdown. De acolo va alege opțiunea Settings.

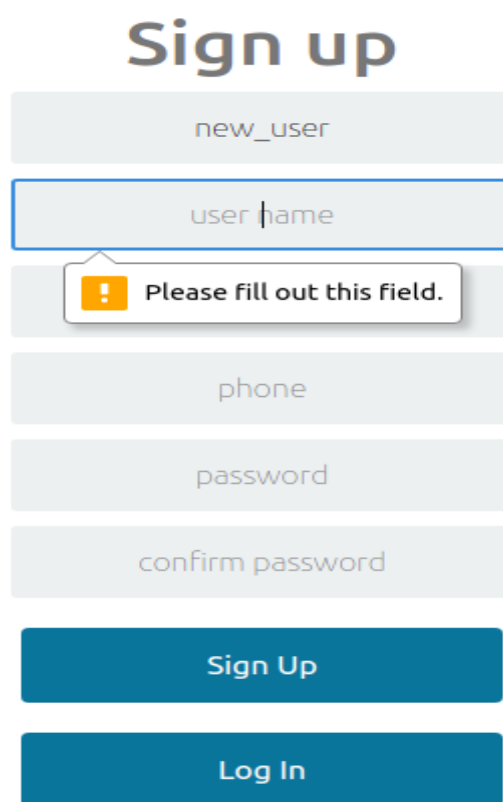
După ce pagina de Settings a fost încărcată, utilizatorul va putea observa în stânga detaliile personale ale contului, iar în dreapta câmpurile text aferente schimbării detaliilor personale (ex: parola, email, username).

Utilizatorul va completa câmpurile necesare schimbării și va apăsa butonul UPDATE. Pagina se va reîncărcă automat și se vor observa noile detalii ale contului schimbate.

4.3. TEST 3 - Înregistrarea unui utilizator

În acest caz, pentru a înregistra un nou utilizator nefiind încă înregistrat, din prima pagină acesta va putea să acceseze butonul de “sign up” pentru a-și crea un cont. Toate câmpurile din formular trebuie completate pentru a putea să înregistrăm un nou utilizator, iar de asemenea username-ul trebuie să fie diferit față de ce există deja în baza de date.

User-ul va trebui să completeze toate câmpurile, deoarece sunt obligatorii, altfel va fii imposibilă înregistrarea.



The image shows a 'Sign up' form with the following elements:

- Title: **Sign up**
- Form fields (all with light gray placeholder text):
 - new_user
 - user name (highlighted with a blue border)
 - phone
 - password
 - confirm password
- Buttons: **Sign Up** and **Log In** (both in blue)
- Validation message: A yellow warning icon and the text "Please fill out this field." is displayed over the 'user name' field.

Doar după ce utilizatorul scrie corect și parola și partea de confirmare a parolei, aceasta se va insera criptată în baza de date, altfel acesta va primi un mesaj tip flash de eroare.

```
if not (sign_up_pers1.check_pass(pass_conf) and sign_up_pers1.check_email()):  
    flash('Please check your sign up details and try again.')  
    return redirect("/sign_up")
```

Sign Up

Log In

Please check your sign
up details and try
again.

