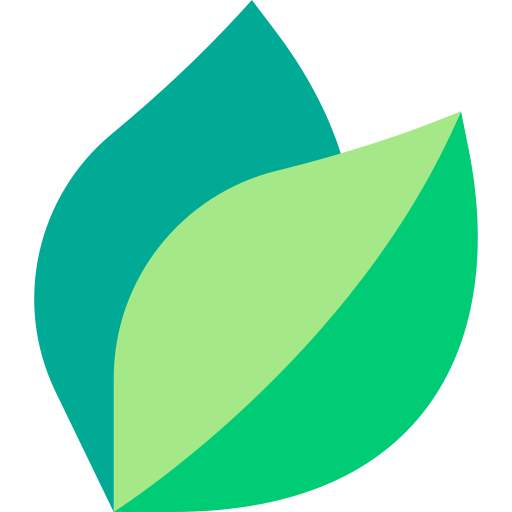
******

***Proiect I.S.W.***

TeaWeb Application



***Manager*:** Olteanu Eduard Daniel

***Development*:** Momoi Alexandru, Olteanu Eduard Daniel

***Unit Testing*:** Momoi Alexandru

***Front-end Development / U.I.*:** Gheorge Luca-Teodor

***Documentație*:** Gheorghe Luca-Teodor,Teleki Ferenc-Tibor

***Database*:** Teleki Ferenc-Tibor, Olteanu Eduard Daniel

***Facultatea de Inginerie Electrică și Știința Calculatoarelor***

*Calculatoare Anul IV*

**Cuprins**

[1. Introducere 3](file:///C:\Users\Tibi\Downloads\Documentatie_AstroSwipe.docx#_Toc93085278)

[2. Front-end Mocks 3](file:///C:\Users\Tibi\Downloads\Documentatie_AstroSwipe.docx#_Toc93085279)

[Landing Page 3](file:///C:\Users\Tibi\Downloads\Documentatie_AstroSwipe.docx#_Toc93085280)

[Login Page 3](file:///C:\Users\Tibi\Downloads\Documentatie_AstroSwipe.docx#_Toc93085281)

Feedback Page ......................................................................................................................................4

3. Biblioteci folosite ......................................................................................................................................5

4. Arhitectura Aplicației 6

6. Testare ................................................................................................................................................12

7. Securitate ...........................................................................................................................................12

8. Development ......................................................................................................................................13

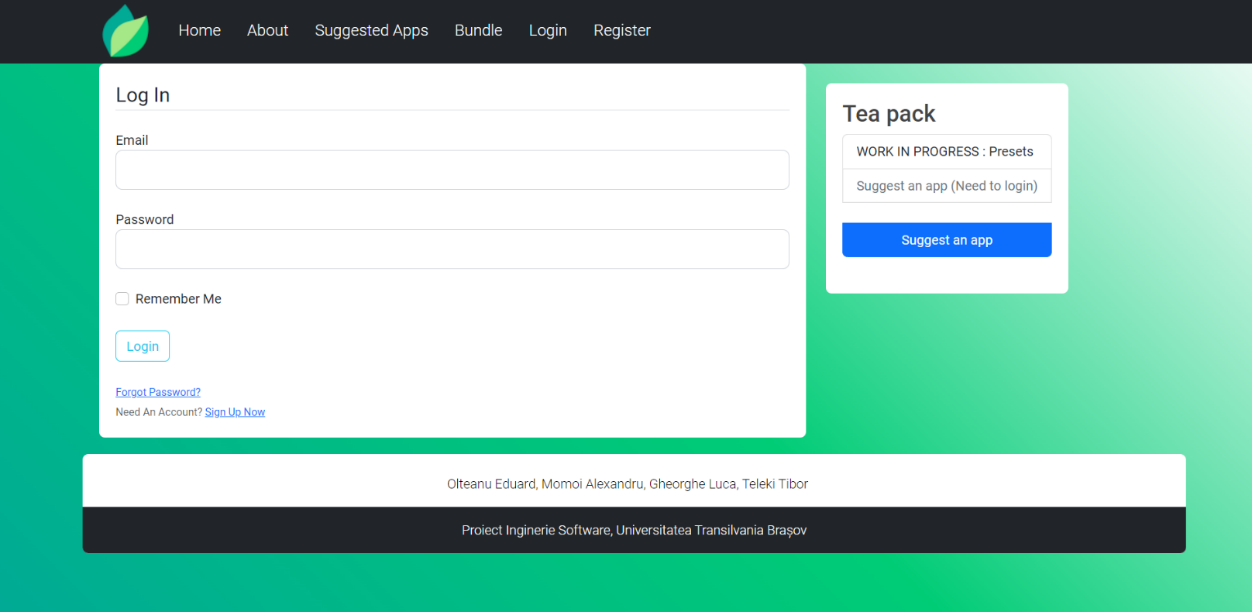
8. Bibliografie..........................................................................................................................................15

# **1.Introducere:**

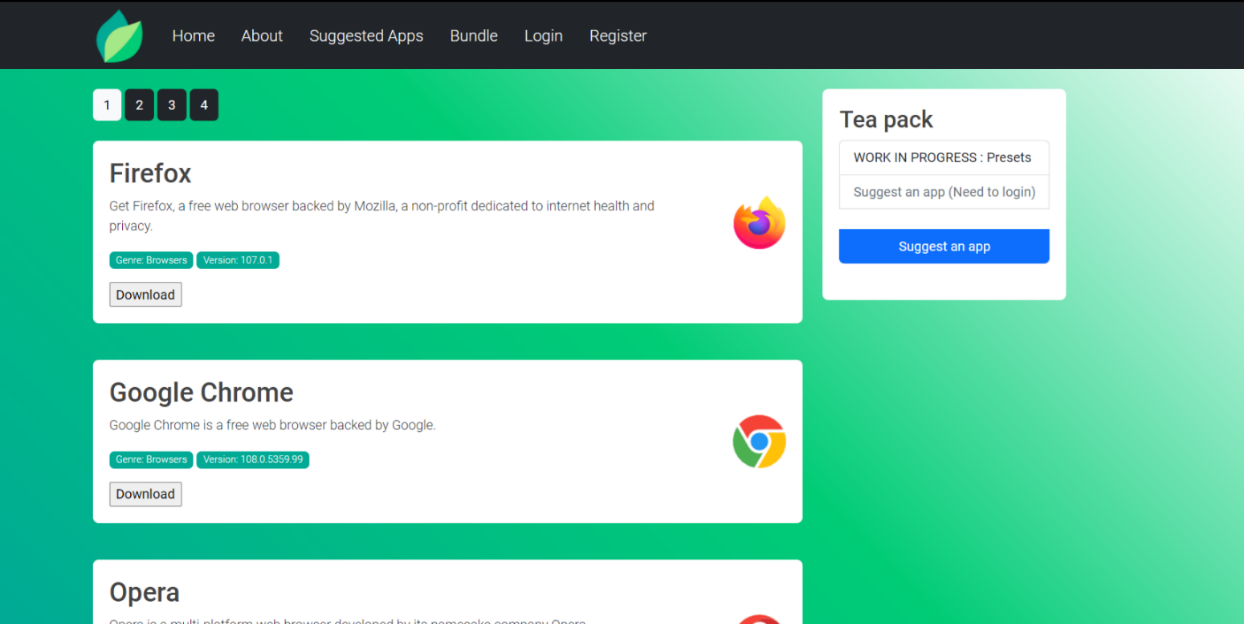
TeaWeb este o aplicație web care folosește Flask API pentru a oferi utilizatorilor un AppStore unde își pot regăsi aplicațiile preferate. Aceasta îi oferă clientului posibilitatea de a descărca și instala mai multe aplicații individual, dar poate opta și pentru bundle-uri prestabilite.

1. **Front-end mocks:**

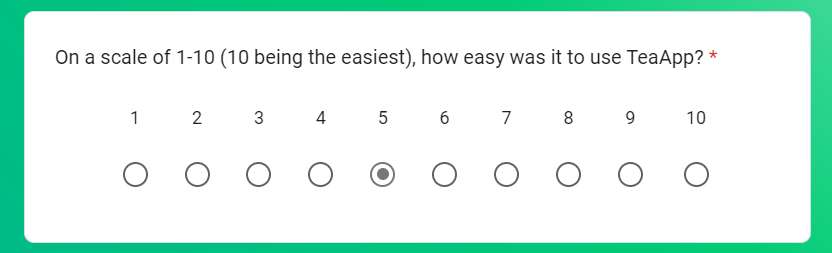
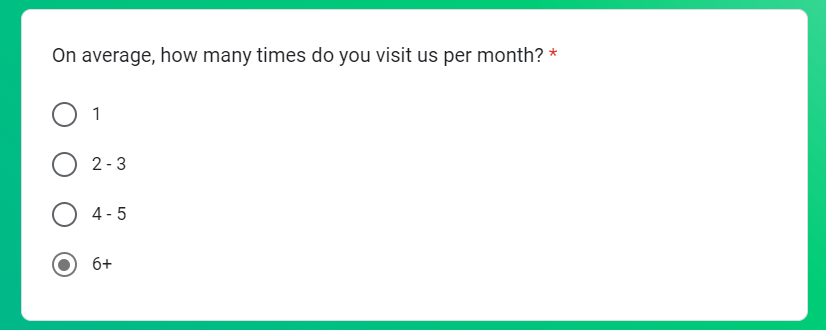
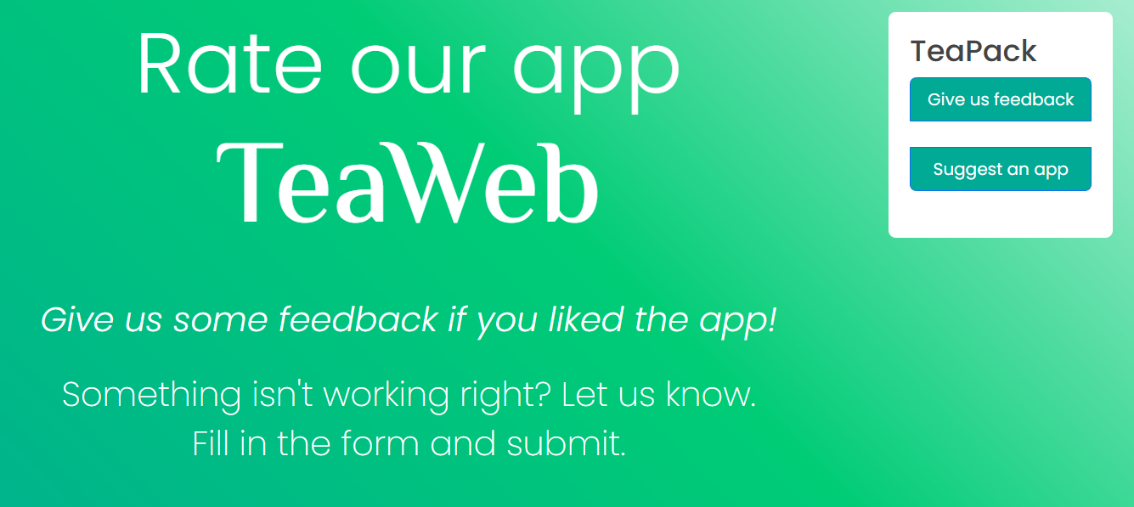
* **Login Page:** Ne-am dorit să oferim utilizatorilor o interfață simplă, ce poate fi utilizată cu ușurință, pentru a se înregistra și loga.

****

* **Landing Page:** Obiectivul principal a fost contruirea unui design simplu și intuitiv, care le poate oferi utilizatorilor o experiență simplă în a descărca aplicațiile dorite.



* **Feedback Page:** Am dorit să le oferit utilizatorilor posibilitatea de a ne ajută să îmbunătățim platforma. Am creat un mail al proiectului pentru a primi răspunsurile de la chestionare și pentru a citi feedback-ul primit.

****

# 

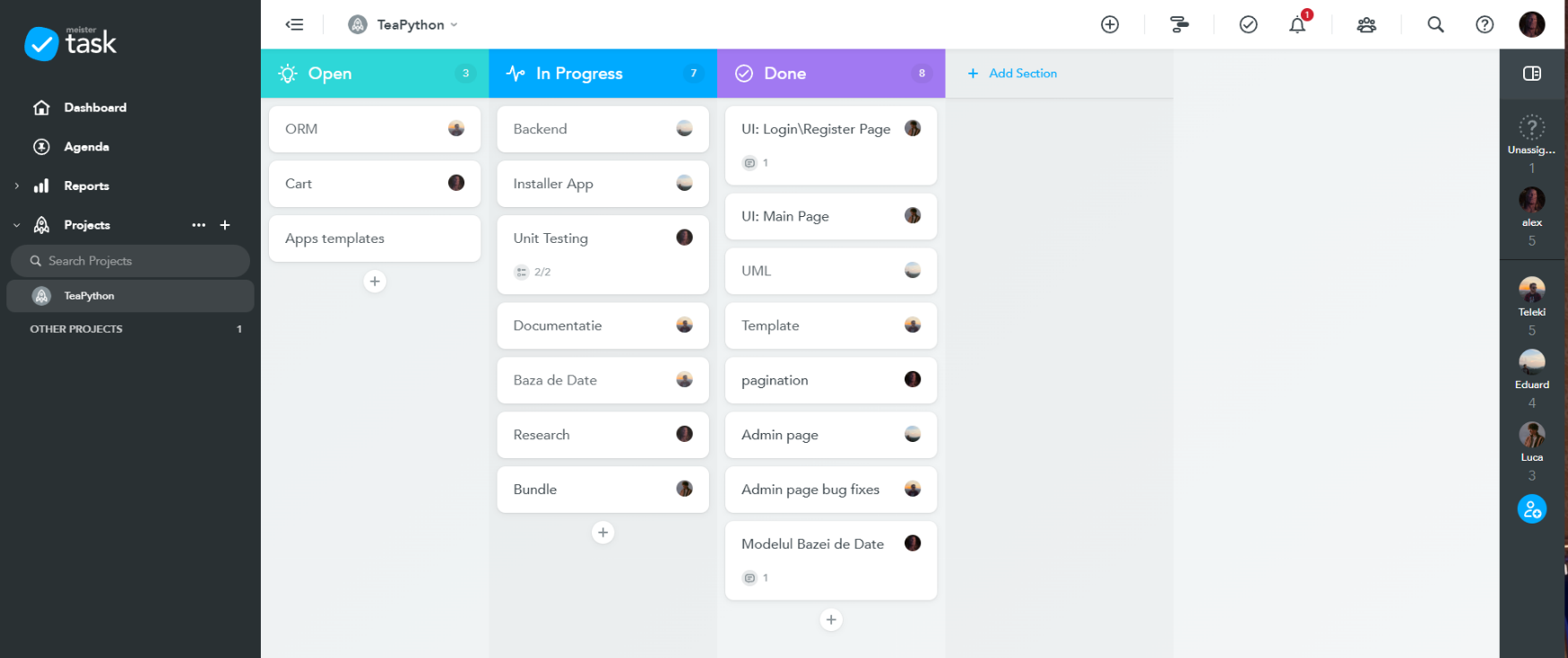
# **2.Biblioteci folosite**

* Bcrypt-4.0.1 - Soluție pentru criptarea parolelor și a informațiilor
* Click-8.1.3 – Librărie în Python pentru generarea automată de pagini
* Dnspython-2.2.1 – Folosită pentru actualizări dinamice
* email-validator-1.3.0 – Librării pentru validarea mail-ului
* Flask-2.2.2 – Utilizată pentru dezvoltarea aplicațiilor web
* Flask-Bcrypt-1.0.1 – Extensie Flask
* Flask-Login-0.6.2 – Folositor pentru managementul user-ilor aplicației
* Flask-SQLAlchemy-3.0.2 – Suport pentru SQL Alchemy
* Flask-WTF-1.0.1 – Include încărcarea de fișiere
* greenlet-2.0.1
* idna-3.4
* importlib-metadata-5.0.0
* itsdangerous-2.1.2
* Jinja2-3.1.2
* MarkupSafe-2.1.1
* SQLAlchemy-1.4.44
* Werkzeug-2.2.2
* WTForms-3.0.1

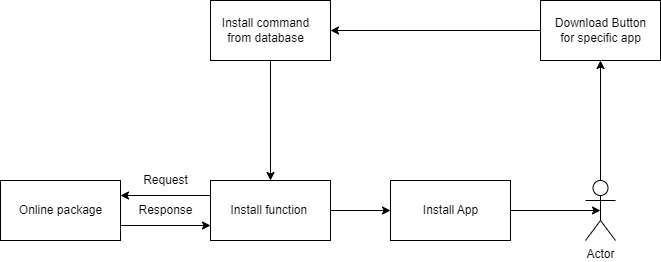
# 

# **3.Arhitectura Aplicației**

* **Management-ul Echipei**

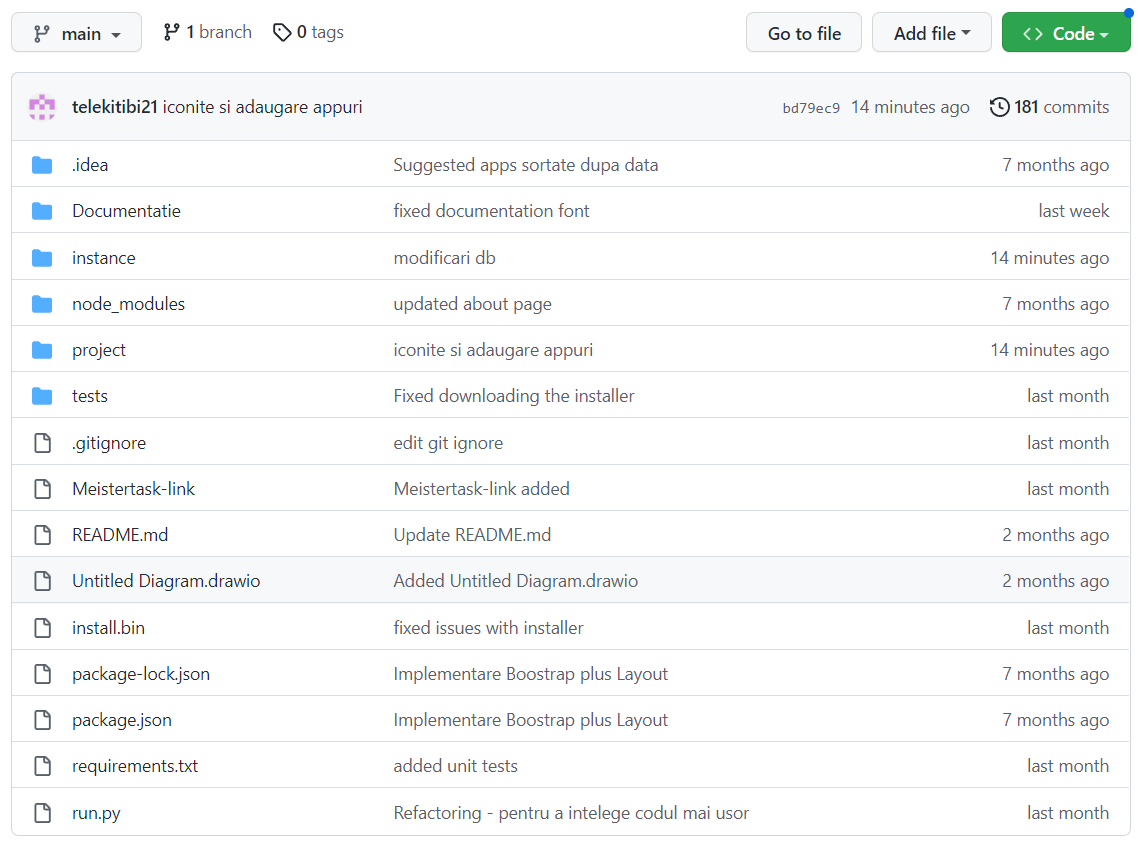
Pentru management-ul echipei am utilizat ca și board pentru task-uri, aplicația ***meistertask*****.**

***Sursa:*** *1*

* **Diagramă Installer**

* **Versionarea Codului**

Pentru versionarea codului am utilizat GitHub, mediu în care am creat branch-uri de dezvoltare pentru diferite etape ale proiectului.



***Sursa:*** *2*

# 

# **4. Management**

## Etape Realizare:

* 1. Realizarea diagramelor U.M.L.
  2. Realizare mock-urilor pentru partea de front-end
  3. Implementarea segmentelor de bază (atât front-end cât si backend):
     + Sign-Up & Login;
     + Posibilitatea de a putea crea sugestii;
     + Editare cont;
     + Editare sugestii;
     + Installere.

## Cerințe Funcționalitate:

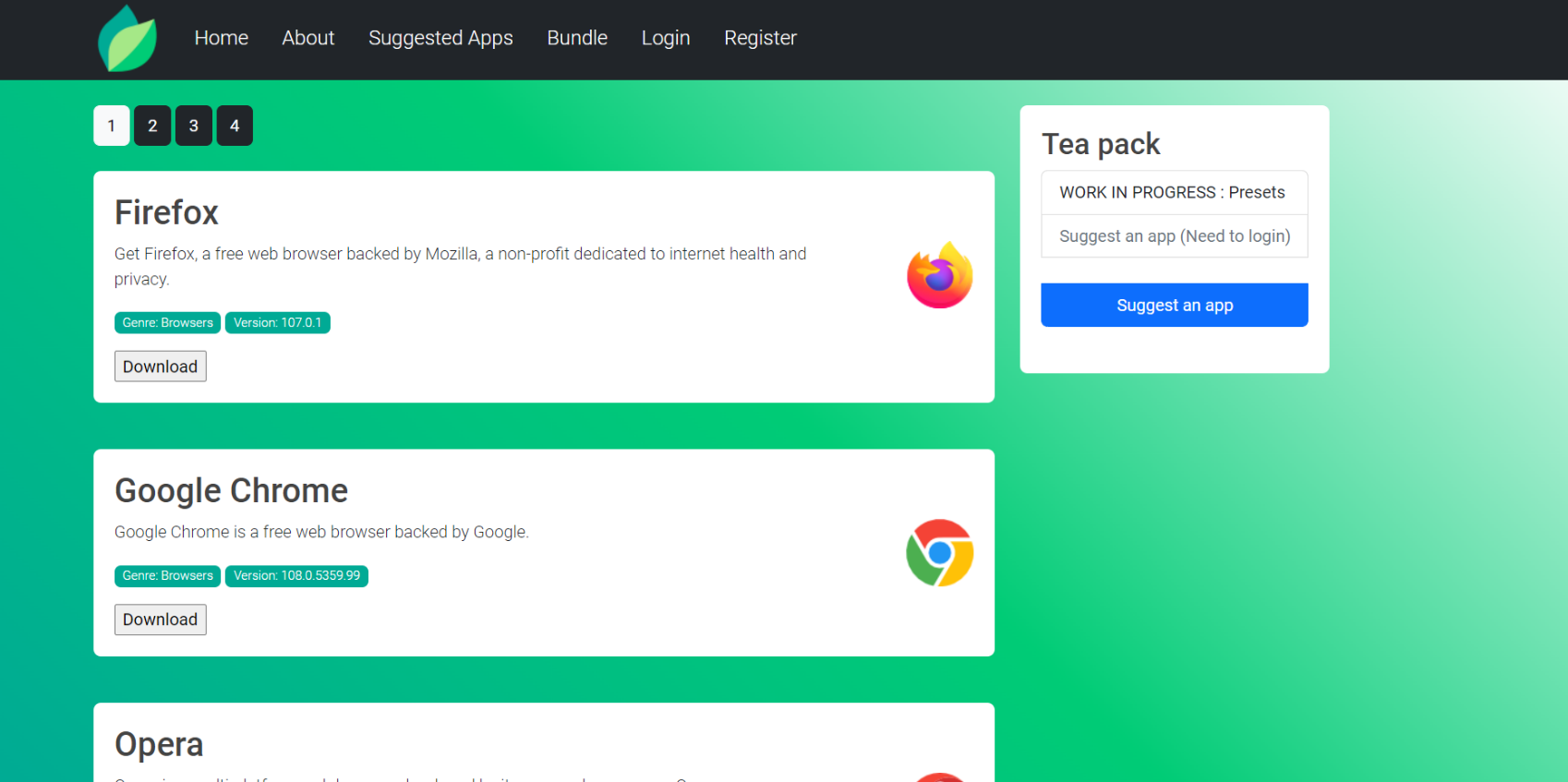
* + 1. Front-end:
       - Main Page: Login/Sign-up;
       - Home Page;
       - Aplicații Disponibile;
       - Profil.
    2. Back-end:
       - Salvare date utilizatori;
       - Crearea intaller-ului ***dorit***;
       - Selectarea aplicațiilor;
       - Descărcarea aplicațiilor.

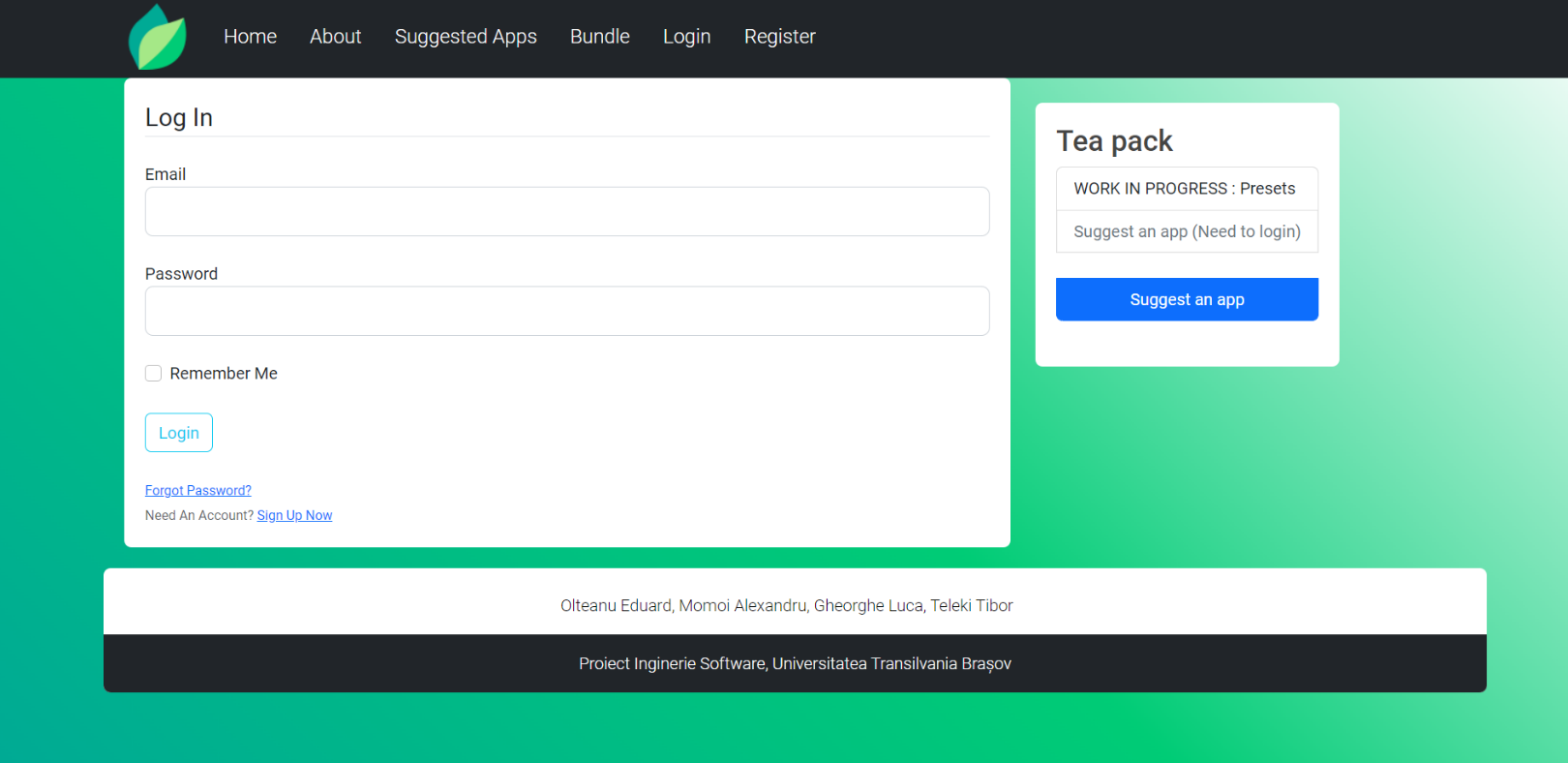
## Cerințe Non-funcționale:

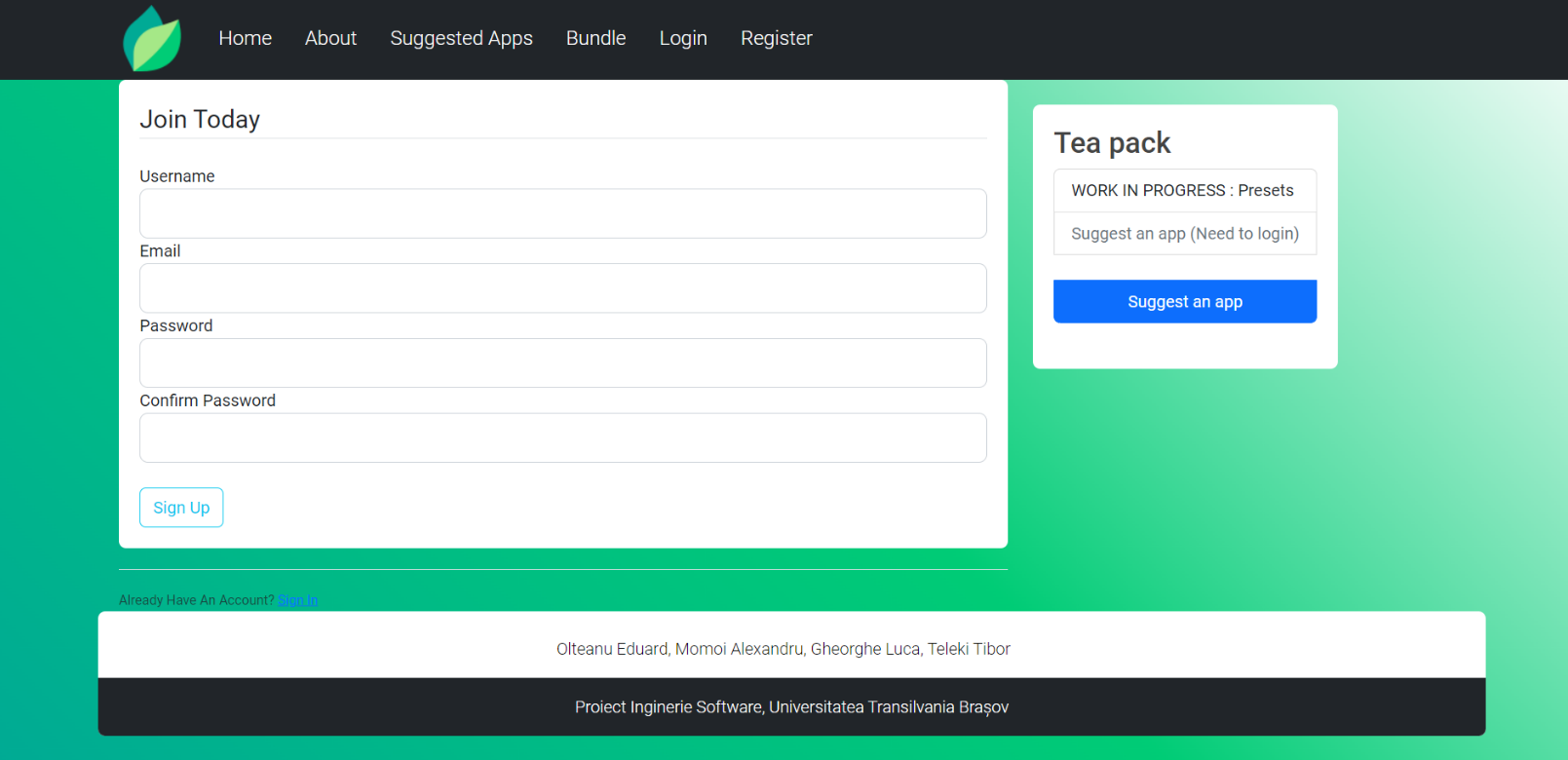
1. Front-end:
   * + - HTML;
       - CSS;
       - Python.
2. Back-end:
   * + - Python.
3. Database:
   * + - SqLite.

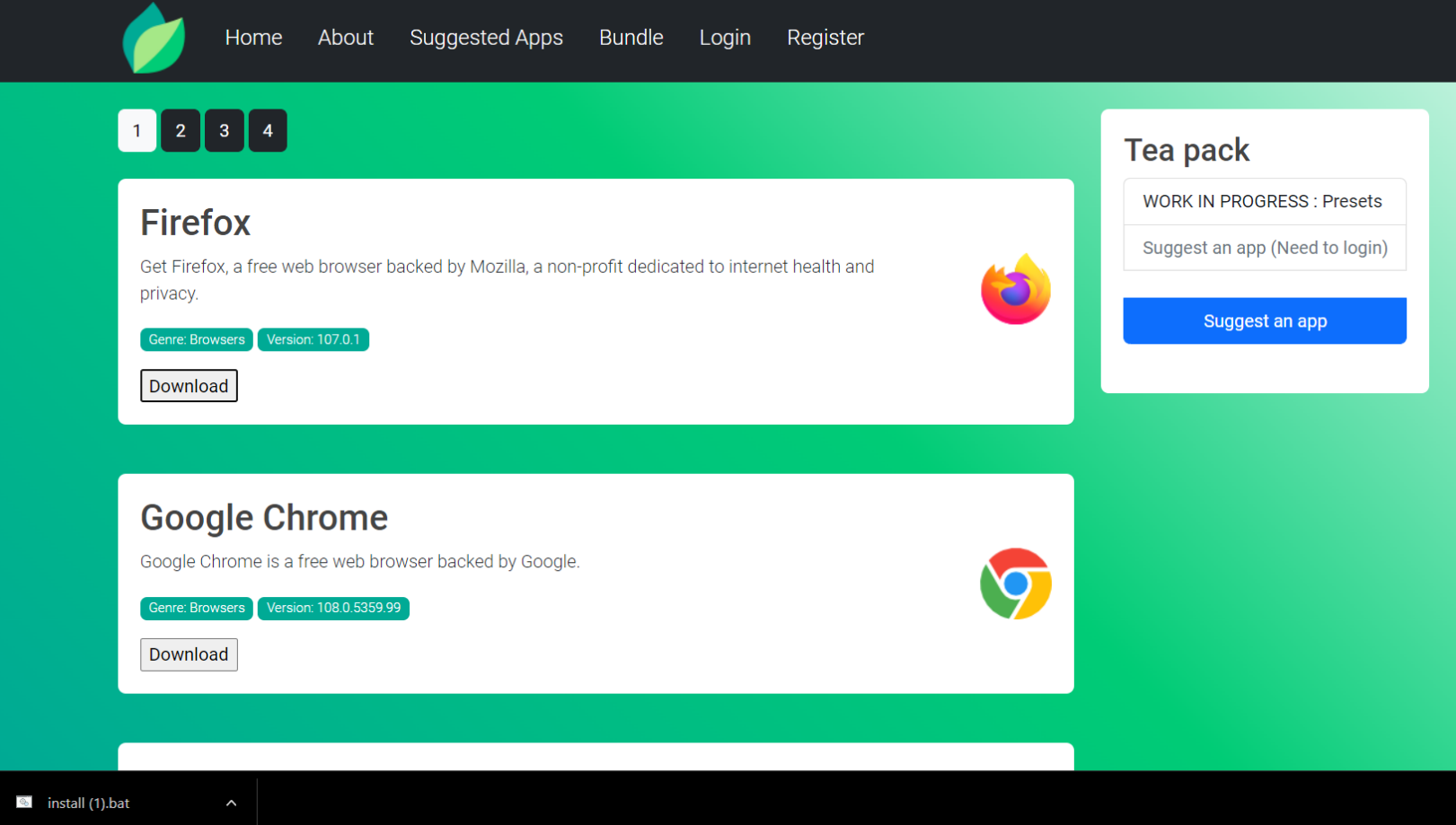
# **5.Tutorial**

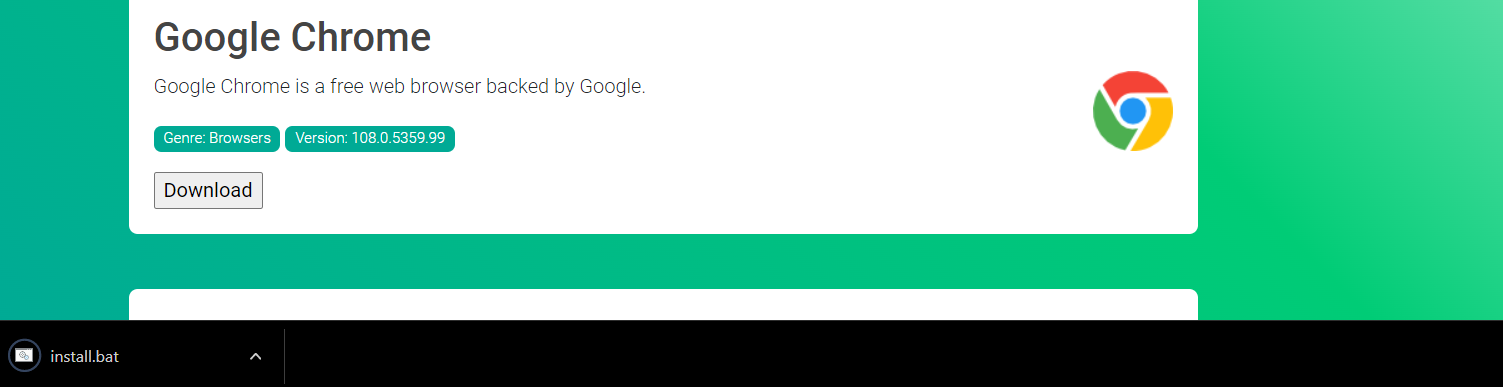
# 5.1Utilizare

**Pasul 1:** Se deschide pagina web.

**Pasul 2:** În cazul în care ai cont, apeși pe butonul ‘*Login*’.

**Pasul 3:** În cazul în care nu există cont apasă pe butonul ‘*Register*’.

**Pasul 4:** Se alege aplicația dorită, se apasă pe download și va apărea fişierul ***.bat***.

**Pasul 5:** Se descarcă aplicația dorită.

# 

# 5.2. Tutorial aplicaţie (Configurare)

**Pasul 1:** Instalare folosind comanda ***pip.***

curl <https://bootstrap.pypa.io/get-pip.py> -o get-pip.py

python3 get-pip.py

sudo apt install python3.10-venv

**Pasul 2:** Se creează un "virtual environment” în folder-ul proiectului rulând comanda:

python3 -m venv teaenv

**Pasul 3:** Se activează “virtual environment-ul” cu pricina rulând comanda:

Source teaenv/bin/activate

**Pasul 4:** Se instalează cerințele:

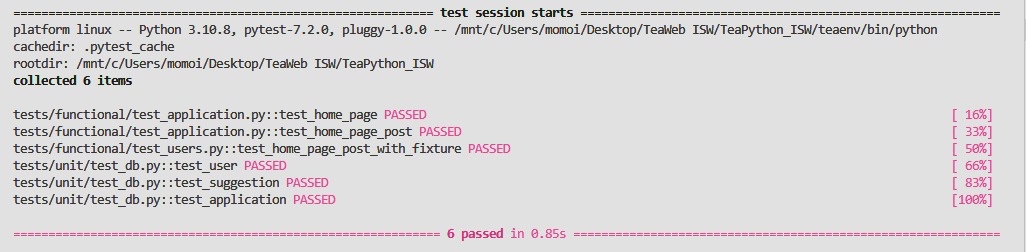
pip install -r requirements.txt

**Pasul 5:** Se rulează aplicația:

python3 run.py

\*Pentru rularea comenzilor de mai sus se va folosi Command Prompt

# **6. Testare**



Pentru testare, am folosit pytest. Pytest este un testing framework Python. Poate fi folosit pentru a scrie diferite tipuri de teste software, inclusiv teste unitare, teste de integrare, teste end-to-end și teste funcționale.

Framework-ul pytest, este o unealtă pentru scrierea, organizarea și executarea testelor în limbajul de programare Python. Pytest oferă flexibilitate în ceea ce privește modul în care testele pot fi executate și îndeplinește aspectele cheie ale unui mediu de testare excelent: testele sunt plăcute de scris, se pot scrie rapid folosind funcții de ajutor (fixtures), pot fi executate cu o singură comandă și rulează rapid.

Am grupat testele în două foldere, “unit” și “functional “. Pytest permite rularea testelor, căutând numai fișiere din directoriu ce încep cu numele de “test”.

Pentru a rula un test, deschidem o fereastră nouă de terminal, după care rulăm comanda python -m pytest -v. Pentru a rula doar anumite teste, spre exemplu unit tests, rulăm comanda python -m pytest tests/unit -v.

# **7. Securitate**

Flask este un framework web open-source pentru Python care oferă o serie de caracteristici pentru a ajuta la construirea de aplicații web sigure. Flask include mecanisme pentru a preveni atacurile comune precum cross-site scripting (XSS), cross-site request forgery (CSRF) și SQL injection.

Flask de asemenea ofera un sistem de gestionare a sesiunilor bazat pe cookie-uri pentru a asigura autentificarea utilizatorilor. Flask poate fi configurat pentru a utiliza HTTPS, pentru a asigura criptarea traficului web.

Flask nu include însă o soluție integrată pentru autorizare, aceasta trebuie implementată separat. In general, Flask oferă un set solid de caracteristici pentru a ajuta la construirea de aplicații web sigure, dar dezvoltatorii trebuie să fie conștienți de riscurile de securitate și să implementeze best-practice-uri pentru a se asigura că aplicația este protejată în mod adecvat.

Un exemplu de securitate implement de noi:

application = Application.query.get\_or\_404(id\_application)

if current\_user.role != 'admin':

abort(403)

Această linie de cod face parte dintr-o funcție care ar putea fi utilizată pentru a obține sau actualiza informații despre o aplicație specifică din baza de date.

Prima linie de cod, "application = Application.query.get\_or\_404(id\_application)", caută în baza de date pentru o aplicație cu un ID specific. Dacă aplicația nu este găsită, se va returna o eroare HTTP 404 (nu a fost găsită).

A doua linie de cod, "if current\_user.role != 'admin':", verifică rolul utilizatorului curent. Dacă rolul nu este "admin", se execută următoarea linie de cod "abort(403)" care va returna o eroare HTTP 403 (interzis).

Acest cod ajută la securitate prin restricționarea accesului la informațiile despre aplicație doar pentru utilizatorii cu rolul "admin". Astfel se asigură că doar utilizatorii autorizați au acces la această informație și se previne accesul neautorizat de către alți utilizatori.

# **8. Development**

Dezvoltarea aplicaţiei a urmat arhitectura MVC- Model, View, Controller.

Dupa cum se poate observa în cadrul diagramei de mai sus, utilizatorul cere date serverului prin intermediul Controller-ului, acesta preia datele printr-un Model, adică un format al datelor potrivit pentru acţiunea cerută, şi le trimite în partea de View, unde sunt trimise înapoi către utilizator, în cazul nostru pe o pagina web, unde sunt afişate corespunzator. De exemplu, în aplicaţia noastră, returnăm aplicații în care introducem diferite date despre ele.

Mai mult decât atât, în cadrul dezvoltării aplicaţiei, am ales să utilizăm o bază de date online deoarece doream ca aplicaţia sa fie portabilă şi să nu existe problemă asupra existenţei datelor şi a incompatibilităţii programelor sau a versiunii acestora în cadrul fiecărui membru al echipei.

# **8.1. Baza de date - ORM**

Am implementat mai multe clase care se mapează la tabelele bazei de date utilizând ORM-ul SQLAlchemy.

Clasa User este un model pentru un utilizator și moşteneste de la db.Model si UserMixin. Are mai multe câmpuri precum id\_user ,username, email, password și un câmp role. Fiecare câmp este definit ca o coloană cu un tip de date specific și are anumite constrângeri precum unique, nullable și primary\_key. Câmpul suggestion este, de asemenea, definit ca o relație cu modelul Suggestion, care este definit mai târziu în cod.

Clasa Application este un model pentru o aplicație și, de asemenea, moşteneşte de la db.Model. Are mai multe câmpuri precum id\_application,name, description, genre, install\_command, image\_file și version.

Clasa Suggestion este un model pentru o sugestie, și are câmpuri precum id\_suggestion,name, description, genre, install\_command, image\_file și date\_sugested. De asemenea, are o cheie străină id\_user care se referă la modelul User.

Funcția load\_user decorată cu @login\_manager.user\_loader este o funcție de reîncărcare utilizată de Flask-Login pentru a reîncărca obiectul utilizator din ID-ul utilizatorului stocat în sesiune.

Funcția init\_db() creează tabelele bazei de date pentru toate modelele definite în cod prin apelarea db.create\_all().

# **9.** **Bibliografie**

[2] <https://www.meistertask.com/app/project/1FzFAJpz/teascript>

[3] <https://github.com/Eduard2609/TeaPython_ISW>