Cofinanțat de Uniunea Europeană

Proiect cofinanțat prin Programul Educație și Ocupare 2021-2027

Activitatea A3.Susținerea angajabilității absolvenților de învățământ terțiar prin realizarea de analize în vederea anticipării unor nevoi orizontale de dezvoltare a ofertei educaționale (L13-L24)

Sub-activitatea A3.1.Elaborarea de analize în vederea anticipării unor nevoi orizontale de dezvoltare a ofertei educaționale, pe domeniile de studii/specializările vizate de proiect (L13-L24)

Metodologie de investigare a oportunităților de angajare pe piața muncii locală și regională pentru absolvenții din domeniile de studii vizate

Autori:
Larisa Stănilă – Coordonator elaborare analize de piață / CEAP
Silvana Crivoi – Expert elaborare analize de piață / EEAP

CUPRINS

Capitolul 1 Context, rolul analizei și cadrul metodologic 3

1.1. Contextul, rolul analizei și domeniul de aplicare 3

1.2. Metodologia de investigare a oportunităților de angajare 3

Capitolul 2. Perspectivă generală asupra anunțurilor de recrutare identificate 8

2.1. Constituirea bazei finale și delimitarea universurilor de analiză 9

2.2. Caracterizarea descriptivă a oportunităților de angajare identificate 10

Anexe 15

Capitolul 1
Context, rolul analizei și cadrul metodologic

1.1. Contextul, rolul analizei și domeniul de aplicare

Prezenta metodologie este elaborată în cadrul activității A3, „Susținerea angajabilității absolvenților de învățământ terțiar prin realizarea de analize în vederea anticipării unor nevoi orizontale de dezvoltare a ofertei educaționale", și al **subactivității A3.1, „Elaborarea de analize în vederea anticipării unor nevoi orizontale de dezvoltare a ofertei educaționale, pe domeniile de studii/specializările vizate de proiect"**, din **proiectul SMART – Studenți motivați să ajungă antreprenori de succes (cod SMIS 311669)**.

Rolul metodologiei este de a asigura un cadru unitar și reproductibil pentru investigarea oportunităților de angajare și pentru constituirea bazei textuale necesare analizei competențelor solicitate pe piața muncii. Informațiile colectate permit identificarea cerințelor formulate de angajatori și raportarea acestora la profilurile de formare ale programelor de studii analizate.

Domeniul de aplicare cuprinde șapte specializări: Inginerie mecanică, Ingineria materialelor, Inginerie și management, Agronomie, Ingineria mediului, Ingineria produselor alimentare și Inginerie chimică. Investigația a avut în vedere anunțuri compatibile cu pregătirea universitară și cu nivelul de calificare al absolvenților de studii superioare, identificate în cadrul etapelor succesive de căutare desfășurate în perioada mai–iulie 2026.

Rezultatele au caracter exploratoriu, nu sunt exhaustive și nu au reprezentativitate statistică, din următoarele motive:

(1) nu toate angajările sunt precedate de anunțuri pe platformele online, unele angajări realizându-se prin recomandări sau prin contactarea directă a potențialilor candidați;

(2) intervalul de colectare a anunțurilor oferă informații doar cu privire la acest interval de timp, rezultatele neputând fi extrapolate pentru alte perioade ale anului.

Din acest motiv, se recomandă implementarea unui mecanism de aplicare continuă a metodologiilor „Metodologia de investigare a oportunităților de angajare pe piața muncii locală și regională pentru absolvenții din domeniile de studii vizate" și „Metodologia de explorare a competențelor specifice vizate de angajatorii de pe piața locală și regională". Aceste anunțuri reprezintă baza textuală pentru implementarea metodologiei de explorare a competențelor specifice vizate de angajatorii de pe piața locală și regională.

1.2. Metodologia de investigare a oportunităților de angajare

Procesul metodologic sintetizat în Figura 1 a fost organizat ca un flux în șapte pași succesivi. Fiecare pas utilizează rezultatele celui anterior, de la selectarea și testarea

platformelor și definirea criteriilor de căutare până la constituirea unei baze de date validate și încadrate pe specializări. Software-ul Python 3.14.6 reprezintă instrumentul central al fluxului, fiind utilizat pentru testarea automată a platformelor, colectarea, extragerea și structurarea datelor, precum și pentru sprijinirea identificării anunțurilor eligibile.

Pasul 1 **Selectarea și testarea platformelor de recrutare**. Într-o primă etapă au fost identificate platformele relevante pentru identificarea oportunităților de angajare: eJobs, Hipo.ro, BestJobs și ANOFM. Platformele au fost testate atât prin căutări manuale, pentru verificarea relevanței și a structurii rezultatelor, cât și prin căutări automate realizate cu scripturi dezvoltate în Python 3.14.6, pentru verificarea accesibilității paginilor și a posibilității de colectare a datelor. Pentru căutarea finală, platforma ANOFM a fost exclusă din etapele de căutare iunie-iulie 2026 din cauza problemelor tehnice constatate la momentul testării automate, iar colectarea a continuat pe eJobs, Hipo.ro și BestJobs.

Pasul 2 **Stabilirea criteriilor de căutare**. După validarea platformelor au fost definite criteriile utilizate pentru construirea interogărilor și s-a realizat filtrarea inițială a rezultatelor. Căutările au fost configurate în funcție de:

(1) județele și municipiile de interes

Municipiu/Oraș	Județ	Regiune de dezvoltare (NUTS II)
București	Municipiul București	București-Ilfov
Ilfov	Ilfov	București-Ilfov
Târgoviște	Dâmbovița	Sud-Muntenia
Ploiești	Prahova	Sud-Muntenia
Câmpina	Prahova	Sud-Muntenia
Pitești	Argeș	Sud-Muntenia
Câmpulung	Argeș	Sud-Muntenia
Curtea de Argeș	Argeș	Sud-Muntenia
Brașov	Brașov	Centru
Săcele	Brașov	Centru
Codlea	Brașov	Centru
Făgăraș	Brașov	Centru

(2) domeniul de activitate

Domeniile de activitate au fost corelate cu specializările analizate și au inclus, cu prioritate, inginerie, producție, construcții, management, agricultură, chimie, mediu, calitate și industria alimentară.

(3) nivelul de studii solicitat
Au fost incluse doar anunțurile care solicită studii superioare.

(4) nivelul de experiență.
Au fost selectate toate nivelurile de experiență (fără experiență, entry level, middle level, senior, executive), ceea ce a oferit flexibilitatea de a filtra ulterior anunțurile după nivelurile de experiență dorite, precum și analiza pentru niveluri de experiență superioare.

Pasul 3 Colectarea linkurilor către anunțurile de angajare. Prin intermediul scripturilor dezvoltate în Python 3.14.6 au fost parcurse paginile de rezultate ale platformelor selectate, utilizând biblioteca Selenium pentru automatizarea navigării și identificarea anunțurilor relevante. Pentru fiecare combinație de criterii de căutare au fost colectate URL-urile anunțurilor individuale și informațiile minimale necesare localizării acestora în etapa următoare. După finalizarea colectării, linkurile au fost agregate într-o bază unică și supuse unui proces de eliminare a duplicatelor, astfel încât fiecare anunț să fie procesat o singură dată.

Mențiuni specifice: Fluxul tehnic a fost adaptat particularităților fiecărei platforme. Pentru eJobs au fost utilizate scripturi dedicate colectării linkurilor din paginile de rezultate. Pentru BestJobs, scriptul a acționat repetat opțiunea „Încarcă mai mult", extinzând lista de rezultate înainte de extragerea URL-urilor unice. Pentru Hipo.ro au fost colectate linkurile asociate anunțurilor identificate pe baza criteriilor de căutare stabilite anterior.

Pasul 4 Extragerea și structurarea datelor. După constituirea bazei de linkuri unice, a fost realizată extragerea automată a informațiilor din conținutul fiecărui anunț prin intermediul unor scripturi distincte dezvoltate în Python 3.14.6. Scripturile au accesat fiecare URL colectat și au extras câmpurile relevante pentru analiză. Datele rezultate au fost prelucrate și organizate cu ajutorul bibliotecii Pandas, fiind ulterior salvate în fișiere Excel și CSV pentru etapele de validare și analiză.

Câmpurile colectate au inclus titlul postului, angajatorul, localitatea și județul, domeniul de activitate, tipul de contract, regimul de lucru, descrierea companiei, cerințele și responsabilitățile postului, salariul și certificările solicitate, acolo unde aceste informații au fost disponibile.

Mențiuni specifice: Procesul de extragere a fost adaptat structurii informaționale a fiecărei platforme. Pentru eJobs au fost utilizate scripturi separate de cele destinate colectării linkurilor. Pentru Hipo.ro au fost extrase atât câmpurile structurate disponibile în pagină, cât și variabila generală *continut_anunt*, utilizată în situațiile în care secțiunile individuale nu puteau fi delimitate automat. În cazul platformei BestJobs, scripturile au identificat și extras informațiile relevante din structura HTML specifică fiecărui anunț.

Pasul 5 Verificarea eligibilității și încadrarea pe specializări. Datele extrase au fost procesate și filtrate manual și automat din perspectiva compatibilității postului cu formarea universitară specifică. Eligibilitatea a fost stabilită pe baza titlului postului, domeniului de activitate, angajatorului, cerințelor, responsabilităților, descrierii postului și conținutului general al anunțului.

Mențiuni specifice: Au fost excluse posturile predominant operative sau de execuție — de exemplu, operator, muncitor, manipulant, instalator, electrician, mecanic auto ori tehnician fără cerință clară de studii superioare. Au fost păstrate rolurile de inginer, specialist, manager, coordonator, responsabil de calitate, responsabil de mediu, tehnolog, analist, proiectant și alte poziții compatibile cu pregătirea universitară.

Încadrarea a urmărit particularitățile fiecărei specializări. Pentru Ingineria mediului au fost selectate roluri privind protecția mediului, HSE/QHSE, tratarea apei, emisiile și infrastructura edilitară; pentru Ingineria produselor alimentare, posturi din industria alimentară, calitate, laborator, siguranță alimentară și tehnologie alimentară; iar pentru Inginerie chimică, roluri asociate proceselor chimice, activității de laborator, industriei farmaceutice, polimerilor și materiilor prime industriale. Posturile de arhitect au fost eliminate din categoria „Inginerie și management", întrucât presupun o specializare distinctă.

Pasul 6 Nișarea rezultatelor metodologiei de investigare a oportunităților de angajare la piața muncii regională

După constituirea bazei complete de anunțuri și validarea acestora din perspectiva eligibilității, a fost realizată extragerea automată a subsetului corespunzător Regiunii Sud-Muntenia. Operațiunea a fost efectuată prin intermediul unor scripturi dezvoltate în Python 3.14.6, care au filtrat anunțurile pe baza localității și județului asociate fiecărui post. Au fost selectate exclusiv anunțurile aferente județelor Argeș, Dâmbovița și Prahova incluse în aria de investigare.

Mențiuni specifice: În etapele inițiale ale metodologiei au fost analizate atât localități din Regiunea Sud-Muntenia, cât și din Regiunea Centru și din regiunea București-Ilfov. Această abordare a urmărit creșterea numărului de anunțuri disponibile pentru analiză și captarea unei imagini cât mai relevante a cererii de competențe din zona de influență a județului Dâmbovița. Au fost incluse centre economice și universitare dezvoltate, precum București, Ploiești, Pitești și Brașov, care oferă un număr ridicat de oportunități de angajare și reprezintă destinații frecvente pentru mobilitatea profesională a absolvenților din județele învecinate. În consecință, analiza competențelor solicitate de angajatori beneficiază de o bază empirică mai extinsă și mai diversificată. Ulterior, utilizarea filtrării automate în Python permite restrângerea bazei de date la nivelul teritorial de interes și realizarea unor analize dedicate exclusiv Regiunii Sud-Muntenia.

Pasul 7 Nișarea rezultatelor metodologiei de investigare a oportunităților de angajare la piața muncii locală – municipiul Târgoviște

În etapa finală a fluxului metodologic a fost realizată selectarea automată a anunțurilor relevante pentru municipiul Târgoviște, utilizând filtre dezvoltate în Python 3.14.6. Procedura a permis extragerea unui subset local din baza de date regională și multiregională constituită anterior, fără a fi necesară repetarea procesului de colectare.

Mențiuni specifice: Abordarea metodologică a fost concepută astfel încât rezultatele să poată fi analizate la diferite niveluri teritoriale. Pornind de la o bază de date extinsă, care include oportunități de angajare din principalele centre economice accesibile absolvenților din județul Dâmbovița, datele pot fi nișate succesiv la nivel regional sau local. În cazul municipiului Târgoviște, filtrarea automată permite identificarea particularităților pieței locale a muncii. Astfel, metodologia oferă atât o perspectivă largă asupra competențelor solicitate în zona de mobilitate profesională a absolvenților, cât și o analiză focalizată asupra oportunităților disponibile în proximitatea principalului centru urban al județului Dâmbovița.



Figura 1. Fluxul metodologic de investigare a oportunităților de angajare pe piața muncii locală și regională

Sursă: realizată de autori

Rezultatul metodologic. Aplicarea celor șapte pași a condus la constituirea unei baze structurate de anunțuri de angajare. Pentru fiecare specializare a fost adăugată o coloană în fișierul Excel, iar anunțurile eligibile au fost marcate corespunzător. Baza păstrează conținutul textual și încadrarea tematică și susține analiza competențelor solicitate pe piața muncii în raport cu profilurile programelor de studii. În secțiunea **Anexe** se regăsesc cu titlu de exemplu scripturile utilizate pentru (a) Pasul 3 - colectarea anunțurilor de pe platforma Ejobs și (b) Pasul 4 - extragerea datelor din anunțurile eligibile (Notă: în funcție de setările/resetările aplicate site-ului, scripturile pot necesita modificări.)

Capitolul 2. Perspectivă generală asupra anunțurilor de recrutare identificate

2.1. Constituirea bazei finale și delimitarea universurilor de analiză

Baza de date utilizată pentru investigarea oportunităților de angajare a fost constituită progresiv, prin runde succesive de căutare desfășurate în perioada mai–iulie 2026 pe platformele de recrutare selectate. Procesul a urmărit identificarea unui volum cât mai divers de anunțuri potențial relevante, urmată de verificarea și filtrarea acestora în raport cu domeniile de studii analizate și cu nivelul de calificare vizat. După colectarea anunțurilor, rezultatele provenite din rundele și platformele diferite au fost reunite într-o bază comună. Aceasta a fost supusă unui proces de consolidare care a inclus eliminarea dublurilor, verificarea relevanței tematice și revizuirea individuală a cazurilor aflate la limita criteriilor de includere. Relevanța unui anunț nu a fost stabilită exclusiv pe baza titlului postului sau a categoriei în care acesta era încadrat de platformă, ci prin examinarea ansamblului informațiilor disponibile privind cerințele educaționale, responsabilitățile, domeniul de activitate și conținutul tehnic al postului. În urma acestei etape a fost constituită o bază extinsă de 407 anunțuri unice și relevante tematic.

Pentru investigarea oportunităților de angajare accesibile absolvenților a fost delimitat, din baza extinsă, un subset distinct format din 139 de anunțuri entry-level sau compatibile cu nivelul entry-level. Separarea celor două universuri de analiză răspunde unor obiective diferite: baza de 139 de anunțuri este utilizată pentru caracterizarea oportunităților de acces pe piața muncii, în timp ce baza extinsă de 407 anunțuri, care include și poziții pentru niveluri medii și avansate de experiență, este păstrată pentru analiza cerințelor și competențelor solicitate de angajatori.

Încadrarea unui anunț în categoria entry-level sau compatibilă cu nivelul entry-level nu s-a bazat exclusiv pe eticheta de experiență furnizată de platforma de recrutare. Au fost analizate cerințele efective privind experiența profesională, responsabilitățile postului și nivelul de autonomie sau coordonare asociat poziției. Au fost considerate accesibile la începutul carierei anunțurile adresate explicit candidaților entry-level, juniorilor, studenților sau proaspeților absolvenți, precum și posturile care solicitau maximum 0–2 ani de experiență, prezentau experiența anterioară doar ca avantaj sau nu formulau cerințe incompatibile cu începutul parcursului profesional. Au fost încadrate în afara subsetului descriptiv pozițiile care solicitau explicit un nivel mai ridicat de experiență sau presupuneau responsabilități evidente de coordonare, management ori expertiză profesională avansată.

Delimitarea celor două universuri analitice permite separarea oportunităților efectiv relevante pentru intrarea absolvenților pe piața muncii de posturile care descriu etape profesionale ulterioare. În același timp, păstrarea bazei tematice extinse permite valorificarea informațiilor privind competențele și cerințele profesionale formulate de angajatori pentru niveluri diferite de experiență.

2.2. Caracterizarea descriptivă a oportunităților de angajare identificate

Distribuția anunțurilor pe platforme

Distribuția anunțurilor evidențiază o concentrare puternică pe BestJobs, de unde provin 85,61% dintre oportunitățile incluse în baza descriptivă. eJobs reprezintă 7,91%, ANOFM 5,04%, iar Hipo 1,44%.

Tabelul 1. Distribuția anunțurilor pe platforme

Platformă	N	%
BestJobs	119	85,61%
eJobs	11	7,91%
ANOFM	7	5,04%
Hipo	2	1,44%
Total	**139**	**100,00%**

Sursa: prelucrare proprie pe baza anunțurilor incluse în analiza descriptivă.

Această structură reflectă procesul de colectare și disponibilitatea anunțurilor relevante identificate pe fiecare platformă și nu trebuie interpretată ca o măsură directă a cotei de piață a platformelor de recrutare. În cadrul bazei analizate, BestJobs a reprezentat însă principala sursă de oportunități identificate ca accesibile persoanelor aflate la începutul carierei.

Distribuția oportunităților în funcție de aria geografică

Pentru analiza geografică, fiecare anunț a fost clasificat o singură dată, chiar dacă includea mai multe localizări. În cazul anunțurilor cu localizări multiple a fost utilizată localizarea considerată cea mai accesibilă din perspectiva absolvenților din Târgoviște. Au fost delimitate patru categorii: Dâmbovița, Prahova, București–Ilfov și alte județe limitrofe incluse în aria relevantă. Categoria „alte județe limitrofe" include Argeș, Brașov, Giurgiu și Teleorman, în timp ce Prahova și Ilfov au fost tratate distinct, având în vedere ponderea lor în baza analizată și, în cazul Ilfovului, integrarea funcțională cu piața muncii din București.

Tabelul 2. Distribuția oportunităților pe arii geografice relevante

Arie geografică	N	%
București–Ilfov	75	54,0%
Prahova	29	20,9%
Alte județe limitrofe	29	20,9%
Dâmbovița	6	4,3%
Total	**139**	**100,0%**

Notă: În cazul anunțurilor cu localizări multiple, clasificarea a urmat ordinea de prioritate Dâmbovița – Prahova – București-Ilfov – alte județe limitrofe, astfel încât fiecare anunț să fie numărat o singură dată.

Sursa: prelucrare proprie pe baza anunțurilor incluse în analiza descriptivă.

Rezultatele indică o concentrare puternică a oportunităților în zona București–Ilfov, căreia îi sunt asociate 54,0% dintre anunțurile analizate. Prahova concentrează 20,9% dintre oportunități, aceeași pondere revenind celorlalte județe limitrofe incluse în aria geografică relevantă, în timp ce numai 4,3% dintre anunțuri sunt localizate în Dâmbovița.

Structura evidențiază dimensiunea relativ redusă a ofertei locale și rolul mobilității geografice în accesul absolvenților din Târgoviște la oportunități de ocupare. București–Ilfov constituie principalul pol de ocupare din aria analizată, completat de Prahova și de celelalte județe limitrofe.

Distribuția anunțurilor în funcție de domeniul de studii asociat

Cele mai multe oportunități sunt asociate domeniului Inginerie și management (34,5%), urmat de Inginerie mecanică (27,3%). Împreună, cele două domenii concentrează 61,8% din oferta de posturi entry-level sau compatibile cu nivelul entry-level identificată în baza analizată.

Tabelul 3. Distribuția anunțurilor pe domenii de studii asociate

Domeniu de studii asociat	N	%
Inginerie și management	48	34,5%
Inginerie mecanică	38	27,3%
Ingineria produselor alimentare	17	12,2%
Inginerie chimică	14	10,1%
Ingineria materialelor	9	6,5%
Agronomie	7	5,0%
Ingineria mediului	6	4,3%
Total	**139**	**100,0%**

Sursa: prelucrare proprie pe baza anunțurilor incluse în analiza descriptivă.

Ponderi mai reduse, dar relevante, se înregistrează pentru Ingineria produselor alimentare (12,2%) și Inginerie chimică (10,1%). Ingineria materialelor reprezintă 6,5%, Agronomia 5,0%, iar Ingineria mediului 4,3%. Distribuția sugerează diferențe între domeniile de studii în ceea ce privește volumul oportunităților accesibile la începutul carierei, cu o concentrare mai accentuată în zona rolurilor asociate managementului tehnic și ingineriei mecanice.

Distribuția anunțurilor pe familii ocupaționale

Structura pe familii ocupaționale evidențiază o diversitate ridicată a rolurilor accesibile la începutul carierei. Cea mai bine reprezentată categorie este Calitate, audit și conformitate (20,1%), urmată de Producție și inginerie de proces (14,4%) și Proiectare și dezvoltare tehnică (10,1%). Împreună, aceste trei categorii concentrează 44,6% dintre oportunitățile analizate.

O reprezentare importantă au și rolurile din ofertare, vânzări tehnice și dezvoltare comercială (8,6%), precum și cele din mentenanță, service și punere în funcțiune și laborator, testare și analize (câte 7,9%).

Tabelul 4. Distribuția anunțurilor pe familii ocupaționale

Familie ocupațională	N	%
Calitate, audit și conformitate	28	20,1%
Producție și inginerie de proces	20	14,4%
Proiectare și dezvoltare tehnică	14	10,1%
Ofertare, vânzări tehnice și dezvoltare comercială	12	8,6%
Mentenanță, service și punere în funcțiune	11	7,9%
Laborator, testare și analize	11	7,9%
Achiziții, aprovizionare și supply chain	8	5,8%
Mediu, SSM/EHS și sustenabilitate	7	5,0%
Roluri tehnice multiple / funcție nespecificată	7	5,0%
Management de proiect și coordonare tehnică	6	4,3%
Logistică, planificare și operațiuni	6	4,3%
Cercetare-dezvoltare și inovare	5	3,6%
Agronomie și activități agricole	4	2,9%
Total	**139**	**100,0%**

Sursa: prelucrare proprie pe baza anunțurilor incluse în analiza descriptivă.

Rezultatele arată că oportunitățile accesibile absolvenților nu se limitează la posturile de proiectare sau producție propriu-zisă, ci includ într-o măsură importantă funcții de calitate și conformitate, activități comerciale cu componentă tehnică, laborator, mentenanță, aprovizionare și alte funcții suport ale proceselor industriale.

Distribuția anunțurilor în funcție de nivelul de experiență

Din totalul celor 139 de anunțuri incluse în analiza descriptivă, 28,8% sunt poziții explicit încadrate ca entry-level, iar 71,2% au fost clasificate ca fiind compatibile cu nivelul entry-level sau ca poziții cu nivel mixt de experiență.

Ponderea ridicată a celei de-a doua categorii arată că oportunitățile accesibile persoanelor aflate la începutul carierei nu sunt limitate la posturile etichetate explicit ca entry-level. În numeroase cazuri, compatibilitatea a rezultat din analiza cerințelor efective ale anunțului: experiență redusă, caracterul opțional al experienței anterioare sau absența responsabilităților specifice unui nivel profesional avansat. Utilizarea exclusivă a etichetelor de experiență furnizate de platformele de recrutare ar fi condus, în consecință, la o subestimare a volumului oportunităților potențial accesibile absolvenților.

Tabelul 5. Distribuția anunțurilor după nivelul de experiență

Nivel standardizat	N	%
Compatibil entry-level / nivel mixt	99	71,2%
Entry-level explicit	40	28,8%
Total	**139**	**100,0%**

Sursa: prelucrare proprie pe baza anunțurilor incluse în analiza descriptivă.

În completarea caracterizării descriptive, figura de mai jos oferă o reprezentare sintetică a denumirilor de posturi identificate în baza analizată.

Dimensiunea mai mare a unor termeni indică o recurență mai ridicată a acestora în titlurile posturilor, sugerând concentrarea oportunităților de angajare în jurul unor profiluri precum inginer proiectant, inginer producție, inginer mecanic, inginer calitate, specialist în domeniul calității sau inginer chimist. Reprezentarea are un rol ilustrativ și completează analiza statistică descriptivă, fără a substitui clasificarea riguroasă a anunțurilor pe domenii de studii, ocupații și niveluri de experiență.

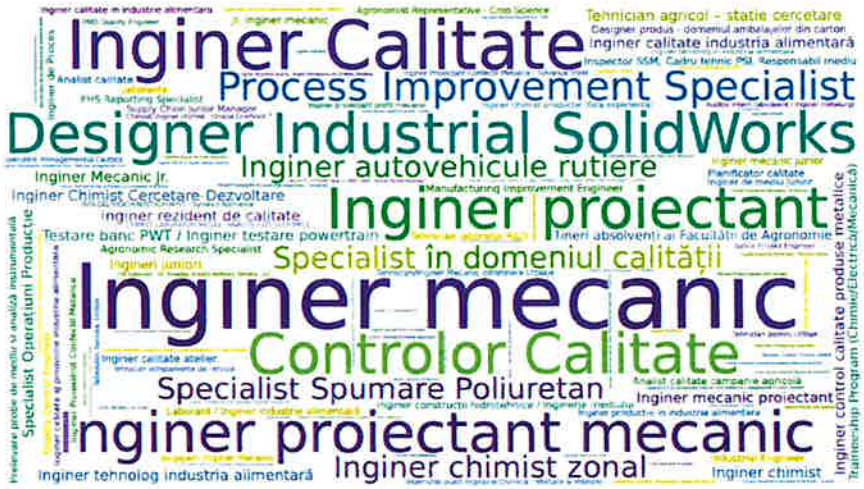


Figura 2. Reprezentarea vizuală a celor mai frecvente denumiri ale posturilor din baza finală de anunțuri

Sursă: prelucrare proprie a autorilor, pe baza bazei finale de anunțuri colectate în cadrul proiectului

Reprezentarea anunțurilor în funcție de domeniul de studii asociat evidențiază predominanța oportunităților asociate domeniilor Inginerie și management și Inginerie mecanică, urmate de Ingineria produselor alimentare și Inginerie chimică. Domeniile Ingineria materialelor, Agronomie și Ingineria mediului sunt mai puțin reprezentate în baza analizată. Dimensiunea denumirilor reflectă frecvența relativă a asocierii anunțurilor cu fiecare domeniu de studii.

Concentrarea asupra specializărilor inginerești trebuie interpretată și în raport cu filtrarea inițială a anunțurilor, realizată în funcție de domeniile de studii vizate. Prin urmare, reprezentarea descrie structura anunțurilor selectate și nu distribuția tuturor domeniilor de studii solicitate pe piața muncii regională.



Figura 3 Reprezentarea anunțurilor după domeniul de studiu asociat
Sursă: prelucrare proprie a autorilor, pe baza bazei finale de anunțuri colectate în cadrul proiectului

În ansamblu, analiza descriptivă evidențiază o concentrare a oportunităților identificate în zona București–Ilfov și în familiile ocupaționale asociate calității, producției și proiectării, cu o reprezentare mai ridicată a domeniilor Inginerie și management și Inginerie mecanică. Rezultatele descriu structura bazei constituite prin metodologia prezentată și trebuie interpretate în limitele procesului de colectare și ale criteriilor tematice aplicate, fără a fi extrapolate la ansamblul pieței muncii.

Anexe

Scripturile sunt prezentate cu titlu ilustrativ și reflectă structura platformei la momentul colectării datelor. Modificările ulterioare ale structurii paginilor web pot necesita actualizarea selectorilor și a parametrilor utilizați.

Script Python pentru colectarea linkurilor anunțurilor de pe platforma eJobs

```
"""Script eJobs pentru colectarea anunțurilor eligibile de pe platforma eJobs."""

import re
import time
from urllib.parse import urljoin

import pandas as pd
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
```

```
# ============================================================
# SETĂRI PRINCIPALE
# ============================================================

SEARCH_URL = "https://www.ejobs.ro/locuri-de-munca/bucuresti,ilfov,targoviste,ploiesti,campina,pitesti,campulung,curtea-de-arges,brasov,sacele,codlea,fagaras/nivel-studii-absolvent/administratie---sector-public,agrara,alimentara,chimica,comert---retail,constructii,energetica,lemn---pvc,masini---auto,medicina---sanatate,navala---aeronautica,manufactura,protectia--mediului,textila,transport---logistica---import---export"

BASE_URL = "https://www.ejobs.ro"

# Pentru rulare completă: 100
MAX_PAGES = 100

OUTPUT_EXCEL_FINAL = "ejobs_linkuri_joburi_final.xlsx"
OUTPUT_CSV_FINAL = "ejobs_linkuri_joburi_final.csv"
OUTPUT_EXCEL_PARTIAL = "ejobs_linkuri_joburi_partial.xlsx"
OUTPUT_CSV_PARTIAL = "ejobs_linkuri_joburi_partial.csv"


# ============================================================
# BROWSER
# ============================================================

def setup_driver():
    options = Options()
    options.add_argument("--window-size=1440,1200")
    options.add_argument("--disable-notifications")
    options.add_argument("--lang=ro-RO")

    # Pentru rulare în fundal, poți activa:
    # options.add_argument("--headless=new")

    driver = webdriver.Chrome(options=options)
    driver.set_page_load_timeout(45)
    return driver


# ============================================================
# FUNCȚII UTILITARE
# ============================================================

def clean_text(text):
    return re.sub(r"\s+", " ", text or "").strip()


def normalize_link(href):
    if not href:
        return ""

    href = urljoin(BASE_URL, href)
    href = href.split("?")[0].split("#")[0]
    return href


def is_real_job_link(href):
```

```
    """
    Păstrează doar linkuri reale de anunțuri eJobs.

    Exemplu valid:
    https://www.ejobs.ro/user/locuri-de-munca/asistent-manager/1965510
    """
    if not href:
        return False

    href = normalize_link(href)
    pattern = r"https://www\.ejobs\.ro/user/locuri-de-munca/[^/]+/\d+$"
    return re.match(pattern, href) is not None


def accept_cookies_if_present(driver):
    """
    Închide bannerul de cookies dacă apare.
    Dacă nu apare, scriptul continuă normal.
    """
    possible_texts = [
        "Accept",
        "Accept toate",
        "Acceptă",
        "Accepta",
        "Sunt de acord",
        "De acord",
        "OK",
    ]

    try:
        buttons = driver.find_elements(By.TAG_NAME, "button")
        for button in buttons:
            try:
                text = clean_text(button.text)
                if text in possible_texts:
                    button.click()
                    time.sleep(1)
                    print("Banner cookies închis.")
                    return
            except Exception:
                pass
    except Exception:
        pass


def build_page_url(search_url, page_number):
    """
    Pagina 1: URL-ul inițial.
    Pagina 2: URL-ul inițial + /pagina2.
    Pagina 3: URL-ul inițial + /pagina3.
    """
    clean_url = search_url.rstrip("/")

    if page_number == 1:
        return clean_url

    return f"{clean_url}/pagina{page_number}"
```

```
# ============================================================
# SCROLL
# ============================================================

def scroll_slowly(driver, steps=18, pixels_per_step=700, pause=0.7):
    """Face scroll lent pentru a permite încărcarea cardurilor de joburi."""
    for _ in range(steps):
        driver.execute_script(f"window.scrollBy(0, {pixels_per_step});")
        time.sleep(pause)

    driver.execute_script("window.scrollTo(0, 0);")
    time.sleep(1)


# ============================================================
# COLECTARE LINKURI CU CONTEXT
# ============================================================

def get_candidate_job_links_with_context(driver, page_number, page_url):
    """Colectează linkuri de joburi împreună cu informații de context."""
    rows = []
    anchors = driver.find_elements(
        By.CSS_SELECTOR,
        "a[href*='/user/locuri-de-munca/']",
    )

    print(f"Linkuri candidate în DOM: {len(anchors)}")

    for idx, anchor in enumerate(anchors, start=1):
        try:
            href = normalize_link(anchor.get_attribute("href"))

            if not is_real_job_link(href):
                continue

            info = driver.execute_script(
                """
                const a = arguments[0];
                const r = a.getBoundingClientRect();
                let container = a.closest('article, li, div');
                let containerText = container ? container.innerText : a.innerText;

                return {
                    x: Math.round(r.x),
                    y: Math.round(r.y + window.scrollY),
                    width: Math.round(r.width),
                    height: Math.round(r.height),
                    visible: !!(r.width && r.height),
                    linkText: a.innerText || '',
                    containerText: containerText || ''
                };
                """,
                anchor,
            )

            rows.append(
                {
```

```
                "pagina_sursa": page_number,
                "url_pagina_sursa": page_url,
                "nr_in_dom": idx,
                "link": href,
                "link_text": clean_text(info.get("linkText", "")),
                "container_text": clean_text(info.get("containerText", "")),
                "x": info.get("x"),
                "y": info.get("y"),
                "width": info.get("width"),
                "height": info.get("height"),
                "visible": info.get("visible"),
            }
        )
    except Exception as exc:
        print(f"Eroare la linkul candidat {idx}: {exc}")

return rows


def filter_likely_main_results(df):
    """
    Filtrare prudentă:
    - păstrăm doar elemente vizibile;
    - păstrăm doar elemente cu dimensiuni reale;
    - eliminăm duplicatele de pe aceeași pagină;
    - ordonăm după poziția în pagină.
    """
    if df.empty:
        return df

    filtered = df.copy()
    filtered = filtered[filtered["visible"] == True]
    filtered = filtered[filtered["width"] > 0]
    filtered = filtered[filtered["height"] > 0]
    filtered = filtered[filtered["y"] >= 0]
    filtered = filtered[
        (filtered["link_text"].str.len() > 0)
        | (filtered["container_text"].str.len() > 20)
    ]

    filtered = filtered.sort_values(
        by=["pagina_sursa", "y", "x"],
        ascending=True,
    )

    filtered = filtered.drop_duplicates(
        subset=["pagina_sursa", "link"],
        keep="first",
    )

    return filtered


# ============================================================
# SALVARE
# ============================================================

def prepare_final_dataframe(rows):
```

```
    df = pd.DataFrame(rows)

    if df.empty:
        return df

    df = df.sort_values(
        by=["pagina_sursa", "y", "x"],
        ascending=True,
    )

    # Eliminăm duplicatele globale.
    # Dacă același job apare pe mai multe pagini, păstrăm prima apariție.
    df = df.drop_duplicates(
        subset=["link"],
        keep="first",
    )

    return df


def save_partial(rows):
    df = prepare_final_dataframe(rows)

    if df.empty:
        return

    df.to_excel(OUTPUT_EXCEL_PARTIAL, index=False)
    df.to_csv(OUTPUT_CSV_PARTIAL, index=False, encoding="utf-8-sig")
    print(f"Salvare parțială: {len(df)} linkuri unice până acum.")


def save_final(df):
    if df.empty:
        print("Nu am nimic de salvat.")
        return

    columns_for_final_file = [
        "pagina_sursa",
        "link",
        "link_text",
        "container_text",
        "url_pagina_sursa",
        "x",
        "y",
        "width",
        "height",
        "visible",
    ]

    existing_columns = [
        col for col in columns_for_final_file if col in df.columns
    ]
    df_final = df[existing_columns].copy()

    df_final.to_excel(OUTPUT_EXCEL_FINAL, index=False)
    df_final.to_csv(OUTPUT_CSV_FINAL, index=False, encoding="utf-8-sig")

    print()
```

```
print("Fișiere finale salvate:")
print(OUTPUT_EXCEL_FINAL)
print(OUTPUT_CSV_FINAL)


# ============================================================
# COLECTARE DE PE MAI MULTE PAGINI
# ============================================================

def collect_links_from_search_pages(driver, search_url, max_pages):
    all_rows = []

    for page_number in range(1, max_pages + 1):
        page_url = build_page_url(search_url, page_number)

        print()
        print("=" * 80)
        print(f"Deschid pagina {page_number}/{max_pages}:")
        print(page_url)
        print("=" * 80)

        try:
            driver.get(page_url)
        except Exception as exc:
            print(f"Eroare la deschiderea paginii {page_number}: {exc}")
            break

        time.sleep(6)
        accept_cookies_if_present(driver)

        scroll_slowly(
            driver=driver,
            steps=18,
            pixels_per_step=700,
            pause=0.7,
        )

        rows = get_candidate_job_links_with_context(
            driver=driver,
            page_number=page_number,
            page_url=page_url,
        )

        df_page_all = pd.DataFrame(rows)

        if df_page_all.empty:
            print(
                f"Pagina {page_number}: nu am găsit linkuri candidate. Oprire."
            )
            break

        df_page_filtered = filter_likely_main_results(df_page_all)
        page_count = len(df_page_filtered)

        print(
            f"Pagina {page_number}: linkuri reale după filtrare: {page_count}"
        )
```

```
    if page_count == 0:
        print(f"Pagina {page_number}: 0 linkuri după filtrare. Oprire.")
        break

    all_rows.extend(df_page_filtered.to_dict("records"))
    save_partial(all_rows)
    time.sleep(3)

  return prepare_final_dataframe(all_rows)


# ============================================================
# MAIN
# ============================================================

def main():
  driver = setup_driver()

  try:
    df_links = collect_links_from_search_pages(
      driver=driver,
      search_url=SEARCH_URL,
      max_pages=MAX_PAGES,
    )

    print()
    print("=" * 80)
    print(f"TOTAL FINAL LINKURI UNICE: {len(df_links)}")
    print("=" * 80)

    if not df_links.empty:
      print(df_links[["pagina_sursa", "link"]].to_string(index=False))

    save_final(df_links)
    input("Apasă Enter ca să închizi browserul...")
  finally:
    driver.quit()

if __name__ == "__main__":
  main()
```

Script Python pentru extragerea datelor din anunțurile selectate de pe platforma eJobs

```
import time
import re
import os

import pandas as pd

from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By


# ============================================================
# SETARI PRINCIPALE
# ============================================================
```

```
INPUT_EXCEL = "ejobs_linkuri_joburi_final.xlsx"

OUTPUT_EXCEL_FINAL = "ejobs_detalii_joburi_final.xlsx"
OUTPUT_CSV_FINAL = "ejobs_detalii_joburi_final.csv"

OUTPUT_EXCEL_PARTIAL = "ejobs_detalii_joburi_partial.xlsx"
OUTPUT_CSV_PARTIAL = "ejobs_detalii_joburi_partial.csv"

# Cate linkuri vrei sa procesezi din fisierul cu linkuri.
MAX_JOBS_TO_EXTRACT = 1000

# Salveaza progresul dupa fiecare N joburi.
SAVE_EVERY = 10

# Pauza intre anunturi.
DELAY_BETWEEN_JOBS = 2

# Daca True, incearca sa continue din fisierul partial existent.
RESUME_FROM_PARTIAL = True


# ============================================================
# BROWSER
# ============================================================

def setup_driver():
    options = Options()
    options.add_argument("--window-size=1440,1200")
    options.add_argument("--disable-notifications")
    options.add_argument("--lang=ro-RO")

    # Pentru rulare completa, poti activa headless.
    # Daca vrei sa vezi browserul, lasa linia comentata.
    # options.add_argument("--headless=new")

    driver = webdriver.Chrome(options=options)
    driver.set_page_load_timeout(45)

    return driver


# ============================================================
# FUNCTII UTILITARE
# ============================================================

def clean_text(text):
    return re.sub(r"\s+", " ", text or "").strip()


def clean_multiline(text):
    lines = [clean_text(line) for line in (text or "").splitlines()]
    lines = [line for line in lines if line]
    return "\n".join(lines)


def get_page_text(driver):
    try:
```

```
        return driver.find_element(By.TAG_NAME, "body").text
    except Exception:
        return ""


def first_text(driver, selectors):
    for selector in selectors:
        try:
            elements = driver.find_elements(By.CSS_SELECTOR, selector)

            for element in elements:
                text = clean_text(element.text)

                if text:
                    return text

        except Exception:
            pass

    return ""


def remove_cookie_text(text):
    if not text:
        return ""

    cookie_markers = [
        "Administrare cookies",
        "Folosim cookie-uri",
        "Modifică setările",
        "Modifica setarile",
        "Acceptă toate",
        "Accepta toate",
        "Vezi politica noastră de cookies",
        "Vezi politica noastra de cookies",
    ]

    cleaned = text

    for marker in cookie_markers:
        index = cleaned.find(marker)
        if index != -1:
            cleaned = cleaned[:index]

    return cleaned.strip()


# ============================================================
# COOKIES / BUTOANE / SCROLL
# ============================================================

def accept_cookies_if_present(driver):
    possible_texts = [
        "Accept",
        "Accept toate",
        "Acceptă",
        "Accepta",
        "Acceptă toate",
```

```
    "Accepta toate",
    "Sunt de acord",
    "De acord",
    "OK",
]

try:
    elements = driver.find_elements(By.XPATH, "//*")

    for element in elements:
        try:
            text = clean_text(element.text)

            if text in possible_texts and element.is_displayed():
                driver.execute_script("arguments[0].click();", element)
                time.sleep(1)
                print("Banner cookies inchis.")
                return

        except Exception:
            pass

except Exception:
    pass


def click_see_more_buttons(driver):
    possible_texts = [
        "Vezi mai mult",
        "Vezi mai multe",
        "Vezi tot",
        "Arată mai mult",
        "Arata mai mult",
    ]

    clicked = 0

    try:
        elements = driver.find_elements(By.XPATH, "//*")

        for element in elements:
            try:
                text = clean_text(element.text)

                if text in possible_texts and element.is_displayed():
                    driver.execute_script("arguments[0].click();", element)
                    clicked += 1
                    time.sleep(0.7)

            except Exception:
                pass

    except Exception:
        pass

    if clicked:
        print(f"Am apasat pe {clicked} butoane/linkuri 'Vezi mai mult'.")
```

```
def scroll_job_page(driver):
    for _ in range(10):
        driver.execute_script("window.scrollBy(0, 700);")
        time.sleep(0.5)

    time.sleep(1)

    driver.execute_script("window.scrollTo(0, 0);")
    time.sleep(0.5)


# ============================================================
# EXTRAGERE DIN TEXT
# ============================================================

def extract_title_company_from_page_text(page_text):
    lines = [clean_text(line) for line in page_text.splitlines()]
    lines = [line for line in lines if line]

    markers = [
        "Raportează conținut ilegal",
        "Raporteaza continut ilegal",
    ]

    marker_index = None

    for i, line in enumerate(lines):
        if line in markers:
            marker_index = i
            break

    if marker_index is None:
        return "", ""

    after_marker = lines[marker_index + 1:]

    stop_values = [
        "Aplică",
        "Aplica",
        "Aplică extern",
        "Aplica extern",
        "Candidatul Ideal",
        "Candidatul ideal",
    ]

    useful_lines = []

    for line in after_marker:
        if line in stop_values:
            break

        if line:
            useful_lines.append(line)

    if len(useful_lines) >= 2:
        return useful_lines[0], useful_lines[1]
```

```
    if len(useful_lines) == 1:
        return useful_lines[0], ""

    return "", ""


def find_section(page_text, start_labels, stop_labels):
    text = clean_multiline(page_text)

    start_pattern = "|".join(re.escape(label) for label in start_labels)
    stop_pattern = "|".join(re.escape(label) for label in stop_labels)

    pattern = rf"({start_pattern})\s*:?\s*(.*?)(?=\n(?:{stop_pattern})\s*:?\s*|$)"

    match = re.search(pattern, text, flags=re.IGNORECASE | re.DOTALL)

    if not match:
        return ""

    return clean_multiline(match.group(2))


def extract_job_criteria(page_text):
    lines = [clean_text(line) for line in page_text.splitlines()]
    lines = [line for line in lines if line]

    criteria = []
    start_index = None

    for i, line in enumerate(lines):
        if line.lower() == "criterii job":
            start_index = i + 1
            break

    if start_index is None:
        return {}

    stop_labels = [
        "Candidatul Ideal",
        "Candidatul ideal",
        "Descrierea jobului",
        "Descriere job",
        "Actiuni job",
        "Acțiuni job",
        "Salvează",
        "Salveaza",
        "Ascunde",
    ]

    stop_labels_lower = [x.lower() for x in stop_labels]

    skip_values = [
        "Aplică",
        "Aplica",
        "Aplică extern",
        "Aplica extern",
        "Vezi mai mult",
        "Vezi mai multe",
```

```
    "Vezi mai puțin",
    "Vezi mai putin",
]

skip_values_lower = [x.lower() for x in skip_values]

for line in lines[start_index:]:
    low = line.lower()

    if low in stop_labels_lower:
        break

    if low in skip_values_lower:
        continue

    if line:
        criteria.append(line)

result = {}

for idx, value in enumerate(criteria, start=1):
    result[f"criteriu_{idx}"] = value

result["criterii_job_toate"] = " | ".join(criteria)

return result


# ============================================================
# EXTRAGERE DETALII DINTR-UN ANUNT
# ============================================================

def extract_job_details(driver, job_url):
    driver.get(job_url)
    time.sleep(5)

    accept_cookies_if_present(driver)
    time.sleep(1)

    click_see_more_buttons(driver)
    time.sleep(1)

    scroll_job_page(driver)

    click_see_more_buttons(driver)
    time.sleep(1)

    page_text = get_page_text(driver)
    page_text = remove_cookie_text(page_text)

    title_from_text, company_from_text = extract_title_company_from_page_text(page_text)

    title = title_from_text or first_text(driver, [
        "h1",
        "[data-test='job-title']",
        ".job-title",
    ])
```

```
company = company_from_text or first_text(driver, [
    "[data-test='company-name']",
    ".company-name",
    "a[href*='/companie/']",
    "a[href*='/company/']",
])

criteria_data = extract_job_criteria(page_text)

ideal_candidate = find_section(
    page_text,
    start_labels=[
        "Candidatul Ideal",
        "Candidatul ideal",
        "Profilul candidatului",
        "Cerinte",
        "Cerințe",
        "Ce ne dorim de la tine",
    ],
    stop_labels=[
        "Descrierea jobului",
        "Descriere job",
        "Responsabilitati",
        "Responsabilități",
        "Beneficii",
        "Alte informatii",
        "Alte informații",
        "Despre companie",
        "Descrierea companiei",
        "Joburi similare",
    ],
)

job_description = find_section(
    page_text,
    start_labels=[
        "Descrierea jobului",
        "Descriere job",
        "Responsabilitati",
        "Responsabilități",
        "Ce vei face",
        "Rolul tau",
        "Rolul tău",
    ],
    stop_labels=[
        "Descrierea companiei",
        "Despre companie",
        "Beneficii",
        "Oferta",
        "Oferta (bonusuri și beneficii)",
        "Oferta (bonusuri si beneficii)",
        "Alte informatii",
        "Alte informații",
        "Joburi similare",
        "Aplică",
        "Aplica",
        "Salvează",
        "Salveaza",
```

```
        "Publicat",
        "Reactualizat",
        "Expiră",
        "Expira",
    ],
)

company_description = find_section(
    page_text,
    start_labels=[
        "Descrierea companiei",
        "Despre companie",
    ],
    stop_labels=[
        "Publicat",
        "Reactualizat",
        "Expiră",
        "Expira",
        "Joburi similare",
        "CANDIDAT",
        "COMPANIE",
    ],
)

data = {
    "link": job_url,
    "titlu_job": title,
    "companie": company,
    "candidatul_ideal": ideal_candidate,
    "descrierea_jobului": job_description,
    "descrierea_companiei": company_description,

    # Coloana utila pentru verificare.
    # Poate fi stearsa dupa validarea completa.
    "page_text_debug": page_text[:8000],
}

data.update(criteria_data)

return data


# ============================================================
# CITIRE LINKURI / RELUARE
# ============================================================

def load_job_links():
    df = pd.read_excel(INPUT_EXCEL)

    if "link" not in df.columns:
        raise ValueError("Fisierul de input nu are coloana 'link'.")

    links = df["link"].dropna().astype(str).tolist()

    # Limitam la primele 1000.
    links = links[:MAX_JOBS_TO_EXTRACT]

    return links
```

```
def load_existing_partial_results():
    if not RESUME_FROM_PARTIAL:
        return []

    if not os.path.exists(OUTPUT_EXCEL_PARTIAL):
        return []

    try:
        df_existing = pd.read_excel(OUTPUT_EXCEL_PARTIAL)

        if df_existing.empty:
            return []

        rows = df_existing.to_dict("records")
        print(f"Am incarcat {len(rows)} rezultate existente din fisierul partial.")

        return rows

    except Exception as e:
        print(f"Nu am putut citi fisierul partial: {e}")
        return []


# ============================================================
# SALVARE
# ============================================================

def order_columns(df_out):
    preferred_columns = [
        "link",
        "titlu_job",
        "companie",
        "candidatul_ideal",
        "descrierea_jobului",
        "descrierea_companiei",
        "criterii_job_toate",
    ]

    criteria_columns = sorted(
        [
            col for col in df_out.columns
            if col.startswith("criteriu_") and col != "criterii_job_toate"
        ],
        key=lambda x: int(x.replace("criteriu_", ""))
    )

    debug_columns = [
        "page_text_debug",
        "eroare",
    ]

    ordered_columns = []

    for col in preferred_columns:
        if col in df_out.columns:
            ordered_columns.append(col)
```

```
    for col in criteria_columns:
        if col not in ordered_columns:
            ordered_columns.append(col)

    for col in df_out.columns:
        if col not in ordered_columns and col not in debug_columns:
            ordered_columns.append(col)

    for col in debug_columns:
        if col in df_out.columns:
            ordered_columns.append(col)

    return df_out[ordered_columns]


def save_dataframe(rows, excel_path, csv_path):
    df_out = pd.DataFrame(rows)

    if df_out.empty:
        print("Nu am date de salvat.")
        return

    # Eliminam duplicatele dupa link, pastrand ultima varianta.
    if "link" in df_out.columns:
        df_out = df_out.drop_duplicates(subset=["link"], keep="last")

    df_out = order_columns(df_out)

    df_out.to_excel(excel_path, index=False)
    df_out.to_csv(csv_path, index=False, encoding="utf-8-sig")

    print(f"Salvat: {excel_path}")
    print(f"Salvat: {csv_path}")


def save_partial(rows):
    print()
    print("Salvare partiala...")
    save_dataframe(rows, OUTPUT_EXCEL_PARTIAL, OUTPUT_CSV_PARTIAL)


def save_final(rows):
    print()
    print("Salvare finala...")
    save_dataframe(rows, OUTPUT_EXCEL_FINAL, OUTPUT_CSV_FINAL)


# ============================================================
# MAIN
# ============================================================

def main():
    links = load_job_links()

    print(f"Am incarcat {len(links)} linkuri din {INPUT_EXCEL}.")
    print(f"Voi procesa maximum {MAX_JOBS_TO_EXTRACT} linkuri.")
```

```
rows = load_existing_partial_results()

already_done = set()

for row in rows:
    link = str(row.get("link", "")).strip()
    if link:
        already_done.add(link)

remaining_links = [link for link in links if link not in already_done]

print(f"Deja procesate: {len(already_done)}")
print(f"Ramase de procesat: {len(remaining_links)}")

if not remaining_links:
    print("Nu mai sunt linkuri de procesat.")
    save_final(rows)
    return

driver = setup_driver()

try:
    total_remaining = len(remaining_links)

    for index, link in enumerate(remaining_links, start=1):
        global_index = len(already_done) + index

        print()
        print("=" * 80)
        print(f"[{index}/{total_remaining}] Global [{global_index}/{len(links)}]")
        print("Extrag date din:")
        print(link)
        print("=" * 80)

        try:
            data = extract_job_details(driver, link)
            rows.append(data)

            print("Titlu:", data.get("titlu_job", ""))
            print("Companie:", data.get("companie", ""))
            print("Candidat ideal - caractere:", len(data.get("candidatul_ideal", "")))
            print("Descriere job - caractere:", len(data.get("descrierea_jobului", "")))
            print("Descriere companie - caractere:", len(data.get("descrierea_companiei", "")))
            print("Criterii job:", data.get("criterii_job_toate", ""))

        except Exception as e:
            print(f"Eroare la linkul {link}: {e}")

            rows.append({
                "link": link,
                "eroare": str(e),
            })

        if index % SAVE_EVERY == 0:
            save_partial(rows)

        time.sleep(DELAY_BETWEEN_JOBS)
```

```
    finally:
        driver.quit()

    save_partial(rows)
    save_final(rows)

    print()
    print("Procesare finalizata.")
    print("Fisiere generate:")
    print(OUTPUT_EXCEL_FINAL)
    print(OUTPUT_CSV_FINAL)
    print(OUTPUT_EXCEL_PARTIAL)
    print(OUTPUT_CSV_PARTIAL)


if __name__ == "__main__":
    main()
```