



**UNIVERSITATEA TEHNICĂ**  
**DIN CLUJ-NAPOCA**

## **PROIECTARE SOFTWARE**

### **Documentație proiect final**

*Grădina Botanică*

*Arhitectura Client/Server, aplicație web, cu protocol HTTP*

Student: Adela Iosif

Grupa: 30233

2023-2024

## Cuprins

1.	Introducere .....	3
1.1.	Obiectiv .....	3
1.2.	Cerințe .....	3
1.3.	Enunțul problemei .....	3
2.	Analiza problemei .....	4
2.1.	Diagrama cazurilor de utilizare.....	4
2.2.	Diagrame de activități .....	5
3.	Proiectare.....	10
3.1.	Diagrame de clase .....	10
3.1.1.	Diagramă de clase – aplicația server.....	10
3.1.2.	Diagramă de clase – aplicația client .....	12
3.2.	Diagrama entitate-relație .....	12
3.3.	Diagrame de secvență .....	13
4.	Implementare.....	18
4.1.	Instrumente utilizate .....	18
4.2.	Șabloane de proiectare .....	19
4.2.1.	Șablonul creațional Singleton .....	19
4.2.2.	Șablonul structural Proxy.....	19
4.2.3.	Șablonul comportamental Strategy .....	19
4.2.4.	Șablonul comportamental Template Method.....	19
4.2.5.	Șablonul creațional Factory.....	20
4.3.	Descrierea aplicației.....	20

## 1. Introducere

### 1.1. Obiectiv

Obiectivul acestui proiect este familiarizarea cu șablonul arhitectural Client/Server, cu șabloanele arhitecturale orientate pe servicii (SOA) și cu șabloanele de proiectare.

Pentru persistența informației se va utiliza o bază de date relațională (SQL Server, MySQL, etc.).

### 1.2. Cerințe

Transformați aplicația implementată la tema 3 într-o aplicație client/server astfel încât să utilizați minim 5 șabloane de proiectare (minim un șablon de proiectare creațional, un șablon de proiectare comportamental și un șablon de proiectare structural) și o arhitectură orientată pe servicii SOA pentru comunicare între aplicația server și aplicația client.

- ❖ În faza de analiză se va realiza diagrama cazurilor de utilizare și diagramele de activități pentru fiecare caz de utilizare (Observație: numărul diagramelor de activități trebuie să fie egal cu numărul de cazuri de utilizare din diagrama cazurilor de utilizare).
- ❖ În faza de proiectare se vor realiza:
  - 2 diagrame de clase corespunzătoare aplicației soft server și aplicației soft client respectând principiile DDD și folosind o arhitectură orientată pe servicii (SOA) și minim 5 șabloane de proiectare;
  - diagrama entitate-relație corespunzătoare bazei de date;
  - diagrame de secvență corespunzătoare tuturor cazurilor de utilizare (Observație: numărul diagramelor de secvență trebuie să fie cel puțin egal cu numărul de cazuri de utilizare din diagrama cazurilor de utilizare).
- ❖ În faza de implementare se va scrie cod pentru îndeplinirea tuturor funcționalităților precizate de diagrama cazurilor de utilizare utilizând:
  - proiectarea dată de diagrama de clase;
  - unul dintre următoarele limbaje de programare: C#, C++, Java, Python.
- ❖ Finalizarea temei va consta în predarea unui director ce va cuprinde:
  - Un fișier cu diagramele UML realizate;
  - Baza de date;
  - Aplicația soft;
  - Documentația (minim 20 pagini) - un fișier care cuprinde:
    - numele studentului, grupa;
    - enunțul problemei;
    - instrumente utilizate;
    - justificarea limbajului de programare ales;
    - descrierea diagramelor UML;
    - descrierea aplicației.

### 1.3. Enunțul problemei

Dezvoltați o **aplicație client/server** care poate fi utilizată într-o **grădină botanică**. Aplicația va avea 3 tipuri de utilizatori: vizitator al grădinii botanice, angajat al grădinii botanice și administrator.

Utilizatorii de tip **vizitator** pot efectua următoarele operații fără autentificare:

- ❖ Vizualizarea listei tuturor plantelor din grădina botanică sortată după tip și specie (vizualizarea include și redarea unor imagini cu toate plantele din grădina botanică; între 1 și 3 imagini pentru fiecare plantă);
- ❖ Filtrarea listei plantelor după următoarele criterii: tip, specie, plante carnivore, zona grădină botanică;
- ❖ Căutarea unei plante după denumire.

Utilizatorii de tip **angajat** al grădinii botanice pot efectua următoarele operații după autentificare:

- ❖ Toate operațiile permise utilizatorilor de tip vizitator;
- ❖ Operații CRUD în ceea ce privește persistența plantelor din grădina botanică;
- ❖ Salvare liste cu plantele în mai multe formate: csv, json, xml, doc;

- ❖ Vizualizarea unor statistici legate de plantele din grădina botanică utilizând grafice (structură radială, structură inelară, de tip coloană, etc.).

Utilizatorii de tip **administrator** pot efectua următoarele operații după autentificare:

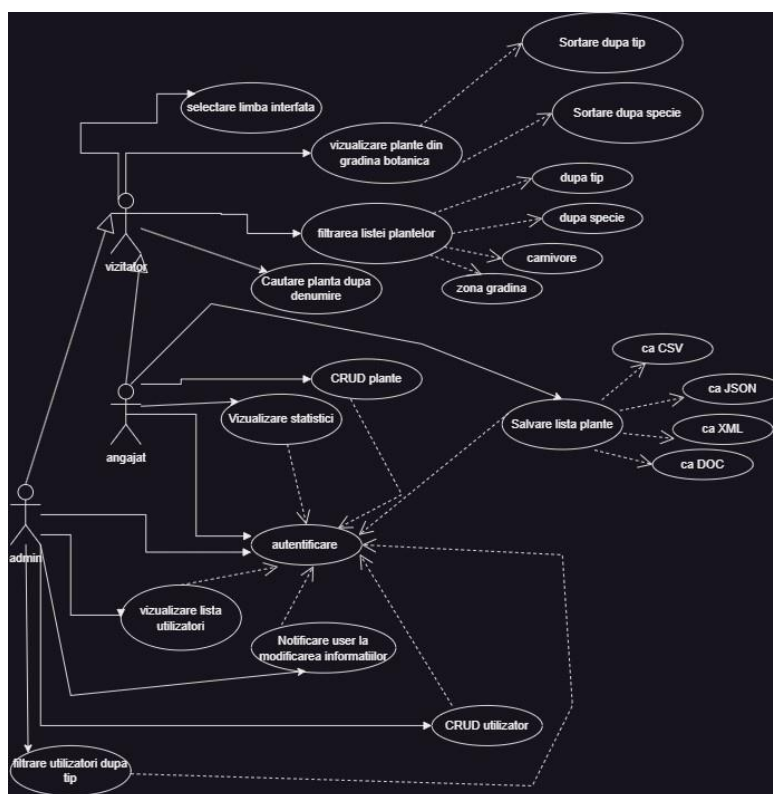
- ❖ Toate operațiile permise utilizatorilor de tip vizitator;
- ❖ Operații CRUD pentru informațiile legate de utilizatorii care necesită autentificare;
- ❖ Vizualizarea listei utilizatorilor care necesită autentificare și filtrarea acestora după tipul utilizatorilor;
- ❖ Notificarea fiecărui utilizator care necesită autentificare prin cel puțin 2 variante (email, SMS, WhatsApp, Skype, etc.) la orice modificare a informațiilor de autentificare aferente acelui utilizator.

**Interfața grafică a aplicației va fi disponibilă în cel puțin 3 limbi de circulație internațională.**

## 2. Analiza problemei

### 2.1. Diagrama cazurilor de utilizare

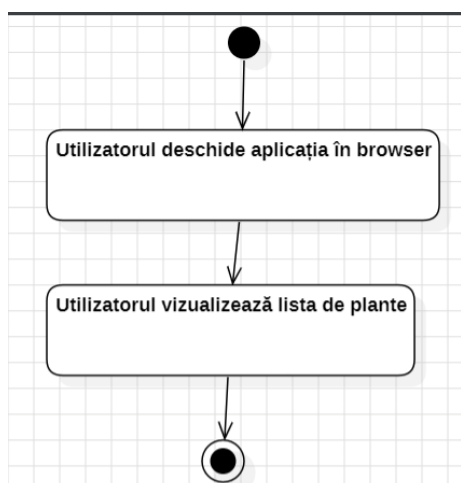
În diagrama de use case (figura 1) se pot identifica cei trei actori: vizitator, angajat și admin. Astfel, vizitatorii sunt persoane nelogate care pot accesa aplicația și pot vizualiza lista plantelor din grădina botanică sortată după tip și specie (acum având ocazia să vizualizeze imagini cu fiecare plantă), dar pot opta și pentru filtrarea listei plantelor după anumite criterii: tip, specie, plante carnivore, zonă grădină, având posibilitatea și de a căuta o plantă după nume. Mai mult, actorii de tip angajat și admin se pot loga, apoi având posibilitatea de efectuare a operațiilor corespunzătoare vizitatorului, alături de altele suplimentare specifice fiecărui rol. Astfel, utilizatorul de tip angajat are ca operații specifice salvarea listei cu plante în diferite formate (csv, json, xml, doc), operațiile CRUD pe plantele din grădina botanică, dar și posibilitatea vizualizării unor statistici despre plante (proportia plantelor carnivore din grădina botanică, numărul de plante din fiecare zonă a grădinii). Administratorul are ca acțiuni specifice rolului operațiile CRUD pe utilizatori, vizualizarea și filtrarea listei după tipul de utilizator, la care se adaugă o funcționalitate mai complexă, anume notificarea utilizatorului prin email și pe WhatsApp de fiecare dată când îi sunt modificate datele de autentificare.



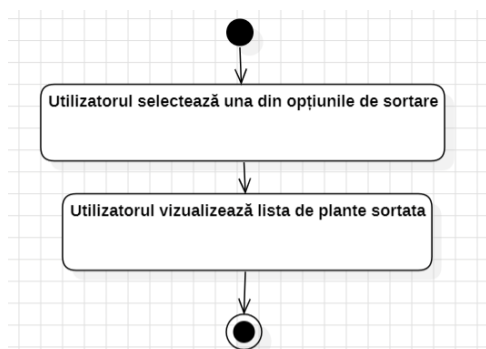
**Figura 1.** Diagrama de use case

## 2.2. Diagrame de activități

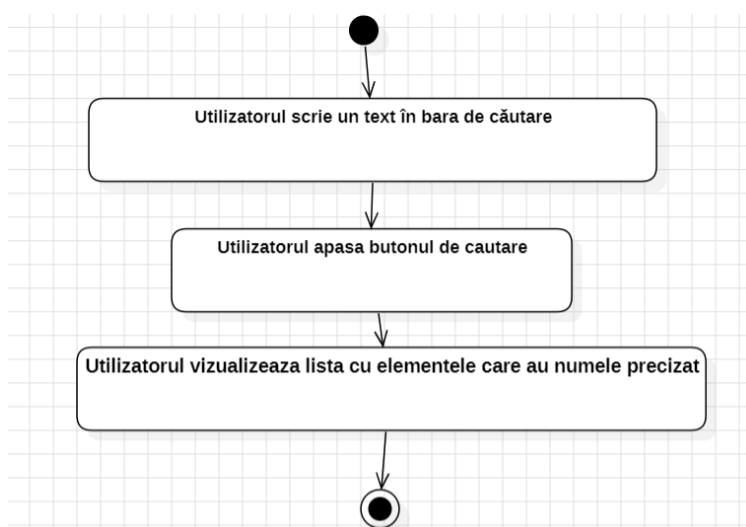
Diagramele de activități sunt utile pentru a descrie fluxurile de lucru dintr-un proces de business, arătând secvența de activități și deciziile care trebuie luate. Astfel, aceste diagrame pot detalia comportamentul unui caz de utilizare, ilustrând fluxul de activități și interacțiunile dintre componentele sistemului (figurile 2-15).



**Figura 2.** Diagramă de activitate – vizualizarea plantelor



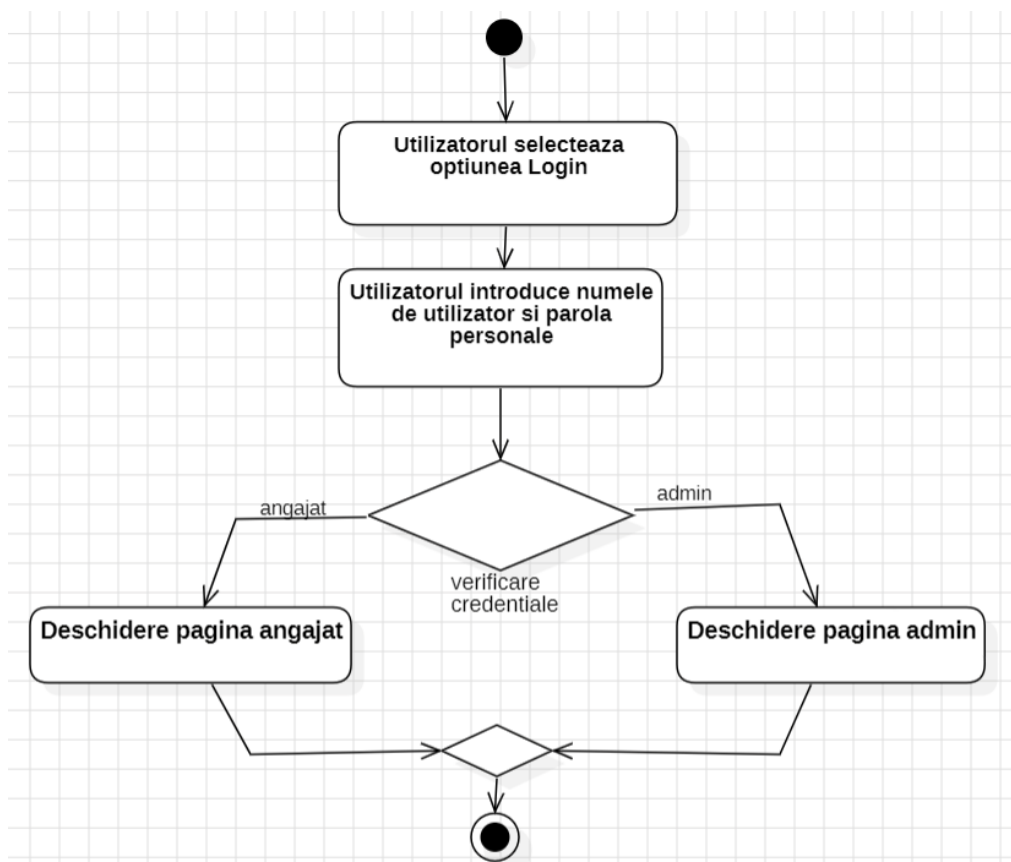
**Figura 3.** Diagramă de activitate – sortarea plantelor după tip sau specie



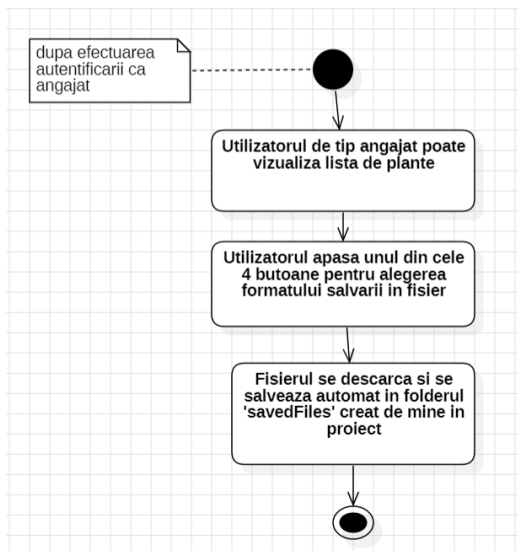
**Figura 4.** Diagramă de activitate – căutarea unei plante după denumire



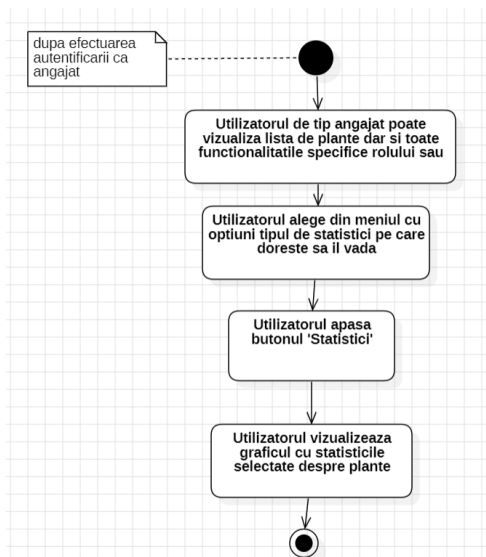
**Figura 5.** Diagramă de activitate – filtrarea plantelor după criteriul specificat



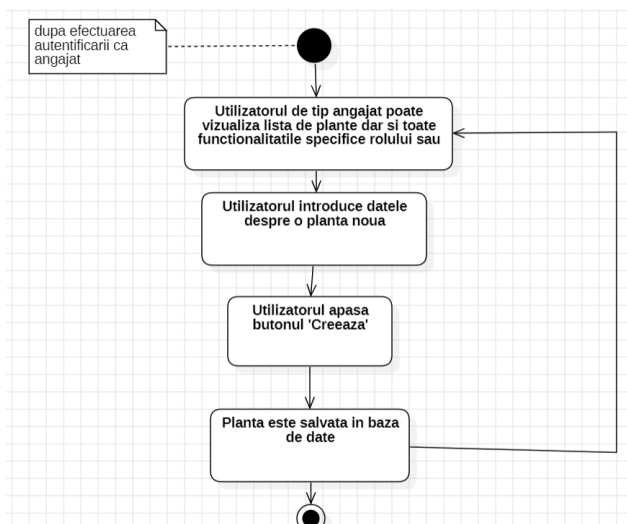
**Figura 6.** Diagramă de activitate – Autentificare



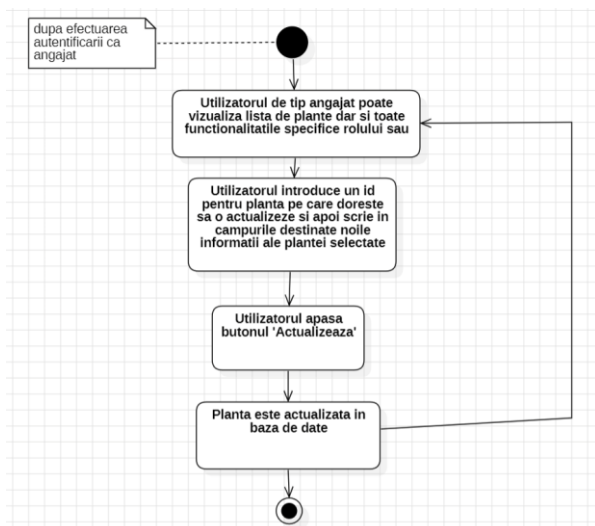
**Figura 7.** Diagramă de activitate – Salvarea unui fișier cu lista de plante, în formatul ales



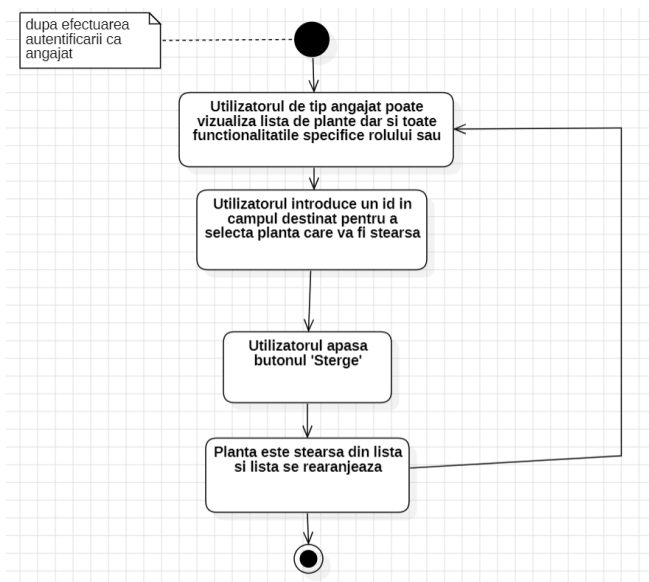
**Figura 8.** Diagramă de activitate – Vizualizarea statisticilor despre plante



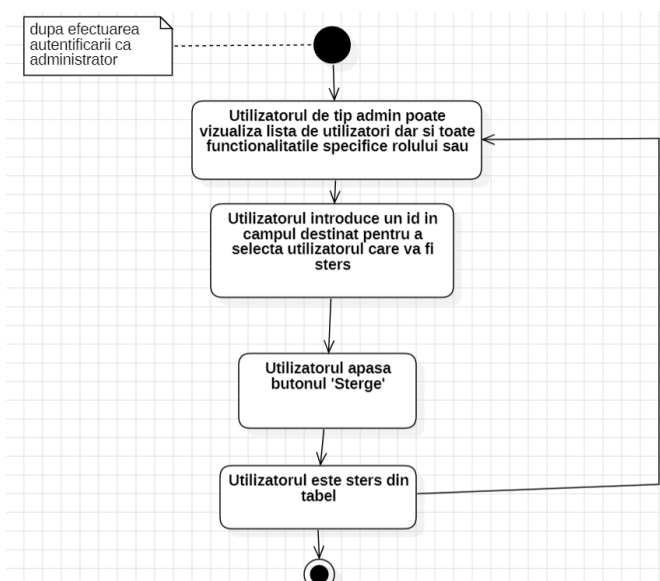
**Figura 9.** Diagramă de activitate – Adăugarea unei plante



**Figura 10.** Diagramă de activitate – Actualizarea unei plante

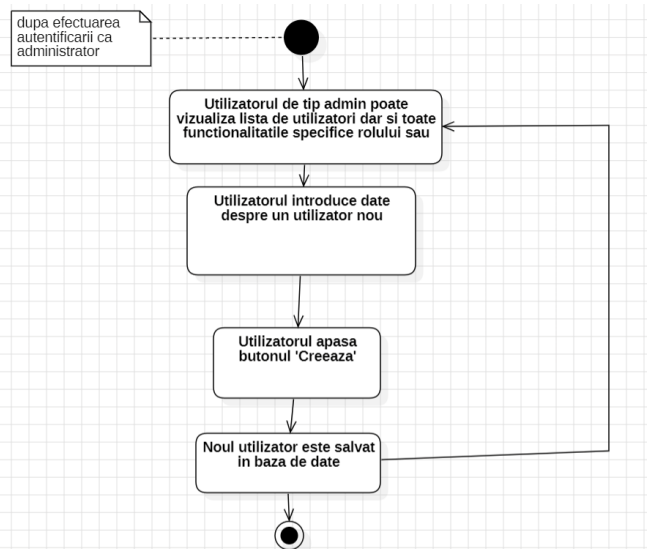


**Figura 11.** Diagramă de activitate – Ștergerea unei plante

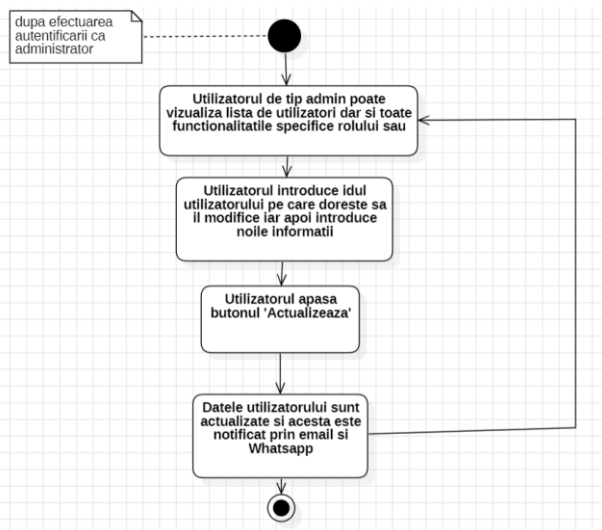


**Figura 12.** Diagramă de activitate – Ștergerea unui utilizator

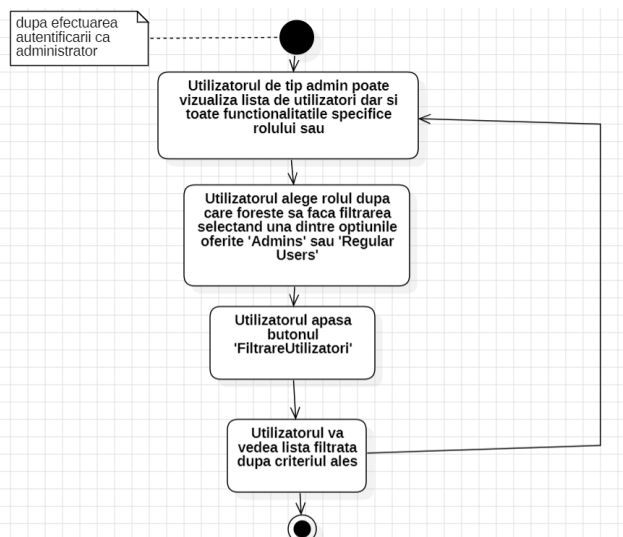




**Figura 13.** Diagramă de activitate – Adăugarea unui utilizator nou



**Figura 14.** Diagramă de activitate – Actualizarea unui utilizator



**Figura 15.** Diagramă de activitate – Filtrare utilizatori după rol

### 3. Proiectare

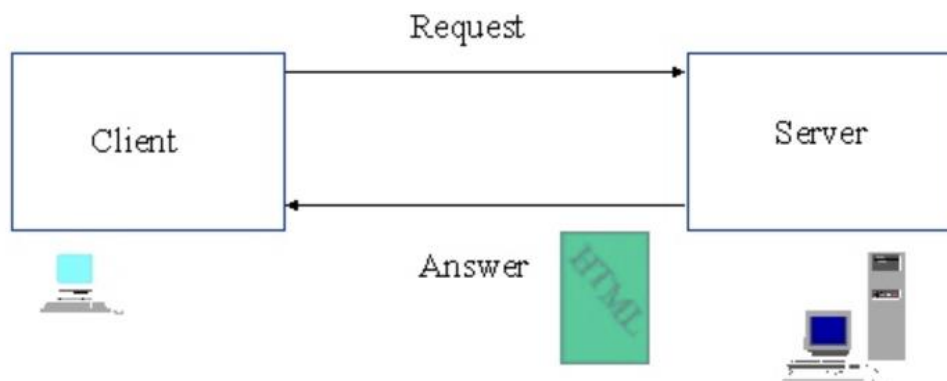
Aplicația web proiectată de mine este una de tip **client/server**, respectând arhitectura SOA pentru comunicarea între aplicația client și aplicația server. În acest stil de proiectare componentele aplicațiilor sunt distribuite ca servicii. Aceste servicii sunt unități de funcționalitate independentă care comunică între ele printr-o rețea, de obicei prin intermediul unor protocoale (de rețea standardizate, ex: http).

Pentru a construi aplicații robuste, scalabile și ușor de întreținut, care să reflecte în mod fidel nevoile și cerințele de business, principiile DDD sunt fundamentale. Astfel, Domain-Driven Design este o abordare de proiectare a software-ului care se concentrează pe modelarea domeniului problemei. De aceea, se folosește un limbaj comun și consistent în întreaga aplicație (atât în codul sursă cât și în interfețelele utilizator). Mai mult, se creează un rich domain model (clasele Plant și User), care să reflecte cât mai bine regulile și comportamentele afacerii. Există clase precum plantServiceFull și UserService care ar putea fi interpretate ca servicii de domeniu (Domain Services) responsabile pentru gestionarea operațiilor complexe legate de plante și utilizatori (utilizate pentru a menține modelul de domeniu curat și coerent).

**SOAP** (Simple Object Access Protocol) este un protocol de mesagerie bazat pe XML, folosit pentru schimbul de informații structurate între aplicații printr-o rețea, în principal folosind HTTP/HTTPS. În proiectul meu, SOAP este utilizat pentru a crea și consuma servicii web: attributele [WebService] și [WebServiceBinding] specifică faptul că metodele expuse în clasă sunt accesibile ca metode SOAP (în clasele *userService* și *plantServiceFull*); apelurile către serviciile web sunt făcute folosind proxy-uri SOAP generate automat (**localhost1.plantServiceFull** și **localhostU.userService**).

#### 3.1. Diagrame de clase

Soluția mea utilizează protocolul http (figura 16), care funcționează, pe scurt, astfel: clientul inițiază comunicarea (o cerere HTTP către server); serverul primește cererea, o procesează și generează un răspuns; clientul primește răspunsul și afișează conținutul utilizatorului.



**Figura 16.** Arhitectura client-server, cu protocolul HTTP

##### 3.1.1. Diagramă de clase – aplicația server

Pachetul service conține clasele necesare implementării tuturor operațiilor care au loc pe backend (figura 17), pe server, și vor fi apelate de către aplicația client prin intermediul comenzilor HTTP. Astfel, acest pachet este parte din stratul de server al aplicației și furnizează funcționalitățile necesare pentru a procesa cererile primite de la clienți și pentru a răspunde în mod adecvat acestora.



### 3.1.2. Diagramă de clase – aplicația client

Aplicația client (figura 19) oferă o interfață prietenoasă utilizatorilor pentru a gestiona și explora informațiile legate de plante și utilizatori în cadrul sistemului. Ea aduce o serie de funcționalități utile, cum ar fi sortarea, filtrarea și gestionarea utilizatorilor, pentru a oferi o experiență eficientă și plăcută utilizatorilor.

Pentru a comunica cu serverul, aplicația client folosește un API de servicii web pentru a efectua operațiile de listare, filtrare și gestionare a datelor.

Ea oferă o serie de funcționalități precum: listarea plantelor și a utilizatorilor, filtrarea și căutarea plantelor, interfață multilingvă, operații CRUD pe plante sau utilizatori, salvarea listei cu plante în diferite formate, trimiterea de notificări către utilizator la efectuarea actualizării datelor sale, și altele.

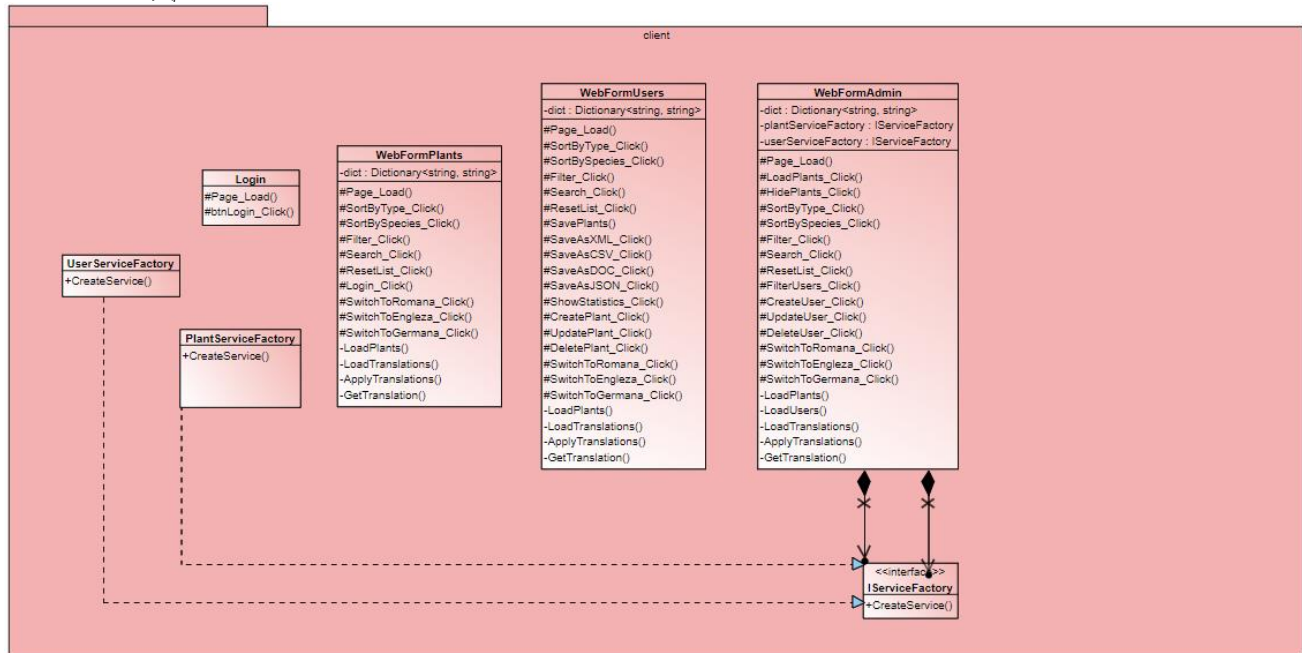


Figura 19. Diagramă de clase – clientul

### 3.2. Diagrama entitate-relație

Diagrama bazei de date este neschimbată. Acest tip de diagramă (figura 20) este un model vizual care arată conexiunile entităților din baza de date. Astfel, baza de date este formată din două tabele: Users și Plant. Tabela Users este folosită pentru păstrarea datelor fiecărui utilizator (nume de utilizator, parolă, atribut pentru verificarea statutului de administrator). De asemenea, tabela Plant stochează toate plantele prezente în grădina botanică, cu atributele lor: nume, specie, tipul de plantă, atribut care reține dacă planta este carnivoră sau nu, locația plantei în cadrul grădinii, dar și câte o imagine pentru fiecare plantă.

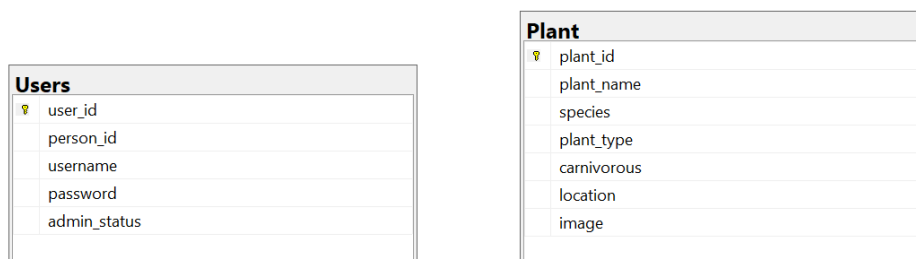
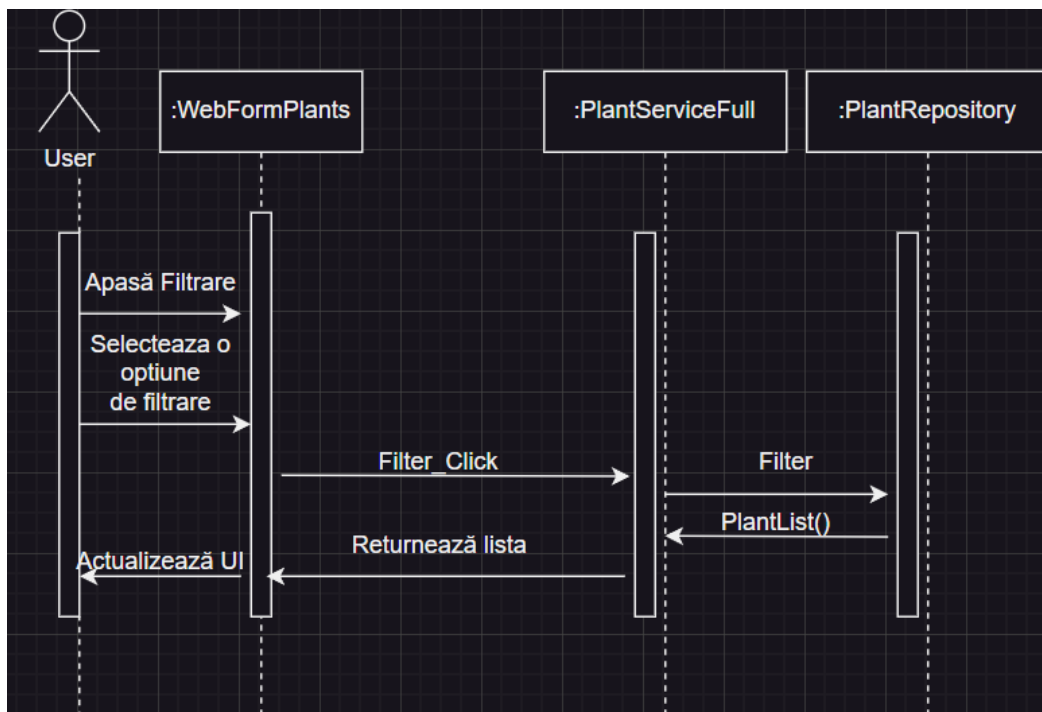


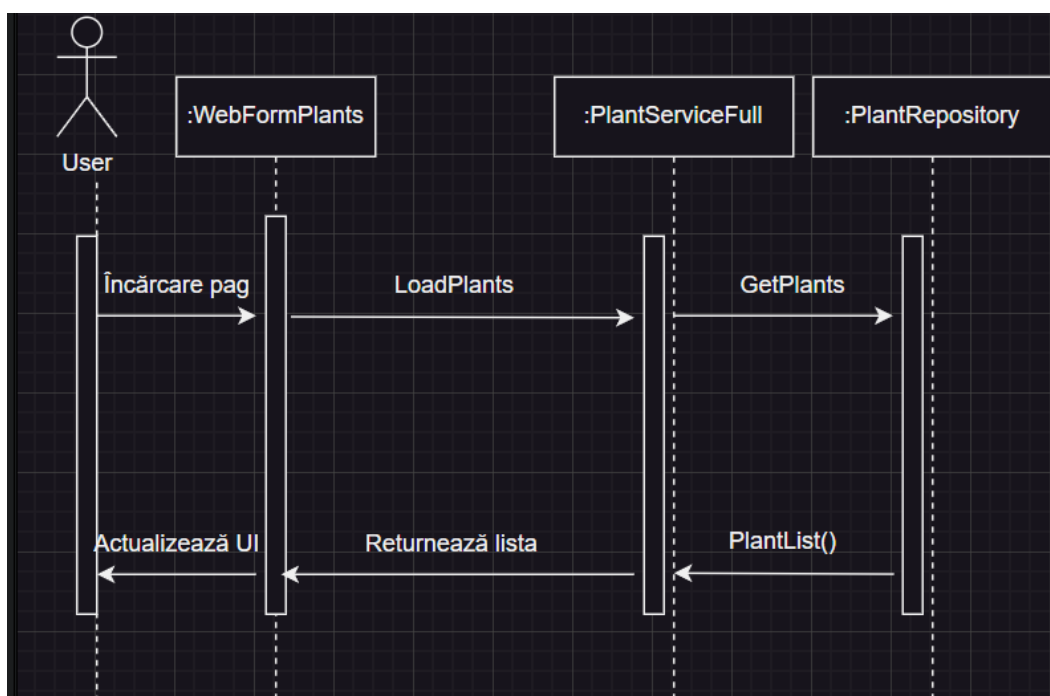
Figura 20. Diagrama entitate-relație

### 3.3. Diagrame de secvență

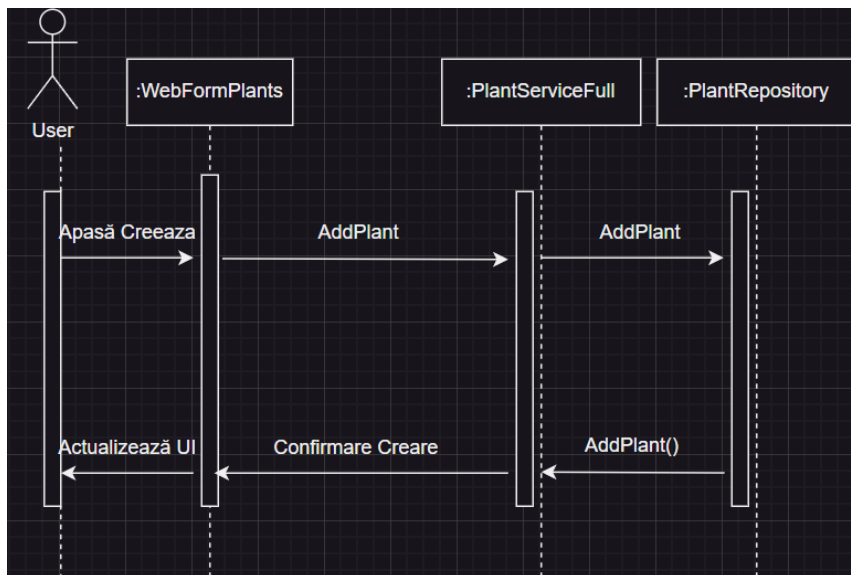
Diagramele de secvență (figurile 21-36) oferă o reprezentare grafică a modului în care diferite obiecte și componente interacționează între ele într-un anumit scenariu sau flux de lucru. Ele sunt folosite pentru a ilustra ordinea în care mesajele sunt trimise între obiecte într-o secvență de acțiuni specifice. Ele arată ordinea în care obiectele comunică între ele și schimbă mesaje pentru a îndeplini anumite sarcini.



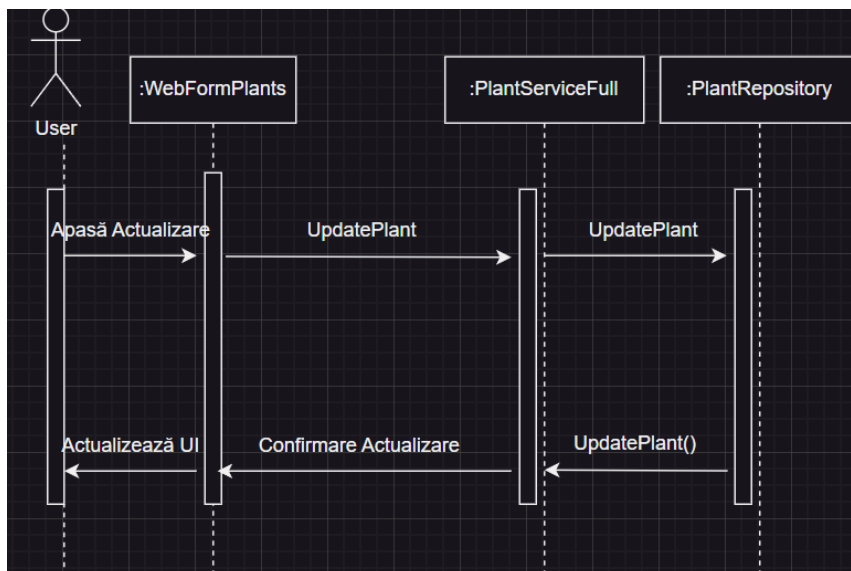
**Figura 21.** Diagramă de secvență – Filtrare plante



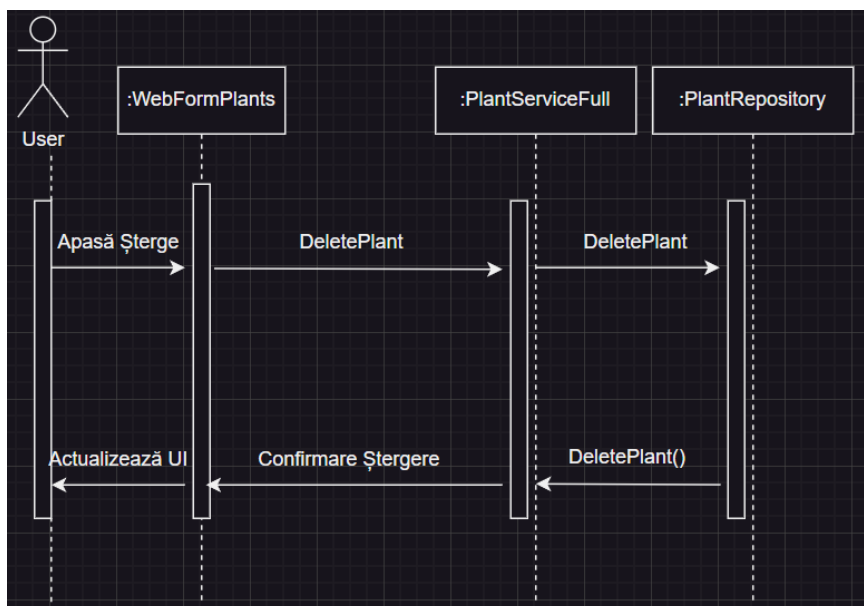
**Figura 22.** Diagramă de secvență – Vizualizare plante



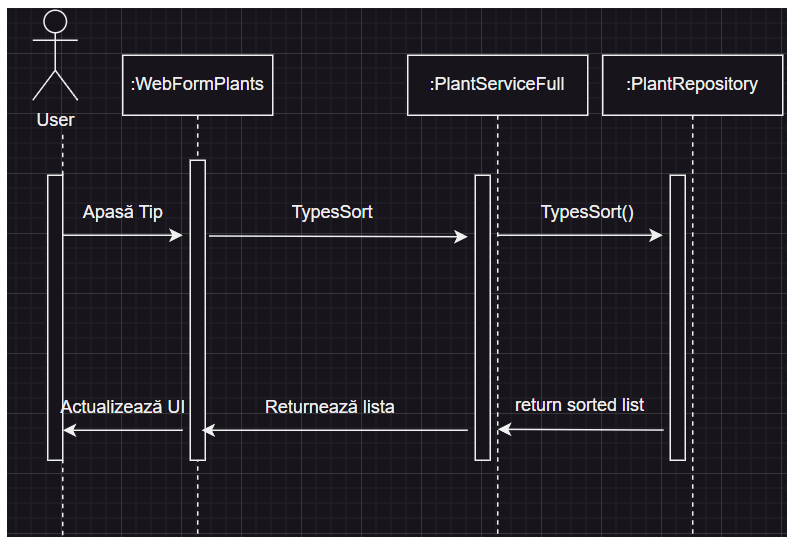
**Figura 23.** Diagramă de secvență – Crearea unei noi plante



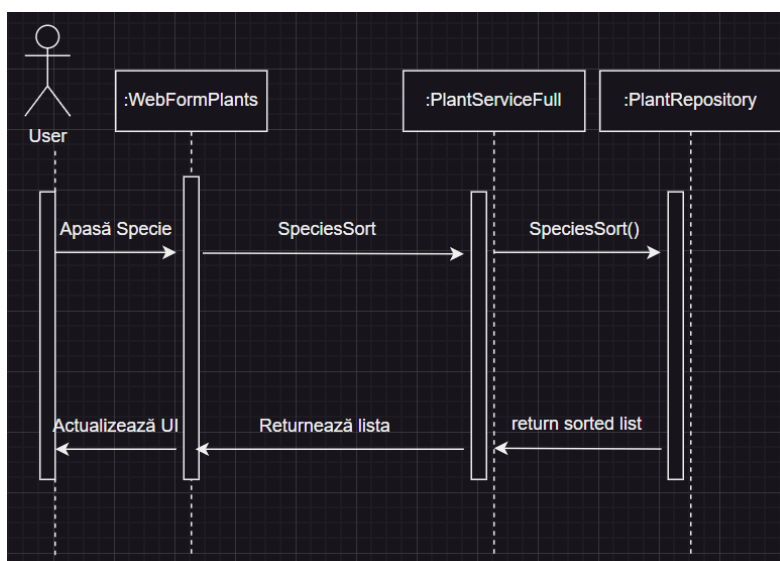
**Figura 24.** Diagramă de secvență – Actualizare plantă



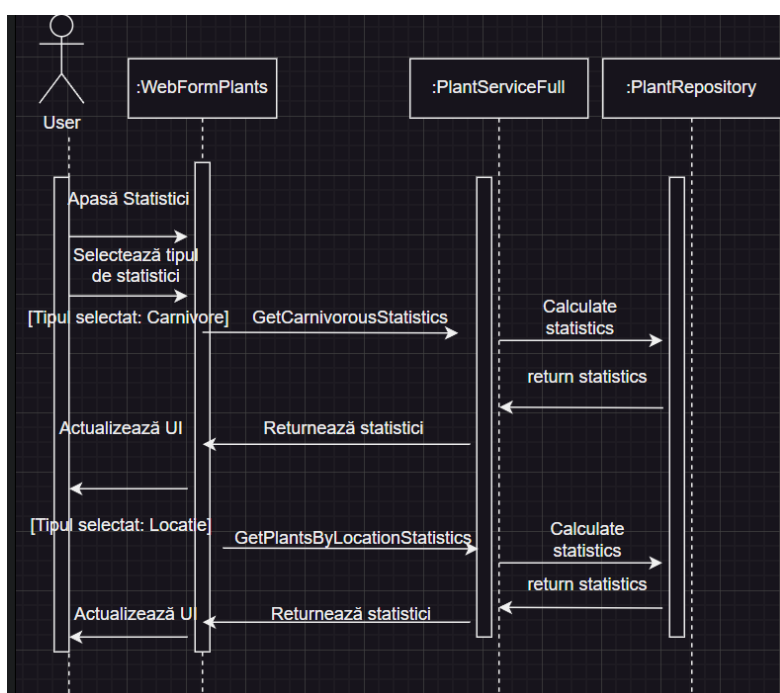
**Figura 25.** Diagramă de secvență – Ștergerea unei plante



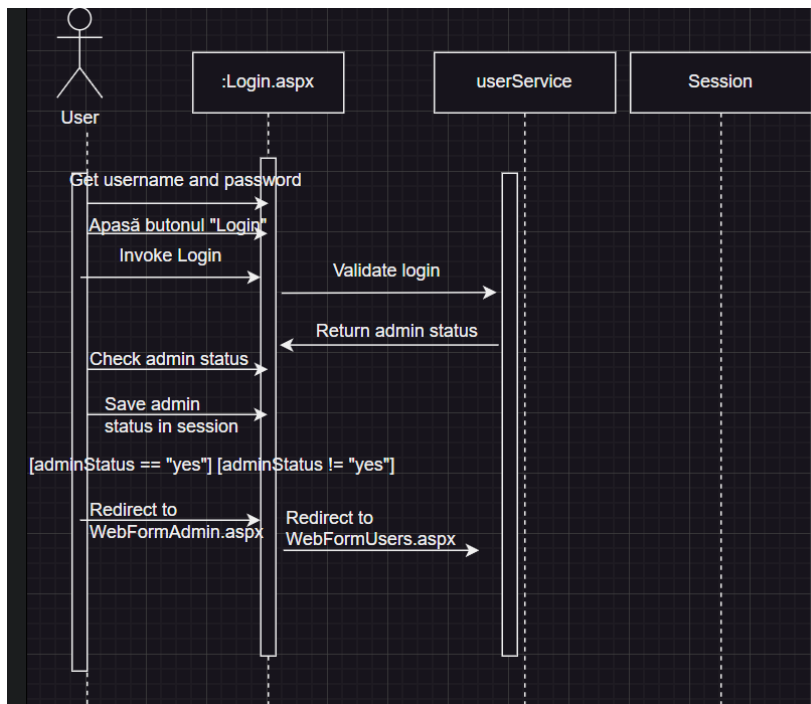
**Figura 26.** Diagramă de secvență – Sortare plante după tip



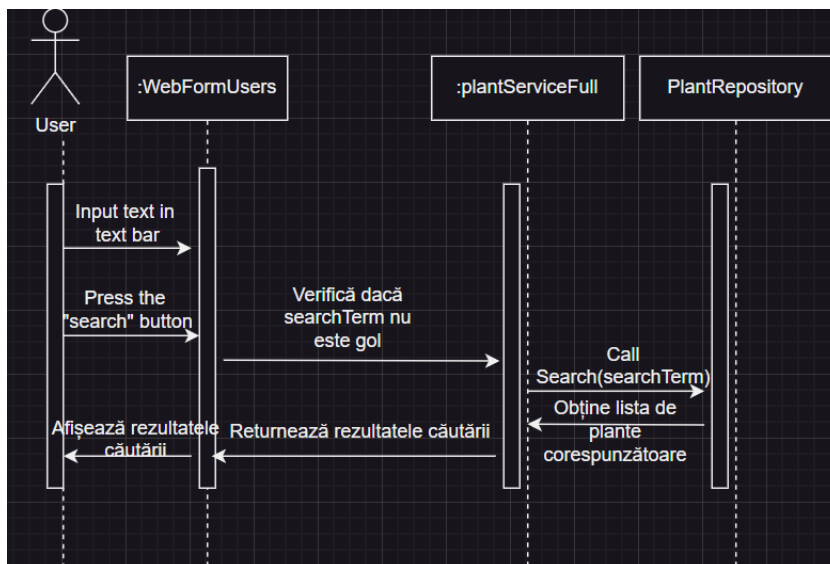
**Figura 27.** Diagramă de secvență – Sortare plante după specie



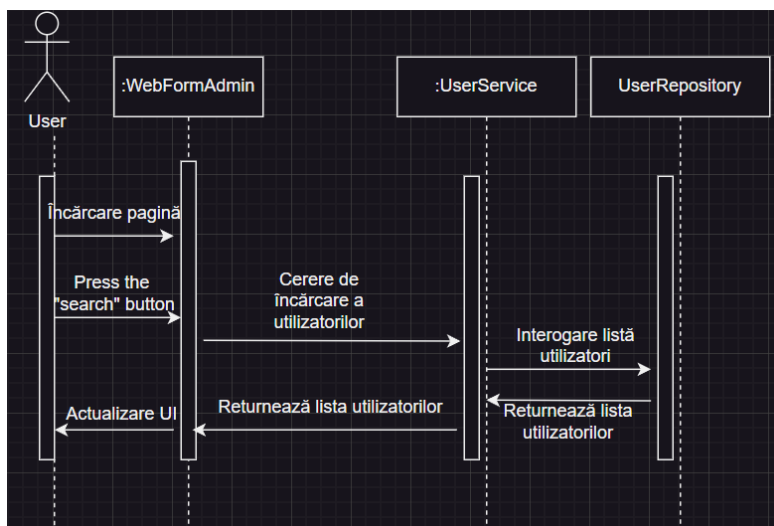
**Figura 28.** Diagramă de secvență – Statistici plante



**Figura 29.** Diagramă de secvență – Autentificare

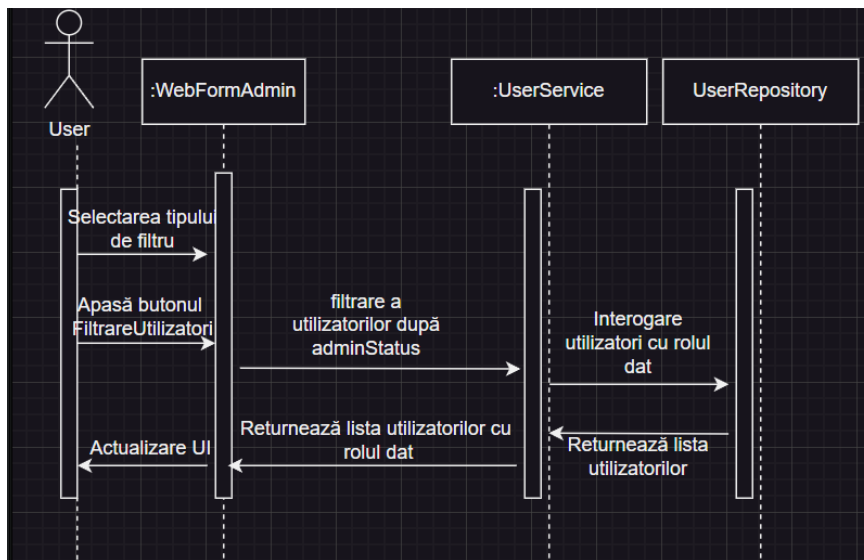


**Figura 30.** Diagramă de secvență – Căutare plantă după nume

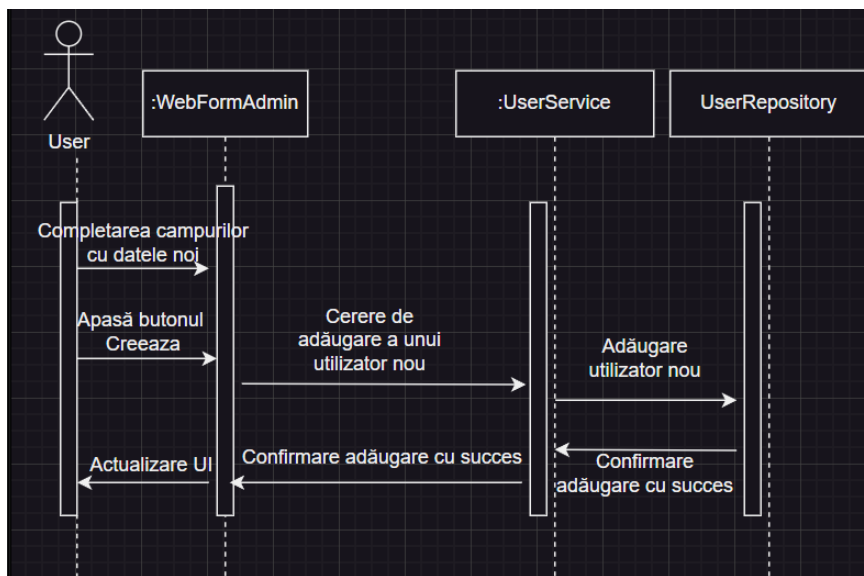


**Figura 31.** Diagramă de secvență – Vizualizare utilizatori

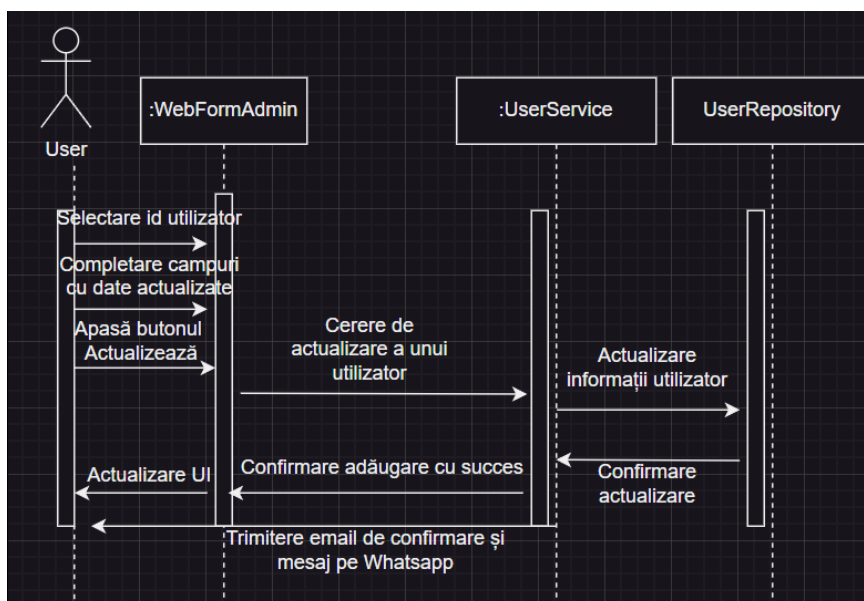




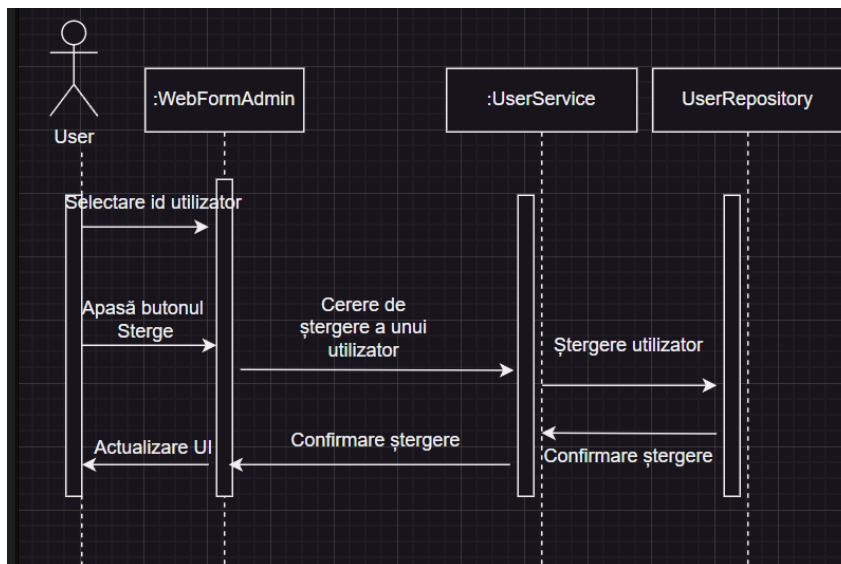
**Figura 32.** Diagramă de secvență – Filtrare Utilizatori după rol



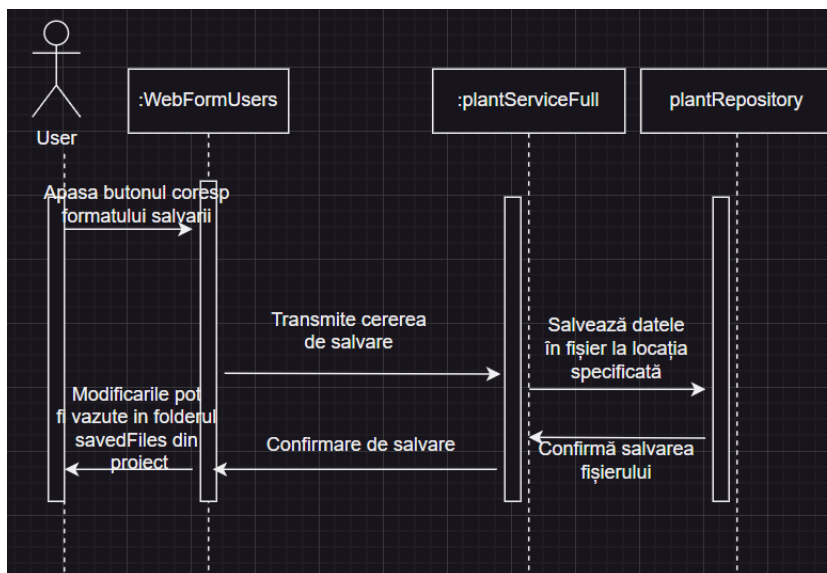
**Figura 33.** Diagramă de secvență – Crearea unui nou utilizator



**Figura 34.** Diagramă de secvență – Actualizarea unui utilizator



**Figura 35.** Diagramă de secvență – Ștergerea unui utilizator



**Figura 36.** Diagramă de secvență – Salvarea listei de plante în fișier

## 4. Implementare

### 4.1. Instrumente utilizate

În soluția prezentată se utilizează:

- Limbajul de programare C# (în IDE-ul Visual Studio)
- Framework-ul ASP.NET
- HTML, CSS
- Sistemul de gestionare de baze de date relaționale SQL Server
- StarUML (diagrame de activitate)
- Visual Paradigm (diagrame de pachete), draw.io (diagrame de secvență)

C# este un limbaj de programare de bază pentru dezvoltarea aplicațiilor în .NET Framework, iar ASP.NET este un framework puternic pentru dezvoltarea aplicațiilor web în .NET. Acestea oferă o gamă largă de funcționalități și biblioteci care facilitează dezvoltarea aplicațiilor web și gestionarea comunicării client-server. În .NET există o serie de biblioteci care facilitează lucrul cu HTTP, cum ar fi HttpClient sau HttpRequest, care permit aplicației să facă cereri HTTP către alte servere și să primească răspunsuri.

HTML am utilizat pentru structura și marcarea conținutului paginii web. Acesta definește elemente precum titluri, butoane, casete de text, liste și tabele, iar CSS este folosit în proiect pentru stilizarea aspectului paginii web.

## 4.2. Șabloane de proiectare

Șabloanele de proiectare oferă o modalitate standardizată de a aborda anumite probleme, promovând practici de proiectare și codare mai bune.

### 4.2.1. Șablonul creațional Singleton

Singleton este un șablon de proiectare creațional care asigură existența unei singure instanțe a unei clase în cadrul unei aplicații și oferă acces la acea instanță de oriunde din cod.

În proiectul meu, am implementat acest design creațional în clasa **Repository**. Clasa are un constructor privat (`private Repository(string dbName)`) și o metodă statică (`public static Repository GetInstance(string dbName)`) care returnează o instanță a clasei **Repository**. De asemenea, am o variabilă statică privată (`private static Repository singleInstance`) care este inițializată cu valoarea null. Metoda `GetInstance` este utilizată pentru a obține o instanță a clasei **Repository** (dacă variabila `singleInstance` este null, atunci este creată o nouă instanță a clasei **Repository** cu numele bazei de date specificate, altfel este returnată instanța existentă). În esență, atunci când se dorește folosirea clasei **Repository**, trebuie apelată metoda `GetInstance(string dbName)` și furnizat numele bazei de date. Aceasta va garanta că se va primi întotdeauna aceeași instanță a clasei **Repository**, indiferent de câte ori se va folosi în aplicație.

### 4.2.2. Șablonul structural Proxy

Șablonul structural Proxy este un design pattern care permite controlul accesului la un obiect, înlocuindu-l cu un obiect proxy care acționează ca o reprezentare intermediară a acestuia. Scopul principal al șablonului Proxy este de a oferi un substitut sau o înlocuire pentru alt obiect, pentru a controla accesul la acesta sau pentru a adăuga funcționalități suplimentare.

În proiectul meu, am implementat acest design structural prin intermediul clasei **UserServiceProxy**, ea servind ca un intermediar între consumatorul serviciului și implementarea reală a serviciului. Interfața **IUserService** reprezintă contractul pentru serviciul web. Această interfață conține metodele pe care consumatorul le poate apela. Clasa **userService** implementează interfața **IUserService** și conține logica reală a serviciului web.

### 4.2.3. Șablonul comportamental Strategy

Șablonul Strategy este un șablon de proiectare comportamental care permite definirea unei familii de algoritmi, încapsularea fiecărui algoritm și tratarea acestora ca obiecte interschimbabile.

În proiectul meu, am implementat acest design comportamental folosind mai multe clase. Interfața **IFileSaveStrategy** definește operația **Save**, pe baza căreia toate strategiile de salvare în fișier vor fi implementate. Ea oferă un contract comun pentru toate strategiile. Clasa **FileSaveContext** acționează ca și context pentru utilizarea strategiei. Ea acceptă o strategie specifică prin intermediul metodei **SetFileSaveStrategy** și apoi utilizează acea strategie pentru a executa operația **SaveFile**. Implementările specifice ale strategiilor se regăsesc în clasele: **CsvFileSaveStrategy**, **XmlFileSaveStrategy**, **DocFileSaveStrategy**, **JsonFileSaveStrategy** – aceste clase implementează interfața **IFileSaveStrategy** și definesc algoritmi specifici pentru salvarea în formatul specific. În cadrul serviciului **plantServiceFull** am utilizat **FileSaveContext** pentru a seta strategia corespunzătoare în funcție de cerințele de salvare și apoi am folosit contextul pentru a salva plantele în diferite formate de fișiere.

### 4.2.4. Șablonul comportamental Template Method

Acest șablon este utilizat pentru definirea unei structuri algoritmice fixe într-o clasă de bază, cu anumite metode care pot fi suprascrise în clasele derivate. În esență, acest șablon definește scheletul algoritmului în clasa de bază și permite subclasselor să își modifice comportamentul prin suprascrierea anumitor metode. Acest lucru face codul mai modular și mai ușor de întreținut, permițând adăugarea ușoară a noi formate de fișiere pentru salvare în viitor.

În proiectul meu, am implementat acest design comportamental în clasa **WebFormUsers**, folosind metoda **SavePlants**. Această metodă este responsabilă pentru definirea structurii algoritmice generale pentru salvarea plantelor în diferite formate de fișiere. Această metodă

primește ca parametru o funcție (**Action saveMethod**), care reprezintă metoda specifică de salvare care urmează să fie executată. Metodele **SaveAsXML\_Click**, **SaveAsCSV\_Click**, **SaveAsDOC\_Click**, **SaveAsJSON\_Click** sunt apelate în momentul în care utilizatorul dorește să salveze plantele în diferite formate de fișiere. În interiorul acestor metode, se apelează **SavePlants**, oferindu-i ca argument metoda specifică de salvare corespunzătoare formatului de fișier dorit.

#### 4.2.5. Șablonul creațional Factory

Scopul șablonului este de a oferi o modalitate de creare a unui obiect fără a expune logica de creare a obiectului direct în codul clientului. Acest lucru aduce beneficii precum creșterea flexibilității, creșterea ușurinței de întreținere și reducerea cuplării între componente. Am folosit Factory pentru a abstractiza crearea de obiecte de serviciu pentru plante și utilizatori.

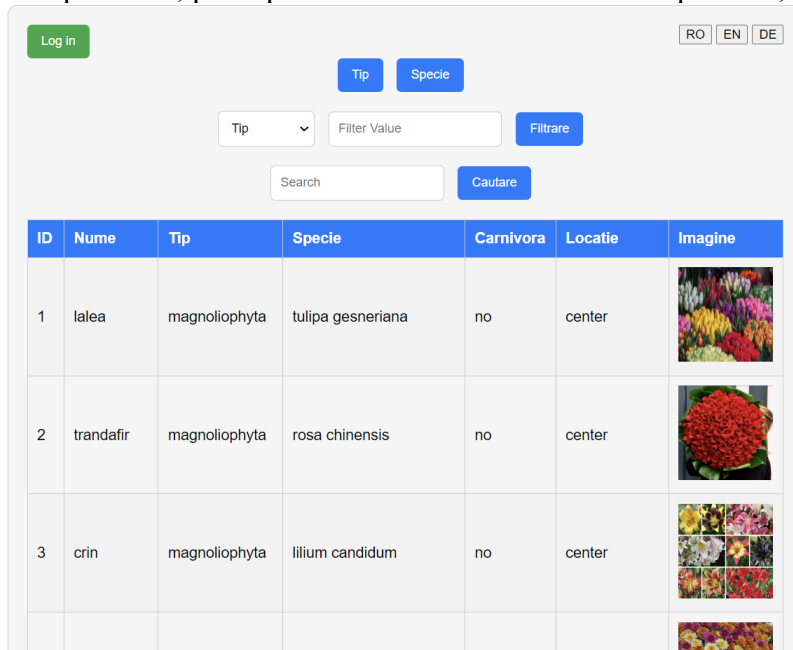
Asfel, interfața **IServiceFactory** servește ca un contract pentru toate fabricile de servicii. Această interfață conține o singură metodă, **CreateService()**, care este responsabilă pentru crearea și returnarea unui serviciu specific. Clasele concrete **UserServiceFactory** și **PlantServiceFactory** implementează interfața **IServiceFactory**. Aceste clase sunt responsabile pentru crearea obiectelor specifice de serviciu pentru utilizatori și plante. Am utilizat acestea în clasa **WebFormAdmin**, unde am declarat două variabile private, **plantServiceFactory** și **userServiceFactory**, de tipul **IServiceFactory**. În metoda **Page\_Load**, ai inițializat aceste fabrici de servicii. Apoi, în metoda **Page\_Load**, am folosit **userServiceFactory** pentru a crea un serviciu de utilizatori și am apelat metoda **GetUsers()** pentru a obține lista de utilizatori. Același lucru este aplicat și pentru serviciul de plante în metoda **LoadPlants()**.




În concluzie, prin utilizarea șablonului Factory, am izolat logica de creare a obiectelor de serviciu în clase separate, ceea ce face codul mai modular și mai ușor de întreținut. Acest lucru face mai ușoară schimbarea sau extinderea serviciilor în viitor, deoarece modificările necesare trebuie făcute doar în clasele de fabrică, fără a afecta codul clientului.

#### 4.3. Descrierea aplicației

La deschiderea aplicației se va afișa pagina de start (figura 37), mai exact cea în care utilizatorul este doar vizitator, acesta nefiind înregistrat în cont (nu îi este atribuit niciun rol). Astfel, un vizitator poate să sorteze plantele după tip sau specie, apăsând unul din butoanele corespunzătoare. Mai mult, plantele din grădina botanică pot fi vizualizate în listă, împreună cu atributele lor: id, nume, tip, specie, dacă este carnivoră sau nu, zona din grădina botanică unde se găsește, o imagine reprezentativă plantei.

De asemenea, pagina se va deschide implicit în limba română, dar există opțiunea de a alege limba preferată, prin apăsarea unuia dintre butoanele specifice, localizate în dreapta-sus.



ID	Nume	Tip	Specie	Carnivora	Locatie	Imagine
1	lălea	magnoliophyta	tulipa gesneriana	no	center	
2	trandafir	magnoliophyta	rosa chinensis	no	center	
3	crin	magnoliophyta	lilium candidum	no	center	

**Figura 37.** Pagina de start a aplicației (vizitator)

Pagina de vizitator conține și opțiunile de căutare a unei plante după nume, sau filtrare după anumite criterii (tip, specie, zonă, carnivore – trebuie selectat criteriul general, apoi introdus explicit în textbox-ul destinat; figura 38). De asemenea, în subsolul paginii, sub tabel, există și opțiunea de resetare a listei, pentru o utilizare mai fluentă a aplicației.

ID	Nume	Tip	Specie	Carnivora	Locatie	Imagine
6	pin	gymnospermae	pinus pinea	no	left entrance	
8	juniper	gymnospermae	juniperus communis	no	right entrance	

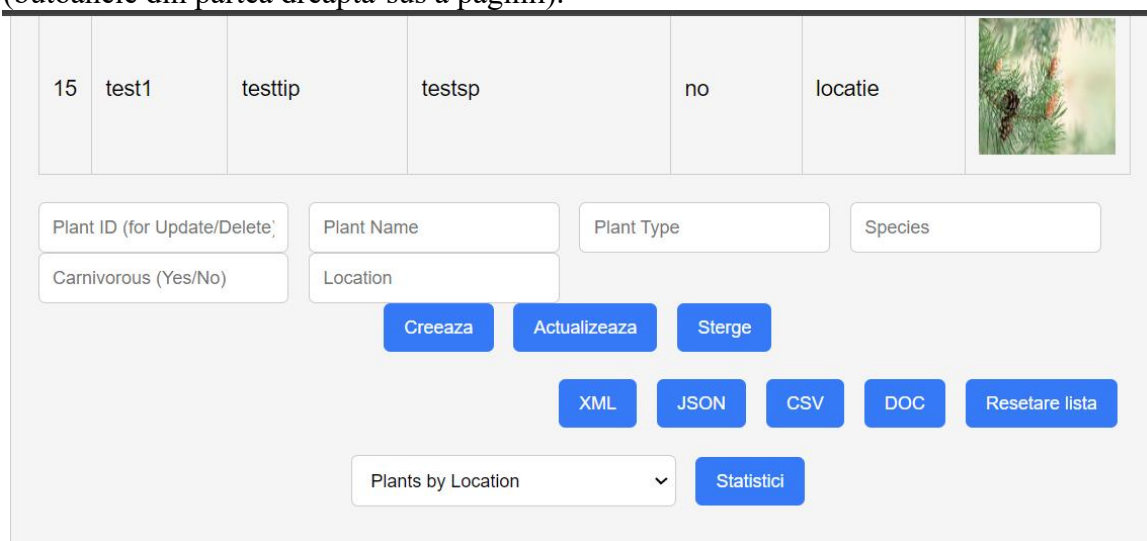
**Figura 38.** Filtrarea plantelor după un anumit criteriu


În pagina de start a aplicației web există și opțiunea de Login, care va deschide o pagină nouă (figura 39), unde se poate face autentificarea (în rolul de angajat sau administrator).

**Figura 39.** Pagina de autentificare

Logarea ca angajat oferă toate funcționalitățile disponibile și unui vizitator, la care se adaugă operații de creare, citire, actualizare, ștergere a plantelor din grădina botanică, posibilitatea de a salva listele cu plante în mai multe formate (csv, json, xml, doc) și vizualizarea unor statistici despre plantele din grădina botanică (figura 40).

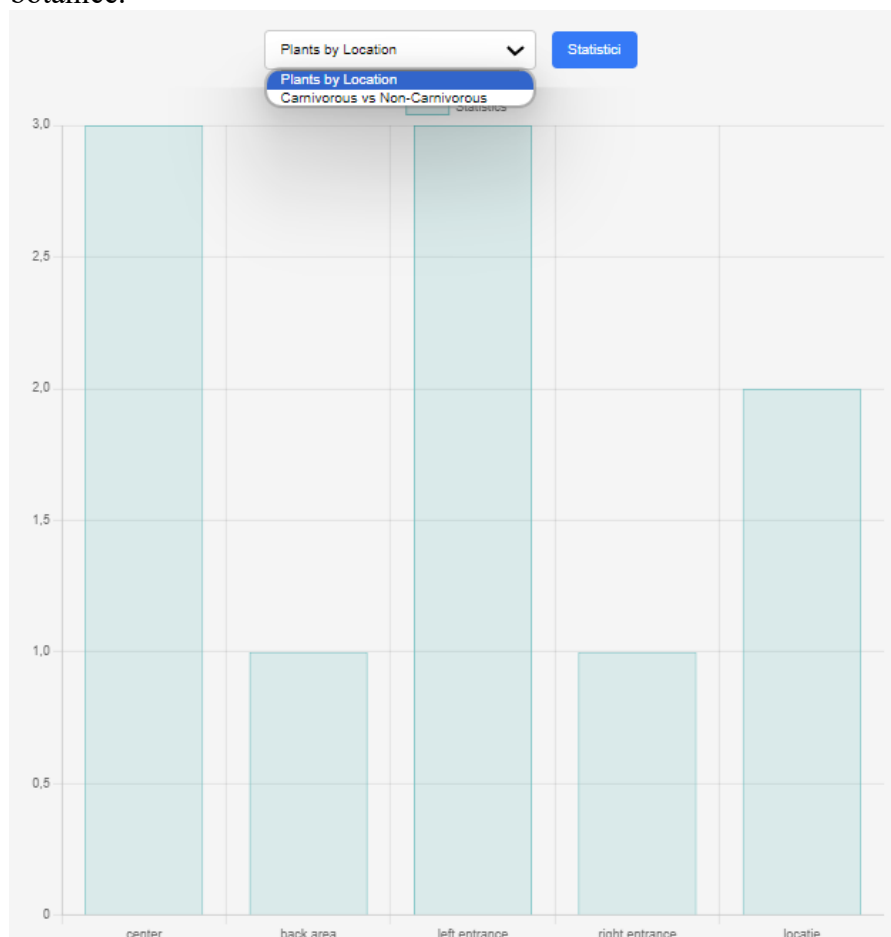
De asemenea, există opțiunea de selectare a limbii preferate: română, engleză sau germană (butoanele din partea dreaptă-sus a paginii).



15	test1	testtip	testsp	no	locatie	
----	-------	---------	--------	----	---------	---

**Figura 40.** Opțiunile suplimentare din pagina de angajat

Astfel, angajatul are opțiunea de vizualizare a unor statistici în ceea ce privește plantele din grădina botanică (figura 41). Am realizat două reprezentări; în prima se vede distribuția plantelor carnivore în grădina botanică, iar în cea de-a doua, numărul de plante din fiecare zonă a grădinii botanice.



**Figura 41.** Statistici despre plantele din grădina botanică

Dacă ne înregistrăm ca admin (figura 42) vom putea efectua operațiile disponibile unui vizitator, dar și operații de creare, citire, actualizare și ștergere a utilizatorilor din baza de date. De asemenea, administratorul va putea vizualiza lista tuturor utilizatorilor și îi va putea filtra după tipul lor (angajați/administratori).

Limba standard în care se va deschide pagina este româna, dar similar și celorlalte pagini ale aplicației, utilizatorul are posibilitatea de a selecta limba favorită (apăsând unul din butoanele specifice din partea de sus a paginii).

RO EN DE

Tip Specie

Type Filter Value Filtrare

All Users FiltrareUtilizatori

Search Cautare

User ID	Username	Password	Admin Status
2	adeliosif@yahoo.com	aaa	no
3	Lucca	lucarus	no
5	admin	admin	yes
6	sara12	1234	no
7	anapop	1112	yes
17	testPassword	testUsername	no
18	Ana_pop	pass1	no
42	user@test.com	testpass1	no
43	user@test2.com	testpass2	no
44	user@test3.com	testpass3	no
48	user@test7.com	testpass7	no
49	user@test4.com	uuuu888	yes

Creeaza Actualizeaza Sterge

User ID (for Update/Delete) Username Password Admin

Incarca lista cu plante Ascunde lista cu plante Resetare lista

**Figura 42.** Pagina de administrator

Pentru a filtra doar tipul dorit de angajat (figura 43), se va alege preferința, utilizând meniul pus la dispoziție (All Users/Admins/Regular Users).

Admins FiltrareUtilizatori

All Users Admins Regular Users

Cautare

User ID	Username	Password	Admin Status
5	admin	admin	yes
7	anapop	1112	yes
49	user@test4.com	uuuu888	yes

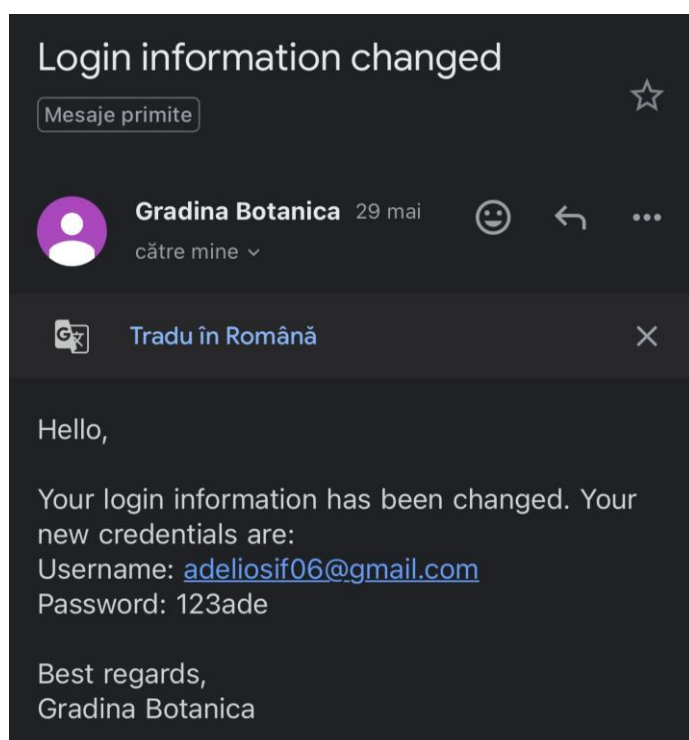
**Figura 43.** Filtrarea angajaților după rol

Pentru a actualiza informațiile despre un utilizator (figura 44), se va introduce id-ul user-ului, noul email (email valid, pe care se va primi notificarea despre schimbarea credențialelor), parola și statusul modificate, dacă este cazul. După apăsarea butonului „Actualizeaza”, se va trimite un email la adresa dată (figura 45), dar și un mesaj pe Whatsapp (figura 46), pentru a informa utilizatorul cu privire la modificarea informațiilor de conectare.

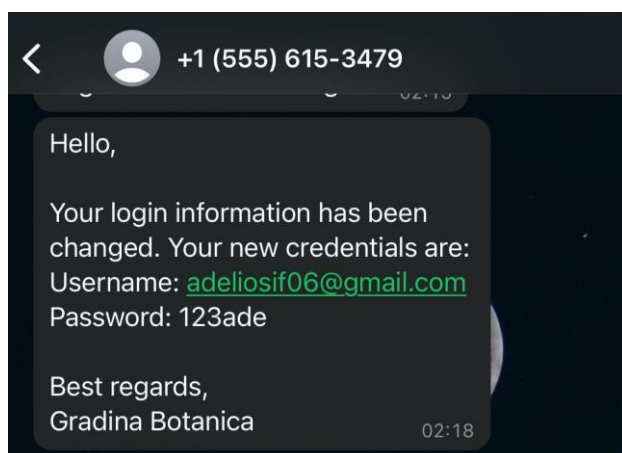
42	user@test.com	testpass1	no
43	adeliosif@yahoo.com	new_pass	yes
44	user@test3.com	testpass3	no
48	user@test7.com	testpass7	no
49	user@test4.com	uuuu888	yes

Creeaza
Actualizeaza
Sterge

**Figura 44.** Actualizarea unui utilizator



**Figura 45.** Notificarea utilizatorului pe email



**Figura 46.** Notificarea utilizatorului pe WhatsApp