UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SPECIALIZAREA: COMPLEMENTE DE ȘTIINȚA CALCULATOARELOR

PROIECTAREA UNEI APLICAȚII CARE RESPECTĂ PATTERN-UL DE PROIECTARE CLIENT – SERVER

GEDDERTH CRISTIAN

CSC

2023

Cuprins

Capitolul 1. Enunțul problemei. 3
Capitolul 2. Instrumente utilizate. 4
Capitolul 3. Justificarea limbajului de programare ales. 5
Capitolul 4. Descrierea diagramelor UML. 6
4.1. Descrierea diagramei entitate – relație. 6
4.2. Descrierea diagramei cazurilor de utilizare. 7
4.3. Descrierea diagramei de clase. 9
4.3.1. Model. 9
4.3.2. Repository. 10
4.3.3. Service 11
4.3.4. Controller. 12
4.3.4. Descrierea diagramei de clase ca întreg. 14
4.3.5. Descrierea aplicației client. 16
4.3.6. Descrierea aplicației client și aplicației server. 17
Capitolul 5. Descrierea aplicației. 18

Capitolul 1. Enunțul problemei.

Dezvoltați o aplicație 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;
- 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, txt;
- 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:

- Operații CRUD pentru informațiile legate de utilizatorii care necesită autentificare;
- Vizualizarea listei utilizatorilor care necesită autentificareși filtrarea acesteia după tipul utilizatorilor.
-

Interfața grafică a aplicației va fi disponibilă în cel puțin 3 limbi de circulație internațională(implicit limba română).

Capitolul 2. Instrumente utilizate.

Pentru realizarea aplicației s-au folosit următoarele instrumente:

- **Star UML:** este un program foarte util și ușor de utilizat pentru a realiza diferite diagrame. În cadrul acestui proiect cu Star UML s-au realizat următoarele diagrame: diagrama de clase, diagrama de utilizare de cazuri și diagrama entitate – relație a schemei bazei de date.
- **IDE-ul IntelliJ IDEA:** este unul dintre cele mai populare și puternice medii de dezvoltare integrate pentru programatorii Java. IDE-ul oferă o varietate de funcții utile, cum ar fi: completarea automata a codului, refactorizare, generare automata de cod (constructori, gettere, settere), proiectarea interfețelor grafice prin drag & drop de elemente grafice.
- **MySQL Workbench & Server:** MySQL este un sistem de gestiune a bazelor de date relaționale, open-source și gratuit pentru utilizare. Acesta oferă o performanță rapidă, scalabilitate și fiabilitate, ceea ce îl face unul dintre cele mai populare SGBD-uri utilizate în prezent. Pentru a crea baza de date s-a folosit aplicația client MySQL Workbench.
- **Spring**: Framework-ul Spring este unul dintre cele mai populare și influente framework-uri de dezvoltare a aplicațiilor în limbajul de programare Java. A fost creat pentru a facilita dezvoltarea aplicațiilor de tip enterprise, oferind o abordare modulară și un set bogat de funcționalități.
- **React:** React este o bibliotecă JavaScript extrem de populară și influentă pentru construirea interfețelor utilizator (UI) interactive și reutilizabile. Dezvoltată de către Facebook, React a devenit un instrument esențial în dezvoltarea aplicațiilor web moderne. React utilizează un concept numit "componente", care reprezintă unități independente de interfață utilizator și logică asociată. Aceste componente pot fi combinate pentru a forma interfețe complexe, iar React se ocupă de actualizarea eficientă a acestora în funcție de schimbările de stare.
- **Maven:** este un instrument open-source pentru gestionarea proiectelor de software, care este larg utilizat în dezvoltarea de aplicații Java. Scopul său principal este de a simplifica procesul de construire și gestionare a proiectelor, prin intermediul utilizării unui sistem de gestionare a dependențelor și a unui model de proiect bine definit. Maven utilizează un fișier de configurare XML numit "pom.xml" pentru a defini informațiile

și dependențele proiectului, precum și pentru a specifica etapele de construire și testare a acestuia. Utilizând aceste informații, Maven poate automatiza procesul de construire a proiectului, inclusiv descărcarea și instalarea automată a bibliotecilor și framework-urilor necesare.

Capitolul 3. Justificarea limbajului de programare ales.

Pentru implementarea aplicației server s-a ales limbajul Java iar pentru dezvoltarea aplicației client s-a optat pentru limbajul Javascript.

Există mai multe motive pentru care Java este o alegere bună pentru a rezolva cerința proiectului, printre care:

1. **Portabilitate**: Java este un limbaj de programare portabil, ceea ce înseamnă că codul scris în Java poate fi executat pe orice platformă care are un mediu de rulare Java instalat. Aceasta face Java o alegere bună pentru proiecte care trebuie să ruleze pe mai multe platforme diferite.
2. **Performanță**: Java este un limbaj de programare performant, cu un sistem de gestionare a memoriei eficient și un mecanism de gestionare a garbage collection. Acestea permit aplicațiilor Java să ruleze rapid și eficient.
3. **Siguranța**: Java este un limbaj de programare sigur, deoarece încorporază o serie de funcționalități de securitate pentru a preveni vulnerabilitățile de securitate ale aplicațiilor. De exemplu, Java are un model de securitate sandbox care protejează utilizatorii de posibile amenințări de securitate.
4. **Biblioteci și framework-uri**: Java are o gamă largă de biblioteci și framework-uri disponibile pentru a ajuta la dezvoltarea aplicațiilor, inclusiv biblioteci pentru interfața utilizatorului (Swing), gestionarea bazelor de date (JDBC), testarea unitară (JUnit), dezvoltarea web (Spring, Struts) și multe altele.
5. **Popularitatea și comunitatea**: Java este un limbaj de programare popular, utilizat de o mulțime de dezvoltatori și organizații din întreaga lume. Există o comunitate activă și mare de dezvoltatori Java care pot oferi ajutor și resurse pentru proiecte Java.

În concluzie, Java este o alegere bună pentru a rezolva cerința proiectului, datorită portabilității, performanței, siguranței, bibliotecilor și framework-urilor disponibile, precum și datorită popularității și comunității sale puternice.

Există mai multe motive pentru care Javascript cu React este o alegere bună pentru a rezolva cerința proiectului, printre care:

1. **Popularitate și adoptare extinsă:** JavaScript este un limbaj de programare larg adoptat în industria dezvoltării web, ceea ce asigură o compatibilitate excelentă și o experiență fluidă pentru utilizatori.
2. **Performanță și interactivitate:** Framework-ul React utilizează virtual DOM pentru a actualiza interfața utilizator într-un mod eficient, oferind o performanță excelentă și o experiență interactivă fluidă.
3. **Componente reutilizabile:** React permite dezvoltatorilor să creeze componente reutilizabile, ceea ce facilitează dezvoltarea și menținerea aplicației client.
4. **Ecosistem bogat:** Ecosistemul React oferă o gamă largă de biblioteci și pachete adiționale care extind funcționalitățile de bază ale React, furnizând instrumente și soluții pentru diverse cerințe de dezvoltare.
5. **Curba de învățare accesibilă:** JavaScript și React au o curba de învățare accesibilă, iar comunitatea puternică de dezvoltatori oferă suport, tutoriale și documentație detaliată.

Capitolul 4. Descrierea diagramelor UML.

În programul Star UML s-au realizat următoarele diagrame: diagrama entitate – relație a schemei bazei de date, diagrama cazurilor de utilizare și diagrama de clase.

4.1. Descrierea diagramei entitate – relație.

În Figura 1 este prezentată schema entitate – relație a bazei de date. Schema bazei de date este formată din 2 tabele independente: *Plante* și *Utilizatori*.

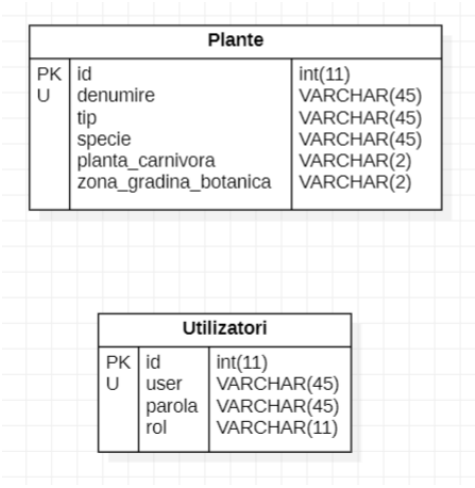

Figura 1. Diagrama entitate - relație.

Tabela *Plante* este formată din următoarele coloane: *id* (de tipul int), *denumire* (de tipul VARCHAR), *tip* (de tipul VARCHAR), *specie* (de tipul VARCHAR), *planta_carnivora* (de tipul VARCHAR), *zona_gradina_botanica* (de tipul VARCHAR). Cheia primară a tabelei *Plante* este coloana *id*. Toate coloanele au constrângerea *Not NULL* și coloana *denumire* mai are și constrângerea *UNIQUE*. Această tabelă păstrează informații despre plantele din grădina botanică.

Tabela *Utilizatori* este formată din următoarele coloane: *id* (de tipul int), *user* (de tipul VARCHAR), *parola* (de tipul VARCHAR), *rol* (de tipul VARCHAR). Cheia primară a tabelei este coloana *id*. Toate coloanele au constrângerea *Not NULL* și coloana *user* mai are și constrângerea *UNIQUE*. Tabela păstrează datele informații despre utilizatorii care necesită autentificare.

4.2. Descrierea diagramei cazurilor de utilizare.

În Figura 2 este prezentată diagrama cazurilor de utilizare.

Se poate observa că utilizatorii de tip *Vizitator* nu necesită autentificare. Acesta poate vizualiza lista de plante din baza de date, poate filtra această listă în funcție de diferite criterii și poate căuta o plantă după denumire.

Din diagramă se constată că în aplicație trebuie implementată funcția de *Autentificare* care este extinsă de funcția *Eșec autentificare*. În cazul în care autentificarea nu s-a efectuat cu succes funcția *Eșec autentificare* va fi apelată.

Utilizatorul de tip *Angajat* necesită autentificare. Acesta poate realiza toate operațiile pe care le poate efectua utilizatorii de tip *Vizitator*. Pe lângă aceste operații *Angajatul* poate efectua operații de tip CRUD asupra persistenței plantelor. *Angajatul* poate salva listele de plante în diferite formate: csv, json, xml, txt și de asemenea poate vizualiza anumite statistici legate de plantele din grădina botanică.

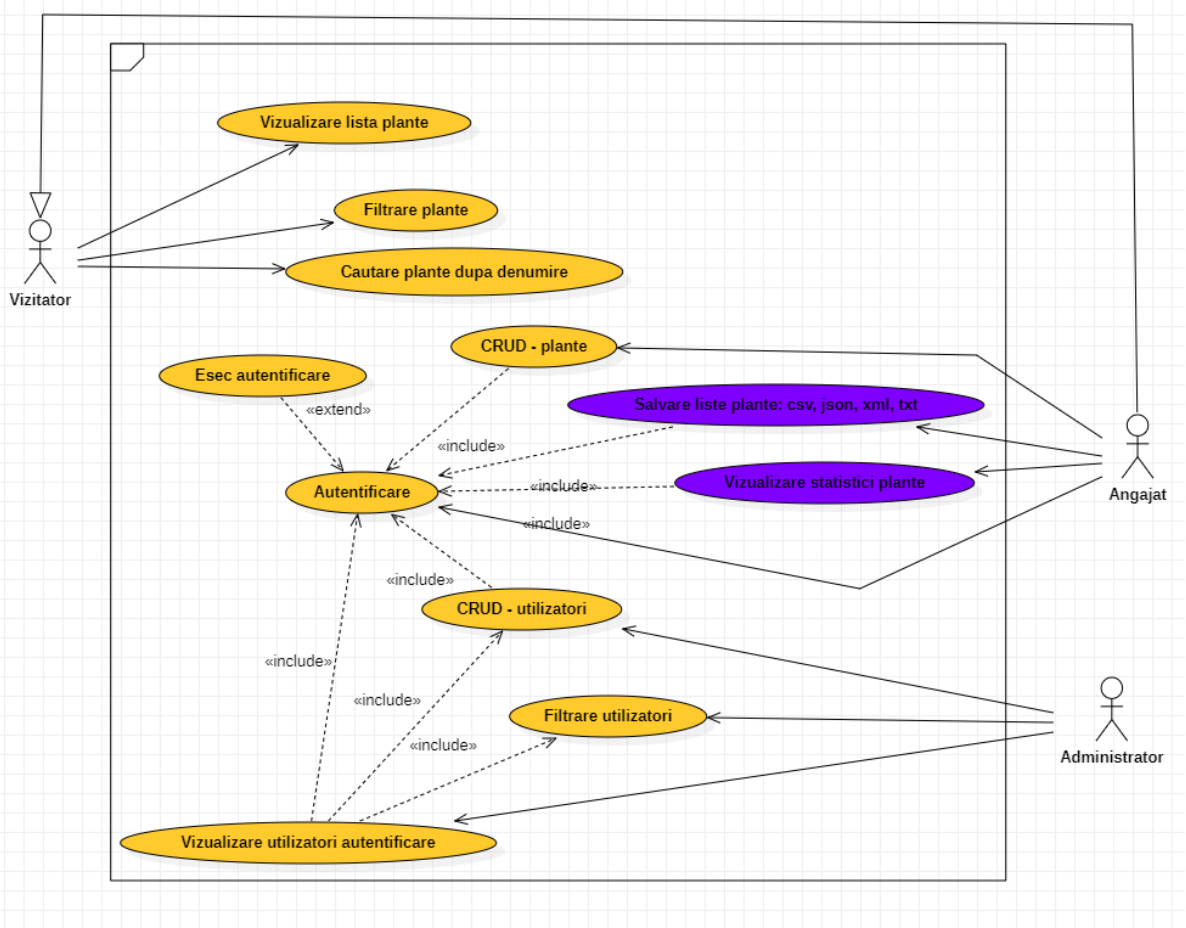

Figura 2. Diagrama cazurilor de utilizare.

Funcționalitățile de a efectua operații CRUD, de a salva lista de plante în diferite formate și vizualizarea statisticilor sunt incluse de funcționalitatea de *Autentificare*, adică operațiile CRUD sunt disponibile odată ce autentificarea s-a realizat cu succes.

Utilizatorul de tip *Administrator* necesită autentificare. Acesta poate efectua operații de tip CRUD asupra utilizatorilor. Această funcționalitate este de asemenea inclusă de funcționalitatea de *Autentificare*. Realizarea operațiilor CRUD asupra utilizatorilor include și vizualizarea tuturor utilizatorilor care necesită autentificare și filtrarea acestora după rol.

Funcționalitățile colorate cu galben sunt implementate în aplicația server iar cele cu mov sunt implementate în aplicația client.

4.3. Descrierea diagramei de clase.

Aplicația server s-a implementat respectând pattern-ul **layered architecture**.

Diagrama de clase a aplicației **server** este structurată pe 4 pachete: *Model*, *Repository*, *Service* și *Controller*. Aplicația este structurată în acest fel pentru a putea respecta pattern-ul de proiectare MVP.

4.3.1. Model.

Figura 3. Model.

Modelul, Figura 3, conține logica de business. În interiorul pachetului sunt implementate următoarele clase: *Plant*, *User*.

Clasa *Plant* corespunde entității *Plante* din baza de date. Starea clasei este caracterizată de atributele: *id*, *name*, *type*, *species*, *carnivorous*, *zone*. Toate atributele sunt de tip *String*, mai puțin atributul *id* care este de tip *Long*, și sunt declarate *private*. Comportamentul clasei este definit de metode. Metodele specifice acestei clase sunt metodele de tip constructor, get și set pentru toate atribute și de metoda *equals*, care verifică dacă există 2 obiecte identice de tipul *Plant*. Clasa are implementați 1 constructor.

Clasa *User* corespunde entității *Utilizatori* din baza de date. Starea clasei este caracterizată de atributele: *id*, *user*, *password*, *role*. Toate atributele sunt de tip *String*, mai puțin atributul *id* care este de tip *Long*, și sunt declarate *private*. Comportamentul clasei este definit de metode. Metodele specifice clasei sunt metode de tip constructor, get și set pentru atribute și de metoda *equals.* Clasa are implementați 1 constructor

Clasa *Repository* are un singur atribut, care este declarat *private*: *SessionFactory*. Comportamentul clasei este definit de metodele: *buildSessionFactory* care construiește un *SessionFactory*. În cadrul metodei sunt specificate proprietățile sesiunii, cum ar fi: tipul bazei de date, credențialele bazei de date și altele. A doua metodă din această clasă este metoda *getSessionFactory* care returnează sesiunea creată de metoda anterioară.

4.3.2. Repository.

În Figura 4 este prezentat pachetul *Repository*, care conține interfețele *IUserRepository* și *IPlantRepository*. Ambele interfețe moștenesc interfața generică *CrudRepository* din pachetul *SpringJPA*, interfața care conține metode CRUD. Metodele definite în interfețele *IUserRepository*, *IPlantRepository și CrudRepository* vor fi implementate automat în momentul rulării programului, astfel nefiind necesară oferirea unei implementări de către programator. Metodele din *IUserRepository* și *IPlantRepository* nu necesită implementare deoarece numele acestora au fost date respectând convenția de denumire a SpringJPA. În funcție de cuvintele cheie din numele metodei, în momentul rulării programului metoda va fi implementată automat.

În rândurile următoare voi explica pe scurt metodele din cele 2 interfețe:

- **findByUserAndPassword(in name:String, in role:String): User**: metoda caută un utilizator după nume și parolă și returnează un obiect de tip User. Cuvântul cheie *findBy* se referă la acțiunea de a căuta în baza de date în funcție de atributele specificate de după;
- **findByRole(in role:String): List<User>:** metoda returnează o listă de obiecte *User* în funcție de *Stringul role* primit ca parametru;

- **countByCarnivorous(): List<Object> și countByZone(): List<Object>:** returnează numărul obiectelor de tip *Plant* pentru categoria *carnivorous* și *zone*;
- **findByOrderByTypeAscSpeciesAsc(): List<Plant>**: metoda returnează o listă de obiecte de tip *Plant* ordanate ascendent după tip și specie. Ordonarea obiectelor în listă se realizează prin folosirea cuvântului cheie *OrderBy* în numele metodei;
- **filterPlants(in criteria:String, in value:String): List<Plant>:** metoda returnează o listă de obiecte de tip *Plant* în funcție de parametrul *criteria* și *filter*. Criteria reprezintă un atribut al clasei *Plant* și *filter* reprezintă valoarea acestui atribut.

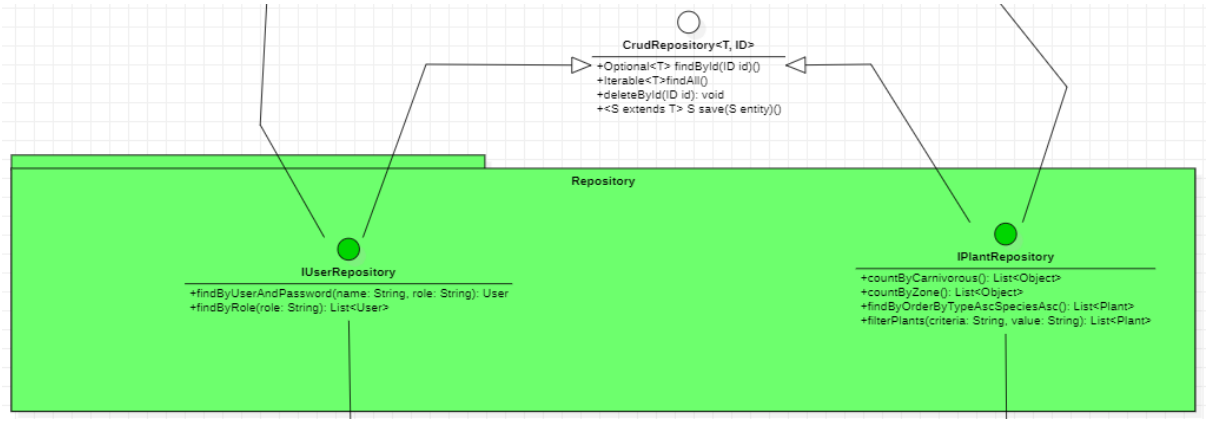

Figura 4. Repository.

4.3.3. Service.

În figura 5 este prezentat pachetul *Service* care conține clasele *UserService* și *PlantService*. Pachetul *Service* accesează metodele necesare din pachetul *Repository* pentru a accesa baza de date.

Pachetul *UserService* are ca atribut un obiect de tip *IUserRepository* și are următoarele metode:

- **getUsers(): List<User>:** metoda returnează lista cu toți utilizatorii;
- **getUserById(in id:Long): User**: metoda returnează un utilizator în funcție de id-ul primit ca parametru;
- **insertUser(in user:User): User**: metoda adaugă în baza de date un utilizator primit ca parametru și returnează utilizatorul adăugat;
- **updateUser(in updatedUser): User**: metoda actualizează un utilizator existent, primit ca parametru, și returnează utilizatorul actualizat;

- **deleteUser(in id:Long): String**: metoda șterge un utilizator din baza de date în funcție de id-ul primit ca parametru, și returnează un String care reprezintă dacă operația s-a realizat cu succes sau nu.
- **getUserCredentials(in name:String, in password:String)**: User: metoda returnează un utilizator în funcție de name-ul și password-ul primite ca parametru;
- **findUsersByRoles(in role:String): List<Users>:** metoda returnează o listă de utilizatori în funcție de role-ul primit ca parametru.

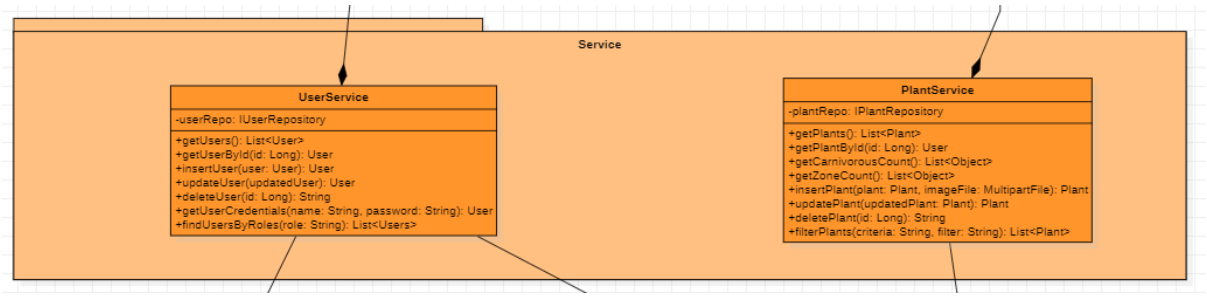

Figura 5. Service.

4.3.4. Controller.

În figura 6 este prezentat pachetul *Controller* care conține 3 clase: *LoginController, AdminController, EmployeeController*. Fiecare controller conține un obiect de tipul claselor din pachetul *Service*, în funcție de operațiile pe care le realizează fiecare controller. Clasele de tip controller conțin endpoint-uri API care realizează legătura cu aplicația de client. Metodele din clasele controller returnează obiectele într-un fișier de tip JSON pentru a putea fi interpretate de aplicația client.

Clasa *LoginController* conține un obiect de tip *UserService* și implmentează metoda:

- **getUserCredentials(in name:String, in password:String): User:** metoda accesează cu același nume din clasa UserService și returnează user-ul aferent numelui și parolei ăntr-un fișier de tip JSON.

Clasa *AdminController* conține un obiect de tip *UserService* și implementează metodele:

- **getAllUsers(): List<User>:** metoda accesează metoda getUsers() din clasa UserService și returnează lista cu toți utilizatorii. Se folosește metoda http GET;

- **getUser(in userId:Long): User:** metoda accesează metoda *getUserById(id: Long)* din clasa *UserService* și returnează în format JSON utilizatorul aferent id-ului primit ca parametru. Se folosește metoda http GET;
- **insertUser(in user:User): User:** metoda accesează metoda cu același nume din UserService și primește un fișier JSON cu utilizatorul care trebuie inserat. Se folosește metoda http POST;
- **updateUser(in user:User): User**: metoda accesează metoda cu același nume din UserService și primește un fișier JSON cu utilizatorul care trebuie actualizat. Se folosește metoda http PUT;
- **deleteUser(in id:Long): String:** metoda accesează metoda cu același nume din UserService și primește un fișier JSON cu id-ul utilizatorului care trebuie șters. Se folosește metoda http DELETE;
- **findByRole(in role:String): List<User>:** metoda accesează metoda findUSersByRole din UserService și returnează un fișier JSON cu lista utilizatorilor. Se folosește metoda http GET;

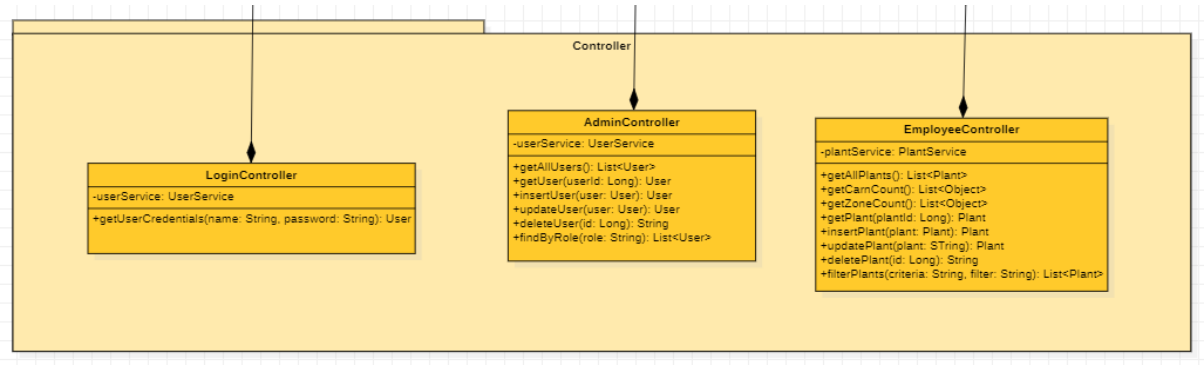

Figura 6. Controller.

Clasa *EmployeeController* conține un obiect de tip *PlantService* și implementează metodele:

- **getAllPlants(): List<Plant>:** metoda accesează metoda getPlants() din clasa *PlantService* și returnează lista cu toate plantele. Se folosește metoda http GET;
- **getPlant(in plantId:Long): Plant:** metoda accesează metoda *getPlantById(id: Long)* din clasa *PlantService* și returnează în format JSON planta aferent id-ului primit ca parametru. Se folosește metoda http GET;
- **insertPlant(in palnt:Plant): Plant:** metoda accesează metoda cu același nume din PlantService și primește un fișier JSON cu utilizatorul care trebuie inserat. Se folosește metoda http POST;

- **updatePlant(in palnt:Plant): Plant**: metoda accesează metoda cu același nume din PlantService și primește un fișier JSON cu planta care trebuie actualizată. Se folosește metoda http PUT;
- **deletePlant(in id:Long): String:** metoda accesează metoda cu același nume din PlantService și primește un fișier JSON cu id-ul plantei care trebuie șterasă. Se folosește metoda http DELETE;
- **filterPlants(in criteria:String, in filter:String): List<Plant>** metoda accesează metoda cu același nume din PlantService și returnează un fișier JSON cu lista plantelor ale căror criteria respectă valoarea filter. Se folosește metoda http GET;
- **getCarnCount(): List<Object> și getZoneCount(): List<Object>:** metodele accesează metodele cu acelși nume din PlantService și returnează numărul fiecărei apariții pentru fiecare valoare pentru atributele carnivorous și zone.

4.3.4. Descrierea diagramei de clase ca întreg.

În 7 este prezentată diagrama de clase completă. Se poate observa cum comunicare se face doar între pachetele adiacente:

- Clasele din *Model* reprezintă entitățile din baza de date;
- Interfețele din *Repository* folosesc metodele de tip get și set pentru a obține și pentru a seta atributele entităților *Plant* și *User* și accesează baza de date;
- Clasele din pachetul *Service* conțin câte un obiect de tipul interfețelor din *Repository*;
- Clasele din pachetul *Controller* reprezintă endpointr-urile de legătură cu aplicația client și accesează metodele necesare din pachetul *Service*.

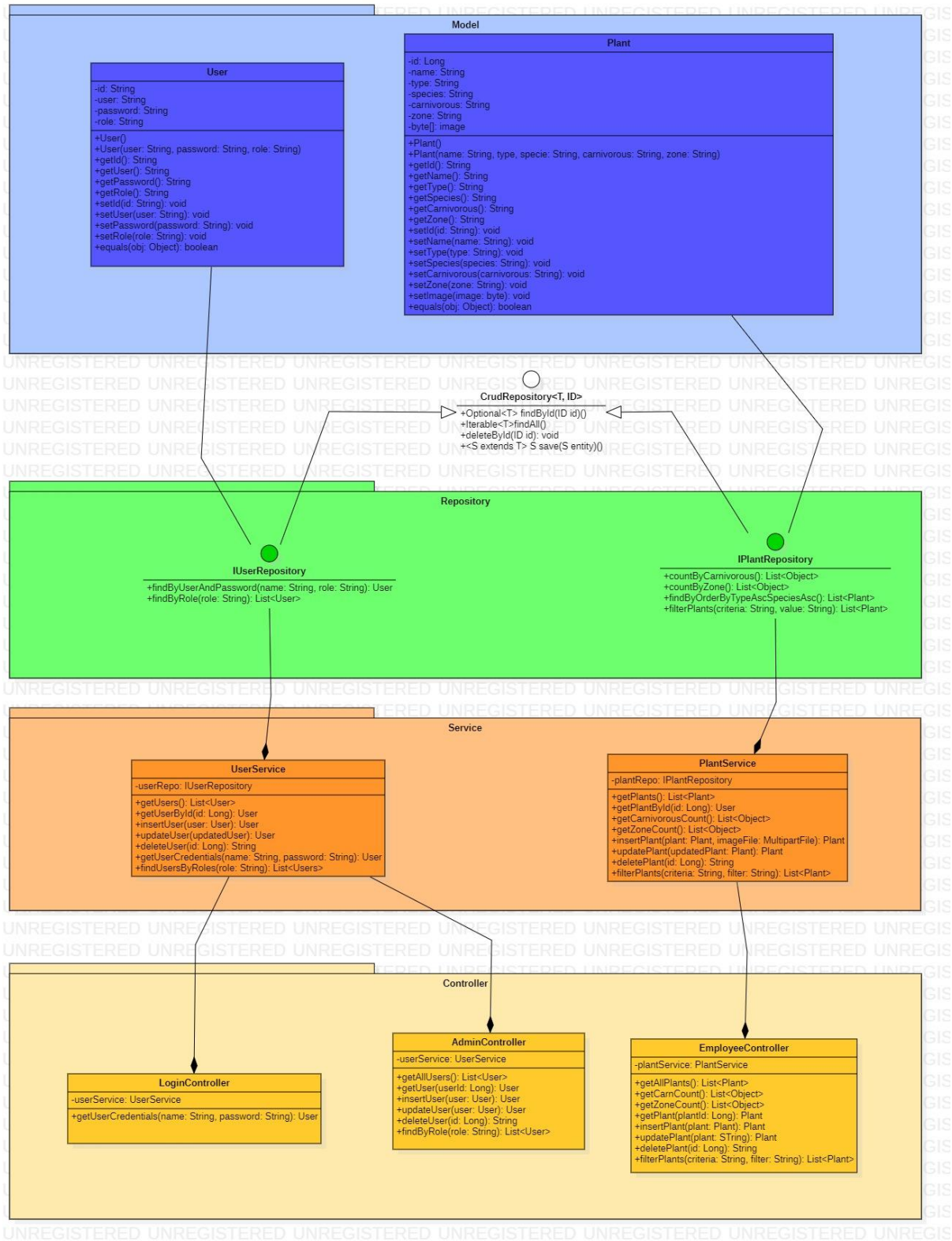

Figura 7. Diagrama de clase.

4.3.5. Descrierea aplicației client.

În figura 8 este prezentă structura principală a aplicației de client. Fiecare fișier conține cod html și css pentru a randa pagina respectivă, dar și funcții care au rolul de a se lega de endpoint-urile din aplicația server.

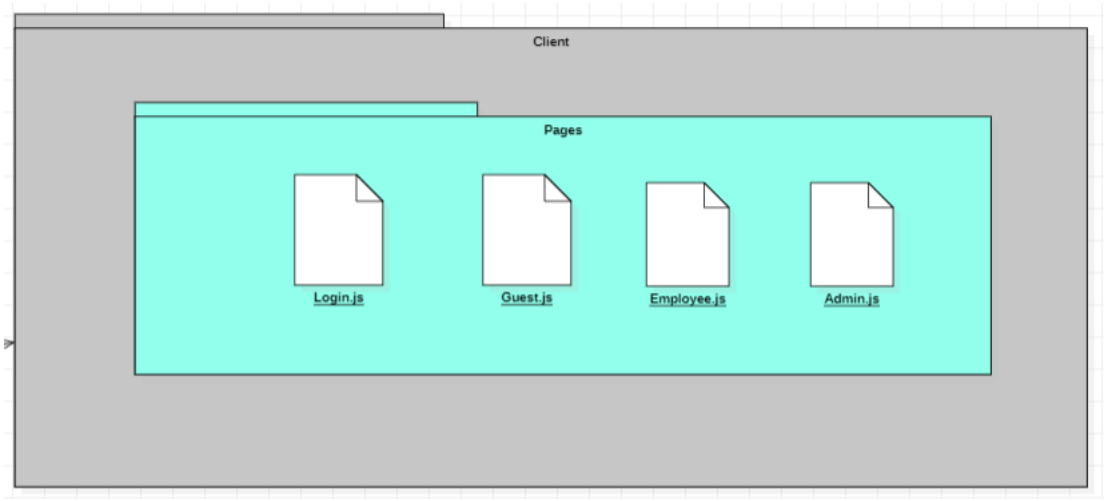

Figura 8. Aplicația client.

- Fișierul **Login.js** randează fereastra principală de unde utilizatorii se vor autentifica sau de unde vizitatorii pot realiza anumite operații. Funcția de autentificare se leagă la endpoint-ul aferent pentru a verifica dacă datele utilizatorului se regăsesc în baza de date.
- Fișierul **Guest.js** randează fereastra pentru vizitatori și prin funcția *useEffect* se obțin toate plantele din baza de date și prin funcția handleFilterPlants se realizează operația de filtrare a plantelor după anumite criterii;
- Fișierul **Admin.js** randează fereastra pentru utilizatorii de tip administrator. Prin funcția *useEffect* se obțin toți utilizatorii din baza de date. Prin funcția *handleSubmit* se realizează operația de inserare și modificare a unui nou utilizator în baza de date. Prin metoda *handleDeleteUser* se realizează operația de ștergere și prin metoda *handleFilterUsers* se realizează operația de filtrare. Toate funcțiile menționate se cuplează la endpoint-ul corespunzător din aplicația de server.
- Fișierul **Employee.js** randează fereastra pentru utilizatorii de tip angajat. Prin funcția *useEffect* se obțin toate plantele din baza de date. Prin funcția *handleSubmit* se realizează operația de inserare și modificare a unei noi plante în baza de date. Prin metoda *handleDeletePlant* se realizează operația de ștergere și prin metoda

handleFilterPlants se realizează operația de filtrare. Toate funcțiile menționate se cuplează la endpoint-ul corespunzător din aplicația de server. Funcțiile *downloadJsonData, downloadXmlData, downloadCsvData, downlaodTxtData* realizează descărcarea listei de plante în diferite formate.

4.3.6. Descrierea aplicației client și aplicației server.

În figura 9 este prezentată diagrama aplicației client – server.

Tehnologia de comunicare dintre aplicații este JSON. Datele transmise între aplicații sunt formatate în format JSON. Aplicația client îi trimite aplicației server http-request-uri cu date în format JSON iar aplicația server îi va răspunde trimițând răspunsuri în format JSON.

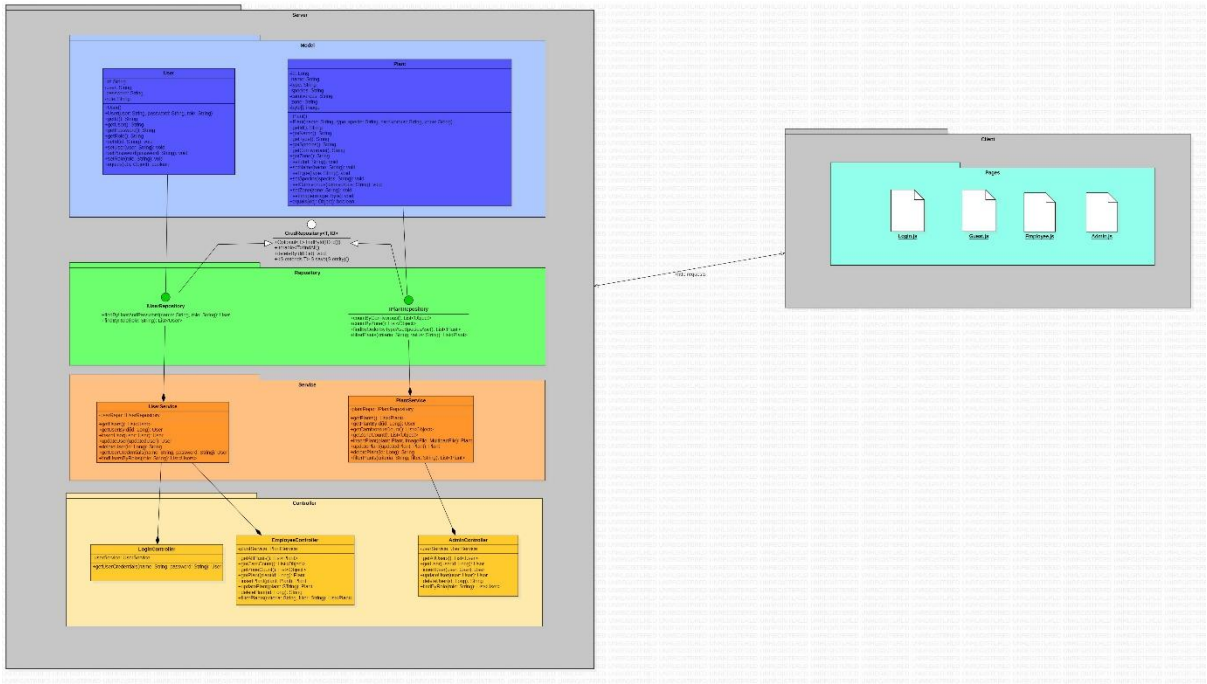

Figura 9. Client – Server

Capitolul 5. Descrierea aplicației.

În figura 10 este prezentată pagina de autentificare.

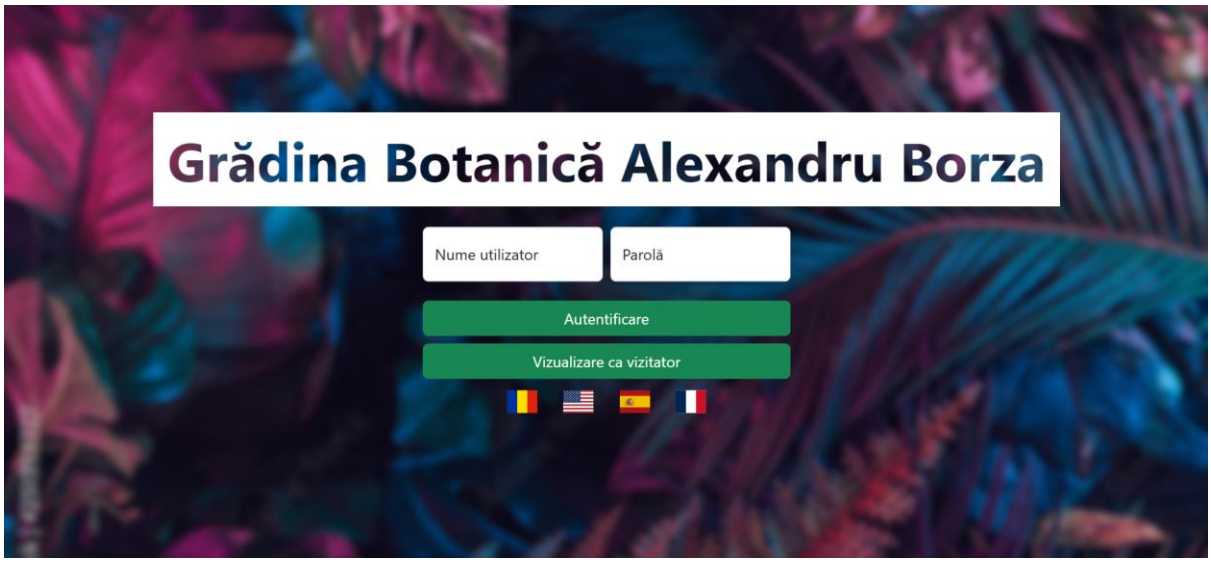

Figura 10. Pagină Login.

În figura 11 este prezentată pagina pentru vizitatori care vor vizualiza plantele sub formă de carduri.

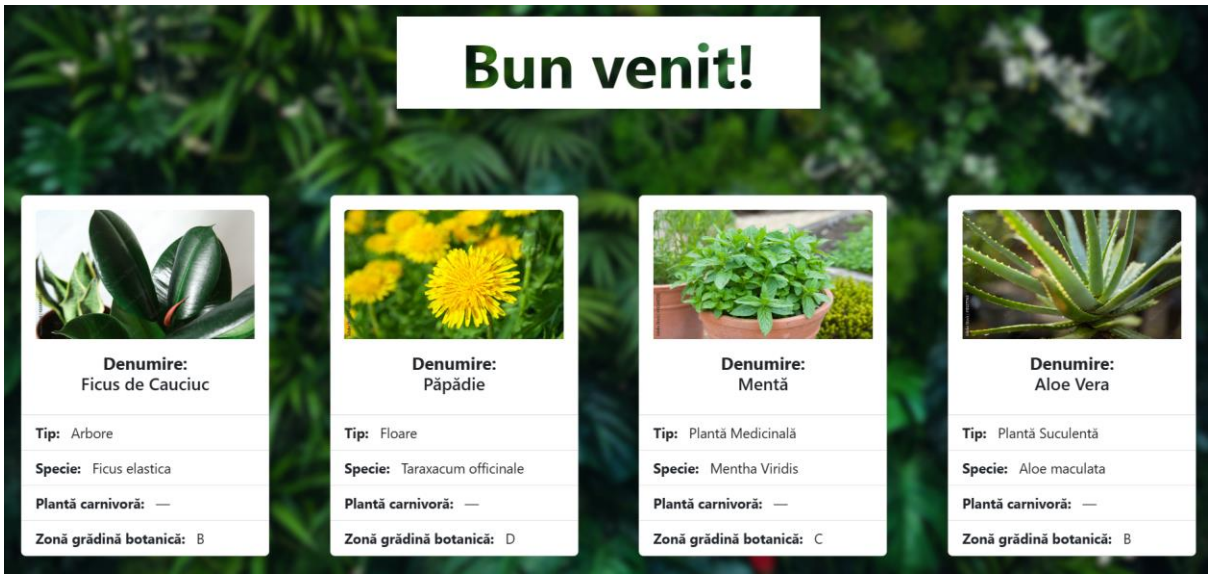

Figura 11. Pagina pentru vizitator.

În figura 12 este prezentat formular de filtrare pentru vizitatori.

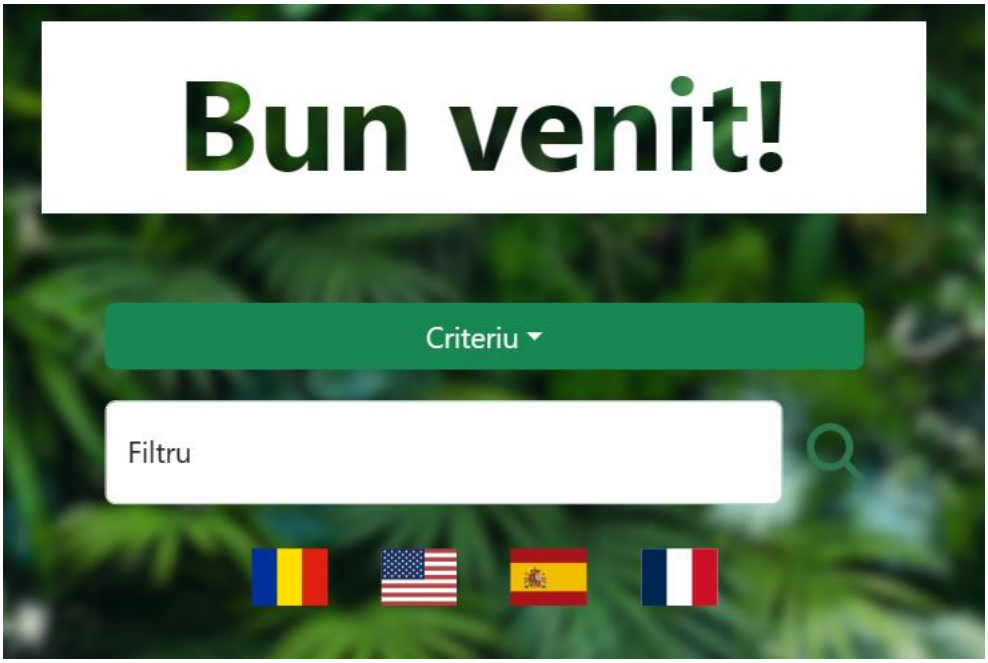

Figura 12. Formular filtrare vizitator.

În figura 13 este prezentată fereastra pentru administratori iar în figura 14 sunt prezentate ferestrele de tip modal pentru adăugare și modificare utilizatori.

Figura 13. Fereastră administratori.

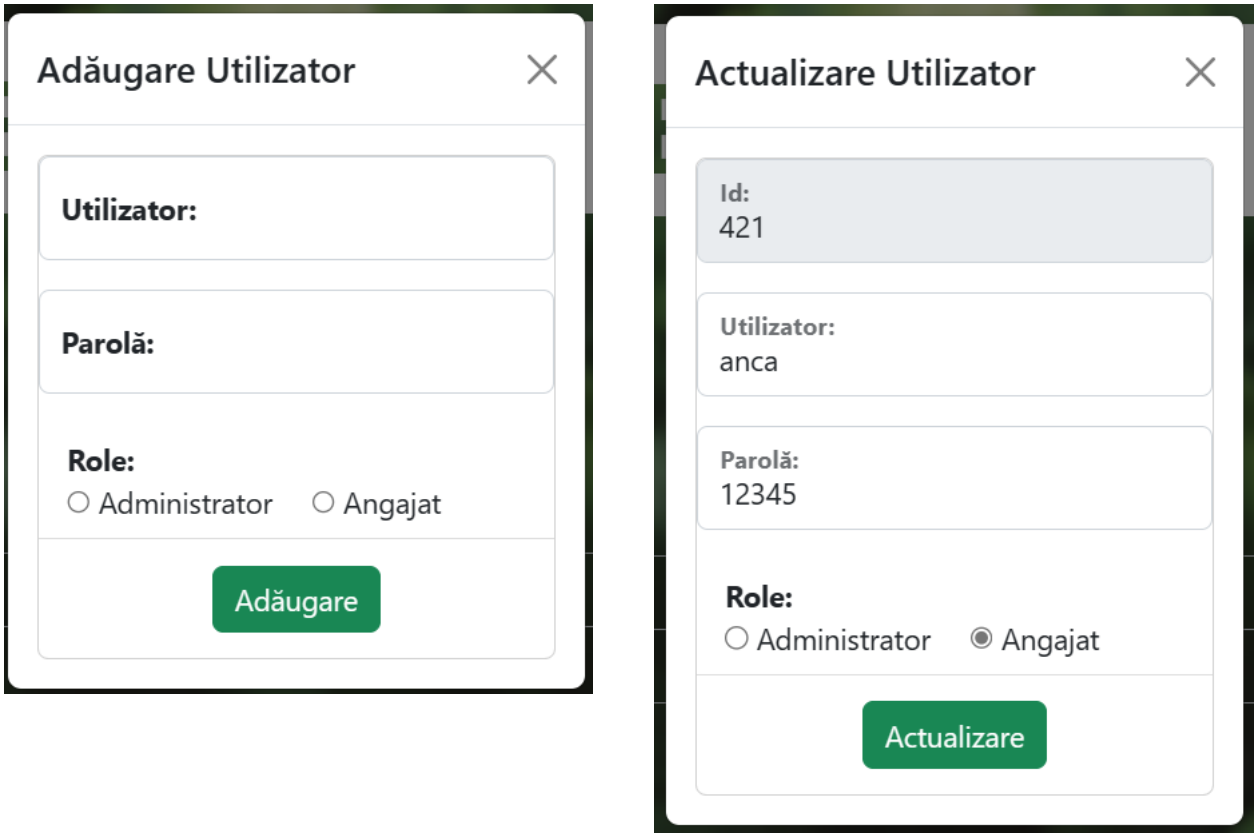

Figura 14 Modale.

În figura 15 este prezentată fereastra pentru angajați.

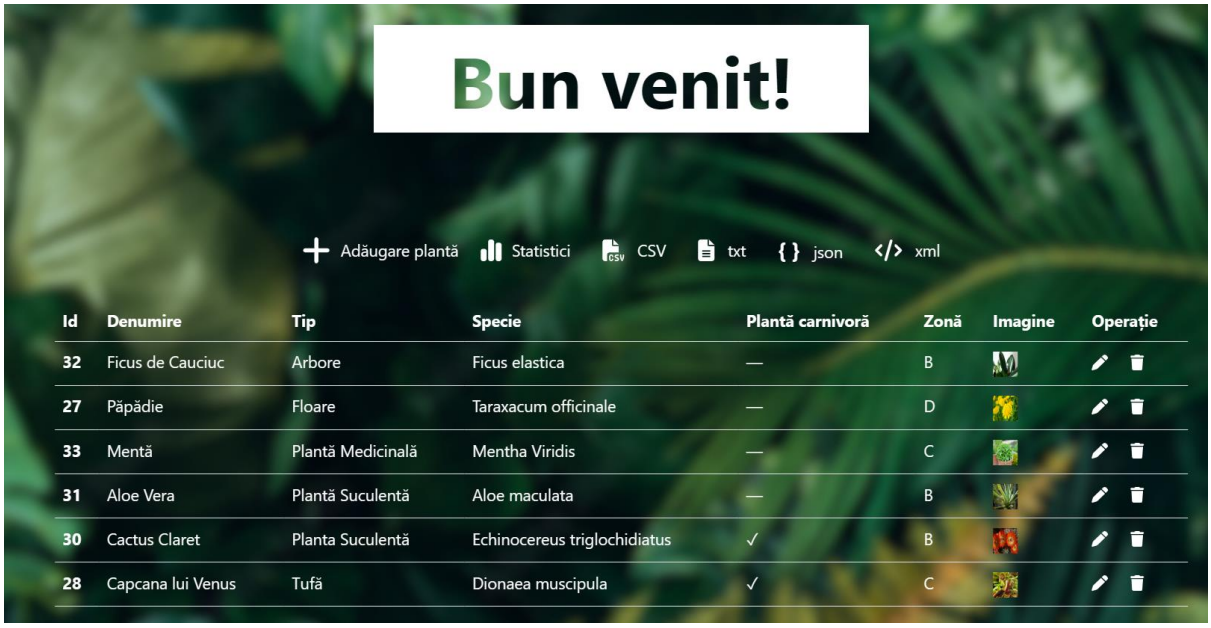

Figura 15. Fereastră angajat.

În figura 16 este prezentată pagina cu statistici legate despre plante.

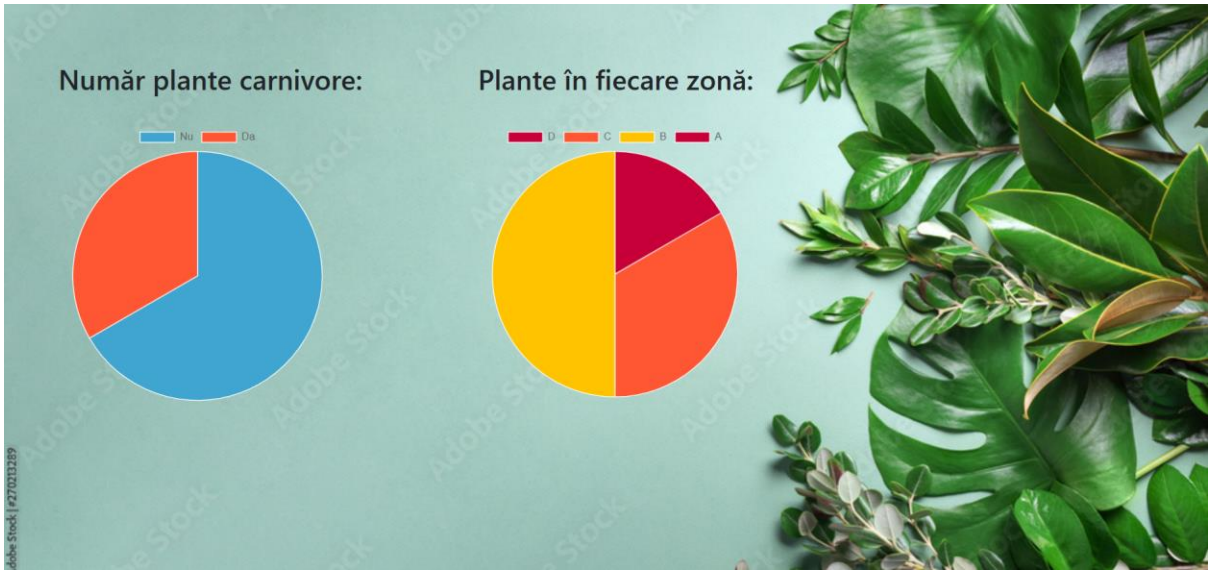

Figura 16. Fereastră statistici.