UNIVERSITATEA TEHNICĂ DIN CLUJ-NAPOCA
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SPECIALIZAREA: COMPLEMENTE DE ȘTIINȚA CALCULATOARELOR

PROIECTAREA UNEI APLICAȚII CARE RESPECTĂ PATTERN-UL DE PROIECTARE MODEL – VIEW – CONTROLLER (MVC)

GEDDERTH CRISTIAN

CSC

2023

Cuprins

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.

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.

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 autentificare.

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.
- **Hibernate**: este un framework open-source pentru Java, utilizat frecvent pentru a facilita accesul la baze de date relationale. Acesta oferă o metodă simplă și eficientă pentru a realiza operațiuni de persistență a datelor. Hibernate utilizează o varietate de tehnici pentru a asigura o performanță ridicată și o integrare ușoară cu alte framework-uri și librării Java. Prin intermediul conceptelor de ORM și de sesiune, Hibernate oferă dezvoltatorilor un set puternic de instrumente pentru a realiza aplicații robuste și scalabile.
- **API-ul swing:** este o bibliotecă grafică pentru crearea de aplicații desktop Java. Aceasta oferă o serie de componente GUI (Graphic User Interface). Biblioteca oferă o varietate de componente GUI, cum ar fi butoane, casete de text, etichete, liste, tabele, check-box-uri, butoane radio, etc.
- **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.

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.

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*.

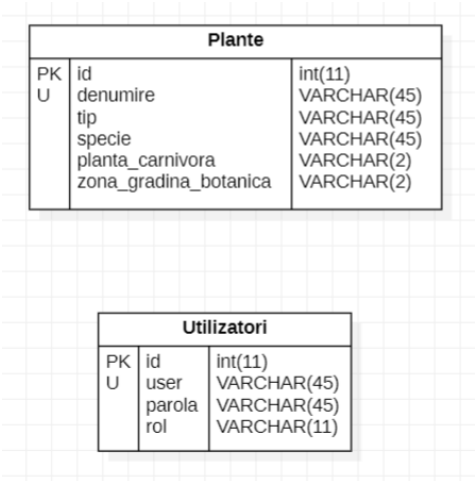

Figura 1. Diagrama entitate - relație.

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.

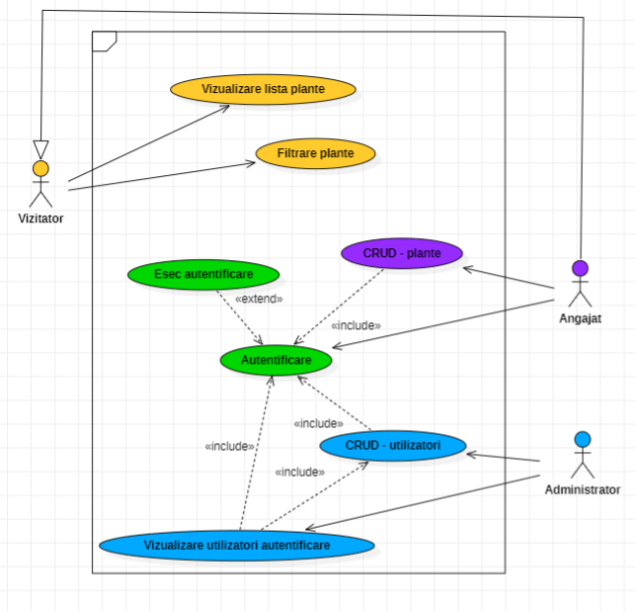

Figura 2. Diagrama cazurilor de utilizare.

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 și de asemenea poate filtra această listă în funcție de diferite criterii.

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. Funcționalitatea de a efectua operații CRUD este inclusă 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.

4.3. Descrierea diagramei de clase.

Diagrama de clase este structurată pe 3 pachete: *Model*, *View*, *Controller*. Aplicația este structurată în acest fel pentru a putea respecta pattern-ul de proiectare MVP.

4.3.1. Model.

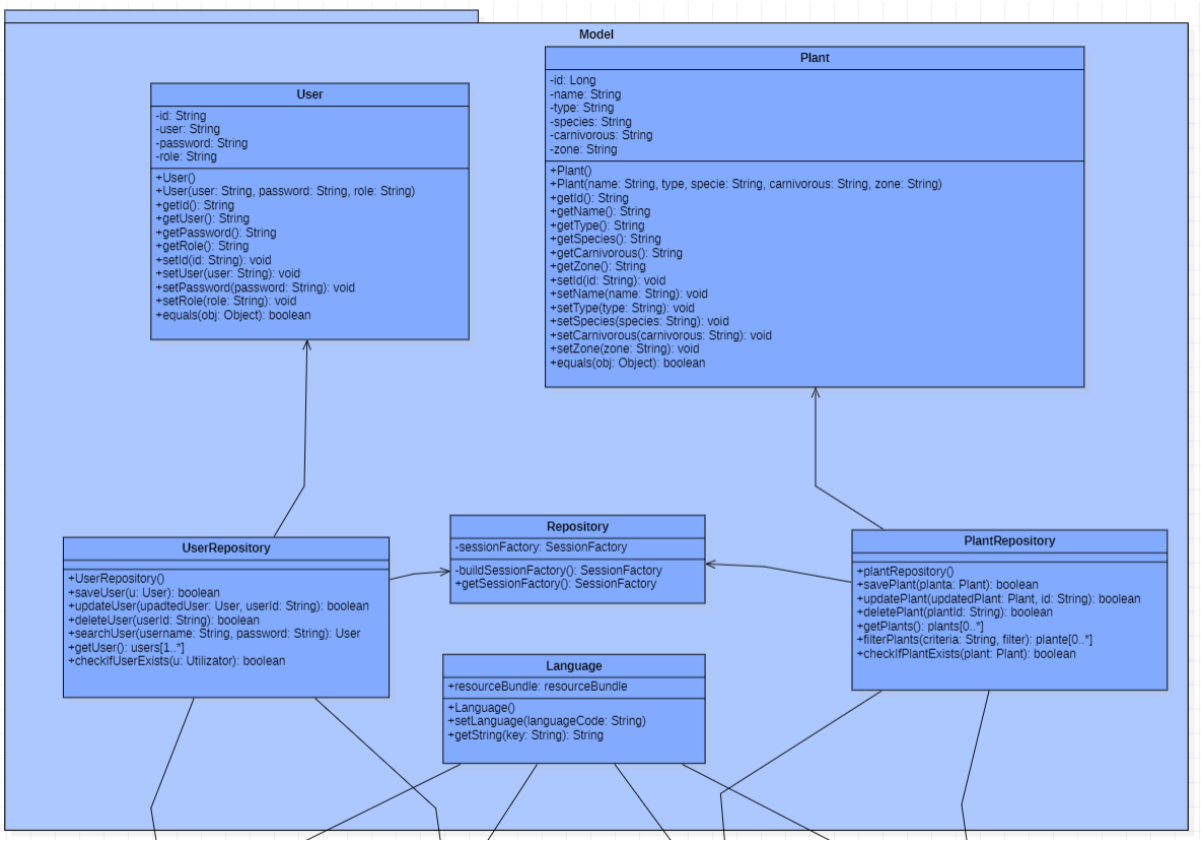

Figura 3. Model.

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

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 2 constructori: unul *default* și unul care folosește toate atributele mai puțin atributul *id*.

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 2 constructori: unul *default* și unul care folosește toate atributele mai puțin atributul *id*.

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.

Clasa *Plant* și *PlantRepository* se află în relație de asociere directă, deoarece clasa *PlantRepository* folosește metodele de tip get și set din clasa *Plant*. Analog se poate spune și despre clasele *User* și *UserRepository*.

Clasele *PlantRepository* și *UserRepository* se află în relație de asociere directă cu clasa *Repository*. Aceste două clase au rolul de a implementa operațiile CRUD asupra entităților corespunzătoare din baza de date. Clasa *PlantaPersistenta* implementează aceste operații prin următoarele metode:

- **savePlant(plant: Plant): boolean** – metoda realizează operația de *INSERT* în baza de date în tabela *Plante* și primește ca parametru obiectul *planta* de tip *Planta*. Metoda returnează o valoare booleană, aceasta fiind *true* dacă operația s-a executat cu succes și *false* în caz contrar.
- **updatePlant(updatedPlant: Plant, id: String): boolean** – metoda realizează operația de *UPDATE* în baza de date în tabela *Plante* și primește ca parametru un String care reprezintă id-ul plantei care se dorește a fi modificată și obiectul *planta* de tip *Planta*. Metoda returnează o valoare booleană, aceasta fiind *true* dacă operația s-a executat cu succes și *false* în caz contrar.
- **deletePlant(plantId: String): boolean** – metoda realizează operația de *DELETE* în baza de date în tabela *Plante* și primește ca parametru un String care reprezintă id-ul plantei care se dorește a fi ștearsă. Metoda returnează o valoare booleană, aceasta fiind *true* dacă operația s-a executat cu succes și *false* în caz contrar.

- **getPlants(): plants[0..*] –** metoda realizează operația de *READ* asupra bazei de date, practic un "SELECT * FROM plante" și returnează lista completă de plante salvate în baza de date. Funcția nu primește parametrii și poate returna de la 0 la mai multe plante.
- **filterPlants(criteria: String, filter: String): plants[0..*] –** metoda realizează operația de *READ* condiționat asupra bazei de date, practic un "SELECT * FROM plante WHERE criteria = 'filter '' și returnează lista filtrată de plante bazată pe coloana și criteriu. Metoda primește doi parametrii: două String-uri- coloana și criteriul. Aceasta poate returna de la 0 la mai multe plante.
- **checkIfPlantExists(plant: Planta): boolean –** metoda verifică dacă în baza de date există o plantă care este identică cu *plant*. Metoda primește un parametru de tip *Plant* și returnează o valoare booleană. Daca metoda returnează *true*, înseamnă că în baza de date există deja o plantă identică cu *plant,* altfel metoda va returna *false*.

Clasa *UserRepository* implementează operații CRUD executate pe tabela *Utilizatori*. Metodele *addUser, updateUser, deleteUser, getUsers* și *checkIfUserExists* au fost implemente la fel ca metodele echivalente din clasa *PlantRepository*. Ceea ce diferă la metoda *getUsers* este că aceasta va returna întotdeauna cel puțin un utilizator. Utilizatorul returnat de metoda va fi însuși utilizatorul care a apelat această funcție. Funcția **searchUser(user: String, password: String): user** primște ca parametrii 2 String-uri care reprezintă un user și o parolă și returnează obiectul *User* care corespunde aceleui user și parole. Metoda este utilizată pentru a realiza funcționalitatea de autentificare.

Clasa *Language* are un atribut privat de tip *ResourceBundle* în care sunt salvate textele traduse în 4 limbi: română, engleză, spaniolă, franceză. Clasa are un constructor fără parametrii care inițializează ResourceBundle-ul pentru limba romană. Clasa mai are o metodă de tip setLanguage care primește ca parametru un *String* și are rolul de a seta codul limbii. De asemenea, clasa mai are o metodă numită *getString* care primește ca parametru un *String* numit *key* și returnează *String*-ul corespunzător *key*-ului din *ResourceBundle*. Această clasă extinde clasa predefinită *Observable*.

4.3.2. View.

În Figura 4 este prezentat pachetul *View*, care conține clasele care implementează interfețele grafice utilizator: *LoginView*, *AdminView*, *EmployeeView*, *GuestView*. Clasele care implementează interfețele grafice utilizator (GUI) au ca atribute elemente grafice din biblioteca *Swing*. Fiecare asemenea clasă conține un constructor fără parametrii și metode de tip *get* care returnează elementele grafice din biblioteca *Swing*.

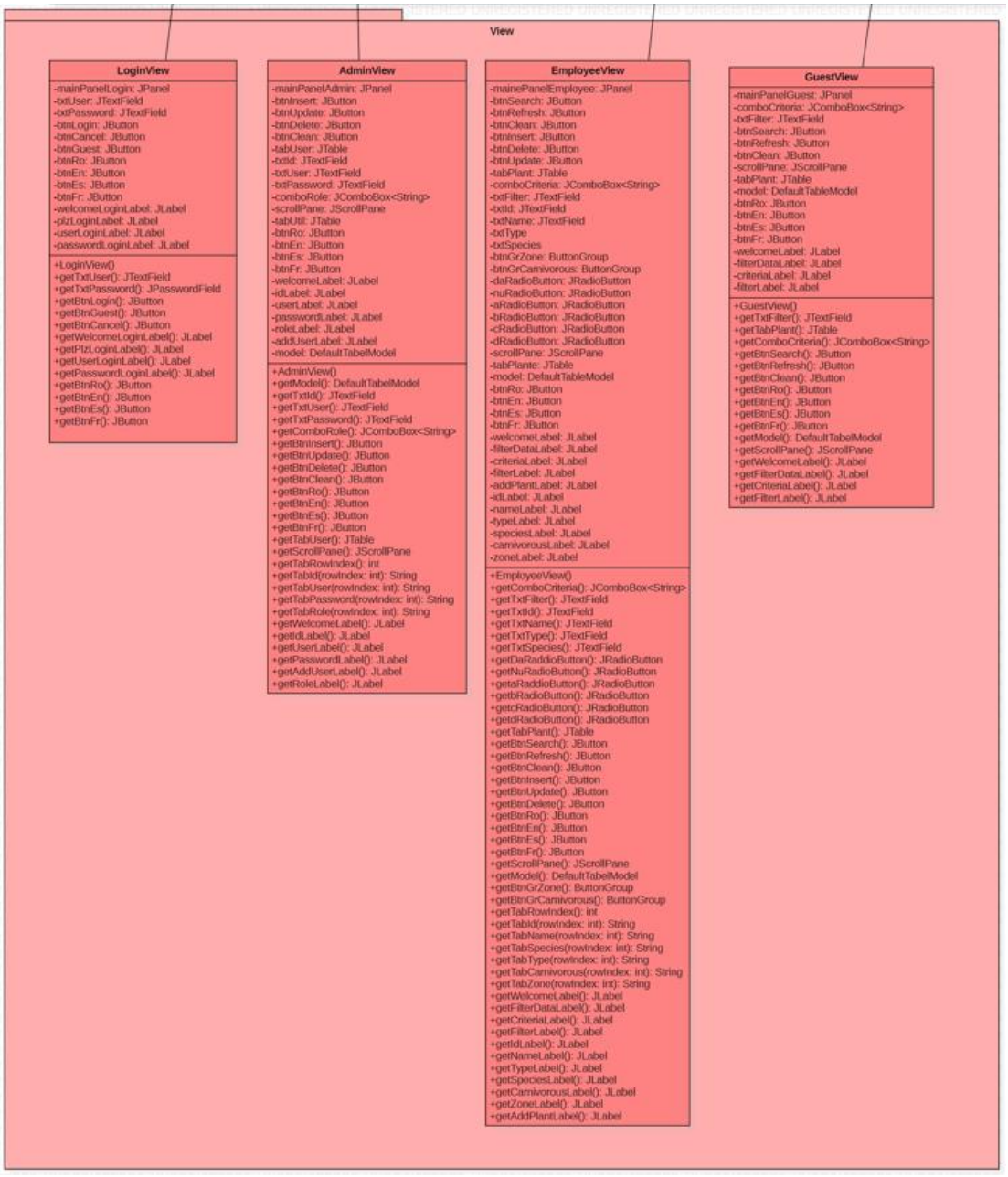

Figura 4. View.

4.3.3. Controller.

În Figura 5 este prezentat pachetul *Controller* care conține câte un Controller pentru fiecare interfață grafică. Fiecare Controller are ca atribut declarat privat: o clasă din pachetul *View*, clasa *PlantRepository* sau *UserRepository* și clasa *Language*. Fiecare Controller implementează interfața predefinită *Observer* care observă schimbarea obiectului de tip *Language*.

Fiecare clasă de tip Controller are un constructor fără parametrii în care: se creează un nou obiect de tipul atributelor fiecărui controller, se adaugă un *Observer* pentru obiectul de tip *Language* și se definesc metodele de tip *ActionListener* care definesc comportamentul evenimentelor realizate în urma interacționării utilizatorului cu elementele grafice.

În rândurile care urmează voi descrie pe scurt metodele din clasele de tip Controller:

- **btnLoginClick():** realizează autentificarea pentru utilizatorii de tip angajat și administrator, după apăsarea butonului Login și deschide o fereastră grafică corespunzătoare rolului;
- **setGuestView():** realizează deschiderea ferestrei pentru utilizatorii de tip vizitator după apăsarea butonului *Vizualizare ca vizitator*;
- **setAdminview():** realizează deschiderea ferestrei pentru utilizatorii de tip administrator după autentificare;
- **setEmployeeView():** realizează deschiderea ferestrei pentru utilizatorii de tip angajat după autentificare;
- **update(Observable o, Object arg):** este metoda din interfața *Observer* și apelează metoda *updateView()*;
- **updateView():** are rolul de a obține traducerile necesare în funcție de limba aleasă de utilizator,
- **insertUserClick(), insertPlantClick():** după apăsarea butonului *Insert* se va realiza operația de INSERT în baza de date;

- **updateUserClick(), updatePlantClick():** după apăsarea butonului *Update* se va realiza operația de UPDATE în baza de date;
- **deleteUserClick(), deletePlantClick():** după apăsarea butonului *Delete* se va realiza operația de DELETE în baza de date;
- **populateTable():** populează tabelele din GUI cu datele din baza de date;
- **refreshTable():** repopulează tabelele din GUI după apăsarea butonului *Refresh*;
- **cleanFields():** șterge datele din elementele grafice de input după apăsarea butonului *Curățare*;
- **setRowCount():** setează numărul de rânduri al tabelului din GUI;
- **showSelectedRowData():** afișează în elementele de input din GUI, datele dintr-un rând din tabel după ce s-a efectuat click pe rândul respectiv.
- **Metode de tip set:** sunt implementate diferite metode de tip set care setează elementele grafice din interfața utilizator.

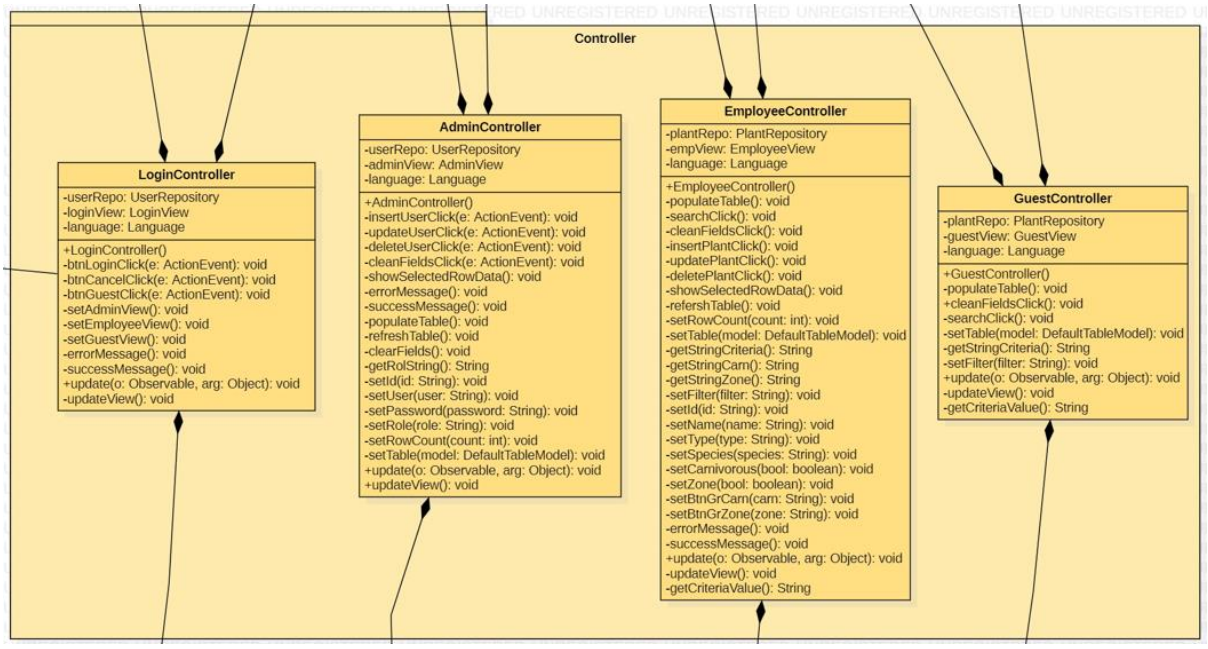

Figura 5. Controller.

4.3.4. Descrierea diagramei de clase ca întreg.

În Figura 6 este prezentată diagrama de clase completă. Se observă independența dintre pachetele *Model* și *View*. Clasele din Pachetul *Controller* se află în relație de compoziție de clasele din pachetele *Model* și *View*:

- **LoginController**: are ca atribute un obiect de tip *LoginView*, un obiect de tip *UserRepository* și un obiect de tip *Language*;
- **AdminController**: are ca atribute un obiect de tip *Adminview*, un obiect de tip *UserRepository* și un obiect de tip *Language*;
- **EmployeeController**: are ca atribute un obiect de tip *EmployeeView*, un obiect de tip *PlantRepository* și un obiect de tip *Language*;
- **GuestController**: are ca atribute un obiect de tip *GuestView*, un obiect de tip *PlantRepository* și un obiect de tip *Language*;.

Clasa *LoginController* se află în relație de compoziție cu clasa *Main* deoarece în *Main* se instanțiază un obiect de tip LoginController.

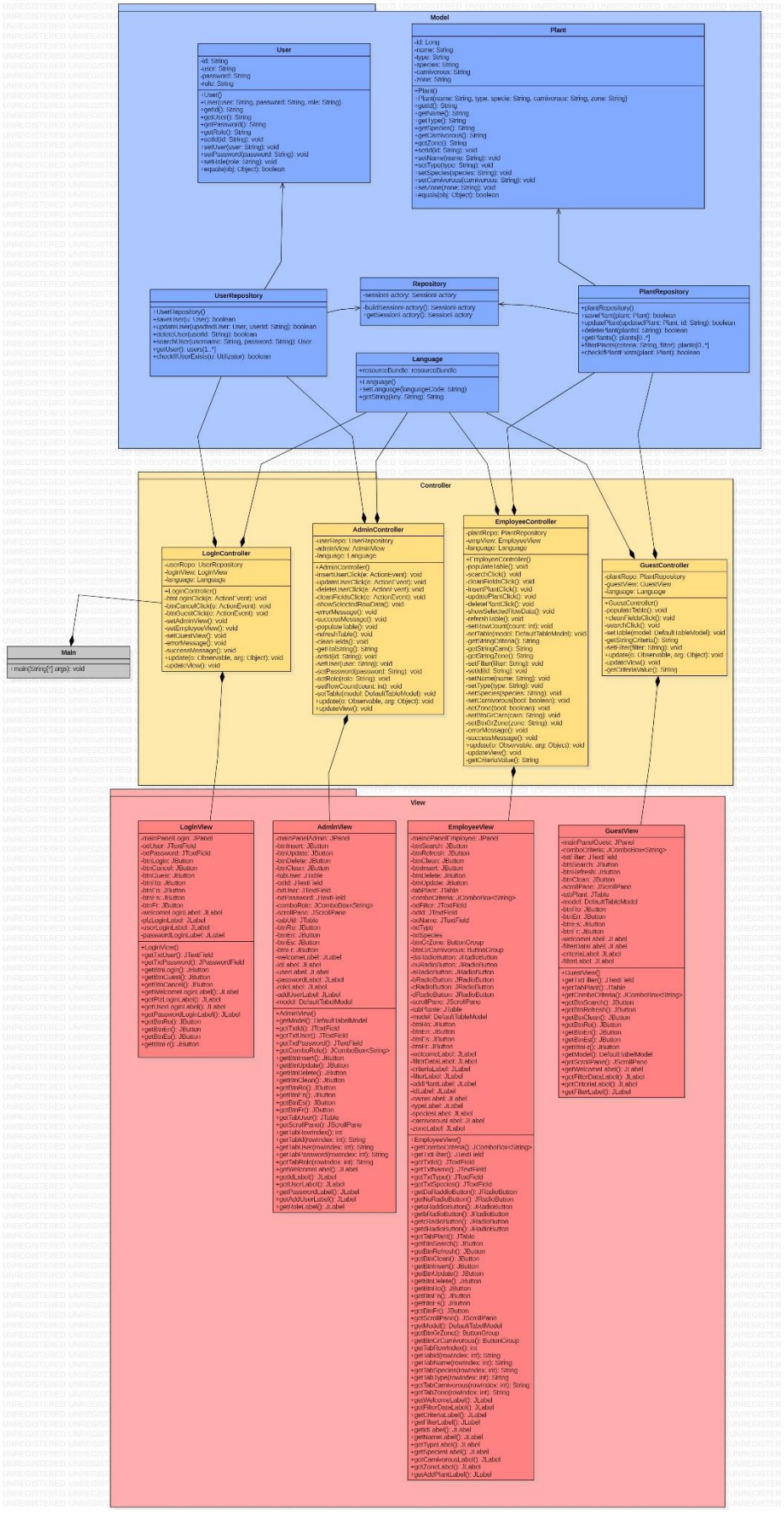

Figura 6. Diagrama de clase.

Capitolul 5. Descrierea aplicației.

Aplicația este formată din 4 ferestre: fereastra de pornire, fereastra pentru utilizatorul de tip vizitator, fereastra pentru utilizatorul de tip administrator și fereastra pentru utilizatorul de tip angajat. În Figura 7 este prezentat ecranul de pornire. Utilizatorii de tip administrator și angajat vor completa câmpurile *Utilizator* și *Parolă* și vor apăsa pe butonul *Autentificare* pentru a se putea autentifica. Utilizatorii de tip vizitator vor apăsa pe butonul *Vizualizare ca vizitator* pentru a fi redirecționați la ecranul corespunzător. Butonul *Renunță* are rolul de a opri aplicația. Iconițele care reprezintă steagul unei țări au rolul de a actualiza interfața grafică în limba corespunzătoare fiecărei țări. Aceste iconițe sunt prezente în fiecare fereastră.

Figura 7. Ecran de pornire.

În Figura 8 este prezentat ecranul pentru utilizatorul de tip vizitator. Acesta conține un tabel în care sunt afișate datele din baza de date din entitatea *Plante*. În dreptul etichetei *Criteriu* se află un câmp de tip dropdown din care vizitatorul poate alege criteriul pe baza căruia dorește să filtreze datele. Câmpurile disponibile sunt: Tip, Specie, Plantă carnivoră, Zonă grădină botanică. Sub acest dropdown este poziționat o casetă de text unde vizitatorul poate introduce valoarea pe baza căreia să se efectueze filtrarea. Butonul *Caută* are rolul de a lansa căutarea pe baza criteriul ales. Butonul *Reîmprospătează* are rolul de reinițializa tabelul astfel încât după o filtrare să fie afișate din nou toate datele din tabel. Butonul *Curățare* are rolul de a șterge valorile introduse în caseta text.

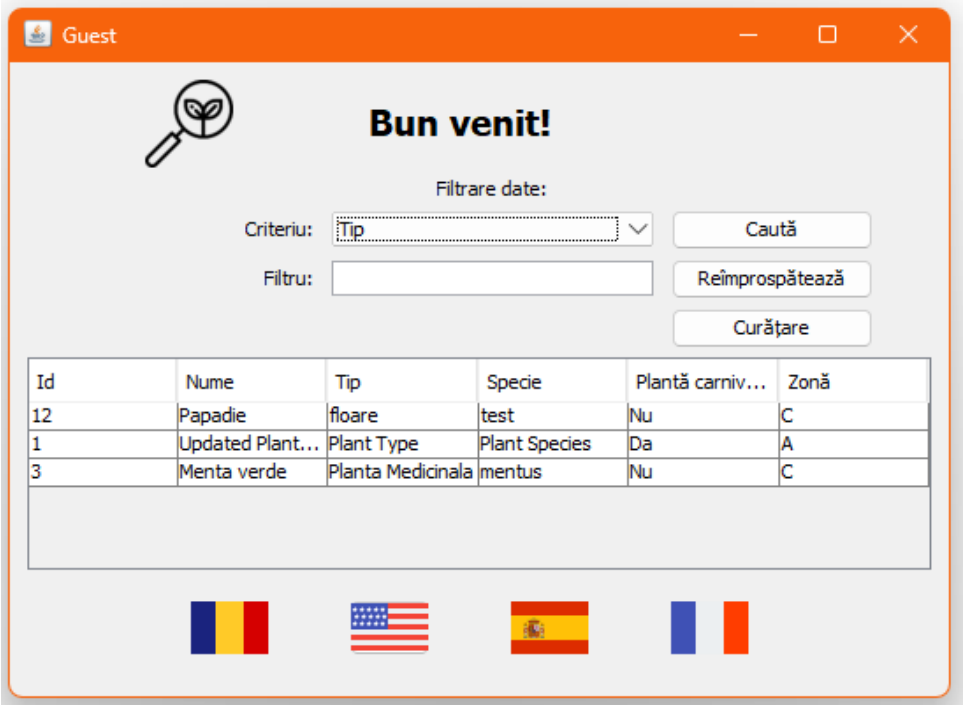

Figura 8. Ecran vizitator.

În Figura 9 este prezentat ecranul utilizatorului de tip administrator. Ecranul conține un tabel în care sunt afișați utilizatorii care necesită autentificare. Din acest ecran administratorii pot efectua operații CRUD pe utilizatorii care necesită autentificare. Ecranul conține 4 câmpuri de text și un dropdown. Primul câmp de text conține id-ul utilizatorului, câmp care este needitabil. În celelalte câmpuri se introduce utilizatorul și parola. Dropdown-ul conține valori care reprezintă rolul utilizatorului: administrator sau angajat.

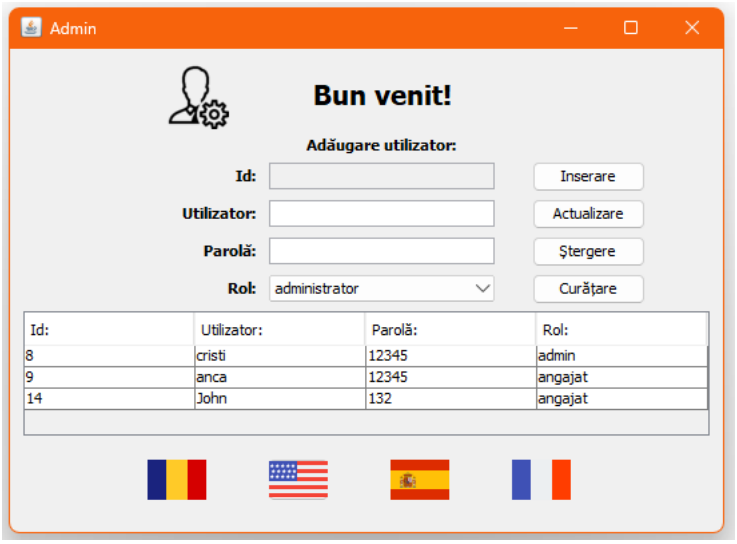

Figura 9. Ecran administrator.

Pentru a introduce un utilizator nou, se vor completa câmpurile *Utilizator*, *Parolă*, *Rol*, câmpul *id* fiind generat automat de baza de date, și în final se apasă butonul *Adăugare*. Pentru a realiza o modificare a unui utilizator, se va apăsa pe rândul din tabel în care sunt salvate datele acestuia. Datele vor fi afișate apoi în casetele text echivalente, astfel administratorul putând modifica datele. Operația se propagă în baza de date apăsând butonul *Actualizare*. Pentru a șterge un

utilizator, se va apăsa pe rândul corespunzător din tabel și apoi pe butonul *Ștergere*. Butonul *Curățare* are aceeași funcționalitate ca butonul cu același nume din ecranul pentru vizitatori. După apăsarea butoanelor care execută operații CRUD va fi afișat un mesaj de eroare sau de succes.

În Figura 10 este prezentat ecranul pentru utilizatorii de tip angajat. Acesta conține un tabel unde sunt afișate datele din entitatea *Plante*. În partea superioară a ecranului se află zona de filtrare a plantelor. Funcționalitățile și modul de utilizare au fost descrise în paragrafele anterioare la ecranul de tip vizitator. Butoanele *Caută*, *Reîmprospătează* și *Curățare* au aceleași funcționalități ca butoanele echivalente din ecranul pentru vizitatori. În partea mediană a ecranului, sunt poziționate casetele text, butoanele radio și butoanele de acțiune (*Adăugare*, *Actualizare*, *Ștergere*) care sunt menite pentru a realiza operațiile de inserare, modificare și ștergere a datelor. Procesul se desfășoară asemănător ca în ecranul de tip administrator.

Figura 10. Ecran angajat.

De asemenea, după realizarea operațiilor de autentificare, inserare, modificare, ștergere se va afișa un mesaj de eroare sau de succes, Figura 11.

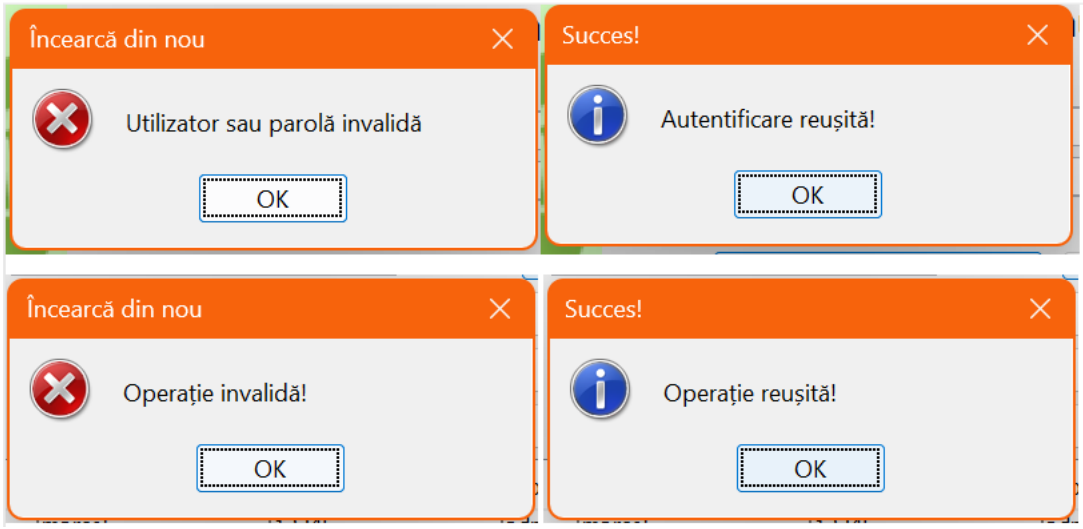

Figura 11. Mesaje.