**Tema 3 Proiectare Software**

**Determinarea traseului optim utilizând transportul in comun**

**Arhitectura MVC**

*Gradinariu Robert-Iulian*

*Grupa 30233*

1. ***Cerinta***

Problema 20

Dezvoltați o aplicație care poate fi utilizată într-un oraș pentru determinarea traseului optim utilizând transportul în comun. Aplicația va avea 3 tipuri de utilizatori: călător, angajat al firmei de transport în comun și administrator.

Utilizatorii de tip călător pot efectua următoarele operații fără autentificare:

❖ Vizualizarea listei tuturor liniilor de transport în comun sortată după număr;

❖ Vizualizarea traseului optim după specificarea stației de plecare și a stației de sosire (reprezentare grafică);

❖ Căutarea unei linii de transport în comun după număr.

Utilizatorii de tip angajat pot efectua următoarele operații după autentificare:

❖ Toate operațiile permise utilizatorilor de tip călător;

❖ Operații CRUD în ceea ce privește persistența liniilor de transport în comun;

❖ Salvare liste cu liniile de transport în comun în mai multe formate: csv, json, xml, doc;

❖ Vizualizarea unor statistici legate de liniile de transport în comun utilizând grafice (structură radială, structură inelară, de tip coloană, etc.).

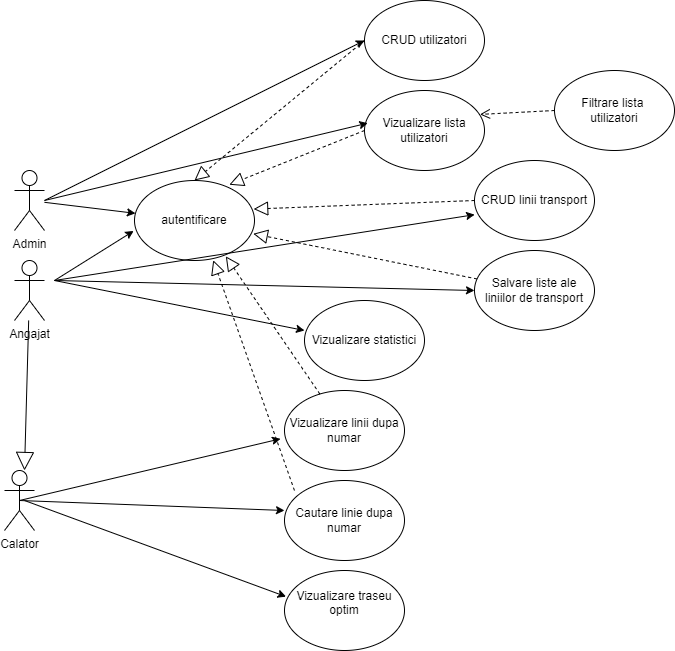
Utilizatorii de tip administrator pot efectua următoarele operații după autentificare:

❖ Operații CRUD pentru informațiile legate de utilizatorii care necesită autentificare;

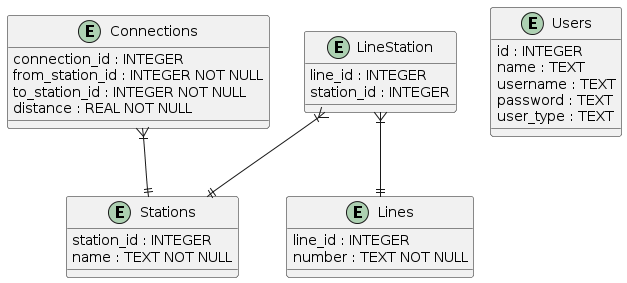
❖ Vizualizarea listei utilizatorilor care necesită autentificare și filtrarea acesteia după tipul utilizatorilor.

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

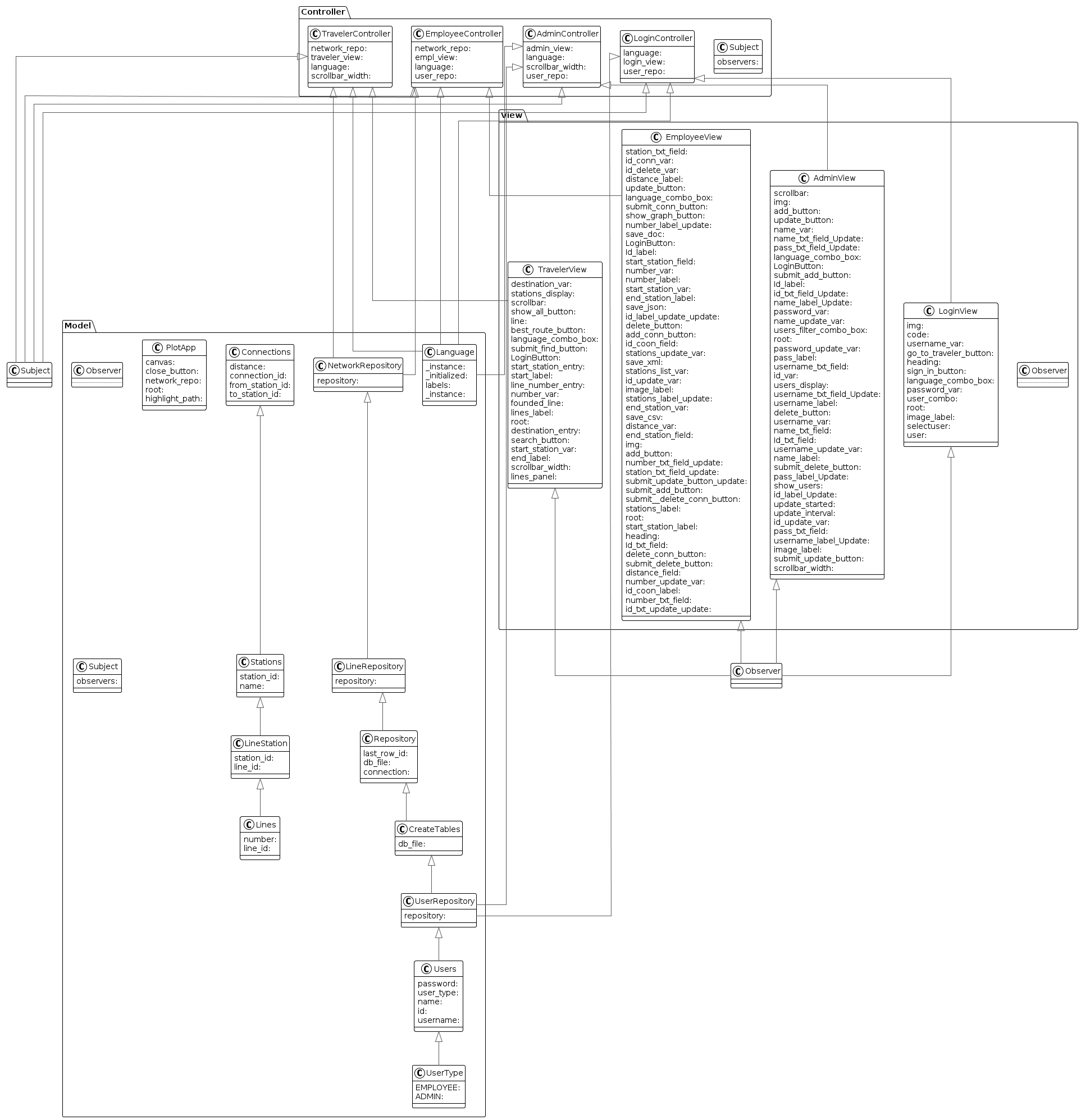
1. ***Diagrame***



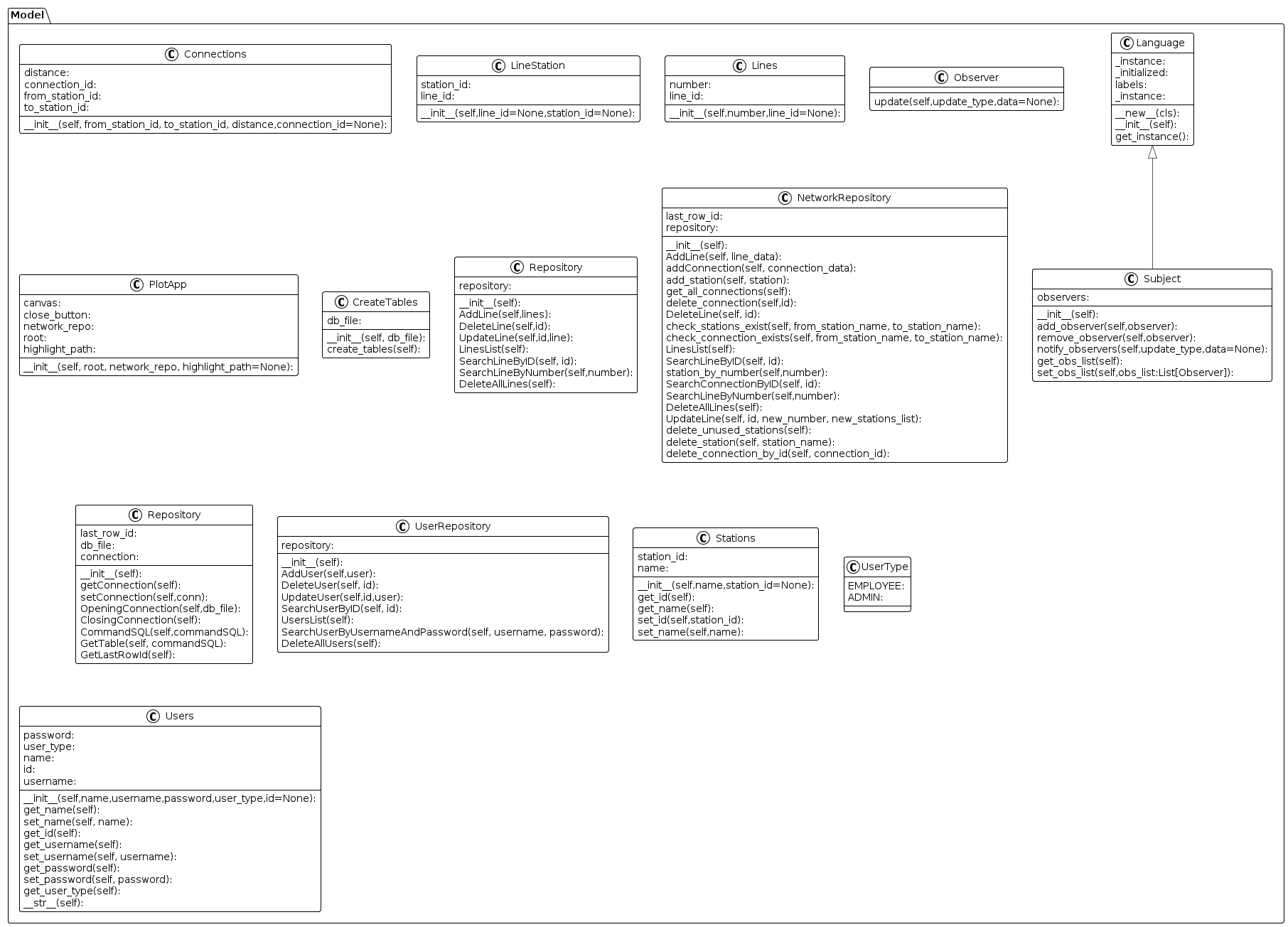
**Diagrama Cazurilor de Utilizare – figura 1**



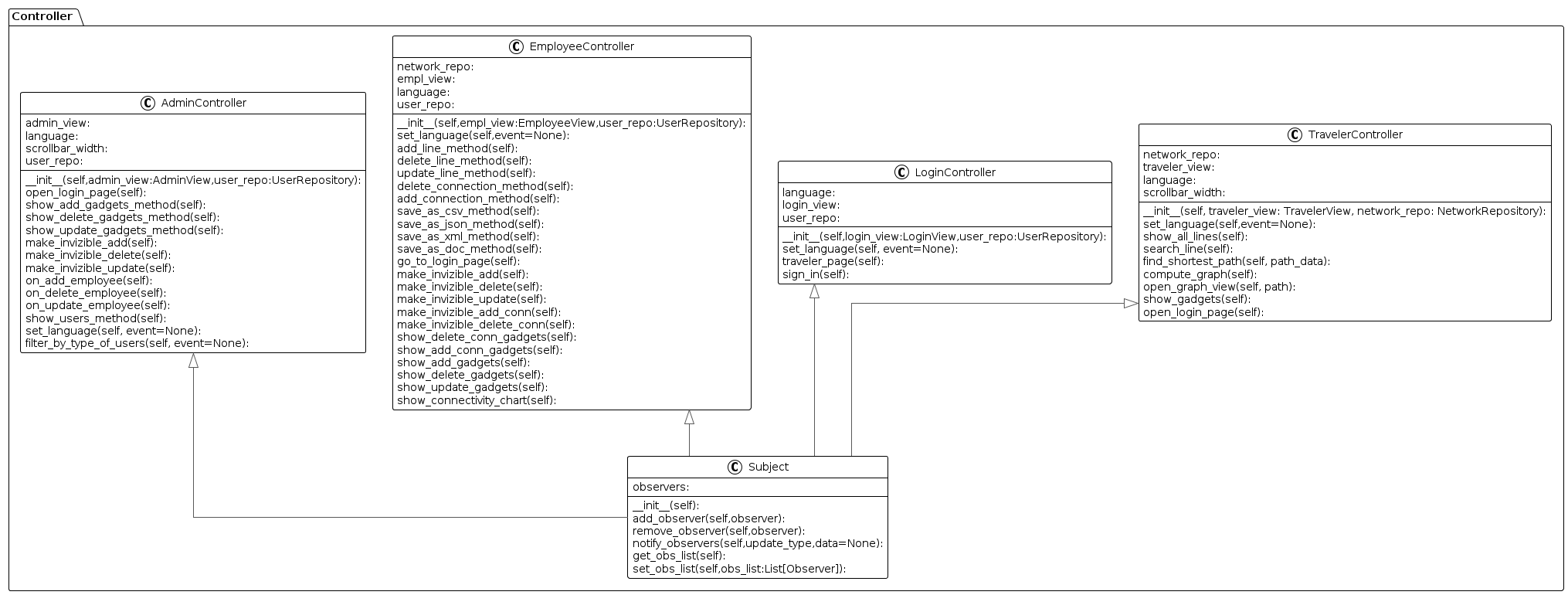
**Diagrama Entitate-Relatie – figura 2**



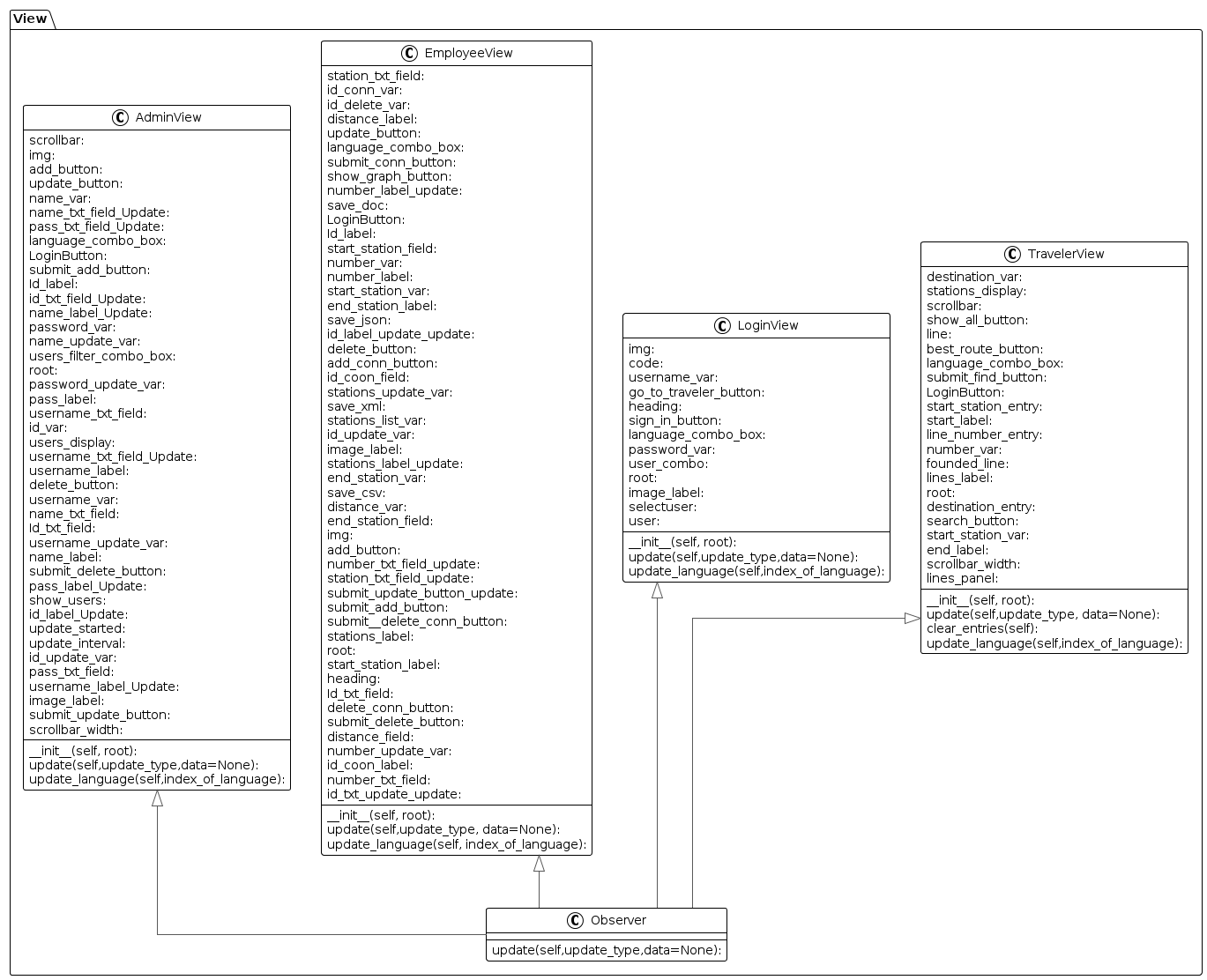
**Diagrama de Clase – figura 3**



**Diagrama Pachetului Model – figura 4**



**Diagrama Pachetului Controller– figura 5**



**Diagrama Pachetului View – figura 6**

1. **Descrierea Diagramelor UML**

*Diagrama Cazurilor de Utilizare*

Aceasta diagrama descrie toate cazurile posibile de utilizare a aplicatiei si este o unealtă utilizată în ingineria software pentru a defini și a reprezenta interacțiunile dintre actori (utilizatori) și un sistem software. Această diagramă este un instrument esențial în etapa de analiză și proiectare a unui sistem software și facilitează înțelegerea și comunicarea între echipele de dezvoltare și clienți.

Actorii sunt entități externe sistemului care interacționează cu acesta. Actorii aplicatiei dezvoltate sunt : calatorul , adminul si angajatul liniilor de transport.

Cazurile de utilizare sunt acțiunile sau secvențele de acțiuni pe care un actor le poate executa în cadrul sistemului. In aplicatia mea cazurile de utilizare sunt separate in functie de actor, unele fiind comune pentru diferiti actori ( de ex calator si angajat ).

*Diagrama Bazei de Date*

Diagrama bazei de date este o reprezentare vizuală a structurii unei baze de date, care include tabelele, relațiile între tabele și coloanele acestora. Este un instrument esențial în proiectarea și modelarea bazelor de date în domeniul ingineriei software și al bazelor de date. Diagrama bazei de date oferă o perspectivă cuprinzătoare asupra modului în care datele sunt organizate și interconectate în cadrul sistemului.

Baza de date a aplicatiei curente consta in 5 tabele care contin informatii despre User, Lines ( liniile de transport in comun), LineStation si Connections.

*Diagrama de Clase*

Diagrama de clase este una dintre cele mai fundamentale și utilizate diagrame în ingineria software, specificată în limbajul de modelare unificat (UML). Această diagramă este folosită pentru a reprezenta structura statică a unui sistem software, evidențiind clasele de obiecte, atributele acestora, metodele și relațiile dintre ele.

Diagrama actuala de clase reprezinta cele 3 pachete importante care modeleaza arhitectura MVC: Model, View si Controller.

**Pachetul Model:**

Descriere:

Pachetul Model reprezintă componenta responsabilă pentru gestionarea și manipularea datelor aplicației. Conține logica de business și regulile de validare a datelor.

*Responsabilități:*

*Stocarea și Gestionarea Datelor:*

Modelul este responsabil pentru stocarea și gestionarea datelor aplicației.

Asigură persistența datelor și comunicarea cu surse externe de date, cum ar fi bazele de date.

*Logica de Business:*

Modelul conține logica de business a aplicației, inclusiv calculele și procesarea datelor. Implementează regulile de validare a datelor pentru a asigura integritatea și consistența acestora.

*Independență de Interfață:*

Modelul este independent de interfața cu utilizatorul și nu conține nicio referință la elementele de interfață. Poate fi reutilizat în diferite contexte și cu diferite interfețe.

**Pachetul View:**

*Descriere:*

Pachetul View reprezintă componenta responsabilă pentru afișarea informațiilor către utilizator și pentru interacțiunea cu acesta. Se ocupă de prezentarea datelor și de primirea acțiunilor utilizatorului.

*Responsabilități:*

*Afișarea Datelor:*

View-ul este responsabil pentru prezentarea datelor primite de la Controller într-un mod comprehensibil și atrăgător. Folosește elemente precum butoane, câmpuri de text, liste etc. pentru a afișa informațiile.

*Interacțiunea cu Utilizatorul:*

View-ul permite utilizatorului să interacționeze cu aplicația prin intermediul evenimentelor, cum ar fi clicuri de mouse sau apăsări de taste. Trimite acțiunile utilizatorului către Controller pentru procesare.

**Pachetul Controller:**

*Descriere:*

Pachetul Controller reprezintă componenta care gestionează fluxul de date între Model și View. Primește acțiunile utilizatorului din View, le procesează și actualizează Modelul și View-ul în consecință.

*Responsabilități:*

*Gestionarea Fluxului de Date:*

Controller-ul primește acțiunile utilizatorului din View și decide cum să le proceseze. Comunică cu Modelul pentru a actualiza datele și cu View-ul pentru a actualiza afișarea.

*Procesarea Acțiunilor Utilizatorului:*

Controller-ul conține logica de procesare a acțiunilor utilizatorului. Poate efectua operații precum validarea datelor, apelarea metodelor din Model și actualizarea View-ului.

*Coordonarea Între Model și View:*

Controller-ul acționează ca un intermediar între Model și View. Obține datele necesare din Model și le pregătește pentru a fi afișate în View. Trimite acțiunile utilizatorului din View către Model pentru a actualiza datele.

Acestea sunt rolurile principale ale fiecărui pachet din arhitectura MVC. Prin separarea responsabilităților între Model, View și Controller, această arhitectură facilitează organizarea, dezvoltarea și mentenanța aplicațiilor software.

1. ***Instrumente utilizate si justificarea limbajului de programare ales***

***Python*** este un limbaj de programare de nivel înalt, interpretat, care promovează scrierea unui cod clar și lizibil. Este cunoscut pentru sintaxa sa simplă și pentru bibliotecile sale extinse, făcându-l ideal pentru dezvoltarea rapidă a aplicațiilor în diverse domenii, de la web la știința datelor.

**Motivele alegerii Python:**

* **Versatilitate:** Poate fi utilizat atât pentru scripting, cât și pentru dezvoltarea de aplicații complexe.
* **Biblioteci Vaste:** Dispune de o gamă largă de biblioteci și framework-uri, cum ar fi Tkinter pentru interfețe grafice și SQLite3 pentru interacțiunea cu baze de date, care facilitează dezvoltarea rapidă a funcționalităților necesare.
* **Comunitate Mare:** Are o comunitate vastă de dezvoltatori, oferind sprijin extensiv și resurse de învățare.
* **Cross-platform:** Poate fi rulat pe majoritatea sistemelor de operare fără modificări semnificative ale codului.

***SQLite*** este un sistem de gestionare a bazei de date relaționale, încorporat în aplicație, care nu necesită un server de baze de date separat. Este ideal pentru aplicații de dimensiuni mici până la medii, prototipuri și proiecte care necesită o bază de date simplă și eficientă.

**Motivele alegerii SQLite:**

* **Ușurință în Utilizare:** Nu necesită configurare sau administrare separată.
* **Portabilitate:** Baza de date este stocată într-un singur fișier pe disc, ușor de copiat și distribuit.
* **Integrare ușoară cu Python:** Python vine cu suport încorporat pentru SQLite prin modulul **sqlite3**, facilitând manipularea datelor.

***Tkinter*** este biblioteca standard de interfață grafică pentru Python, furnizând instrumente pentru crearea de ferestre, butoane, câmpuri de text și alte elemente de interfață grafică. Este o alegere populară pentru proiectele care necesită o interfață grafică simplă, datorită integrării sale strânse cu Python.

**Motivele alegerii Tkinter:**

* **Integrare Strânsă cu Python:** Permite dezvoltarea interfețelor grafice direct în codul Python, fără a fi necesare alte limbaje sau instrumente.
* **Simplu de Învățat și de Folosit:** Are o curvă de învățare blândă și este bine documentat, fiind ideal pentru începători și pentru proiecte cu termene strânse.
* **Cross-platform:** Interfețele create cu Tkinter pot fi rulate pe Windows, macOS și Linux fără modificări majore.

1. ***Descrierea Aplicatiei***

***Clasa Repository (Model):***

Această clasă gestionează conexiunea cu baza de date SQLite și execută operațiuni generale SQL, cum ar fi execuția de comenzi și interogări. Este o componentă centrală pentru manipularea datelor, folosită de alte clase pentru a accesa și modifica baza de date.

***Clasa CreateTables (Model):***

Descriere: Responsabilă cu inițializarea bazei de date prin crearea tabelelor necesare: Lines pentru liniile de transport și Users pentru utilizatori. Această clasă folosește Repository pentru a executa comenzi SQL de creare a tabelelor, asigurând structura inițială necesară aplicației.

***Clasa Lines (Model):***

Reprezintă o linie de transport, având atribute pentru identificator, numărul liniei și lista stațiilor. Această clasă este utilizată pentru a modela și manipula datele referitoare la liniile de transport în cadrul aplicației.

***Clasa NetworkRepository :***

Conține logica specifică pentru manipularea datelor liniilor de transport. Permite adăugarea, ștergerea, actualizarea și interogarea liniilor de transport folosind metodele oferite de Repository pentru a executa operațiuni SQL corespunzătoare.

***Clasa User (Model):***

Model pentru un utilizator al sistemului, având atribute pentru nume, username, parolă și tipul utilizatorului (angajat sau admin). Aceasta clasă este folosită pentru a reprezenta utilizatorii în cadrul aplicației.

***Clasa UserRepository :***

Similar cu LineRepository, această clasă gestionează operațiunile specifice pentru utilizatori, inclusiv adăugarea, ștergerea, actualizarea și căutarea utilizatorilor în baza de date. Utilizează Repository pentru a interacționa cu baza de date.

***Clasa AdminController***

Este responsabilă pentru gestionarea interacțiunii dintre interfața grafică a administratorului (AdminView) și modelul de date al aplicației (UserRepository). Aceasta implementează logica de control pentru diferite acțiuni ale administratorului, cum ar fi adăugarea, ștergerea și actualizarea angajaților, precum și afișarea listei de utilizatori.

AdminController se ocupă de legarea evenimentelor generate de butoanele din interfața grafică cu metodele corespunzătoare din controller. De asemenea, gestionează vizibilitatea și starea widget-urilor în funcție de acțiunea selectată de administrator.

Clasa AdminController extinde clasa Subject, ceea ce sugerează că aceasta este parte a unui model Observer, unde AdminView este un observator al modificărilor din controller. Controllerul notifică observatorii atunci când au loc anumite evenimente, cum ar fi schimbarea limbii sau navigarea către pagina de autentificare.

***Clasa LoginController*** este responsabilă pentru gestionarea interacțiunii dintre interfața grafică de autentificare (LoginView) și modelul de date al utilizatorilor (UserRepository). Aceasta implementează logica de control pentru procesul de autentificare și navigarea către alte pagini ale aplicației.

LoginController se ocupă de legarea evenimentelor generate de butoanele din interfața de autentificare cu metodele corespunzătoare din controller. De exemplu, atunci când utilizatorul apasă butonul "Sign In", controllerul preia datele introduse (username, password, user\_type) și le verifică în baza de date a utilizatorilor folosind metoda SearchUserByUsernameAndPassword din UserRepository. În funcție de tipul de utilizator autentificat (Admin sau Employee), controllerul deschide fereastra corespunzătoare și creează o instanță a controllerului asociat (AdminController sau EmployeeController).

De asemenea, LoginController permite navigarea către pagina Traveler prin intermediul butonului "Go to Traveler" și gestionează schimbarea limbii prin intermediul unui ComboBox.

Clasa LoginController extinde clasa Subject, ceea ce indică faptul că aceasta este parte a unui model Observer, unde LoginView este un observator al modificărilor din controller. Controllerul notifică observatorii atunci când au loc anumite evenimente, cum ar fi schimbarea limbii sau autentificarea cu succes.

În ansamblu, LoginController joacă un rol esențial în gestionarea procesului de autentificare și în coordonarea navigării între diferite pagini ale aplicației în funcție de tipul de utilizator autentificat.

***Clasa EmployeeController*** este responsabilă pentru gestionarea interacțiunii dintre interfața grafică a angajatului (EmployeeView) și modelul de date al aplicației (NetworkRepository și UserRepository). Aceasta implementează logica de control pentru diferite acțiuni ale angajatului, cum ar fi adăugarea, ștergerea și actualizarea liniilor de transport în comun, gestionarea conexiunilor între stații, salvarea datelor în diferite formate (CSV, JSON, XML, DOC) și afișarea graficului de conectivitate.

EmployeeController se ocupă de legarea evenimentelor generate de butoanele din interfața grafică cu metodele corespunzătoare din controller. De exemplu, atunci când angajatul apasă butonul "Add Line", controllerul preia datele introduse (numărul liniei și lista de stații) și le adaugă în baza de date folosind metoda AddLine din NetworkRepository. Similar, controllerul gestionează ștergerea și actualizarea liniilor, adăugarea și ștergerea conexiunilor între stații.

De asemenea, EmployeeController oferă funcționalități de export a datelor în diferite formate, precum CSV, JSON, XML și DOC, prin intermediul metodelor save\_as\_csv\_method, save\_as\_json\_method, save\_as\_xml\_method și save\_as\_doc\_method. Aceste metode utilizează biblioteci specifice pentru a genera fișierele respective cu datele liniilor de transport în comun.

O altă funcționalitate importantă este afișarea graficului de conectivitate între stații. Metoda show\_connectivity\_chart utilizează biblioteca NetworkX pentru a construi un graf pe baza conexiunilor dintre stații și afișează acest graf într-un canvas folosind biblioteca Matplotlib.

Clasa EmployeeController extinde clasa Subject, ceea ce indică faptul că aceasta este parte a unui model Observer, unde EmployeeView este un observator al modificărilor din controller. Controllerul notifică observatorii atunci când au oc anumite evenimente, cum ar fi adăugarea, ștergerea sau actualizarea liniilor, salvarea datelor în diferite formate sau afișarea graficului de conectivitate.

In plus, EmployeeController permite navigarea către pagina de autentificare prin intermediul metodei go\_to\_login\_page. Atunci când angajatul apasă butonul "Login", controllerul creează o nouă fereastră pentru LoginView și un nou LoginController asociat.

Clasa EmployeeController utilizează, de asemenea, clasa Language pentru a gestiona schimbarea limbii aplicației. Atunci când angajatul selectează o limbă din ComboBox-ul corespunzător, controllerul notifică observatorii cu privire la schimbarea limbii.

***Clasa TravelerController*** este responsabilă de gestionarea interacțiunilor dintre interfața utilizatorului (TravelerView) și modelul de date (NetworkRepository). Ea acționează ca un intermediar între cele două componente, primind acțiuni de la utilizator prin intermediul butoanelor și a altor elemente din interfața grafică și apelând metodele corespunzătoare din repository pentru a prelua sau a modifica datele.

Principalele funcționalități ale clasei TravelerController includ:

* Afișarea tuturor liniilor de transport disponibile (show\_all\_lines)
* Căutarea unei linii specifice după numărul său (search\_line)
* Calcularea celui mai scurt traseu între două stații (compute\_graph și find\_shortest\_path)
* Deschiderea unei ferestre separate pentru vizualizarea grafică a traseului (open\_graph\_view)
* Gestionarea schimbării limbii interfației (set\_language)
* Deschiderea paginii de autentificare (open\_login\_page)

Clasa utilizează, de asemenea, un model Observer pentru a notifica vederea (TravelerView) atunci când au loc anumite acțiuni, permițând vederii să se actualizeze în consecință.