Universitatea Tehnică de Construcții din București
Facultatea de Hidrotehnică
Specializarea: Automatică și Informatică Aplicată

Sistem de gestionare a proprietatilor imobiliare

Coordonator ştiinţific:
Sl.dr.ing. Oana FLANGEA
Drd.ing Gabriela OLTEANU

Absolvent:
Remus-Gabriel ROSCA

București, 2025

CUPRINS

INTRODUCERE

Motivarea alegerii temei

Am ales tema “Sistem de gestionare a proprietatilor imobiliare “ deoarece in ultimii ani piata imobiliara a cunoscut o dezvoltare semnificativa, volumul proprietatilor imobiliare este in crestere continua iar prin acest sistem se poate realiza gestionarea acestora mult mai

usor.Proprietarii, administratorii de cladiri si agetiile imobiliare se confrunta tot mai des cu dificultati legate de evidenta chiriasilor,plata chiriilor, achizitionarea de noi proprietati. Astfel, lipsa unui sustem centralizat de gestionare a acestor informatii poate aduce pe termen lung pierderi de timp, comunicraea ineficienta dintre vanzator si cumparator si in final pieredea profitului.

Un sistem software dedicat poate automatiza aceste procese, oferind rapiditate siguranta in gestionarea datelor. Alegerea acestei teme se bazeaza ,asadar, pe nevoia de reala de optimizare a activitatilor din cadrul domeniului imobiliar.

Obiectivele propuse in cadrul lucrarii

Obiectivul principal al acestui proiect este realizarea unei aplicatii dezoltatea in limbajul Java, care sa permita gestionarea eficienta a proprietatilor imobiliare,prin aplicarea conceptelor fundamentale ale programarii orientate pe obiecte si a conectarii acesteia la baza de date.

Al doilea obiectiv il reprezinta crearea claselor necesare functionarii aplicatiei, si respectiv a metodelor necesare.In crearea acestor clase se propune utilizarea principiilor POO: incapsulare, mostenire si polimorfism.

Un alt obiectiv este implementarea functionalitatilor de baza ale aplicatiei: citirea ,stergerea si afisarea proprieatilor, crearea utilizatorilor, introducerea acestora in baza de date pentru o gestionarea mai eficienta si mai usoara.

De asemenea se va avea in vedere implementaeea mecanismelor de validare si de tratarea erorilor, pentru a asigura corectitudinea informatiilor si stabilitatea aplicatiei.

Structura

Lucrarea este compusă din 5 capitole şi un capitol ce concretizează concluziile, contribuțiile şi perspectivele de dezvoltare viitoare. Proiectul prezintă contribuțiile cu privire la sistemele informatice pentru gestionarea disfuncţionalităţilor unei aplicaţii, semnalate de client, sau altfel spus, aplicatiile de ticheting.

In cadrul primului capitol am prezentat conceptul de ticketing, unde este intalnit, de ce este benefic, unde si cine il poate folosi si ce beneficii aduce folosirea acestuia in cadrul companiilor mici si nu numai.

In cel de al doilea capitol am descris succint tehnologiile pe care le-am folosit pentru dezvoltarea aplicatiei si limbajele de programare folosite in scrierea acesteia. Am folosit programe precum Visual Studio Community 2015 si SQL Server, dar si limbaje precum C#, JavaScript, CSS, HTML, JQuery.

In cadrul capitolului trei am prezentat modul de functionare al fiecarui modul cuprins in aplicatie, astfel explicand care este rolul modulelor si cum isi aduc acestea aportul pentru buna functionare a aplicatiei.

Capitolul patru este o scurta prezentare al modului in care conturile noi se creeaza in cadrul aplicatiei, cum sunt ele gestionate de catre administrator si rolurile pe care un utilizator le poate avea in aplicatie, roluri pe care este bazat si accesul in cadrul modulelor.

In ultimul capitol am descris aplicatia in sine, modul in care aceasta se configureaza, cum se foloseste si am prezentat succint un scenariu de crearea a unui utilizator nou, configurarea

clientului pe care acesta va fi atribuit si modul in care utilizatorul introduce un tichet in sistem, este preluat mai departe de un alt utilizator helpdesk si solutioneaza tichetul.

1. TEHNOLOGII SOFTWARE FOLOSITE PENTRU DEZVOLTAREA APLICATIEI

1.1.Visual Studio Code

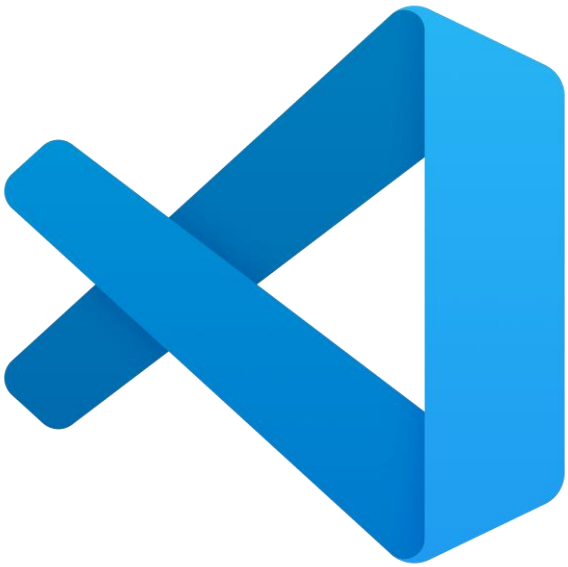
Visual Studio Code este un editor de cod sursa cross-platform, dezvoltat de Microsoft, bazat pe arhitectura Electron. Acesta combina simplitatea unui editor de text cu functionalitati avansate de mediu de dezvoltare integrat(IDE) ,fiind optimizat pentru procesele moderne de editare,build si debugging.

Spre deosebire de IDE-urile traditionale care pot fi consumatoare de resurse, VS Code adopra o arhitectura modulara:

-Nucleu Light-Weight: Motorul principal este axat pe viteza si performanta in gestionarea fisierelor de mari dimensiuni.

-Language Server Protocol(LSP) : Permite suportul pentru multiple limbaje de programare prin procese separate, oferind analiza statica si auto-complete fara a ingreuna interfata utilizatorului.

-Integrare cu Sisteme de Fisiere: Suport nativ pentru spatii de lucru si gestionarea proiectelor complexe.



1.2.Java

Java este un limbaj de programare pe obiect, de nivel inalt, dezvoltat de compania Sun Microsystems(in prezent detinut de Oracle). Acesta este conceput pentru a fi portabil, sigur si eficient, permitand, rularea aplicatiilor pe diferite platforme hardware si sisteme de operare, confrom ,principiului “write once, run anywhere”.
Limbajul Java ofera suport nativ pentru programarea orientata pe obiect, incluzand concepte precum incapsularea , mostenirea si polimorfismul, facilitand dezvoltarea de aplicatii modulare, usor de intretinut si extensibile.

Datorita stabilitatii, securitatii si performantei sale , Java este utilizat pe scara larga in dezvoltarea aplicatiilor enterprise,sistemelor distribuite si aplicatiilor comerciale.

1.3. SQL

SQL(Structured Query Language) este limbajul strandarg folosit pentru a comunica cu bazele de date relationale. Spre deosebire de limbajele procedurale(Pythonsau Java), SQL este un limbaj delarativ, ceea ce inseamna ca utilizatorul specifica ce date doreste sa obtina sau sa modifice,nu si cum sa execute procesul tehnic de extragere a acestor date.

2. IMPLEMENTAREA APLICATIEI

Aplicatia pe care am realizat-o este alcatuita din mai multe clase esentiale pentru functionarea acesteia. Fiecare clasa reprezinta un fisier .java separat, conectate intre ele prin importuri si prin baza de date. Pentru implementare am folosit Java(interfata cu utilizatorul) si SQL(conectivitate si utilizare baza de date).

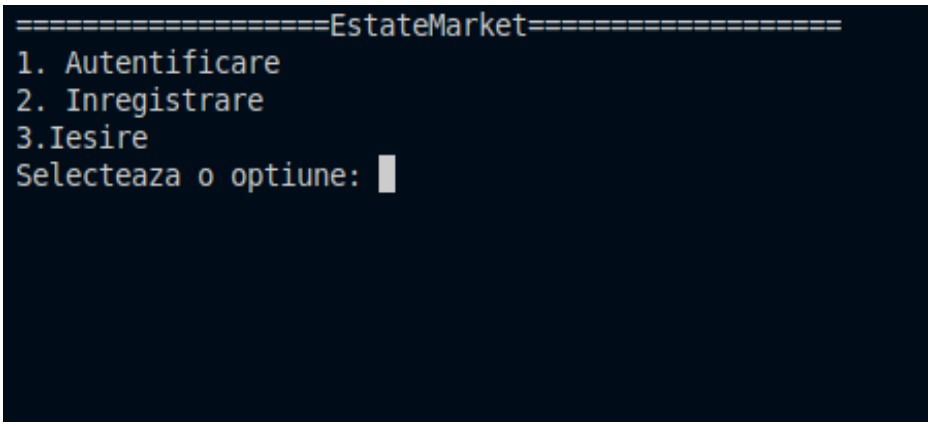
2.1. PaginaPrincipala

Aceasta clasa este cea care este apelata odata cu rularea programului. Aceasta pe langa informatiile de inceput, realizeaza si unele procese de fundal esentiale pentru functionarea aplicatiei in sine.

```java
import java.util.Scanner;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;


public  class PaginaPrincipala {

    Run | Debug | Run main | Debug main
    public static void main(String[] args){


        ServiciuAuth serviciuAuth = new ServiciuAuth();
        Scanner scanner = new Scanner(System.in);

        while(true){
        System.out.println("====================EstateMarket====================");

        System.out.println("1. Autentificare");
        System.out.println("2. Inregistrare");
        System.out.println("3.Iesire");
        System.out.print("Selecteaza o optiune: ");
        int optiune = scanner.nextInt();

        switch(optiune){
            case 1 -> serviciuAuth.Autentificare();
            case 2 -> serviciuAuth.Inregistrare();
            case 3 -> System.out.println("La revedere!");
            default -> System.out.println("Optiune invalida. Te rog incearca din nou.");
        }
```

Dupa cum se poate observa, prima parte a paginii principale contine 4 importuri necesare, Scanner pentru citire de la tastatura, iar celelalte trei sunt utile pentru rularea in timp real a unor functii.

In functia main a clasei este declarata o clasa numita “ServiciuAuth” care va realiza functiile de inregistrare si autentificare, si clasa Scanner necesara pentru citirea de la tastatura.
Liniile urmatoare realizeaza interfata ce o vede utilizatorul , direct in terminal, interfata ce contine: titlul aplicatiei, 3 optiuni din care utilizatorul poate alege, si functionalitatea alegerii(utilizatorul poate alege 1,2 sau 3)



Cea de-a doua parte a acestei clase se axeaza mai mult pe functionalitatea backend a aplicatiei.

Titlul aplicatiei este unul sugestiv, EstateMarket, format din doua cuvinte.(Estate – proprietate imobiliara, si Market- magazin).

```
ConfigDB.activeazaWAL();
ProprietateDAO dao = new ProprietateDAO();
UtilizatorDAO udao = new UtilizatorDAO();
TranzactiiDAO tdao = new TranzactiiDAO();

ServiciuChirii ServiciuChirii = new ServiciuChirii(udao, tdao, dao);
ScheduledExecutorService scheduler =  Executors.newScheduledThreadPool(10);

scheduler.scheduleWithFixedDelay(()->{

    ServiciuChirii.proceseazaChirii();

}, 0, 30, TimeUnit.SECONDS);
```

Clasa “PaginaPrincipala.java” se ocupa de asemenea cu activarea caracteristicii WAL(Write Ahead Logging) esentiala pentru eficientizarea gestionarii bazei de date, instantierea claselor de tip DAO(Data Access Object) menite pentru izolarea functiilor sql de restul aplicatiei, si rularea procesului de bussiness, folosind un thread (scheduler – clasa ScheduledExecutorService care gestioneaza threaduri in mode repetat) care ruleaza o data la 30 de secunde functia “proceseazaChirii” din clasa “ServiciuChirii”.

Aceste clase mentionate vor fi descrise in paginile urmatoare, intr-un mod detaliat.

2.2. ConfgDB

Clasa “ConfigDB” este clasa care realizeaza configuratia bazei de date si a caracteristicii WAL.

```java
import java.sql.*;



public class ConfigDB {


    private static final String url = "jdbc:sqlite:realestate.db";

    public static void activeazaWAL(){
        try(Connection conn = DriverManager.getConnection(url);
         Statement stmt = conn.createStatement()){

            stmt.execute("PRAGMA  journal_mode=WAL;");


         }catch(SQLException e){
            System.out.println("Eroare la activarea WAL: " + e.getMessage());
         }
    }
}
```

Pentru a realiza aceasta configurare este nevoie de libraria java.sql.* pentru a stabilii conexiunea cu baza de date. Astfel ,in prima linie a clasei este declarat url-ul bazei de date realestate.db care va fi conectat la aplicatie prin API-ul JDBC(Java Database Connectivity) , cu ajutorul instrumentului SqLite. Acest url va fi folosit pentru fiecare adaugare , modifica sau stergere a elementelor in sau din baza de date.

Clasa contine si functia “activeazaWAL” care executa activarea aceasta caracteristica de Write Ahead Logging. Pentru a activa aceasta caracteristica trebuie realizata conexiunea cu baza de date realestate.db ,folosind clasa **DriverManager**(Managerul de Drivere) care gestioneaza conexiunile de tip **Connection.** Rolul clasei DriverManager este cel de a identifica driverul potrivit pentru baza de date la care se vrea conexiunea pe baza unui url ca cel declarat mai sus,prin functia **getConnection()** careia ii va fi transmis url-ul ca parametru. Pe langa conexiune este nevoie si de un **Statement,** mai exact o instructie sau serie de instructiuni ,care are rolul de a fi executat.

Activarea WAL in sine se realizeaza folosind obiectul stmt de tip Statement, folosind cuvantul cheie PRAGMA pentru a specifica conexiunii ca se doreste modifcarea configurarii bazei de date, si anume schimbarea modului de scriere din RollBack(baza de date se blocheaza daca se doreste inserarea unui element nou) , in Write Ahead Logging(se scrie noile comenzi intr-un fisier separat cu extensia .wal, iar apoi se ruleaza acele comenzi).

Intreaga operatie este incadrata intr-un bloc try-catch pentru a getiona eventualele erori care pot aparea la conexiunea dintre aplicatie si baza de date).

2.3. Clasa Utilizator

Dupa cum sugereaza numele, clasa Utilizator are rolul de a crea un model abstract al unui singur utilizator, model ce poate fi replicat ori de cate ori este nevoie. Aceasta clasa este formata din 3 categorii de instructiuni, declararea elementelor unui utilizator(nume ,prenume, s.a.m.d), constructorul clasei , si crearea functiilor necesare pentru accesarea variabilelor din exteriorul de catre alte clase.

```java
public class Utilizator {
    private  String nume;
    private String prenume;
    private String numeutilizator;
    private String email;
    private String parola;
    private String tara;
    private String oras;
    private String telefon;
    private double balanta =0;
    private double profit_total=0;
    private double profit_vanzari=0;
    private double profit_chirii=0;


    public Utilizator(String nume, String prenume,String numeutilizator,String email,String parola,
    String tara,String oras,String telefon,double balanta){
            this.nume = nume;
            this.prenume = prenume;
            this.numeutilizator = numeutilizator;
            this.email = email;
            this.parola = parola;
            this.tara = tara;
            this.oras = oras;
            this.telefon = telefon;
            this.balanta = balanta;
    }

    public Utilizator(){

    }
```

In secventa de cod de mai sus, se pot observa variabilele clasei (nume, prenume, numeutilizator,...) si tot ce tine de informatiile unui utilizator, folosind cuvantul cheie **private** care are rolul de a interzice altor clase sa acceseze variabilele in mod direct. Este prezentat de asemena si constructorul, o metode esentiala pentru programarea pe obiecte. Acest constructor este apelat automat la fiecare instantiere de tip Utilizator, si tine rolul de “reteta”. El pregateste obiectul pentru utilizare, setand valorile necesare.

Astfel, constructorul Utilizator primeste ca parametrii toate variabilele de care are nevoie din exterior(ex. String nume, String email), si atribuie aceste atribute variabilelor clasei folosind cuvantul cheie **this.**

In afara acestui constructor ,este declarat un alt constructor complet gol, ceea ce poate oferi acestei clase mai multe tipui de apelare.Clasa poate functiona atat cu parametrii cat si cu un constructor gol, in functie de necesitate.

```java
public void setNume(String nume){
    this.nume = nume;

}

public void setPrenume(String prenume){
    this.prenume = prenume;
}

public void setNumeUtilizator(String numeutilizator){
    this.numeutilizator = numeutilizator;
}

public void setEmail(String email){
    this.email = email;
}

public void setParola(String parola){
    this.parola = parola;
}
```

Functii Setter

Urmatoarea secventa de cod pe care am scris-o in clasa Utilizator este cea de functii setter si getter necesare pentru a accesa informatiile utilizatorului dintr-o clasa externa. Functiile setter sunt de tip void si primesc un argument din exterior, iar cele getter returneaza informatia ceruta.

```java
public String getNume(){
    return nume;
}

public String getPrenume(){
    return prenume;
}

public String getNumeUtilizator(){
    return numeutilizator;
}

public String getEmail(){
    return email;

}
public String getParola(){
    return parola;
}
```

Functii getter

Aceasta metoda de accesare a informatiilor dintr-o clasa se numeste Incapsulare si este unul dintre conceptele fundamentale OOP. Este folosit pentru a proteja informatiile de accesari accidentale si pentru a oferi o logica mai buna in dezvoltarea aplicatiei.

2.4.Clasa “model_proprietate”

Aceasta clasa are aceeasi structura ca si clasa Utilizator.Singura diferenta este modificarea campurilor pentru informatiile proprietatii. Acest model de proprietate va fi util pentru instantierea si adaugarea de noi proprietati in aplicatie.

```
public class model_proprietate {
    private int ID;
    private String titlu;
    private String descriere;
    private String locatie;
    private double pret;
    private String tipProprietate;
    private double suprafata;
    private  String numeUtilizator;
    private boolean deInchiriat;
    private boolean inchiriat= false;
    private boolean deVanzare=false;
    private String status;
    private double profit;

    public model_proprietate(){

    }
```

Metodele getter si setter contin aceeasi structura ca in clasa Utilizator.

```
public void setID(int ID){
    this.ID = ID;
}

public void setStatus(String status){
    this.status = status;
}
```

```java
public int getID(){
    return ID;
}

public String getStatus(){
    return status;
}
```

2.5. Clasa ServiciuAuth

Clasa ServiciuAuth joaca un rol critic in cadrul aplicatiei, aceasta implementand logica de inregistrare si autentificare ,folosind de asemenea si baza de date. Clasa contine doua metode pentru fiecare actiune(sign-up sau sign-in).

Inainte de implementarea celor doua metode este nevoie de anumiti pasi cum ar fi importul librariilor necesare si declararea campurilor ce urmeaza a fi utilizate.

```java
import java.sql.*;
import java.util.Scanner;


public class ServiciuAuth {

    private final Scanner scanner = new Scanner(System.in);

    private String nume;
    private String prenume;
    private String numeutilizator;
    private String parola;
    private String email;
    private String tara;
    private String oras;
    private String telefon;
    private double balanta = 0;




    String url = "jdbc:sqlite:realestate.db";
```

Link-ul pentru baza de date , cum a fost specificat anterior, este necesar pentru conexiunea la baza de date.

Metoda de intregistrare:

```
public void Inregistrare(){
    try(Connection conn = DriverManager.getConnection(url)){

    System.out.println("=====================Inregistrare==================");
    System.out.println("Nume:");
    nume = scanner.nextLine();
    System.out.println("Prenume:");
    prenume = scanner.nextLine();
    System.out.print("Nume utilizator: ");
    numeutilizator =scanner.nextLine();
    System.out.print("Parola: ");
    parola= scanner.nextLine();
    System.out.print("Email: ");
    email = scanner.nextLine();
    System.out.print("Tara:");
    tara = scanner.nextLine();
    System.out.print("Oras:");
    oras = scanner.nextLine();
    System.out.print("Numar de telefon:");
    telefon = scanner.nextLine();
```

Aceasta metoda presupune in prima faza,citirea tuturor datelor de intregistrare de la tastatura, si salvarea acestora in campurile declarate la inceputul clasei.

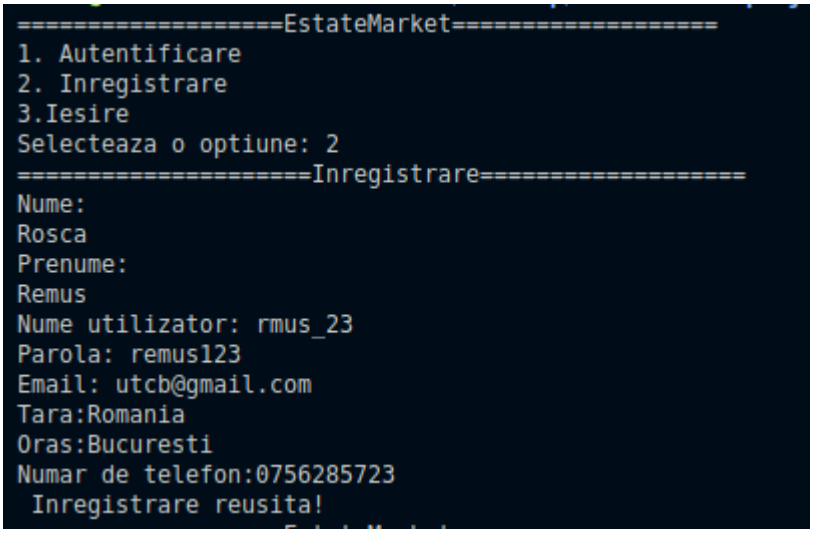
```
String sql = "INSERT INTO Utilizatori(Nume, Prenume,NumeUtilizator,Parola, Email, Tara, Oras, Telefon,Balanta) VALUES(?,?,?,?,?,?,?,?,?)";


PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setString(1,nume);
    pstmt.setString(2,prenume);
    pstmt.setString(3, numeutilizator);
    pstmt.setString(4,parola);
    pstmt.setString(5,email);
    pstmt.setString(6,tara);
    pstmt.setString(7,oras);
    pstmt.setString(8,telefon);
    pstmt.setDouble(9,balanta);


    pstmt.executeUpdate();
    System.out.println(" Inregistrare reusita!");

    conn.close();


}catch (SQLException e) {
    System.out.println(" Eroare la conectare: " + e.getMessage());
}
```

Urmatoarea secventa de cod din metoda prezinta logica de inregistrare in baza de date a utilizatorului.Prima linie contine codul SQL necesar pentru introducerea unui item in tabelul numit Utilizatori.Acest tabel contine 9 campuri carora trebuie atribuite 9 valori. Aceste valori sunt setate folosind un PreparedStatement. Diferenta dintre PreparedStatement si Statement este ca PreparedStatement asteapta manual parametrii pentru a completa instructiunea SQL. Acest lucru ajuta la securitatea bazei de date, prevenind atacurile de tip SQL Injection.

Acest Statement va primi instructiunea SQL iar apoi va seta fiecarei valori(notata cu ?),informatia corecta folosind metoa **set(** exemplu: pentru valoare numelui se va folosi metoda setString deoarece este un sir de caractere , iar acestei metode ii vor fi date 2 parametrii, primul pentru pozitia unde va fi introdus Stringul, iar al doilea pentru Stringul care trebuie transmise

codului SQL Dupa ce au fost introduse toate informatiile in instructiunea SQL, acea instructiune va fi executata folosind metoda **executeUpdate().**

Intreaga secventa de cod a functiei Inregistrare este incapsulata intr-un block try-catch pentru gestionarea erorilor de conectare si de scriere de la tastatura.



Mai sus am atasat un exemplu de inregistrare care este finalizata cu mesajul “Inregistarre reusita!”. Interfata de inregistrare este una cat se poate de simpla, fiind realizata in terminalul IDE-ului.

	Nume	Prenume	NumeUtilizator	Parola	Email	Tara	Oras	Telefon	Balanta
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	Rosca	Remus	rmus_23	remus123	utcb@gmail.com	Romania	Bucuresti	756285723	0.0

Se poate observa ca inregistrarea a fost intradevar realizata cu succes, utilizatorul fiind introdus in baza de date , mai exact in tabelul utilizatorilor. Balanta utilizatorului este automat setata la 0.0.

Metoda de autentificare:

Spre deosebire de metoda inregistrare, aceasta metoda are rolul de a verifica informatiile furnizate de utilizator in baza de date a utilizatorilor inregistrati pentru a efectua autentificarea acestuia in aplicatie.

```java
public void Autentificare(){
    System.out.println("====================Autentificare==================");
    System.out.print("Nume utilizator: ");
    numeutilizator =scanner.nextLine();
    System.out.print("Parola: ");
    parola= scanner.nextLine();

    String sql = "SELECT * FROM Utilizatori WHERE NumeUtilizator = ? AND Parola = ?";

   try(Connection conn = DriverManager.getConnection(url);
        PreparedStatement pstmt = conn.prepareStatement(sql)){

            pstmt.setString(1, numeutilizator);
            pstmt.setString(2, parola);

            ResultSet rs = pstmt.executeQuery();
            if(rs.next()){
                System.out.println("Autentificare reusita!");
                Utilizator utilizator = new Utilizator(rs.getString("Nume"),
                rs.getString("Prenume"),
                rs.getString("NumeUtilizator"),
                rs.getString("Email"),
                rs.getString("Parola"),
                rs.getString("Tara"),
                rs.getString("Oras"),
                rs.getString("Telefon"),
                rs.getDouble("Balanta"));

                Autentificat au = new Autentificat(utilizator);
                au.meniu();


            }else{
                System.out.println(" Nume utilizator sau parola incorecta.");
            }
```

Structura metodei este similara cu cea a metodei de inregistrare. Conexiunea la baza de date se realizeaza in aceeasi maniera, folosind DriverManager si PreparedStatemen.
Diferentele constau in informatiile transmise de utilizator(numele utilizatorului si parola), si in instructiunea SQL. Daca se gaseste in baza de date un utilizator cu numele si parola corespunzatoate se va crea un obiect utilizator caruia i vor fi atribuite toate informatiile din baza de date folosind metoda **getString()** din clasa PreparedStatement. De asemenea codul creeaza un obiect de tip Autentificat din care se va apela metoda **meniu().**Aceasta metoda face practic transferul dintre pagina de autentificare si pagina principala a utilizatorului autentificat.
In cazul in care nu se gaseste in baza de date nici un utilizator cu informatiile corespunzatoare, codul returneaza un mesaj specific.

```
==================EstateMarket==================
1. Autentificare
2. Inregistrare
3.Iesire
Selecteaza o optiune: 1
====================Autentificare==================
Nume utilizator: rmus_23
Parola: remus123
Autentificare reusita!
```

In exemplul de mai sus se observa ca aceasta metoda este functionala iar autentificarea este realizata cu succes folosind credentialele salvate in baza de date.

In cazul in care informatiile transmise nu se afla in baza de date aplicatia returneaza un mesaj corespunzator.

```
====================Autentificare==================
Nume utilizator: marcel
Parola: marcel
 Nume utilizator sau parola incorecta.
```

2.6. Clasa ProprietateDAO

Clasa ProprietateDAO este responsabila pentru comunicarea directa cu baza de date SQLite. Aceasta incapsuleaza toata logica SQL, oferind restului aplicatiei metode simple de manipulare a obiectelor de tip model_proprietate. Aceasta clasa va realiza de fapt manipularea in diefrite forme ale proprietatilor inregistrare in baza de date.

```java
import java.sql.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class ProprietateDAO {

    Scanner scanner = new Scanner(System.in);


    private  int ID=0;
   private final String url = "jdbc:sqlite:realestate.db?busy_timeout=5000";

    public ProprietateDAO(){

    }
```

Clasa este structurata in mai multe metode cum ar fi metoda de afisare a proprietatilor, sau metoda de filtrare.

Metoda de afisare:

Aceasta este o metoda utilitara interna care proceseaza rezultatele returnate de SQL.

```
public void afisareRS(ResultSet rs,List<model_proprietate> proprietati) throws SQLException{
    int contor = 0;
    while(rs.next()){
        contor++;
        model_proprietate prop = new model_proprietate();

        prop.setTitlu(rs.getString("Titlu"));
        prop.setDescriere(rs.getString("Descriere"));
        prop.setLocatie(rs.getString("Locatie"));
        prop.setPret(rs.getDouble("Pret"));
        prop.setTipProprietate(rs.getString("TipProprietate"));
        prop.setSuprafata(rs.getDouble("Suprafata"));
        prop.setNumeUtilizator(rs.getString("NumeUtilizator"));
        prop.setID(rs.getInt("ID"));
        prop.setProfit(rs.getDouble("Profit"));

        String tipCumparare = rs.getString("TipCumparare");

        if(tipCumparare != null)
            if(tipCumparare.equalsIgnoreCase("Inchiriere"))
                prop.setDeInchiriat(optiune: true);
            else if(tipCumparare.equalsIgnoreCase("Vanzare"))
                prop.setDeVanzare(optiune: true);

        prop.setStatus(rs.getString("Status"));
        proprietati.add(prop);


        if(prop.getDeInchiriat()==true)
            System.out.printf("%d %s | %s | %.2f EUR /luna | %s| %s | %s | %s %n",contor,
        prop.getTitlu(), prop.getLocatie(), prop.getPret(), prop.getTipProprietate(),"inchiriere",prop.getNumeUtilizator(),prop.getStatus());

        else if(prop.getDeVanzare()==true)
            System.out.printf("%d %s | %s | %.2f EUR | %s | %s | %s | %s %n",contor,
        prop.getTitlu(), prop.getLocatie(), prop.getPret(), prop.getTipProprietate(),"vanzare",prop.getNumeUtilizator(),prop.getStatus());

    }
}
```

In esenta ,aceasta metoda extrage informatiile din fiecare coloana a tabelului (ID, Titlu, Pret, etc.) si creeaza un obiect Java. De asemenea, include logica de afisare diferentiata in consola pentru anunturile de tip “Vanzare” fata de cele de tip "Inchiriere" (ex: adaugarea sufixului "/luna" la pret).

Metoda de filtrare:

Aceasta metoda permite utilizatorilor sa caute proprietati bazate pe anumite criterii.

```
public void Filtrare(String tip_proprietate , String locatie, double suprafata, double pret){


    List<model_proprietate> proprietati_filtrate = new ArrayList<>();

    double minSuprafata = suprafata -30;
    double maxSuprafata = suprafata +30;
    double minPret = pret -1500;
    double maxPret = pret +1500;

    String sql = "SELECT * FROM Proprietati WHERE TipProprietate = ? AND Locatie = ? AND Suprafata BETWEEN ? AND ? AND Pret BETWEEN ? AND ?";
```

Pregatirea acestei functii consta in ,declararea antetului functiei(functia contine 4 parametrii), declararea unui vector de tipul “model_proprietate” care va stoca toate proprietatile

cautate, si cele 4 variabile du[a care se va efectua filtrarea. De asemenea este nevoie de instructiunea SQL

```
try(Connection conn = DriverManager.getConnection(url);
    PreparedStatement pstmt = conn.prepareStatement(sql)){

        pstmt.setString(1,tip_proprietate);
        pstmt.setString(2,locatie);
        pstmt.setDouble(3,minSuprafata);
        pstmt.setDouble(4,maxSuprafata);
        pstmt.setDouble(5,minPret);
        pstmt.setDouble(6,maxPret);
        System.out.println("Lista proprietatilor disponibile:");

        ResultSet rs = pstmt.executeQuery();

        afisareRS(rs, proprietati_filtrate);
```

Filtrarea in sine sed realizeaza folosind PreparedStatement care transmite informatia SQL catre baza de date,setarea parametrilor in instructiunea SQL, executarea instructiunii si afisarea proprietatilor cerute folosind metoda de afisare prezentata mai sus.

In aceeasi maniera se realizeaza toate functiile legate de gestionarea proprietatilor(de exemplu adaugarea de proprietati,stergerea, actualizarea proprietatilor, afisarea proprietatilor detinute de utilizator,proprietatile care contin chirii active,actualizarea cumparatorului,si actualizarea profitului).

2.7. Clasa Tranzactii

Pe langa utilizatori si proprietati, orice aplicatie de acest tip are nevoie de un sistem de tranzactionare. In aplicatia EstateMarket,pentru a putea realiza tranzactii este nevoie de un sablon al unui astfel de proces. Aici intervine clasa Tranzactii, care are rolul de a crea campuri privind numele cumparatorului, numele vanzatorului, pretul tranzactiei etc.

```
public class Tranzactii {

    private String vanzator;
    private String cumparator;
    private String titlu_proprietate;
    private double pret;
    private String tip_achizitie;

    public Tranzactii(){

    }
```

Urmand functionalitatea clasei Utilizator sau clasei model_proprietate, clasa Tranzactii are nevoie de metode de accesare sau setare a acestor campuri.

```
public String getTitluProprietate(){
    return titlu_proprietate;
}

public double getPret(){
    return pret;
}

public String getTipAchizitie(){
    return tip_achizitie;
}

public String getVanzator(){
    return vanzator;
}

public String getCumparator(){
    return cumparator;
}
```

```java
public String getTitluProprietate(){
    return titlu_proprietate;
}

public double getPret(){
    return pret;
}

public String getTipAchizitie(){
    return tip_achizitie;
}

public String getVanzator(){
    return vanzator;
}

public String getCumparator(){
    return cumparator;
}
```

2.8. Clasa TranzactiiDAO

Rolul acestei clase ,asa cum reiese din denumire, este de a gestiona fiecare tranzactie efectuata pe aplicatie, indiferent de utilizator. Astfel , aplicatia poate extrage datele corespunzatoare procesarii chiriilor spre exemplu, sau transferul de proprietar in cazul in care proprietatea a fost cumparata definitiv.

```java
public class TranzactiiDAO {

    private final String url = "jdbc:sqlite:realestate.db?busy_timeout=5000";

    public void TransferProprietar(String nume_cumparator, int ID){
        String sql = "UPDATE Proprietati SET NumeUtilizator = ? WHERE ID =?";

        try(Connection conn = DriverManager.getConnection(url);
            PreparedStatement pstmt = conn.prepareStatement(sql)){

                pstmt.setString(1, nume_cumparator);
                pstmt.setInt(2,ID);

                pstmt.executeUpdate();


            }catch(SQLException e){
                System.out.println("Eroare la conectare: "+ e.getMessage());
            }
    }
```

La fel ca si clasa ProprietateDAO ,se incepe declarand linkul bazei de date catre care se va realiza conexiunea.
Prima functie este cea de transfer a proprietarului, care modifica numele de utilizator a vechiului proprietar cu numele noului proprietar , imediat dupa ce tranzactia a fost efectuata.

```java
public void InregistrareTranzactie(String nume_vanzator, String nume_cumparator,
                                   String titlu_proprietate,String tip_proprietate, double suma_tranzactie,
                                       boolean  vanzare, boolean inchiriere){


    String sql = "INSERT INTO Tranzactii(NumeVanzator ,TitluProprietate, TipProprietate, PretAchizitie, NumeCumparator, TipAchizitie) VALUES(?,?,?,?,?,?

    try(Connection conn = DriverManager.getConnection(url);
        PreparedStatement pstmt = conn.prepareStatement(sql)){

            pstmt.setString(1,nume_vanzator);
            pstmt.setString(2,titlu_proprietate);
            pstmt.setString(3,tip_proprietate);
            pstmt.setDouble(4,suma_tranzactie);
            pstmt.setString(5,nume_cumparator);
            if(vanzare) pstmt.setString(6, "Vanzare");
            else if(inchiriere) pstmt.setString(6, "Inchiriere");

            pstmt.executeUpdate();
```

A doua functie, care este esentiala in randul tranzactiilor, este **InregistrareTranzactie** care salveaza tranzactia nou efectuata in baza de date(nume vanzator, nume cumparator , titlu proprietate, pret de vanzare/inchiriere etc).
In exemplul de mai jos se observa tranzactiile inregistrare in baza de date.

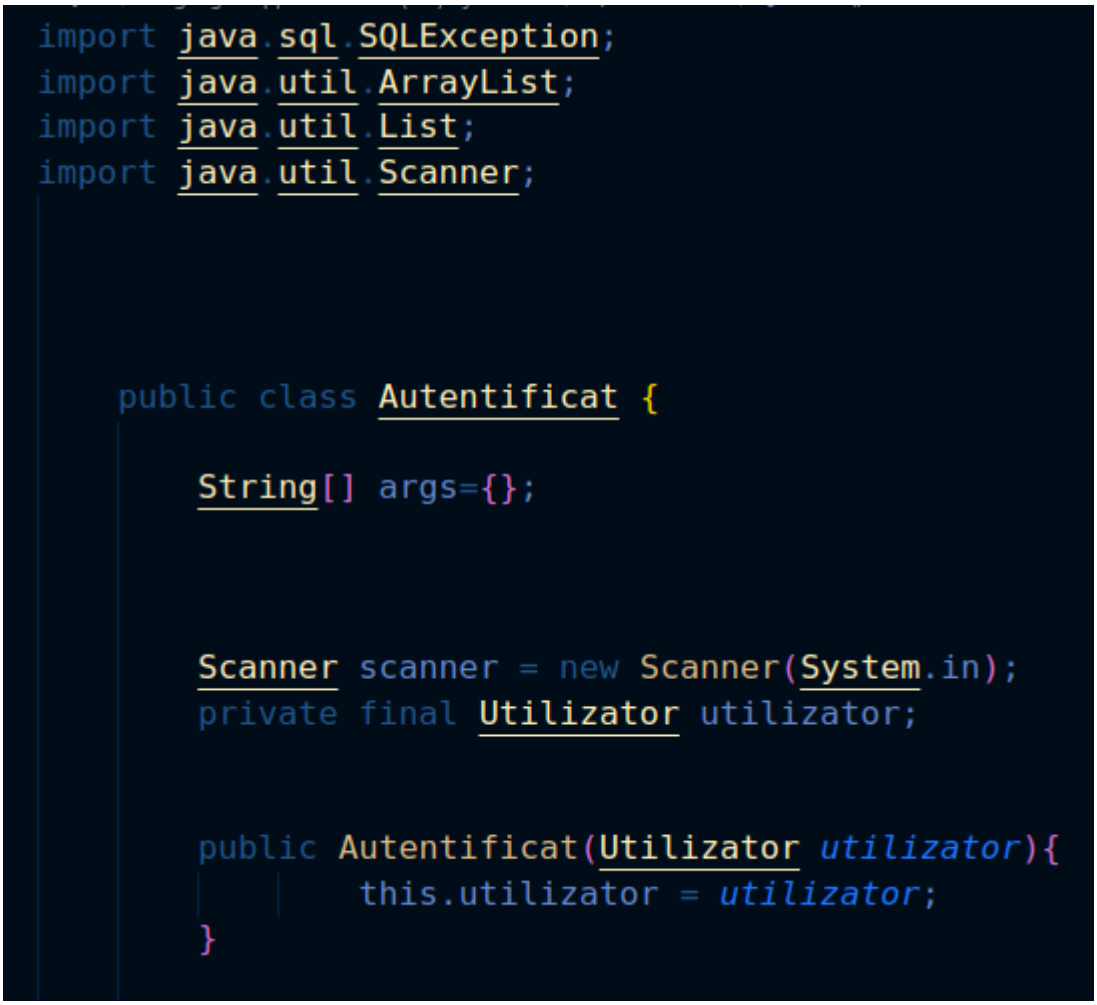
ID	NumeVanzator	TitluProprietate	TipProprietate	PretAchizitie	NumeCumparator	TipAchizitie
Filter	Filter	Filter	Filter	Filter	Filter	Filter
7	denisa123	Casa Vacante Parang	Rezidentiala	60000.0	rmus_23	Vanzare
8	denisa123	Casa Vacante Parang	Rezidentiala	60000.0	rmus_23	Vanzare
9	denisa123	Casa Vacante Parang	Rezidentiala	60000.0	rmus_23	Vanzare
10	rmus_23	Garsoniera Centrala	Rezidentiala	55000.0	denisa123	Vanzare
11	denisa123	Garsoniera Centrala	Rezidentiala	55000.0	rmus_23	Vanzare
12	rmus_23	Garsoniera Centrala	Rezidentiala	300.0	denisa123	Inchiriere
13	denisa123	Garsoniera Centrala	Rezidentiala	30000.0	rmus_23	Vanzare

```java
public List<Tranzactii>  AfisareTranzactii(String nume_utilizator){
    List<Tranzactii> tranzactii = new ArrayList<>();
    String sql = "SELECT * FROM Tranzactii WHERE NumeCumparator = ? OR NumeVanzator = ?";

    try(Connection conn = DriverManager.getConnection(url);
        PreparedStatement pstmt = conn.prepareStatement(sql)){
            pstmt.setString(1,nume_utilizator);
            pstmt.setString(2,nume_utilizator);

            ResultSet rs = pstmt.executeQuery();

            while(rs.next()){
                Tranzactii tranzactie = new Tranzactii();

                tranzactie.setTitluProprietate(rs.getString("TitluProprietate"));
                tranzactie.setVanzator(rs.getString("NumeVanzator"));
                tranzactie.setCumparator(rs.getString("NumeCumparator"));
                tranzactie.setPret(rs.getDouble("PretAchizitie"));
                tranzactie.setTipAchizitie(rs.getString("TipAchizitie"));

                tranzactii.add(tranzactie);
            }


        }catch(SQLException e){
            System.out.println("Eroare la conectare: "+ e.getMessage());
        }


        return tranzactii;
}
```

Ultim functie a acestei clase este cea de afisare a tranzactiilor unei anumite proprietati.Aceasta functie ajuta in momentul in care un utilizator doreste sa isi vada si sa verifice tranzactiile pe care le-a efectuat(vanzator sau cumparator).

2.9. Clasa Autentificat

Aceasta clasa este clasa motor in functionarea aplicatiei, Este cea mai complexa clasa din punct de vedere a codului scris si al logicii din spate.
A doua functie, care este esentiala in randul tranzactiilor, este **InregistrareTranzactie**
care salveaza tranzactia nou efectuata in baza de date(nume vanzator, nume cumparator , titlu proprietate, pret de vanzare/inchiriere etc).



Inceputul clasei este unul simplu. Se observa implementarea claselor de care este nevoie pentru functionarea aplicatiei(SQLException pentru gestionarea erorilor bazei de date, ArrayList si List pentru stocarea informatiilor extrase din baza de date, iar libraria Scanner pentru citirea datelor de la tastatura.)

Se observa declararea unui sir de Stringuri numit args care va fi necesar pentru apelarea ulterioara a functiei principale din clasa.

Constructorul clasei contine ca si parametru un obiect de tipul Utilizator, obiect ce ii este transmis din functia autentificare impreuna cu toate atributele si informatiile utilizatorului autentificat.

```java
public void meniu() throws SQLException{
    System.out.println();
    System.out.println();
    System.out.println();
    System.out.println("Bine ai venit "+ utilizator.getNume() + " " + utilizator.getPrenume());
    System.out.println("==================EstateMarket====================");

    ProprietateDAO dao = new ProprietateDAO();


    while (true) {
        System.out.println("1.Afiseaza proprietati");
        System.out.println("2.Cauta proprietate");
        System.out.println("3.Adauga proprietate");
        System.out.println("4.Proprietatile mele");
        System.out.println("5.Profilul meu");
        System.out.println("6.Balanta");
        System.out.println("7.Deconectare");

        int optiune = scanner.nextInt();
        scanner.nextLine();
```

Functia principala a clasei Autntificat este meniul aplicatiei. In aceasta functie vor fi gestionate toate actiunile pe care un utilizator le realizeaza in contul lui.
Ca orice aplicatie, EstateMarket isi saluta utilizatorul autentificat intr-o maniera prietenoasa.

Meniul in sine este format din 7 optiuni puse la dispozitia utilizatorului. Intregul meniu este incapsulat intr-o bucla **while(true)** pentru ca aplicatia sa functioneze pana la inchiderea procesului.

Se poate observa ca aplicatia asteapta de la utilizator introducerea unui numar de la 1 la 7 pentru a-si alege optiunea.

Pentru ca programul sa realizeze instructiunile pentru optiunea data de utilizator, am folosit structura **switch-case** structura esentiala in orice limbaj de programare.

Optiunea 1: Afisarea de Proprietati si cumpararea /inchirierea acestora

```
System.out.println("==================Proprietati==================");
List<model_proprietate> lista_proprietati = dao.AfisajProprietati();
System.out.println("\n\n");

System.out.println("1.Selectare proprietate");
System.out.println("2.Inapoi");
int optiune_proprietati = scanner.nextInt();
scanner.nextLine();


switch(optiune_proprietati){
    case 1 -> {
        System.out.println(lista_proprietati.size());
        System.out.println("Introdu numarul proprietatii dorite:");
         int numar_proprietate = scanner.nextInt();
        scanner.nextLine();
        if(numar_proprietate>=0 && numar_proprietate<=lista_proprietati.size()){
            model_proprietate prop_selectata = lista_proprietati.get(numar_proprietate-1);
            System.out.println(prop_selectata.getID());
            afisareAnunt(prop_selectata, numar_proprietate);
            System.out.println("Doriti achizitionarea/inchirierea acestei proprietati?");
            System.out.println("1.Da");
            System.out.println("2.Nu");

            int optiune_achizitie = scanner.nextInt();
            scanner.nextLine();

            if(optiune_achizitie==1){

                if(prop_selectata.getDeVanzare())
                    procesareVanzare(prop_selectata,dao);
                else if(prop_selectata.getDeInchiriat())
                    procesareChirie(prop_selectata, dao);

            }
            else if(optiune_achizitie ==2){
                System.out.println("Tranzactie anulata!");
                meniu();
```

Pentru a selecta o anumita proprietate este nevoie de crearea unui vector care va contine toate anunturile publicate. Acest lucru se face folosind functia **dao.AfisajProprietati(),** unde dao este o instanta a **clasei** ProprietateDAO.

Selectarea se realizaeaza selectand optiunea **Selectare Proprietate** iar apoi scrierea numarului proprietatii dorite.

```
Lista proprietatilor disponibile:
1 Apartament 2 camere Centru Vechi | Bucuresti | 60000.00 EUR | Apartament | vanzare | rmus_23 | ACTIV
2 Vila Contemporana | Snagov | 250000.00 EUR | Apartament | vanzare | denis12 | ACTIV

Accounts

1.Selectare proprietate
2.Inapoi
```

In acest exemplu am creat 2 proprietati pe care le-am publicat(cuvantul ACTIV la final).

```
==================Proprietati==================
Lista proprietatilor disponibile:
1 Apartament 2 camere Centru Vechi | Bucuresti | 60000.00 EUR | Apartament | vanzare | rmus_23 | ACTIV
2 Vila Contemporana | Snagov | 250000.00 EUR | Apartament | vanzare | denis12 | ACTIV



1.Selectare proprietate
2.Inapoi
1
2
Introdu numarul proprietatii dorite:
2
2


Ai ales proprietatea numarul:2
Detalii proprietate:
Titlu: Vila Contemporana
Descriere: 4 camere
Locatie: Snagov
Pret: 250000.0 EUR
Tip proprietate: Apartament
Suprafata: 100.0
Proprietate adaugata de utilizatorul: denis12



Doriti achizitionarea/inchirierea acestei proprietati?
1.Da
2.Nu
```

In momentul in care utilizatorul alege o proprietate aplicatia afiseaza toate informatiile proprietatii respective, inclusiv locatia, tipul de proprietate, suprafat si utilizatorul care a public anuntul respectiv(in acest caz **denis12**).

Dupa afisarea informatiilor aplicatia ofera utilizatorului posibilitatea de a achizitionarea sau inchirierea proprietatii(in functie de modul in care a fost publicata).

In cazul in care utilizatorul doreste achizitionarea, aplicatia va apela metoda **procesareVanzare()** sau **procesareChirie().**

```java
public void procesareVanzare(model_proprietate prop_selectata,ProprietateDAO dao){

    UtilizatorDAO cumparatordao = new UtilizatorDAO(utilizator);
    UtilizatorDAO vanzatordao = new UtilizatorDAO();
    Utilizator vanzator = vanzatordao.ReturneazaUtilizator(prop_selectata.getNumeUtilizator());

    if(utilizator.getBalanta() < prop_selectata.getPret()) System.out.println("Fonduri insuficiente!");

    else{

    tdao.InregistrareTranzactie(vanzator.getNumeUtilizator(), utilizator.getNumeUtilizator(),
    prop_selectata.getTitlu(), prop_selectata.getTipProprietate(),prop_selectata.getPret(), prop_selectata.getDeVanzare()
    , prop_selectata.getDeInchiriat());
    cumparatordao.RetragereBani(prop_selectata.getPret(), utilizator.getNumeUtilizator());
    vanzatordao.AdaugareBani(prop_selectata.getPret(), vanzator.getNumeUtilizator());
    utilizator.setBalanta(cumparatordao.BalantaActualizata(utilizator.getNumeUtilizator()));
    prop_selectata.setStatus(status: "INACTIV");
    dao.AcualizareStatus(prop_selectata.getStatus(), prop_selectata.getID());
    tdao.TransferProprietar(utilizator.getNumeUtilizator(),prop_selectata.getID());
    System.out.println("Tranzactie efectuata cu succes! Accesati Proprietatile mele pentru a vedea noua ta proprietate.");
    }


}
```

Pentru ca vanzarea sa se efectueze este nevoie de DAO pentru utilizator si vanzator si de obiectul de tip utilizator pentru vanzator.

In cazul in care balanta cumparatorului(in cazul nostru utilizatorul conectat) este mai mica decat pretul proprietatii , se afiseaza un mesaj corespunzator “Fonduri Insuficiente”.

```
Ai ales proprietatea numarul:2
Detalii proprietate:
Titlu: Vila Contemporana
Descriere: 4 camere
Locatie: Snagov
Pret: 250000.0 EUR
Tip proprietate: Apartament
Suprafata: 100.0
Proprietate adaugata de utilizatorul: denis12



Doriti achizitionarea/inchirierea acestei proprietati?
1.Da
2.Nu
1
Fonduri insuficiente!
```

In caz opus se incepe operatia de inregistrare tranzactie,retragerea banilor din contul utilizatorului, adaugarea sumei in contul vanzatorului, actualizarea statusului proprietatii(din ACTIV in INACTiV pentru a nu mai fi vizibil pe lista proprietatilor disponibile). Dupa efectuarea operatiilor se afiseaza un mesaj corespunzator.

```
public void procesareChirie(model_proprietate prop_selectata, ProprietateDAO dao){

    try{

    UtilizatorDAO cumparatordao = new UtilizatorDAO(utilizator);
    UtilizatorDAO vanzatordao = new UtilizatorDAO();
    Utilizator vanzator = vanzatordao.ReturneazaUtilizator(prop_selectata.getNumeUtilizator());



        tdao.InregistrareTranzactie(vanzator.getNumeUtilizator(), utilizator.getNumeUtilizator(),
        prop_selectata.getTitlu(), prop_selectata.getTipProprietate(),prop_selectata.getPret(), prop_selectata.getDeVanzare()
        , prop_selectata.getDeInchiriat());
         cumparatordao.RetragereBani(prop_selectata.getPret(), utilizator.getNumeUtilizator());
         vanzatordao.AdaugareBani(prop_selectata.getPret(), vanzator.getNumeUtilizator());
         utilizator.setBalanta(cumparatordao.BalantaActualizata(utilizator.getNumeUtilizator()));
         prop_selectata.setStatus(status: "CHIRIE");
         prop_selectata.setProfit(prop_selectata.getPret());
         dao.AcualizareStatus(prop_selectata.getStatus(), prop_selectata.getID());
         dao.ActualizareCumparator(utilizator.getNumeUtilizator(), prop_selectata.getID());
         dao.ActualizareProfit(prop_selectata.getProfit(), prop_selectata.getID());
    }catch(Exception e){
        e.printStackTrace();
    }
```

In mod similar se efectueaza si procesarea chiriei, insa acest proces va fi efectuat o singura data in momentul in care utilizatoul inchiriaza proprietatea. Continuarea platii chiriilor se va face intr-o alta clasa.

In aceasta metoda,numele proprietarului nu va fi modificat. Vor modificate insa, statusul proprietatii(CHIRIE), si se va adauga numele chiriasului.

Optiunea 2: Filtrare Proprietati

Aceasta optiune cere de la utilizator tip de proprietate,locatie, o marja de pret, si o marja de suprafata dupa care realizeaza filtrarea.

```java
case 2 -> {
    System.out.println("====================CautareProprietate====================");

    System.out.println("Introduceti datele de cautare:");

    String tip_proprietate_f;
    String locatie_f;
    double suprafata_f;
    double pret_f;

    System.out.println("Tip Proprietate:");
    tip_proprietate_f = scanner.nextLine();
    System.out.println("Locatie:");
    locatie_f = scanner.nextLine();
    System.out.println("Suprafata:");
    suprafata_f = scanner.nextDouble();
    scanner.nextLine();
    System.out.println("Pret:");
    pret_f = scanner.nextDouble();
    scanner.nextLine();
    dao.Filtrare(tip_proprietate_f, locatie_f, suprafata_f, pret_f);
}
```

Pentru a incepe filtrarea, trebuie apelata metoda Filtrare din dao, metoda care a fost explicat-o anterior.

```
====================CautareProprietate====================
Introduceti datele de cautare:
Tip Proprietate:
Apartament
Locatie:
Snagov
Suprafata:
100
Pret:
250000
Lista proprietatilor disponibile:
1 Vila Contemporana | Snagov | 250000.00 EUR | Apartament | vanzare | denis12 | ACTIV
```

Optiunea 3: Adaugare Proprietate

```
case 3 -> {
    System.out.println("====================Adauga proprietate====================")

    model_proprietate proprietate = new model_proprietate();
    proprietate.setNumeUtilizator(utilizator.getNumeUtilizator());
    System.out.println();
    System.out.println();

    System.out.println("Titlu:");
    String titlu = scanner.nextLine();
    proprietate.setTitlu(titlu);

    System.out.println("Descriere:");
    String descriere = scanner.nextLine();
    proprietate.setDescriere(descriere);

    System.out.println("Locatie:");
    String locatie = scanner.nextLine();
    proprietate.setLocatie(locatie);

    System.out.println("Pret:");
    double pret = scanner.nextDouble();
    proprietate.setPret(pret);
    scanner.nextLine();

    System.out.println("TipProprietate:");
    String tipProprietate = scanner.nextLine();
    proprietate.setTipProprietate(tipProprietate);

    System.out.println("Suprafata:");
    double suprafata = scanner.nextDouble();
    proprietate.setSuprafata(suprafata);
    scanner.nextLine();

    System.out.println("Selecteaza tipul de tranzactie:");
    System.out.println("1.Vanzare");
    System.out.println("2.Inchiriere");
    int optiune_tranzactie = scanner.nextInt();
```

Aceasta secventa de cod implementeaza logica de adaugare a unui anunt, interogand utilizatorul cu intrebari cum ar fi titlu pe care doreste sa il dea noii proprietati si descrierea acesteia.

```
System.out.println("Selecteaza tipul de tranzactie:")
System.out.println("1.Vanzare");
System.out.println("2.Inchiriere");
int optiune_tranzactie = scanner.nextInt();

if(optiune_tranzactie==1){
    proprietate.setDeVanzare(optiune: true);
}

else if(optiune_tranzactie==2){
    proprietate.setDeInchiriat(optiune: true);

}

proprietate.setStatus(status: "INACTIV");



dao.AdaugaProprietate(proprietate);
```

In momentul in care utilizatorul alege un tip de tranzatie, aplicatia seteaza o valoare boolean a proprietatii care mai apoi va fi transmisa catre baza de date. Aceasta valoare este ori de vanzare ori de inchiriere.
In final, se apeleaza functia **AdaugareProprietate()** din dao.

Optiunea 4: Proprietatile Mele
Daca utilizatorul alege aceasta optiune, aplicatia ii ofera o lista cu toate proprietatile detiunte si cele date spre inchiriere.

```
case 4 -> {

    System.out.println("==================Proprietatile mele====================");
    List<model_proprietate> lista_proprietatile_mele;
    lista_proprietatile_mele = dao.ProprietatileMele(utilizator.getNumeUtilizator());

    System.out.println("\n\n");
    System.out.println("1.Selectare proprietate");
    System.out.println("2.Inapoi");

    int optiune_proprietatile_mele  = scanner.nextInt();
    scanner.nextLine();
    if(optiune_proprietatile_mele==1){
        System.out.println("Introdu numarul proprietatii dorite:");
        int numar_proprietate = scanner.nextInt();
        scanner.nextLine();
        System.out.println(lista_proprietatile_mele.size());
        if(numar_proprietate>0 && numar_proprietate<=lista_proprietatile_mele.size()){
            model_proprietate prop_selectata = lista_proprietatile_mele.get(numar_proprieta
            afisareAnunt(prop_selectata, numar_proprietate);

            System.out.println("Optiuni:");


            if(prop_selectata.getStatus().equals("CHIRIE")){
                System.out.println("1.Anulare chirie");
                System.out.println("2.Vizualizare profit");
                System.out.println("3.Inapoi");
```

Dupa listarea proprietatilor detinute, utilizatorul are posibilitatea de a alege o proprietate. In momentul in care o proprietate a fost alesa,aplicatia ofera optiune de gestionare a acesteia.

In cazul in care este data spre chirie, utilizatorul poate anula chiria respectiva, sau poate vizualiza profitul proprietatii(doar in cazul in care este proprietarul acesteia).

```
if(prop_selectata.getStatus().equals("CHIRIE")){
    System.out.println("1.Anulare chirie");
    System.out.println("2.Vizualizare profit");
    System.out.println("3.Inapoi");

    int optiune_anunt = scanner.nextInt();
    scanner.nextLine();

    switch (optiune_anunt) {
        case 1 -> {
            System.out.println("Chirie anulata cu succes!");
            ServiciuChirii schirii = new ServiciuChirii(udao: null, tdao: null, dao);
            schirii.IncetareChirie(prop_selectata, dao);
        }
        case 2 -> {
            if(prop_selectata.getNumeUtilizator().equals(utilizator.getNumeUtilizator()))
                System.out.println("Profitul proprietatii: " + prop_selectata.getProfit()+ "EUR/n");
            else System.out.println("Nu poti vedea profitul acestei proprietati deoarece nu esti proprietarul acesteia!/n");
        }
        case 3 -> meniu();
        default -> { System.out.println("Optiune invalida! Te rog incearca din nou!");
        }
    }
}
```

Daca proprietatea este inactiva(nu este publicata in lista proprietatilor disponibile),utilizatorul are posibilitatea de a publica anuntul sau nu. Daca utilizatorul doreste publicarea unui anunt, aplicatia deschide o interfata de editare a proprietatii pentru ca utilizatorul sa modifice unele informatii dupa bunul plac. Dupa confirmarea modificarilor, procesul de

adaugare se realizeaza apelandu-se metoda **ActualizareProprietate(),** si modificandu-se statusul anuntului din **INACTIV** in **ACTIV.**

```
else if(prop_selectata.getStatus().equals("INACTIV")){


System.out.println("Doriti sa publicati anunt pentru aceasta proprietate?");
System.out.println("1.Da");
System.out.println("2.Nu");
int optiune_publicare = scanner.nextInt();
scanner.nextLine();

if(optiune_publicare == 1){
    System.out.println("==================Editare Anunt==================");
    System.out.println("Tip achizitie(1-Vanzare/2-Inchiriere):");
    int optiune_achizitie_edit = scanner.nextInt();
    scanner.nextLine();
    if(optiune_achizitie_edit ==1){
            prop_selectata.setDeVanzare(optiune: true);
            prop_selectata.setDeInchiriat(optiune: false);
                                                        }
    else if(optiune_achizitie_edit == 2){
            prop_selectata.setDeVanzare(optiune: false);
            prop_selectata.setDeInchiriat(optiune: true);
    }
    if(prop_selectata.getDeVanzare()){
        System.out.println("Pret proprietate ");
        double pret_proprietate_edit = scanner.nextDouble();
        scanner.nextLine();
        prop_selectata.setPret(pret_proprietate_edit);

        System.out.println("Doriti sa confirmati datele?(1-Da/2-Nu)");
        int optiune_confirmare = scanner.nextInt();
        scanner.nextLine();
        prop_selectata.setStatus(status: "ACTIV");
        if(optiune_confirmare==1){
            System.out.println(prop_selectata.getStatus());
            dao.ActualizareProprietate(prop_selectata.getPret(), prop_selectata.getDeVanzare(),
            prop_selectata.getDeInchiriat(), prop_selectata.getStatus(), prop_selectata.getTitlu());
```

Optiunea 5: Profil

Ca orice aplicatie sau sitweb, este nevoie de vizualizarea profilului, motivele fiind diverse(verificarea datelor, modificarea datelor care pot fi modificate etc.)

EstateMarket ofera un meniu pentru vizualizarea informatiilor din cont, dar si pentru vizualizarea Balantei din cont.

```
    case 5 -> {
        System.out.println("==================Profil==================");
        System.out.println("Nume:" + utilizator.getNume());
        System.out.println("Prenume:" + utilizator.getPrenume());
        System.out.println("NumeUtilizator:" + utilizator.getNumeUtilizator());
        System.out.println("Email:" + utilizator.getEmail());
        System.out.println("Parola:" + utilizator.getParola());
        System.out.println("Tara:"+ utilizator.getTara());
        System.out.println("Oras:"+ utilizator.getOras());
        System.out.println("Telefon:"+ utilizator.getTelefon());
        System.out.println("Balanta: "+ utilizator.getBalanta());
        System.out.println("\n\n");
    }
```

Optiunea 6: Balanta

Aceasta optiune ofera utilizatorului posibilitatea de a verifica soldul, de a adauga sau retrage bani din cont, si de a vizualiza tranzactiile efectuate.

```
        System.out.println("Oras:"+ utilizator.getOras());
        System.out.println("Telefon:"+ utilizator.getTelefon());
        System.out.println("Balanta: "+ utilizator.getBalanta());
        System.out.println("\n\n");
    }
    case 6 -> {
        System.out.println("==================Balanta====================");
        System.out.println("Balanta ta este de :"+ utilizator.getBalanta()+ " EUR");
        System.out.println("1.Tranzactiile mele");
        System.out.println("2.Adauga bani");
        System.out.println("3.Retrage bani");
        System.out.println("4.Inapoi");
        int optiune_balanta = scanner.nextInt();
        scanner.nextLine();
        UtilizatorDAO utilizatordao = new UtilizatorDAO(utilizator);

        switch(optiune_balanta){

            case 1 -> {
                System.out.println("==================Tranzactiile mele====================");
                List<Tranzactii> tranzactii = new ArrayList<>();

                tranzactii = tdao.AfisareTranzactii(utilizator.getNumeUtilizator());

                for(int i=0;i<tranzactii.size();i++){
                    Tranzactii tranzactie = tranzactii.get(i);

                    System.out.println((i+1)+" " +tranzactie.getTitluProprietate()+" | "+(
                    tranzactie.getVanzator().equals(utilizator.getNumeUtilizator())?"+" +tranzactie.getPret():"-"+tranzactie.getPret())+
                    " | " + tranzactie.getVanzator() + "-> "+ tranzactie.getCumparator() );
                }
```

```
    case 2 -> {
        System.out.println();
        System.out.println("Introdu suma dorita:");
        double suma_adaugare = scanner.nextDouble();
        scanner.nextLine();

        utilizatordao.AdaugareBani(suma_adaugare, utilizator.getNumeUtilizator());
        utilizator.setBalanta(utilizatordao.BalantaActualizata(utilizator.getNumeUtilizator()));
        System.out.println("Bani adaugati cu succes!");

    case 3 -> {
        System.out.println("Introdu suma dorita:");
        double suma_retragere = scanner.nextDouble();
        scanner.nextLine();
        utilizatordao.RetragereBani(suma_retragere, utilizator.getNumeUtilizator());
        utilizator.setBalanta(utilizatordao.BalantaActualizata(utilizator.getNumeUtilizator()));
        System.out.println("Bani retrasi cu succes!");

    case 4 ->{
            meniu();
    }

    default -> System.out.println("Optiune invalida! Te rog incearca din nou");
```

Ultima optiune este cea de deconectare, care deconecteaza utilizatorul din cont si al redirectiona catre pagina principala.

2.10. Clasa ChirieInfo

Clasa ChirieInfo are rolul de a crea un alt model pentru proprietatile aflate in chirie.Aceasta clasa este folosita impreuna cu ServiciuChirii pentru automatizarea chirilor existente, pe toata perioada

functionarii aplicatiei. EstateMarket nefiind o aplicatie ce ruleaza live 24/7 aceste procese pot fi realizate doar cand aplicatia ruleaza din terminal.

```
    private String titlu;
    private String vanzator;
    private String chirias;
    private double pret;
    private String tip_proprietate;
    private double profit;
    private int ID;



    public ChirieInfo(String titlu, String vanzator, String chirias,double pret,String tip_proprietate,int ID,double profit){
        this.titlu = titlu;
        this.chirias =chirias;
        this.vanzator = vanzator;
        this.pret =pret;
        this.tip_proprietate= tip_proprietate;
        this.ID = ID;
        this.profit = profit;

    }

    public void setProfit(double profit){
        this.profit = this.profit + profit;
    }


    public String getTitlu(){
        return titlu;
    }

    public String getVanzator(){
        return vanzator;
    }

    public String getChirias(){
        return chirias;
    }
```

Structura acestei clase este similara cu structura clasei Utilizator si prezinta aceleasi functionalitati.

2.11. Clasa ServiciuChirii

Aceasta clasa este practic motorul procesarii chiriilor “in timp real”.Contine metode precum **procesareChirii()** care realizeaza trannzactiile pentru toate chiriile existente in baza de date. De asemenea contine si metoda pentru incetarea unei anumite chirii, in functie de dorinta vanzatorului sau a cumparatorului.

```java
public void proceseazaChirii(){

try{
    List<ChirieInfo> chirii = dao.ChiriiActive();

    for(ChirieInfo chirie : chirii){


        Utilizator proprietar = udao.ReturneazaUtilizator(chirie.getVanzator());
        Utilizator chirias = udao.ReturneazaUtilizator(chirie.getChirias());

        tdao.InregistrareTranzactie(proprietar.getNumeUtilizator(),
                                    chirias.getNumeUtilizator(), chirie.getTitlu(),
                                     chirie.getTipProprietate(), chirie.getPret(), vanzare: false, inchiriere: true);

        udao.AdaugareBani(chirie.getPret(), proprietar.getNumeUtilizator());
        udao.RetragereBani(chirie.getPret(), chirias.getNumeUtilizator());
        chirie.setProfit(chirie.getPret());
        dao.ActualizareProfit(chirie.getProfit(), chirie.getID());


    }
    }catch(Exception e){
        System.out.println("Eroare la extragerea chiriilor"+ e.getMessage());
    }
}
```

```java
public void IncetareChirie(model_proprietate prop_selectata,ProprietateDAO dao){
    UtilizatorDAO vanzatordao = new UtilizatorDAO();
    Utilizator vanzator = vanzatordao.ReturneazaUtilizator(prop_selectata.getNumeUtilizator());
    prop_selectata.setStatus(status: "ACTIV");
    dao.AcualizareStatus(prop_selectata.getStatus(), prop_selectata.getID());
    dao.ActualizareCumparator(nume_cumparator: "", prop_selectata.getID());

}
```

Aceasta clasa este rulata la dechiderea aplicatiei, pentru a oferi un oarecare grad de realism aplicatiei.In viata reala chiria nu se opreste brusc, acest grad de realism nu a fost realizatf momentan).

```java
ServiciuChirii ServiciuChirii = new ServiciuChirii(udao, tdao, dao);
ScheduledExecutorService scheduler =  Executors.newScheduledThreadPool(10);

scheduler.scheduleWithFixedDelay(()->{

    ServiciuChirii.proceseazaChirii();

}, 0, 30, TimeUnit.SECONDS);
```

Acest proces este realizat in clasa PaginaPrincipala, acolo unde am activat WAL-ul si am configurat baza de date. Pentru efectuarea chiriilor am folosit un thread care ruleaza in fundal pe tot parcursul rularii programului.Acest thread denumit **scheduler** are rolul de a apela functia proceseazaChirii() din clasa ServiciuChirii o data la 30 de secunde.

Pentru ca intreaga aplicatie sa ruleze corespunzator a fost nevoie de un script cu rolul de a automatiza pregatirile necesare rularii.

Acest script este alcatuit din doua procese.Primul proces este cel de compilare al intregului proiect, adica toate fisiere java sunt mai intai compilate. Al doilea proces este rularea efectiva a aplicatiei, adica rularea PaginiiPrincipale(cea care contine meniul de inregfistrare si autentificare).

CONCLUZII SI CONTRIBUTII PERSONALE

Concluzii

EstateMarket este o aplicatie pe care am construit-o pentru a rezolva problema cautarii de locuinte. Ideea a plecat de la dorinta de a avea o platforma simpla, unde utilizatorii sa gaseasca rapid apartamente sau case, fara sa se piarda in meniuri complicate. Este un proiect la care am lucrat cu placere si care mi-a permis sa pun in practica notiunile invatate la facultate.

Contributii personale

In cadrul acestui proiect m-am ocupat de intreaga arhitectura a bazei de date folosind SQLite, unde am creat tabelele necesare si am definit relatiile dintre date pentru a evita duplicarea informatiilor. Am scris codul pentru interfata de tip text, gestionand citirea input-urilor de la tastatura si afisarea rezultatelor intr-un format lizibil in terminal. O mare parte din munca mea a constat in scrierea interogarilor SQL pentru operatiunile fundamentale, adica adaugarea de proprietati noi, vizualizarea celor existente si stergerea anunturilor care nu mai sunt valabile. De asemenea, am implementat o logica de filtrare care proceseaza cerintele utilizatorului si returneaza doar acele randuri din baza de date care respecta criteriile de pret sau locatie.

Perspective de viitor

Pe viitor intentionez sa extind functionalitatea aplicatiei prin adaugarea unui sistem de autentificare care sa protejeze baza de date de modificari neautorizate. De asemenea, imi propun sa creez o functie de export care sa genereze rapoarte automate in format text sau CSV cu cele mai ieftine proprietati din baza de date. O alta directie de dezvoltare ar fi implementarea unor statistici avansate, cum ar fi calcularea pretului mediu pe zona, direct prin comenzi SQL complexe. In final, mi-ar placea sa transform acest motor de cautare intr-un serviciu web, pastrand SQLite ca solutie de stocare dar adaugand o interfata grafica accesibila din browser.

BIBLIOGRAFIE

1.https://www.imobiliare.ro/vanzare-imobiliare/bucuresti?cq_src=google_ads&cq_cmp=22708597878&cq_term=&cq_plac=&cq_net=x&cq_plt=gp&utm_source=google&utm_medium=cpc&utm_campaign=%5BMAX%5D+%5BPerformance_Big-9%5D+%5BUsr_Aquisition%5D&gad_source=1&gad_campaignid=22718336872&gbraid=0AAAAAD_tjVy5cPlx-KvDZ9vwV3ffX64qL&gclid=CjwKCAiAjojLBhAlEiwAcjhrDk94JKWcgOONFINo4NY6GTCOjOllfDOLmmb-rfOVD0so2NBNhE8klRoC62wQAvD_BwE

2. https://stackoverflow.com/questions

3. https://sqlite.org/

4. https://www.java.com/en/

5. https://www.w3schools.com/java/

6. https://geniusee.com/single-blog/how-to-create-a-real-estate-app

7. https://www.imobiliare.net/