

# Vintage Rent

Documentatie  
(Programare Avansata pe Obiecte)

Dinu Florin-Silviu  
grupa 231

## Cuprins

<b>1</b>	<b>Introducere</b>	<b>1</b>
<b>2</b>	<b>Main</b>	<b>1</b>
<b>3</b>	<b>GUI</b>	<b>1</b>
3.1	MainGUI . . . . .	1
3.2	Add . . . . .	3
3.3	Rapoarte . . . . .	3
3.4	Log . . . . .	4
<b>4</b>	<b>Conexiunea la data sourceuri</b>	<b>4</b>
<b>5</b>	<b>Modele</b>	<b>4</b>
<b>6</b>	<b>Serviciul de audit</b>	<b>4</b>
<b>7</b>	<b>Rapoarte</b>	<b>4</b>
7.1	Raport client . . . . .	5
<b>8</b>	<b>Clase ajutatoare</b>	<b>5</b>
8.1	Pair<T, U> . . . . .	5
<b>9</b>	<b>Resurse</b>	<b>5</b>
<b>10</b>	<b>Alte mentiuni despre cod</b>	<b>5</b>
<b>11</b>	<b>Workflow GitHub</b>	<b>5</b>

# Introducere

Aplicatia permite gestionarea unui magazin care inchiriaza camere vintage. Avem utilizatori care pot fi angajati, clienti sau administratori de site. Acestora le sunt puse la dispozitie camere de diverse tipuri si obiective. Ei pot alege sa inchirieze oricare pereche de camera si obiectiv doresc. Daca apar probleme, angajatii pot adauga penalizari inchirierii.

Ca o prima idee despre ce si cum se poate face, atasez diagrama relationala a bazei de date. Proiectul a fost gandit pentru un anumit workflow care este implementat si in aplicatia Java.

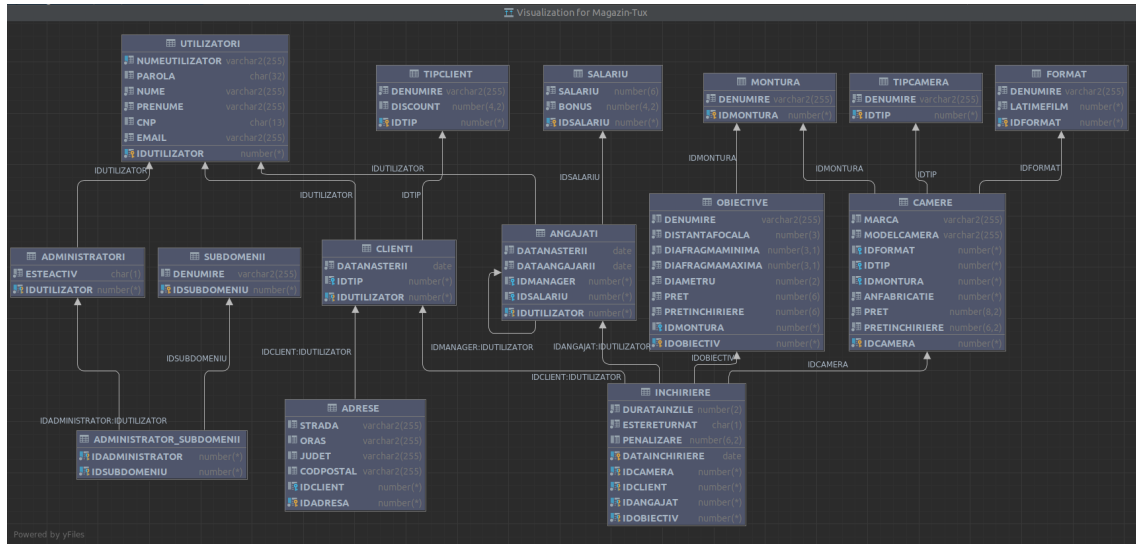


Figura 1: Diagrama relationala

## Main

Entry pointul in program se afla in **org.vintage.Main**, metoda **main**. Mai intai se initializeaza tema pentru GUI, apoi se porneste serviciul de audit **CsvLogger**. Apoi din main se lanseaza un thread separat si se asteapta sa se faca joinul. Acest lucru ne permite sa detectam cand programul este inchis si sa logam aceasta operatie.

Threadul deschis de main incepe prin a afisa splash screenul. In timpul afisarii se deschide un thread nou pentru conectarea la **baza de date Oracle** si se verifica fisierele **csv** pentru a avea putea fi procesate de al doilea tip de datasource. Atributul atomic boolean **isOracleUp** tine cont daca s-a putut realiza conexiunea la baza de date. Daca aceasta s-a realizat, atunci se va folosi default baza de date, daca nu, atunci se va folosi default csvul. Datasourceul poate fi schimbat in timpul rularii programului din meniul corespunzator, deci in aceasta etapa doar definim defaulturile.

Dupa ce threadul care face conexiunile se termina, se rezuma threadul **tmain** care va face dispose splash screenului dupa minim 3 secunde (in total cu timpul de conectare la data sourceuri). Apoi se incepe cu MainGUI care reprezinta interfata grafica principala.

## GUI

### MainGUI

File	CRUD	Datasources	Rapoarte	Log	About			
DURATA_IN_...	ESTE_RETUR...	PENALIZARE	DATA_INCHI...	IDCAMERA	IDCLIENT	IDOBIECTIV	IDANGAJAT	
1	true	0.0	2023-02-08	1	5	1	10	
1	true	0.0	2023-02-28	1	5	1	10	
5	true	0.0	2022-01-01	1	5	1	2	
2	true	10.0	2022-01-05	1	5	2	2	
3	true	11.0	2022-02-02	2	6	4	3	
10	true	20.0	2022-03-02	3	7	5	4	
11	true	0.0	2022-04-03	4	8	6	3	
3	true	0.0	2022-05-04	5	9	7	4	
5	true	30.0	2022-05-05	6	5	8	2	
4	true	0.0	2022-05-10	7	5	9	2	
4	true	0.0	2022-05-11	1	6	10	2	
5	true	0.0	2022-05-12	3	6	1	10	
6	true	50.0	2022-05-13	1	7	2	11	
9	false	0.0	2022-05-14	2	9	4	4	
1	true	0.0	2023-02-06	1	5	1	10	
1	true	0.0	2023-02-28	1	5	3	2	
1	true	0.0	2023-02-28	1	5	1	4	
Remove				Add				

Figura 2: Main GUI

Pentru tema am folosit **com.formdev.flatlaf** si am initializat **FlatDarkLaf**. Mai jos este codul dependentei din **pom.xml**.

---

```

<dependency>
  <groupId>com.formdev</groupId>
  <artifactId>flatlaf</artifactId>
  <version>3.0</version>
</dependency>

```

---

Interfata grafica principala consta dintr-o **bara de meniu** si un **gridbaglayout** in care este pus un **JTable** si doua butoane de **remove** si **add**.

**Bara de meniu** are mai multe elemente:

1. File - de aici se poate iesi din program
2. CRUD - contine tabelele pe care se poate lucra si se va face update la alegerea oricarui tabel, evident ca folosind datele din data sourceul selectat
3. Datasources - contine sursele de date disponibile si se va face update la alegerea oricarei surse de date
4. Rapoarte - contine rapoartele
5. Log - contine un item pentru afisarea logurilor
6. About - afiseaza informatii despre program

**JTable** este evident continut intr-un **JScrollPane** si are un data source dat prima data de **Rent** pentru **read**. Contine un eveniment de setare a valorii in caz de **update**. Unele tabele pot fi sortate, altele nu, depinde de la caz la caz. Oricum, chiar si in cele care pot fi sortate, operatiunile decurg normal, deoarece se obtine indexul corect al liniei si coloanei.

**Butonul remove** va sterge randul selectat atunci cand este apasat.

**Butonul add** va deschide o fereastră personalizata in functie de fiecare tabela in parte. In continuare voi vorbi despre ferestrele de adaugare.

## Add

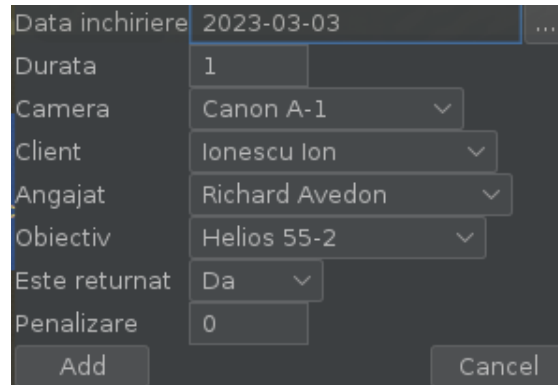


Figura 3: Add GUI

Ferestrele de adaugare sunt facute in functie de fiecare clasa in parte. Ca element comun toate contin un **gridbaglayout**, un buton de adaugare si unul de iesire, precum si mai multe **JLabel**-uri.

Elementele optionale pe care le pot contine sunt:

1. **TextField** pentru text normal
2. **ComboBox** in cazul unei chei externe pentru alegerea in functie de informatiile din tabelul de legatura
3. **DatePickerImpl** pentru alegerea datelor calendaristice

Daca primul este evident, pentru celelalte doua voi detalia.

**ComboBox** reprezinta un combo box pentru a alege datele in functie de ce se afla in tabelul de legatura. Ca element, am creat **org.gui.custom.ComboBox** pentru a putea opera cu o cheie si o valoare. Cheia reprezinta valoarea campului de legatura, iar valoarea ceea ce se va afisa.

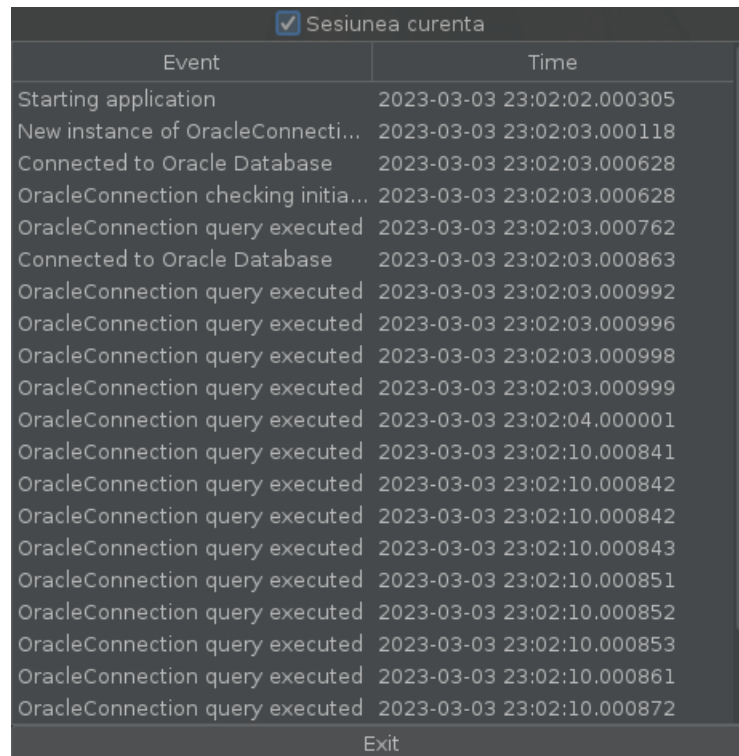
**DatePickerImpl** reprezinta un date picker de la **org.jdatepicker.jdatepicker**. Acesta poate vine cu un panou care se afiseaza dinamic pentru alegerea datei calendaristice. Codul pentru dependenta adaugat in **pom.xml** este:

```
<dependency>
  <groupId>org.jdatepicker</groupId>
  <artifactId>jdatepicker</artifactId>
  <version>1.3.4</version>
</dependency>
```

## Rapoarte

**Raport clienti** reprezinta raportul in care se pot afisa date despre ce a inchiriat un client. Acesta contine un **ComboBox** pentru a alege clientul cu un eveniment de schimbare astfel incat sa se actualizeze automat datele din **JTable**-ul de mai jos. De asemenea, contine si un buton de exit.

## Log



<input checked="" type="checkbox"/> Sesiunea curenta	
Event	Time
Starting application	2023-03-03 23:02:02.000305
New instance of OracleConnecti...	2023-03-03 23:02:03.000118
Connected to Oracle Database	2023-03-03 23:02:03.000628
OracleConnection checking initia...	2023-03-03 23:02:03.000628
OracleConnection query executed	2023-03-03 23:02:03.000762
Connected to Oracle Database	2023-03-03 23:02:03.000863
OracleConnection query executed	2023-03-03 23:02:03.000992
OracleConnection query executed	2023-03-03 23:02:03.000996
OracleConnection query executed	2023-03-03 23:02:03.000998
OracleConnection query executed	2023-03-03 23:02:03.000999
OracleConnection query executed	2023-03-03 23:02:04.000001
OracleConnection query executed	2023-03-03 23:02:10.000841
OracleConnection query executed	2023-03-03 23:02:10.000842
OracleConnection query executed	2023-03-03 23:02:10.000842
OracleConnection query executed	2023-03-03 23:02:10.000843
OracleConnection query executed	2023-03-03 23:02:10.000851
OracleConnection query executed	2023-03-03 23:02:10.000852
OracleConnection query executed	2023-03-03 23:02:10.000853
OracleConnection query executed	2023-03-03 23:02:10.000861
OracleConnection query executed	2023-03-03 23:02:10.000872

Exit

Figura 4: Log GUI

Contine logurile sesiunii curente, dar si un **JCheckBox** astfel incat sa poata fi vazute toate. Acestea sunt scrise intr-un fisier CSV si preluate de acolo intr-un **JTable** continut intr-un **JScrollPane**.

## Conexiunea la data sourceuri

### Modele

### Serviciul de audit

**CsvLogger** este numele serviciului de audit care scrie logurile intr-un fisier csv. Acesta extinde **java.util.logging.Logger** si este de tip singleton. Are un constructor privat si singurul mod in care poate fi creata instanta este prin metoda statica **getInstance**.

**Metoda de scriere** `log` primeste un mesaj de tip `String` si il scrie in fisier.

**Metodele de citire** `readLog()` si `readLogToday()` citesc intreg fisierul de log, respectiv doar cel de la inceputul sesiunii si pana acum si corespund optiunilor din interfata grafica. Ambele returneaza un **Pair<List<String>, List<List<String>>** cu headerul, respectiv corpul fisierului.

## Rapoarte

Rapoartele si alte actiuni pot fi accesate direct din **org.actions.MainService**.

## Raport client

Acesta primește un întreg care reprezintă ID-ul clientului și un tip de baza de date. Returnează un **Map<String, String>** reprezentând maparea tuturor datelor obținute. Ca mențiune separată, pentru a număra camerele distincte am optat pentru o integrare comună între CSV și bazele de date, așa că am efectuat o singură interogare și le pun într-un **HashSet<Integer>**, iar apoi extrag dimensiunea lui.

## Clase ajutatoare

### Pair<T, U>

Deoarece în Java nu există un tip pair nativ, am scris o clasă care folosește două templateuri, T și U, și creează o pereche mapând obiectele pe atributele first și second. Acestea sunt publice, dar se pot accesa și prin getteri.

## Resurse

## Alte mențiuni despre cod

## Workflow GitHub

Acțiunea de publicare a unei versiuni noi pe GitHub are următorii pași:

1. Checkout
2. Setare Java 17 Oracle
3. Preluarea versiunii proiectului cu ajutorul Maven din pom.xml
4. Buildul făcut de Maven
5. Uploadul artefactului de release (fișierul .jar) în mediul acțiunii
6. Crearea pachetului de release
7. Atasarea artefactului de release la pachet

Toate aceste lucruri se pot face atât manual cât și la fiecare push pe main. De aceea pentru fiecare versiune nouă se va lucra pe un branch separat.

Fișierul de workflow se află în `./.github/workflows/main.yml`.