

Vintage Rent

Documentatie
(Programare Avansata pe Obiecte)

Dinu Florin-Silviu
grupa 231

Cuprins

1	Introducere	1
2	Main	1
3	GUI	2
3.1	MainGUI	2
3.2	Add	3
3.3	Rapoarte	4
3.4	Log	4
4	Conexiunea la data sourceuri	4
5	Modele	6
5.1	Model	6
5.2	ModelList	6
5.3	LinkModelToDatabase	6
6	*Model	6
7	Serviciul de audit	7
8	Rapoarte	7
8.1	Raport client	8
9	Clase ajutatoare	8
9.1	Pair<T, U>	8
9.2	ModelInit	8
10	Resurse	8
11	Workflow GitHub	8
12	Alte mentiuni despre cod	9

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.

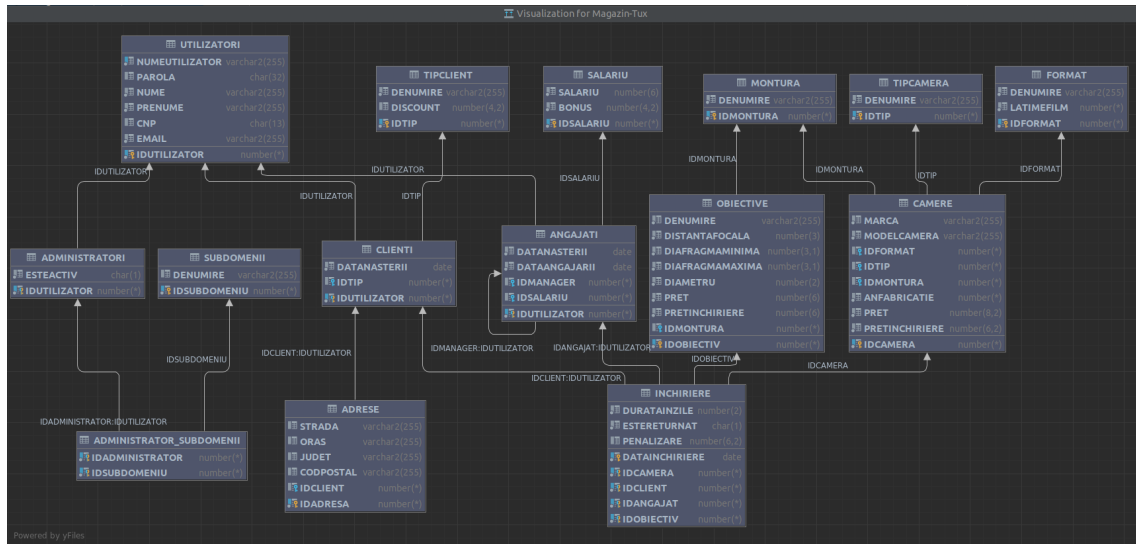


Figura 1: Diagrama relationala

Pentru conexiunea la baza de date Oracle, va trebui sa intrati in folderul **docker_oracle_21c** si sa rulati **docker-compose up**. Dupa acest lucru, adaugati in baza de date urmatorul utilizator:

```
CREATE USER C##TUX IDENTIFIED BY "fmiLove";  
GRANT ALL PRIVILEGES TO C##TUX;
```

Motivul pentru care acest utilizator exista, este ca sa nu se faca niciodata conexiunea cu SYSTEM sau alt utilizator privilegiat, ca si in viata reala.

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 personalizată în funcție de fiecare tabelă în parte. În continuare voi vorbi despre ferestrele de adăugare.

Add

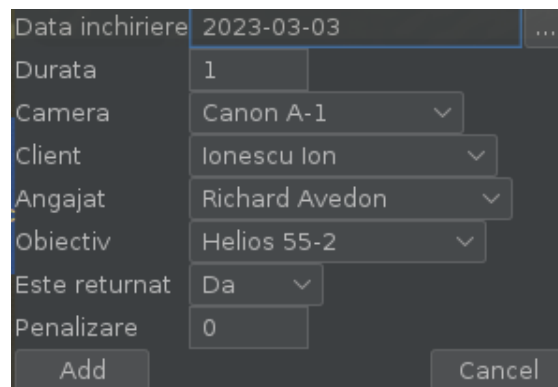


Figura 3: Add GUI

Ferestrele de adăugare sunt făcute în funcție de fiecare clasă în parte. Ca element comun toate contin un **gridbaglayout**, un buton de adăugare și unul de ieșire, precum și mai multe **JLabel**-uri.

Elementele optionale pe care le pot conține sunt:

1. **TextField** pentru text normal
2. **ComboBox** în cazul unei chei externe pentru alegerea în funcție de informațiile din tabelul de legatură
3. **DatePickerImpl** pentru alegerea datelor calendaristice

Dacă primul este evident, pentru celelalte două voi detalia.

ComboBox reprezintă un combo box pentru a alege datele în funcție de ce se află în tabelul de legatură. Ca element, am creat **org.gui.custom.ComboBoxItem** pentru a putea opera cu o cheie și o valoare. Cheia reprezintă valoarea câmpului de legatură, iar valoarea ceea ce se va afișa.

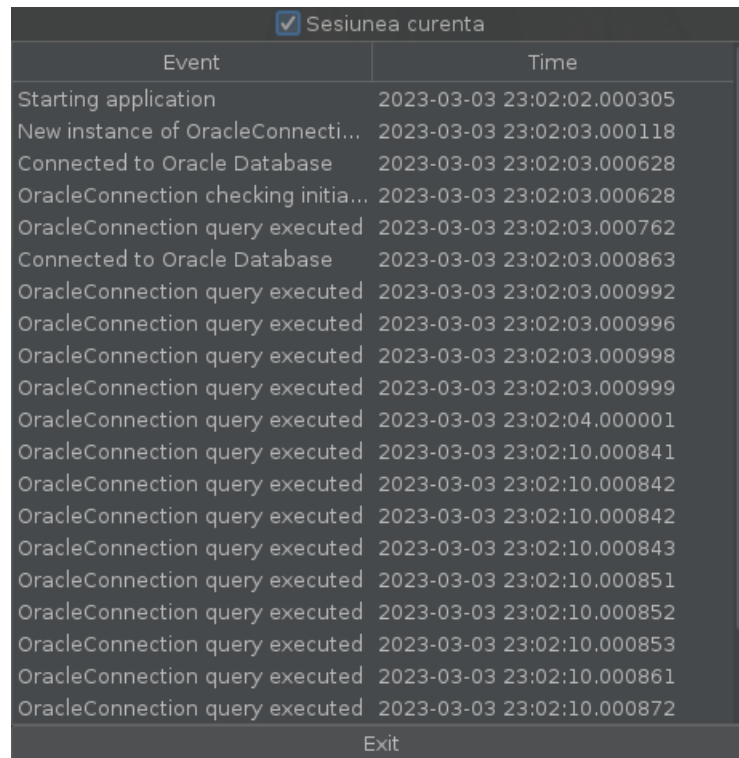
DatePickerImpl reprezintă un date picker de la **org.jdatepicker.jdatepicker**. Acesta poate vine cu un panou care se afișează dinamic pentru alegerea datei calendaristice. Codul pentru dependența adăugat în **pom.xml** este:

```
<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 **JComboBox** 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

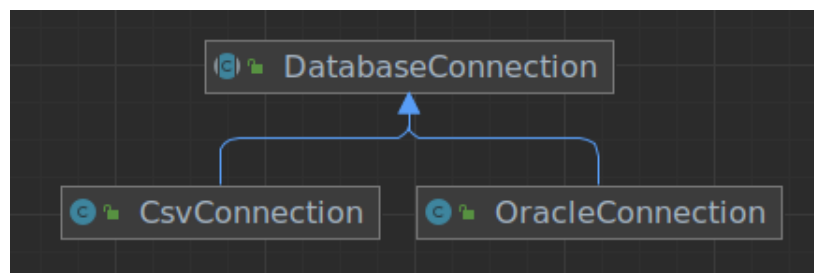


Figura 5: Diagrama claselor care se ocupa de relatia cu data sourceurile

Clasa abstracta `org.database.DatabaseConnection` contine attributele si metodele necesare de implementat de catre **OracleConnection** si **CsvConnection** care o mostenesc.

Deoarece clasa nu poate fi initializata, ea functioneaza si pe post de **factory** pentru celelalte clase. Astfel ca e nevoie de un **public enum DatabaseType** pentru a da tipul de data source care se doreste in metoda **getInstance**. De asemenea, pentru ca celelalte 2 clase sunt de tip **singleton**, aceasta are un pool de conexiuni pe care il completeaza pe masura ce se creaza instante din clasele noi. Pool-ul este in **protected static ArrayList<DatabaseConnection> instances**.

OracleConnection este clasa care mosteneste DatabaseConnection si implementeaza metodele acesteia de conectare la baza de date si manipulare a datelor. Evident ca toate sunt in functie de cerintele necesare sistemului de gestiune a bazelor de date Oracle careia ii trimite queryuri. Aceste queryuri nu sunt protejate la un atac de tip sql injection, ramane in lista de todo de implementat o astfel de protectie.

CsvConnection este clasa care, de asemenea, mosteneste DatabaseConnection si implementeaza metodele de manipulare a fisierelor Csv. Aici sunt cateva particularitati.

In primul rand, fisierele Csv nu au integritate referentiala, iar programul nu este un sistem de gestiune a lor asemanator cu unul al unei baze de date. Asadar, utilizatorul trebuie sa fie atent la modificarea fisierelor astfel incat sa nu strice integritatea lor.

Pentru manipularea datelor am folosit **com.opencsv.opencsv**. Codul din pom.xml pentru aceasta dependenta este urmatorul:

```
<dependency>
  <groupId>com.opencsv</groupId>
  <artifactId>opencsv</artifactId>
  <version>5.7.1</version>
</dependency>
```

Deoarece apar diferente fata de bazele de date care au deja rezultate implementate ca **ResultSet**, am folosit **com.mockrunner.mockrunner-jdbc** pentru a simula acest tip de date astfel incat metodele sa aiba o valoare returnata standard. Codul din pom.xml pentru acesta este urmatorul:

```
<dependency>
  <groupId>com.mockrunner</groupId>
  <artifactId>mockrunner-jdbc</artifactId>
  <version>2.0.1</version>
</dependency>
```

Deoarece clasa este folosita si de serviciul de audit, aceasta impelmenteaza cateva metode care nu se regasesc in celelalte conexiuni. Aceste metode sunt **insertNoLog** care face inserarea fara a scrie acest lucru in loguri (altfel s-ar crea o bucla infinita) si **readAllNoLog** care citeste un fisier de log, la fel fara a scrie acest lucru in loguri. In serviciul de audit se face un upcasting catre aceasta clasa la instanta de DatabaseConnection pentru a-i folosi metodele specifice.

Pentru ca resursele pot sa nu fie alocate deja, am facut un hack si am inclus un fisier **CSV/mock.csv** careia sa-i ia calea in caz ca doreste sa creeze un fisier care inca nu exista al unei tabele. Acest comportament poate fi observat in metoda **createAndINsertDynamically**.

Modele

Model

Clasa abstracta **Model** contine ceea ce este necesar pentru a crea un model concret:

1. Atributul de instanta pentru a fi singleton
2. Atributul de tip de baza de date pentru conectare
3. Atributul de nume cu getter si setter
4. Metoda statica de preluare a instantei pentru a asigura proprietatea de singleton
5. Metoda de setare a unui tip de baze de date
6. O clasa interioara statica si abstracta denumita **AbstractInnerModel** unde vor fi puse attributele necesare fiecarui model atunci cand le preia din data source

ModelList

Clasa **ModelList** `<T extends Model.AbstractInnerModel>` contine un template pentru a limita tipul de date al listelor pe care le creaza la cel specificat, precum si un atribut privat de tip **List**`<T>` unde va stoca lista.

Are 3 constructori, unul parametrizat pentru initializarea listei cu o lista data, altul neparametrizat care initializeaza lista implicit cu un `ArrayList`, iar altul parametrizat cu un boolean pentru a initializa cu un `SortedList`.

Contine o clasa denumita **SortedList**`<T>` care mosteneste `ArrayList` si face override pe metodele de adaugare, actualizare si stergere specifice astfel incat sa poata pastra lista sortata.

Contine mai multe metode de a manipula lista continuta.

LinkModelToDatabase

Contine mai multe metode necesare unui model pentru a lucra cu baza de date (CRUD), precum si o metoda specifica aplicatiei pentru a scrie datele din baza de date intr-un fisier CSV.

In semnatura sa, **public interface LinkModelToDatabase**`<T extends ModelList, U extends Model.AbstractInnerModel>` apar 2 templateuri, unul pentru a fi sigur ca este de tip `ModelList`, iar altul pentru a fi sigur ca extinde clasa de campuri continuta de fiecare model in parte.

*Model

Prin acest lucru definesc orice model. Deoarece modelele functioneaza pe acelasi principiu, dar cu o alta multime de campuri mapate din data source, este de preferat sa fac o descriere generala pentru a putea implementa orice alt model pe viitor.

Clasele modelelor concrete, extind clasa `Model` si implementeaza interfata `LinkModelToDatabase` oferindu-i valori corecte pentru template. Un exemplu de astfel de semnatura este: **public class RentModel extends Model implements LinkModelToDatabase** `< ModelList < RentModel.InnerRentModel>, RentModel.InnerRentModel>`.

Contin o clasa interioara statica ce extinde `AbstractInnerModel` in care se definesc campurile care se vor utiliza.

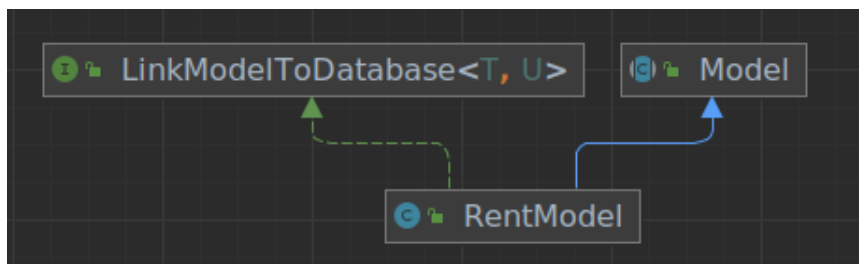


Figura 6: Diagrama RentModel

Mentionez in continuare modalitatea standard de implementare pentru cele mai importante metode:

1. `getTableModel` definește un array de string-uri cu coloanele si apoi creaza si returneaza un `DefaultTableModel`
2. `transferToModelList` transfera un `ResultSet` in `ModelList`-ul corespunzator
3. `getData()` incepe prin a defini un map cu denumirea generala a tabelelor si locul unde se pot gasi in baza de date. Apoi preia `ResultSet`-uri din mai multe tabele, iar in afara de cel principal, pe celelalte le pune intr-un `Map<Integer, String>` pentru a avea cheia primara si valoarea care se doreste extrasa. Apoi, pentru fiecare inregistrara din tabela principala, va mapa datele obtinute din celelalte tabele si va popula `ModelList`-ul. Am ales aceasta metoda deoarece este comuna atat bazelor de date relationale, cat si fisierelor csv.
4. `updateData` primeste un rand de actualizat sub forma unui `ModelList` si mapeaza pe baza lui toate campurile ce pot fi actualizate, apoi intr-o mapare separata va pune campurile care definesc cheia primara. La sfarsit ia obiectul data sourceului si apeleaza functia de update
5. `deleteRow` preia un rand sub forma unui `ModelList` si creaza clauza where specifica stergerii pe care o trimite mai departe obiectului data sourceului si apeleaza functia delete.
6. `throwIntoCsv` verifica daca are o conexiune la o baza de date, apoi ia pe rand toate rezultatele, inclusiv din metadatele `ResultSet`-ului coloanele, apoi apeleaza metoda de creare si inserare din `CsvConnection`
7. `insertRow` preia un model ce extinde `AbstractInnerModel` si creaza o lista de valori de forma **`List<Pair<String, String>>`** in care adauga pe rand valorile. Daca in spate modelul are o cheia primara care se incrementeaza, acesta exclude valoarea noua a cheii primare.

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 bază 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 ca 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.

ModelInit

Această clasă se ocupă cu initializarea modelului pentru fișierele CSV corespunzătoare tabelor. Dacă acestea nu sunt găsite, programul va lua pe rând tabelele din baza de date și le va arunca datele în fișiere CSV corespunzătoare pentru ca modelele să funcționeze și cu acest tip de fișiere.

Resurse

Folderul de resurse este împărțit în:

1. CSV cu fișierele CSV
 - (a) Fișierele corespunzătoare tabelor
 - (b) Folderul Log
2. META-INF cu manifestul
3. Migrations cu fișierul sql care este rulat pentru a popula baza de date
4. Spalsh cu imaginea pentru spalsh screen

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`.

Alte mentiuni despre cod

Codul se refera la un produs MVP de gestionare a unui magazin de vanzare a camerelor foto vintage. Pe parcurs am spus ce poate fi imbunatatit, iar in fisierul <https://github.com/fredtux/VintageRent/blob/main/README.MD> se pot gasi mai multe informatii referitoare la indeplinirea cerintelor.