Aplicația de gestiune a unei biblioteci: jLib

Referat

Alin Clincea, 221

1. Cerințele problemei

1.1 Cerințe comune

Proiectați și implementați o aplicație care folosește interfețe grafice, folosind Java Swing sau JavaFX, pentru rezolvarea problemei propuse (Enunțurile de mai jos!). Se va evidenția o arhitectură de aplicație pe 3 niveluri:

1. *interfață (nivel prezentare) – interfața grafică;*
2. *logica aplicației (nivel logic) – domeniul problemei;*
3. *persistența datelor (nivel date) – repository;*

Interfața trebuie să cuprindă elementele fundamentale ale graficii în Java: gestionari de poziționare, tratarea evenimentelor, interfețe de tip Listener și clase interne pentru gestionarea evenimentelor. Totodată, trebuie ca interfețele să cuprindă meniuri, meniuri de context, acceleratori, etc.

Accesul în cadrul aplicației se va face pe bază de: user și password.

Este importantă separarea la nivel de clase a codului ce ține de GUI de cel ce ține de logica aplicației (acțiunile propriu-zise).

Se va ține cont de modelarea corectă cu clase și pachete a sistemului.

1.2 Problema aleasă

5. Biblioteca

Interfața trebuie să includă următoarele facilități:

- vizualizarea informațiilor despre o carte selectată: un mic rezumat, coperta cărții, preț etc.

- căutarea cărților după diverse criterii: autor, titlu, an de publicație, editură etc.

- împrumutarea cărților și gestiunea împrumuturilor.

2. Structura, fluxul și implementarea aplicației

2.1 Funcționalitate

Aplicația (denumită *jLib*) permite gestionarea a trei entități principale de conținut: **cărțile** (obiectele stocate într-o bibliotecă), **cititorii** (subiecții care consumă serviciile bibliotecii) și **bibliotecarii** (subiecții care gestionează serviciile bibliotecii).

2.2 Arhitectură

jLib este concepută după patternul *model-view-controller* (MVC[1]), în care datele sunt gestionate cu ajutorul unei baze de date (de tip SQLite).

Utilizatorul interacționează cu interfața grafică (*views*), comenzile sunt preluate de unități de cod (*controllers*), care la rândul lor interacționează cu reprezentarea datelor (*models*)

[1] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

extrase din stocare (baza de date). Sunt implementate operațiile CRUD[2] pentru toate entitățile de conținut.

2.2.1 Entry point

Inițializarea aplicației se realizează prin intermediul *App.java* – o clasă derivată din *Application* (clasă parte din platforma JavaFX).

2.2.2 Views

jLib are trei elemente principale de interfață grafică: **ecran de login** (*/views/login.fxml*), **ecranul principal** (*/views/main.fxml*), **dialog pentru funcția de împrumutare a cărților** (*/views/borrow_dialog.fxml*).

2.2.3 Controllers

Fiecărui *view* îi corespunde un *controller*, care conține și gestionează datele și acțiunile generate de interacțiunile utilizatorului cu *view*-ul:

/views/login.fxml	->	*/controllers/LoginController.java*
/views/main.fxml	->	*/controllers/MainController.java*
/views/borrow_dialog.fxml	->	*/controllers/BorrowController.java*

2.2.4 Resources

Elementele grafice afișate în *view*-uri (imagini, stiluri CSS) sunt plasate în folderul *resources* (elementele permanente). Elementele din proiect, utilizate pentru demonstrație, sunt plasate în subfolderele din *storage* (grupate pe categorii: *covers*, *readers*, *librarians*).

2.2.5 Models

Fiecărei entități principale de conținut îi corespunde un *model* clase care reprezintă atributele și metodele fiecărei entități, în corelație cu modalitatea de stocare a acestora în baza de date:

/models/Book.java	->	cărți
/models/Reader.java	->	cititori
/models/Librarian.java	->	bibliotecari

2.2.6 Storage

Baza de date (*/storage/jlib.db*) cuprinde patru tabele:

1. **books**: atributele cărților;
2. **borrows**: relația dintre cărți și cititori; atributele împrumuturilor;
3. **readers**: atributele cititorilor;
4. **librarians**: atributele bibliotecarilor.

Modulul necesar pentru utilizarea din Java a bazei de date SQLite este driverul JDBC *sqlite-jdbc-3.34.0.jar*, plasat în folderul *lib* și încărcat ca *referenced library*.

[2] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete

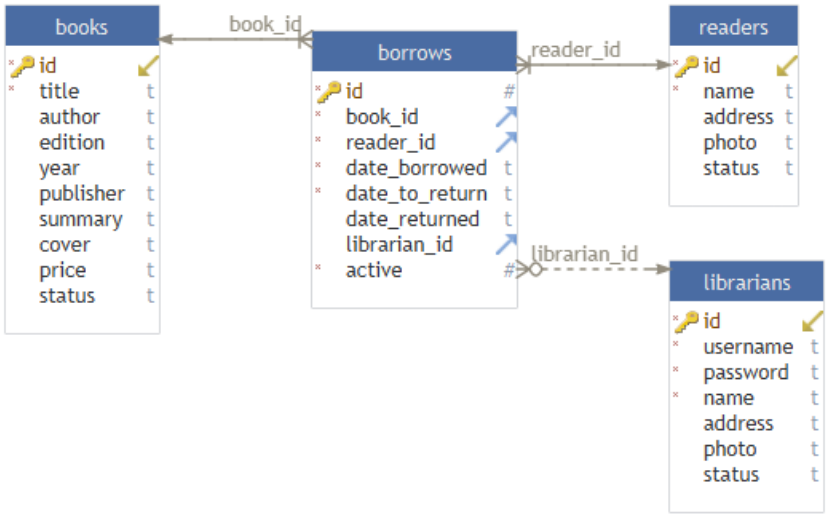

Figura 1: Schema bazei de date

2.2.7 Utils

jLib mai conține și două clase utilitare (*utils*):

1. *Database.java*: pentru gestiunea interacțiunilor cu baza de date (conexiune, interogări);
2. *AutoComplete.java*: pentru extinderea funcționalității unui *ComboBox*;

2.3 Flux

Fluxul aplicației este reprezentat schematic în Figura 2.

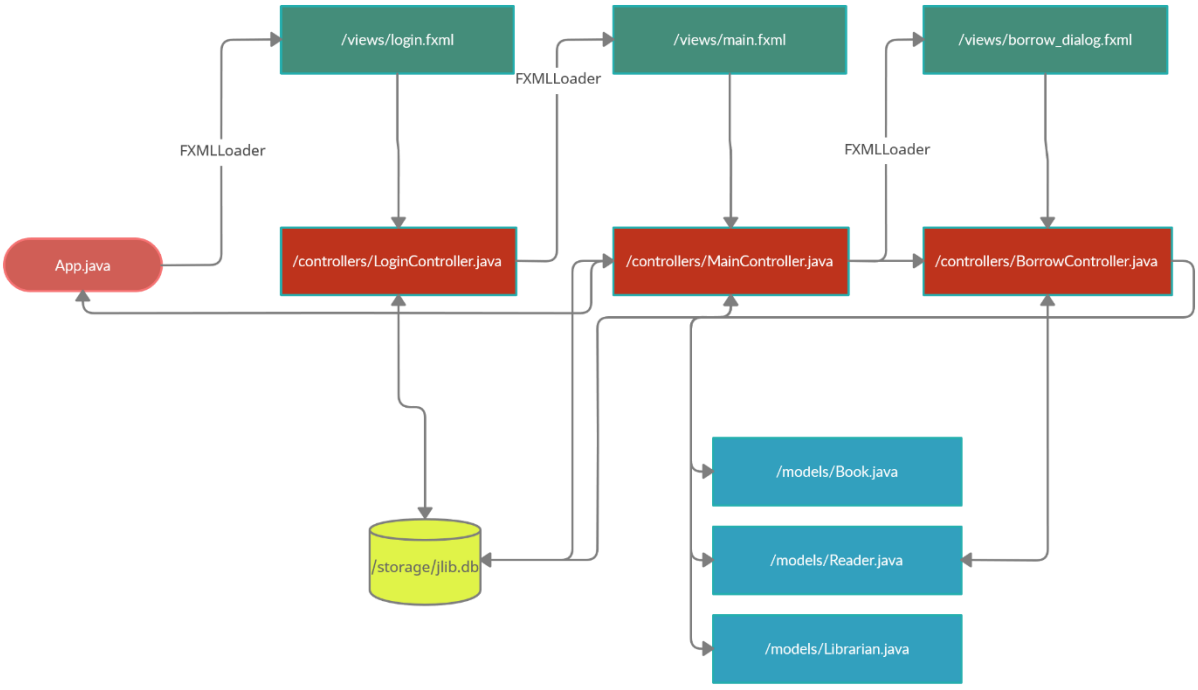

Figura 2: Fluxul aplicației

Inițializarea aplicației are loc în *App.java*. Într-un câmp static al clasei se va păstra o referință către instanța clasei, pentru acces din controllere (ex. în metoda *logout()* din *MainController.java*). Inițializarea referinței are loc prin constructor.

Prin intermediul *FXMLLoader* (clasă a JavaFX) se încarcă primul *view* (*login.fxml*) și *controllerul* său (*LoginController.java*). Această fereastră este fără cadru (*borderless*), astfel încât este necesară implementarea funcționalității de mutare a ferestrei și de închidere a acesteia.

În **ecranul de login** utilizatorul trebuie să introducă *username* și *password* corecte (valorile inițiale sunt *admin* și *pass*); verificarea corectitudinii se face prin interogarea bazei de date. În cazul în care datele nu sunt corecte, utilizatorul este informat prin UI.

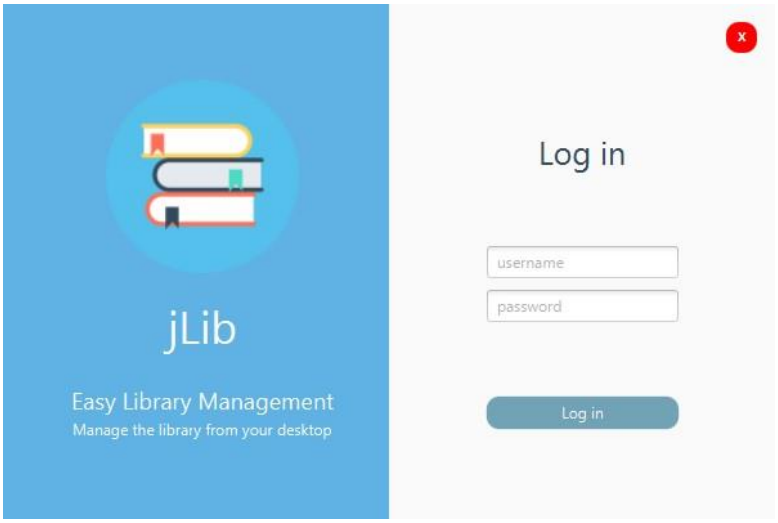

Figura 3: Ecranul de login

Dacă autentificarea s-a realizat cu succes, se încarcă **ecranul principal** al aplicației. Acesta este compus din: meniu, bandă superioară de informare, container cu 3 taburi, bandă inferioară (fără rol funcțional în versiunea curentă a aplicației, dar cu potențial de extindere – ex. consolă de activități etc.).

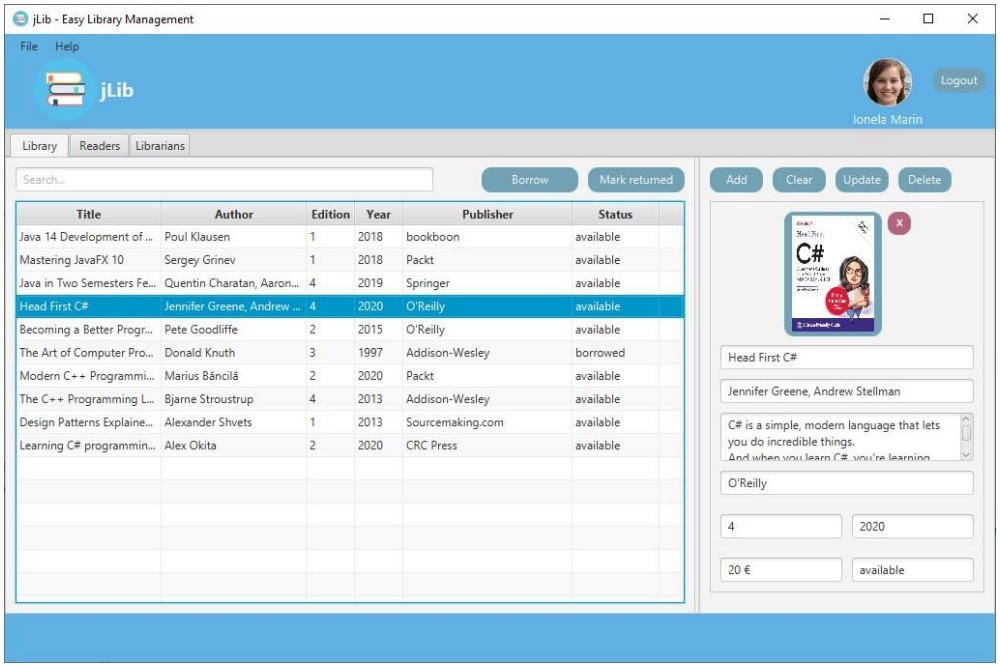

Figura 4: Tabul Library, cu o carte selectată

Containerele din acest ecran sunt scalabile potrivit dimensiunii ferestrei.

În conținutul fiecărui tab este folosită o interfață după patternul *master-detail*, în care datele sunt afișate în format tabular (*TableView*), alături de o zonă în care sunt afișate detaliile rândului selectat din tabel. Acțiunile de gestionare a datelor se declanșează prin butoanele din zona superioară a elementului *detail*.

Taburile conțin și câmpuri de căutare, care realizează o filtrare instantanee a conținutului tabelelor, fără a mai fi necesar mecanismul „clasic" de selecție prin *ComboBox* a criteriului de căutare, urmat de apăsarea unui buton.

Particularitatea din tabul *Library* este aceea că permite, prin butoanele *Borrow* și *Mark returned*, gestionarea împrumuturilor (în ambele cazuri este necesară o selecție în tabel).

Butonul *Borrow* declanșează afișarea view-ului *borrow_dialog.fxml*, conceput ca o **fereastră de dialog modală**, care are ca scop realizarea unei asocieri între cartea selectată (identificată prin *id* din baza de date) și cititorul care poate fi selectat cu ajutorul unui *ComboBox* care realizează o filtrare instantanee a cititorilor disponibili.

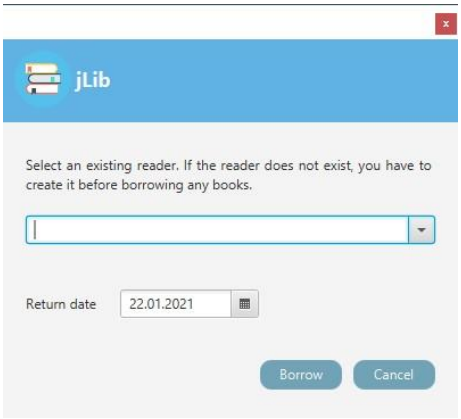

Figura 5: Dialogul modal de împrumut al cărților

2.4 Implementare

Toate denumirile și conținutul textual din aplicație (ex. comentarii, javadoc), inclusiv cele din interfața grafică, sunt în engleză (există posibilitatea implementării unui mecanism de traducere a interfeței grafice, în ipotetice versiuni viitoare).

Codul a fost optimizat folosind recomandările de *linting* din IntelliJ IDEA (ex. preferința pentru expresii lambda, acolo unde este posibil; reducerea unor expresii condiționale; reducerea unor tipuri explicite – *A<Reader, String>()*, cu *A<>()*).

Este păstrată o referință la instanța curentă a aplicației, vizibilă pentru controllere sub forma unui câmp static al clasei *App*, pentru facilitarea funcției de *logout* (anterior implementării finale, codul era duplicat în *App.java* și în *MainController.java*).

Conexiunea la baza de date se realizează o singură dată, are referințe în toate controllerele și este menținută pe toată durata execuției aplicației prin intermediul clasei utilitare */utils/Database.java*.

Widgeturile au un *fx:id*, care este adăugat în controllerele corespunzătoare folosind adnotarea *@FXML*.

Acțiunile determinate de butoane sunt realizate ca metode, de regulă fără parametri, care sunt atașate butoanelor prin FXML (ex. *onAction="#closeWindow"*, în manieră similară combinației HTML și JavaScript).

Acțiunile conținute în metodele apelabile din butoane sunt refolosite în cazul meniurilor contextuale (implementate, în această versiune a aplicației, doar pentru tabelul de cărți).

Există widgeturi pentru care este necesară validarea inputului, astfel încât să fie prevenite manifestări neplanificate ale aplicației (ex. *TextFields* sau *DataPicker* care nu ar trebui să fie goale). De asemenea, pentru anumite acțiuni mai importante (ex. ștergerea de obiecte) ar putea fi implementată confirmarea din partea utilizatorului.

Transmiterea de date între controllere (ex. *id*-ul bibliotecarului sau al cărții selectate) se realizează prin:

1. expunerea unui câmp în controllerul-destinație;
2. înainte de crearea scenei pentru destinație, crearea unei referințe către destinație în controllerul-sursă;
3. accesarea și editarea câmpului prin referința la destinație;
4. încărcarea și afișarea scenei.

Spre exemplificare: instanța *MainController* transmite către *BorrowController* valorile pentru: *id*-ul cărții selectate, *id*-ul bibliotecarului care efectuează împrumutul, referința către instanța *MainController* (pentru actualizarea statusului din tabelul de cărți – *mainController.displayBooks()*):

```
FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("/jlib/views/borrow_dialog.fxml"));
Parent window = fxmlLoader.load();

BorrowController borrowController = fxmlLoader.getController();
borrowController.selectedBookId = id;
borrowController.currentLibrarianId = currentLibrarianId;
borrowController.mainController = this;

Stage stage = new Stage();
stage.initStyle(StageStyle.UTILITY);
stage.initModality(Modality.APPLICATION_MODAL);
stage.setResizable(false);
stage.setScene(new Scene(window));
stage.show();
```

Pentru execuția de cod în controllere, după etapa de încărcare a view-urilor FXML, este implementată interfața *javafx.fxml.Initializable*, care impune folosirea metodei *initialize()*. Dacă s-ar încerca execuția codului ce accesează obiecte FXML injectate, prin utilizarea constructorilor controllerelor, se pune problema unor referințe nule întrucât la momentul execuției constructorilor etapa de injecție FXML nu este încă parcursă.

Datele obținute din *jlib.db* sunt adăugate în tipuri *ObservableList<T>*, care ulterior sunt utilizate pentru popularea tabelelor, capturarea selecțiilor etc.

Funcționalitatea de autocompletare pentru *ComboBox*-ul din dialogul de împrumutare este implementată folosind o clasă utilitară (*/utils/AutoComplete.java*) importată dintr-o sursă externă.

Ori de câte ori este necesară interacțiunea cu baza de date sunt urmate regulile:

1. dacă interogarea presupune utilizarea inputului direct al utilizatorului (ex. textul unui *TextField*), sunt folosite *prepared statements* (pentru a evita sau limita posibilitatea de injecții SQL);
2. atunci când interacțiunea cu baza de date este mediată (ex. se folosește un *id* sau un câmp la care utilizatorul nu are acces direct) sunt folosite interogări SQL simple, direct din controllere.

Imaginile pe care utilizatorii le pot modifica (ex. coperți, fotografii de cititori sau de bibliotecari) nu sunt stocate în baza de date (deși ar fi fost posibil prin utilizarea tipului *blob*). Numai URL-ul acestora este stocat (în formă externă, ex. *file:/D:/Documents/Facultate/Teme/Tehnologii%20Java/Tema/jLib/src/jlib/storage/covers/Head%20First%20C%23%20%5BStellman,%204,%202020%5D.jpg*), ele fiind păstrate în *file system*.

Baza de date poate fi îmbunătățită printr-o concepție coerentă asupra modului de gestionare a datelor (ex. dacă se urmărește păstrarea istoricului obiectelor, atunci interogările de tip DELETE trebuie să fie în concret implementate ca UPDATE, cu marcarea respectivului obiect ca indisponibil pentru interfața grafică curentă, dar vizibil în eventuale interfețe de istoric, rapoarte etc.). De asemenea, datele autorilor pot fi stocate într-un tabel separat, care să fie asociat printr-un tabel de relație cu cărțile (permițând asocieri de mai mulți autori la o carte, unicitatea informației despre autori și refolosirea acesteia); totuși, aceste dezvoltări nu fac obiectul temei propuse.

3. Mediul de dezvoltare

Aplicația a fost realizată folosind următoarea platformă software:

1. **sistem de operare**: Windows 10 (x64);
2. **JDK**: AdoptOpenJDK jdk-15.0.1+9 x64 Windows[3];
3. **JavaFX**: JavaFX 11.0.2 Windows[4]; SceneBuilder 11.0.0 Windows[5];
4. **IDE**: Visual Studio Code 1.52.1 x64 Windows[6]; Java Extension Pack 0.12.1;
5. **database management**: SQLite JDBC Driver 3.34.0[7]; DB Browser for SQLite 3.12.1 x64 Windows[8];
6. **software configuration management**: Git for Windows x64 2.29.0[9]; TortoiseGit 2.11.0 x64 Windows[10,11].

[3] https://adoptopenjdk.net/releases.html?variant=openjdk15&jvmVariant=hotspot
[4] https://gluonhq.com/products/javafx/
[5] https://gluonhq.com/products/scene-builder/
[6] https://code.visualstudio.com/Download
[7] https://github.com/xerial/sqlite-jdbc, https://repo1.maven.org/maven2/org/xerial/sqlite-jdbc/3.34.0/sqlite-jdbc-3.34.0.jar
[8] https://sqlitebrowser.org/dl/
[9] https://gitforwindows.org/
[10] https://tortoisegit.org/
[11] https://github.com/ucv-cs/Tehnologii-Java/tree/master/Tema/jLib (*privat*)