МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИРОДНИЧО-ГУМАНІТАРНИЙ ФАХОВИЙ КОЛЕДЖ

ДВНЗ «Ужгородський національний університет»

Реєстраційний №

Дата

Дацко Павло Володимирович

студент 4-го курсу

денної форми навчання

ІНП № 253029ФМБД

**З В І Т**

ПРОХОДЖЕННЯ ВИРОБНИЧО-ТЕХНОЛОГІЧНОЇ ПРАКТИКИ

Галузь знань 12 «Інформаційні технології»,

Спеціальність 121

«Інженерія програмного забезпечення»

Розробка системи каталогізації книг

(індивідуальне завдання)

|  |  |
| --- | --- |
| Допущено до захисту | Науковий керівник: |
| Дата «  »   2025 р.    (підпис) | Викл. Кузьма Л.В. |

**Практика захищена «**  **»**    **2025 р.**

**з оцінкою «** **»**

**Підписи членів комісії**

Ужгород – 2025

**ЗМІСТ**

ВСТУП 4

[1 ТЕОРЕТИЧНА ЧАСТИНА 6](#_Toc195835978)

[1.1 Загальні відомості про засоби розробки 6](#_Toc195835979)

[1.2 Загальні відомості про Java 6](#_Toc195835980)

[1.3 Загальні відомості про IntelijIdea 8](#_Toc195835981)

[1.4 Загальні відомості про SQLite 9](#_Toc195835982)

[1.5 Огляд аналогів 10](#_Toc195835983)

[1.6 Опис предметної області та постанова задачі 11](#_Toc195835984)

[1.7 Обґрунтування вибору методу розв’язку задачі 13](#_Toc195835985)

[2 ПРАКТИЧНА ЧАСТИНА 14](#_Toc195835986)

[2.1 Інформаційна структура моделі та алгоритм 14](#_Toc195835987)

[2.2 Функціональні частини програми 15](#_Toc195835988)

[2.3 Інтерфейс та керівництво користувача 18](#_Toc195835989)

[2.4 Тестування проєкту 20](#_Toc195835990)

[ВИСНОВКИ 27](#_Toc195835991)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 28](#_Toc195835992)

[Додаток 1 Бази даних 29](#_Toc195835993)

[Додаток 2. Схема класів 30](#_Toc195835994)

[Додаток 3. Документація до програмного коду 31](#_Toc195835995)

[Додаток 4. Лістинг програми 41](#_Toc195835996)

Актуальність теми проекту зумовлена необхідністю ефективного управління бібліотечними ресурсами в умовах зростання обсягів книжкових фондів та запитів користувачів. У сучасному світі, з поширенням цифрових технологій, бібліотеки стикаються з викликами швидкого пошуку, обліку та організації книг. Відсутність централізованої системи каталогізації може призводити до труднощів у доступі до інформації про книги, управління їх наявністю та відстеження історії видачі. Система каталогізації книг, розроблена з використанням Java, JavaFX та MySQL, покликана вирішити ці проблеми, забезпечуючи прозорість, зручність та оптимізацію процесів управління бібліотечними фондами.

Метою проекту є створення зручного та надійного інструменту для бібліотекарів та адміністраторів бібліотек, який дозволить вести детальний облік книг, їх характеристик, статусів наявності та історії видачі. Така система сприятиме підвищенню ефективності управління бібліотечними ресурсами, спрощенню пошуку книг та оптимізації роботи бібліотеки загалом.

Предметом дослідження є аналіз потреб бібліотек у каталогізації книг, а також вивчення проблем, пов’язаних із відсутністю систематизованого підходу до управління книжковими фондами. Будуть розглянуті сучасні практики та методології каталогізації, щоб інтегрувати найкращі з них у розроблювану систему.

Об’єктом дослідження є система каталогізації книг, створена за допомогою Java, JavaFX та SQLite. Ця система призначена стати ключовим елементом бібліотечної інфраструктури, забезпечуючи високу точність обліку, легкий доступ до актуальної інформації про книги та ефективність у прийнятті управлінських рішень.

КОРОТКІ ВІДОМОСТІ ПРО БАЗУ ПРАКТИКИ

Базою для виробничо-технологічної практики було обрано ТОВ «Комплексна система захисту інформації», яке розташовано за адресою: м.Ужгород, площа Ш.Перефі,47.

Керівником фірми ТОВАРИСТВО З ОБМЕЖЕНОЮ ВІДПОВІДАЛЬНІСТЮ «Комплексна система захисту інформації» є Андрусь Василь Юрійович.

Види діяльності:

* консультування з питань інформатизації;
* комп'ютерне програмування;
* діяльність із керування комп'ютерним устаткованням;
* інша діяльність у сфері інформаційних технологій і комп'ютерних систем.

Основним завданням від бази практики є створення програмного забезпечення для автоматизації обліку роботи з клієнтами.

# **ТЕОРЕТИЧНА ЧАСТИНА**

## **Загальні відомості про засоби розробки**

Вибір реляційної бази даних зумовлений практичною експертизою та попереднім досвідом у роботі з реляційними системами керування базами даних (СКБД). Це спростило і прискорило процес розробки програмного забезпечення.

Для зберігання даних вибрано SQLite, оскільки існує попередній досвід роботи з цією СКБД та успішне використання її протягом навчання.

У якості інструмента для адміністрування та управління базою даних обрано SQLiteStudio, оскільки він надавав широкий функціонал і дозволяв нам поглибитися у роботу з СКБД.

Щодо середовища розробки, то вибором стала IntelliJ IDEA. За період навчання отримано поради від наукового керівника і врахувано власний досвід використання цього середовища під час навчання.

Під час аналізу фреймворків для створення користувацького інтерфейсу вирішено використовувати JavaFX. Цей вибір був зумовлений новаторським підходом та широким спектром компонентів, що надає JavaFX порівняно з іншими аналогами.

Обрані нами рішення допоможуть ефективно розробляти програмне забезпечення для створення системи обліку комп’ютерів на підприємстві.

## **Загальні відомості про Java**

Java (вимовляється Джава) — об'єктно-орієнтована мова програмування, випущена 1995 року компанією «Sun Microsystems» як основний компонент платформи Java. З 2009 року мовою займається компанія «Oracle», яка того року придбала «Sun Microsystems». В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи.

«Oracle» надає компілятор Java та віртуальну машину Java, які задовольняють специфікації Java Community Process, під ліцензією GNU General Public License.

Мова значно запозичила синтаксис із C і C++. Зокрема, взято за основу об'єктну модель С++, проте її модифіковано. Усунуто можливість появи деяких конфліктних ситуацій, що могли виникнути через помилки програміста та полегшено сам процес розроблення об'єктно-орієнтованих програм. Ряд дій, які в С/C++ повинні здійснювати програмісти, доручено віртуальній машині. Передусім Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням, що в порівнянні, наприклад, з C++ зменшує швидкість роботи програм. За необхідності таких дій Java дозволяє викликати підпрограми, написані іншими мовами програмування.

Java вплинула на розвиток J++[en], що розроблялась компанією «Microsoft». Роботу над J++ було зупинено через судовий позов «Sun Microsystems», оскільки ця мова програмування була модифікацією Java. Пізніше в новій платформі «Microsoft» .NET випустили J#, щоб полегшити міграцію програмістів J++ або Java на нову платформу. З часом нова мова програмування С# стала основною мовою платформи, перейнявши багато чого з Java. J# востаннє включався в версію Microsoft Visual Studio 2005. Мова сценаріїв JavaScript має схожу із Java назву і синтаксис, але не пов'язана із Java.

Спочатку мова називалася Oak («дуб») і розроблялася Джеймсом Гослінгом для програмування побутових електронних пристроїв. Згодом вона була перейменована в Java і стала використовуватися для написання клієнтських застосунків і серверного програмного забезпечення. Названа на честь марки кави Java, яка, в свою чергу, отримала найменування однойменного острова (Ява), тому на офіційній емблемі мови зображена чашка з паркою кавою. Існує й інша версія походження назви мови, пов'язана з алюзією на каво-машину як приклад побутового устаткування, для програмування якого спочатку мова створювалася [2].

## **Загальні відомості про IntelijIdea**

IntelliJ IDEA — комерційне інтегроване середовище розробки для різних мов програмування (Java, Python, Scala, PHP та ін.) від компанії JetBrains. Система поставляється у вигляді урізаної по функціональності безкоштовної версії «Community Edition» і повнофункціональної комерційної версії «Ultimate Edition», для якої активні розробники відкритих проектів мають можливість отримати безкоштовну ліцензію. Програмний код Community-версії поширюються рамках ліцензії Apache 2.0. Бінарні збірки підготовлені для Linux, Mac OS X і Windows.

Перша версія IntelliJ IDEA з'явилася у січні 2001 року й швидко здобула популярність, як перша Java IDE із широким набором інтегрованих інструментів для рефакторингу, що дозволяла програмістам швидко реорганізовувати програмний код програм. Дизайн середовища орієнтовано на продуктивність праці програмістів, дозволяючи їм сконцентруватися на розробці функціональності, тоді як IntelliJ IDEA бере на себе виконання рутинних операцій.

Починаючи з шостої версії продукту IntelliJ IDEA надає інтегрований інструментарій для розробки графічного користувацького інтерфейсу.

З версії 9.0 є безкоштовний варіант Community Edition з відкритими кодами. Програмний код відкритої версії IntelliJ IDEA Community Edition поширюються рамках ліцензії Apache 2.0. Бінарні пакунки підготовлені для Linux, Mac OS X і Windows.

До складу IntelliJ IDEA включені напрацювання, створені в результаті спільної роботи з компанією Google, яка використовувала IntelliJ IDEA як базис для свого нового відкритого середовища розробки Android Studio. Завдяки співпраці істотно розширені штатні можливості IntelliJ IDEA з розробки застосунків для платформи Android [7].

## **Загальні відомості про SQLite**

SQLite — полегшена реляційна система керування базами даних, реалізована як компактна бібліотека, що підтримує більшу частину стандарту SQL-92. Початковий код SQLite поширюється як суспільне надбання (public domain), що дозволяє використовувати його без будь-яких обмежень і безкоштовно для будь-яких цілей, включаючи комерційні, академічні чи особисті проєкти. Фінансову підтримку розробки SQLite забезпечує консорціум, до складу якого входять провідні технологічні компанії, такі як Adobe, Oracle, Mozilla, Nokia, Bentley Systems і Bloomberg, що свідчить про довіру до цієї технології в індустрії.

З 2018 року SQLite, поряд із форматами JSON і CSV, рекомендований Бібліотекою Конгресу США як стандарт для зберігання структурованих даних, що підкреслює його надійність, універсальність і довгострокову перспективу використання. SQLite вирізняється своєю портативністю, мінімальними вимогами до ресурсів і можливістю роботи без окремого серверного процесу, що робить її ідеальним вибором для вбудованих систем, мобільних додатків, настільних програм і невеликих проєктів, таких як система каталогізації книг, розроблювана на Java, JavaFX і SQLite.

У 2005 році проєкт SQLite отримав престижну нагороду Google-O'Reilly Open Source Awards, що підтверджує його значний внесок у розвиток відкритих технологій і визнання серед спільноти розробників. SQLite підтримує основні можливості реляційних баз даних, включаючи транзакції, індекси, тригери та представлення, при цьому зберігаючи компактність — розмір бібліотеки становить лише близько 1 МБ. Це дозволяє інтегрувати її навіть у пристрої з обмеженими ресурсами, такі як IoT-пристрої чи смартфони.

SQLite широко застосовується в різноманітних програмах і системах завдяки своїй простоті, високій продуктивності та кросплатформенності. Наприклад, вона використовується у веббраузерах (Mozilla Firefox, Google Chrome), операційних системах (Android, iOS, macOS), а також у популярних застосунках, таких як Skype і Dropbox. У контексті розробки системи каталогізації книг SQLite є оптимальним вибором, оскільки забезпечує швидкий доступ до даних, ефективне управління бібліотечними записами (такими як інформація про книги, авторів, жанри та історію видачі) і легку інтеграцію з Java через JDBC-драйвер. Крім того, SQLite підтримує одночасний доступ до бази даних у багатопотокових програмах, що може бути корисним для реалізації функцій пошуку та оновлення записів у реальному часі.

Для забезпечення надійності SQLite використовує механізм журналювання (WAL — Write-Ahead Logging), який гарантує цілісність даних навіть у разі збоїв системи. Це особливо важливо для бібліотечних систем, де втрата даних про книги чи транзакції може призвести до серйозних незручностей. Завдяки відсутності необхідності в складній конфігурації чи адмініструванні, SQLite дозволяє розробникам зосередитися на логіці програми, а не на управлінні базою даних, що спрощує створення таких систем, як зазначена система каталогізації книг.

Відповідність пам’яті користувача: З попередніх розмов (16 квітня 2025, 17 квітня 2025) відомо, що ви працюєте над проєктом "Library Management System" із використанням Java 22, JavaFX 22 і SQLite. SQLite ідеально підходить для вашого проєкту завдяки своїй легкості та простоті інтеграції через JDBC, що відповідає вашим потребам у створенні інтерфейсу для управління бібліотечними записами, включаючи функції редагування, видалення та пагінацію, як зазначено в попередніх запитах [6].

## **Огляд аналогів**

Система **Торгсофт** для книгарень, представлена на сайті torgsoft.ua, є спеціалізованим програмним забезпеченням для автоматизації управління книжковими магазинами. Вона оптимізує бізнес-процеси, але має як переваги, так і недоліки. Торгсофт забезпечує повний облік книг, включаючи залишки, штрихкодування та продажі. Підтримка імпорту/експорту даних у XLS/XLSX спрощує роботу з постачальниками. Синхронізація з вебсайтами оновлює дані про наявність і ціни, автоматизує замовлення та сповіщення клієнтів через SMS/Viber. Програма підходить для малих і великих книгарень, має безстрокову ліцензію без абонплати та пропонує 30-денну демоверсію. Інтеграція з логістикою (Нова Пошта, Укрпошта), платіжними (Monobank, Privat24) і бухгалтерськими системами (M.E.Doc) спрощує операції. Інтерфейс може бути складним для користувачів без досвіду, а навчання вимагає часу. Разовий платіж за ліцензію може бути високим для малих книгарень, попри відсутність абонплати. Налаштування під специфічні потреби може бути обмеженим, що створює труднощі для нестандартних процесів. Для повноцінної роботи потрібне додаткове обладнання (сканери, принтери), що збільшує витрати [8].

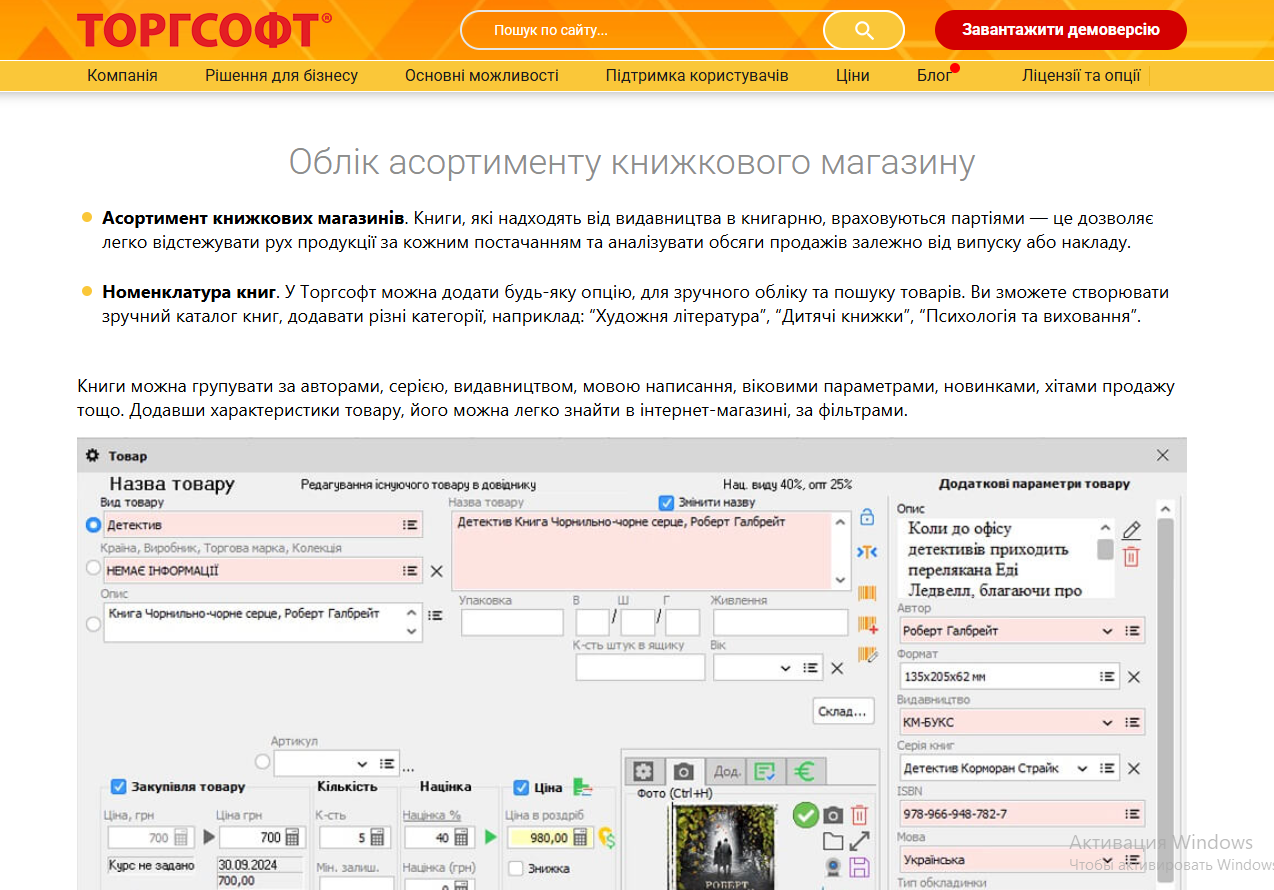


Рисунок 1.1 Аналог проєкту

## Опис предметної області та постанова задачі

Розглядається можливість розробки системи журналу обліку роботи з клієнтами для підприємства з використанням Java для бекенд-розробки, JavaFX для створення графічного інтерфейсу користувача, а також MySQL для бази даних. Головною метою системи є автоматизація процесів ведення записів про взаємодії з клієнтами, включно з управлінням зустрічами, переговорами, угодами, а також історією обслуговування та відгуками клієнтів.

Основні функції системи включатимуть реалізацію механізмів CRUD (створення, читання, оновлення, видалення) для записів про кожного клієнта, його переваги, потреби, історію взаємодії та результати співпраці. Важливим аспектом системи стане можливість реєстрації та аутентифікації користувачів, включаючи менеджерів з продажу, службу підтримки та адміністраторів системи, що дозволить забезпечити диференційований доступ до даних залежно від ролі та завдань.

Проект має на меті підвищення ефективності взаємодії з клієнтами, оптимізацію процесів продажу та служби підтримки, а також зниження ризиків втрати важливої інформації про клієнтів. Ключовою задачею стане забезпечення безпеки зберігання та обробки персональних даних клієнтів, для чого будуть використані сучасні механізми шифрування та захисту інформації.

Необхідно також створити зручний і інтуїтивно зрозумілий графічний інтерфейс, який дозволить користувачам легко навігувати по системі та ефективно управляти інформацією про клієнтів. Важливим є застосування передових технологій веб-розробки для створення клієнтської частини системи, а також ретельне тестування юзабіліті для забезпечення високої якості користувацького досвіду.

Таким чином, розробка системи журналу обліку роботи з клієнтами має стати ключовим інструментом для ефективного управління клієнтською базою, підвищення задоволеності клієнтів та сприяння прийняттю обґрунтованих рішень в сфері продажів і обслуговування.

## Обґрунтування вибору методу розв’язку задачі

Розглядається можливість розробки системи каталогізації книг для бібліотеки з використанням Java для бекенд-розробки, JavaFX для створення графічного інтерфейсу користувача, а також SQLite для бази даних. Головною метою системи є автоматизація процесів обліку книжкового фонду, включаючи управління інформацією про книги, їх наявність, видачу, повернення та відгуки користувачів.

Основні функції системи включатимуть реалізацію механізмів CRUD (створення, читання, оновлення, видалення) для записів про кожну книгу, її автора, жанр, рік видання, статус наявності та історію видачі. Важливим аспектом стане можливість реєстрації та аутентифікації користувачів, таких як бібліотекарі, адміністратори та читачі, що забезпечить диференційований доступ до даних залежно від ролі: від повного доступу для бібліотекарів до обмеженого для читачів.

Проект має на меті підвищення ефективності управління бібліотечними ресурсами, спрощення пошуку книг, оптимізацію процесів видачі та зниження ризиків втрати інформації про книжковий фонд. Ключовою задачею стане забезпечення безпеки зберігання даних, зокрема інформації про користувачів і книги, для чого будуть використані сучасні механізми шифрування та захисту бази даних SQLite.

Необхідно створити зручний і інтуїтивно зрозумілий графічний інтерфейс, який дозволить користувачам легко навігувати по системі, швидко знаходити книги та управляти записами. Важливим є застосування сучасних технологій JavaFX для створення клієнтської частини, а також ретельне тестування юзабіліті для забезпечення комфортного користувацького досвіду.

Таким чином, розробка системи каталогізації книг стане ключовим інструментом для ефективного управління бібліотечним фондом, підвищення якості обслуговування читачів і сприяння обґрунтованим рішенням у процесі організації бібліотечної роботи.

# **ПРАКТИЧНА ЧАСТИНА**

Практична частина складається з окремих підрозділів, які створюються на основі частин розробленого програмного сервісу. В практичній частині описано всі функціональні частини, інтерфейс та керівництво користувача. Протестовано головні модулі, описано вхідні і вихідні дані. В даному розділі детально описано алгоритм роботи розробленої системи.



## **2.1 Інформаційна структура моделі та алгоритм**

Алгоритмічні моделі, які лягли в основу розробки програми:

* Реалізація алгоритму авторизації користувача
* Створення інформативних сторінок для показу в реальному часі всіх книг які є в наявності.
* Реалізація CRUD та інформаційного наповнення програми.

ERD-діаграма БД (рисунок 2.1) наглядно відображає всі таблиці БД та їх поля. БД включає такі таблиці:

* «users» — перелік користувачів адмін-панелі;
* «status» — допоміжна таблиця, яка встановлює актуальний статус наявності книги;
* «books» — перелік всіх книг які є в бібліотеці;
* «booktheytook» — перелік книг які взяли відвідувачі;
* «ganre» — допоміжна таблиця, в встановлює жанри книг;
* «author» — допоміжна таблиця, яка встановлює автора книги;

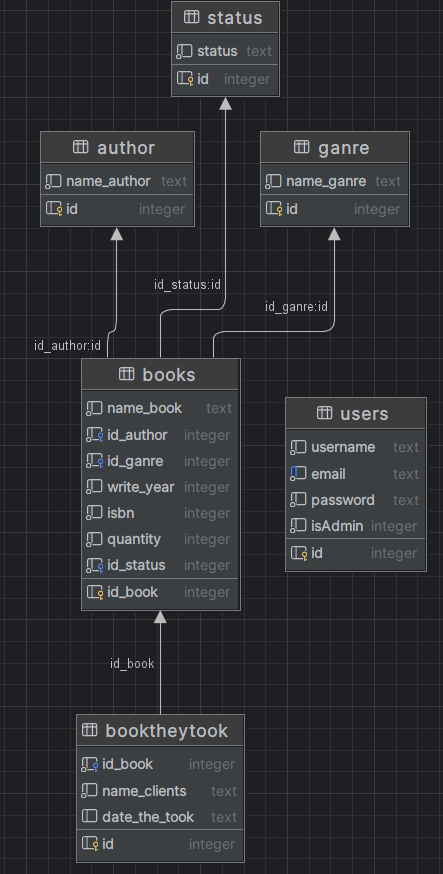


Рисунок 2.1 ERD-діаграма бази даних

## **2.2 Функціональні частини програми**

Для реалізації проєкту, використано середовище IntellijIDEA (рисунок 2.2).

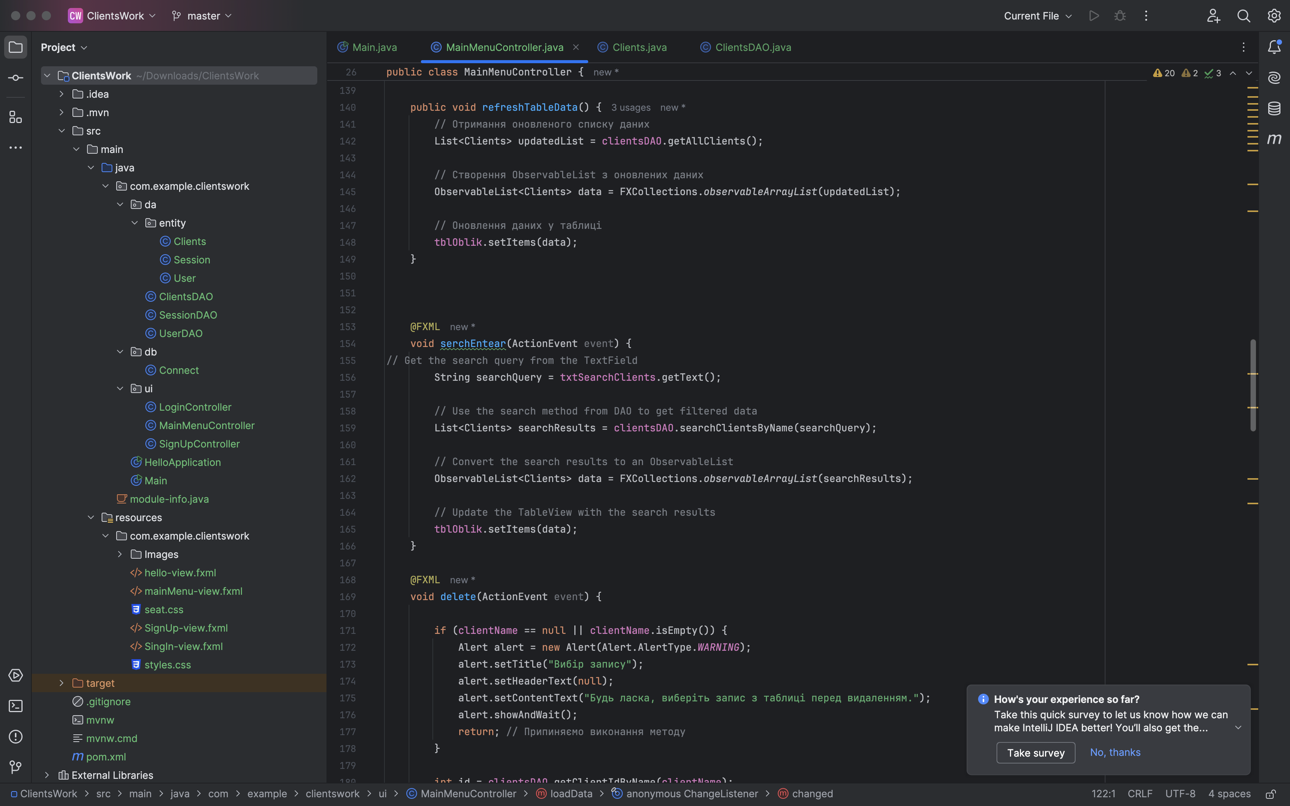


Рисунок 2.2 Проєкт в IntellijIDEA

Структура проєкту реалізована за актуальним алгоритмом створення проектів, де всі файли групуються логічно в залежності від їх призначення, тобто каталог «ui» містить лиш ті підкаталоги і файли, що стосуються користувацького інтерфейсу.

Структура проєкту зображена на рисунку 2.3.

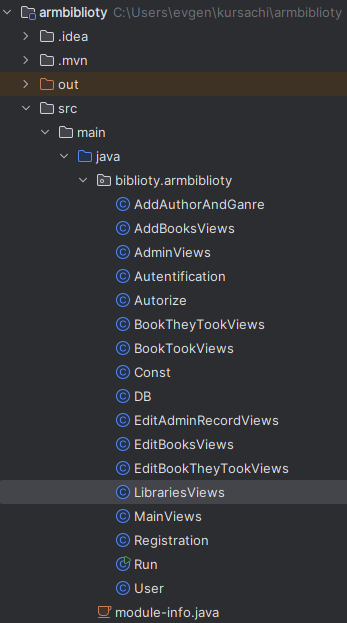
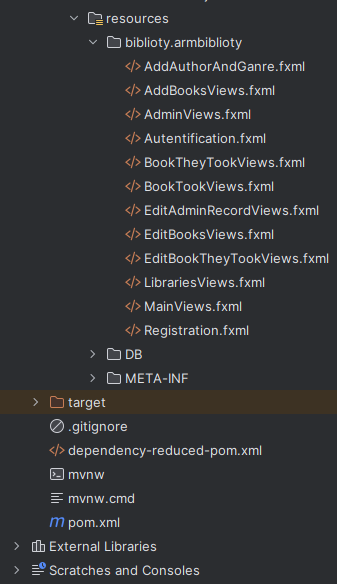
 

Рисунок 2.3 Структура проєкту

Опис головних функціональних файлів проєкту викладено в таблиці 2.1.

Таблиця 2.1 Опис файлів

|  |  |
| --- | --- |
| **Бібліотеки:**  java.sql.Connection; | **Опис**  Бібліотека для роботи з базами данних і використання її в проєкті |
| javafx.application.Application; | Бібліотека для роботи з користувацбким інтерфейсом |
| java.util.List; | Клас ArrayList представляє узагальнену колекцію, яка успадковує свою функціональність від класу AbstractList і застосовує інтерфейс List . |
| java.util.Scanner; | Сканер - це клас на мові Java , який дозволяє розраховувати дані з різних джерел. В тому числі і з консоллю |
| java.io.IOException | Клас IOException є базовим класом для винятків, що виникають при доступі до даних за допомогою потоків, файлів та каталогів . Бібліотека базових класів включає такі типи, кожен з яких є похідним класом IOException: DirectoryNotFoundException. EndOfStreamException. |
| **Змінні:** | **Опис** |
| Map<String, Integer> authorMap | Масив з даними авторів книг. |
| String tableName | Змінна для отримання назви таблиці |
| String loginText | Змінна, яка зчитує ввід логіна користувачем. |
| String loginPassword | Змінна, яка зчитує ввід пароля користувачем. |
| User(int userId, String nameUser, String email, String password, int isAdmin) | Конструктор для роботи з даними користувача |
| User user | Змінна проходить по всьому масиву користувачів |

*Продовження таблиці 2.4*

|  |  |
| --- | --- |
| int bookId | Змінна, яка отримує ідентифікатор книги |
| ObservableList<Book> booksData | Масив, який отримує в себе всі книги |
| String name | Змінна, яка записує імʼя відвідувача бібліотеки |
| Connection dbConnection | Статичне з'єднання для одноразового використання бази данних |
| public Record(int id, int bookId, String clientName, String bookName, String dateTook) | Конструктор для отримання даних по книгах які взяли з бібліотеки |
| private String email; | Змінна, яка записує емейл |

## **2.3 Інтерфейс та керівництво користувача**

* Інтерфейс системи обліку комп’ютерів створюється з використанням JavaFX, що дозволяє розробити графічний користувацький інтерфейс (GUI) із зручною взаємодією для користувачів. Компоненти інтерфейсу включають:
* JavaFX Scene Builder — візуальний інструмент для створення графічного інтерфейсу, який дозволяє розробникам легко створювати компоненти інтерфейсу без прямого написання коду. Він служить "скелетом" програми.
* Java код — виконує логіку взаємодії інтерфейсу з користувачем і базою даних MySQL, реагуючи на дії користувача, такі як натискання кнопок, введення даних у форми тощо. Це дозволяє системі динамічно відповідати на запити користувача і забезпечує інтерактивність додатку.
* Робота з системою починається з головного вікна програми, яке забезпечує навігаційне меню для доступу до основних функцій системи.

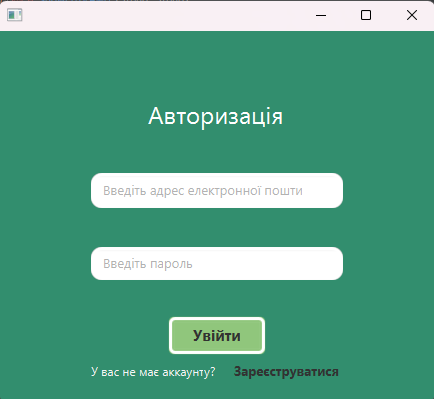


Рисунок 2.4 Вікно аутентифікації

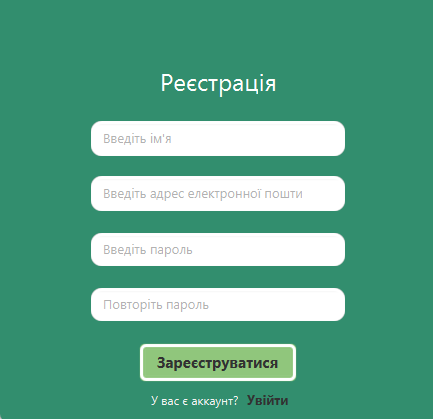


Рисунок 2.5 Вікно реєстрації

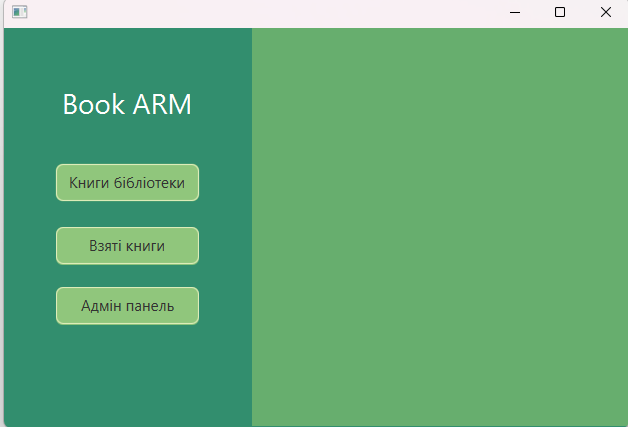


Рисунок 2.6 Головне меню адміністратора

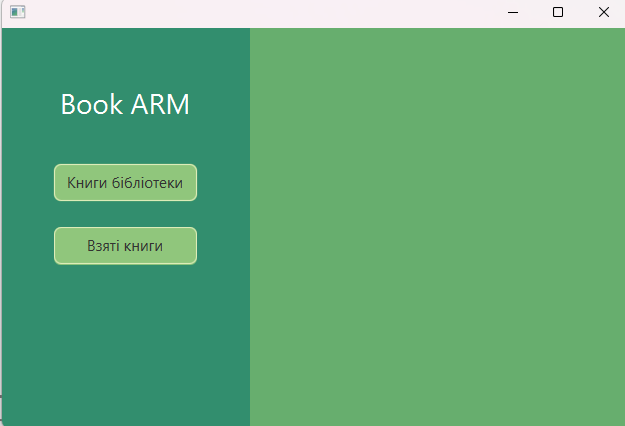


Рисунок 2.7 Головне меню користувача

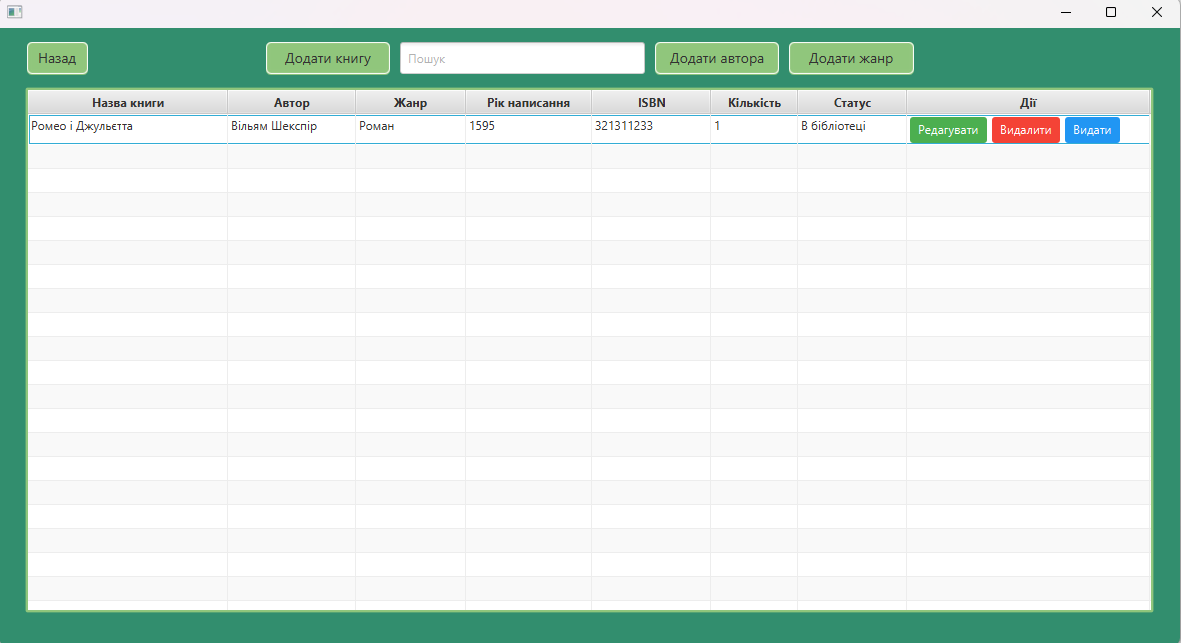


Рисунок 2.8 Вікно з каталогом книг

## **2.4 Тестування проєкту**

Тестування проводиться на платфомі Windows.

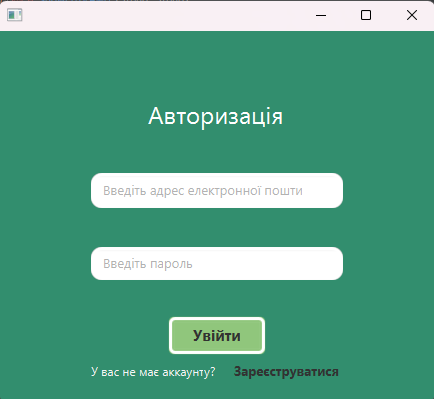


Рисунок 2.9 Вікно аутентифікації

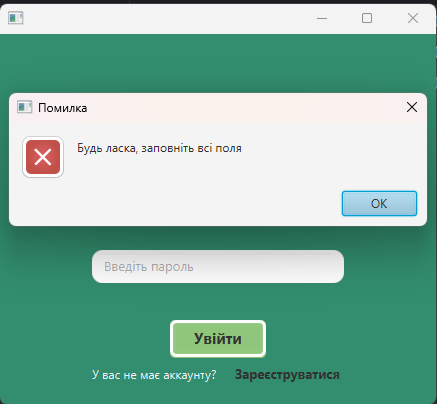


Рисунок 2.10 Реакція додатку на пусті поля

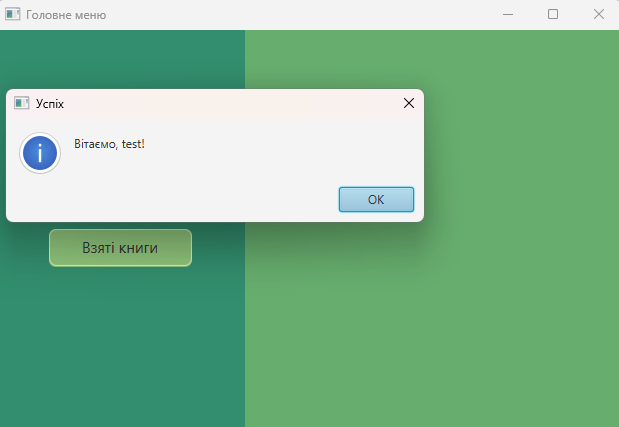


Рисунок 2.11 Успішна автентифікація

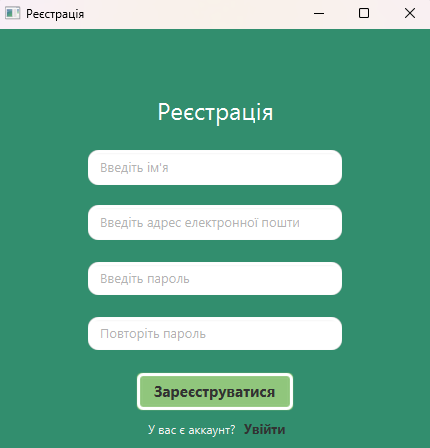


Рисунок 2.12 Вікно реєстрації

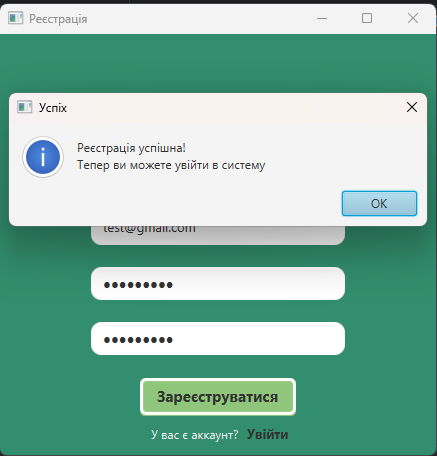


Рисунок 2.13 Успішна реєстрація

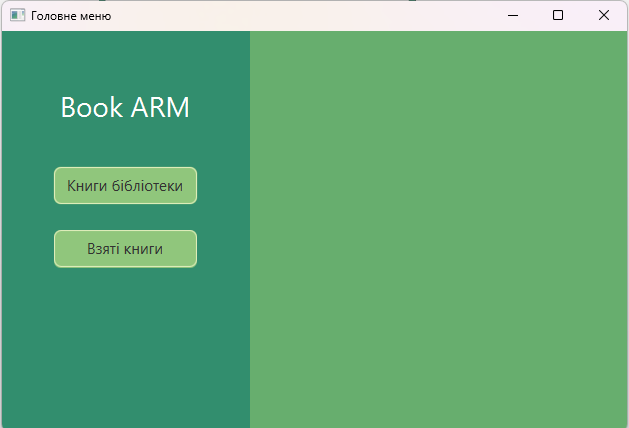


Рисунок 2.14 Головне меню

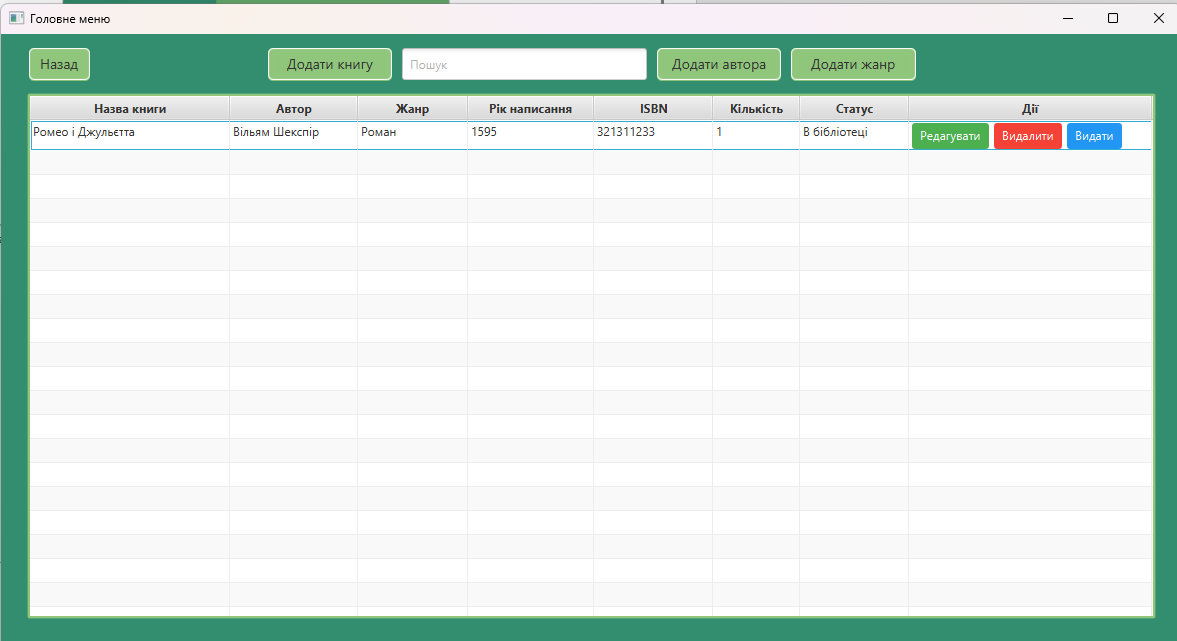


Рисунок 2.15 Вікно з каталогом книг

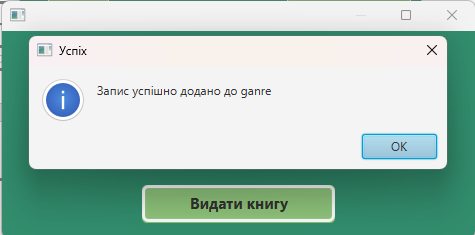


Рисунок 2.16 Вікно додавання жанру книг і успішне додавання жанру

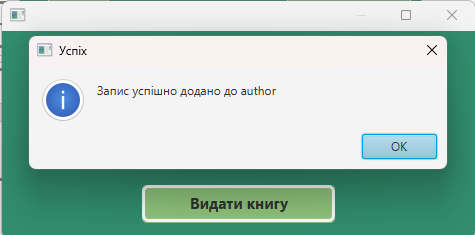
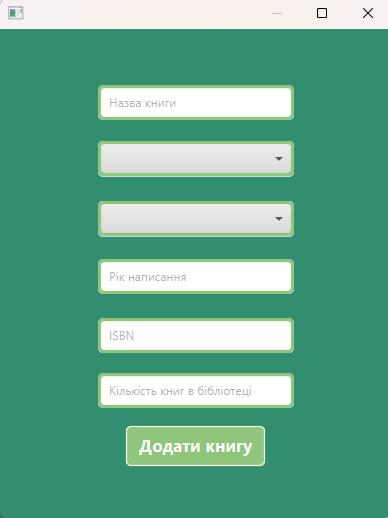
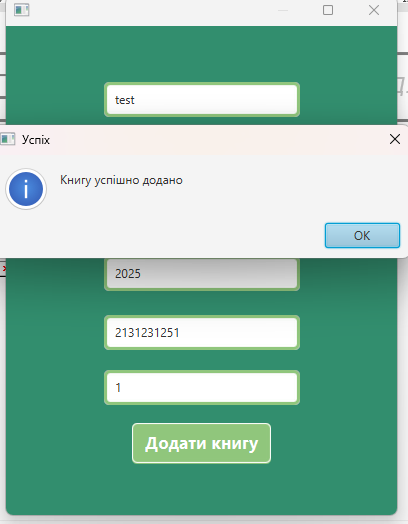


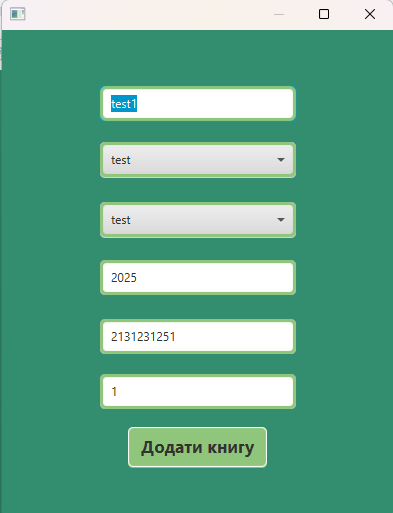
Рисунок 2.17 Вікно додавання автору книг і успішне додавання автора



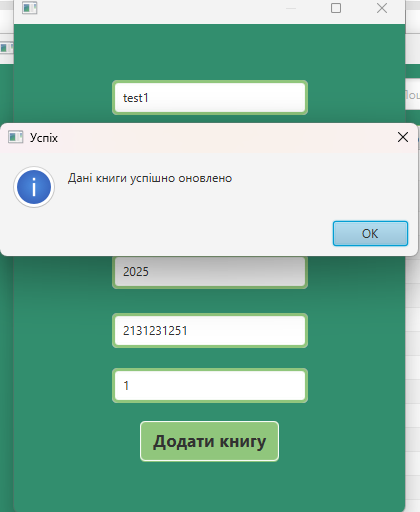
Рисунко 2.18 Вікно додавання книг



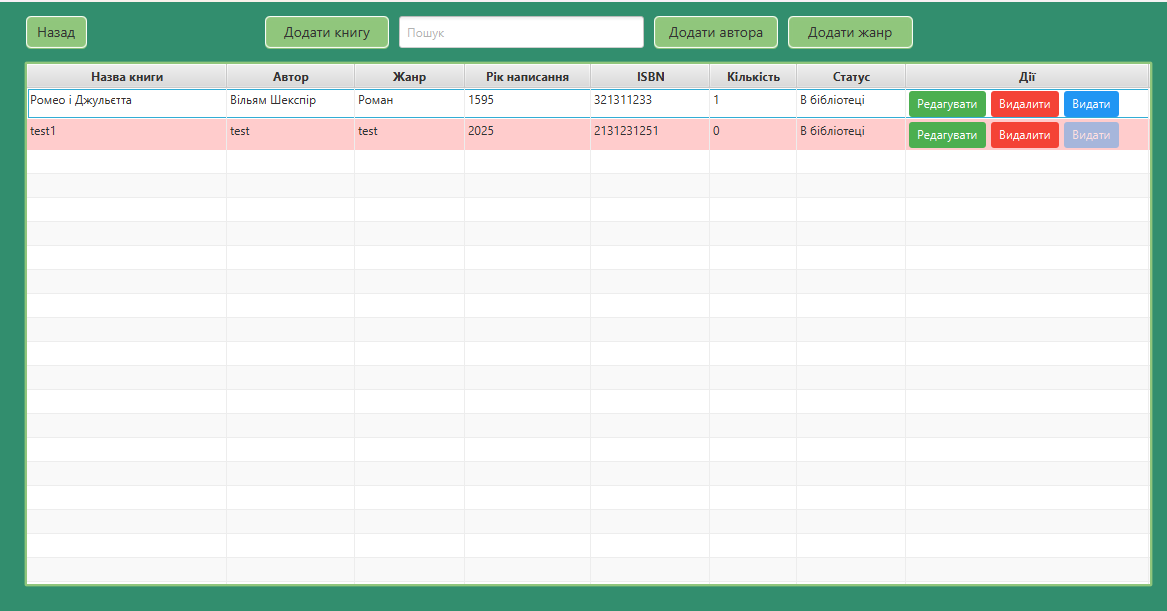
Рисунко 2.19 Успішно додана книга



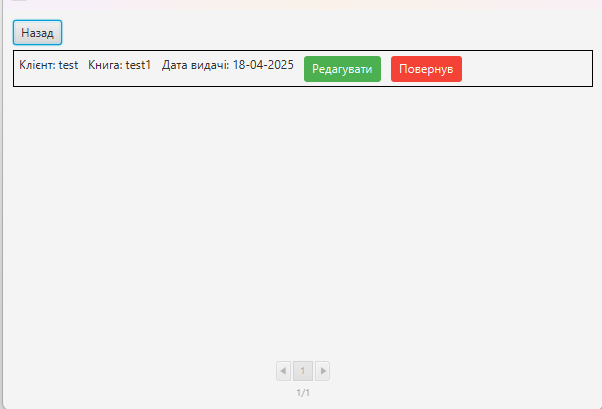
Рисунко 2.20 Вікно редагування



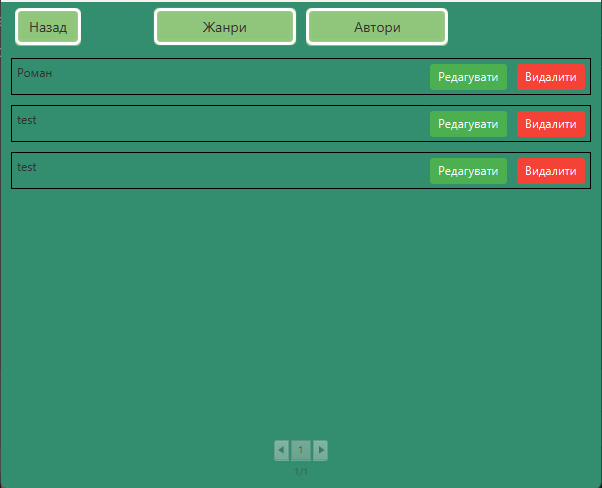
Рисунко 2.21 Успішне редагування

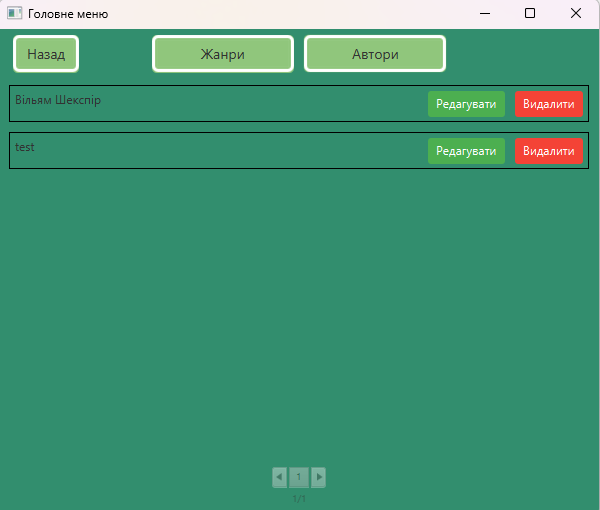


Рисунко 2.22 Такий вигляд має таблиця коли всі екземпляри книг видані

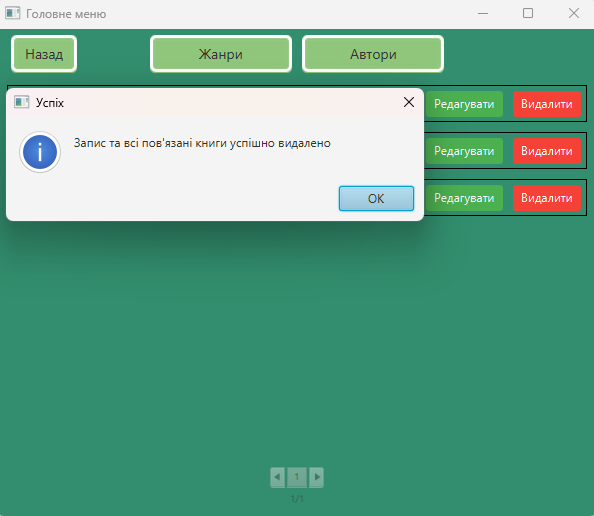


Рисунко 2.23 Вікно з взятими книгами

  
Рисунко 2.24 Адмін панель з жанрами



Рисунко 2.25 Адмін панель з авторами

  
Рисунко 2.26 Успішно видалений запис

ВИСНОВКИ

Висновок до проекту системи каталогізації книг підкреслює важливість і необхідність впровадження інтегрованої та автоматизованої системи для управління бібліотечним фондом, розробленої з використанням Java, JavaFX та SQLite. Успішне впровадження такої системи не лише підвищить ефективність управління книгами, але й дозволить бібліотеці оптимізувати процеси видачі, пошуку та обліку літератури, підвищити задоволеність користувачів і забезпечити зручний доступ до бібліотечних ресурсів.

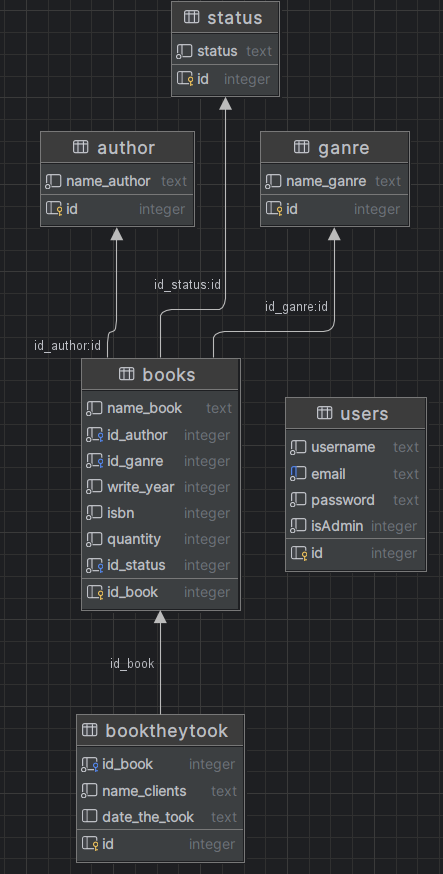
Проект базується на ретельному аналізі потреб бібліотеки в точному обліку книг, управлінні інформацією про їх наявність, авторів, жанри та історію видачі, враховуючи проблеми, що виникають через відсутність централізованого підходу. Розробка та впровадження системи каталогізації книг стане ключовим елементом у стратегії підвищення якості обслуговування читачів і підтримки ефективної роботи бібліотеки.

З огляду на вищезазначене, проект системи каталогізації книг є важливим кроком у майбутнє, що дозволяє оптимізувати бібліотечні процеси, забезпечити швидкий доступ до інформації та адаптуватися до зростаючих потреб користувачів. Розробка такої системи відповідає сучасним тенденціям цифровізації бібліотечної діяльності та є критично важливим елементом для забезпечення високого рівня задоволеності читачів і ефективності роботи бібліотеки в цілому.

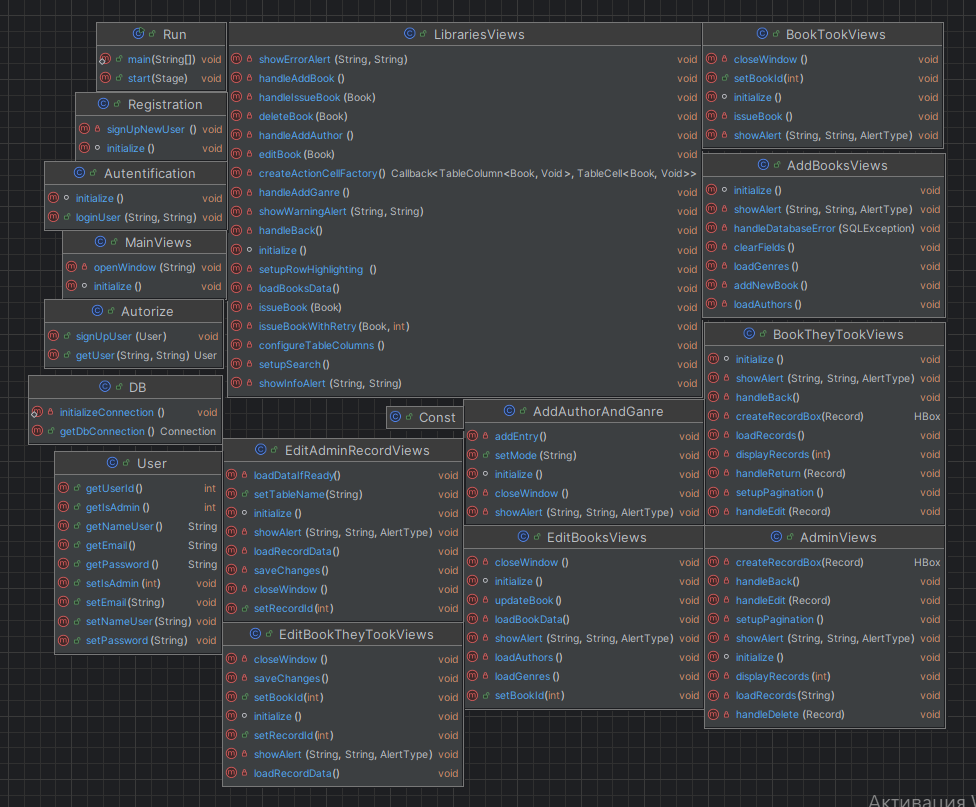
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сайт «ПГФК». Про підприємство. URL: https://www.college.uzhnu.edu.ua/ (дата звернення 20.03.2025);
2. Wikipedia. 2024. JAVA. URL: uk.wikipedia.org/wiki/Java (дата звернення 21.03.2025);
3. Wikipedia. 2025. OOP. URL: uk.wikipedia.org/wiki/oop (дата звернення 22.03.2025);
4. Socialite. 2025. Socialite Providers. URL: socialiteproviders.com/about/ (дата звернення 25.03.2025);
5. Wikipedia. 2025. SceneBuilder. URL: uk.wikipedia.org/wiki/ SceneBuilder (дата звернення 27.03.2025);
6. Wikipedia. 2025. SQLite. URL: uk.wikipedia.org/wiki/SQLite (дата звернення 28.03.2025);
7. Wikipedia IntellijIDEA URL: https://uk.wikipedia.org/wiki/IntelliJ\_IDEA (дата звернення 01.04.2025);
8. ТОРГСОФТ. 2025. ТОРГСОФТ URL: https://torgsoft.ua/soft/programa-dlya-knigarn/ (дата звернення 09.04.2025).

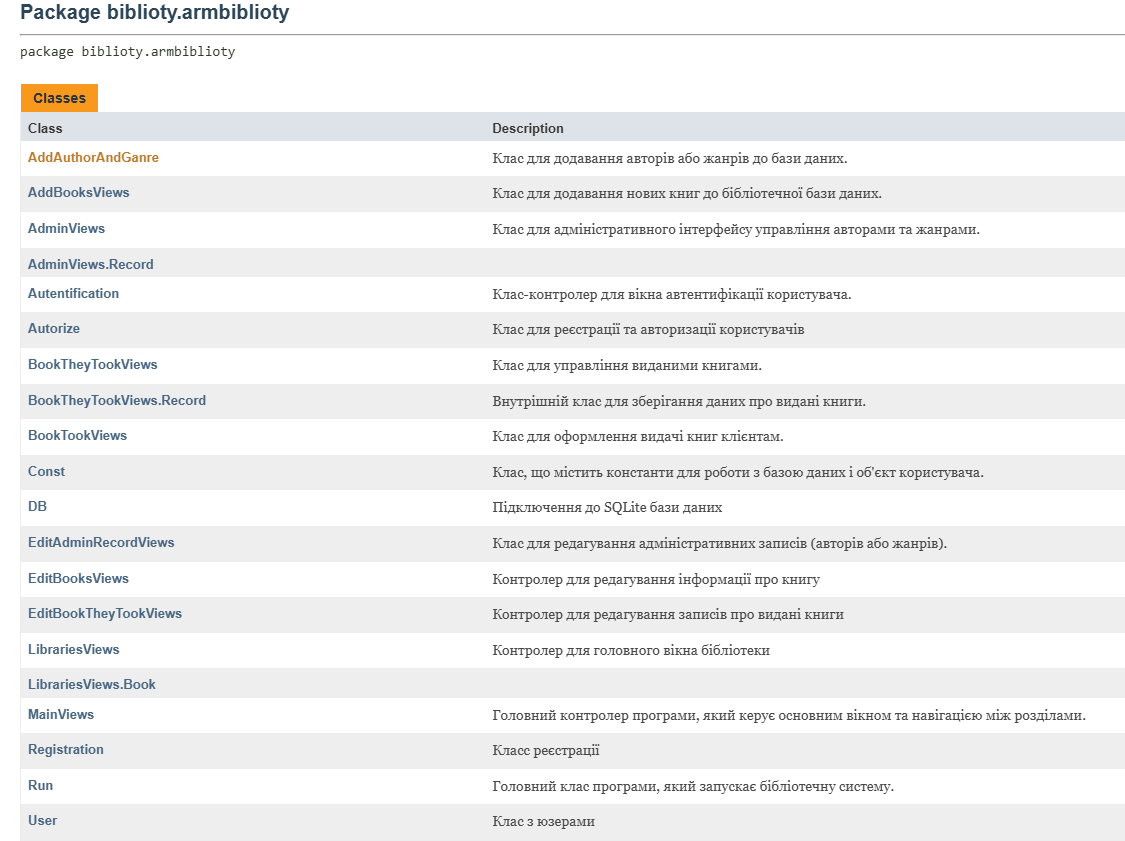
Додаток 1 Бази даних

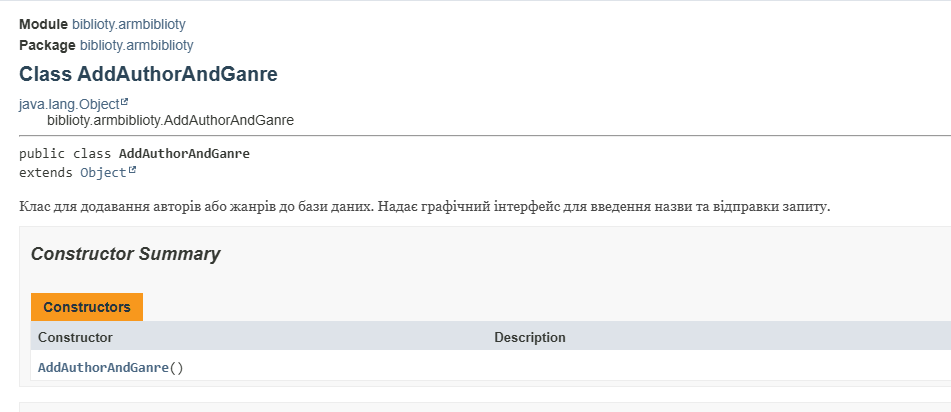


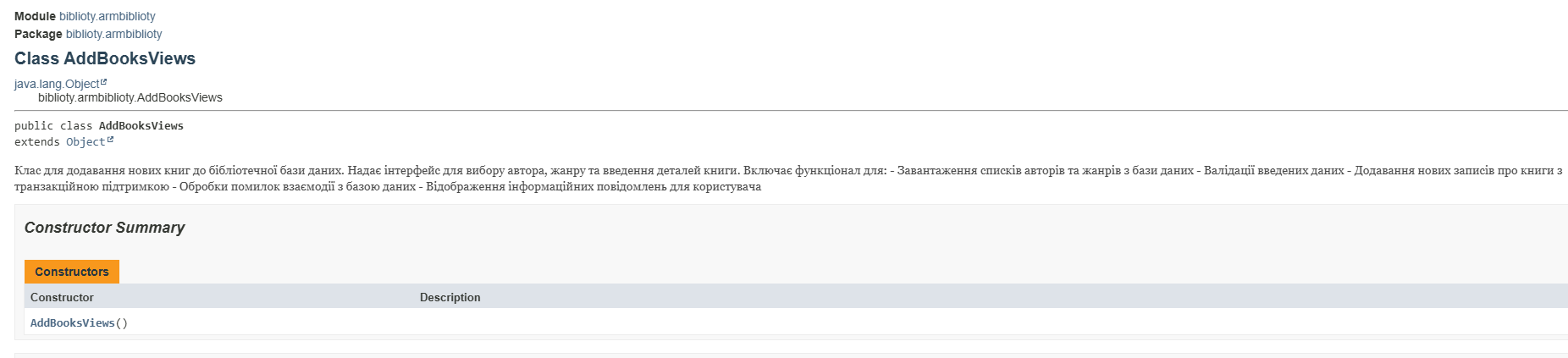
Додаток 2. Схема класів

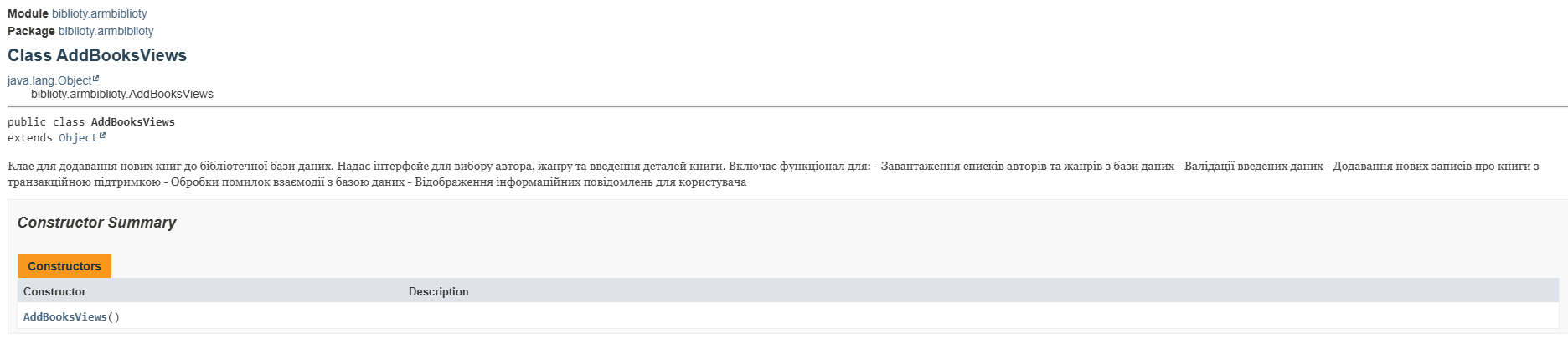


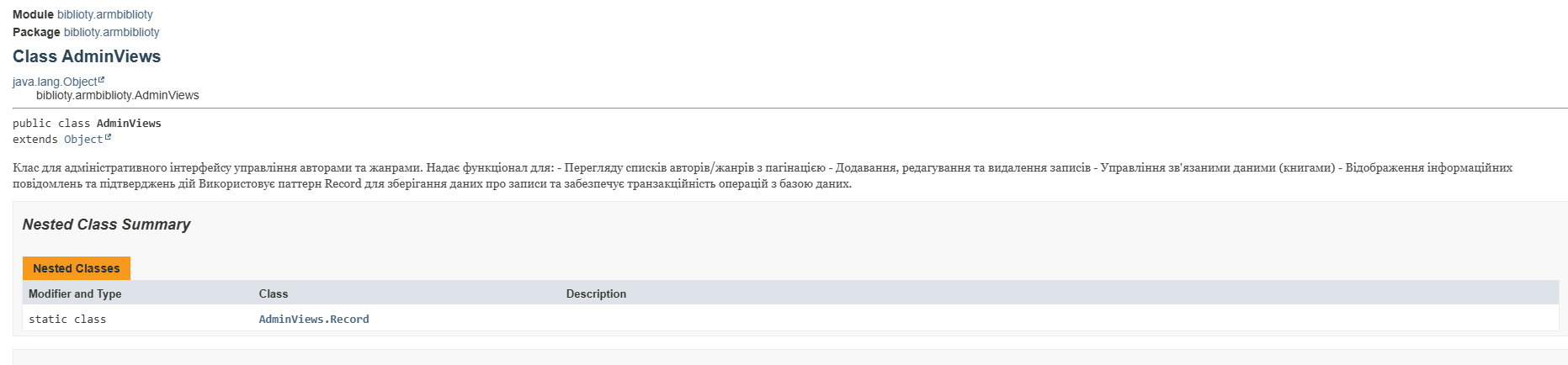
Додаток 3. Документація до програмного коду

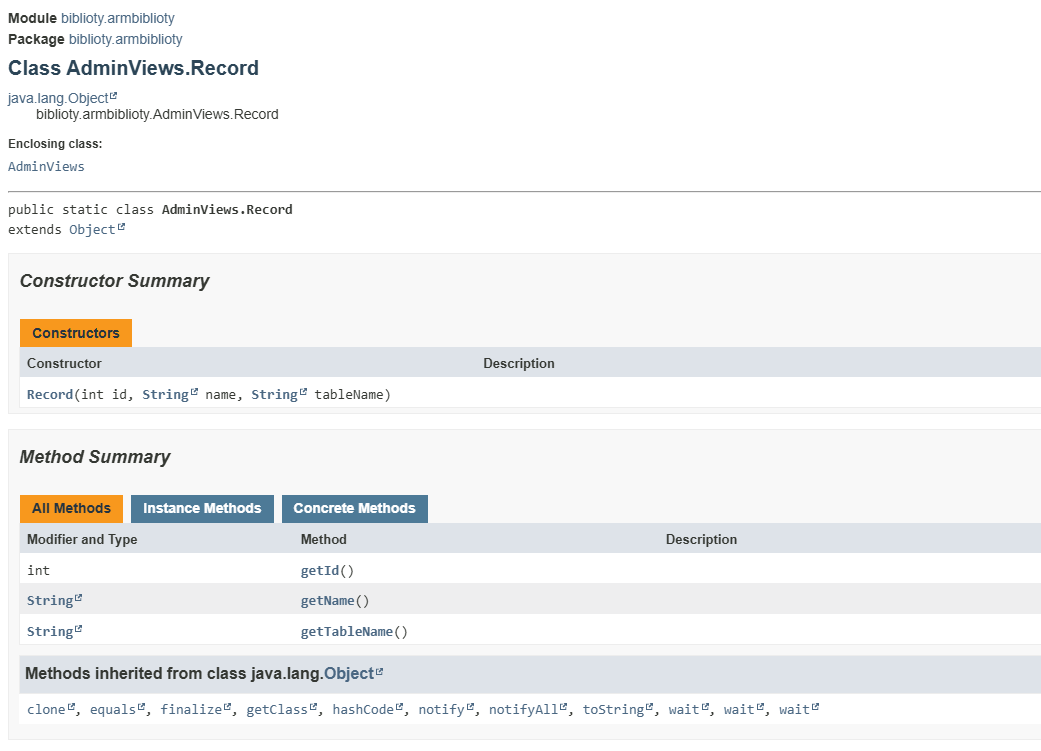


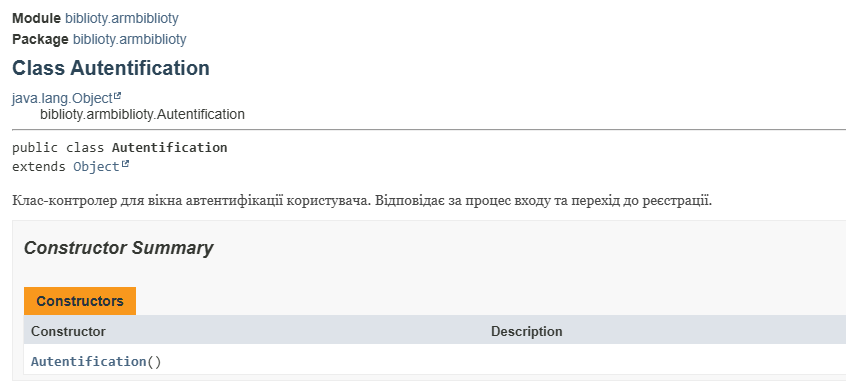


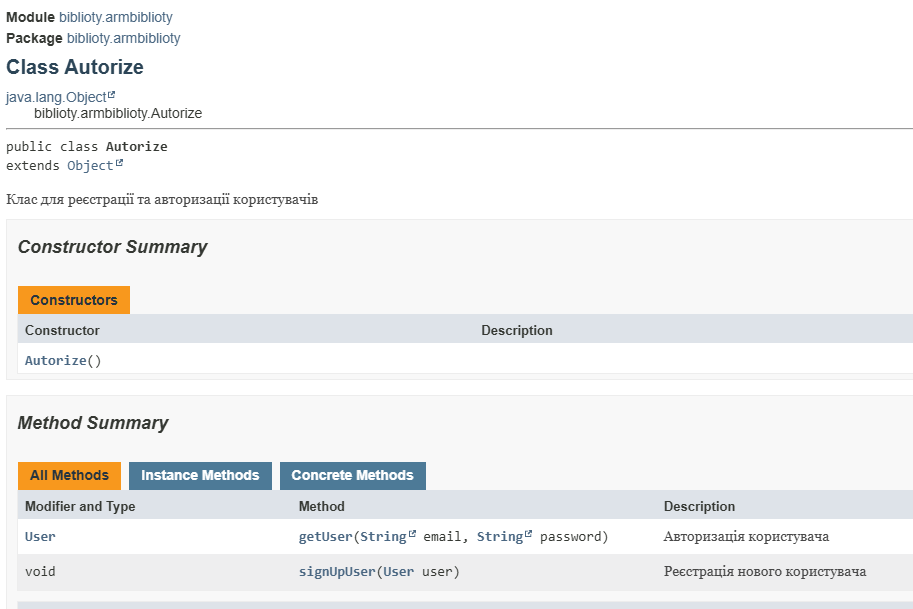


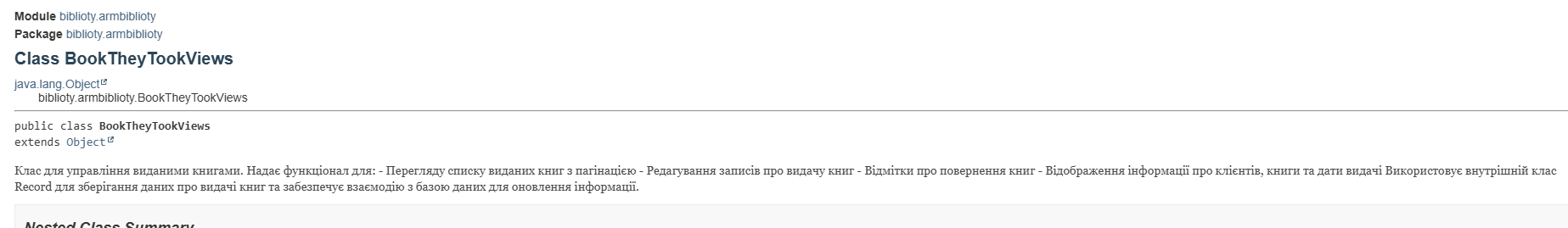
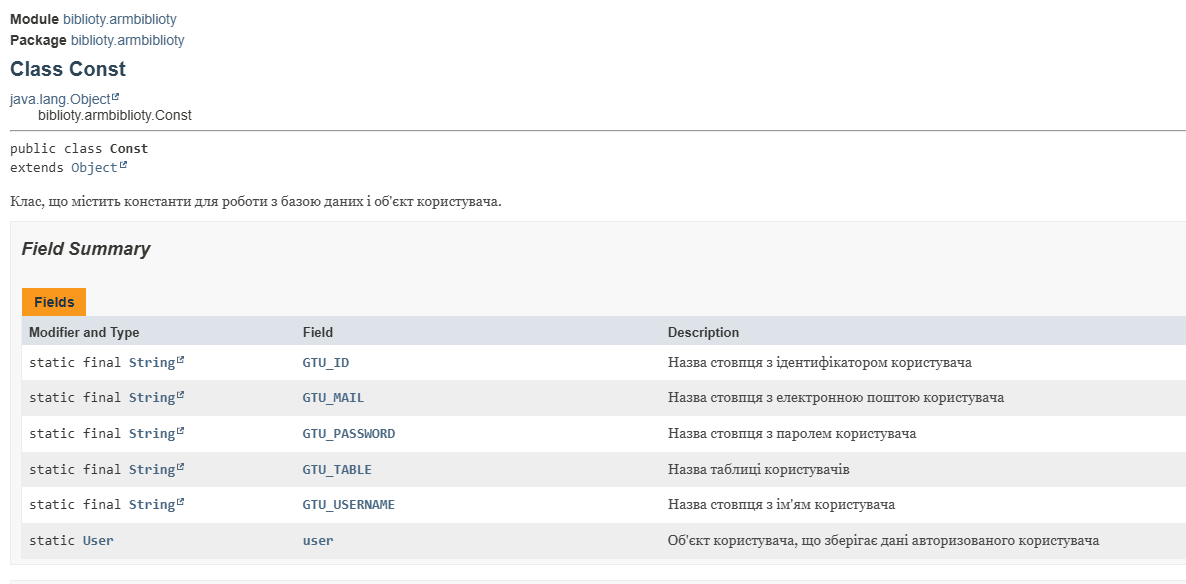


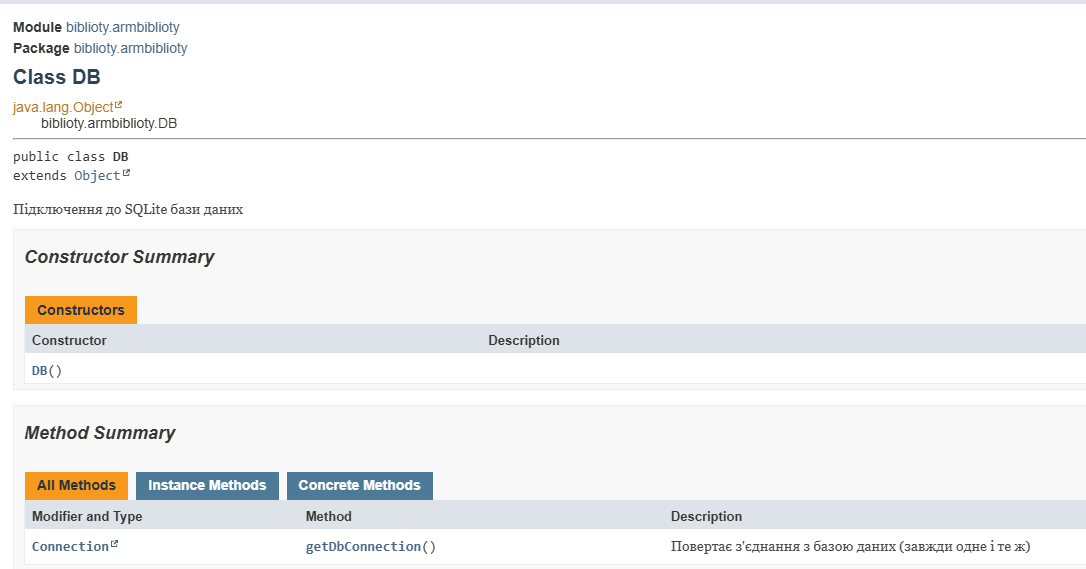




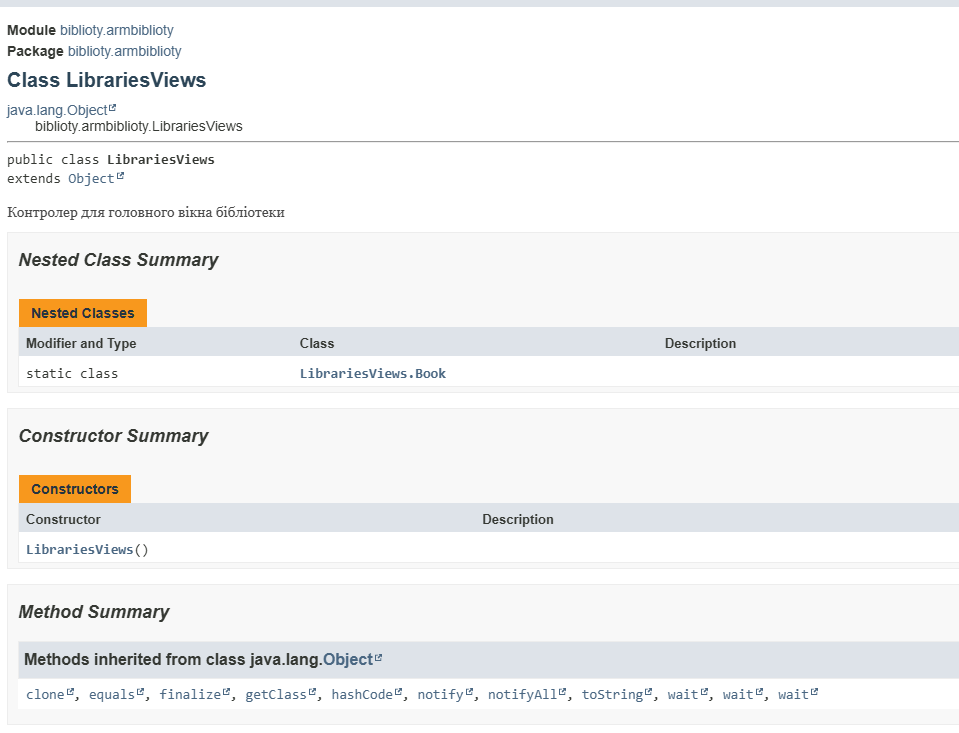










Додаток 4. Лістинг програми

**AddBooksViews**

package biblioty.armbiblioty;

import java.net.URL;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.HashMap;

import java.util.Map;

import java.util.ResourceBundle;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.ChoiceBox;

import javafx.scene.control.TextField;

public class AddBooksViews {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button AddBooks;

@FXML

private ChoiceBox<String> GanreChoicBox;

private Map<String, Integer> ganreMap = new HashMap<>();

@FXML

private TextField ISBNField;

@FXML

private TextField NameBookField;

@FXML

private TextField QuantityField;

@FXML

private ChoiceBox<String> authorChoicBox;

private Map<String, Integer> authorMap = new HashMap<>();

@FXML

private TextField writeYearField;

@FXML

void initialize() {

loadAuthors();

loadGenres();

AddBooks.setOnAction(event -> {

try {

addNewBook();

} catch (SQLException e) {

handleDatabaseError(e);

}

});

}

private void loadAuthors() {

try {

DB db = new DB();

try (Connection conn = db.getDbConnection()) {

String query = "SELECT id, name\_author FROM author";

try (PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

ObservableList<String> authors = FXCollections.observableArrayList();

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name\_author");

authors.add(name);

authorMap.put(name, id);

}

authorChoicBox.setItems(authors);

}

}

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося завантажити авторів", Alert.AlertType.ERROR) }

private void loadGenres() {

try {

DB db = new DB();

try (Connection conn = db.getDbConnection()) {

String query = "SELECT id, name\_ganre FROM ganre";

try (PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

ObservableList<String> genres = FXCollections.observableArrayList();

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name\_ganre");

genres.add(name);

ganreMap.put(name, id);

}

GanreChoicBox.setItems(genres);

}

}

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося завантажити жанри", Alert.AlertType.ERROR);

}

}

private void addNewBook() throws SQLException {

// Перевірка заповненості полів

if (NameBookField.getText().isEmpty() ||

authorChoicBox.getValue() == null ||

GanreChoicBox.getValue() == null ||

writeYearField.getText().isEmpty() ||

ISBNField.getText().isEmpty() ||

QuantityField.getText().isEmpty()) {

showAlert("Попередження", "Будь ласка, заповніть всі поля", Alert.AlertType.WARNING);

return;

}

// Перевірка числових полів

try {

Integer.parseInt(writeYearField.getText());

Integer.parseInt(ISBNField.getText());

int quantity = Integer.parseInt(QuantityField.getText());

if (quantity <= 0) {

showAlert("Попередження", "Кількість має бути більше 0", Alert.AlertType.WARNING);

return;

}

} catch (NumberFormatException e) {

showAlert("Помилка", "Рік, ISBN та кількість мають бути числами", Alert.AlertType.ERROR);

return;

}

// Отримання ID автора та жанру

int authorId = authorMap.get(authorChoicBox.getValue());

int ganreId = ganreMap.get(GanreChoicBox.getValue());

DB db = new DB();

Connection conn = null;

try {

conn = db.getDbConnection();

conn.setAutoCommit(false); // Починаємо транзакцію

String query = "INSERT INTO books (name\_book, id\_author, id\_ganre, write\_year, isbn, quantity, id\_status) " +

"VALUES (?, ?, ?, ?, ?, ?, 1)"; // id\_status = 1 (Доступна)

try (PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, NameBookField.getText());

stmt.setInt(2, authorId);

stmt.setInt(3, ganreId);

stmt.setInt(4, Integer.parseInt(writeYearField.getText()));

stmt.setInt(5, Integer.parseInt(ISBNField.getText()));

stmt.setInt(6, Integer.parseInt(QuantityField.getText()));

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

conn.commit(); // Підтверджуємо транзакцію

showAlert("Успіх", "Книгу успішно додано", Alert.AlertType.INFORMATION);

clearFields();

} else {

conn.rollback(); // Відкатуємо транзакцію

showAlert("Помилка", "Не вдалося додати книгу", Alert.AlertType.ERROR);

}

}

} catch (SQLException e) {

if (conn != null) {

conn.rollback(); // Відкатуємо транзакцію при помилці

}

throw e;

} finally {

if (conn != null) {

try {

conn.setAutoCommit(true); // Повертаємо автоматичний режим

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

private void handleDatabaseError(SQLException e) {

if (e.getMessage().contains("database is locked")) {

showAlert("Помилка", "База даних зайнята. Спробуйте ще раз через декілька секунд.", Alert.AlertType.ERROR);

} else {

showAlert("Помилка", "Помилка бази даних: " + e.getMessage(), Alert.AlertType.ERROR);

}

}

private void clearFields() {

NameBookField.clear();

authorChoicBox.setValue(null);

GanreChoicBox.setValue(null);

writeYearField.clear();

ISBNField.clear();

QuantityField.clear();

}

private void showAlert(String title, String message, Alert.AlertType type) {

Alert alert = new Alert(type);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

}}

AddBookViews

package biblioty.armbiblioty;

import java.net.URL;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.HashMap;

import java.util.Map;

import java.util.ResourceBundle;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.ChoiceBox;

import javafx.scene.control.TextField;

public class AddBooksViews {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button AddBooks;

@FXML

private ChoiceBox<String> GanreChoicBox;

private Map<String, Integer> ganreMap = new HashMap<>();

@FXML

private TextField ISBNField;

@FXML

private TextField NameBookField;

@FXML

private TextField QuantityField;

@FXML

private ChoiceBox<String> authorChoicBox;

private Map<String, Integer> authorMap = new HashMap<>();

@FXML

private TextField writeYearField;

@FXML

void initialize() {

loadAuthors();

loadGenres();

AddBooks.setOnAction(event -> {

try {

addNewBook();

} catch (SQLException e) {

handleDatabaseError(e);

}

});

}

private void loadAuthors() {

try {

DB db = new DB();

try (Connection conn = db.getDbConnection()) {

String query = "SELECT id, name\_author FROM author";

try (PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

ObservableList<String> authors = FXCollections.observableArrayList();

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name\_author");

authors.add(name);

authorMap.put(name, id);

}

authorChoicBox.setItems(authors);

}

}

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося завантажити авторів", Alert.AlertType.ERROR);

}

}

private void loadGenres() {

try {

DB db = new DB();

try (Connection conn = db.getDbConnection()) {

String query = "SELECT id, name\_ganre FROM ganre";

try (PreparedStatement stmt = conn.prepareStatement(query);

ResultSet rs = stmt.executeQuery()) {

ObservableList<String> genres = FXCollections.observableArrayList();

while (rs.next()) {

int id = rs.getInt("id");

String name = rs.getString("name\_ganre");

genres.add(name);

ganreMap.put(name, id);

}

GanreChoicBox.setItems(genres);

}

}

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося завантажити жанри", Alert.AlertType.ERROR);

}

}

private void addNewBook() throws SQLException {

// Перевірка заповненості полів

if (NameBookField.getText().isEmpty() ||

authorChoicBox.getValue() == null ||

GanreChoicBox.getValue() == null ||

writeYearField.getText().isEmpty() ||

ISBNField.getText().isEmpty() ||

QuantityField.getText().isEmpty()) {

showAlert("Попередження", "Будь ласка, заповніть всі поля", Alert.AlertType.WARNING);

return;

}

// Перевірка числових полів

try {

Integer.parseInt(writeYearField.getText());

Integer.parseInt(ISBNField.getText());

int quantity = Integer.parseInt(QuantityField.getText());

if (quantity <= 0) {

showAlert("Попередження", "Кількість має бути більше 0", Alert.AlertType.WARNING);

return;

}

} catch (NumberFormatException e) {

showAlert("Помилка", "Рік, ISBN та кількість мають бути числами", Alert.AlertType.ERROR);

return;

}

// Отримання ID автора та жанру

int authorId = authorMap.get(authorChoicBox.getValue());

int ganreId = ganreMap.get(GanreChoicBox.getValue());

DB db = new DB();

Connection conn = null;

try {

conn = db.getDbConnection();

conn.setAutoCommit(false); // Починаємо транзакцію

String query = "INSERT INTO books (name\_book, id\_author, id\_ganre, write\_year, isbn, quantity, id\_status) " +

"VALUES (?, ?, ?, ?, ?, ?, 1)"; // id\_status = 1 (Доступна)

try (PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, NameBookField.getText());

stmt.setInt(2, authorId);

stmt.setInt(3, ganreId);

stmt.setInt(4, Integer.parseInt(writeYearField.getText()));

stmt.setInt(5, Integer.parseInt(ISBNField.getText()));

stmt.setInt(6, Integer.parseInt(QuantityField.getText()));

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

conn.commit(); // Підтверджуємо транзакцію

showAlert("Успіх", "Книгу успішно додано", Alert.AlertType.INFORMATION);

clearFields();

} else {

conn.rollback(); // Відкатуємо транзакцію

showAlert("Помилка", "Не вдалося додати книгу", Alert.AlertType.ERROR);

}

}

} catch (SQLException e) {

if (conn != null) {

conn.rollback(); // Відкатуємо транзакцію при помилці

}

throw e;

} finally {

if (conn != null) {

try {

conn.setAutoCommit(true); // Повертаємо автоматичний режим

conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

}

}

private void handleDatabaseError(SQLException e) {

if (e.getMessage().contains("database is locked")) {

showAlert("Помилка", "База даних зайнята. Спробуйте ще раз через декілька секунд.", Alert.AlertType.ERROR);

} else {

showAlert("Помилка", "Помилка бази даних: " + e.getMessage(), Alert.AlertType.ERROR);

}

}

private void clearFields() {

NameBookField.clear();

authorChoicBox.setValue(null);

GanreChoicBox.setValue(null);

writeYearField.clear();

ISBNField.clear();

QuantityField.clear();

}

private void showAlert(String title, String message, Alert.AlertType type) {

Alert alert = new Alert(type);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

}

**AdminViews**

package biblioty.armbiblioty;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.HBox;

import javafx.scene.layout.Priority;

import javafx.scene.layout.Region;

import javafx.scene.layout.VBox;

import javafx.stage.Modality;

import javafx.stage.Stage;

import java.io.IOException;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class AdminViews {

@FXML private Button genresBtn;

@FXML private Button authorsBtn;

@FXML private Button backBtn;

@FXML private VBox recordsVBox;

@FXML private Pagination pagination;

private static final int RECORDS\_PER\_PAGE = 8;

private List<Record> allRecords = new ArrayList<>();

private String currentTable = null;

public static class Record {

private final int id;

private final String name;

private final String tableName;

public Record(int id, String name, String tableName) {

this.id = id;

this.name = name;

this.tableName = tableName;

}

public int getId() { return id; }

public String getName() { return name; }

public String getTableName() { return tableName; }

}

@FXML

void initialize() {

genresBtn.setOnAction(event -> loadRecords("ganre"));

authorsBtn.setOnAction(event -> loadRecords("author"));

backBtn.setOnAction(event -> handleBack());

setupPagination();

}

private void handleBack() {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("MainViews.fxml"));

Parent root = loader.load();

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.show();

Stage currentStage = (Stage) backBtn.getScene().getWindow();

currentStage.close();

} catch (Exception e) {

showAlert("Помилка", "Не вдалося повернутися: " + e.getMessage(), Alert.AlertType.ERROR);

}

}

private void loadRecords(String tableName) {

currentTable = tableName;

allRecords.clear();

DB db = new DB();

try (Connection conn = db.getDbConnection();

PreparedStatement stmt = conn.prepareStatement(

"SELECT id, name\_" + (tableName.equals("author") ? "author" : "ganre") +

" AS name FROM " + tableName);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

allRecords.add(new Record(

rs.getInt("id"),

rs.getString("name"),

tableName

));

}

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося завантажити записи: " + e.getMessage(), Alert.AlertType.ERROR);

}

setupPagination();

}

private void setupPagination() {

if (allRecords.isEmpty()) {

pagination.setPageCount(1);

pagination.setDisable(true);

recordsVBox.getChildren().clear();

return;

}

int pageCount = (int) Math.ceil((double) allRecords.size() / RECORDS\_PER\_PAGE);

pagination.setPageCount(pageCount);

pagination.setDisable(pageCount <= 1);

pagination.setCurrentPageIndex(0);

pagination.currentPageIndexProperty().addListener((obs, oldIndex, newIndex) -> {

displayRecords(newIndex.intValue());

});

displayRecords(0);

}

private void displayRecords(int pageIndex) {

recordsVBox.getChildren().clear();

int startIndex = pageIndex \* RECORDS\_PER\_PAGE;

int endIndex = Math.min(startIndex + RECORDS\_PER\_PAGE, allRecords.size());

for (int i = startIndex; i < endIndex; i++) {

Record record = allRecords.get(i);

recordsVBox.getChildren().add(createRecordBox(record));

}

}

private HBox createRecordBox(Record record) {

HBox recordBox = new HBox(10);

recordBox.setStyle("-fx-border-color: black; -fx-border-width: 1; -fx-padding: 5;");

recordBox.setMaxWidth(Double.MAX\_VALUE);

Label nameLabel = new Label(record.getName());

nameLabel.setMaxWidth(Double.MAX\_VALUE);

HBox.setHgrow(nameLabel, Priority.ALWAYS);

// Спейсер для вирівнювання кнопок праворуч

Region spacer = new Region();

HBox.setHgrow(spacer, Priority.ALWAYS);

Button editBtn = new Button("Редагувати");

editBtn.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white;");

editBtn.setOnAction(event -> handleEdit(record));

Button deleteBtn = new Button("Видалити");

deleteBtn.setStyle("-fx-background-color: #f44336; -fx-text-fill: white;");

deleteBtn.setOnAction(event -> handleDelete(record));

recordBox.getChildren().addAll(nameLabel, spacer, editBtn, deleteBtn);

return recordBox;

}

private void handleEdit(Record record) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("EditAdminRecordViews.fxml"));

Parent root = loader.load();

EditAdminRecordViews controller = loader.getController();

controller.setRecordId(record.getId());

controller.setTableName(record.getTableName());

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadRecords(currentTable);

} catch (IOException e) {

showAlert("Помилка", "Не вдалося завантажити форму редагування: " + e.getMessage(),

Alert.AlertType.ERROR);

}

}

private void handleDelete(Record record) {

Alert confirmation = new Alert(Alert.AlertType.CONFIRMATION);

confirmation.setTitle("Підтвердження");

confirmation.setHeaderText(null);

confirmation.setContentText("Ви впевнені, що хочете видалити '" + record.getName() + "'?");

if (confirmation.showAndWait().orElse(ButtonType.CANCEL) != ButtonType.OK) {

return;

}

DB db = new DB();

try (Connection conn = db.getDbConnection()) {

conn.setAutoCommit(false);

// Видаляємо зв'язані записи з таблиці books

String deleteBooksQuery = "DELETE FROM books WHERE id\_" +

(record.getTableName().equals("author") ? "author" : "ganre") +

" = ?";

try (PreparedStatement stmt = conn.prepareStatement(deleteBooksQuery)) {

stmt.setInt(1, record.getId());

stmt.executeUpdate();

}

// Видаляємо сам запис

String deleteRecordQuery = "DELETE FROM " + record.getTableName() + " WHERE id = ?";

try (PreparedStatement stmt = conn.prepareStatement(deleteRecordQuery)) {

stmt.setInt(1, record.getId());

stmt.executeUpdate();

}

conn.commit();

showAlert("Успіх", "Запис та всі пов'язані книги успішно видалено", Alert.AlertType.INFORMATION);

loadRecords(currentTable);

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося видалити запис: " + e.getMessage(), Alert.AlertType.ERROR);

}

}

private void showAlert(String title, String message, Alert.AlertType type) {

Alert alert = new Alert(type);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

}

**Autentification**

package biblioty.armbiblioty;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

/\*\*

\* Клас-контролер для вікна автентифікації користувача.

\*/

public class Autentification {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button btnLogIn;

@FXML

private Button btnSgnIn;

@FXML

private TextField txtEmail;

@FXML

private PasswordField txtPassword;

/\*\*

\* Ініціалізує контролер після завантаження користувальницького інтерфейсу (FXML).

\*/

@FXML

void initialize() {

// Обробник події для кнопки входу

btnLogIn.setOnAction(event -> {

String loginText = txtEmail.getText().trim();

String loginPassword = txtPassword.getText().trim();

if (!loginText.equals("") && !loginPassword.equals(""))

loginUser(loginText, loginPassword);

else

System.out.println("login is empty ");

});

// Обробник події для кнопки реєстрації

btnSgnIn.setOnAction(event -> {

Stage stage = (Stage) btnSgnIn.getScene().getWindow();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("Registration.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

stage.setScene(new Scene(root));

});

}

/\*\*

\* Метод для автентифікації користувача.

\* @param loginText логін користувача

\* @param passwordTxt пароль користувача

\*/

public void loginUser(String loginText, String passwordTxt) {

Autorize autorize = new Autorize();

User userFromDatabase = autorize.getUser(loginText, passwordTxt);

if (userFromDatabase != null) {

Const.user = userFromDatabase;

System.out.println(Const.user.getIsAdmin());

Stage stage = (Stage) btnLogIn.getScene().getWindow();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("MainViews.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

stage.setScene(new Scene(root));

System.out.println("login");

}

}

}

**Autorize**

package biblioty.armbiblioty;

import javafx.scene.control.Alert;

import org.apache.commons.codec.digest.DigestUtils;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

/\*\*

\* Клас реєстрації та авторизації

\*/

public class Autorize {

DB db = new DB();

/\*\*

\* Реєстрація користувача

\* @param user

\*/

public void signUpUser(User user) {

String insert = "INSERT INTO " + Const.GTU\_TABLE + "(" +

Const.GTU\_USERNAME + "," + Const.GTU\_MAIL + "," +

Const.GTU\_PASSWORD + "," + "isAdmin" + ")" + "VALUES(?,?,?,?)";

try {

PreparedStatement prSt = db.getDbConnection().prepareStatement(insert);

prSt.setString(1, user.getNameUser());

prSt.setString(2, user.getEmail());

prSt.setString(3, DigestUtils.sha1Hex(user.getPassword()));

prSt.setInt(4, 0);

int rowsAffected = prSt.executeUpdate();

// Логування результату виконання

if (rowsAffected > 0) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setHeaderText("Успіх");

alert.setContentText("Дані було успішно додано");

alert.show();

System.out.println("User registered successfully!");

} else {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setHeaderText("Помилка");

alert.setContentText("Не вдалося додати користувача");

alert.show();

System.out.println("Error: User not registered.");

}

} catch (Exception e) {

e.printStackTrace();

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setHeaderText("Помилка");

alert.setContentText("Такий користувач уже існує");

alert.show();

System.out.println("Error during registration: " + e.getMessage());

}

}

/\*\*

\* Авторизація користувача

\* @param email

\* @param password

\* @return

\*/

public User getUser(String email, String password) {

User user = null;

String hashedPassword = DigestUtils.sha1Hex(password);

String sqlStatement = String.format("SELECT \* FROM %s WHERE %s = ? AND %s = ?", Const.GTU\_TABLE, Const.GTU\_MAIL, Const.GTU\_PASSWORD);

try {

PreparedStatement preparedStatement = db.getDbConnection().prepareStatement(sqlStatement);

preparedStatement.setString(1, email);

preparedStatement.setString(2, hashedPassword);

ResultSet resultSet = preparedStatement.executeQuery();

// Логування результату виконання

if (resultSet.next()) {

int userId = resultSet.getInt("id");

String userName = resultSet.getString("username");

String loginFromDatabase = resultSet.getString("email");

String passwordFromDatabase = resultSet.getString("password");

int isAdmin = resultSet.getInt("isAdmin");

user = new User(userId, userName, loginFromDatabase, passwordFromDatabase, isAdmin);

} else {

System.out.println("No user found with given email and password.");

}

} catch (SQLException e) {

e.printStackTrace();

System.out.println("SQL Error: " + e.getMessage());

}

return user;

}

}

**BookTheyTook**

package biblioty.armbiblioty;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.HBox;

import javafx.scene.layout.VBox;

import javafx.stage.Modality;

import javafx.stage.Stage;

import java.net.URL;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import java.util.ResourceBundle;

public class BookTheyTookViews {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button backBtn;

@FXML

private VBox recordsVBox;

@FXML

private Pagination pagination;

private static final int RECORDS\_PER\_PAGE = 8;

private List<Record> allRecords = new ArrayList<>();

public static class Record {

private final int id;

private final int bookId;

private final String clientName;

private final String bookName;

private final String dateTook;

public Record(int id, int bookId, String clientName, String bookName, String dateTook) {

this.id = id;

this.bookId = bookId;

this.clientName = clientName;

this.bookName = bookName;

this.dateTook = dateTook;

}

public int getId() { return id; }

public int getBookId() { return bookId; }

public String getClientName() { return clientName; }

public String getBookName() { return bookName; }

public String getDateTook() { return dateTook; }

}

@FXML

void initialize() {

backBtn.setOnAction(event -> handleBack());

loadRecords();

setupPagination();

}

private void handleBack() {

try {

// Завантажуємо вікно LibrariesViews

FXMLLoader loader = new FXMLLoader(getClass().getResource("MainViews.fxml"));

Parent root = loader.load();

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.show();

// Закриваємо поточне вікно

Stage currentStage = (Stage) backBtn.getScene().getWindow();

currentStage.close();

} catch (Exception e) {

showAlert("Помилка", "Не вдалося повернутися: " + e.getMessage(), Alert.AlertType.ERROR);

}

}

private void loadRecords() {

allRecords.clear();

DB db = new DB();

Connection conn = null;

PreparedStatement stmt = null;

ResultSet rs = null;

try {

conn = db.getDbConnection();

String query = "SELECT bt.id, bt.id\_book, bt.name\_clients, bt.date\_the\_took, b.name\_book " +

"FROM booktheytook bt " +

"JOIN books b ON bt.id\_book = b.id\_book";

stmt = conn.prepareStatement(query);

rs = stmt.executeQuery();

while (rs.next()) {

allRecords.add(new Record(

rs.getInt("id"),

rs.getInt("id\_book"),

rs.getString("name\_clients"),

rs.getString("name\_book"),

rs.getString("date\_the\_took")

));

}

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося завантажити записи: " + e.getMessage(), Alert.AlertType.ERROR);

e.printStackTrace();

} finally {

try {

if (rs != null) rs.close();

if (stmt != null) stmt.close();

// З'єднання не закриваємо, як потрібно

} catch (SQLException ex) {

ex.printStackTrace();

}

}

}

private void setupPagination() {

if (allRecords.isEmpty()) {

pagination.setPageCount(1);

pagination.setDisable(true);

recordsVBox.getChildren().clear();

return;

}

int pageCount = (int) Math.ceil((double) allRecords.size() / RECORDS\_PER\_PAGE);

pagination.setPageCount(pageCount);

pagination.setDisable(pageCount <= 1);

pagination.setCurrentPageIndex(0);

pagination.currentPageIndexProperty().addListener((obs, oldIndex, newIndex) -> {

displayRecords(newIndex.intValue());

});

displayRecords(0);

}

private void displayRecords(int pageIndex) {

recordsVBox.getChildren().clear();

int startIndex = pageIndex \* RECORDS\_PER\_PAGE;

int endIndex = Math.min(startIndex + RECORDS\_PER\_PAGE, allRecords.size());

for (int i = startIndex; i < endIndex; i++) {

Record record = allRecords.get(i);

HBox recordBox = createRecordBox(record);

recordsVBox.getChildren().add(recordBox);

}

}

private HBox createRecordBox(Record record) {

HBox recordBox = new HBox(10);

recordBox.setStyle("-fx-border-color: black; -fx-border-width: 1; -fx-padding: 5;");

Label clientLabel = new Label("Клієнт: " + record.getClientName());

Label bookLabel = new Label("Книга: " + record.getBookName());

Label dateLabel = new Label("Дата видачі: " + record.getDateTook());

Button editBtn = new Button("Редагувати");

editBtn.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white;");

editBtn.setOnAction(event -> handleEdit(record));

Button returnBtn = new Button("Повернув");

returnBtn.setStyle("-fx-background-color: #f44336; -fx-text-fill: white;");

returnBtn.setOnAction(event -> handleReturn(record));

recordBox.getChildren().addAll(clientLabel, bookLabel, dateLabel, editBtn, returnBtn);

return recordBox;

}

private void handleEdit(Record record) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("EditBookTheyTookViews.fxml"));

Parent root = loader.load();

EditBookTheyTookViews controller = loader.getController();

controller.setRecordId(record.getId());

controller.setBookId(record.getBookId());

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadRecords();

setupPagination();

} catch (Exception e) {

showAlert("Помилка", "Не вдалося відкрити форму редагування: " + e.getMessage(), Alert.AlertType.ERROR);

}

}

private void handleReturn(Record record) {

Alert confirmation = new Alert(Alert.AlertType.CONFIRMATION);

confirmation.setTitle("Підтвердження");

confirmation.setHeaderText(null);

confirmation.setContentText("Ви впевнені, що книга повернута клієнтом '" + record.getClientName() + "'?");

if (confirmation.showAndWait().orElse(ButtonType.CANCEL) != ButtonType.OK) {

return;

}

DB db = new DB();

Connection conn = null;

try {

conn = db.getDbConnection();

String query = "DELETE FROM booktheytook WHERE id = ?";

try (PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, record.getId());

stmt.executeUpdate();

}

showAlert("Успіх", "Запис успішно видалено", Alert.AlertType.INFORMATION);

loadRecords();

setupPagination();

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося видалити запис: " + e.getMessage(), Alert.AlertType.ERROR);

}

}

private void showAlert(String title, String message, Alert.AlertType type) {

Alert alert = new Alert(type);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

}

**BookTookViews**

package biblioty.armbiblioty;

import java.net.URL;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.util.ResourceBundle;

import javafx.fxml.FXML;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

public class BookTookViews {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button btnRequest;

@FXML

private TextField txtPIB; // Змінено з PasswordField на TextField для зручності введення

private int bookId;

public void setBookId(int bookId) {

this.bookId = bookId;

}

@FXML

void initialize() {

btnRequest.setOnAction(event -> {

try {

issueBook();

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося видати книгу: " + e.getMessage(), Alert.AlertType.ERROR);

}

});

}

private void issueBook() throws SQLException {

// Перевірка заповненості поля ПІБ

if (txtPIB.getText() == null || txtPIB.getText().trim().isEmpty()) {

showAlert("Попередження", "Будь ласка, введіть ПІБ клієнта", Alert.AlertType.WARNING);

return;

}

DB db = new DB();

Connection conn = db.getDbConnection(); // Використовуємо існуюче з'єднання

// Отримуємо поточну дату у форматі "день-місяць-рік"

String currentDate = LocalDate.now().format(DateTimeFormatter.ofPattern("dd-MM-yyyy"));

// Додаємо запис про видачу

String query = "INSERT INTO booktheytook (id\_book, name\_clients, date\_the\_took) VALUES (?, ?, ?)";

try (PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, bookId);

stmt.setString(2, txtPIB.getText().trim());

stmt.setString(3, currentDate);

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

showAlert("Успіх", "Книгу успішно видано клієнту: " + txtPIB.getText(), Alert.AlertType.INFORMATION);

closeWindow();

}

} catch (SQLException e) {

showAlert("Помилка", "Не вдалося видати книгу: " + e.getMessage(), Alert.AlertType.ERROR);

throw e; // Перезкидання виключення для обробки вище

}

}

private void closeWindow() {

Stage stage = (Stage) btnRequest.getScene().getWindow();

stage.close();

}

private void showAlert(String title, String message, Alert.AlertType type) {

Alert alert = new Alert(type);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

}

**Const**

package biblioty.armbiblioty;

/\*\*

\* Клас, що містить константи для роботи з базою даних і об'єкт користувача.

\*/

public class Const {

/\*\* Назва таблиці користувачів \*/

public static final String GTU\_TABLE = "users";

/\*\* Назва стовпця з ідентифікатором користувача \*/

public static final String GTU\_ID = "id";

/\*\* Назва стовпця з ім'ям користувача \*/

public static final String GTU\_USERNAME = "username";

/\*\* Назва стовпця з паролем користувача \*/

public static final String GTU\_PASSWORD = "`password`";

/\*\* Назва стовпця з електронною поштою користувача \*/

public static final String GTU\_MAIL = "`email`";

/\*\* Об'єкт користувача, що зберігає дані авторизованого користувача \*/

public static User user = null;

}

**DB**

package biblioty.armbiblioty;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.nio.file.Paths;

import java.nio.file.Path;

/\*\*

\* Підключення до SQLite бази даних

\*/

public class DB {

private static Connection dbConnection; // Статичне з'єднання для одноразового використання

static {

try {

// Ініціалізація з'єднання при завантаженні класу

initializeConnection();

} catch (SQLException e) {

System.err.println("Помилка ініціалізації з'єднання: " + e.getMessage());

e.printStackTrace();

}

}

private static void initializeConnection() throws SQLException {

if (dbConnection == null || dbConnection.isClosed()) {

// Шлях до бази даних у папці resources/DB

Path dbPath = Paths.get("src", "main", "resources", "DB", "biblioteka.db");

String url = "jdbc:sqlite:" + dbPath.toString() + "?busy\_timeout=5000";

dbConnection = DriverManager.getConnection(url);

System.out.println("Підключення до SQLite встановлено: " + dbConnection);

// Увімкнення підтримки foreign keys

dbConnection.createStatement().execute("PRAGMA foreign\_keys = ON");

}

}

/\*\*

\* Повертає з'єднання з базою даних (завжди одне і те ж)

\* @return об'єкт Connection

\* @throws SQLException

\*/

public Connection getDbConnection() throws SQLException {

if (dbConnection == null || dbConnection.isClosed()) {

initializeConnection(); // Переініціалізація, якщо з'єднання втрачено

}

return dbConnection;

}

// Метод closeConnection видалено, оскільки з'єднання не закриватиметься

}

**MainViews**

package biblioty.armbiblioty;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.stage.Stage;

public class MainViews {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button AdminBtn;

@FXML

private Button BookTheyTookBtn;

@FXML

private Button LibrariesBtn;

@FXML

void initialize() {

if(Const.user.getIsAdmin()!=1){

AdminBtn.setVisible(false);

}

AdminBtn.setOnAction(event -> {

openWindow("AdminViews.fxml");

});

LibrariesBtn.setOnAction(event -> {

openWindow("LibrariesViews.fxml");

});

BookTheyTookBtn.setOnAction(event -> {

openWindow("BookTheyTookViews.fxml");

});

}

/\*\*

\* Метод для відкриття нового вікна.

\*

\* @param fxmlFile ім'я файлу FXML для завантаження

\*/

private void openWindow(String fxmlFile) {

Stage stage = (Stage) AdminBtn.getScene().getWindow();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource(fxmlFile));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

stage.setScene(new Scene(root));

}

}

**Registration**

package biblioty.armbiblioty;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.stage.Stage;

import java.io.IOException;

import java.net.URL;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.ResourceBundle;

/\*\*

\* Класс реєстрації

\*/

public class Registration {

@FXML

private ResourceBundle resources;

@FXML

private URL location;

@FXML

private Button btnLogIn;

@FXML

private Button btnSignIn;

@FXML

private TextField txtEmail;

@FXML

private PasswordField txtPassword;

@FXML

private PasswordField txtRepitPassword;

@FXML

private TextField txtUserName;

/\*\*

\* ініціалізація класу

\*/

@FXML

void initialize() {

btnSignIn.setOnAction(event -> {

signUpNewUser();

});

btnLogIn.setOnAction(event-> {

Stage stage = (Stage) btnLogIn.getScene().getWindow();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("Autentification.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

stage.setScene(new Scene(root));

});

}

/\*\*

\* Реэстрація користувача

\*/

private void signUpNewUser () {

Autorize autorize= new Autorize();

String userNameReg = txtUserName.getText();

String passwordReg = txtPassword.getText();

String emailReg = txtEmail.getText();

String returnPass = txtRepitPassword.getText();

if(txtPassword.getText().length()<8){

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setHeaderText("Помилка");

alert.setContentText("Пароль надто короткий");

alert.show();

return;

}

if (!txtPassword.getText().equals(txtRepitPassword.getText())) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setHeaderText("Помилка");

alert.setContentText("Паролі не співпадають");

alert.show();

} else {

ArrayList<String> arrList = new ArrayList<String>();

arrList.addAll(Arrays.asList(userNameReg, passwordReg, emailReg,returnPass));

if (arrList.stream().anyMatch(text -> text.equals(""))) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setHeaderText("Помилка");

alert.setContentText("Заповніть всі поля");

alert.show();

return;

}

User user = new User(userNameReg, emailReg, passwordReg);

autorize.signUpUser(user);

txtUserName.setText("");

txtRepitPassword.setText("");

txtPassword.setText("");

txtEmail.setText("");

Stage stage = (Stage) btnSignIn.getScene().getWindow();

FXMLLoader loader = new FXMLLoader();

loader.setLocation(getClass().getResource("Autentification.fxml"));

try {

loader.load();

} catch (IOException e) {

e.printStackTrace();

}

Parent root = loader.getRoot();

stage.setScene(new Scene(root));

}

}

}

**Run**

package biblioty.armbiblioty;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.stage.Stage;

import java.io.IOException;

public class Run extends Application {

@Override

public void start(Stage stage) throws IOException {

FXMLLoader fxmlLoader = new FXMLLoader(Run.class.getResource("Autentification.fxml"));

Scene scene = new Scene(fxmlLoader.load());

stage.setScene(scene);

stage.show();

}

public static void main(String[] args) {

launch();

}

}

**User**

package biblioty.armbiblioty;

/\*\*

\* Клас з юзерами

\*/

public class User {

private int isAdmin;

private int userId;

private String nameUser;

private String password;

private String email;

/\*\*

\* Конструктор юзер для реєстрації

\*

\* @param userId

\* @param nameUser

\* @param email

\* @param password

\* @param isAdmin

\*/

public User(int userId, String nameUser, String email, String password, int isAdmin) {

this.userId = userId;

this.nameUser = nameUser;

this.password = password;

this.email = email;

this.isAdmin = isAdmin;

}

/\*\*

\* Конструктор юзер для авторизації

\* @param nameUser

\* @param email

\* @param password

\*/

public User(String nameUser, String email, String password) {

this.nameUser = nameUser;

this.password = password;

this.email = email;

}

public int getIsAdmin() {

return isAdmin;

}

public void setIsAdmin(int isAdmin) {

this.isAdmin = isAdmin;

}

public User() {}

public String getNameUser() {

return nameUser;

}

public int getUserId() {

return userId;

}

public void setNameUser(String nameUser) {

this.nameUser = nameUser;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

**LibrariesViews**

package biblioty.armbiblioty;

import java.net.URL;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ResourceBundle;

import java.util.concurrent.Executors;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.collections.transformation.FilteredList;

import javafx.collections.transformation.SortedList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.scene.layout.HBox;

import javafx.stage.Modality;

import javafx.stage.Stage;

import javafx.util.Callback;

public class LibrariesViews {

@FXML private ResourceBundle resources;

@FXML private URL location;

@FXML private TableColumn<Book, Integer> ISBNColumn;

@FXML private TableView<Book> booksTable;

@FXML private TableColumn<Book, String> AutorBook;

@FXML private TableColumn<Book, String> GanreColumn;

@FXML private TableColumn<Book, String> NameBook;

@FXML private TableColumn<Book, Integer> QuantityColumn;

@FXML private TextField SearchField;

@FXML private Button AddBooks;

@FXML private Button BackBtn;

@FXML private Button AddAuthor;

@FXML private Button AddGanre;

@FXML private TableColumn<Book, String> StatusColumn;

@FXML private TableColumn<Book, Integer> yearWrite;

@FXML private TableColumn<Book, Void> ActionColumns;

private ObservableList<Book> booksData = FXCollections.observableArrayList();

@FXML

void initialize() {

AddBooks.setOnAction(event -> handleAddBook());

BackBtn.setOnAction(event -> handleBack());

AddAuthor.setOnAction(event -> handleAddAuthor());

AddGanre.setOnAction(event -> handleAddGanre());

try {

// Налаштування колонок

configureTableColumns();

// Завантаження даних

loadBooksData();

// Налаштування пошуку

setupSearch();

// Підсвічування рядків

setupRowHighlighting();

} catch (Exception e) {

showErrorAlert("Помилка ініціалізації", e.getMessage());

e.printStackTrace();

}

}

private void configureTableColumns() {

NameBook.setCellValueFactory(new PropertyValueFactory<>("name"));

AutorBook.setCellValueFactory(new PropertyValueFactory<>("author"));

GanreColumn.setCellValueFactory(new PropertyValueFactory<>("genre"));

yearWrite.setCellValueFactory(new PropertyValueFactory<>("year"));

QuantityColumn.setCellValueFactory(new PropertyValueFactory<>("availableQuantity"));

StatusColumn.setCellValueFactory(new PropertyValueFactory<>("status"));

ISBNColumn.setCellValueFactory(new PropertyValueFactory<>("isbn"));

ActionColumns.setCellFactory(createActionCellFactory());

}

private void setupRowHighlighting() {

booksTable.setRowFactory(tv -> new TableRow<Book>() {

@Override

protected void updateItem(Book book, boolean empty) {

super.updateItem(book, empty);

if (book == null || empty) {

setStyle("");

} else {

setStyle(book.getAvailableQuantity() <= 0 ? "-fx-background-color: #ffcccc;" : "");

}

}

});

}

private void setupSearch() {

FilteredList<Book> filteredData = new FilteredList<>(booksData, p -> true);

SearchField.textProperty().addListener((observable, oldValue, newValue) -> {

filteredData.setPredicate(book -> {

if (newValue == null || newValue.isEmpty()) return true;

String lowerCaseFilter = newValue.toLowerCase();

return book.getName().toLowerCase().contains(lowerCaseFilter) ||

book.getAuthor().toLowerCase().contains(lowerCaseFilter) ||

book.getGenre().toLowerCase().contains(lowerCaseFilter) ||

String.valueOf(book.getYear()).contains(lowerCaseFilter) ||

String.valueOf(book.getIsbn()).contains(lowerCaseFilter);

});

});

SortedList<Book> sortedData = new SortedList<>(filteredData);

sortedData.comparatorProperty().bind(booksTable.comparatorProperty());

booksTable.setItems(sortedData);

}

private synchronized void loadBooksData() {

booksData.clear();

DB db = new DB();

Connection conn = null;

PreparedStatement stmt = null;

ResultSet rs = null;

try {

conn = db.getDbConnection();

conn.setNetworkTimeout(Executors.newSingleThreadExecutor(), 5000);

String query = "SELECT b.id\_book, b.name\_book, a.name\_author, g.name\_ganre, " +

"b.write\_year, b.quantity, s.status, b.isbn, " +

"(SELECT COUNT(\*) FROM booktheytook bt WHERE bt.id\_book = b.id\_book) AS issued\_count " +

"FROM books b " +

"JOIN author a ON b.id\_author = a.id " +

"JOIN ganre g ON b.id\_ganre = g.id " +

"JOIN status s ON b.id\_status = s.id";

stmt = conn.prepareStatement(query);

rs = stmt.executeQuery();

while (rs.next()) {

int totalQuantity = rs.getInt("quantity");

int issuedCount = rs.getInt("issued\_count");

int availableQuantity = totalQuantity - issuedCount;

booksData.add(new Book(

rs.getInt("id\_book"),

rs.getString("name\_book"),

rs.getString("name\_author"),

rs.getString("name\_ganre"),

rs.getInt("write\_year"),

totalQuantity,

availableQuantity,

rs.getString("status"),

rs.getInt("isbn")

));

}

} catch (SQLException e) {

showErrorAlert("Помилка завантаження даних", e.getMessage());

e.printStackTrace();

} finally {

try {

if (rs != null) rs.close();

if (stmt != null) stmt.close();

// З'єднання не закриваємо, оскільки воно завжди відкрите

} catch (SQLException ex) {

ex.printStackTrace();

}

}

}

private Callback<TableColumn<Book, Void>, TableCell<Book, Void>> createActionCellFactory() {

return param -> new TableCell<>() {

private final Button editBtn = new Button("Редагувати");

private final Button deleteBtn = new Button("Видалити");

private final Button issueBtn = new Button("Видати");

private final HBox buttons = new HBox(5, editBtn, deleteBtn, issueBtn);

{

editBtn.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white;");

deleteBtn.setStyle("-fx-background-color: #f44336; -fx-text-fill: white;");

issueBtn.setStyle("-fx-background-color: #2196F3; -fx-text-fill: white;");

editBtn.setOnAction(event -> editBook(getCurrentBook()));

deleteBtn.setOnAction(event -> deleteBook(getCurrentBook()));

issueBtn.setOnAction(event -> handleIssueBook(getCurrentBook()));

}

private Book getCurrentBook() {

return getTableView().getItems().get(getIndex());

}

@Override

protected void updateItem(Void item, boolean empty) {

super.updateItem(item, empty);

if (empty) {

setGraphic(null);

} else {

Book book = getCurrentBook();

issueBtn.setDisable(book.getAvailableQuantity() <= 0);

setGraphic(buttons);

}

}

};

}

private void handleIssueBook(Book book) {

if (book.getAvailableQuantity() <= 0) {

showWarningAlert("Попередження", "Книга відсутня в бібліотеці!");

return;

}

try {

issueBookWithRetry(book, 3);

loadBooksData();

} catch (Exception e) {

showErrorAlert("Помилка видачі книги", e.getMessage());

}

}

private void issueBookWithRetry(Book book, int maxRetries) throws Exception {

int attempts = 0;

Exception lastException = null;

while (attempts < maxRetries) {

try {

issueBook(book);

return;

} catch (SQLException e) {

lastException = e;

attempts++;

if (attempts >= maxRetries) break;

try {

Thread.sleep(100 \* attempts);

} catch (InterruptedException ie) {

Thread.currentThread().interrupt();

throw new RuntimeException(ie);

}

}

}

throw lastException != null ? lastException : new Exception("Невідома помилка");

}

private void issueBook(Book book) throws SQLException {

FXMLLoader loader = new FXMLLoader(getClass().getResource("BookTookViews.fxml"));

try {

Parent root = loader.load();

BookTookViews controller = loader.getController();

controller.setBookId(book.getId());

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

} catch (Exception e) {

throw new SQLException("Помилка при відкритті форми видачі: " + e.getMessage());

}

}

private void editBook(Book book) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("EditBooksViews.fxml"));

Parent root = loader.load();

EditBooksViews controller = loader.getController();

controller.setBookId(book.getId());

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadBooksData();

} catch (Exception e) {

showErrorAlert("Помилка редагування", e.getMessage());

}

}

private void deleteBook(Book book) {

Alert confirmation = new Alert(Alert.AlertType.CONFIRMATION);

confirmation.setTitle("Підтвердження");

confirmation.setHeaderText(null);

confirmation.setContentText("Ви впевнені, що хочете видалити книгу '" + book.getName() + "'?");

if (confirmation.showAndWait().orElse(ButtonType.CANCEL) != ButtonType.OK) {

return;

}

DB db = new DB();

Connection conn = null;

try {

conn = db.getDbConnection();

conn.setAutoCommit(false);

conn.setNetworkTimeout(Executors.newSingleThreadExecutor(), 5000);

String deleteIssuedQuery = "DELETE FROM booktheytook WHERE id\_book = ?";

try (PreparedStatement stmt = conn.prepareStatement(deleteIssuedQuery)) {

stmt.setInt(1, book.getId());

stmt.executeUpdate();

}

String deleteBookQuery = "DELETE FROM books WHERE id\_book = ?";

try (PreparedStatement stmt = conn.prepareStatement(deleteBookQuery)) {

stmt.setInt(1, book.getId());

stmt.executeUpdate();

}

conn.commit();

loadBooksData();

showInfoAlert("Успіх", "Книгу успішно видалено");

} catch (SQLException e) {

try {

if (conn != null) conn.rollback();

} catch (SQLException ex) {

ex.printStackTrace();

}

showErrorAlert("Помилка видалення", "Не вдалося видалити книгу: " + e.getMessage());

} finally {

try {

if (conn != null) {

conn.setAutoCommit(true);

// З'єднання не закриваємо

}

} catch (SQLException ex) {

ex.printStackTrace();

}

}

}

@FXML

private void handleAddBook() {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("AddBooksViews.fxml"));

Parent root = loader.load();

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadBooksData();

} catch (Exception e) {

showErrorAlert("Помилка додавання книги", e.getMessage());

}

}

@FXML

private void handleBack() {

try {

// Завантажуємо головне вікно (MainViews)

FXMLLoader loader = new FXMLLoader(getClass().getResource("MainViews.fxml"));

Parent root = loader.load();

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.show();

// Закриваємо поточне вікно

Stage currentStage = (Stage) BackBtn.getScene().getWindow();

currentStage.close();

} catch (Exception e) {

showErrorAlert("Помилка повернення", e.getMessage());

}

}

@FXML

private void handleAddAuthor() {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("AddAuthorAndGanre.fxml"));

Parent root = loader.load();

AddAuthorAndGanre controller = loader.getController();

controller.setMode("author"); // Встановлюємо режим для автора

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadBooksData(); // Оновлюємо дані після додавання

} catch (Exception e) {

showErrorAlert("Помилка додавання автора", e.getMessage());

}

}

@FXML

private void handleAddGanre() {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource("AddAuthorAndGanre.fxml"));

Parent root = loader.load();

AddAuthorAndGanre controller = loader.getController();

controller.setMode("ganre"); // Встановлюємо режим для жанру

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadBooksData(); // Оновлюємо дані після додавання

} catch (Exception e) {

showErrorAlert("Помилка додавання жанру", e.getMessage());

}

}

private void showErrorAlert(String title, String message) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

private void showInfoAlert(String title, String message) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

private void showWarningAlert(String title, String message) {

Alert alert = new Alert(Alert.AlertType.WARNING);

alert.setTitle(title);

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();

}

public static class Book {

private final int id;

private final String name;

private final String author;

private final String genre;

private final int year;

private final int totalQuantity;

private final int availableQuantity;

private final String status;

private final int isbn;

public Book(int id, String name, String author, String genre, int year,

int totalQuantity, int availableQuantity, String status, int isbn) {

this.id = id;

this.name = name;

this.author = author;

this.genre = genre;

this.year = year;

this.totalQuantity = totalQuantity;

this.availableQuantity = availableQuantity;

this.status = status;

this.isbn = isbn;

}

// Гетери

public int getId() { return id; }

public String getName() { return name; }

public String getAuthor() { return author; }

public String getGenre() { return genre; }

public int getYear() { return year; }

public int getTotalQuantity() { return totalQuantity; }

public int getAvailableQuantity() { return availableQuantity; }

public String getStatus() { return status; }

public int getIsbn() { return isbn; }

}

}