МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ПРИРОДНИЧО-ГУМАНІТАРНИЙ ФАХОВИЙ КОЛЕДЖ

ДВНЗ «Ужгородський національний університет»

Реєстраційний №

Дата

Король Максим Іванович

студент 3-го курсу

денної форми навчання

ІНП № 253087ФМБД

**З В І Т**

ПРОХОДЖЕННЯ ВИРОБНИЧО-ТЕХНОЛОГІЧНОЇ ПРАКТИКИ

Галузь знань 12 «Інформаційні технології»,

Спеціальність 121

«Інженерія програмного забезпечення»

Розробка системи обліку медичних товарів

(індивідуальне завдання)

|  |  |
| --- | --- |
| Допущено до захисту | Науковий керівник: |
| Дата «  »   2025 р.    (підпис) | Викл. Ходанич П.Я. |

**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 запасами медичних товарів у закладах охорони здоров’я в умовах зростання попиту на медичні послуги та ресурси. У сучасному світі, з поширенням цифрових технологій, медичні установи стикаються з викликами швидкого пошуку, обліку та організації товарів, таких як ліки, медичне обладнання та витратні матеріали. Відсутність централізованої системи обліку може призводити до труднощів у відстеженні наявності товарів, контролю термінів придатності та управління поставками. Система обліку медичних товарів, розроблена з використанням Java, JavaFX та MySQL, покликана вирішити ці проблеми, забезпечуючи прозорість, зручність та оптимізацію процесів управління медичними ресурсами.

Метою проекту є створення зручного та надійного інструменту для працівників медичних закладів, який дозволить вести детальний облік медичних товарів, їх характеристик, статусів наявності та історії руху. Така система сприятиме підвищенню ефективності управління запасами, спрощенню доступу до інформації про товари та оптимізації роботи медичних установ загалом.

Предметом дослідження є аналіз потреб медичних закладів у систематизації обліку товарів, а також вивчення проблем, пов’язаних із відсутністю автоматизованого підходу до управління запасами. Будуть розглянуті сучасні практики та методології обліку, щоб інтегрувати найкращі з них у розроблювану систему.

Об’єктом дослідження є система обліку медичних товарів, створена за допомогою Java, JavaFX та SQLite. Ця система призначена стати ключовим елементом інфраструктури медичних закладів, забезпечуючи високу точність обліку, легкий доступ до актуальної інформації про товари 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**

"IBS Міні-Аптека" — це економічна програма для автоматизації обліку та управління в аптеках України, розроблена компанією IBS (ТОВ "Інновація-Н"). Вона призначена для малих аптек і невеликих аптечних мереж, забезпечуючи швидке впровадження з мінімальними витратами. Програма автоматизує ключові процеси: облік медикаментів за серіями, контроль термінів придатності, замовлення постачальникам без використання платформ ЛІКІС і МОРІОН, продаж за штрих-кодами, націнку товарів із контролем регульованих цін, інвентаризацію та формування аналітичних звітів. Система підтримує роботу з сучасним обладнанням, зокрема фіскальними реєстраторами та сканерами штрих-кодів, а також дозволяє віддалений доступ до бази даних через інтернет. "IBS Міні-Аптека" включає робочі місця фармацевта-касира та оператора, забезпечує підтримку програми "Доступні ліки" та пропонує безкоштовне встановлення, налаштування й навчання персоналу. Вартість стартує від 4950 грн, із гнучкими моделями оплати (оренда або купівля ліцензії).

Програма автоматизує облік медикаментів, продажів і замовлень, що спрощує управління аптекою та зменшує ручну роботу. Облік за серіями з контролем термінів придатності допомагає уникнути прострочення. Система є економічно вигідною (від 4950 грн) і підходить для малих аптек. Вона працює без підключення до ЛІКІС і МОРІОН, спрощуючи замовлення товарів. Підтримка фіскальних реєстраторів і сканерів забезпечує сумісність із сучасним обладнанням. Віддалений доступ до бази даних дозволяє управляти аптекою з будь-якого місця. Програма підтримує державну програму "Доступні ліки", полегшуючи облік відшкодувань. Аналітичні звіти допомагають аналізувати ефективність бізнесу. Безкоштовне встановлення, налаштування та навчання зменшують початкові витрати. Гнучка модель оплати (оренда або купівля) адаптується до потреб користувача.

"IBS Міні-Аптека" орієнтована на малі аптеки, тому її функціонал може бути недостатнім для великих мереж із складними процесами. У порівнянні з комплексним рішенням "IBS Аптека", програма має обмежені можливості, наприклад, відсутність спеціалізованих модулів для великих мереж. Для роботи потрібне додаткове обладнання (комп’ютер, сканер, фіскальний реєстратор), що збільшує початкові витрати. Відсутність інтеграції з платформами ЛІКІС і МОРІОН може ускладнити роботу для аптек, які використовують ці системи. Технічна підтримка після початкового періоду може вимагати додаткової оплати, що впливає на довгострокові витрати [8].

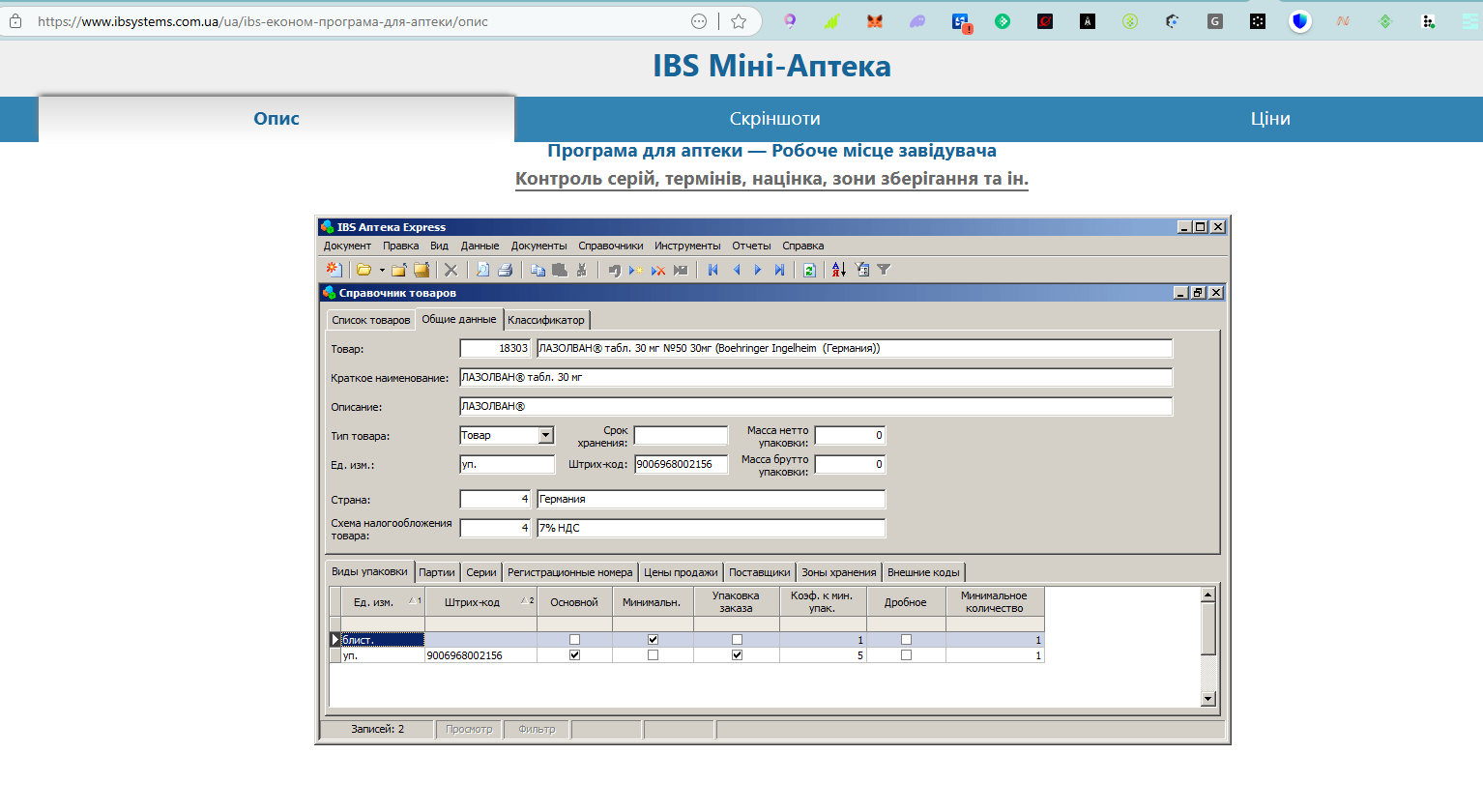


Рисунок 1.1 Аналог проєкту IBS Міні-Аптека

## Опис предметної області та постанова задачі

Розглядається можливість розробки системи обліку медичних товарів 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 управління запасами медичних товарів, включаючи облік медикаментів, медичного обладнання, витратних матеріалів, контроль термінів придатності, а також відстеження поставок і продажів.

Основні функції системи включатимуть реалізацію механізмів CRUD (створення, читання, оновлення, видалення) для записів про кожен медичний товар, його характеристики, кількість, терміни придатності, постачальників та історію руху. Важливим аспектом системи стане можливість реєстрації та аутентифікації користувачів, включаючи фармацевтів, адміністраторів аптек і менеджерів зі складського обліку, що дозволить забезпечити диференційований доступ до даних залежно від ролі та завдань.

Проект має на меті підвищення ефективності управління запасами медичних товарів, оптимізацію процесів закупівлі та реалізації, а також зниження ризиків втрат через прострочення чи помилки в обліку. Ключовою задачею стане забезпечення безпеки зберігання та обробки даних, зокрема інформації про постачальників і товари, 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 медичні товари. Важливим є застосування передових технологій для створення клієнтської частини системи, а також ретельне тестування юзабіліті для забезпечення високої якості користувацького досвіду.

Таким чином, розробка системи обліку медичних товарів має стати ключовим інструментом для ефективного управління запасами, підвищення якості обслуговування в медичних закладах і аптеках, а також сприяння прийняттю обґрунтованих управлінських рішень.

## Обґрунтування вибору методу розв’язку задачі

Розглядається можливість розробки системи обліку медичних товарів для медичних закладів або аптек з використанням Java для бекенд-розробки, JavaFX для створення графічного інтерфейсу користувача, а також SQLite для бази даних. Головною метою системи є автоматизація процесів управління запасами медичних товарів, включаючи облік медикаментів, медичного обладнання, витратних матеріалів, контроль їх наявності, руху, термінів придатності та відгуків користувачів.

Механізми CRUD: Реалізація створення, читання, оновлення та видалення записів про кожен медичний товар, його назву, категорію, виробника, кількість, терміни придатності та історію руху.

Реєстрація та аутентифікація: Система підтримує доступ для фармацевтів, адміністраторів і менеджерів зі складського обліку з диференційованим доступом до даних залежно від ролі.

Облік і контроль: Автоматизація управління запасами, включаючи контроль термінів придатності та відстеження руху товарів.

Проект спрямований на підвищення ефективності управління запасами медичних товарів, спрощення пошуку та обліку товарів, оптимізацію процесів закупівлі й реалізації, а також зниження ризиків втрат через помилки чи прострочення. Ключовою задачею є забезпечення безпеки даних, зокрема інформації про товари та постачальників, за допомогою сучасних механізмів шифрування та захисту бази даних SQLite.

Система матиме зручний і інтуїтивно зрозумілий графічний інтерфейс, розроблений за допомогою JavaFX, що дозволить користувачам легко навігувати та управляти інформацією про медичні товари. Застосування сучасних технологій та ретельне тестування юзабіліті забезпечать високу якість користувацького досвіду.

Розробка системи обліку медичних товарів стане ключовим інструментом для ефективного управління запасами, підвищення якості обслуговування в медичних закладах і аптеках, а також сприяння обґрунтованим рішенням у процесі організації роботи з медичними ресурсами.

# **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 наявни медичних товарів.
* 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 медичних товарів;
* «Product» — перелік всіх медичних товарів які є в наявності;
* «Orders» — перелік всіх замовлень;
* «category» — допоміжна таблиця, в встановлює категорію товару;
* «Clients» — допоміжна таблиця, яка містить в собі всі дані клієнта;

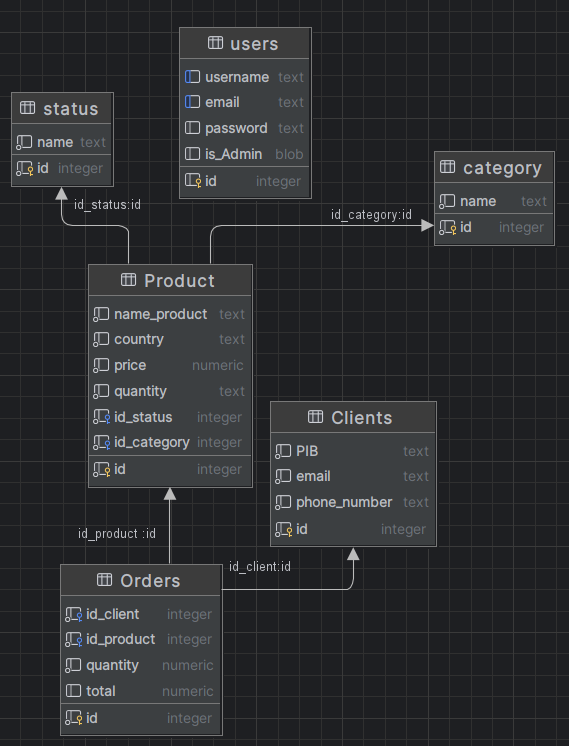


Рисунок 2.1 ERD-діаграма бази даних

## **2.2 Функціональні частини програми**

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

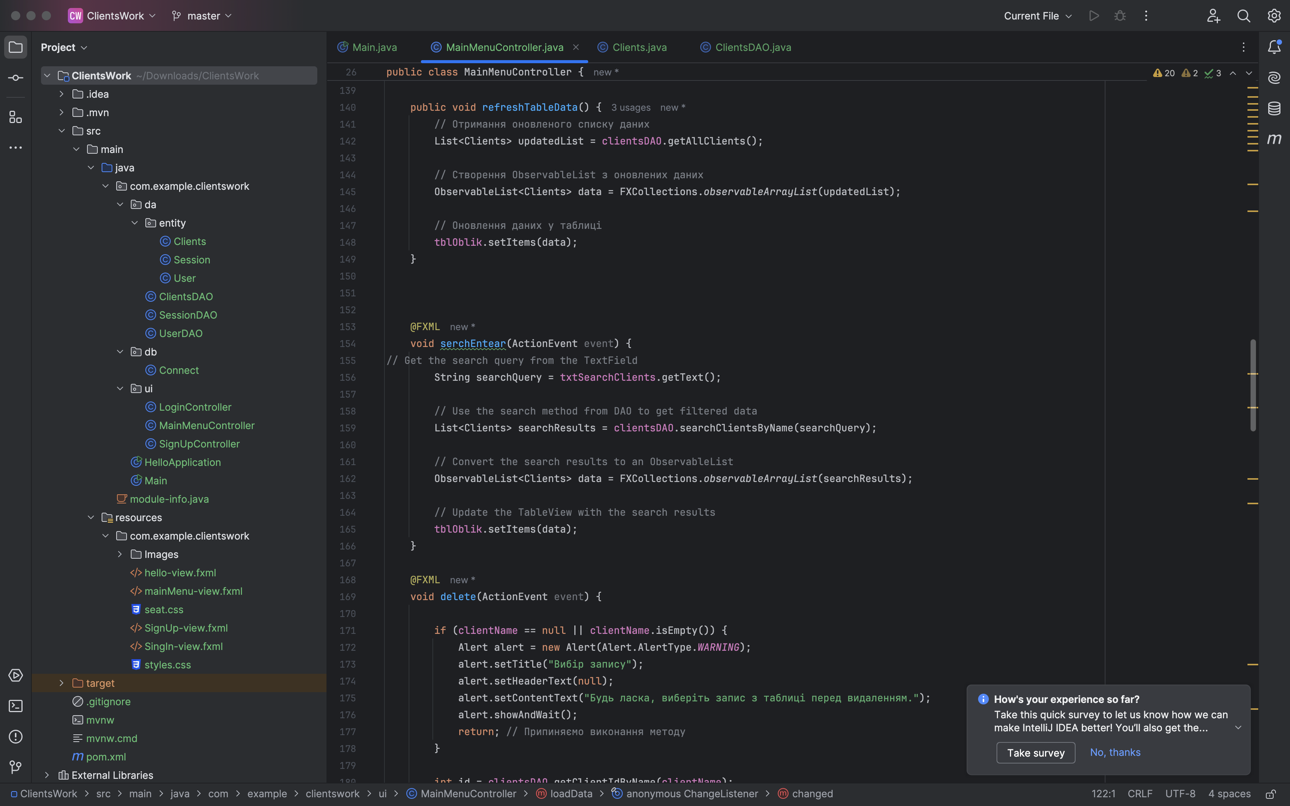


Рисунок 2.2 Проєкт в IntellijIDEA

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.

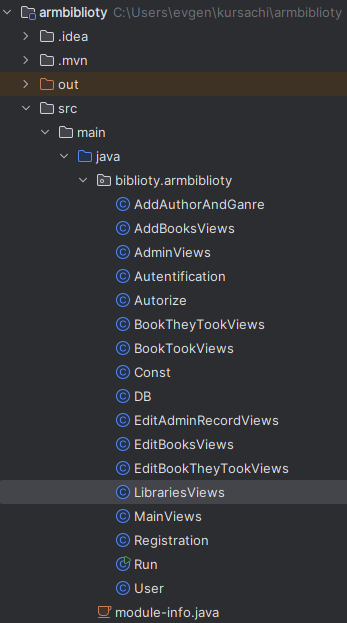
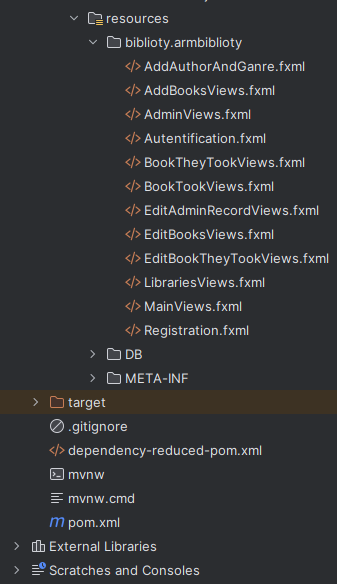
 

Рисунок 2.3 Структура проєкту

Опис головних функціональних файлів проєкту викладено в таблиці 2.1.

Таблиця 2.1 Опис файлів

|  |  |
| --- | --- |
| **Бібліотеки:** | **Опис** |
| java.sql.Connection; | 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** |
| int categoryId | Змінна, яка зберігає в собі ідентифікатор категорії |
| 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*

|  |  |
| --- | --- |
| List<Record> allRecords | Масив, який зберігає в собі всі замовлення |
| ObservableList<String> category | Масив, який отримує в себе всі категорії |
| String name | Змінна, яка записує імʼя назву медичних товарів |
| Connection dbConnection | Статичне з'єднання для одноразового використання бази данних |
| ObservableList<String> clients | Масив, який отримує в себе всіх клієнтів які зареєстровані в базі даних |
| String currentTable | Змінна, яка вказує поточну таблицю |
| int clientId | Змінна, яка зберігає в собі ідентифікатор клієнта |

## **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.

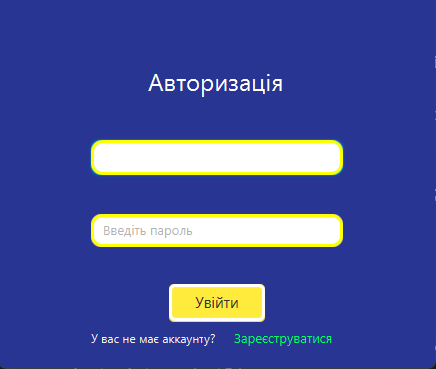


Рисунок 2.4 Вікно аутентифікації

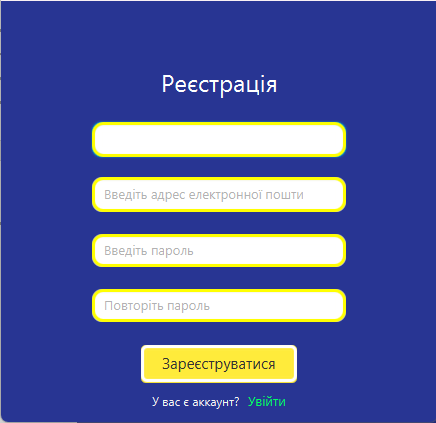


Рисунок 2.5 Вікно реєстрації

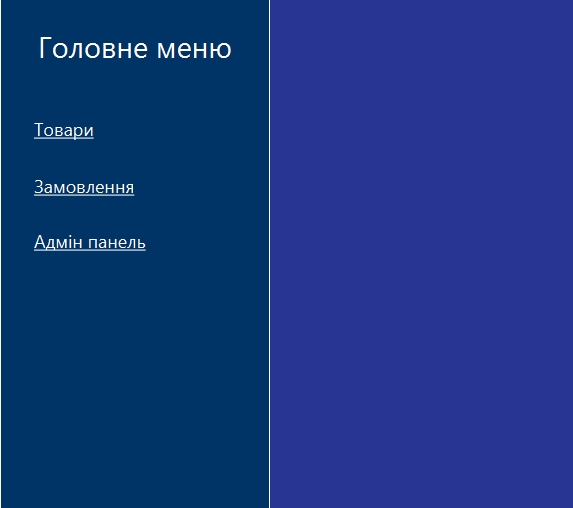


Рисунок 2.6 Головне меню адміністратора

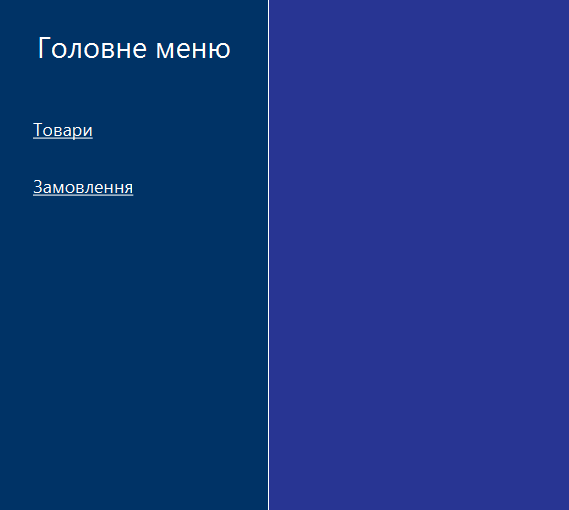


Рисунок 2.7 Головне меню користувача

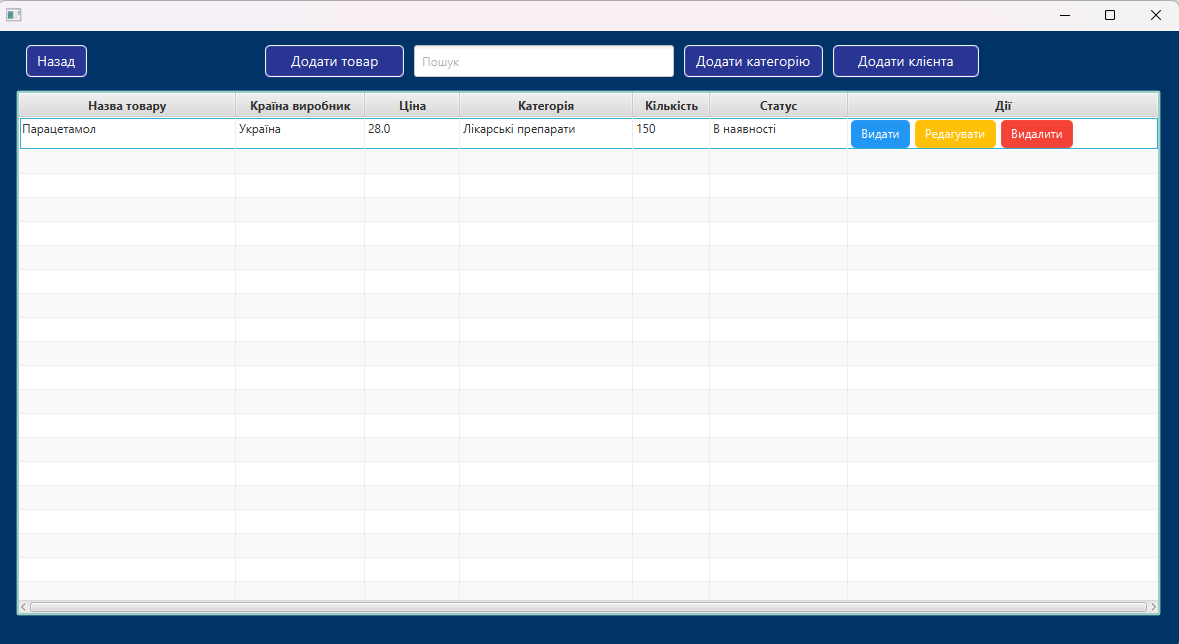


Рисунок 2.8 Вікно з каталогом медичних товарів

## **2.4 Тестування проєкту**

Тестування проводиться на платфомі Windows.

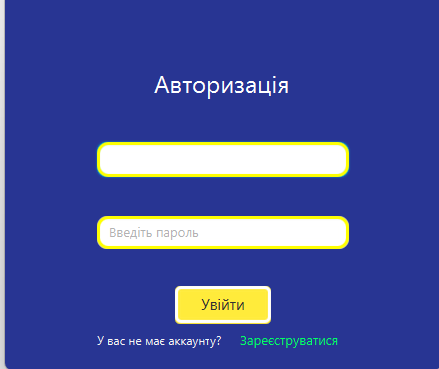


Рисунок 2.9 Вікно аутентифікації

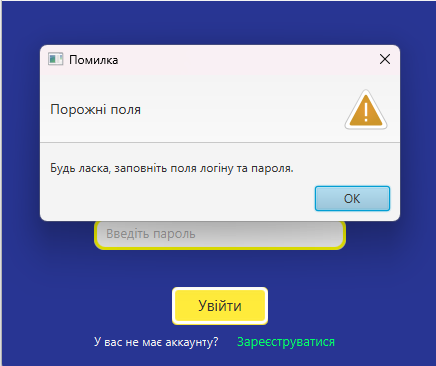


Рисунок 2.10 Реакція додатку на пусті поля

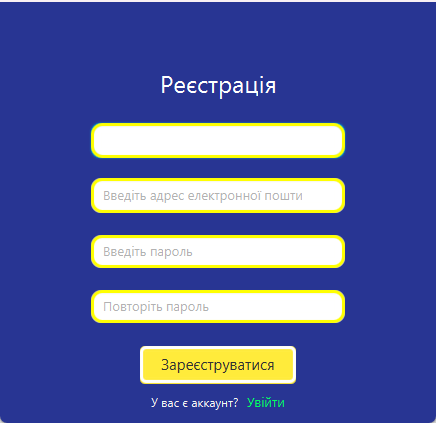


Рисунок 2.11 Вікно реєстрації

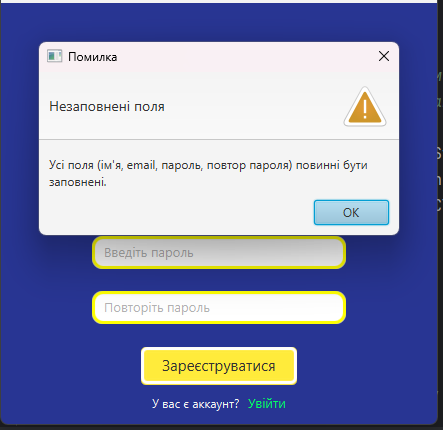


Рисунок 2.12 Реакція додатку на пусті поля

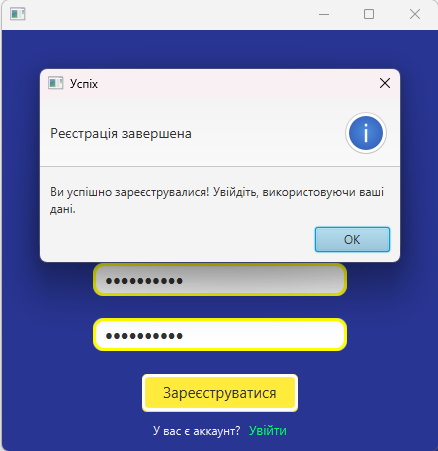


Рисунок 2.13 Успішна реєстрація

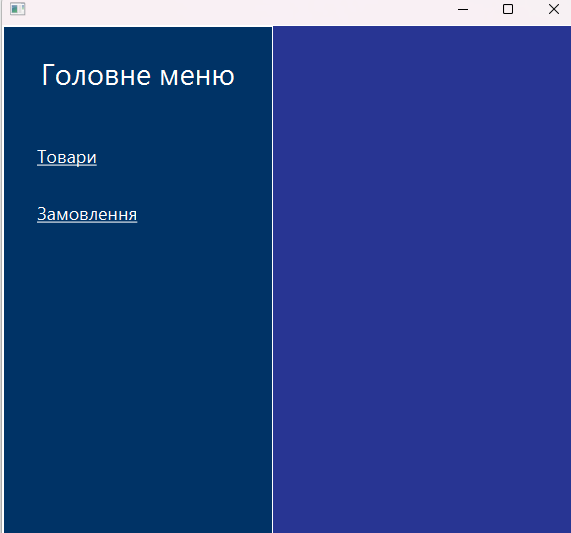


Рисунок 2.14 Головне меню

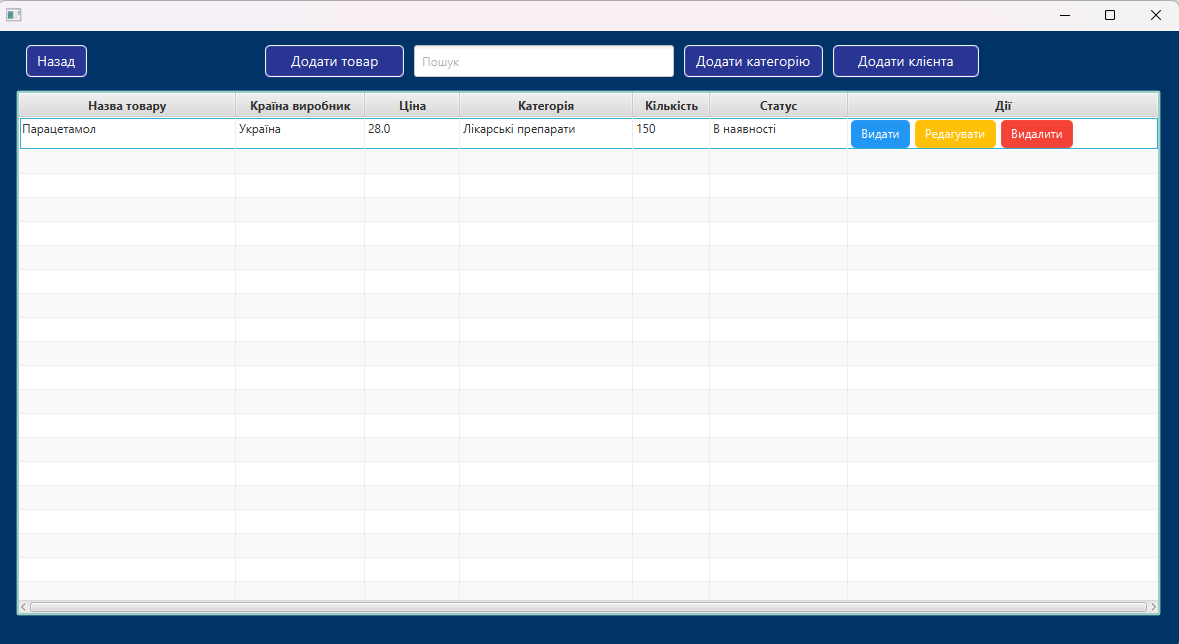


Рисунок 2.15 Вікно з каталогом книг

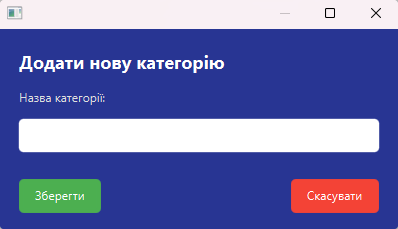


Рисунок 2.16 Вікно додавання категорії

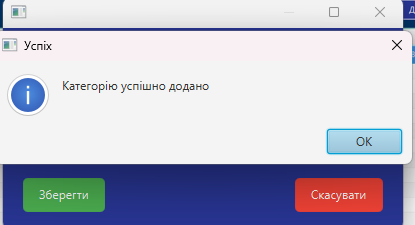


Рисунок 2.17 Успішне додавання категорії

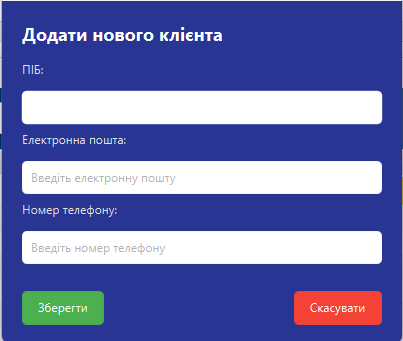


Рисунок 2.18 Вікно додавання клієнта

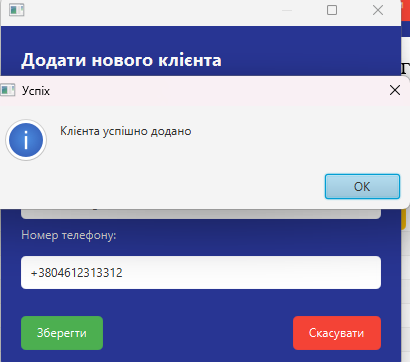
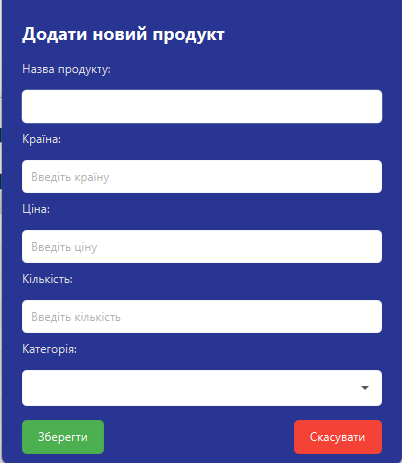
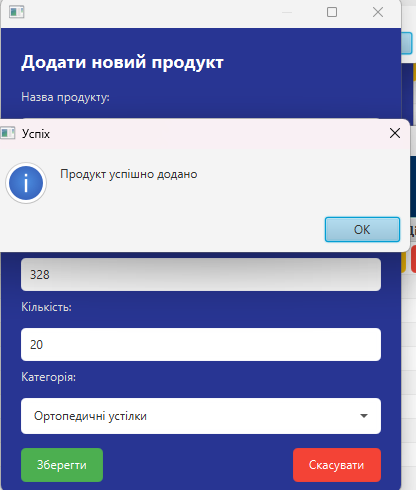


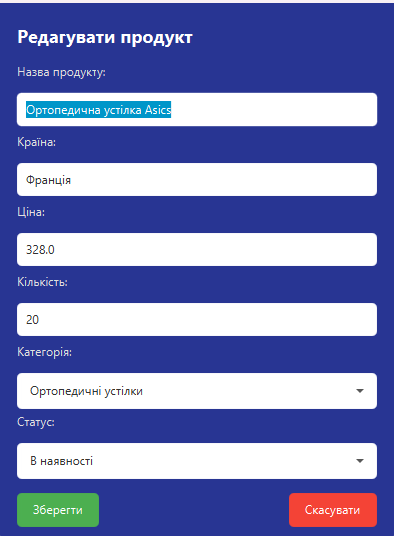
Рисунок 2.20 Успішно доданий клієнт



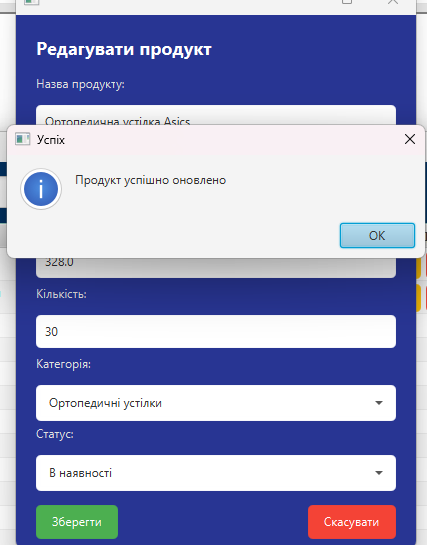
Рисунко 2.21 Вікно додавання товарів



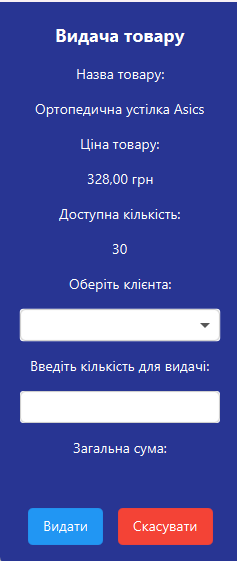
Рисунко 2.22 Успішно доданий товар



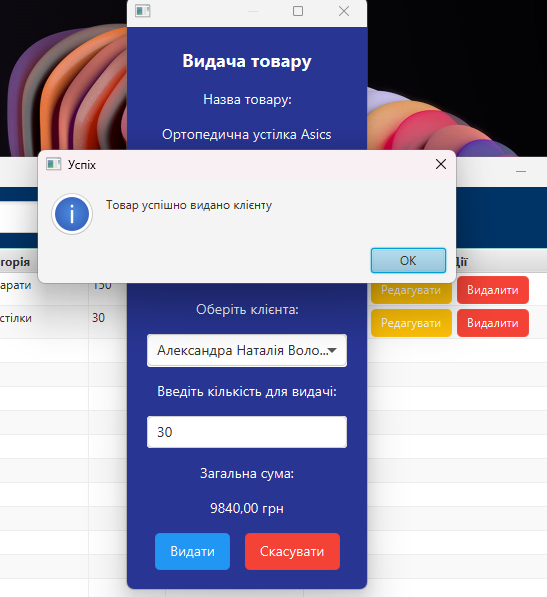
Рисунко 2.23 Вікно редагування

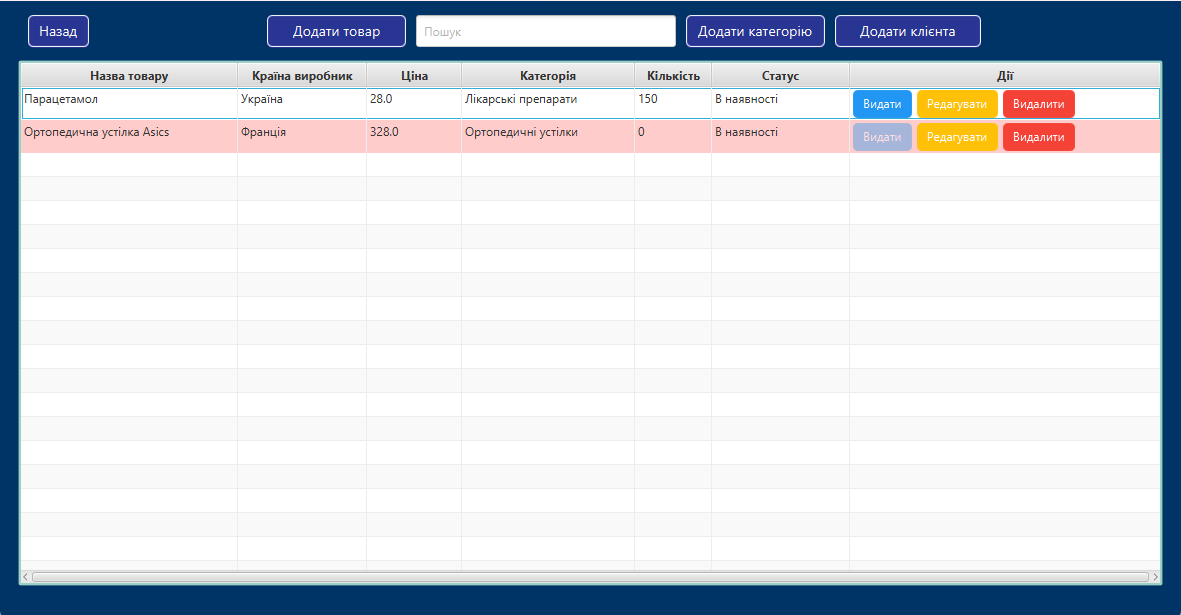


Рисунко 2.24 Успішне редагування

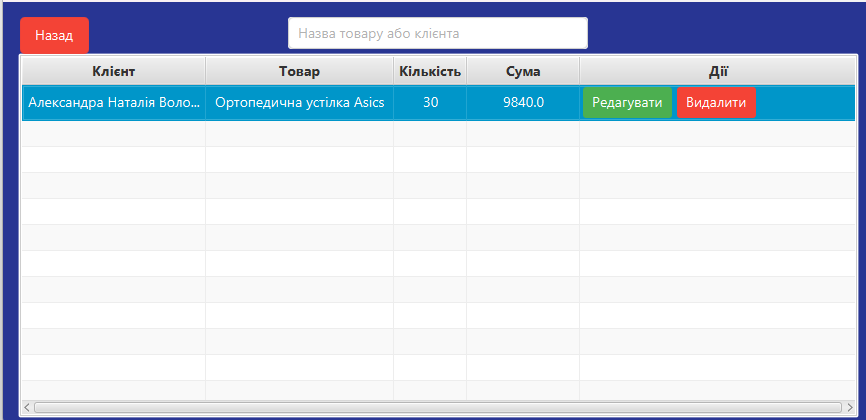


Рисунко 2.25 Вікно видачі товару

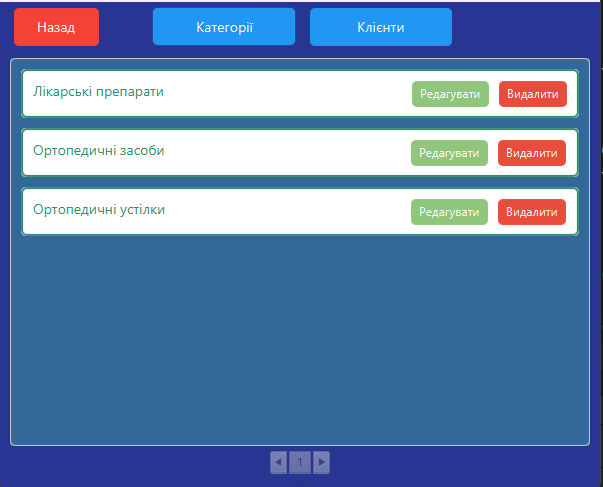
  
Рисунок 2.26 Успішно виданий товар

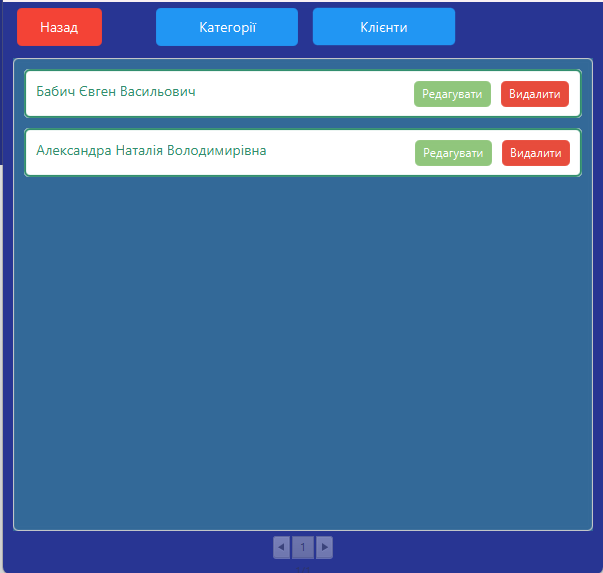


Рисунко 2.27 Такий вигляд має таблиця коли всі товари видані

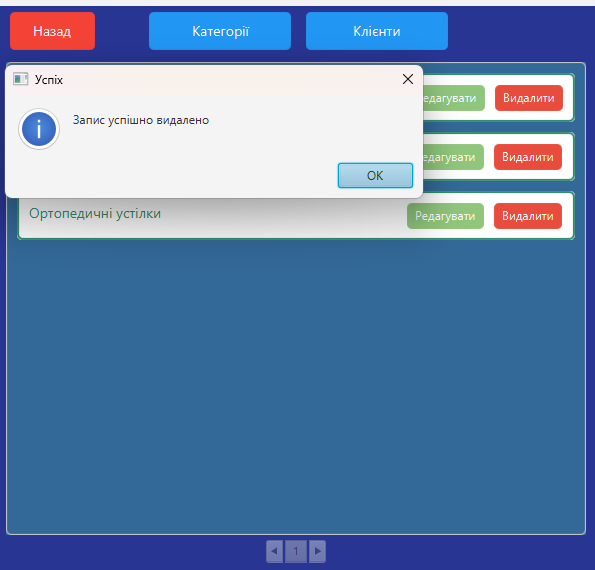


Рисунко 2.28 Вікно з виданими товарами

  
Рисунко 2.29 Адмін панель з категоріями



Рисунко 2.30 Адмін панель з клієнтами

  
Рисунко 2.26 Успішно видалений запис

На цьому тестування можна вважати завершеним. В процесі тестування помилок виявлено не було.

ВИСНОВКИ

Розробка системи обліку медичних товарів, створеної з використанням Java, JavaFX та SQLite, підкреслює важливість і необхідність впровадження автоматизованого інструменту для ефективного управління запасами медичних закладів і аптек. Успішне впровадження такої системи не лише підвищить точність обліку медикаментів, медичного обладнання та витратних матеріалів, але й оптимізує процеси закупівлі, реалізації, контролю термінів придатності та відстеження руху товарів. Це сприятиме підвищенню якості обслуговування клієнтів, зменшенню ризиків втрат через прострочення та забезпечить зручний доступ до актуальної інформації про запаси.

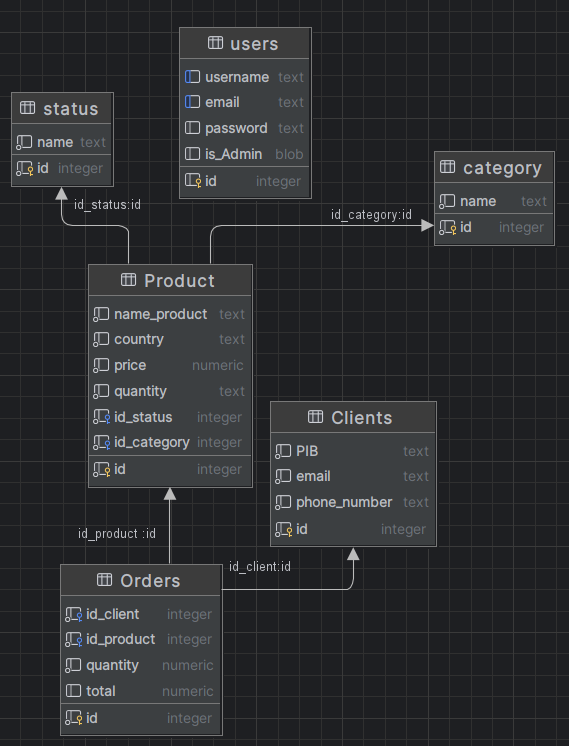
Проект ґрунтується на детальному аналізі потреб медичних закладів у централізованому підході до управління запасами, враховуючи проблеми, пов’язані з відсутністю автоматизації, такі як помилки в обліку, затримки в поставках чи нестача товарів. Система обліку медичних товарів стане ключовим елементом у стратегії підвищення ефективності роботи аптек і медичних установ, забезпечуючи швидкий доступ до даних про товари, їх характеристики, постачальників та історію руху. Крім того, використання сучасних механізмів шифрування для захисту даних гарантує безпеку конфіденційної інформації, що є критично важливим у медичній сфері.

Впровадження такої системи є не лише кроком до цифровізації, але й інвестицією в майбутнє медичної логістики, адже вона дозволяє не тільки економити час і ресурси, але й створює фундамент для інтеграції з іншими системами, наприклад, для автоматизації звітності чи прогнозування попиту. З огляду на це, проект системи обліку медичних товарів є важливим внеском у модернізацію управління медичними ресурсами, відповідаючи сучасним викликам цифрової трансформації. Він забезпечує надійну основу для підвищення ефективності роботи, зниження операційних витрат і, найголовніше, покращення якості медичних послуг для кінцевих споживачів.

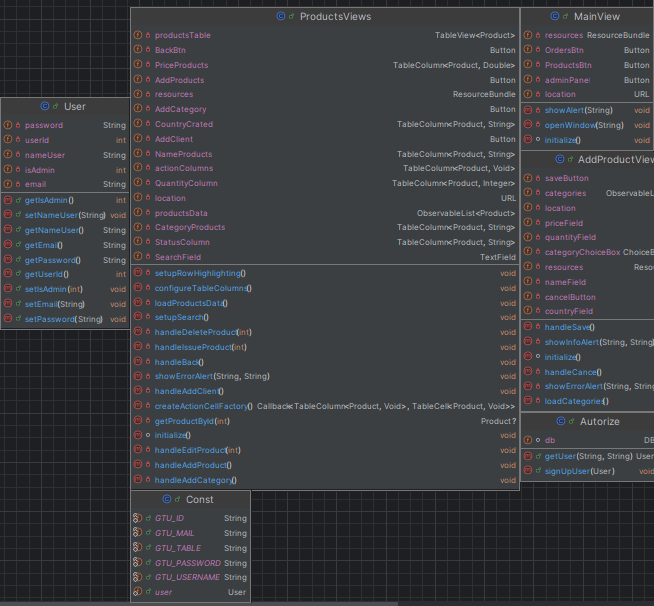
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

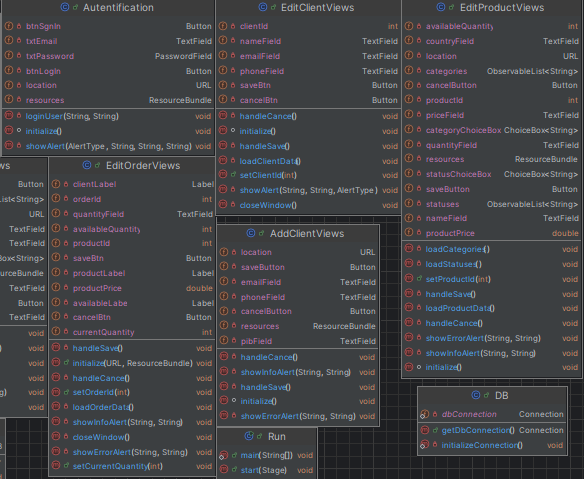
1. Сайт «ПГФК». Про підприємство. URL: https://www.college.uzhnu.edu.ua/ (дата звернення 20.03.2025);
2. Wikipedia. 2024. JAVA. URL: uk.wikipedia.org/wiki/Java (дата звернення 21.03.2025);
3. Wikipedia. 2025. OOP. URL: uk.wikipedia.org/wiki/oop (дата звернення 20.03.2025);
4. Socialite. 2025. Socialite Providers. URL: socialiteproviders.com/about/ (дата звернення 25.03.2025);
5. Wikipedia. 2025. SceneBuilder. URL: uk.wikipedia.org/wiki/ SceneBuilder (дата звернення 21.03.2025);
6. Wikipedia. 2025. SQLite. URL: uk.wikipedia.org/wiki/SQLite (дата звернення 24.03.2025);
7. Wikipedia. IntellijIDEA URL: https://uk.wikipedia.org/wiki/IntelliJ\_IDEA (дата звернення 03.04.2025);
8. IBS Міні-Аптека. 2025. IBS Міні-Аптека URL: https://www.ibsystems.com.ua/ua/ibs-економ-програма-для-аптеки/опис (дата звернення 06.04.2025).

Додаток 1 Бази даних

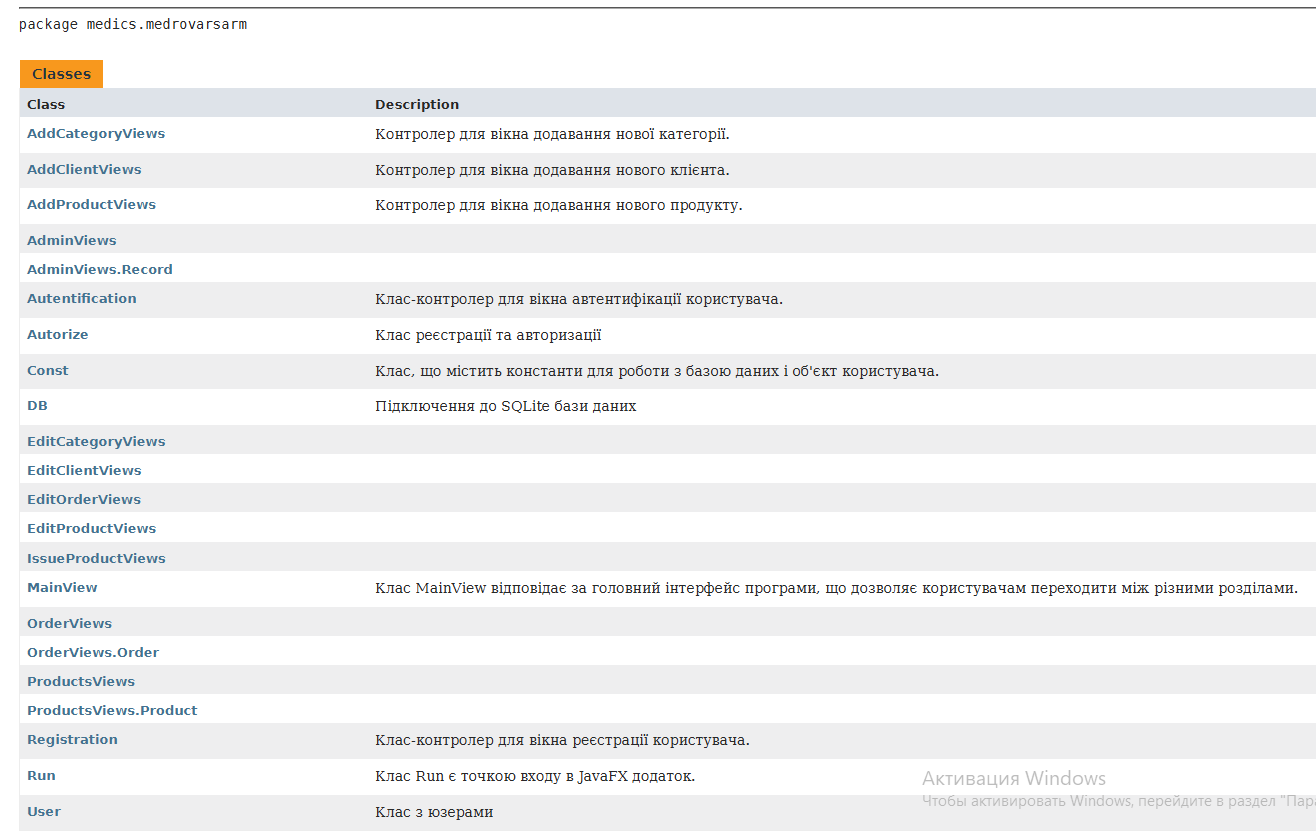


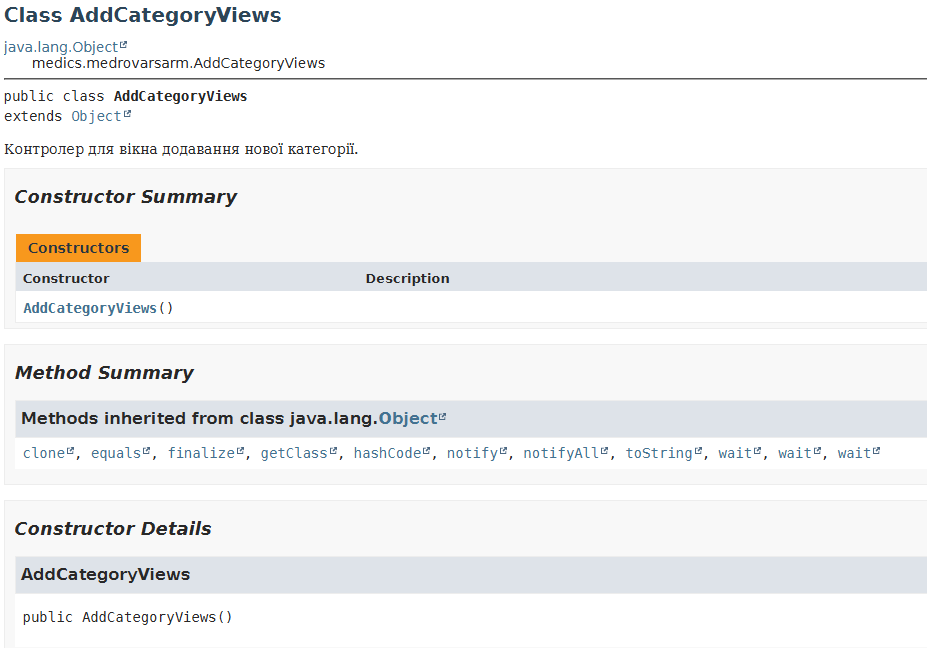
Додаток 2. Схема класів

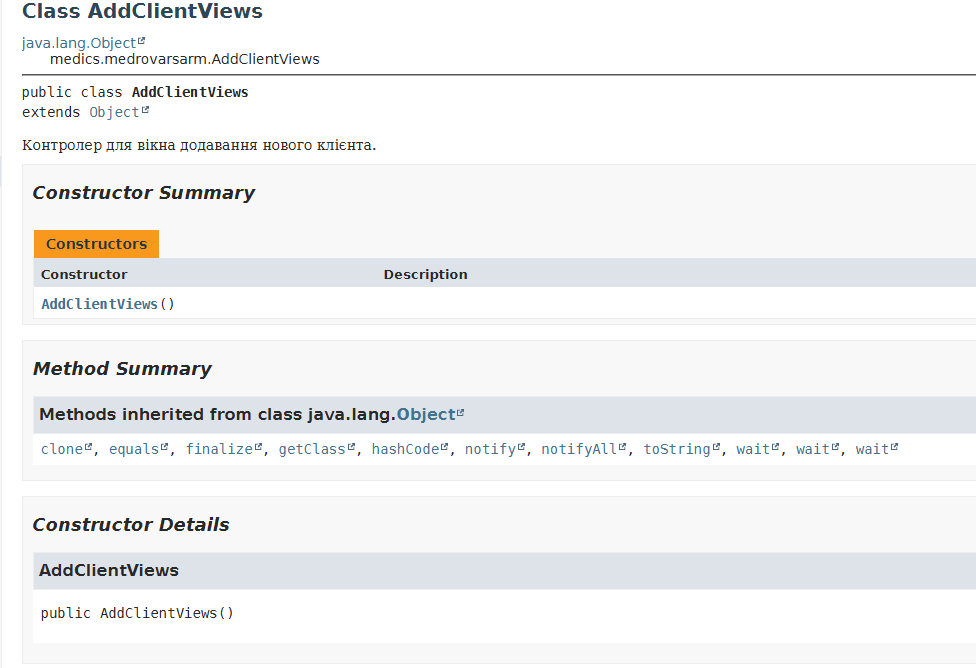


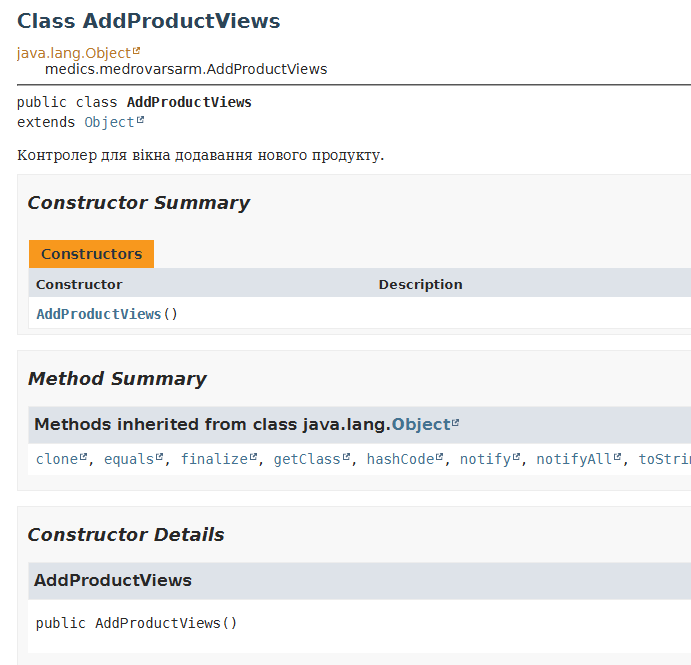


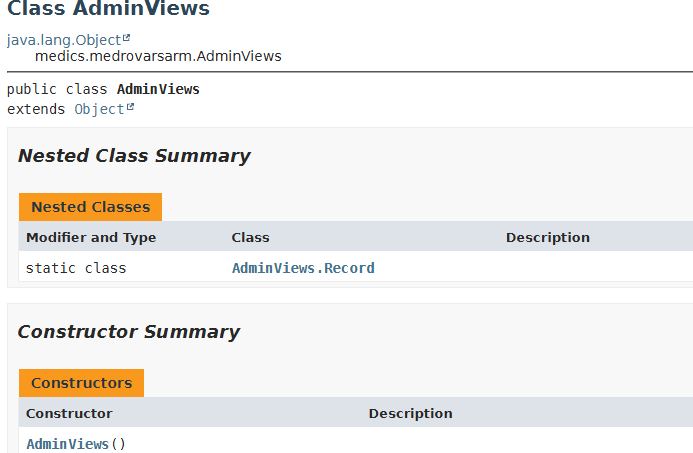
Додаток 3. Документація до програмного коду



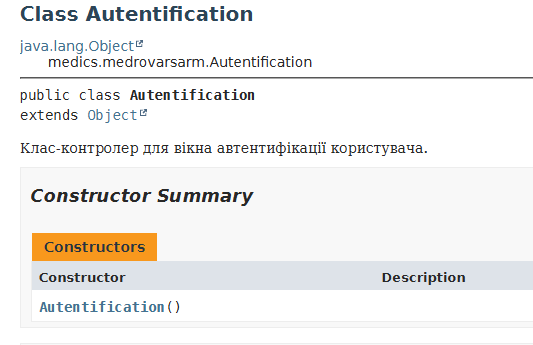


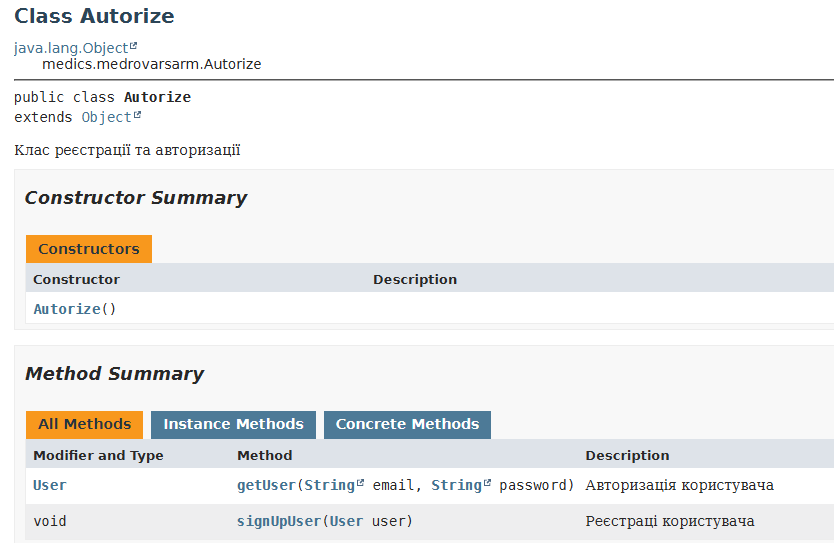




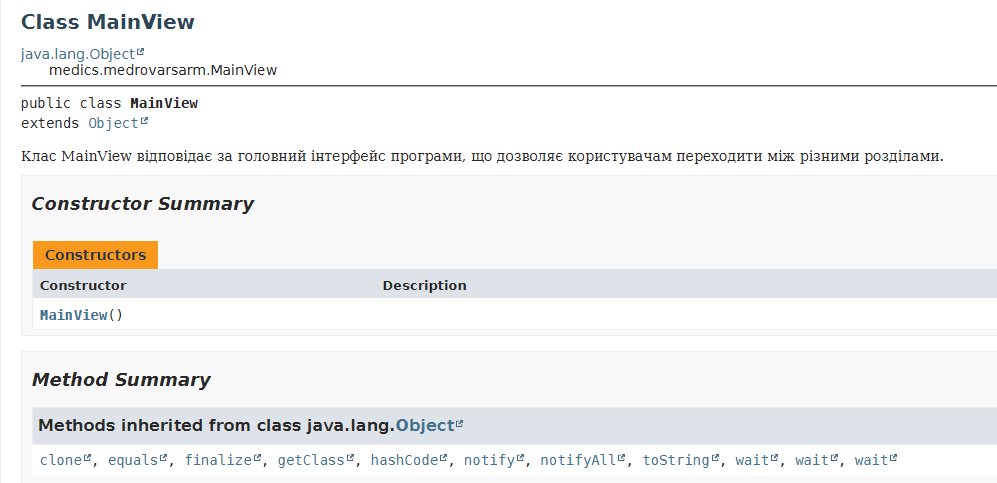


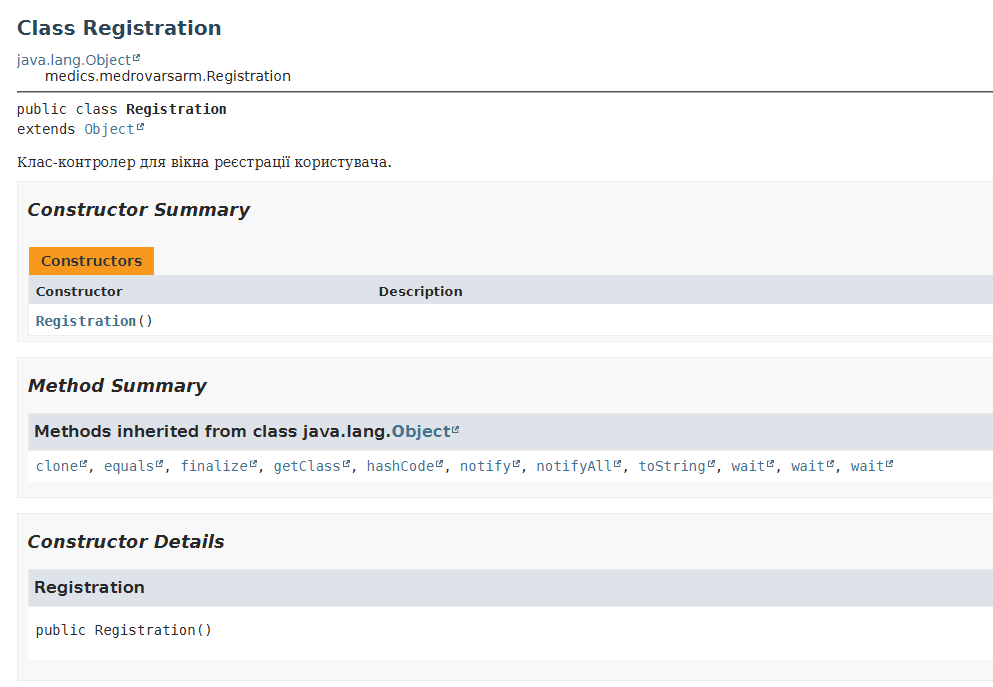




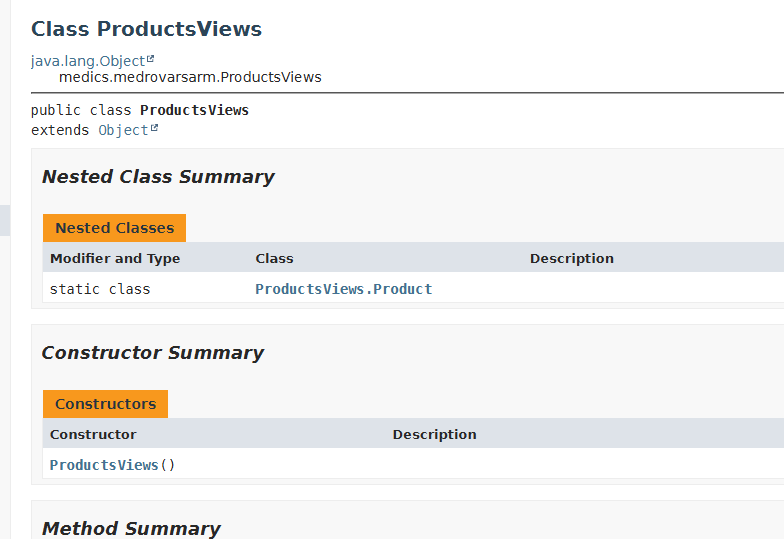












Додаток 4. Лістинг програми

**AddCategoryViews**

package medics.medrovarsarm;

import java.net.URL;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

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;

/\*\*

\* Контролер для вікна додавання нової категорії.

\*/

public class AddCategoryViews {

@FXML private ResourceBundle resources;

@FXML private URL location;

@FXML private TextField categoryNameField;

@FXML private Button saveButton;

@FXML private Button cancelButton;

/\*\*

\* Ініціалізує контролер, налаштовує обробники подій для кнопок.

\*/

@FXML

void initialize() {

saveButton.setOnAction(event -> handleSave());

cancelButton.setOnAction(event -> handleCancel());

}

/\*\*

\* Зберігає нову категорію в базі даних.

\*/

private void handleSave() {

String categoryName = categoryNameField.getText().trim();

if (categoryName.isEmpty()) {

showErrorAlert("Помилка", "Назва категорії не може бути порожньою");

return;

}

DB db = new DB();

Connection conn = null;

try {

conn = db.getDbConnection();

String query = "INSERT INTO category (name) VALUES (?)";

PreparedStatement stmt = conn.prepareStatement(query);

stmt.setString(1, categoryName);

stmt.executeUpdate();

showInfoAlert("Успіх", "Категорію успішно додано");

handleCancel();

} catch (SQLException e) {

showErrorAlert("Помилка бази даних", "Не вдалося додати категорію: " + e.getMessage());

} finally {

try {

if (conn != null) {

// Connection is not closed as per DB class assumption

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

/\*\*

\* Закриває вікно без збереження.

\*/

private void handleCancel() {

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

stage.close();

}

/\*\*

\* Показує повідомлення про помилку.

\*

\* @param title Заголовок повідомлення

\* @param message Текст повідомлення

\*/

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();

}

/\*\*

\* Показує інформаційне повідомлення.

\*

\* @param title Заголовок повідомлення

\* @param message Текст повідомлення

\*/

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();

}

}}

**AddClientViews**

package medics.medrovarsarm;

import java.net.URL;

import java.sql.Connection;

import java.sql.PreparedStatement;

import java.sql.SQLException;

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;

/\*\*

\* Контролер для вікна додавання нового клієнта.

\*/

public class AddClientViews {

@FXML private ResourceBundle resources;

@FXML private URL location;

@FXML private TextField pibField;

@FXML private TextField emailField;

@FXML private TextField phoneField;

@FXML private Button saveButton;

@FXML private Button cancelButton;

/\*\*

\* Ініціалізує контролер, налаштовує обробники подій для кнопок.

\*/

@FXML

void initialize() {

saveButton.setOnAction(event -> handleSave());

cancelButton.setOnAction(event -> handleCancel());

}

/\*\*

\* Зберігає нового клієнта в базі даних.

\*/

private void handleSave() {

String pib = pibField.getText().trim();

String email = emailField.getText().trim();

String phone = phoneField.getText().trim();

if (pib.isEmpty() || email.isEmpty() || phone.isEmpty()) {

showErrorAlert("Помилка", "Усі поля обов'язкові для заповнення");

return;

}

DB db = new DB();

Connection conn = null;

try {

conn = db.getDbConnection();

String query = "INSERT INTO Clients (PIB, email, phone\_number) VALUES (?, ?, ?)";

PreparedStatement stmt = conn.prepareStatement(query);

stmt.setString(1, pib);

stmt.setString(2, email);

stmt.setString(3, phone);

stmt.executeUpdate();

showInfoAlert("Успіх", "Клієнта успішно додано");

handleCancel();

} catch (SQLException e) {

showErrorAlert("Помилка бази даних", "Не вдалося додати клієнта: " + e.getMessage());

} finally {

try {

if (conn != null) {

// Connection is not closed as per DB class assumption

}

} catch (Exception ex) {

ex.printStackTrace();

}

}

}

/\*\*

\* Закриває вікно без збереження.

\*/

private void handleCancel() {

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

stage.close();

}

/\*\*

\* Показує повідомлення про помилку.

\*

\* @param title Заголовок повідомлення

\* @param message Текст повідомлення

\*/

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();

}

/\*\*

\* Показує інформаційне повідомлення.

\*

\* @param title Заголовок повідомлення

\* @param message Текст повідомлення

\*/

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();

}

}

**AdminViews**

package medics.medrovarsarm;  
  
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 categoriesBtn;  
 @FXML private Button clientsBtn;  
 @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() {  
 categoriesBtn.setOnAction(event -> loadRecords("category"));  
 clientsBtn.setOnAction(event -> loadRecords("Clients"));  
 backBtn.setOnAction(event -> handleBack());  
  
 setupPagination();  
 }  
  
 private void handleBack() {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("MainViews.fxml"));  
 Parent root = loader.load();  
 Stage stage = (Stage) backBtn.getScene().getWindow();  
 stage.setScene(new Scene(root));  
 } 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, " + (tableName.equals("Clients") ? "PIB AS name" : "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-background-color: #FFFFFF; -fx-border-color: #328E6E; -fx-border-width: 2; -fx-border-radius: 5; -fx-background-radius: 5; -fx-padding: 10;");  
 recordBox.setMaxWidth(Double.*MAX\_VALUE*);  
  
 Label nameLabel = new Label(record.getName());  
 nameLabel.setStyle("-fx-font-size: 14; -fx-text-fill: #328E6E;");  
 nameLabel.setMaxWidth(Double.*MAX\_VALUE*);  
 HBox.*setHgrow*(nameLabel, Priority.*ALWAYS*);  
  
 Region spacer = new Region();  
 HBox.*setHgrow*(spacer, Priority.*ALWAYS*);  
  
 Button editBtn = new Button("Редагувати");  
 editBtn.setStyle("-fx-background-color: #90C67C; -fx-text-fill: white; -fx-font-size: 12; -fx-border-radius: 5; -fx-background-radius: 5;");  
 editBtn.setOnAction(event -> handleEdit(record));  
  
 Button deleteBtn = new Button("Видалити");  
 deleteBtn.setStyle("-fx-background-color: #E74C3C; -fx-text-fill: white; -fx-font-size: 12; -fx-border-radius: 5; -fx-background-radius: 5;");  
 deleteBtn.setOnAction(event -> handleDelete(record));  
  
 recordBox.getChildren().addAll(nameLabel, spacer, editBtn, deleteBtn);  
 return recordBox;  
 }  
  
 private void handleEdit(Record record) {  
 try {  
 String fxmlFile = record.getTableName().equals("Clients")  
 ? "EditClientViews.fxml"  
 : "EditCategoryViews.fxml";  
  
 FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));  
 Parent root = loader.load();  
  
 if (record.getTableName().equals("Clients")) {  
 EditClientViews controller = loader.getController();  
 controller.setClientId(record.getId());  
 } else {  
 EditCategoryViews controller = loader.getController();  
 controller.setCategoryId(record.getId());  
 }  
  
 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", 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);  
  
 if (record.getTableName().equals("category")) {  
 String deleteProductsQuery = "DELETE FROM Product WHERE id = ?";  
 try (PreparedStatement stmt = conn.prepareStatement(deleteProductsQuery)) {  
 stmt.setInt(1, record.getId());  
 stmt.executeUpdate();  
 }  
 } else if (record.getTableName().equals("Clients")) {  
 String deleteOrdersQuery = "DELETE FROM Orders WHERE id = ?";  
 try (PreparedStatement stmt = conn.prepareStatement(deleteOrdersQuery)) {  
 stmt.setInt(1, record.getId());  
 stmt.executeUpdate();  
 }  
 }  
  
 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 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 medics.medrovarsarm;

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 medics.medrovarsarm;

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;

}

}

**AddProductViews**

package medics.medrovarsarm;

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 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;

import javafx.stage.Stage;

/\*\*

\* Контролер для вікна додавання нового продукту.

\*/

public class AddProductViews {

@FXML private ResourceBundle resources;

@FXML private URL location;

@FXML private TextField nameField;

@FXML private TextField countryField;

@FXML private TextField priceField;

@FXML private TextField quantityField;

@FXML private ChoiceBox<String> categoryChoiceBox;

@FXML private Button saveButton;

@FXML private Button cancelButton;

private ObservableList<String> categories = FXCollections.observableArrayList();

/\*\*

\* Ініціалізує контролер, завантажує категорії та налаштовує обробники подій.

\*/

@FXML

void initialize() {

loadCategories();

categoryChoiceBox.setItems(categories);

saveButton.setOnAction(event -> handleSave());

cancelButton.setOnAction(event -> handleCancel());

}

/\*\*

\* Завантажує список категорій з бази даних.

\*/

private void loadCategories() {

DB db = new DB();

Connection conn = null;

PreparedStatement stmt = null;

ResultSet rs = null;

try {

conn = db.getDbConnection();

String query = "SELECT name FROM category";

stmt = conn.prepareStatement(query);

rs = stmt.executeQuery();

while (rs.next()) {

categories.add(rs.getString("name"));

}

} catch (SQLException e) {

showErrorAlert("Помилка бази даних", "Не вдалося завантажити категорії: " + e.getMessage());

} finally {

try {

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

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

} catch (SQLException ex) {

ex.printStackTrace();

}

}

}

/\*\*

\* Зберігає новий продукт у базі даних з id\_status = 1.

\*/

private void handleSave() {

String name = nameField.getText().trim();

String country = countryField.getText().trim();

String priceText = priceField.getText().trim();

String quantityText = quantityField.getText().trim();

String category = categoryChoiceBox.getValue();

if (name.isEmpty() || country.isEmpty() || priceText.isEmpty() || quantityText.isEmpty() ||

category == null) {

showErrorAlert("Помилка", "Усі поля обов'язкові для заповнення");

return;

}

double price;

int quantity;

try {

price = Double.parseDouble(priceText);

quantity = Integer.parseInt(quantityText);

} catch (NumberFormatException e) {

showErrorAlert("Помилка", "Ціна та кількість повинні бути коректними числами");

return;

}

DB db = new DB();

Connection conn = null;

try {

conn = db.getDbConnection();

conn.setAutoCommit(false);

String categoryQuery = "SELECT id FROM category WHERE name = ?";

PreparedStatement categoryStmt = conn.prepareStatement(categoryQuery);

categoryStmt.setString(1, category);

ResultSet categoryRs = categoryStmt.executeQuery();

int categoryId = categoryRs.next() ? categoryRs.getInt("id") : -1;

if (categoryId == -1) {

showErrorAlert("Помилка", "Некоректна категорія");

return;

}

String query = "INSERT INTO Product (name\_product, country, price, quantity, id\_status, id\_category) " +

"VALUES (?, ?, ?, ?, 1, ?)";

PreparedStatement stmt = conn.prepareStatement(query);

stmt.setString(1, name);

stmt.setString(2, country);

stmt.setDouble(3, price);

stmt.setString(4, String.valueOf(quantity));

stmt.setInt(5, categoryId);

stmt.executeUpdate();

conn.commit();

showInfoAlert("Успіх", "Продукт успішно додано");

handleCancel();

} catch (SQLException e) {

try {

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

} catch (SQLException ex) {

ex.printStackTrace();

}

showErrorAlert("Помилка бази даних", "Не вдалося додати продукт: " + e.getMessage());

} finally {

try {

if (conn != null) {

conn.setAutoCommit(true);

}

} catch (SQLException ex) {

ex.printStackTrace();

}

}

}

/\*\*

\* Закриває вікно без збереження.

\*/

private void handleCancel() {

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

stage.close();

}

/\*\*

\* Показує повідомлення про помилку.

\*

\* @param title Заголовок повідомлення

\* @param message Текст повідомлення

\*/

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();

}

/\*\*

\* Показує інформаційне повідомлення.

\*

\* @param title Заголовок повідомлення

\* @param message Текст повідомлення

\*/

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();

}

}

**EditCategory Views**

package medics.medrovarsarm;  
  
import javafx.fxml.FXML;  
import javafx.scene.control.Alert;  
import javafx.scene.control.Button;  
import javafx.scene.control.TextField;  
import javafx.stage.Stage;  
  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.SQLException;  
  
public class EditCategoryViews {  
  
 @FXML private TextField nameField;  
 @FXML private Button saveBtn;  
 @FXML private Button cancelBtn;  
  
 private int categoryId;  
  
 public void setCategoryId(int categoryId) {  
 this.categoryId = categoryId;  
 loadCategoryData();  
 }  
  
 @FXML  
 void initialize() {  
 saveBtn.setOnAction(event -> handleSave());  
 cancelBtn.setOnAction(event -> handleCancel());  
 }  
  
 private void loadCategoryData() {  
 DB db = new DB();  
 try (Connection conn = db.getDbConnection();  
 PreparedStatement stmt = conn.prepareStatement(  
 "SELECT name FROM category WHERE id = ?")) {  
  
 stmt.setInt(1, categoryId);  
 var rs = stmt.executeQuery();  
  
 if (rs.next()) {  
 nameField.setText(rs.getString("name"));  
 }  
 } catch (SQLException e) {  
 showAlert("Помилка", "Не вдалося завантажити дані категорії", Alert.AlertType.*ERROR*);  
 }  
 }  
  
 private void handleSave() {  
 if (nameField.getText().isEmpty()) {  
 showAlert("Помилка", "Назва категорії не може бути порожньою", Alert.AlertType.*ERROR*);  
 return;  
 }  
  
 DB db = new DB();  
 try (Connection conn = db.getDbConnection();  
 PreparedStatement stmt = conn.prepareStatement(  
 "UPDATE category SET name = ? WHERE id = ?")) {  
  
 stmt.setString(1, nameField.getText());  
 stmt.setInt(2, categoryId);  
 stmt.executeUpdate();  
  
 showAlert("Успіх", "Категорію успішно оновлено", Alert.AlertType.*INFORMATION*);  
 closeWindow();  
 } catch (SQLException e) {  
 showAlert("Помилка", "Не вдалося оновити категорію: " + e.getMessage(), Alert.AlertType.*ERROR*);  
 }  
 }  
  
 private void handleCancel() {  
 closeWindow();  
 }  
  
 private void closeWindow() {  
 Stage stage = (Stage) saveBtn.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 medics.medrovarsarm;

/\*\*

\* 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 medics.medrovarsarm;  
  
  
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()) {  
  
 Path dbPath = Paths.*get*( "DB", "Medics.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*;  
 }  
  
}

**MainViews**

package medics.medrovarsarm;  
  
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.Alert;  
import javafx.scene.control.Button;  
import javafx.stage.Stage;  
  
*/\*\*  
 \* Клас MainView відповідає за головний інтерфейс програми, що дозволяє користувачам переходити між різними розділами.  
 \*/*public class MainView {  
  
 @FXML  
 private ResourceBundle resources;  
  
 @FXML  
 private URL location;  
  
 @FXML  
 private Button ProductsBtn;  
  
 @FXML  
 private Button OrdersBtn;  
  
 @FXML  
 private Button adminPanel;  
 */\*\*  
 \* Ініціалізує компонент, перевіряє права адміністратора та налаштовує обробники подій для кнопок.  
 \*/* @FXML  
 void initialize() {  
 if (Const.*user*.getIsAdmin() != 1) {  
 adminPanel.setVisible(false); // Приховуємо панель адміністратора, якщо користувач не є адміністратором  
 }  
 ProductsBtn.setOnAction(event -> {  
 openWindow("ProductsViews.fxml");  
  
 });  
 OrdersBtn.setOnAction(event -> {  
  
 openWindow("OrderViews.fxml");  
 });  
  
  
 adminPanel.setOnAction(event -> openWindow("AdminViews.fxml"));  
  
 }  
  
 */\*\*  
 \* Метод для відкриття нового вікна.  
 \*  
 \* @param fxmlFile ім'я файлу FXML для завантаження  
 \*/* private void openWindow(String fxmlFile) {  
 Stage stage = (Stage) adminPanel.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));  
 }  
  
 */\*\*  
 \* Метод для відображення інформаційного повідомлення.  
 \*  
 \* @param message текст повідомлення  
 \*/* private void showAlert(String message) {  
 Alert alert = new Alert(Alert.AlertType.*INFORMATION*);  
 alert.setTitle("Інформація");  
 alert.setHeaderText(null);  
 alert.setContentText(message);  
 alert.showAndWait();  
 }  
}

**Registration**

package medics.medrovarsarm;

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 medics.medrovarsarm;

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 medics.medrovarsarm;

/\*\*

\* 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;

}

}

**ProductsViews**

package medics.medrovarsarm;

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 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 ProductsViews {

@FXML private ResourceBundle resources;

@FXML private URL location;

@FXML private Button AddCategory;

@FXML private Button AddClient;

@FXML private Button AddProducts;

@FXML private Button BackBtn;

@FXML private TableColumn<Product, String> CategoryProducts;

@FXML private TableColumn<Product, String> CountryCrated;

@FXML private TableColumn<Product, String> NameProducts;

@FXML private TableColumn<Product, Double> PriceProducts;

@FXML private TableColumn<Product, Integer> QuantityColumn;

@FXML private TextField SearchField;

@FXML private TableColumn<Product, String> StatusColumn;

@FXML private TableView<Product> productsTable;

@FXML private TableColumn<Product, Void> actionColumns;

private ObservableList<Product> productsData = FXCollections.observableArrayList();

@FXML

void initialize() {

AddProducts.setOnAction(event -> handleAddProduct());

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

AddCategory.setOnAction(event -> handleAddCategory());

AddClient.setOnAction(event -> handleAddClient());

try {

configureTableColumns();

loadProductsData();

setupSearch();

setupRowHighlighting();

} catch (Exception e) {

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

e.printStackTrace();

}

}

private void configureTableColumns() {

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

CategoryProducts.setCellValueFactory(new PropertyValueFactory<>("category"));

CountryCrated.setCellValueFactory(new PropertyValueFactory<>("country"));

PriceProducts.setCellValueFactory(new PropertyValueFactory<>("price"));

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

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

actionColumns.setCellFactory(createActionCellFactory());

}

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

return new Callback<>() {

@Override

public TableCell<Product, Void> call(final TableColumn<Product, Void> param) {

return new TableCell<>() {

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

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

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

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

{

issueBtn.setStyle("-fx-background-color: #2196F3; -fx-text-fill: white; -fx-background-radius: 5; -fx-padding: 5 10; -fx-font-family: 'Roboto';");

editBtn.setStyle("-fx-background-color: #FFC107; -fx-text-fill: white; -fx-background-radius: 5; -fx-padding: 5 10; -fx-font-family: 'Roboto';");

deleteBtn.setStyle("-fx-background-color: #f44336; -fx-text-fill: white; -fx-background-radius: 5; -fx-padding: 5 10; -fx-font-family: 'Roboto';");

issueBtn.setOnAction(event -> {

int productId = getTableView().getItems().get(getIndex()).getId();

handleIssueProduct(productId);

});

editBtn.setOnAction(event -> {

int productId = getTableView().getItems().get(getIndex()).getId();

handleEditProduct(productId);

});

deleteBtn.setOnAction(event -> {

int productId = getTableView().getItems().get(getIndex()).getId();

handleDeleteProduct(productId);

});

}

@Override

public void updateItem(Void item, boolean empty) {

super.updateItem(item, empty);

if (empty) {

setGraphic(null);

} else {

Product product = getTableView().getItems().get(getIndex());

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

setGraphic(hbox);

}

}

};

}

};

}

private void handleIssueProduct(int productId) {

try {

Product product = getProductById(productId);

if (product == null) {

showErrorAlert("Помилка", "Продукт не знайдено");

return;

}

URL fxmlLocation = getClass().getResource("IssueProductViews.fxml");

if (fxmlLocation == null) {

showErrorAlert("Помилка", "Не вдалося знайти IssueProductViews.fxml");

return;

}

FXMLLoader loader = new FXMLLoader(fxmlLocation);

Parent root = loader.load();

IssueProductViews controller = loader.getController();

if (controller == null) {

showErrorAlert("Помилка", "Контролер IssueProductViews не ініціалізований");

return;

}

controller.setProductId(productId);

controller.setAvailableQuantity(product.getAvailableQuantity());

controller.setProductPrice(product.getPrice());

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadProductsData();

} catch (Exception e) {

showErrorAlert("Помилка", "Не вдалося відкрити форму видачі товару: " + e.getMessage());

e.printStackTrace();

}

}

private void handleEditProduct(int productId) {

try {

URL fxmlLocation = getClass().getResource("EditProductViews.fxml");

if (fxmlLocation == null) {

showErrorAlert("Помилка", "Не вдалося знайти EditProductViews.fxml");

return;

}

FXMLLoader loader = new FXMLLoader(fxmlLocation);

Parent root = loader.load();

EditProductViews controller = loader.getController();

if (controller == null) {

showErrorAlert("Помилка", "Контролер EditProductViews не ініціалізований");

return;

}

controller.setProductId(productId);

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadProductsData();

} catch (Exception e) {

showErrorAlert("Помилка", "Не вдалося відкрити форму редагування товару: " + e.getMessage());

e.printStackTrace();

}

}

private void handleDeleteProduct(int productId) {

Product product = getProductById(productId);

if (product == null) {

showErrorAlert("Помилка", "Продукт не знайдено");

return;

}

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

confirmation.setTitle("Підтвердження видалення");

confirmation.setHeaderText(null);

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

if (confirmation.showAndWait().get() == ButtonType.OK) {

DB db = new DB();

try (Connection conn = db.getDbConnection();

PreparedStatement stmt = conn.prepareStatement("DELETE FROM Product WHERE id = ?")) {

stmt.setInt(1, productId);

stmt.executeUpdate();

loadProductsData();

} catch (SQLException e) {

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

}

}

}

private Product getProductById(int productId) {

DB db = new DB();

try (Connection conn = db.getDbConnection();

PreparedStatement stmt = conn.prepareStatement(

"SELECT p.id, p.name\_product, c.name as category\_name, p.country, " +

"p.price, p.quantity as total\_quantity, s.name as status\_name, " +

"(SELECT COALESCE(SUM(o.quantity), 0) FROM Orders o WHERE o.id\_product = p.id) as ordered\_quantity " +

"FROM Product p " +

"JOIN category c ON p.id\_category = c.id " +

"JOIN status s ON p.id\_status = s.id " +

"WHERE p.id = ?")) {

stmt.setInt(1, productId);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

int totalQuantity = rs.getInt("total\_quantity");

int orderedQuantity = rs.getInt("ordered\_quantity");

int availableQuantity = totalQuantity - orderedQuantity;

return new Product(

rs.getInt("id"),

rs.getString("name\_product"),

rs.getString("category\_name"),

rs.getString("country"),

rs.getDouble("price"),

totalQuantity,

availableQuantity,

rs.getString("status\_name")

);

}

} catch (SQLException e) {

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

}

return null;

}

private synchronized void loadProductsData() {

productsData.clear();

DB db = new DB();

try (Connection conn = db.getDbConnection();

PreparedStatement stmt = conn.prepareStatement(

"SELECT p.id, p.name\_product, c.name as category\_name, p.country, " +

"p.price, p.quantity as total\_quantity, s.name as status\_name, " +

"(SELECT COALESCE(SUM(o.quantity), 0) FROM Orders o WHERE o.id\_product = p.id) as ordered\_quantity " +

"FROM Product p " +

"JOIN category c ON p.id\_category = c.id " +

"JOIN status s ON p.id\_status = s.id");

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

int totalQuantity = rs.getInt("total\_quantity");

int orderedQuantity = rs.getInt("ordered\_quantity");

int availableQuantity = totalQuantity - orderedQuantity;

productsData.add(new Product(

rs.getInt("id"),

rs.getString("name\_product"),

rs.getString("category\_name"),

rs.getString("country"),

rs.getDouble("price"),

totalQuantity,

availableQuantity,

rs.getString("status\_name")

));

}

} catch (SQLException e) {

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

}

}

private void setupRowHighlighting() {

productsTable.setRowFactory(tv -> new TableRow<Product>() {

@Override

protected void updateItem(Product product, boolean empty) {

super.updateItem(product, empty);

if (product == null || empty) {

setStyle("");

} else {

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

}

}

});

}

private void setupSearch() {

FilteredList<Product> filteredData = new FilteredList<>(productsData, p -> true);

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

filteredData.setPredicate(product -> {

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

String lowerCaseFilter = newValue.toLowerCase();

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

product.getCategory().toLowerCase().contains(lowerCaseFilter) ||

product.getCountry().toLowerCase().contains(lowerCaseFilter) ||

String.valueOf(product.getPrice()).contains(lowerCaseFilter) ||

String.valueOf(product.getAvailableQuantity()).contains(lowerCaseFilter);

});

});

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

sortedData.comparatorProperty().bind(productsTable.comparatorProperty());

productsTable.setItems(sortedData);

}

@FXML

private void handleAddProduct() {

try {

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

Parent root = loader.load();

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadProductsData();

} catch (Exception e) {

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

}

}

@FXML

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) {

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

}

}

@FXML

private void handleAddCategory() {

try {

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

Parent root = loader.load();

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadProductsData();

} catch (Exception e) {

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

}

}

@FXML

private void handleAddClient() {

try {

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

Parent root = loader.load();

Stage stage = new Stage();

stage.initModality(Modality.APPLICATION\_MODAL);

stage.setScene(new Scene(root));

stage.showAndWait();

loadProductsData();

} catch (Exception e) {

showErrorAlert("Помилка додавання клієнта", 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();

}

public static class Product {

private final int id;

private final String name;

private final String category;

private final String country;

private final double price;

private final int totalQuantity;

private final int availableQuantity;

private final String status;

public Product(int id, String name, String category, String country, double price,

int totalQuantity, int availableQuantity, String status) {

this.id = id;

this.name = name;

this.category = category;

this.country = country;

this.price = price;

this.totalQuantity = totalQuantity;

this.availableQuantity = availableQuantity;

this.status = status;

}

public int getId() { return id; }

public String getName() { return name; }

public String getCategory() { return category; }

public String getCountry() { return country; }

public double getPrice() { return price; }

public int getTotalQuantity() { return totalQuantity; }

public int getAvailableQuantity() { return availableQuantity; }

public String getStatus() { return status; }

}

}

**OrderViews**

package medics.medrovarsarm;  
  
import java.io.IOException;  
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 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 OrderViews {  
  
 @FXML private ResourceBundle resources;  
 @FXML private URL location;  
 @FXML private TableView<Order> ordersTable;  
 @FXML private TableColumn<Order, String> clientColumn;  
 @FXML private TableColumn<Order, String> productColumn;  
 @FXML private TableColumn<Order, Integer> quantityColumn;  
 @FXML private TableColumn<Order, Double> totalColumn;  
 @FXML private TableColumn<Order, Void> actionColumn;  
 @FXML private TextField searchField;  
 @FXML private Button backBtn;  
  
 private ObservableList<Order> ordersData = FXCollections.*observableArrayList*();  
  
 @FXML  
 void initialize() {  
 configureTableColumns();  
 loadOrdersData();  
 setupSearch();  
  
 backBtn.setOnAction(event -> {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("MainViews.fxml"));  
 Parent root = loader.load();  
 Stage stage = (Stage) backBtn.getScene().getWindow();  
 stage.setScene(new Scene(root));  
 } catch (IOException e) {  
 showErrorAlert("Помилка", "Не вдалося повернутися до головного меню");  
 }  
 });  
 }  
  
 private void configureTableColumns() {  
 clientColumn.setCellValueFactory(new PropertyValueFactory<>("clientName"));  
 productColumn.setCellValueFactory(new PropertyValueFactory<>("productName"));  
 quantityColumn.setCellValueFactory(new PropertyValueFactory<>("quantity"));  
 totalColumn.setCellValueFactory(new PropertyValueFactory<>("total"));  
  
 actionColumn.setCellFactory(createActionCellFactory());  
 }  
  
 private Callback<TableColumn<Order, Void>, TableCell<Order, Void>> createActionCellFactory() {  
 return param -> new TableCell<>() {  
 private final Button editBtn = new Button("Редагувати");  
 private final Button deleteBtn = new Button("Видалити");  
 private final HBox buttons = new HBox(5, editBtn, deleteBtn);  
  
 {  
 editBtn.setStyle("-fx-background-color: #4CAF50; -fx-text-fill: white;");  
 deleteBtn.setStyle("-fx-background-color: #f44336; -fx-text-fill: white;");  
  
 editBtn.setOnAction(event -> {  
 Order order = getTableView().getItems().get(getIndex());  
 handleEditOrder(order);  
 });  
  
 deleteBtn.setOnAction(event -> {  
 Order order = getTableView().getItems().get(getIndex());  
 handleDeleteOrder(order);  
 });  
 }  
  
 @Override  
 protected void updateItem(Void item, boolean empty) {  
 super.updateItem(item, empty);  
 if (empty) {  
 setGraphic(null);  
 } else {  
 setGraphic(buttons);  
 }  
 }  
 };  
 }  
  
 private void handleEditOrder(Order order) {  
 try {  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("EditOrderViews.fxml"));  
 Parent root = loader.load();  
  
 EditOrderViews controller = loader.getController();  
 controller.setOrderId(order.getId());  
 controller.setCurrentQuantity(order.getQuantity());  
  
 Stage stage = new Stage();  
 stage.initModality(Modality.APPLICATION\_MODAL);  
 stage.setScene(new Scene(root));  
 stage.showAndWait();  
  
 loadOrdersData();  
 } catch (IOException e) {  
 showErrorAlert("Помилка", "Не вдалося відкрити форму редагування");  
 }  
 }  
  
 private void handleDeleteOrder(Order order) {  
 Alert confirmation = new Alert(Alert.AlertType.CONFIRMATION);  
 confirmation.setTitle("Підтвердження видалення");  
 confirmation.setHeaderText(null);  
 confirmation.setContentText("Ви впевнені, що хочете видалити це замовлення?");  
  
 if (confirmation.showAndWait().get() == ButtonType.OK) {  
 DB db = new DB();  
 try (Connection conn = db.getDbConnection();  
 PreparedStatement stmt = conn.prepareStatement(  
 "DELETE FROM Orders WHERE id = ?")) {  
  
 stmt.setInt(1, order.getId());  
 stmt.executeUpdate();  
  
 showInfoAlert("Успіх", "Замовлення успішно видалено");  
 loadOrdersData();  
 } catch (SQLException e) {  
 showErrorAlert("Помилка", "Не вдалося видалити замовлення");  
 }  
 }  
 }  
  
 private void loadOrdersData() {  
 ordersData.clear();  
 DB db = new DB();  
 try (Connection conn = db.getDbConnection();  
 PreparedStatement stmt = conn.prepareStatement(  
 "SELECT o.id, c.PIB as client\_name, p.name\_product as product\_name, " +  
 "o.quantity, o.total " +  
 "FROM Orders o " +  
 "JOIN Clients c ON o.id\_client = c.id " +  
 "JOIN Product p ON o.id\_product = p.id ");  
 ResultSet rs = stmt.executeQuery()) {  
  
 while (rs.next()) {  
 ordersData.add(new Order(  
 rs.getInt("id"),  
 rs.getString("client\_name"),  
 rs.getString("product\_name"),  
 rs.getInt("quantity"),  
 rs.getDouble("total")  
 ));  
 }  
 } catch (SQLException e) {  
 showErrorAlert("Помилка", "Не вдалося завантажити дані замовлень");  
 }  
 }  
  
 private void setupSearch() {  
 FilteredList<Order> filteredData = new FilteredList<>(ordersData, p -> true);  
  
 searchField.textProperty().addListener((observable, oldValue, newValue) -> {  
 filteredData.setPredicate(order -> {  
 if (newValue == null || newValue.isEmpty()) {  
 return true;  
 }  
  
 String lowerCaseFilter = newValue.toLowerCase();  
 return order.getProductName().toLowerCase().contains(lowerCaseFilter) ||  
 order.getClientName().toLowerCase().contains(lowerCaseFilter);  
 });  
 });  
  
 SortedList<Order> sortedData = new SortedList<>(filteredData);  
 sortedData.comparatorProperty().bind(ordersTable.comparatorProperty());  
 ordersTable.setItems(sortedData);  
 }  
  
 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();  
 }  
  
 public static class Order {  
 private final int id;  
 private final String clientName;  
 private final String productName;  
 private final int quantity;  
 private final double total;  
  
 public Order(int id, String clientName, String productName, int quantity,  
 double total) {  
 this.id = id;  
 this.clientName = clientName;  
 this.productName = productName;  
 this.quantity = quantity;  
 this.total = total;  
 }  
  
 public int getId() { return id; }  
 public String getClientName() { return clientName; }  
 public String getProductName() { return productName; }  
 public int getQuantity() { return quantity; }  
 public double getTotal() { return total; }  
 }  
}