МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА КРОС-ПЛАТФОРМНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ «БІБЛІОТЕКА»**

Виконав студент 2 курсу, групи 8.1219-з
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
І.В. Панасенко
(ініціали та прізвище)
Керівник завідувач кафедри фундаментальної математики, доцент, д.т.н. Гребенюк С. М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення

(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри
кафедри програмної інженерії,
к.ф.–м.н., доцент
_____ Лісняк А.О.
(підпис)

« ____ » _____ 2020 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Панасенко Ілля Валерійович
(прізвище, ім'я та по батькові)

1. Тема роботи розробка крос-платформної інформаційної системи «Бібліотека»

керівник роботи завідувач кафедри фундаментальної математики, доцент, д.т.н. Гребенюк С. М.
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » 05 2020 року № 577

2. Строк подання студентом роботи 29.11.2020

3. Вихідні дані до роботи: 1. Постановка задачі.
2. Перелік літератури.
3. Перелік задач до розв'язання.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
1. Постановка задачі.
2. Ознайомлення з теоретичними відомостями.
3. Розробка крос-платформного додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	21.08.2020	
2.	Збір вихідних даних.	28.08.2020	
3.	Обробка методичних та теоретичних	26.09.2020	
	джерел.		
4.	Розробка першого розділу.	03.10.2020	
5.	Розробка другого розділу.	15.10.2020	
6.	Розробка третього розділу.	27.10.2020	
6.	Оформлення та нормоконтроль	30.11.2020	
	кваліфікаційної роботи.		
7.	Захист кваліфікаційної роботи.	9.12.2020	

Студент _____ І.В. Панасенко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.М. Гребенюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.В. Кудін
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка крос-платформної інформаційної системи «Бібліотека»: 48 с., 19 рис., 8 джерел, 2 додатка.

JAVA, БАЗА ДАНИХ, XML, JAXB, SCENE BUILDER

Об’єкт дослідження – процес розробки крос-платформної інформаційної системи «Бібліотека».

Мета роботи: розробити систему для автоматизації обліку книг.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проектування, конструювання програмних забезпечень.

У кваліфікаційній роботі розглянуто основні етапи розробки автоматизованої системи обліку книг.

При розробці системи був проведений аналіз предметної області, обрана архітектура і платформа реалізації, розроблена функціональна і інформаційна модель системи, спроектована архітектура системи, реалізована інформаційна система на базі JavaFX та мови програмування Java, оформлена супровідна документація. В результаті роботи отримана автоматизована система обліку книг.

ABSTRACT

Master's qualifying paper «Development of Cross–Platform Information System «Library»: 48 pages., 19 figures, 8 references, 2 supplements.

JAVA, DATABASE, XML, JAXB, SCENE BUILDER

Object of the study – is the process of development a cross–platform information system «Library».

Aim of the study – to develop a system for automation of bookkeeping.

Methods of research – methods of collecting and analyzing software requirements, methods of modeling, designing, designing software.

In the qualification work the main stages of development of the automated system of book accounting.

During the development of the system the subject area was analyzed, the architecture and implementation platform were selected, the functional and information model of the system, the information system based on JavaFX and Java programming languages was implemented, and accompanying documentation was prepared. As a result of work the automated system of the account of books is received.

ЗМІСТ

ВСТУП

Актуальність дослідження визначається тим, що у сфері обліку книг для домашнього використання, існує багато різноманітних автоматизованих систем. Однак, при більш детальному ознайомленні існуючими системами обліку книг було виявлено, що функціоналу недостатньо або забагато, не завжди існує підтримка різних платформ, системи налагоджені для роботі лише на конкретних операційних системах. Отже, в наш час особливу актуальність має розробка системі для обліку книг для домашнього використання зі зрозумілим інтерфейсом, необхідними функціями, можливостями, та не перевантаженим надмірним функціоналом.

Основна мета розробки такої системі полягає в наділенні її оптимальнім набором функціоналу та можливістю без складнощів працювати на різних платформах.

Для досягнення такої мети потрібно виконати наступні задачі:

– збір інформації для написання спеціальних вимог до системи (технічного завдання);

– створення плану стадій та етапів розробки;

– розробка концепції та архітектури для реалізації інформаційної системи;

– проектування інформаційної системи;

– реалізація інформаційної системи;

– оформлення документації до проекту.

В результаті виконання роботи планується мати крос-платформну систему, яка проста в використанні, інтуїтивно зрозуміла та наділена виключно необхідним функціоналом

1 ОПИС ОСНОВНИХ ВАРІАНТІВ РЕАЛІЗАЦІЇ ПРОЕКТУ

База даних – набір впорядкованих даних, що зберігаються, зазвичай, в електронному вигляді. Для керування даними створено системи. за допомогою яких відбувається адміністрування, системи керування базами даних (СКБД). Ці системи разом з даними, а також додатками, пов’язані з ними, називаються системами баз даних, або, базами даних.

Існує багато різних баз даних, які використовують в залежності від потреб в тій чи іншій ситуації.

Елементи в реляційної базі даних організовані у вигляді набору таблиць за допомогою стовпців і рядками. Така структура забезпечує найбільш гнучкий доступ до структурованої інформації.

Інформація в об'єктно-орієнтованій базі даних представляється у формі об'єкта, як в об'єктно-орієнтованому програмуванні.

Розподілена база даних складається з двох або більше файлів, розташованих на різних вузлах. При використанні такої бази, вона може зберігатися на різних комп’ютерах, які при цьому можуть бути розташовані на різних місцях.

Сховища даних, які є централізованими репозиторіями для даних, являють собою тип бази даних, спеціально створеної для швидкого виконання запитів і аналізу.

База даних NoSQL, або нереляційна база даних, дозволяє працювати з неструктурованими або слабо структурованими даними (на відміну від реляційної бази даних). NoSQL набирає популярність по мірі поширення та ускладнення веб-додатків.

1.1 Обґрунтування використання XML в базах даних

Існує ряд причин для використання XML для зберігання даних.

Зокрема, для XML вони включають:

– зберігається структура та порядок документу;

– вхідними даними можуть бути файли XML, та щоб запобігти подвійному моделюванню даних;

– XML дуже добре підходить для розріджених даних, глибоко вкладених даних і змішаного вмісту (наприклад, тексту з вбудованими тегами розмітки);

– файл бази даних XML можна зрозуміти та прочитати навіть не маючи досвіду в базах даних;

– метадані часто доступні в форматі XML;

– семантичні веб-дані доступні в форматі RDF / XML;

Стів О'Коннелл наводить одну єдину причину використання XML при творені баз даних: все більш широке використання XML для передачі даних означає, що «дані витягуються з баз даних і розміщюються в документи XML і навпаки». Це може бути більш ефективним (з точки зору витрат на перетворення) і простішим – зберігати дані у форматі XML.

1.2 Крос-платформність та вибір мови для написання додатку

Крос-платформність – властивість програмного забезпечення працювати більш ніж на одній програмній (в тому числі – операційній системі) або апаратній платформі; технології, що дозволяють досягти такої властивості. Крос-платформність дозволяє суттєво скоротити витрати на розробку нового або адаптацію існуючого програмного забезпечення [1].

В залежності від засобів реалізації розподілення можна виокремити крос-платформність на рівні мов програмування (а також інструментів таких мов:

компіляторів та редакторів зв'язків), апаратної платформи та операційної системи, середовища виконання.

Існують декілька методів та прийомів для досягненні необхідного рівня крос-платформності.

Крос-платформність на рівні мов програмування досягається створенням додатку без орієнтування на платформу. Багатоплатформними на даний момент є більшість сучасних високорівневих мов програмування, до таких мов відносяться C, C++, Pascal крос-платформні мови на рівні компіляції, бо для них є компілятори під різні платформи.

Крос-платформність на рівні редакторів зв'язків досягається реалізацією для різних платформ крос-платформних бібліотек, які, незалежно від платформи, реалізують інтерфейс, в тому числі стандартизованих бібліотек. До прикладу, багато бібліотек мови Сі стандартизовані. Існує також велика кількість нестандартних крос-платформних бібліотек: Qt, SDL, FLTK, GTK+, STL, Boost, OpenCL, OpenGL, OpenAL.

Крос-платформність на рівні середовищ виконання забезпечується реалізацією ними можливостей, які необхідні програмам незалежно від платформи. Для цього існує поняття «контракт» – це декларований набір таких можливостей, так званий обов’язок, який покладається на середовище, щоб забезпечити виконання програми. Ці обов'язки реалізуються через інтерпретатор, протоколи, файлові потоки, системні виклики, віртуальну машину тощо.

Java і C# – крос-платформні мови на рівні виконання, це означає, що їх виконавчі файли без попередньої перекомпіляції можна запускати на різних платформах.

PHP, ActionScript, Perl, Python, Tcl і Ruby – крос-платформні інтерпретовані мови, їх інтерпретатори існують для багатьох різних платформ.

Крос-платформність на апаратному рівні досягається реалізацією форматом їх представлення та однакових машинних команд, механізмами

адресації пам'яті, системами переривань тощо. Шляхом віртуалізації відповідних ресурсів та механізмів дана модель може бути реалізована.

Тож, цілком обґрунтовано була вибрана Java, як мова написання додатку кваліфікаційної роботі, яка є крос-платформною мовою на рівні виконання, що в свою чергу дає змогу запускати програмний код, який створено на даній мові на будь-якому пристрої та на будь-якій платформі, що є однією з основних вимог даної роботи.

1.3 Детальний огляд мові програмування Java

Java – жорстко типізована об'єктно-орієнтована мова програмування, розроблена компанією Sun Microsystems (в подальшому придбаної компанією Oracle). Розробка ведеться співтовариством, організованим через Java Community Process, мова і основні реалізації її технології поширюються за ліцензією GPL. Права на торговельну марку належать корпорації Oracle [3].

Найбільш поширеною та однією з мов, що найшвидше розвивається, є Java. Вже багато років вона входить до топ-3 мов для написання додатків. Java замислювалася як універсальна мова програмування, яку можна буде застосовувати для будь-яких завдань будь-якого рівня. На теперішній час створено багато версій мови Java. Поточною версією є Java 12, що вийшла в березні 2019 року. Java перетворилася з просто універсальної мови в цілу платформу і екосистему, яка об'єднує різні технології, що використовуються при вирішенні цілого ряду різних задач: від створення десктопних додатків до написання сервісів та навіть веб-порталів. Java активно застосовується для створення програмного забезпечення для цілого ряду пристроїв: звичайних ПК, планшетів, смартфонів, пристроїв розумного дому, та інше. Одна з найпоширеніших платформ для смартфонів є ОС Android, яка використовує Java, як мову написання додатків.

Для розробки на даній мові програмування створено комплект інструментів, який називається JDK, або Java Development Kit. Однак, незважаючи на одну мову програмування в даному випадку, існують різноманітні реалізації даного продукту.

Найбільш розповсюджені це Oracle JDK і OpenJDK. На відміну від Oracle JDK, якою займається виключно Oracle, програмним продуктом OpenJDK займається багато користувачів, включно з Oracle. Найбільші відмінності полягають в ліцензуванні. Згідно з угодою при купівлі даного продукту, Oracle дозволяє використання Oracle JDK лише для некомерційних цілей, таких, як навчання в вищому навчальному закладі, чи для розробки, тестування та демонстрації створених додатків. Однак, якщо необхідна підтримка, необхідно буде придбати комерційну версію продукту. В свою чергу, OpenJDK абсолютно безплатна для будь-якого виду користування.

Функціонально ці дві системи практично нічим не відрізняються, але якщо подивитися на них з точки зору продуктивності, тут можна побачити відмінності. Oracle JDK працює значно швидше, ніж OpenJDK. Також розробники відмічають, що Oracle JDK працює стабільніше, аніж його безкоштовний аналог OpenJDK.

1.3.1 Віртуальна машина Java

Не зважаючи на те, що Java є мовою високого рівня, програма яка виконана на ній, не може бути відразу виконана. Її спочатку треба відкомпілювати, тобто перетворити у послідовність машинних команд – об'єктний модуль.

Але він також не може бути виконаний одразу: об'єктний модуль треба ще скомпонувати з бібліотеками, що використовувалися при створенні додатку, і дозволити перехресні посилання між секціями об'єктного модуля, отримавши в результаті завантажувальний модуль, що є вже повністю готовою до виконання програмою.

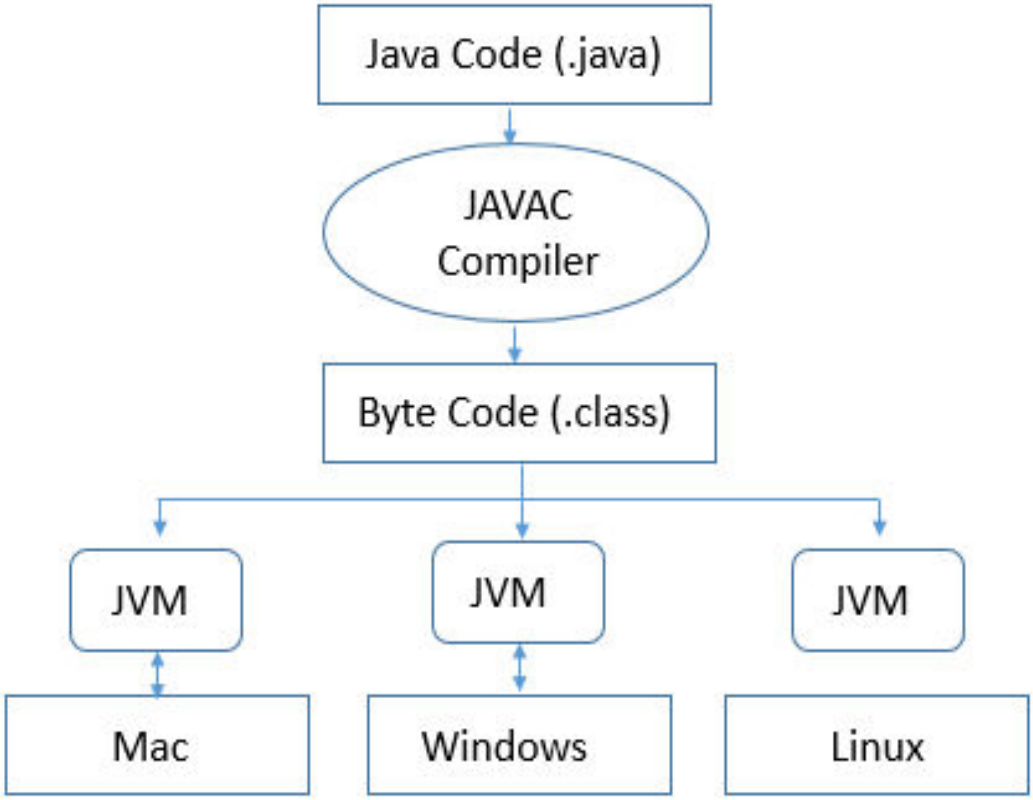

Рисунок 1.1 – Структура JVM

Головна особливість технології Java – програма компілюється відразу в машинні команди так званої віртуальної машини Java (JVM, JavaVirtualMachine). Сукупність команд разом з системою їх виконання називається віртуальною машиною Java. Віртуальна машина Java повністю стекова, це означає, що потрібна адресація осередків пам'яті і велика кількість регістрів. Тому команди JVM короткі та мають довжину 1 байт, у зв’язку з чим команди JVM називають байт-кодами (bytecodes), хоча зустрічаються команди довжиною 2 і 3 байти. Якщо поглянути на статистичні данні, середня довжина команди складає 1,8 байту. В специфікації віртуальної машини Java (VMS,

VirtualMachineSpecification) знаходиться повний опис команд та всієї архітектури JVM

Важливою особливістю Java є стандартні функції, що викликаються в програмі, та підключаються до неї тільки на етапі виконання, при цьому не включаються в байт-коди. Для мови Java характерна така річ, як динамічна компоновка (dynamicbinding). Це зменшує обсяг скомпільованого коду.

На першому етапі написана на мові Java програма, трансформується компілятором в байт-коди. При цьому компіляція не залежить від архітектури пристрою, чи процесора. Вона може бути виконана після написання програми один раз. Байт-коди можуть записуватися в одному або декількох файлах, можуть передаватися по мережі або зберігатися в зовнішній пам'яті. Файли з байт кодами мають невеликий розмір. Отримані в результаті компіляції байт-коди можна буде виконувати на будь-якому комп'ютері, де встановлено JVM. Так застосовується один з головних принципів Java «Writeonce, runanywhere» – «Написано один раз, виконується де завгодно»[2].

1.3.2 Крос-платформність Java додатків

Через можливу несумісність програмних інтерфейсів різних графічних оболонок та операційних систем користувача недостатньо просто перекомпілювати вже написаній код.

Програма, що написана на мові Java, компілюється в двійковий модуль, що в свою чергу складається з команд постпроцесора Java, який для кожної платформи свій. Такий модуль містить байт-код, призначений для виконання Java-інтерпретатором.

Якщо треба створити крос-платформний додаток, який має працювати на декількох машинах, існує можливість не компілювати його декілька разів. Можливо відкомпілювати та налагодити програму на найбільш зручній

платформі для розробника. Як результат отримуємо крос-платформний додаток, який може працювати в будь якому середовищі, яке підтримує Java постпроцесор.

Від платформи також залежить внутрішня реалізація класів. Для використання можливостей цих бібліотек всі необхідні завантажувальні модулі пропонуються у готовому вигляді у віртуальній машині Java. Наприклад, для Windows, як операційної системи, поставляються бібліотеки для динамічного завантаження DLL, з функціональністю стандартних Java класів.

Програмісти не повинні турбуватися про відмінності створюваного програмного інтерфейсу конкретних операційних систем. Все це дає змогу створювати додатки, які не потребують зміни коду при перенесенні на різні платформи.

2 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

2.1 Основні аналоги проекту

BookSnake – повноцінна база даних, зі зрозумілим інтерфейсом, яка дає змогу зберігати інформацію про книги користувача. Особливість даної програми в тому, що вона може до облікового запису кожної книги приєднати файл цієї книги. Відкриття файлу відбувається у вікні програми. При додаванні облікового запису нової книги користувач вводить автора, назву книги, ключові слова.

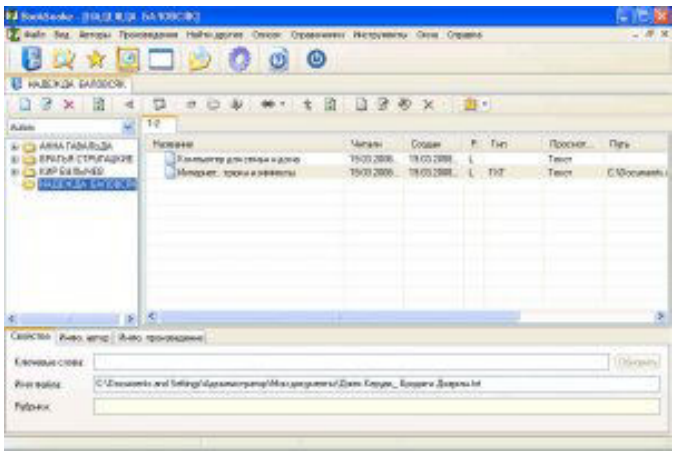

Рисунок 2.1 – Програма BookSnake

Unicat – програма, яка є універсальним каталогізатором, за допомогою якого можливо зберігати облікові записи не лише книг, але й фільмів, використовувати як телефонний довідник, та інше. Можливо створення декількох рівнів каталогів. Також важлива особливість є в тому, що програма може самостійно аналізувати обраний користувачем об’єкт. Програма має змогу експорту та імпорту вмісту.

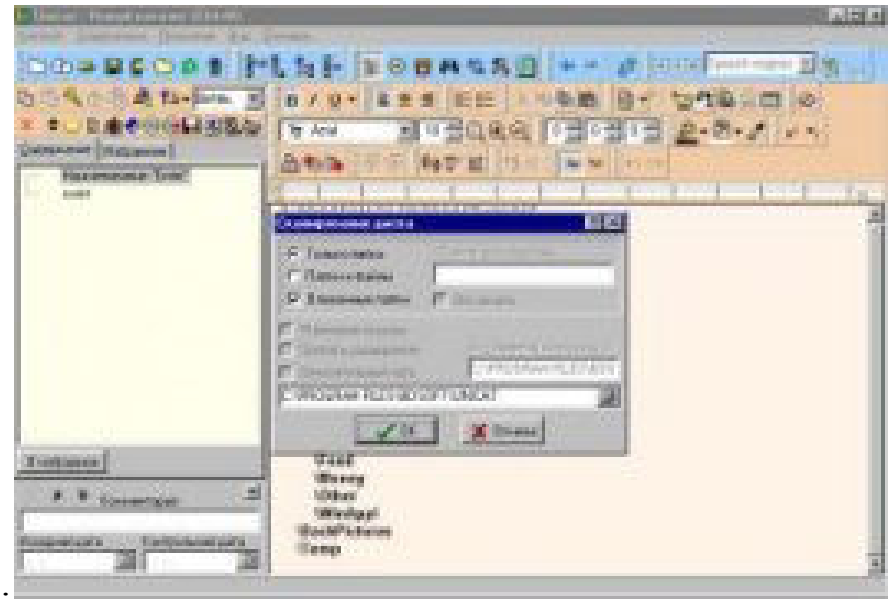

Рисунок 2.2 – Програма Unicat

Отже, керуючись вже створеними додатками для керування та обліку книг у домашній бібліотеці обираємо функціонал нашого створюваного додатку. Перш за все повинно бути реалізовано додавання облікових записів книг з такими полями, як назва книги, автор, жанр книги, видавництво, можливість додавання за бажанням коментарів чи опису книги. Також необхідно реалізувати можливість додавання через спеціальну кнопку шляху, що вказує де лежить книга на конкретному комп'ютері з можливістю, за бажанням користувача, показати місцезнаходження книги шляхом відкриття провідника та підсвічуванням обраної книги.

2.2 Вибір середовища програмування

Для створення проекту було обрано середовище розробки IntelliJ IDEA – продукт компанії JetBrains. Даний інструмент дозволяє використовувати чимало мов для програмування, наприклад, Java, Python – для Back end, та JavaScript для Front end та багато інших.

Історія створення даного середовища розпочинає свій шлях з 2001 року, саме тоді народилася первинна версія середовища програмування у Java. Шалене визнання йому приносить набір засобів з рефакторінга. З того часу стало значно легше робити реорганізацію початкового коду програм. Зростає швидкість виконання операцій, що допомагає кожному програмісту приділити більше уваги функції, ніж нескінченним розрахункам. Тепер ті завдання, що раніше потребували чимало часу, можна виконати миттєво у цьому середовищі.

Розробка графіки інтерфейсів стала можливою вже у шостій версії. IntelliJ IDEA – середовище, що має широкий вибір засобів не тільки для створення програмного забезпечення, а також має здатність сполучатись з такими інструментами як JUnit, Apache Ant, CVS, Subversion. Через шість років після створення першої версії розробники презентували плагін для мови Ruby.

Дев'ята версія існує як Community Edition і Ultimate Edition. Тепер можна вільно користуватися та розповсюджувати версію Community, Ultimate – платна версія. Спільне у них є те, що вони легко інтегруються іншими популярними системами, а також із Scala, Kotlin, Groovy. Відмінність комерційної версії від безкоштовної – підтримка Java EE, UML-діаграм, можливість розрахувати покриття коду, управління іншими версіями, мовами, фреймворками.

2.3 Основні інструменти керування та складання проекту

Мавен(Maven) – засіб для з'єднання Java коду у єдиний проект, а саме – компіляція, створення таких важливих речей як jar, дистрибутив, генерація усіх документів. Для збирання проектів через командний рядок необхідно записувати в bat / sh скрипт. В залежності від платформи можуть бути інші скрипти та інструменти. Ant і Maven – найпопулярніші засоби для збірки. Але Мавен має більше переваг:

– Windows чи Linux – немає різниці, бо у будь-якій операційній системі можна зібрати проект, це буде один той самий документ;

– Maven управляє дуже складними залежностями: проект використовує бібліотеки, а ті побічні бібліотеки теж користуються різними версіями. Інструмент вирішує суперечності між різними версіями, якщо необхідно – здійснює перехід на новий рівень цих залежностей;

- continuous integration – командний рядок може створити збірку на сервері автоматично;

- дивовижно проста інтеграція з іншим середовищем. Відкриття та збірка проектів за допомогою цього інструменту не потребує налаштувань, його можна використовувати надалі у проекті. Maven зберігає усі необхідні налаштування навіть з різних середовищ. Це перестрахування від будь-якого дублікату, помилки. Єдиний файл і є остаточним матеріалом для подальшої роботи;

- декларативний опис проекту.

Як було зазначено раніше, система створена з використанням інструменту для роботи з проектами Maven.

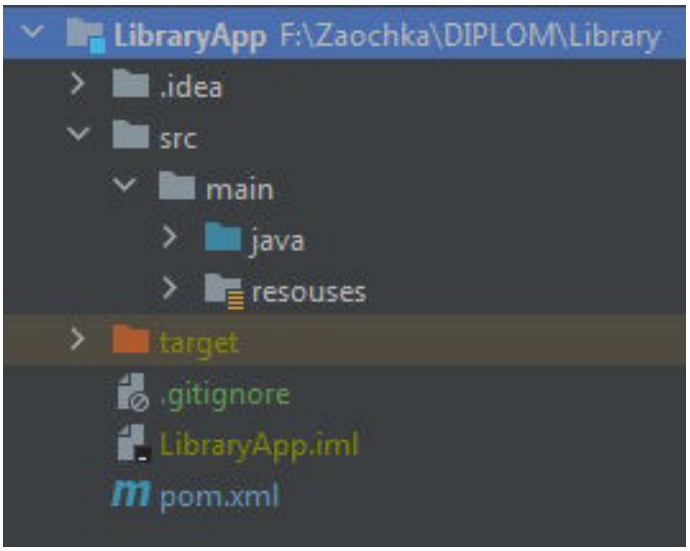

Рисунок 2.3 – Структура проекту

Каталог проекту складається з:

- .idea – містить файли середовища програмування;

- .src – є найбільш важливим каталогом проекту Maven. Все, що є частиною артефакту, будь-то jar або war, присутнє тут.

Його підкаталоги:

- src / main / java – вихідний код Java для артефакту;
- src / main / resources – файли конфігурації, мультимедійні файли та інше;
- target – містить службові файли, які утворюються при компілюванні проекту;
- gitignore – містить перелік файлів, які не потребують завантаження в систему контролю версій GitHub;
- libraryApp.iml – файли конфігурації створюваної програми;
- pom.xml – визначає залежності та модулі під час життєвого циклу збірки проекту Maven.

Отже, проект містить багато підкаталогів та файлів які підтримують функціонування системи в цілому.

2.4 Метод зберігання даних

Для зберігання даних у базі даних нашого додатку був обраній метод запису у XML файл. Для цього біла використана бібліотека JAXB (Java Architechture for XML Binding). За допомогою цієї бібліотеки можливо записати наші данні у файл XML, який потім можна буде відкрити даною програмою. Бібліотека JAXB вже присутня в JDK, тому додатково нічого не встановлювалось.

Бібліотека JAXB має в своєму арсеналі дві функції, це маршалювання (marshalling), та демаршалювання (unmarshalling). Перша відповідає за запис об’єктів у XML файл, друга за зворотнє перетворення файлу XML у Java об’єкти.

3 СТРУКТУРА ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОЕКТУ

3.1 Розробка структури проекту

Архітектура проекту передбачає використання XML файлу як сховища даних, що будуть використовуватися як для зберігання даних при закритті додатку так і при перенесенні бази даних на іншу машину чи будь-який інший пристрій з підтримкою JVM.

При роботі додатку данні з файлу XML переносяться у список для роботи по додаванню чи редагуванню раніше створених облікових записів для книг. При закінченні роботи чи при користувацькому запиті данні будуть збережені у файлі XML зі списку, у якому відбувалася робота.

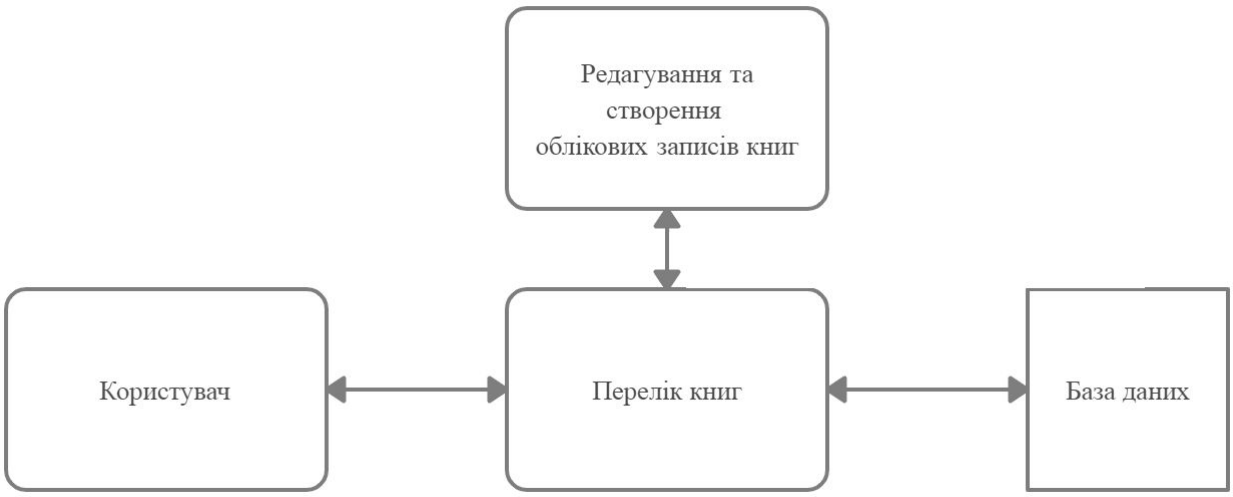

Рисунок 3.1 – Діаграма переміщення даних

На нижче приведеній UML діаграмі відображені всі класи, методи та поля, які будуть необхідні при створенні додатку.

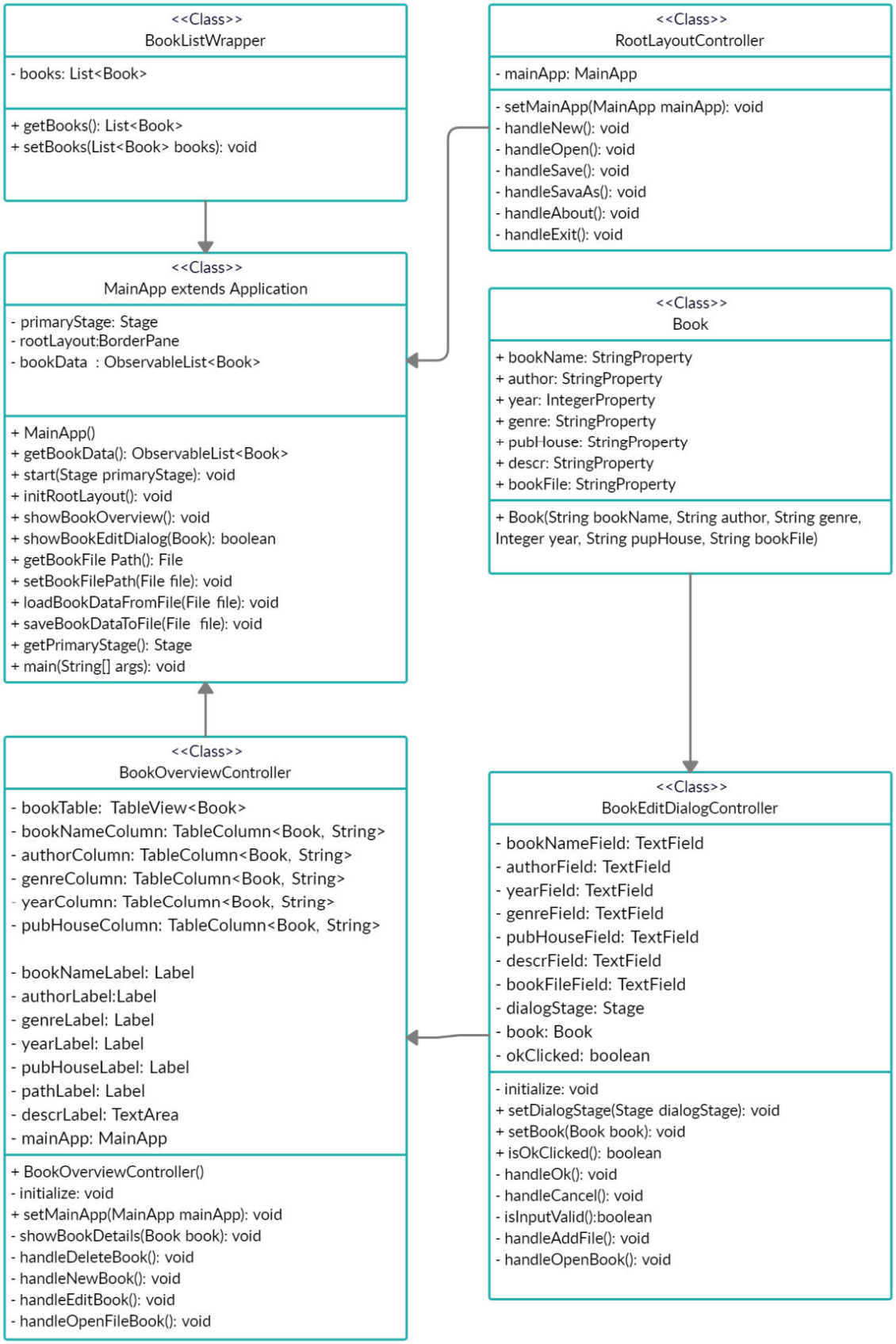

Рисунок 3.2 – UML діаграма додатку

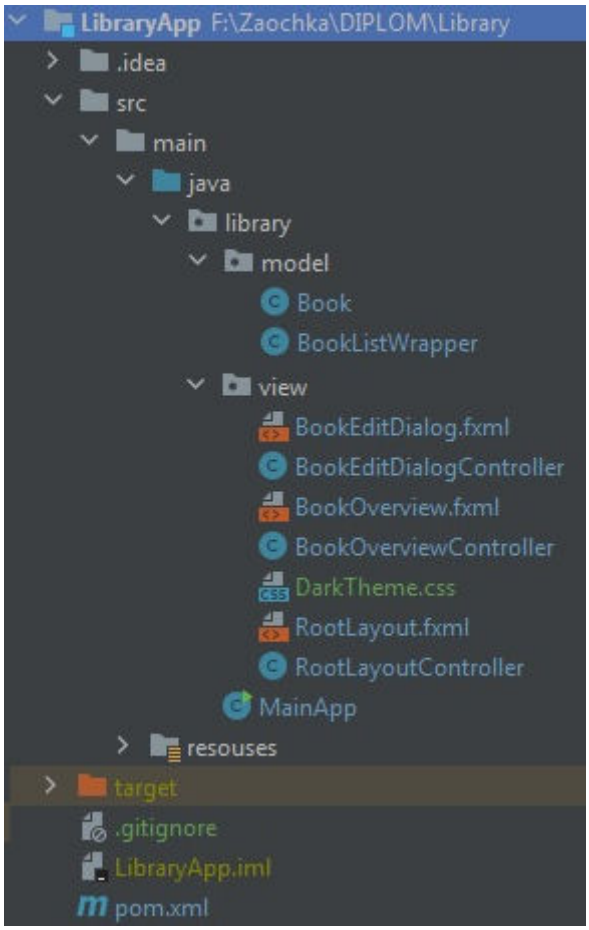

Рисунок 3.3 – Загальний вигляд проекту

Всі поля які необхідні для реалізації даного проекту знаходиться в класі-моделі Book.

```
private final StringProperty bookName;
private final StringProperty author;
private final IntegerProperty year;
private final StringProperty genre;
private final StringProperty pubHouse;
private final StringProperty descr;
private final StringProperty bookFile;
```

Рисунок 3.4 – Поля ініціалізації класу-моделі Book

При реалізації проекту використано саме StringProperty та IntegerProperty, бо у JavaFX використовують, переважно, Properties якщо мова йде про класи-

моделі. Property слідкує за змінами, наприклад у bookName чи інших. Цей функціонал допомагає нам зберегти синхронність даних та їх відображення.

Також в цьому класі створено конструктор Book, в який занесено всі змінні, без яких неможливе створення облікового запису на кожну з книг, та ініціалізовано його змінні як null.

```
public Book() {
    this( bookName: null,  author: null,  genre: null,  year: 0,  pupHouse: null,  bookFile: null);
}

/**
 * Constructor with some initial data.
 */
public Book(String bookName, String author, String genre, Integer year, String pupHouse, String bookFile) {
    this.bookName = new SimpleStringProperty(bookName);
    this.author = new SimpleStringProperty(author);
    this.genre = new SimpleStringProperty(genre);
    this.year = new SimpleIntegerProperty(year);
    this.pubHouse = new SimpleStringProperty(pupHouse);
    this.descr = new SimpleStringProperty( initialValue: "");
    this.bookFile = new SimpleStringProperty(bookFile);
}
```

Рисунок 3.5 – Конструктор класу-моделі Book

Також присвоєні полям гетери та сетери, щоб будо можливо звертатися до цього класу з інших класів.

3.2 Опис інтерфейсу проекту

Перше, що зустрічає користувача при взаємодії з даним програмним продуктом, є основне вікно програми, яке складається зі стандартного меню зверху, де знаходяться інструменти для створення нової бібліотеки, зберігання поточної, відкриття раніше збереженої.

Також є пункт «Help», в якому можливо переглянути інформацію про автора проекту, та його версію.

Основне поле головного вікна безпосередньо займає таблиця з переліком доданих книг, та деякими основними характеристиками тих чи інших книг.

Наявна можливість сортування кожного стовбця за алфавітом від А до Я, та навпаки.

Поле з правого боку основного вікна програми є полем перегляду більш детальної інформації про кожну з раніше доданих книг, редагування раніше створених книг, чи створення нових.

Для перегляду інформації необхідно підсвітити в основному полі ту чи іншу книгу. Створення нового облікового запису для книги можливо натисканням кнопки «New» в даному полі.

Для редагування запису про раніше додану книгу необхідно натиснути кнопку «Edit», після чого відкриється вікно редагування.

Щоб видалити обліковій запис раніше доданої книги необхідно натиснути кнопку «Delete».

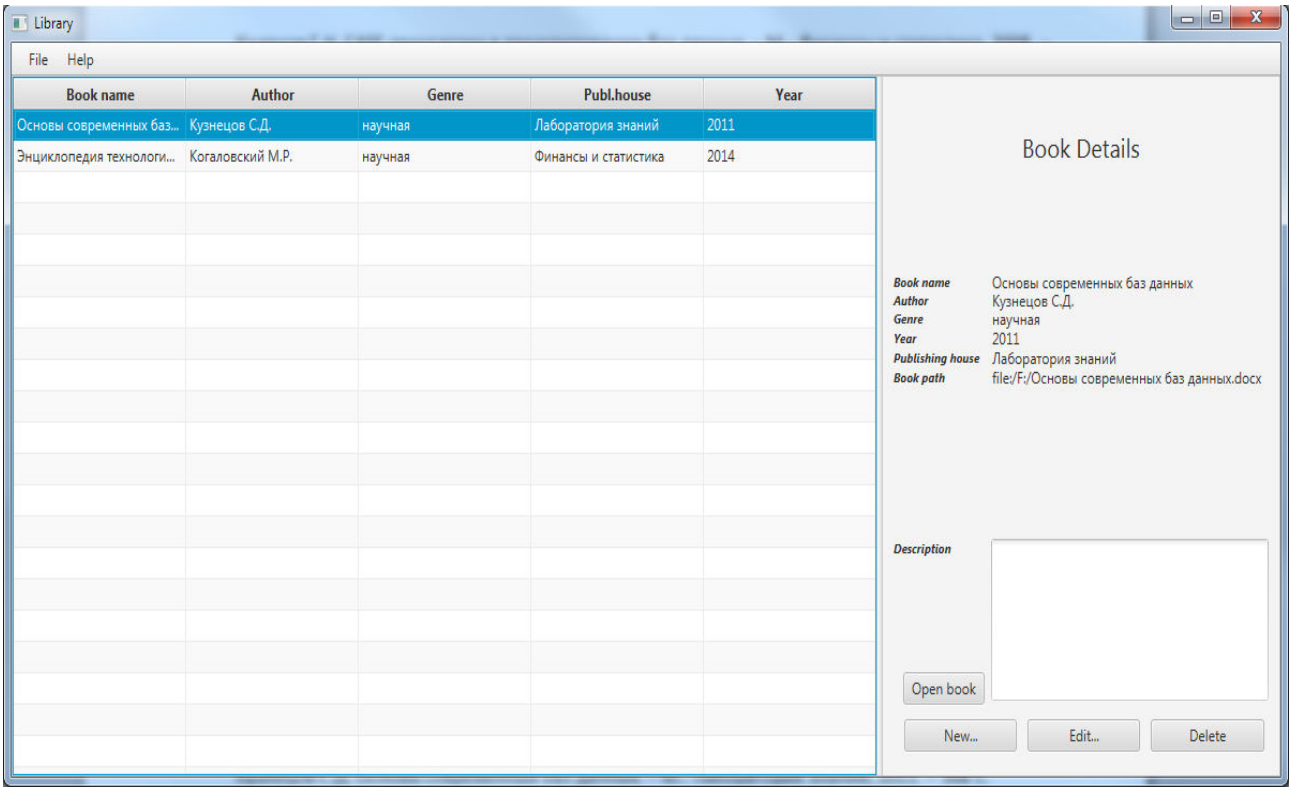

Рисунок 3.6 – Основне вікно додатку

3.3 Детальній опис додатку

3.3.1 Головна сторінка

За головну сторінку при роботі з програмою відповідає 4 класи, почнемо з BookOverview.fxml та його контролера BookOverviewController.

У файлі BookOverview.fxml мовою html описана розмітка сторінки. Для цього використано додаток JavaFX Scene Builder від корпорації Oracl.

У файлі BookOverviewController описані ті чи інші дії користувача на головній сторінці.

Анотацією @FXML позначені методи та поля, до яких fxml файл повинен мати доступ.

Після завантаження fxml файлу викликається метод initialize(), при цьому вже повинні бути ініціалізовані поля.

Метод setCellValueFactory(...) визначає поле, яке буде використано в класі Book для кожного конкретного стовпця з таблиці.

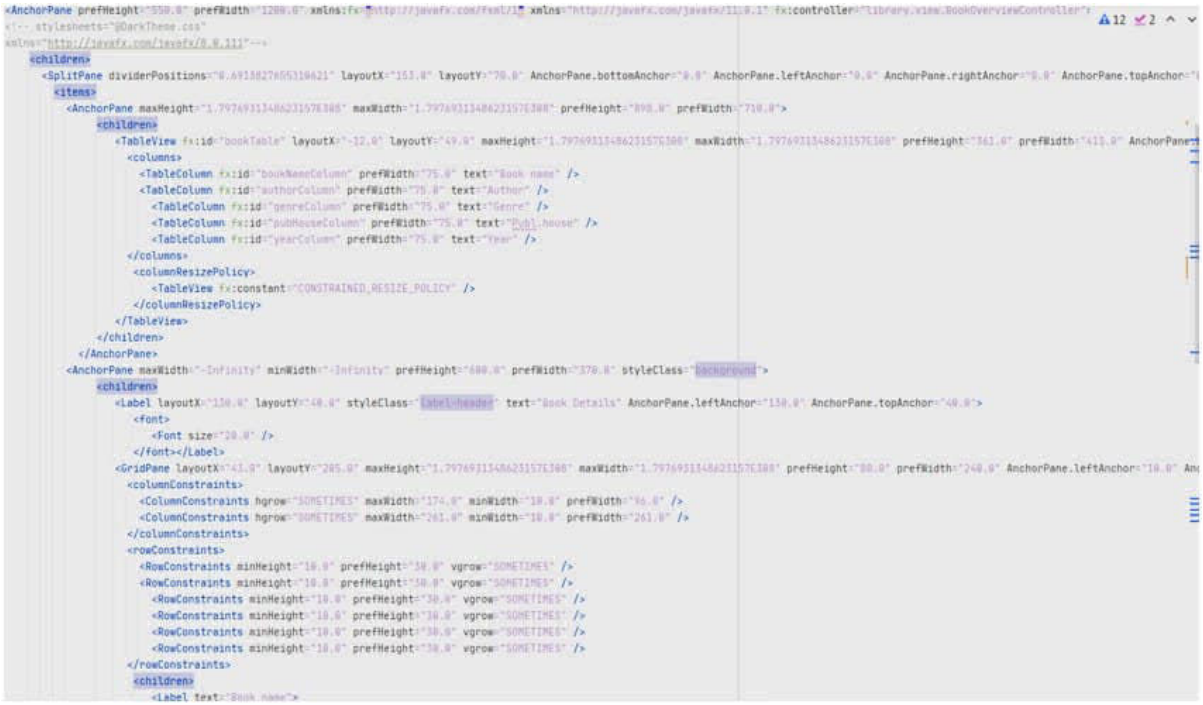

Рисунок 3.7 – Загальний вигляд класу fxml головної сторінки

За кнопки меню зверху та за таск бар відповідає RootLayout.fxml та його контролер RootLayoutController.

В RootLayout.fxml мовою html описана розмітка сторінки.

В RootLayoutController визначаються реакції програми на дії користувача, зокрема на такі кнопки меню, які створюють нову бібліотеку, відкривають раніше створену, зберігають. Кнопка «About» дозволяю отримати інформацію, в якій міститься відомості про програму, її версію та автора. Кнопка виходу «Exit» дозволяє завершити роботу програми.

Для відображення додаткової інформації про книгу користувач має змогу натиснути на запис у таблиці зліва, для того щоб програма відобразила цю додаткову інформацію контролер прослуховує зміни. Для цього в JavaFX існує інтерфейс ChangeListener. в якому знаходиться метод changed(...). В цьому методі міститься декілька параметрів. Це newValue, oldValue та observable.

```
@FXML
private void initialize() {
    // Initialize the person table with the two columns.
    bookNameColumn.setCellValueFactory(cellData -> cellData.getValue().bookNameProperty());
    authorColumn.setCellValueFactory(cellData -> cellData.getValue().authorProperty());
    genreColumn.setCellValueFactory(cellData -> cellData.getValue().genreProperty());
    yearColumn.setCellValueFactory(cellData -> cellData.getValue().yearProperty().asString());
    pubHouseColumn.setCellValueFactory(cellData -> cellData.getValue().pubHouseProperty());


    // Clear person details.
    showBookDetails(null);

    // Listen for selection changes and show the person details when changed.
    bookTable.getSelectionModel().selectedItemProperty().addListener(
            (observable, oldValue, newValue) -> showBookDetails(newValue));
}
```

Рисунок 3.8 – Слухач змін у класі контролері

Існує можливість відсортувати раніше введені користувачем данні за алфавітом чи за роком видання, та ін. Користувач за бажанням може додати опис книги в спеціальне поле на свій розсуд.

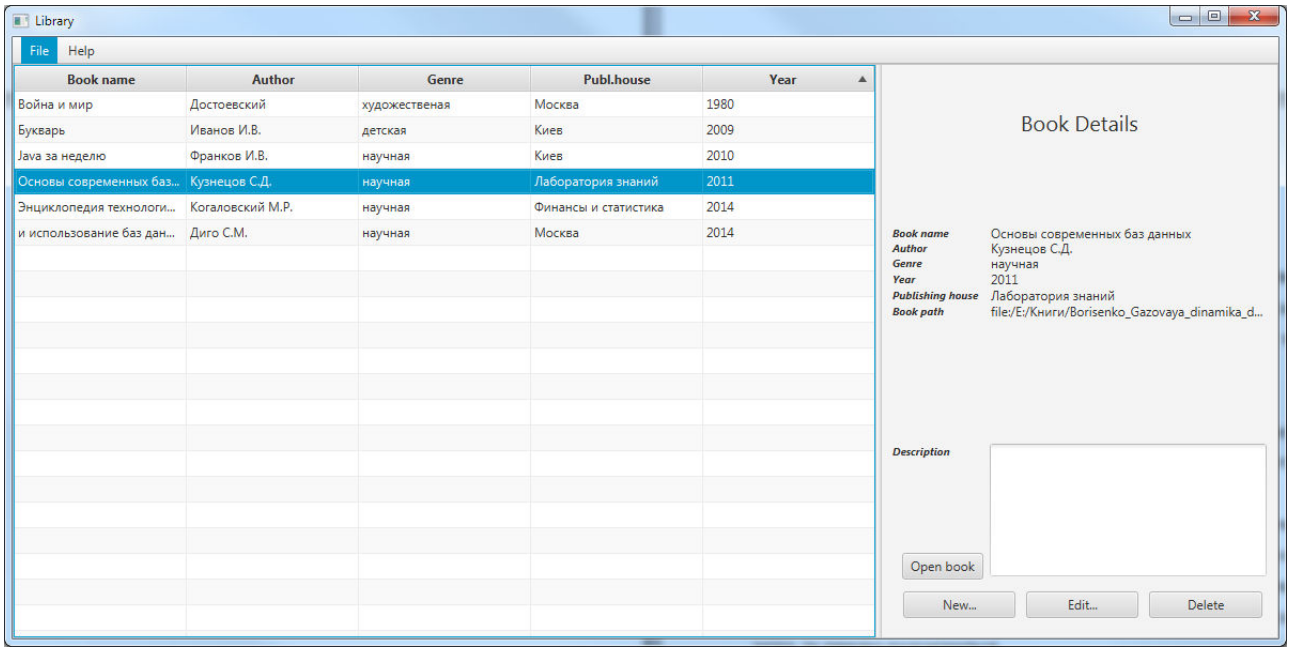

Рисунок 3.9 – Сортування облікових записів книг за роком видання

Також можливо відкрити папку, де лежить книга натиснувши «Open book», за цією дією відкривається папка провідника операційної системи, в якій буде підсвічено обрану користувачем книгу.

```
    @FXML
    private void handleOpenFileBook() throws IOException {
        Process p = new ProcessBuilder( ...command: "explorer.exe", "/select," + pathLabel.getText()).start();
    }

}
```

Рисунок 3.10 – Функціонал кнопки Open Book

3.3.2 Сторінка створення та редагування

За додавання нових облікових записів для книг, чи їх редагування відповідають 2 класи, це BookEditDialog.fxml та BookEditDialogController.

В BookEditDialog.fxml мовою html описана розмітка сторінки. BookEditDialogController визначає реакції програми на дії користувача.

Данні, що вводить користувач проходять перевірку на валідність.

Реалізовано функціонал кнопки «Add File». Програма зчитує місцезнаходження обраного користувачем файлу та записує його як параметр в обліковому записі стосовно кожної книги. Щоб згодом користувач зміг перейти до цього файлу шляхом натискання на кнопку «Open book».

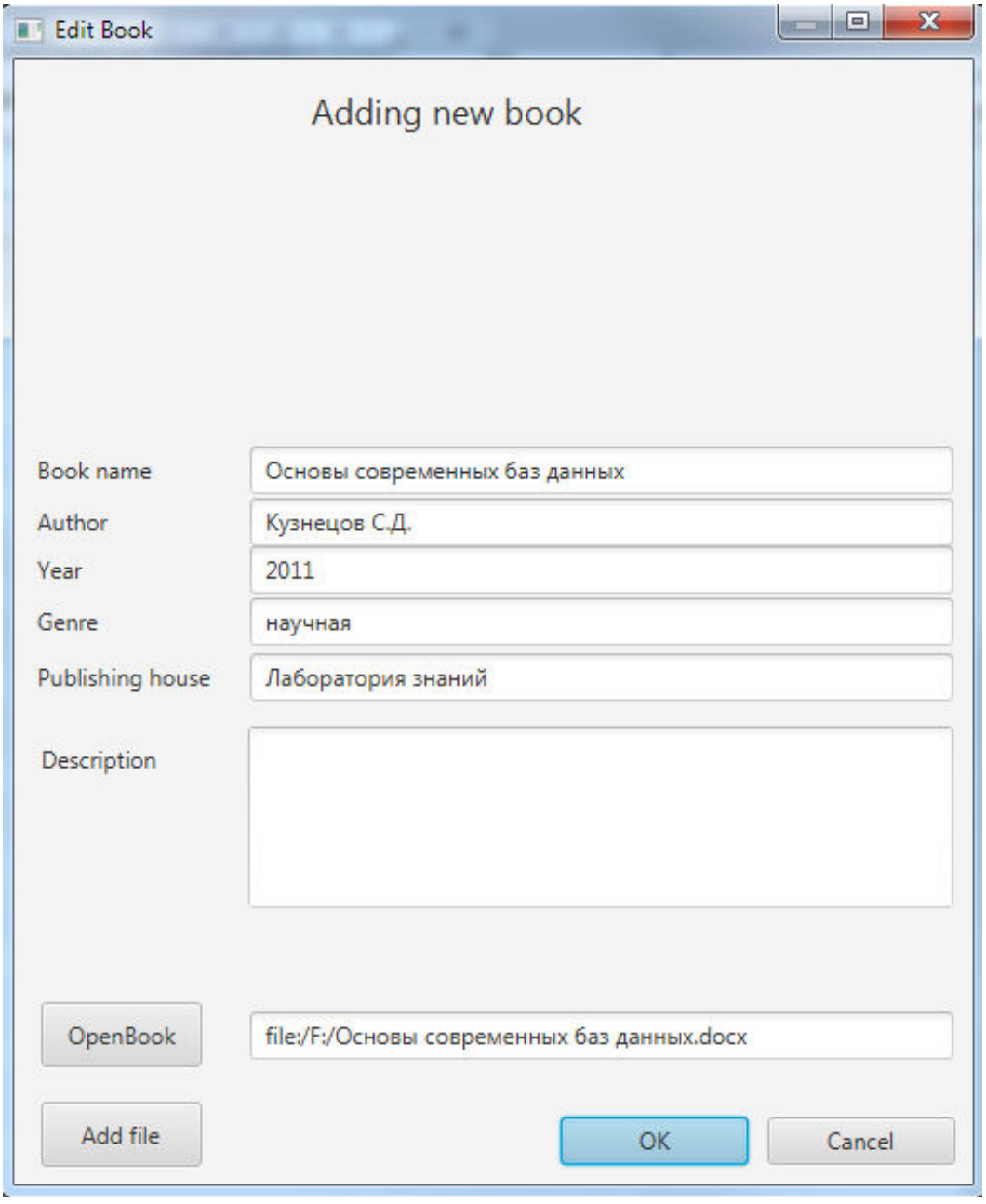

Рисунок 3.11 – Вікно створення та редагування

3.3.3 Перевірка валідності введених користувачем даних

Все що користувач пише в поля вводу, проходить перевірку на відповідність до очікування, що може міститися в кожній з них.

Наприклад, неможливо в місце для вводу року, колі була видана книга вписати дуже ранній рік, або рік, що ще не настав.

```
private boolean isInputValid() {
    String errorMessage = "";

    if (bookNameField.getText() == null || bookNameField.getText().length() == 0) {
        errorMessage += "No valid book name!\n";
    }
    if (authorField.getText() == null || authorField.getText().length() == 0) {
        errorMessage += "No valid author!\n";
    }
    if (genreField.getText() == null || genreField.getText().length() == 0) {
        errorMessage += "No valid genre!\n";
    }

    Calendar cal = Calendar.getInstance();

    if (Integer.parseInt(yearField.getText()) > cal.getWeekYear() ||
            Integer.parseInt(yearField.getText()) < 1500 ||
            yearField.getText() == null || yearField.getText().length() == 0) {
        errorMessage += "No valid year!\n";
    } else {
        // try to parse the year into an int.
        try {
            Integer.parseInt(yearField.getText());
        } catch (NumberFormatException e) {
            errorMessage += "No valid year (must be an integer)!\n";
        }
    }

    if (pubHouseField.getText() == null || pubHouseField.getText().length() == 0) {
        errorMessage += "No valid pub house!\n";
    }

    if (errorMessage.length() == 0) {
        return true;
    } else {
        // Show the error message.
        Alert alert = new Alert(AlertType.ERROR);
        alert.initOwner(dialogStage);
        alert.setTitle("Invalid Fields");
        alert.setHeaderText("Please correct invalid fields");
        alert.setContentText(errorMessage);

        alert.showAndWait();

        return false;
    }
}
```

Рисунок 3.12 – Метод перевірки валідності введених даних

3.3.4 Механізм зберігання даних бібліотеки

Запис даних у список BookData, в якому зберігається вся інформація про введені користувачем облікові записи книг відбувається за допомогою класу BookListWrapper, що знаходиться у каталозі model.

```
package library.model;

import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import java.util.List;

/**
 * Helper class to wrap a list of books. This is used for saving the
 * list of books to XML.
 */
@XmlRootElement(name = "books")
public class BookListWrapper {

    private List<Book> books;

    @XmlElement(name = "book")
    public List<Book> getBooks() {
        return books;
    }

    public void setBooks(List<Book> books) {
        this.books = books;
    }
}
```

Рисунок 3.13 – Клас, що виконує запис до списку

Дані, які користувач хоче зберегти у базу даних XML, знаходяться у списку BookData в класі MainApp. Бібліотека JAXB потребує, щоб зовнішній клас наших даних був поміченій анотацією @XmlRootElement. Тип змінної bookData є ObservableList, в який користувач не може втрутитись. Був створений клас-обгортка, в якому знаходиться колекція записів, і на якому

стоїть анотація @XmlRootElement. @XmlRootElement визначає ім'я кореневого елемента.

Щоб клас MainApp зберігав та зчитував данні нашого додатку він наділеній функціоналом.

```
public void saveBookDataToFile(File file) {
    try {
        JAXBContext context = JAXBContext
                .newInstance(BookListWrapper.class);
        Marshaller m = context.createMarshaller();
        m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

        // Wrapping our person data.
        BookListWrapper wrapper = new BookListWrapper();
        wrapper.setBooks(bookData);

        // Marshalling and saving XML to the file.
        m.marshal(wrapper, file);

        // Save the file path to the registry.
        setBookFilePath(file);
    } catch (Exception e) { // catches ANY exception
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText("Could not save data");
        alert.setContentText("Could not save data to file:\n" + file.getPath());

        alert.showAndWait();
    }
}
```

Рисунок 3.14 – Метод, який виконує зберігання бази даних

Файли зберігаються у файл XML, який можливо перенести на інший комп’ютер чи будь який інший пристрій, де встановлено створену програму по обліку книг. Запустивши її, та відкривши в ній цей файл, можливо повністю відтворити бібліотеку книг.

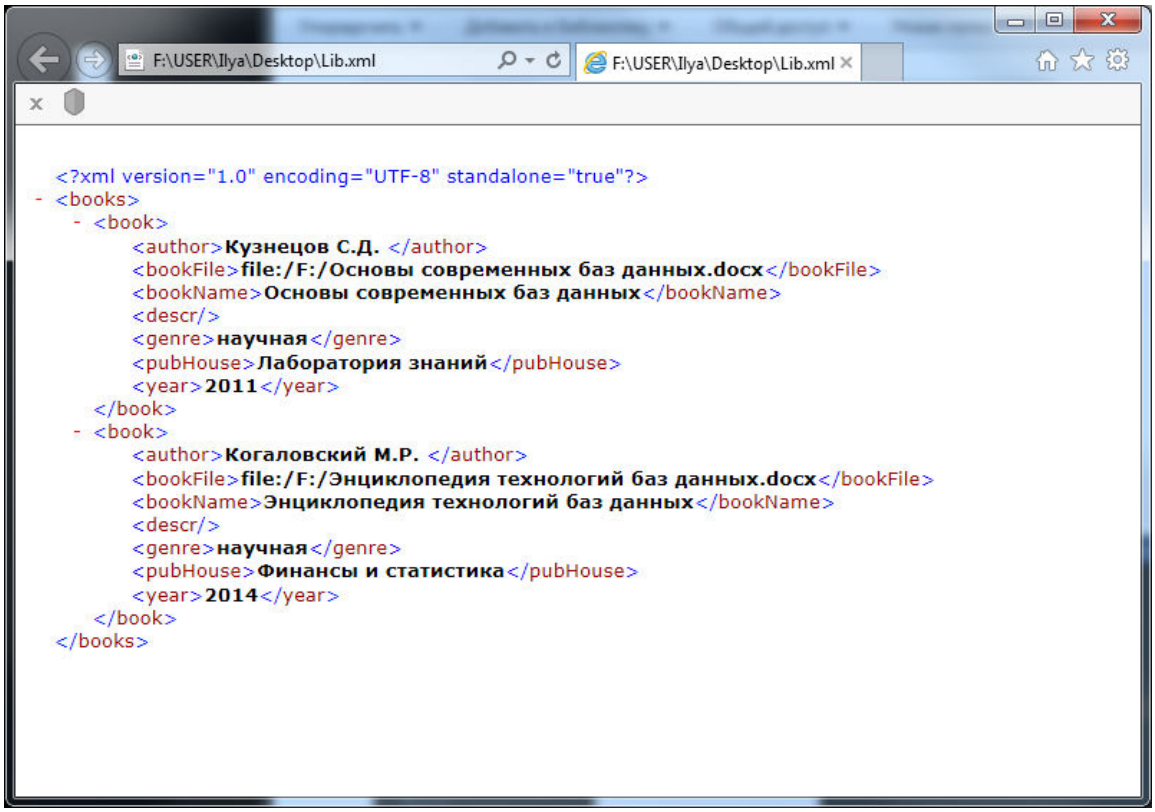

Рисунок 3.15 – Вигляд заповненої бази даних XML

ВИСНОВКИ

В результаті виконання роботи було розроблено інформаційну систему з автоматизації обліку книг. У відповідності з поставленими задачами технічного завдання були виконані наступні етапи створення технічного проекту:

- зроблено аналіз предметної області;
- обрана архітектура побудови і платформа реалізації системи;
- спроектована інформаційна система за допомогою мови програмування Java;
- розроблена UML діаграма та діаграма потоку даних;
- реалізований процес зберігання за допомогою бібліотеки JAXB.

Результатом реалізації цих задач є розробка на основі платформи JavaFX діючого прототипу системи автоматизованого обліку книг. Основними перевагами створеної інформаційної системи «Бібліотека» є її крос-платформність, легкість, та швидкість розгортання, та наявність функціоналу, який вигідно виокремлює її від інших існуючих аналогічних програмних засобів.

ПЕРЕЛІК ПОСИЛАНЬ

1. https://uk.wikipedia.org/Багатоплатформність
2. Горбань А.Г. Програмування в Java . К.: Финансы и статистика, 2008, 310 с.
3. https://coggle.it/diagram/XX9LlrstyIQa9wW2/t/%D1%96%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D0%B8–%D1%80%D0%B8%D1%81%D1%83%D0%BD%D0%BE%D0%BA1
4. Карпова Т.С. Базы данных: модели, разработка, реализация. СПб.: Питер, 2002, 304 с.
5. Кузнецов С.Д. Основы современных баз данных. М.: Лаборатория знаний, 2011, 368 с.
6. Белов В.С. Информационно–аналитические системы. Основы проектирования и применения: учебное пособие. М.: Евразийский открытый институт, 2010, 111 с.
7. ГОСТ 34.601–90. Межгосударственный стандарт. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания, Взамен ГОСТ 24.601–86, ГОСТ 24.602–86; введ. 01.01.1992, М.: Стандартинформ, 2009, 65 с.
8. Диго С.М. Проектирование и использование баз данных: учебник для студентов вузов, М.: Финансы и статистика, 2014, 364 с.
9. Дейт К. Введение в баз данных. М: Вильямс, 2001, 485 с.
10. Советов Б.Я., Цехановский В.В., Чертовской В.Д. Базы данных. Теория и практика, М.: Юрайт, 2014, 464 с.

Додаток А

Класс MainApp

```
package library;

import javafx.application.Application;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.image.Image;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.BorderPane;
import javafx.stage.Modality;
import javafx.stage.Stage;
import library.model.Book;
import library.model.BookListWrapper;
import library.view.BookEditDialogController;
import library.view.BookOverviewController;
import library.view.RootLayoutController;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import javax.xml.bind.Unmarshaller;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.prefs.Preferences;

public class MainApp extends Application {

    private Stage primaryStage;
    private BorderPane rootLayout;

    /**
     * The data as an observable list of Book.
     */
    private final ObservableList<Book> bookData = FXCollections.observableArrayList();


    /**
     * Constructor
     */
    public MainApp() throws FileNotFoundException {

        // Add some sample data
        bookData.add(new Book("Основы современных баз данных", "Кузнецов С.Д. ", "научная", 2011,
"Лаборатория знаний", "erfref"));

        bookData.add(new Book("Энциклопедия технологий баз данных", "Когаловский М.Р. ", "научная",
2014, "Финансы и статистика","erf"));

    }
```

```
public ObservableList<Book> getBookData() {
    return bookData;
}

@Override
public void start(Stage primaryStage) {
    this.primaryStage = primaryStage;
    this.primaryStage.setTitle("Library");
    this.primaryStage.setWidth(1200);
    this.primaryStage.setHeight(600);

    initRootLayout();

    showBookOverview();
}


/**
 * Initializes the root layout and tries to load the last opened
 * person file.
 */
public void initRootLayout() {
    try {
        // Load root layout from fxml file.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/RootLayout.fxml"));
        rootLayout = (BorderPane) loader.load();

        // Show the scene containing the root layout.
        Scene scene = new Scene(rootLayout);
        primaryStage.setScene(scene);

        // Give the controller access to the main app.
        RootLayoutController controller = loader.getController();
        controller.setMainApp(this);

        primaryStage.show();
    } catch (IOException e) {
        e.printStackTrace();
    }

    // Try to load last opened person file.
    File file = getBookFilePath();
    if (file != null) {
        loadBookDataFromFile(file);
    }
}

/**
 * Shows the person overview inside the root layout.
 */
public void showBookOverview() {
    try {
        // Load person overview.
        FXMLLoader loader = new FXMLLoader();
        loader.setLocation(MainApp.class.getResource("view/BookOverview.fxml"));
        AnchorPane bookOverview = (AnchorPane) loader.load();

        // Set person overview into the center of root layout.
        rootLayout.setCenter(bookOverview);

        // Give the controller access to the main app.
```

```
            BookOverviewController controller = loader.getController();
            controller.setMainApp(this);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public boolean showBookEditDialog(Book book) {
        try {
            // Load the fxml file and create a new stage for the popup dialog.
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(MainApp.class.getResource("view/BookEditDialog.fxml"));
            AnchorPane page = (AnchorPane) loader.load();


            // Create the dialog Stage.
            Stage dialogStage = new Stage();
            dialogStage.setTitle("Edit Book");
            dialogStage.initModality(Modality.WINDOW_MODAL);
            dialogStage.initOwner(primaryStage);
            Scene scene = new Scene(page);
            dialogStage.setScene(scene);

            // Set the book into the controller.
            BookEditDialogController controller = loader.getController();
            controller.setDialogStage(dialogStage);
            controller.setBook(book);

            // Set the dialog icon.
            dialogStage.getIcons().add(new Image("file:resources/images/edit.png"));

            // Show the dialog and wait until the user closes it
            dialogStage.showAndWait();

            return controller.isOkClicked();
        } catch (IOException e) {
            e.printStackTrace();
            return false;
        }
    }

    /**
     * @return
     */
    public File getBookFilePath() {
        Preferences prefs = Preferences.userNodeForPackage(MainApp.class);
        String filePath = prefs.get("filePath", null);
        if (filePath != null) {
            return new File(filePath);
        } else {
            return null;
        }
    }

    /**
     * @param file the file or null to remove the path
     */
    public void setBookFilePath(File file) {
        Preferences prefs = Preferences.userNodeForPackage(MainApp.class);
        if (file != null) {
            prefs.put("filePath", file.getPath());
```

```
        // Update the stage title.
        primaryStage.setTitle("LibraryApp – " + file.getName());
    } else {
        prefs.remove("filePath");

        // Update the stage title.
        primaryStage.setTitle("LibraryApp");
    }
}

public void loadBookDataFromFile(File file) {
    try {
        JAXBContext context = JAXBContext
                .newInstance(BookListWrapper.class);
        Unmarshaller um = context.createUnmarshaller();

        // Reading XML from the file and unmarshalling.
        BookListWrapper wrapper = (BookListWrapper) um.unmarshal(file);

        bookData.clear();
        bookData.addAll(wrapper.getBooks());

        // Save the file path to the registry.
        setBookFilePath(file);

    } catch (Exception e) { // catches ANY exception
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText("Could not load data");
        alert.setContentText("Could not load data from file:\n" + file.getPath());

        alert.showAndWait();
    }
}

/**
 * Saves the current person data to the specified file.
 *
 * @param file
 */
public void saveBookDataToFile(File file) {
    try {
        JAXBContext context = JAXBContext
                .newInstance(BookListWrapper.class);
        Marshaller m = context.createMarshaller();
        m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

        // Wrapping our person data.
        BookListWrapper wrapper = new BookListWrapper();
        wrapper.setBooks(bookData);

        // Marshalling and saving XML to the file.
        m.marshal(wrapper, file);

        // Save the file path to the registry.
        setBookFilePath(file);
    } catch (Exception e) { // catches ANY exception
        Alert alert = new Alert(AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText("Could not save data");
        alert.setContentText("Could not save data to file:\n" + file.getPath());

        alert.showAndWait();
```

```
        }
    }

    /**
     * Returns the main stage.
     *
     * @return
     */
    public Stage getPrimaryStage() {
        return primaryStage;
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Додаток Б

Опис контролерів

```
package library.view;

import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.control.Alert.AlertType;
import library.MainApp;
import library.model.Book;

import java.io.*;


public class BookOverviewController {
    @FXML
    private TableView<Book> bookTable;
    @FXML
    private TableColumn<Book, String> bookNameColumn;
    @FXML
    private TableColumn<Book, String> authorColumn;
    @FXML
    private TableColumn<Book, String> genreColumn;
    @FXML
    private TableColumn<Book, String> yearColumn;
    @FXML
    private TableColumn<Book, String> pubHouseColumn;

    @FXML
    private Label bookNameLabel;
    @FXML
    private Label authorLabel;
    @FXML
    private Label genreLabel;
    @FXML
    private Label yearLabel;
    @FXML
    private Label pubHouseLabel;
    @FXML
    private Label pathLabel;

    @FXML
    private TextArea descrLabel;

    // Reference to the main application.
    private MainApp mainApp;


    /**
     * The constructor.
     * The constructor is called before the initialize() method.
     */
    public BookOverviewController() {

    }

    /**
     * Initializes the controller class. This method is automatically called
     * after the fxml file has been loaded.
     */
```

```
@FXML
private void initialize() {
    // Initialize the person table with the two columns.
    bookNameColumn.setCellValueFactory(cellData –> cellData.getValue().bookNameProperty());
    authorColumn.setCellValueFactory(cellData –> cellData.getValue().authorProperty());
    genreColumn.setCellValueFactory(cellData –> cellData.getValue().genreProperty());
    yearColumn.setCellValueFactory(cellData –> cellData.getValue().yearProperty().asString());
    pubHouseColumn.setCellValueFactory(cellData –> cellData.getValue().pubHouseProperty());


    // Clear person details.
    showBookDetails(null);

    // Listen for selection changes and show the person details when changed.
    bookTable.getSelectionModel().selectedItemProperty().addListener(
            (observable, oldValue, newValue) –> showBookDetails(newValue));
}

public void setMainApp(MainApp mainApp) {
    this.mainApp = mainApp;

    // Add observable list data to the table
    bookTable.setItems(mainApp.getBookData());
}

private void showBookDetails(Book book) {
    if (book != null) {
        // Fill the labels with info from the book object.
        bookNameLabel.setText(book.getBookName());
        authorLabel.setText(book.getAuthor());
        genreLabel.setText(book.getGenre());
        yearLabel.setText(Integer.toString(book.getYear()));
        pubHouseLabel.setText(book.getPubHouse());
        descrLabel.setText(book.getDescr());
        pathLabel.setText(book.getBookFile());

        descrLabel.setWrapText(true);
    } else {
        // Book is null, remove all the text.
        bookNameLabel.setText("");
        authorLabel.setText("");
        genreLabel.setText("");
        yearLabel.setText("");
        pubHouseLabel.setText("");
        descrLabel.setText("");
        pathLabel.setText("");
    }
}

@FXML
private void handleDeleteBook() {
    int selectedIndex = bookTable.getSelectionModel().getSelectedIndex();
    if (selectedIndex >= 0) {
        bookTable.getItems().remove(selectedIndex);
    } else {
        // Nothing selected.
        Alert alert = new Alert(AlertType.WARNING);
        alert.initOwner(mainApp.getPrimaryStage());
        alert.setTitle("No Selection");
        alert.setHeaderText("No Book Selected");
        alert.setContentText("Please select a book in the table.");

        alert.showAndWait();
```

```
        }
    }

    @FXML
    private void handleNewBook() throws FileNotFoundException {
        Book tempBook = new Book();
        boolean okClicked = mainApp.showBookEditDialog(tempBook);
        if (okClicked) {
            mainApp.getBookData().add(tempBook);
        }
    }

    @FXML
    private void handleEditBook() {
        Book selectedBook = bookTable.getSelectionModel().getSelectedItem();
        if (selectedBook != null) {
            boolean okClicked = mainApp.showBookEditDialog(selectedBook);
            if (okClicked) {
                showBookDetails(selectedBook);
            }

        } else {
            // Nothing selected.
            Alert alert = new Alert(AlertType.WARNING);
            alert.initOwner(mainApp.getPrimaryStage());
            alert.setTitle("No Selection");
            alert.setHeaderText("No Book Selected");
            alert.setContentText("Please select a book in the table.");

            alert.showAndWait();
        }
    }

    @FXML
    private void handleOpenFileBook() throws IOException {
        //Runtime.getRuntime().exec("explorer.exe /select," + tempFileBook.getBookFile().toString());
        Process p = new ProcessBuilder("explorer.exe", "/select," + pathLabel.getText()).start();
    }

}
```

```
package library.view;

import javafx.scene.image.ImageView;
import javafx.stage.FileChooser;
import library.model.Book;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.TextField;
import javafx.scene.control.TextArea;
import javafx.scene.image.Image;
import javafx.stage.Stage;

import java.io.File;
import java.io.IOException;
import java.util.Calendar;

/**
 * Dialog to edit details of a book.
 */
```

```
public class BookEditDialogController {

    @FXML
    private TextField bookNameField;
    @FXML
    private TextField authorField;
    @FXML
    private TextField yearField;
    @FXML
    private TextField genreField;
    @FXML
    private TextField pubHouseField;
    @FXML
    private TextArea descrField;

    @FXML
    private TextField bookFileField;


    private Stage dialogStage;
    private Book book;
    private boolean okClicked = false;

    @FXML
    private void initialize() {
    }

    public void setDialogStage(Stage dialogStage) {
        this.dialogStage = dialogStage;
        descrField.setWrapText(true);

        // Set the dialog icon.
        this.dialogStage.getIcons().add(new Image("file:resources/images/edit.png"));
    }

    public void setBook(Book book) {
        this.book = book;

        bookNameField.setText(book.getBookName());
        authorField.setText(book.getAuthor());
        yearField.setText(Integer.toString(book.getYear()));
        genreField.setText(book.getGenre());
        pubHouseField.setText(book.getPubHouse());
        descrField.setText(book.getDescr());
        bookFileField.setText(book.getBookFile());
    }

    public boolean isOkClicked() {
        return okClicked;
    }

    @FXML
    private void handleOk() {
        if (isInputValid()) {
            book.setBookName(bookNameField.getText());
            book.setAuthor(authorField.getText());
            book.setYear(Integer.parseInt(yearField.getText()));
            book.setGenre(genreField.getText());
            book.setPubHouse(pubHouseField.getText());
            book.setDescr(descrField.getText());
            book.setBookFile(bookFileField.getText());

            okClicked = true;
```

```
            dialogStage.close();
        }
    }

    /**
     * Called when the user clicks cancel.
     */
    @FXML
    private void handleCancel() {
        dialogStage.close();
    }

    /**
     * Validates the user input in the text fields.
     *
     * @return true if the input is valid
     */
    private boolean isInputValid() {
        String errorMessage = "";

        if (bookNameField.getText() == null || bookNameField.getText().length() == 0) {
            errorMessage += "No valid book name!\n";
        }
        if (authorField.getText() == null || authorField.getText().length() == 0) {
            errorMessage += "No valid author!\n";
        }
        if (genreField.getText() == null || genreField.getText().length() == 0) {
            errorMessage += "No valid genre!\n";
        }

        Calendar cal = Calendar.getInstance();

        if (Integer.parseInt(yearField.getText()) > cal.getWeekYear() ||
                Integer.parseInt(yearField.getText()) < 1500 ||
                yearField.getText() == null || yearField.getText().length() == 0) {
            errorMessage += "No valid year!\n";
        } else {
            // try to parse the year into an int.
            try {
                Integer.parseInt(yearField.getText());
            } catch (NumberFormatException e) {
                errorMessage += "No valid year (must be an integer)!\n";
            }
        }

        if (pubHouseField.getText() == null || pubHouseField.getText().length() == 0) {
            errorMessage += "No valid pub house!\n";
        }

        if (errorMessage.length() == 0) {
            return true;
        } else {
            // Show the error message.
            Alert alert = new Alert(AlertType.ERROR);
            alert.initOwner(dialogStage);
            alert.setTitle("Invalid Fields");
            alert.setHeaderText("Please correct invalid fields");
            alert.setContentText(errorMessage);

            alert.showAndWait();

            return false;
        }
```

```
    }

    @FXML
    private void handleAddFile() throws IOException {

        FileChooser chooser = new FileChooser();
        FileChooser.ExtensionFilter extFilter =
                new FileChooser.ExtensionFilter("Book files", "*.pdf", "*.djvu", "*.docx");
        chooser.getExtensionFilters().add(extFilter);
        chooser.setTitle("Open File");
        File bookFile1 = chooser.showOpenDialog(new Stage());
        String bookfile = bookFile1.toURI().toString();
        bookFileField.setText(bookfile);
    }

    @FXML
    private void handleOpenBook() throws IOException {
        Process p = new ProcessBuilder("explorer.exe", "/select," + bookFileField.getText()).start();
    }
}


package library.view;

import library.MainApp;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.stage.FileChooser;

import java.io.File;

public class RootLayoutController {



    // Reference to the main application
    private MainApp mainApp;


    public void setMainApp(MainApp mainApp) {
        this.mainApp = mainApp;
    }

    /**
     * Creates an empty address book.
     */
    @FXML
    private void handleNew() {
        mainApp.getBookData().clear();
        mainApp.setBookFilePath(null);
    }

    /**
     * Opens a FileChooser to let the user select an address book to load.
     */
    @FXML
    private void handleOpen() {
        FileChooser fileChooser = new FileChooser();

        // Set extension filter
        FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter(
```

```
            "XML files (*.xml)", "*.xml");
    fileChooser.getExtensionFilters().add(extFilter);

    // Show save file dialog
    File file = fileChooser.showOpenDialog(mainApp.getPrimaryStage());

    if (file != null) {
        mainApp.loadBookDataFromFile(file);
    }
}

/**
 * Saves the file to the person file that is currently open. If there is no
 * open file, the "save as" dialog is shown.
 */
@FXML
private void handleSave() {
    File bookFile = mainApp.getBookFilePath();
    if (bookFile != null) {
        mainApp.saveBookDataToFile(bookFile);
    } else {
        handleSaveAs();
    }
}

/**
 * Opens a FileChooser to let the user select a file to save to.
 */
@FXML
private void handleSaveAs() {
    FileChooser fileChooser = new FileChooser();

    // Set extension filter
    FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter(
            "XML files (*.xml)", "*.xml");
    fileChooser.getExtensionFilters().add(extFilter);

    // Show save file dialog
    File file = fileChooser.showSaveDialog(mainApp.getPrimaryStage());

    if (file != null) {
        // Make sure it has the correct extension
        if (!file.getPath().endsWith(".xml")) {
            file = new File(file.getPath() + ".xml");
        }
        mainApp.saveBookDataToFile(file);
    }
}

/**
 * Opens an about dialog.
 */
@FXML
private void handleAbout() {
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("AddressApp");
        alert.setHeaderText("About");
        alert.setContentText("Author: Ilya Panasenko\nFor: ZNU diploma project");

        alert.showAndWait();
}

/**
```

```
     * Closes the application.
     */
    @FXML
    private void handleExit() {
        System.exit(0);
    }

}
```