

STATE UNIVERSITY OF TELECOMMUNICATIONS
EDUCATIONAL AND SCIENTIFIC INSTITUTE OF INFORMATION
TECHNOLOGIES

Department of software engineering

Explanatory note
to bachelor thesis
for the bachelor's degree of higher education

on the topic: «**CREATION OF BOOK COLLECTION APPLICATION USING
THE C# LANGUAGE**»

Done by: student of the 4th year, group PD-42
121 Software engineering

(шифр і назва спеціальності/спеціалізації)

Idrissi Hesham

(прізвище та ініціали)

Head

Gamaniuk I.M.

(прізвище та ініціали)

Рецензент

(прізвище та ініціали)

Kyiv – 2023

STATE UNIVERSITY OF TELECOMMUNICATIONS
EDUCATIONAL AND SCIENTIFIC INSTITUTE OF INFORMATION
TECHNOLOGIES

Department of software engineering
Degree of higher education - «Bachelor»
Specialty – 121 «Software engineering»

I APPROVE

Head of Department
Software engineering

Negodenko O.V.
“ ” _____ 2023

T A S K
FOR THE STUDENT'S BACHELOR THESIS

IDRISSI HESHAM

(прізвище, ім'я, по батькові)

1. The topic: «Creation of book collection application using the C# language»

Head: Gamaniuk I.M., senior lecturer

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Approved by order of higher education «24» February 2023 № 26.

2. Deadline for submission of work by the student «1» June 2023

3. Input data to work

3.1 Scientific and technical literature on issues related to accounting software;

3.2 Online editor of UML diagrams;

3.3 Visual Studio integrated development environment;

4. Content of the settlement and explanatory note (list of issues to be developed)

4.1 Accounting of the book collection as a component of the collection process;

4.2 Analysis of available tools and technologies for organizing accounting of books in the collection;

4.3 Information system modeling and design;

4.4 Testing;

5. List of demonstration material (name of the main slides)

5.1. Title slide

5.2. Purpose, object and subject of research

5.3. Tasks of the graduate thesis

5.4. Analysis of analogues

- 5.5. Software requirements
- 5.6. Implementation software
- 5.7. Functional requirements
- 5.8. Put in information about the book
- 5.9. Object interaction
- 5.10. Book presentation and business logic layers
- 5.11. State diagram
- 5.12. Object diagram
- 5.13. Architecture design
- 5.14. Screen forms
- 5.15. Conclusions

6. Issue date of the task « 25 » February 2023

CALENDAR PLAN

№ in order	The name of the stages of the bachelor's work	The term of performance of work stages	Note
1	Selection of scientific and technical literature	25.02.23-27.02.23	Done
2	Analysis and research of existing analogues	28.02.23-10.03.23	Done
3	Modeling, system design	13.03.23-24.03.23	Done
4	Development of the main functionality of the system	27.03.23-28.04.23	Done
5	Conclusions and design of the work	08.05.23-12.05.23	Done
6	Development of mandatory demonstration materials	22.05.23-26.05.23	Done
7	Preliminary work protection	23.05.23	Done
8	Submission of work	01.06.23	

Student _____
(підпис)

Idrissi Hesham
(прізвище та ініціали)

Head of work _____
(підпис)

Gamaniuk I.M.
(прізвище та ініціали)

ABSTRACT

The text part of the bachelor thesis: 47 pp., 1 table, 40 figures, 2 appendices, 15 sources.

INTEGRATED DEVELOPMENT ENVIRONMENT (IDE), CRUD REQUESTS, JSON FORMAT, UNIFIED MODELING LANGUAGE (UML).

Keeping records of a book collection involves a large amount of data, inefficient processing by human resources and a high risk of errors.

Object of study: The process of accounting for books in the collection.

Subject of study: Application for accounting of books in the collection.

The purpose of the work: Automatisation of the accounting process of books in the collection.

Research methods are a unified process of creating software.

Based on the results of the completed work, an application for book collection accounting was developed, which allows implementing information support and updating the database.

Using the created application will allow you to get reliable and timely information about the availability of books in the collection.

CONTENT

INTRODUCTION.....	8
1 ACCOUNTING OF THE BOOK COLLECTION AS A COMPONENT OF THE COLLECTION PROCESS	9
1.1 Organization of accounting in the modern collection process.....	9
1.2 Accounting in conditions of digitization of life.....	9
2 ANALYSIS OF AVAILABLE TOOLS AND TECHNOLOGIES FOR ORGANIZING ACCOUNTING OF BOOKS IN THE COLLECTION.....	11
2.1 Analysis of existing programs for accounting of books in the collection.....	11
2.2 Selection of software development technologies and tools.....	16
2.2.1 C# language.....	16
2.2.2 Visual Studio integrated development environment.....	17
2.2.3 The .Net Framework platform.....	18
2.2.4 Online UML diagram editor UMLetino.....	18
2.2.5 JSON format.....	19
3 INFORMATION SYSTEM MODELING AND DESIGN.....	21
3.1 Vision of the system.....	21
3.1 Analysis of the subject area model.....	23
3.1.1 Electronic tables for displaying the subject area.....	23
3.2 Model of precedents.....	27
3.3 Design model.....	31
3.3.1 Activity design.....	31
3.3.2 Designing the sequence of method calls and object interaction.....	33
3.3.3 Object states.....	35
3.3.4 Designing classes and relationships between objects.....	37
3.3.5 State design.....	39
3.3.6 System deployment design.....	41
3.5 Architecture.....	42
4 TESTING.....	44
CONCLUSIONS.....	45
REFERENCES.....	46
ANNEX A.....	48
ANNEX B.....	58

INTRODUCTION

Today, people and information technologies are closely related to each other. In the conditions of modern information development, it is necessary to continuously improve the current activity of people by automating its life processes. Involvement of information technologies in book collection accounting will allow reliable storage of data, make their display more convenient, speed up search, dramatically increase processing speed and accuracy of results.

There are a certain number of programs for accounting for a book collection. In most cases, it is a cumbersome piece of software that covers almost every aspect of a person's book collection accounting and is not convenient when a private, lightweight application is needed.

Therefore, the chosen topic of work is relevant.

Object of study: The process of accounting for books in the collection.

Subject of study: Application for accounting of books in the collection.

The purpose of the work: Automatisation of the accounting process of books in the collection.

The following tasks were solved in the study:

- study of the method of organizing the accounting process of books in the collection;
- definition of basic functions;
- application development according to modeling.

Research methodology: unified software creation process.

Practical significance of the results: this product can be used to account for a collection of books.

1 ACCOUNTING OF THE BOOK COLLECTION AS A COMPONENT OF THE COLLECTION PROCESS

1.1 Organization of accounting in the modern collection process

The purpose of human activity is to create value. One of the types of value creation is book collecting. The process of book collecting plays an important role, because book collecting as a component of the overall process of value creation is the final stage of the movement of a product from the entertainment sphere to the sphere of cultural heritage.

Clear, timely, properly organized accounting of books in the collection contributes to strengthening control over the availability of books, providing collectors with information and accelerating the exchange of books between collectors.

The software allows a person to compile book accounting records of book purchases and book movement in terms of book exchange between collectors.

1.2 Accounting in conditions of digitization of life

The growing role of information technologies as a factor in social life led to the transition to an information society and the formation of digitization of life, which is a defining trend of global socio-economic development. The latest stage is characterized by constant technological innovations, the production of information products and services, the use of computer networks and the global information space for effective communication.

Significant technological changes stimulate the modernization of accounting science, contribute to the development of the methodology and organization of the accounting process, and also actualize the problem of positioning the accounting system.

Further scientific achievements in this area are extremely important - new concepts, development of certain types of accounting, because in the conditions of the development of the information society and the digital economy, a number of prerequisites for the formation of a new accounting paradigm arise.

Digital technologies are currently used in all areas of social life: in the system of public administration, economy, business, social sphere. Such a transformation speeds up economic and social processes, makes them more qualitative.

At the same time, under the influence of modern information systems and information technologies, significant changes are taking place in accounting methodology and practice, which actualize the feasibility of developing an accounting paradigm adequate to the new conditions.

At the same time, during the last decades, problems related to the decrease in the functionality of accounting have accumulated, caused by its conservatism, the lack of informational value of accounting information for interested parties, which led to a number of studies at the fundamental level, and at the practical level - the search for ways to update accounting and increase the level of its compliance information to user requests [1].

2 ANALYSIS OF AVAILABLE TOOLS AND TECHNOLOGIES FOR ORGANIZING ACCOUNTING OF BOOKS IN THE COLLECTION

2.1 Analysis of existing programs for accounting of books in the collection

The book accounting program in the collection is an important tool for effective collecting. Automated systems can improve collection productivity. With their help, you can optimize not only the book exchange process, but also book accounting. Control of the number and balance of books of the same type helps to optimize the planning of their exchange.

Software products contain functionality that facilitates the performance of many operations. When they are used, it is possible to exchange information faster and improve collection productivity.

The most popular programs in this niche are:

1. “Reading List: Book Tracker”

Easily track books you’ve read, books you’re reading, and books you want to read. Record your progress by adding the start and finish date of your books. [2].

The program has the following interface (Figure 2.1 – 2.4):



Figure 2.1 – Registration of your books



Figure 2.2 – Record reading progress

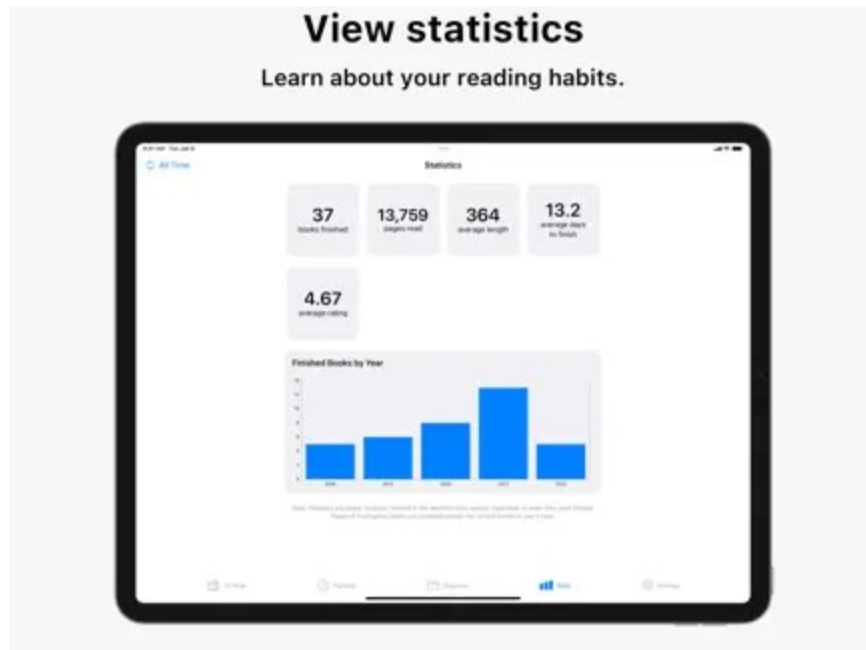


Figure 2.3 – Overview of statistics



Figure 2.4 – Information search

2. “Turn - Reading Tracker & Timer”

Turn - Reading Tracker is designed as a book tracker and library tracker to help you track your reading habits and organize your book collection effortlessly while you read. Some of the features are detailed below [3].

The program interface is as follows (Figure 2.5, 2.6):

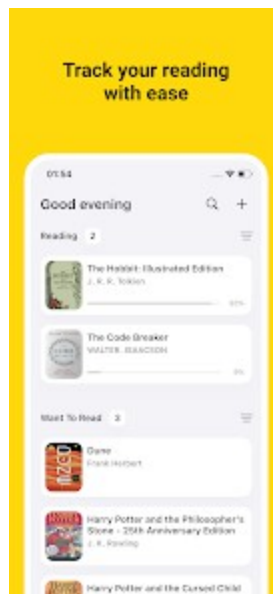


Figure 2.5 – Tracking the reading

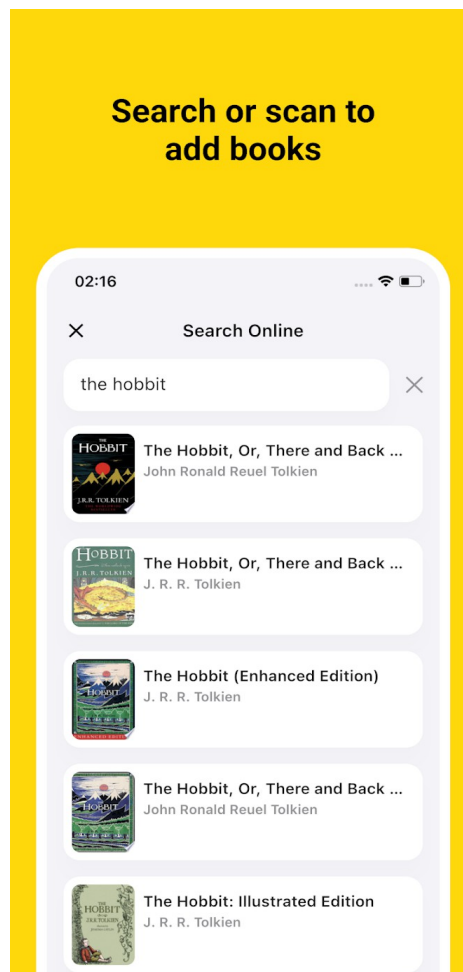


Figure 2.6 – Search and add books

3. “Bookmory - reading tracker”

Bookmory helps you track your reading, manage your books, build a lasting reading habit, and better remember what you read.

Add books, e-books or audiobooks to your bookshelf.

Use the timer to track your reading. Improve your reading habits with insightful stats. Stay motivated with reading goals.

Remember what you read by writing and reviewing notes. [4].

The program interface is as follows (Figure 2.7, 2.8):

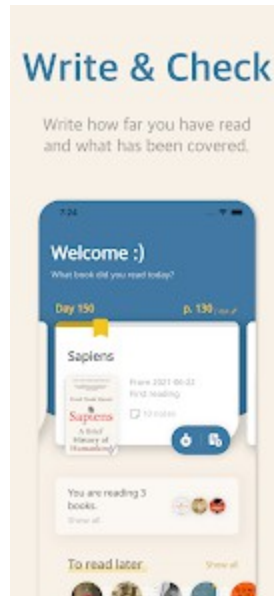


Figure 2.7 – Reading progress

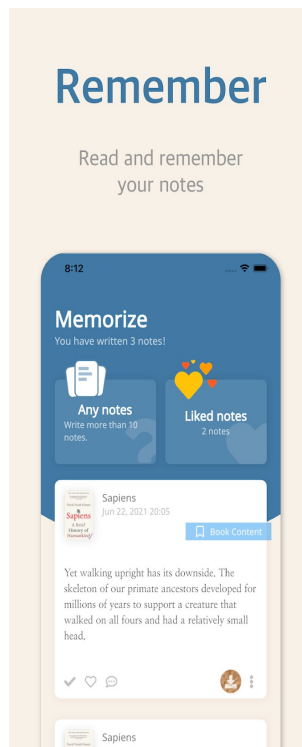


Figure 2.8 – Actions with notes

So, the above-mentioned programs have a convenient interface and good functionality, but they lack privacy, ease of use, lightweight, and ease of maintenance. However, the created project takes this into account.

The results of the image analysis are in the Table 2.1.

Table 2.1 — Analysis of analogues

	Reading List: Book Tracker	Turn - Reading Tracker & Timer	Bookmory - reading tracker	BookSea
Private	-	-	-	+
Simple using	+	-	+	+
Lightweight	+	+	-	+
Ease of maintenance	-	+	+	+

2.2 Selection of software development technologies and tools

To create software for accounting of books in the collection, it is necessary to choose development tools. Development tools include: programming language, integrated development environment, database management system, editor of UML diagrams.

2.2.1 C# language

To implement this project in the Microsoft Visual Studio development environment, the C# programming language is used, which is an element of the .Net Framework. For the development of any application on the Windows operating system, the .Net environment is created, while the C# language is used together with the .Net Framework. Therefore, at the moment, the combination of C# and .Net is the most productive for programmers.

C# is a modern, universal, object-oriented programming language developed by Microsoft.

This programming language is very easy to learn. It is important to note that an application written in C# can be deployed on any operating system such as Android, iOS, Windows, or a cloud platform.

There are many important features of C# that make it more useful and unique compared to other languages:

- very fast, its compilation and execution time do not take much time;
- has a rich set of library functions and data types;
- is one of the modern programming languages;
- is type-safe code that can only access a memory location and has execute permission. Thus, it improves the security of the application;
- to solve large problems, programming in C# divides the problem into smaller modules called functions or procedures, each of which has a specific responsibility, which is why C# is called a structured programming language.

- very fast, its compilation and execution time do not take much time.

But C# also has certain disadvantages:

- is completely based on the Microsoft .Net platform, so this language is not flexible;
- after making changes in the written code, it must be recompiled [6].

2.2.2 Visual Studio integrated development environment

An Integrated Development Environment (IDE) is software for creating programs that combines common developer tools into a single Graphical User Interface (GUI).

An IDE consists of:

- source code editor: a text editor that can help you write software code with features such as syntax highlighting with visual hints, language-specific autocompletion, and error checking as you write code.
- local build automation: compilation of computer source code into binary code, packaging of binary code and launch of automated tests.
- debugger: a program for testing other programs that can graphically display the location of the error in the original code [7].

Microsoft Visual Studio is an integrated environment created by Microsoft Corporation for developing graphical user interface, console, web applications, web applications, mobile applications, cloud applications and web services, etc. With the help of this IDE, you can make managed, as well as write your own code. Visual Studio (VS) uses various software platforms, which include: Windows Store, Microsoft Silverlight, Windows API. VS is used to write code in such languages as: C#, C++, VB (Visual Basic), Python, JavaScript, etc. [8].

2.2.3 The .Net Framework platform

.Net Framework is a software development platform produced by Microsoft Corporation for building and running Windows applications. The platform consists of developer tools, programming languages, and libraries for writing desktop and web applications, as well as web services and games. It supports various programming languages. Thus, developers can choose the language to develop the required application. Visual Basic and C# are popular.

The .NET Framework has a set of standard class libraries. A class library is a set of methods and functions that can be used for a given task. Programs using the .NET framework can run on all Windows platforms [9].

2.2.4 Online UML diagram editor UMLetino

UMLetino is an online tool for building and editing UML diagrams.

Main features: web application without installation, export of files in PNG, EPS, PDF, JPG, SVG formats, simple modifications of UML elements based on markup, saving of diagrams in browser storage, support for all types of UML diagrams.

UMLetino supports the following types of UML diagrams: class diagrams, use case diagrams, sequence diagrams, state diagrams, deployment diagrams, activity diagrams, and others [5].

It looks like this (Figure 2.9):

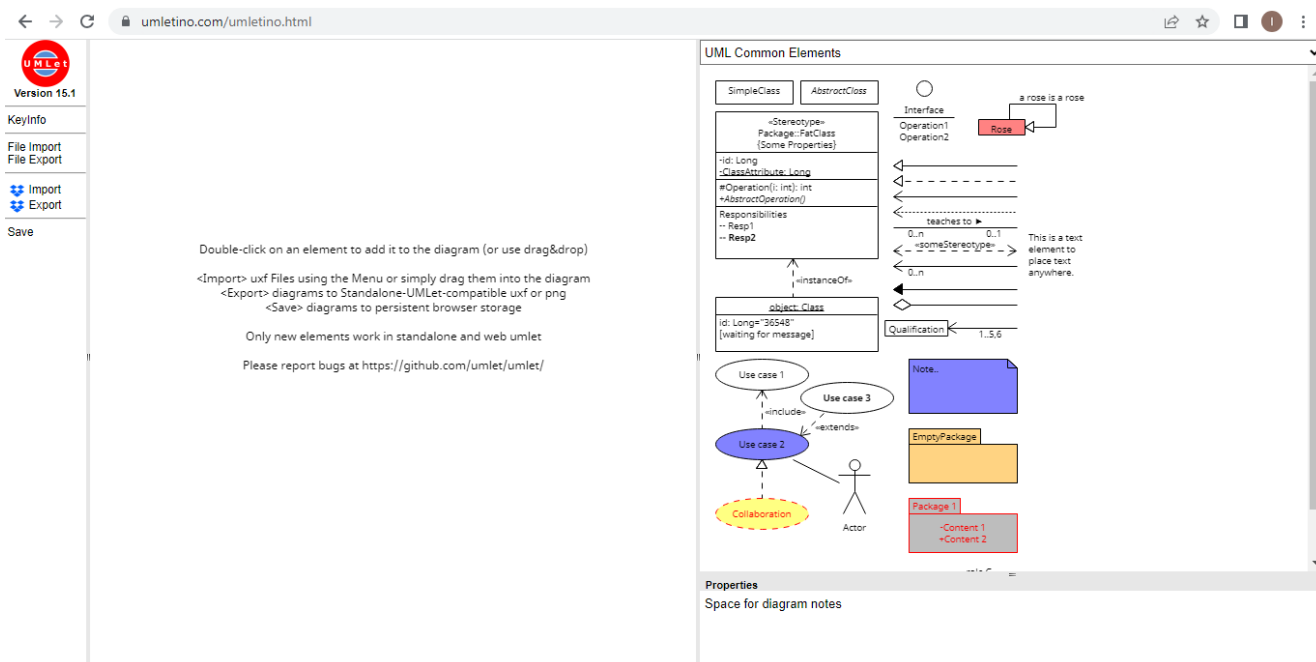


Figure 2.9 – UMLetino interface

2.2.5 JSON format

There is information: a list of books, a list of authors, a list of author endorsements for books, etc. The question is where to store all this information.

Currently, a popular way to implement data storage in applications is to use files in JSON format.

JSON (Figure. 2.10) is a text format for exchanging data between computers. JSON is text-based and human-readable. This format allows you to describe objects and other data structures. It is mainly used to transmit structured information over the network.

JSON is based on two rules:

1. a set of name-value pairs (embodied in different languages as an object, record, structure, dictionary, hash table, key list or associative array);
2. an ordered list of values (implied in many languages as an array, vector, list, or sequence).

```
[
  {
    "Id":1,
    "Title":"C# 8.0 and .NET Core 3.0",
    "IssueDate":"2019-10-31T00:00:00",
    "Summary":"In C# 8.0 and .NET Core 3.0 \u2013 Modern Cross-Platform Development, Fourth Edition, expert teacher Mark J. Price gives you everything you need to start programming C# applications."
  },
  {
    "Id":2,
    "Title":"Head First C#",
    "IssueDate":"2021-01-26T00:00:00",
    "Summary":"Dive into C# and create apps, user interfaces, games, and more using this fun and highly visual introduction to C#, .NET Core, and Visual Studio."
  },
  {
    "Id":3,
    "Title":"C# 8.0 Pocket Reference",
    "IssueDate":"2019-12-10T00:00:00",
    "Summary":"When you need answers about using C# 8.0, this tightly focused and practical book tells you exactly what you need to know without long intros or bloated samples."
  }
]
```

Figure 2.10 – JSON example

It is advantageous to use json files when you need to store loosely coupled structures, but JSON becomes inefficient when storing structures that are tightly coupled.

3 INFORMATION SYSTEM MODELING AND DESIGN

3.1 Vision of the system

A system vision is a conceptual document that defines the place and role of the information system under development.

Vision of the system

Date of amendment: 01.05.2022

Version: Final version.

Date: 01.05.2022

Description: Final version. All clarifications are taken into account.

Introduction:

A reliable private application.

Positioning:

Economic prerequisites:

- Existing software products are free, have a convenient interface and good functionality, but do not provide privacy, their use without the Internet, ease and simplicity of maintenance.

- To ensure the operation of the application, it is not planned to spend money on the Internet and support by programming specialists.

- To reduce the costs of using existing database management systems, it is planned to store data in JSON format.

Formulation of the problem:

- It is necessary to create an application that records the books in the collection, but which ensures privacy, does not involve the use of the Internet and database management systems, and also does not involve the support of programming specialists.

System place:

- The system is intended for private use. It does not interact with other systems.

- The system is intended for a person who likes to collect books. It records the books in the collection.

Interested persons:

- User - for accounting of books in the collection.

Basic high-level tasks:

- Accounting of books in the collection;
- Search and presentation of information about books in the collection.

User level tasks:

- Management of information about books
- Search for books by information
- Management of information about authors
- Search for authors by information

Review:

- Product perspective: the system will serve only the user and will not interact with other systems (Figure 3.1).

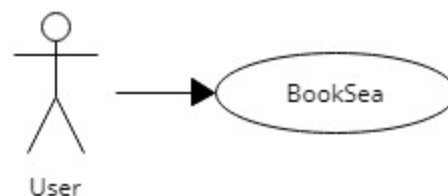


Figure 3.1 – Context diagram

A Context Diagram shows the interaction between the system and other actors (external factors) with which the system is designed to interact. The context diagram shows the entire system as a single process [10].

Advantages of the system:

- Private system.
- Does not require funds for use and support.

Assumptions and dependencies:

- It is assumed that the application will be interesting for private individuals and they will buy it.

Cost and pricing:

- Cost of system development calculated separately.

Licensing and installation:

- It is planned to carry out licensing (certification) of the system.

Install the system after licensing.

Main properties of the system:

- Management of information about books.
- Search for books by information.
- Management of information about authors.
- Search for authors by information.

Other requirements and restrictions:

- The application must be private.
- Internet will not be required when using.
- Will not use a lot of computer RAM.
- Use the .NET framework.
- Use JSON data format.

3.1 Analysis of the subject area model

3.1.1 Electronic tables for displaying the subject area

Spreadsheets can be used to represent the subject environment. With its help, it is easier to understand the structure of the construction of the UML diagram of the subject area, and later to write the code.

For this, several tables are created (Figure 3.2, 3.3):

Book			
Id	Title	IssueDate	Summary
1	C# 8.0 and .NET Core 3.0	31 Oct. 2019	In C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development, Fourth Edition, expert teacher Mark J. Price gives you everything you need to start programming C# applications.
2	Head First C#	26 Jan. 2021	Dive into C# and create apps, user interfaces, games, and more using this fun and highly visual introduction to C#, .NET Core, and Visual Studio.
3	C# 8.0 Pocket Reference	10 Dec. 2019	When you need answers about using C# 8.0, this tightly focused and practical book tells you exactly what you need to know without long intros or bloated samples.

Author		
Id	FirstName	LastName
1	Mark	J. Price
2	Andrew	Stellman
3	Jospeh	Albahari
4	Ben	Albahari

Figure 3.2 – Book and Author spreadsheets

BookAuthor			
Id	BookId	AuthorId	
1	1	1	1
2	2	2	2
3	3	3	3
4	3	4	4

Figure 3.3 – Fixing spreadsheets of books by authors

One book can have many authors and one author can have many books. Thus, the relation here is many to many.

Based on the tables, we can construct the following diagram of the subject area (Figure 3.4):

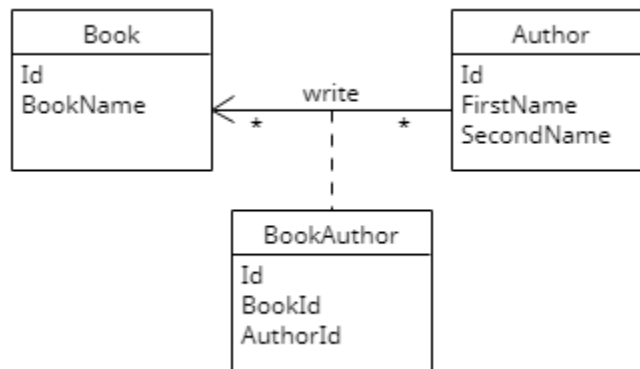


Figure 3.4 – Domain model diagram

JSON notation objects are formatted using code (Figure 3.5 — 3.7):

```

internal class Book : IId
{
    Ссылка 8
    public int Id { get; set; }
    Ссылка 3
    public string Title { get; set; }
    Ссылка 2
    public DateTime IssueDate { get; set; }
    Ссылка 2
    public string Summary { get; set; }

    Ссылка 5
    public Book(string title, DateTime issueDate, string summary)
    {
        Title = title;
        IssueDate = issueDate;
        Summary = summary;
    }
    Ссылка 0
    public override string ToString()
    {
        return string.Format(Id + " " + Title + " " + IssueDate + " " + Summary);
    }
}
  
```

Figure 3.5 – Book model

```

internal class Author : IID
{
    — ССЫЛКИ
    public int Id { get; set; }
    — ССЫЛКИ
    public string FirstName { get; set; }
    Ссылка 3
    public string LastName { get; set; }

    — ССЫЛКИ
    public Author(string firstName, string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }
    Ссылка 0
    public override string ToString()
    {
        return string.Format(Id + " " + FirstName + " " + LastName);
    }
}

```

Figure 3.6 – Author model

```

internal class BookAuthor : IID
{
    — ССЫЛКИ
    public int Id { get; set; }
    — ССЫЛКИ
    public int BookId { get; set; }
    Ссылка 3
    public int AuthorId { get; set; }

    — ССЫЛКИ
    public BookAuthor(int bookId, int authorId)
    {
        BookId = bookId;
        AuthorId = authorId;
    }
    — ССЫЛКИ
    public override string ToString()
    {
        return string.Format(Id + " " + BookId + " " + AuthorId);
    }
}

```

Figure 3.7 – BookAuthor model

3.2 Model of precedents

The UML diagram of precedents (Use-Case Diagram) models the behavior of the system. This diagram defines the interactions between the system and its participants (actors). Use cases and actors in diagrams describe what the system does and how participants use it, but not how the system works internally [11].

A diagram of book precedents is presented in Figure 3.8.

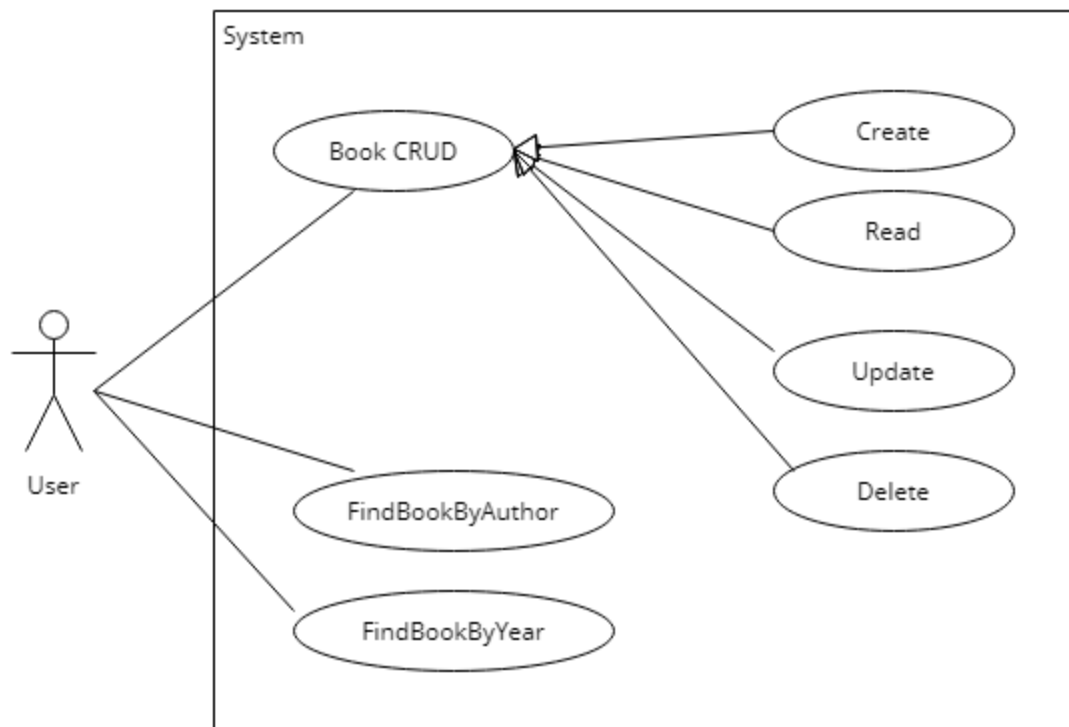


Figure 3.8 – User actions on books

The Book CRUD precedent is implemented at three levels: Presentation Layer, Business Logic Layer, and Data Access Layer. At the Business Logic Layer, Book CRUD is implemented in the BookBLL class (Figure 3.9).

```

internal class BookBLL
{
    DB db;

    Ссылка: 1
    public BookBLL(DB db)
    {
        this.db = db;
    }

    Ссылка: 1
    public int CreateBook(string title, DateTime issueDate, string summary)
    {
        Book book = new Book(title, issueDate, summary);
        return db.DBBook.AddItem(book);
    }

    Ссылка: 2
    public Book GetBookById(int id)
    {
        return db.DBBook.GetItemById(id);
    }

    Ссылка: 1
    public bool DeleteBookByBook(Book book)
    {
        return db.DBBook.DeleteItemByItem(book);
    }

    Ссылка: 0
    public bool UpdateBook(Book oldBook, string title, DateTime issueDate, string summary)
    {
        Book newBook = new Book(title, issueDate, summary);
        return db.DBBook.Update(oldBook, newBook);
    }

    Ссылка: 1
    public List<Book> GetAllBooks()
    {
        return db.DBBook.Items;
    }
}

```

Figure 3.9 – CRUD methods

The Find book by author precedent is implemented at three levels: Presentation Layer, Business Logic Layer, and Data Access Layer. At the Business Logic Layer, the precedent is implemented in the BookServiceBLL class by GetBooksByAuthorId() method (Figure 3.10).

```

public List<Book> GetBooksByAuthorId(int id)
{
    List<Book> booksByAuthorId = new List<Book>();
    foreach (Author a in db.DBAuthor.Items)
    {
        if (a.Id == id)
        {
            foreach (BookAuthor ba in db.DBBookAuthor.Items)
            {
                if (a.Id == ba.AuthorId)
                {
                    foreach (Book b in db.DBBook.Items)
                    {
                        if (b.Id == ba.BookId)
                        {
                            booksByAuthorId.Add(b);
                        }
                    }
                }
            }
        }
    }
    return booksByAuthorId;
}

```

Figure 3.10 – Find book by author method

The Find book by year precedent is implemented at three levels: Presentation Layer, Business Logic Layer, and Data Access Layer. At the Business Logic Layer, the precedent is implemented in the BookServiceBLL class by GetBooksByYear() method (Figure 3.11).

```

public List<Book> GetBooksByYear(int year)
{
    List<Book> books = new List<Book>();
    foreach (Book b in db.DBBook.Items)
    {
        if (b.IssueDate.Year == year)
        {
            books.Add(b);
        }
    }
    return books;
}

```

Figure 3.11 – Find book by year method

A diagram of book precedents is presented in Figure 3.12.

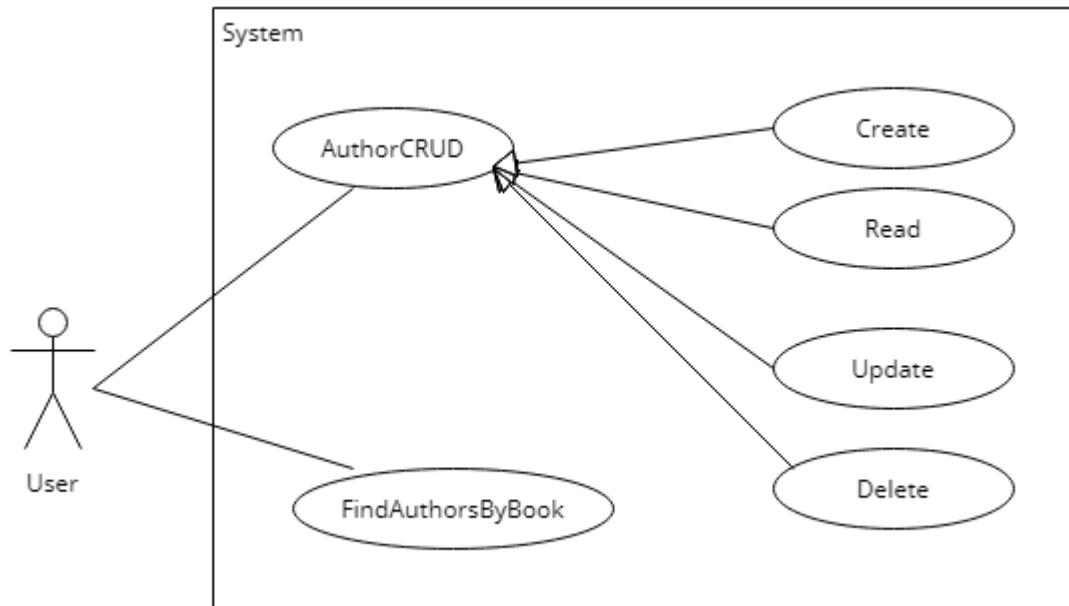


Figure 3.12 – User actions on authors

The Author CRUD precedent is implemented at three levels: Presentation Layer, Business Logic Layer, and Data Access Layer. At the Business Logic Layer, the precedent is implemented in the AuthorBLL class (Figure 3.13).

```

public int CreateAuthor(string firstName, string lastName)
{
    Author author = new Author(firstName, lastName);
    return db.DBAuthor.AddItem(author);
}
Ссылка: 0
public Author GetAuthorById(int id)
{
    return db.DBAuthor.GetItemById(id);
}
Ссылка: 0
public bool DeleteAuthorByAuthor(Author author)
{
    return db.DBAuthor.DeleteItemByItem(author);
}
Ссылка: 0
public bool UpdateAuthor(Author oldAuthor, string firstName, string lastName)
{
    Author newAuthor = new Author(firstName, lastName);
    return db.DBAuthor.Update(oldAuthor, newAuthor);
}
Ссылка: 2
public List<Author> GetAllAuthors()
{
    return db.DBAuthor.Items;
}

```

Figure 3.13 – CRUD methods

The Find authors by book precedent is implemented at three levels: Presentation Layer, Business Logic Layer, and Data Access Layer. At the Business Logic Layer, the precedent is implemented in the AuthorServiceBLL class by GetAuthorsByBook() method (Figure 3.14).

```

public List<Author> GetAuthorsByBook(int id)
{
    List<Author> authors = new List<Author>();
    foreach (Book book in db.DBBook.Items)
    {
        if (book.Id == id)
        {
            foreach (BookAuthor ba in db.DBBookAuthor.Items)
            {
                if (book.Id == ba.BookId)
                {
                    foreach (Author a in db.DBAuthor.Items)
                    {
                        if (a.Id == ba.AuthorId)
                        {
                            authors.Add(a);
                        }
                    }
                }
            }
        }
    }
    return authors;
}

```

Figure 3.14 – Find authors by book method

3.3 Design model

3.3.1 Activity design

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams are intended to model both computational and organizational processes (i.e., workflows), as well as the data flows intersecting with the related activities. Although activity diagrams primarily show the overall flow of control, they can also include elements showing the flow of data between activities through one or more data stores. [12].

The diagram shown in Figure 3.15 describes the workflow for entering information about books and the authors who wrote them.

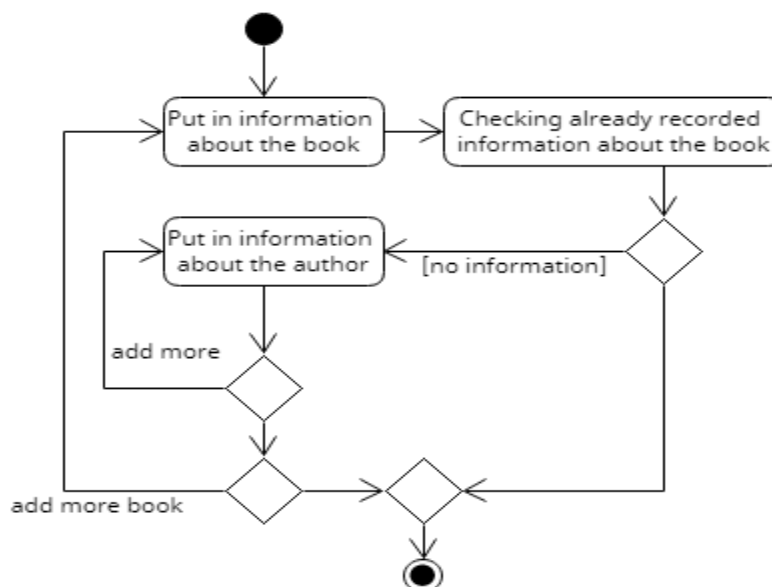


Figure 3.15 – Workflow for entering information about books and the authors

At the Presentation Layer, the workflow is implemented in the BookPL class by CreateBook() method (Figure 3.16).

```

private bool CheckBookAvailability(string title, DateTime issueDate)
{
    return bookBLL.CheckBookAvailability(title, issueDate);
}
Ссылка 3
public void CreateBook(bool addAuthor)
{
    string title = Helper.StringInputCheck("Create Book Title?: ", 50);
    DateTime issueDate = Helper.DateTimeInputCheck("Issue Date?: ");

    bool isAvailable = CheckBookAvailability(title, issueDate);

    if (!isAvailable)
    {
        string summary = Helper.StringInputCheck("Summary?: ", 150);

        int bookId = bookBLL.CreateBook(title, issueDate, summary);
        while (addAuthor)
        {
            int authorId = authorPL.CreateAuthor();
            bookAuthorBLL.CreateBookAuthor(bookId, authorId);
            Console.WriteLine("Add more authors?(y/n): ");
            if ("y" != Console.ReadLine())
            {
                addAuthor = false;
            }
        }
    }
    else { Console.WriteLine("The book is Available"); }
}
  
```

Figure 3.16 – Create book method

3.3.2 Designing the sequence of method calls and object interaction

A sequence diagram or system sequence diagram (SSD) shows process interactions arranged in time sequence in the field of software engineering. It depicts the processes and objects involved and the sequence of messages exchanged between the processes and objects needed to carry out the functionality. Sequence diagrams are typically associated with use case realizations [13].

The following diagram (Figure 3.17, 3.19) describes the interaction of objects implementing the user case for obtaining information about all books and the authors who wrote them.

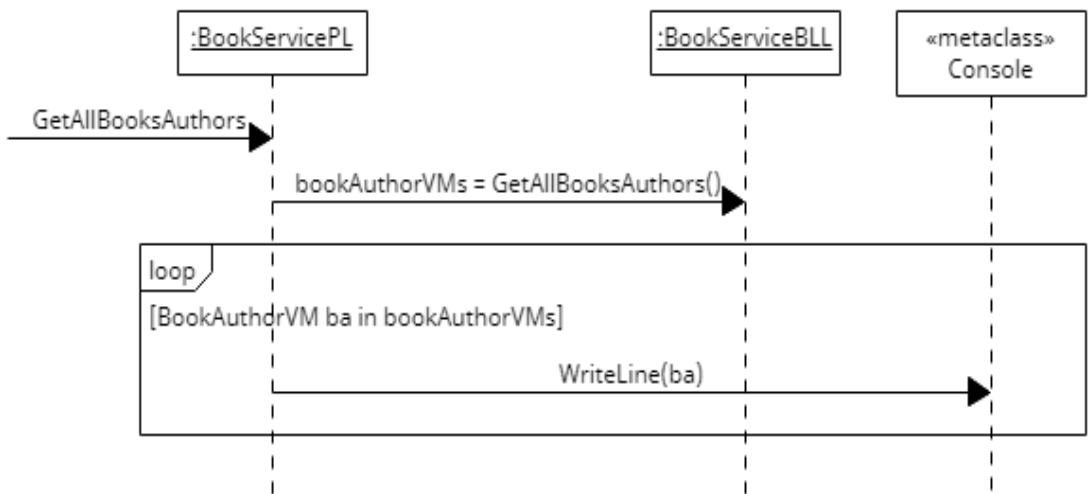


Figure 3.16 – GetAllBooksAuthors() method on Presentation Layer

```
public void GetAllBooksAuthors()
{
    List<BookAuthorVM> bookAuthorVMs = bookServiceBLL.GetAllBooksAuthors();
    foreach (BookAuthorVM ba in bookAuthorVMs)
    {
        Console.WriteLine(ba);
    }
}
```

Figure 3.17 – The implementation of the sequence diagram shown in Figure 3.16

At the Business Logic Layer, objects interact as shown in Figure 3.18.

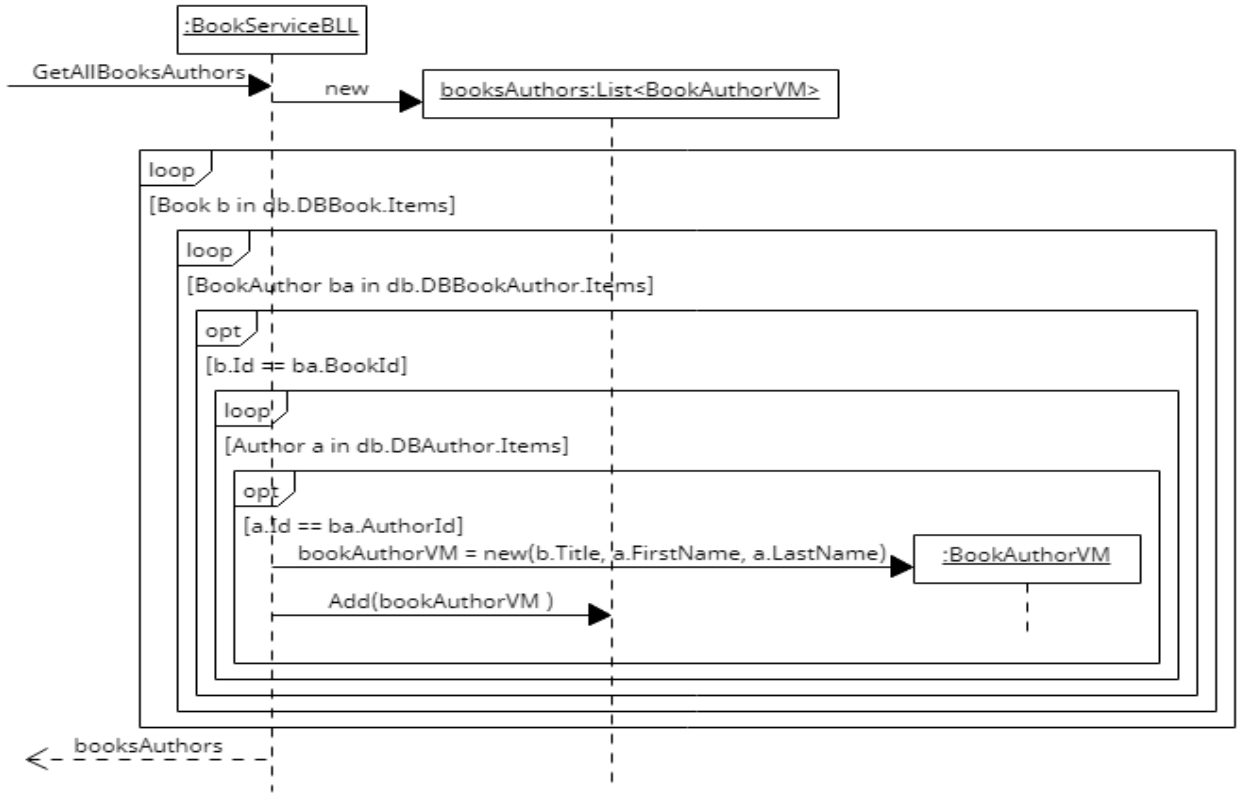


Figure 3.18 – GetAllBooksAuthors() method on Business Logic Layer

Figure 3.19 shows the GetAllBooksAuthors() method in code on Business Logic Layer.

```
public List<BookAuthorVM> GetAllBooksAuthors()
{
    List<BookAuthorVM> booksAuthors = new List<BookAuthorVM>();
    foreach (Book b in db.DBBook.Items)
    {
        foreach(BookAuthor ba in db.DBBookAuthor.Items)
        {
            if (b.Id == ba.BookId)
            {
                foreach (Author a in db.DBAuthor.Items)
                {
                    if (a.Id == ba.AuthorId)
                    {
                        BookAuthorVM bookAuthorVM = new BookAuthorVM(b.Title, a.FirstName, a.LastName);
                        booksAuthors.Add(bookAuthorVM);
                    }
                }
            }
        }
    }
    return booksAuthors;
}
```

Figure 3.19 – The implementation of the sequence diagram shown in Figure 3.18

3.3.3 Object states

We have a precedent for entering information about the book, as well as adding the authors who wrote it (Figure 3.20).

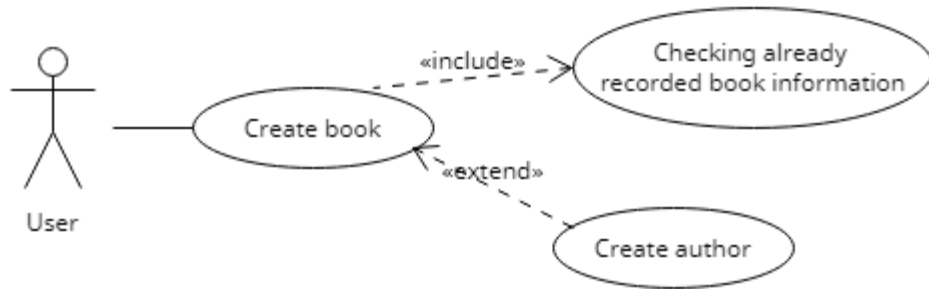


Figure 3.20 – Precedent for adding a book to a collection

Результат виконання прецеденту Create book зображено на діаграмах об'єктів (Рисунок 3.21, 3.22).

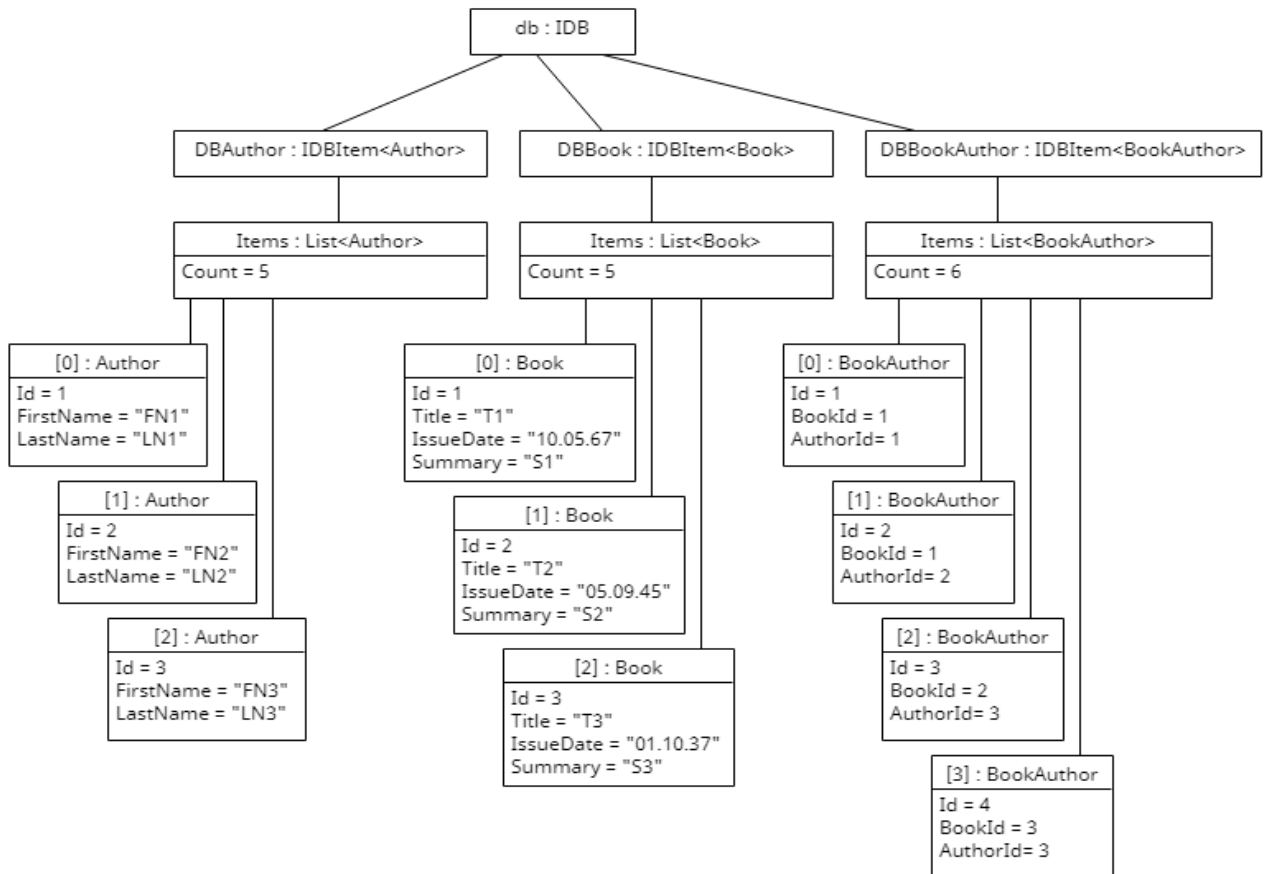


Figure 3.21 – Before event - add book and authors

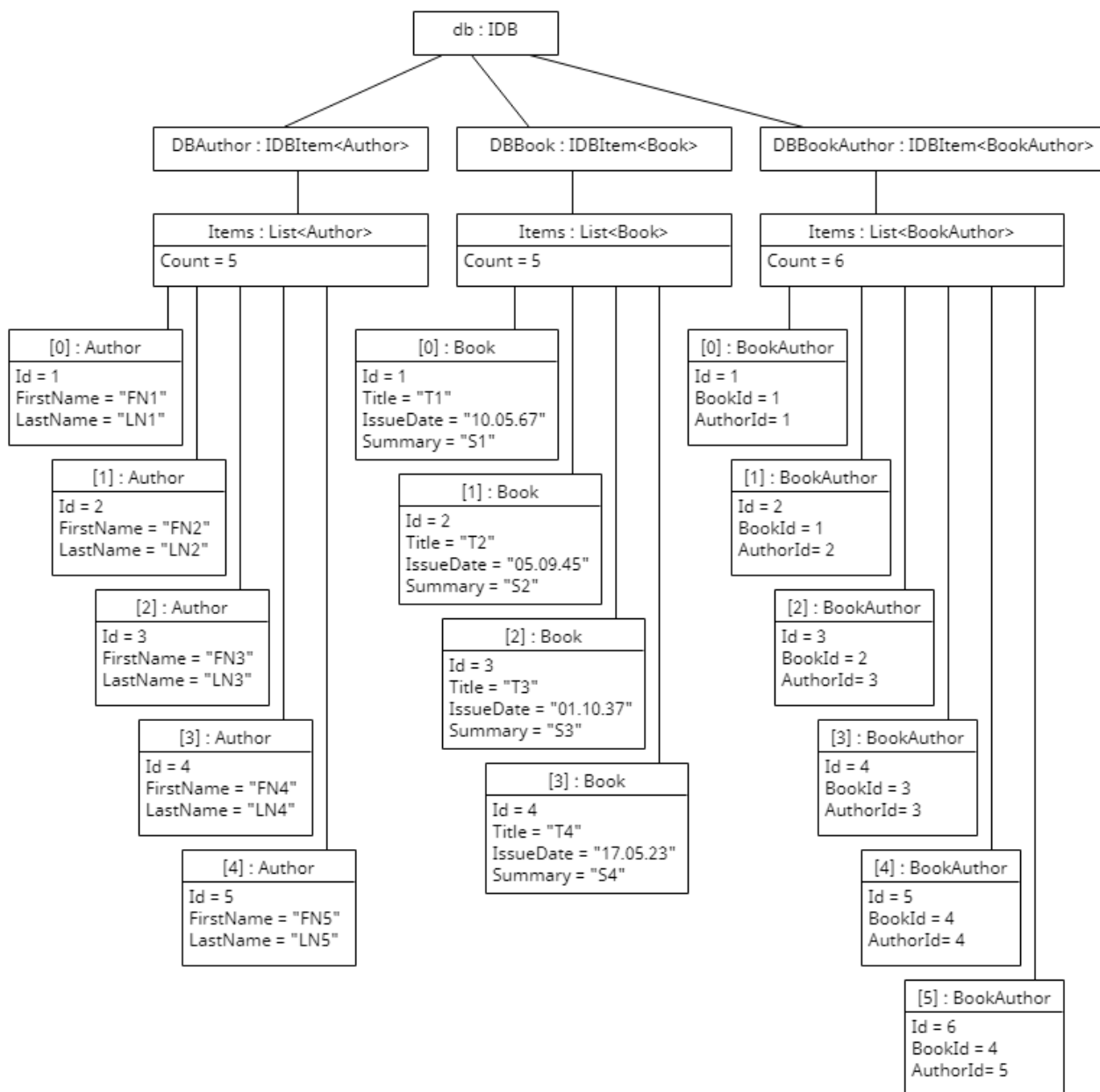


Figure 3.22 – After event - add book and authors

Before the precedent was executed, there were 3 books in the DBBook database, 3 authors in DBAuthor, and 4 author fixations for books in DBBookAuthor. After the precedent, there were 4 books in the DBBook database, 5 authors in DBAuthor, and 6 author fixations for books in DBBookAuthor.

Зміну кількості об'єктів в базі даних DBBook можна спостерігати і в Visual Studio під час розробки (Рисунок 3.23).

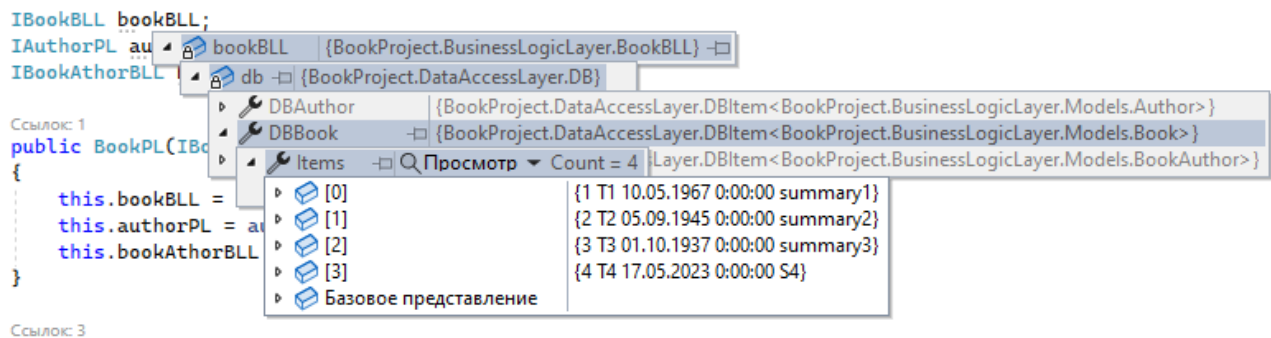


Figure 3.23 – After event - add book and authors (4 books)

3.3.4 Designing classes and relationships between objects

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects [14].

The class diagram related to the book on the Presentation Layer and Business Logic Layer is presented in Figure 3.24.

The `BookPL` class has the `IBookPL` interface and belongs to the Presentation Layer. It has a reference to an object of type `BookBLL`, which has the interface `IBookBLL` and belongs to the Business Logic Layer. At the Presentation Layer, the `BookPL` class has methods that interact with the user and use View Model classes to display information for the user. Also, the methods of the `BookPL` class refer to the methods of the `BookBLL` class to implement the corresponding services at the Business Logic Layer.

On the Business Logic Layer, the `BookBLL` class has methods that implement business logic and interact with models, and also refer to the Data Access Layer classes to implement the corresponding services on the Data Access Layer. Also, the `BookPL` class refers to the `AuthorPL` class and the `BookAuthorBLL` class to implement the service of pinning authors for the books they have written.

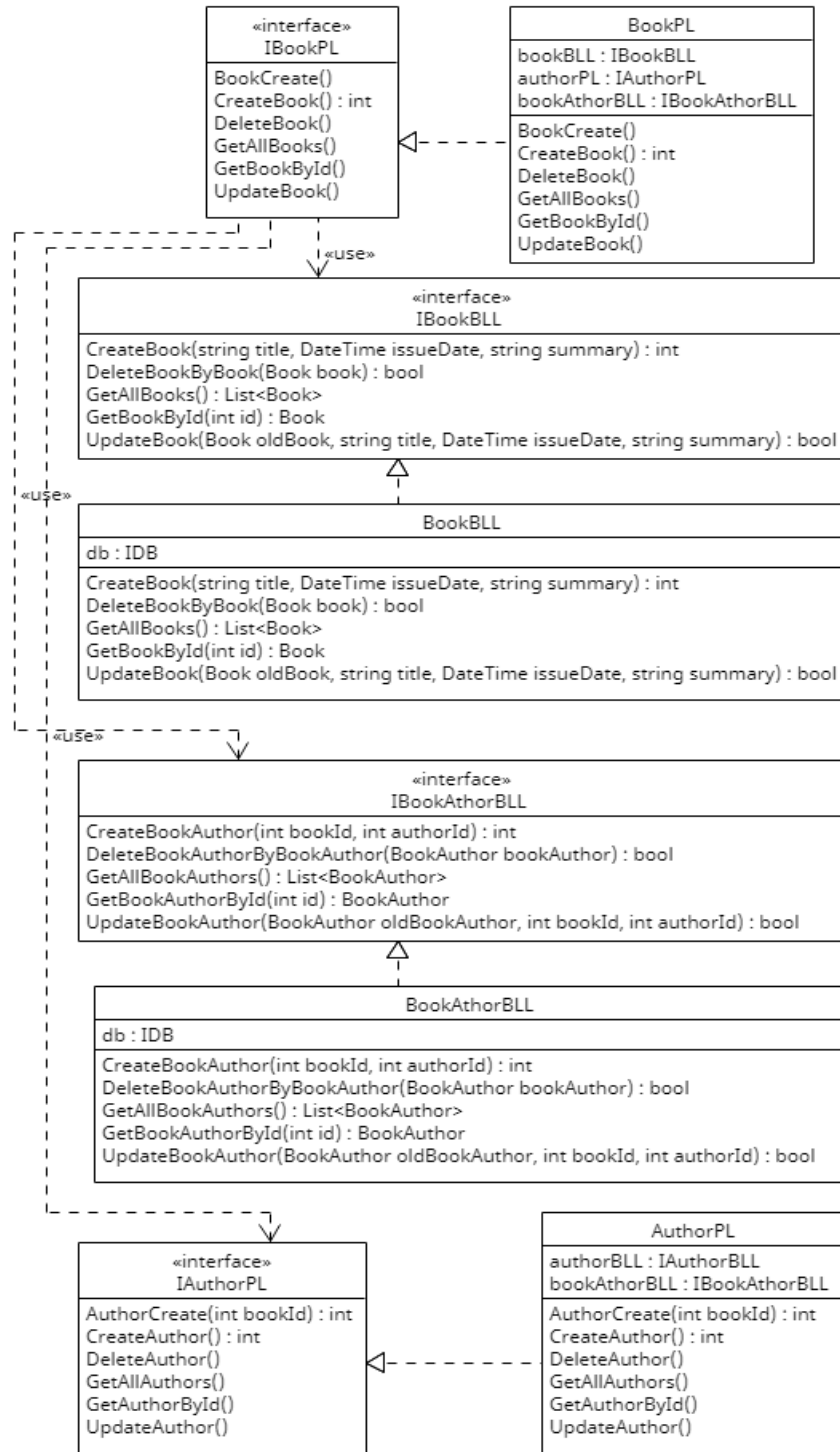


Figure 3.24 – Class diagram related to the book

3.3.5 State design

A state diagram is a diagram that highlights the event-driven behavior of an object. It consists of states, transitions, events and actions. It is used to illustrate the dynamic view of the system. A state diagram is especially important for modeling interface behavior [15].

The following diagram of the menu states (Figure 3.25) shows the process of transition from the main menu to the menu of information services about books, to the menu of information services about authors, and to the menu of management of books and authors. These three menus are also divided into sub-menus. From each menu there is an exit to a lower level.

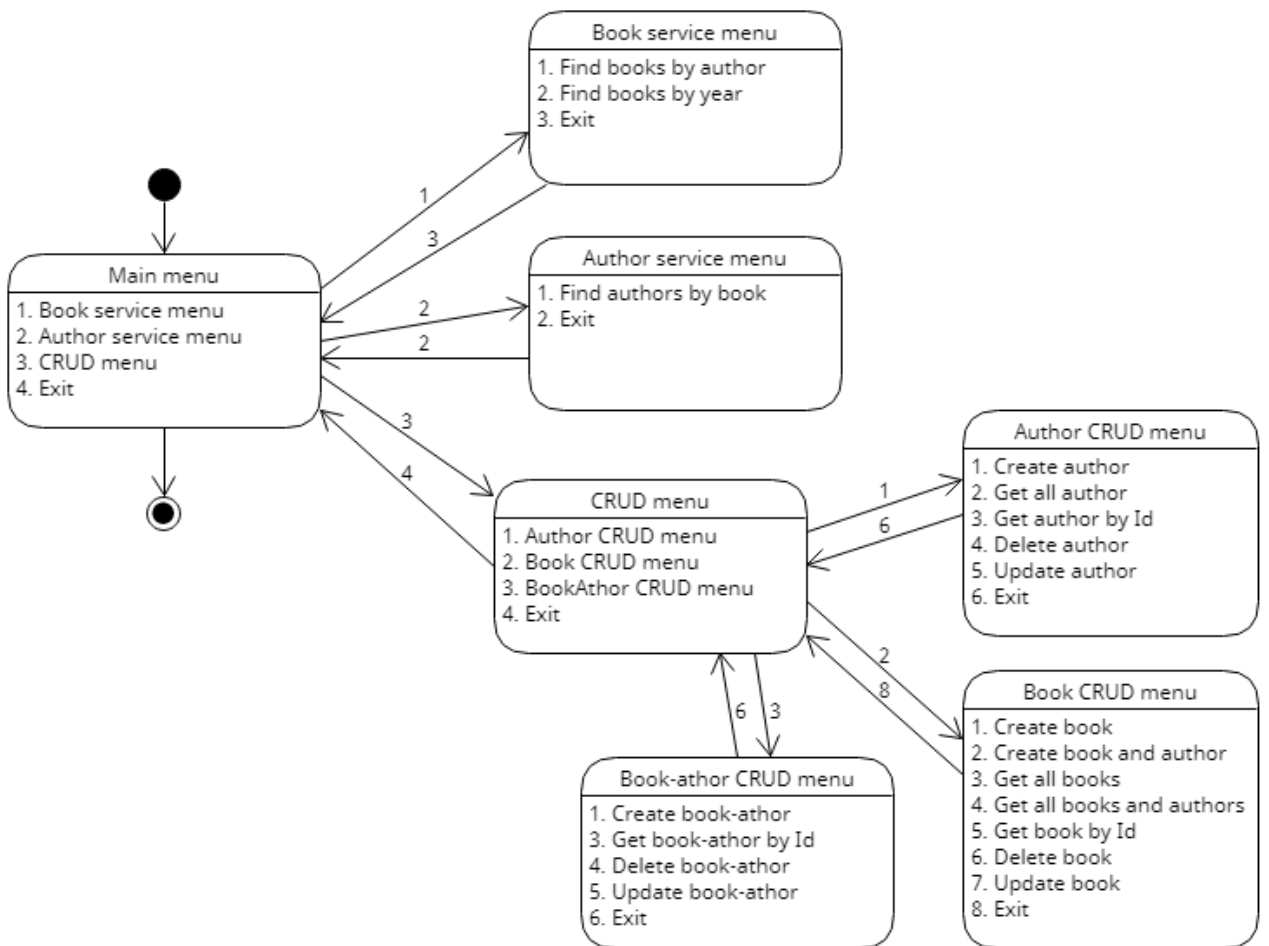


Figure 3.25 – Menu states

The main menu is implemented by the code presented in Figure 3.26.

```
internal class MainMenu : IMenu
{
    IMenu bookServiceMenu;
    IMenu authorServiceMenu;
    IMenu crudMenu;
    Ссылка 1
    public MainMenu(IMenu bookServiceMenu, IMenu authorServiceMenu, IMenu crudMenu) {
        this.bookServiceMenu = bookServiceMenu;
        this.authorServiceMenu = authorServiceMenu;
        this.crudMenu = crudMenu;
    }
    Ссылка 7
    public void Run() {
        bool flag = true;
        while (flag)
        {
            Console.WriteLine("1 - Book Service Menu");
            Console.WriteLine("2 - Author Service Menu");
            Console.WriteLine("3 - CRUD Menu");
            Console.WriteLine("4 - Exit");
            int menuNumber = Helper.IntInputCheck("-> ");
            switch (menuNumber) {
                case 1:
                    bookServiceMenu.Run();
                    break;
                case 2:
                    authorServiceMenu.Run();
                    break;
                case 3:
                    crudMenu.Run();
                    break;
                case 4:
                    flag = false;
                    break;
                default:
                    Console.WriteLine("Error");
                    break;
            }
        }
    }
}
```

Figure 3.26 – Main menu code

3.3.6 System deployment design

The application is deployed as shown in Figure 3.27.

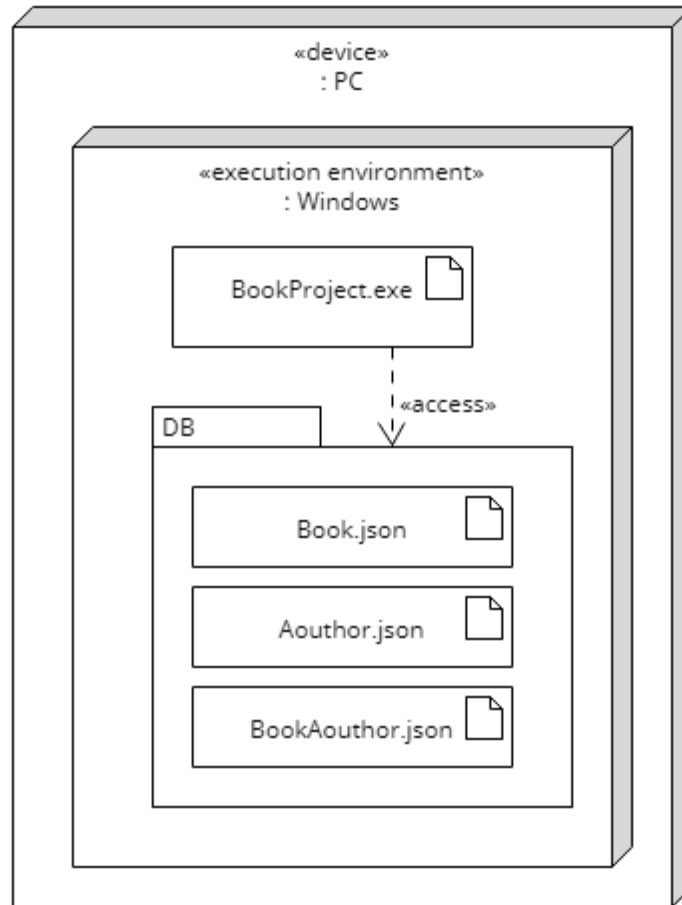


Figure 3.27 – Deployment diagram

3.5 Architecture

The application consists of three layers: Presentation Layer, Business Logic Layer, Data Access Layer.

The architecture is represented by a package diagram (Figure 3.28).

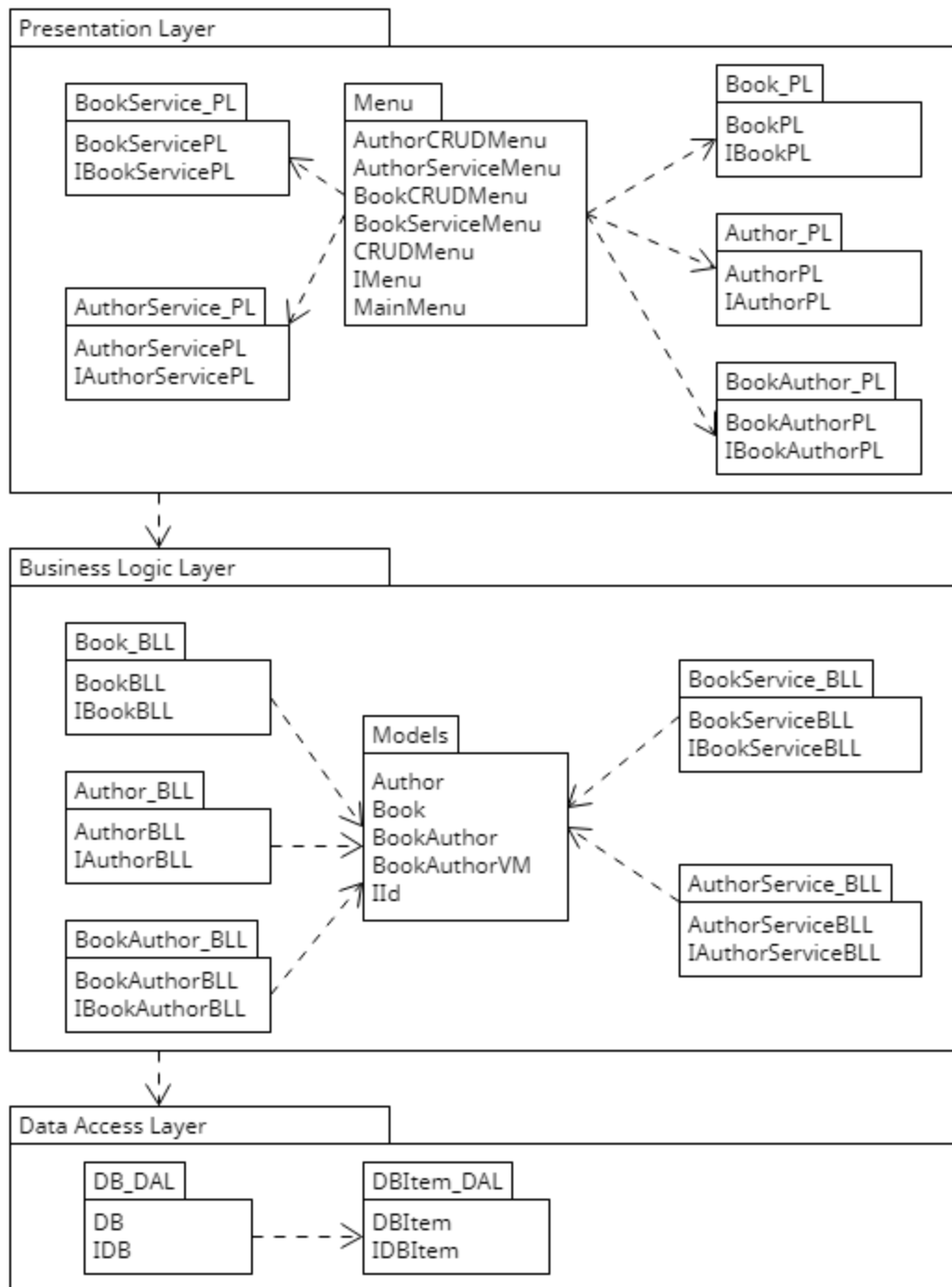


Figure 3.28 – Architecture

У програмі пакети виглядають так (Рисунок 3.29):

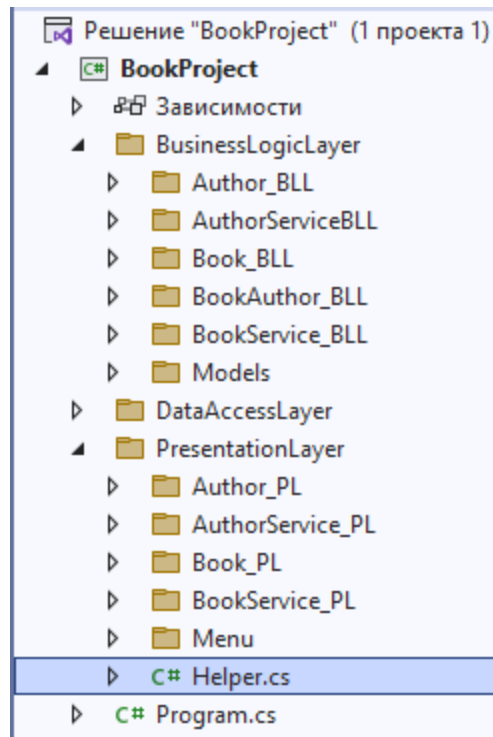


Figure 3.29 – View of packages in the project

4 TESTING

Unit testing was performed in this project.

Testing was done on the `DBItem<T>` class. The `DBItemTest` class has methods that test:

1. `AddItem_ToItems()`: whether a new object is added to the database;
2. `AddItemAdd1ToId()`: is added to Id 1;
3. `GetItemGetById()`: is it possible to find the desired object by Id;
4. `DeleteItem_Test()`: whether the item is deleted;
5. `UpdateItemTrue()`: whether the selected item changes correctly;
6. `UpdateItemFalse()`: does the selected item change correctly;
7. `UpdateItemOldListAndNewListAreEquivalent()`: whether the selected item in the collection changes.

The test results are presented below (Figure 4.1):

The screenshot shows a test runner window titled "Обозреватель тестов". The status bar indicates "Запуск тестов завершен: тестов запущено в 303 мс: 7 (пройдено: 7, не пройдено: 0, пропущено: 0)". The main table displays the following test results:

Тестирование	Длительность	Признаки	Сообщение об ошибке
BookProjectTest (7)	28 мс		
BookProjectTest (7)	28 мс		
DBItemTest (7)	28 мс		
AddItem_Add1ToId	1 мс		
AddItemToItems	22 мс		
DeleteItemTest	1 мс		
GetItem_GetById	2 мс		
UpdateItem_False	1 мс		
UpdateItem_OldListAndNewList...	1 мс		
UpdateItem_True	< 1 мс		

On the right side, a summary panel titled "Сводка по группе" shows "BookProjectTest" with "Тесты в группе: 7" and "Общая длительность" (partially visible). Under "Результаты", it shows "7 Пройден".

Figure 4.1 – Test results

CONCLUSIONS

1. The model of the subject field (diagram of the subject field) has been modulated.
2. Modulated the case model (case set and use case diagram).
3. The design model (diagrams of activities, sequences, classes) has been modulated.
4. The application has been created.


REFERENCES

1. Accounting in the digital economy [Electronic resource] – Resource access mode: <http://ibo.wunu.edu.ua/index.php/ibo/article/view/405>.
2. Reading List: Book Tracker [Electronic resource] – Resource access mode: <https://apps.apple.com/us/app/reading-list-book-tracker/id1217139955>.
3. Turn - Reading Tracker & Timer [Electronic resource] – Resource access mode: <https://play.google.com/store/apps/details?id=com.jackwradford.readingtracker>.
4. Bookmory - reading tracker [Electronic resource] – Resource access mode: <https://play.google.com/store/apps/details?id=net.tonysoft.bookmory>.
5. UMLet - Free UML Tools for fast UML diagrams [Electronic resource] – Resource access mode: <https://www.umlet.com>.
6. Advantages of C# [Electronic resource] – Resource access mode: <https://www.codeguru.com/csharp/c-sharp-advantages>.
7. What Is An IDE (Integrated Development Environment)? [Electronic resource] – Resource access mode: <https://aws.amazon.com/what-is/ide/>.
8. Introduction to Visual Studio [Electronic resource] – Resource access mode: <https://www.geeksforgeeks.org/introduction-to-visual-studio/>.
9. What is .NET Framework? Explain Architecture & Components [Electronic resource] – Resource access mode: <https://www.guru99.com/net-framework.html>.
10. Context diagram | IST Project Management Office - University [Electronic resource] – Resource access mode: <https://uwaterloo.ca/ist-project-management-office/tools-and-templates/tools/context-diagram#:~:text=Context%20diagrams%20show%20the%20interactions,system%20will%20be%20part%20of>.
- 11 Use-case diagrams [Электронный ресурс] – Режим доступа до ресурсу: <https://www.ibm.com/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case>.
12. Activity diagram [Electronic resource] – Resource access mode: https://en.wikipedia.org/wiki/Activity_diagram.
13. Sequence diagram [Electronic resource] – Resource access mode: https://en.wikipedia.org/wiki/Sequence_diagram.


14. Class diagram [Electronic resource] – Resource access mode: https://en.wikipedia.org/wiki/Class_diagram.

15. All You Need to Know about State Diagrams - Visual Paradigm [Electronic resource] – Resource access mode: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/about-state-diagrams>.

ANNEX A



STATE UNIVERSITY OF TELECOMMUNICATIONS
Educational and scientific institute of INFORMATION
TECHNOLOGIES
Department of software engineering



Creation of book collection application using the C#
language

Performed by a 4th year student
group PD-42
Idrissi Hesham
Head of work

Senior lecturer of the Department of Software Engineering Gamaniuk Igor Mykhailovych

Kyiv – 2023

PURPOSE, OBJECT AND SUBJECT OF RESEARCH

- **The purpose of the work:** Automatisation of the accounting process of books in the collection.
- **Object of study:** The process of accounting for books in the collection.
- **Subject of study:** Application for accounting of books in the collection.

TASKS OF THE GRADUATE THESIS

1. Analysis of existing similar applications
2. Define of the business model.
3. Define of functional requirements
4. Define of non-functional requirements
5. Develop application design
6. Develop the program architecture
7. Implement the application

3

ANALYSIS OF ANALOGUES



Reading List: Book Tracker



Turn - Reading Tracker & Timer



Bookmory - reading tracker

4

ANALYSIS OF ANALOGUES

	Reading List: Book Tracker	Turn - Reading Tracker & Timer	Bookmory - reading tracker	BookSea
Private	-	-	-	+
Simple using	+	-	+	+
Lightweight	+	+	-	+
Ease of maintenance	-	+	+	+

5

SOFTWARE REQUIREMENTS

Functional requirements:

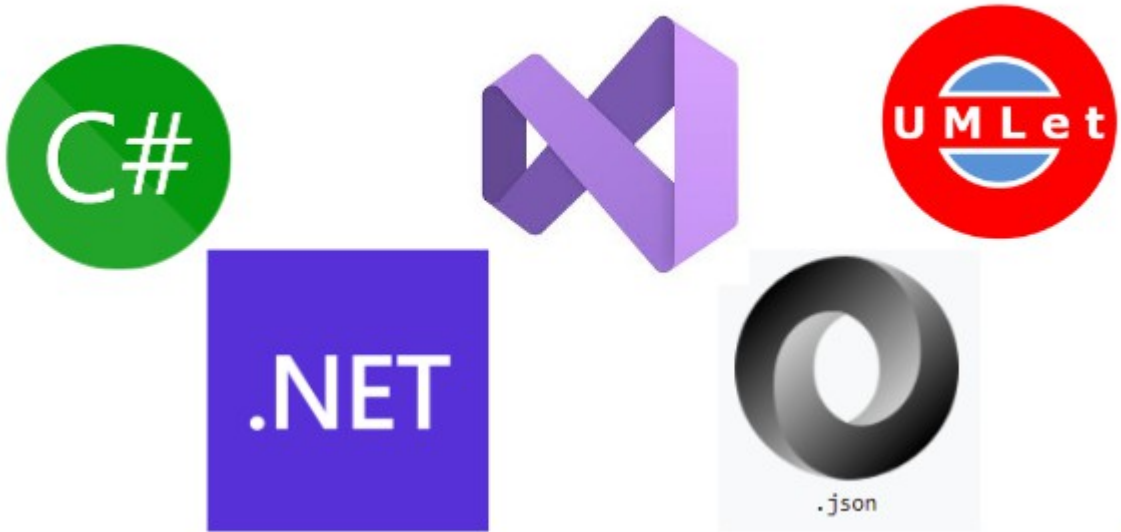
- Management of information about books
- Search for books by information
- Management of information about authors
- Search for authors by information

Non-functional requirements:

- Private, Simple using
- Lightweight
- .NET application
- JSON data interchange format

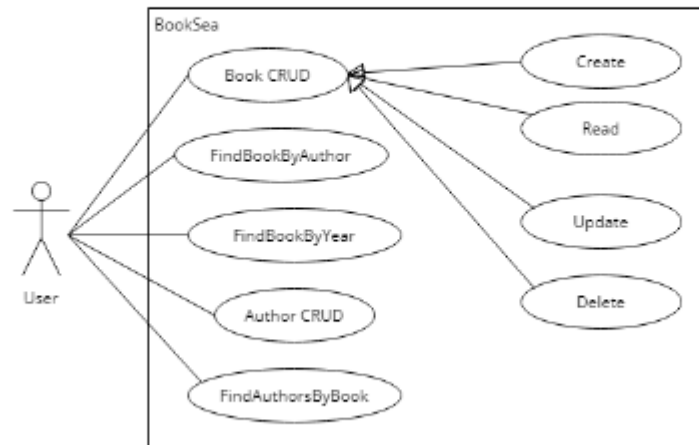
6

IMPLEMENTATION SOFTWARE



7

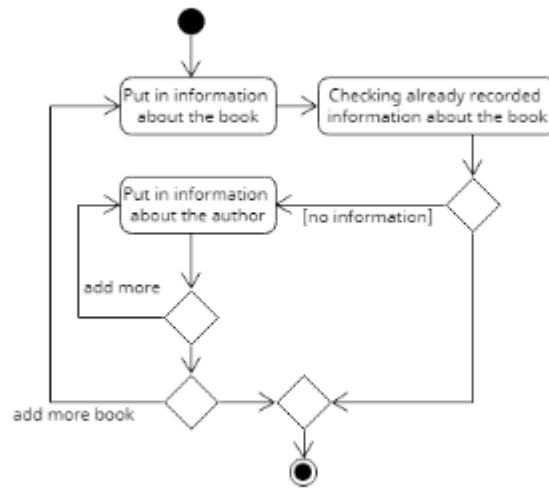
FUNCTIONAL REQUIREMENTS



Use case diagram

8

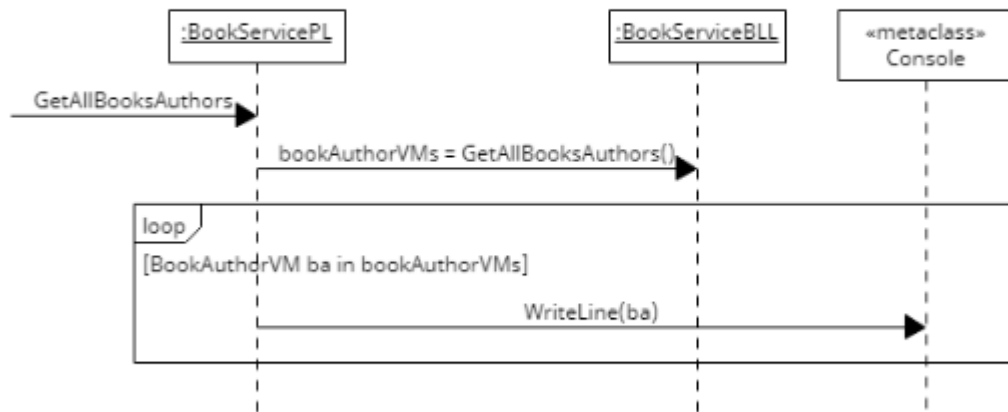
PUT IN INFORMATION ABOUT THE BOOK



Activity diagram

9

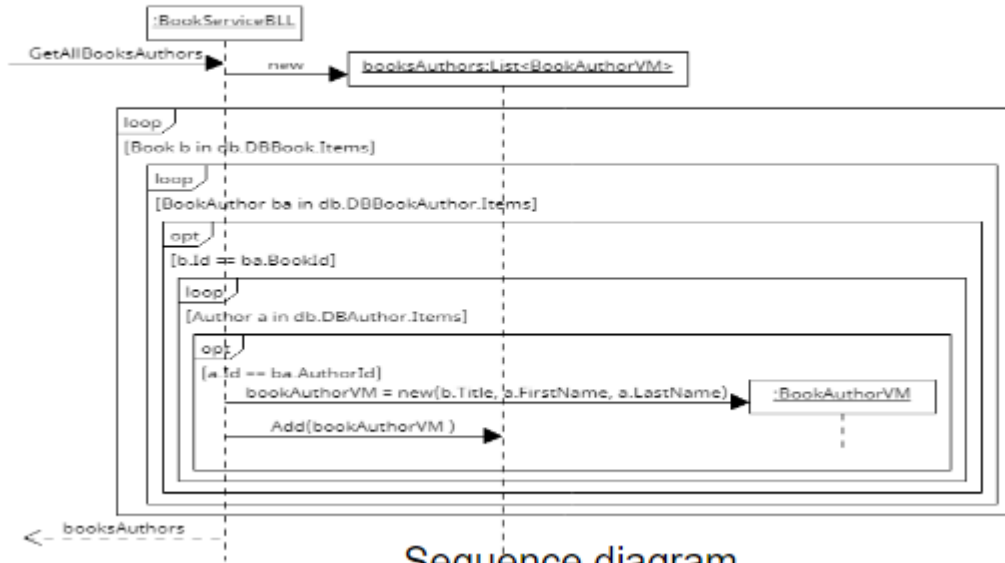
OBJECT INTERACTION. PRESENTATION LAYER



Sequence diagram

10

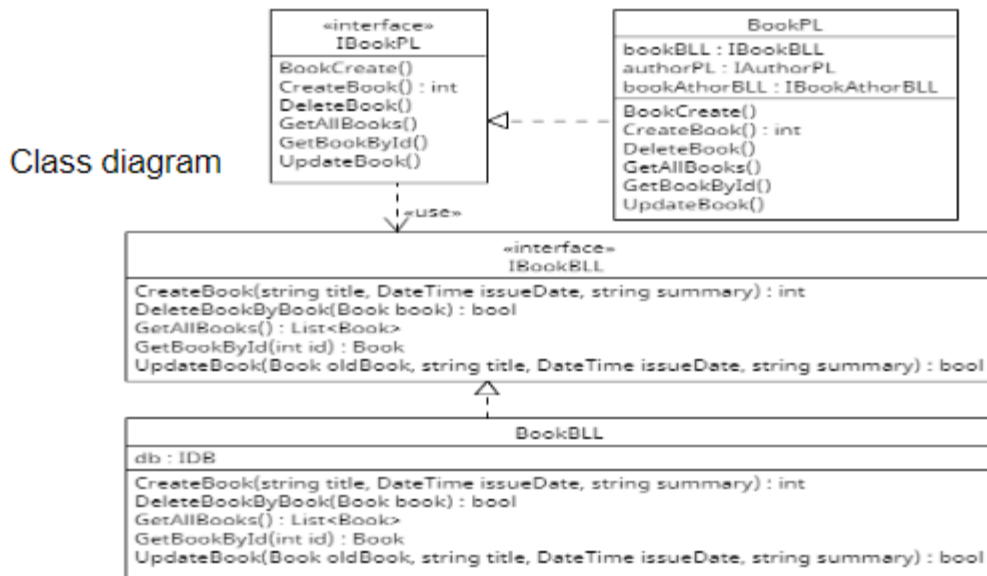
OBJECT INTERACTION. BUSINESS LOGIC LAYER



Sequence diagram

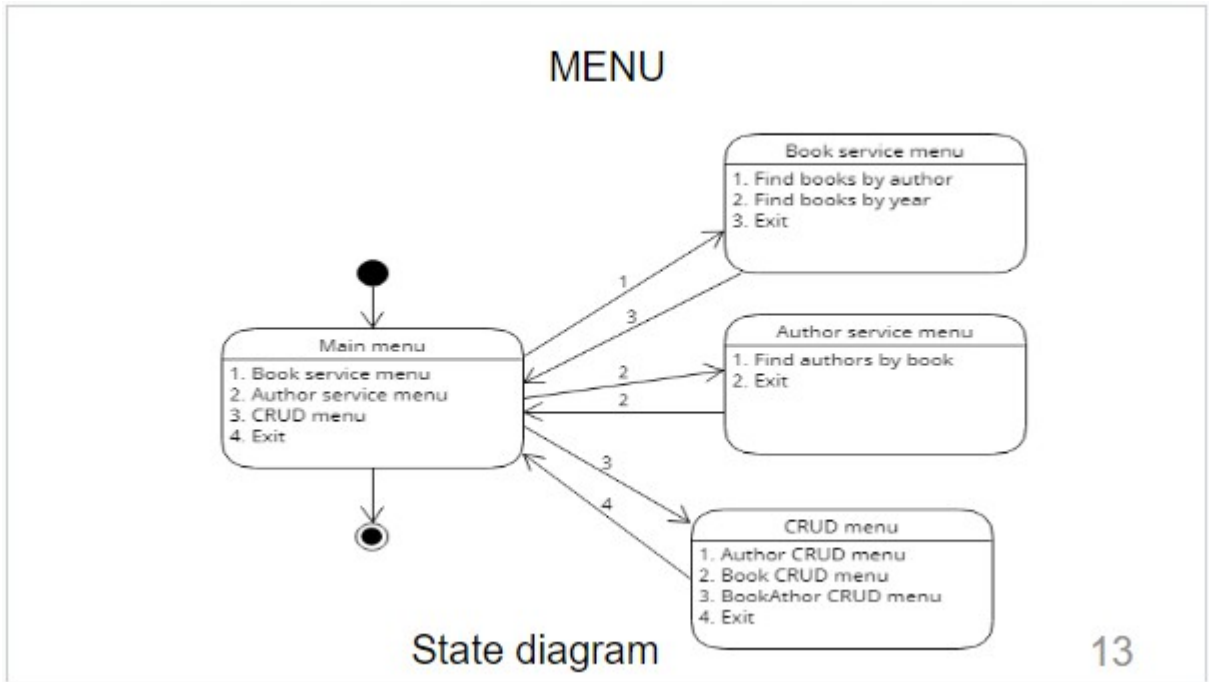
11

BOOK PRESENTATION AND BUSINESS LOGIC LAYERS

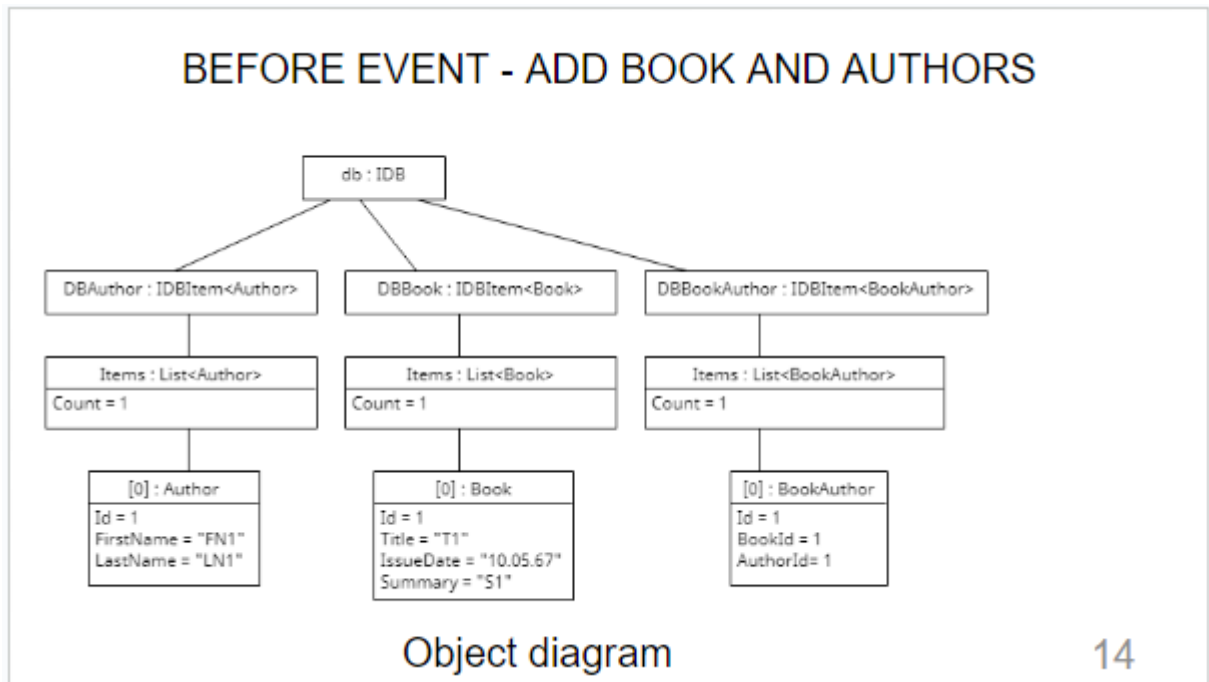


Class diagram

12



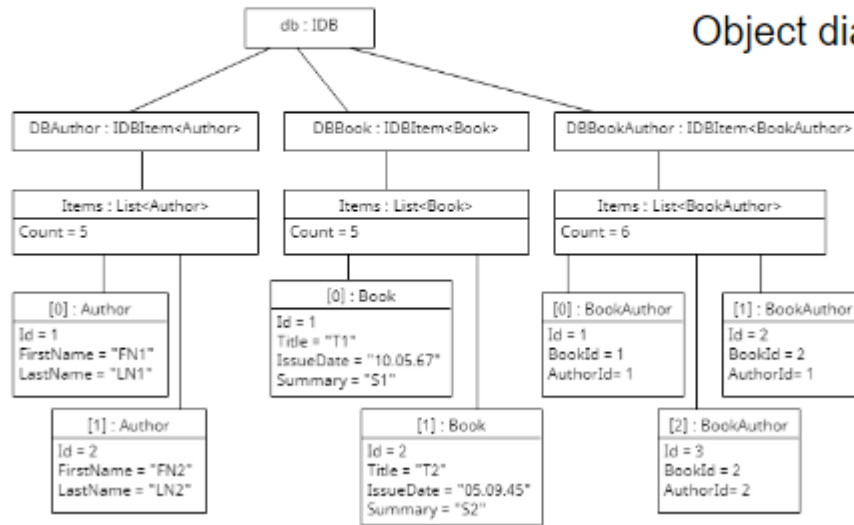
13



14

AFTER EVENT - ADD BOOK AND AUTHORS

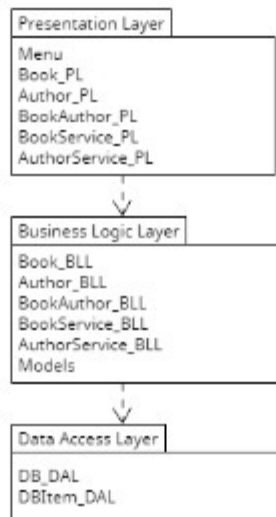
Object diagram



15

ARCHITECTURE DESIGN

Package diagram



16

APPROVAL OF RESEARCH RESULTS

1) Idricci Hesham T. Using a sequence diagram to simulate the case of getting a list of all books and the authors that written them into a book collection accounting application / T. Hesham Idrizzi , I.M. Gamaniuk// International scientific and practical conference "Modern aspects of digitization and informatization in software and computer engineering". Accepted for publication.

18

CONCLUSIONS

1. The model of the subject field (diagram of the subject field) has been modulated.
2. Modulated the case model (case set and use case diagram).
3. The design model (diagrams of activities, sequences, classes) has been modulated.
4. The application has been created.

19

THANK YOU FOR YOUR ATTENTION!

ANNEX B

```

using BookProject.BusinessLogicLayer.Models;
using BookProject.DataAccessLayer;

namespace BookProject.BusinessLogicLayer.Author_BLL
{
    internal class AuthorBLL : IAuthorBLL
    {
        IDB db;

        public AuthorBLL(IDB db)
        {
            this.db = db;
        }
        public int CreateAuthor(string firstName, string lastName)
        {
            Author author = new Author(firstName, lastName);
            return db.DBAuthor.AddItem(author);
        }
        public Author GetAuthorById(int id)
        {
            return db.DBAuthor.GetItemById(id);
        }
        public bool DeleteAuthorByAuthor(Author author)
        {
            return db.DBAuthor.DeleteItemByItem(author);
        }
        public bool UpdateAuthor(Author oldAuthor, string firstName, string lastName)
        {
            Author newAuthor = new Author(firstName, lastName);
            return db.DBAuthor.Update(oldAuthor, newAuthor);
        }
        public List<Author> GetAllAuthors()
        {
            return db.DBAuthor.Items;
        }
    }
}
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.BusinessLogicLayer.Author_BLL
{
    internal interface IAuthorBLL
    {
        int CreateAuthor(string firstName, string lastName);
        bool DeleteAuthorByAuthor(Author author);
        List<Author> GetAllAuthors();
        Author GetAuthorById(int id);
        bool UpdateAuthor(Author oldAuthor, string firstName, string lastName);
    }
}
using BookProject.BusinessLogicLayer.Models;
using BookProject.DataAccessLayer;

namespace BookProject.BusinessLogicLayer.AuthorServiceBLL
{
    internal class AuthorServiceBLL : IAuthorServiceBLL

```

```

{
    IDB db;

    public AuthorServiceBLL(IDB db)
    {
        this.db = db;
    }
    public List<Author> GetAuthorsByBook(int id)
    {
        List<Author> authors = new List<Author>();
        foreach (Book book in db.DBBook.Items)
        {
            if (book.Id == id)
            {
                foreach (BookAuthor ba in db.DBBookAuthor.Items)
                {
                    if (book.Id == ba.BookId)
                    {
                        foreach (Author a in db.DBAuthor.Items)
                        {
                            if (a.Id == ba.AuthorId)
                            {
                                authors.Add(a);
                            }
                        }
                    }
                }
            }
        }
        return authors;
    }
}
}
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.BusinessLogicLayer.AuthorServiceBLL
{
    internal interface IAuthorServiceBLL
    {
        List<Author> GetAuthorsByBook(int id);
    }
}
using BookProject.BusinessLogicLayer.Models;
using BookProject.DataAccessLayer;

namespace BookProject.BusinessLogicLayer.Book_BLL
{
    internal class BookBLL : IBookBLL
    {
        IDB db;

        public BookBLL(IDB db)
        {
            this.db = db;
        }

        public int CreateBook(string title, DateTime issueDate, string summary)
        {

```

```

        Book book = new Book(title, issueDate, summary);
        return db.DBBook.AddItem(book);
    }
    public Book GetBookById(int id)
    {
        return db.DBBook.GetItemById(id);
    }
    public bool DeleteBookByBook(Book book)
    {
        return db.DBBook.DeleteItemByItem(book);
    }
    public bool UpdateBook(Book oldBook, string title, DateTime issueDate, string
summary)
    {
        Book newBook = new Book(title, issueDate, summary);
        return db.DBBook.Update(oldBook, newBook);
    }
    public List<Book> GetAllBooks()
    {
        return db.DBBook.Items;
    }
    public bool CheckBookAvailability(string title, DateTime issueDate)
    {
        bool isAvailable = false;
        foreach (Book b in db.DBBook.Items)
        {
            if (b.Title == title && b.IssueDate == issueDate)
            {
                isAvailable = true;
                break;
            }
        }
        return isAvailable;
    }
}
}
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.BusinessLogicLayer.Book.BLL
{
    internal interface IBookBLL
    {
        int CreateBook(string title, DateTime issueDate, string summary);
        bool DeleteBookByBook(Book book);
        List<Book> GetAllBooks();
        Book GetBookById(int id);
        bool UpdateBook(Book oldBook, string title, DateTime issueDate, string summary);
        bool CheckBookAvailability(string title, DateTime issueDate);
    }
}
using BookProject.BusinessLogicLayer.Models;
using BookProject.DataAccessLayer;

namespace BookProject.BusinessLogicLayer.BookAuthor.BLL
{
    internal class BookAthorBLL : IBookAthorBLL
    {
        IDB db;
    }
}

```

```

public BookAThorBLL(IDB db)
{
    this.db = db;
}

public int CreateBookAuthor(int bookId, int authorId)
{
    BookAuthor bookAuthor = new BookAuthor(bookId, authorId);
    return db.DBBookAuthor.AddItem(bookAuthor);
}
public BookAuthor GetBookAuthorById(int id)
{
    return db.DBBookAuthor.GetItemById(id);
}
public bool DeleteBookAuthorByBookAuthor(BookAuthor bookAuthor)
{
    return db.DBBookAuthor.DeleteItemByItem(bookAuthor);
}
public bool UpdateBookAuthor(BookAuthor oldBookAuthor, int bookId, int authorId)
{
    BookAuthor newBookAuthor = new BookAuthor(bookId, authorId);
    return db.DBBookAuthor.Update(oldBookAuthor, newBookAuthor);
}
public List<BookAuthor> GetAllBookAuthors()
{
    return db.DBBookAuthor.Items;
}
}
}
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.BusinessLogicLayer.BookAuthor_BLL
{
    internal interface IBookAThorBLL
    {
        int CreateBookAuthor(int bookId, int authorId);
        bool DeleteBookAuthorByBookAuthor(BookAuthor bookAuthor);
        List<BookAuthor> GetAllBookAuthors();
        BookAuthor GetBookAuthorById(int id);
        bool UpdateBookAuthor(BookAuthor oldBookAuthor, int bookId, int authorId);
    }
}
using BookProject.BusinessLogicLayer.Models;
using BookProject.DataAccessLayer;

namespace BookProject.BusinessLogicLayer.BookService_BLL
{
    internal class BookServiceBLL : IBookServiceBLL
    {
        IDB db;

        public BookServiceBLL(IDB db)
        {
            this.db = db;
        }
        public List<BookAuthorVM> GetAllBooksAuthors()
        {

```

```

List<BookAuthorVM> booksAuthors = new List<BookAuthorVM>();
foreach (Book b in db.DBBook.Items)
{
    foreach (BookAuthor ba in db.DBBookAuthor.Items)
    {
        if (b.Id == ba.BookId)
        {
            foreach (Author a in db.DBAuthor.Items)
            {
                if (a.Id == ba.AuthorId)
                {
                    BookAuthorVM bookAuthorVM = new BookAuthorVM(b.Title,
a.FirstName, a.LastName);
                    booksAuthors.Add(bookAuthorVM);
                }
            }
        }
    }
}
return booksAuthors;
}
public List<Book> GetBooksByAuthorId(int id)
{
    List<Book> booksByAuthorId = new List<Book>();
    foreach (Author a in db.DBAuthor.Items)
    {
        if (a.Id == id)
        {
            foreach (BookAuthor ba in db.DBBookAuthor.Items)
            {
                if (a.Id == ba.AuthorId)
                {
                    foreach (Book b in db.DBBook.Items)
                    {
                        if (b.Id == ba.BookId)
                        {
                            booksByAuthorId.Add(b);
                        }
                    }
                }
            }
        }
    }
    return booksByAuthorId;
}
public List<Book> GetBooksByYear(int year)
{
    List<Book> books = new List<Book>();
    foreach (Book b in db.DBBook.Items)
    {
        if (b.IssueDate.Year == year)
        {
            books.Add(b);
        }
    }
    return books;
}
}

```

```

}
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.BusinessLogicLayer.BookService_BLL
{
    internal interface IBookServiceBLL
    {
        List<BookAuthorVM> GetAllBooksAuthors();
        List<Book> GetBooksByAuthorId(int id);
        List<Book> GetBooksByYear(int year);
    }
}
namespace BookProject.BusinessLogicLayer.Models
{
    internal class Author : IID
    {
        public int Id { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }

        public Author(string firstName, string lastName)
        {
            FirstName = firstName;
            LastName = lastName;
        }

        public Author()
        {
        }

        public override string ToString()
        {
            return string.Format(Id + " " + FirstName + " " + LastName);
        }
    }
}
namespace BookProject.BusinessLogicLayer.Models
{
    internal class Book : IID
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public DateTime IssueDate { get; set; }
        public string Summary { get; set; }

        public Book(string title, DateTime issueDate, string summary)
        {
            Title = title;
            IssueDate = issueDate;
            Summary = summary;
        }

        public Book()
        {
        }

        public override string ToString()
        {

```

```

        return string.Format("\n" + Id + " " + Title + " " +
IssueDate.ToShortDateString() + "\n" + Summary);
    }
}
namespace BookProject.BusinessLogicLayer.Models
{
    internal class BookAuthor : IId
    {
        public int Id { get; set; }
        public int BookId { get; set; }
        public int AuthorId { get; set; }

        public BookAuthor(int bookId, int authorId)
        {
            BookId = bookId;
            AuthorId = authorId;
        }

        public BookAuthor()
        {
        }

        public override string ToString()
        {
            return string.Format(Id + " " + BookId + " " + AuthorId);
        }
    }
}
namespace BookProject.BusinessLogicLayer.Models
{
    internal class BookAuthorVM
    {
        public string Title { get; set; }
        public string AuthorFirstName { get; set; }
        public string AuthorLastName { get; set; }

        public BookAuthorVM(string title, string authorFirstName, string authorLastName)
        {
            Title = title;
            AuthorFirstName = authorFirstName;
            AuthorLastName = authorLastName;
        }
        public override string ToString()
        {
            return string.Format(Title + " " + AuthorFirstName + " " + AuthorLastName);
        }
    }
}
namespace BookProject.BusinessLogicLayer.Models
{
    internal interface IId
    {
        int Id { get; set; }
    }
}
namespace BookProject.DataAccessLayer
{

```



```

internal class Configuration
{
    public static string DBPath { get; set; } = @"C:\Temp2\BookDB";
}
}
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.DataAccessLayer
{
    internal class DB : IDB
    {
        public DBItem<Book> DBBook { get; set; } = new DBItem<Book>();
        public DBItem<Author> DBAuthor { get; set; } = new DBItem<Author>();
        public DBItem<BookAuthor> DBBookAuthor { get; set; } = new DBItem<BookAuthor>();
        public DB()
        {
            Initialize();
        }
        public void DbSerialize()
        {
            DBBook.SerializeJSON();
            DBAuthor.SerializeJSON();
            DBBookAuthor.SerializeJSON();
        }
        private void Initialize()
        {
            DBBook.DeserializeJSON();
            if (DBBook.Items.Count == 0)
            {
                BookInitialize();
            }
            DBAuthor.DeserializeJSON();
            if (DBAuthor.Items.Count == 0)
            {
                AuthorInitialize();
            }
            DBBookAuthor.DeserializeJSON();
            if (DBBookAuthor.Items.Count == 0)
            {
                BookAuthorInitialize();
            }
        }
        private void BookInitialize()
        {
            Book book1 = new Book("C# 8.0 and .NET Core 3.0", DateTime.Parse("31 Oct.
2019"), "In C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development, Fourth Edition,
expert teacher Mark J. Price gives you everything you need to start programming C#
applications.");
            DBBook.AddItem(book1);
            Book book2 = new Book("Head First C#", DateTime.Parse("26 Jan. 2021"), "Dive
into C# and create apps, user interfaces, games, and more using this fun and highly
visual introduction to C#, .NET Core, and Visual Studio.");
            DBBook.AddItem(book2);
            Book book3 = new Book("C# 8.0 Pocket Reference", DateTime.Parse("10 Dec.
2019"), "When you need answers about using C# 8.0, this tightly focused and practical
book tells you exactly what you need to know without long intros or bloated samples.");
            DBBook.AddItem(book3);
        }
    }
}

```

```

private void AuthorInitialize()
{
    Author author1 = new Author("Mark", "J. Price");
    DBAuthor.AddItem(author1);
    Author author2 = new Author("Andrew", "Stellman");
    DBAuthor.AddItem(author2);
    Author author3 = new Author("Jospeh", "Albahari");
    DBAuthor.AddItem(author3);
    Author author4 = new Author("Ben", "Albahari");
    DBAuthor.AddItem(author4);
}
private void BookAuthorInitialize()
{
    BookAuthor bookAuthor1 = new BookAuthor(1, 1);
    DBBookAuthor.AddItem(bookAuthor1);
    BookAuthor bookAuthor2 = new BookAuthor(2, 2);
    DBBookAuthor.AddItem(bookAuthor2);
    BookAuthor bookAuthor3 = new BookAuthor(3, 3);
    DBBookAuthor.AddItem(bookAuthor3);
    BookAuthor bookAuthor4 = new BookAuthor(3, 4);
    DBBookAuthor.AddItem(bookAuthor4);
}
}
}
using System.Text.Json;
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.DataAccessLayer
{
    internal class DBItem<T> : IDBItem<T> where T : IID
    {
        private int counter = 1;
        public List<T> Items { get; set; } = new List<T>();
        public int AddItem(T item)
        {
            item.Id = counter++;
            Items.Add(item);
            return item.Id;
        }
        public T GetItemById(int id)
        {
            T result = default(T);
            foreach (T item in Items)
            {
                if (item.Id == id)
                {
                    result = item;
                    break;
                }
            }
            return result;
        }
        public bool DeleteItemByItem(T item)
        {
            return Items.Remove(item);
        }
        public bool Update(T oldItem, T newItem)
        {

```

```

        newItem.Id = oldItem.Id;
        bool result = Items.Remove(oldItem);
        Items.Add(newItem);
        return result;
    }
    public bool SerializeJSON()
    {
        //string path = @"C:\Temp2\BookDB";
        string path = Configuration.DBPath;
        if (Directory.Exists(path))
        {
            path = path + @"\";
        }
        else
        {
            path = string.Empty;
        }
        string jsonString = string.Empty;
        string fileName = String.Format(path + Items.ToString() + ".json");
        try
        {
            jsonString = JsonSerializer.Serialize(Items);

            File.WriteAllText(fileName, jsonString);
        }
        catch (Exception ex) { Console.WriteLine(ex.Message); return false; }
        return true;
    }
    public List<T> DeserializeJSON()
    {
        string path = Configuration.DBPath;
        if (Directory.Exists(path))
        {
            path = path + @"\";
        }
        else
        {
            path = string.Empty;
        }
        string jsonString = string.Empty;
        try
        {
            jsonString = File.ReadAllText(String.Format(path + Items.ToString() +
".json"));

            Items = JsonSerializer.Deserialize<List<T>>(jsonString);
            counter = GetMaxId();
            return Items;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine("Verify that entity classes and their base classes have
a default constructor");
            return null;
        }
    }
    private int GetMaxId()
    {

```

```

        int maxId = 1;
        foreach (T item in Items)
        {
            if (item.Id > maxId)
            {
                maxId = item.Id;
            }
        }
        return (maxId + 1);
    }
}
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.DataAccessLayer
{
    internal interface IDB
    {
        DBItem<Author> DBAuthor { get; set; }
        DBItem<Book> DBBook { get; set; }
        DBItem<BookAuthor> DBBookAuthor { get; set; }
        void DbSerialize();
    }
}
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.DataAccessLayer
{
    internal interface IDBItem<T> where T : IIId
    {
        List<T> Items { get; set; }

        int AddItem(T item);
        bool DeleteItemByItem(T item);
        T GetItemById(int id);
        bool Update(T oldItem, T newItem);
        bool SerializeJSON();
        List<T> DeserializeJSON();
    }
}
using BookProject.BusinessLogicLayer.Author_BLL;
using BookProject.BusinessLogicLayer.BookAuthor_BLL;
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.PresentationLayer.Author_PL
{
    internal class AuthorPL : IAuthorPL
    {
        IAuthorBLL authorBLL;
        IBookAthorBLL bookAthorBLL;

        public AuthorPL(IAuthorBLL authorBLL, IBookAthorBLL bookAthorBLL)
        {
            this.authorBLL = authorBLL;
            this.bookAthorBLL = bookAthorBLL;
        }
        public int CreateAuthor()
        {

```

```

        string firstName = Helper.StringInputCheck("CreateAuthor Author FirstName?: ", 30);
        string lastName = Helper.StringInputCheck("CreateAuthor Author LastName?: ", 30);
        return authorBLL.CreateAuthor(firstName, lastName);
    }
    public int AuthorCreate(int bookId)
    {
        int athorId = CreateAuthor();
        return bookAThorBLL.CreateBookAuthor(bookId, athorId);
    }
    public void GetAllAuthors()
    {
        List<Author> authors = authorBLL.GetAllAuthors();
        foreach (Author author in authors)
        {
            Console.WriteLine(author);
        }
    }
}
namespace BookProject.PresentationLayer.Author_PL
{
    internal interface IAuthorPL
    {
        int AuthorCreate(int bookId);
        int CreateAuthor();
        void GetAllAuthors();
    }
}
using BookProject.BusinessLogicLayer.AuthorServiceBLL;
using BookProject.BusinessLogicLayer.Book_BLL;
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.PresentationLayer.AuthorService_PL
{
    internal class AuthorServicePL : IAuthorServicePL
    {
        IAuthorServiceBLL authorServiceBLL;
        IBookBLL bookBLL;

        public AuthorServicePL(IAuthorServiceBLL authorServiceBLL, IBookBLL bookBLL)
        {
            this.authorServiceBLL = authorServiceBLL;
            this.bookBLL = bookBLL;
        }
        public void GetAuthorsByBook()
        {
            List<Book> books = bookBLL.GetAllBooks();
            foreach (Book b in books)
            {
                Console.WriteLine(b);
            }
            int id = Helper.IntInputCheck("GetAuthorsByBook Book Id?: ");
            List<Author> authors = authorServiceBLL.GetAuthorsByBook(id);
            foreach (Author a in authors)
            {
                Console.WriteLine(a);
            }
        }
    }
}

```

```

    }
}
}
namespace BookProject.PresentationLayer.AuthorService_PL
{
    internal interface IAuthorServicePL
    {
        void GetAuthorsByBook();
    }
}
using BookProject.BusinessLogicLayer.Book_BLL;
using BookProject.BusinessLogicLayer.BookAuthor_BLL;
using BookProject.BusinessLogicLayer.Models;
using BookProject.PresentationLayer.Author_PL;

namespace BookProject.PresentationLayer.Book_PL
{
    internal class BookPL : IBookPL
    {
        IBookBLL bookBLL;
        IAuthorPL authorPL;
        IBookAthorBLL bookAthorBLL;

        public BookPL(IBookBLL bookBLL, IAuthorPL authorPL, IBookAthorBLL bookAthorBLL)
        {
            this.bookBLL = bookBLL;
            this.authorPL = authorPL;
            this.bookAthorBLL = bookAthorBLL;
        }
        private bool CheckBookAvailability(string title, DateTime issueDate)
        {
            return bookBLL.CheckBookAvailability(title, issueDate);
        }
        public void CreateBook(bool addAuthor)
        {
            string title = Helper.StringInputCheck("Create Book Title?: ", 50);
            DateTime issueDate = Helper.DateTimeInputCheck("Issue Date?: ");

            bool isAvailable = CheckBookAvailability(title, issueDate);

            if (!isAvailable)
            {
                string summary = Helper.StringInputCheck("Summary?: ", 150);

                int bookId = bookBLL.CreateBook(title, issueDate, summary);
                while (addAuthor)
                {
                    int authorId = authorPL.CreateAuthor();
                    bookAthorBLL.CreateBookAuthor(bookId, authorId);
                    Console.WriteLine("Add more authors?(y/n): ");
                    if ("y" != Console.ReadLine())
                    {
                        addAuthor = false;
                    }
                }
            }
            else { Console.WriteLine("The book is Available"); }
        }
    }
}

```

```

}
public void GetBookById()
{
    int id = Helper.IntInputCheck("Get Book by Id. Id?: ");
    Book book = bookBLL.GetBookById(id);
    if (book != default(Book))
    {
        Console.WriteLine(book);
    }
    else
    {
        Console.WriteLine("Not found");
    }
}
public void DeleteBook()
{
    int id = Helper.IntInputCheck("Delete Book by Id. Id?: ");
    Book book = bookBLL.GetBookById(id);
    if (book != default(Book))
    {
        Console.WriteLine(book);
        Console.WriteLine(" Delete?(y/n): ");
        if ("y" == Console.ReadLine())
        {
            if (bookBLL.DeleteBookByBook(book))
            {
                Console.WriteLine("Deleted");
            }
            else
            {
                Console.WriteLine("Not Deleted");
            }
        }
    }
    else
    {
        Console.WriteLine("Not found");
    }
}

public void GetAllBooks()
{
    foreach (Book book in bookBLL.GetAllBooks())
    {
        Console.WriteLine(book);
    }
}
}
namespace BookProject.PresentationLayer.Book_PL
{
    internal interface IBookPL
    {
        void CreateBook(bool addAuthor);
        void DeleteBook();
        void GetAllBooks();
        void GetBookById();
    }
}

```

```

}
using BookProject.BusinessLogicLayer.Author_BLL;
using BookProject.BusinessLogicLayer.BookService_BLL;
using BookProject.BusinessLogicLayer.Models;

namespace BookProject.PresentationLayer.BookService_PL
{
    internal class BookServicePL : IBookServicePL
    {
        IBookServiceBLL bookServiceBLL;
        IAuthorBLL authorBLL;

        public BookServicePL(IBookServiceBLL bookServiceBLL, IAuthorBLL authorBLL)
        {
            this.bookServiceBLL = bookServiceBLL;
            this.authorBLL = authorBLL;
        }
        public void GetAllBooksAuthors()
        {
            List<BookAuthorVM> bookAuthorVMs = bookServiceBLL.GetAllBooksAuthors();
            foreach (BookAuthorVM ba in bookAuthorVMs)
            {
                Console.WriteLine(ba);
            }
        }
        public void GetBooksByAuthor()
        {
            List<Author> authors = authorBLL.GetAllAuthors();
            foreach (Author a in authors)
            {
                Console.WriteLine(a);
            }
            int id = Helper.IntInputCheck("Get Books By Author Id?: ");
            List<Book> books = bookServiceBLL.GetBooksByAuthorId(id);
            foreach (Book b in books)
            {
                Console.WriteLine(b);
            }
        }
        public void GetBooksByYear()
        {
            int year = Helper.IntInputCheck("Get Books By Year Year?: ");
            List<Book> books = bookServiceBLL.GetBooksByYear(year);
            foreach (Book b in books)
            {
                Console.WriteLine(b);
            }
        }
    }
}
namespace BookProject.PresentationLayer.BookService_PL
{
    internal interface IBookServicePL
    {
        void GetAllBooksAuthors();
        void GetBooksByAuthor();
        void GetBooksByYear();
    }
}

```



```

}
using BookProject.PresentationLayer.Author_PL;

namespace BookProject.PresentationLayer.Menu
{
    internal class AuthorCRUDMenu : IMenu
    {
        IAuthorPL authorPL;

        public AuthorCRUDMenu(IAuthorPL authorPL)
        {
            this.authorPL = authorPL;
        }
        public void Run()
        {
            bool flag = true;
            while (flag)
            {
                Console.WriteLine("1 - Create Author");
                Console.WriteLine("2 - Get All Authors");
                Console.WriteLine("3 - Get Author by Id");
                Console.WriteLine("4 - Delete Author");
                Console.WriteLine("5 - Update Author");
                Console.WriteLine("6 - Exit");
                int menuNumber = Helper.IntInputCheck("-> ");
                switch (menuNumber)
                {
                    case 1:
                        authorPL.CreateAuthor();
                        break;
                    case 2:
                        authorPL.GetAllAuthors();
                        break;
                    case 3:
                        Console.WriteLine("Get Author by Id");
                        break;
                    case 4:
                        Console.WriteLine("Delete Author");
                        break;
                    case 5:
                        Console.WriteLine("Update Author");
                        break;
                    case 6:
                        flag = false;
                        break;
                    default:
                        Console.WriteLine("Error");
                        break;
                }
            }
        }
    }
}
using BookProject.PresentationLayer.AuthorService_PL;

namespace BookProject.PresentationLayer.Menu
{
    internal class AuthorServiceMenu : IMenu

```

```

{
    IAuthServicePL authServicePL;

    public AuthorServiceMenu(IAuthServicePL authServicePL)
    {
        this.authServicePL = authServicePL;
    }

    public void Run()
    {
        bool flag = true;
        while (flag)
        {
            Console.WriteLine("1 - Find Authors By Book");
            Console.WriteLine("2 - Exit");
            int menuNumber = Helper.IntInputCheck("-> ");
            switch (menuNumber)
            {
                case 1:
                    authServicePL.GetAuthorsByBook();
                    break;
                case 2:
                    flag = false;
                    break;
                default:
                    Console.WriteLine("Error");
                    break;
            }
        }
    }
}

using BookProject.PresentationLayer.Book_PL;
using BookProject.PresentationLayer.BookService_PL;

namespace BookProject.PresentationLayer.Menu
{
    internal class BookCRUDMenu : IMenu
    {
        IBookPL bookPL;
        IBookServicePL bookServicePL;

        public BookCRUDMenu(IBookPL bookPL, IBookServicePL bookServicePL)
        {
            this.bookPL = bookPL;
            this.bookServicePL = bookServicePL;
        }

        public void Run()
        {
            bool flag = true;
            while (flag)
            {
                Console.WriteLine("1 - Create Book");
                Console.WriteLine("2 - Create Book and Author");
                Console.WriteLine("3 - Get All Books");
                Console.WriteLine("4 - Get All Books and Authors");
                Console.WriteLine("5 - Get Book by Id");
            }
        }
    }
}

```

```

        Console.WriteLine("6 - Delete Book");
        Console.WriteLine("7 - Update Book");
        Console.WriteLine("8 - Exit");
        int menuNumber = Helper.IntInputCheck("-> ");
        switch (menuNumber)
        {
            case 1:
                bookPL.CreateBook(false);
                break;
            case 2:
                bookPL.CreateBook(true);
                break;
            case 3:
                bookPL.GetAllBooks();
                break;
            case 4:
                bookServicePL.GetAllBooksAuthors();
                break;
            case 5:
                bookPL.GetBookById();
                break;
            case 6:
                bookPL.DeleteBook();
                break;
            case 7:
                Console.WriteLine("Update Book");
                break;
            case 8:
                flag = false;
                break;
            default:
                Console.WriteLine("Error");
                break;
        }
    }
}
}
}
using BookProject.PresentationLayer.BookService_PL;

namespace BookProject.PresentationLayer.Menu
{
    internal class BookServiceMenu : IMenu
    {
        IBookServicePL bookServicePL;

        public BookServiceMenu(IBookServicePL bookServicePL)
        {
            this.bookServicePL = bookServicePL;
        }

        public void Run()
        {
            bool flag = true;
            while (flag)
            {
                Console.WriteLine("1 - Find books by author");
                Console.WriteLine("2 - Find books by year");
            }
        }
    }
}

```



```

        Console.WriteLine("Error");
        break;
    }
}
}
}
}
namespace BookProject.PresentationLayer.Menu
{
    internal class MainMenu : IMenu
    {
        IMenu bookServiceMenu;
        IMenu authorServiceMenu;
        IMenu crudMenu;
        public MainMenu(IMenu bookServiceMenu, IMenu authorServiceMenu, IMenu crudMenu) {
            this.bookServiceMenu = bookServiceMenu;
            this.authorServiceMenu = authorServiceMenu;
            this.crudMenu = crudMenu;
        }
        public void Run() {
            bool flag = true;
            while (flag)
            {
                Console.WriteLine("1 - Book Service Menu");
                Console.WriteLine("2 - Author Service Menu");
                Console.WriteLine("3 - CRUD Menu");
                Console.WriteLine("4 - Exit");
                int menuNumber = Helper.IntInputCheck("-> ");
                switch (menuNumber) {
                    case 1:
                        bookServiceMenu.Run();
                        break;
                    case 2:
                        authorServiceMenu.Run();
                        break;
                    case 3:
                        crudMenu.Run();
                        break;
                    case 4:
                        flag = false;
                        break;
                    default:
                        Console.WriteLine("Error");
                        break;
                }
            }
        }
    }
}
namespace BookProject.PresentationLayer.Menu
{
    internal interface IMenu
    {
        void Run();
    }
}
namespace BookProject.PresentationLayer
{

```

```

internal static class Helper
{
    public static string StringInputCheck(string message, int messageLength)
    {
        string str = "";
        bool flag = true;
        while (flag)
        {
            Console.Write(message);
            str = Console.ReadLine();
            if (str.Length < messageLength)
            {
                flag = false;
            }
            else
            {
                Console.WriteLine("No enough capacity please enter less characters");
            }
        }
        return str;
    }
    public static int IntInputCheck(string message)
    {
        int result = 0;
        bool flag = true;
        while (flag)
        {
            Console.Write(message);
            string str = Console.ReadLine();
            flag = !int.TryParse(str, out result);
            if (flag)
            {
                Console.WriteLine("It must be digits try again");
            }
        }
        return result;
    }
    public static DateTime DateTimeInputCheck(string message)
    {
        DateTime result = DateTime.Now;
        bool flag = true;
        while (flag)
        {
            Console.Write(message);
            string str = Console.ReadLine();
            flag = !DateTime.TryParse(str, out result);
            if (flag)
            {
                Console.WriteLine("It must be date try again");
            }
        }
        return result;
    }
    public static void AddMore(Action action)
    {
        bool flag = true;
        do
        {

```

```

        action();
        Console.WriteLine("Add more?(y/n): ");
        if ("y" != Console.ReadLine())
        {
            flag = false;
        }
    } while (flag);
}
}
}
using BookProject.BusinessLogicLayer.Author_BLL;
using BookProject.BusinessLogicLayer.AuthorServiceBLL;
using BookProject.BusinessLogicLayer.Book_BLL;
using BookProject.BusinessLogicLayer.BookAuthor_BLL;
using BookProject.BusinessLogicLayer.BookService_BLL;
using BookProject.DataAccessLayer;
using BookProject.PresentationLayer.Author_PL;
using BookProject.PresentationLayer.AuthorService_PL;
using BookProject.PresentationLayer.Book_PL;
using BookProject.PresentationLayer.BookService_PL;
using BookProject.PresentationLayer.Menu;

IDB db = new DB();

IBookAthorBLL bookAthorBLL = new BookAthorBLL(db);
IAuthorBLL authorBLL = new AuthorBLL(db);
IAuthorPL authorPL = new AuthorPL(authorBLL, bookAthorBLL);
IMenu authorCRUDMenu = new AuthorCRUDMenu(authorPL);

IBookBLL bookBLL = new BookBLL(db);
IBookPL bookPL = new BookPL(bookBLL, authorPL, bookAthorBLL);
IBookServiceBLL bookServiceBLL = new BookServiceBLL(db);
IBookServicePL bookServicePL = new BookServicePL(bookServiceBLL, authorBLL);
IMenu bookCRUDMenu = new BookCRUDMenu(bookPL, bookServicePL);
IMenu crudMenu = new CRUDMenu(authorCRUDMenu, bookCRUDMenu);
IMenu bookServiceMenu = new BookServiceMenu(bookServicePL);
IAuthorServiceBLL authorServiceBLL = new AuthorServiceBLL(db);
IAuthorServicePL authorServicePL = new AuthorServicePL(authorServiceBLL, bookBLL);
IMenu authorServiceMenu = new AuthorServiceMenu(authorServicePL);
IMenu mainMenu = new MainMenu(bookServiceMenu, authorServiceMenu, crudMenu);
mainMenu.Run();
db.DbSerialize();

Console.ReadKey();

```