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: «**CREATING SOFTWARE FOR LEARNING ENGLISH IDIOMS USING C#**»

Done by: student of the 4th year, group PD-42
121 Software engineering
(шифр і назва спеціальності/спеціалізації)

Sabir Habib
(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

SABIR HABIB
(прізвище, ім'я, по батькові)

1. The topic: «Creating software for learning English idioms using C#»
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
   4.1 Learning English idioms as a component of the learning process;
   4.2 Analaysis of available tools and technologies for organizing learning English idioms;
   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 idiom
5.9. Object iteraction
5.10. Idiom 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 _____ Sabir Habib
(підпис) (прізвище та ініціали)

Head of work _____ Gamaniuk I.M.
(підпис) (прізвище та ініціали)

ABSTRACT

The text part of the bachelor thesis: 41 pp., 2 table, 34 figures, 2 appendices, 10 sources.

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

In order to understand native speakers well, it is necessary to understand idioms. In order for native speakers to treat you with respect, it is necessary to use idioms. Therefore, it is necessary to pay a lot of attention to learning idioms.

Object of study: learning English idiom.

Subject of study: application for learning English idiom.

The purpose of the work: improving English language learning by using application for learning English idioms.

Research methods are a unified process of creating software.

Based on the results of the completed work, an application for learning English idiom was developed, which allows implementing information support and updating the database.

Using the created application will allow you to learn English idiom.

CONTENT

INTRODUCTION

Globalization processes capture more and more regions. A good knowledge of an international language comes to the fore. One such language is English. Native English speakers often use idioms in conversation.

In order to understand native speakers well, it is necessary to understand idioms. In order for native speakers to treat you with respect, it is necessary to use idioms. Therefore, it is necessary to pay a lot of attention to learning idioms.

Object of study: learning English idiom.

Subject of study: application for learning English idiom.

The purpose of the work: improving English language learning by using application for learning English idioms.

The following tasks were solved in the study:

- study of the method of organizing the learning English idiom;
- 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 learn English idiom.

1 LEARNING ENGLISH IDIOMS AS A COMPONENT OF THE LEARNING ENGLISH PROCESS

1.1 Learning English process

The process of learning English includes many disciplines. The main disciplines are:

- learning words;
- studying grammar;
- writing works;
- translation of texts;
- reading literature and newspapers;
- watching movies;
- study of dialogues;
- learning colloquial phrases;
- study of idioms;
- communication with native speakers;
- study of behavior patterns of native speakers;

and many others.

The process of learning English is iterative. At each iteration, it is necessary to pass the disciplines mentioned earlier. With each new iteration, the level of English language proficiency increases.

This process can be depicted using an activity diagram (Figure 1.1).

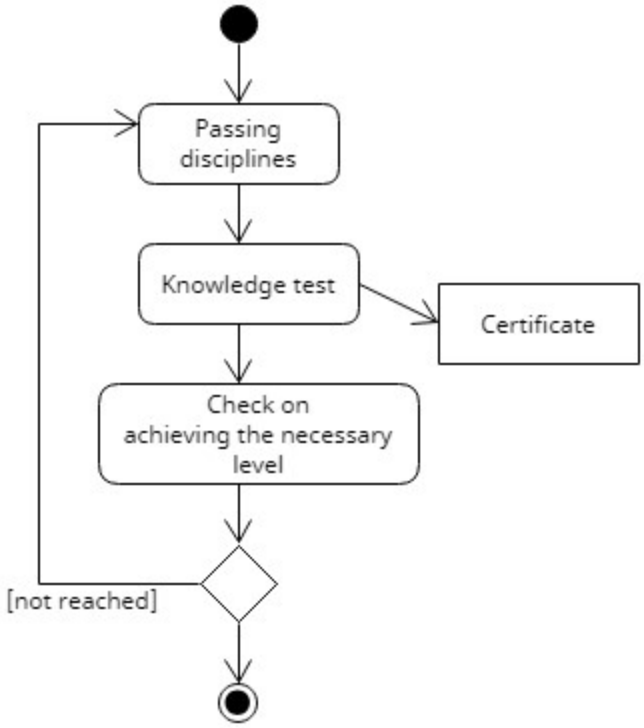


Figure 1.1 — Language learning process

1.2 Understanding the Importance of Idioms in English Language Learning

An idiom is a phrase or saying that is commonly used in everyday English to express certain ideas or opinions. Understanding English idioms is important because they require a deeper familiarity of the English language to comprehend what someone means when they use them in conversation.

Idioms may seem complicated at first, but they can actually be a lot of fun to learn. If you're interested in building your English skills, read on to find out why idioms are so important to your English language learning.

Idioms Give You a New Way to Express Yourself in the English Language.

The meaning of an idiom generally depends on the specific context in which it is used. When someone in America tells you to 'break a leg', for example, they aren't saying that in a literal sense, but instead are wishing you good luck, usually before a

performance. Similarly, if someone asks you to ‘think outside the box’, they mean that you should use a different approach than what you might normally do.

Idioms are particularly useful because they give you a new, creative way to express yourself. Rather than saying ‘You’re correct’, you could say ‘You hit the nail on the head’, which is a more complex and interesting expression. Idioms can also be quite humorous to use, which allows you to express yourself in a more genuine way, including showing off your personality and sense of humor [1].

1.3 Learning English idiom process

The learning English idiom process can be depicted in an activity diagram (Figure 1.2).

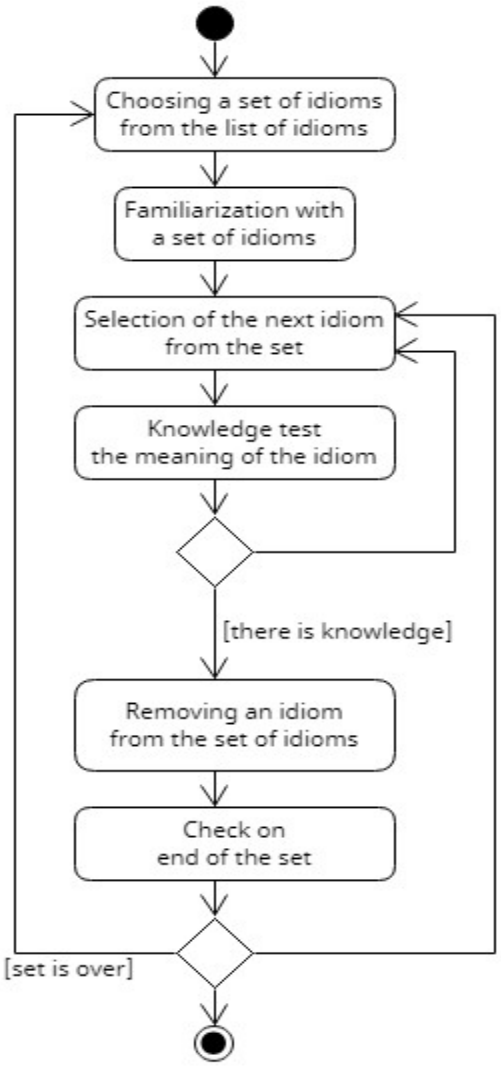


Figure 1.2 — The process of learning idioms

2 ANALAYSIS OF AVAILABLE TOOLS AND TECHNOLOGIES FOR ORGANIZING LEARNING ENGLISH IDIOMS

2.1 Analysis of existing programs for learning English idioms

The most popular programs in this niche are:

1. “All English Idioms & Phrases”

This is helpful app to you can learn Idioms , Phrases and Proverbs in English very easily and effectively. It has more than 10000 idioms and phrases, it will help you search easily and effectively [2].

The program has the following interface (Figure 2.1, 2.2):

Figure 2.1 – Main sections

Figure 2.2 – Categories of idioms

2. “English Idioms Cards”

Learn English idioms in a fun and easy way via 1000 flashcards with pictures and pronunciation.

In our app you will find 1000 flashcards containing idiom meanings, example sentences, pictures and correct pronunciation [3].

The program interface is as follows (Figure 2.3):

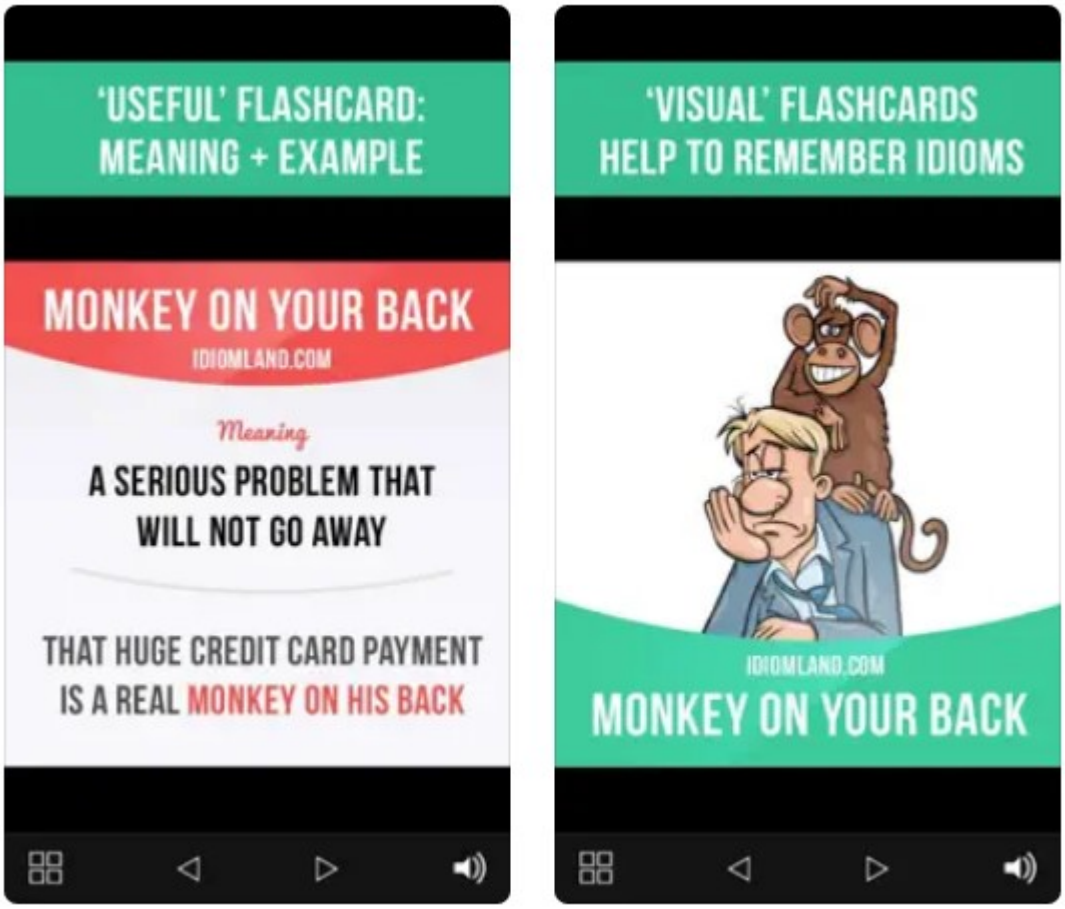


Figure 2.3 – Visual flashcards help to remember idioms

3. "English Idioms and Phrases"

This app is dedicated to one of the most difficult lexical topics in the English language - Idioms.

All English language learners face these unusual phrases from the very beginning of their curriculum. When you meet them for the first time you get confused and usually try to translate each word and only after that you try to understand the meaning of the whole phrase. But very quickly you realize that you can not understand it at all. Such extraordinary phrases are called idioms. To understand what they mean we have to learn them [4].

The program interface is as follows (Figure 2.4):

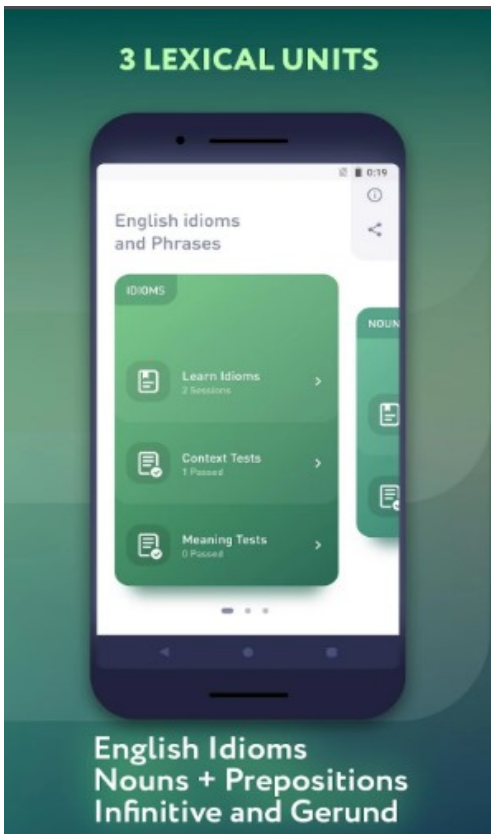


Figure 2.4 – Lexical units

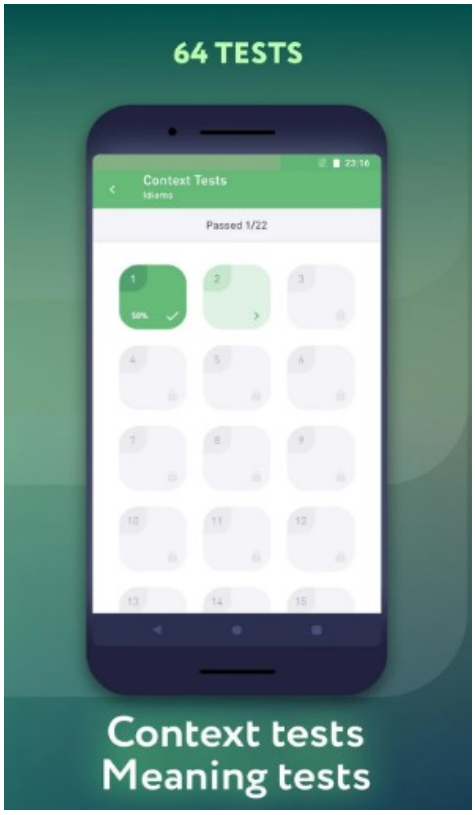


Figure 2.5 – Tests

So, the above-mentioned programs have a convenient interface and good functionality, but they lack privacy, lightweight, to work offline, and quiz. 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

	All English Idioms & Phrases	English Idioms Cards	English Idioms and Phrases	English Idioms Learning
Private using	+	-	+	+
Light weight	+	+	-	+
Multi operating system	-	+	+	+
To work offline	+	+	-	+
Quiz	-	-	+	+

2.2 Selection of software development technologies and tools

To create software for learning English idioms, 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

The Visual Studio IDE is a creative launching pad that you can use to edit, debug, and build code, and then publish an app. Over and above the standard editor and debugger that most IDEs provide, Visual Studio includes compilers, code completion tools, graphical designers, and many more features to enhance the software development process. [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 2.6):

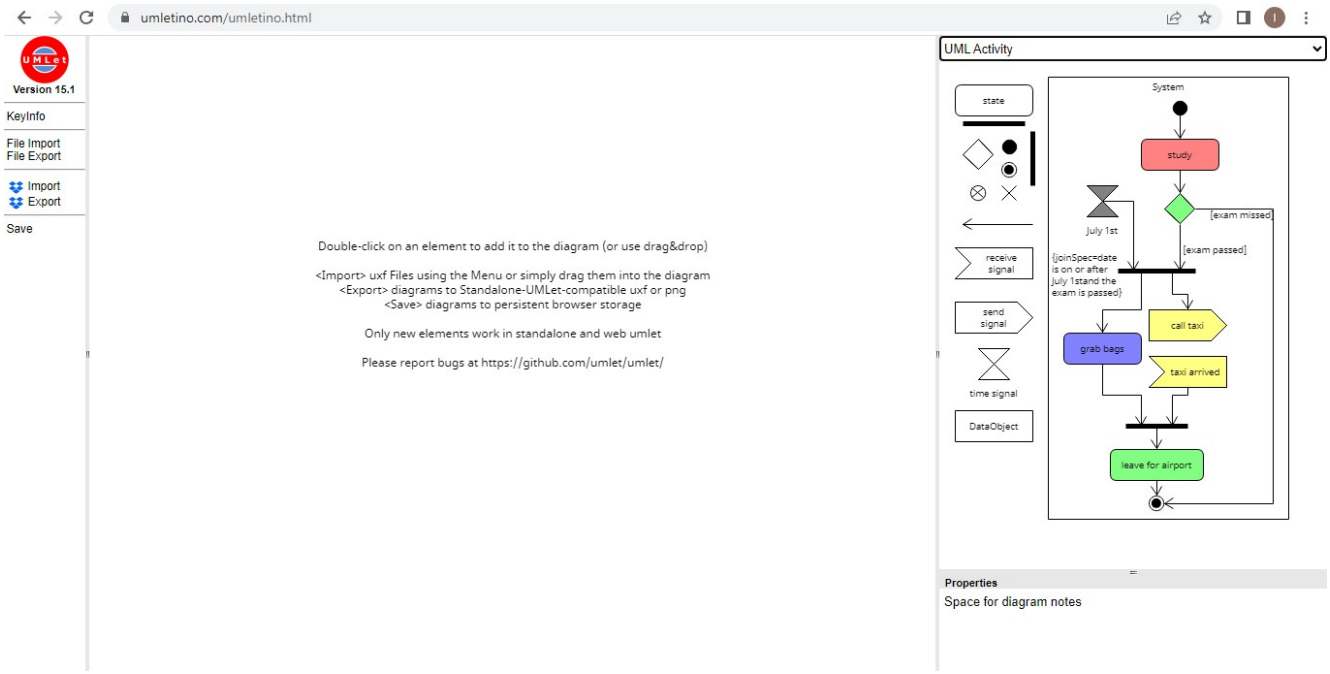


Figure 2.6 – UMLetino

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. 2.7) 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,
    "Phrase":"in a brown study",
    "Learned":false
  },
  {
    "Id":2,
    "Phrase":"study animal",
    "Learned":false
  },
  {
    "Id":3,
    "Phrase":"work like a horse",
    "Learned":false
  }
]
```

Figure 2.7 – 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: 02.05.2022

Version: Final version.

Date: 02.05.2022

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

Introduction:

This software is a reliable and private application.

Positioning:

Economic prerequisites:

- Existing software products are free, have a convenient interface and good functionality, but do not provide privacy. It is impossible to use them without the Internet. They are not light and easy to maintain.

- To ensure the operation of the application, it is not planned to spend money on the Internet, as well as spend money on support by programmers.

- 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 provides learning of English idioms, but which ensures privacy, does not involve the use of the Internet and database management systems.

System place:

- The application is intended for private use. It does not interact with other applications.

- The application is intended for a person who likes to learn idioms. It has idioms, their meanings and their examples. There are also incorrect meanings for forming options during the learning process.

Interested persons:

- User - for learning idioms from the English language.

Basic high-level tasks:

- Learning idioms.

User level tasks:

- Management of idioms, their meanings, their examples and their incorrect meanings used in the process of studying idioms.
- Show learned idioms.
- Display of unlearned idioms.

Review:

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

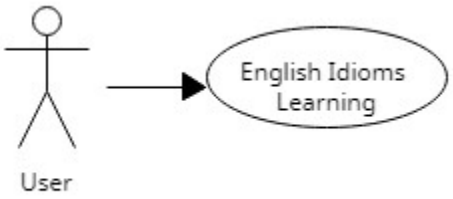


Figure 3.1 – Context diagram

A context diagram shows the place and role of an application in the overall system of interaction between applications and users.

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:

- The cost of the application will be determined by the consumer market.

Licensing and installation:

- It is planned to use the application after licensing (certification).

Main properties of the system:

- Learning idioms.
- Management of idioms and their meanings.

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):

Idiom

Id	Phrase	Done
1	Idiom 1	TRUE
2	Idiom 2	FALSE

Meaning

Id	Idiom meaning	Idiom Id
1	Meaning 1	1
2	Meaning 2	2

Wrong meaning

Id	Idiom wrong meaning	Meaning Id
1	Wrong meaning 1	1
2	Wrong meaning 2	1
3	Wrong meaning 3	1
4	Wrong meaning 4	2
5	Wrong meaning 5	2
6	Wrong meaning 6	2

Figure 3.2 – Book, Meaning, Wrong Meaning spreadsheets

A single idiom can have many meanings, and a single meaning can have many false meanings. Thus, it is a one-to-many relationship.

Based on the tables, we can construct the following diagram of the subject area (Figure 3.3):

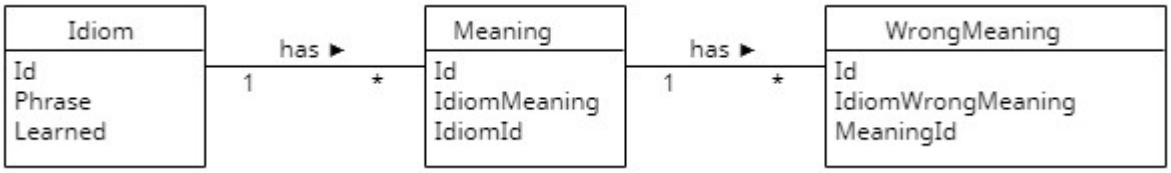


Figure 3.3 – Domain model diagram

Models are formatted using code (Figure 3.4 — 3.6):

```
namespace IdiomProject
{
    Ссылок: 10
    internal class Idiom : IId
    {
        Ссылок: 9
        public int Id { get;set; }
        Ссылок: 2
        public string Phrase { get; set; }
        Ссылок: 2
        public bool Learned { get; set; }
        Ссылок: 0
        public Idiom() { }
        Ссылок: 3
        public Idiom(string phrase, bool learned)
        {
            Phrase = phrase;
            Learned = learned;
        }
        Ссылок: 0
        public override string ToString()
        {
            return String.Format(Id + " " + Phrase + " " + Learned);
        }
    }
}
```

Figure 3.4 – Idiom model

```
namespace IdiomProject
{
    Ссылок: 10
    internal class Meaning : IId
    {
        Ссылок: 9
        public int Id { get; set; }
        Ссылок: 2
        public string IdiomMeaning { get; set; }
        Ссылок: 2
        public int IdiomId { get; set; }

        Ссылок: 0
        public Meaning() { }
        Ссылок: 3
        public Meaning(string meaningText, int idiomId)
        {
            IdiomMeaning = meaningText;
            IdiomId = idiomId;
        }
        Ссылок: 0
        public override string ToString()
        {
            return String.Format(Id + " " + IdiomMeaning + " " + IdiomId);
        }
    }
}
```

Figure 3.5 – Meaning model

```
namespace IdiomProject
{
    Ссылок: 2
    internal class WrongMeaning
    {
        Ссылок: 1
        public int Id { get; set; }
        Ссылок: 2
        public string WrongMeaningText { get; set; }
        Ссылок: 2
        public int MeaningId { get; set; }

        Ссылок: 0
        public WrongMeaning() { }

        Ссылок: 0
        public WrongMeaning(string wrongMeaningText, int meaningId)
        {
            WrongMeaningText = wrongMeaningText;
            MeaningId = meaningId;
        }

        Ссылок: 0
        public override string ToString()
        {
            return String.Format(Id + " " + WrongMeaningText + " " + MeaningId);
        }
    }
}
```

Figure 3.6 – WrongMeaning model

3.2 Model of precedents

A UML-diagram of precedents (Use-Case Diagram) models the actions of the application. This diagram shows the interaction between users and the application. The diagram does not show how the application works internally.

A use-case diagram is presented in Figure 3.7.

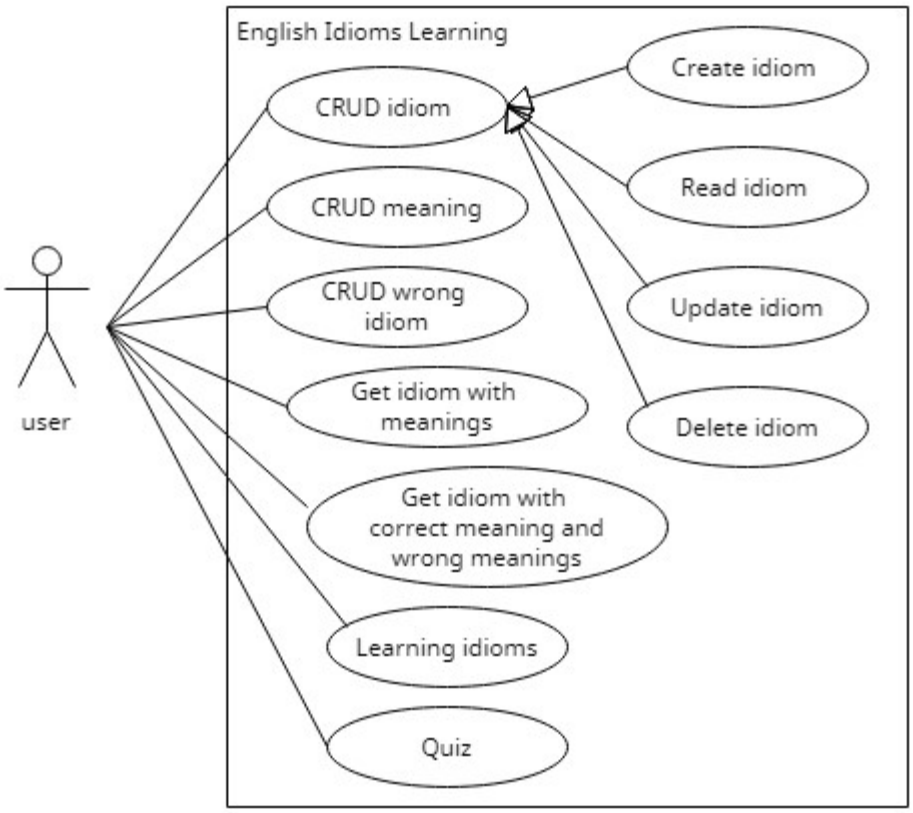


Figure 3.7 – User actions

The Idiom CRUD precedent is implemented at three levels: Presentation Layer, Business Logic Layer, and Data Access Layer. At the Business Logic Layer, Idiom CRUD is implemented in the IdiomBLL class (Figure 3.8).

```
public int CreateIdiom(string phrase, bool learned)
{
    return db.DBIdiom.AddItem(Factory.CreateIdiom(phrase, learned));
}
Ссылок: 2
public bool CheckIdiomAvailability(string phrase)...
Ссылок: 1
public Idiom GetIdiomById(int id)
{
    return db.DBIdiom.GetItemById(id);
}
Ссылок: 1
public bool UpdateIdiom(Idiom oldIdiom, string phrase, bool learned)
{
    return db.DBIdiom.Update(oldIdiom, Factory.CreateIdiom(phrase, learned));
}
Ссылок: 0
public bool DeleteIdiomByIdiom(Idiom idiom)
{
    return db.DBIdiom.DeleteItemByItem(idiom);
}
Ссылок: 2
public List<Idiom> GetAllIdioms()
{
    return db.DBIdiom.Items;
}
```

Figure 3.8 – CRUD methods

The activity diagram of the learning idioms precedent is shown on Figure 3.9.

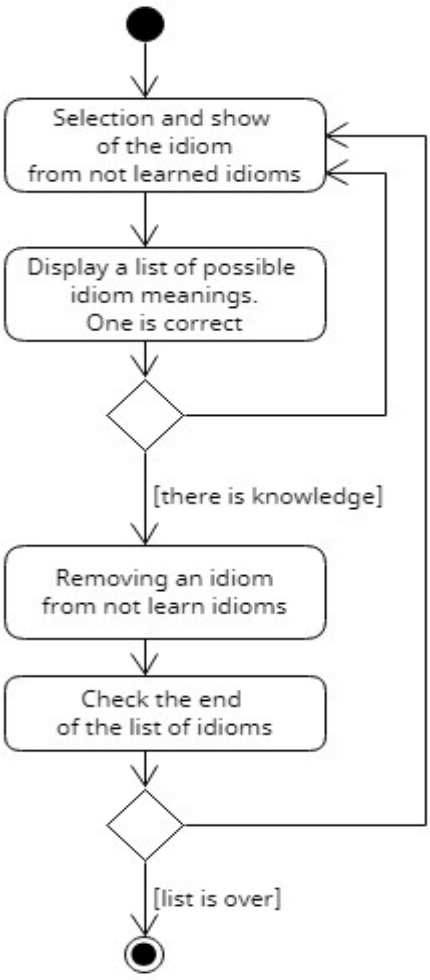


Figure 3.9 – Activity diagram

The learning idioms 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 IdiomServiceBLL class by Study() method (Figure 3.10). At the Presentation Layer, the precedent is implemented in the IdiomServicePL class by Study() method (Figure 3.11).

```
public List<IdiomLearn> Study(ref string idiomText) {
    List<IdiomLearn> idiomLearns = new List<IdiomLearn>();
    int randomNumber = 1 + new Random().Next(4);
    foreach (var i in db.DBIdiom.Items) {
        if (i.Learned == false) {
            int idCounter = 1;
            bool flag = false;
            idiomText = i.Phrase;
            foreach (var m in db.DBMeaning.Items) {
                if (m.IdiomId == i.Id) {
                    foreach (var wm in db.DBWrongMeaning.Items) {
                        if (wm.MeaningId == m.Id) {
                            if (idiomLearns.Count + 1 == randomNumber) {
                                flag = true;
                                idiomLearns.Add(new IdiomLearn(idCounter++, m.IdiomMeaning, true, i));
                                idiomLearns.Add(new IdiomLearn(idCounter++, wm.WrongMeaningText, false));
                            }
                            else {
                                idiomLearns.Add(new IdiomLearn(idCounter++, wm.WrongMeaningText, false));
                            }
                        }
                    }
                    if (flag == false) {
                        idiomLearns.Add(new IdiomLearn(idCounter++, m.IdiomMeaning, true, i));
                    }
                }
            }
            return idiomLearns;
        }
    }
    return idiomLearns;
}
```

Figure 3.10 – Learning idioms precedent at Business Logic Layer

```
public void Study()
{
    string idiomText = "";
    List<IdiomLearn> idiomLearns = idiomServiceBLL.Study(ref idiomText);
    Console.WriteLine(idiomText);
    foreach (var il in idiomLearns)
    {
        Console.WriteLine(il);
    }
    Console.Write("Enter number -> ");
    int number = int.Parse(Console.ReadLine());
    foreach (var il in idiomLearns)
    {
        if (il.Id == number)
        {
            if (il.RealIdiom == true)
            {
                il.Idiom.Learned = true;
                Console.WriteLine("Right");
            }
            else
            {
                Console.WriteLine("Wrong");
            }
        }
    }
}
```

Figure 3.11 – Learning idioms precedent at Presentation Layer

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. [10].

The diagram shown in Figure 3.12 describes the workflow for entering information about idioms and their meanings.

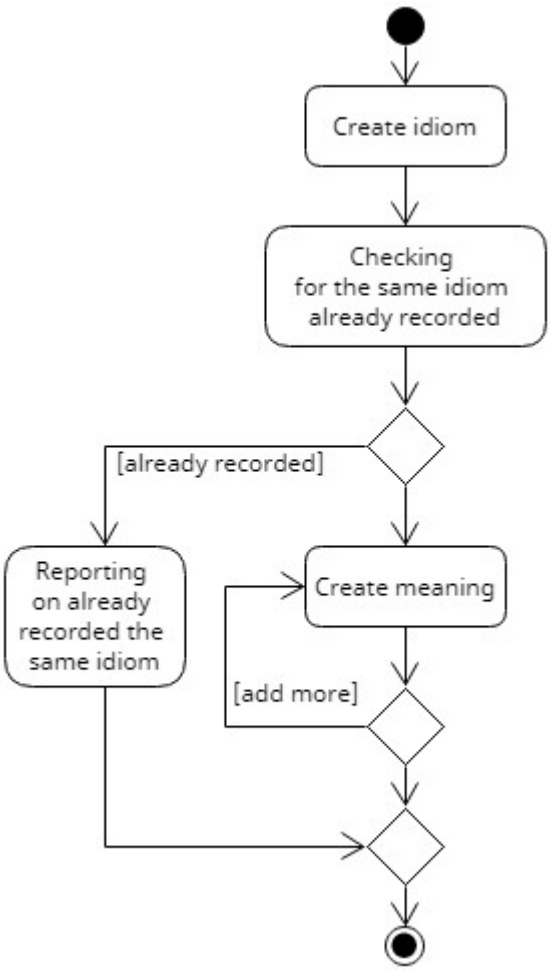


Figure 3.12 – Workflow for entering information about idioms and their meanings

At the Presentation Layer, the workflow is implemented in the IdiomPL class by CreateIdiom() method (Figure 3.13).

```
public void CreateIdiom()
{
    string phrase = Helper.StringInputCheck("Create Idiom phrase?: ", 150);
    bool isAvailable = idiomBll.CheckIdiomAvailability(phrase);
    if (!isAvailable)
    {
        bool learned = false;
        int idIdiom = idiomBll.CreateIdiom(phrase, learned);
        bool addMeaning = true;
        while (addMeaning)
        {
            int idMeaning = meaningPL.CreateMeaning(idIdiom);
            Console.Write("Add more meanings?(y/n): ");
            if ("y" != Console.ReadLine())
            {
                addMeaning = false;
            }
        }
    }
    else
    {
        Console.WriteLine("This idiom already exists in the idiom list!");
    }
}
```

Figure 3.13 – Create idiom method

3.3.2 Designing the sequence of method calls and object interaction

With the help of a sequence diagram, you can model the interaction of objects over time. It shows which objects interact and which messages are transmitted between them. They usually show methods that implement certain use-cases.

The following diagram (Figure 3.14) describes the interaction of objects implementing the user case for Create idiom. Objects of all three levels interact.

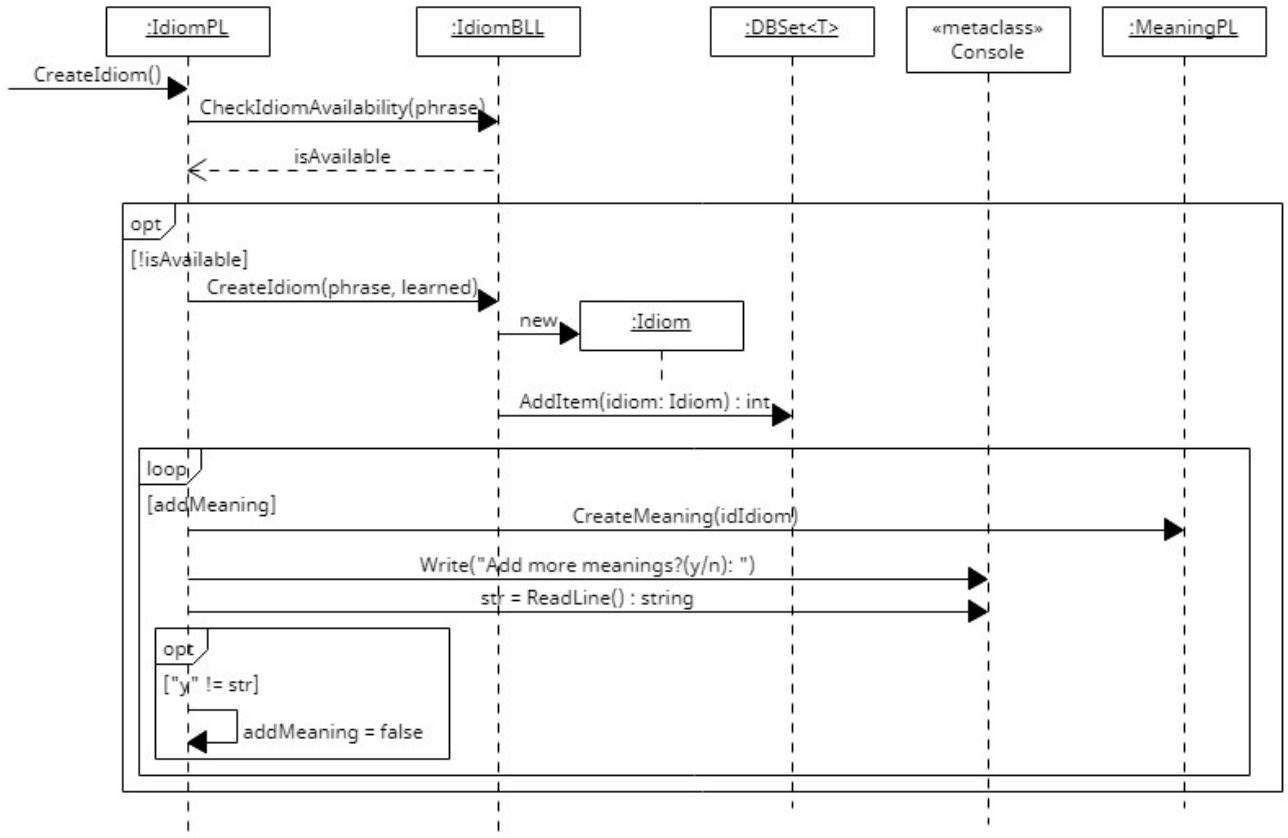


Figure 3.14 – CreateIdiom() method

The code is presented in Figure 3.13.

3.3.3 Designing classes and relationships between objects

The class diagram shows the static structure of the system. It shows classes, class attributes, class methods, and relationships between objects of these classes.

The class diagram related to the idiom on the Presentation Layer and Business Logic Layer is presented in Figure 3.15.

The IdiomPL class has the IIdiomPL interface and belongs to the Presentation Layer. It has a reference to an object of type IdiomBLL, which has the interface IIdiomBLL and belongs to the Business Logic Layer. At the Presentation Layer, the IdiomPL class has methods that interact with the user. Also, the methods of the IdiomPL class refer to the methods of the IdiomBLL class to implement the corresponding services at the Business Logic Layer.

On the Business Logic Layer, the IdiomBLL 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 IdiomPL class refers to the MeaningPL class to implement the service of pinning meanings for the idioms.

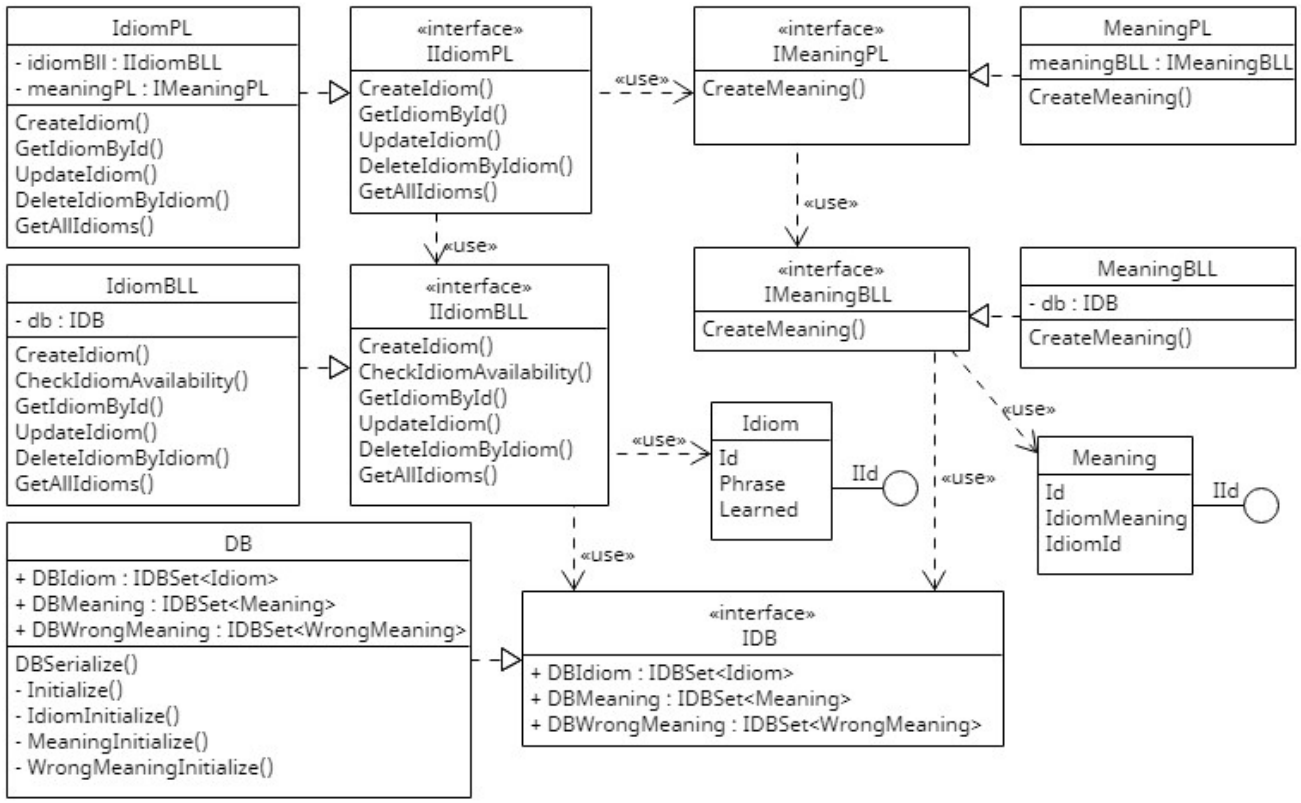


Figure 3.15 – Class diagram related to the idiom

3.3.4 Designing objects

Modeling of objects and their states is carried out using an object diagram. At certain moments of the application's operation, there is a certain number of objects. Objects have their states and their parameters have certain values.

Objects can refer to each other. Objects can be part of other objects.

An object diagram shows a slice of the system at a certain point in time. Object diagrams refer to structural diagrams.

The object diagram is used to show the data structure at a certain moment of the system's operation during the analysis of certain phases of the application's operation.

For example, when learning idioms, the application has four objects of meaning. One of them has the correct meaning. This object has a reference to the object idiom.

This situation can be depicted on the object diagram (Figure 3.16).

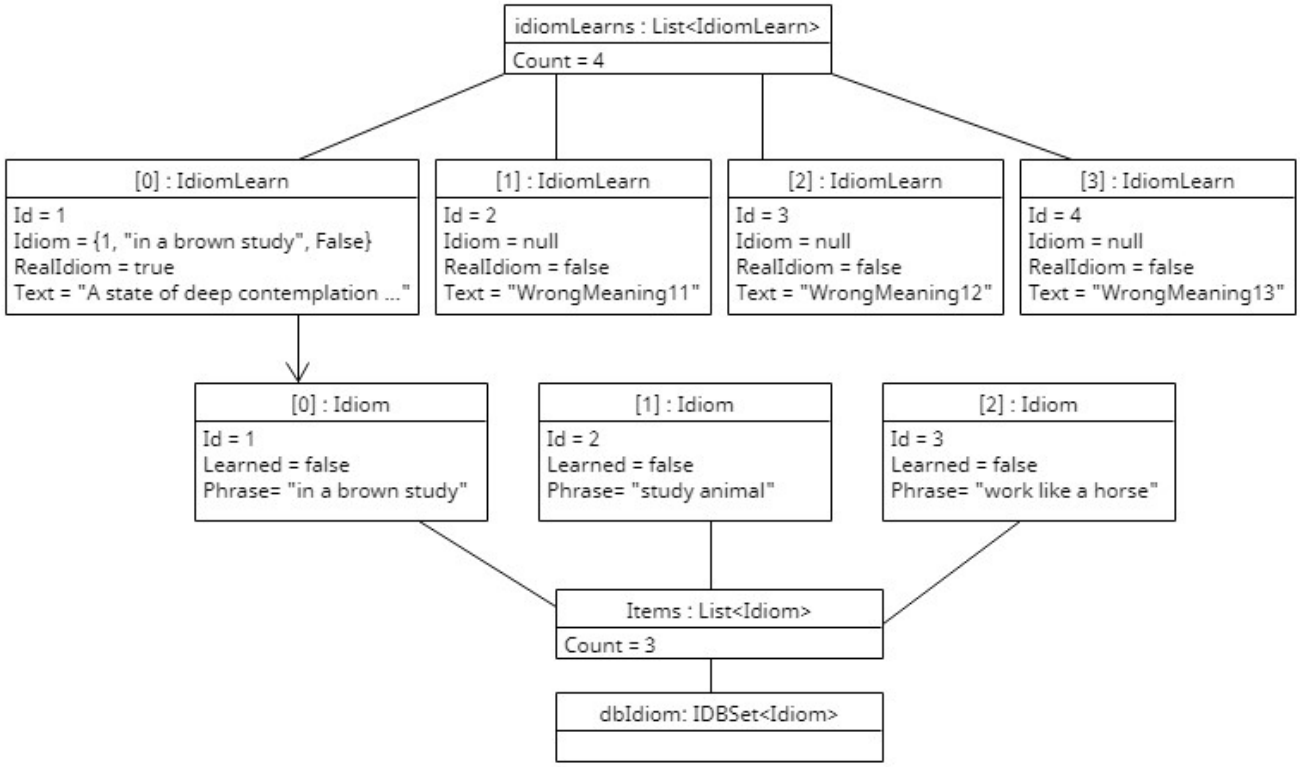


Figure 3.16 – Slice of the system during idiom learning

When the execution of the program is stopped, Visual Studio provides an opportunity to observe the states of objects.

Колекція idiomLearns представлена на Рисунку 3.17.

Figure 3.17 – Collection of objects for idiom learning

An object with the correct value is shown in Figure 3.18. The Idiom object property has a reference to the idiom whose value it has.

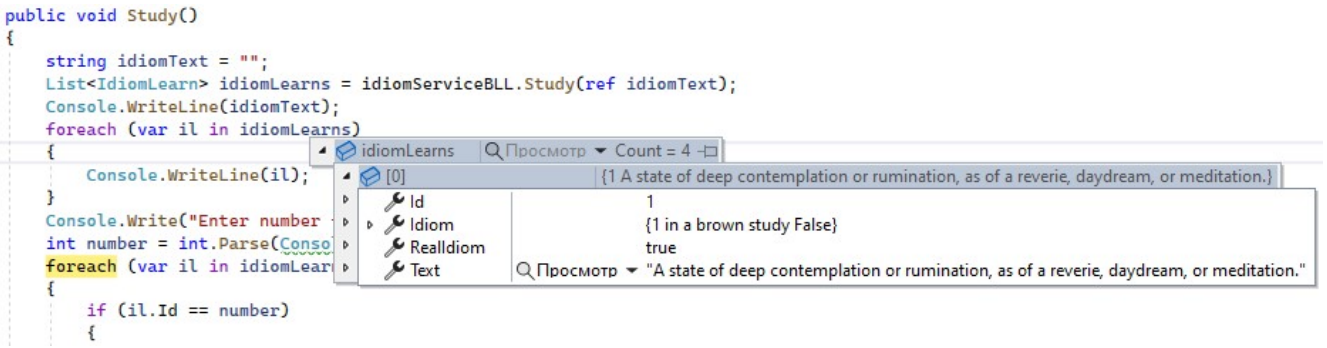


Figure 3.18 – The value of the object's properties

Колекція ідіом представлена на Рисунку 3.19.

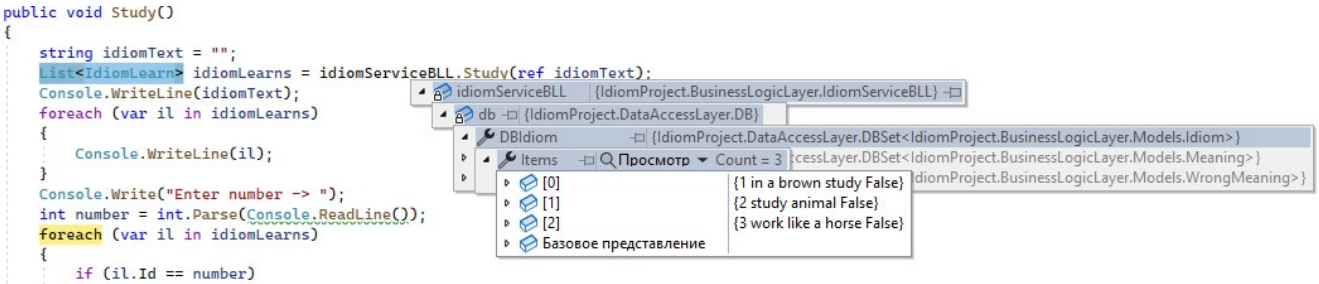


Figure 3.19 – Objects of idioms

An idiom object that is referenced by an object with the correct idiom meaning (Figure 3.20).

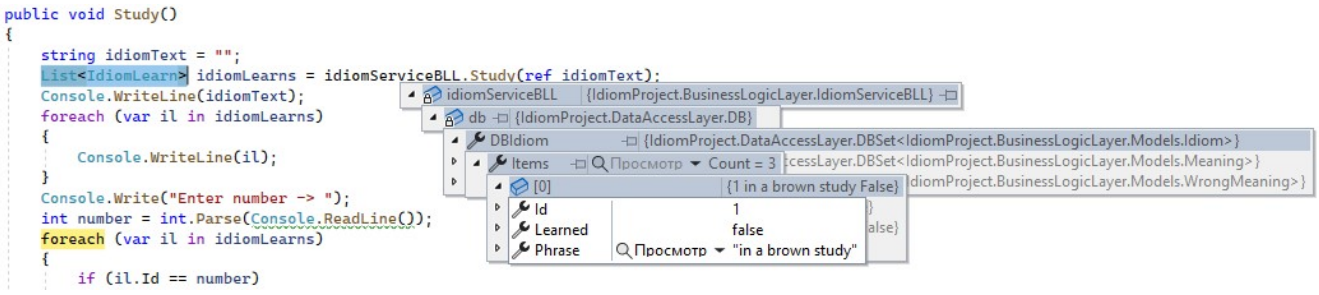


Figure 3.20 – The object of the idiom that is being studied at this moment

3.3.5 State design

A state diagram modulates object states and transitions between states. It shows states, transitions, events, and actions performed between states. This is a dynamic view of the system. A state diagram should be used when modeling a user interface (for example: a menu).

The following diagram of the menu states (Figure 3.21) shows the process of transition from the main menu to the Idiom learning menu and to the Idiom management menu. From each menu there is an exit to a higher level.

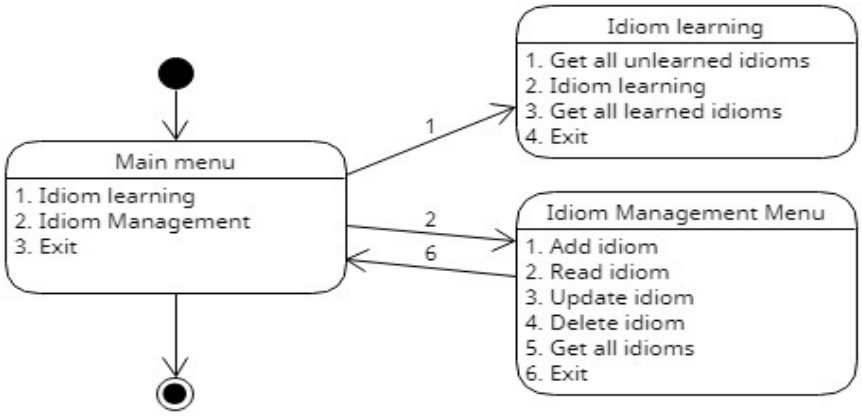


Figure 3.21 – Menu states

The main menu is implemented by the code presented in Figure 3.22.

```
internal class MainMenu
{
    IMenu idiomLearningMenu;
    IMenu idiomManagementMenu;
    Ссылок: 0
    public MainMenu(IMenu idiomLearningMenu, IMenu idiomManagementMenu)
    {
        this.idiomLearningMenu = idiomLearningMenu;
        this.idiomManagementMenu = idiomManagementMenu;
    }
    Ссылок: 0
    public void Run()
    {
        bool flag = true;
        while (flag)
        {
            Console.WriteLine("1 - Idiom Learning Menu");
            Console.WriteLine("2 - Idiom Management Menu");
            Console.WriteLine("3 - Exit");
            int menuNumber = Helper.IntInputCheck("-> ");
            switch (menuNumber)
            {
                case 1:
                    idiomLearningMenu.Run();
                    break;
                case 2:
                    idiomManagementMenu.Run();
                    break;
                case 3:
                    flag = false;
                    break;
                default:
                    Console.WriteLine("Error");
                    break;
            }
        }
    }
}
```

Figure 3.22 – Main menu code

3.3.6 System deployment design

The application is deployed as shown in Figure 3.23.

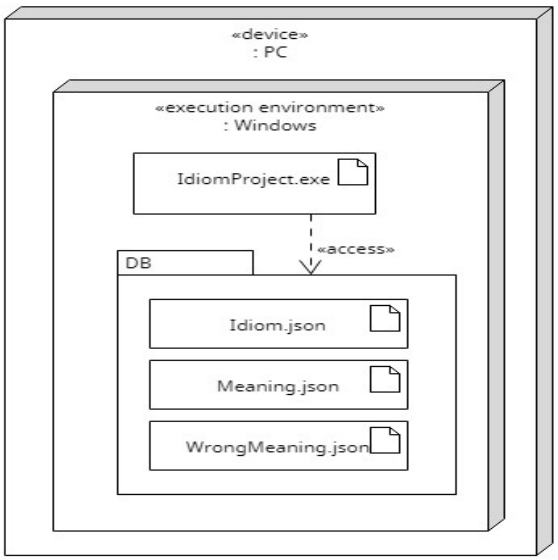


Figure 3.23 – Deployment diagram

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 3.24).

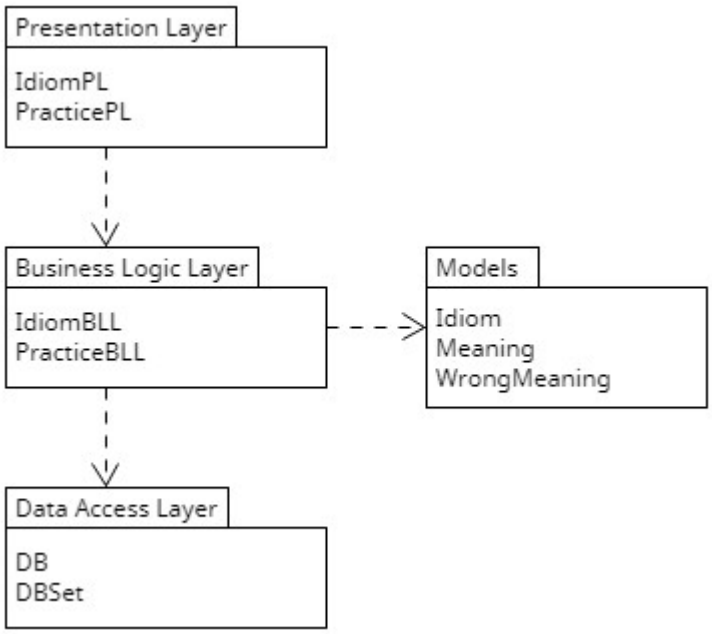


Figure 3.24 – Architecture

In the program, the packages look like this (Figure 3.25):

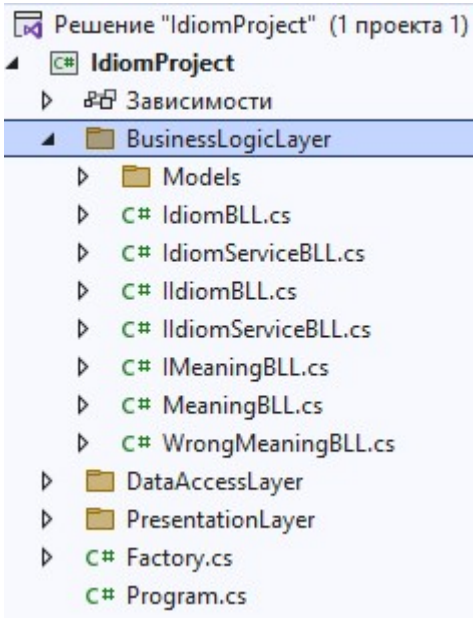


Figure 3.25 – View of packages in the project

4 TESTING

Manual testing was performed in this project.

The test results are presented below (Table 4.1):

Table 4.1 — Result of testing

	Testing
CRUD Idiom	+
Practice	+
CRUD Wrong Meaning	+
Get all idioms	+

CONCLUSIONS

1. It has been done analysis of existing similar applications.

2. The model of the subject field (domain diagram) has been modulated.

3. Modulated the case model (case set and use case diagram).

4. The design model of all use cases (diagrams of activities, sequences) has been modulated.

5. A three-leyer architecture (package diagram) has been formed.

6. The structure of the application (class diagram) has been elaborated.

7. The application has been created.

8. The application has been tested.

REFERENCES

1. Understanding the Importance of Idioms in English Language Learning [Electronic resource] – Resource access mode: https://www.ascenglish.com/blog/2019/05/understanding-the-importance-of-idioms-in-english-language-learning.

2. All English Idioms & Phrases [Electronic resource] – Resource access mode: https://play.google.com/store/apps/details?id=com.app.englishidioms&hl=en_US.

3. English Idioms Cards [Electronic resource] – Resource access mode: https://apps.apple.com/us/app/english-idioms-cards/id991895435.

4. English Idioms and Phrases [Electronic resource] – Resource access mode: https://play.google.com/store/apps/details?id=com.sevenlynx.idioms.

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.

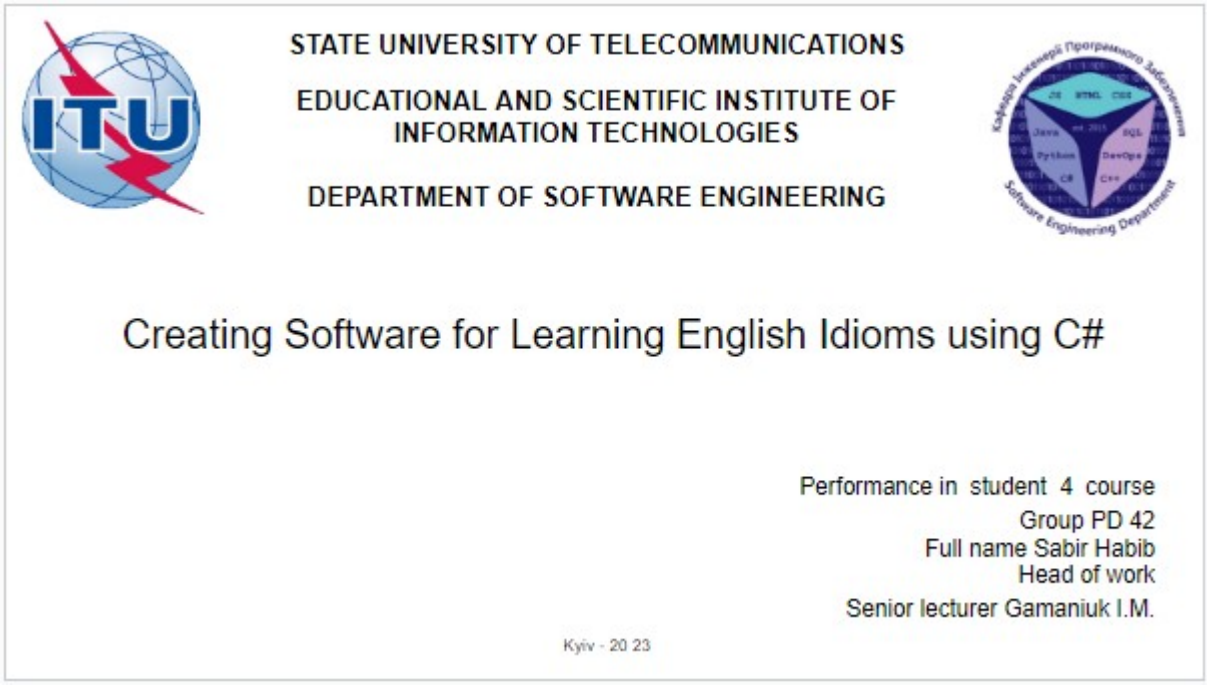
7. Visual Studio 2022 [Electronic resource] – Resource access mode: https://visualstudio.microsoft.com/#vs-section.

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. Activity diagram [Electronic resource] – Resource access mode: https://en.wikipedia.org/wiki/Activity_diagram.

ANNEX A



STATE UNIVERSITY OF TELECOMMUNICATIONS

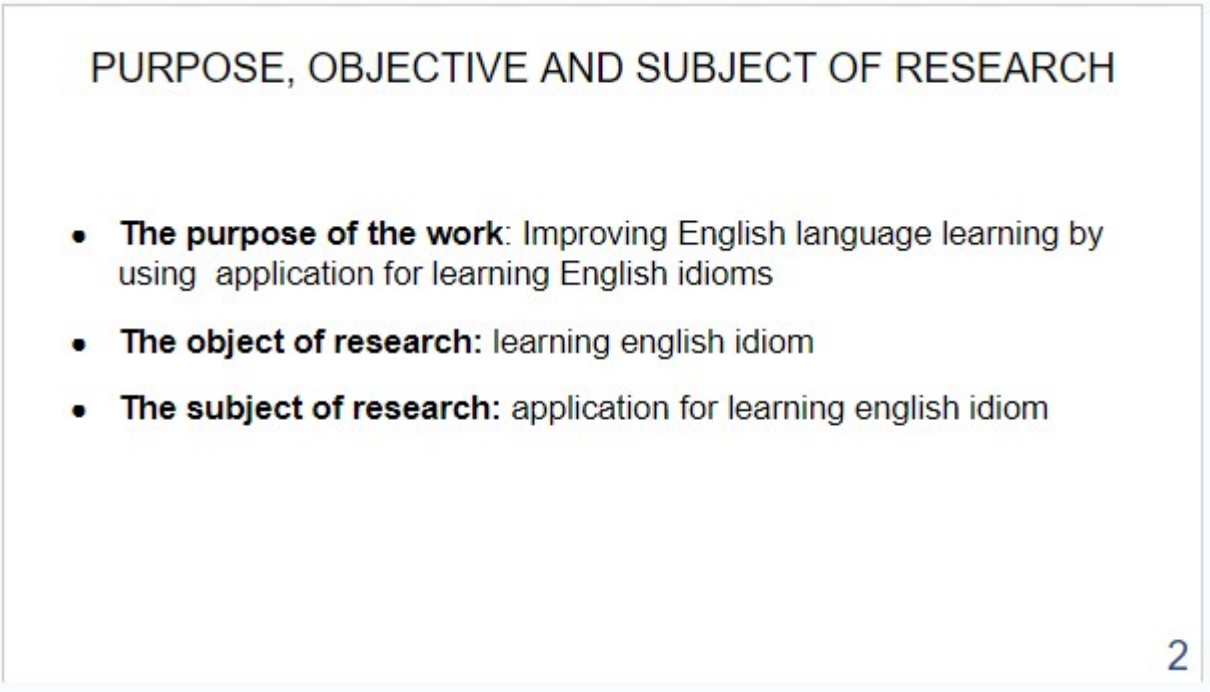
EDUCATIONAL AND SCIENTIFIC INSTITUTE OF INFORMATION TECHNOLOGIES

DEPARTMENT OF SOFTWARE ENGINEERING

Creating Software for Learning English Idioms using C#

Performance in student 4 course
Group PD 42
Full name Sabir Habib
Head of work
Senior lecturer Gamaniuk I.M.

Kyiv - 20 23



PURPOSE, OBJECTIVE AND SUBJECT OF RESEARCH

- **The purpose of the work**: Improving English language learning by using application for learning English idioms
- **The object of research:** learning english idiom
- **The subject of research:** application for learning english idiom

2

TASKS OF THE GRADUATE THESIS

1. Analysis of existing similar applications
2. Determination of application requirements
3. Using tools to create application
4. Application modeling and design
5. Develop application
6. Testing

ANALYSIS OF ANALOGUES

	All English Idioms & Phrases	English Idioms Cards	English Idioms and Phrases	English Idioms Learning
Private using	+	-	+	+
Light weight	+	+	-	+
Multi operating system	-	+	+	+
To work offline	+	+	-	+
Quiz	-	-	+	+

4

APPLICATION REQUIREMENTS

Functional requirements:

- Create, Read, Update, Delete idioms
- Practice of learning idioms

Non-functional requirements:

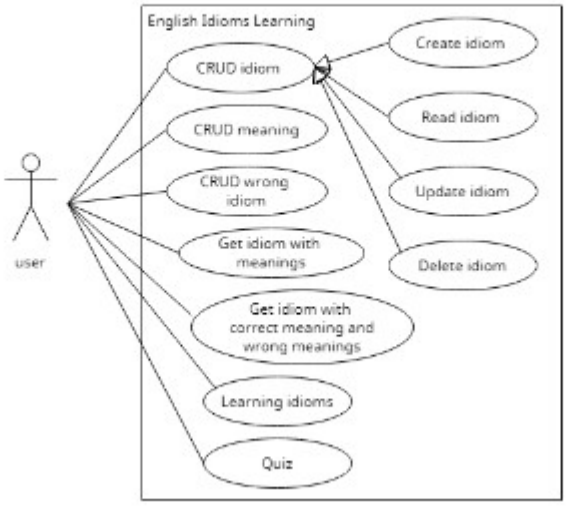
- English languge
- Private using
- Light weight
- Multi operating system

5

IMPLEMENTATION SOFTWARE

6

Functional requirements



7

Domain

Idiom

Id	Phrase	Done
1	Idiom 1	TRUE
2	Idiom 2	FALSE

Meaning

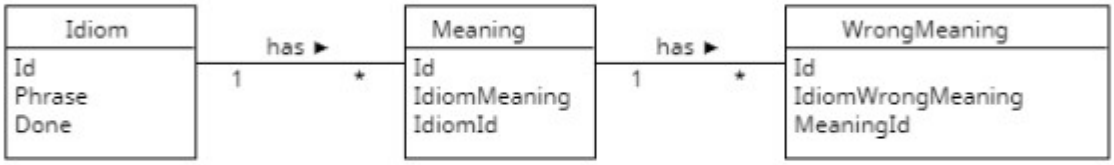
Id	Idiom meaning	Idiom Id
1	Meaning 1	1
2	Meaning 2	2

Wrong meaning

Id	Idiom wrong meaning	Meaning Id
1	Wrong meaning 1	1
2	Wrong meaning 2	1
3	Wrong meaning 3	1
4	Wrong meaning 4	2
5	Wrong meaning 5	2
6	Wrong meaning 6	2

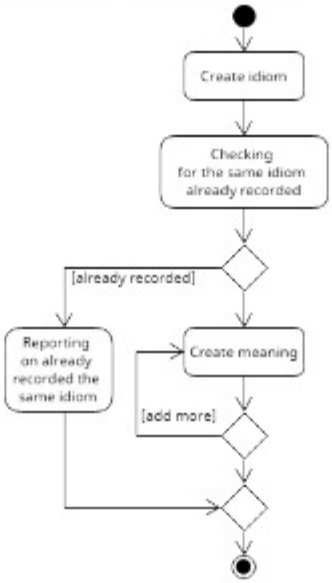
8

Domain Diagram



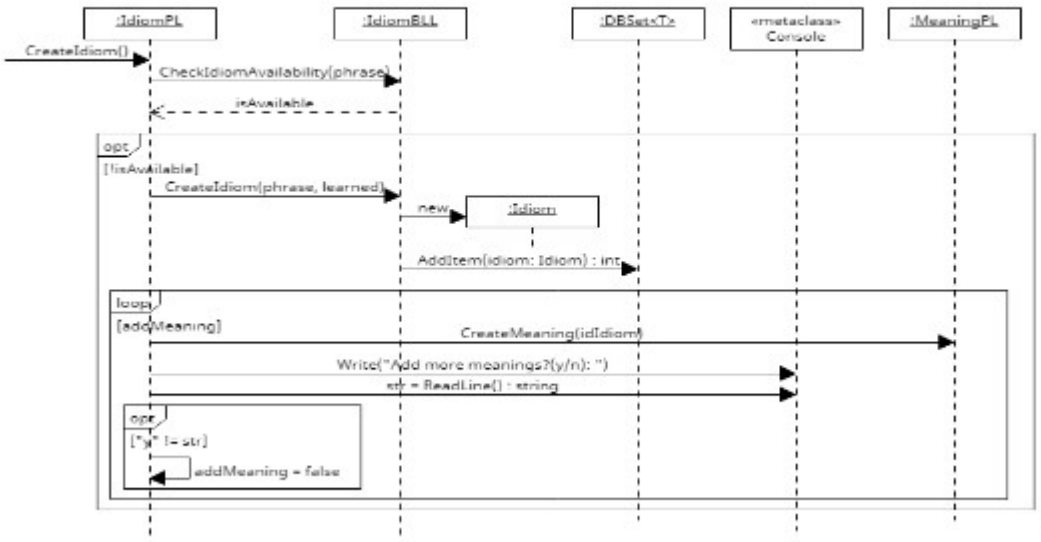
9

Active Diagram



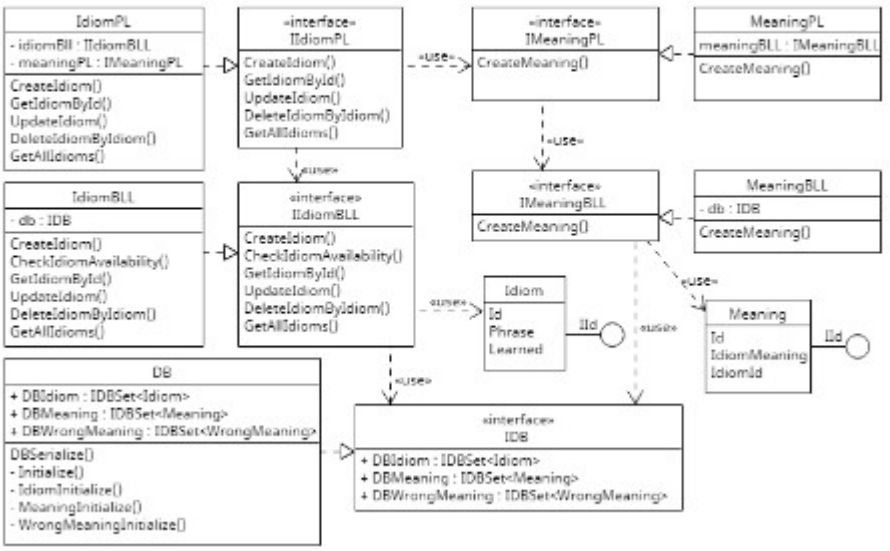
10

Sequence Diagram

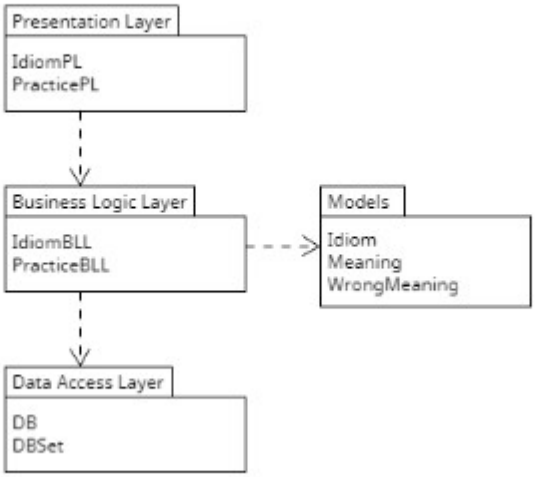


11

Class Diagram

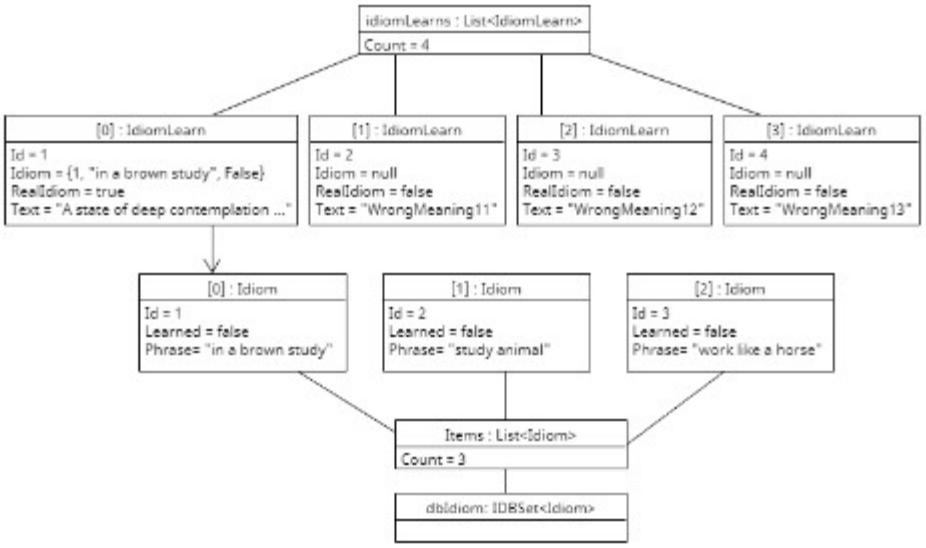


Architecture

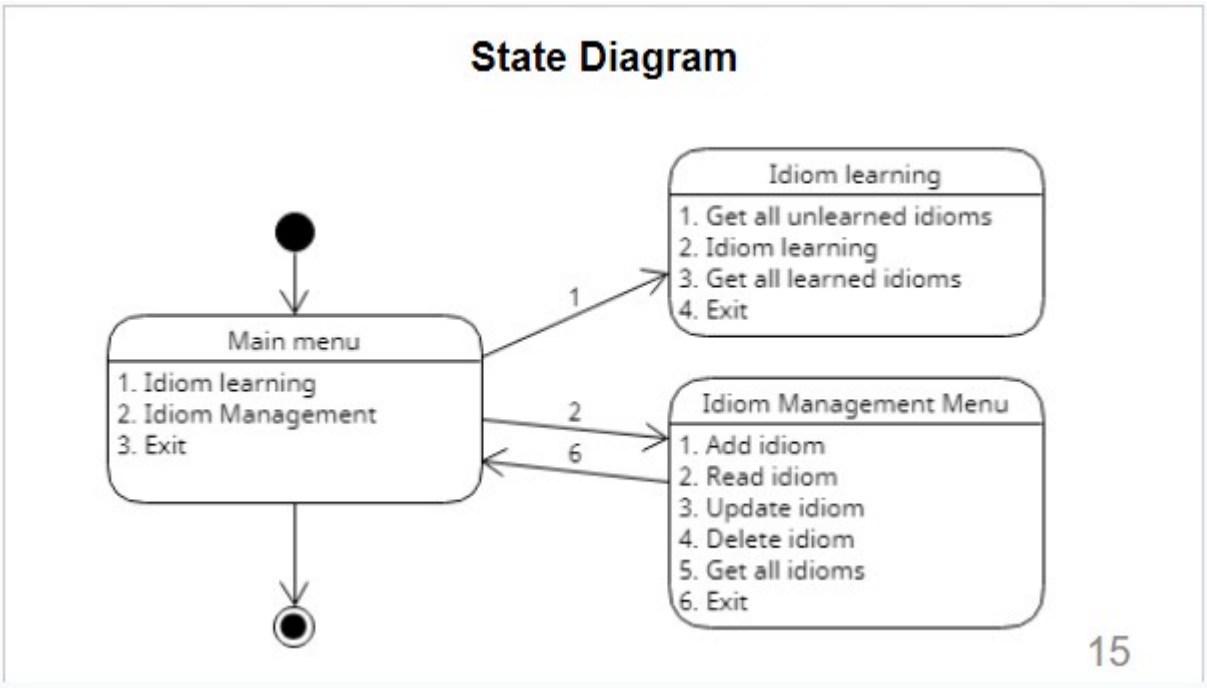


13

Object Diagram



14


State Diagram
Idiom learning
1. Get all unlearned idioms
2. Idiom learning
3. Get all learned idioms
4. Exit
1
Main menu
1. Idiom learning
2. Idiom Management
3. Exit
2
6
Idiom Management Menu
1. Add idiom
2. Read idiom
3. Update idiom
4. Delete idiom
5. Get all idioms
6. Exit
15

Testing

	Testing
CRUD Idiom	+
Practice	+
CRUD Wrong Meaning	+
Get all idioms	+

CONCLUSION

1. Done analysis of existing similar applications
2. Done application modeling and design
3. Done develop application
4. Done testing

THANK YOU FOR ATTENTION!

ANNEX B

```
namespace IdiomProject.BusinessLogicLayer.Models
{
    internal class Idiom : IId
    {
        public int Id { get; set; }
        public string Phrase { get; set; }
        public bool Learned { get; set; }
        public Idiom() { }
        public Idiom(string phrase, bool learned)
        {
            Phrase = phrase;
            Learned = learned;
        }
        public override string ToString()
        {
            return string.Format(Id + " " + Phrase + " " + Learned);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.BusinessLogicLayer.Models
{
    internal class IdiomLearn
    {
        public int Id { get; set; }
        public string Text { get; set; }
        public bool RealIdiom { get; set; } = false;
        public Idiom Idiom { get; set; }

        public IdiomLearn(int id, string text, bool realIdiom)
        {
            Id = id;
            Text = text;
            RealIdiom = realIdiom;
        }
        public IdiomLearn(int id, string text, bool realIdiom, Idiom idiom)
        {
            Id = id;
            Text = text;
            RealIdiom = realIdiom;
            Idiom = idiom;
        }
        public override string ToString()
        {
            return string.Format(Id + " " + Text);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.BusinessLogicLayer.Models
{
    internal interface IId
    {
```

```
        int Id { get; set; }
    }
}
namespace IdiomProject.BusinessLogicLayer.Models
{
    internal class Meaning : IId
    {
        public int Id { get; set; }
        public string IdiomMeaning { get; set; }
        public int IdiomId { get; set; }

        public Meaning() { }
        public Meaning(string meaningText, int idiomId)
        {
            IdiomMeaning = meaningText;
            IdiomId = idiomId;
        }
        public override string ToString()
        {
            return string.Format(Id + " " + IdiomMeaning + " " + IdiomId);
        }
    }
}
namespace IdiomProject.BusinessLogicLayer.Models
{
    internal class WrongMeaning : IId
    {
        public int Id { get; set; }
        public string WrongMeaningText { get; set; }
        public int MeaningId { get; set; }

        public WrongMeaning() { }

        public WrongMeaning(string wrongMeaningText, int meaningId)
        {
            WrongMeaningText = wrongMeaningText;
            MeaningId = meaningId;
        }

        public override string ToString()
        {
            return string.Format(Id + " " + WrongMeaningText + " " + MeaningId);
        }
    }
}
using IdiomProject.BusinessLogicLayer.Models;
using IdiomProject.DataAccessLayer;
using IdiomProject.PresentationLayer;
namespace IdiomProject.BusinessLogicLayer
{
    internal class IdiomBLL : IIdiomBLL
    {
        IDB db;

        public IdiomBLL(IDB db)
        {
            this.db = db;
        }
        public int CreateIdiom(string phrase, bool learned)
        {
            return db.DBIdiom.AddItem(Factory.CreateIdiom(phrase, learned));
        }
        public bool CheckIdiomAvailability(string phrase)
        {
            bool isAvailable = false;
```

```
            foreach (Idiom i in db.DBIdiom.Items)
            {
                if (i.Phrase == phrase)
                {
                    isAvailable = true;
                    break;
                }
            }
            return isAvailable;
        }
        public Idiom GetIdiomById(int id)
        {
            return db.DBIdiom.GetItemById(id);
        }
        public bool UpdateIdiom(Idiom oldIdiom, string phrase, bool learned)
        {
            return db.DBIdiom.Update(oldIdiom, Factory.CreateIdiom(phrase, learned));
        }
        public bool DeleteIdiomByIdiom(Idiom idiom)
        {
            return db.DBIdiom.DeleteItemByItem(idiom);
        }
        public List<Idiom> GetAllIdioms()
        {
            return db.DBIdiom.Items;
        }
    }
}
using IdiomProject.BusinessLogicLayer.Models;
using IdiomProject.DataAccessLayer;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.BusinessLogicLayer
{
    internal class IdiomServiceBLL : IIdiomServiceBLL
    {
        IDB db;

        public IdiomServiceBLL(IDB db)
        {
            this.db = db;
        }
        public List<IdiomLearn> Study(ref string idiomText) {
            List<IdiomLearn> idiomLearns = new List<IdiomLearn>();
            int randomNumber = 1 + new Random().Next(4);
            foreach (var i in db.DBIdiom.Items) {
                if (i.Learned == false) {
                    int idCounter = 1;
                    bool flag = false;
                    idiomText = i.Phrase;
                    foreach (var m in db.DBMeaning.Items) {
                        if (m.IdiomId == i.Id) {
                            foreach (var wm in db.DBWrongMeaning.Items) {
                                if (wm.MeaningId == m.Id) {
                                    if (idiomLearns.Count + 1 == randomNumber) {
                                        flag = true;
                                        idiomLearns.Add(new IdiomLearn(idCounter++,
m.IdiomMeaning, true, i));
                                        idiomLearns.Add(new IdiomLearn(idCounter++,
wm.WrongMeaningText, false));
                                    }
```

```
                                    else {
                                        idiomLearns.Add(new IdiomLearn(idCounter++,
wm.WrongMeaningText, false));
                                    }
                                }
                            }
                            if (flag == false) {
                                idiomLearns.Add(new IdiomLearn(idCounter++,
m.IdiomMeaning, true, i));
                            }
                        }
                    }
                    return idiomLearns;
                }
            }
            return idiomLearns;
        }

    }
}
using IdiomProject.BusinessLogicLayer.Models;

namespace IdiomProject.BusinessLogicLayer
{
    internal interface IIdiomBLL
    {
        int CreateIdiom(string phrase, bool learned);
        bool CheckIdiomAvailability(string phrase);
        Idiom GetIdiomById(int id);
        bool UpdateIdiom(Idiom oldIdiom, string phrase, bool learned);
        List<Idiom> GetAllIdioms();
    }
}
using IdiomProject.BusinessLogicLayer.Models;

namespace IdiomProject.BusinessLogicLayer
{
    internal interface IIdiomServiceBLL
    {
        List<IdiomLearn> Study(ref string idiomText);
    }
}
namespace IdiomProject.BusinessLogicLayer
{
    internal interface IMeaningBLL
    {
        int CreateMeaning(string meaningText, int idiomId);
    }
}
using IdiomProject.DataAccessLayer;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.BusinessLogicLayer
{
    internal class MeaningBLL : IMeaningBLL
    {
        IDB db;

        public MeaningBLL(IDB db)
        {
            this.db = db;
```

```
        }
        public int CreateMeaning(string meaningText, int idiomId)
        {
            return db.DBMeaning.AddItem(Factory.CreateMeaning(meaningText, idiomId));
        }
    }
}
using IdiomProject.DataAccessLayer;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.BusinessLogicLayer
{
    internal class WrongMeaningBLL
    {
        IDB db;

        public WrongMeaningBLL(IDB db)
        {
            this.db = db;
        }
        public int CreateWrongMeaning(string wrongMeaningText, int meaningId)
        {
            return db.DBWrongMeaning.AddItem(Factory.CreateWrongMeaning(wrongMeaningText,
meaningId));
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.DataAccessLayer
{
    internal class Configuration
    {
        public static string DBPath { get; set; } = @"C:\Temp2\IdiomDB";
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using IdiomProject.BusinessLogicLayer.Models;

namespace IdiomProject.DataAccessLayer
{
    internal class DB : IDB
    {
        public IDBSet<Idiom> DBIdiom { get; set; } = new DBSet<Idiom>();
        public IDBSet<Meaning> DBMeaning { get; set; } = new DBSet<Meaning>();
        public IDBSet<WrongMeaning> DBWrongMeaning { get; set; } = new
DBSet<WrongMeaning>();
        public DB()
        {
            Initialize();
        }
        public void DBSerialize()
        {
```

```
            try
            {
                DBIdiom.SerializeJSON();
                DBMeaning.SerializeJSON();
                DBWrongMeaning.SerializeJSON();
            }
            catch(Exception ex)
            {
                Console.WriteLine(ex.Message);
            }

        }
        private void Initialize()
        {
            try
            {
                DBIdiom.DeserializeJSON();
            }
            catch (Exception ex)
            {
                //Console.WriteLine(ex.Message);
                IdiomInitialize();
            }
            try
            {
                DBMeaning.DeserializeJSON();
            }
            catch (Exception ex)
            {
                //Console.WriteLine(ex.Message);
                MeaningInitialize();
            }
            try
            {
                DBWrongMeaning.DeserializeJSON();
            }
            catch (Exception ex)
            {
                //Console.WriteLine(ex.Message);
                WrongMeaningInitialize();
            }
        }

        //private void Initialize()
        //{
        //    IdiomInitialize();
        //    MeaningInitialize();
        //    WrongMeaningInitialize();
        //}
        private void IdiomInitialize()
        {
            DBIdiom.AddItem(Factory.CreateIdiom("in a brown study", false));
            DBIdiom.AddItem(Factory.CreateIdiom("study animal", false));
            DBIdiom.AddItem(Factory.CreateIdiom("work like a horse", false));
        }
        private void MeaningInitialize()
        {
            DBMeaning.AddItem(Factory.CreateMeaning("A state of deep contemplation or
rumination, as of a reverie, daydream, or meditation.", 1));
            DBMeaning.AddItem(Factory.CreateMeaning("Someone who studies very hard or
very often.", 2));
            DBMeaning.AddItem(Factory.CreateMeaning("To work with great intensity, energy,
and persistence.", 3));
        }
        private void WrongMeaningInitialize()
```

```
        {
            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning11", 1));
            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning12", 1));
            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning13", 1));

            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning21", 2));
            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning22", 2));
            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning23", 2));

            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning31", 3));
            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning32", 3));
            DBWrongMeaning.AddItem(Factory.CreateWrongMeaning("WrongMeaning33", 3));
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.Json;
using System.Text.Json.Serialization;
using System.Threading.Tasks;
using IdiomProject.BusinessLogicLayer.Models;

namespace IdiomProject.DataAccessLayer
{
    internal class DBSet<T> : IDBSet<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;
            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\IdiomDB";
            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
            {
                var options = new JsonSerializerOptions
                {
                    IncludeFields = true,
                };

                jsonString = JsonSerializer.Serialize(Items, options);

                File.WriteAllText(fileName, jsonString);
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message + "\n" + "Verify that entity classes and
their base classes have a default constructor");
            }
            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"));
                var options = new JsonSerializerOptions
                {
                    IncludeFields = true,
                };

                Items = JsonSerializer.Deserialize<List<T>>(jsonString, options);
                counter = GetMaxId();
                return Items;
            }
            catch (Exception ex)
            {
                throw new Exception(ex.Message + "\n" + "Verify that entity classes and
their base classes have a default constructor");
            }
        }
        private int GetMaxId()
        {
            int maxId = 1;
            foreach (T item in Items)
            {
                if (item.Id > maxId)
```

```
                {
                    maxId = item.Id;
                }
            }
            return maxId + 1;
        }
    }
}
using IdiomProject.BusinessLogicLayer.Models;

namespace IdiomProject.DataAccessLayer
{
    internal interface IDB
    {
        IDBSet<Idiom> DBIdiom { get; set; }
        IDBSet<Meaning> DBMeaning { get; set; }
        IDBSet<WrongMeaning> DBWrongMeaning { get; set; }
    }
}
using IdiomProject.BusinessLogicLayer.Models;

namespace IdiomProject.DataAccessLayer
{
    internal interface IDBSet<T> where T : IId
    {
        List<T>? Items { get; set; }

        int AddItem(T item);
        bool DeleteItemByItem(T item);
        List<T> DeserializeJSON();
        T GetItemById(int id);
        bool SerializeJSON();
        bool Update(T oldItem, T newItem);
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.PresentationLayer.Menu
{
    internal interface IMenu
    {
        void Run();
    }
}
using IdiomProject.PresentationLayer;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.PresentationLayer.Menu
{
    internal class MainMenu
    {
        IMenu idiomLearningMenu;
        IMenu idiomManagementMenu;
        public MainMenu(IMenu idiomLearningMenu, IMenu idiomManagementMenu)
        {
            this.idiomLearningMenu = idiomLearningMenu;
            this.idiomManagementMenu = idiomManagementMenu;
```

```
        }
        public void Run()
        {
            bool flag = true;
            while (flag)
            {
                Console.WriteLine("1 - Idiom Learning Menu");
                Console.WriteLine("2 - Idiom Management Menu");
                Console.WriteLine("3 - Exit");
                int menuNumber = Helper.IntInputCheck("-> ");
                switch (menuNumber)
                {
                    case 1:
                        idiomLearningMenu.Run();
                        break;
                    case 2:
                        idiomManagementMenu.Run();
                        break;
                    case 3:
                        flag = false;
                        break;
                    default:
                        Console.WriteLine("Error");
                        break;
                }
            }
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject.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 System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using IdiomProject.BusinessLogicLayer;
using IdiomProject.BusinessLogicLayer.Models;

namespace IdiomProject.PresentationLayer
{
    internal class IdiomPL : IIdiomPL
    {
        IIdiomBLL idiomBll;
        IMeaningPL meaningPL;

        public IdiomPL(IIdiomBLL idiomBll, IMeaningPL meaningPL)
        {
            this.idiomBll = idiomBll;
            this.meaningPL = meaningPL;
        }
        public void CreateIdiom()
        {
            string phrase = Helper.StringInputCheck("Create Idiom phrase?: ", 150);
            bool isAvailable = idiomBll.CheckIdiomAvailability(phrase);
            if (!isAvailable)
```

```
            {
                bool learned = false;
                int idIdiom = idiomBll.CreateIdiom(phrase, learned);
                bool addMeaning = true;
                while (addMeaning)
                {
                    int idMeaning = meaningPL.CreateMeaning(idIdiom);
                    Console.Write("Add more meanings?(y/n): ");
                    if ("y" != Console.ReadLine())
                    {
                        addMeaning = false;
                    }
                }
            }
            else
            {
                Console.WriteLine("This idiom already exists in the idiom list!");
            }
        }
        public void GetAllIdioms()
        {
            List<Idiom> idioms = idiomBll.GetAllIdioms();
            foreach (Idiom i in idioms)
            {
                Console.WriteLine(i);
            }
        }

    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using IdiomProject.BusinessLogicLayer;
using IdiomProject.BusinessLogicLayer.Models;

namespace IdiomProject.PresentationLayer
{
    internal class IdiomServicePL : IIdiomServicePL
    {
        IIdiomServiceBLL idiomServiceBLL;

        public IdiomServicePL(IIdiomServiceBLL idiomServiceBLL)
        {
            this.idiomServiceBLL = idiomServiceBLL;
        }
        public void Study()
        {
            string idiomText = "";
            List<IdiomLearn> idiomLearns = idiomServiceBLL.Study(ref idiomText);
            Console.WriteLine(idiomText);
            foreach (var il in idiomLearns)
            {
                Console.WriteLine(il);
            }
            Console.Write("Enter number -> ");
            int number = int.Parse(Console.ReadLine());
            foreach (var il in idiomLearns)
            {
                if (il.Id == number)
                {
                    if (il.RealIdiom == true)
                    {
```

```
                        il.Idiom.Learned = true;
                        Console.WriteLine("Right");
                    }
                    else
                    {
                        Console.WriteLine("Wrong");
                    }
                }
            }
        }
    }
}
namespace IdiomProject.PresentationLayer
{
    internal interface IIdiomPL
    {
        void CreateIdiom();
        void GetAllIdioms();
    }
}
namespace IdiomProject.PresentationLayer
{
    internal interface IIdiomServicePL
    {
        void Study();
    }
}
namespace IdiomProject.PresentationLayer
{
    internal interface IMeaningPL
    {
        int CreateMeaning(int idIdiom);
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using IdiomProject.BusinessLogicLayer;

namespace IdiomProject.PresentationLayer
{
    internal class MeaningPL : IMeaningPL
    {
        IMeaningBLL meaningBLL;

        public MeaningPL(IMeaningBLL meaningBLL)
        {
            this.meaningBLL = meaningBLL;
        }
        public int CreateMeaning(int idIdiom)
        {
            string meaningText = Helper.StringInputCheck("Create Meaning text?: ", 150);
            return meaningBLL.CreateMeaning(meaningText, idIdiom);
        }
    }
}
using IdiomProject.BusinessLogicLayer;
using IdiomProject.BusinessLogicLayer.Models;
using IdiomProject.DataAccessLayer;
using IdiomProject.PresentationLayer;
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;

namespace IdiomProject
{
    internal class Factory
    {
        public static Idiom CreateIdiom(string phrase, bool learned)
        {
            return new Idiom(phrase, learned);
        }
        public static Meaning CreateMeaning(string meaningText, int idiomId)
        {
            return new Meaning(meaningText, idiomId);
        }
        public static WrongMeaning CreateWrongMeaning(string wrongMeaningText, int
meaningId)
        {
            return new WrongMeaning(wrongMeaningText, meaningId);
        }
        public static IDB CreateDB()
        {
            return new DB();
        }
        private static IIdiomBLL CreateIdiomBLL(IDB db)
        {
            return new IdiomBLL(db);
        }
        private static IIdiomPL CreateIdiomPL(IIdiomBLL idiomBLL, IMeaningPL meaningPL)
        {
            return new IdiomPL(idiomBLL, meaningPL);
        }
        private static IMeaningBLL CreateMeaningBLL(IDB db)
        {
            return new MeaningBLL(db);
        }
        private static IMeaningPL CreateMeaningPL(IMeaningBLL meaningBLL)
        {
            return new MeaningPL(meaningBLL);
        }
        private static IIdiomServiceBLL CreateIdiomServiceBLL(IDB db)
        {
            return new IdiomServiceBLL(db);
        }
        private static IIdiomServicePL CreateIdiomServicePL(IIdiomServiceBLL
idiomServiceBLL)
        {
            return new IdiomServicePL(idiomServiceBLL);
        }
        public static void Start()
        {
            IDB db = CreateDB();
            IIdiomBLL idiomBLL = CreateIdiomBLL(db);
            IMeaningBLL meaningBLL = CreateMeaningBLL(db);
            IMeaningPL meaningPL = CreateMeaningPL(meaningBLL);
            IIdiomPL idiomPL = CreateIdiomPL(idiomBLL, meaningPL);

            IIdiomServiceBLL idiomServiceBLL = CreateIdiomServiceBLL(db);
            IIdiomServicePL idiomServicePL = CreateIdiomServicePL(idiomServiceBLL);
            idiomServicePL.Study();
            idiomServicePL.Study();
            idiomServicePL.Study();
            idiomServicePL.Study();
```

```
            Console.ReadKey();

            //idiomPL.CreateIdiom();
            //idiomPL.GetAllIdioms();
        }
    }
}
using IdiomProject;

Factory.Start();

Console.ReadKey();
```