Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до магістерської кваліфікаційної роботи

магістр
(освітньо-кваліфікаційний рівень)

на тему: Розробка методу та програмного забезпечення для підвищення ефективності засвоєння студентами знань і навичок

Виконав: студент II курсу

групи 1ПІ-18м

спеціальності

121 – інженерія програмного забезпечення
(повне найменування вищого навчального закладу)

Кавка О. О.
(прізвище та ініціали)

Керівник: к.т.н., доц. Романюк О. В.
(прізвище та ініціали)

Рецензент: к.т.н., доц. Іванчук Я. В.
(прізвище та ініціали)

ВНТУ – 2019

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 – інженерія програмного забезпечення

УЗГОДЖЕНО	ЗАТВЕРДЖУЮ
Директор ТОВ «Дельфи»	Завідувач кафедри ПЗ
Бондар Н. П.	д.т.н., професор Романюк О. Н.
«___» _____ 2019 року	«___» _____ 2019 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Кавці Олексію Олександровичу

1. Тема роботи: «Розробка методу та програмного забезпечення для підвищення ефективності засвоєння студентами знань і навичок», керівник роботи: Романюк Оксана Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від "__"_____2019 року №___

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи: набір тем дисципліни «Алгоритми та структури даних», алгоритмічні задачі з архівів Codeforces, Timus Online Judge, E-Olymp, крива забування Еббінгауза, метод інтервального повторення – метод Лейтнера, кількість груп задач: 10.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз та порівняльна характеристика методів та інструментальних засобів підвищення ефективності засвоєння студентами знань та навичок; розробка методів для підвищення ефективності формування практичних навичок із розв'язування алгоритмічних задач в галузі інженерії програмного забезпечення; розробка програмних засобів реалізації модифікованого методу Лейтнера; тестування розробленого програмного продукту; економічна частина; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; мета, об'єкт, предмет та завдання дослідження; наукова новизна та практична цінність; крива забування Еббінгауза; оригінальний метод Лейтнера; модифікований метод Лейтнера; удосконалений метод оцінювання складності задач; архітектура системи; ієрархія класів; ER-модель предметної галузі; тестування системи; використання системи; аналіз ефективності розробленої модифікації методу Лейтнера;

економічне обґрунтування; апробація, публікації, впровадження; заключний слайд.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Романюк О.В., к.т.н., доцент кафедри ПЗ		
5	Бальзан М.В. к.е.н., доцент кафедри ЕПВМ		

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної області та порівняльна характеристика методів підвищення ефективності вивчення алгоритмів та структур даних	04.09.2019 – 30.09.2019	Виконано
2	Розробка модифікованого методу Лейтнера для підвищення ефективності формування практичних навичок розв’язування алгоритмічних задач в програмній інженерії	30.09.2019 – 20.10.2019	Виконано
3	Розробка удосконаленого методу оцінювання складності програмних алгоритмічних задач	21.10.2019 – 27.10.2019	Виконано
4	Розробка програмних засобів реалізації модифікованого методу Лейтнера	28.10.2019 – 10.11.2019	Виконано
5	Тестування розробленого програмного продукту	11.11.2019 – 15.11.2019	Виконано
6	Економічна частина	16.11.2019 – 21.11.2019	Виконано

Студент _____ Кавка О.О.
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ Романюк О.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

У магістерській кваліфікаційній роботі «Розробка методу та програмного забезпечення для підвищення ефективності засвоєння студентами знань і навичок» розроблено модифікацію методу Лейтнера для формування практичних навичок з використання алгоритмів та структур даних при розв’язуванні програмних алгоритмічних задач, а також запропоновано удосконалений метод оцінювання складності алгоритмічних задач.

У ході виконання роботи було проаналізовано існуючі методи підвищення ефективності засвоєння студентами знань та навичок, а також існуючі програмні аналоги. Було розроблено та перевірено роботу програмного забезпечення для впровадження та дослідження розроблених методів за допомогою таких засобів: мова програмування C#, фреймворки ASP.NET Core 3.0, Entity Framework Core та gRPC for .NET Core.

ABSTRACT

Master's qualification work "Development of method and software for improving students' mastering of knowledge and skills" developed a modification of Leitner's method for forming practical skills in the use of algorithms and data structures in solving software algorithmic problems, as well as the improved method of estimating the complexity of algorithmic problems.

In the course of work, the existing methods of improving the students' mastering of knowledge and skills were analyzed, as well as the existing program analogues. Software has been developed and tested to implement and research the developed methods using the following tools: C# programming language, ASP.NET Core 3.0, Entity Framework Core, and gRPC for .NET Core.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасне сусрпільство живе в епоху інформаційних технологій. Google, Microsoft, Apple, Uber – ці та інші IT-компанії кардинально змінили світ. Провідні IT-компанії сприяють розвитку молодих фахівців, шляхом організації та фінансування численних конкурсів, хакатонів та олімпіад. Такі заходи сприяють розвитку саме знань та навичок з використання алгоритмів та структур даних.

Знання та вміння використовувати алгоритми та структури даних – один з фундаментальних критеріїв, що характеризують високопрофесійного фахівця у галузі інформаційних технологій. Такі навички як здатність підібрати оптимальний спосіб розв’язання задачі, оцінити швидкодію алгоритму, споживання системних ресурсів – навички, пов’язані із розв’язанням алгоритмічних задач – високо цінуються на міжнародному ринку праці у сфері інформаційних технологій. Вже у найближчому майбутньому інформаційні технології в Україні можуть стати першочерговою з економічної точки зору галуззю. Таким чином, постає важливе завдання розвитку професійних та кваліфікованих фахівців, що будуть конкурентоспроможними на світовому ринку, здатними на розробку інноваційних технологічних рішень. Однак, на даний момент не існує спеціалізованого методу, що призначений для розвитку навичок використання алгоритмів та структур даних.

. Традиційно для підвищення рівня засвоєння знань використовують метод інтервального повторення інформації Лейтнера [1], що базується на дослідженні темпів забування інформації Германа Еббінгауза і періодичному повторенні інформації, яка ще не засвоєна. Особливу ефективність цей метод демонструє при вивченні іноземних мов [2]. Однак механізм засвоєння навичок та знань з використання алгоритмів та структур даних має складнішу природу і вимагає під час навчання поступового переходу від засвоєння менш складних навичок до більш складних [3].

У зв’язку з цим, важливою задачею є розробка методу для засвоєння навичок та знань з використання алгоритмів та структур даних, який дозволить ранжувати алгоритмічні знання за ступенем їх складності та пропонувати для повторення незасвоєні алгоритмічні зрання за зростанням їх складності.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення ефективності засвоєння студентами знань та навичок з використання алгоритмів та структур даних шляхом розробки та впровадження нових методів та засобів навчання.

Основними задачами дослідження є:

- аналіз існуючих методів формування знань у галузі інформаційних технологій;
- розробка нових або модифікація існуючих методів для підвищення ефективності засвоєння знань та навичок у галузі програмної інженерії;
- розробка програмної системи, на основі якої буде можливе впровадження та дослідження методів навчання;
- тестування розробленої програмної системи.

Об’єкт дослідження – процес підвищення ефективності засвоєння студентами знань та навичок з використання алгоритмів та структур даних.

Предмет дослідження – методи та засоби підвищення ефективності засвоєння студентами знань та навичок з використання алгоритмів та структур даних.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів для дослідження предметної області; теорія проектування баз даних для розробки бази даних програмної системи; теорія проектування архітектури програмного забезпечення для розробки архітектури програмної системи, метод комп’ютерного моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку набув метод Лейтнера, який, на відміну від відомого, враховує складність освоєних задач, що дає можливість підвищити ефективність засвоєння студентами навичок з використання алгоритмів та структур даних.

2. Удосконалено метод оцінювання складності алгоритмічних програмних задач, який, на відміну від існуючих, враховує кількість успішних розв'язків, відсоток успішних розв'язків та середню кількість спроб до успішного розв'язку, що підвищує об'єктивність оцінки.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритм та програмні засоби для підвищення ефективності засвоєння студентами навичок використання алгоритмів та структур даних.

Впровадження. Результати дослідження використовуються на підприємстві ТОВ «Дельфи» для навчання стажерів, що підтверджується відповідним актом.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати:

- аналіз особливостей застосування методу Лейтнера для формування практичних навичок розв'язування алгоритмічних задач в програмній інженерії [3];
- модифікація методу Лейтнера для підвищення ефективності засвоєння практичних навичок розв'язування алгоритмічних задач [4];
- розробка методу оцінювання складності алгоритмічних програмних задач [5].

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародних конференціях:

- XII міжнародна науково-практична конференція «Інформаційні технології і автоматизація – 2019» (Одеса, 2019);
- XXXVI Міжнародна інтернет-конференція «Інновації науки XXI століття» (Вінниця, 2019).
- Міжнародна науково-практична конференція «Електронні інформаційні ресурси в освіті і науці: створення, використання, доступ» (Вінниця, 2019).

Публікації. Основні результати досліджень опубліковано в 3 наукових працях, у тому числі 3 – у матеріалах конференцій.

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку літератури, що містить 29 найменувань, 5 додатків. Робота містить 45 ілюстрацій, 11 таблиць.

1 АНАЛІЗ ТА ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА МЕТОДІВ ТА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ЗАСВОЄННЯ СТУДЕНТАМИ ЗНАНЬ ТА НАВИЧОК

1.1 Аналіз методів підвищення ефективності засвоєння студентами знань та навичок

Існує декілька методів підвищення ефективності вивчення інформації [6]. До найбільш поширених відносяться:

- Метод локусів, або «палац пам’яті». Назва методу походить від латинського слова «locus», що означає «місце». Ще одна назва методу – «просторова мнемоніка» – описує його суть. Ідея полягає у тому, щоб уявити певну знайому локацію – квартиру, будинок, палац – і створити асоціативні зв’язки між фактами та певними «локусами» – місцями. Таким чином, уявна прогулянка знайомими місцями допомагає відтворити у пам’яті відповідну асоційовану інформацію.

- Мнемоніка. Це – набір способів запам’ятовування, що зазвичай використовується для вивчення структурованих даних – наприклад, кольорів веселки, порядкова класифікація зірок чи літери, перед якими префікс «з-» перетворюється на «с-». До найбільш поширених технік мнемонічного запам’ятовування належать: акронім, акровірш, ключові слова, рифмізація, образ-ім’я та формування ланцюжка.

- Метод фрагментування. Декілька елементів, які необхідно запам’ятати, об’єднуються в одну групу, запам’ятати яку простіше, ніж всі елементи окремо. Часто застосовується для запам’ятовування номерів телефону чи банківських карток.

- Метод інтервальних повторів. Ідея цієї техніки полягає в тому, що інформація постійно повторюється відповідно до певних зростаючих інтервалів.

- Метод сторітелінгу – створення історії і яскравих образів, що обгортають інформацію, яку необхідно запам’ятати.

Проаналізовані методи запам’ятовування орієнтовані виключно на запам’ятовування інформації, проте не мають на меті сприяти розвитку практичних навичок застосування тих чи інших знань.

Окрім того, такі методи як мнемоніка, метод локусів, метод фрагментування, метод сторітелінгу – не є стандартизованими, а доволі абстрактними, що унеможливлює їх впровадження у програмній системі. Серед розглянутих методів підвищення ефективності засвоєння знань та навичок лише метод інтервальних повторів може бути модифікований та впроваджений у автоматизованій програмній системі.

Розглянуті методи призначені для запам’ятовування інформації, а не для формування практичних навичок. Вони доволі загальними і не пропонують чіткої послідовності дій, необхідних до виконання для досягнення мети – розвитку навичок використання алгоритмів та структур даних для розв’язання програмних задач. Лише метод інтервальних повторень має потенціал відповідної модифікації. Таким чином, у контексті теми магістерської кваліфікаційної роботи лише цей метод є перспективним для подальшого аналізу та розробки.

1.2 Порівняльна характеристика методів оцінювання складності алгоритмічних програмних задач

У результаті проведеного дослідження було виявлено два найпоширеніші способи оцінювання складності алгоритмічних задач, які використовуються у розглянутих онлайн-сервісах, що надають архів алгоритмічних задач та інструменти перевірки правильності розв’язків.

Перший спосіб – це відображення кількості осіб, що розв’язали задачу. Така статистична інформація доступна у всіх архівах алгоритмічних задач. Подібний підхід складно назвати методом оцінювання складності задачі, проте він дає користувачу приблизне уявлення про складність задачі. Суттєвим недоліком цього методу є те, що отриманий показник сильно залежить він популярності задачі.

Другий спосіб – використання формули, що враховує «вік» задачі та кількість осіб, що її розв’язали. Такий метод використовує сервіс Timus Online Judge (рис. 1.1).

Архив задач. Том 12

Решенные задачи: показывать | скрывать • Упорядочить по: номеру | авторам | сложности ← Том 11

	Номер	Название	Источник	Авторы	Сложность
✓	2100	Свадебный обед	Чемпионат УрФУ среди юниоров 2016	3599	30
	2101	Рыцарский щит	Чемпионат УрФУ среди юниоров 2016	64	1469
–	2102	Миша и криптография	Чемпионат УрФУ среди юниоров 2016	843	127
	2103	Корпоративная почта	Чемпионат УрФУ среди юниоров 2016	95	1035
	2104	Игра с полоской	Чемпионат УрФУ среди юниоров 2016	377	281
	2105	Алиса и Боб на велосипедах	Чемпионат УрФУ среди юниоров 2016	471	226
	2106	Десериализация	Чемпионат УрФУ среди юниоров 2016	137	739
	2107	Oppa Funcan Style	Чемпионат УрФУ среди юниоров 2016	66	1430
	2108	Олег и маленькие пони	Чемпионат УрФУ среди юниоров 2016	303	347
	2109	Туризм на Марсе	Чемпионат УрФУ среди юниоров 2016	252	415
	2110	Удалить или максимизировать	Чемпионат УрФУ среди юниоров 2016	127	793
	2111	Платон	Вузовско-академическая олимпиада по информатике 2019	544	89
	2112	Полевые логи	Вузовско-академическая олимпиада по информатике 2019	81	541
	2113	Пережить потоп	Вузовско-академическая олимпиада по информатике 2019	117	387
	2114	Моё ремесло	Вузовско-академическая олимпиада по информатике 2019	56	745
	2115	День знаний	Вузовско-академическая олимпиада по информатике 2019	261	182
	2116	Он вам не конь	Вузовско-академическая олимпиада по информатике 2019	77	565
	2117	Полифемовы тройки	Вузовско-академическая олимпиада по информатике 2019	90	492
	2118	Шифровка	Петрозаводск лето 2018. t.me/umnik_team Contest	11	2233
	2119	Древесная оболочка	Петрозаводск лето 2018. t.me/umnik_team Contest	32	1116
	2120	Средний вес дерева	Петрозаводск лето 2018. t.me/umnik_team Contest	21	1512
	2121	Пересечение парабол	Петрозаводск лето 2018. t.me/umnik_team Contest	58	689
	2122	Хэмминг	Петрозаводск лето 2018. t.me/umnik_team Contest	9	2468

Рисунок 1.1 – Приклад оцінювання складності задачі на сервісі Timus Online Judge

Сама формула не є опублікованою, доступний лише її опис на сайті вищевказаного сервісу [7]. Цей метод більш точний, ніж перший, але він не вирішує проблему залежності показника складності він популярності задачі. Формула складності задачі, що використовується сервісом Timus Online Judge, піддається критиці за недостовірність [8].

Проведене дослідження виявило відсутність наукових робіт на тему оцінки складності алгоритмічних програмних задач, а також суттєві недоліки

практичних аналогів, що полягають у впливі популярності задачі на оцінку її складності.

1.3 Аналіз інструментальних засобів підвищення ефективності засвоєння студентами знань та навичок

Інструментальними засобами для розгляду є так звані «online judge» сервіси, які містять умови різноманітних алгоритмічних задач, а також тестову систему, що дозволяє автоматично перевіряти запропоновані користувачами розв'язки. Таких сервісів є багато, розглянемо найбільш популярні та функціональні.

Codeforces – проект, розроблений у 2010 році Михайлом Мірзаяновим [9]. На сайті регулярно проводяться онлайн-змагання з розв'язання програмних алгоритмічних задач, є великий архів задач з виставленими тегами, що вказують на клас завдання, а також інтегрована тестова система (рис. 1.2). Сайтом користується велика кількість учасників з багатьох країн світу.

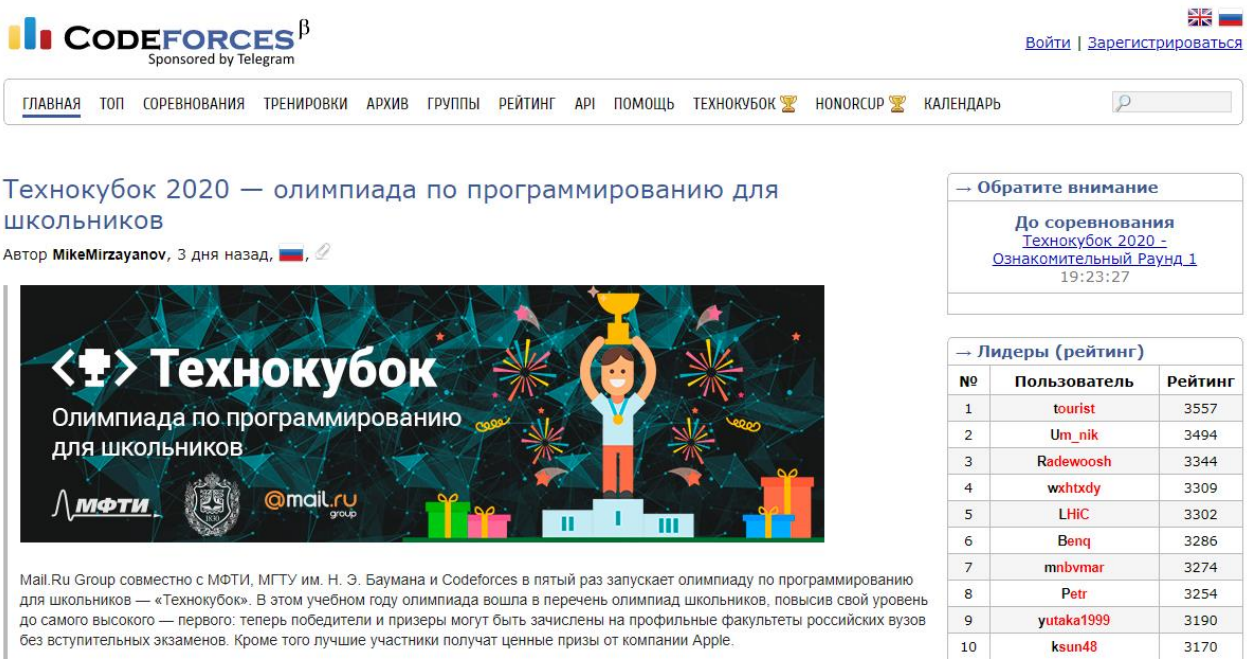

Рисунок 1.2 – Головна сторінка сайту Codeforces

Timus Online Judge – один з перших і найпопулярніших «online judge» сервісів в Україні, Росії та Білорусі (рис. 1.3) [10]. Має архів з більш ніж тисячі задач, для кожної з яких доступні теги, а також складність завдання – відносний показник, що базується на кількості учасників, що розв'язали задачу.

Рисунок 1.3 – Картка користувача на сайті Timus Online Judge

E-Olymp – Інтернет-портал організаційно-методичного забезпечення дистанційних олімпіад з програмування для обдарованої молоді навчальних закладів України [11]. Портал розроблено Житомирським державним університетом ім. Івана Франка, кафедрою прикладної математики та інформатики за фінансової підтримки МОН та Державного комітету України з питань науки, інновацій та інформатизації в рамках Державної програми «Інформаційні та комунікаційні технології в освіті і науці» у 2009-2010 роках. Є

великий архів задач (більш ніж 6000), для яких доступна статистика розв’язків, що дає змогу визначити складність задачі, але задачі гірше протеговані, ніж на інших розглянутих ресурсах. Наявна тестова система (рис. 1.4).

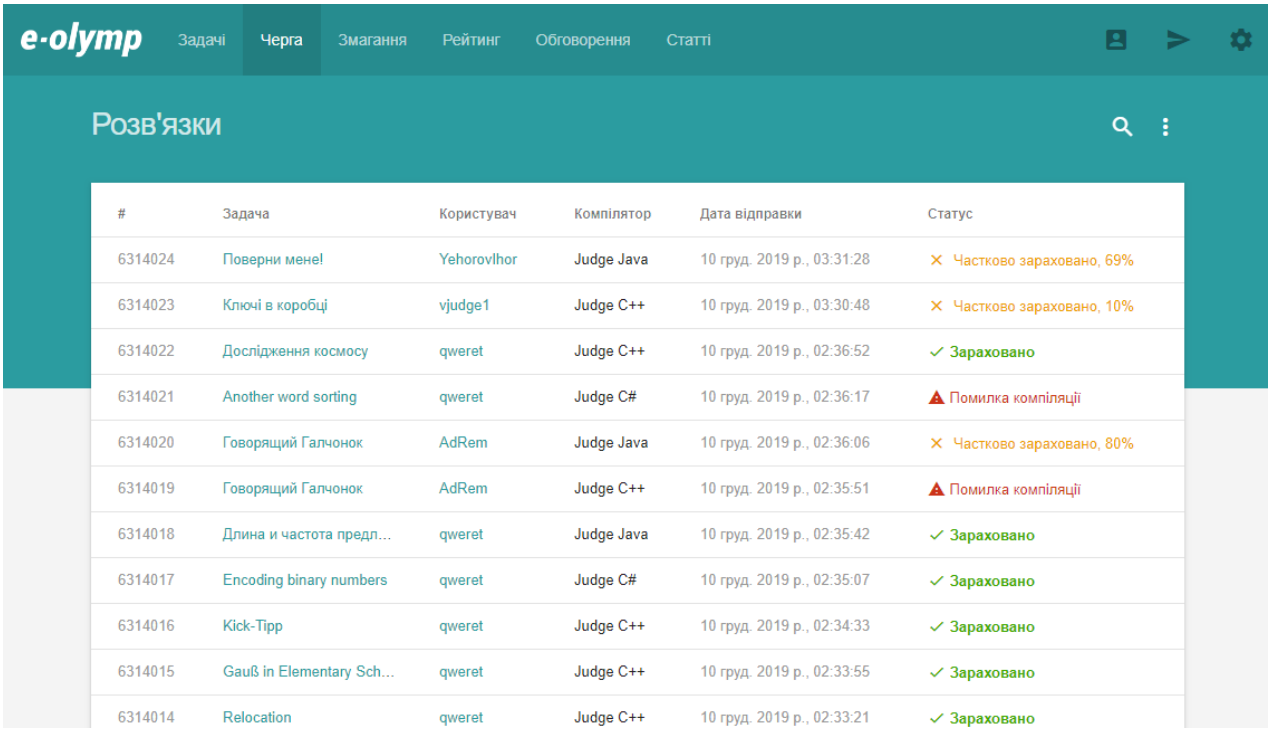

Рисунок 1.4 – Сторінка сайту E-Olymp

У всіх розглянутих сервісів є суттєвий недолік: відсутність модуля підбору задач. Користувач вимушений самостійно шукати задачі, опираючись на теги, опис задачі та доступну інформацію про складність. Подібний підхід до розвитку навичок з використання алгоритмів та структур даних для розв’язання програмних задач є неефективним. Відсутність чіткого плану навчання, низька ефективність процесу вивчення ставлять під сумнів результативність використання користувачем вищезгаданих сервісів. Таким чином, є актуальною проблема розробки методу та відповідних інструментальних засобів, спрямованих на підвищення ефективності засвоєння студентами знань і навичок з використання алгоритмів та структур даних для розв’язання програмних задач.

1.4 Постановка задачі

У зв’язку з відсутністю спеціалізованих методів, спрямованих на ефективне формування навичок з використання алгоритмів та структур даних, перед початком розробки було поставлено наступні задачі:

- розробити метод, спрямований на ефективне формування навичок з використання алгоритмів та структур даних;
- розробити метод оцінювання складності алгоритмічних програмних задач, на об’єктивність якого не впливає популярність задачі;
- розробити програмну систему для впровадження методу;
- розробити веб-складову системи для доступу через мережу Інтернет;
- забезпечити можливість впровадження декількох методів підбору задач та їх одночасного використання на різних групах користувачів;
- провести тестування системи, виправити виявлені помилки.

Необхідно забезпечити достовірність зібраних даних, шляхом унеможливлення використання одним користувачем різних методів підбору задач або виключенням таких користувачів із вибірки даних для аналізу.

1.5 Висновки

У результаті аналізу предметної області було розглянуто існуючі методи та інструментальні засоби підвищення ефективності засвоєння студентами знань та навичок, а також методи оцінки складності алгоритмічних задач. Зроблено висновок про необхідність розробки спеціалізованого методу для формування навичок з використання алгоритмів та структур даних, оскільки не існує відповідного спеціалізованого методу, а найближчі аналоги – онлайн-сервіси, що надають архів алгоритмічних задач, а також тестову систему для перевірки правильності розв’язків – не пропонують жодного механізму інтелектуального підбору задач. Також було зроблено висновок про необхідність розробки методу оцінювання складності алгоритмічних програмних задач, на об’єктивність якого не впливає популярність задачі. Найкращим рішенням даної проблеми є розробка програмної системи, на базі якої буде впроваджено відповідний метод.

2 РОЗРОБКА МЕТОДІВ ДЛЯ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ ФОРМУВАННЯ ПРАКТИЧНИХ НАВИЧОК ІЗ РОЗВ'ЯЗУВАННЯ АЛГОРИТМІЧНИХ ЗАДАЧ В ГАЛУЗІ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз методу Лейтнера

За основу взято дослідження пам'яті, проведене німецьким психологом Германом Еббінгаузом у 1885 році. Вчений запропонував метод запам'ятовування беззмістовних закритих складів, що не викликають жодних змістовних асоціацій. У процесі дослідження було виявлено, що після першого безпомилкового повторення серії таких складів "забування" відбувається дуже швидко. Протягом першої години забувається до 60% всієї отриманої інформації. Через 10 годин після запам'ятовування, у пам'яті залишається 35% вивченого. Надалі процес забування відбувається повільно, і через 6 днів у пам'яті залишається 20% від початкової кількості вивчених складів. Стільки ж залишається у пам'яті через місяць (рисунок 2.1) [12].

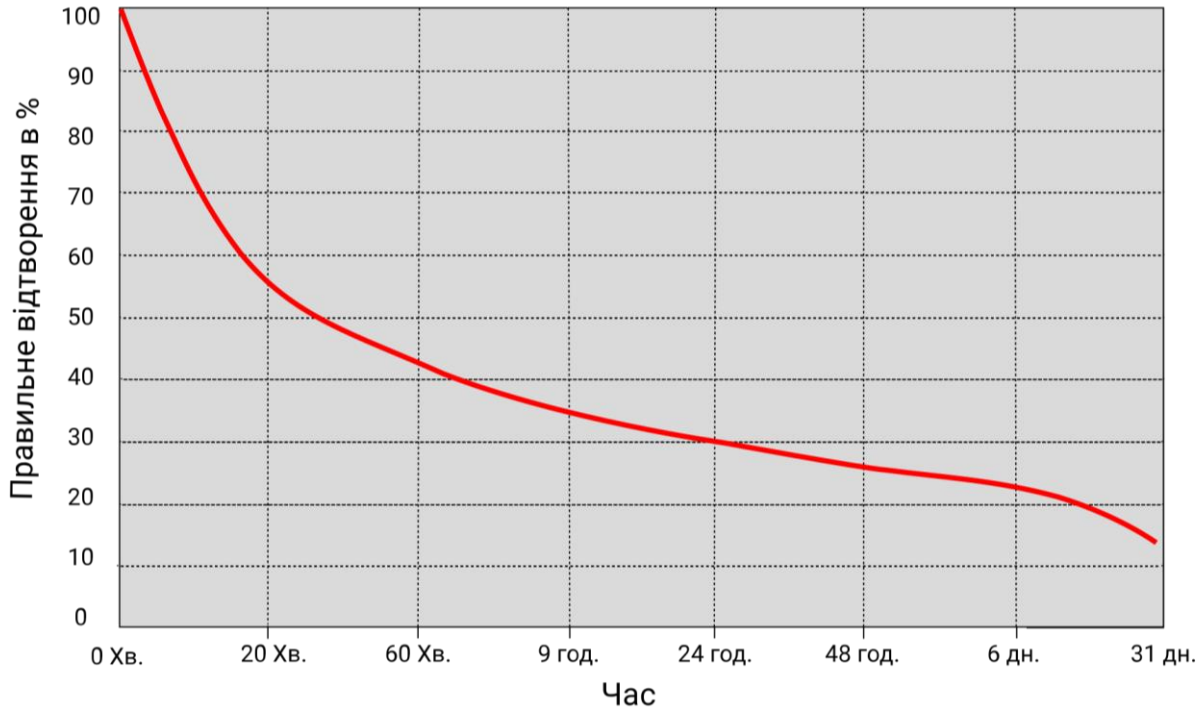

Рисунок 2.1 – Крива забування Еббінгауза

На основі результатів дослідження Еббінгауза було розроблено техніку інтервальних повторень, що полягає в повторенні засвоєного матеріалу через певні зростаючі інтервали часу. Ідея про те, що інтервальне повторення можна використовувати для покращення процесу навчання, вперше була запропонована в книзі «Психологія навчання» («Psychology of Study») професора Алека Мейса у 1932 році [13].

Відкритою проблемою техніки інтервального повторення є підбір часових інтервалів між повторами [14].

В 1970-х роках німецький вчений та журналіст Себастьян Лейтнер запропонував метод для ефективного запам’ятовування і повторення за допомогою флеш-карток. Метод полягає у тому, що картки з інформацією діляться на групи. Картки з першої групи демонструються учневі найчастіше, картки з другої групи – трохи рідше, і так далі. Якщо учень правильно пам’ятає інформацію з картки, вона переміщується у наступну групу. Якщо помиляється – картка переміщується у попередню або першу групу (рис. 2.2) [1].

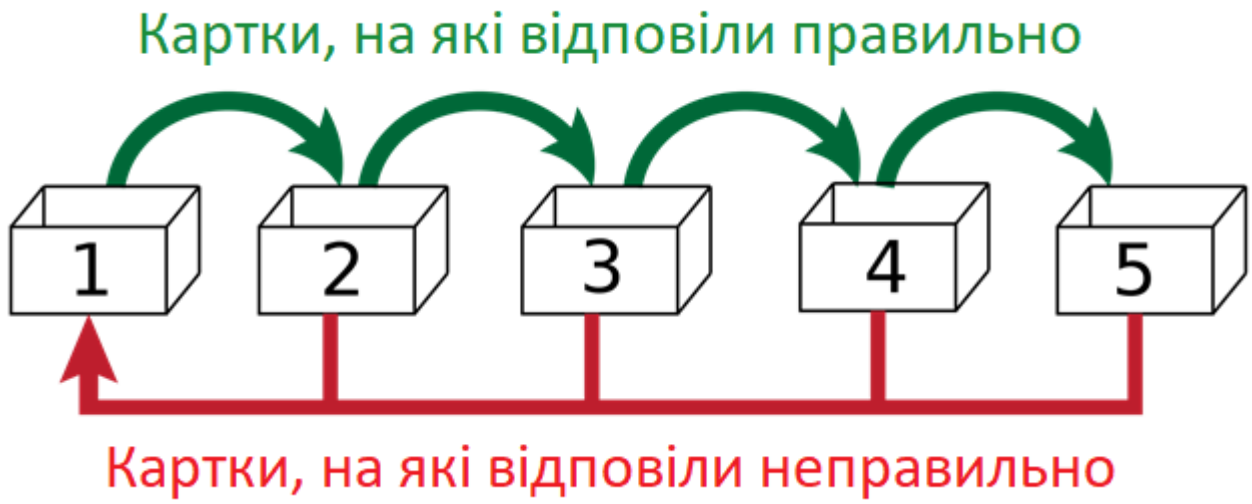

Рисунок 2.2 – Візуалізація варіації методу Лейтнера

Метод Лейтнера набув широкого розповсюдження, особливо у галузі вивчення іноземних мов. Ось приклади сервісів, що використовують метод Лейтнера:

- LinguaLeo – сервіс для вивчення іноземних мов, використовує метод для вивчення слів [15];
- MemRise – сервіс, що орієнтований переважно на вивчення іноземних мов, а також на набуття певних загальних знань [16];
- Mnemosyne – програмний продукт, що призначений для запам’ятовування визначеної користувачем інформації [17].

Тим не менш, попри наявність великої кількості сервісів, що застосовують метод Лейтнера для вивчення іноземних мов, він ще не використовувався для набуття навичок у галузі програмної інженерії.

2.2 Розробка модифікованого методу Лейтнера для підвищення ефективності засвоєння студентами знань та навичок

Метод Лейтнера працює для запам’ятовування інформації, яку можна поділити на фрагменти, а також однозначно визначити правильність запам’ятовування. Саме тому цей метод широко застосовується при вивченні словника іноземної мови, тощо. Утім, формування практичних навичок - значно складніше, ніж механічне запам’ятовування. Проблема полягає у тому, що на успішність розв’язання алгоритмічної задачі в області програмної інженерії впливають не лише теоретичні знання. Окрім знання алгоритмів, структур даних та обмежень у їх використанні, необхідно мати досвід розробки програмного коду для вказаних алгоритмів, розуміння типових задач і методів їх розв’язання, а також здатність звести задану задачу до типового вигляду. Для прикладу, розглянемо наступну задачу.

«Дано N міст і M доріг. Кожна дорога двостороння і з’єднує рівно два міста. Скільки нових доріг потрібно побудувати, щоб існував шлях між будь-якою парою міст?»

Проілюструємо задачу схемою, зображеною на рисунку 2.3. Для розв’язання задачі необхідно помітити, що вона легко зводиться до типової задачі – підрахунку кількості компонент зв’язності. Здатність робити це залежить від досвіду розв’язування типових задач. Після цього можна

підрахувати кількість компонент зв'язності за допомогою пошуку в ширину, пошуку в глибину чи іншого алгоритму обходу графа. Цей етап потребує знання відповідних алгоритмів та вміння їх імплементувати.

Окрім коректності роботи розробленого програмного засобу, здатного розв'язати поставлену задачу, існує не менш важливий показник, а саме ресурси, яких потребує програма для розв'язання задачі. Якщо розроблений додаток розв'язує задачу, але потребує для цього на порядок більше часу чи оперативної пам'яті, ніж це необхідно при оптимальному виборі алгоритму, задача не може вважатись розв'язаною правильно.

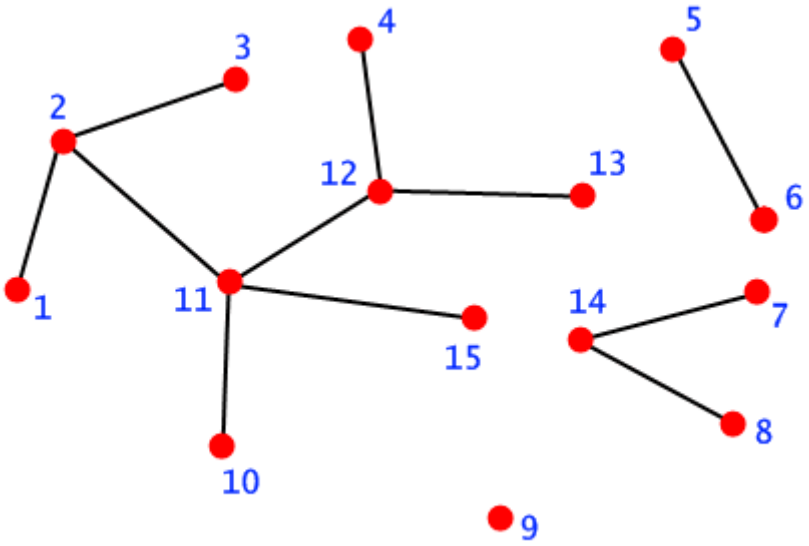

Рисунок 2.3 – Схема міст і доріг між ними

Суттєвою проблемою є нерівнозначність флеш-карток і алгоритмічних задач. При традиційному застосуванні методу Лейтнера конкретна флеш-картка і є інформацією, яку потрібно засвоїти. У поточному ж випадку, конкретна алгоритмічна задача - це лише прояв інформації, яку потрібно засвоїти. Разом з тим, задачі неоднорідні за своєю складністю, і це також потрібно враховувати.

Таким чином, для застосування метода Лейтнера у контексті формування навичок з розв'язання алгоритмічних задач у програмній інженерії, необхідно внести два корективи.

Перший коректив – зміна механізму вибору завдання. У класичному формулюванні, учневі пропонуються одні і ті ж картки. У поточному випадку, учневі пропонуватимуться лише задачі, які він ще не розв’язав – з відповідним рівнем складності.

Другий коректив – це формулювання комплексних критеріїв успішності, що будуть застосовуватись при оцінці розв’язку задачі.

Запропоновано наступні критерії успішності:

1. Коректність розв’язку – здатність розробленого додатку дати правильну відповідь при будь-яких коректних вхідних даних. Це - основний критерій успішності розв’язку.

2. Споживання системних ресурсів – процесорний час та інші ресурси системи, у якій виконується розроблений додаток. Залежно від вхідних даних, вимоги до системних ресурсів можуть відрізнятись [18], тому при оцінці обчислювальної складності алгоритмів [19] в загальному випадку береться найгірший можливий випадок – максимальний показник використання ресурсу. Цей критерій може бути поділений на підпункти:

- Максимальний процесорний час, необхідний для розв’язання задачі.
- Максимальне одночасне споживання оперативної пам’яті.

Перший критерій – коректність розв’язку – це основний критерій, на якому базується метод Лейтнера у його традиційному формулюванні. Його гарантує наявність вичерпного набору тестів.

Другий критерій – споживання системних ресурсів – зазвичай включається у перший, стаючи його частиною. Якщо запропоноване рішення споживає більше системних ресурсів, ніж відведено на вирішення задачі, рішення вважається неправильним.

На основі вищевикладеного, було розроблено модифікований метод Лейтнера, що може бути використаний для формування навичок з використання алгоритмів та структур даних. У цій модифікації картки – це алгоритми, структури даних, техніки розв’язання задач – наприклад, «пошук у глибину», «дерево відрізків», «динамічне програмування» тощо.

Картки розбиваються на десять груп. У першій групі знаходяться найменш засвоєні картки. У десятій групі – найкраще засвоєні картки. У випадку, якщо учень проходить тестування успішно, картка переміщується до наступної по порядку групи. Якщо результат тестування незадовільний, картка переміщується до попередньої групи. Якщо учень не проходить тестування для картки протягом 7 днів, картка переміщується до попередньої групи.

Частота повторень для групи характеризується послідовністю Фібоначчі і наведена у таблиці 2.1.

Таблиця 2.1

Частота повторень для групи

Номер групи	Частота повторень (дні)
1	1
2	2
3	3
4	5
5	8
6	13
7	21
8	34
9	55
10	89

Наприклад, картки з першої групи пропонуються для тестування щодня. Картки з десятої групи – раз на три місяці.

Тестування полягає у розв’язуванні набору з п’яти задач, пов’язаних з темою картки. Підбір задач є ключовим фактором успішності застосування методу.

Схему алгоритму модифікації методу Лейтнера наведено на рисунку 2.4.

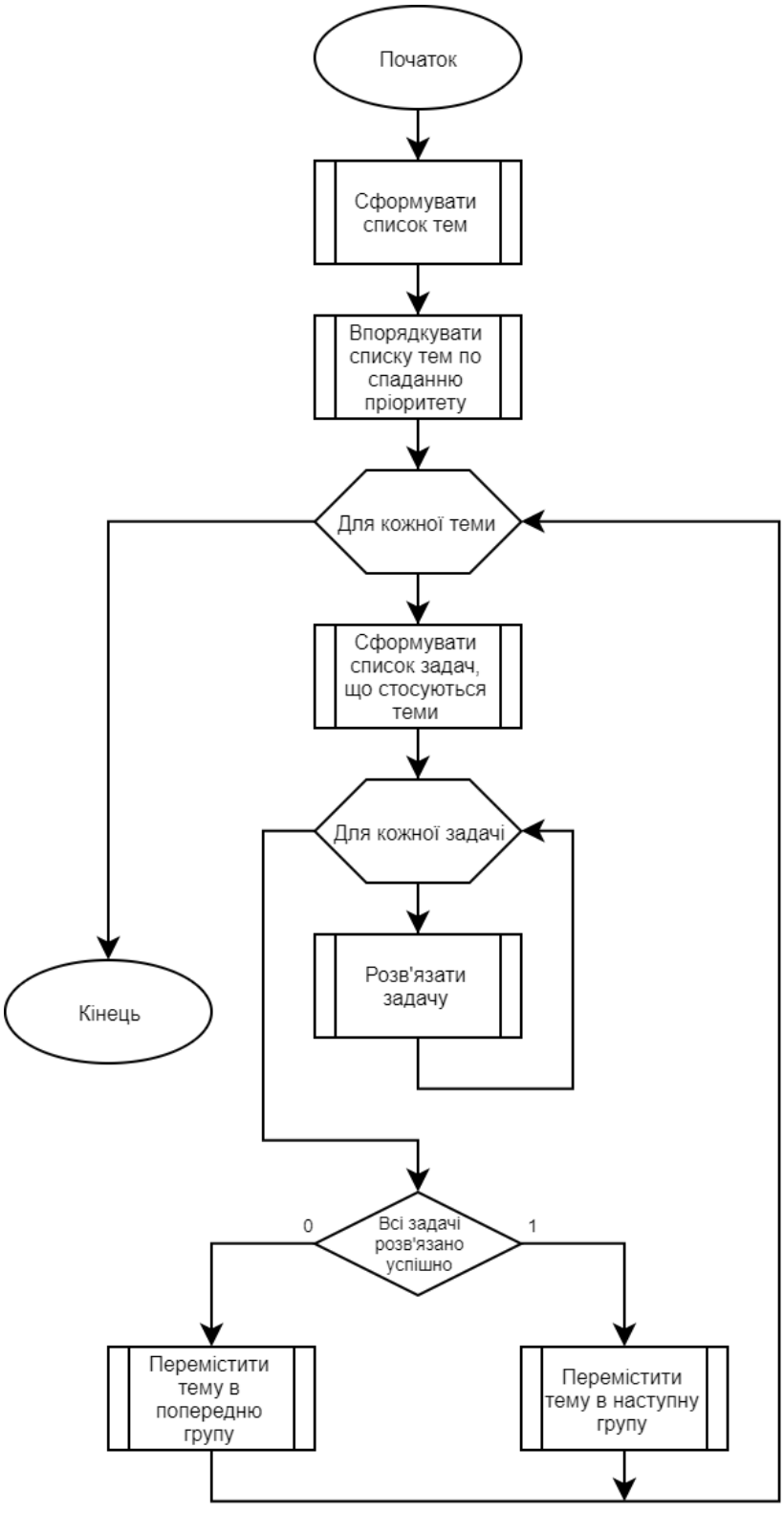

Рисунок 2.4 – Схема алгоритму модифікації методу Лейтнера

Задачі розбиваються на десять груп за рівнем складності. Для цього вони сортуються за складністю і діляться на десять рівних груп. При тестуванні використовуються задачі з відповідним рівнем складності. Оскільки задача може

бути пов’язана з декількома картками, для тестування відбираються задачі, які користувач може розв’язати. Наприклад, у користувача є картка «дерево (теорія графів)», що відноситься до п’ятої групи, і картка «алгоритм Крускала», що відноситься до першої групи. Таким чином, для тестування не буде використано задачу п’ятої групи складності, що пов’язана з обома вказаними картками, оскільки учень не в достатній мірі володіє темою «алгоритм Крускала».

Оскільки на початку тренувань учневі потрібно розв’язувати велику кількість задач, може виникати ситуація, у якій протягом одного дня учень не встигає пройти тестування для всіх запланованих карток. Для цього учневі потрібно визначити пріоритет кожної картки – таким чином, в першу чергу на тестування подаються картки з вищим пріоритетом.

2.3 Розробка удосконаленого методу оцінювання складності алгоритмічних програмних задач

Для кожної задачі визначається числова характеристика складності. Складність задачі базується на статистичних даних і залежить від:

- кількості людей, що успішно розв’язали задачу;
- кількості людей, що зробили щонайменше одну спробу розв’язку;
- середньої кількості спроб, необхідних для розв’язку задачі.

Числова характеристика складності – це ціле невід’ємне число. Значення 0 характеризує максимально просту задачу – задачу, яку з першого разу розв’язують усі, хто намагається. Максимальне значення складності не обмежується. Запропоновано формулу 2.1:

$$X = \left\lceil \frac{A * R^2}{S * \sqrt[5]{S}} * 100 \right\rceil, \qquad (2.1)$$

де X – числова характеристика складності задачі;

S – кількість осіб, що розв’язали задачу;

A – кількість осіб, що намагались розв’язати задачу;

R – середня кількість спроб, за яку було досягнуто правильне рішення.

Для підвищення точності, для тренування не використовуються задачі, які намагались розв’язати менше 100 осіб.

Особливості наведеної формули:

- Показник складності перебуває у зворотній пропорційності до відсотку осіб, що успішно розв’язали задачу.
- Показник складності перебуває у квадратичній залежності від середньої кількості спроб, необхідних для розв’язку задачі, оскільки більша кількість спроб характеризує складнішу задачу.
- Показник складності перебуває у зворотній залежності від кореня п’ятого степеня кількості осіб, що розв’язали задачу, оскільки більша кількість осіб, що розв’язали задачу, свідчить про меншу складність задачі.
- Результат множиться на 100 і округлюється вгору до найближчого цілого числа. Це дозволяє більш точно диверсифікувати задачі за складністю.

Розглянемо декілька можливих варіантів задач.

Задачу 1 намагалось розв’язати 2000 осіб, з яких 1600 успішно впорались із завданням. У середньому для цього знадобилось 1,5 спроби. Підставивши цю інформацію у формулу, отримаємо числову характеристику складності завдання.

$$X = \left\lceil \frac{2000 * 1.5^2}{1600 * \sqrt[5]{1600}} * 100 \right\rceil = 65.$$

Розрахована числова характеристика складності задачі становить 65.

Задачу 2 намагалось розв’язати 800 осіб, з яких 400 успішно впорались із завданням. У середньому для цього знадобилось 1,5 спроби.

Підставивши цю інформацію у формулу, отримаємо числову характеристику складності завдання.

$$X = \left\lceil \frac{800 * 1.5^2}{400 * \sqrt[5]{400}} * 100 \right\rceil = 136.$$

Показник підвищився, оскільки відсоток осіб, що впорались із задачею, зменшився.

Задачу 3 намагалось розв’язати 100000 осіб, з яких 50000 успішно впорались із завданням. У середньому для цього знадобилось 1,5 спроби. Підставивши цю інформацію у формулу, отримаємо числову характеристику складності завдання.

$$X = \left\lceil \frac{100000 * 1.5^2}{50000 * \sqrt[5]{50000}} * 100 \right\rceil = 52.$$

Не зважаючи на те, що порівняно з першими двома задачами середня кількість спроб не змінилась, а відсоток тих, хто розв’язав задачу, знизився, характеристика складності завдання знизилась. Це пов’язано зі збільшенням кількості осіб, що впорались із завданням.

У мережі Інтернет доступні безкоштовні онлайн-сервіси, що містять архіви алгоритмічних задач, а також систему тестування програмних рішень – такі як, Codeforces [7], Timus Online Judge [8], E-Olymp [9]. Використання таких масових сервісів дає доступ до статистики розв’язків алгоритмічних задач.

2.4 Висновки

У ході роботи над другим розділом магістерської кваліфікаційної роботи було проведено дослідження методу Лейтнера, сформульовано основні критерії перевірки коректності розв’язку програмної алгоритмічної задачі, розроблено та описано модифікацію методу Лейтнера для формування навичок з використання алгоритмів та структур даних для розв’язання програмних задач, розроблено та описано метод розрахунку числової характеристики складності задачі на основі статистичних даних.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ МОДИФІКОВАНОГО МЕТОДУ ЛЕЙТНЕРА

3.1 Розробка архітектури програмної системи

Програмна система поділятиметься на декілька взаємодіючих систем, як це зображено на рисунку 3.1.

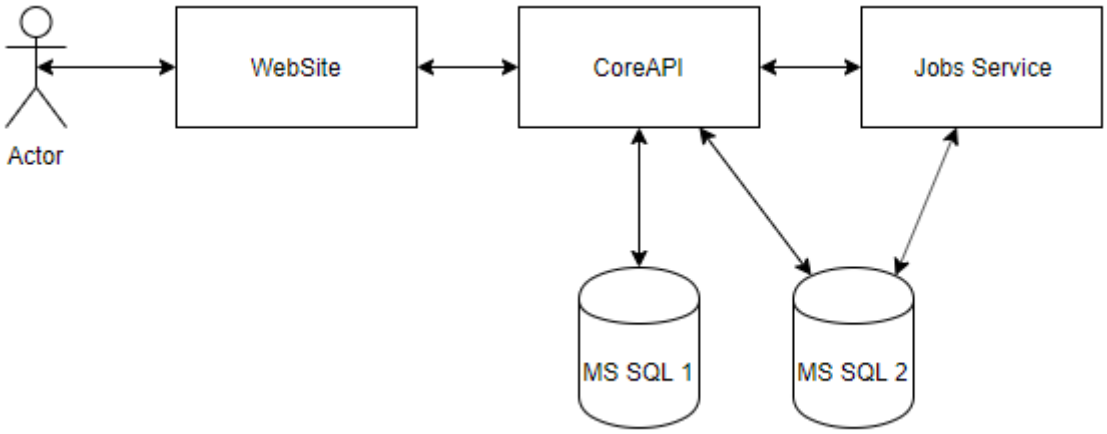

Рисунок 3.1 – Архітектура програмної системи

Actor – це користувач системи. Він взаємодіє із системою через WebSite – користувацький веб-інтерфейс.

WebSite – веб-сайт, що надає користувачу можливість створити та увійти в обліковий запис; прив’язати облікові записи інших сервісів, таких як Codeforces (це необхідно для відслідковування результатів розв’язування задач). Веб-сайт призначений для взаємодії з сервісом не лише звичайних користувачів, але і адміністратора. Для розмежування доступу буде використано рольову модель керування доступом (role-based access control).

CoreAPI – «ядро» системи, що обробляє запити на роботу з даними.

Jobs Service – сервіс, що виконує певні періодичні або неперіодичні процедури. До періодичних процедур можна віднести синхронізацію даних з архівом задач, генерацію списку тренування, а також розсилку електронних листів користувачам. До неперіодичних процедур можна віднести: верифікацію

електронної адреси користувача. Цей сервіс, зокрема, впроваджує розроблену модифікацію методу Лейтнера у модулі підбору задач для користувача, а також метод оцінювання складності алгоритмічних задач з архівів задач.

MS SQL 1 – реляційна база даних, що зберігає інформацію про користувачів, задачі, категорії задач тощо.

MS SQL 2 – реляційна база даних, що використовується для координації роботи Jobs Service.

Розроблена архітектура забезпечує наступні функціональні характеристики:

- Можливість розширити набір користувацьких інтерфейсів за рахунок імплементації нового сервісу. На поточний момент доступний лише веб-інтерфейс, але для зручності користувачів можуть бути розроблені боти для месенджерів, розширення для браузерів, мобільні додатки тощо.
- Можливість хостити компоненти за допомогою Docker та Kubernetes.
- Можливість масштабувати більш високонавантажені компоненти системи незалежно від інших.
- За рахунок використання бази даних MS SQL Server для синхронізації, стан кожного job-процесу може бути записаний і відновлений. Це важливо у випадку технічних неполадок, таких як позапланове вимкнення живлення на сервері.

Наведені функціональні характеристики сприятимуть подальшій розробці та підтримці системи.

3.2 Варіантний аналіз і обґрунтування вибору програмних засобів

Перед початком розробки програмного коду необхідно дослідити доступні програмні засоби та відібрати такі, що найкраще відповідають розроблюваній системі. Для розробки складових WebSite, CoreAPI та Job Service буде проведено аналіз мов програмування, а саме: C++, C# та Java.

C++ – компільована мова програмування високого рівня, розроблена Б’ярне Страуструпом. Перша версія мови була випущена у 1979 році.

Найактуальніша версія мови C++ 17 вийшла у 2017 році. C++ успадкувала синтаксис і основні аспекти мови C. По суті, C++ є надмножиною мови C, на що також вказує початкова назва – «C with classes» – «Сі з класами». Мова C++ підтримує парадигму об'єктноорієнтованого програмування, характеризується високою швидкодією за рахунок того, що код програми компілюється у машинний код, а також має велику кількість сторонніх бібліотек, що спрощують процес розробки. До недоліків мови відноситься відсутність автоматичного керування пам'яттю («збір сміття» – «garbage collection»), що є потенційно небезпечним, адже допускає існування помилок, що призводять до втрат пам'яті (явище, коли частина пам'яті, виділеної для виконання програми, не вивільняється після завершення її використання). Натомість, ручне керування пам'яттю надає розробнику більше контролю над роботою програми. Мова C++ не є кросплатформною. Програма, написана мовою C++ та скомпільована для операційної системи Windows, не працюватиме під керуванням операційної системи Linux – для цього її потрібно скомпілювати заново з допомогою спеціалізованого компілятора [20].

C# – мова програмування високого рівня, розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде в корпорації Microsoft. Перша версія мови була випущена у 2002 році. Найактуальніша версія мови C# 7.0 вийшла 7 березня 2017 року. Мова програмування C# була розроблена під впливом C++ та Java, має C-подібний синтаксис. Підтримує парадигму об'єктно-обієнтованого програмування. Є кросплатформною, оскільки код програми компілюється в IL (Intermediate Language), інша назва CIL (Common Intermediate Language) або MSIL (Microsoft Intermediate Language), що інтерпретується віртуальною машиною CLR (Common Language Runtime). Має дещо нижчу швидкодію, ніж мова C++, проте вищу, ніж Java. Характеризується наявністю великої кількості сторонніх бібліотек, автоматичного керування пам'яттю та вбудованою реалізацією деяких шаблонів проектування – наприклад, шаблон Спостерігач (Observer) реалізується мовною конструкцією event [21].

Java – мова програмування високого рівня, випущена компанією «Sun Microsystems» у 1995 році. Найактуальніша версія мови Java Standard Edition 10 вийшла 20 березня 2018 року. Має С-подібний синтаксис. Підтримує парадигму об’єктно-орієнтованого програмування. Є кросплатформною, оскільки код програми компілюється в байт-код, що при виконанні інтерпретується віртуальною машиною для конкретної платформи. Має дещо гіршу швидкодію у порівнянні з мовами C++ та C#. Характеризується наявність великої кількості сторонніх бібліотек. Java використовує автоматичний збирач сміття для керування пам'яттю під час життєвого циклу об'єкта. Програміст вирішує, коли створювати об'єкти, а віртуальна машина відповідальна за звільнення пам'яті після того, як об'єкт стає непотрібним. Коли до певного об'єкта вже не залишається посилань, збирач сміття може автоматично прибирати його із пам'яті [22].

Результати порівняльного аналізу наведено у таблиці 3.1.

Таблиця 3.1

Порівняльний аналіз мов програмування

Критерій	C++	C#	Java
Підтримка об’єктно-орієнтованого програмування	+ (1.0)	+ (1.0)	+ (1.0)
Автоматичне керування пам’яттю	- (0.0)	+ (0.5)	+ (0.5)
Кросплатформність	- (0.0)	+ (0.4)	+ (0.4)
Висока швидкодія	+ (0.5)	+/- (0.4)	+/- (0.2)
Наявність та доступність сторонніх бібліотек	+ (0.7)	+ (0.7)	+ (0.7)
Сума	2.2	3.0	2.8

У результаті проведеного аналізу для розробки було обрано мову програмування C#. Веб-розробка мовою програмування C# здійснюється за допомогою фреймворка ASP.NET. Найактуальнішою версією фреймворку ASP.NET є ASP.NET Core 3.0, випущена у 2019 році. Основною перевагою перед ASP.NET MVC є кросплатформність, що надає гнучкості при виборі хостингу. Саме тому для розробки веб-додатку була використана ASP.NET Core 3.0. Для розробки клієнтської частини додатку було обрано фреймворки jQuery та Bootstrap, оскільки використання готових стилів та функцій дозволить прискорити процес розробки. Окрім цього, для розробки користувацького інтерфейсу буде використано рушій представлень Razor, що дасть змогу спростити вихідний код шляхом розробки допоміжних функцій для формування елементів інтерфейсу.

Хостинг веб-сервісів, розроблених за допомогою фреймворку ASP.NET Core 3.0, може відбуватися на комп'ютері під керуванням як операційної системи Windows, так і операційної системи Linux. Для хостингу веб-сервісу під керуванням операційної системи Linux використовується веб-сервер Apache. Для хостингу веб-сервісу під керуванням операційної системи Windows використовується веб-сервер Internet Information Services (IIS). Для хостингу розроблюваного веб-сервісу було обрано саме операційну систему Windows, оскільки подібний підхід спрощує процес відлагодження розроблюваного проекту на етапі розробки. У якості системи керування базами даних було обрано Microsoft SQL Server 2012, оскільки Microsoft SQL Server вирізняється наявністю великої кількості допоміжних інструментів, що спрощують процес розробки та підтримки системи. Вибір версії 2012 року обумовлений вищою надійністю «старіших» версій продукту. Використання іншої системи керування базами даних – MySQL, що характеризується високою продуктивністю та кросплатформністю, є неможливим у зв'язку з інтеграційними проблемами, що стосуються переходу на .NET Core 3.0.

Для розробки Job Service буде використано фреймворк Hangfire, що надає простий API для створення та виконання завдань [23].

3.3 Вибір середовища розробки

Для ефективної розробки програмного забезпечення застосовуються інтегровані середовища розробки (IDE – Integrated Development Environment). Інтегроване середовище розробки – це комплексне програмне рішення для розробки програмного забезпечення, що складається з редактора вихідного коду, інструментів для автоматизації компіляції та відлагодження програм. Більшість сучасних середовищ розробки мають вбудовану функцію автодоповнення коду. З метою вибору середовища розробки для порівняння було обрано середовища Visual Studio 2019, Project Rider та Visual Studio Code [24].

Microsoft Visual Studio – серія продуктів корпорації Microsoft, що підтримують розробку для .NET, а також розробку на C++, Python, Node.js, JavaScript/TypeScript (рис. 3.2). Microsoft Visual Studio надає засоби для розробки консольних додатків, програм з графічним інтерфейсом, зокрема з підтримкою технологій Windows Forms та WPF, а також веб-сайти, веб-додатки тощо.

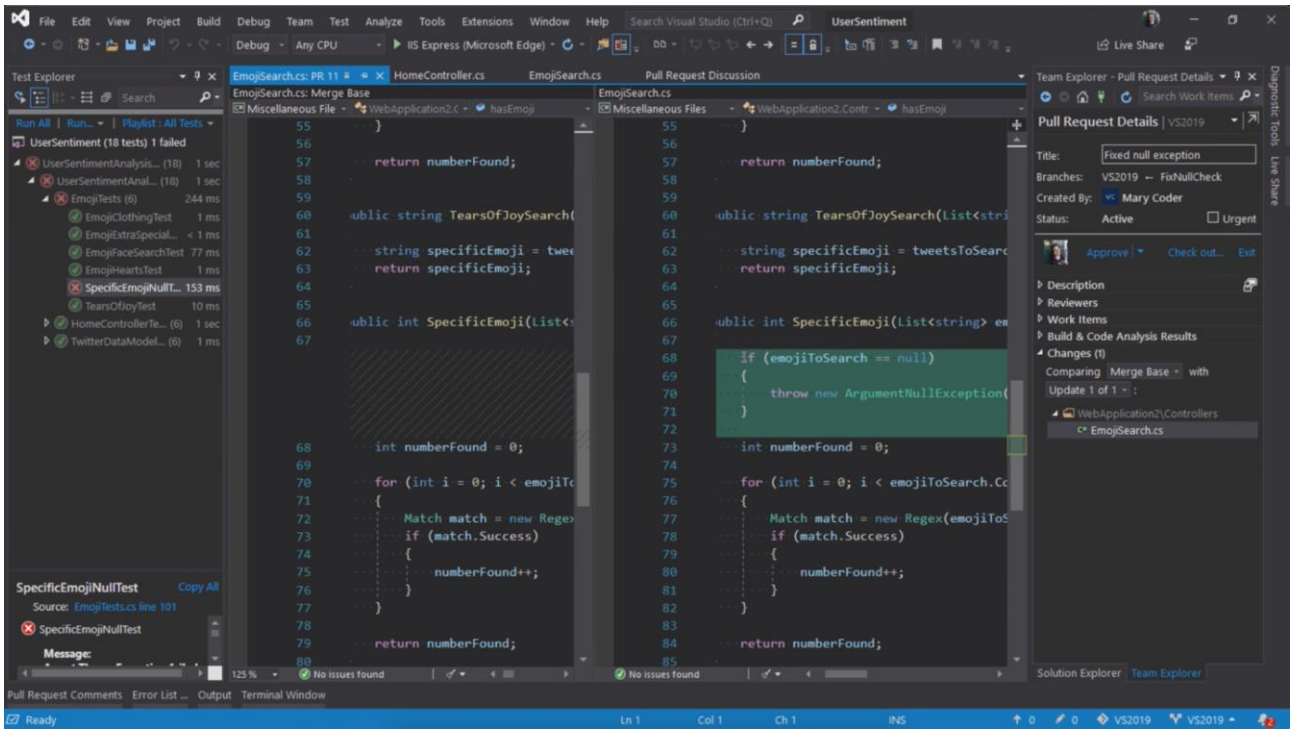

Рисунок 3.2 – Вікно Visual Studio 2019

Project Rider – продукт компанії JetBrains для розробки для .NET. Як і інші середовища розробки від компанії JetBrains, характеризується потужним механізмом автодоповнення та статичного аналізу коду (рис. 3.3).

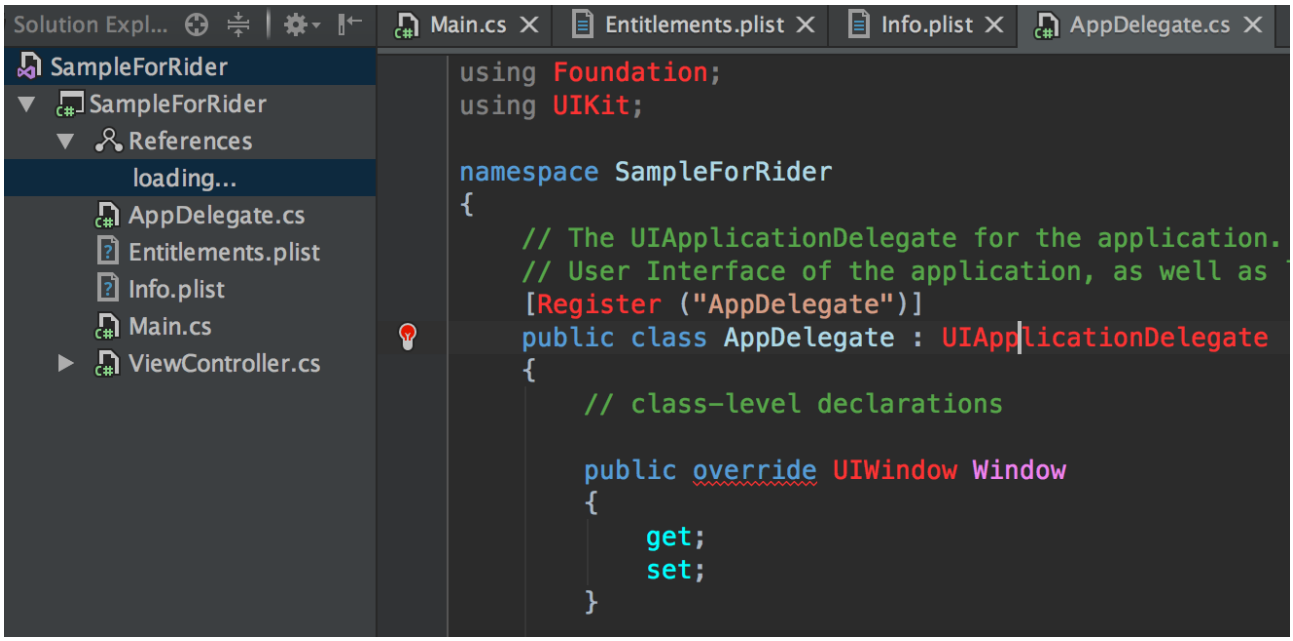

Рисунок 3.3 – Вікно Project Rider

Visual Studio Code – засіб для створення, редагування та зневадження сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

Результати проведеного порівняльного аналізу наведено у таблиці 3.2.

Таблиця 3.2

Порівняльний аналіз IDE

Критерій	Visual Studio 2019	Project Rider	Visual Studio Code
1	2	3	4
Вбудована підтримка модульного тестування	+ (0.5)	+ (0.5)	+ (0.5)

Продовження таблиці 3.2

1	2	3	4
Вбудована підтримка контролю версій	+ (0.6)	+ (0.6)	+ (0.6)
Вбудована підтримка nuget	+ (0.5)	+ (0.5)	+/- (0.2)
Автодоповнення коду	+ (0.9)	+ (0.9)	+ (0.9)
Можливість додання розширень	+ (0.5)	+ (0.5)	+ (0.5)
Наявність безкоштовної версії	+ (0.9)	- (0.0)	+ (0.9)
Сума	3.9	3.0	3.6

У результаті аналізу було прийнято рішення про використання інтегрованого середовища розробки Microsoft Visual Studio 2019, оскільки воно вирізняється наявністю безкоштовної версії – Microsoft Visual Studio 2019 Community, а також має зручний інтерфейс для роботи з nuget.

Для роботи з Microsoft SQL Server 2012 використовуватиметься SQL Server Management Studio 11, оскільки ця утиліта є офіційним засобом для конфігурування, керування та адміністрування компонентів Microsoft SQL Server.

3.4 Розробка бази даних

Першим етапом проектування бази даних є розробка універсального відношення, що полягає у виборі інформаційних об’єктів та описі їх найбільш важливих та суттєвих характеристик. У одне універсальне відношення включаються всі важливі атрибути і воно може містити всі дані, які будуть розміщені в базі даних в майбутньому. Атрибут – це характеристика певного інформаційного об’єкта бази даних, властивість сутності.

При використанні універсального відношення виникає декілька проблем:

- Надмірність даних.
- Аномалія оновлення.
- Аномалія включення.
- Аномалія видалення.

Проблема надмірності виникає, коли дані повторюються. При необхідності внесення змін, які стосуються значення атрибута одного об'єкта, потрібно відшукати й змінити всі дані, що дублюються у різних кортежах та відображають змінюване значення атрибута. Універсальне відношення зберігає в одному кортежі дані про декілька об'єктів, тому при доданні у базу даних про новий об'єкт, яке виконується шляхом вставки нового кортежу, дані про інші об'єкти можуть бути ще невідомі, а це спричинить за собою наявність порожніх полів у новому кортежі [25]. Зворотна проблема до включення є аномалія видалення. Вона полягає у тому, що інформація, яка зберігалась тільки в кортежі, який видаляється, може бути втрачена. В універсальне відношення необхідно включити атрибути, які представлені у таблиці 3.3.

Таблиця 3.3

Перелік атрибутів предметної області

№	Назва атрибута	Ім’я поля	Коментар
1	2	3	4
1	ID користувача	UserId	Ідентифікаційний номер користувача у системі
2	Логін користувача	Login	Логін, з яким користувач авторизується у системі
3	Email користувача	Email	Адреса електронної пошти користувача
4	Хеш пароля користувача	PasswordHash	Хеш пароля користувача, що використовується для перевірки коректності пароля
5	Сіль пароля користувача	PasswordSalt	Рядок, що використовується при обчисленні хешу пароля

Продовження таблиці 3.3

1	2	3	4
6	Індикатор підписки на розсилку	SubscribedToEmails	Вказує на те, що згоден користувач на одержання електронних листів
7	ID ролі	RoleId	Ідентифікаційний номер ролі
8	Ім’я ролі	RoleName	Ім’я ролі
9	ID архіву	ArchiveId	Ідентифікаційний номер архіву задач
10	Ім’я архіву	ArchiveName	Ім’я архіву задач – наприклад Codeforces, TImus, E-Olymp
11	ID облікового запису	AccountId	Ідентифікаційний номер облікового запису
12	Дескриптор облікового запису	AccountDescriptor	Дексриптор облікового запису в системі архіву задач – наприклад, логін користувача на сайті Codeforces
13	ID задачі	TaskId	Ідентифікаційний номер задачі
14	Назва задачі	TaskTitle	Назва задачі в архіві
15	Гіперпосилання на задачу	TaskUrl	Гіперпосилання на задачу в архіві
16	Складність задачі	TaskComplexity	Складність задачі
17	ID спроби	SolutionId	Ідентифікаційний номер спроби розв’язку задачі
18	Результат спроби	SolutionResult	Результат спроби розв’язку задачі
19	Дата спроби	SolutionDate	Дата спроби розв’язку задачі
20	ID тегу	TagId	Ідентифікаційний номер тегу
21	Назва тегу	TagTitle	Назва тегу в архіві задач
22	ID теми	TopicId	Ідентифікатор теми в системі
23	Назва теми	TopicTitle	Назва теми
24	ID навички	SkillId	Ідентифікаційний номер навички користувача
25	Рівень навички	SkillLevel	Рівень володіння навичкою користувачем

Продовження таблиці 3.3

1	2	3	4
26	Пріоритет вивчення навички	Priority	Пріоритет вивчення навички користувачем
27	Дата останнього проходження тестування	LastPracticeDate	Дата останнього проходження тестування

Відповідно до таблиці 3.3 потужність універсальної множини – 27.

Наступним етапом розробки бази даних є розробка ER-моделі предметної області.

ER-модель – мета-модель даних, що дозволяє описувати концептуальні схеми предметної області. За допомогою такої моделі виділяють найсуттєвіші елементи (вузли, блоки) моделі і встановлюють зв'язки між ними [26].

На основі інформаційних об’єктів бази даних в ER-моделі можна виділити такі сутності та ключі:

- Користувач (<ID користувача>).
- Роль (<ID ролі>).
- Архів (<ID архіву>).
- Обліковий запис (<ID облікового запису>).
- Задача (<ID задачі>).
- Спроба (<ID спроби>).
- Тег (<ID тегу>).
- Тема (<ID теми>).
- Навичка (<ID навички>).

В базу даних необхідно включити атрибути, що описують наступні інформаційні об’єкти: користувач, роль, архів, обліковий запис, задача, спроба, тег, тема, навичка. Кожен з цих інформаційних об’єктів має такі характеристики:

- Користувач (<ID користувача>, Логін користувача, Email користувача, Хеш пароля користувача, Сіль пароля користувача, Індикатор підписки на розсилку).
- Роль (<ID ролі>, Ім’я ролі).
- Архів (<ID архіву>, Ім’я архіву).
- Обліковий запис (<ID облікового запису>, Дескриптор облікового запису).
- Задача (<ID задачі>, Назва задачі, Гіперпосилання на задачу, Складність задачі).
- Спроба (<ID спроби>, Результат спроби, Дата спроби).
- Тег (<ID тегу>, Назва тегу).
- Тема (<ID теми>, Назва теми).
- Навичка (<ID навички>, Рівень навички, Пріоритет вивчення навички, Дата останнього проходження тестування).

Зв’язок сутностей «Користувач – Роль» характеризується класом належності «обов’язковий» з боку сутності «Користувач» та класом залежності «необов’язковий» з боку сутності «Роль», а також типом зв’язку N:M, оскільки користувач може мати декілька ролей, а роль може бути у декількох користувачів. У кожного користувача має бути як мінімум одна роль. Можлива ситуація, у якій жоден користувач не має певної ролі.

Зв’язок сутностей «Користувач – Обліковий запис» характеризується класом належності «необов’язковий» з боку сутності «Користувач» та класом залежності «необов’язковий» з боку сутності «Обліковий запис», а також типом зв’язку N:M, оскільки користувач може мати декілька облікових записів, а може не мати жодного. Розроблювана система не передбачає підтвердження володіння обліковим записом на сайті архіву задач для приєднання, тому можлива ситуація, у якій користувач відмітить обліковий запис, що не належить йому. Для того, щоб не блокувати роботу справжнього власника облікового запису, дозволяється відповідність облікового запису декільком користувачам. Окрім цього, обліковий запис може не мати відповідного йому користувача, оскільки публічна

інформація про облікові записи буде збиратись незалежно від факту реєстрації користувача на розроблюваному сервісі.

Зв’язок сутностей «Обліковий запис – Архів» характеризується класом належності «обов’язковий» з боку сутності «Обліковий запис» та класом залежності «необов’язковий» з боку сутності «Архів», а також типом зв’язку N:1, оскільки обліковий запис належить конкретному архіву задач, у якого може бути багато облікових записів. Можлива ситуація, у якій певному архіву задач не відповідає жоден обліковий запис – до першої синхронізації даних.

Зв’язок сутностей «Обліковий запис – Спроба» характеризується класом належності «необов’язковий» з боку сутності «Обліковий запис» та класом залежності «обов’язковий» з боку сутності «Спроба», а також типом зв’язку 1:N, оскільки для певного облікового запису може не бути спроб розв’язку задачі – у випадку новоствореного облікового запису. Але кожна спроба розв’язку виконується від імені конкретного облікового запису.

Зв’язок сутностей «Спроба – Задача» характеризується класом належності «обов’язковий» з боку сутності «Спроба» та класом залежності «необов’язковий» з боку сутності «Задача», а також типом зв’язку N:1, оскільки спроба виконується для конкретної задачі, і задачі може відповідати багато спроб розв’язку. Можлива ситуація, у якій для конкретної задачі немає спроб розв’язку.

Зв’язок сутностей «Задача – Архів» характеризується класом належності «обов’язковий» з боку сутності «Задача» та класом залежності «необов’язковий» з боку сутності «Архів», а також типом зв’язку N:1, оскільки задача завжди належить одному архіву, і у архіві може бути багато задач. Можлива ситуація, у якій в архіві немає жодної задачі – до першої синхронізації даних.

Зв’язок сутностей «Задача – Тег» характеризується класом належності «необов’язковий» з боку сутності «Задача» та класом залежності «обов’язковий» з боку сутності «Тег», а також типом зв’язку N:M, оскільки задача може не мати тегів, а може мати декілька. Тег існує поки існує хоча б одна задача з таким тегом, і може належати багатьох задачам.

Зв’язок сутностей «Тег – Архів» характеризується класом належності «обов’язковий» з боку сутності «Тег» та класом залежності «необов’язковий» з боку сутності «Архів», а також типом зв’язку N:1, оскільки тег актуальний для задач з конкретного архіву, а в архіві може бути декілька тегів.

Зв’язок сутностей «Тег – Тема» характеризується класом належності «обов’язковий» з боку сутності «Тег» та класом залежності «обов’язковий» з боку сутності «Тема», а також типом зв’язку 1:N, оскільки тег обов’язково відповідає певній темі, а тема – декільком тегам з різних архівів.

Зв’язок сутностей «Користувач – Навичка» характеризується класом належності «обов’язковий» з боку сутності «Користувач» та класом залежності «обов’язковий» з боку сутності «Навичка», а також типом зв’язку 1:N, оскільки у користувача обов’язково є декілька навичок, а навичка обов’язково відповідає певному користувачу.

Зв’язок сутностей «Навичка – Тема» характеризується класом належності «обов’язковий» з боку сутності «Навичка» та класом залежності «необов’язковий» з боку сутності «Тема», а також типом зв’язку N:1, оскільки навичка відповідає конкретній темі, а у теми може бути декілька навичок – відповідно до кількості зареєстрованих користувачів. До реєстрації першого користувача в системі немає жодної сутності навички.

Модель ER-діаграм екземплярів сутностей наведено у таблиці 3.4.

Таблиця 3.4

Характеристика зв’язків між сутностями предметної області

Назва сутності 1	Назва сутності 2	Тип зв’язку	Назва зв’язку	Клас належності
1	2	3	4	5
Користувач	Роль	N:M	Володіє 1	обов., необ.
Користувач	Обліковий запис	N:M	Володіє 2	необ., необ.
Обліковий запис	Архів	N:1	Належить 1	обов., необ.
Обліковий запис	Спроба	1:N	Володіє 3	необ., обов.

Продовження таблиці 3.4

1	2	3	4	5
Спроба	Задача	N:1	Відповідає 1	обов., необ.
Задача	Архів	N:1	Належить 2	обов., необ.
Задача	Тег	N:M	Відповідає 2	необ., обов.
Тег	Архів	N:1	Належить 3	обов., необ.
Тег	Тема	1:N	Відповідає 3	обов., обов.
Користувач	Навичка	1:N	Володіє 4	обов., обов.
Навичка	Тема	N:1	Належить 4	обов., необ.

Для позначення зв’язку кожному з них присвоюється унікальна назва, що зазвичай є дієсловом. У випадку повторного використання одного і того ж дієслова, до назви додається числове позначення – ціле число, починаючи з одиниці.

На основі даних таблиці 3.4 можна побудувати результуючу ER-діаграму типів предметної галузі, наведену на рисунку 3.4.

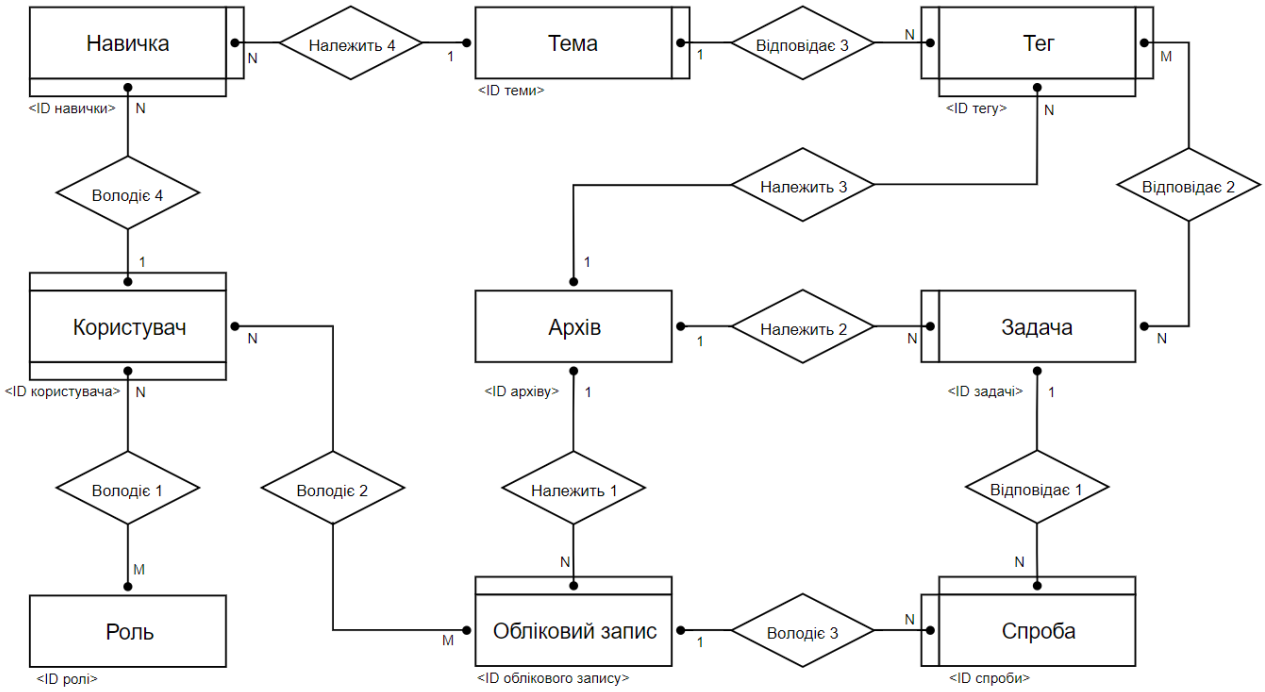

Рисунок 3.4 – ER-діаграма предметної області

Наступний етапом розробки бази даних є нормалізація відношень.

Нормалізація – розбиття однієї таблиці на декілька таблиць, що володіють кращими властивостями включення, зміни або видалення даних. Остаточна мета нормалізації зводиться до отримання такого проекту бази даних в якому кожен факт з'являється лише в одному місці, тобто виключена надмірність інформації.

Нормалізація проводиться за алгоритмом та визначеннями нормальних форм. Універсальне відношення R розбивається на такі відношення:

- R1 (<ID користувача>, Логін користувача, Email користувача, Хеш пароля користувача, Сіль пароля користувача, Індикатор підписки на розсилку);
- R2 (<ID ролі >, Ім’я ролі);
- R3 (<ID архіву>, Ім’я архіву);
- R5 (<ID облікового запису >, Дескриптор облікового запису);
- R12 (<ID задачі>, Назва задачі, Гіперпосилання на задачу, Складність задачі);
- R10 (<ID спроби >, Результат спроби, Дата спроби);
- R16 (<ID тегу >, Назва тегу);
- R21 (<ID теми >, Назва теми);
- R23 (<ID навички >, Рівень навички, Пріоритет вивчення навички, Дата останнього проходження тестування).

Кортежі відношень не повторюються, атрибути атомарні, всі ключі є простими, відсутні транзитивні залежності, присутній один ключ у кожному відношенні. Тому можна зробити висновок, що всі відношення знаходяться у нормальній формі Бойса-Кодда.

Наступним етапом розробки бази даних є одержання попередніх відношень методом «суть-зв’язок».

Модель «суть-зв'язок» визначає значення даних в контексті їх взаємозв'язку з іншими даними. Існує 8 правил, якими користуються для одержання набору відношень, вільних від проблем вставки, видалення і модифікації (тобто без порожніх полів і надлишково дубльованих даних, а також

таких, що не зберігають в одному записі дані про декілька об'єктів) на основі розроблених інформаційних моделей. Для кожного випадку використовується певне правило, згідно з яким генеруються попередні відношення.

Правило 4: якщо степінь бінарного зв’язку 1:N, і клас належності N-зв’язної суті є обов’язковим, то досить використати по одному відношенню на кожну суть, при умові, що ключ кожної суті служить у якості первинного ключа для відповідного відношення. Додатково, ключ однозв’язної суті повинен бути доданий, як атрибут у відношенні, що відводиться N-зв’язній суті.

Правило 6: якщо ступінь бінарного зв'язку M:N, то для зберігання даних потрібні три відношення: по одному для кожної суті, причому ключ кожної суті служить як первинний ключ для відповідного відношення, та одного відношення для зв'язку. Зв'язок повинен мати серед своїх атрибутів і ключ суті кожної зі зв'язних сутей.

На основі цих правил, використовуючи множину атрибутів з таблиці 3.3, ER-діаграму типів з рисунку 3.4, можна представити попередні відношення, що подані у таблиці 3.5.

Таблиця 3.5

Попередні відношення

Ім’я зв’язку	Правило	Попередні відношення	Додаткові атрибути
1	2	3	4
Володіє 1	6	R1(ID користувача, …)	Логін користувача, Email користувача, Хеш пароля користувача, Сіль пароля користувача, Індикатор підписки на розсилку
		R2(ID ролі, …)	Ім’я ролі
		R3(ID користувача, ID ролі)	(для організації зв’язку)

Продовження таблиці 3.5

1	2	3	4
Володіє 2	6	*R4(ID користувача, …)	Логін користувача, Email користувача, Хеш пароля користувача, Сіль пароля користувача, Індикатор підписки на розсилку
		*R5(ID облікового запису, …)	Дескриптор облікового запису
		R6(ID користувача, ID облікового запису)	(для організації зв’язку)
Належить 1	4	R7(ID облікового запису, …)	Дескриптор облікового запису, ID архіву
		R8(ID архіву, …)	Ім’я архіву
Володіє 3	4	*R9(ID облікового запису, …)	Дескриптор облікового запису, ID архіву
		*R10(ID спроби, …)	Результат спроби, Дата спроби, ID облікового запису
Відповідає 1	4	R11(ID спроби, …)	Результат спроби, Дата спроби, ID облікового запису, ID задачі
		*R12(ID задачі, …)	Назва задачі, Гіперпосилання на задачу, Складність задачі
Належить 2	4	R13(ID задачі, …)	Назва задачі, Гіперпосилання на задачу, Складність задачі, ID архіву
		*R14(ID архіву, …)	Ім’я архіву
Відповідає 2	6	*R15(ID задачі, …)	Назва задачі, Гіперпосилання на задачу, Складність задачі, ID архіву
		*R16(ID тегу, …)	Назва тегу
		R17(ID задачі, ID тегу)	(для організації зв’язку)
Належить 3	4	*R18(ID тегу, …)	Назва тегу, ID архіву
		*R19(ID архіву, …)	Ім’я архіву

Продовження таблиці 3.5

1	2	3	4
Відповідає 3	4	R20(ID тегу, …)	Назва тегу, ID архіву, ID теми
		R21(ID теми, …)	Назва теми
Володіє 4	4	*R22(ID користувача, …)	Логін користувача, Email користувача, Хеш пароля користувача, Сіль пароля користувача, Індикатор підписки на розсилку
		*R23(ID навички, …)	Рівень навички, Пріоритет вивчення навички, Дата останнього проходження тестування, ID користувача
Належить 4	4	R24(ID навички, …)	Рівень навички, Пріоритет вивчення навички, Дата останнього проходження тестування, ID користувача, ID теми
		*R25(ID теми, …)	Назва теми

У таблиці 3.5 відношення R4, R5, R9, R10, R12, R14, R15, R16, R18, R19, R22, R23, R25 відмічені зірочкою, що означає, що ці відношення включаються у інші, а саме:

- Відношення R4, R22 включаються у відношення R1.
- Відношення R5, R9 включаються у відношення R7.
- Відношення R14, R19 включаються у відношення R8.
- Відношення R10 включається у відношення R11.
- Відношення R12, R15 включаються у відношення R13.
- Відношення R16, R18 включаються у відношення R20.
- Відношення R25 включається у відношення R21.
- Відношення R23 включається у відношення R24.

Відношення, що включаються в інші, можуть бути вилучені. Після видалення вищевказаних відношень, отримано таку підмножину відношень, які описують предметну область:

- R1 (<ID користувача>, Логін користувача, Email користувача, Хеш пароля користувача, Сіль пароля користувача, Індикатор підписки на розсилку).
- R2 (<ID ролі>, Ім’я ролі).
- R3 (<ID користувача, ID ролі>) – для організації зв’язку.
- R6 (<ID користувача, ID облікового запису>) – для організації зв’язку.
- R7 (<ID облікового запису>, Дескриптор облікового запису, ID архіву).
- R8 (<ID архіву>, Ім’я архіву).
- R11 (<ID спроби>, Результат спроби, Дата спроби, ID облікового запису, ID задачі).
- R13 (<ID задачі>, Назва задачі, Гіперпосилання на задачу, Складність задачі, ID архіву).
- R17 (<ID задачі, ID тегу>) – для організації зв’язку.
- R20 (<ID тегу>, Назва тегу, ID архіву, ID теми).
- R21 (<ID теми>, Назва теми).
- R24 (<ID навички>, Рівень навички, Пріоритет вивчення навички, Дата останнього проходження тестування, ID користувача, ID теми).

Кінцевим етапом проектування бази даних є розробка схеми бази даних, що описує структуру і взаємозв’язки у реляційній базі даних. Схема бази даних наведена в додатку В.

3.5 Розробка програмних модулів системи

Програмну систему розділено на декілька проектів, кожен з яких компілюється в DLL-файл (Dynamic Link Library). Діаграму залежностей між проектами наведено на рисунку 3.5.

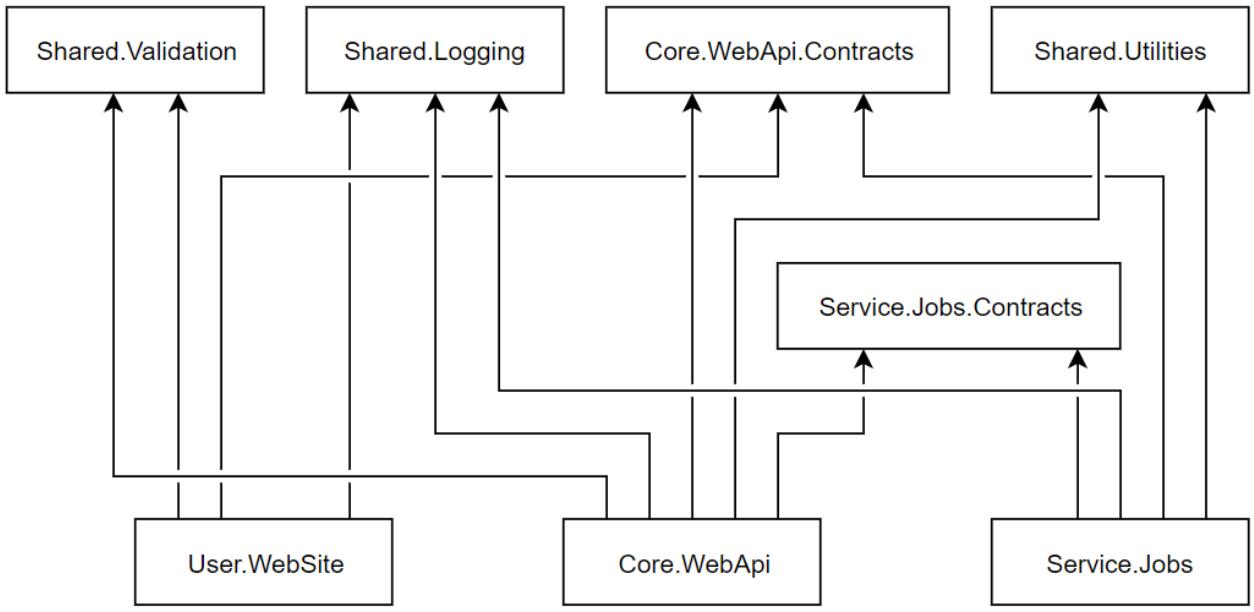

Рисунок 3.5 – Діаграма залежностей між проектами

Поділ на проекти виконано відповідно до логічного призначення кожної фрагменту коду:

- Shared.Validation – проект, що містить компоненти, що виконують перевірку правильності даних. Наприклад, перевірка логіна та пароля користувача на відповідність правилам виконується як на веб-сайті, так і на рівні внутрішнього API.

- Shared.Logging – проект, що містить код, що забезпечує інфраструктуру логування. Незважаючи на наявність потужних бібліотек для логування, має сенс використання інтерфейсу ILogger, абстрагованого від конкретної реалізації, з метою забезпечення можливості подальшого переходу на альтернативну бібліотеку.

- Shared.Utilities – проект, що містить набір утиліт, що спрощують роботу з даними і водночас відділені від бізнес-логіки, наприклад CryptographyUtilities.

- Core.WebApi.Contracts – проект, що містить опис API, який надається сервісом CoreAPI – а саме, доступні сервіси, а також моделі даних, з якими працюють ці сервіси. Для цього використовується технологія gRPC for .NET Core.

- Service.Jobs.Contracts – проект, що містить строго типізований опис API, який надається сервісом Jobs Service.
- User.WebSite – веб-додаток, що забезпечує користувацький доступ до системи.
- Core.WebApi – проект, що керує даними. Він надає API для створення, оновлення, читання та видалення даних, та виконує додаткові пов’язані з цим дії – наприклад, запуск процедури підбору задач для користувача після реєстрації.
- Service.Jobs – проект, що містить реалізацію процедур сервісу Jobs Service.

Такий поділ системи забезпечує низьку зв’язність компонентів, що спрощує процес розробки та підтримки.

Для роботи модуля підбору задач потрібні вихідні дані – а саме, інформація про задачі, облікові записи користувачів, а також спроби розв’язку задач. Для отримання цієї інформації розроблено окремі програмні компоненти, що виконують синхронізації бази даних з сторонніми сервісами, такими як Codeforces, Timus Online Judge та E-Olymp. При розробці потрібно враховувати відмінності між API, що надаються різними сервісами. Наприклад, Codeforces надає зручний API для доступу до даних, і повертає результат у форматі JSON. У випадку Timus Online Judge необхідно виконувати парсинг HTML сторінки для отримання даних.

Оскільки операція синхронізації доволі затратна з точки зору системних ресурсів, було прийнято рішення запускати її раз на добу – о 02:00 за всесвітнім координованим часом (UTC). Такий час обрано з тієї причини, що вищенаведені сервіси переважно використовуються користувачами з Європи та Азії. Низька активність у період синхронізації дозволить зменшити навантаження на сервери серверів, а також знизити ризик нестабільної роботи процесу синхронізації.

Процес синхронізації поділяється на три складові:

- Синхронізація архіву задач.
- Синхронізація облікових записів.
- Синхронізація спроб розв’язування задач.

Для кожного процесу створюється окремий клас, що реалізує логіку синхронізації даних. Ієрархію класів наведено на рисунку 3.6.

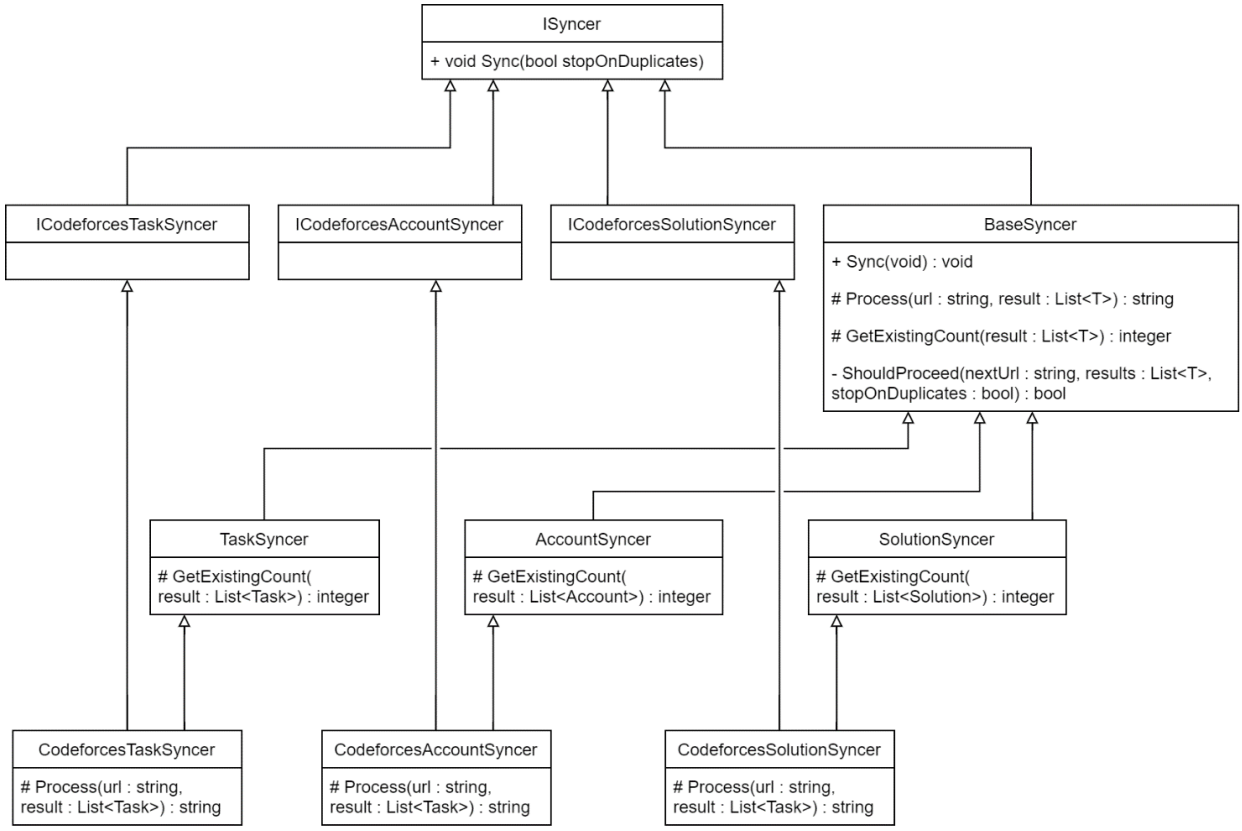

Рисунок 3.6 – Ієрархія класів модуля синхронізації даних

Оскільки повна ієрархія класів містить багато елементів, на рисунку наведено лише ту її частину, що стосується синхронізації з сервісом Codeforces.

Процес синхронізації запускається раз на добу з параметром stopOnDuplicates=true, що вказує на те, що у разі, якщо усі елементи, знайдені на останній обробленій сторінці, вже наявні у базі даних, необхідно припинити синхронізацію. Це дозволяє уникнути надлишкової роботи з обробки вже наявних даних. Тим не менш, можлива ситуація, спричинена, наприклад, технічним збоєм, у якій частина даних буде пропущена. Для того, щоб синхронізувати такі дані, щомісяця відбуватиметься запуск процесу синхронізації з параметром stopOnDuplicates=false.

Ключовим елементом розроблюваної системи є модуль підбору задач. Існує два можливі тригери процесу підбору задач для користувача:

- Особистий запит користувача. За допомогою веб-інтерфейсу користувач може виконати запит на отримання набору задач для розв’язування.

- Періодичне формування набору задач для користувача, результати якого згодом надсилаються на електронну пошту користувача у разі, якщо він погодився на отримання листів електронною поштою.

Підбір задач для користувача виконується за наступним алгоритмом:

1. Формування списку тем, які необхідно тренувати. Для цього з таблиці Skills обираються рядки, що стосуються поточного користувача, а від дати останнього тестування минуло достатньо часу. Кількість днів після останнього тренування визначається за таблицею 2.1.

2. Формування списку тегів із вказанням пріоритету. Для цього з таблиці Skills витягується пріоритет теми, а в відповідних рядків таблиці Tags – ідентифікатор тегу.

3. Завантажується список задач, що відповідають заданим тегам. На цьому етапі завантажуються усі задачі, які пов’язані хоча б з одним тегом зі сформованого раніше списку. Для підвищення швидкодії, виконання пошуку можна виконувати паралельно для різних архівів, оскільки задача не може мати тегів, що стосуються інших архівів.

4. Список задач фільтрується, відсіюючи вже розв’язані користувачем задачі.

5. Список задач фільтрується за рівнем складності відповідно до теми. Для кожної обраної теми у порядку спадання пріоритету виконується пошук 10 задач із сформованого раніше списку задач. Після кожної ітерації обрані елементи вилучаються зі списку. При пошуку виконується перевірка, чи не має поточна задача тегу, що пов’язаний з навичкою, рівень володіння якою у користувача нижчий за рівень складності задачі.

6. Сформований список задач, згрупований за темою і відсортований у порядку спадання пріоритету, відображається перед користувачем за допомогою веб-інтерфейсу, або надсилається на відповідну поштову адресу.

Блок-схема алгоритму підбору задач наведена на рисунку 3.7.

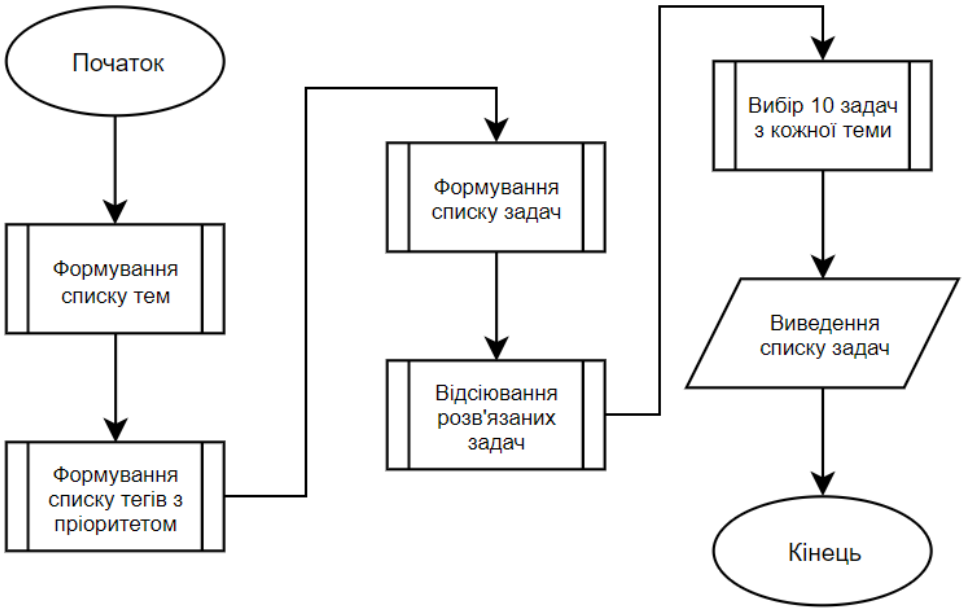

Рисунок 3.7 – Схема алгоритму підбору задач

При розробці програмного продукту було використано систему контролю версій Git з використанням хмарного репозиторію GitHub, а також утиліти для системи контролю версій з графічним інтерфейсом SourceTree.

3.6 Висновки

У третьому розділі було проведено варіантний аналіз та вибір засобів розробки та інтегрованого середовища розробки, а також розроблено архітектуру програмної системи. Було розроблено універсальне відношення, ER-модель предметної галузі, спроектовано відношення та виконано їх нормалізацію, а також проведено оцінку спроектованих відношень. У результаті виконання цих дій було одержано структуру розроблюваної бази даних. Було розроблено програмні модулі, описано структуру компонентів, що забезпечують синхронізацію даних та алгоритм підбору задач.

4 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

4.1 Вибір методики тестування програмної системи

Тестування програмного забезпечення –процес дослідження програмного продукту, що має на меті виявлення інформації про якість продукту стосовно контексту його використання. Процес тестування означає як пошук помилок або інших дефектів, так і випробування програмного забезпечення для його оцінки. Критерії оцінювання:

1. Відповідність програмного продукту технічному завданню.
2. Коректність отриманих результатів для будь-яких дозволених вхідних даних.
3. Відповідність вимогам, що стосуються споживання системних ресурсів.
4. Відповідність вимогам, що стосуються сумісності з програмним забезпеченням та операційними системами.

Зважаючи на факт того, що кількість можливих тестів навіть для примітивних програмних компонент є дуже великою, стратегія тестування полягає в тому, щоб максимізувати покриття програмного продукту тестами, витративши на тестування прийнятну кількість часу.

Існує багато видів тестування: одні зазвичай виконують самі розробники, а інші – спеціалізовані групи. Деякі види тестування перераховані нижче [27]:

1. Модульне тестування (unit testing) – тестування окремої складової програмної системи (функції, класу, сервісу тощо), що виконується окремо від інших компонентів.
2. Інтеграційне тестування – це виконання декількох компонентів або підсистем, що має на меті перевірку коректності взаємодії компонентів.
3. Регресивне тестування – перевірка, спрямована на виявлення багів у програмі, що вже пройшла цей набір тестів. Ці тести зазвичай виконуються вручну, і обов’язково покривають найважливішу частину функціональності продукту.

4. Системне тестування – виконання програмного продукту в його остаточній конфігурації. Часто цей вид тестування автоматизують за допомогою спеціальних інструментів. Підтримка автоматичних системних тестів має значну більшу ефективність, ніж модульне (блокове) тестування.

Існує два основних методи тестування: тестування «білої скриньки», тестування «чорної скриньки» [28].

Перший вид тестування має на меті з'ясування обставин, в яких поведінка програми не відповідає її специфікації. Тестові ж дані використовуються тільки у відповідності зі специфікацією програми (без урахування знань про її внутрішню структуру).

Стратегія «білої скриньки», або стратегія тестування, керованого логікою програми, дозволяє досліджувати внутрішню структуру програми. В цьому випадку QA-інженер отримує тестові дані шляхом аналізу логіки програми. При використанні даної стратегії часто не використовується специфікація програми.

Стратегія «білої скриньки» надлишкова, оскільки існує специфікація вимог до системи. Необхідність отримання інформації шляхом аналізу програмного коду виникає лише у разі відсутності специфікації та особи, відповідальної за опис вимог до програмного продукту – що трапляється у випадку необхідності підтримувати так званий «legacy code» – програмний код, що був розроблений іншим розробником або групою розробників.

Для тестування програмного продукту було обраного стратегію «чорної скриньки».

4.2 Хід тестування

Тестування програмної системи проводиться за методикою «чорної скриньки». Вона базується на використанні шаблонів тестування (тест-кейсів). Це означає, що буде створено декілька тест-кейсів для перевірки правильності роботи основних функцій програмного додатку. Розроблені для перевірки правильності роботи сайту тест-кейси описано в таблиці 4.1.

Таблиця 4.1

Тест-кейси

<table>
<tr><td rowspan="4">Код тест-кейса</td><td colspan="3">Опис тест-кейса</td></tr>
<tr><td colspan="2">Хід тестування</td><td>Очікуваний результат</td></tr>
<tr><td>Дата тестування</td><td>Результат</td><td>Примітка</td></tr>
<tr><td colspan="3">2</td></tr>
<tr><td rowspan="3">1</td><td colspan="2">3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td></tr>
<tr><td rowspan="3">001</td><td colspan="3">Тест реєстрації в системі</td></tr>
<tr><td colspan="2">1. Відкрити сайт.
2. Натиснути на пункт меню «Register».
3. В поле «Login» ввести текст «testX», де символи X замінити на номер виконання тесту.
4. В поле «Email» ввести текст «testX@example.com», де символи X замінити на номер виконання тесту.
5. В поле «Password» ввести текст «123456».
В поле «Password confirmation» ввести текст «123456».
6. Натиснути кнопку «Register».</td><td>У системі зареєстровано нового користувача.
Після реєстрації користувач залишається авторизованим, на що вказує відсутність пунктів «Login» та «Register» на навігаційному меню.</td></tr>
<tr><td>11.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td rowspan="3">002</td><td colspan="3">Тест авторизації в системі</td></tr>
<tr><td colspan="2">1. Відкрити сайт.
2. Натиснути на пункт меню «Login».
3. В поля «Login» та «Password» ввести дані користувача, створеного в тесті 001.
4. Натиснути кнопку «Login».</td><td>Користувач залишається авторизованим, на що вказує відсутність пунктів «Login» та «Register» на навігаційному меню.</td></tr>
<tr><td>11.11.2019</td><td>Пройдено</td><td>-</td></tr>
</table>

Продовження таблиці 4.1

<table>
<tr><td rowspan="3">1</td><td colspan="3">2</td></tr>
<tr><td colspan="2">3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td></tr>
<tr><td rowspan="3">003</td><td colspan="3">Тест виходу з системи</td></tr>
<tr><td colspan="2">1. Повторити кроки, описані у тесті 001.
2. Натиснути на пункт меню «Logout».
3. Натиснути на кнопку «Logout».</td><td>Користувач припиняє бути авторизованим, на що вказує наявність пунктів «Login» та «Register» на навігаційному меню.</td></tr>
<tr><td>11.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td rowspan="3">004</td><td colspan="3">Тест доступу до адміністративних ресурсів</td></tr>
<tr><td colspan="2">1. Відкрити сайт.
2. Авторизуватись на сайті з логіном «admin» та паролем «123456789».
3. Натиснути на пункт меню «System Configuration».</td><td>Сторінка «System Configuration» відображається.</td></tr>
<tr><td>12.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td rowspan="3">005</td><td colspan="3">Тест редагування адреси електронної пошти</td></tr>
<tr><td colspan="2">1. Авторизуватись на сайті.
2. Натиснути на пункт меню «Settings».
3. Змінити значення поля «Email».
4. Натиснути кнопку «Save».</td><td>Відображається нова адреса електронної пошти.</td></tr>
<tr><td>12.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td rowspan="3">006</td><td colspan="3">Тест редагування списку пов’язаних облікових записів</td></tr>
<tr><td colspan="2">1. Авторизуватись на сайті.
2. Натиснути на пункт меню «Settings».
3. В секції «Add Account» обрати архів за допомогою radio button «Archive» та вписати у поле «Descriptor» існуючий дескриптор користувача.
4. Натиснути кнопку «Save».</td><td>Після оновлення сторінки у списку пов’язаних обліковий записів відображається новий.</td></tr>
<tr><td>12.11.2019</td><td>Пройдено</td><td>-</td></tr>
</table>

Продовження таблиці 4.1

1	2		
	3	4	
	5	6	7
007	Тест синхронізації інформації про облікові записи		
	1. Відкрити SQL Server Management Studio. 2. Підключитись до бази даних. 3. Виконати запит SELECT COUNT(1) FROM Accounts	Отримано ненульове значення.	
	12.11.2019	Пройдено	-
008	Тест синхронізації інформації про задачі		
	1. Відкрити SQL Server Management Studio. 2. Підключитись до бази даних. 3. Виконати запит SELECT COUNT(1) FROM Tasks	Отримано ненульове значення.	
	13.11.2019	Пройдено	-
009	Тест синхронізації інформації про спроби розв’язку задач		
	1. Відкрити SQL Server Management Studio. 2. Підключитись до бази даних. 3. Виконати запит SELECT COUNT(1) FROM Solutions	Отримано ненульове значення.	
	13.11.2019	Пройдено	-
010	Тест редагування пріоритетності тем		
	1. Авторизуватись на сайті. 2. Натиснути на пункт меню «Settings». 3. В секції «Topics» змінити показник пріоритету однієї з тем. 4. Натиснути кнопку «Save».	Після оновлення сторінки відображаються оновлені дані.	
	14.11.2019	Пройдено	-

Продовження таблиці 4.1

<table>
<tr><td rowspan="3">1</td><td colspan="3">2</td></tr>
<tr><td colspan="2">3</td><td>4</td></tr>
<tr><td>5</td><td>6</td><td>7</td></tr>
<tr><td rowspan="3">011</td><td colspan="3">Тест коректності підбору задач</td></tr>
<tr><td colspan="2">1. Авторизуватись на сайті.
2. Натиснути на пункт меню «Analysis».
3. Зачекати, поки формується результат.</td><td>Генерація підбірки задач займає менше 20 секунд. До кожної задачі доступне гіперпосилання на задачу.</td></tr>
<tr><td>14.11.2019</td><td>Пройдено</td><td>-</td></tr>
<tr><td rowspan="3">012</td><td colspan="3">Тест коректності відображення сайту у різних браузерах</td></tr>
<tr><td colspan="2">1. Відкрити сайт за допомогою браузера Google Chrome
2. Відкрити сайт за допомогою браузера Opera
3. Відкрити сайт за допомогою браузера Edge</td><td>Елементи сайту відображаються однаково, стилі не відрізняються.</td></tr>
<tr><td>15.11.2019</td><td>Пройдено</td><td>-</td></tr>
</table>

При тестуванні програми за методикою тестування «чорної скриньки» помилок не виявлено.

4.3 Розробка інструкції користувача

Користувацький інтерфейс розроблено у мінімалістичному дизайні, що підвищує зручність користування, адже інтерфейс не перевантажено елементами керування.

Користувач починає роботу з сайтом з головної сторінки. Ця сторінка містить навігаційну панель (рис. 4.1).

Рисунок 4.1 – Навігаційна панель сайту

Для початку роботи з системою необхідно авторизуватись на сайті. Зробити це можна, перейшовши на відповідну сторінку, натиснувши на пункт «Register» на навігаційному меню.

Після переходу на сторінку реєстрації, користувач повинен заповнити наведені поля, такі як логін, email, пароль та підтвердження пароля (рис. 4.2).

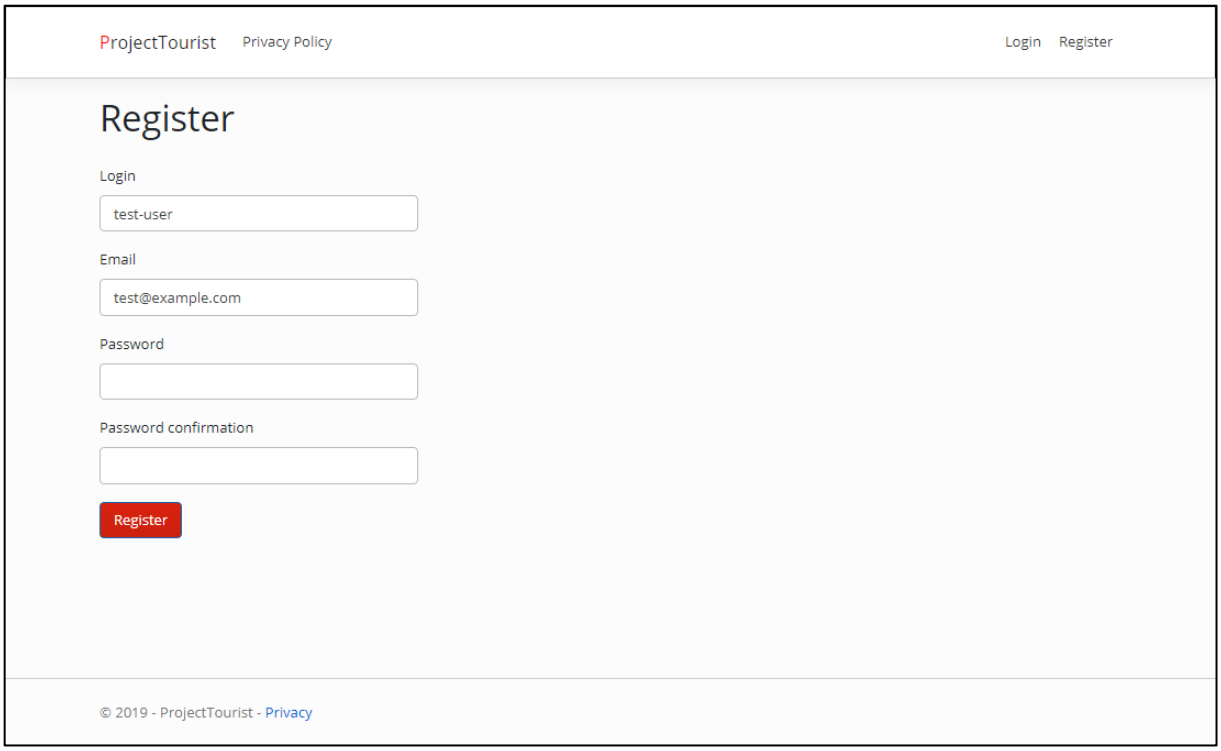

Рисунок 4.2 – Сторінка реєстрації

Існують наступні вимоги до значень полів, наведених на сторінці реєстрації:

- Поле «Логін» має бути не порожнім, мати довжину від 3 до 32 символів латинського алфавіту незалежно від капіталізації літер, а також цифри. Логін може містити не більше одного спеціального символу з наступного переліку: нижнє підкреслення, собака, крапка, плюс та мінус.
- Поле «Email» має бути не порожнім та містити правильну адресу електронної пошти.
- Поле «Пароль» має бути не порожнім та містити від 8 до 32 символів.
- Поля «Пароль» та «Підтвердження пароля» повинні співпадати.

У випадку введення неприйнятних значень, відповідне повідомлення буде відображене поруч з полем введення, як це зображено на рисунку 4.3.

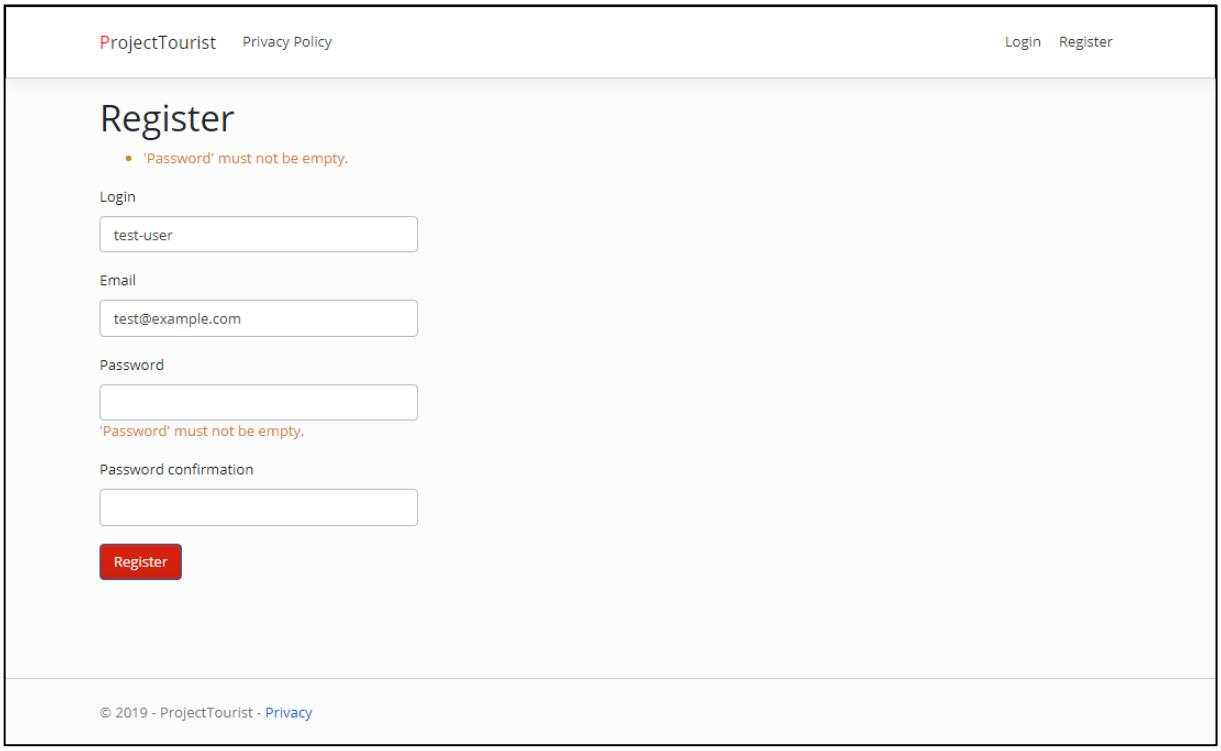

Рисунок 4.3 – Сторінка реєстрації при введенні неправильних даних

У випадку коректності введених даних користувача буде зареєстровано.

До початку використання системи користувач повинен підключити пов’язані облікові записи з архівів алгоритмічних задач. Для цього необхідно перейти на сторінку налаштувань і у секції «Облікові записи» додати відповідні облікові записи (рис. 4.4).

Оскільки розроблена програмна система використовує виключно публічну інформацію, то підключення облікового запису не потребує від користувача жодних додаткових дій, окрім вказання дескриптора облікового запису. У поточному випадку дескриптор – це певний унікальний рядок, що однозначно вказує на обліковий запис. Це може бути нікнейм, адреса електронної пошти чи автоматично згенерований числовий ідентифікатор.

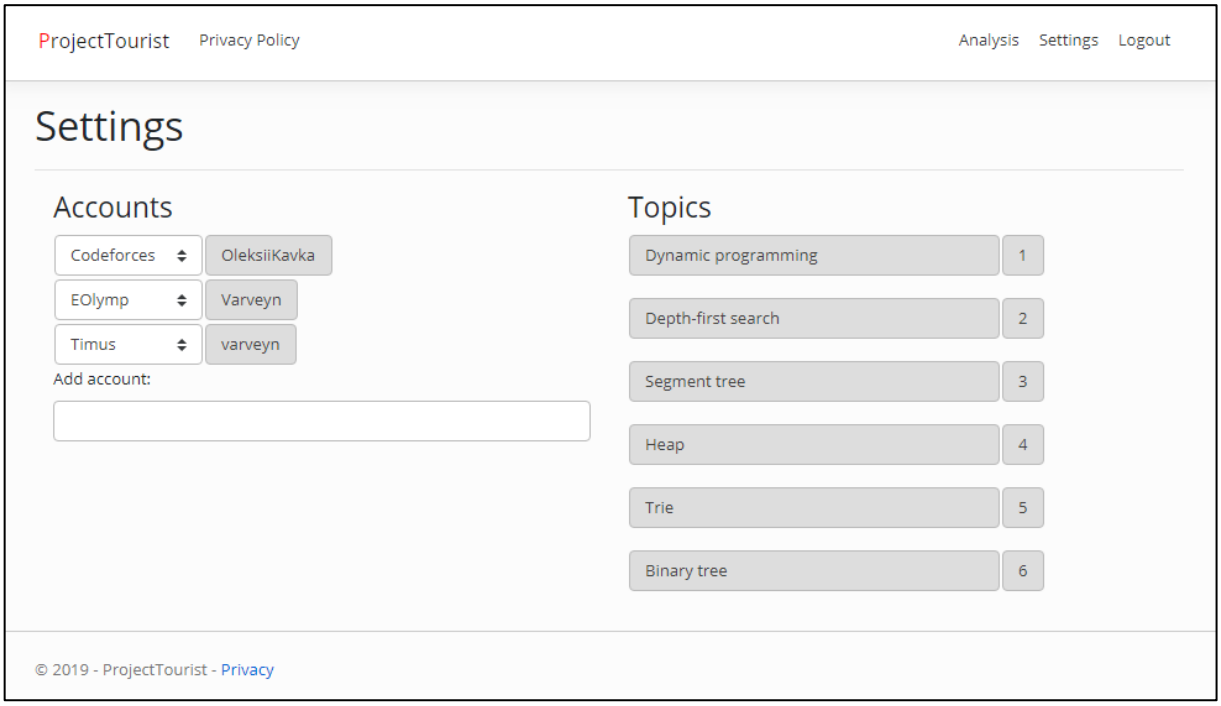

Рисунок 4.4 – Сторінка налаштувань

Після того як користувач прив’язав хоча б один обліковий запис, він може скористатись модулем підбору задач. Для цього необхідно натиснути на пункт «Аналіз» на навігаційному меню. Після нетривалого очікування буде сформовано список задач, які рекомендовано розв’язувати для розвитку навичок застосування тих чи інших методів розв’язання програмних алгоритмічних задач. Приклад підбірки задач наведено на рисунку 4.5.

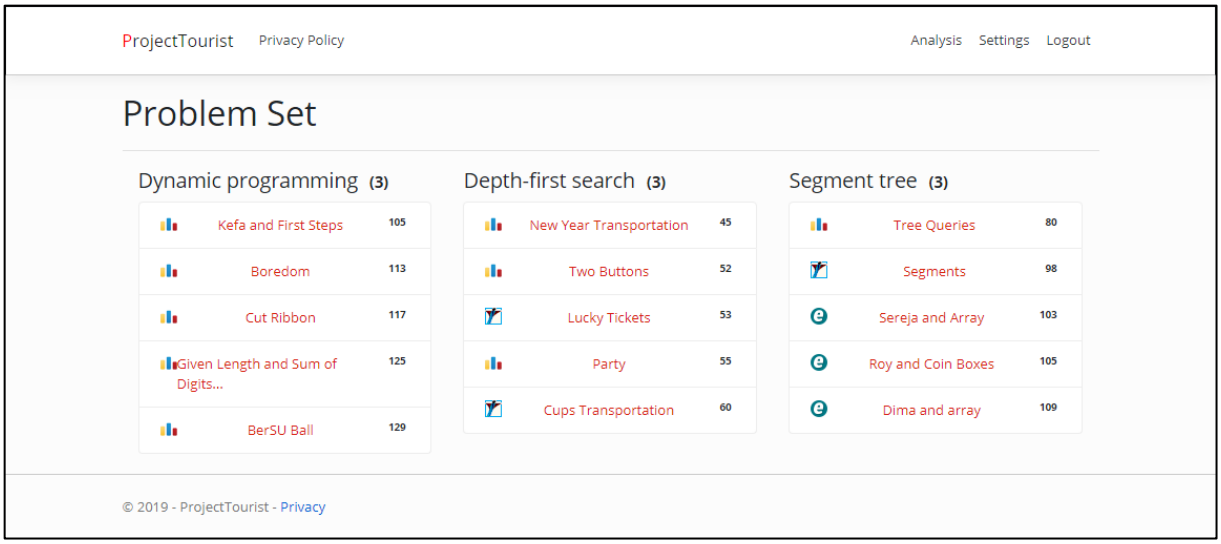

Рисунок 4.5 – Підбірка задач для тренування

На сторінці відображається список задач, що містить п’ять задач на кожну тему. Список містить гіперпосилання на задачу, а також піктограму, що вказує на те, з якого саме архіву поточна задача. Після натиснення на заголовок задачі, користувач перейде на сайт архіву задач, де зможе ознайомитись з умовою задачі та почати практикуватись.

Таким чином, розроблена інструкція користувача містить всю необхідну для використання системи інформацію.

4.4 Аналіз ефективності розробленої модифікації методу Лейтнера

Оскільки метою магістерської кваліфікаційної роботи є підвищення ефективності засвоєння студентами знань та навичок з використання алгоритмів та структур, то необхідно провести дослідження, що дозволить визначити, чи було досягнуто поставленої мети.

Перш за все, необхідно визначити критерій оцінювання – певну числову характеристику, на основі якої може бути зроблено висновок про ефективність розробленої модифікації методу Лейтнера.

Ключовим показником володіння навичкою використання алгоритму чи структури даних є здатність розв’язувати більш складні задачі, що пов’язані з відповідною темою. Таким чином, головною складовою критерію оцінювання ефективності методу є факт досягнення студентом рівня кваліфікації, на якому він здатний розв’язувати задачі певного рівня складності. Іншим важливим чинником є кількість часу, що знадобилась для досягнення відповідного рівня кваліфікації.

На основі наведених вище параметрів може бути побудований графік, що відобразить залежність рівня кваліфікації студента стосовно конкретної теми від часу. Рівень кваліфікації характеризується максимальним рівнем складності задачі, яку розв’язав користувач за заданий період часу.

На основі теоретичних положень, викладених у другому розділі, можна спрогнозувати вигляд такого графіка для випадку використання користувачем модифікації методу Лейтнера (рис. 4.6).

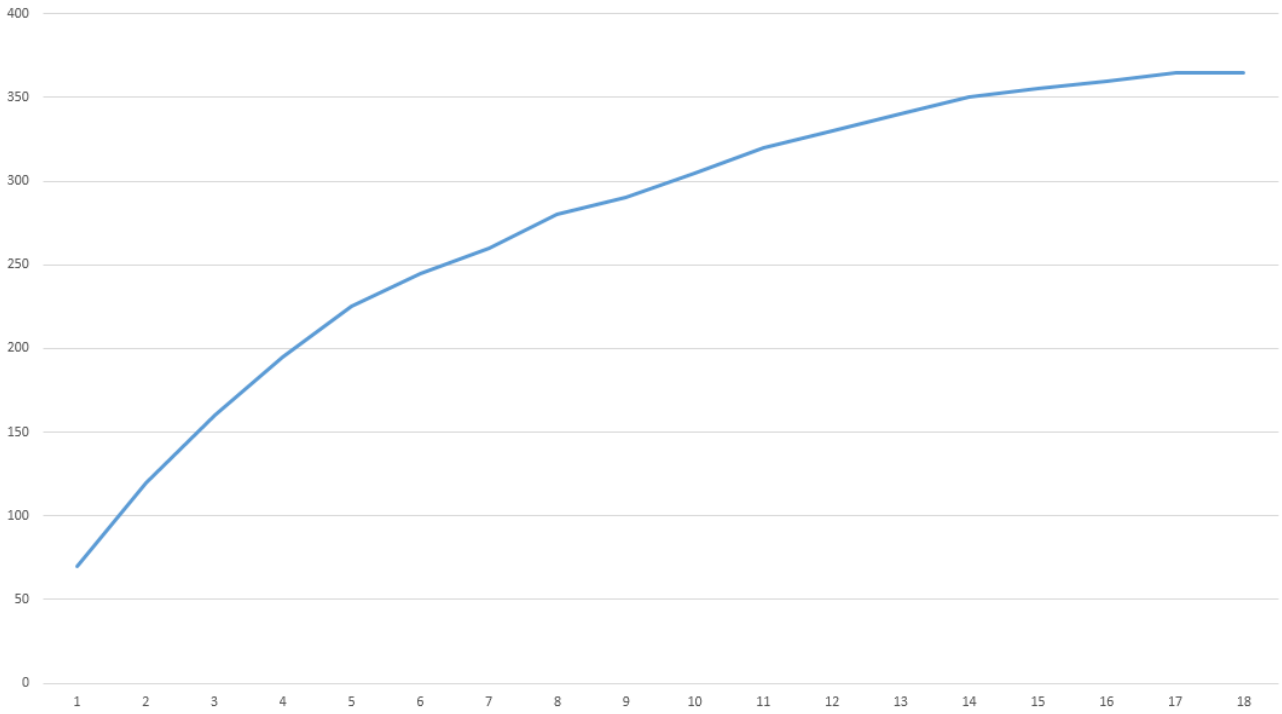

Рисунок 4.6 – Залежність кваліфікації від часу при використанні модифікації методу Лейтнера

Для виконання порівняння було використано інформацію про рішення задач користувачами сервісу Codeforces. Серед розглянутих у першому розділі сервісів він характеризується найбільшою аудиторією та найкращим тегуванням задач, що дозволяє досягнути вищої точності оцінювання.

Досліджуваний період часу склад 18 місяців від першої спроби розв’язку задачі. При цьому враховувались результати лише тих користувачів, які розв’язували не менше 15 задач щомісяця.

Наведений графік репрезентує кваліфікацію окремого користувача для окремої теми. Для того, щоб отримати усереднений графік, що характеризує ефективність засвоєння навичок з використання алгоритмів та структур даних без застосування спеціалізованих методів навчання, було проведено наступні перетворення:

- кортежі даних для користувача об’єднувались та усереднювались шляхом розрахунку середнього арифметичного значення, в результаті

чого було утворено відповідний графік, що характеризував середню залежність кваліфікації користувача від часу по всіх темах;

- отримані в результаті попереднього перетворення дані було усереднено аналогічно до попереднього кроку, в результаті чого було отримано результуючий графік, наведений на рисунку 4.7.

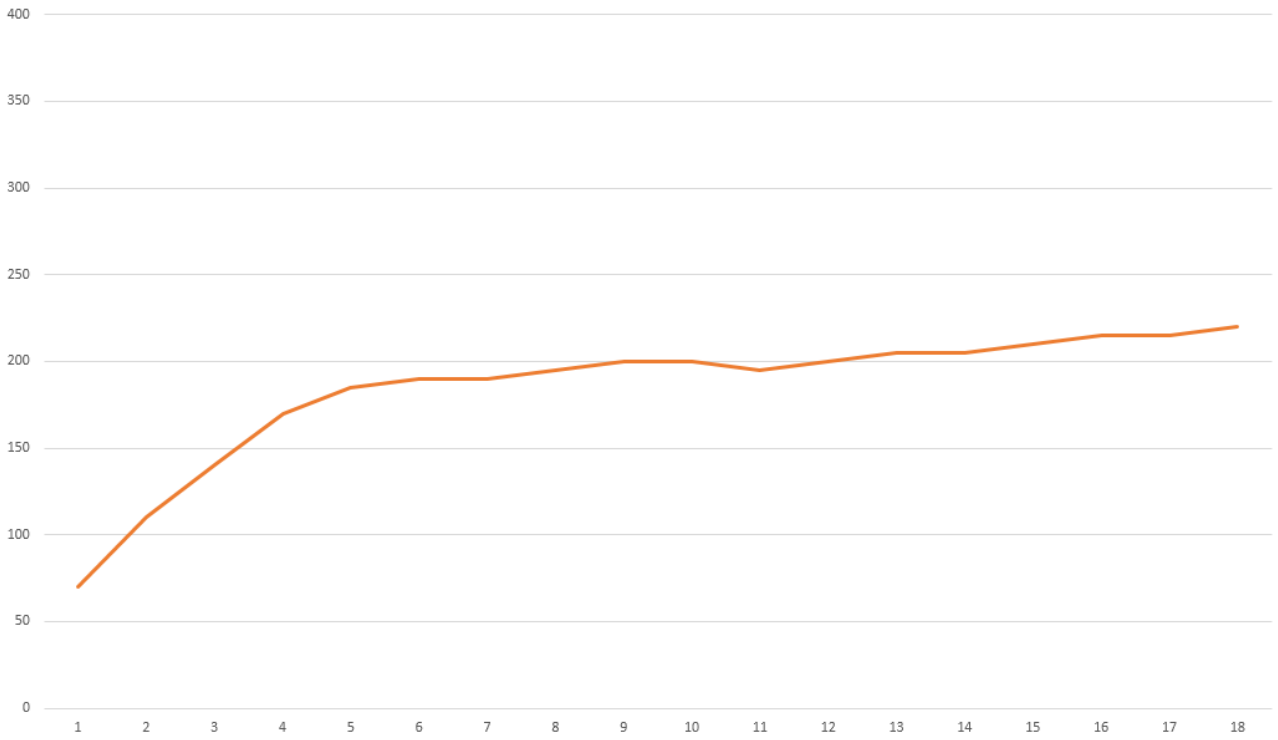

Рисунок 4.7 – Залежність кваліфікації від часу без використання спеціалізованих методів

Отримані дані свідчать про те, що розвиток кваліфікації практично припиняється після п’яти місяців, і може навіть знижуватись. Такі результати можуть бути спричинені відсутністю регулярної практики.

Для порівняння графіків їх було накладено (рис. 4.8).

Порівняння графіків свідчить про те, що використання модифікації методу Лейтнера підвищує ефективність засвоєння знань та навичок з використання алгоритмів та структур даних, а також дозволяє досягнути вищого рівня кваліфікації, порівняно з безсистемним вивченням.

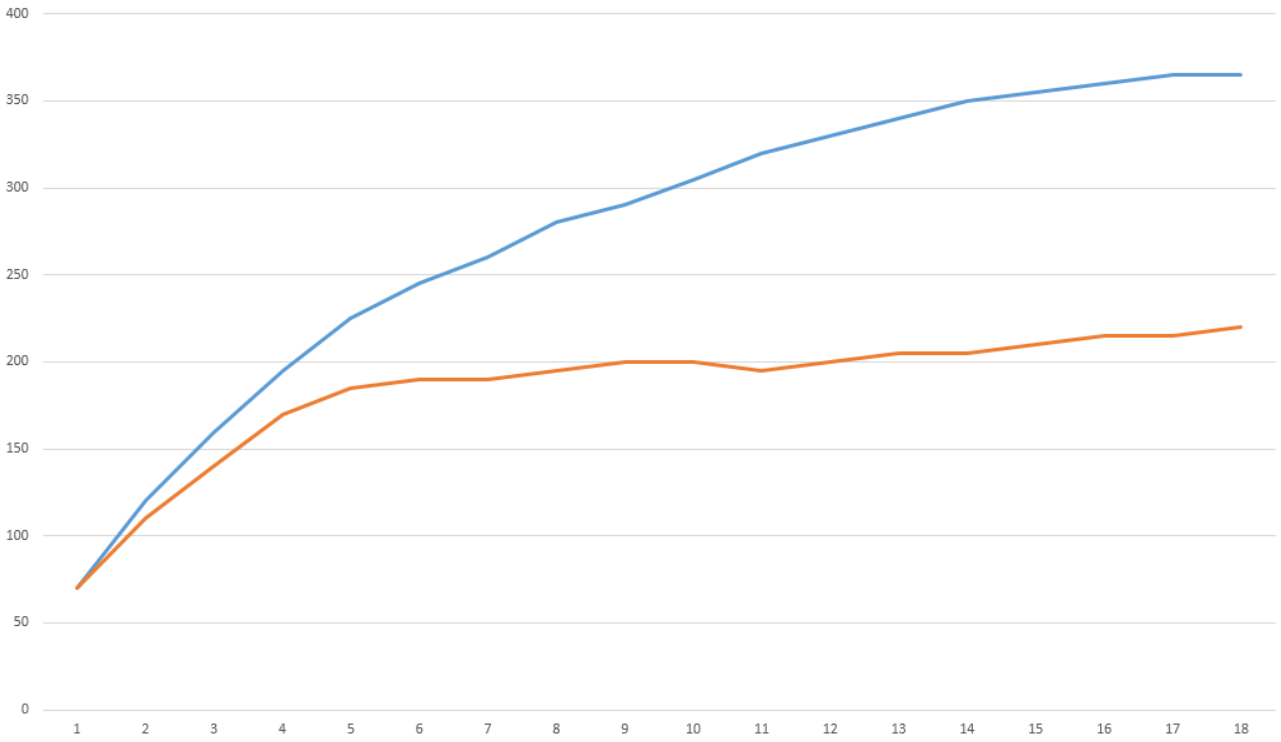

Рисунок 4.8 – Порівняння результатів дослідження

Таким чином, розроблена модифікація методу Лейтнера має теоретичну перспективу суттєво підвищити ефективність засвоєння студентами знань та навичок з використання алгоритмів та структур даних.

4.5 Висновки

У четвертому розділі магістерської кваліфікаційної роботи було розглянуто види та методики тестування, обрано методику тестування «чорної скриньки», складено набір тест-кейсів для тестування програмного продукту, проведено тестування, а також розроблено інструкцію користувача, у якій наведено покроковий опис дій, необхідних для користування системи. Було проведено аналіз ефективності розробленої модифікації методу Лейтнера і виявлено, що його використання може підвищити ефективність засвоєння студентами знань та навичок з використання алгоритмів та структур даних.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки [29]. Для проведення технологічного аудиту було залучено 2-х незалежних експертів, а саме доцент кафедри програмного забезпечення, кандидат технічних наук Романюк О. В. та доцент кафедри галузевого машинобудування, кандидат технічних наук Іванчук Я. В.

Проведено оцінювання комерційного потенціалу розробки за 12-ма критеріями за 5-ти бальною шкалою. Результати оцінювання комерційного потенціалу розробки наведено в таблиці 5.1.

Таблиця 5.1

Результати оцінювання комерційного потенціалу розробки

Критерій	Романюк О. В.	Іванчук Я. В.
	Бали, виставлені експертами	
1	4	4
2	3	3
3	3	4
4	4	4
5	3	4
6	3	4
7	3	3
8	3	4
9	4	4
10	4	3
11	3	4
12	4	4
Сума балів	41	45
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^2 СБ_i}{2} = 43$	

Отже, з отриманих даних таблиці 5.1 видно, що нова розробка має високий рівень комерційного потенціалу.

5.2 Прогнозування витрат на виконання науково-дослідні роботи та впровадження результатів

Для розробки нового програмного продукту необхідні такі витрати. Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} * t, \quad (5.1)$$

де М – місячний посадовий оклад конкретного розробника;

$Т_р$ – кількість робочих днів у місяці, $Т_р$ – 21 день;

t – кількість днів роботи розробника, t = 50 днів.

Результати розрахунку заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2

Розрахунки основної заробітної плати

Працівник	Оклад М, грн	Оплата за робочий день, грн	Кількість днів роботи, t	Витрати на оплату праці, грн
Науковий керівник	6000	285,71	5	1428,57
Інженер-програміст	3500	166,67	50	8333,33
Всього				9761,90

Розрахуємо додаткову заробітну плату:

$$З_{дод} = 9761{,}90 * 0{,}1 = 976{,}19 \text{ (грн)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати (5.2):

$$Н_{зп} = (З_о + З_р) * \frac{\beta}{100} \quad (5.2)$$

Підставивши у формулу (5.2) відомі дані, отримаємо значення нарахування на заробітну плату операторів $Н_{зп}$:

$$Н_{зп} = (9761,90 + 976,19) * \frac{36,3}{100} = 3897,93 \text{ (грн)}$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за формулою (5.3):

$$А = \frac{Ц * Н_а}{100} * \frac{Т}{12}, \quad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань, відсоток (для програмного забезпечення 25%);

Т – термін використання (Т = 3 місяці).

Розрахунок наведено в таблиці 5.3.

Таблиця 5.3

Розрахунок амортизаційних відрахувань

Найменування	Балансова вартість, грн	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань, грн
Персональний комп’ютер	9000	25	3	562,5
Всього				562,5

Розрахуємо витрати на комплектуючі. Для розрахунку витрат на комплектуючі використовується формула (5.4).

$$К = \sum_1^n Н_i * Ц_i * К_i, \quad (5.4)$$

де n – кількість комплектуючих;

$Н_i$ - кількість комплектуючих і-го виду;

$Ц_i$ – покупна ціна комплектуючих і-го виду, грн;

$К_i$ – коефіцієнт транспортних витрат (приймемо $К_i$ = 1,1)

Проведемо оцінку витрат на комплектуючі, що було використано для розробки програмного забезпечення. Результати занесено до таблиці 5.4.

Таблиця 5.4

Витрати на комплектуючі, що було використано для розробки програмного забезпечення

Найменування матеріалу	Одиниці виміру	Ціна, грн	Витрачено	Вартість витрачених матеріалів, грн
Флешка	шт.	200	1	200
Пачка паперу	уп.	120	1	120
Ручка	шт.	5	1	5
Всього з урахуванням транспортних витрат				357,5

Витрати на електроенергію розраховуються за формулою (5.5):

$$В_е = В * П * Ф * К_п, \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,7 грн/кВт);

П – установлена потужність комп’ютера (П = 0,6 кВт);

Ф – фактична кількість годин роботи комп’ютера (Ф = 200 год);

$К_п$ – коефіцієнт використання потужності ($К_п < 1$, $К_п = 0,9$).

Підставивши відомі дані в формулу (5.5), отримаємо величину витрат на електроенергію:

$$В_е = 1,7 * 0,6 * 200 * 0,9 = 183,6 \text{ (грн)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $В_{ін}$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які виконували дану роботу. Розрахунок виконується за формулою (5.6):

$$В_{ін} = (1 \dots 3) * (З_о + З_{дод}) \quad (5.6)$$

Отже, розраховуємо інші витрати:

$$В_{ін} = 1 * (9761,90 + 976,19) = 10738,09 \text{ (грн)}$$

Сума всіх попередніх статей витрат дає витрати на виконання розробки програмного забезпечення, що обчислюється за формулою (5.7):

$$В = З_о + З_{дод} + Н_{зп} + А + К + В_е + В_{ін} \quad (5.7)$$

Підставивши обчислені раніше величини у формулу (5.7), отримаємо значення вартості розробки:

$$В = 9761,9 + 976,19 + 3897,93 + 562,5 + 357,5 + 183,6 +$$
$$10738,09 = 26477,71 \text{ (грн)}$$

Розрахуємо загальну вартість наукової роботи $В_{заг}$ за формулою (5.8):

$$В_{заг} = \frac{В}{\alpha}, \quad (5.8)$$

де α – частка витрат, які безпосередньо здійснює виконавець поточного етапу роботи, у відносних одиницях – приймаємо рівним 1.

$$В_{заг} = \frac{26477,71}{1} = 26477,71 \text{ (грн)}$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою (5.9):

$$ЗВ = \frac{В_{заг}}{\beta}, \quad (5.9)$$

де β – коефіцієнт, що характеризує етап (стадію) виконання даної роботи.

$$В_{заг} = \frac{26477,71}{0,9} = 29419,68 \text{ (грн)}$$

Таким чином, загальні витрати становлять 29419,68 грн.

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою (5.10):

$$\Delta П_i = \sum_1^n (\Delta П_я * N + П_я * \Delta N)i, \quad (5.10)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

N – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технології зменшяться на 15 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 15 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 300 користувачів, протягом другого року – на 250 користувачів, протягом третього року – 175 користувачів. Реалізація інформаційної технології до впровадження результатів наукової розробки складала 800 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 250 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 15 \cdot 800 + (250 + 15) \cdot 300 = 91500 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 15 \cdot 800 + (250 + 15) \cdot (300 + 250) = 157750 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 15 \cdot 800 + (250 + 15) \cdot (300 + 250 + 175) = 204125 \text{ грн.}$$

5.4 Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$ вкладених інвестицій розраховується за формулою (5.11):

$$Е_{абс} = (ПП - PV), \quad (5.11)$$

де ПП – вартість чистих прибутків,

PV – теперішня вартість інвестицій.

Схематична характеристика руху платежів, таких як інвестиції та додаткові прибутки, наведено на рисунку 5.1.

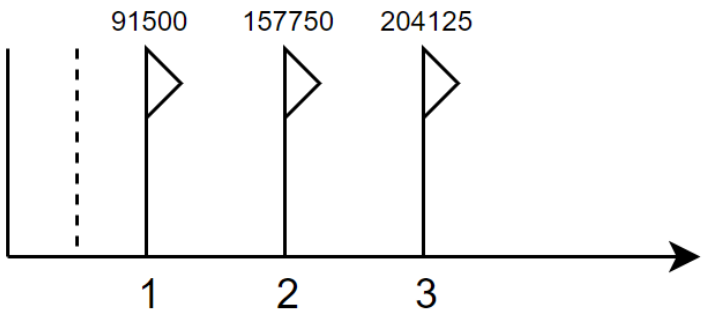

Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів магістерської кваліфікаційної роботи

Вартість чистого прибутку розраховується за формулою (5.12):

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.12)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої магістерської кваліфікаційної роботи, грн;

t – період часу, протягом якого виявляються результати впровадженої магістерської кваліфікаційної роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні. Для України цей показник знаходиться на рівні 0,1.

Розрахуємо вартість чистого прибутку:

$$ПП = \frac{29419,68}{(1+0,1)^0} + \frac{91500}{(1+0,1)^1} + \frac{157750}{(1+0,1)^2} + \frac{204125}{(1+0,1)^3} = 396335,533 \text{ (грн).}$$

На основі вартості чистого прибутку розраховуємо $Е_{абс}$:

$$Е_{абс} = 396335,533 - 29419,68 = 366915,853 \text{ (грн).}$$

Оскільки $Е_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів магістерської кваліфікаційної роботи буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою (5.13):

$$Е_в = \sqrt[T]{1 + \frac{Е_{абс}}{PV}} - 1, \quad (5.13)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій, грн;

Т – життєвий цикл наукової розробки, роки.

Таким чином,

$$E_в = \sqrt[3]{1 + \frac{366915,853}{29419,68}} - 1 = 1,37944145.$$

Показник відносної ефективності вкладених в наукову розробку інвестицій становить 137,94%.

Розрахована величина порівнюється з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає мінімальну дохідність, за якої можливе інвестування в розробку. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою (5.14):

$$\tau = d + f, \qquad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках. В 2019 році в Україні цей показник становить 0,2;

f – показник, що характеризує ризикованість вкладень. Приймемо його рівним 0,1.

Підставивши ці показники у формулу (5.14), отримаємо значення мінімальної ставки:

$$\tau = 0,2 + 0,1 = 0,3.$$

Оскільки показник відносної ефективності (137,94%) вкладених в наукову розробку інвестицій перевищує значення мінімальної ставки (30%), то розробка є такою, що має інвестиційну привабливість.

Термін окупності вкладених у реалізацію наукового проекту визначається за формулою (5.15):

$$T_{ок} = \frac{1}{E_в}, \quad (5.15)$$

Термін окупності становить 0,72 року.

5.5 Висновки

Під час виконання економічної частини магістерської кваліфікаційної роботи на основі розрахунків було доведено, що розробка методу та програмного забезпечення для підвищення ефективності засвоєння студентами знань та навичок є доцільною та має економічне обґрунтування.

На основі виконаних підрахунків одержано наступні результати:

- витрати на розробку та її впровадження складають 29,42 тис. грн., що не перевищує задане у технічному завданні значення;
- абсолютний ефект розробки складає 396,34 тис. грн. за три роки, що також задовольняє значення задане у технічному завданні;
- відносна норма дохідності була досягнута і складає 137,94%;
- термін окупності системи, що розробляється, складає менше одного року, що вписується в задані в технічному завданні часові рамки та є показником доцільності розробки.

Таким чином, отримані вище результати доводять корисність, доцільність та необхідність даної розробки.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи було розроблено метод та програмне забезпечення для підвищення ефективності засвоєння студентами знань та навичок.

1. Виконано аналіз предметної області, у результаті якого було зроблено висновок про актуальність розробки методів та засобів підвищення ефективності засвоєння студентами знань та навичок. Проведено дослідження існуючих методів підвищення ефективності навчання, а також існуючих програмних систем, що спрямовані на розвиток знань та навичок користувача. Проаналізовано метод Лейтнера, виявлено його недоліки та обмеження області застосування методу. Розглянуто існуючі методи оцінювання складності програмних алгоритмічних задач, виявлено їх недоліки.

2. Запропоновано модифікацію методу Лейтнера, що відрізняється від існуючого методу можливістю застосування для формування практичних навичок з розв’язування програмних алгоритмічних задач з використання алгоритмів та структур даних. Запропоновано новий метод оцінювання складності задач, що відрізняється від існуючих більшою кількістю врахованих чинників, що підвищує його об’єктивність.

3. Розроблено архітектуру програмної системи для впровадження розроблених методів, схему бази даних, а також програмні модулі. Для розробки використано мову програмування C#, а також фреймворки ASP.NET Core 3.0, Entity Framework Core, gRPC for .NET Core. При розробці використовувалось інтегроване середовище розробки Microsoft Visual Studio 2017 Community, а також утиліту для керування базою даних Microsoft SQL Server Management Studio.

4. Виконано тестування розробленої системи, у результаті якого було зроблено висновок про відповідність програмного продукту критеріям якості. Розроблено інструкцію користувача, яка містить всю необхідну для користування програмною системою інформацію.

Результати роботи було впроваджено на ТОВ «Дельфи» для навчання стажерів.

Отримані в магістерській кваліфікаційній роботі наукові та практичні результати можна використати для підвищення ефективності засвоєння студентами знань та навичок з використання алгоритмів та структур даних.

Проведено тестування розроблених методів та програмного забезпечення для підвищення ефективності засвоєння студентами знань та навичок, у результаті якого було встановлено високий технічний рівень та комерційний потенціал розробки. Виконано розрахунок економічного ефекту від можливого впровадження розроблених методів та програмного забезпечення, в результаті якого підтверджено доцільність фінансування цієї наукової розробки.

Підсумовуючи усе вищесказане, можна зробити висновок, що задачі магістерської кваліфікаційної роботи було виконано у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Система Лейтнера – Википедия. URL: https://ru.wikipedia.org/wiki/Система_Лейтнера (дата звернення: 25.10.2019)
2. Демчук С. В., Романюк О. В. Підвищення ефективності формування словникового запасу людини на основі аналізу результатів тестування вивченого обсягу іноземних слів. *Матеріали XLVII науково-технічної конференції підрозділів ВНТУ*. Вінниця, 14-23 березня 2018 р. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/20636/5102.pdf
3. Романюк О. В., Кавка О. О. Особливості застосування методу Лейтнера для формування практичних навичок розв'язування алгоритмічних задач в програмній інженерії. *Збірник доповідей XII Міжнародної науково-практичної конференції "Інформаційні технології і автоматизація – 2019"*, Одеса, 2019. С. 18–19.
4. Романюк О. В., Кавка О.О. Модифікація методу Лейтнера для покращення засвоєння студентами знань та навичок використання алгоритмів та структур даних у програмній інженерії. *Збірник тез доповідей XXXV Міжнародної науково-практичної інтернет-конференції «Світові тенденції сучасних наукових досліджень»*, Вінниця, 2019. С. 16–20.
5. Романюк О. В., Кавка О. О. Метод оцінювання складності алгоритмічних задач статистичним методом. *Збірник матеріалів Міжнародної науково-практичної конференції «Електронні інформаційні ресурси в освіті і науці: створення, використання, доступ»*, Вінниця, 2019.
6. Топ-5 методов запоминания информации – СКБ Контур. URL: https://kontur.ru/articles/5140 (дата звернення: 25.10.2019)
7. Часто задаваемые вопросы @ Timus Online Judge. URL: https://acm.timus.ru/help.aspx?topic=faq (дата звернення: 25.10.2019)
8. Соответствие сложности задач КФ и Тимуса – Codeforces. URL: https://codeforces.com/blog/entry/13435?locale=ru#comment-183449 (дата звернення: 25.10.2019)

9. Help – Codeforces. URL: https://codeforces.com/help (Last accessed: 25.10.2019)
10. Timus Online Judge. URL: http://acm.timus.ru (Last accessed: 25.10.2019)
11. E-Olymp. URL: https://www.e-olymp.com/uk/ (Last accessed: 25.10.2019)
12. Кривая забывания – Википедия. URL: https://ru.wikipedia.org/wiki/Кривая_забывания (дата звернення: 25.10.2019)
13. Интервальное повторение – Википедия. URL: https://ru.wikipedia.org/wiki/Интервальные_повторения (дата звернення: 25.10.2019)
14. The right time to learn: mechanisms and optimization of spaced learning. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5126970/ (Last accessed: 25.10.2019)
15. Какие методики обучения использует Lingualeo | Lingualeo Блог. URL: https://corp.lingualeo.com/ru/2016/09/23/otkuda-u-leo-nogi-rastut-o-metodikah-lingualeo/ (дата звернення: 25.10.2019)
16. Memrise. Учи язык. Мир ждёт тебя. URL: https://www.memrise.com/ru/ (дата звернення: 25.10.2019)
17. Mnemosyne – Википедия. URL: https://ru.wikipedia.org/wiki/Mnemosyne (дата звернення: 25.10.2019)
18. Быстрая сортировка – Википедия. URL: https://ru.wikipedia.org/wiki/Быстрая_сортировка#Оценка_сложности_алгоритма (дата звернення: 25.10.2019)
19. Time complexity – Wikipedia. URL: https://en.wikipedia.org/wiki/Time_complexity (Last accessed: 25.10.2019)
20. Stroustrup B. The C++ Programming Language, 4th edition. Addison-Wesley, 2013 – 1368 p.
21. A. Troelsen, Ph. Japikse. C# 6.0 and the .NET 4.6 Framework, 7th edition. Apress, 2015. 1625 p.
22. Шилдт Г. Java 8. Руководство для начинающих, 6-е издание. Москва, 2017 – 720 с.

23. Hangfire Overview. URL: https://www.hangfire.io/overview.html (Last accessed: 25.10.2019)
24. TOP IDE Index. URL: http://pypl.github.io/IDE.html (Last accessed: 25.10.2019)
25. Романюк О. Н., Савчук Т. О. Організація баз даних і знань. Вінниця: УНІВЕРСУМ, 2003. 123 с.
26. Пєтух А. М. Романюк О. В., Романюк О. Н. Бази даних. Мови запитів, управління транзакціями, розподілена обробка даних. URL: http://posibnyky.vntu.edu.ua/db/index.htm (дата звернення: 11.11.2019)
27. Поняття про тестування програмного забезпечення. URL: http://helpiks.org/5-101392.html (дата звернення: 11.11.2019)
28. Степанченко И. В. Методы тестирования программного обеспечения. Волгоград: ВолгГТУ, 2006. 74 с.
29. Козловський В.О. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт – Вінниця: ВНТУ, 2012. – 22 с.

ДОДАТКИ

ДОДАТОК А Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

УЗГОДЖЕНО	ЗАТВЕРДЖУЮ
Директор ТОВ «Дельфи»	Зав. кафедри ПЗ, д.т.н., проф.
Бондар Н. П.	Романюк О. Н.
«___» _____ 2019 р.	«___» _____ 2019 р.

Технічне завдання
на магістерську кваліфікаційну роботу
на тему: «Розробка методу та програмного забезпечення для підвищення ефективності засвоєння студентами знань і навичок»
за спеціальністю: 121 – «Інженерія програмного забезпечення»

Керівник магістерської кваліфікаційної роботи:
_____ к.т.н., доц. Романюк О. В.
«___» _____ 2019 р.
Виконав:
_____ ст. гр. 1ПІ-18м Кавка О. О.
«___» _____ 2019 р.

Вінниця – 2019 р.

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методу та програмного забезпечення для підвищення ефективності засвоєння студентами знань і навичок».

Галузь застосування – системи навчання студентів.

2. Підстава для розробки

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №____ ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки

Метою роботи є підвищення ефективності засвоєння студентами знань та навичок з використання алгоритмів та структур даних шляхом розробки та впровадження нових методів навчання.

Призначення роботи – розробка нових або модифікація існуючих методів, що характеризуватимуться вищим показником засвоюваності знань.

4. Вихідні дані для проведення науково-дослідної роботи

Перелік основних літературних джерел, на основі яких буде виконуватись магістерська кваліфікаційна робота:

1. Система Лейтнера – Википедия. URL: https://ru.wikipedia.org/wiki/Система_Лейтнера (дата звернення: 25.10.2019)
2. Демчук С. В., Романюк О. В. Підвищення ефективності формування словникового запасу людини на основі аналізу результатів тестування вивченого обсягу іноземних слів. *Матеріали XLVII науково-технічної конференції підрозділів ВНТУ*. Вінниця, 14-23 березня 2018 р. URL: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/20636/5102.pdf

5. Технічні вимоги

Вихідні дані до роботи: набір тем дисципліни «Алгоритми та структури даних», алгоритмічні задачі з архівів Codeforces, Timus Online Judge, E-Olymp, крива забування Еббінгауза, метод інтервального повторення, метод Лейтнера, кількість груп задач: 10.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт

- пояснювальна записка до магістерської кваліфікаційної роботи;
- технічне завдання;
- лістинг програмної системи.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Термін виконання етапів роботи
1	2	3
1	Аналіз предметної області та порівняльна характеристика методів підвищення ефективності вивчення алгоритмів та структур даних	04.09.2019 – 30.09.2019
2	Розробка методу підвищення ефективності формування практичних навичок розв'язування алгоритмічних задач в програмній інженерії та методу оцінювання складності алгоритмічних задач	30.09.2019 – 27.10.2019
3	Розробка програмного забезпечення	28.10.2019 – 10.11.2019

1	2	3
4	Тестування розробленого програмного продукту	11.11.2019 – 15.11.2019
5	Економічна частина	16.11.2019 – 21.11.2019

10. Порядок контролю на прийняття

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б Акт впровадження

УЗГОДЖЕНО	ЗАТВЕРДЖУЮ
Директор ТОВ «Дельфи»	Завідувач кафедри ПЗ
Бондар Н. П.	д.т.н., професор Романюк О. Н.
«___» _____ 2019 року	«___» _____ 2019 року

АКТ ВПРОВАДЖЕННЯ № ____
результатів науково-дослідної роботи

Замовник: ТОВ «Дельфи»

Цим актом підтверджується, що результат роботи «Розробка методу та програмного забезпечення для підвищення ефективності засвоєння студентами знань і навичок»,
(найменування теми)

що виконав студент групи 1ПІ-18м ВНТУ Кавка Олексій Олександрович,
(виконавці)

за договором про творчу співдружність без взаємних грошових розрахунків, на громадських засадах виконана з _____ по _____
(строки виконання)

відповідно до теми, затвердженої наказом №___ від «____» _____ 2019 року
впроваджено у ТОВ «Дельфи»
(найменування організації, де здійснювалось впровадження)

1. Вид впроваджених результатів: експлуатація програмного забезпечення
(експлуатація виробу, роботи, технології)
2. Характеристика масштабу впровадження: одиничне
(унікальне, одиничне, партія, масове, серійне)
3. Форма впровадження: дослідний зразок
4. Новизна результатів науково-дослідної роботи: якісно нові
(піонерські, принципово нові, якісно нові, модифікації, модернізація старих розробок)
5. Впроваджені: в системі навчання стажерів
6. Соціальний та науково-технічний ефект: спеціальне призначення
(охорона навколишнього середовища, поліпшення й оздоровлення умова праці, удосконалення структури керування, науково-технічних напрямків, спеціальне призначення)

Від виконавця:	Від ТОВ «Дельфи»:
Студент групи 1ПІ-18м	Директор
_____ Кавка О. О.	_____ Бондар Н. П.

Зав. кафедри ПЗ: д.т.н., професор _____ Романюк О. Н.

ДОДАТОК В Схема бази даних

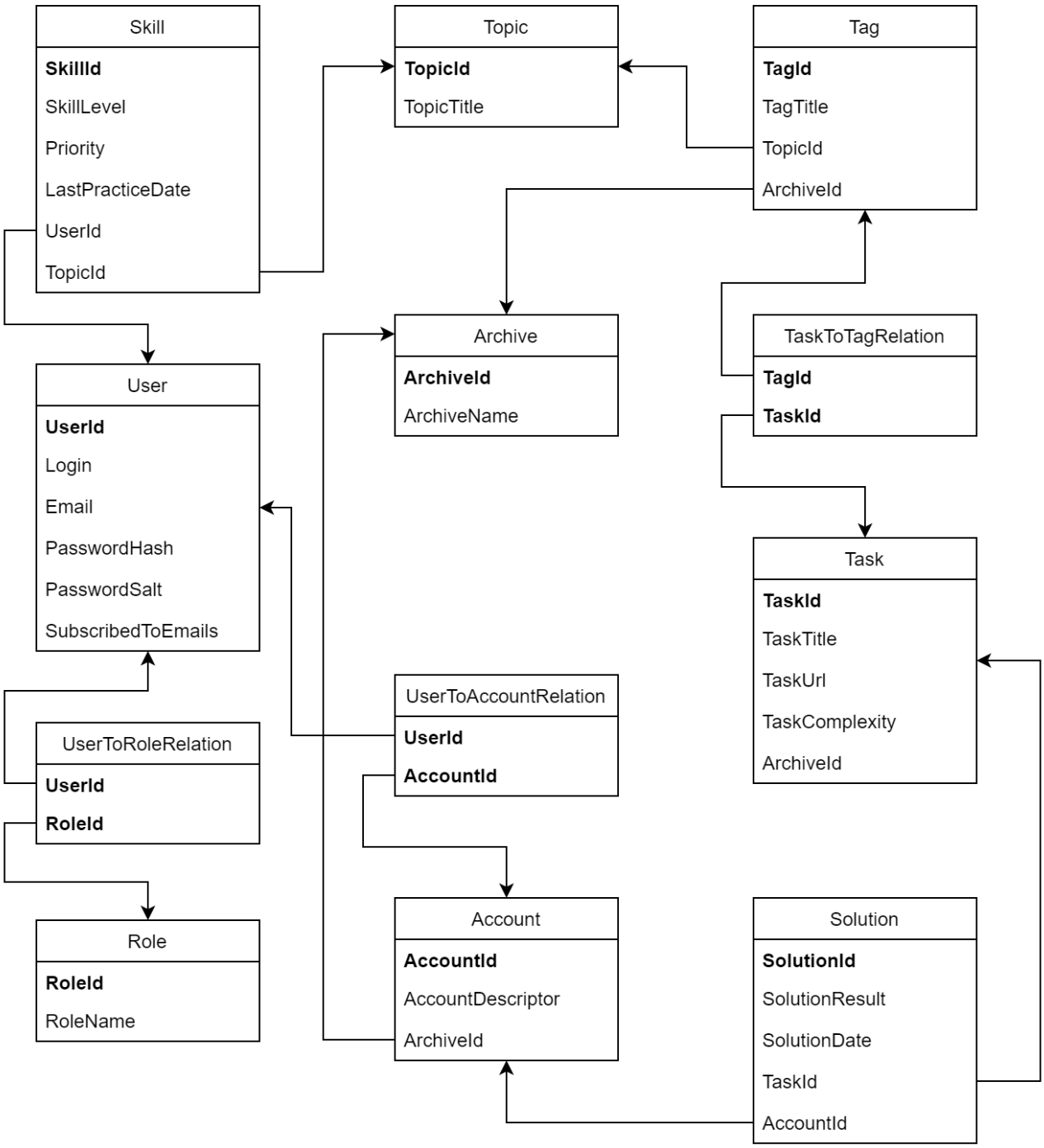

Рисунок В.1 – Схема бази даних

ДОДАТОК Г Графічні матеріали

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д.т.н., професор _____ О. Н. Романюк

Науковий керівник, к.т.н., доцент _____ О. В. Романюк

Рецензент, к.т.н., доцент _____ Я. В. Іванчук

Нормоконтроль, к.т.н., доцент _____ О. В. Романюк

Виконавець, студент групи 1ПІ-18м _____ О. О. Кавка

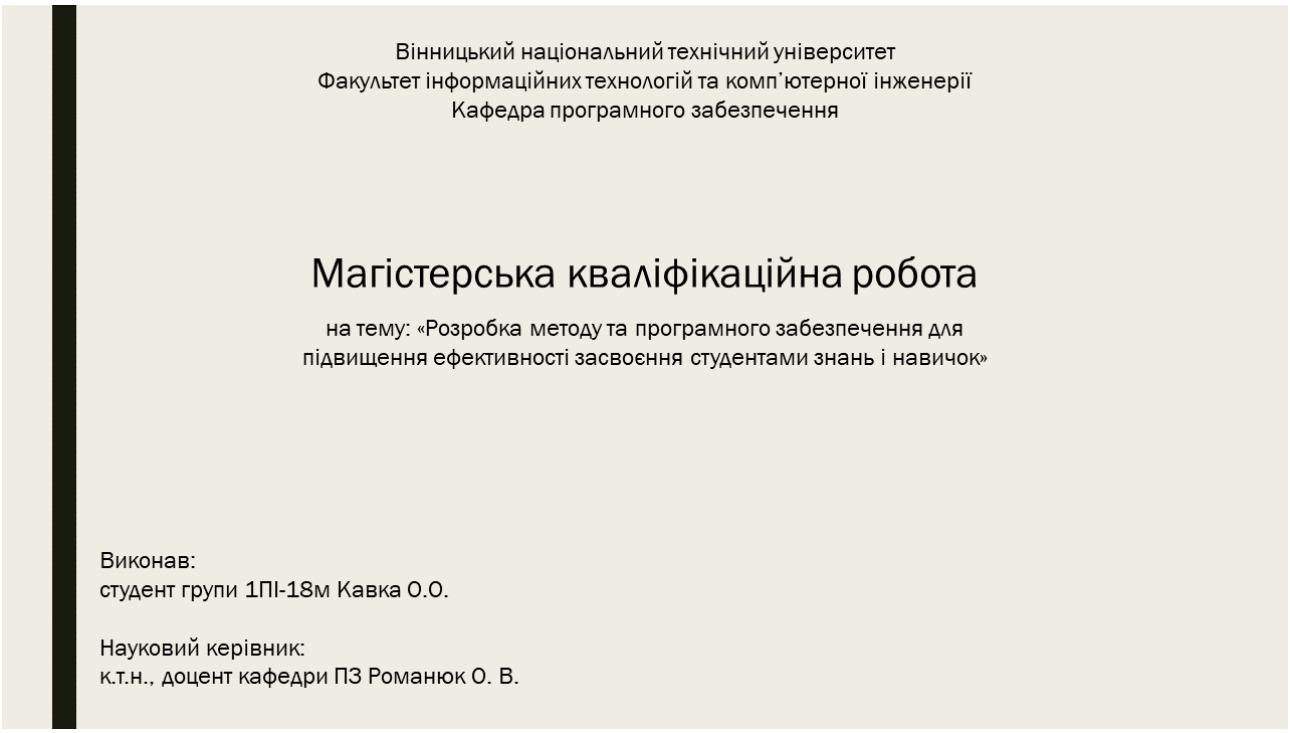

Рисунок Г.1 – Титульний слайд

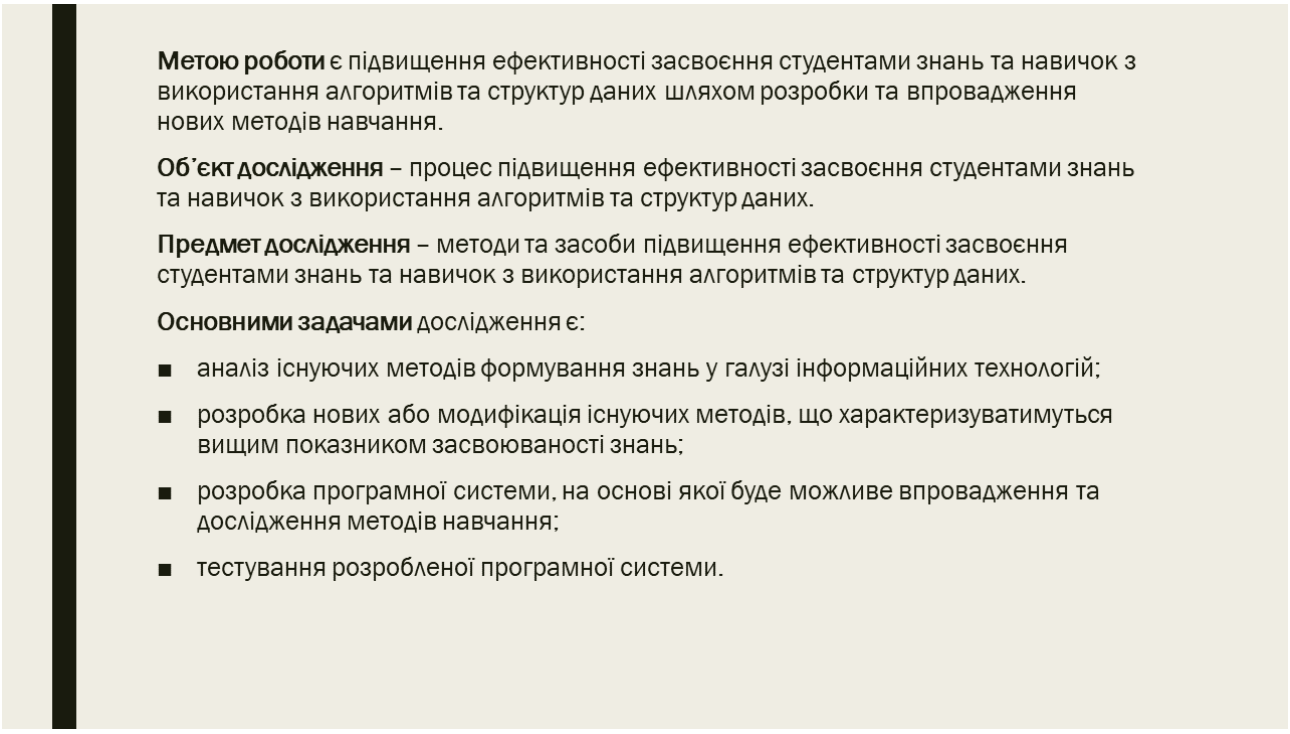

Рисунок Г.2 – Мета, об'єкт, предмет та завдання дослідження

Наукова новизна:

- Подальшого розвитку набув метод Лейтнера, який, на відміну від відомого, враховує складність освоєних задач, що дає можливість підвищити ефективність засвоєння студентами навичок з використання алгоритмів та структур даних.
- Удосконалено метод оцінювання складності алгоритмічних програмних задач, який, на відміну від існуючих, враховує кількість успішних розв'язків, відсоток успішних розв'язків та середню кількість спроб до успішного розв'язку, що підвищує об'єктивність оцінки.

Практична цінність:

- Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритм та програмні засоби для підвищення ефективності засвоєння студентами навичок використання алгоритмів та структур даних.

Рисунок Г.3 – Наукова новизна та практична цінність

Рисунок Г.4 – Крива забування Еббінгауза

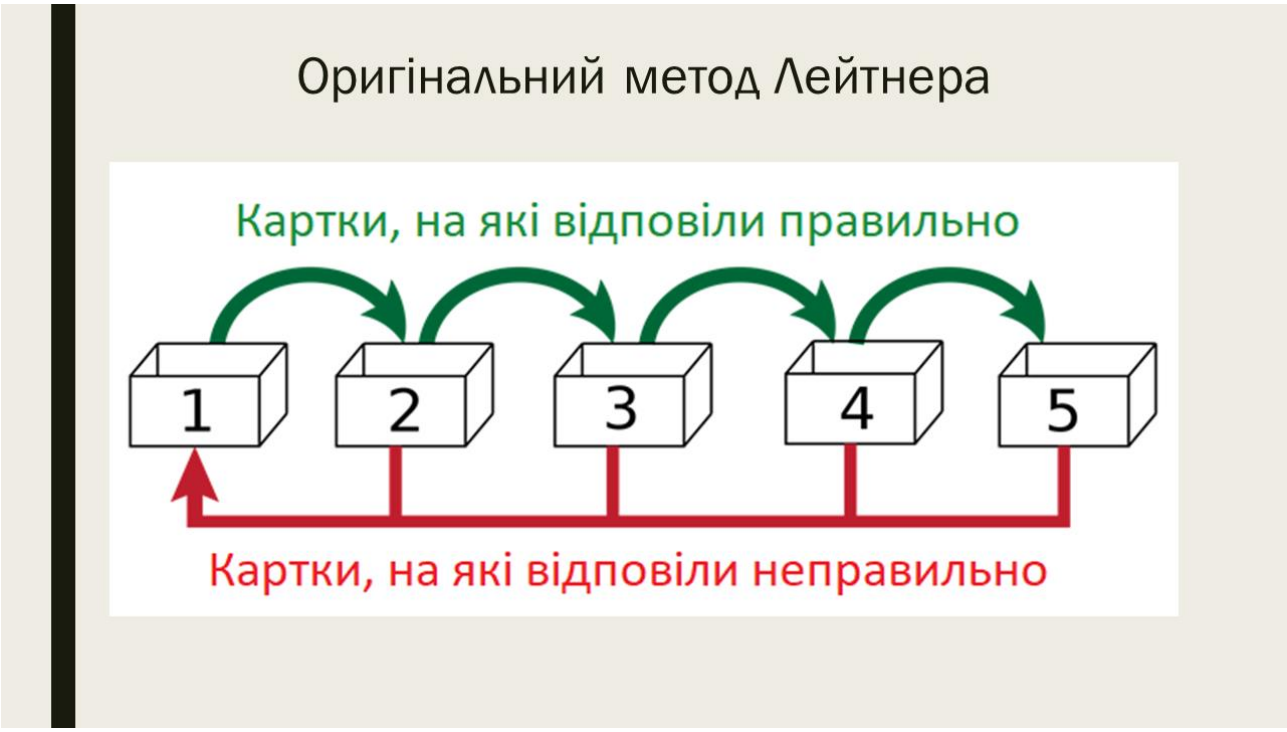

Рисунок Г.5 – Оригінальний метод Лейтнера

Рисунок Г.6 – Модифікований метод Лейтнера

Модифікований метод Лейтнера

Номер групи	Частота повторень (дні)
1	1
2	2
3	3
4	5
5	8
6	13
7	21
8	34
9	55
10	89

Рисунок Г.7 – Модифікований метод Лейтнера

Удосконалений метод оцінювання складності задач

$$X = \left\lceil \frac{A * R^2}{S * \sqrt[5]{S}} * 100 \right\rceil,$$

де

- X – числова характеристика складності задачі;
- S – кількість осіб, що розв'язали задачу;
- A – кількість осіб, що намагались розв'язати задачу;
- R – середня кількість спроб, за яку було досягнуто правильне рішення

Рисунок Г.8 – Удосконалений метод оцінювання складності задач

Удосконалений метод оцінювання складності задач

Задачу 1 намагалось розв'язати 2000 осіб, з яких 1600 успішно впорались із завданням. У середньому для цього знадобилось 1,5 спроби

$$X = \left\lceil \frac{2000 * 1.5^2}{1600 * \sqrt[5]{1600}} * 100 \right\rceil = 65$$

Задачу 2 намагалось розв'язати 800 осіб, з яких 400 успішно впорались із завданням. У середньому для цього знадобилось 1,5 спроби

$$X = \left\lceil \frac{800 * 1.5^2}{400 * \sqrt[5]{400}} * 100 \right\rceil = 136$$

Задачу 3 намагалось розв'язати 100000 осіб, з яких 50000 успішно впорались із завданням. У середньому для цього знадобилось 1,5 спроби

$$X = \left\lceil \frac{100000 * 1.5^2}{50000 * \sqrt[5]{50000}} * 100 \right\rceil = 52$$

Рисунок Г.9 – Удосконалений метод оцінювання складності задач

Архітектура системи

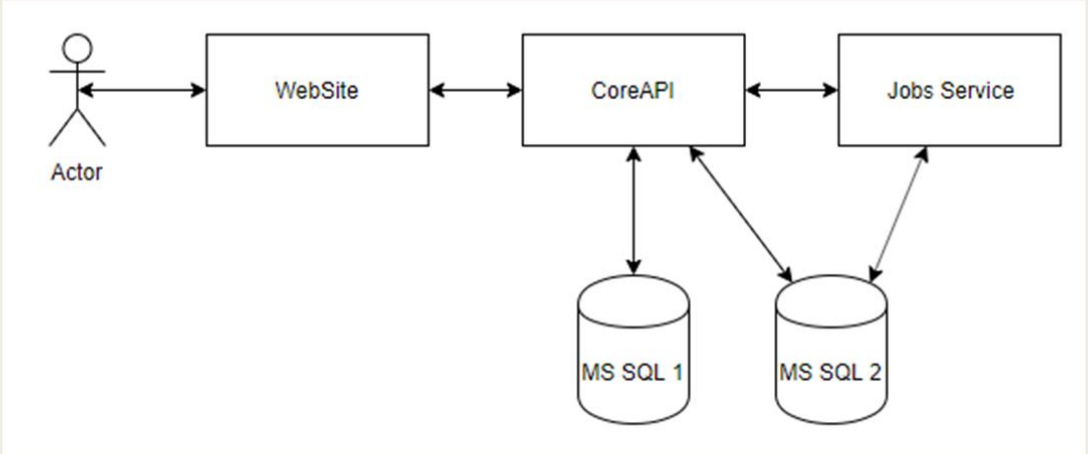

Рисунок Г.10 – Архітектура системи

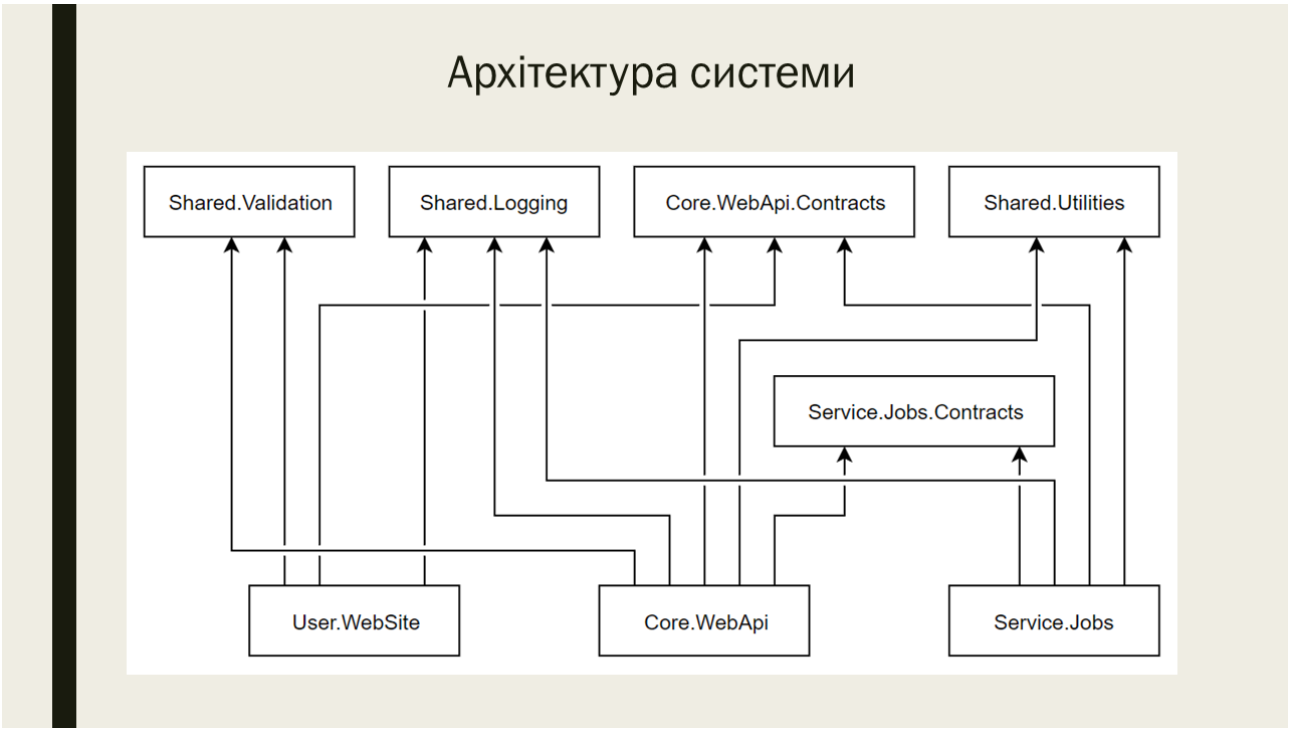

Рисунок Г.11 – Архітектура системи

Рисунок Г.12 – Ієрархія класів

Рисунок Г.13 – ER-модель предметної галузі

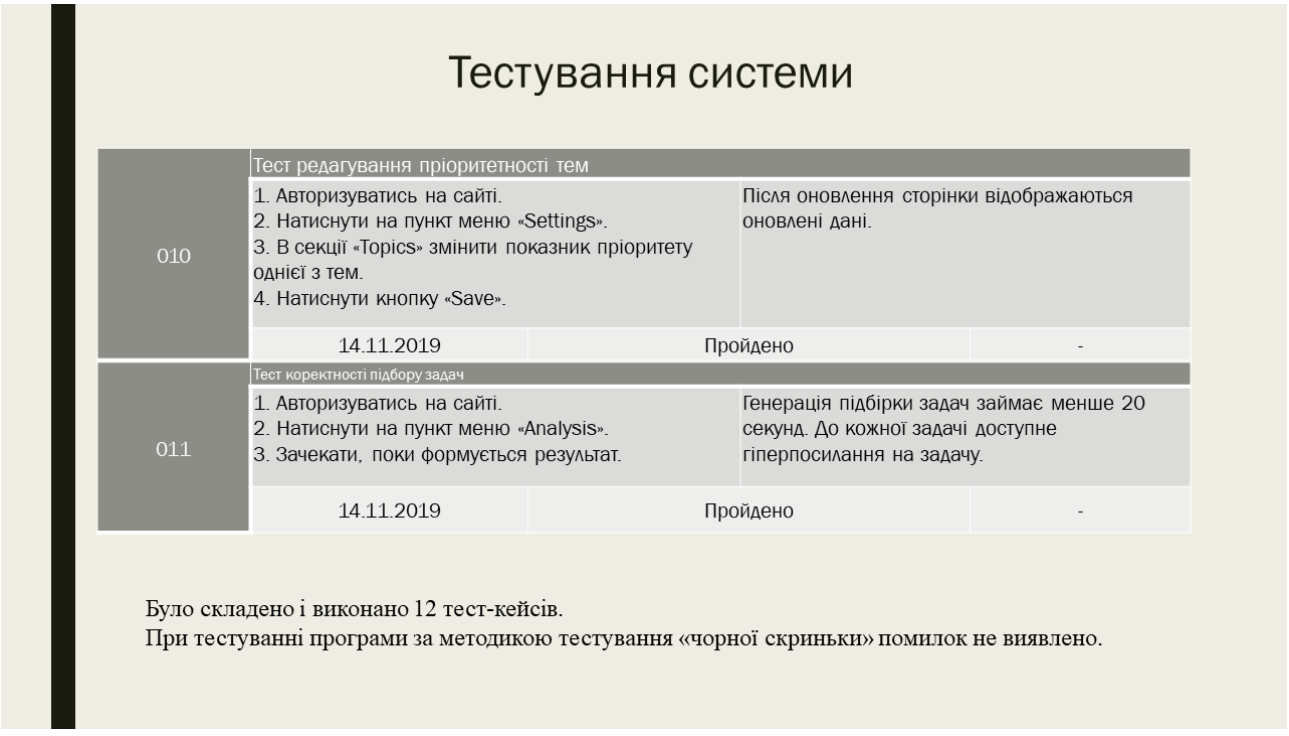

Рисунок Г.14 – Тестування системи

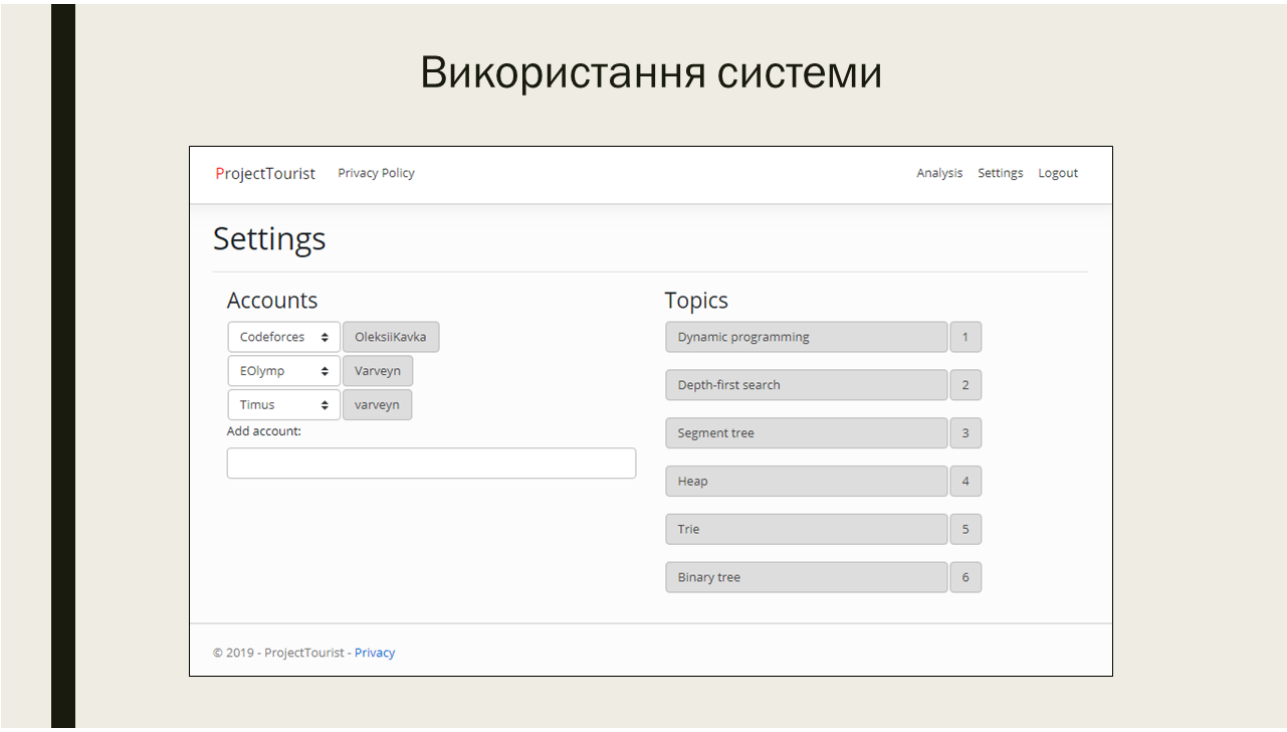

Рисунок Г.15 – Використання системи

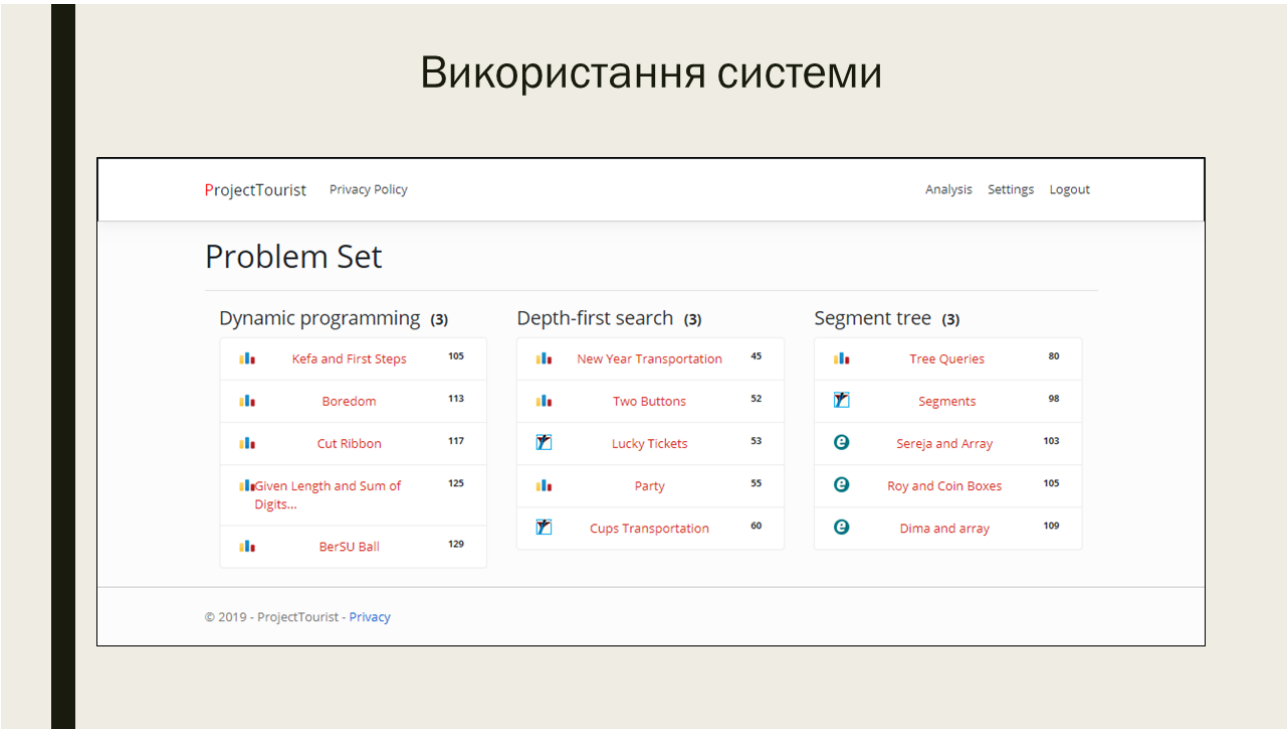

Рисунок Г.16 – Використання системи

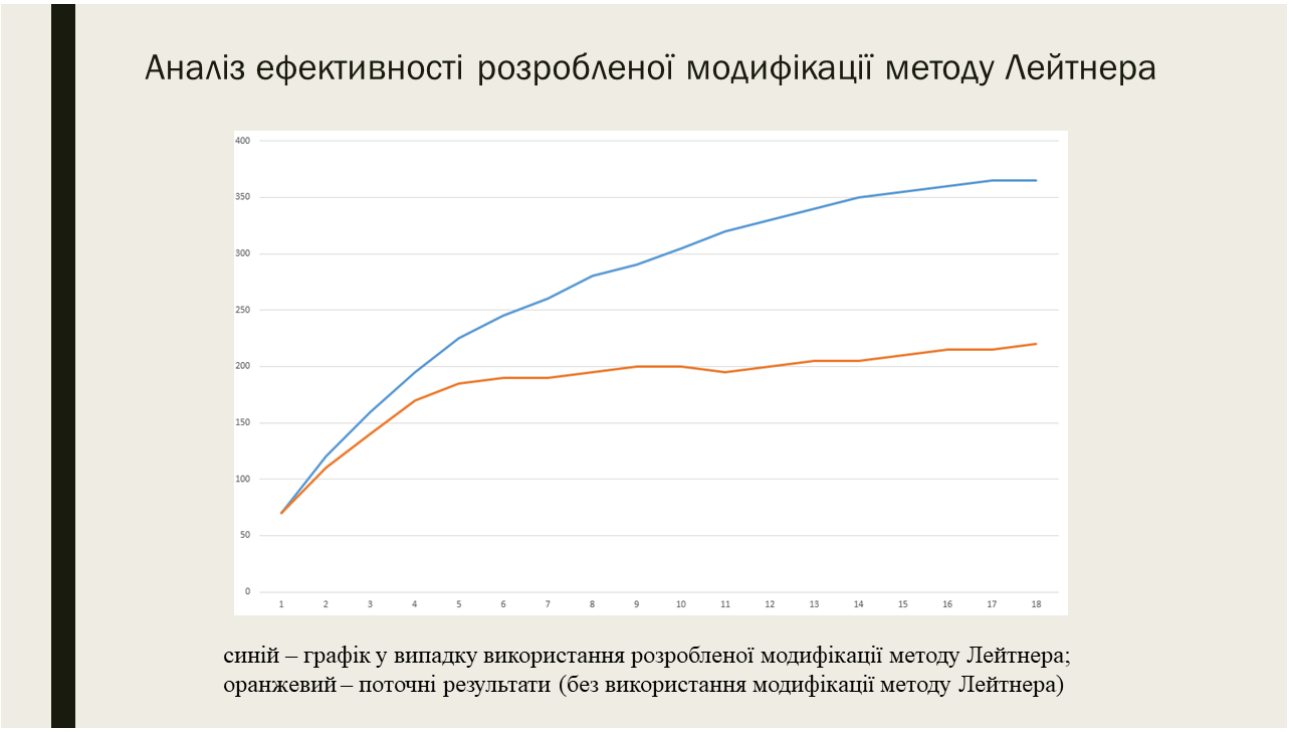

Рисунок Г.17 – Аналіз ефективності розробленої модифікації методу Лейтнера

Рисунок Г.18 – Економічне обґрунтування

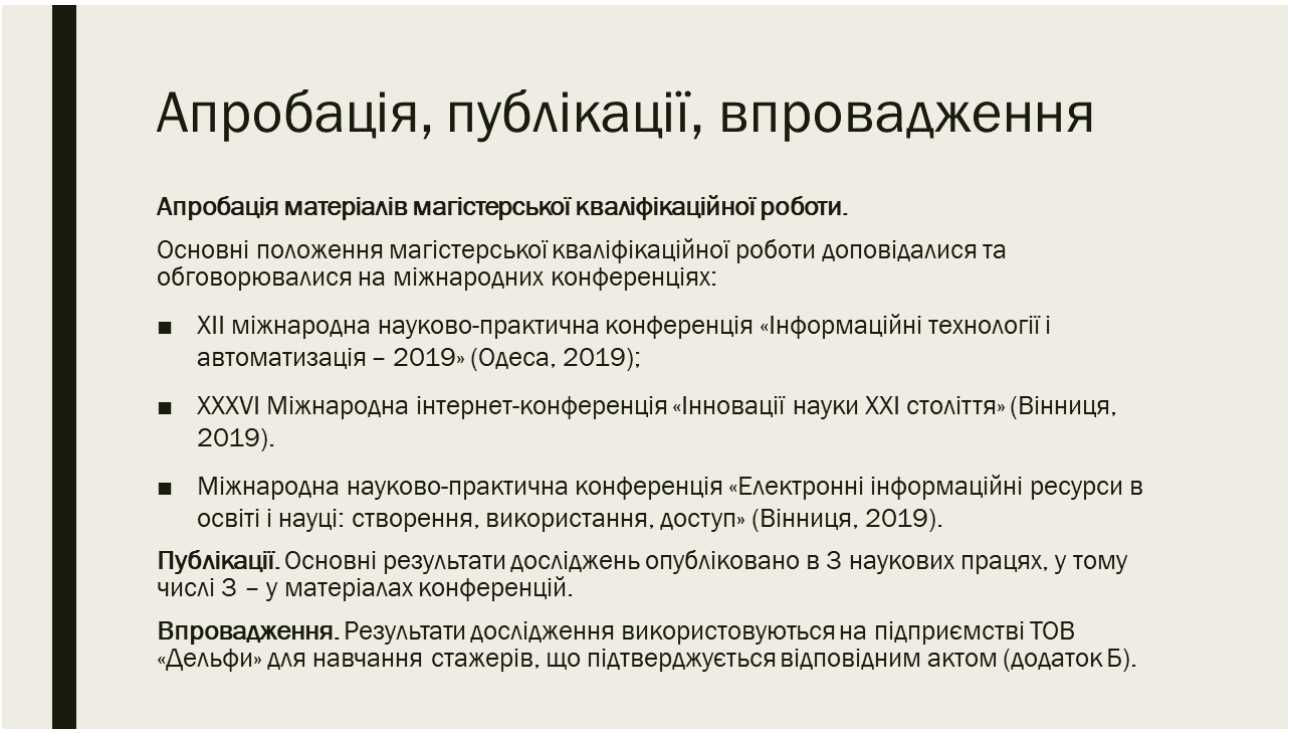

Рисунок Г.19 – Апробація, публікації, впровадження

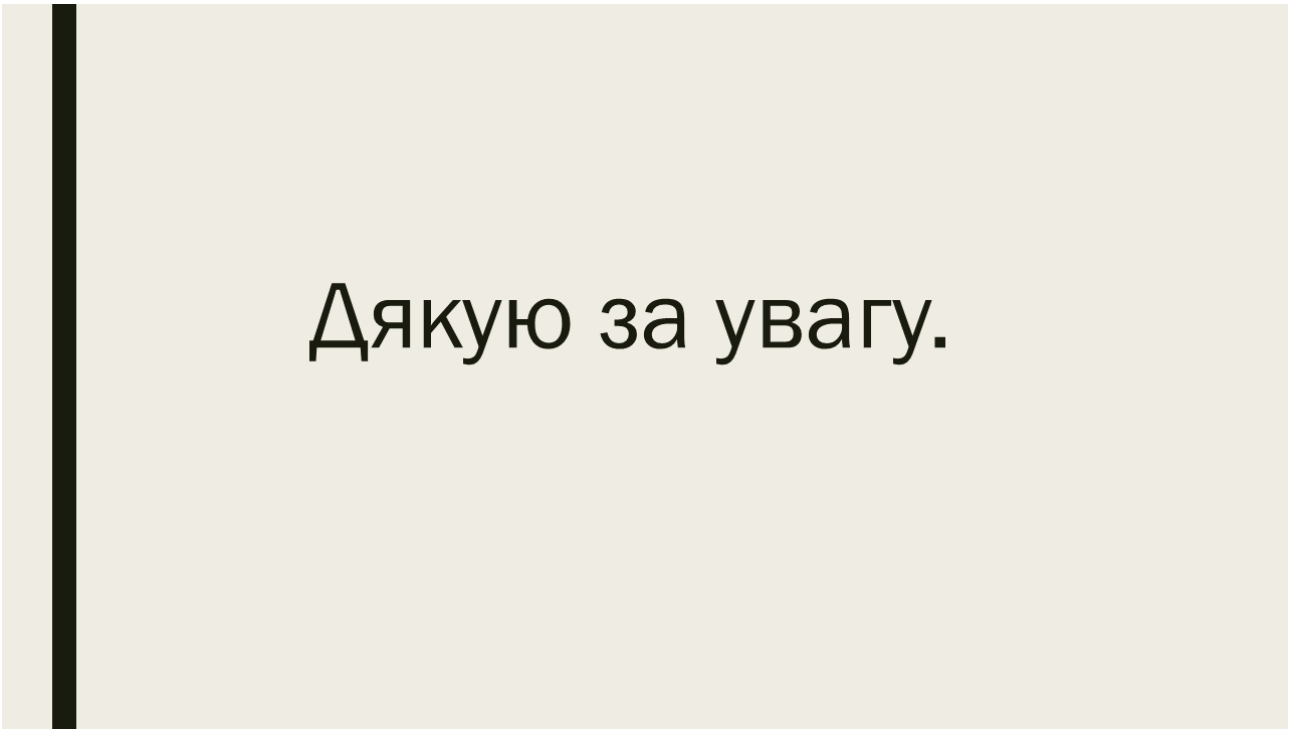

Рисунок Г.20 – Заключний слайд

ДОДАТОК Д Лістинг коду

Лістинг файлу Core.WebApi.Contracts/AuthorizationService.proto

```
syntax = "proto3";

option csharp_namespace = "Core.WebApi.Contracts.Authorization";

package Authorization;

service AuthorizationService {
	rpc Register (RegistrationRequest) returns (RegistrationResponse);
	rpc Login (LoginRequest) returns (LoginResponse);
}

message RegistrationRequest {
	string login = 1;
	string email = 2;
	string password = 3;
}

message RegistrationResponse {
	RegistrationStatus status = 1;
	repeated string errors = 2;

	enum RegistrationStatus {
		SUCCESS = 0;
		DUPLICATE = 1;
		INVALID_REQUEST = 2;
		UNKNOWN_ERROR = 3;
	}
}

message LoginRequest {
	string login = 1;
	string password = 2;
}

message LoginResponse {
	LoginStatus status = 1;
	repeated string errors = 2;

	enum LoginStatus {
		SUCCESS = 0;
		INVALID_CREDENTIALS = 1;
		INVALID_REQUEST = 2;
		UNKNOWN_ERROR = 3;
	}
}
```

Лістинг файлу Core.WebApi/AuthorizationService.cs

```
using Core.DataAccessLayer;
using Core.DataAccessLayer.Contracts.Models;
using Core.DataAccessLayer.Extensions;
using Core.WebApi.Contracts.Authorization;
using Core.WebApi.Data.Exceptions;
using Core.WebApi.Data.Validators;
using Grpc.Core;
using Shared.Utilities.Cryptography;
```

```
using System;
using System.Threading.Tasks;
using AuthorizationServiceContract = Core.WebApi.Contracts.Authorization.AuthorizationService;

namespace Core.WebApi.Services
{
   public class AuthorizationService : AuthorizationServiceContract.AuthorizationServiceBase
   {
      private readonly CoreDbContext _dbContext;

      public AuthorizationService(CoreDbContext dbContext)
      {
         _dbContext = dbContext;
      }

      public override Task<LoginResponse> Login(LoginRequest request, ServerCallContext context)
      {
         var result = new LoginResponse();

         try
         {
            ValidateRequest(request);
            ValidateCredentials(request);

            result.Status = LoginResponse.Types.LoginStatus.Success;
         }
         catch (InvalidRequestException ex)
         {
            result.Status = LoginResponse.Types.LoginStatus.InvalidRequest;
            result.Errors.Add(ex.Message);
         }
         catch (InvalidCredentialsException)
         {
            result.Status = LoginResponse.Types.LoginStatus.InvalidCredentials;
         }
         catch (Exception)
         {
            result.Status = LoginResponse.Types.LoginStatus.UnknownError;
         }

         return Task.FromResult<LoginResponse>(result);
      }

      public override Task<RegistrationResponse> Register(RegistrationRequest request, ServerCallContext context)
      {
         var result = new RegistrationResponse();

         try
         {
            ValidateRequest(request);

            var user = MapUserEntity(request);

            _dbContext.Users.Add(user);
            _dbContext.SaveChanges();

            result.Status = RegistrationResponse.Types.RegistrationStatus.Success;
         }
         catch (InvalidRequestException ex)
         {
            result.Status = RegistrationResponse.Types.RegistrationStatus.InvalidRequest;
            result.Errors.Add(ex.Message);
```

```
    }
    catch (DuplicatedEntityException)
    {
        result.Status = RegistrationResponse.Types.RegistrationStatus.Duplicate;
    }
    catch (Exception)
    {
        result.Status = RegistrationResponse.Types.RegistrationStatus.UnknownError;
    }

    return Task.FromResult<RegistrationResponse>(result);
}



private User MapUserEntity(RegistrationRequest request)
{
    var salt = CryptographyUtility.GetSalt();
    var passwordHash = CryptographyUtility.GetPasswordHash(request.Password, salt);

    return new User
    {
        Login = request.Login,
        Email = request.Email,
        PasswordHash = passwordHash,
        PasswordSalt = salt
    };
}

private void ValidateRequest(LoginRequest request)
{
    var validationResult = new LoginRequestValidator()
        .Validate(request);

    if (!validationResult.IsValid)
    {
        throw new InvalidRequestException("Login request is not valid.");
    }
}

private void ValidateRequest(RegistrationRequest request)
{
    var validationResult = new RegistrationRequestValidator()
        .Validate(request);

    if (!validationResult.IsValid)
    {
        throw new InvalidRequestException("Registration request is not valid.");
    }
}

private void ValidateCredentials(LoginRequest request)
{
    var user = _dbContext.Users.FindByLogin(request.Login);

    if (user == null)
    {
        throw new InvalidCredentialsException($"User with login '{request.Login}' does not exist.");
    }

    var passwordHash = CryptographyUtility.GetPasswordHash(request.Password, user.PasswordSalt);
```

```
            if (passwordHash != user.PasswordHash)
            {
                throw new InvalidCredentialsException($"User '{request.Login}' tries to login with invalid password.");
            }
        }
    }
}
```

Лістинг файлу Core.WebApi/Startup.cs

```
using Core.DataAccessLayer;
using Core.WebApi.Startup.ConfigurationExtensions;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace Core.WebApi.Startup
{
    public class Startup
    {
        public IConfiguration Configuration { get; set; }

        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddGrpc();

            var coreDbConnectionString = Configuration.GetConnectionString("CoreDataBase");
            services.AddDbContext<CoreDbContext>(options => options.UseMySQL(coreDbConnectionString));
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.UseGrpcServices();
                endpoints.ConfigureGetGrpcResponse();
            });
        }
    }
}
```

Лістинг файлу Core.WebApi/GrpcExtensions.cs

```
using Core.WebApi.Services;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Routing;

namespace Core.WebApi.Startup.ConfigurationExtensions
{
   internal static class GrpcExtensions
   {
      internal static void UseGrpcServices(this IEndpointRouteBuilder endpoints)
      {
         endpoints.MapGrpcService<AuthorizationService>();
      }

      internal static void ConfigureGetGrpcResponse(this IEndpointRouteBuilder endpoints)
      {
         endpoints.MapGet("/", async context =>
         {
            await context.Response.WriteAsync(
               "Communication with gRPC endpoints must be made through a gRPC client. " +
               "To learn how to create a client, visit: https://go.microsoft.com/fwlink/?linkid=2086909");
         });
      }
   }
}
```

Лістинг файлу UserConfiguration.cs

```
using Core.DataAccessLayer.Contracts.Models;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;

namespace Core.DataAccessLayer.Configurations
{
   internal class UserConfiguration : IEntityTypeConfiguration<User>
   {
      public void Configure(EntityTypeBuilder<User> builder)
      {
         builder.HasKey(x => x.Id);

         builder.HasIndex(x => x.Login).IsUnique();

         builder.Property(x => x.Login)
            .HasMaxLength(32)
            .IsRequired();

         builder.Property(x => x.Email)
            .HasMaxLength(320)
            .IsRequired();

         builder.Property(x => x.PasswordHash)
            .HasMaxLength(64)
            .IsRequired();

         builder.Property(x => x.PasswordSalt)
            .HasMaxLength(64)
            .IsRequired();
      }
   }
}
```

Лістинг файлу Shared.Utilities/CryptographyUtility.cs

```
using System;
using System.Linq;
using System.Security.Cryptography;
using System.Text;

namespace Shared.Utilities.Cryptography
{
   public static class CryptographyUtility
   {
      private const string Pepper = "eixBRwvu27vavY1wblkHpChVNOFoaYxc";

      private const int MinSaltLength = 16;

      public static string GetSalt(int saltLength = 16)
      {
         if (saltLength < MinSaltLength)
         {
            throw new ArgumentOutOfRangeException(nameof(saltLength), saltLength, $"Minimal salt length is
{MinSaltLength}.");
         }

         using (RNGCryptoServiceProvider random = new RNGCryptoServiceProvider())
         {
            var data = new byte[saltLength];
            random.GetNonZeroBytes(data);
            return ByteArrayToHashString(data);
         }
      }

      public static string GetPasswordHash(string password, string salt)
      {
         using (SHA384 hasher = SHA384.Create())
         {
            byte[] data = Encoding.ASCII.GetBytes(password + salt + Pepper);
            data = hasher.ComputeHash(data);
            return ByteArrayToHashString(data);
         }
      }

      private static string ByteArrayToHashString(byte[] salt)
      {
         return string.Concat(salt.Select(x => x.ToString("X2")));
      }
   }
}
```

Лістинг файлу User.WebSite/wwwroot/css/site.css

```
/* Please see documentation at https://docs.microsoft.com/aspnet/core/client-side/bundling-and-minification
for details on configuring this project to bundle and minify static web assets. */

a.navbar-brand {
  white-space: normal;
  text-align: center;
  word-break: break-all;
}

a.navbar-brand::first-letter {
    color: red;
}
```

```
/* Provide sufficient contrast against white background */
a {
  color: #0366d6;
}

.btn-primary {
  color: #fff;
  background-color: #1b6ec2;
  border-color: #1861ac;
}

.nav-pills .nav-link.active, .nav-pills .show > .nav-link {
  color: #fff;
  background-color: #1b6ec2;
  border-color: #1861ac;
}

/* Sticky footer styles -------------------------------------------------- */
html {
  font-size: 14px;
}
@media (min-width: 768px) {
  html {
    font-size: 16px;
  }
}

.border-top {
  border-top: 1px solid #e5e5e5;
}
.border-bottom {
  border-bottom: 1px solid #e5e5e5;
}

.box-shadow {
  box-shadow: 0 .25rem .75rem rgba(0, 0, 0, .05);
}

button.accept-policy {
  font-size: 1rem;
  line-height: inherit;
}

/* Sticky footer styles
-------------------------------------------------- */
html {
  position: relative;
  min-height: 100%;
}

body {
  /* Margin bottom by footer height */
  margin-bottom: 60px;
}
.footer {
  position: absolute;
  bottom: 0;
  width: 100%;
  white-space: nowrap;
  line-height: 60px; /* Vertically center the text there */
}
```

Лістинг файлу User.WebSite/Controllers/AuthorizationController.cs

```
using Core.WebApi.Contracts.Authorization;
using Microsoft.AspNetCore.Authentication;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Security.Claims;
using System.Threading.Tasks;
using User.WebSite.ViewModels.Authorization;
using static Core.WebApi.Contracts.Authorization.AuthorizationService;
using static Core.WebApi.Contracts.Authorization.LoginResponse.Types;
using static Core.WebApi.Contracts.Authorization.RegistrationResponse.Types;

namespace User.WebSite.Controllers
{
   public class AuthorizationController : Controller
   {
      private readonly AuthorizationServiceClient _authorizationService;

      public AuthorizationController(
         AuthorizationServiceClient authorizationService)
      {
         _authorizationService = authorizationService;
      }

      [HttpGet]
      [Route("register")]
      public IActionResult GetRegisterPage()
      {
         if (HttpContext.User.Identity.IsAuthenticated)
         {
            return View("AlreadyLoggedIn");
         }

         return View("Register");
      }

      [HttpPost]
      [Route("register")]
      public async Task<IActionResult> Register(RegisterViewModel model)
      {
         if (HttpContext.User.Identity.IsAuthenticated)
         {
            return View("AlreadyLoggedIn");
         }

         if (!ModelState.IsValid)
         {
            return View("Register");
         }

         var request = MapToRegistrationRequest(model);
         var registrationResponse = _authorizationService.Register(request);

         if (registrationResponse.Status == RegistrationStatus.Success)
         {
            await Login(model.Login);
            return Redirect("/start");
         }
```

```
    else
    {
        return View("Register");
    }
}

[HttpGet]
[Route("login")]
public IActionResult GetLoginPage()
{
    if (HttpContext.User.Identity.IsAuthenticated)
    {
        return View("AlreadyLoggedIn");
    }

    return View("Login");
}

[HttpPost]
[Route("login")]
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (HttpContext.User.Identity.IsAuthenticated)
    {
        return View("AlreadyLoggedIn");
    }

    if (!ModelState.IsValid)
    {
        return View("Login");
    }

    var request = MapToLoginRequest(model);
    var registrationResponse = _authorizationService.Login(request);

    if (registrationResponse.Status == LoginStatus.Success)
    {
        await Login(model.Login);
        return Redirect("/start");
    }
    else
    {
        return View("Login");
    }
}

[HttpGet]
[Route("logout")]
public IActionResult GetLogoutPage()
{
    if (HttpContext.User.Identity.IsAuthenticated)
    {
        return View("Logout");
    }

    return Redirect("/start");
}

[HttpPost]
[Route("logout")]
public async Task<IActionResult> Logout()
{
```

```
            if (HttpContext.User.Identity.IsAuthenticated)
            {
                await HttpContext.SignOutAsync();
            }

            return Redirect("/start");
        }


        private RegistrationRequest MapToRegistrationRequest(RegisterViewModel model)
        {
            return new RegistrationRequest
            {
                Login = model.Login,
                Email = model.Email,
                Password = model.Password
            };
        }

        private LoginRequest MapToLoginRequest(LoginViewModel model)
        {
            return new LoginRequest
            {
                Login = model.Login,
                Password = model.Password
            };
        }

        private async Task Login(string login)
        {
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, login),
                new Claim(ClaimTypes.Role, "User")
            };

            var identity = new ClaimsIdentity(claims, CookieAuthenticationDefaults.AuthenticationScheme);
            var principal = new ClaimsPrincipal(identity);

            HttpContext.User = principal;
            await HttpContext.SignInAsync(
                scheme: CookieAuthenticationDefaults.AuthenticationScheme,
                principal: principal,
                properties: new AuthenticationProperties
                {
                    ExpiresUtc = DateTime.Now.AddDays(2)
                });
        }
    }
}
```

Лістинг файлу User.WebSite/ViewsShared/_Layout.cshtml

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - Project Tourist</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
    <link rel="stylesheet" href="~/lib/simplex/bootstrap.min.css" />
    <link rel="stylesheet" href="~/css/site.css" />
```

```
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow
mb-3">
            <div class="container">
                <a class="navbar-brand" href="/start">ProjectTourist</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-
controls="navbarSupportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" href="/privacy">Privacy Policy</a>
                        </li>
                    </ul>
                </div>
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                    <ul class="navbar-nav">
                        @if (!Context.User.Identity.IsAuthenticated)
                        {
                            <li class="nav-item">
                                <a class="nav-link text-dark" href="/login">Login</a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark" href="/register">Register</a>
                            </li>
                        }
                        else
                        {
                            <li class="nav-item">
                                <a class="nav-link text-dark" href="/analysis">Analysis</a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark" href="/settings">Settings</a>
                            </li>
                            <li class="nav-item">
                                <a class="nav-link text-dark" href="/logout">Logout</a>
                            </li>
                        }
                    </ul>
                </div>
            </div>
        </nav>
    </header>
    <div class="container">
        <main role="main" class="pb-3">@RenderBody()</main>
    </div>
    <footer class="border-top footer text-muted">
        <div class="container">
            &copy; 2019 - ProjectTourist - <a href="/privacy">Privacy</a>
        </div>
    </footer>
    <script src="~/lib/jquery/dist/jquery.min.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
    <script src="~/js/site.js" asp-append-version="true"></script>
    @RenderSection("Scripts", required: false)
</body>
</html>
```

Лістингфайлу User.WebSite/Views/Account/GetSettings.cshtml

```
@using User.WebSite.Data;

@model User.WebSite.ViewModels.Account.SettingsViewModel

@{
    ViewData["Title"] = "Settings";
}

<h1>Settings</h1>
<hr />

<div class="container">
    <div class="row">
        <div class="col-lg-6">
            <h3>Accounts</h3>
            @foreach (var account in Model.Accounts)
            {
                <table>
                    <tr>
                        <td>
                            <select class="custom-select" asp-for="@account.Key" asp-
items="Html.GetEnumSelectList<Archive>()">
                                <option selected="selected" value="">Please select</option>
                            </select>
                        </td>
                        <td>
                            <text class="input-group-text">@account.Value</text>
                        </td>
                    </tr>
                </table>
            }
            <div class="form-group">
                <label for="usr">Add account:</label>
                <input type="text" class="form-control">
            </div>
        </div>
        <div class="col-lg-6">
            <h3>Topics</h3>
            @foreach (var topic in Model.Topics)
            {
                <div class="form-group">
                    <table>
                        <tr>
                            <td>
                                <text class="input-group-text" style="width:300px">@topic.Key</text>
                            </td>
                            <td>
                                <text class="input-group-text">@topic.Value</text>
                            </td>
                        </tr>
                    </table>
                </div>
            }
        </div>
    </div>
</div>

@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Лістинг файлу User.WebSite/Views/Analytics/GetProblemSet.cshtml

```
@model User.WebSite.ViewModels.Analytics.ProblemSetViewModel

@{
    ViewData["Title"] = "Problem Set";
}

<h1>Problem Set</h1>
<hr />

<div class="container">
    <div class="row">
        @foreach (var topic in Model.Tasks)
        {
            <div class="col-lg-4">
                <h4>@topic.Key <span class="badge">(3)</span></h4>
                <ul class="list-group">
                    @foreach (var task in topic.Value)
                    {
                        <li class="list-group-item d-flex justify-content-between align-content-lg-center">
                            @switch (task.Archive)
                            {
                                case Data.Archive.Codeforces:
                                    <img style="height:16px;width:16px;" src="~/resources/codeforces-logo.png" />
                                    break;

                                case Data.Archive.EOlymp:
                                    <img style="height:16px;width:16px;" src="~/resources/eolymp-logo.png" />
                                    break;

                                case Data.Archive.Timus:
                                    <img style="height:16px;width:16px;" src="~/resources/timus-logo.png" />
                                    break;
                            }

                            <a class="btn-link" href="example.com"> @task.Url </a>
                            <span class="badge">@task.Complexity</span>
                        </li>
                    }
                </ul>
            </div>
        }
    </div>
</div>
```