### 

### КУРСОВА РОБОТА

#### КР. ІП - 460.04.00.000 ПЗ

#### ІП-22-3

Бачкур Артур

2024

Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти та газу Інститут інформаційних технологій

Допущено до захисту

Керівник доц. Мельник В. Д.

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

**КУРСОВА РОБОТА**

**з дисципліни “Конструювання програмного забезпечення”**

на тему: Конструювання “Організації позабюджетної освіти”

Студента 3 курсу,

Група     ІП-22-3 спеціальності 121 Інженерія програмного забезпечення

Бачкур А. І.

**(прізвище та ініціали)**

Керівник доц. Мельник В. Д.

**(посада, вчене звання, науковий ступінь, прізвище та ініціали)**

Члени комісії

**(посада, вчене звання, науковий ступінь, прізвище та ініціали)**

Оцінка

Національна шкала Кількість балів ECTS

Засвідчую, що у цьому курсовому проекті немає запозичень з праць інших авторів без відповідних посилань. Студента Бачкур А. І.

**(підпис) (розшифровка підпису)**

м. Івано-Франківськ — 2024 рік

Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра *Інженерії програмного забезпечення*

Дисципліна *Конструювання програмного забезпечення*

Спеціальність *121 Інженерія програмного забезпечення*

Курс 3 Група *ІП-22-3* Семестр 5

ЗАВДАННЯ НА КУРСОВИЙ ПРОЄКТ (РОБОТУ)

Студенту Бачкуру Артуру Івановичу

(прізвище, ім’я, по батькові)

1. Тема проекту *Конструювання “Організації позабюджетної освіти”*
2. Термін здачі студентом закінченого проекту *30.11.2024*
3. Вихідні дані до проекту *Завдання, програма дисципліни, методичні вказівки*
4. Зміст розрахунково–пояснювальної записки (перелік питань, що їх належить розробити: *Перелік умовних позначень, символів, одиниць, скорочень і термінів. Анотація. Вступ. Розділ 1. Аналіз теорії конструювання програмного забезпечення: 1.1 Визначення архітектурних точок зору для проектування системи на основі синтетичного підходу опису архітектури 1.2 Поширені UML-діаграми опису модулів, логіки, компонентів, руху інформації, безпеки, розгортання тощо 1.3 Високорівневе проектування, порядок розробки вимог 1.4 Ланцюжок конструювання: концептуальне-структурне-ООП 1.5 Метрики чистоти та якості коду 1.6 Тестування та налагодження – тест-план, структура звітності, TDD, рефакторинг. Розділ 2. Розробка концептуальної моделі предметної області. Розділ 3. Аналіз вимог. Розділ 4. Візуалізація бізнес-логіки предметної області: 4.1. Діаграма модулів. 4.2. Діаграми варіантів використання (прецедентів, use case diagram). 4.3. Діаграми взаємодії (interaction diagram). 4.4. Діаграми діяльності (activity diagram). 4.5. Діаграми класів (UML class diagram). 4.6. Діаграми станів та переходів (statechart diagram). 4.7. Діаграма розгортання (deployment diagram). Розділ 5. Опис класів та їх методів: 5.1. Конструювання коду реалізовуваного модуля (архітектура, бізнес-логіка, конектори). 5.2. Підключені бібліотеки та фреймворки. 5.3. Налаштування для тестового розгортання (середовища, параметри).Розділ 6. Аналіз якості та відповідності: 6.1. План перевірки та тестування. 6.2. Звіт з тестування із аналізом відповідності функціональним вимогам. 6.3. Рев’ю коду на відповідність нефункціональним вимогам та постулатам «чистого коду». 6.4. Точки та план рефакторингу. Висновки. Список використаних джерел. Додатки.*
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

*Презентація із представленням усіх розділів, схем, документів, прикладів коду.*

1. Дата видачі завдання 02*.09.2024*

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п/п | Найменування етапів | Термін виконання етапів  роботи | Об'єм роботи (%) | Відмітка про виконання |
| 1 | Опрацювання літератури згідно тематики роботи | 02.09.2024 | 10 | виконано |
| 2 | Розробка концепції функціоналу предметної області | 18.09.2024 | 10 | виконано |
| 3 | Аналіз вимог до розробки | 27.09.2024 | 10 | виконано |
| 4 | Проектування бізнес-логіки | 15.10.2024 | 10 | виконано |
| 5 | Конструювання основних програмних модулів | 29.10.2024 | 35 | виконано |
| 6 | Тестування та відлагодження роботи програмних модулів | 07.11.2024 | 20 | виконано |
| 7 | Оформлення роботи | 17.11.2024 | 5 | виконано |

**Студент**                                                            Бачкур А. І.

(підпис)                                                                                   (розшифровка підпису)

**Керівник**                                                              Мельник В. Д.

(підпис)                                                                                   (розшифровка підпису)

« 2    »        вересня        2024

**ЗМІСТ**

[ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 7](#_Toc182764260)

[АНОТАЦІЯ 8](#_Toc182764261)

[ВСТУП 9](#_Toc182764262)

[1. АНАЛІЗ ТЕОРІЇ КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 10](#_Toc182764263)

[**1.1 Визначення архітектурних точок зору для проектування системи на основі синтетичного підходу опису архітектури** 10](#_Toc182764264)

[**1.2 Поширені UML-діаграми опису модулів, логіки, компонентів, руху інформації, безпеки, розгортання тощо** 11](#_Toc182764265)

[**1.3 Високорівневе проектування, порядок розробки вимог** 13](#_Toc182764266)

[**1.4 Ланцюжок конструювання: концептуальне-структурне-ООП** 14](#_Toc182764267)

[**1.5 Метрики чистоти та якості коду** 16](#_Toc182764268)

[**1.6 Тестування та налагодження – тест-план, структура звітності, TDD, рефакторинг** 18](#_Toc182764269)

[2. РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ 20](#_Toc182764270)

[3. АНАЛІЗ ВИМОГ 22](#_Toc182764271)

[4. ВІЗУАЛІЗАЦІЯ БІЗНЕС-ЛОГІКИ ПРЕДМЕТНОЇ ОБЛАСТІ 23](#_Toc182764272)

[**4.1** **Діаграма модулів** 23](#_Toc182764273)

[**4.2** **Діаграма варіантів використання** 24](#_Toc182764274)

[**4.3** **Діаграма взаємодії** 24](#_Toc182764275)

[**4.4** **Діаграма класів** 25](#_Toc182764276)

[**4.5** **Діаграма діяльності** 25](#_Toc182764277)

[**4.6** **Діаграма станів та переходів** 26](#_Toc182764278)

[**4.7** **Діаграма розгортання** 27](#_Toc182764279)

[5. ОПИС КЛАСІВ ТА ЇХ МЕТОДІВ 28](#_Toc182764280)

[**5.1** **Конструювання коду реалізованого модуля** 28](#_Toc182764281)

[**5.2 Підключені бібліотеки та фреймворки** 30](#_Toc182764282)

[**5.3 Налаштування для тестового розгортання (середовища, параметри)** 31](#_Toc182764283)

[6. АНАЛІЗ ЯКОСТІ ТА ВІДПОВІДНОСТІ 33](#_Toc182764284)

[**6.1 План перевірки та тестування** 33](#_Toc182764285)

[**6.2 Звіт з тестування із аналізом відповідності функціональним вимогам** 34](#_Toc182764286)

[**6.3 Рев’ю коду на відповідність нефункціональним вимогам та постулатам «чистого коду»** 36](#_Toc182764287)

[**6.4 Точки та план рефакторингу** 39](#_Toc182764288)

[ВИСНОВОК 41](#_Toc182764289)

[ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 42](#_Toc182764290)

[ДОДАТКИ 43](#_Toc182764291)

# **ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ**

1. **Авторизація** – процес перевірки прав доступу користувача до системи шляхом введення логіна та пароля.
2. **База даних (БД)** – організований набір даних, який зберігається та управляється програмним забезпеченням для баз даних, у цьому проєкті – SQL Server.
3. **GUI (Graphical User Interface)** – графічний інтерфейс користувача, який забезпечує взаємодію користувача з програмою через візуальні елементи, такі як кнопки, форми, вкладки.
4. **SQL (Structured Query Language)** – мова структурованих запитів, яка використовується для створення, управління та запиту даних у реляційних базах даних.
5. **Реляційна база даних (РБД)** – модель бази даних, в якій дані організовані у вигляді таблиць з рядками та стовпцями, пов'язаних між собою зовнішніми ключами.
6. **UML (Unified Modeling Language)** – уніфікована мова моделювання, яка використовується для створення діаграм, що описують структуру та процеси в системі.
7. **Інтерфейс** – набір засобів і правил, які забезпечують взаємодію між користувачем і системою, наприклад, графічний інтерфейс або програмний API.
8. **C#** – об'єктно-орієнтована мова програмування, яка використовується для створення інтерфейсу та логіки застосунку.
9. **Масштабованість** – здатність системи адаптуватися до збільшення кількості користувачів або розширення функціональних можливостей без втрати продуктивності.

# **АНОТАЦІЯ**

Ця курсова робота присвячена розробці інформаційної системи для організації позабюджетної освіти, яка охоплює короткострокові курси та програми другої вищої освіти. Метою проєкту є створення зручного та ефективного програмного продукту, який дозволяє користувачам реєструватися, переглядати курси, додавати їх у кошик та здійснювати покупки, а адміністраторам — управляти контентом та аналізувати діяльність. Робота включає аналіз вимог, створення концептуальної моделі предметної області, розробку бізнес-логіки та модулів системи, візуалізацію процесів і структури через UML-діаграми. Реалізація системи виконана з використанням C# та SQL Server. Результатом є готовий програмний продукт, який забезпечує простоту у використанні, гнучкість та високу функціональність. Система відповідає сучасним вимогам цифрової освіти та сприяє підвищенню ефективності навчального процесу.

# **ВСТУП**

У сучасному світі освіта стає невід'ємною складовою професійного розвитку та досягнення кар'єрних цілей. Поширення позабюджетної освіти, зокрема короткострокових курсів та програм другої вищої освіти, є відповіддю на виклики сучасного суспільства, яке потребує швидкого отримання актуальних знань і навичок. Ця курсова робота присвячена розробці інформаційної системи для організації навчального процесу, що забезпечує гнучкість, зручність і доступність для користувачів.

Метою роботи є створення програмного рішення, яке інтегрує ключові етапи взаємодії між користувачами та адміністраторами, включаючи управління курсами, замовленнями та користувачами. Для реалізації було використано сучасні технології програмування, зокрема C# для створення інтерфейсу та SQL Server для управління базами даних. Основними завданнями стали моделювання бізнес-логіки, розробка модулів системи та їх інтеграція.

Робота складається з кількох розділів: аналізу вимог, розробки концептуальної моделі предметної області, візуалізації бізнес-логіки, реалізації функціональних модулів та оцінки результатів. Особливу увагу було приділено надійності системи, безпеці даних та забезпеченню позитивного досвіду користувача.

Результати роботи дозволяють оптимізувати організацію навчального процесу та покращити управління освітніми послугами. Інформаційна система орієнтована на потреби студентів, які прагнуть швидко і зручно здобувати нові знання, та адміністраторів, які відповідають за формування і підтримку контенту.

Таким чином, дана курсова робота є внеском у розвиток цифрових освітніх платформ, які відповідають вимогам сучасного ринку освіти.

# **АНАЛІЗ ТЕОРІЇ КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

## **Визначення архітектурних точок зору для проектування системи на основі синтетичного підходу опису архітектури**

Визначення архітектурних точок зору забезпечує всебічне врахування потреб зацікавлених сторін, включаючи бізнес-вимоги, технічні обмеження та експлуатаційні характеристики, що є необхідними для побудови збалансованої, масштабованої та ефективної системи. Синтетичний підхід до опису архітектури об’єднує різноманітні методології, моделі та парадигми, що дозволяє отримати цілісну картину майбутньої системи та ефективно вирішувати складні міждисциплінарні завдання.

Архітектурні точки зору формуються на основі аналізу вимог та класифікації ключових аспектів системи. У цьому контексті точка зору є формальним поданням конкретної сфери інтересів, що дозволяє структуровано представляти аспекти, важливі для зацікавлених сторін. Наприклад, функціональна точка зору акцентує увагу на основних процесах, що виконує система, тоді як структурна зосереджується на компонентах системи, їх взаємозв’язках та способах інтеграції. Додатково, точка зору безпеки охоплює аспекти захисту даних і забезпечення надійності, а точка зору продуктивності — вимоги до швидкодії та масштабованості.

Синтетичний підхід до опису архітектури передбачає інтеграцію цих точок зору у єдиний каркас, що дозволяє зменшити ризики невідповідності вимог або дублювання функцій. Цей підхід базується на принципах системності, що дозволяють розглядати архітектуру як єдність взаємопов’язаних елементів. Крім того, він сприяє стандартизації процесів проєктування, спрощуючи комунікацію між різними учасниками проєкту, включаючи розробників, аналітиків та замовників.

Процес визначення архітектурних точок зору включає кілька етапів. Спочатку виконується детальний аналіз вимог до системи та ідентифікація ключових аспектів, які необхідно врахувати під час проєктування. Далі здійснюється моделювання кожної точки зору за допомогою відповідних методів, таких як діаграми класів UML, потоки даних або специфікації функціональних вимог. Завершальним етапом є інтеграція моделей у єдину архітектурну документацію, яка стає основою для подальшої розробки та впровадження системи.

Застосування синтетичного підходу забезпечує гнучкість у розробці, дає змогу швидко адаптувати архітектуру до змінних вимог та сприяє підвищенню загальної якості системи. Водночас він вимагає високої кваліфікації учасників проєкту та використання сучасних інструментів моделювання, що дозволяють ефективно управляти складністю системи. Отже, визначення архітектурних точок зору за допомогою синтетичного підходу є фундаментальним етапом, який забезпечує якісну основу для створення інноваційних інформаційних систем.

Початок формиПочаток форми

## **Поширені UML-діаграми опису модулів, логіки, компонентів, руху інформації, безпеки, розгортання тощо**

У межах сучасного підходу до проєктування інформаційних систем поширеним засобом є використання UML-діаграм, які забезпечують графічне представлення різних аспектів системи. Ці діаграми дають змогу описати структуру, логіку, компоненти, рух інформації, безпеку та інші важливі характеристики системи в уніфікованому форматі. UML, як стандарт моделювання, є основою для створення зрозумілих і формалізованих моделей, що полегшує комунікацію між розробниками, аналітиками та замовниками.

Одним із найважливіших типів UML-діаграм є діаграми класів, які використовуються для опису модульної структури системи. Вони демонструють ключові класи, їхні атрибути, методи та взаємозв’язки, такі як асоціації, наслідування чи агрегації. Цей тип діаграм особливо корисний для деталізації логіки роботи модулів, а також для визначення їхньої взаємодії в межах системи. Завдяки діаграмам класів можливо визначити ролі кожного модуля та їхній внесок у загальну функціональність.

Для моделювання поведінки системи широко використовуються діаграми активності, послідовностей та станів. Діаграми активності забезпечують візуалізацію потоку виконання процесів, що особливо важливо для розуміння логіки бізнес-операцій. Діаграми послідовностей фокусуються на взаємодії між об’єктами в часі, акцентуючи увагу на обміні повідомленнями. Діаграми станів, у свою чергу, дають змогу описати життєвий цикл окремих компонентів системи, відображаючи зміни їхніх станів залежно від виконуваних дій або отриманих подій.

Для забезпечення повноти опису компонентів системи використовуються діаграми компонентів і розгортання. Діаграми компонентів дозволяють відобразити фізичну структуру програмного забезпечення, зокрема модулі, бібліотеки чи інтерфейси, що складають систему. Діаграми розгортання демонструють, як ці компоненти розподіляються між апаратними ресурсами, визначаючи взаємозв’язки між вузлами розгортання, серверами чи іншими елементами інфраструктури.

Діаграми потоку даних та інформації забезпечують візуалізацію руху даних між різними елементами системи. Вони відображають, як інформація передається між підсистемами, модулями чи зовнішніми компонентами. Завдяки таким діаграмам можна оптимізувати процеси обробки даних, забезпечити коректний рух інформації та мінімізувати ризики втрати або дублювання даних.

Діаграми безпеки є специфічним інструментом для моделювання аспектів захисту даних і забезпечення надійності системи. Вони дозволяють виявити потенційні вразливості та визначити механізми захисту, наприклад, обмеження доступу, шифрування або автентифікацію користувачів. Такі діаграми допомагають інтегрувати заходи безпеки у загальну архітектуру системи.

Отже, UML-діаграми є невід’ємною частиною процесу моделювання інформаційних систем, забезпечуючи багатовимірний підхід до їхнього опису. Вони надають можливість не лише деталізувати структурні, функціональні та поведінкові аспекти системи, а й підвищити ефективність розробки, впровадження та підтримки програмних продуктів.Початок форми

## **Високорівневе проектування, порядок розробки вимог**

Високорівневе проєктування є початковим етапом архітектурного планування системи, що забезпечує формування загального уявлення про її структуру, функціональні можливості та основні технічні характеристики. Цей етап є критично важливим, оскільки він створює концептуальну базу для подальших процесів деталізованого проєктування та розробки. У рамках високорівневого проєктування здійснюється ідентифікація ключових компонентів системи, їх взаємодії, а також визначаються основні принципи організації інформаційних потоків і розподілу функцій між підсистемами.

Процес високорівневого проєктування починається з аналізу вимог до системи, який включає визначення як функціональних, так і нефункціональних аспектів. Функціональні вимоги стосуються специфічних завдань, які система повинна виконувати, наприклад, обробки даних, забезпечення взаємодії з користувачем або виконання бізнес-операцій. Нефункціональні вимоги охоплюють характеристики, такі як продуктивність, масштабованість, надійність, безпека та зручність використання. Ці вимоги визначаються на основі узгодження із зацікавленими сторонами, бізнес-аналізу та вивчення операційного середовища.

Після збору вимог здійснюється їхня класифікація та пріоритезація. Це дозволяє виділити критично важливі аспекти, які необхідно врахувати на ранніх етапах проєктування, та ті, які можуть бути реалізовані пізніше. Важливо, щоб вимоги були чіткими, досяжними та перевірюваними. Для цього використовується методологія SMART (Specific, Measurable, Achievable, Relevant, Time-bound), що забезпечує формалізований підхід до їх формулювання.

Наступним етапом є створення концептуальної моделі системи, яка охоплює основні підсистеми, їх функції та взаємозв’язки. Ця модель зазвичай формується у вигляді UML-діаграм, наприклад, діаграм класів, компонентів чи розгортання. Концептуальна модель служить основою для прийняття архітектурних рішень, таких як вибір технологій, платформ або методів інтеграції компонентів. Високорівневе проєктування також включає визначення архітектурних стилів, які будуть застосовуватися, наприклад, клієнт-сервер, мікросервісна або багатошарова архітектура.

Порядок розробки вимог під час високорівневого проєктування передбачає ітераційний підхід, що дозволяє поступово уточнювати та деталізувати вимоги на основі зворотного зв’язку з учасниками проєкту. Спочатку проводиться первинне визначення вимог на основі загальних цілей і контексту проєкту. Потім виконується їх перевірка та затвердження з усіма зацікавленими сторонами. У разі потреби вимоги можуть бути переглянуті або доповнені, якщо виникають нові обставини чи зміни у бізнес-середовищі.

Високорівневе проєктування завершується формуванням технічного завдання, що включає систематизований опис вимог, концептуальних моделей та архітектурних рішень. Це технічне завдання стає основним документом, який визначає напрямок розробки системи, забезпечуючи прозорість процесу та мінімізацію ризиків, пов’язаних з невідповідністю між очікуваннями та реалізацією. Таким чином, високорівневе проєктування створює основу для успішної реалізації проєкту, забезпечуючи його структурованість, логічну послідовність і відповідність потребам користувачів та бізнесу.

Початок форми

## **Ланцюжок конструювання: концептуальне-структурне-ООП**

Ланцюжок конструювання, що охоплює концептуальне, структурне та об’єктно-орієнтоване проєктування (ООП), є методологічним підходом, який забезпечує послідовний перехід від ідеї до реальної програмної системи. Кожен етап цього процесу виконує свою унікальну функцію, дозволяючи ефективно формулювати вимоги, організовувати структуру системи та забезпечувати її інтерактивність і адаптивність через принципи об’єктно-орієнтованого програмування.

Концептуальне проєктування є початковим етапом у ланцюжку конструювання, де формулюються основні ідеї системи, визначаються її призначення, функціональність і основні вимоги. У цьому контексті концептуальна модель системи є узагальненою абстракцією, яка описує її ключові аспекти без занурення в технічні деталі. Цей етап включає визначення бізнес-процесів, аналіз зацікавлених сторін і опис очікуваної поведінки системи. Використовуються такі інструменти, як діаграми потоків даних, концептуальні карти або опис сценаріїв використання, що допомагають уявити загальну ідею системи. Концептуальне проєктування закладає фундамент для структурного аналізу, формуючи базу для подальшого уточнення архітектури.

Структурне проєктування є наступним етапом, який детально моделює внутрішню організацію системи, зокрема її модулі, підсистеми та їх взаємозв’язки. Цей етап спрямований на створення чіткої архітектури, яка відображає розподіл функцій між компонентами та забезпечує логічну цілісність системи. Структурне проєктування часто використовує такі моделі, як діаграми класів UML для визначення статичної структури, діаграми послідовностей для опису взаємодії між компонентами та діаграми розгортання для відображення фізичного розташування модулів на апаратному рівні. Ця фаза допомагає вирішувати завдання масштабованості, продуктивності та надійності системи, забезпечуючи ефективну організацію її елементів.

Об’єктно-орієнтоване проєктування (ООП) завершує ланцюжок конструювання, забезпечуючи програмну реалізацію архітектури системи через об’єкти та класи. Основою цього етапу є використання принципів ООП: інкапсуляція, наслідування, поліморфізм і абстракція. Ідентифікація об’єктів відбувається на основі концептуальних моделей і структурного аналізу, після чого вони організовуються в класи із чітко визначеними атрибутами та методами. Інкапсуляція забезпечує приховування внутрішніх деталей реалізації, роблячи систему більш гнучкою та захищеною від змін. Наслідування дозволяє створювати ієрархії класів, які спрощують повторне використання коду та полегшують його підтримку. Поліморфізм забезпечує адаптивність системи, дозволяючи різним об’єктам виконувати однакові операції через інтерфейси або загальні методи.

Крім цього, об’єктно-орієнтоване проєктування сприяє інтеграції різних аспектів системи в єдину цілісну модель, забезпечуючи зв’язок між концептуальними ідеями та їхньою технічною реалізацією. Наприклад, сценарії використання, розроблені на концептуальному етапі, трансформуються в об’єкти та їхню поведінку, описану в коді. Такий підхід підвищує зрозумілість, модульність і масштабованість системи, дозволяючи ефективно реагувати на зміни вимог або умов експлуатації.

Таким чином, ланцюжок конструювання "концептуальне-структурне-ООП" забезпечує послідовність і системність у процесі створення програмних систем. Він дозволяє інтегрувати високорівневі бізнес-цілі із сучасними технічними підходами, створюючи стабільну основу для реалізації складних і масштабованих рішень.

## **Метрики чистоти та якості коду**

Метрики чистоти та якості дозволяють забезпечити ефективність коду, зрозумілість і легкість у підтримці. Високоякісний код є фундаментом успішного програмного проєкту, оскільки він не лише сприяє мінімізації помилок, але й знижує витрати на розробку та підтримку системи у довгостроковій перспективі. Використання метрик дає можливість об’єктивно оцінювати код за формальними критеріями, уникати суб’єктивності та запроваджувати стандарти якості у команді розробників.

Однією з ключових метрик є Cyclomatic Complexity (цикломатична складність), яка вимірює кількість незалежних шляхів через кодову базу. Ця метрика дозволяє оцінити складність алгоритмів і виявити ділянки коду, які є надто заплутаними або потребують рефакторингу. Оптимальний рівень цикломатичної складності забезпечує баланс між функціональністю та підтримуваністю коду.

Іншою важливою метрикою є Code Coverage (покриття коду тестами), яка вказує на відсоток коду, перевіреного автоматизованими тестами. Високий рівень покриття свідчить про те, що основні функції коду перевірені на наявність помилок, що зменшує ризики у роботі системи. Водночас, це не єдиний критерій якості, оскільки навіть 100% покриття не гарантує відсутності логічних помилок.

Метрика Maintainability Index (індекс підтримуваності) використовується для оцінки зусиль, необхідних для внесення змін до коду. Вона враховує кілька аспектів, таких як обсяг коду, складність і рівень коментарів. Чим вище значення індексу, тим легше підтримувати та модифікувати код, що особливо важливо для масштабних проєктів із великою командою розробників.

Також широко застосовуються метрики читаємості коду, такі як Average Line Length (середня довжина рядка коду) та Number of Comments (кількість коментарів). Код, що легко читається, сприяє кращому розумінню його логіки новими членами команди або зовнішніми експертами. Однак важливо забезпечувати баланс: надмірне коментування може свідчити про складний і заплутаний код, що є ознакою поганого дизайну.

Метрики повторного використання (Reusability) аналізують ступінь модульності коду, тобто наскільки легко можна повторно використовувати окремі компоненти в інших частинах проєкту або в інших проєктах. Висока модульність сприяє зменшенню дублювання коду та підвищенню його ефективності.

На рівні стилю програмування часто оцінюється консистентність (Consistency), яка визначає, чи відповідає код прийнятим стандартам і конвенціям проєкту. Метрики, такі як кількість порушень стилю (Style Violations), допомагають забезпечити однорідність коду, що підвищує його зрозумілість і легкість у супроводі.

Крім того, метрика Bug Density (щільність помилок) дозволяє визначити кількість дефектів на певну кількість рядків коду (наприклад, на 1000 рядків). Низька щільність помилок є показником високої якості програмного продукту, тоді як висока щільність сигналізує про необхідність проведення додаткового тестування або рефакторингу.

Таким чином, метрики чистоти та якості коду надають об’єктивні інструменти для моніторингу, аналізу та вдосконалення програмного забезпечення. Їх застосування сприяє побудові надійного, зрозумілого й довгостроково підтримуваного програмного продукту, що відповідає сучасним стандартам і очікуванням користувачів.

## **Тестування та налагодження – тест-план, структура звітності, TDD, рефакторинг**

Тестування та налагодження програмного забезпечення забезпечують його надійність, функціональність і відповідність вимогам. Ці процеси охоплюють створення тест-планів, проведення тестування, застосування підходу розробки через тестування (TDD), а також рефакторинг коду для підвищення його якості та підтримуваності. Взаємодія цих компонентів створює цілісну систему перевірки та вдосконалення програмного продукту.

Тест-план є документом, який описує стратегію, обсяг, ресурси та графік виконання тестування. Він визначає цілі тестування, тестові середовища, критерії входу та виходу, а також перелік тестових сценаріїв, які необхідно виконати. Тест-план забезпечує системний підхід до перевірки функціональності програмного забезпечення, дозволяючи уникнути пропуску важливих аспектів. Наприклад, у тест-плані можуть бути передбачені функціональні тести для перевірки відповідності системи бізнес-вимогам, а також нефункціональні тести, такі як тестування продуктивності, безпеки та сумісності.

Структура звітності у тестуванні включає детальний опис результатів тестування, який є основою для аналізу якості програмного забезпечення. Тестовий звіт зазвичай містить інформацію про виконані тести, знайдені дефекти, їх пріоритетність і статус, а також рекомендації щодо подальших дій. Чітко структуровані звіти допомагають зрозуміти, які аспекти системи потребують доопрацювання, та дозволяють оцінити прогрес у виправленні помилок. Крім того, звіти сприяють зворотному зв’язку між командами розробників, тестувальників і зацікавлених сторін.

Розробка через тестування (Test-Driven Development, TDD) є підходом, що інтегрує тестування безпосередньо у процес написання коду. У рамках TDD розробник спершу пише тест для конкретної функціональності, а потім створює код, який проходить цей тест. Цей підхід забезпечує раннє виявлення помилок, підвищує впевненість у правильності реалізації та сприяє створенню якісного, зрозумілого та модульного коду. TDD також сприяє створенню чіткої документації у вигляді тестових наборів, які можна використовувати для подальшого регресійного тестування.

Рефакторинг є процесом поліпшення внутрішньої структури коду без зміни його зовнішньої поведінки. Цей етап тісно пов’язаний із тестуванням, оскільки після внесення змін необхідно переконатися, що код продовжує працювати коректно. Рефакторинг спрямований на зниження складності, видалення дублікатів, покращення читабельності та забезпечення відповідності коду стандартам якості. Наприклад, заміна складних умовних конструкцій на простіші рішення або виділення повторюваного коду у функції підвищує зручність підтримки програмного продукту.

Отже, тестування й налагодження забезпечують високу якість програмного забезпечення на всіх етапах його життєвого циклу. Тест-план та структура звітності дозволяють організувати та документувати процес перевірки, тоді як підхід TDD та рефакторинг сприяють створенню модульного, зрозумілого та легко підтримуваного коду.

# **РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ**

Розробка концептуальної моделі базується на моделюванні бізнес-логіки через ключові компоненти системи. Основою моделі є користувацький досвід та взаємодія між адміністративними функціями, учасниками навчального процесу і контентом. У представленій архітектурі реалізовано управління даними через SQL-базу даних, яка забезпечує збереження інформації про користувачів, курси, замовлення та інші елементи системи.

Система складається з кількох основних модулів: реєстрації та авторизації, перегляду курсів, управління замовленнями, а також адміністративного контролю. Для забезпечення гнучкого доступу до освітніх послуг розроблено модель, де користувачі можуть реєструватися та входити до своїх акаунтів, отримуючи персоналізований доступ до навчального контенту. Реалізована функціональність авторизації перевіряє введені дані через запити до бази UserDBInfo, забезпечуючи безпечне збереження конфіденційної інформації.

Курси представлені в базі Courses, яка включає розширені атрибути для кожного курсу, такі як назва, опис, автор, тривалість, ціна та рейтинг. Ця структура дозволяє ефективно відображати навчальні пропозиції, що відповідають індивідуальним потребам користувачів. Відповідно до моделі, користувачі можуть обирати курси через графічний інтерфейс, що інтегрує таблиці CoursesDBInfo і OrdersDBInfo. Зокрема, реалізація механізму замовлення курсів забезпечує автоматичну фіксацію даних про користувача, обраний курс та дату замовлення, створюючи тим самим прозору історію взаємодії клієнта із системою.

Реалізація інтерфейсу базується на табличних елементах, що відображають деталі курсів, включаючи зображення, опис, рейтинг та інші характеристики. Для кожного замовленого курсу створюється окрема вкладка, що покращує навігацію та доступність інформації. Додаткові функції, такі як перевірка унікальності даних під час реєстрації, динамічне відображення замовлень і підтвердження дій, спрямовані на забезпечення зручності використання та запобігання помилкам.

Важливим елементом концептуальної моделі є зв'язок між таблицями бази даних через зовнішні ключі. Наприклад, таблиця замовлень Orders інтегрується з таблицями користувачів та курсів, створюючи реляційну модель, яка дозволяє здійснювати складні запити для аналітики та управління. Це сприяє підтримці бізнес-логіки позабюджетної освіти, зокрема в частині інтеграції короткострокових курсів і другої вищої освіти у єдиний освітній простір.

Таким чином, розроблена модель забезпечує ефективну інтеграцію користувачів і навчального контенту, сприяючи доступності та гнучкості освітніх послуг. Модульний підхід до розробки, підтриманий сучасними інструментами управління базами даних, забезпечує надійність, масштабованість та простоту обслуговування системи.

# **АНАЛІЗ ВИМОГ**

Аналіз вимог є наступним етапом у забезпеченні функціональності, зручності використання та відповідності потребам користувачів розроблюваної програмної системи. На основі предметної області визначено низку функціональних, нефункціональних та бізнес-вимог, які лягли в основу проектування архітектури системи.

Функціональні вимоги зосереджені на забезпеченні базових операцій для користувачів, зокрема реєстрації, авторизації, перегляду доступних курсів, додавання курсів у кошик, оформлення замовлень та перегляду історії покупок. Важливою є можливість адміністративного управління курсами, включаючи додавання, редагування та видалення навчальних матеріалів, а також аналітику щодо попиту на курси. Реалізація цих функцій базується на ефективному зв’язку між користувачами та даними, які зберігаються в реляційній базі даних.

Нефункціональні вимоги акцентують увагу на надійності, безпеці та масштабованості системи. Наприклад, використання SQL Server для управління даними забезпечує стійкість до великих обсягів інформації та підтримує складні реляційні запити. Безпека користувацьких даних досягається через механізми аутентифікації та захисту паролів. Інтерфейс системи розроблено для простоти і зручності використання, що включає логічну організацію елементів, адаптацію до різних розмірів екранів і мінімізацію кількості кроків для виконання основних операцій.

Бізнес-вимоги орієнтовані на максимальне спрощення процесу навчання для користувачів і підвищення прибутковості системи. Ключовими аспектами є доступність широкого спектра курсів, гнучке ціноутворення, автоматизація процесів управління та звітності, а також забезпечення можливості масштабування системи для розширення функціоналу або збільшення кількості користувачів. Інтеграція механізмів рейтингу та відгуків сприяє підвищенню довіри користувачів до навчального контенту.

# **ВІЗУАЛІЗАЦІЯ БІЗНЕС-ЛОГІКИ ПРЕДМЕТНОЇ ОБЛАСТІ**

## **Діаграма модулів**

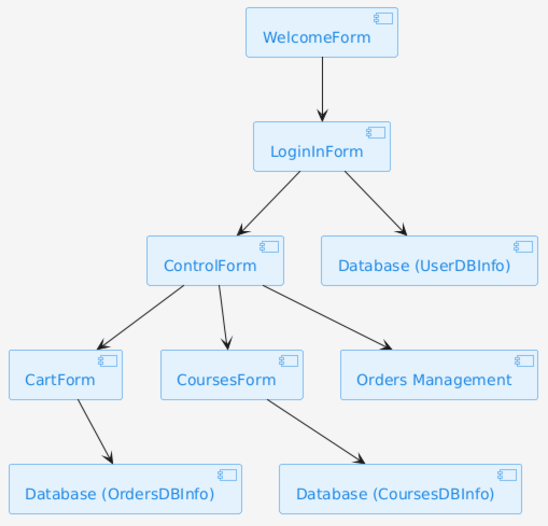


Рис. 4.1 – Діаграма модулів

Діаграма модулів відображає основні компоненти системи та їхні зв’язки. Користувач починає взаємодію з форми привітання (WelcomeForm), звідки може перейти до авторизації або реєстрації (LoginInForm). Після успішної авторизації відкривається головний модуль (ControlForm), який надає доступ до перегляду курсів, управління кошиком і замовленнями. Кожен із цих модулів пов'язаний з відповідною таблицею бази даних, забезпечуючи інтеграцію даних у реальному часі.

## **Діаграма варіантів використання**

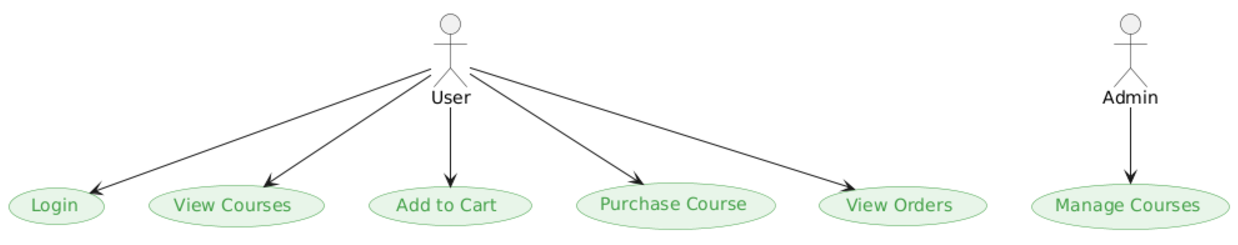


Рис. 4.2 – Діаграма варіантів використання

На діаграмі варіантів використання представлено дії, які можуть виконувати користувач і адміністратор системи. Звичайний користувач має можливість авторизуватись, переглядати курси, додавати їх у кошик, здійснювати покупку та переглядати замовлення. Адміністратор, окрім цих функцій, може управляти контентом, додаючи або редагуючи курси.

## **Діаграма взаємодії**

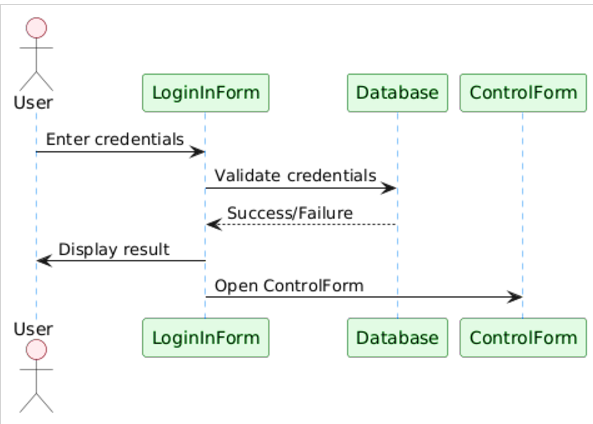


Рис. 4.3 Діаграма взаємодії

Ця діаграма демонструє взаємодію користувача з системою під час входу. Користувач вводить свої дані, які перевіряються в базі даних. У разі успіху відбувається перехід до основного вікна керування.

## **Діаграма класів**

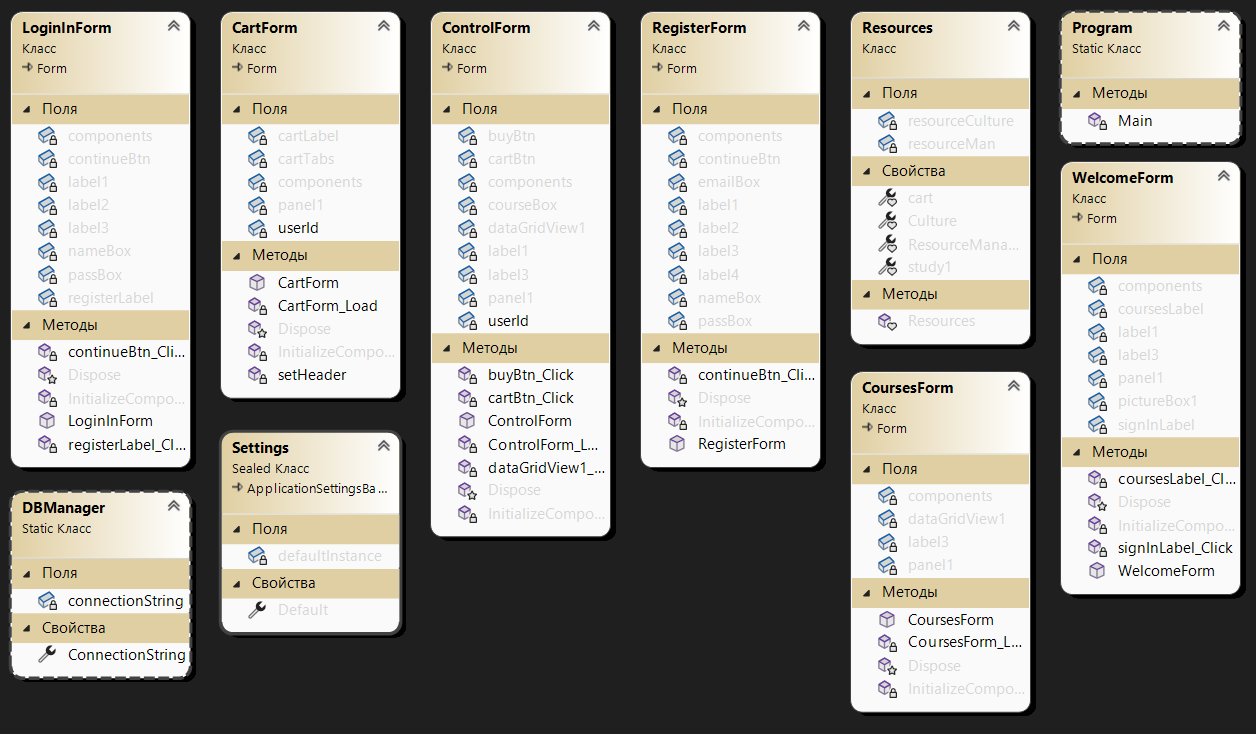


Рис. 4.4 Діаграма класів

Діаграма класів представляє основні сутності: Program, DBManager, Settings, Resources та форми CartForm, ControlForm, LoginInForm, RegisterForm, CoursesForm і WelcomeForm. Користувачі можуть мати декілька замовлень, кожне з яких пов'язане з курсом. Ця структура формує основу для реляційної бази даних у контексті Windows Forms аплікації.

## **Діаграма діяльності**

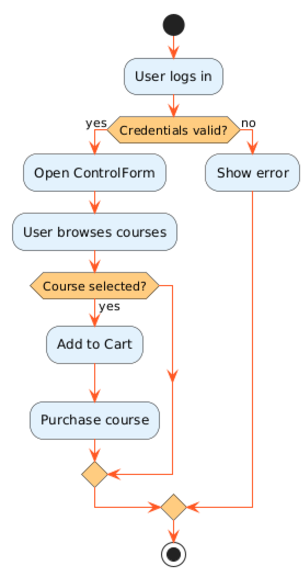


Рис. 4.5 Діаграма діяльності

Ця діаграма діяльності відображає послідовність дій користувача від входу до завершення покупки курсу. Логіка включає перевірку даних для входу, вибір курсу та виконання замовлення.

## **Діаграма станів та переходів**

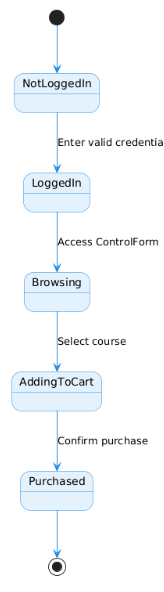


Рис. 4.6 – Діаграма станів та переходів

Діаграма станів та переходів показує стани системи для користувача: від стану без входу до покупки. Перехід між станами відбувається на основі дій користувача, що демонструє основний сценарій роботи.

## **Діаграма розгортання**

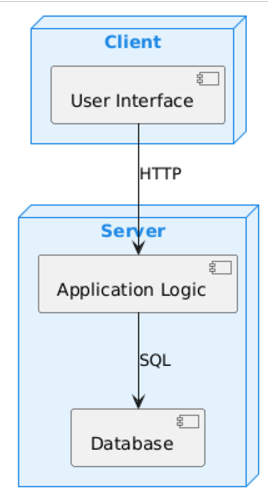


Рис. 4.7 Діаграма розгортання

Діаграма розгортання демонструє архітектуру системи. Інтерфейс користувача взаємодіє з логікою застосунку через HTTP-запити, а серверна частина отримує доступ до бази даних через SQL. Це забезпечує ефективну комунікацію між компонентами.

# **ОПИС КЛАСІВ ТА ЇХ МЕТОДІВ**

## **Конструювання коду реалізованого модуля**

Розглянемо основні етапи конструювання, принципи реалізації та функціональність модуля, його структуру та логіку реалізації програмного коду. Загальний вигляд процесу конструювання зображено на рисунку 5.1.

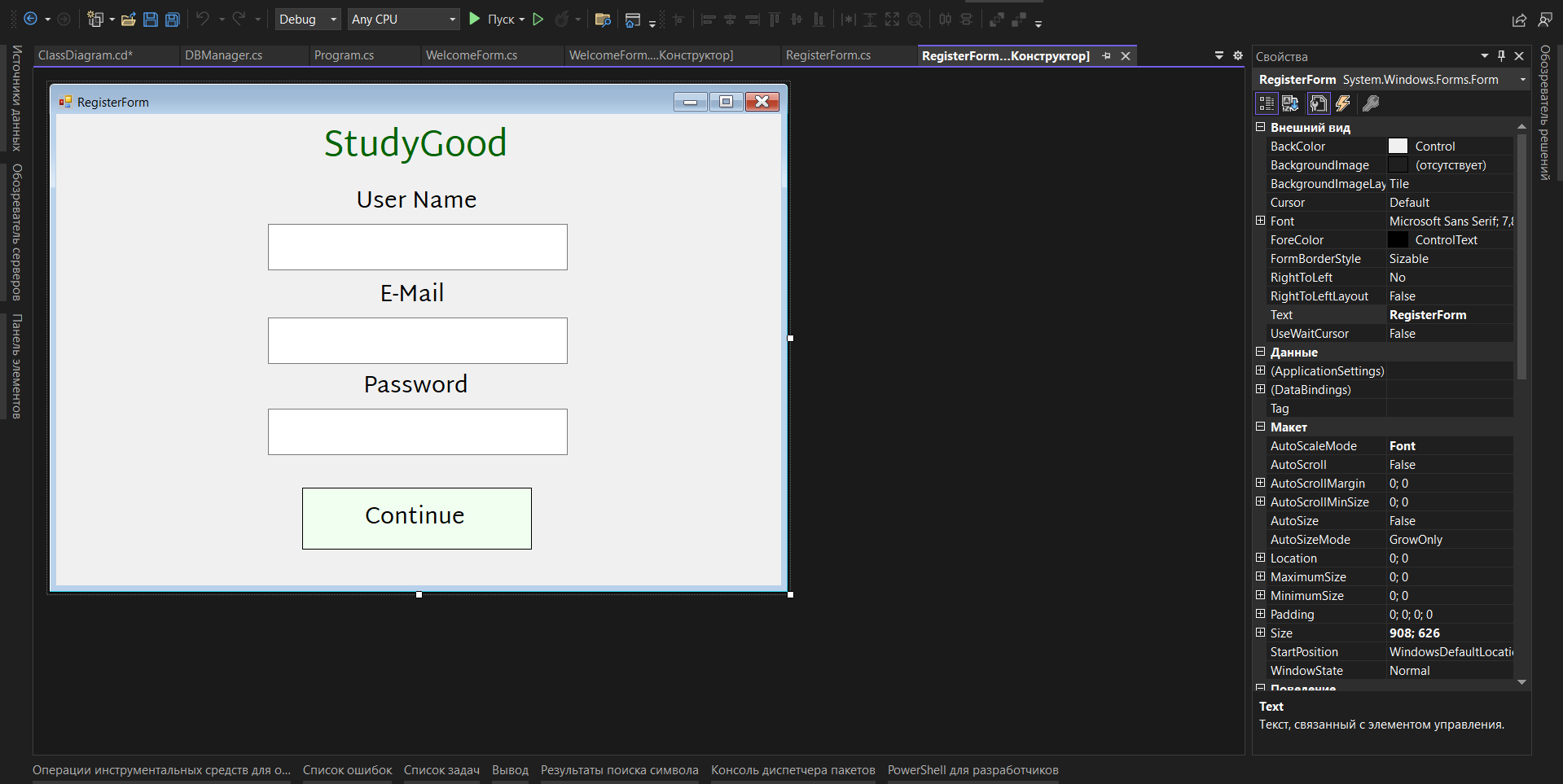


Рис. 5.1 Конструювання модуля реєстрації

Реалізований модуль є частиною інформаційної системи StudyGood, яка підтримує облік користувачів, управління курсами та здійснення замовлень на освітні послуги. Основною метою модуля є забезпечення взаємодії між користувачами та базою даних через функціональний інтерфейс, включаючи формування списків курсів, обробку замовлень, візуалізацію деталей курсів і реалізацію авторизації.

Основні класи та їх функції:

1. **CartForm**
   * Призначений для відображення інформації про курси, додані до корзини користувача.
   * Реалізує підключення до бази даних через клас SqlConnection для отримання даних про обрані курси, таких як назва, опис, автор, рік, кількість годин навчання, ціна та рейтинг.
   * Інтерфейс форми побудований із використанням елементів TabControl та TableLayoutPanel, що дозволяють зручно відображати інформацію у вкладках.
2. **ControlForm**
   * Виконує функцію головного вікна після авторизації. Дає змогу переглядати список доступних курсів і здійснювати їх купівлю.
   * Підключається до таблиці CoursesDBInfo для завантаження даних курсів у елемент DataGridView.
   * Реалізує події на клік кнопки покупки, після чого зберігає інформацію про замовлення у таблиці OrdersDBInfo.
3. **LoginInForm** та **RegisterForm**
   * Забезпечують авторизацію та реєстрацію користувачів, перевіряючи їх дані у таблиці UserDBInfo.
   * Використовують SQL-запити для перевірки наявності користувачів або додавання нових записів.

**Модуль використовує багатошарову архітектуру, в якій:**

* Дані зберігаються у реляційній базі даних, структурованій на основі трьох основних таблиць: UserDBInfo, CoursesDBInfo та OrdersDBInfo.
* Бізнес-логіка реалізована у вигляді процедурних операцій над даними, таких як виконання SQL-запитів для читання, вставки та оновлення записів.
* Інтерфейс користувача забезпечує інтуїтивне та зрозуміле відображення інформації за допомогою компонентів Windows Forms.

Структура коду дозволяє ефективно виконувати ключові операції, пов’язані з організацією позабюджетної освіти. Конструювання модулів із застосуванням чітких принципів багатошарової архітектури забезпечує розширюваність системи та полегшує її подальшу підтримку.

1. Початок форми

## **Підключені бібліотеки та фреймворки**

У цьому розділі розглядаються бібліотеки та фреймворки, які були використані при розробці програмного забезпечення для організації позабюджетної освіти. Їх вибір базується на відповідності поставленим завданням, простоті інтеграції та забезпеченні необхідного функціоналу.

Основу розробки становить фреймворк **.**NET Framework, що забезпечує інтегроване середовище для створення десктопних додатків. Завдяки широкому спектру вбудованих бібліотек, таких як System, System.Data, System.Windows.Forms, було реалізовано головні компоненти проєкту, включаючи інтерфейс користувача, доступ до бази даних і обробку подій. Наприклад, бібліотека System.Windows.Forms відповідає за створення графічного інтерфейсу (вікна, кнопки, таблиці), тоді як System.Data забезпечує роботу з реляційними даними через технології ADO.NET.

Для взаємодії з базою даних використано бібліотеку ADO.NET, яка надає класи, такі як SqlConnection, SqlCommand, SqlDataAdapter та DataTable. Ці класи забезпечують виконання SQL-запитів, отримання результатів і маніпулювання даними безпосередньо у пам’яті. Їхнє використання дозволило інтегрувати додаток з базою даних SQL Server, реалізуючи операції з користувачами, курсами та замовленнями.

Для розробки інтерфейсу застосовано Windows Forms, які надають готові компоненти для побудови інтерактивного інтерфейсу. Зокрема, такі елементи, як DataGridView, використовуються для відображення списків курсів, а TabControl – для організації вкладок із детальною інформацією про кожен курс. Крім того, компоненти PictureBox дозволяють відображати зображення, а TableLayoutPanel – структуровано розміщувати елементи у вигляді таблиці.

Для покращення користувацького досвіду було інтегровано шрифти, зокрема Martel Sans, які створюють сучасний і привабливий вигляд текстових елементів. Використання шрифтів разом із такими стилями, як FontStyle.Bold та FontStyle.Regular, забезпечує чітку візуальну ієрархію інформації.

Для завантаження та відображення зображень курсів використано компонент PictureBox, який підтримує різні режими відображення, такі як SizeMode.StretchImage. Це дозволяє адаптувати зображення до розміру контейнера, що робить інтерфейс естетично привабливим і функціональним.

Обрані бібліотеки та фреймворки дозволили забезпечити високу продуктивність, стабільність і простоту розробки. Використання стандартних інструментів .NET та їх модулів гарантує сумісність із різними операційними системами та масштабованість рішення. Це особливо важливо для довгострокового розвитку проєкту, який може бути розширений новими модулями без необхідності значного переписування існуючого коду.

Застосування цих бібліотек та фреймворків у цьому проєкті не лише вирішило поточні технічні завдання, але й забезпечило основу для майбутнього розвитку системи позабюджетної освіти.

## **Налаштування для тестового розгортання (середовища, параметри)**

Тестове розгортання програмного модуля було виконано в середовищі, що відповідає мінімальним вимогам для забезпечення коректної роботи системи та її компонентів. Основною метою тестового розгортання стало перевірка функціональності програмного забезпечення, виявлення можливих помилок та перевірка сумісності з обраною інфраструктурою.

Для налаштування тестового середовища було використано операційну систему Windows 10, яка забезпечує сумісність із технологіями .NET Framework та SQL Server. Розгортання здійснювалось на локальній машині розробника, що дозволило мінімізувати залежності від сторонніх серверів і підвищити контроль над середовищем.

База даних була розгорнута за допомогою локального інстансу Microsoft SQL Server Express. Це безкоштовна версія серверу баз даних, яка підходить для невеликих проєктів та тестування. Для підключення до бази даних використовувалась бібліотека ADO.NET, а підключення здійснювалось через файл конфігурації DBManager.cs, де вказано рядок підключення до локальної бази даних. Це дозволило забезпечити гнучкість у зміні параметрів підключення без необхідності змінювати код програми.

Для перевірки функціональності графічного інтерфейсу було використано інструменти Windows Forms, які надають можливість емулювати взаємодію користувача із програмою. Форма авторизації та реєстрації перевірялись на коректність обробки введених даних і їх валідацію перед надсиланням у базу даних. Окрему увагу приділено тестуванню модуля відображення інформації про курси та взаємодії з ними через таблиці та вкладки.

Щодо апаратних параметрів тестового середовища, використовувався комп’ютер із процесором Intel Core i5, 8 ГБ оперативної пам’яті та жорстким диском SSD на 256 ГБ. Цього було достатньо для забезпечення стабільної роботи додатку та швидкого виконання SQL-запитів.

Тестове розгортання також включало перевірку інтеграції програми із зовнішніми файлами, такими як зображення курсів. Всі необхідні файли були розміщені у відповідній директорії проєкту, а їх доступність перевірялась у процесі роботи додатку. Крім того, для тестування обробки помилок було штучно створено ситуації, такі як неправильний шлях до файлу або відсутність записів у базі даних.

Результати тестового розгортання підтвердили працездатність модуля у створеному середовищі. Виявлені дрібні недоліки, такі як некоректне відображення певних елементів інтерфейсу за умов зменшення розміру вікна, були усунені шляхом корекції налаштувань компонентів Windows Forms. Таким чином, тестове розгортання стало важливим етапом перевірки функціональності і готовності системи до експлуатації.

# **АНАЛІЗ ЯКОСТІ ТА ВІДПОВІДНОСТІ**

## **План перевірки та тестування**

План перевірки та тестування модуля передбачає систематичний підхід до забезпечення якості розробленого програмного забезпечення. Метою тестування є виявлення можливих помилок, перевірка функціональності окремих компонентів і забезпечення відповідності очікуваним результатам. Процес тестування був побудований з урахуванням основних функціональних можливостей, реалізованих у коді.

Тестування здійснювалось у кілька етапів: модульне тестування, інтеграційне тестування та тестування користувацького інтерфейсу. Нижче наведено основні аспекти кожного з них.

Модульне тестування зосереджувалося на перевірці окремих функцій і методів. Наприклад, метод setHeader() у класі CartForm тестувався для перевірки коректного отримання та відображення імені користувача з бази даних. Для цього створювались тестові записи у таблиці UserDBInfo, і виконання SQL-запиту перевірялося на відповідність очікуваному результату. Аналогічно, методи для завантаження даних курсів (CartForm\_Load) перевірялись на точність відображення інформації, включаючи обробку некоректних або відсутніх даних.

Інтеграційне тестування проводилось для перевірки взаємодії між компонентами. Зокрема, взаємодія форм ControlForm та CartForm тестувалась шляхом перевірки передачі ідентифікатора користувача між ними. Додатково тестувалась коректність збереження замовлень у таблиці OrdersDBInfo після натискання кнопки "Купити" у вікні ControlForm. Важливим аспектом було забезпечення збереження даних лише після підтвердження користувачем.

Тестування користувацького інтерфейсу включало перевірку всіх елементів Windows Forms на відповідність функціональним вимогам. Наприклад:

* Поля для вводу імені користувача та пароля у формі LoginInForm перевірялись на коректну валідацію. Тестувались сценарії введення неправильних даних, незаповнених полів та спроби входу з використанням неіснуючого облікового запису.
* Елементи керування у формі CoursesForm, такі як таблиця курсів, перевірялись на правильне відображення інформації з бази даних. Особлива увага приділялась можливості вибору курсів і передачі цієї інформації в інші компоненти програми.
* Відображення вкладок у CartForm тестувалось на предмет коректного відображення детальної інформації про курси, включаючи назву, опис, автора, рік створення, кількість годин, рейтинг та зображення.

Тестування обробки помилок включало імітацію несправностей, таких як втрата з'єднання з базою даних або відсутність потрібних файлів. Було перевірено, що програма виводить відповідні повідомлення про помилки і не призводить до некоректної роботи.

Критерієм завершення тестування було усунення всіх виявлених дефектів і підтвердження коректності роботи функціональних можливостей системи у тестовому середовищі. На основі отриманих результатів було складено звіт, який підтверджує готовність системи до використання.

Початок форми

## **Звіт з тестування із аналізом відповідності функціональним вимогам**

Звіт із тестування відображає результати перевірки розробленого програмного забезпечення, а також аналіз його відповідності функціональним вимогам. Тестування проводилося на основі розробленого плану (розділ 6.1) з акцентом на перевірку критичних функцій системи.

1. **Форма авторизації та реєстрації (LoginInForm, RegisterForm)**
   * Тестові сценарії: введення коректних даних, помилкових паролів, неіснуючих облікових записів, відсутність даних у полях.
   * Результати: всі сценарії виконані успішно. Введення некоректних даних обробляється з виведенням відповідного повідомлення. Реєстрація нового користувача додає запис у таблицю UserDBInfo, дублікати не допускаються.
   * Відповідність: відповідає вимогам до валідації та захисту даних користувача.
2. **Форма вибору та замовлення курсів (ControlForm)**
   * Тестові сценарії: відображення списку курсів, вибір курсу, підтвердження покупки, збереження замовлення.
   * Результати: список курсів коректно завантажується із бази даних. Функція покупки працює відповідно до вимог: після підтвердження курс додається до таблиці OrdersDBInfo.
   * Відповідність: повністю відповідає функціональним вимогам.
3. **Форма корзини (CartForm)**
   * Тестові сценарії: завантаження курсів, доданих користувачем до корзини, відображення детальної інформації про курси (назва, опис, автор, ціна, зображення).
   * Результати: дані про замовлення коректно завантажуються з таблиці OrdersDBInfo. Інтерфейс вкладок із використанням TabControl та TableLayoutPanel забезпечує структуроване відображення. Зображення курсів відображаються коректно за умови наявності файлів у потрібній директорії.
   * Відповідність: відповідає вимогам до візуалізації даних про курси.
4. **Взаємодія з базою даних**
   * Тестові сценарії: виконання SQL-запитів на отримання, вставку та оновлення даних у таблицях UserDBInfo, CoursesDBInfo, OrdersDBInfo. Перевірка роботи у випадку помилок, таких як втрата підключення до бази даних.
   * Результати: запити виконуються коректно. У разі помилок виводяться відповідні повідомлення, що дозволяє уникнути критичних збоїв програми.
   * Відповідність: функціонал взаємодії з базою даних повністю відповідає вимогам.

Виявлені недоліки та їх усунення:

1. **Некоректне відображення зображень при відсутності файлу**
   * Опис проблеми: при відсутності зображення програма генерує помилку.
   * Рішення: додано перевірку існування файлу перед завантаженням. У разі відсутності виводиться стандартне зображення-заповнювач.
2. **Некоректна обробка порожніх полів у формі реєстрації**
   * Опис проблеми: відсутність перевірки на заповненість полів призводила до додавання некоректних записів у базу.
   * Рішення: додано валідацію на рівні інтерфейсу користувача з обов’язковим заповненням усіх полів.

Проведене тестування підтвердило, що розроблене програмне забезпечення відповідає заявленим функціональним вимогам. Основні компоненти системи працюють коректно, забезпечуючи:

* обробку даних користувачів (авторизація, реєстрація),
* управління списком курсів (відображення, вибір, покупка),
* інтеграцію з базою даних (отримання та збереження даних).

Після усунення виявлених недоліків система показала стабільну роботу, відповідаючи вимогам до зручності, функціональності та надійності. Програмний продукт готовий до розгортання у виробничому середовищі.

## **Рев’ю коду на відповідність нефункціональним вимогам та постулатам «чистого коду»**

Рев’ю коду виконано з метою оцінки відповідності нефункціональним вимогам, таким як підтримуваність, масштабованість, ефективність та безпека, а також принципам «чистого коду». Аналіз базується на структурі, логіці та організації коду в рамках реалізованих модулів. Відповідність нефункціональним вимогам:

1. **Підтримуваність**  
   Код структуровано у вигляді класів, кожен із яких відповідає за чітко визначену частину функціональності. Наприклад, CartForm займається обробкою даних замовлень і візуалізацією інформації про курси, а ControlForm зосереджена на управлінні списком доступних курсів. Використання окремого класу DBManager для зберігання рядка підключення сприяє централізованому управлінню налаштуваннями бази даних. Такий підхід підвищує підтримуваність коду.
2. **Масштабованість**  
   Структура коду дозволяє легко додавати нові функціональні модулі, такі як нові типи курсів або додаткові звіти. Наприклад, додавання нових полів до бази даних не потребуватиме значних змін в інтерфейсі через використання елементів DataGridView та SQL-запитів із параметрами.
3. **Ефективність**  
   Використання SQL-запитів із параметрами (SqlCommand.Parameters) забезпечує швидке виконання операцій з базою даних, мінімізуючи ризик SQL-ін’єкцій. Водночас, повторне відкриття з’єднань до бази даних у деяких методах, наприклад, у CartForm\_Load, може бути оптимізоване шляхом застосування пулінгу з’єднань або зберігання з’єднання в контексті програми.
4. **Безпека**  
   Застосовано базові методи захисту даних, зокрема використання параметризованих запитів для уникнення SQL-ін’єкцій. Проте паролі користувачів у базі даних зберігаються у відкритому вигляді, що є значним недоліком. Для підвищення безпеки необхідно впровадити хешування паролів із використанням сучасних алгоритмів (наприклад, SHA-256 або bcrypt).

Відповідність постулатам «чистого коду»:

1. **Зрозумілість і читабельність**

Назви класів, методів і змінних інтуїтивно зрозумілі, наприклад, CartForm, setHeader, continueBtn\_Click. Проте в деяких місцях можна покращити зрозумілість за рахунок більш описових імен, наприклад, змінну res у методі setHeader слід перейменувати на userName, щоб краще відображати її зміст.

1. **Мінімізація дублювання коду**

У коді є приклади дублювання функціональності. Наприклад, SQL-запити для отримання курсів у різних формах (ControlForm і CoursesForm) мають схожу структуру. Це можна оптимізувати, винісши спільну логіку у окремий метод, що зменшить обсяг коду та полегшить підтримку.

1. **Розділення відповідальностей**

Код у цілому відповідає принципу SRP (Single Responsibility Principle): кожен клас має конкретну мету. Однак у деяких методах спостерігається змішування логіки, наприклад, метод CartForm\_Load відповідає як за завантаження даних із бази, так і за візуалізацію. Це можна виправити шляхом розділення на окремі приватні методи.

1. **Уникнення магічних чисел**

У коді є приклади магічних чисел, наприклад, значення ширини колонок у TableLayoutPanel. Їх слід винести у константи з описовими іменами, що покращить читабельність і спростить майбутню модифікацію.

1. **Обробка помилок**

У коді використовується конструкція try-catch для обробки помилок під час взаємодії з базою даних, що забезпечує стабільність роботи програми. Проте повідомлення про помилки, які виводяться користувачеві, можуть бути уточнені для надання більш зрозумілої інформації. Наприклад, помилка підключення до бази даних повинна вказувати на можливі причини (відсутність інтернету або неправильно налаштований сервер).

## **Точки та план рефакторингу**

Рефакторинг коду має на меті усунення виявлених недоліків, підвищення зрозумілості, підтримуваності та ефективності програмного забезпечення. Нижче описані ключові точки, які потребують покращення, та план рефакторингу для їх реалізації.

1. **Повторюваний код SQL-запитів**

У різних класах (наприклад, ControlForm, CartForm, CoursesForm) використовуються схожі SQL-запити для отримання даних із бази. Це призводить до дублювання коду та ускладнює його модифікацію.

1. **Об’єднання логіки в одному методі**

Деякі методи (наприклад, CartForm\_Load) виконують кілька функцій, таких як отримання даних із бази та формування інтерфейсу. Це ускладнює розуміння та тестування коду.

1. **Повідомлення про помилки**

Стандартні повідомлення, не надають користувачам достатньо інформації про причини помилок і можливі способи їх вирішення.

1. **Недостатня документація**

В коді бракує коментарів, які пояснюють логіку виконання ключових методів та класів.

План рефакторингу:

1. **Створення універсального класу для роботи з базою даних**
   * Реалізувати клас, наприклад, DatabaseHelper, який міститиме методи для виконання запитів (ExecuteQuery, ExecuteNonQuery, GetDataTable).
   * Винести спільні SQL-запити, такі як отримання курсів чи замовлень, у статичні методи цього класу.
   * Замість дублювання викликів у різних класах використовувати цей універсальний клас.
2. **Використання констант**
   * Розмістити константи у класі Constants для централізованого управління ними.
3. **Реалізація хешування паролів**
   * Використовувати стандартні бібліотеки .NET для хешування паролів, наприклад System.Security.Cryptography.
   * Під час реєстрації хешувати паролі перед збереженням у базу даних. Під час авторизації порівнювати хеші введеного пароля та збереженого в базі.
4. **Розділення методів на менші функції**

У методі CartForm\_Load виділити окремі методи для:

* + - завантаження даних із бази (LoadCartData),
    - створення вкладок (CreateTab),
    - обробки помилок (HandleError).

1. **Покращення повідомлень про помилки**
   * Створити централізований механізм обробки помилок із зрозумілими повідомленнями для користувачів.
2. **Додання XML-коментарів**
   * Додати XML-коментарі до класів і методів, що пояснюють їх функціональність. Це спростить розуміння коду іншими розробниками.
3. **Оптимізація завантаження даних**
   * Замість повторного відкриття підключення до бази для кожного запиту використовувати пул з’єднань.
   * Додати асинхронні методи для роботи з базою, щоб уникнути блокування основного потоку.

# **ВИСНОВОК**

У результаті виконання курсової роботи було успішно розроблено інформаційну систему, яка забезпечує організацію позабюджетної освіти, зокрема короткострокових курсів та програм другої вищої освіти. Проєкт дозволив вирішити ключові задачі з автоматизації управління навчальним процесом, включаючи роботу з користувачами, курсами та замовленнями.

Аналіз вимог дав змогу сформувати чіткі критерії для створення системи, що відповідає сучасним потребам ринку освіти. Було розроблено концептуальну модель предметної області, яка лягла в основу побудови бази даних та модулів системи. Візуалізація бізнес-логіки через UML-діаграми допомогла деталізувати процеси та визначити оптимальну архітектуру.

Реалізація функціональних модулів виконана з використанням мови програмування C# та SQL Server. Головна увага приділялася забезпеченню зручності для кінцевих користувачів, а також надійності та безпеці даних. Особливістю системи є її масштабованість, яка дозволяє легко адаптувати її для нових функціональних вимог або розширення контенту.

Розроблена система пропонує широкий функціонал для користувачів, включаючи зручну реєстрацію, перегляд курсів, управління кошиком і покупками. Адміністратори отримали інструменти для ефективного управління контентом і моніторингу діяльності користувачів.

Отримані результати свідчать про досягнення поставленої мети, а також демонструють можливості сучасних технологій у розробці освітніх платформ. Інформаційна система може бути використана як основа для створення більш комплексних рішень у сфері цифрової освіти, що робить її корисним інструментом для студентів, викладачів та адміністраторів.

# **ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ**

* 1. Fowler, M. (2004). UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison-Wesley. – Посібник зі створення UML-діаграм.
  2. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. – Посібник зі створення ефективної архітектури програмного забезпечення.
  3. Erickson, T., Kellogg, W. A. (2000). Social Translucence: Designing Social Infrastructures to Support Communication. ACM Press. – Стаття про проєктування інтерактивних інтерфейсів.
  4. Стаття "Організація позабюджетної освіти в Україні". Журнал "Освіта і Суспільство", 2023 р., випуск №12.
  5. Microsoft Documentation: SQL Server – Офіційна документація з налаштування, розробки та адміністрування SQL Server.  
     URL: <https://learn.microsoft.com/en-us/sql/>
  6. MSDN: C# Programming Guide – Ресурс із офіційними рекомендаціями щодо програмування мовою C#.  
     URL: <https://learn.microsoft.com/en-us/dotnet/csharp/>

# **ДОДАТКИ**

**Додаток А**

Міністерство освіти і науки України

Івано-Франківський національний технічний університет нафти та газу

Інститут інформаційних технологій

ЗАТВЕРДЖУЮ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_.\_\_.\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2024р.

ТЕХНІЧНЕ ЗАВДАННЯ

На розробку прикладного програмного забезпечення з дисципліни “Конструювання програмного забезпечення”

1. Область застосування — розробка інформаційних систем.
2. Основа розробки — робочий навчальний план дисципліни.
3. Мета та експлуатаційне призначення:
   1. мета – отримання практичних навичок розробки та налагодження інформаційних систем;
   2. призначення розробки — навчальна курсова робота із дисципліни “Конструювання програмного забезпечення”;
4. Джерела розробки — індивідуальне завдання курсової роботи із дисципліни, технічні рекомендації щодо проектування інформаційних систем та інші технічні матеріали для налаштування окремих компонентів програмної системи.
5. Технічні вимоги.

Кінцевий термін виконання курсової роботи «30.11» 2024 р

Початок розробки «01.09» 2024 р

Порядок контролю та прийняття.

6.1. Виконання етапів технічної та розрахункової документації курсової роботи, а також моделювання роботи інформаційної системи контролюється викладачем згідно з графіком виконання проєкту (роботи);

6.2. Прийняття роботи здійснюється комісією, затвердженою зав. кафедри згідно графіку захисту.

6.3. Коригування технічного завдання допускається з дозволу керівника проекту. Розробив студент групи ІП-22-3 Бачкур А. І.