МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»

Кафедра інженерії програмного забезпечення

КУРСОВА РОБОТА

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

з дисципліни: «Моделювання та аналіз програмного забезпечення»:

«Моделювання онлайн-платформи дистанційного навчання»

студента 4 курсу групи ІПЗ-20-2
спеціальності 121 «Інженерія програмного забезпечення»
Бубенка Олексія Вікторовича
(прізвище, ім'я та по-батькові)

Керівник: доцент кафедри ІПЗ, к.т.ц., доцент Сугоняк І.І.
Дата захисту: " ___ " _____ 20__р.
Національна шкала _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії _____ Сугоняк І.І.
(підпис) (прізвище та ініціали)
_____ Левківський В.Л.
(підпис) (прізвище та ініціали)
_____ Кравченко С.М.
(підпис) (прізвище та ініціали)
_____ Вольський Р.А.
(підпис) (прізвище та ініціали)

Житомир – 2023

АНОТАЦІЯ

Завданням курсової роботи було дослідження основних принципів та методів проєктування програмного забезпечення при розробці власного продукту у вигляді онлайн-платформи дистанційного навчання із використанням інструментальних засобів побудови моделей, що несуть характеристику програмного забезпечення та використовуються на етапах життєвого циклу.

Пояснювальна записка до курсового проєкту на тему «Моделювання онлайн-платформи дистанційного навчання» складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаних джерел та додатків.

Текстова частина викладена на сторінках друкованого тексту.

Пояснювальна записка має сторінки додатків. Список використаних джерел містить 10 найменувань і займає 1 сторінку. В роботі наведено 17 рисунків, загальний обсяг роботи – 68 сторінки.

КЛЮЧОВІ СЛОВА: НАВЧАЛЬНИЙ КУРС, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ОНЛАЙН-ПЛАТФОРМА, UML, ШАБЛОНИ ПРОЄКТУВАННЯ.

ЗМІСТ

ВСТУП

Сьогоднішні освітні заклади стикаються зі складністю навчального процесу, викликаною глобальними трансформаціями та змінами. Онлайн-платформи для дистанційного навчання в освітніх установах стають ключовим інструментом, який дозволяє адаптуватися до сучасного середовища. Вони забезпечують студентам та викладачам можливість отримувати та надавати освіту з будь-якої точки світу, дозволяючи індивідуалізувати процес навчання та сприяючи гнучкому навчанню, яке враховує потреби кожного.

Розширення онлайн-платформ у навчальних закладах створює простір для інновацій та модернізації освітнього процесу. Ці платформи відкривають доступ до різноманітних курсів, ресурсів та інструментів, що покликані полегшити навчання, стимулювати активність студентів та розвивати їхні здібності відповідно до сучасних вимог та потреб на ринку праці.

Метою курсової роботи є дослідження особливостей моделювання та методів проєктування програмного забезпечення за темою курсової роботи, проектування власного програмного забезпечення, бази даних, створення UML діаграм, генерація коду тощо.

Завданням на курсову роботу є:

- аналіз теоретичних засад моделювання програмного забезпечення;
- аналіз та опис вимог користування;
- методи модулювання функцій та поведінки системи;
- проектування об’єктної структури системи;
- фізичне моделювання програмних комплексів;
- кодогенерація із моделей

Об’єктом дослідження курсової роботи є методи моделювання та засоби проектування програмного забезпечення.

Предметом дослідження є можливості моделювання UML-діаграм за допомогою CASE-засобів для проектування програмного забезпечення, що становить основу для подальшого аналізу та дослідження.

РОЗДІЛ 1. АНАЛІЗ ВИМОГ КОРИСТУВАЧА ТА КОНЦЕПТУАЛЬНЕ МОДЕЛЮВАННЯ

1.1 Технічне завдання на розробку системи

Назва системи – онлайн-платформа дистанційного навчання «Scientia».

Призначення системи полягає в інтеграції її в освітні заклади з метою забезпечення доступу до онлайн-платформи для дистанційного навчання через глобальну мережу Інтернет. Ця платформа дозволяє викладачам створювати курси з різних предметів, завантажувати матеріали, тести та завдання для учнів. Окрім цього, вона дозволяє вчителям оцінювати успішність учнів, виставляти оцінки за завдання, вести журнал успішності та надавати конструктивний зворотний зв'язок для поліпшення навчального процесу.

Мета впровадження онлайн-платформи дистанційного навчання "Scientia" полягає у створенні максимально доступного, ефективного та інноваційного середовища для освіти, спрямованого на полегшення доступу до якісної освіти для учнів у освітніх закладах через глобальну мережу Інтернет. Ця платформа має за мету сприяти викладачам у створенні та розробці навчальних курсів, наданні учням можливості доступу до різноманітних матеріалів та інструментів навчання, а також відстеженні та оцінюванні успішності учнів, щоб покращити якість освіти та навчальний процес в цілому.

Вхідні дані включають інформацію про користувачів, навчальні матеріали, дані про успішність учнів.

Технічні вимоги:

- Апаратне забезпечення:

  Процесор: рекомендується Intel Core i3 або аналогічний від AMD.

  Оперативна пам'ять: мінімум 4 ГБ RAM.

  Жорсткий диск: достатній обсяг для зберігання програм та даних.
- Системне забезпечення:

  Операційна система: оновлені версії Windows, такі як Windows 10 або Windows 11, актуальні версії macOS, різні дистрибутиви Linux, такі як Ubuntu, Fedora, Debian тощо.

Веб-браузери: останні версії популярних браузерів, таких як Google Chrome, Mozilla Firefox, Microsoft Edge та Safari.

Функціональні характеристики системи:

- Надання постійного доступу користувачам до актуальної інформації про їхні академічні рахунки та прогрес навчання.
- Вхідні дані формуються на основі навчальних курсів та інформації про користувачів.
- Обмеження кількості спеціалізованих робочих місць - 20.
- Максимальна кількість одночасно працюючих користувачів - 10000.
- Орієнтовна кількість записів у базі даних - 100000000.
- Функціонал звітності, включаючи рахунок за поточний період з даними про динаміку витрат та роздруківку дзвінків за період.
- Вимоги до документації — інструкція користувача, інтерактивна довідка, керівництво по встановленню та конфігурації.
- Перелік звітних форм — статистичні дані про успшність учнів.

Якісні характеристики роботи системи:

- Час, необхідний для дослідженя можливостей системи та адаптації в ній – не більше 12 годин.
- Час відгуку для типових задач - не більше 5 секунд, для складних завдань - не більше 20 секунд.
- Час отримання відповіді від адміністратора на скаргу або звернення – не більше 7 днів.
- Доступність - час, що витрачається на обслуговування системи, не повинно перевищувати 3% від загального часу роботи.
- Середній час безвідмовної роботи - 15 робочих днів.
- Максимальна норма помилок або дефектів - 1 помилка на двадцять тисяч запитів користувачів.

1.2 Обґрунтування вибору засобів моделювання

У сучасному світі програмного забезпечення для моделювання UML-діаграм існує безліч, що надають різноманітні можливості та інструменти для розробки програмного забезпечення та систем. У виборі оптимального CASE-сервісу для моделювання UML-діаграм важливо провести ретельне порівняння доступних програмних засобів, враховуючи їхні можливості, переваги та недоліки. Наступне дослідження дозволить обрати найбільш відповідний інструмент, який найкраще відповідатиме потребам проєкту та сприятиме успішному моделюванню системи.

1. StarUML

StarUML — це безкоштовна платформа моделювання UML з відкритим вихідним кодом. Серед переваг — підтримка стандартів UML 2.0 та MDA, інтуїтивний інтерфейс, можливість розширення через плагіни, генерація коду на різні мови програмування та експорт документації у різноманітні формати. Проте є недоліки: відсутність можливості командної роботи, відсутня офіційна документація та відсутність відкритого коду у нових версіях, що може ускладнити створення власних розширень.

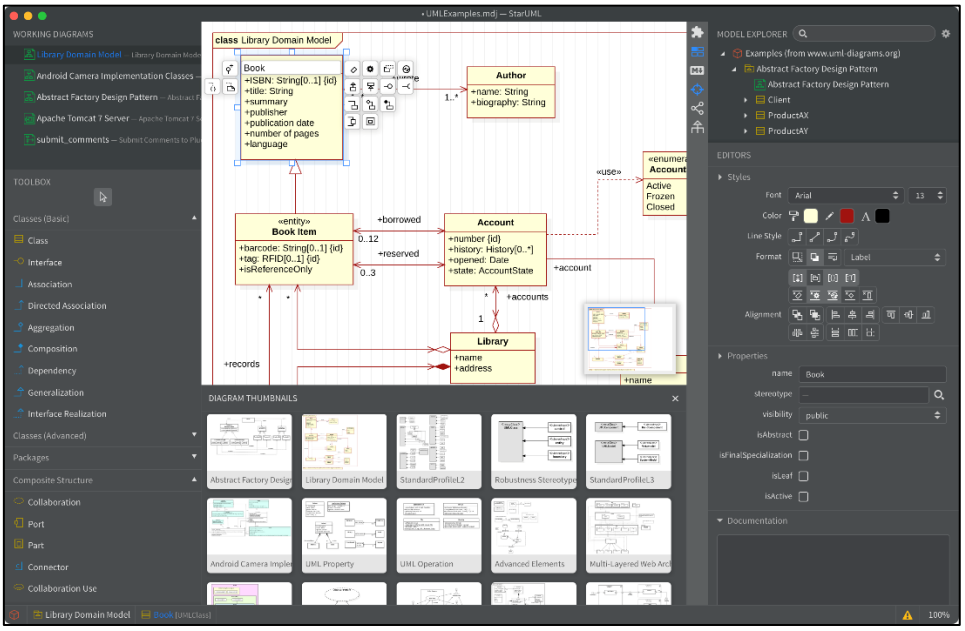


Рис. 1.1. Інтерфейс StarUML

2. Draw.io

Draw.io — це веб-платформа для створення діаграм з безкоштовним доступом та відкритим вихідним кодом. Її основна перевага полягає у зручності та доступності онлайн, що дозволяє користувачам створювати графічні представлення без потреби встановлення програм або залежності від певної операційної системи.

Платформа відкрита для всіх типів користувачів - від початківців до досвідчених фахівців. Вона підтримує широкий спектр діаграм. Окрім цього, вона надає можливість експорту діаграм у різноманітні формати (PNG, JPEG, PDF тощо).

Однією з ключових переваг є те, що платформа є безкоштовною та має відкритий вихідний код. Це сприяє активному розвитку спільноти та можливості розробки власних розширень чи плагінів. Проте, серед недоліків можна виділити обмежену можливість роботи в офлайн-режимі та деякі обмежені можливості для роботи з векторною графікою, порівняно з іншими програмами. В цілому, Draw.io є зручним та потужним інструментом для створення різноманітних діаграм, але може мати обмеження у використанні в офлайн-середовищі.

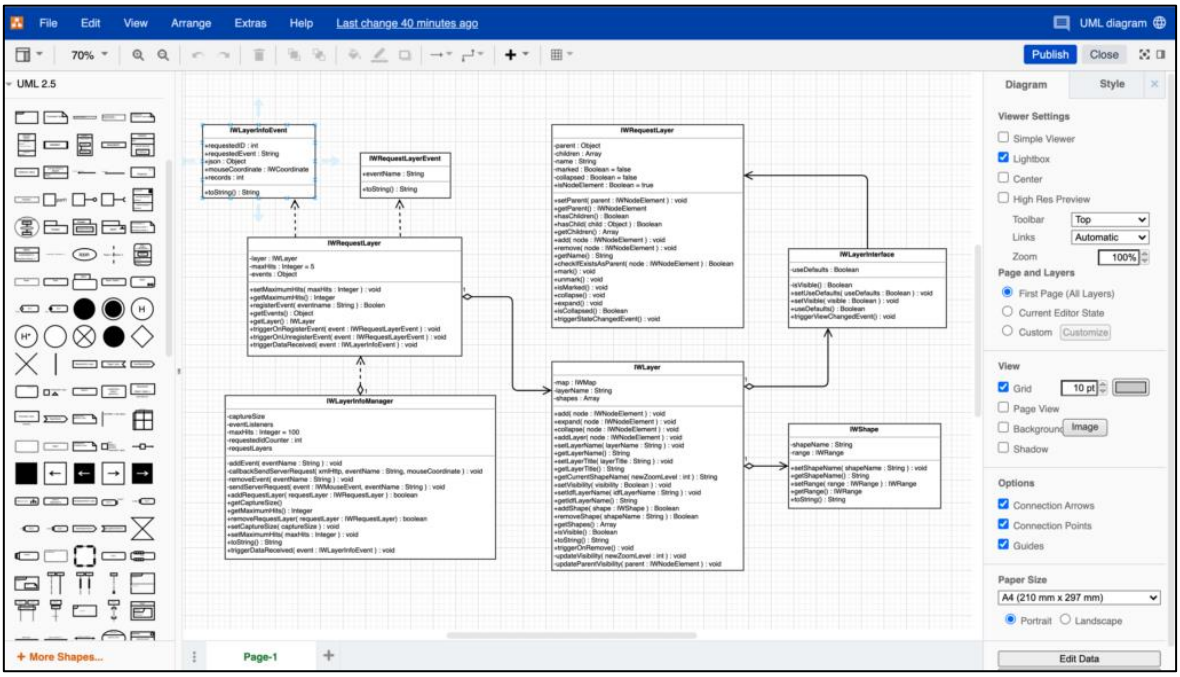


Рис. 1.2. Інтерфейс Draw.io

3. Lucidchart

Lucidchart — це онлайн-інструмент для створення діаграм, включаючи UML. Він має зручний інтерфейс та багато шаблонів для різних типів діаграм. Використання доступне онлайн з будь-якого пристрою. Особливість полягає в можливості спільної роботи над діаграмами у реальному часі. Проте, безкоштовна версія має обмежені функції, можливо потрібна підписка для доступу до додаткового функціоналу. Lucidchart є зручним інструментом для створення різних діаграм, особливо для спільної роботи над ними в команді.

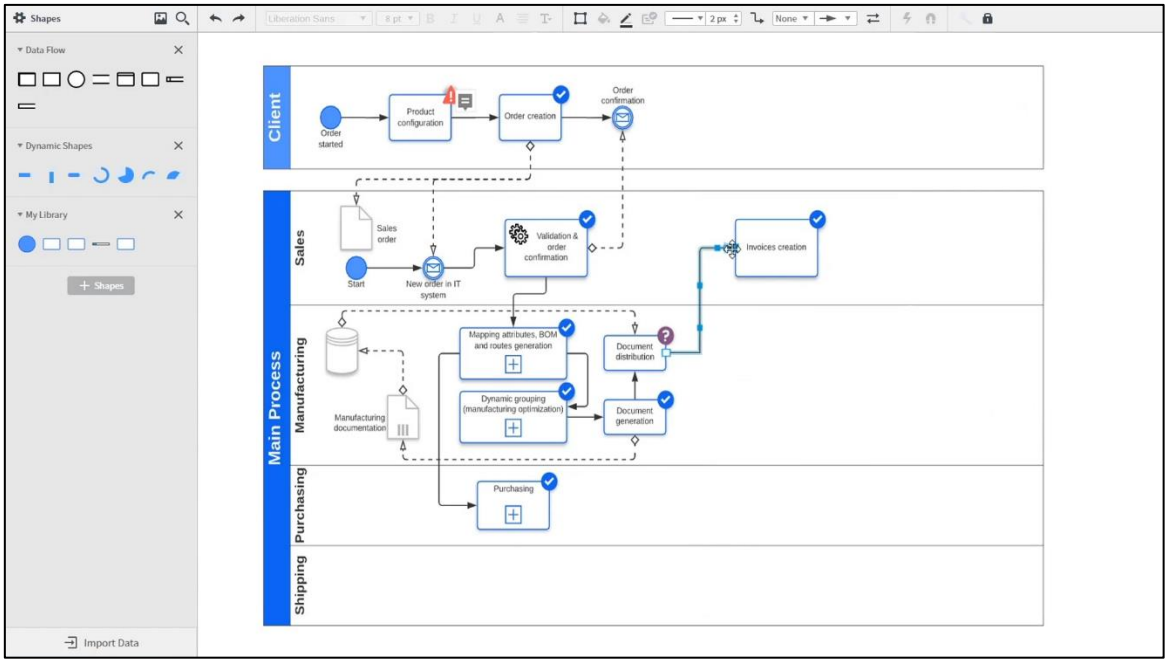


Рис. 1.3. Інтерфейс Lucidchart

Для визначення найкращого CASE-сервісу було сформовано порівняльну таблицю.

Таблиця 1.1.

Характеристика існуючих CASE-сервісів

Характеристика	StarUML	Draw.io	Lucidchart
Доступність	Задовільно	Відмінно	Відмінно
Інтерфейс	Задовільно	Відмінно	Добре
Складні діаграми	Задовільно	Добре	Добре
Шаблони	Задовільно	Відмінно	Відмінно

Продовження таблиці 1.1.

Експорт	Добре	Відмінно	Відмінно
Кодогенерація	Добре	Не задовільно	Не задовільно
Сумісність	Задовільно	Відмінно	Добре
Продуктивність	Добре	Добре	Добре

Після детального порівняння різних параметрів інструментів для побудови діаграм, було вирішено використовувати Draw.io для створення діаграм, оскільки він найбільш відповідає нашим потребам щодо зручності, функціональності та сумісності. Під час вибору Draw.io також було враховано попередній досвід використання цього інструменту, що вплинув на позитивне рішення про його використання в подальшій роботі. Однак для проведення кодогенерації було вирішено використовувати StarUML, таким чином, використовуючи переваги обох інструментів для оптимізації процесу моделювання.**Аналіз вимог до програмного продукту**

Для забезпечення безперервного виконання необхідних функцій і відповідності встановленим стандартам та специфікаціям, наведено високорівневі вимоги, які повинна задовольняти веб-орієнтована онлайн-платформа для дистанційного навчання. Ці вимоги допомагають забезпечити безперебійну роботу системи та виконання всіх необхідних стандартів і специфікацій.

Бізнес вимоги

1. Основні цілі: проект створюється з метою спростити взаємодію між вчителем та учнями в умовах дистанційного навчання, насамперед для закладів середньої освіти.
2. Представлення проекту: проект буде реалізовано у вигляді веб-платформи, яка надає користувачам доступ до перегляду та створення навчального контенту та інших інструментів для ефективного дистанційного навчання.

Платформа буде містити структуровану інформацію про навчальні курси та надавати різноманітні можливості для інтерактивного навчання, включаючи відеоуроки, завдання, тестування та інші функції, спрямовані на найкраще засвоєння навчального матеріалу. Крім того, платформа надасть можливість вчителям та учням взаємодіяти, зокрема відстежувати прогрес у навчанні.

Вимоги користувачів

1. Створення навчальних тестів та отримання вичерпної аналітики за їх результатами.
2. Розміщення оголошень на навчальних курсах, сповіщаючи учасників про важливу інформацію заздалегідь.
3. Створення та редагування плану занять курсу за допомогою спеціального конструктора.
4. Моніторинг та аналіз результатів навчання.
5. Можливість ведення електронної книги наказів.
6. Отримання сповіщень про основні події на навчальних курсах.
7. Перегляд персоналізованого розкладу занять.

Вимоги адміністратора сервісу

1. Можливість обробляти запити навчальних закладів на приєднання до платформи або від’єднання від платформи.
2. Можливість надання зворотнього зв'язку для користувачів

Характеристика об’єкта комп’ютеризації

Користувач сервісу буде мати можливість створення та ведення курсів, створення оголошень та редагування плану курсу, оцінювання робіт та можливість виставляти відмітки щодо відвідуваності. Користувачі сервісу отримуватимуть сповіщення про основні події на навчальних курсах, наприклад, сповіщення про те, що завдання треба виконати до певної дати. Також у користувачів буде можливість перегляду зведеної статистики щодо успішності навчання.

Функціональні вимоги

1. Авторизація у сервісі: сервіс повинен надавати можливість користувачам входити в систему та визначати їхню роль — учень, вчитель, керівник навчального закладу тощо.
2. Приєднання до платформи та від'єднання від неї: сервіс повинен забезпечувати можливістю перегляду та обробки запитів від навчальних закладів, які виражають бажання приєднатись до платформи. Також сервіс має надавати можливість розглядати запити навчальних закладів про від'єднання від платформи.
3. Ведення навчального курсу: сервіс повинен надавати користувачам можливість вносити зміни в інформацію щодо плану курсу, включаючи дати проведення занять, методи оцінювання завдань тощо. А також у користувача повинна бути можливість створення оголошень на курсі.
4. Виконання завдань: сервіс повинен надавати можливість користувачам завантажувати виконані роботи через відповідні форми та після їх перевірки користувачу сервіс повинен надсилати відповідні сповіщення.

Нефункціональні вимоги

1. Сприйняття
   - час, необхідний для адаптації та дослідження всіх можливостей сервісі – не більше 8 годин.
   - час відгуку для типових задач - не більше 5 секунд, для складних завдань - не більше 20 секунд.
   - інтерфейс має бути простим та інтуїтивно зрозумілим.
2. **Надійність**
   - доступність - час, що витрачається на обслуговування системи не повинно перевищувати 3% від загального часу роботи.
   - середній час безвідмовної роботи -15 робочих днів.

- максимальна норма помилок або дефектів - 1 помилка на двадцять тисяч запитів користувачів.

3. **Продуктивність**
   - система повинна підтримувати мінімум 5000 одночасно працюючих користувачів, пов'язаних із загальною базою даних.
4. **Можливість експлуатації**
   - система повинна бути готовою до масштабування, щоб забезпечити ефективну роботу навіть при зростанні кількості користувачів. У разі потреби, система повинна мати можливість легко розширювати ресурси, без негативного впливу на її функціональність та продуктивність.

Аналіз вимог користувачів дав підстави для визначення варіантів використання додатку планування розваг та меню готелів. На рис. 1.4 наведено діаграму використання програмної системи.

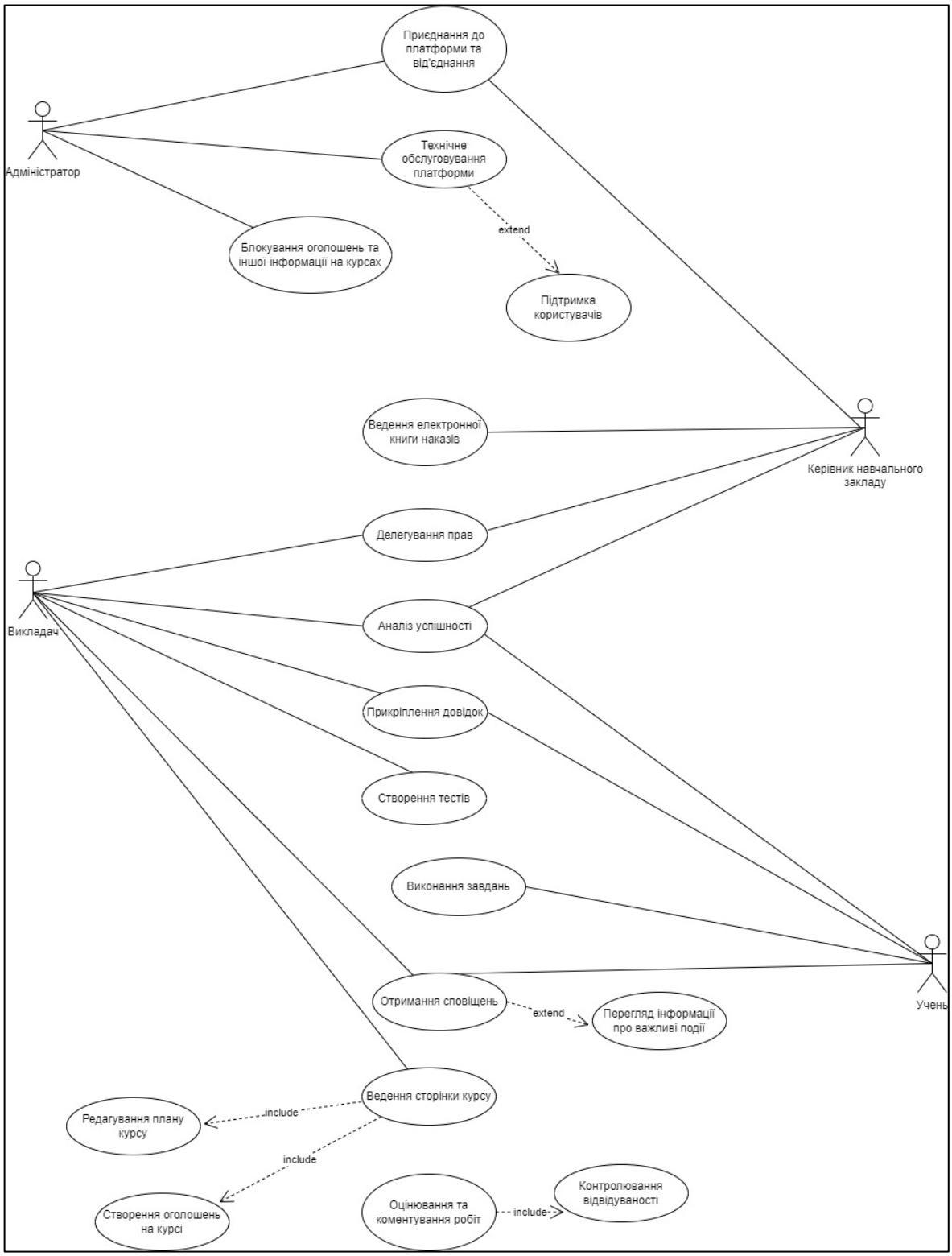


Рис. 1.4. Варіанти використання сервісу

Висновки до першого розділу

Під час виконання даного розділу, було проаналізовано поставлене завдання, визначено вимоги для моделювання. Також детально розглянуто різні CASE-інструменти та обрано той, який найбільше підходить для наших потреб. У результаті цього аналізу було систематично проаналізовано всі види вимог до системи та створено діаграму використання.

РОЗДІЛ 2. РОЗРОБКА МОДЕЛІ ПРОГРАМНОГО КОМПЛЕКСУ НА ЛОГІЧНОМУ РІВНІ

2.1 Алгоритм роботи та стани програмної системи

Ефективним інструментом для візуалізації послідовності дій та станів програмної системи є діаграма активності. Ця діаграма дозволяє уявно зобразити робочі процеси, логіку прийняття рішень, послідовність виконання операцій, виокремити вузли, цикли та паралельні дії, відображаючи суть та логіку певного процесу чи функціональної системи.

Така діаграма будується з певної обмеженої кількості фігур, які з’єднані.

Кожен елемент на такій діаграмі має своє позначення:

- Початковий вузол діяльності — початок потоку виконання діяльності, позначений кругом.
- Контроль потоку — вказує послідовність виконання між діями, зображений стрілкою.
- Дія — конкретна дія, яка виконується в системі, представлена заокругленим прямокутником.
- Вузол-рішення — визначає напрямок потоку в залежності від умови, позначений ромбом з розгалуженням.
- Вузол-злиття — об'єднує різні шляхи виконання, позначений ромбом після рішення.
- Вузол-розгалуження — розділяє процес на паралельні потоки без умови, зображений зафарбованим прямокутником зі стрілками.
- Об'єднуючий вузол — об'єднує паралельні потоки, позначений зафарбованим прямокутником зі стрілками.
- Фінальний вузол діяльності — завершує діаграму, позначений кругом з контуром.
- Доріжка — групує дії, які виконує один актор або групує дії в одному потоці.

Було побудовано діаграми створення та виконання практичного завдання на навчальному курсі та взаємодії вчителя з конструктором плану занять.

Діаграма «Створення та виконання практичного завдання на навчальному курсі» відображає послідовність дій вчителя та учня при роботі з практичним завданням.

Пояснення:

- Початковий вузол діяльності: момент початку взаємодії з платформою
- Дія: вибір навчального курсу
- Дія: створення нового завдання
- Вузол-рішення: чи є потреба прикріпляти файли для завдання?
    - Так:
        - Дія: прикріплення необхідних файлів до завдання
        - Дія: публікація завдання
    - Ні:
        - Дія: публікація завдання
- Дія: сповіщення учня про те, що на навчальному курсі з’явилось нове завдання
- Дія: перегляд завдання учнем
- Дія: завантаження виконаного завдання
- Вузол-рішення: виконане завдання потребує завантаження файлів?
    - Так:
        - Дія: завантаження необхідних файлів до виконаного завдання
        - Дія: підтвердження завантаження виконаного завдання
    - Ні:
        - Дія: підтвердження завантаження виконаного завдання
- Дія: сповіщення про те, що учень завантажив завдання
- Дія: перегляд виконаного учнем завдання та його перевірка

- Вузол-рішення: виконане завдання потрібно доопрацювати?
    - Так:
        - Дія: сповіщення про те, що роботу потрібно доопрацювати
    - Ні:
        - Дія: виставлення оцінки за роботу
- Дія: сповіщення учня про те, що роботу було оцінено
- Дія: перегляд виставленої оцінки вчителем
- Фінальний вузол

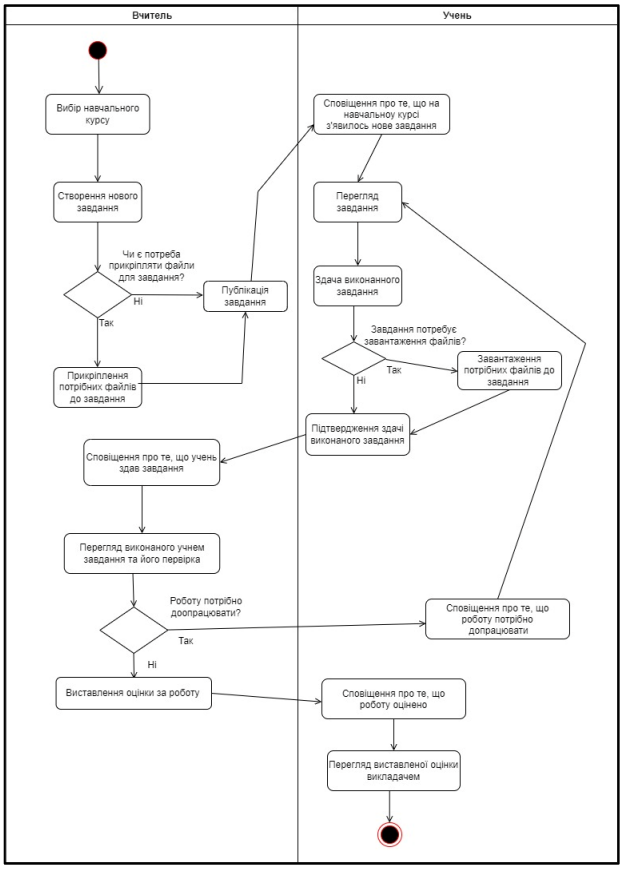


Рис. 2.1. Діаграма активностей створення та виконання практичного завдання на навчальному курсі

Діаграма «Взаємодія вчителя з конструктором плану занять» відображає послідовність дій вчителя в процесі роботи над планом занять навчального курсу.

Пояснення:

- Початковий вузол діяльності: момент початку взаємодії з платформою
- Дія: перехід до сторінки потрібного навчального курсу
- Дія: перехід до конструктора занять обраного навчального курсу
- Вузол-розгалуження
    - Дія: додавання заняття
        - Дія: модальне вікно з відображення полів заняття
        - Дія: заповнення полів заняття (тема, дата, домашнє завдання)
        - Дія: додавання форми оцінювання заняття
        - Вузол-рішення: додати ще одну форму оцінювання заняття?
            - Так:
                - Дія: додавання форми оцінювання заняття
                - Дія: підтвердження додавання заняття до списку занять
            - Ні:
                - Дія: підтвердження додавання заняття до списку занять
    - Дія: редагування заняття
        - Дія: модальне вікно з відображенням полів заняття та поточних значень
        - Вузол-розгалуження
            - Зміна теми заняття
            - Зміна дати заняття
            - Зміна форми оцінюваня заняття
            - Вузол-рішення: додати ще одну форму оцінювання заняття?
                - Так:

        - Дія: додавання форми оцінювання заняття
    - Зміна домашнього завдання до заняття
- Дія: видалення заняття
    - Дія: модальне вікно з підтвердженням видалення заняття
    - Дія: підтвердження видалення заняття
- Вузол-злиття
- Відображення переліку занять курсу
- Фінальний вузол

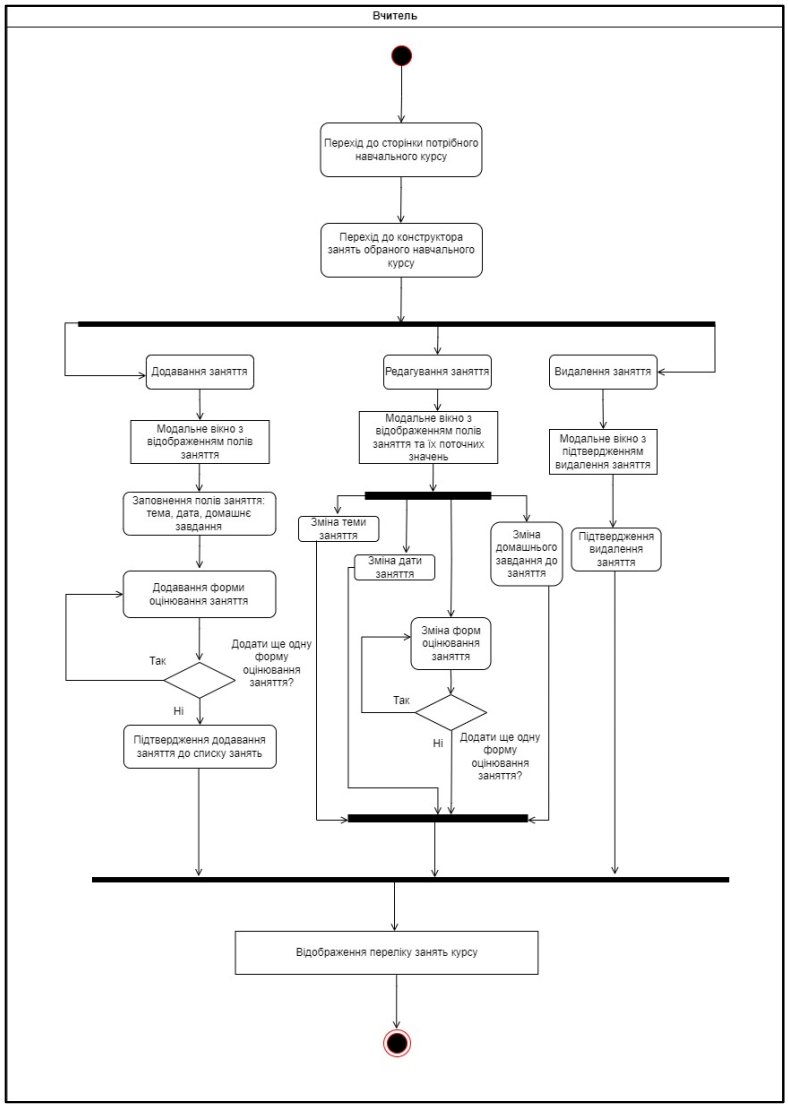


Рис. 2.2. Діаграма активностей взаємодії вчителя з конструктором плану занять

2.2 Взаємодія об’єктів системи

Об’єктно орієнтована модель системи зображується у вигляді діаграми класів. Діаграма класів показує структуру системи шляхом зображення класів, інтерфейсів, перерахувань, атрибутів, методів та зв’язків між об’єктами діаграми.

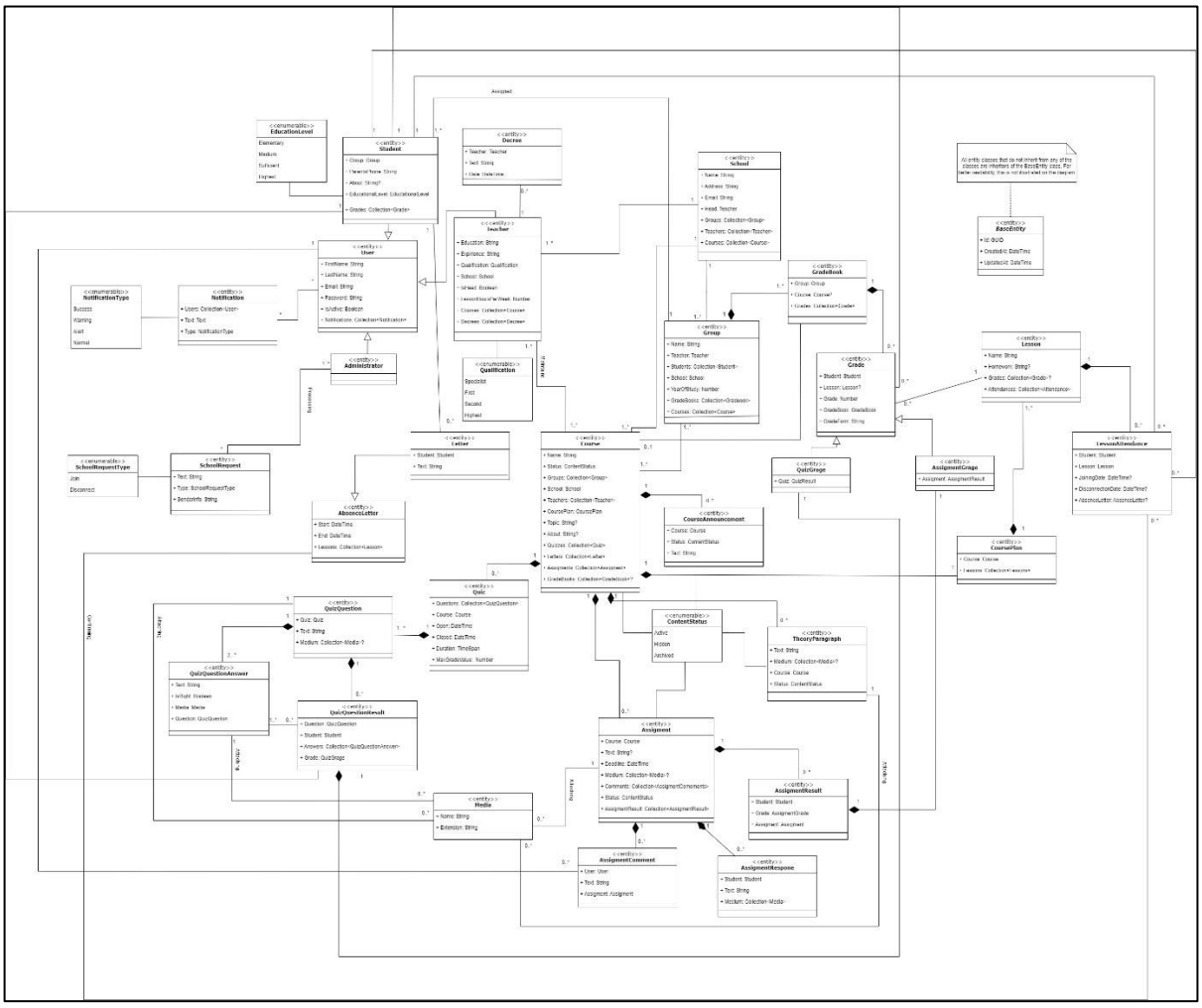
На діаграмі класів класи визначають основні структурні одиниці системи, включаючи атрибути та методи об'єктів. Інтерфейси слугують для визначення контрактів, забезпечуючи взаємодію між класами, а перерахування використовується для обмеження можливих значень, що дозволяє зручно та зрозуміло визначати константи для об'єктів чи параметрів у системі. Ці елементи спільно моделюють структуру та взаємодії в програмній системі, полегшуючи розуміння та управління системою.

Для вказівки доступності полів або методів у діаграмі класів використовуються спеціальні символи: + (публічний), - (приватний), # (захищений).

Взаємодія класів та інших об’єктів діаграми класів між собою зображується за допомогою наступних зв’язків:

- Наслідування: один клас успадковує властивості та методи іншого, батьківскього, класу.
- Асоціація: зв'язок, коли об'єкти одного класу пов'язані з об'єктами іншого класу.
- Агрегація: зв’язок, коли один клас може містити об'єкти іншого класу, які можуть існувати незалежно.
- Композиція: зв’язок, коли один клас є невід'ємною частиною іншого класу і не може існувати окремо від нього.
- Реалізація: зв’язок, коли клас забезпечує реалізацію всіх методів, описаних у інтерфейсі, який він реалізує.

На основі проаналізованих даних було побудовано діаграму класів для відображення взаємозв'язків між класами та іншими об’єктами системи.



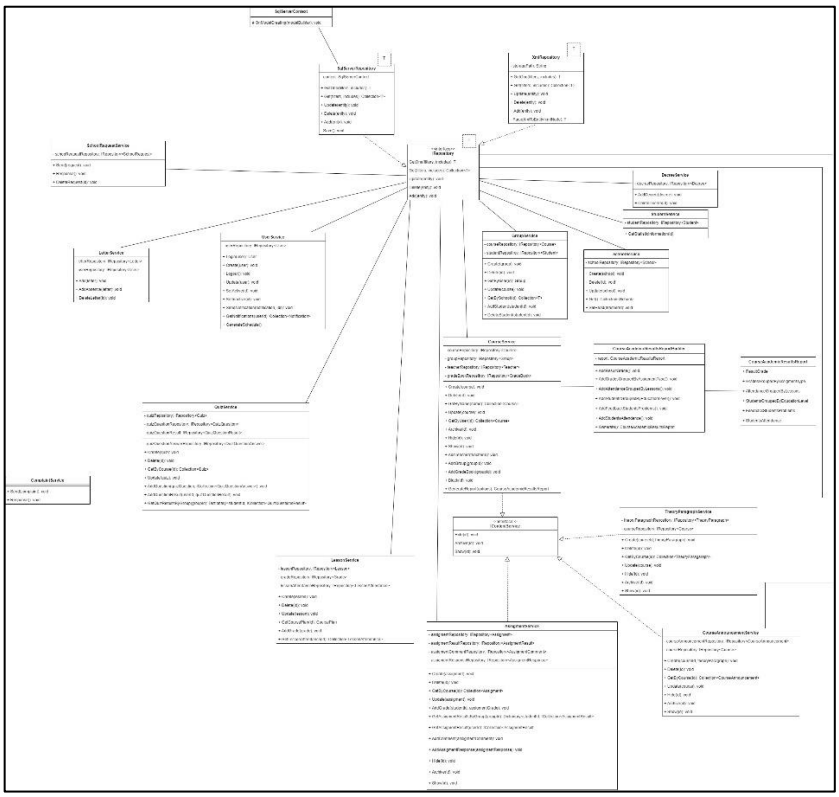


Рис. 2.3 – 2.4. Діаграма класів

Всі класи можна поділити на два типи: класи моделей та класи сервісів, які виконують операції з моделями.

Класи моделей — це ключові компоненти програмного коду, що відповідають за відображення та організацію даних у системі. Вони також представляють структури та об'єкти, які відображають таблиці баз даних.

Опис класів моделей:

- *BaseEntity*: це абстрактний клас моделі, який унаслідують всі інші класи, він містить ті поля, які повинні всі моделі системи,
- *User, Teacher, Student, Administrator*: класи, які представляють сутності основних учасників системи: викладачів, студентів та адміністраторів відповідно, загальним для учасників системи є клас User, який в свою чергу наслідується від BaseEntity,
- *Notification*: описує сутність сповіщення, наслідується від BaseEntity,
- *Decree*: описує сутність наказу у книзі наказів школи, наслідується від BaseEntity,
- *SchoolRequest*: описує сутність запиту на приєднання до платформи, наслідується від BaseEntity,
- *School*: описує сутність школи, наслідується від BaseEntity
- *Group*: описує сутність навчальної групи (класу), наслідується від BaseEntity,
- *Course*: описує сутність навчального курсу, наслідується від BaseEntity,
- *Lesson*: описує сутність уроку, наслідується від BaseEntity,
- *LessonAttendance*: описує сутність відсутності на уроці, наслідується від BaseEntity,
- *Grade*: описує сутність оцінки учня, наслідується від BaseEntity,
- *GradeBook*: описує сутність журналу оцінок, наслідується від BaseEntity,
- *Grade, AssigmentGrade, QuizGrade*: клас Grade представляє загальну оцінку студента за курс, наслідується від BaseEntity, клас

AssigmentGrade — оцінка за конкретне практичне завдання, а клас QuizGrade — оцінки за тестування, останні наслідуються від Grade,

- *Letter, AbsenceLetter: клас Letter* — відображає загальний лист від імені студента, наслідується від BaseEntity, тоді як AbsenceLetter — це конкретний тип листа, який повідомляє про відсутність чи відпустку.
- *Quiz, QuizQuestion, QuizQuestionAnswer, QuizQuestionResult*: ці сутності представляють окремі аспекти структури тестової системи: клас Quiz представляє набір питань для тестування, клас QuizQuestion представляє конкретне питання в цьому тесті, клас QuizQuestionAnswer представляє варіант відповіді на питання, клас QuizQuestionResult представляє результат за певне питання, всі наслідуються від BaseEntity,
- *Assigment, AssigmentResult*: клас Assigment представляє конкретне завдання, яке студент повинен виконати, а клас AssigmentResult представляє інформацію про результати виконання цього завдання певним студентом, включаючи оцінку та додаткові деталі, всі наслідуються від BaseEntity,
- *Media*: описує сутність мультимедійного файлу, наслідується від BaseEntity,
- *CoursePlan, CourseAnnouncement, TheoryParagraph*: клас CoursePlan описує сутність навчального плану курсу, клас CourseAnnouncement описує сутність оголошення на навчальному курсі, клас TheoryParagraph описує сутність для теоретичної статті на навчальному курсі, всі наслідуються від BaseEntity.

Класи сервісів описують функціонал, пов'язаний з бізнес-логікою програми. Ці класи включають методи для взаємодії з моделями даних, виконання операцій над ними, а також управління різними аспектами функціонування застосунку.

Опис класів та інтерфейсів сервісів:

- *LetterService*: надає функціонал для керування операціями над листами
- *ShoolRequestService*: надає функціонал для керування операціями над запитами приєднання до платформи,
- *QuizService*: надає функціонал для керування операціями над тестами,
- *LessonService*: надає функціонал для керування операціями над уроками,
- *GroupService:* надає функціонал для керування операціями над групами (класами),
- *StudentService:* надає функціонал для керування операціями над учнями,
- *DecreeService:* надає функціонал для керування операціями над наказами у книзі наказів школи,
- *SchoolService:* надає функціонал для керування операціями над школам.
- *IContentService:* представляє інтерфейс, що має методи для приховування, архівування та відображення елементів.
- *TheoryParagraphService:* надає функціонал для керування операціями над теоретичною статтею навчального курсу, реалізує інтерфейс IContentService
- *ComplaintService:* надає функціонал для зворотнього зв'язку з користувачами,
- *CourseService:* надає функціонал для керування операціями над навчальними курсами, реалізує інтерфейс IContentService,
- *AssigmentService:* надає функціонал для керування операціями над практичними завданнями, реалізує інтерфейс IContentService,
- *CourseAnnouncementService:* надає функціонал для керування операціями над оголошеннями на навчальних курсах, реалізує інтерфейс IContentService.

Також на діаграмі класів були використані перерахування для створення колекції іменованих констант, які представляють фіксований набір значень.

При проектуванні важливо врахувати можливі труднощі, з якими програма може зіткнутися. Однією з таких труднощів є необхідність розширення або зміни логіки програми, що може вимагати значних зусиль та часу для вирішення. Для таких ситуацій були створені певні шаблони проєктування, які надають рекомендації з розв'язання типових проблем.

Розглянувши об’єкт звіту про успішність, можна дійти висновку, що він є доволі складним та комплексним, а саме включає в себе кінцеву оцінку учнів, згруповані оцінки по типу оцінювання, згруповану відвідуваність по заняттям, відгук вчителя про проблемні теми учнів, відсоток відвіданих занять учнями. Однак кінцевому користувачу не обов’язково необхідна інформація про всі ці параметри, його можуть цікавити лише деякі з них.

Для вирішення цієї задачі використаємо породжувальний шаблон проєктування «Будівельник». Винесемо конструювання об'єкту звіту про успішність за межі класу у спеціальний клас будівельник CourseAcademicResultsReportBuilder. Таким чином ми зможемо отримувати різні відображення об'єктів, які міститимуть лише необхідну інформацію, використовуючи один і той самий програмний код.

Розділивши застосунок на три шари: шар презентації, шар бізнес-логіки та шар доступу до даних, було виявлено, що на рівні доступу до даних спосіб роботи з даними є однаковим для всіх сутностей домену. Тому доцільним є реалізація загального абстрактного способу роботи з даними, що дозволить нам також не турбуватись про те, яке сховище використовує шар доступу до даних.

Отже, шаблон проєктування «Репозиторій» дозволяє відокремити бізнес-логіку та дані, а інтерфейс містить усі методи CRUD-операцій: читання, видалення, оновлення, створення, що є загальним для всіх сутностей. Для реалізації цього шаблону проєктування було реалізовано інтерфейс IRepository та класи XmlRepository та SqlServerRepositry, які його реалізують.

2.3 Комунікації і послідовність взаємодії об’єктів системи

Зв'язки та взаємодія об'єктів у системі виявляються через діаграми послідовності у мові UML. Ці діаграми відтворюють процес обміну повідомленнями між об'єктами протягом визначеного періоду. Кожна така діаграма описує конкретний сценарій чи варіант використання, демонструючи послідовність подій, які відбуваються між різними об'єктами. Однією з таких є діаграма послідовності.

Діаграма послідовності в UML відображає взаємодію об'єктів системи в часовому порядку, відтворюючи послідовність їх повідомлень та дій.

Розглянемо ключові елементи діаграми послідовності:

- Об’єкт (актор) — це елемент, який може бути фізичною сутністю або представляти класи програм, модулі системи чи функції,
- Лінія життя — це характеристика кожного об’єкта, яка відображає тривалість існування об’єкта на діаграмі,
- Активаційна скринька — це елемент, що відображає часовий проміжок, протягом якого об'єкт знаходиться у стані активації в діаграмі активності, вказуючи час та послідовність виконання певної дії,
- Синхронні повідомлення — це, коли запит передається з очікуванням відповіді, і процес зупиняється, доки не отримає відповідь,
- Асинхронні повідомлення — це передача команди чи інформації до іншого об’єкта без очікування зворотного зв’язку, і може бути виконана неодноразово без обмежень,
- Відповіді можуть бути синхронними або асинхронними, де синхронні очікують підтвердження для продовження, а асинхронні просто реєструють дії без зупинки процесу використання ресурсу,

- Прямокутник з написом "ALT" всередині і з двома або більше гілками виходу відображає альтернативний сценарій в діаграмі послідовності. Кожна гілка представляє альтернативний шлях виконання, який може бути обраний залежно від умови чи події, що виникла під час основного сценарію,
- Прямокутник з написом "OPT" всередині вказує на додатковий шлях виконання, де користувач може обирати один із можливих методів виконання певного процесу. Це подібно до використання фільтрів у пошуку, де користувач сам обирає параметри для отримання результатів,
- Прямокутник з написом "LOOP" всередині вказує на циклічну дію, яка виконується постійно протягом певного проміжку часу чи життєвого циклу,
- Завершення — кінець життєвого циклу об’єкта.

Було вирішено побудувати діаграму послідовності для наочного відображення взаємодії об'єктів під час створення та виконання практичного завдання. Це дозволить продемонструвати логіку та послідовність подій у процесі його виконання.

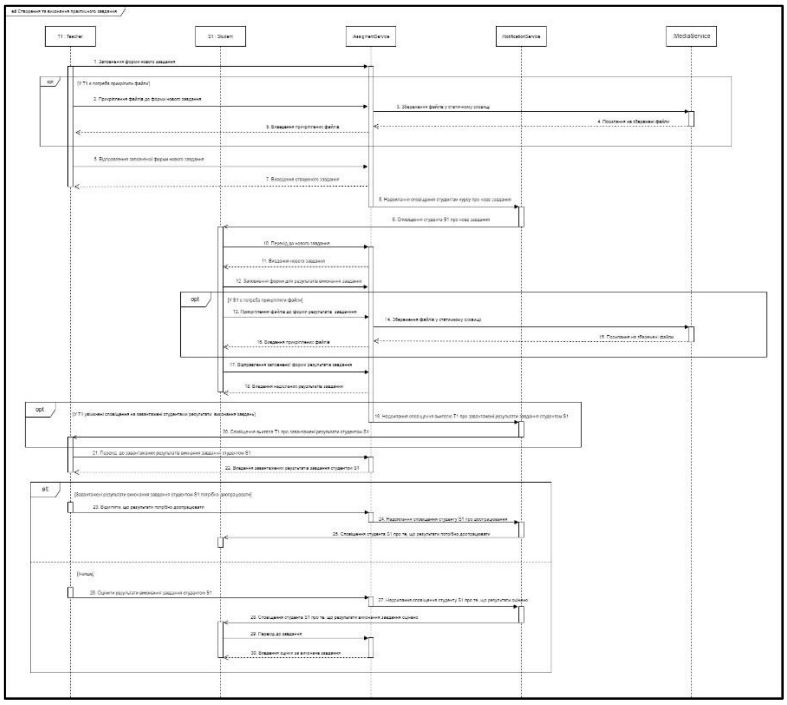


Рис. 2.5. Діаграма послідовності створення та виконання практичного завдання

На діаграмі послідовності створення та виконання практичного завдання відображено послідовність взаємодії між об'єктами Teacher, Student, AssigmentService, NotificationService, MediaService. Діаграма демонструє обмін повідомленнями між об'єктами та їх взаємодію у конкретних сценаріях виконання. Наприклад, Teacher та Student можуть спілкуватися через AssigmentService для виконання завдання, можуть використовувати NotificationService для сповіщень чи MediaService для обробки медіафайлів під час виконання та створення завдання. На цій діаграмі відображена прив'язка до часу через послідовність подій та їхній хронологічний порядок.

Діаграма кооперації демонструє взаємодії між різними елементами системи без прив'язки до часу, на відміну від діаграми послідовності. Вона відображає візуальну модель зв'язків між цими елементами у варіанті використання, аналізуючи обмін повідомленнями між об'єктами. На діаграмі показані об'єкти та способи, якими вони спілкуються в межах цього сценарію використання.

Розглянемо ключові елементи діаграми кооперації:

- Екземпляри акторів і класів: відображення окремих об'єктів, які беруть участь у виконанні конкретного варіанту використання,
- Асоціації між екземплярами: зв'язки між об'єктами, які демонструють способи їх взаємодії,
- Повідомлення: відображення обміну інформацією між екземплярами акторів і класів, що реалізують обрані сценарії використання, вони зображені стрілками або лініями з написами.

Діаграми кооперації було побудовано для відображення структури взаємодії при проходженні тестування студентом та при перевірці завантажених робіт і формування звіту успішності.

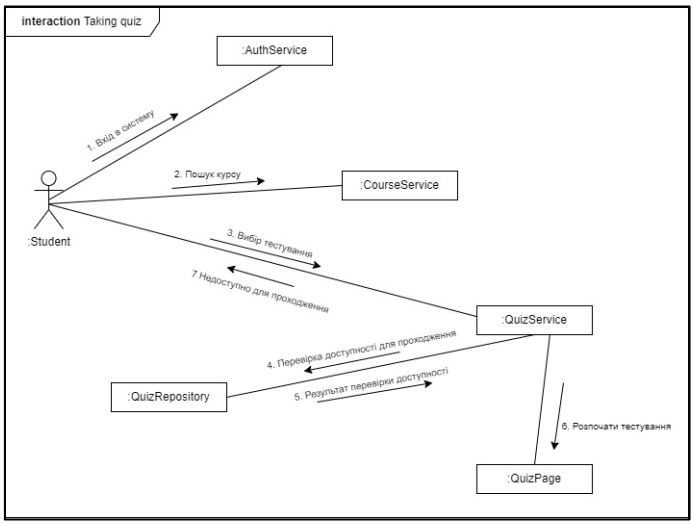


Рис. 2.6. Діаграма кооперації проходження тестування студентом

Діаграма кооперації "Проходження тестування студентом" включає об'єкти, що взаємодіють у процесі тестування. Учасники включають студента, який проходить тест, сервіси та репозиторій, необхідні для цього. Наприклад, AuthService використовується для аутентифікації студента, CourseService та QuizService - для доступу до курсу та тестування, а QuizRepository — для отримання даних тесту. Також присутня взаємодія з QuizPage, що представляє інтерфейс користувача для проходження тесту.

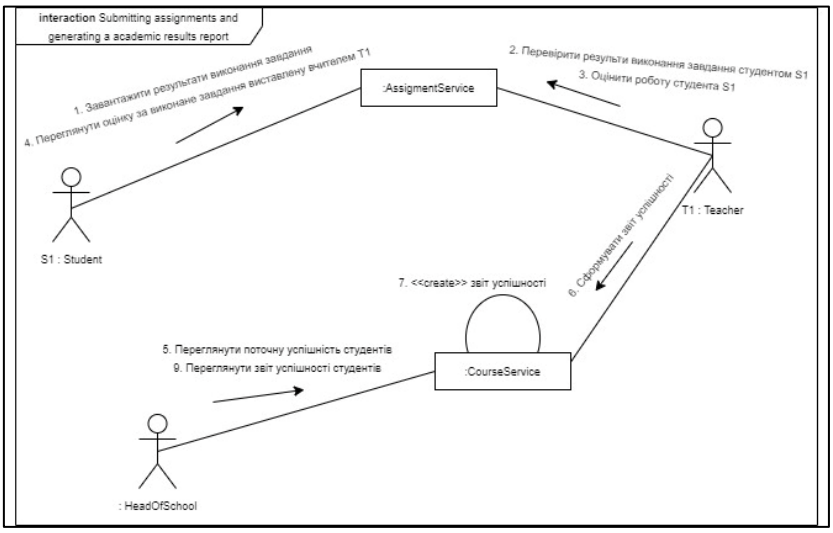


Рис. 2.7. Діаграма кооперації перевірки завантажених робіт та формування звіту успішності

Діаграма кооперації "Перевірка завантажених робіт та формування звіту успішності" включає в себе взаємодію об'єктів, таких як Teacher, Student, HeadOfSchool та AssigmentService. На цій діаграмі відображається обмін повідомленнями, виклики методів та передача інформації між цими об'єктами під час процесу перевірки завантажених робіт студентами та формування звіту про успішність.

Висновки до другого розділу

У цьому розділі було розроблено ключові структурні та взаємодійні моделі системи за допомогою діаграм активностей, класів, послідовностей та кооперації. Діаграми активностей дозволили відобразити послідовність операцій та процесів в системі. Діаграма класів надає уявлення про структуру системи, включаючи класи, методи та взаємозв'язки між ними. Діаграми послідовності та кооперації дозволили візуалізувати взаємодію між об'єктами системи під час виконання конкретних сценаріїв.

РОЗДІЛ 3. ФІЗИЧНА МОДЕЛЬ ТА ПРОТОТИП ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Взаємодія компонентів системи

Для візуалізації компонентів та їх взаємозв'язків у системі застосовується діаграма компонентів. Ця діаграма допомагає показати структуру програмних компонентів та їх взаємодію в сервісно-орієнтованих проектах. Вона зосереджується на відображенні залежностей між компонентами, їхніми інтерфейсами та іншими важливими аспектами системи, забезпечуючи чітке уявлення про архітектурну структуру програмного продукту.

Розглянемо ключові елементи діаграми компонентів:

- Компонент — це окремий модуль або блок функціональності програмної системи, який може містити підкомпоненти
- Інтерфейс — це зв'язок між компонентами, позначений у вигляді кола, вони бувають "обов’язкові" (required), позначені колом у кінці лінії, та "доступні" (provided), позначені сокетом.
- Порт — це квадрат, прикріплений до компонента, який вказує на те, що інтерфейс надається внутрішнім компонентом і функціонує як засіб передачі даних та керування,
- Пакет — це група взаємопов'язаних компонентів, які утворюють велику систему,
- Асоціація — це лінія, що показує зв'язок між компонентами,
- Залежність — це пунктирна лінія, що вказує напрямок залежності між компонентами,
- Делегація — це лінія між компонентами, що показує передачу функціональності чи відповідальності від одного компонента іншому через їх інтерфейси.

Було розроблено діаграму компонентів, що охоплює всю систему згідно з поставленим завданням.

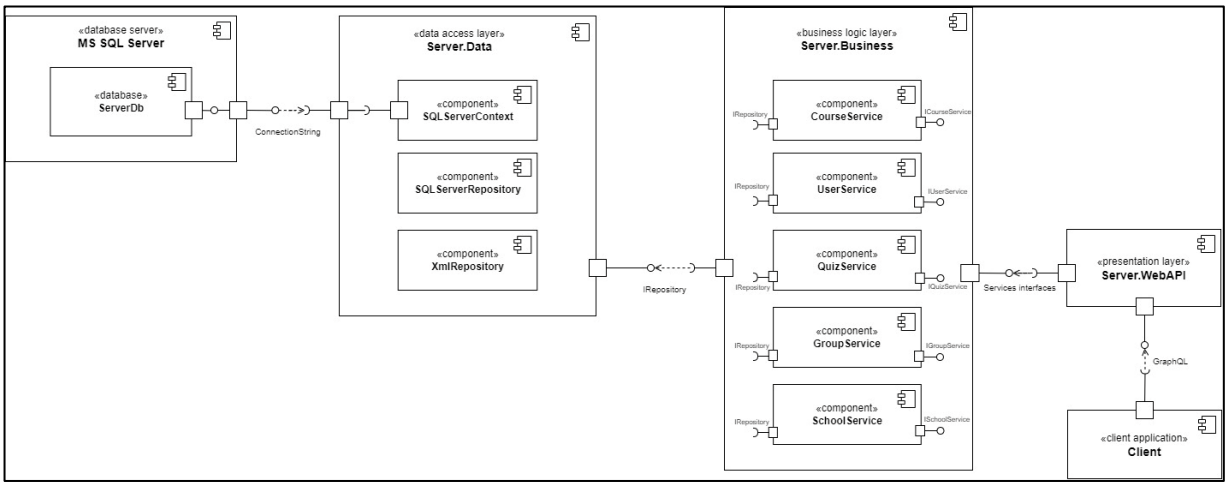


Рис. 3.1. Діаграма компонентів застосунку

При проектуванні застосунку була використана тришарова архітектура, яка складається з таких компонентів:

Серверна частина складається з декількох рівнів:

- ServerDb: цей рівень є базою даних, що надає інтерфейс через connection string для взаємодії з компонентом Server.Data.
- Server.Data: Він взаємодіє з базою даних і надає інтерфейс IRepository для компонента Server.Business.
- Server.Business: Виконує бізнес-логіку, використовуючи інтерфейс IRepository, що отримує від Server.Data, та надає інтерфейси сервісів для компонента Server.WebApi.

Клієнтська частина представлена компонентом Client, який використовує GraphQL інтерфейс, наданий Server.WebApi, для взаємодії з сервером. Такий підхід дозволяє чітко розділити відповідальності та забезпечити структурованість у взаємодії між різними частинами системи.

3.2 Архітектура програмного комплексу та його розгортання

Діаграма розгортання — це діаграма для візуалізації фізичної структури системи, яка відображає конфігурацію апаратних та програмних ресурсів у реальному середовищі. Ця діаграма вказує на робочі екземпляри компонентів, тобто показує, як програмне забезпечення розгорнуте на фізичних об'єктах, таких як апаратне забезпечення чи середовища виконання.

Розглянемо ключові елементи діаграми розгортання:

- Вузол: представлений прямокутником з ім'ям, він відображає фізичний об'єкт, такий як сервер або обладнання, де працює програмне забезпечення,
- Компонент: прямокутник з ім'ям, який вказує на програмний модуль або сервіс, що розгортається в одному чи кількох вузлах.
- Взаємозв'язок: лінія, яка з'єднує вузли або компоненти, вказує на зв'язок між ними, але не конкретизує його природу.
- Канал комунікації: лінія або стрілка, яка показує напрямок комунікації між вузлами, які можуть бути мережами або засобами передачі даних.
- Артефакт: прямокутник з ім'ям, що представляє ресурси, які зберігаються або обробляються, такі як файли чи бази даних.

Було створено діаграму розгортання застосунку, яка відображає фізичну структуру системи згідно з поставленою задачею.

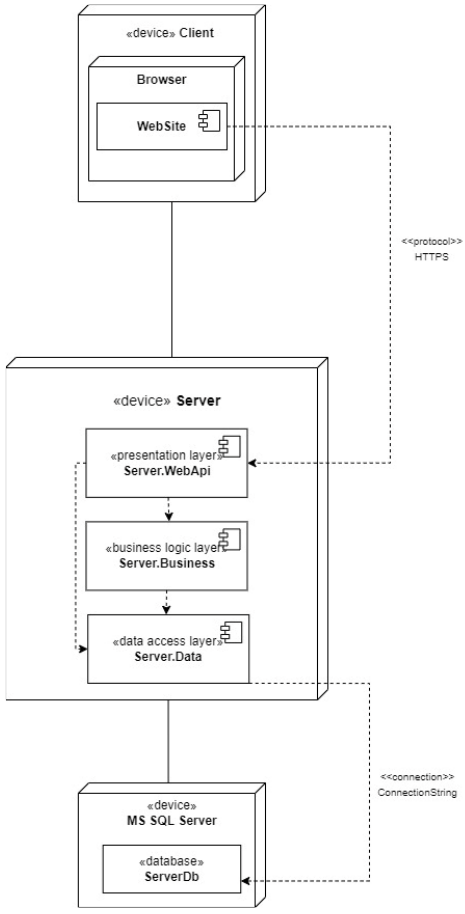


Рис. 3.2. Діаграма розгортання застосунку

На діаграмі розгортання є три пристрої: Client, Server і MS SQL Server. У Client є Browser, в якому розташований WebSite, що взаємодіє з сервером через HTTPS. На пристрої Server знаходяться компоненти Server.WebApi, Server.Business та Server.Data, які надають інтерфейс, виконують бізнес-логіку та взаємодіють з базою даних відповідно, також ці компоненти взаємодіють між собою. MS SQL Server представляє сервер бази даних, що зберігає та обробляє дані для системи.

3.3 Генерування програмного коду для прототипу програмного комплексу

Генерування коду здійснюється у StarUML на основі побудованої раніше діаграми класів. Цей CASE-сервіс автоматично створює код на доступних мовах програмування, таких як C#, PHP, Java та інші, відповідно до обраної мови програмування.

Кроки для генерації коду в StarUML виглядають наступним чином:

1. Відкрийте StarUML.
2. Побудуйте або відкрийте наявну діаграму класів.
3. Встановіть розширення для вибраної мови програмування через Tools > Extension Manager. Виберіть необхідне розширення та встановіть його.
4. Після встановлення розширення перезавантажте сервіс StarUML.
5. Переконайтеся, що налаштування оновилися, вибравши Tools > встановлене розширення > Generate code.
6. Виберіть модель, на основі якої потрібно згенерувати код, та вкажіть шлях для збереження результату.
7. Перейдіть за обраним шляхом та переконайтеся щодо наявності згенерованих файлів коду.

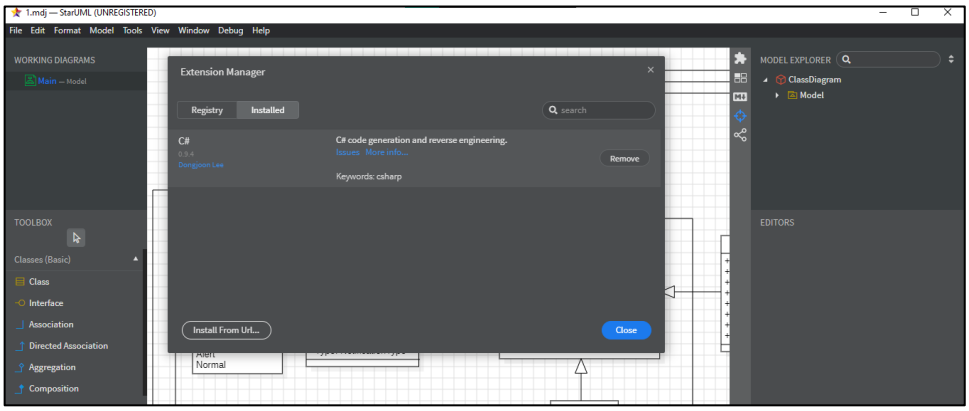


Рис. 3.3. Завантажене розширення для кодогенерації на C#

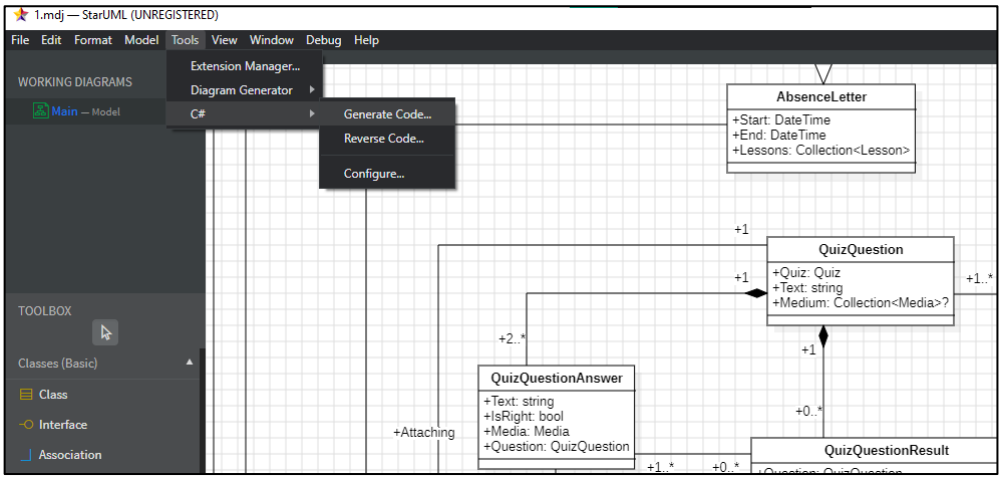
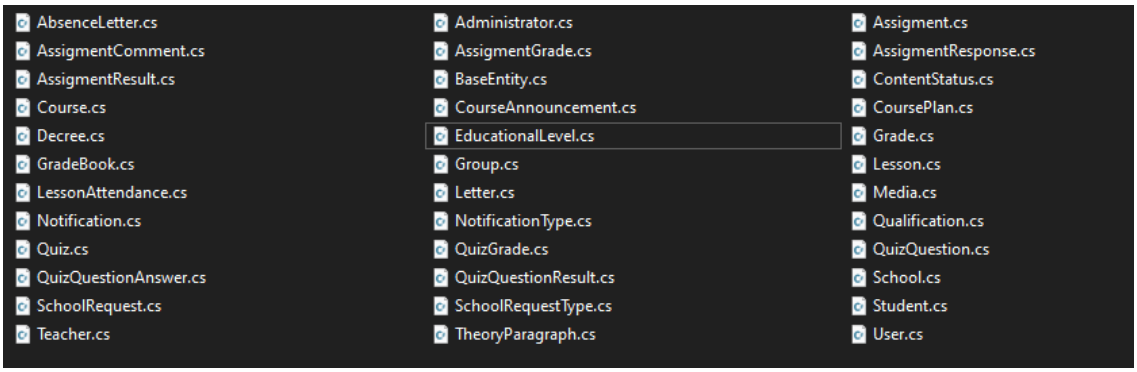


Рис. 3.4. Запуск кодогенерації



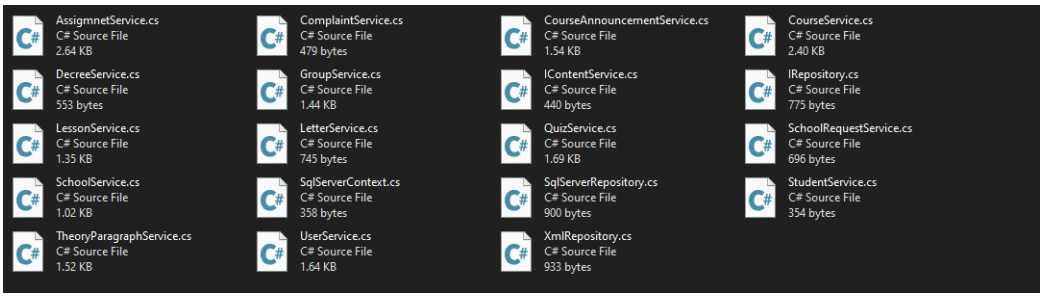


Рис. 3.5. – 3.6. Згенеровані файли під час кодогенерації

Висновки до третього розділу

У цьому розділі було змодельовано загальну структуру вузлів системи та їх взаємодії. Було створено дві важливі діаграми: діаграма компонентів, яка відображає взаємодію окремих модулів системи та діаграма розгортання, що представляє фізичне розташування цих компонентів на різних вузлах системи.

Окрім цього, результатом роботи над цим розділом стало генерування програмного коду на основі попередньо побудованої діаграми класів. Цей згенерований код є основою програмного забезпечення і відображає структуру та функціональність системи, визначені під час аналізу та моделювання.

ВИСНОВКИ

Під час виконання курсової роботи отримані теоретичні знання з проектування та моделювання програмного забезпечення було вдало застосовано на практиці.

Під час виконання першого розділу курсової роботи, було проведено аналіз задачі та визначено вимоги для моделювання. Були ретельно розглянуті різні CASE-інструменти для вибору оптимального для потреб проекту інструменту. У результаті цього аналізу були систематично проаналізовані всі види вимог до системи та створено діаграму використання.

Другий розділ був присвячений розробці ключових структурних та взаємодійних моделей системи. Застосовані діаграми активностей, класів, послідовностей та кооперації дозволили зобразити послідовність операцій, структуру системи, а також взаємодію об'єктів та компонентів під час виконання конкретних сценаріїв.

У третьому розділі було змодельовано загальну структуру вузлів системи та їх взаємодії через діаграми компонентів та розгортання. Крім того, в результаті цієї роботи був згенерований програмний код на основі попередньо побудованої діаграми класів. Цей згенерований код є основою програмного забезпечення та відображає структуру та функціональність системи, визначені під час аналізу та моделювання.

У результаті виконання курсової роботи була розроблена UML модель програмного комплексу для онлайн-платформи дистанційного навчання. Ця модель буде використана як основа для подальшої розробки застосунку.

Виконання поставлених завдань дозволило поглибити розуміння особливостей моделювання та аналізу програмних комплексів, що стало основним методом дослідження в рамках курсової роботи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Уніфікована мова моделювання UML. Загальна характеристика. [Електронний ресурс]. Режим доступу: https://svitppt.com.ua/informatika/unifikovana-mova-modelyuvannya-uml-zagalna-harakteristika.html
2. Рефакторинг Гуру. [Електронний ресурс]. Режим доступу: https://refactoring.guru/uk
3. UML 2.0 in Action: A project-based tutorial: A detailed and practical walk-through showing how to apply – MUMBAI: Packt Publishing Ltd., 2005
4. The Unified Modeling Language UML. [Електронний ресурс]. Режим доступу: https://www.uml-diagrams.org/
5. Class Diagram. [Електронний ресурс]. Режим доступу: https://www.uml-diagrams.org/class-diagrams-overview.html
6. Components Diagram. [Електронний ресурс]. Режим доступу: https://www.uml-diagrams.org/component-diagrams.html
7. Actiity Diagram. [Електронний ресурс]. Режим доступу: https://www.uml-diagrams.org/activity-diagrams.html
8. Sequence Diagram. [Електронний ресурс]. Режим доступу: https://www.uml-diagrams.org/sequence-diagrams.html
9. Use cases diagram. [Електронний ресурс]. Режим доступу: https://www.uml-diagrams.org/communication-diagrams.html
10. Deployment diagram. [Електронний ресурс]. Режим доступу: https://www.uml-diagrams.org/deployment-diagrams-overview.html

ДОДАТКИ

Лістинг класу AbsenceLetter:

```
public class AbsenceLetter : BaseEntity
{
    public DateTime Start;

    public DateTime End;

    public IEnumerable<Lesson> Lessons;
}
```

Лістинг класу Administrator:

```
public class Administrator : User
{
}
```

Лістинг класу Assigment:

```
public class Assignment : BaseEntity
{
    public Course Course;

    public string? Text;

    public DateTime Deadline;

    public IEnumerable<Media> Medium;

    public IEnumerable<AssignmentComment> Comments;

    public ContentStatus Status;

    public IEnumerable<AssignmentResult> AssigmentResults;

}
```

Лістинг класу AssigmentComment:

```
public class AssignmentComment : BaseEntity
{
    public User User;

    public string Text;

    public Assignment Assignment;

}
```

Лістинг класу AssigmentGrade:

```
public class AssignmentGrade : Grade
{
    public AssignmentResult Assignment;
}
```

Лістинг класу AssigmentResponse:

```
public class AssignmentResponse : BaseEntity
{
    public Student Student;

    public string Text;

    public IEnumerable<Media> Medium;

}
```

Лістинг класу AssigmentResult:

```
public class AssignmentResult : BaseEntity
{
    public Student Student;

    public AssignmentGrade Grade;

    public Assignment Assignment;
}
```

Лістинг класу BaseEntity:

```
public abstract class BaseEntity
{
    public Guid Id;

    public DateTime CreatedAt;

    public DateTime UpdatedAt;
}
```

Лістинг перерахування ContentStatus:

```
public enum ContentStatus
{
    Active,
    Hidden,
    Archived
}
```

Лістинг класу Course:

```
public class Course : BaseEntity
{
    public string Name;

    public ContentStatus Status;

    public IEnumerable<Group> Groups;

    public School School;

    public IEnumerable<Teacher> Teachers;
```

```
    public CoursePlan CoursePlan;

    public string? Topic;

    public string? About;

    public IEnumerable<Quiz> Quizzes;

    public IEnumerable<Letter> Letters;

    public IEnumerable<Assignment> Assigments;

    public IEnumerable<GradeBook>? GradeBooks;
}
```

Лістинг класу CourseAnnouncement:

```
public class CourseAnnouncement : BaseEntity
{
    public Course Course;

    public ContentStatus Status;

    public string Text;
}
```

Лістинг класу CoursePlan:

```
public class CoursePlan : BaseEntity
{
    public Course Course;

    public IEnumerable<Lesson> Lessons;
}
```

Лістинг класу Decree:

```
public class Decree : BaseEntity
{
    public Teacher Teacher;

    public string Text;

    public DateTime Date;
}
```

Лістинг перерахування EducationLevel:

```
public enum EducationalLevel
{
    Elementary,
    Medium,
    Sufficient,
```

```
    Highest
}
```

Лістинг класу Grade:

```
public class Grade : BaseEntity
{
    public Student Student;

    public Lesson? Lesson;

    public int GradeNumber;

    public GradeBook GradeBook;

    public string GradeForm;
}
```

Лістинг класу GradeBook:

```
public class GradeBook : BaseEntity
{
    public Group Group;

    public Course? Course;

    public IEnumerable<Grade> Grades;
}
```

Лістинг класу Group:

```
public class Group : BaseEntity
{
    public string Name;

    public Teacher Teacher;

    public IEnumerable<Student> Students;

    public School School;

    public int YearOfStudy;

    public IEnumerable<GradeBook> GradeBooks;

    public IEnumerable<Course> Courses;
}
```

Лістинг класу Lesson:

```
public class Lesson : BaseEntity
{
    public string Name;

    public string? Homework;

    public IEnumerable<Grade> Grades;
```

```
    public IEnumerable<LessonAttendance> Attendances;
}
```

Лістинг класу LessonAttendance:

```
public class LessonAttendance : BaseEntity
{
    public Student Student;

    public Lesson Lesson;

    public DateTime? JoiningDate;

    public DateTime? DisconnectionDate;

    public AbsenceLetter AbsenceLetter;
}
```

Лістинг класу Letter:

```
public class Letter : AbsenceLetter
{
    public Student Student;

    public string Text;
}
```

Лістинг класу Media:

```
public class Media : BaseEntity
{
    public string Name;

    public string Extension;
}
```

Лістинг класу Notification:

```
public class Notification : BaseEntity
{
    public IEnumerable<User> Users;

    public string Text;

    public NotificationType Type;
}
```

Лістинг перерахування NotificationType:

```
public enum NotificationType
{
    Success,
    Warning,
    Alert,
    Normal
}
```

Лістинг класу Qualification:

```
public enum Qualification
{
    Specialist,
    First,
    Second,
    Highest
}
```

Лістинг класу Quiz:

```
public class Quiz : BaseEntity
{
    public IEnumerable<QuizQuestion> Questions;

    public Course Course;

    public DateTime Open;

    public DateTime Closed;

    public TimeSpan Duration;

    public double MaxGradeValue;
}
```

Лістинг класу QuizGrade:

```
public class QuizGrade : Grade
{
    public ICollection<QuizQuestionResult> Results;

    public Student Student;
}
```

Лістинг класу QuizQuestion:

```
public class QuizQuestion : BaseEntity
{
    public Quiz Quiz;

    public string Text;

    public IEnumerable<Media>? Medium;
}
```

Лістинг класу QuizQuestionAnswer:

```
public class QuizQuestionAnswer : BaseEntity
{
    public string Text;

    public bool IsRight;

    public Media Media;

    public QuizQuestion Question;
}
```

Лістинг класу QuizQuestionResult:

```
public class QuizQuestionResult : BaseEntity
{
    public QuizQuestion Question;

    public Student Student;

    public IEnumerable<QuizQuestionAnswer> Answers;

    public QuizGrade Grade;
     }
```

Лістинг класу School:

```
public class School : BaseEntity
{
    public string Name;

    public string Address;

    public string Email;

    public IEnumerable<Group> Groups;

    public IEnumerable<Teacher> Teachers;

    public IEnumerable<Course> Courses;
}
```

Лістинг класу SchoolRequest:

```
public class SchoolRequest : BaseEntity
{
    public string Text;

    public SchoolRequestType Type;

    public string SenderInfo;
}
```

Лістинг перерахування SchoolRequestType:

```
public enum SchoolRequestType
{
    Join,
    Disconnect
}
```

Лістинг класу Student:

```
public class Student : User
{
    public Group Group;

    public string ParentsPhone;

    public string? About;
```

```
    public EducationalLevel EducationalLevel;

    public IEnumerable<Grade> Grades;

}
```

Лістинг класу Teacher:

```
public class Teacher : User
{
    public string Education;

    public string Expierence;

    public Qualification Qualification;

    public School School;

    public bool IsHead;

    public double LessonHoursPerWeek;

    public IEnumerable<Course> Courses;

    public IEnumerable<Decree> Decrees;
}
```

Лістинг класу TheoryParagraph:

```
public class TheoryParagraph : BaseEntity
{
    public string Text;

    public IEnumerable<Media> Medium;

    public Course Course;

    public ContentStatus Status;
}
```

Лістинг класу User:

```
public class User : BaseEntity
{
    public string FirstName;

    public string LastName;

    public string Email;

    public string Password;

    public bool IsActive;

    public IEnumerable<Notification> Notification;
}
```

Лістинг класу AssigmentService:

```
public class AssigmnetService : IContentService
{

    public AssigmnetService()
    {
    }

    private IRepostory<Assigmnet> assigmentRepository;

    private IRepostory<AssigmentResult> assigmentResultRepository;

    private IRepostory<AssigmentComment> assigmentCommentRepository;

    private IRepostory<AssigmentResponse> assigmentResponseRepository;

    /// <summary>
    /// @param assigment
    /// @return
    /// </summary>
    public void Create(object assigment) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Delete(object id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public IEnumerable<Assigment> GetByCourse(object id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param assigment
    /// @return
    /// </summary>
    public void Update(void assigment) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param studentId
    /// @param assigmentGrade
    /// @return
```

```
/// </summary>
public void AddGrade(void studentId, void assigmentGrade) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param groupId
/// @return
/// </summary>
public Dictionary<studentId> GetAssigmentResultByGroup(void groupId) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param userId
/// @return
/// </summary>
public IEnumerable<AssigmentResult> GetAsigmentResult(void userId) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param assigmentComment
/// @return
/// </summary>
public void AddComment(void assigmentComment) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param assigmentResponse
/// @return
/// </summary>
public void AddAssigmentResponse(void assigmentResponse) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Hide(void id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Archive(void id) {
    // TODO implement here
```

```
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Show(void id) {
        // TODO implement here
        return null;
    }
}
```

Лістинг класу ComplaintService:

```
public class ComplaintService {

    public ComplaintService() {
    }

    /// <summary>
    /// @param complaint
    /// @return
    /// </summary>
    public void Send(object complaint) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @return
    /// </summary>
    public void Response() {
        // TODO implement here
        return null;
    }
}
```

Лістинг класу CourseAnnouncementService:

```
public class CourseAnnouncementService : IContentService {

    public CourseAnnouncementService() {
    }

    private IRepository<CourseAnnouncement> courseAnnouncementRepository;

    public IRepository<Course> courseRepository;

    /// <summary>
    /// @param courseId
    /// @param theoryParagraph
    /// @return
    /// </summary>
    public void Create(Guid courseId, Guid theoryParagraph) {
```

```
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Delete(Guid id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public IEnumerable<CourseAnnouncement> GetByCourse(Guid id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param course
/// @return
/// </summary>
public void Update(Course course) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Hide(object id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Archive(object id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Show(object id) {
    // TODO implement here
    return null;
}
```

```
}
```

Лістинг класу CourseService:

```
public class CourseService : IContentService {

    public CourseService() {
    }

    private IRepository<Course> courserepository;

    private IRepository<Group> groupRepository;

    private IRepository<Teacher> teacherRepository;

    private IRepository<GradeBook> gradeBookRepository;

    /// <summary>
    /// @param course
    /// @return
    /// </summary>
    public void Create(Course course) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Delete(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param name
    /// @return
    /// </summary>
    public IEnumerable<Course> GetByName(string name) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param course
    /// @return
    /// </summary>
    public void Update(Course course) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
```

```
public IEnumerable<Course> GetByUser(Guid id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Hide(Guid id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Archive(Guid id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void Show(Guid id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param teacherId
/// @return
/// </summary>
public void AddTeacher(Guid teacherId) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param groupId
/// @return
/// </summary>
public void AddGroup(Guid groupId) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param drageBookId
/// @return
/// </summary>
public void AddGradeBook(Guid drageBookId) {
    // TODO implement here
    return null;
```

```
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Block(Guid id) {
        // TODO implement here
        return null;
    }
      }
```

Лістинг класу DecreeService:

```
public class DecreeService {

    public DecreeService() {
    }

    private IRepository<Decree> decreeRepository;

    /// <summary>
    /// @param decree
    /// @return
    /// </summary>
    public void AddDecree(Decree decree) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void DeleteDecree(Guid id) {
        // TODO implement here
        return null;
    }
}
```

Лістинг класу GroupService:

```
public class GroupService {

    public GroupService() {
    }

    private IRepository<Course> courseRepository;

    private IRepository<Student> studentRepository;

    /// <summary>
    /// @param group
    /// @return
    /// </summary>
    public void Create(Group group) {
        // TODO implement here
```

```
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Delete(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public Group GetByUser(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param course
    /// @return
    /// </summary>
    public void Update(Group group) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public IEnumerable<T> GetBySchool(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param studentId
    /// @return
    /// </summary>
    public void AddStudent(Guid studentId) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param studentId
    /// @return
    /// </summary>
    public void DeleteStudent(Guid studentId) {
        // TODO implement here
        return null;
    }
```

```
    }
```

Лістинг інтерфейсу IContentService:

```
public interface IContentService {

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Hide(Guid id);

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Archive(Guid id);

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Show(Guid id);

    }
```

Лістинг інтерфейсу IRepository:

```
public interface IRepository<T> where T : class
{
    /// <summary>
    /// @param filters
    /// @param includes
    /// @return
    /// </summary>
    public T GetOne(object filters, object includes);

    /// <summary>
    /// @param filters
    /// @param includes
    /// @return
    /// </summary>
    public IEnumerable<T> Get(object filters, object includes);

    /// <summary>
    /// @param entity
    /// @return
    /// </summary>
    public void Update(T entity);

    /// <summary>
    /// @param entity
    /// @return
    /// </summary>
    public void Delete(T entity);

    /// <summary>
    /// @param entity
```

```
    /// @return
    /// </summary>
    public void Add(T entity);
      }
```

Лістинг класу LessonService:

```
public class LessonService {

    public LessonService() {
    }

    private IRepository<Lesson> lessonRepository;

    private IRepository<Grade> gradeRepository;

    private IRepository<LessonAttendance> lessonAttendanceRepository;

    /// <summary>
    /// @param lesson
    /// @return
    /// </summary>
    public void Create(Lesson lesson) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Delete(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param lesson
    /// @return
    /// </summary>
    public void Update(Lesson lesson) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public CoursePlan GetCoursePlan(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param grade
    /// @return
```

```
    /// </summary>
    public void AddGrade(void grade) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public IEnumerable<LessonAttendance> GetLessonAttendance(Guid id) {
        // TODO implement here
        return null;
    }

}
```

Лістинг класу LetterService:

```
public class LetterService {

    public LetterService() {
    }

    private IRepository<Letter> letterRepository;

    private IRepository<User> userRepository;

    /// <summary>
    /// @param letter
    /// @return
    /// </summary>
    public void Add(Letter letter) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param letter
    /// @return
    /// </summary>
    public void AddAbsence(Letter letter) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @return
    /// </summary>
    public void DeleteLetter() {
        // TODO implement here
        return null;
    }

}
```

Лістинг класу QuizService:

```
public class QuizService
{

    public QuizService() {
    }

    private IRepository<Quiz> quizRepository;

    private IRepository<QuizQuestion> quizQuestionRepository;

    private IRepository<QuizQuestionResult> quizQuestionResultRepository;

    public IRepository<QuizQuestionAnswer> quizQuestionAnswerRepository;

    /// <summary>
    /// @param quiz
    /// @return
    /// </summary>
    public void Create(Quiz quiz) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Delete(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public IEnumerable<Quiz> GetByCourse(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param quiz
    /// @return
    /// </summary>
    public void Update(Quiz quiz) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param quizQuestion
    /// @return
    /// </summary>
    public void AddQuestion(QuizQuestion quizQuestion) {
        // TODO implement here
        return null;
```

```
    }

    /// <summary>
    /// @param userId
    /// @param quizQuestionResult
    /// @return
    /// </summary>
    public void AddQuestionResult(Guid userId, Guid quizQuestionResult) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param groupId
    /// @return
    /// </summary>
    public Dictionary<Student, IEnumerable<QuizQuestionResult>> GetQuizResultByGroup(Guid groupId) {
        // TODO implement here
        return null;
    }
      }
```

Лістинг класу SchoolRequestService:

```
public class SchoolRequestService
{

    public SchoolRequestService() {
    }

    private IRepository<SchoolRequest> schoolRequestRepository;

    /// <summary>
    /// @param request
    /// @return
    /// </summary>
    public void Send(void request) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @return
    /// </summary>
    public void Response() {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @return
    /// </summary>
    public void DeleteResponse() {
        // TODO implement here
        return null;
    }
```

```
}
```

Лістинг класу SchoolService:

```
public class SchoolService
{

    public SchoolService() {
    }

    private IRepository<School> schoolRepository;

    /// <summary>
    /// @param school
    /// @return
    /// </summary>
    public void Create(School school) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Delete(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param school
    /// @return
    /// </summary>
    public void Update(void school) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @return
    /// </summary>
    public IEnumerable<School> Get() {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param teacherId
    /// @return
    /// </summary>
    public void SetHead(Guid teacherId) {
        // TODO implement here
        return null;
    }

}
```

Лістинг класу SqlServerContext:

```
public class SqlServerContext {

    public SqlServerContext() {
    }

    /// <summary>
    /// @param modelBuilder
    /// @return
    /// </summary>
    protected void OnModelCreating(void modelBuilder) {
        // TODO implement here
        return null;
    }

     }
```

Лістинг інтерфейсу SqlServerRepository:

```
public interface SqlServerRepository<T> : IRepository<T> where T : class {

    SqlServerContext context { get; set; }

    /// <summary>
    /// @param filters
    /// @param includes
    /// @return
    /// </summary>
    public T GetOne(object filters, object includes);

    /// <summary>
    /// @param filters
    /// @param includes
    /// @return
    /// </summary>
    public IEnumerable<T> Get(object filters, object includes);

    /// <summary>
    /// @param entity
    /// @return
    /// </summary>
    public void Update(T entity);

    /// <summary>
    /// @param entity
    /// @return
    /// </summary>
    public void Delete(T entity);

    /// <summary>
    /// @param entity
    /// @return
    /// </summary>
    public void Add(T entity);
```

```
    /// <summary>
    /// @return
    /// </summary>
    public void Save();
}
```

Лістинг класу StudentService:

```
public class StudentService
{

    public StudentService() {
    }

    private IRepository<Student> studentRepository;

    /// <summary>
    /// @param id
    /// </summary>
    public void getStatisticInformation(Guid id) {
        // TODO implement here
    }

     }
```

Лістинг класу TheoryParagraphService:

```
public class TheoryParagraphService : IContentService {

    public TheoryParagraphService() {
    }

    private IRepository<TheoryParagraph> theoryParagraphRepository;

    private IRepository<Course> courseRepository;

    /// <summary>
    /// @param courseId
    /// @param theoryParagraph
    /// @return
    /// </summary>
    public void Create(Guid courseId, Guid theoryParagraph) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Delete(Guid id) {
        // TODO implement here
        return null;
    }
```

```
    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public IEnumerable<TheoryParagraph> GetBtCourse(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param course
    /// @return
    /// </summary>
    public id Update(Course course) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Hide(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Archive(Guid id) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param id
    /// @return
    /// </summary>
    public void Show(Guid id) {
        // TODO implement here
        return null;
    }

        }
```

Лістинг класу UserService:

```
public class UserService
{

    public UserService()
    {
    }
```

```
private IRepository<User> userrepository;

/// <summary>
/// @param user
/// @return
/// </summary>
public User Login(User user) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param user
/// @return
/// </summary>
public void Create(User user) {
    // TODO implement here
    return null;
}

/// <summary>
/// @return
/// </summary>
public void Logout() {
    // TODO implement here
    return null;
}

/// <summary>
/// @param user
/// @return
/// </summary>
public void Update(UserSecretsIdAttribute user) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void SetActive(Guid id) {
    // TODO implement here
    return null;
}

/// <summary>
/// @param id
/// @return
/// </summary>
public void SetInactive(Guid id) {
    // TODO implement here
    return null;
}

/// <summary>
```

```
    /// @param notifications
    /// @param ids
    /// @return
    /// </summary>
    public void SendNotification(Notification notification, Guid ids) {
        // TODO implement here
        return null;
    }

    /// <summary>
    /// @param userId
    /// @return
    /// </summary>
    public IEnumerable<Notification> GetNotifications(Guid userId) {
        // TODO implement here
        return null;
    }

    public void GenerateSchedule() {
        // TODO implement here
    }

        }
```

Лістинг інтерфейсу XmlRepository:

```
public interface XmlRepository<T> : IRepository<T> where T : class
{
    string StoragePath { get; set; }

    /// <summary>
    /// @param filters
    /// @param includes
    /// @return
    /// </summary>
    public T GetOne(object filters, object includes);

    /// <summary>
    /// @param filters
    /// @param includes
    /// @return
    /// </summary>
    public IEnumerable<T> Get(object filters, object includes);

    /// <summary>
    /// @param entity
    /// @return
    /// </summary>
    public void Update(T entity);

    /// <summary>
    /// @param entity
    /// @return
    /// </summary>
    public void Delete(T entity);

    /// <summary>
```

```
/// @param entity
/// @return
/// </summary>
public void Add(T entity);

/// <summary>

public T ParseXmlToEntity(object xmlNode)
```