Український державний університет

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Система формування індивідуальної освітньої траєкторії здобув освіти»

за освітньою програмою: «12 Інженерія програмного забезпечення»

зі спеціальності: «121 Інженерія програмного забезпечення»

Виконав: студент групи «ПЗ20130»

Курнір / Богдан КУШНІР /

(підпис студента) (Ім'я ПРІЗВИЩЕ)

Керівник: / доцент, Олена КУРОП'ЯТНИЙ

(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Нормоконтролер: / доцент, Світлана ВОЛКОВА

(підпис) (посада, Ім'я ПРІЗВИЩЕ)

Засвідчую, що у цій роботі немає запозичень праць інших авторів без відповідних посилань

Студент Курнір

(підпис)

Дніпро – 2023 рік

Ministry of Education and Science of Ukraine
Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note

to Bachelor's Thesis

on the topic: «The system of formation of an individual educational trajectory of an education seeker»
according to educational curriculum «12 Software engineering»
in the Speciality: «121 Software engineering»

Done by the student of the group PZ20130: / /

Scientific Supervisor: / /

Normative controller: / /

Dnipro – 2023

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Факультет «Комп'ютерні технології і системи»
Кафедра: «Комп'ютерні інформаційні технології»
Рівень вищої освіти: бакалавр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ /Вадим ГОРЯЧКІН/
(підпис)
Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу бакалавра
студенту Кушніру Богдану Сергійовичу
1. Тема роботи: «Система формування індивідуальної освітньої траєкторії здобувача освіти»
Керівник роботи:
затверджені наказом № 1209 ст від 07.12.2022
2. Строк подання студентом роботи: __.__.202_ р.
3. Вихідні дані до роботи: _____

4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати):
Вступ, Збір та аналіз вимог, зовнішнє проєктування, внутрішнє проєктування, проєктування архітектури системи, розробка програми, тестування та налагодження, висновки та рекомендації

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
Презентація
Відео роботи програми

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Технічне завдання: Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.		
2	Робочий проект. Програмування та відладка програми.		
3	Робочий проект. Тестування програми		
4	Робочий проект. Розробка, узгодження і затвердження програмної документації.		
5	Подання кваліфікаційної роботи до кафедри		
6	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії		

Студент _____ _____
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник роботи _____ _____
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра.

30с., 13 рис., 2 табл., 4 додатки, 6 джерел.

Метою даного дослідження є створення системи формування індивідуальної освітньої траєкторії здобувача освіти, яка буде сприяти ефективному розвитку і навчанню учнів. Система базується на використанні сучасних технологій та методологій, зокрема патерну MVC (Model-View-Controller), який забезпечує зручний та зрозумілий інтерфейс користувача та оптимальну організацію даних.

Об'єктом дослідження є методологія розробки системи формування індивідуальної освітньої траєкторії здобувача освіти на основі патерну MVC. Дана методологія є актуальною, оскільки надає можливість адаптувати навчальний процес до потреб кожного учня, сприяє його особистісному зростанню та розвитку.

Предметом дослідження є розробка веб-платформи для формування індивідуальної освітньої траєкторії здобувача освіти з використанням технологій ASP .Net та патерну MVC. Цей сайт надає можливість здобувачам освіти вибирати та налаштовувати свої освітні програми, враховуючи їхні особисті інтереси, навчальні потреби та можливості.

Пояснювальна записка складається зі вступу, 6 розділів, висновків, бібліографічного списку та 4 додатків:

- вступ. У розділі описується актуальність створення системи, наводяться основні завдання та цілі дослідження, 1 сторінка;
- збір та аналіз вимог. У розділі описані загальні вимоги до системи, 1 сторінка;
- зовнішнє проєктування. У розділі описані функціональні вимоги до системи, 3 сторінки;
- внутрішнє проєктування. У розділі описується структура та компоненти системи, 6 сторінок;
- проєктування архітектури системи. У цьому розділі детально описується внутрішня будова системи, 4 сторінки;

– розробка програми. В даному розділі описуються алгоритми роботи ключових модулів системи, 2 сторінки;
– тестування та налагодження, цьому розділі описуються процеси тестування системи, виявлення та виправлення помилок, 2 сторінки;
– додатки містять технічне завдання, опис та специфікацію програми.

Ключові слова: C#, SITES, MVC, WEB, INDIVIDUAL TRAJECTORY, FORMATION SYSTEMS.

ЗМІСТ

ВСТУП

Система формування індивідуальної освітньої траєкторії здобувача освіти є сучасним програмним рішенням, розробленим для надання студентам можливості планування та реалізації своєї освітньої траєкторії з урахуванням їхніх особистих потреб та цілей.

У сучасному освітньому середовищі все більше зростає розмаїтість курсів, програм та спеціалізацій, що доступні студентам. Однак, здобувачам освіти часто важко орієнтуватись в цьому розмаїтті та знайти оптимальний шлях розвитку. Тому система формування індивідуальної освітньої траєкторії здобувача освіти надає цінний інструмент для підтримки студентів у процесі планування своєї освіти.

Основною метою системи є надання можливості студентам створювати індивідуальні освітні траєкторії, враховуючи їхні інтереси, навички, потреби та кар'єрні амбіції. Система допомагає здобувачам освіти визначити необхідні курси, які вони повинні пройти для досягнення своїх навчальних та професійних цілей.

У процесі розробки системи використовується сучасний стек технологій, зокрема фреймворки та інструменти, що дозволяють створити зручний та інтуїтивно зрозумілий інтерфейс для користувачів. Для реалізації функціональності системи використовуються мови програмування, такі як C# та JavaScript, а також бази даних для збереження інформації про дисципліни та студентів.

Система формування індивідуальної освітньої траєкторії здобувача освіти призначена для студентів, навчальних закладів, викладачів та адміністраторів освітніх програм. Вона створена з метою полегшити процес планування освіти та підтримати студентів у досягненні їхніх навчальних та професійних цілей.

1 ЗБІР ТА АНАЛІЗ ВИМОГ

Постановка задачі:

- розробити програмний продукт, що представляє собою систему формування індивідуальної освітньої траєкторії здобувача освіти;
- забезпечити інтерфейс веб-порталу для зручного використання користувачами.

В результаті збору та аналізу вимог від користувачів було визначено, що користувачем системи буде адміністратор, якому будуть надані такі функціональні особливості:

1. Формування індивідуальної освітньої траєкторії здобувача освіти.

Система повинна дозволяти здобувачам освіти обирати дисципліни, які вони бажають вивчати.

Здобувачі освіти повинні мати можливість встановлювати пріоритети для обраних дисциплін.

2. Відображення інформації про обрані дисципліни та семестри.

Система повинна відображати користувачам інформацію про обрані дисципліни у зручному вигляді.

Користувачі повинні мати можливість переглядати детальну інформацію про кожну обрану дисципліну.

3. Можливість редагування обраних дисциплін та семестрів.

Користувачам повинна бути надана можливість редагувати дисципліни, змінювати пріоритети та додавати нові дисципліни та семестри.

4. Генерація груп здобувачів освіти на основі обраних дисциплін та семестрів.

Система повинна автоматично генерувати групи здобувачів освіти на основі обраних ними пріоритетів.

Висновки до розділу 1

В цьому розділі було описано мінімальні потреби для працездатності додатку.

На подальших етапах розробки можна виокремити такі вимоги:

- покращений дизайн сайту та його адаптивність;
- створення фільтрів на сторінках;
- розширення предметної області, додання характеристик до сутностей, наприклад, у студента може бути спеціальність, що додає можливість підбирати дисципліни вже під конкретну спеціальність;
- покращення методу генерації груп.

2 ЗОВНІШНЄ ПРОЄКТУВАННЯ

Функціональне призначення системи полягає в розробці системи формування індивідуальної освітньої траєкторії здобувача освіти. Програмний продукт має забезпечувати зручний інтерфейс веб-порталу для користування користувачами.

Експлуатаційне призначення системи - надання здобувачам освіти можливості обирати дисципліни та семестри, встановлювати пріоритети для обраних дисциплін та семестрів, а також отримувати інформацію про обрані дисципліни та семестри у зручному вигляді.

Функціональні вимоги системи:

- можливість редагування даних дисциплін, студентів та семестрів навчання;
- можливість встановлювати пріоритети студентів по дисциплінах обраного семестру;
- детальний перегляд інформації про кожну обрану дисципліну, студента та семестр;
- зміна пріоритетів для обраних дисциплін та семестрів;
- додавання нових дисциплін, студентів та семестрів;
- автоматична генерація груп здобувачів освіти на основі дисциплін обраного семестрі та розставлених пріоритетів;

Обмеження системи та додаткові можливості:

- забезпечення валідації введеної інформації та повідомлення про некоректність введених даних;
- обмежене виконання фільтрації та пошуку.

Для повного опису функціональних вимог, було розроблено діаграми прецедентів (рис. 2.1 – 2.3), що описують процес взаємодії з системою.

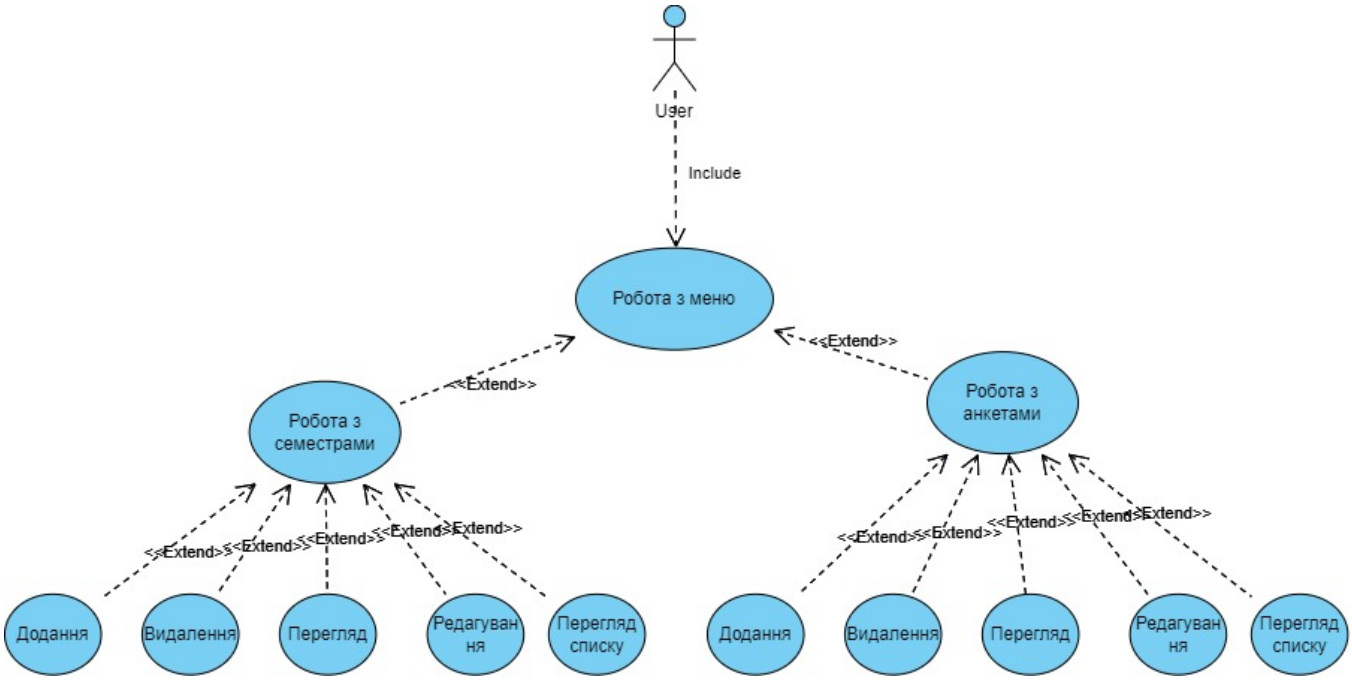

Рисунок 2.1 – Діаграма прецедентів робота з анкетами та семестрами

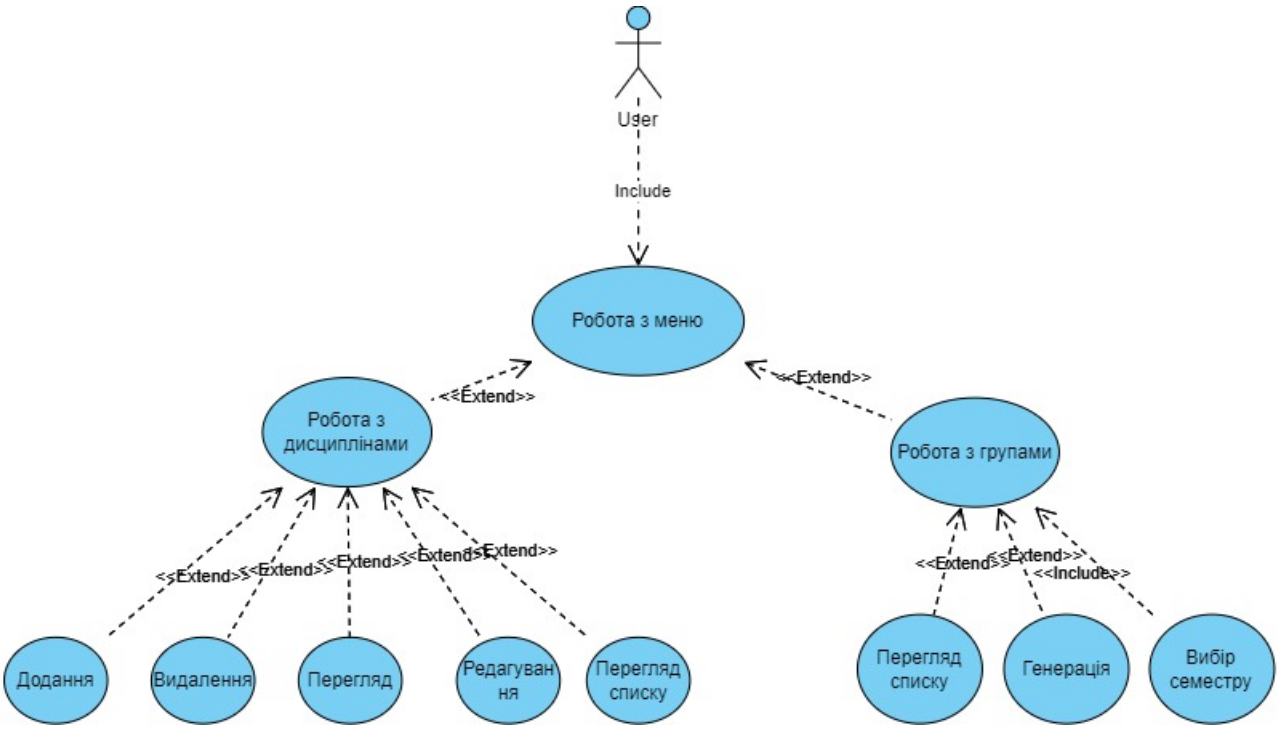

Рисунок 2.2 – Діаграма прецедентів робота з дисциплінами та групами

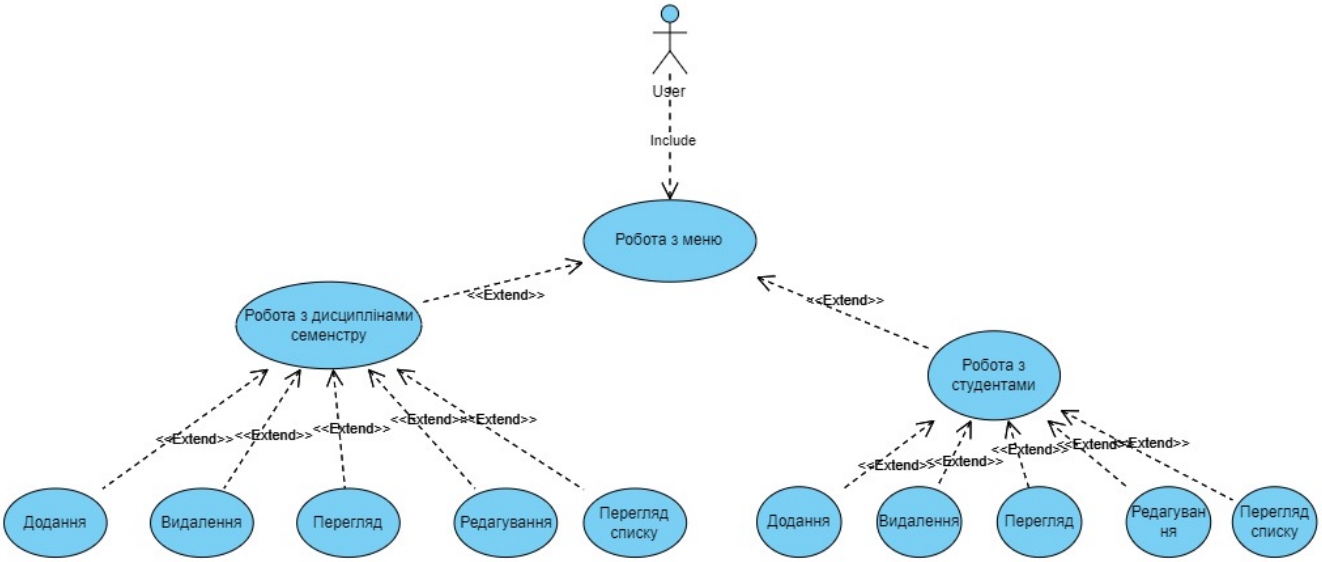

Рисунок 2.3 – Діаграма прецедентів робота з додатковими відомостями дисципліни та студентами

Висновки до розділу 2

У цьому розділі було повністю визначено функціональні вимоги до системи формування індивідуальної освітньої траєкторії здобувача освіти. Описані можливості та обмеження системи. Та процес взаємодії користувача з системою.

3 ВНУТРІШНЄ ПРОЄКТУВАННЯ

При проектуванні системи формування індивідуальної освітньої траєкторії здобувача освіти використовується архітектурний патерн MVC (Model-View-Controller). Цей патерн дозволяє розділити роботу системи на три основні компоненти: моделі, представлення та контролери.

Моделі

Моделі в системі виконують дві основні функції.

Перші моделі використовуються для представлення елементів ведення даних, з якими працюють користувачі. Вони забезпечують збереження даних, введених користувачем, такі як дисципліни та семестри (View Model).

Другі моделі виконують роль посередника між користувачем та базою даних. Вони зберігають в собі дані з бази даних (Data transfer object).

Представлення

Представлення в системі використовуються для відображення даних моделей у вигляді інтерфейсу користувача.

Представлення також може включати додаткові функції, такі як перевірка вмісту даних чи додаткові можливості відображення.

Контролери

Контролери в системі відповідають за обробку подій у представленні та взаємодію з моделями.

Їх основний обов'язок полягає в обробці подій, що виникають у користувача, і отриманні необхідних даних з моделей для подальшого переходу до відповідних представлень.

Контролери виконують роль сполучника між моделями та представленнями.

Завдяки такому поділу на компоненти система буде легшою у супроводженні та розширенні протягом усього періоду її функціонування, незалежно від обсягу та складності [1].

Для наочного відображення структури програми було розроблено діаграми класів (рис. 3.1 – 3.2).

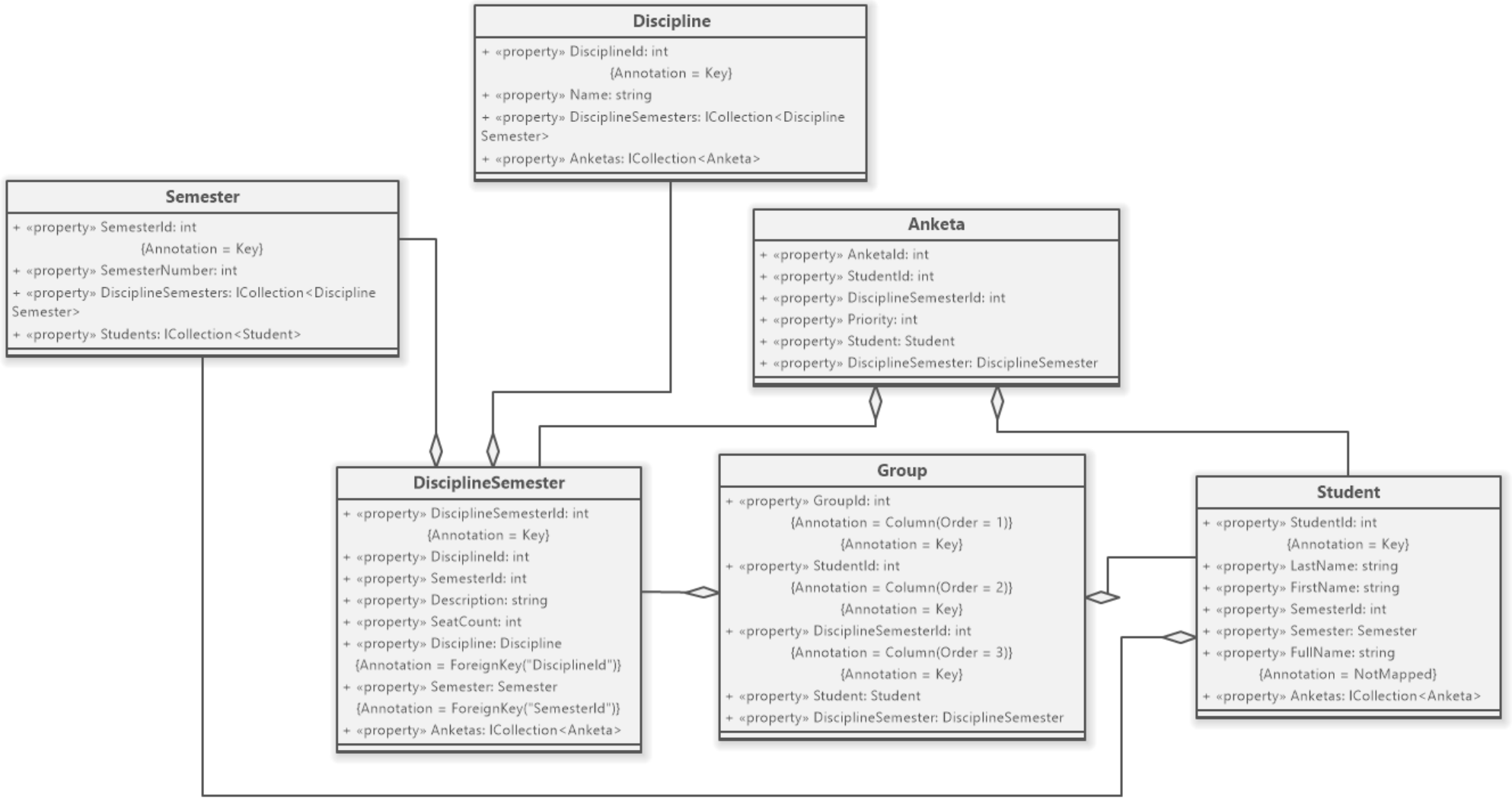

Рисунок 3.1 – Діаграма класів, взаємодія моделей

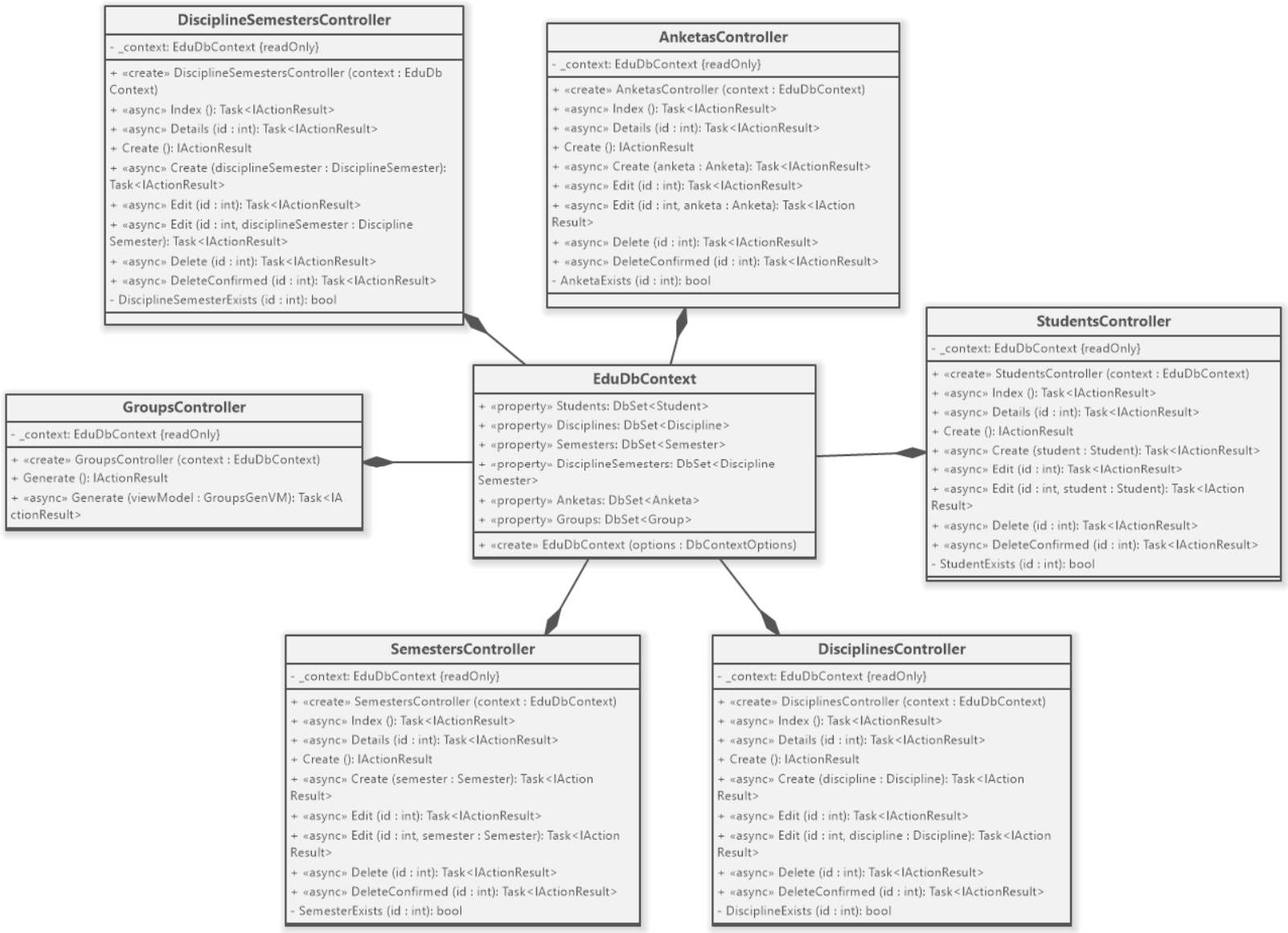

Рисунок 3.2 – Діаграма класів, взаємодія контролерів

Інтерфейс користувача

Однією з важливих частин роботи системи формування індивідуальної освітньої траєкторії здобувача освіти є її інтерфейс користувача. При проєктуванні інтерфейсу необхідно враховувати кольорову палітру елементів, їх розміри та розташування відносно інших елементів та важливості для користувача. Використання підходящих кольорів, розмірів та розташування елементів дозволить забезпечити зручний та привабливий інтерфейс для користувачів.

Засоби акцентування уваги на певних елементах є також важливою складовою частиною системи. Наприклад, можна використовувати галерею картинок, яка супроводжується анімацією для перегляду картинок або анімаційне зникнення елементів при їх видаленні. Такі засоби покращують зручність використання системи та роблять її більш привабливою для користувачів.

Для більшого розуміння класів кольорів в системі формування індивідуальної освітньої траєкторії здобувача освіти наведено класи кольорів, визначені у bootstrap 5, на рисунку 3.3.

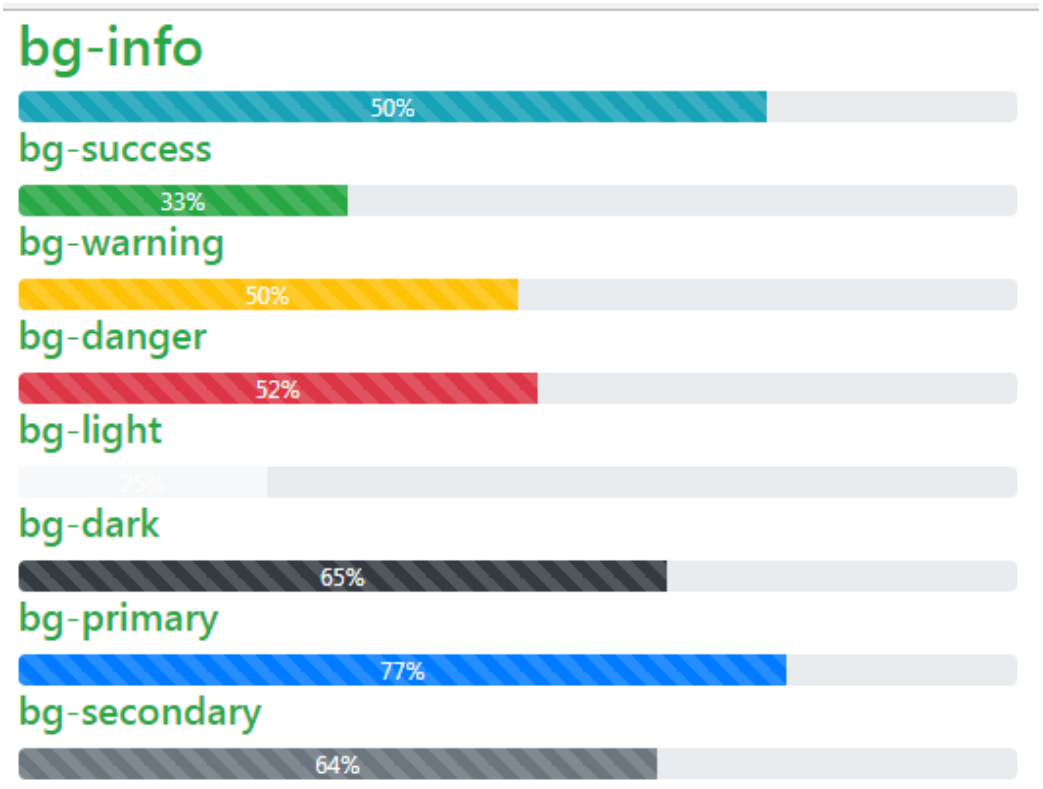

Рисунок 3.3 – Кольори палітри

Обрання відповідного виду діалогу з користувачем є ще одним важливим аспектом для успішної взаємодії системи з користувачем. Неправильний вибір виду діалогу може призвести до незручностей при використанні системи, що зменшує її цінність. В результаті аналізу вимог було визначено, що вид діалогу на основі екранних форм є найбільш підходящим для роботи з системою формування індивідуальної освітньої траєкторії здобувача освіти, оскільки вона вимагає безлічі відповідей від користувача на кожному кроці.

Кожна програма має свої повідомлення про стан роботи системи, для їх виводу на сайті використовуються повідомлення, що представлені в табл. 3.1.

Таблиця 3.1 – Повідомлення системи

Текст	Адресат	Ситуація	Рекомендовані дії
1	2	3	4
Некоректно заповнені дані	Користувач	Дані моделі були заповнені не відповідно до даних, які треба було ввести	Ввести коректні дані, в залежності від поля з помилкою
Запис не знайдено	Користувач	Користувач намагався працювати з даними, яких немає в системі	Оновити сторінку та спробувати ще раз, перевірити введені поля
Сторінку не знайдено	Користувач	Спроба перейти на сторінку яка відсутня у системі	Обрати іншу сторінку

Перелік браузерів на яких повинні правильно працювати усі функції сайту:

- Microsoft Edge v.104.2.14.02 та вище;
- Google Chrome v.105.1.21.34 та вище.

Динаміка системи

Основним модулем системи є генерація груп студентів, цей процес має такий алгоритм роботи:

1. Отримати список анкет з бази даних за вказаним номером семестру та відсортувати їх за пріоритетом.
2. Визначити мінімальний та максимальний пріоритет серед анкет.

3. Створити порожній список груп для збереження.

4. Отримати список дисциплін з бази даних разом з анкетами, що до них відносяться.

5. Видалити дисципліни, для яких відсутні анкети з мінімальним пріоритетом, зі списку дисциплін, які не вдається заповнити.

6. Пройти по пріоритетам від мінімального до максимального.

7. Для кожного пріоритету:

8. Отримати анкети з поточним пріоритетом.

9. Створити групи для кожної анкети, якщо студент не належить до жодної іншої групи.

10. Розрахувати кількість людей по дисциплінам у створених групах.

11. Вибрати дисципліну, для якої набралося найменше людей із дисциплін, які не вдалося заповнити.

12. Додати цю дисципліну до списку видалених дисциплін.

13. Отримати анкети з поточною дисципліною з мінімальним пріоритетом.

14. Для кожної анкети:

15. Знайти анкету з наступним пріоритетом для того ж студента.

16. Додати групу, якщо анкета знайдена і дисципліна не видалена.

17. Перевірити студентів, які не потрапили до жодної групи, і додати їх до групи з анкетою максимального пріоритету, якщо дисципліна не видалена.

18. Видалити всі існуючі групи з бази даних.

19. Зберегти нові групи у базі даних.

20. Отримати список груп з бази даних разом із додатковою інформацією.

21. Повернути часткове представлення зі списком груп для відображення.

Цей алгоритм дозволяє ефективно розподілити студентів на групи, дотримуючись пріоритетів заявок та обмежень по кількості студентів у групах. Він також враховує можливість розформування групи і перерозподілу студентів на інші групи у разі, якщо дисципліна не може бути заповнена повністю.

Висновки до розділу 3

Цей розділ містить опис внутрішньої структури проекту. В ньому описані підтримувані платформи, на яких працює сайт, алгоритм роботи найважливішої складової системи – генерація груп.

Також було описано формат взаємодії користувача з системою й особливості графічного інтерфейсу.

4 ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

Для наочного відображення фізичної структури проекту виконаємо проектування діаграми компонентів (рис. 4.1 – 4.6).

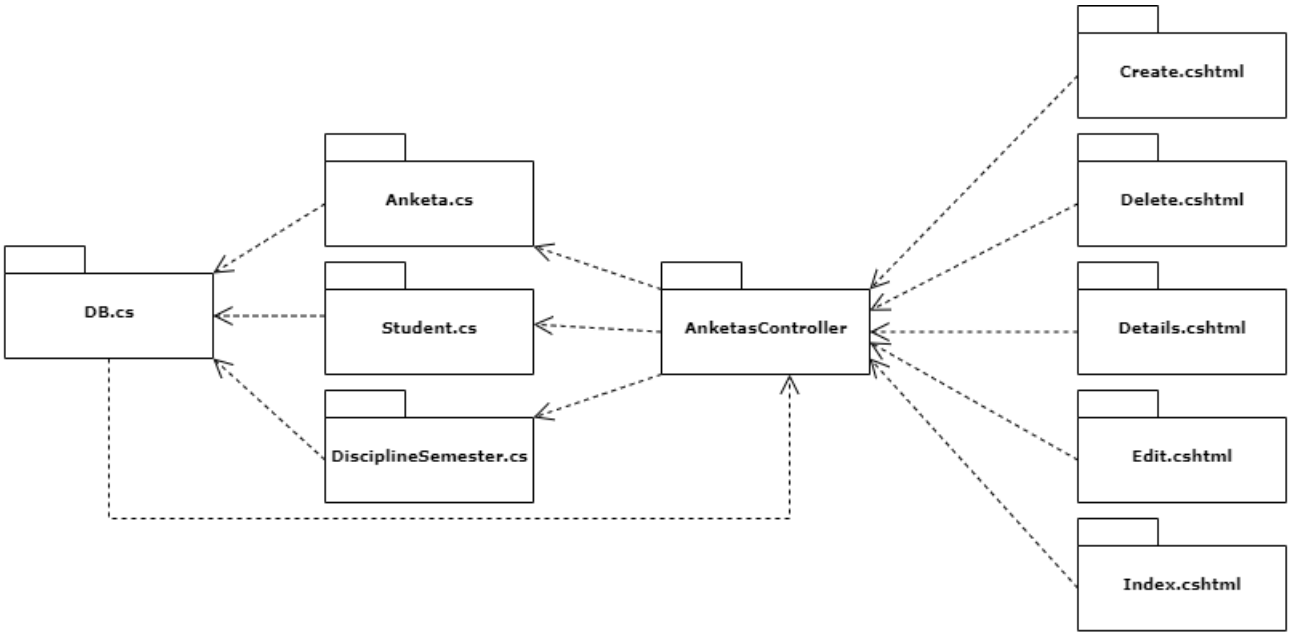

Рисунок 4.1 – Діаграма артефактів (компонентів) щодо контролеру анкет

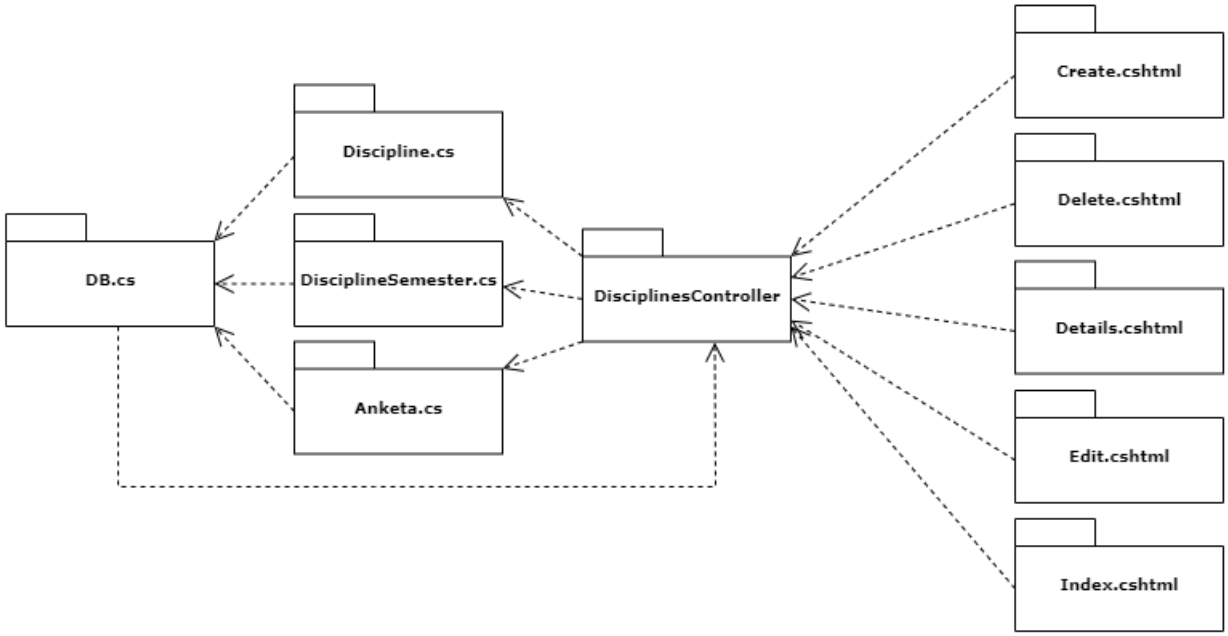

Рисунок 4.2 – Діаграма артефактів (компонентів) щодо контролеру дисциплін

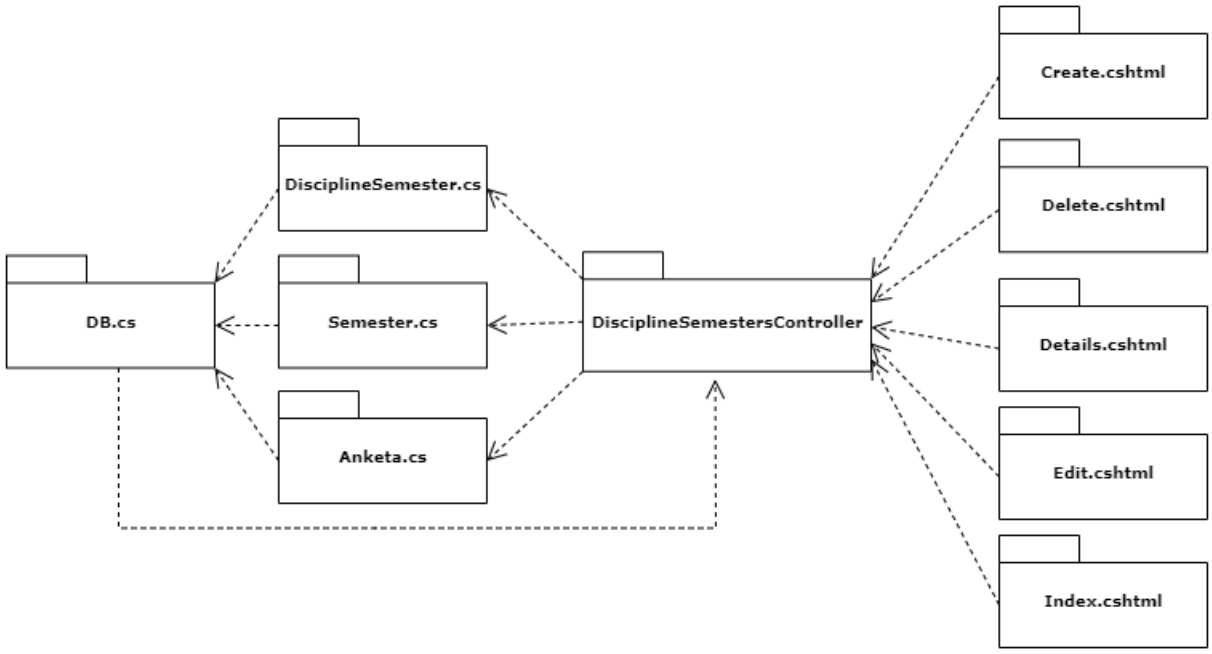

Рисунок 4.3 – Діаграма артефактів (компонентів) щодо контролеру розширених даних дисципліни

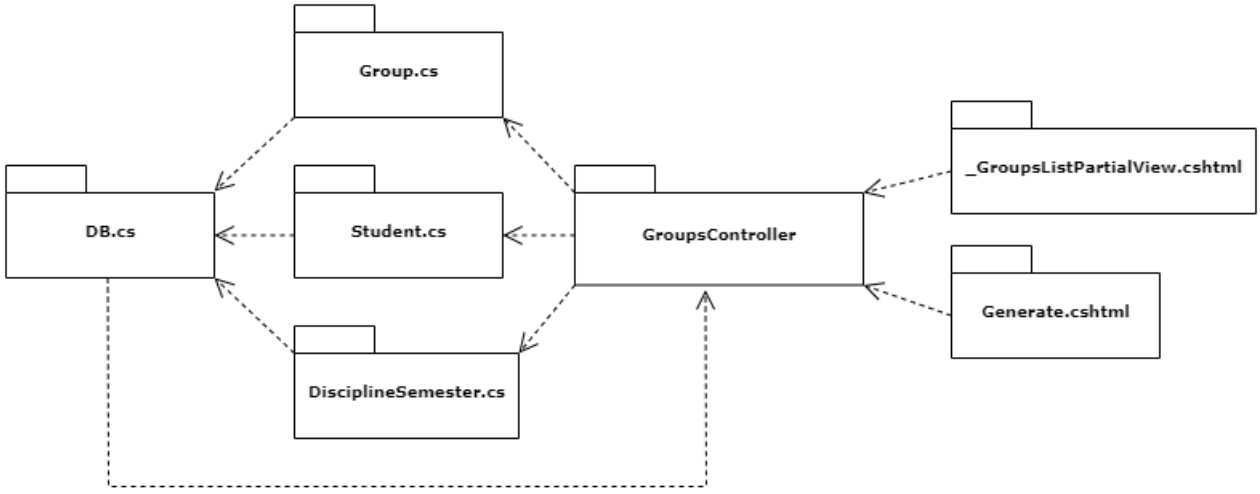

Рисунок 4.4 – Діаграма артефактів (компонентів) щодо контролеру груп

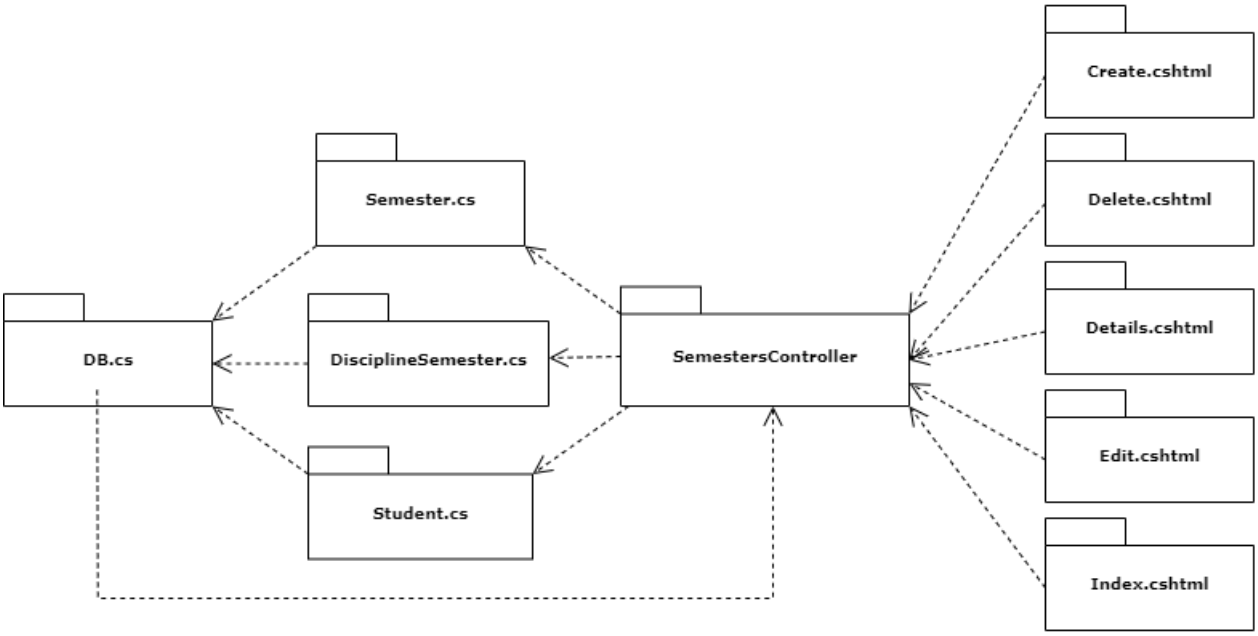

Рисунок 4.5 – Діаграма артефактів (компонентів) щодо контролеру семетрів

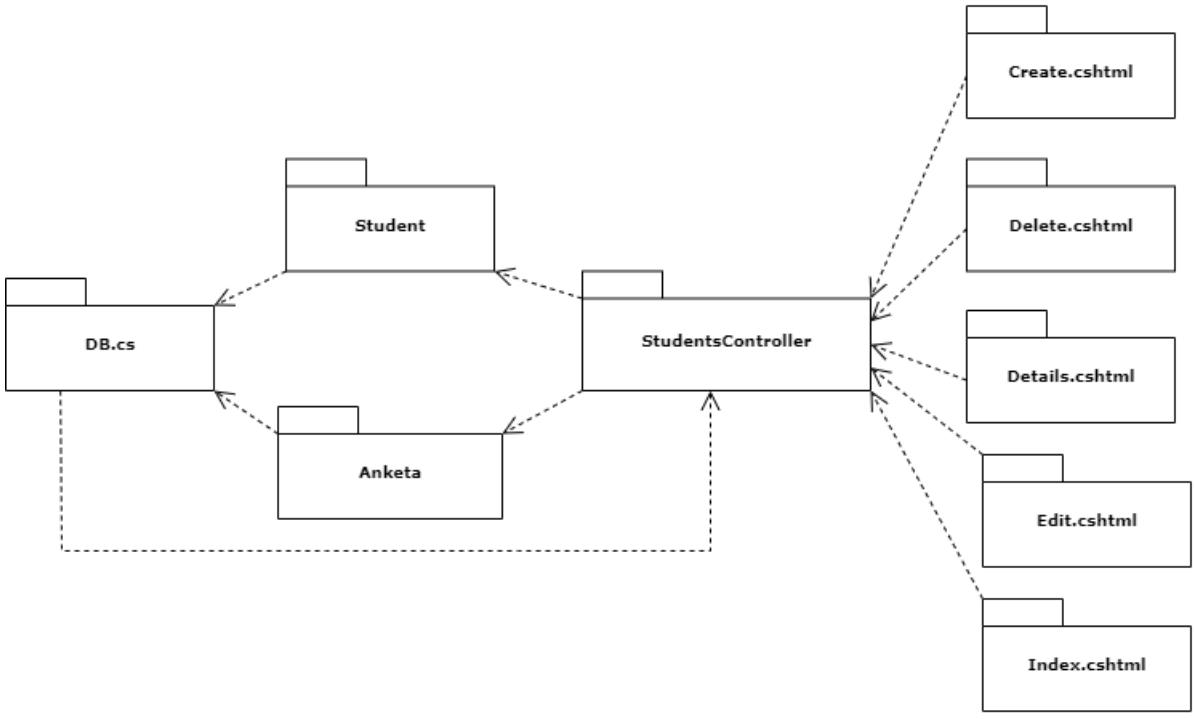

Рисунок 4.6 – Діаграма артефактів (компонентів) щодо контролеру студентів

В розробці програмного забезпечення важливим є детальне проектування бази даних, оскільки відносно її структури надалі будуть будуватись інші компоненти системи методи та засоби їх взаємодії. На рис 4.7 представлено ER-модель розробленої бази даних.

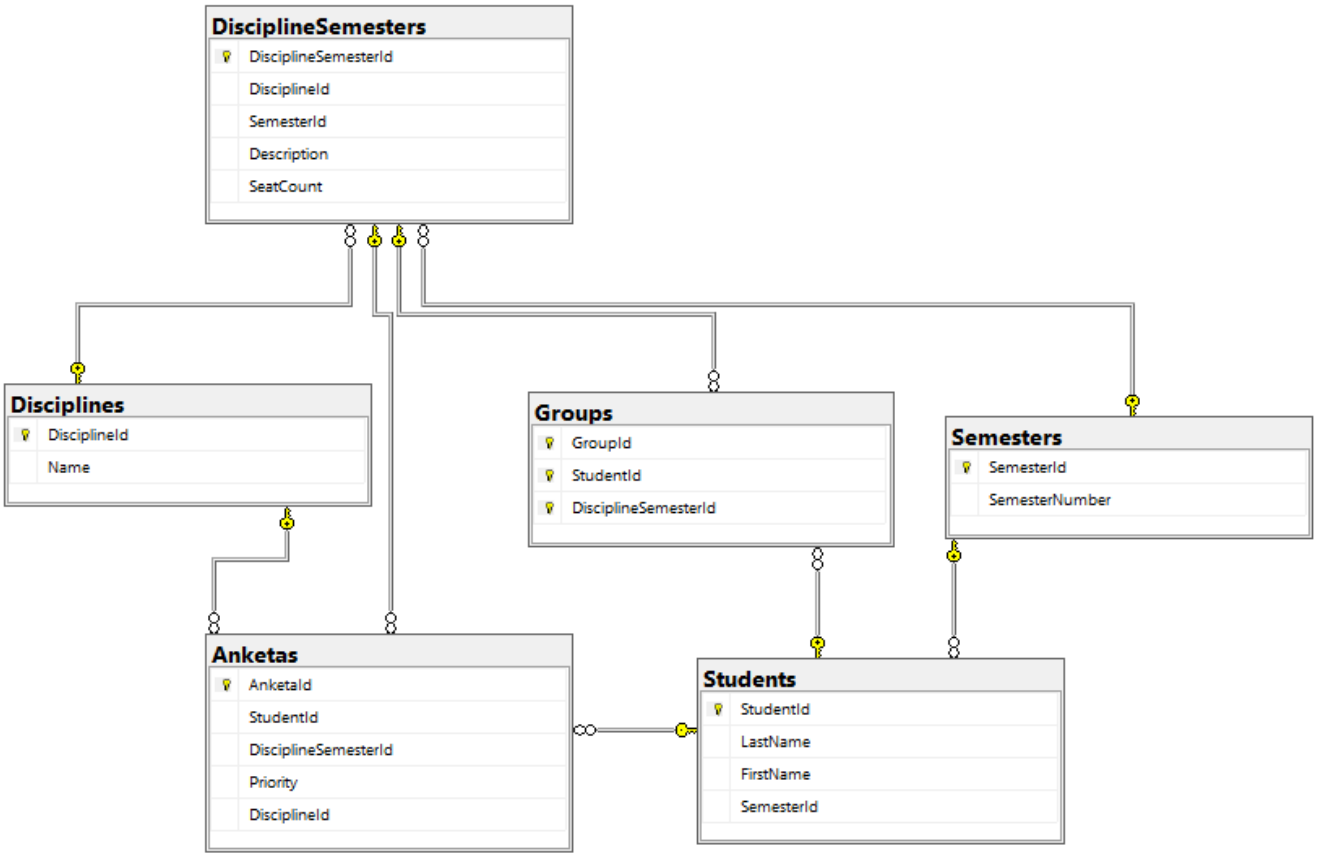

Рисунок 4.7 – ER-модель розробленої бази даних

Для роботи з базою даних програма використовує систему управління реляційними базами даних – Microsoft SQL Server, після розробки ER-моделі в середовищі Visual studio було розроблено відповідні таблиці бази даних.

Висновки до розділу 4

В цьому розділі була описана взаємодія компонентів на фізичному рівні, побудовано структури їх взаємодії на різних рівнях проекту, основними елементами виступали контролери. На діаграмах компонентів за допомогою різних кольорів представлено поділ модулів відповідно до їх ролей.

Також було спроектовано структуру відношень бази даних. Відношення спроектовані відносно поточної ітерації проекту й можуть покращуватись та розширюватись на інших ітераціях.

5 РОЗРОБКА ПРОГРАМИ

У розробці програми для даного проекту були обрані мова програмування C# та фреймворк ASP.Net MVC. Цей вибір має кілька переваг, які впливають на якість та продуктивність сайту.

По-перше, фреймворк ASP.Net MVC є незалежним від системи управління базою даних. Це означає, що можна легко вибрати підходящий ресурс для розміщення сайту залежно від потреб. Такий підхід забезпечує гнучкість та масштабованість проекту.

По-друге, мова HTML-розмітки Razor використовується для побудови сторінок сайту. Це дає можливість гнучко створювати сторінки з будь-яким набором даних. HTML-розмітка Razor сприяє ефективній і зручній роботі з даними та шаблонами сторінок.

По-третє, мова програмування C# є компільованою мовою, що приносить помітний приріст в продуктивності сайту. Автоматичне прибирання сміття в C# дозволяє уникнути проблем з управлінням пам'яттю та сконцентруватися на інших аспектах розробки системи.

Крім того, сайти на ASP.Net MVC легко розміщуються на рішеннях в Azure або IIS. Інтеграція з Visual Studio дозволяє швидко та зручно розгортати сайт зовнішніми ресурсами.

Окрім вибору технологій, програма розробляється методом покрокової деталізації, що означає, що розробка відбувається зверху вниз. Цей підхід дозволяє систематично розробляти та деталізувати програму, що сприяє зручності та ефективності роботи над проектом.

Цей метод дозволяє послідовно розвивати програму, забезпечуючи якість та контроль кожного етапу розробки.

1. Робота з меню.
    1.1. Робота з студентами
        1.1.1. Додання
        1.1.2. Перегляд списку
            1.1.2.1. Редагування

1.1.2.2. Видалення

1.1.2.3. Перегляд деталей

1.2. Робота з дисциплінами

1.2.1. Додання

1.2.2. Перегляд списку

1.2.2.1. Редагування

1.2.2.2. Видалення

1.2.2.3. Перегляд деталей

1.3. Робота з семестрами

1.3.1. Додання

1.3.2. Перегляд списку

1.3.2.1. Редагування

1.3.2.2. Видалення

1.3.2.3. Перегляд деталей

1.4. Налаштування додаткових відомостей дисциплін

1.4.1. Додання

1.4.2. Перегляд списку

1.4.2.1. Редагування

1.4.2.2. Видалення

1.4.2.3. Перегляд деталей

1.5. Робота з анкетами студентів

1.5.1. Додання

1.5.2. Перегляд списку

1.5.2.1. Редагування

1.5.2.2. Видалення

1.5.2.3. Перегляд деталей

1.6. Робота з групами

1.6.1. Генерація груп

1.6.2. Обрання семестру для генерації

1.6.3. Перегляд згенерованого списку

Висновки до розділу 5

У цьому було розглянуто вибір мови програмування та фреймворка для реалізації проекту. Обрано мову C# та фреймворк ASP.Net MVC з метою забезпечення якості та продуктивності сайту.

Фреймворк ASP.Net MVC був обраний через його незалежність від системи управління базами даних. Це дозволяє легко вибрати найкращий ресурс для розміщення сайту відповідно до потреб проекту. Гнучкість та масштабованість цього підходу забезпечують успішну реалізацію проекту.

Мова програмування C# була обрана через її компільований характер. Автоматичне прибирання сміття в C# забезпечує ефективне управління пам'яттю системи.

Також у розділі було також наведено список кроків розробки програми, включаючи роботу з меню, студентами, дисциплінами, семестрами, додатковими відомостями дисциплін, анкетами студентів та групами. Цей список детально описує функціонал програми та послідовність роботи з ним.

6 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ

При тестуванні програми доцільно використати метод тестування чорної скриньки — припущення про помилку, оскільки проект використовує додаткові бібліотеки, що відкидає можливість побудови класів еквівалентності для тестування, також інші методи не зможуть дати достатню якість тестування сайту.

Разом з методом припущення про помилку для налагодження програми буде використовуватись метод індукції та зворотного відстеження.

При використанні методу індукції було проаналізовано випадки помилки та дані чи дії що її спричиняють. Гіпотезами було отримано набір можливих причин помилки.

Під час тестування деякі помилки призводили до зупину програми в конкретному місці, для налагодження програми було використано метод зворотного відстеження.

Припущення про помилку:

1. Що буде якщо у обраному семестрі студенти які аналізувались мали вибір не серед 4 дисциплін кожен, а 2 студенти мали обрати серед двох дисциплін й 5 студентів мали вибір серед 4?
2. Що буде якщо два студенти з першого пункту мали 1 спільну дисципліну з іншими, й по одній різній дисципліні, при цьому інші студенти мають обирати серед 4 дисциплін?
3. Що буде, якщо залишиться студент, який не попав на жодну дисципліну, оскільки неможливо відкрити групи?
4. Що буде, якщо користувач спробує перейти на неіснуючу сторінку?
5. Що буде, якщо не було сформовано жодних груп?

Протокол налагодження програми був записаний до табл. 6.1.

Таблиця 6.1 – Протокол налагодження програми

Опис помилки	Опис ситуації	Способи усунення	Дії, що були застосовані для усунення
1	2	3	4
Після формування груп залишились студенти, які не ввійшли до жодної з груп, оскільки набір відкрити було неможливо	При формуванні групи студенти могли відправитись до груп інших дисциплін, проте вони, чомусь залишились без дисциплін	Перевірка даних студентів та порівняння даних про дисципліни на набір	Після першого формування було додано розміщення студентів до груп, які вдалось відкрити
Після повторного формування груп залишились студенти, які не ввійшли до жодної з груп	При формуванні групи студенти не змогли потрапити до жодної з груп, оскільки не було відкрито груп, які могли б розмістити в себе студента	Перевірка даних студентів та порівняння даних про дисципліни на набір	Після другого формування було додано розміщення студентів до груп, які не вдалось відкрити, тепер є група, яка не може відкритись, проте студент туди записується
Після формування груп не виявилось жодної сформованої, тому клієнт нічого не побачив на сторінці	Після отримання даних про групи, дані були порожні	Перевірити процес відправки даних клієнту та сформовану сторінку зі списком	Було додано перевірки на існування даних й перероблено сторінку, якщо даних не знайдено

Висновки до розділу 6

В цьому розділі описаний процес тестування та налагодження програми. Також описано помилки, що виникали при тестуванні програми та способи їх усунення.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

В рамках дипломної роботи було проведено дослідження та розробка проекту, спрямованого на формування освітньої траєкторії студента. Під час виконання роботи було отримано практичні навички у побудові сайту за архітектурним патерном MVC та досліджено методи та технології розробки сайтів на мові C#.

У процесі дослідження методів розробки сайтів був обраний найбільш доречний стек технологій, що включав в себе такі фреймворки та бібліотеки: ASP.Net Core v.7, Entity Framework v.7, JQuery v.3.6.2 та Bootstrap v.5. Ці технології були використані для побудови функціональності сайту, зокрема для роботи зі стороною клієнта, генерації адаптивних сторінок та стандартизованого інтерфейсу корситувача.

Проект був успішно реалізований згідно поставлених вимог, і на поточній стадії задовольняє потреби користувача.

Отже, в цій дипломній роботі описана актуальна проблематика формування освітньої траєкторії студента та розробку проекту, що забезпечує необхідні функціональні можливості по розміщенню студентів у групах дисциплін відповідно до їх побажань.

Щодо рекомендацій – варто покращити алгоритм розміщення людей по групам, також можна ускладнити систему додавши характеристики студента та дисциплін, наприклад, спеціальності, на яких вони викладаються також варто додати фільтри й пошуки за різними критеріями.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Albahari, J., Albahari, B. C# 9.0 in a Nutshell: The Definitive Reference. – Sebastopol, CA: O'Reilly Media, 2021. – 1080 с.
2. Richter, J. CLR via C# (Developer Reference). – Redmond, WA: Microsoft Press, 2018. – 896 с.
3. Troelsen, A., Japikse, P. Pro C# 7: With .NET and .NET Core. – Berkeley, CA: Apress, 2017. – 1412 с.
4. Deitel, P., Deitel, H. C# 9 and .NET 5 – Modern Cross-Platform Development. – Upper Saddle River, NJ: Pearson, 2021. – 1088 с.
5. Sestoft, P. C# Precisely. – Cambridge, MA: The MIT Press, 2016. – 216 с.
6. Lippert, E., et al. C# in Depth. – Greenwich, CT: Manning Publications, 2019. – 528 с.

ДОДАТКИ

ДОДАТОК А

ЗАТВЕРДЖУЮ
Проректор
Українського державного університету
науки і технології
Анатолій РАДКЕВИЧ

СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01319-01-ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН

Керівник розробки
_____Олена КУРОП'ЯТНИК

Виконавець
_____Богдан КУШНІР

Нормоконтролер
_____Світлана ВОЛКОВА

2023

ЗАТВЕРДЖЕНО
1116130.01319-01-ЛЗ

СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ

Технічне завдання

Листів 15

2023

АНОТАЦІЯ

Документ 1116130.01319-01 «СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ. Технічне завдання» входить до складу програмної документації на програму, яка представляє собою систему для формування освітньої траєкторії студента.

У даному документі представлені призначення та область застосування програмного продукту, основні вимоги, стадії та строки виконання проекту.

ЗМІСТ

ВСТУП

Система формування індивідуальної освітньої траєкторії здобувача освіти є сучасним програмним рішенням, розробленим для надання студентам можливості планування та реалізації своєї освітньої траєкторії з урахуванням їхніх особистих потреб та цілей.

У сучасному освітньому середовищі все більше зростає розмаїтість курсів, програм та спеціалізацій, що доступні студентам. Однак, здобувачам освіти часто важко орієнтуватись в цьому розмаїтті та знайти оптимальний шлях розвитку. Тому система формування індивідуальної освітньої траєкторії здобувача освіти надає цінний інструмент для підтримки студентів у процесі планування своєї освіти.

Основною метою системи є надання можливості студентам створювати індивідуальні освітні траєкторії, враховуючи їхні інтереси, навички, потреби та кар'єрні амбіції. Система допомагає здобувачам освіти визначити необхідні курси, які вони повинні пройти для досягнення своїх навчальних та професійних цілей.

У процесі розробки системи використовується сучасний стек технологій, зокрема фреймворки та інструменти, що дозволяють створити зручний та інтуїтивно зрозумілий інтерфейс для користувачів. Для реалізації функціональності системи використовуються мови програмування, такі як C# та JavaScript, а також бази даних для збереження інформації про дисципліни та студентів.

Система формування індивідуальної освітньої траєкторії здобувача освіти призначена для студентів, навчальних закладів, викладачів та адміністраторів освітніх програм. Вона створена з метою полегшити процес планування освіти та підтримати студентів у досягненні їхніх навчальних та професійних цілей.

1 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ від 07.12.2021 р. «Про призначення керівників та затвердження бакалаврських робіт», затверджений ректором Українського державного університету науки і технологій проф. Пшіньком О. М.

Тема дипломної роботи – «Система формування індивідуальної освітньої траєкторії здобувача освіти». Керівник – доцент Куроп’ятник О. С.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення системи полягає в розробці системи формування індивідуальної освітньої траєкторії здобувача освіти. Програмний продукт має забезпечувати зручний інтерфейс веб-порталу для користування користувачами.

Експлуатаційне призначення системи - надання здобувачам освіти можливості обирати дисципліни та семестри, встановлювати пріоритети для обраних дисциплін та семестрів, а також отримувати інформацію про обрані дисципліни та семестри у зручному вигляді.

.

3 ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

3.1 Вимоги до функціональних характеристик

Функціональні вимоги системи:

- можливість редагування даних дисциплін, студентів та семестрів навчання;
- можливість встановлювати пріоритети студентів по дисциплінах обраного семестру;
- детальний перегляд інформації про кожну обрану дисципліну, студента та семестр;
- зміна пріоритетів для обраних дисциплін та семестрів;
- додавання нових дисциплін, студентів та семестрів;
- автоматична генерація груп здобувачів освіти на основі дисциплін обраного семестрі та розставлених пріоритетів;

Обмеження системи та додаткові можливості:

- забезпечення валідації введеної інформації та повідомлення про некоректність введених даних.
- обмежене виконання фільтрації та пошуку.

3.2 Вимоги до надійності

Вимоги до надійності наступні:

- контроль ведених даних у полів введення: Необхідно забезпечити механізм контролю та валідації введених даних в полі введення. Це допоможе уникнути некоректних даних та можливих помилок при їх обробці;
- повідомлення про стан роботи та результати операцій: При оновленні даних, система повинна забезпечувати відображення відповідних повідомлень, що інформують користувача про стан роботи програми та результати виконаних операцій. Це дозволить користувачеві бути в курсі процесу та виявляти можливі проблеми;

– архівна копія тексту програми: Слід забезпечити збереження архівної копії тексту програми. Це дозволить відновити вихідний код у разі втрати або пошкодження основної версії програми, забезпечуючи можливість швидкої відновлення роботи системи;
– резервна копія бази даних на зовнішньому носії: Необхідно регулярно створювати резервні копії бази даних та зберігати їх на зовнішньому носії, такому як сервер, де розміщена база даних. Це забезпечить можливість відновлення даних у разі втрати, пошкодження або несправності бази даних.

3.3 Умови експлуатації

Програмний продукт повинен використовуватись у приміщеннях, що відповідають умовам роботи електронно-обчислювальних машин (ЕОМ). Це означає, що приміщення повинні мати відповідні кліматичні, санітарні та гігієнічні умови, відповідно до вимог, визначених у ДНАОП 0.00-1.31-99 (див. табл. 3.1).

Таблиця 3.1 – Кліматичні умови

Пора року	Категорія робіт згідно з ГОСТ 12.01-005-88	Температура повітря, град.С	Відносна вологість повітря, %	Швидкість руху повітря, м/с
		Оптимальна	Оптимальна	Оптимальна
Холодна	легка-1-а	22-24	40-60	0,1
	легка-1-б	21-23	40-60	0,1
Тепла	легка-1-а	23-25	40-60	0,1
	легка-1-б	22-24	40-60	0,2

Користувач, який працює з програмним продуктом, повинен мати навички роботи з персональним комп'ютером та бути ознайомленим з документацією до програми. Це допоможе забезпечити ефективне та безперебійне використання програмного продукту, оскільки користувач буде знати, як правильно взаємодіяти з програмою та виконувати необхідні дії.

3.4 Вимоги до складу та параметрів технічних засобів

Для використання розроблюваного програмного продукту на різних пристроях та сервері вимагається наступне обладнання:

- мінімум 2 гігабайти оперативної пам'яті;
- браузер , що підтримується програмним продуктом.

Та мобільних пристроях, що мають:

- мінімум 2 гігабайти оперативної пам'яті;
- операційна система Android версії 9 і вище.

Сервер повинен мати:

- мінімум 2 гігабайти оперативної пам'яті;
- процесор: 32 або 64 розрядний процесор з тактовою частотою 1 ГГц або вище, який підтримує набір інструкцій SSE2;
- мінімум 50 гігабайт простору на жорсткому диску;
- засоби контролю, такі як миша, віддалений робочий стіл, клавіатура;
- програвач CD-R;
- монітор.

3.5 Вимоги до інформаційної та програмної сумісності

Програмний продукт, що розробляється, має підтримку для наступних операційних систем:

- Windows 95 та наступні версії (включно) - це охоплює широкий спектр операційних систем Windows, включаючи Windows 98, Windows XP, Windows 7, Windows 8 і Windows 10;
- операційні системи Android версії 9.0 та вище - це включає усі після Android 9.0 версії, такі як Android 10, Android 11 і майбутні оновлення.

У процесі розробки програмного продукту використовується середовище розробки Visual Studio 2022 Professional, яке забезпечує розширені можливості для розробки програм на різних платформах.

Для управління базами даних використовується система керування базами даних MS SQL Server 2019.

Також, для користувачів програмного продукту необхідна наявність інтернет-браузера для доступу до функціональності програми через веб-інтерфейс.

3.6 Вимоги до маркування і упаковки

Упаковка програмного продукту та документації повинна бути ретельно захищена від можливих пошкоджень, які можуть виникнути під час транспортування або зберігання. Це охоплює механічні пошкодження (удари, подряпини) та кліматичні впливи (вологість, температура).

На зовнішній упаковці програмного продукту слід зазначити наступну інформацію:

- назва продукту – чітко вказати назву програмного продукту, щоб користувачі могли легко його ідентифікувати;
- номер версії – якщо програмний продукт має декілька версій, на упаковці слід зазначити номер поточної версії. Це допоможе користувачам визначити актуальність програми;
- мінімальні системні вимоги – на упаковці слід вказати необхідні системні вимоги для успішного встановлення та роботи програмного продукту. Це можуть бути вимоги до операційних систем, обсягу оперативної пам'яті, процесора та інших апаратних компонентів.

На зворотній стороні упаковки слід вказати наступну інформацію:

- розробник – зазначити ім'я або назву компанії, яка розробляє програмний продукт;
- юридична адреса – вказати повну юридичну адресу розробника, що дозволить зв'язатися з ним у разі потреби або отримання додаткової інформації.

На рис. 3.1. представлено приклад маркування.

Програмний додаток «Система формування індивідуальної освітньої траєкторії здобувача освіти» Мінімальні системні вимоги: – 2 гігабайти оперативної пам’яті; – 50 гігабайт простору на жорсткому диску;	Розробник: Кушнір Б. С. Кафедра «КІТ», УДУНТ м. Дніпро, вул. Лазаряна 2 2023

Рисунок 3.1 – Маркування

3.7 Вимоги до транспортування і зберігання

Транспортування програмного продукту повинно забезпечувати його безпеку, цілісність і захист від несанкціонованого доступу. Програмний виріб, який міститься на оптичному носії даних типу CD-R, потребує відповідної упаковки, щоб забезпечити захист від механічних пошкоджень та атмосферного впливу.

Оптимальним варіантом для захисту програмного продукту на CD-R може бути використання пластикового футляра або паперового конверту. Ці упаковки забезпечують фізичний захист від подряпин, ударів та інших механічних пошкоджень, які можуть виникнути під час транспортування.

Крім того, важливо враховувати атмосферний вплив на програмний продукт. Упаковка повинна бути стійкою до вологості, температурних змін та інших атмосферних факторів, які можуть негативно вплинути на якість CD-R та його вміст.

4 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

Вся документація до програмного додатку повинна задовольняти вимоги до програмної документації.

До складу програмної документації має входити технічне завдання та робочий проект.

До складу робочого проекту мають входити:

- специфікація;
- текст програми;
- опис програми.

Вся документація до програми повинна задовольняти вимогам державного стандарту до оформлення програмних документів [3].

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

В табл. 5.1 приведені стадії та етапи розробки програмного продукту.

Таблиця 5.1 – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	
Робочий проект	Програмування та відладка програми.	
	Тестування програми	
	Розробка, узгодження і затвердження програмної документації.	

6 ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Контроль здійснюється за допомогою виконання набору тестів з метою знаходження помилок в програмному продукті та його специфікації. Контроль виконання роботи забезпечується головним керівником розробки.

Прийом програмного продукту здійснюється уповноваженою комісією.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Albahari, J., Albahari, B. C# 9.0 in a Nutshell: The Definitive Reference. – Sebastopol, CA: O'Reilly Media, 2021. – 1080 с.
2. Richter, J. CLR via C# (Developer Reference). – Redmond, WA: Microsoft Press, 2018. – 896 с.
3. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю.М.Івченко, В.І.Шинкаренко, В.Г.Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В.Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В.Лазаряна, 2009. – 38 с.

ДОДАТОК Б

ЗАТВЕРДЖУЮ
Проректор
Українського державного університету
науки і технології
Анатолій РАДКЕВИЧ

СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ

Специфікація
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01319-01-ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН

Керівник розробки
_____Олена КУРОП'ЯТНИК

Виконавець
_____Богдан КУШНІР

Нормоконтролер
_____Світлана ВОЛКОВА

2023

ЗАТВЕРДЖЕНО
1116130.01319-01-ЛЗ

СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ

Специфікація

Листів 2

2023

Позначення	Найменування	Примітка
	Документація	
1116130.01319-01-ЛЗ	Лист затвердження	
1116130.01319-01 12 01-ЛЗ	Лист затвердження	
1116130.01319-01 12 01	Текст програми	
1116130.01319-01 13 01-ЛЗ	Лист затвердження	
1116130.01319-01 13 01	Опис програми	

ДОДАТОК В

ЗАТВЕРДЖУЮ
Проректор
Українського державного університету
науки і технології
Анатолій РАДКЕВИЧ

СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01319-01 12 01-ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН

Керівник розробки
_____Олена КУРОП'ЯТНИК

Виконавець
_____Богдан КУШНІР

Нормоконтролер
_____Світлана ВОЛКОВА

2023

ЗАТВЕРДЖЕНО
1116130.01319-01 12 01-ЛЗ

СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ

Текст програми

Листів 28

2023

АНОТАЦІЯ

Документ 1116130.01319-01 12 01 «СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ. Текст програми» входить до складу програмної документації на програму, яка представляє собою систему для формування освітньої траєкторії студента.

У даному документі представлений текст програми та схема взаємодії модулів програми. У процесі розробки програмного продукту використовується середовище розробки Visual Studio 2022 Professional, яке забезпечує розширені можливості для розробки програм на різних платформах.

Для управління базами даних використовується система керування базами даних MS SQL Server 2019.

ЗМІСТ

1 СХЕМА ВЗАЄМОДІЇ МОДУЛІВ

На рис. 1.1 – 1.6 приведена схема взаємодії модулів програми.

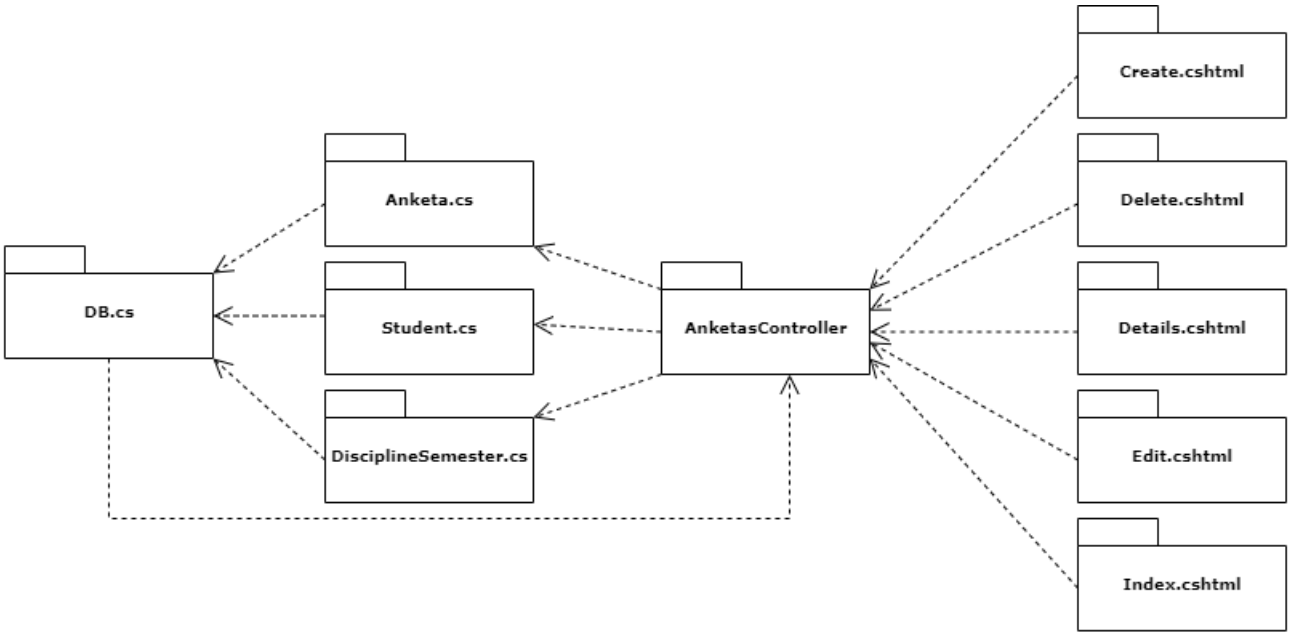

Рисунок 1.1 – Діаграма артефактів (компонентів)

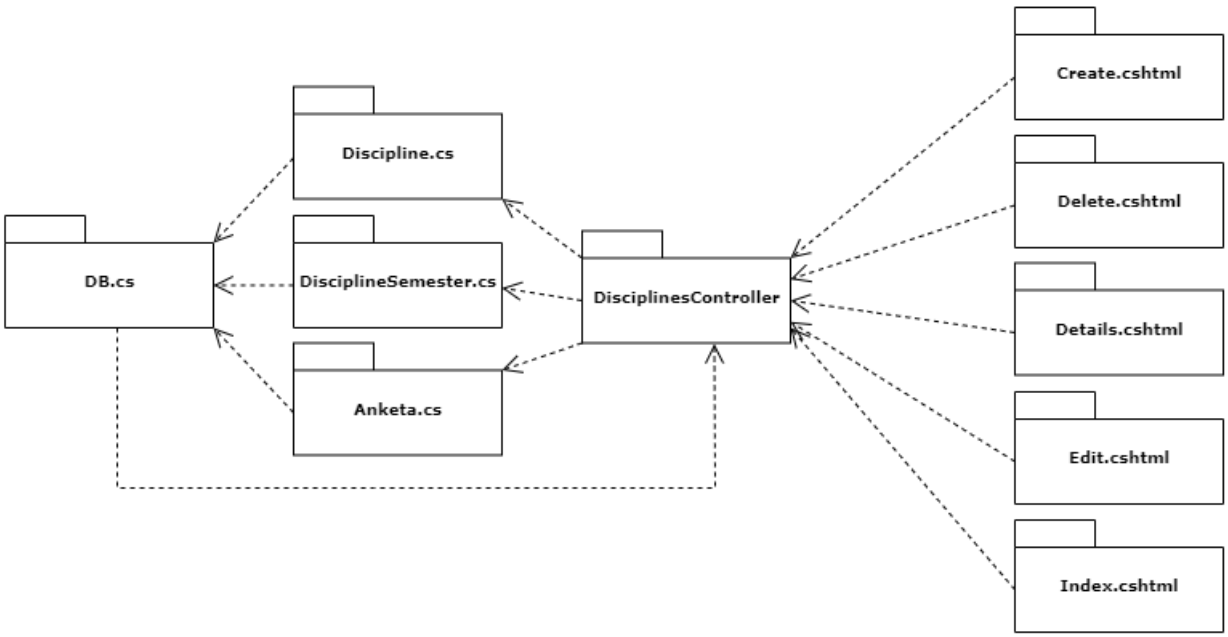

Рисунок 1.2 – Діаграма артефактів (компонентів)

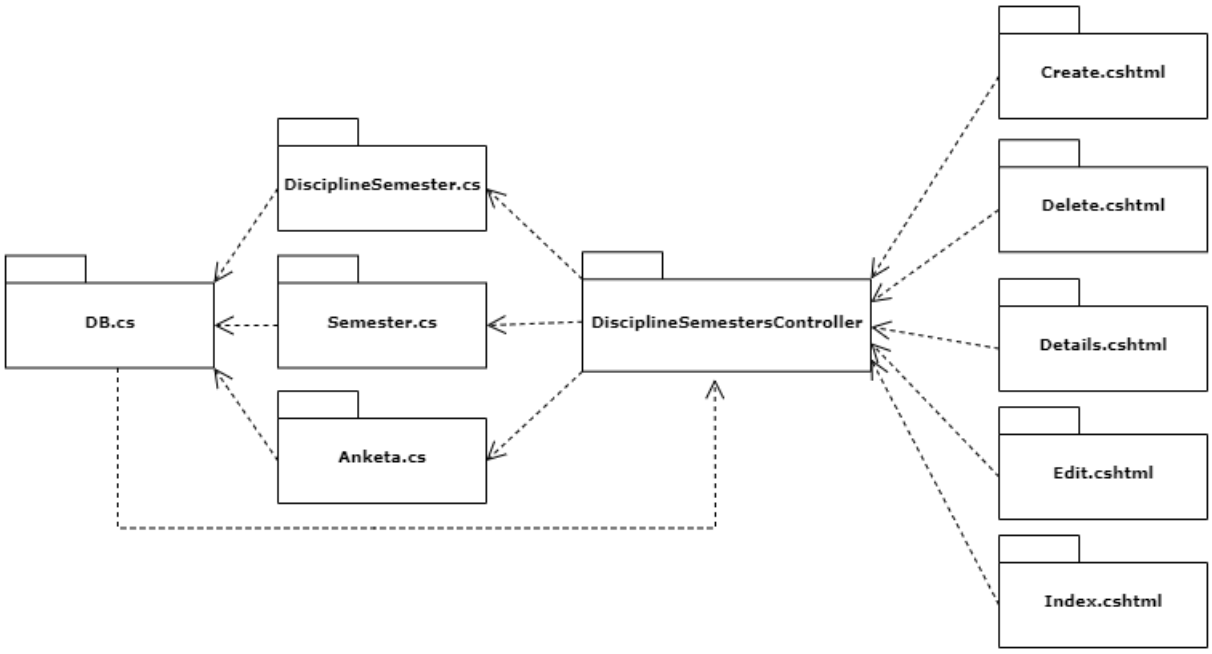

Рисунок 1.3 – Діаграма артефактів (компонентів)

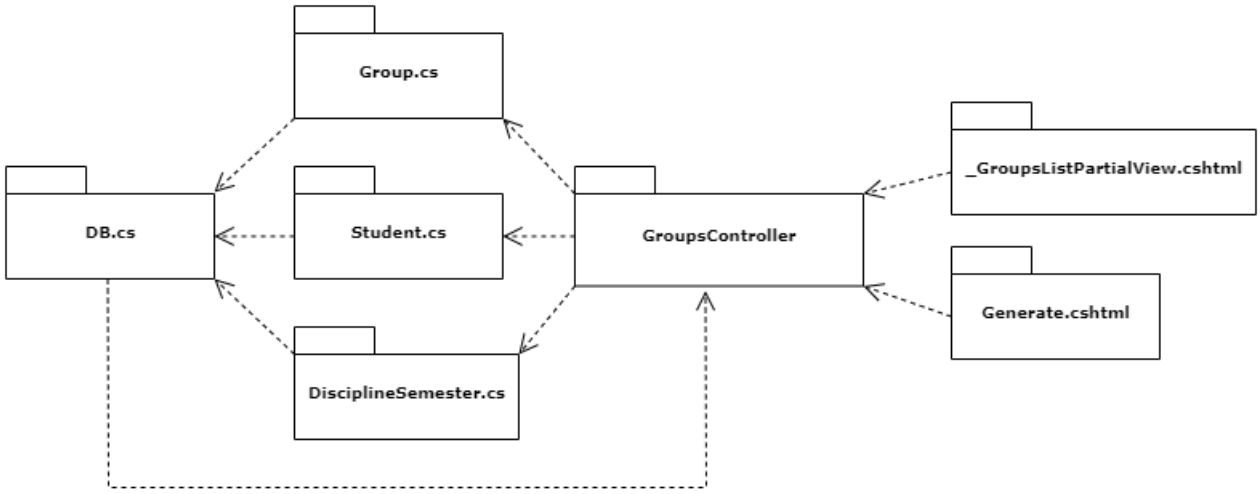

Рисунок 1.4 – Діаграма артефактів (компонентів)

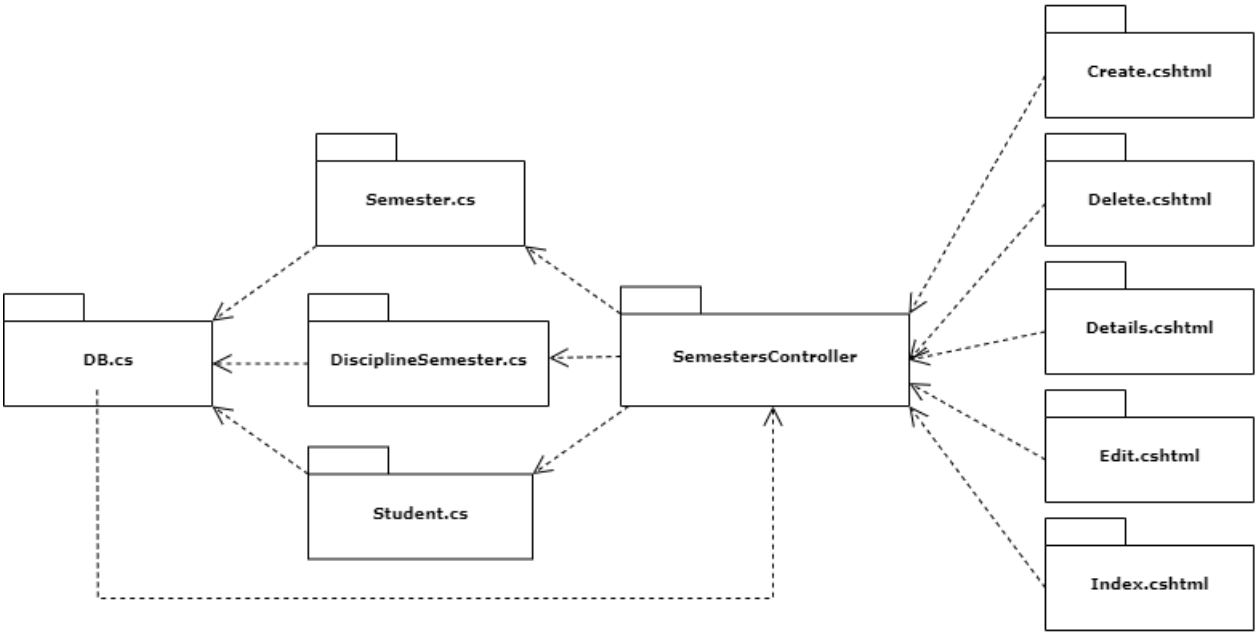

Рисунок 1.5 – Діаграма артефактів (компонентів)

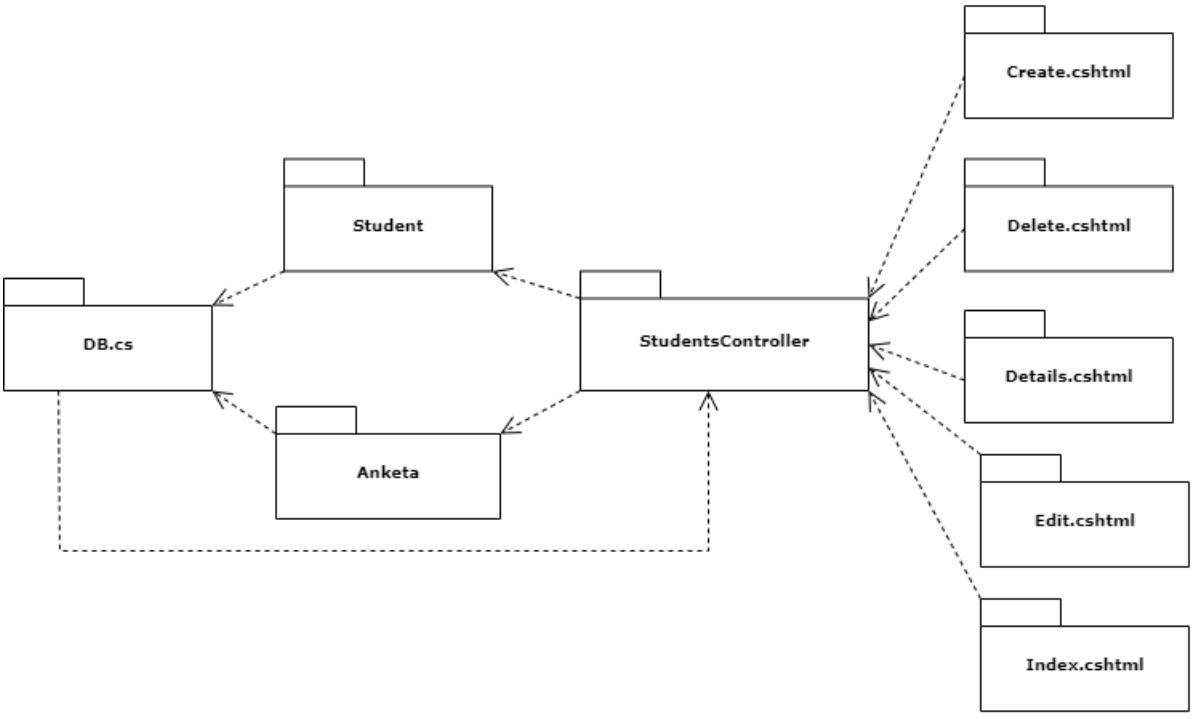

Рисунок 1.6 – Діаграма артефактів (компонентів)

2 ТЕКСТ ПРОГРАМИ

2.1 Модуль AnketasController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MyEdu.Data;
using MyEdu.Models.dbModel;

namespace MyEdu.Controllers
{
    public class AnketasController : Controller
    {
        private readonly EduDbContext _context;

        public AnketasController(EduDbContext context)
        {
            _context = context;
        }

        // GET: Anketas
        public async Task<IActionResult> Index()
        {
            var eduDbContext = _context.Anketas.Include(a => a.DisciplineSemester)
                .ThenInclude(a => a.Discipline)
                .Include(s => s.DisciplineSemester)
                .ThenInclude(s => s.Semester)
                .Include(a => a.Student);
            return View(await eduDbContext.ToListAsync());
        }

        // GET: Anketas/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context.Anketas == null)
            {
                return NotFound();
            }

            var anketa = await _context.Anketas.Include(a => a.DisciplineSemester)
                .ThenInclude(a => a.Discipline)
                .Include(s => s.DisciplineSemester)
                .ThenInclude(s => s.Semester)
                .Include(a => a.Student)
                .FirstOrDefaultAsync(m => m.AnketaId == id);
            if (anketa == null)
            {
                return NotFound();
            }

            return View(anketa);
        }

        // GET: Anketas/Create
        public IActionResult Create()
        {
```

```
            ViewData["DisciplineSemesterId"] = new
SelectList(_context.DisciplineSemesters.Include(ds => ds.Semester).Include(ds =>
ds.Discipline), "DisciplineSemesterId", "NameAndSemester");
            ViewData["StudentId"] = new SelectList(_context.Students, "StudentId",
"FullName");
            return View();
        }

        // POST: Anketas/Create
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("AnketaId,StudentId,DisciplineSemesterId,Priority")] Anketa anketa)
        {
            if (ModelState.IsValid)
            {
                var student = await _context.Students.FindAsync(anketa.StudentId);
                var disciplineSemester = await
_context.DisciplineSemesters.FindAsync(anketa.DisciplineSemesterId);

                if (student.SemesterId != disciplineSemester.SemesterId)
                {
                    ModelState.AddModelError(string.Empty, "Студент та дисципліна повинні
бути з одного семестру.");
                    ViewData["DisciplineSemesterId"] = new
SelectList(_context.DisciplineSemesters.Include(ds => ds.Semester).Include(ds =>
ds.Discipline), "DisciplineSemesterId", "NameAndSemester");
                    ViewData["StudentId"] = new SelectList(_context.Students, "StudentId",
"FullName", anketa.StudentId);
                    return View(anketa);
                }

                _context.Add(anketa);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }

            ViewData["DisciplineSemesterId"] = new
SelectList(_context.DisciplineSemesters.Include(ds => ds.Semester).Include(ds =>
ds.Discipline), "DisciplineSemesterId", "NameAndSemester");
            ViewData["StudentId"] = new SelectList(_context.Students, "StudentId",
"FullName", anketa.StudentId);
            return View(anketa);
        }


        // GET: Anketas/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.Anketas == null)
            {
                return NotFound();
            }

            var anketa = await _context.Anketas.FindAsync(id);
            if (anketa == null)
            {
                return NotFound();
            }
            ViewData["DisciplineSemesterId"] = new
SelectList(_context.DisciplineSemesters.Include(ds => ds.Semester).Include(ds =>
ds.Discipline), "DisciplineSemesterId", "NameAndSemester");
```

```
            ViewData["StudentId"] = new SelectList(_context.Students, "StudentId",
"FullName", anketa.StudentId);
            return View(anketa);
        }

        // POST: Anketas/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("AnketaId,StudentId,DisciplineSemesterId,Priority")] Anketa anketa)
        {
            if (id != anketa.AnketaId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                var student = await _context.Students.FindAsync(anketa.StudentId);
                var disciplineSemester = await
_context.DisciplineSemesters.FindAsync(anketa.DisciplineSemesterId);

                if (student.SemesterId != disciplineSemester.SemesterId)
                {
                    ModelState.AddModelError(string.Empty, "Студент та дисципліна повинні
бути з одного семестру.");
                    ViewData["DisciplineSemesterId"] = new
SelectList(_context.DisciplineSemesters.Include(ds => ds.Semester).Include(ds =>
ds.Discipline), "DisciplineSemesterId", "NameAndSemester");
                    ViewData["StudentId"] = new SelectList(_context.Students, "StudentId",
"FullName", anketa.StudentId);
                    return View(anketa);
                }

                try
                {
                    _context.Update(anketa);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!AnketaExists(anketa.AnketaId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }

            ViewData["DisciplineSemesterId"] = new
SelectList(_context.DisciplineSemesters.Include(ds => ds.Semester).Include(ds =>
ds.Discipline), "DisciplineSemesterId", "NameAndSemester");
            ViewData["StudentId"] = new SelectList(_context.Students, "StudentId",
"FullName", anketa.StudentId);
            return View(anketa);
        }
```

```
        // GET: Anketas/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null || _context.Anketas == null)
            {
                return NotFound();
            }

            var anketa = await _context.Anketas
                .Include(a => a.DisciplineSemester)
                .ThenInclude(ds => ds.Semester)
                .Include(d => d.DisciplineSemester)
                .ThenInclude(d => d.Discipline)
                .Include(a => a.Student)
                .FirstOrDefaultAsync(m => m.AnketaId == id);
            if (anketa == null)
            {
                return NotFound();
            }

            return View(anketa);
        }

        // POST: Anketas/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context.Anketas == null)
            {
                return Problem("Not found this record.");
            }
            var anketa = await _context.Anketas.FindAsync(id);
            if (anketa != null)
            {
                _context.Anketas.Remove(anketa);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool AnketaExists(int id)
        {
          return _context.Anketas.Any(e => e.AnketaId == id);
        }
    }
}
```

2.2 Модуль DisciplinesController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MyEdu.Data;
using MyEdu.Models.dbModel;

namespace MyEdu.Controllers
{
```

```
    public class DisciplinesController : Controller
    {
        private readonly EduDbContext _context;

        public DisciplinesController(EduDbContext context)
        {
            _context = context;
        }

        // GET: Disciplines
        public async Task<IActionResult> Index()
        {
              return View(await _context.Disciplines.ToListAsync());
        }

        // GET: Disciplines/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context.Disciplines == null)
            {
                return NotFound();
            }

            var discipline = await _context.Disciplines
                .FirstOrDefaultAsync(m => m.DisciplineId == id);
            if (discipline == null)
            {
                return NotFound();
            }

            return View(discipline);
        }

        // GET: Disciplines/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: Disciplines/Create
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("DisciplineId,Name")] Discipline
discipline)
        {
            if (ModelState.IsValid)
            {
                _context.Add(discipline);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(discipline);
        }

        // GET: Disciplines/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.Disciplines == null)
            {
                return NotFound();
            }
```

```
            var discipline = await _context.Disciplines.FindAsync(id);
            if (discipline == null)
            {
                return NotFound();
            }
            return View(discipline);
        }

        // POST: Disciplines/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("DisciplineId,Name")]
Discipline discipline)
        {
            if (id != discipline.DisciplineId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(discipline);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!DisciplineExists(discipline.DisciplineId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(discipline);
        }

        // GET: Disciplines/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null || _context.Disciplines == null)
            {
                return NotFound();
            }

            var discipline = await _context.Disciplines
                .FirstOrDefaultAsync(m => m.DisciplineId == id);
            if (discipline == null)
            {
                return NotFound();
            }

            return View(discipline);
        }

        // POST: Disciplines/Delete/5
        [HttpPost, ActionName("Delete")]
```

```
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context.Disciplines == null)
            {
                return Problem("Not found this record.");
            }
            var discipline = await _context.Disciplines.FindAsync(id);
            if (discipline != null)
            {
                _context.Disciplines.Remove(discipline);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool DisciplineExists(int id)
        {
          return _context.Disciplines.Any(e => e.DisciplineId == id);
        }
    }
}
```

2.3 Модуль DisciplineSemestersController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MyEdu.Data;
using MyEdu.Models.dbModel;

namespace MyEdu.Controllers
{
    public class DisciplineSemestersController : Controller
    {
        private readonly EduDbContext _context;

        public DisciplineSemestersController(EduDbContext context)
        {
            _context = context;
        }

        // GET: DisciplineSemesters
        public async Task<IActionResult> Index()
        {
            var eduDbContext = _context.DisciplineSemesters.Include(d =>
d.Discipline).Include(d => d.Semester);
            return View(await eduDbContext.ToListAsync());
        }

        // GET: DisciplineSemesters/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context.DisciplineSemesters == null)
            {
                return NotFound();
            }
```

```
            var disciplineSemester = await _context.DisciplineSemesters
                .Include(d => d.Discipline)
                .Include(d => d.Semester)
                .FirstOrDefaultAsync(m => m.DisciplineSemesterId == id);
            if (disciplineSemester == null)
            {
                return NotFound();
            }

            return View(disciplineSemester);
        }

        // GET: DisciplineSemesters/Create
        public IActionResult Create()
        {
            ViewData["DisciplineId"] = new SelectList(_context.Disciplines, "DisciplineId",
"Name");
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber");
            return View();
        }

        // POST: DisciplineSemesters/Create
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("DisciplineSemesterId,DisciplineId,SemesterId,Description,SeatCount")]
DisciplineSemester disciplineSemester)
        {
            if (ModelState.IsValid)
            {
                _context.Add(disciplineSemester);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["DisciplineId"] = new SelectList(_context.Disciplines, "DisciplineId",
"Name", disciplineSemester.DisciplineId);
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber", disciplineSemester.SemesterId);
            return View(disciplineSemester);
        }

        // GET: DisciplineSemesters/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.DisciplineSemesters == null)
            {
                return NotFound();
            }

            var disciplineSemester = await _context.DisciplineSemesters.FindAsync(id);
            if (disciplineSemester == null)
            {
                return NotFound();
            }
            ViewData["DisciplineId"] = new SelectList(_context.Disciplines, "DisciplineId",
"Name", disciplineSemester.DisciplineId);
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber", disciplineSemester.SemesterId);
            return View(disciplineSemester);
        }
```

```
        // POST: DisciplineSemesters/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("DisciplineSemesterId,DisciplineId,SemesterId,Description,SeatCount")]
DisciplineSemester disciplineSemester)
        {
            if (id != disciplineSemester.DisciplineSemesterId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(disciplineSemester);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!DisciplineSemesterExists(disciplineSemester.DisciplineSemesterId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            ViewData["DisciplineId"] = new SelectList(_context.Disciplines, "DisciplineId",
"Name", disciplineSemester.DisciplineId);
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber", disciplineSemester.SemesterId);
            return View(disciplineSemester);
        }

        // GET: DisciplineSemesters/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null || _context.DisciplineSemesters == null)
            {
                return NotFound();
            }

            var disciplineSemester = await _context.DisciplineSemesters
                .Include(d => d.Discipline)
                .Include(d => d.Semester)
                .FirstOrDefaultAsync(m => m.DisciplineSemesterId == id);
            if (disciplineSemester == null)
            {
                return NotFound();
            }

            return View(disciplineSemester);
        }

        // POST: DisciplineSemesters/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
```

```
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context.DisciplineSemesters == null)
            {
                return Problem("Not found this record.");
            }
            var disciplineSemester = await _context.DisciplineSemesters.FindAsync(id);
            if (disciplineSemester != null)
            {
                _context.DisciplineSemesters.Remove(disciplineSemester);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool DisciplineSemesterExists(int id)
        {
          return _context.DisciplineSemesters.Any(e => e.DisciplineSemesterId == id);
        }
    }
}
```

2.4 Модуль GroupsController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography.Xml;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MyEdu.Data;
using MyEdu.Models.dbModel;
using MyEdu.Models.viewModel;
using Group = MyEdu.Models.dbModel.Group;

namespace MyEdu.Controllers
{
    public class GroupsController : Controller
    {
        private readonly EduDbContext _context;

        public GroupsController(EduDbContext context)
        {
            _context = context;
        }

        // GET: Groups/Generate
        public IActionResult Generate()
        {
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber");
            return View();
        }

        // POST: Groups/Generate
        // To protect from overposting attacks, enable the specific properties you
want to bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
```

```
        [HttpPost]
        public async Task<IActionResult> Generate(GroupsGenVM viewModel)
        {
            List<int> removedDisciplineIdList = new List<int>();
            // Знаходимо анкети, де номер семестру співпадає з вказаним
            var anketaList = await _context.Anketas
                .Include(ds => ds.DisciplineSemester)
                .ThenInclude(s => s.Semester)
                .Include(ds=>ds.DisciplineSemester)
                .ThenInclude(d => d.Discipline)
                .Include(st => st.Student)
                .ThenInclude(s => s.Semester)
                .Where(a => a.Student.Semester.SemesterNumber ==
viewModel.selectedSemesterId)
                .OrderBy(a => a.Priority)
                .ToListAsync();

            // Знаходимо мінімальний і максимальний пріоритет
            int minPriority = 0;
            int maxPriority = 0;

            try {
                minPriority = anketaList.Min(a => a.Priority);
                maxPriority = anketaList.Max(a => a.Priority);
            } catch { }

            List<Group> groupsToSave = new List<Group>();

            try
            {
                var disciplineList = await _context.DisciplineSemesters.Include(x =>
x.Anketas).ToListAsync();

                var emptyAnketasDisciplineId = disciplineList.FindAll(x =>
x.Anketas.All(y => y.Priority != minPriority)).Select(x => x.DisciplineSemesterId);

                removedDisciplineIdList.AddRange(emptyAnketasDisciplineId);

                // Обхід від мінімального до максимального пріоритету
                for (int priority = minPriority; priority <= maxPriority; priority++)
                {
                    // Беремо анкети з поточним пріоритетом
                    var anketaWithPriorityList = anketaList.Where(a => a.Priority ==
priority).ToList();

                    //Формуємо список за пріоритетом priority
                    List<Group> groups = groupsToSave;
                    foreach (var anketa in anketaWithPriorityList)
                    {
                        var group = new Group();
                        group.Student = anketa.Student;
                        group.StudentId = anketa.StudentId;
                        group.DisciplineSemester = anketa.DisciplineSemester;
                        group.DisciplineSemesterId = anketa.DisciplineSemesterId;
                        if (groups.Find(x => x.StudentId == group.StudentId) == null)
                            groups.Add(group);
                    }

                    // Отримуємо кількість людей по дисциплінам
                    var groupWithCount = groups
                        .GroupBy(x => x.DisciplineSemester)
                        .Select(y => new { DisciplineSemester = y.Key, Count =
y.Count() }).ToList();
```

```
                    //Отримуємо групи, які не змогли заповнити
                    var notFullDisciplineGroup = groupWithCount.FindAll(x => x.Count
< x.DisciplineSemester.SeatCount);

                    int minSeatAmongGroups = -1;

                    //Отримуємо дані про кількість людей у групі де набралось
найменше людей
                    if (notFullDisciplineGroup.Count() > 0)
                        minSeatAmongGroups = notFullDisciplineGroup.Min(x =>
x.Count);

                    //Шукаємо мінімальну групу, яку не змогли набрати
                    var minGroup = notFullDisciplineGroup.Find(x => x.Count ==
minSeatAmongGroups);

                    if (minGroup == null)
                        continue;
                    //Розформування групи на інші групи

removedDisciplineIdList.Add(minGroup.DisciplineSemester.DisciplineSemesterId);

                    //Отримуємо список анкет по пріорітету для нашої групи дисципліни
                    var anketaDisciplineSemesterPriorityList =
anketaWithPriorityList.FindAll(x => x.DisciplineSemesterId ==
minGroup.DisciplineSemester.DisciplineSemesterId);

                    //Перебираємо студентів цієї дисципліни
                    foreach (var anketa in anketaDisciplineSemesterPriorityList)
                    {
                        int tempPrior = priority;
                        //Шукаємо анкету студента з пріоритетом + 1
                        Anketa anketaIncPriority = new Anketa();

                        //якщо дисципліна видалена збільшую пріорітет для пошуку
                        while (tempPrior <= maxPriority)
                        {
                            if (anketaIncPriority != null &&
removedDisciplineIdList.Contains(anketaIncPriority.DisciplineSemesterId))
                            {
                                ++tempPrior;
                                continue;
                            }
                            else if (anketaIncPriority != null &&
!removedDisciplineIdList.Contains(anketaIncPriority.DisciplineSemesterId) &&
anketaIncPriority.DisciplineSemesterId != 0)
                            {
                                break;
                            }

                            anketaIncPriority = anketaList
                            .Find(a => (a.Priority == tempPrior + 1) && a.StudentId
== anketa.StudentId);

                            if (tempPrior + 1 > maxPriority)
                            {
                                anketaIncPriority = anketaList
                                    .Find(a => (a.Priority == tempPrior) &&
a.StudentId == anketa.StudentId);
                            }

                            ++tempPrior;
                        }
```

```
                        foreach (var el in removedDisciplineIdList)
                            groups.RemoveAll(x => x.DisciplineSemesterId == el);

                        //Коли знаходжу анкету додаю групу
                        if (anketaIncPriority != null &&
!removedDisciplineIdList.Contains(anketaIncPriority.DisciplineSemesterId))
                        {
                            var group = new Group();
                            group.Student = anketaIncPriority.Student;
                            group.StudentId = anketaIncPriority.StudentId;
                            group.DisciplineSemester =
anketaIncPriority.DisciplineSemester;
                            group.DisciplineSemesterId =
anketaIncPriority.DisciplineSemesterId;
                            groups.Add(group);
                        }
                    }
                }
            }
            catch { }

            try {
                // Отримуємо тих студентів, які не попали до груп
                var studentInGroup = groupsToSave.Select(x => x.StudentId);

                var studentNotInGroup = anketaList.Where(x =>
!studentInGroup.Contains(x.StudentId)).Select(x => x.StudentId);

                foreach (var stud in studentNotInGroup)
                {
                    Student searchedStud = await _context.Students.Include(x =>
x.Anketas).FirstAsync(x => x.StudentId == stud);
                    int maxPrior = searchedStud.Anketas.Max(x => x.Priority);
                    bool added = false;
                    while (!added && maxPrior > 0)
                    {
                        var anket = searchedStud.Anketas.First(x => x.Priority ==
maxPrior);
                        if
(!removedDisciplineIdList.Contains(anket.DisciplineSemesterId))
                        {
                            var group = new Models.dbModel.Group();
                            group.Student = anket.Student;
                            group.StudentId = anket.StudentId;
                            group.DisciplineSemester = anket.DisciplineSemester;
                            group.DisciplineSemesterId = anket.DisciplineSemesterId;
                            groupsToSave.Add(group);
                            added = true;
                        }
                        maxPrior--;
                    }
                }

                // Отримуємо тих студентів, які не попали до жодної з груп, оскільки
вони не сформувались і формуємо одну неповну групу
                studentInGroup = groupsToSave.Select(x => x.StudentId);

                studentNotInGroup = anketaList.Where(x =>
!studentInGroup.Contains(x.StudentId)).Select(x => x.StudentId);

                foreach (var stud in studentNotInGroup)
                {
                    Student searchedStud = await _context.Students.Include(x =>
x.Anketas).FirstAsync(x => x.StudentId == stud);
```

```
                    int maxPrior = searchedStud.Anketas.Max(x => x.Priority);
                    var anket = searchedStud.Anketas.First(x => x.Priority ==
maxPrior);
                    var group = new Models.dbModel.Group();
                    group.Student = anket.Student;
                    group.StudentId = anket.StudentId;
                    group.DisciplineSemester = anket.DisciplineSemester;
                    group.DisciplineSemesterId = anket.DisciplineSemesterId;
                    groupsToSave.Add(group);
                }

            }
            catch { }

            foreach (var el in groupsToSave)
            {
                _context.Add(el);
            }

            var allRecord = await _context.Groups.ToListAsync();

            _context.RemoveRange(allRecord);

            await _context.SaveChangesAsync();

            viewModel.Groups = await _context.Groups.Include(s =>
s.Student).Include(ds => ds.DisciplineSemester).ThenInclude(s =>
s.Semester).ToListAsync();

            return PartialView("~/Views/Groups/_GroupsListPartialView.cshtml",
viewModel);
        }
    }
}
```

2.5 Модуль SemestersController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MyEdu.Data;
using MyEdu.Models.dbModel;

namespace MyEdu.Controllers
{
    public class SemestersController : Controller
    {
        private readonly EduDbContext _context;

        public SemestersController(EduDbContext context)
        {
            _context = context;
        }

        // GET: Semesters
        public async Task<IActionResult> Index()
        {
              return View(await _context.Semesters.ToListAsync());
        }
```

```
        // GET: Semesters/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context.Semesters == null)
            {
                return NotFound();
            }

            var semester = await _context.Semesters
                .FirstOrDefaultAsync(m => m.SemesterId == id);
            if (semester == null)
            {
                return NotFound();
            }

            return View(semester);
        }

        // GET: Semesters/Create
        public IActionResult Create()
        {
            return View();
        }

        // POST: Semesters/Create
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Create([Bind("SemesterId,SemesterNumber")]
Semester semester)
        {
            if (ModelState.IsValid)
            {
                _context.Add(semester);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            return View(semester);
        }

        // GET: Semesters/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.Semesters == null)
            {
                return NotFound();
            }

            var semester = await _context.Semesters.FindAsync(id);
            if (semester == null)
            {
                return NotFound();
            }
            return View(semester);
        }

        // POST: Semesters/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
```

```
        public async Task<IActionResult> Edit(int id, [Bind("SemesterId,SemesterNumber")]
Semester semester)
        {
            if (id != semester.SemesterId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(semester);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!SemesterExists(semester.SemesterId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            return View(semester);
        }

        // GET: Semesters/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null || _context.Semesters == null)
            {
                return NotFound();
            }

            var semester = await _context.Semesters
                .FirstOrDefaultAsync(m => m.SemesterId == id);
            if (semester == null)
            {
                return NotFound();
            }

            return View(semester);
        }

        // POST: Semesters/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context.Semesters == null)
            {
                return Problem("Not found this record.");
            }
            var semester = await _context.Semesters.FindAsync(id);
            if (semester != null)
            {
                _context.Semesters.Remove(semester);
            }

            await _context.SaveChangesAsync();
```

```
            return RedirectToAction(nameof(Index));
        }

        private bool SemesterExists(int id)
        {
          return _context.Semesters.Any(e => e.SemesterId == id);
        }
    }
}
```

2.6 Модуль StudentsController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.EntityFrameworkCore;
using MyEdu.Data;
using MyEdu.Models.dbModel;

namespace MyEdu.Controllers
{
    public class StudentsController : Controller
    {
        private readonly EduDbContext _context;

        public StudentsController(EduDbContext context)
        {
            _context = context;
        }

        // GET: Students
        public async Task<IActionResult> Index()
        {
            var eduDbContext = _context.Students.Include(s => s.Semester);
            return View(await eduDbContext.ToListAsync());
        }

        // GET: Students/Details/5
        public async Task<IActionResult> Details(int? id)
        {
            if (id == null || _context.Students == null)
            {
                return NotFound();
            }

            var student = await _context.Students
                .Include(s => s.Semester)
                .FirstOrDefaultAsync(m => m.StudentId == id);
            if (student == null)
            {
                return NotFound();
            }

            return View(student);
        }

        // GET: Students/Create
        public IActionResult Create()
        {
```

```
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber");
            return View();
        }

        // POST: Students/Create
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult>
Create([Bind("StudentId,LastName,FirstName,SemesterId")] Student student)
        {
            if (ModelState.IsValid)
            {
                _context.Add(student);
                await _context.SaveChangesAsync();
                return RedirectToAction(nameof(Index));
            }
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber", student.SemesterId);
            return View(student);
        }

        // GET: Students/Edit/5
        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.Students == null)
            {
                return NotFound();
            }

            var student = await _context.Students.FindAsync(id);
            if (student == null)
            {
                return NotFound();
            }
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber", student.SemesterId);
            return View(student);
        }

        // POST: Students/Edit/5
        // To protect from overposting attacks, enable the specific properties you want to
bind to.
        // For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
[Bind("StudentId,LastName,FirstName,SemesterId")] Student student)
        {
            if (id != student.StudentId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(student);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
```

```
                {
                    if (!StudentExists(student.StudentId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction(nameof(Index));
            }
            ViewData["SemesterId"] = new SelectList(_context.Semesters, "SemesterId",
"SemesterNumber", student.SemesterId);
            return View(student);
        }

        // GET: Students/Delete/5
        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null || _context.Students == null)
            {
                return NotFound();
            }

            var student = await _context.Students
                .Include(s => s.Semester)
                .FirstOrDefaultAsync(m => m.StudentId == id);
            if (student == null)
            {
                return NotFound();
            }

            return View(student);
        }

        // POST: Students/Delete/5
        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context.Students == null)
            {
                return Problem("Not found this record.");
            }
            var student = await _context.Students.FindAsync(id);
            if (student != null)
            {
                _context.Students.Remove(student);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool StudentExists(int id)
        {
          return _context.Students.Any(e => e.StudentId == id);
        }
    }
}
```

2.7 Модуль Anketa.cs

```
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text.RegularExpressions;
namespace MyEdu.Models.dbModel
{
    [PrimaryKey(nameof(AnketaId))]
    [Index(nameof(StudentId), nameof(DisciplineSemesterId), nameof(Priority), IsUnique =
true)] // Додано атрибут для унікального складного індексу
    public class Anketa
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)] // Додано атрибут для
автоматичного інкрементування
        public int AnketaId { get; set; }
        [DisplayName("Student")]
        public int StudentId { get; set; }

        [DisplayName("Discipline")]
        public int DisciplineSemesterId { get; set; }
        [Range(0, int.MaxValue)]
        public int Priority { get; set; }

        public Student Student { get; set; }
        public DisciplineSemester DisciplineSemester { get; set; }
    }
}
```

2.8 Модуль Discipline.cs

```
using System.ComponentModel.DataAnnotations;

namespace MyEdu.Models.dbModel
{
    public class Discipline
    {
        [Key]
        public int DisciplineId { get; set; }

        public string Name { get; set; }

        public ICollection<DisciplineSemester> DisciplineSemesters { get; set; }
        public ICollection<Anketa> Anketas { get; set; }
    }
}
```

2.9 Модуль DisciplineSemester.cs

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MyEdu.Models.dbModel
{
    public class DisciplineSemester
    {
```

```
        [Key]
        public int DisciplineSemesterId { get; set; }

        [DisplayName("Discipline")]
        public int DisciplineId { get; set; }

        [DisplayName("Semester")]
        public int SemesterId { get; set; }

        public string Description { get; set; }

        [DisplayName("Seat count")]
        [Range(0, int.MaxValue)]
        public int SeatCount { get; set; }

        [NotMapped]

        [DisplayName("Discipline")]
        public string NameAndSemester => Discipline == null || Semester == null ? "":
$"{Discipline.Name} Semester: {Semester.SemesterNumber}";

        [ForeignKey("DisciplineId")]
        public Discipline Discipline { get; set; }

        [ForeignKey("SemesterId")]
        public Semester Semester { get; set; }

        public ICollection<Anketa> Anketas { get; set; }
    }
}
```

2.10 Модуль Group.cs

```
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MyEdu.Models.dbModel
{
    [PrimaryKey(nameof(GroupId), nameof(StudentId), nameof(DisciplineSemesterId))]
    public class Group
    {
        [Key]
        [Column(Order = 1)]
        public int GroupId { get; set; }

        [Key]
        [Column(Order = 2)]

        [DisplayName("Student")]
        public int StudentId { get; set; }

        [Key]
        [Column(Order = 3)]

        [DisplayName("Discipline")]
        public int DisciplineSemesterId { get; set; }

        public Student Student { get; set; }
        public DisciplineSemester DisciplineSemester { get; set; }
    }
}
```

2.11 Модуль Semester.cs

```
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace MyEdu.Models.dbModel
{
    public class Semester
    {
        [Key]
        public int SemesterId { get; set; }
        [DisplayName("Semester number")]
        public int SemesterNumber { get; set; }

        public ICollection<DisciplineSemester> DisciplineSemesters { get; set; }
        public ICollection<Student> Students { get; set; }
    }
}
```

2.12 Модуль Student.cs

```
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace MyEdu.Models.dbModel
{
    public class Student
    {
        [Key]
        public int StudentId { get; set; }
        [DisplayName("Last name")]
        public string LastName { get; set; }

        [DisplayName("First name")]
        public string FirstName { get; set; }

        [DisplayName("Semester")]
        public int SemesterId { get; set; }
        public Semester Semester { get; set; }

        [NotMapped]
        [DisplayName("Full Name")]
        public string FullName => $"{FirstName} {LastName}";
        public ICollection<Anketa> Anketas { get; set; }
    }
}
```

ДОДАТОК Г

ЗАТВЕРДЖУЮ
Проректор
Українського державного університету
науки і технології
Анатолій РАДКЕВИЧ

СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ

Опис програми
ЛИСТ ЗАТВЕРДЖЕННЯ
1116130.01319-01 13 01-ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН

Керівник розробки
_____Олена КУРОП'ЯТНИК

Виконавець
_____Богдан КУШНІР

Нормоконтролер
_____Світлана ВОЛКОВА

2023

ЗАТВЕРДЖЕНО
1116130.01319-01 13 01-ЛЗ

СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ

Опис програми

Листів 30

2023

АНОТАЦІЯ

Документ 1116130.01319-01 13 01 «СИСТЕМА ФОРМУВАННЯ ІНДИВІДУАЛЬНОЇ ОСВІТНЬОЇ ТРАЄКТОРІЇ ЗДОБУВАЧА ОСВІТИ. Опис програми» входить до складу програмної документації на програму, яка представляє собою систему для формування освітньої траєкторії студента.

У даному документі представленні опис програми та її функціональних можливостей. У процесі розробки програмного продукту використовується середовище розробки Visual Studio 2022 Professional, яке забезпечує розширені можливості для розробки програм на різних платформах.

Для управління базами даних використовується система керування базами даних MS SQL Server 2019.

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмна система «система формування індивідуальної освітньої траєкторії здобувача освіти» призначена для надання здобувачам освіти можливості обирати дисципліни та семестри, встановлювати пріоритети для обраних дисциплін та семестрів, а також отримувати інформацію про обрані дисципліни та семестри у зручному вигляді.

Програмне забезпечення необхідне для функціонування:

- фреймворк ASP.Net Core v.7 або вище;
- фреймворк Entity Framework v.7 або вище;
- браузер.

Мови програмування:

- C#;
- JavaScript;
- T-SQL.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Функціональне призначення системи полягає в розробці системи формування індивідуальної освітньої траєкторії здобувача освіти. Програмний продукт має забезпечувати зручний інтерфейс веб-порталу для користування користувачами.

Обмеження системи:

- забезпечення валідації введеної інформації та повідомлення про некоректність введених даних.
- обмежене виконання фільтрації та пошуку.

Функціонально сайт також обмежений використовуваними браузерами:

- Microsoft Edge v.104.2.14.02 та вище;
- Google Chrome v.105.1.21.34 та вище.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1 Алгоритм програми

Основним модулем системи є генерація груп студентів, цей процес має такий алгоритм роботи:

1. Отримати список анкет з бази даних за вказаним номером семестру та відсортувати їх за пріоритетом.
2. Визначити мінімальний та максимальний пріоритет серед анкет.
3. Створити порожній список груп для збереження.
4. Отримати список дисциплін з бази даних разом з анкетами, що до них відносяться.
5. Видалити дисципліни, для яких відсутні анкети з мінімальним пріоритетом, зі списку дисциплін, які не вдається заповнити.
6. Пройти по пріоритетам від мінімального до максимального.
7. Для кожного пріоритету:
8. Отримати анкети з поточним пріоритетом.
9. Створити групи для кожної анкети, якщо студент не належить до жодної іншої групи.
10. Розрахувати кількість людей по дисциплінам у створених групах.
11. Вибрати дисципліну, для якої набралося найменше людей із дисциплін, які не вдалося заповнити.
12. Додати цю дисципліну до списку видалених дисциплін.
13. Отримати анкети з поточною дисципліною з мінімальним пріоритетом.
14. Для кожної анкети:
15. Знайти анкету з наступним пріоритетом для того ж студента.
16. Додати групу, якщо анкета знайдена і дисципліна не видалена.
17. Перевірити студентів, які не потрапили до жодної групи, і додати їх до групи з анкетою максимального пріоритету, якщо дисципліна не видалена.
18. Видалити всі існуючі групи з бази даних.
19. Зберегти нові групи у базі даних.

20. Отримати список груп з бази даних разом із додатковою інформацією.

21. Повернути часткове представлення зі списком груп для відображення.

Цей алгоритм дозволяє ефективно розподілити студентів на групи, дотримуючись пріоритетів заявок та обмежень по кількості студентів у групах. Він також враховує можливість розформування групи і перерозподілу студентів на інші групи у разі, якщо дисципліна не може бути заповнена повністю.

Інші частини роботи програми ґрунтуються на використані архітектурного патерну MVC (Model-View-Controller). Цей патерн дозволяє розділити роботу системи на три основні компоненти: моделі (дані), представлення (вигляд) та контролери (обробники).

3.2 Використані методи

При проектуванні системи формування індивідуальної освітньої траєкторії здобувача освіти використовується архітектурний патерн MVC (Model-View-Controller). Цей патерн дозволяє розділити роботу системи на три основні компоненти: моделі, представлення та контролери.

Моделі

Моделі в системі виконують дві основні функції.

Перші моделі використовуються для представлення елементів ведення даних, з якими працюють користувачі. Вони забезпечують збереження даних, введених користувачем, такі як дисципліни та семестри (View Model).

Другі моделі виконують роль посередника між користувачем та базою даних. Вони зберігають в собі дані з бази даних (Data transfer object).

Представлення

Представлення в системі використовуються для відображення даних моделей у вигляді інтерфейсу користувача.

Представлення також може включати додаткові функції, такі як перевірка вмісту даних чи додаткові можливості відображення.

Контролери

Контролери в системі відповідають за обробку подій у представленні та взаємодію з моделями.

Їх основний обов'язок полягає в обробці подій, що виникають у користувача, і отриманні необхідних даних з моделей для подальшого переходу до відповідних представлень.

Контролери виконують роль сполучника між моделями та представленнями.

3.3 Структура програми

Для наочного відображення складових частин структури програми та їх взаємодії було розроблено діаграми класів (рис. 3.1 – 3.2).

На діаграмах представлено взаємодію класів між собою, через велику кількість класів, діаграми були представлені відносно контролерів сторінок, оскільки контролери є посередником між рівнями даних та представлення, й надають можливість найбільш повно охопити взаємодію між іншими модулями.

Призначення класів:

- AnketasController – використовується для обробки даних пов’язаних з анкетами;
- DisciplinesController – використовується для обробки даних пов’язаних з дисциплінами;
- DisciplineSemestersController – використовується для обробки даних пов’язаних з додатковою інформацією дисциплін;
- GroupsController – використовується для обробки даних пов’язаних з групами на дисципліни;
- SemestersController – використовується для обробки даних пов’язаних з семестрами;
- StudentsController – використовується для обробки даних пов’язаних з студентами;
- Anketa – використовується для збереження даних про анкету;
- Discipline – використовується для збереження даних про дисципліну;

– DisciplineSemester – використовується для збереження даних про додаткові відомості дисципліни;
– Group – використовується для збереження даних про групу на дисципліну;
– Semester – використовується для збереження даних про семестр;
– Student – використовується для збереження даних про студента.

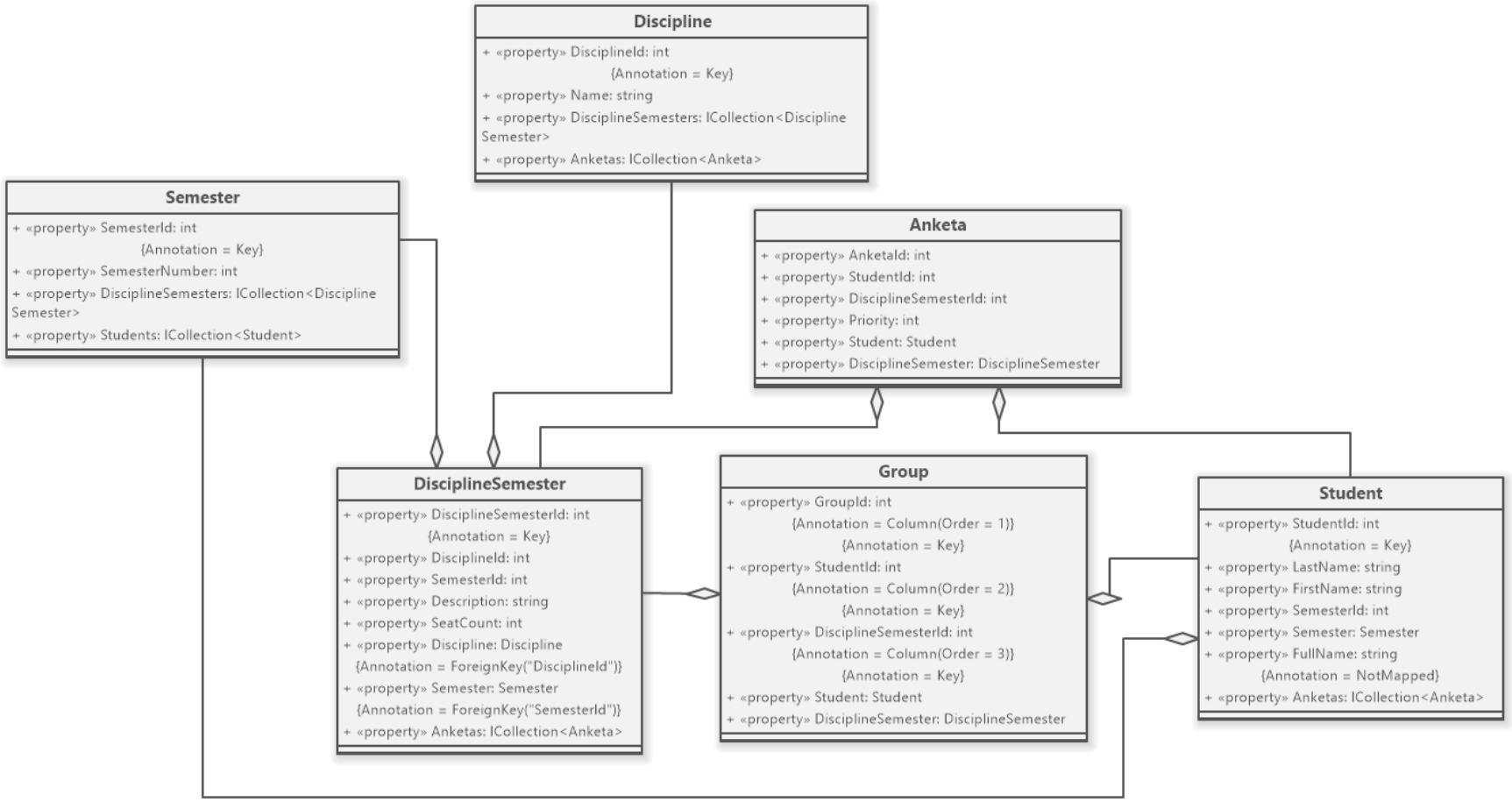

Рисунок 3.1 – Діаграма класів, взаємодія моделей

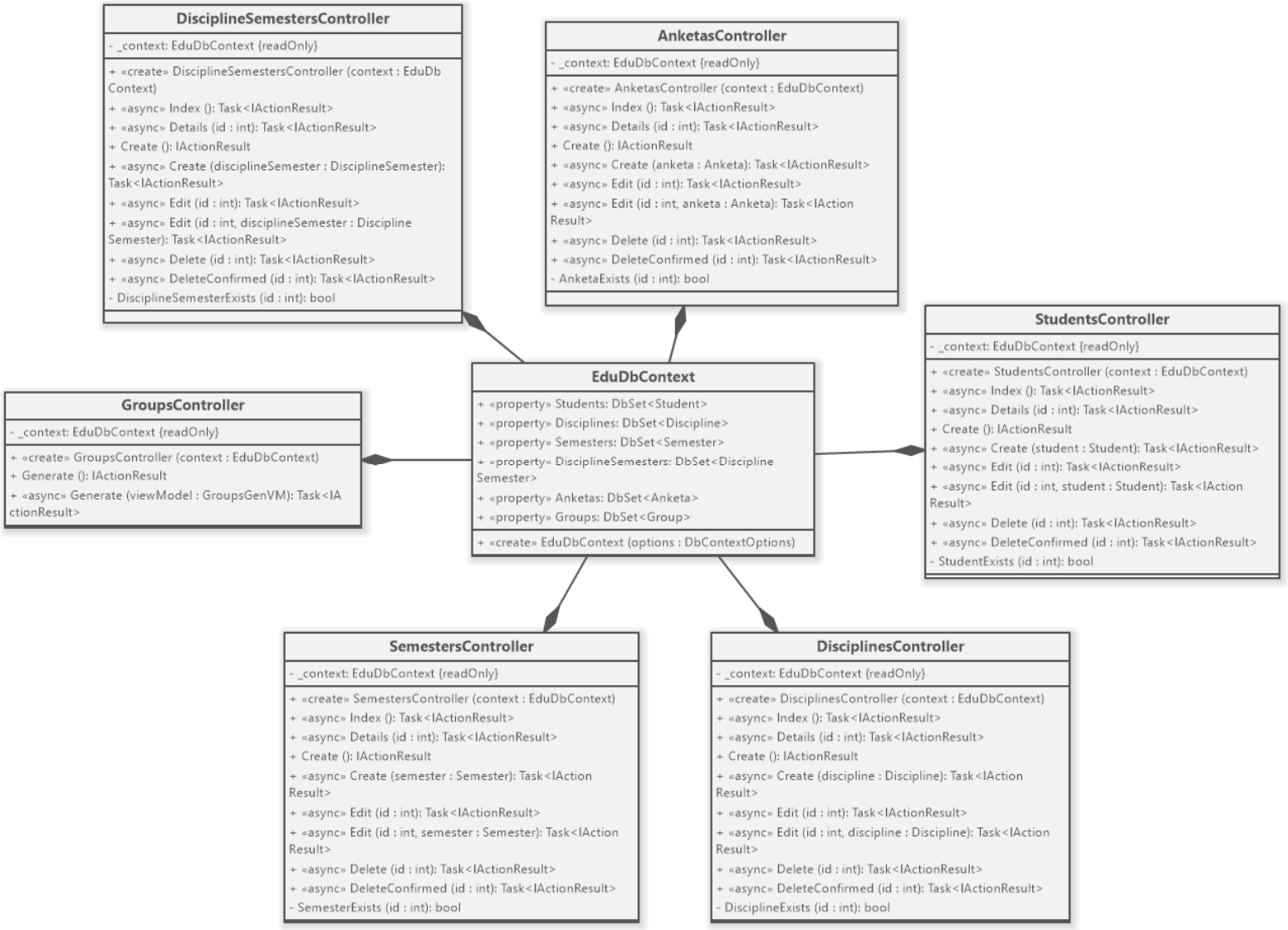

Рисунок 3.2 – Діаграма класів, взаємодія контролерів

3.4 Зв’язки програми з іншими програмами

IIS є потужним веб-сервером, що надає засоби для розгортання та керування веб-сайтами на операційних системах Microsoft він розроблений для адміністрування веб-сайтів на операційних системах Microsoft. Він є вбудованим компонентом операційних систем Windows і надає різноманітні функції для обробки запитів і відповідей на веб-сайтах.

4 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Для використання розроблюваного програмного продукту на різних пристроях та сервері вимагається наступне обладнання:

- мінімум 2 гігабайти оперативної пам'яті;
- браузер , що підтримується програмним продуктом.

Та мобільних пристроях, що мають:

- мінімум 2 гігабайти оперативної пам'яті;
- операційна система Android версії 9 і вище.

Сервер повинен мати:

- мінімум 2 гігабайти оперативної пам'яті;
- процесор: 32 або 64 розрядний процесор з тактовою частотою 1 ГГц або вище, який підтримує набір інструкцій SSE2;
- мінімум 50 гігабайт простору на жорсткому диску;
- засоби контролю, такі як миша, віддалений робочий стіл, клавіатура;
- програвач CD-R;
- монітор.

5 ВИКЛИК ЗАВАНТАЖЕННЯ

Для того, щоб користуватись програмним продуктом необхідно перейти на сайт за посиланням http://mytrayektori.com, або розгорнути його локально на використовуваному комп’ютері.

Для того, щоб підтримувати сайт онлайн треба постійно оновлювати підписку Azure Portal, тому за потреби замовника, сайт може бути тимчасово недоступний.

6 ВХІДНІ ДАНІ

Вхідними даними програми є властивості сутностей. Властивості сутностей описані у вигляді ER-діаграми (рис. 6.1).

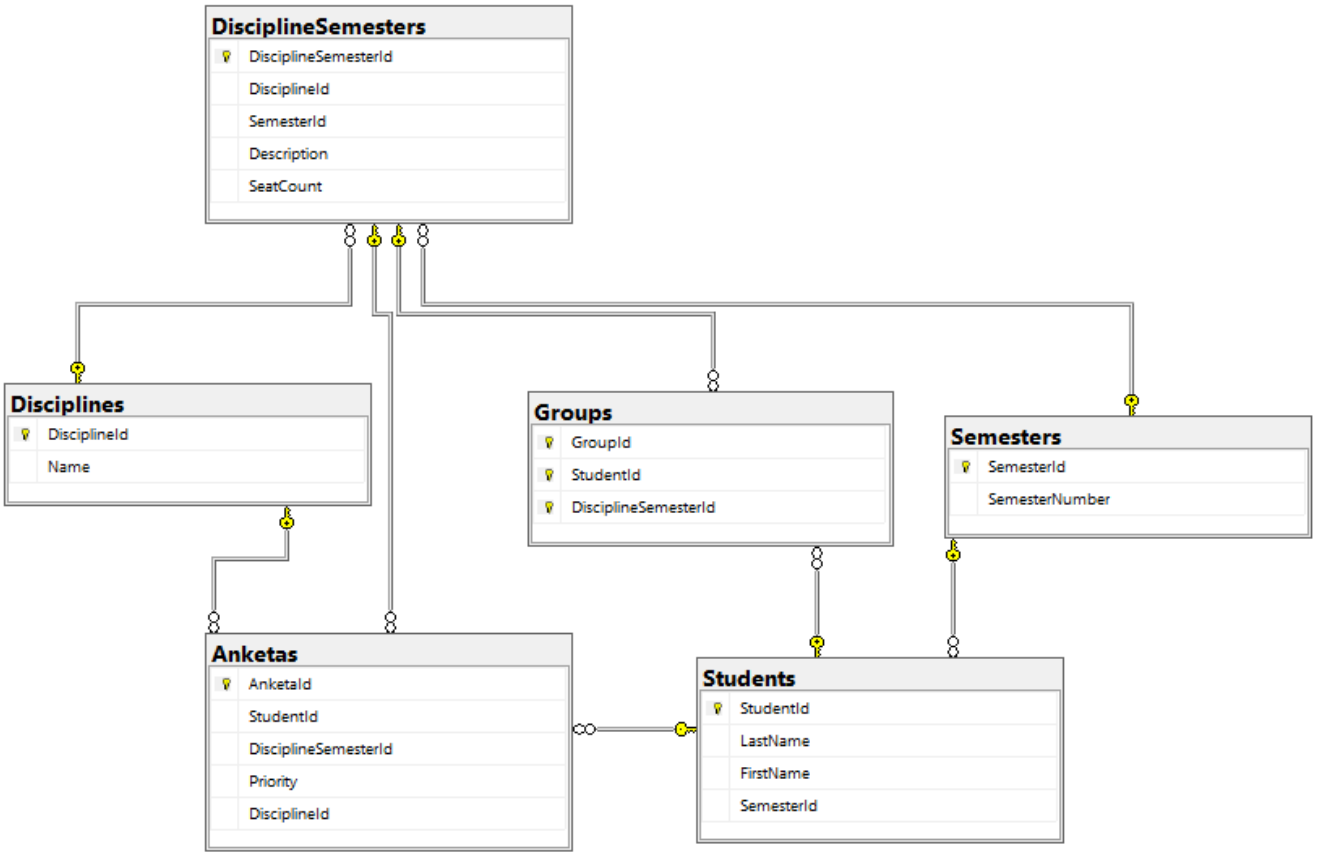

Рисунок 6.1 – ER-модель, що відображає сутності та їх атрибути

Дані вводяться з клавіатури, або, за наявності екранної клавіатури, з використанням екранного маніпулятора.

7 ВИХІДНІ ДАНІ

Вихідними даними програми є:

- згенеровані сторінки з даними сутностей;
- повідомлення про роботу програми;
- поля у базі даних.

8 ОПИС ПРИЗНАЧЕНОГО ДЛЯ КОРИСТУВАЧА ІНТЕРФЕЙСУ

8.1 Опис станів програми

Стани, в яких може знаходитись програма наведено у табл. 8.1.

Таблиця 8.1 – Стани програми

№	Стан	Опис стану	Рекомендовані дії
1	2	3	4
1	Завантаження сторінки	Формується запит на створення представлення	Чекати відображення сторінки
2	Додання запису	На вхід події контролера приходить одна з моделей, програма обробляє дані пов’язані з моделлю і повертає відповідь у вигляді переадресування на сторінку списку обраних сутностей, якщо дані було додано	Чекати відображення списку
3	Додання запису	На вхід події контролера приходить одна з моделей, програма обробляє дані пов’язані з моделлю і виявляється так, що було отримано некоректні дані, програма повертає користувача на сторінку введення даних	Чекати завантаження попередніх даних
4	Редагування запису	На вхід події контролера приходить одна з моделей, програма обробляє дані пов’язані з моделлю і повертає відповідь у вигляді переадресування на сторінку списку обраних сутностей, якщо дані було збережено	Чекати відображення списку
5	Редагування запису	програма обробляє дані пов’язані з моделлю і виявляється так, що було отримано некоректні дані, програма повертає користувача на сторінку введення даних	Чекати завантаження попередніх даних
6	Видалення запису	На вхід події контролера приходить одна з моделей, програма обробляє дані пов’язані з моделлю і видаляє запис та повертає відповідь у вигляді переадресування на сторінку списку обраних сутностей, якщо дані було видалено	Чекати відображення списку

Продовження таблиці 8.1

1	2	3	4
7	Видалення запису	На вхід події контролера приходить одна з моделей, програма обробляє дані пов’язані з моделлю і виявляється так, що було отримано некоректні дані, програма переводить користувача на сторінку помилки	Чекати завантаження сторінки
8	Перегляд запису	На вхід події контролера приходить одна з моделей, програма обробляє дані пов’язані з моделлю і виявляється так, що було отримано некоректні дані, програма переводить користувача на сторінку помилки	Чекати завантаження сторінки
9	Завантаження не існуючої події	У випадку коли користувач намагається отримати сторінку, якої не існує, роутер сайту поверне користувача на сторінку «Не знайдено»	Чекати завантаження сторінки
10	Обрання семестру для генерації груп	Контролер, який відповідає за роботу з групами отримує номер семестру, після чого починає формувати групи під цей семестр, в результаті він повертає сформовані групи, які далі представлення виводить	Чекати завантаження списку груп, або напису про неможливість створити групи за наявними анкетами
11	Сформовані групи не повні	Після формування груп, може виявитись, що групи були неповними, тоді програма розподілить студентів по групам відносно пріоритету, з вказанням, що групу відкрити неможливо	Після завантаження можливих груп, треба перевірити дані анкет, чи не було допущено якусь помилку при заповнені
12	Студента немає в жодному списку	Формування груп не розподілило студента в жодну з наявних дисциплін	Треба перевірити коректність заповнених анкет студента

8.2 Опис керування діалогом

Після завантаження сторінки сайту, користувачу доступний короткий опис призначення сайту та його можливості.

Далі користувач може перейти до керування записами про студентів, керування записами про дисципліну, керування додатковою інформацією про дисципліни, керування анкетами, керування семестрами та до генерації груп.

Кожна складова частина пов’язана з керуванням передбачає такі дії як: відображення записів, перегляд деталей запису, видалення запису, редагування запису. Кожна з цих дій – окрема сторінка, яка відображає дані й можливості відповідно до вказаної дії.

Коли всі необхідні дані заповнено користувач може перейти до генерації груп.

Під час генерації може виникнути декілька сценаріїв роботи. Якщо всі анкети заповнені правильно та вони містять розподіл по пріоритету від 1, то студентів буде розміщено по групам, коли ж анкети не мають пріоритету в 1, проте є в наявності інші анкети по дисциплінам, ці данні вважаються некоректними, тому цього студента буде проігноровано. Після генерації груп, можуть з’явитись студенти, які не потрапили до жодної з груп з причини неможливості відкрити таку, через пріоритети інших студентів, тоді програма надає також список зі студентам та дисциплінами на які неможливо відкрити набір, але найімовірніше добрати групу.

Для генерації по семестру користувачу просто необхідно обрати семестр з випадаючого списку.

При формуванні анкет може виникнути ситуація з коли студент знаходиться не на тому семестрі, на якому проводиться дисципліна, в такому випадку програма повідомить користувача, що неможливо створити таку анкету.

Для виходу з сайту треба закрити відповідну вкладку браузера.

8.3 Формування екранів

Спроектовані сторінки сайту відображенні на рис. 8.1 – 8.14.

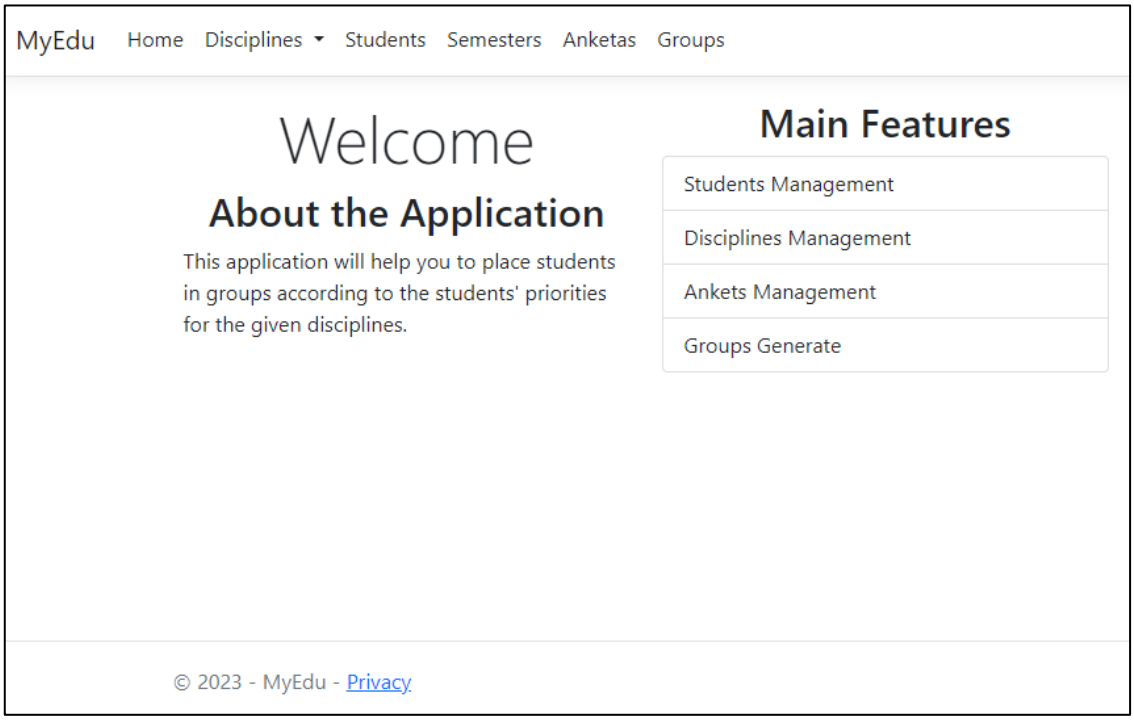

Рисунок 8.1 – Стартова сторінка сайту

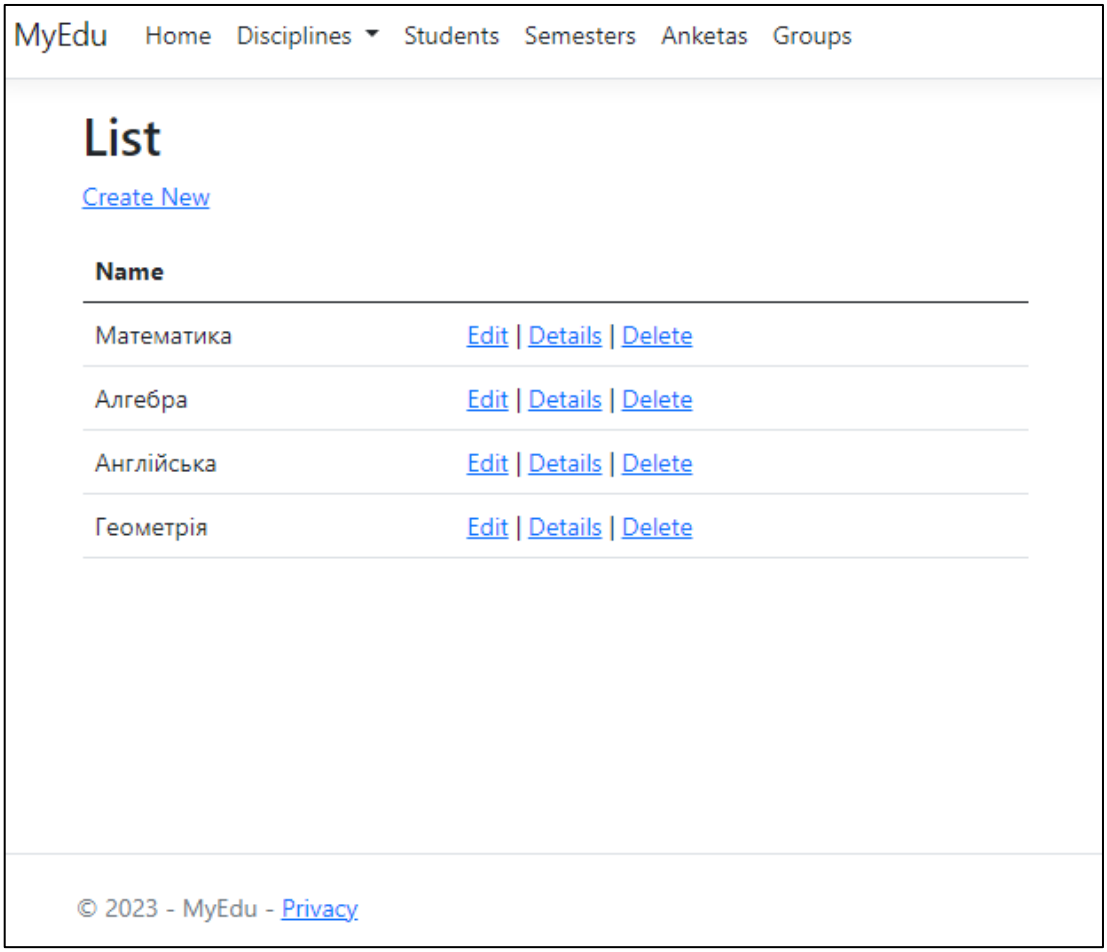

Рисунок 8.2 – Сторінка дисциплін

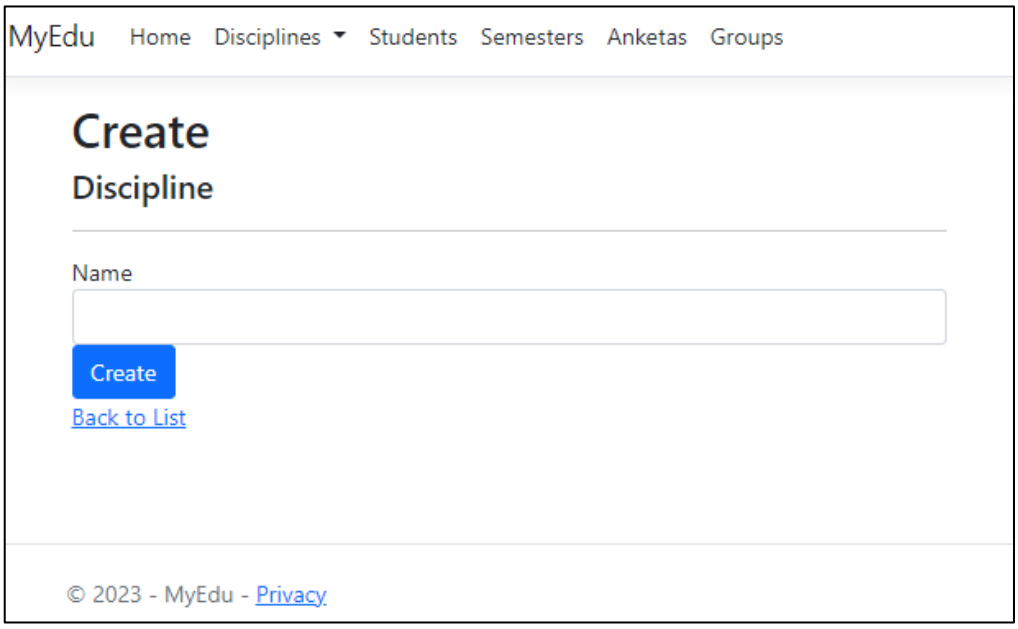

Рисунок 8.3 – Створення дисципліни

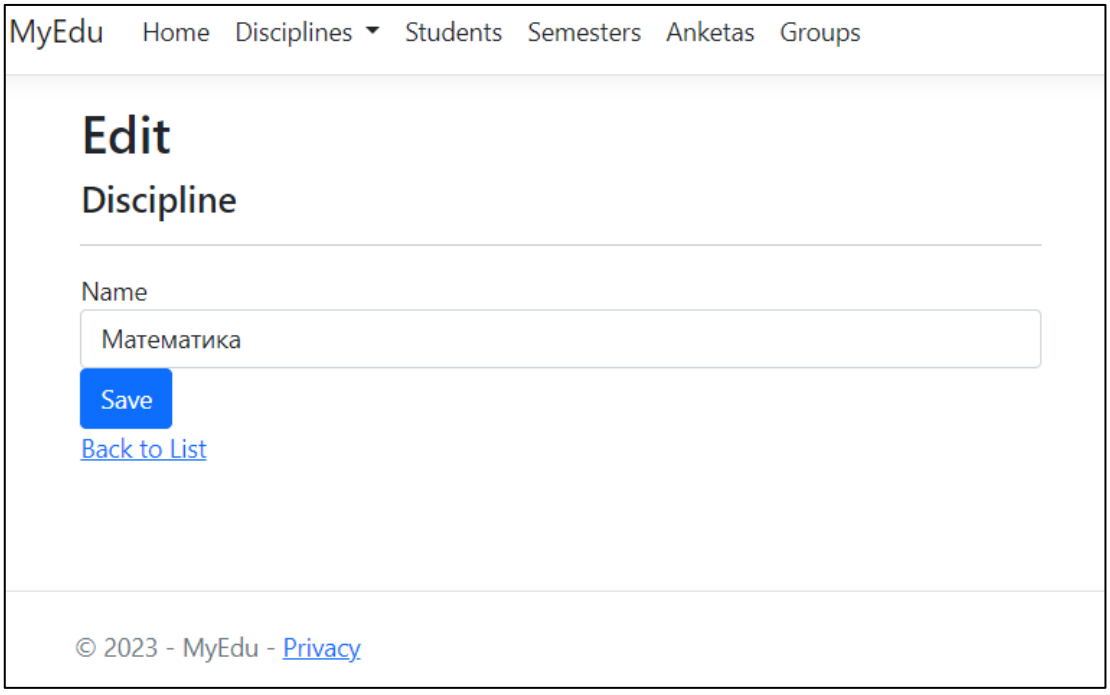

Рисунок 8.4 – Редагування дисципліни

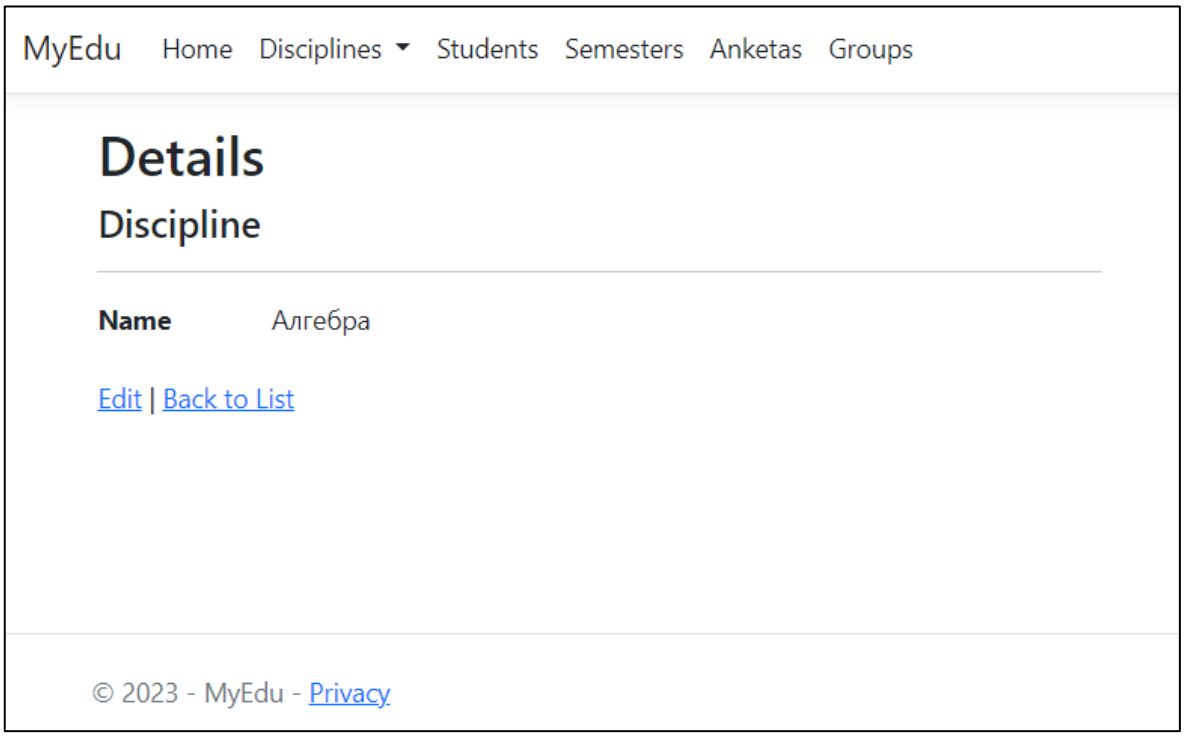

Рисунок 8.5 – Перегляд деталей про дисципліну

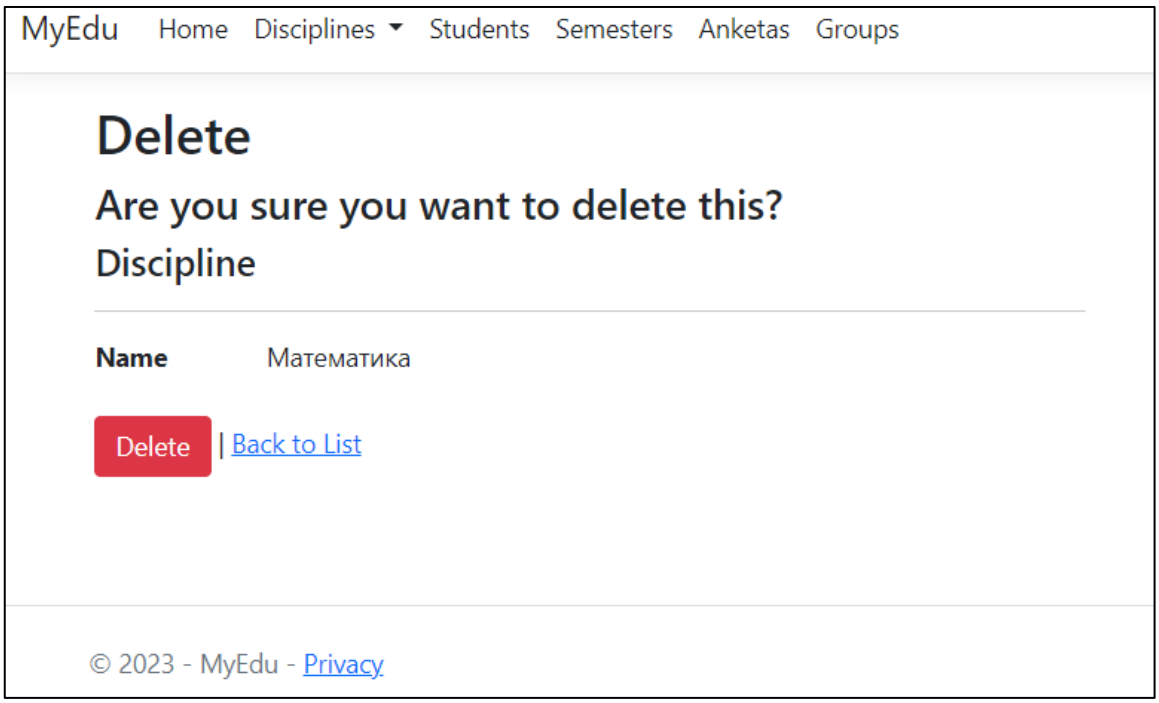

Рисунок 8.6 – Видалення дисципліни

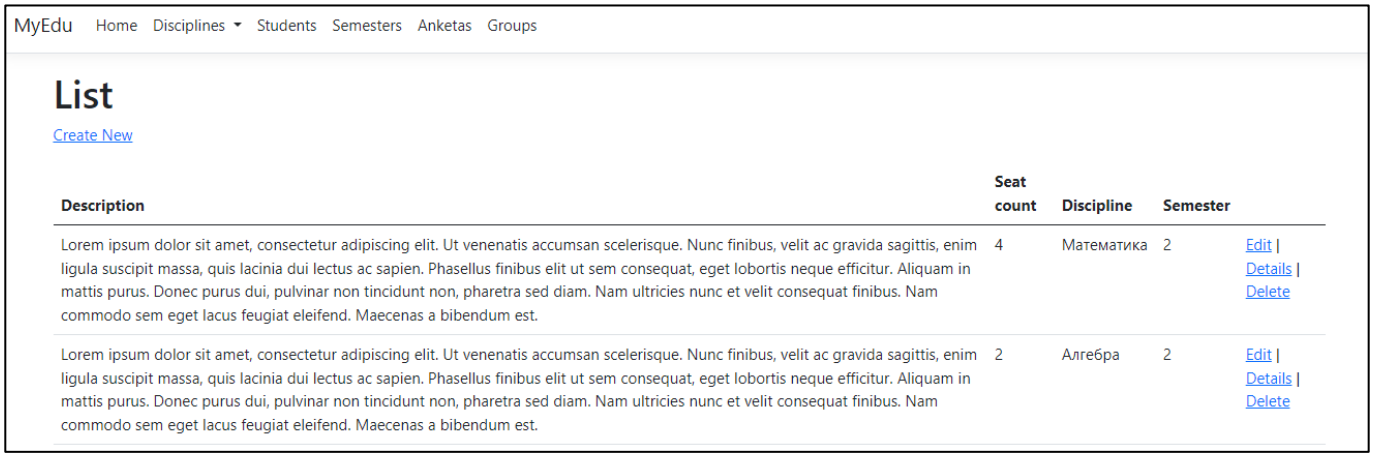

Рисунок 8.7 – Сторінка додаткових налаштувань до дисципліни відносно семестру викладання

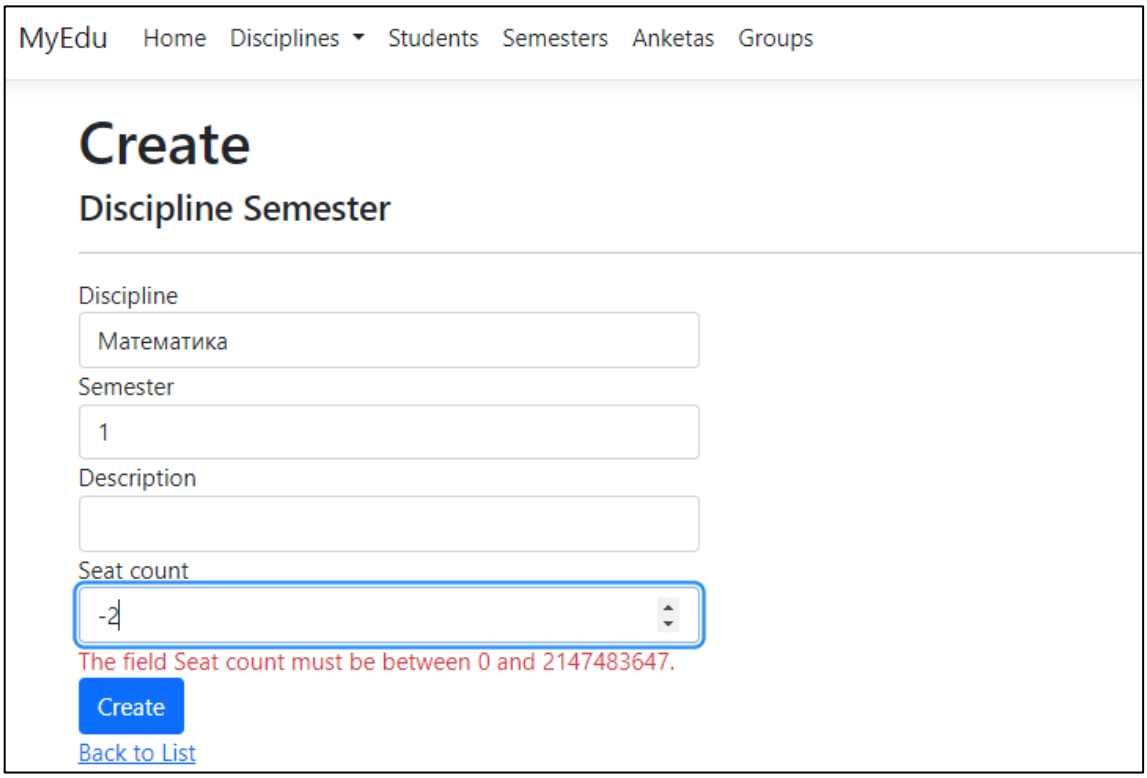

Рисунок 8.8 – Редагування інформації

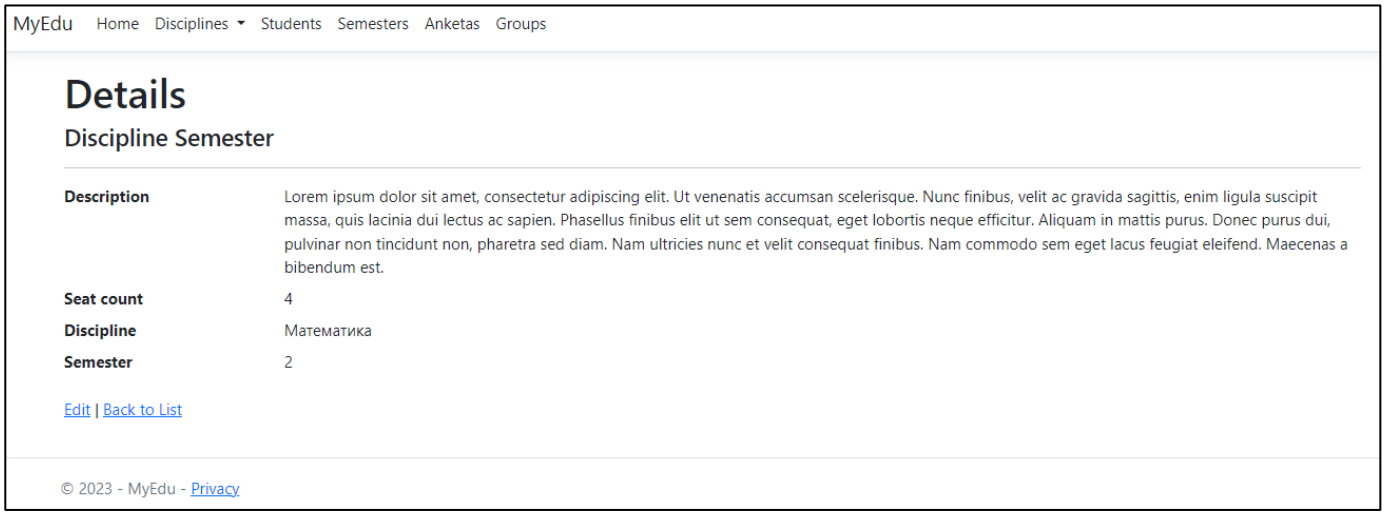

Рисунок 8.9 – Перегляд деталей

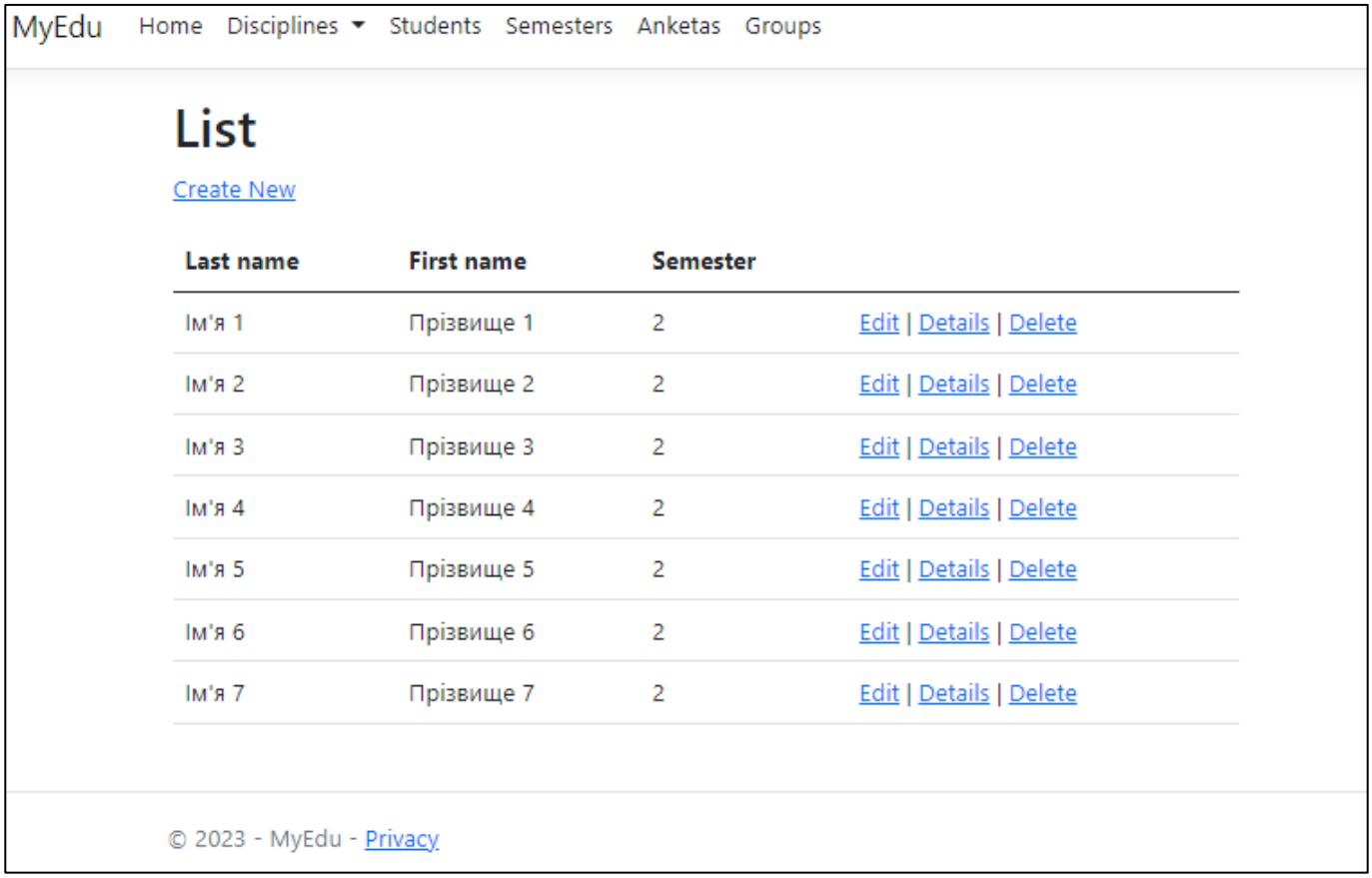

Рисунок 8.10 – Список студентів

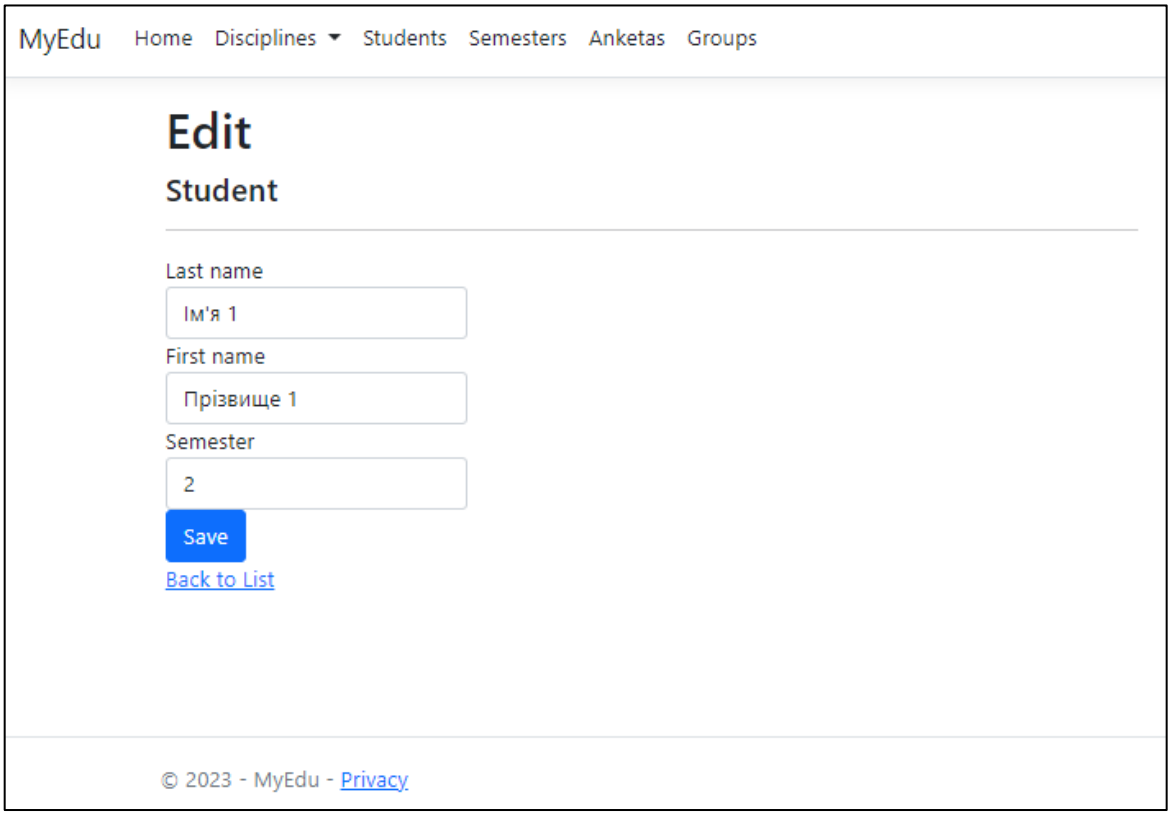

Рисунок 8.11 – Редагування студента

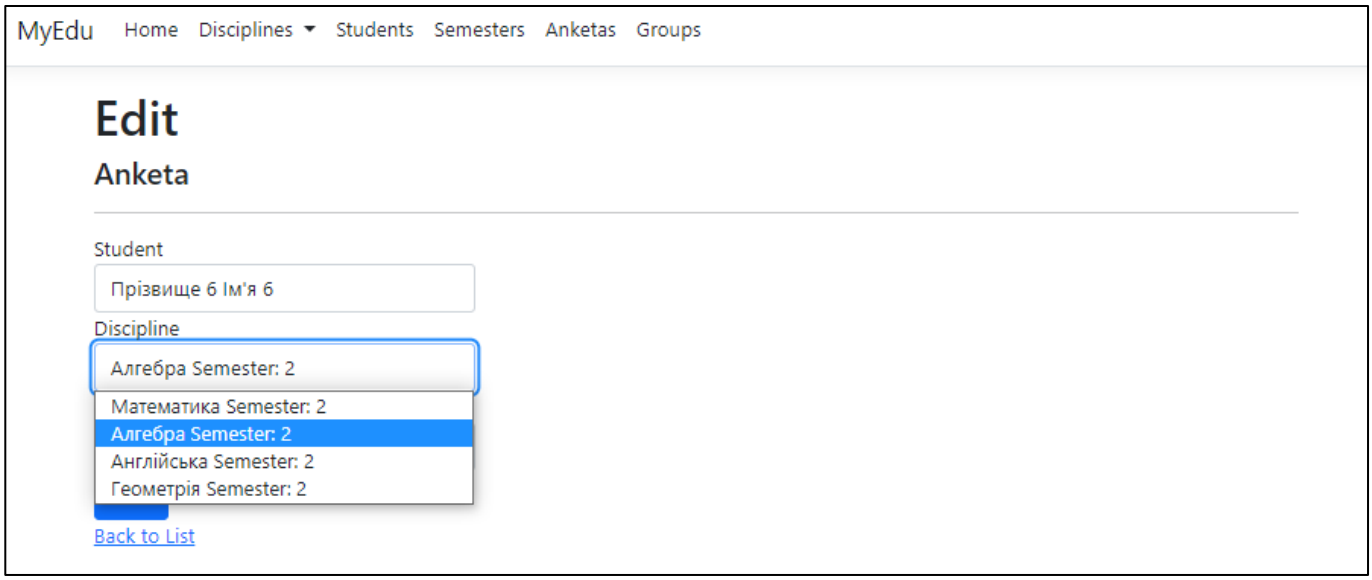

Рисунок 8.12 – Редагування анкети

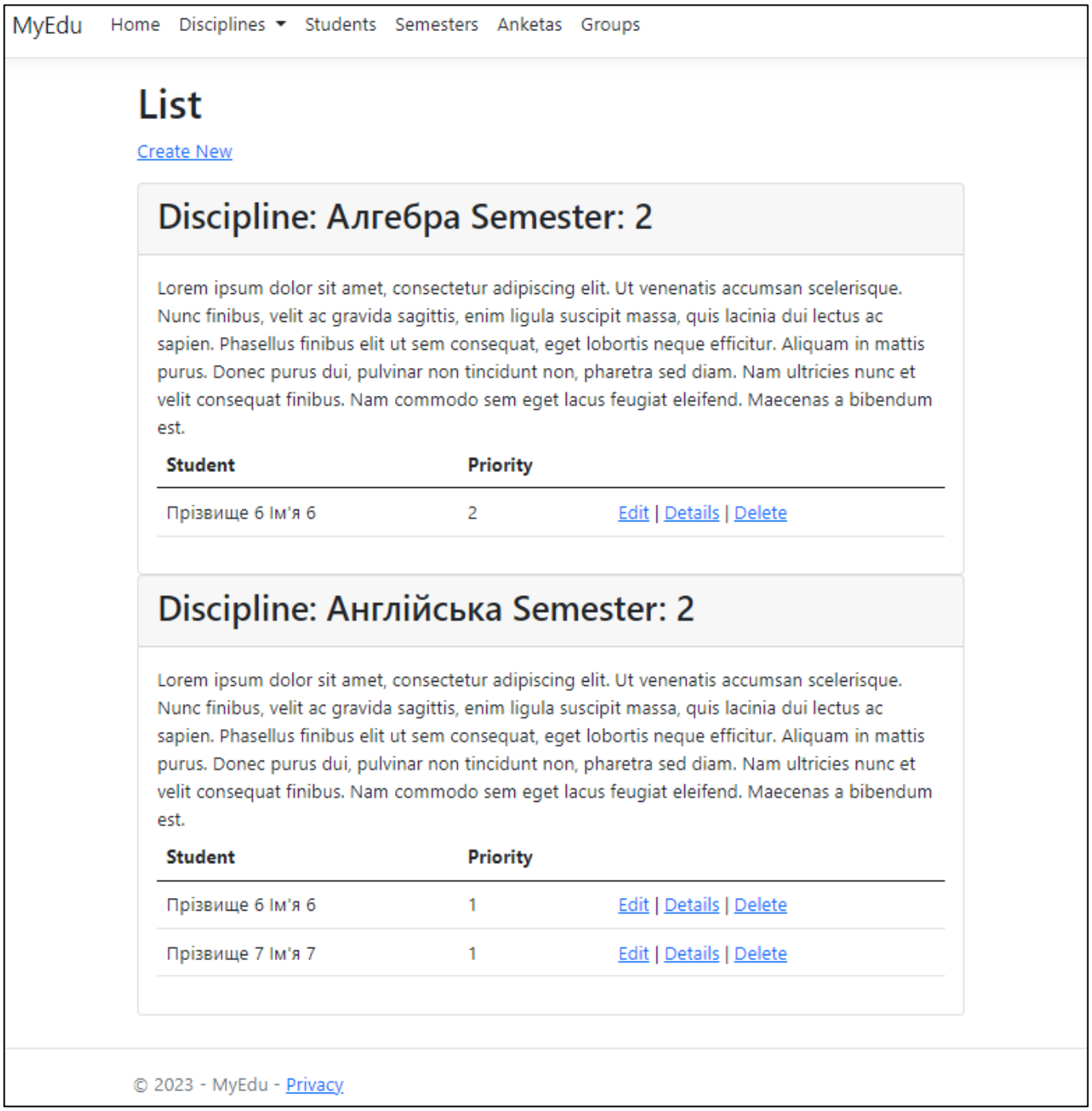

Рисунок 8.13 – Список анкет

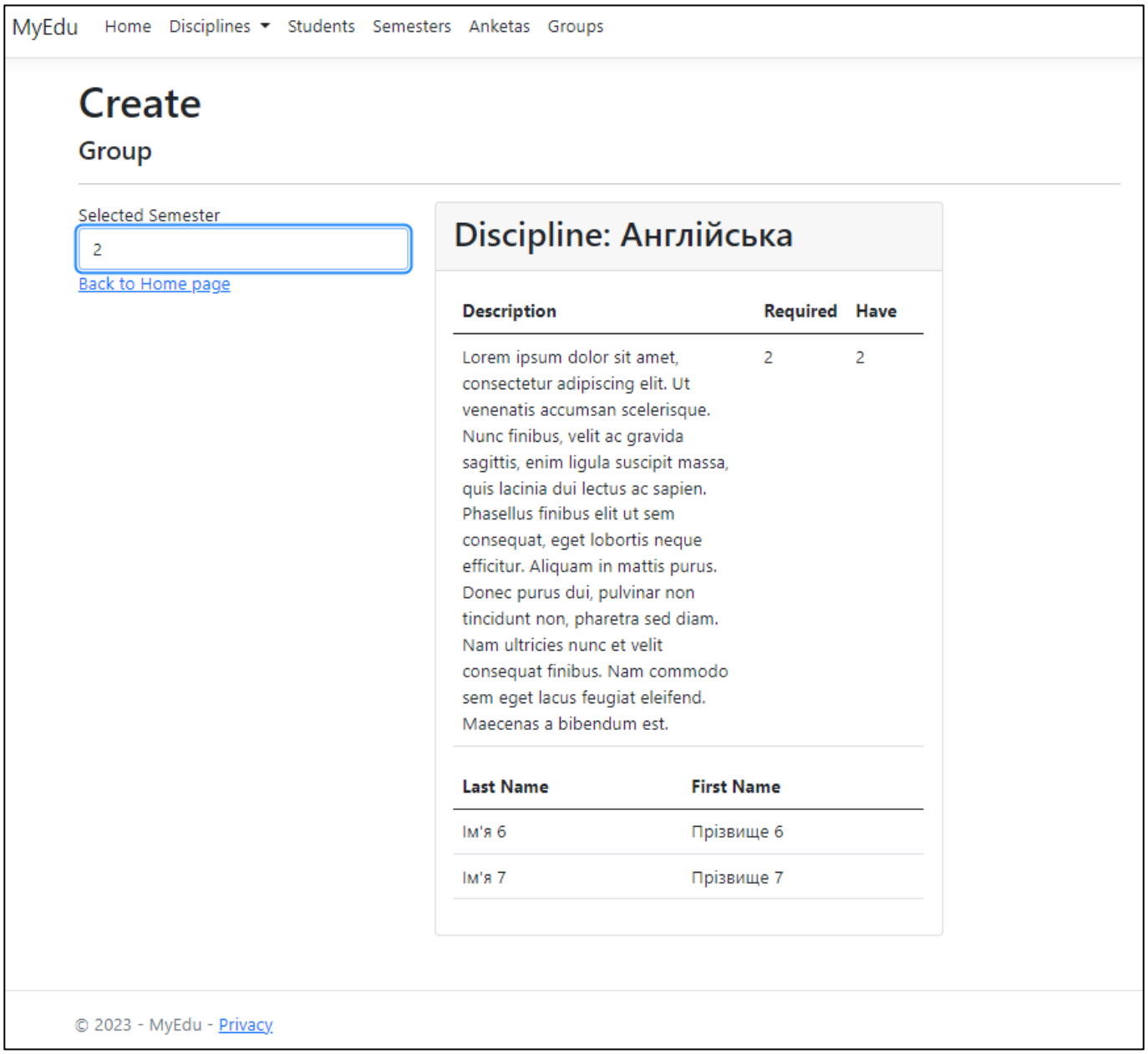

Рисунок 8.14 – Згенерована група

9 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

У разі виникнення питань, пов’язаних з сайтом, можна написати до телеграму за посиланням: https://t.me/BohdanKushnir. Підтримка з 12:00 до 18:00 у будні дні.

10 ПОВІДОМЛЕННЯ

В табл. 10.1 приведені повідомлення користувачу в залежності від ситуацій, що виникли, а також рекомендовані дії.

Таблиця 10.1 – Повідомлення системи

Текст	Адресат	Ситуація	Рекомендовані дії
1	2	3	4
Некоректно заповнені дані	Користувач	Дані моделі були заповнені не відповідно до даних, які треба було ввести	Ввести коректні дані, в залежності від поля з помилкою
Запис не знайдено	Користувач	Користувач намагався працювати з даними, яких немає в системі	Оновити сторінку та спробувати ще раз, перевірити введені поля
Сторінку не знайдено	Користувач	Спроба перейти на сторінку яка відсутня у системі	Обрати іншу сторінку