



Метадані

ДОКУМЕНТ

Заголовок

Бак_пояснювальна_Бережной

Автор

Бережной

Науковий керівник / Експерт

ІД документу

334320281

ОРГАНІЗАЦІЯ

Назва організації

Taras Shevchenko National University of Luhansk

підрозділ

Taras Shevchenko National University of Luhansk

ЗВІТ

Дата звіту

6/15/2026

Дата редагування

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

2.26%

2.26%

КП 1

21183

Кількість слів

2.37%

2.37%

КЦ

184518

Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв



4

Інтервали



0

Мікропробіли



182

Білі знаки



0

Парафрази (SmartMarks)



37

Джерела

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз

Колір тексту

#	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	http://inmad.vntu.edu.ua/portal/static/4B809FDE-F630-454E-8085-656607677BDE.pdf	17 (0.08 %)

2	https://cloud.tencent.com/developer/ask/sof/331262	17 (0.08 %)
3	Echkenko_Shchebenyuk_Diplom_2026 6/1/2026 V. N. Karazin Kharkiv National University (KKNU) (ННІ комп'ютерних наук та штучного інтелекту - кафедра математичного моделювання та аналізу даних)	17 (0.08 %)
4	https://cloud.tencent.com/developer/ask/sof/331262	16 (0.08 %)
5	https://github.com/dotnet/efcore/issues/23344	16 (0.08 %)
6	Каталог товарів із характеристиками та посиланнями на покупку 4/29/2025 Rivne Applied College of Information Technologies (Rivne Applied College of Information Technologies)	16 (0.08 %)
7	https://github.com/dotnet/efcore/issues/19813	15 (0.07 %)
8	https://ela.kpi.ua/bitstream/123456789/44080/1/Zagrebelna_bakalavr.pdf	15 (0.07 %)
9	CRM-система для підтримки роботи адміністратора фітнесстудії 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	14 (0.07 %)
10	Курсова Маруда 6/1/2026 Limited liability company "Holosiiv economy-law professional college" (Limited liability company "Holosiiv economy-law professional college")	14 (0.07 %)

База даних RefBooks



#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
---	-----------	--

Домашня база даних



#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
---	-----------	--

Програма обміну базами даних (1.35 %)



#	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
---	-----------	--

1	Курсова Маруда 6/1/2026 Limited liability company "Holosiiv economy-law professional college" (Limited liability company "Holosiiv economy-law professional college")	56 (7) (0.26 %)
2	БР Жила В.С. ICT 41 Проектування та розробка інформаційної системи для автоматизації обліку діяльності АЗС.DOCX 6/21/2025 National University of Water and Environmental Engineering (National University of Water and Environmental Engineering)	38 (6) (0.18 %)
3	Курсова Робота ООП Пугач Антон, н.кер. Яровий Р.О. 4/29/2026 European University (European University)	35 (5) (0.17 %)
4	Система обліку студентів і груп 5/25/2026 Establishment of the Lviv Regional Council "Sambir Professional Pedagogical College Named after Ivan Fvlvnochak" (Sambir Professional Pedagogical College named after Ivan Fvlvnochak)	31 (4) (0.15 %)

5	CRM-система для підтримки роботи адміністратора фітнесстудії 3/16/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	28 (3) (0.13 %)
6	Каталог товарів із характеристиками та посиланнями на покупку 4/29/2025 Rivne Applied College of Information Technologies (Rivne Applied College of Information Technologies)	21 (2) (0.1 %)
7	Echkenko_Shchebenyuk_Diplom_2026 6/1/2026 V. N. Karazin Kharkiv National University (KKNU) (ННІ комп'ютерних наук та штучного інтелекту - кафедра математичного моделювання та аналізу даних)	17 (1) (0.08 %)
8	Диплом Микуленко Д.А. 6/10/2026 Zaporizhzhia National University (Кафедра електроніки, інформаційних систем та програмного забезпечення (спеціальність 121 Інженерія програмного забезпечення))	17 (2) (0.08 %)
9	Крутогуз Дмитро Сергійович_Розробка інформаційної системи обліку та аналізу відвідуваності студентів_122_бак_денна_наук. кер. Хайдуров В.В. 5/7/2026 European University (European University)	13 (1) (0.06 %)
10	Григорів_кваліфікаційна_на антиплагіат 5/15/2025 King Danylo University (King Danylo University)	12 (2) (0.06 %)
11	Облік підписок та платежів. 5/29/2026 Separate structural department "Lviv Professional College of Lviv National Environmental University" (Відокремлений структурний підрозділ "Львівський фаховий коледж Львівського національного університету природокористування")	11 (1) (0.05 %)
12	Курсова 2 фуд.docx 12/16/2025 National University of Water and Environmental Engineering (National University of Water and Environmental Engineering)	7 (1) (0.03 %)

Інтернет (0.91 %)



#	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
13	https://cloud.tencent.com/developer/ask/sof/331262	41 (3) (0.19 %)
14	https://www.visualchart.com/ContentManagement/Development/Manuals/EN/Fundamentals_of_Computer_Programming_with_CSharp.pdf	36 (4) (0.17 %)
15	https://github.com/dotnet/efcore/issues/23344	25 (2) (0.12 %)
16	https://github.com/dotnet/efcore/issues/19813	20 (2) (0.09 %)
17	http://inmad.vntu.edu.ua/portal/static/4B809FDE-F630-454E-8085-656607677BDE.pdf	17 (1) (0.08 %)
18	https://ela.kpi.ua/bitstream/123456789/44080/1/Zagrebelna_bakalavr.pdf	15 (1) (0.07 %)
19	https://dev.to/muhammad_salem/associations-in-ef-core-14d3	15 (2) (0.07 %)
20	https://romiller.com/2010/07/18/ef-ctp4-tips-tricks-mapping-to-an-existing-database/	12 (1) (0.06 %)
21	https://wirefuture.com/post/how-to-use-entity-framework-core-in-net-8-efficiently	12 (1) (0.06 %)



Список прийнятих фрагментів

#	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)
---	-------	---------------------------------------

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЗ «ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА»

Факультет технологій та інформаційних систем
(назва факультету, інституту)
Кафедра інформаційних технологій та систем
(назва кафедри)

Пояснювальна записка до кваліфікаційної роботи
за першим (бакалаврським) рівнем освіти

на тему:

РОЗРОБКА СИСТЕМИ РЕЄСТРАЦІЇ ВІДВІДУВАННЯ ЗАНЯТЬ СТУДЕНТАМИ НАВЧАЛЬНОЇ ГРУПИ

Виконав: здобувач вищої освіти 4 курсу спеціальності

F2 «Інженерія програмного забезпечення»

(шифр і назва напрямку підготовки, спеціальності)

Олексій БЕРЕЖНОЙ

(прізвище та ініціали) Керівник

Ольга

СМАГІНА

(прізвище та ініціали) Рецензент

Владислав ІЩЕНКО

(прізвище та ініціали)

Лубни - 2026

Зміст

ВСТУП 3

РОЗДІЛ 1. АНАЛІЗ ПІДХОДІВ ТА ІНСТРУМЕНТІВ ДЛЯ РЕЄСТРАЦІЇ ВІДВІДУВАННЯ ЗАНЯТЬ СТУДЕНТАМИ 7

1.1. Аналіз традиційних та цифрових підходів до реєстрації відвідування занять студентами 7

1.2. Теоретичні основи обліку відвідування та його роль у моніторингу якості освітнього процесу 10

1.3. Формування функціональних вимог до застосунку реєстрації відвідування занять студентами 19

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ 30

2.1. Архітектурна модель програмного застосунку 30

2.2. Обґрунтування вибору інструментів розробки та засобів зберігання даних 48

2.3. Алгоритм обробки та аналізу даних журналу відвідування занять 54

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ЙОГО АПРОБАЦІЯ 61

3.1. Реалізація програмного застосунку для реєстрації відвідування занять 61

3.2. Опис інтерфейсу користувача та основних сценаріїв роботи із застосунком 69

3.3. Налаштування, тестування та апробація програмного засобу 75

ЗАГАЛЬНІ ВИСНОВКИ 82

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 87

ДОДАТКИ 90

ВСТУП

Актуальність теми розробки системи реєстрації відвідування занять студентами навчальної групи зумовлена необхідністю підвищення ефективності організації освітнього процесу, оперативного отримання інформації про навчальну активність студентів та зменшення обсягу ручної роботи викладачів і кураторів. У традиційній практиці облік відвідування часто здійснюється за допомогою паперових журналів, окремих електронних таблиць або неформальних записів, що ускладнює пошук потрібної інформації, аналіз динаміки відвідування, формування звітності та виявлення студентів, які потребують додаткової уваги. В умовах цифровізації освіти такі підходи поступово втрачають ефективність, оскільки сучасний освітній процес потребує не лише фіксації факту присутності чи відсутності студента, а й можливості подальшої обробки цих даних, їх узагальнення, фільтрації, візуалізації та використання для прийняття педагогічних і управлінських рішень.

Питання автоматизації освітнього процесу, використання електронних журналів, цифрових систем управління навчанням та інструментів моніторингу навчальної діяльності розглядалися у працях дослідників, які вивчали проблеми інформатизації освіти, цифрової трансформації закладів освіти, педагогічного моніторингу та освітньої аналітики. Зокрема, значна увага приділялася використанню LMS Moodle, електронних журналів, систем управління навчальними даними, засобів збору статистики та аналітичних панелей для контролю активності здобувачів освіти. У цих дослідженнях підкреслюється, що цифрові дані про участь студентів у навчальному процесі можуть бути використані не лише для

адміністративного контролю, а й для раннього виявлення проблем із навчальною дисципліною, прогнозування ризиків академічної неуспішності та підвищення якості освітнього процесу.

Разом із тим у багатьох випадках заклади освіти або окремі викладачі потребують не складної корпоративної інформаційної системи, а простого, зрозумілого й доступного програмного засобу, орієнтованого на потреби конкретної навчальної групи. Такий застосунок має забезпечувати швидке внесення даних про студентів, створення занять, реєстрацію відвідування, перегляд журналу та формування елементарної статистики. Саме тому вибір теми є обґрунтованим, оскільки вона поєднує актуальну освітню проблему з практичним завданням програмної реалізації системи, яка може бути використана в реальному навчальному процесі для підвищення зручності, точності та оперативності обліку відвідування занять студентами.

Метою роботи є розробка програмної системи для реєстрації, збереження, перегляду та аналізу відвідування занять студентами навчальної групи.

Для досягнення поставленої мети необхідно:

- проаналізувати традиційні та цифрові підходи до обліку відвідування занять;
- визначити роль даних про відвідування у моніторингу якості освітнього процесу;
- сформулювати функціональні та нефункціональні вимоги до майбутнього застосунку;
- спроектувати архітектуру системи та модель даних;
- обґрунтувати вибір інструментів розробки і засобів зберігання інформації;
- розробити алгоритм обробки даних журналу відвідування; реалізувати програмний застосунок;
- описати його інтерфейс і основні сценарії використання;
- провести налагодження, тестування та апробацію створеного програмного засобу.

Об'єктом дослідження є процес реєстрації та аналізу відвідування занять студентами навчальної групи.

Предметом дослідження є програмні засоби, моделі даних, алгоритми та інтерфейсні рішення для автоматизації реєстрації відвідування занять студентами.

У роботі використовуються методи аналізу наукових і практичних джерел з питань цифровізації освіти та освітнього моніторингу, метод порівняння традиційних і цифрових підходів до ведення журналу відвідування, метод моделювання для побудови структури даних і архітектури застосунку, метод алгоритмізації для опису процесу обробки журналу відвідування, а також методи програмної реалізації, тестування й апробації створеної системи.

Як інструменти дослідження та розробки використані мова програмування C#, платформа .NET, середовище Visual Studio, технологія Windows Forms, система керування базами даних SQLite, а також засоби роботи з даними, зокрема Entity Framework Core.

У першому розділі роботи розглянуто теоретичні та практичні засади реєстрації відвідування занять студентами. Основну увагу приділено аналізу традиційних і цифрових підходів до ведення журналу відвідування, визначенню їхніх переваг і недоліків, а також обґрунтуванню необхідності автоматизації цього процесу. Окремо висвітлено роль даних про відвідування у моніторингу якості освітнього процесу, контролі навчальної дисципліни, виявленні студентів групи ризику та формуванні звітності. Завершенням розділу стане формулювання функціональних і нефункціональних вимог до програмного застосунку, який має забезпечити зручну реєстрацію, збереження та подальший аналіз інформації про відвідування занять.

У другому розділі розглянуто питання проєктування архітектури програмної системи та вибору засобів її реалізації. У цьому розділі описано загальну структуру застосунку, його основні модулі, логіку взаємодії користувача із системою та модель зберігання даних. Також обґрунтовано вибір мови програмування, платформи розробки, середовища програмування, технології побудови інтерфейсу та засобів роботи з базою даних. Окрему увагу приділено алгоритму обробки журналу відвідування, який передбачатиме підрахунок кількості занять, визначення кількості присутностей і пропусків, обчислення відсотка відвідування та формування підсумкових показників для окремих студентів і навчальної групи загалом.

У третьому розділі описано процес практичної реалізації програмного застосунку для реєстрації відвідування занять студентами. У ньому подано структуру створеного проєкту, охарактеризовано основні програмні компоненти, моделі даних, форми або сторінки інтерфейсу, а також логіку виконання основних операцій: додавання студентів, створення занять, фіксації статусу присутності, перегляду журналу та формування статистики. Крім того, у розділі розглянуто інтерфейс користувача, основні сценарії роботи із застосунком, результати налагодження й тестування, а також апробацію програмного засобу на прикладі умовної або реальної навчальної групи. Це дозволило оцінити працездатність системи, її відповідність сформульованим вимогам і практичну доцільність використання в освітньому процесі.

РОЗДІЛ 1. АНАЛІЗ ПІДХОДІВ ТА ІНСТРУМЕНТІВ ДЛЯ РЕЄСТРАЦІЇ ВІДВІДУВАННЯ ЗАНЯТЬ СТУДЕНТАМИ

1.1. Аналіз традиційних та цифрових підходів до реєстрації відвідування занять студентами

У традиційній освітній практиці реєстрація відвідування занять найчастіше здійснювалася за допомогою паперових журналів, відомостей, окремих списків групи або усного контролю присутності на початку заняття. Викладач або староста навчальної групи відмічав студентів, які були присутні чи відсутні на занятті, після чого ці дані могли використовуватися для підготовки звітів, інформування куратора, адміністрації або самих студентів. Такий підхід є простим і зрозумілим, не потребує складної технічної інфраструктури та може застосовуватися навіть за відсутності комп'ютерної техніки чи доступу до мережі Інтернет. Саме тому паперовий журнал тривалий час залишався основним інструментом фіксації відвідування в багатьох закладах освіти.

Разом із тим традиційний спосіб обліку має низку суттєвих недоліків. По-перше, паперові журнали та окремі відомості є незручними для швидкого пошуку інформації за певний період, дисципліною або конкретним студентом. По-друге, ручне внесення даних підвищує ризик помилок, неточностей, пропусків записів або дублювання інформації. По-третє, формування підсумкової статистики потребує додаткового часу, оскільки викладачеві або куратору необхідно вручну підраховувати кількість пропущених занять, визначати відсоток відвідування, порівнювати показники між студентами чи навчальними групами. Крім того, паперові документи можуть бути пошкоджені або втрачені, а доступ до них обмежений фізичним місцем зберігання. У разі дистанційного або змішаного навчання такі способи стають ще менш ефективними, оскільки не забезпечують оперативного збору та обробки інформації.

Проміжним варіантом між традиційним і повністю автоматизованим підходом є використання електронних таблиць, наприклад Microsoft Excel або Google Sheets. У такому випадку список студентів, дати занять, дисципліни та позначки про присутність зберігаються у табличному вигляді. Перевагою цього підходу є доступність інструментів, можливість швидкого редагування даних, застосування формул для підрахунку пропусків,

фільтрація записів і побудова простих діаграм. Google Sheets також дає змогу організувати спільний доступ до журналу для кількох користувачів. Проте електронні таблиці не завжди забезпечують достатній рівень структурованості даних, контролю введення інформації та захисту від випадкових змін. У складніших випадках таблиця може стати перевантаженою, незручною для користувача та складною для супроводу. Окремим напрямом цифрової реєстрації відвідування є використання онлайн-форм, зокрема Google Forms або Microsoft Forms. За допомогою таких інструментів можна створити форму, у якій студент або викладач вносить інформацію про присутність на занятті. Дані автоматично зберігаються у таблиці, що спрощує їх подальшу обробку. Такий підхід є зручним для швидкого збору інформації, особливо під час дистанційного навчання. Наприклад, викладач може надати студентам посилання на форму або QR-код, після чого відповіді будуть автоматично зібрані в електронному вигляді. Проте цей спосіб має й обмеження: складно гарантувати достовірність самостійної реєстрації студентів, можливе повторне заповнення форми, передача посилання іншим особам або некоректне введення персональних даних. Тому онлайн-форми є корисним допоміжним інструментом, але не завжди можуть замінити повноцінну систему обліку.

У сучасних закладах освіти для контролю навчальної активності студентів також використовуються системи управління навчанням, зокрема Moodle, Google Classroom, Microsoft Teams та інші освітні платформи. Такі системи можуть фіксувати участь студентів у навчальному курсі, активність у середовищі, виконання завдань, проходження тестів, перегляд матеріалів, участь у форумах або онлайн-заняттях. У деяких випадках платформи мають спеціальні модулі або журнали для обліку відвідування. Перевагою таких систем є інтеграція даних про відвідування з іншими навчальними показниками: успішністю, виконанням завдань, результатами тестування та активністю в електронному курсі. Однак використання таких платформ потребує налаштування, адміністрування, наявності облікових записів користувачів і певного рівня цифрової підготовки викладачів та студентів.

Ще одним цифровим підходом є використання QR-кодів, RFID-карток, мобільних застосунків або біометричних технологій. Наприклад, студент може сканувати QR-код під час заняття, прикладати студентський квиток до зчитувача або підтверджувати присутність через мобільний застосунок. Такі способи дають змогу частково автоматизувати процес реєстрації та зменшити час, який викладач витрачає на перевірку присутніх. Водночас ці підходи потребують додаткових технічних засобів, стабільного доступу до мережі, захисту персональних даних і вирішення питань достовірності ідентифікації студента.

1.2. Теоретичні основи обліку відвідування та його роль у моніторингу якості освітнього процесу

Облік відвідування занять студентами є важливим елементом організації освітнього процесу, оскільки дає змогу фіксувати участь здобувачів освіти в навчальних заняттях, аналізувати їхню навчальну активність та своєчасно виявляти проблеми, пов'язані з пропусками занять. У найпростішому розумінні облік відвідування передбачає реєстрацію факту присутності або відсутності студента на конкретному занятті. Проте в сучасних умовах таке трактування є дещо обмеженим, оскільки дані про відвідування можуть використовуватися не лише для адміністративного контролю, а й для педагогічного аналізу, моніторингу якості освітнього процесу та підтримки прийняття управлінських рішень.

Відвідування занять є одним із показників залученості студента до освітнього процесу. Регулярна присутність на заняттях забезпечує можливість систематичного засвоєння навчального матеріалу, участі в обговореннях, виконання практичних завдань, отримання консультацій від викладача та взаємодії з іншими студентами. Особливо важливим цей показник є для дисциплін, у яких значну роль відіграють лабораторні, практичні або семінарські заняття, оскільки саме під час них формуються прикладні вміння та професійні навички. Пропуски занять можуть призводити до фрагментарного засвоєння матеріалу, зниження поточної успішності, ускладнення виконання індивідуальних і командних завдань, а також до зростання ризику академічної неуспішності.

Розвиток науково-методичних підходів до спостереження за навчальними заняттями як інструменту моніторингу якості освіти демонструє тривалу еволюцію - від класичних моделей контролю до сучасних цифрових та AI-орієнтованих систем. Генезис цього напрямку у 1960-1990-х роках пов'язаний із діяльністю школи «ефективного викладання», зокрема працями Дж. Брофі та Т. Гуда, які заклали фундамент систематичного аналізу уроку [5; 6]. Їхній підхід дозволив емпірично пов'язати поведінку викладача з результатами навчання, акцентуючи увагу на часовій структурі заняття та специфіці зворотного зв'язку. Паралельно з цим Р[Медлі [23-25] впровадив концепцію систематичного спостереження (systematic classroom observation) як кількісного методу, що мав на меті замінити суб'єктивні описові оцінки об'єктивною фіксацією подій. Подальший розвиток методології у роботах Г[Тарпа, Р[Ваксмана та Г[Волберга сприяв розширенню об'єкта моніторингу, додавши до нього параметри навчального середовища, культурної чутливості та міжособистісної взаємодії [15]. Попри свою інноваційність, класичний етап характеризувався використанням ручних протоколів, вираженою спрямованістю на дії викладача та низькою валідністю одиничних відвідувань для формування узагальнених висновків про якість викладання.

Трансформація підходів у період 1990-2010-х років позначилася інституціоналізацією моделей моніторингу, серед яких провідне місце посіла рамка професійного викладання Ш. Даніелсон (Danielson Framework for Teaching) [9]. Ця модель структурувала дані за чотирма ключовими доменами - від планування до професійної етики, що забезпечило комплексний аналіз педагогічної діяльності. Водночас система оцінювання взаємодії в класі CLASS (Classroom Assessment Scoring System) Р. Піанти змістила фокус на якість комунікації між викладачем і студентом як визначальний фактор результативності навчання [18]. Сучасний етап розвитку характеризується переглядом ролі адміністративного контролю; прикладом цього є досвід британської інспекції Ofsted [14], яка після 2015 року відмовилася від виставлення оцінок за окремі уроки на користь системного аналізу освітнього процесу. Новітні тенденції моніторингу інтегрують цифрові інструменти та алгоритми штучного інтелекту, що дозволяє автоматизувати аналіз даних, мінімізувати суб'єктивізм та трансформувати відвідування занять із засобу зовнішнього нагляду на механізм безперервного професійного розвитку педагогів.

Сучасна трансформація моніторингу якості освіти характеризується послідовним переходом від емпіричних методів до цифрових систем структурованого спостереження (e-Observation). Впровадження спеціалізованого програмного забезпечення, як-от eWalk, iObservation або модулів у межах університетських систем управління якістю (QMS), дозволило оцифрувати класичні протоколи та стандартизувати індикатори педагогічної ефективності. Використання мобільних пристроїв для фіксації структури заняття, типів активностей та рівня залученості студентів забезпечує накопичення масивів порівнюваних даних, хоча й не нівелює повністю суб'єктивізм експерта-спостерігача.

Паралельно з цифровізацією протоколів відбувається розвиток відеоаналізу занять, що докорінно змінює філософію відвідування. Традиційні «живі» візити дедалі частіше замінюються практиками Lesson Study та відеоцентрованою рефлексією, де запис заняття стає об'єктом колегіального обговорення або індивідуального професійного аналізу. У цьому контексті відеофіксація використовується не як інструмент адміністративного контролю, а як джерело доказової аналітики для безперервного вдосконалення викладацької майстерності.

Важливим етапом еволюції стало виникнення навчальної аналітики (Learning Analytics) як форми «непрямого спостереження». Завдяки інтеграції з LMS-платформами (Moodle, Canvas) та сервісами відеозв'язку (MS Teams, Zoom), поведінка та активність студентів фіксуються автоматично. Це дозволяє оцінювати якість освітнього процесу опосередковано, проте на великих вибірках і з мінімальним рівнем

суб'єктивності, виявляючи кореляції між педагогічними стратегіями та реальними навчальними результатами.

Найбільш прогресивним напрямом є впровадження інструментів штучного інтелекту, здатних до автоматизованого аналізу мовлення та взаємодії. AI-алгоритми дозволяють диференціювати частку мовлення викладача й студентів, класифікувати типи запитань та аналізувати динаміку дискусій. Попри значний потенціал комп'ютерного зору для аналізу уваги аудиторії та жестів, масове впровадження таких рішень у вищій школі наразі стримується етичними обмеженнями, регламентами GDPR та необхідністю дотримання принципів академічної свободи. У сучасних інтегрованих системах забезпечення якості (QA platforms) спостереження за заняттям остаточно втрачає статус ізольованого інструменту [1]. Воно стає елементом комплексної екосистеми, що базується на триангуляції даних: поєднанні результатів відвідувань, студентських опитувань, аналітики LMS та академічної успішності. Таким чином, роль відвідування занять трансформується: воно більше не є достатньою підставою для остаточного оцінювання якості освіти, проте залишається критично важливим якісним інструментом для глибокого аналізу педагогічних практик та професійного розвитку викладацького складу.

Реєстрація відвідування занять студентами є одним із важливих елементів організації освітнього процесу, оскільки дає змогу фіксувати участь здобувачів освіти в навчальних заняттях, контролювати навчальну дисципліну, аналізувати активність студентів і своєчасно виявляти проблеми, пов'язані з пропусками занять. Відвідування не можна розглядати як єдиний показник якості навчання або успішності студента, однак воно є важливим індикатором залученості до освітнього процесу. Регулярна присутність на заняттях створює умови для систематичного засвоєння навчального матеріалу, участі в обговореннях, виконання практичних завдань, лабораторних робіт, консультацій із викладачем та взаємодії з іншими студентами навчальної групи. Тому облік відвідування має не лише адміністративне, а й педагогічне значення.

З педагогічної точки зору облік відвідування виконує не лише контрольну, а й діагностичну функцію. Дані про пропуски можуть свідчити про зниження мотивації студента, проблеми з організацією навчального часу, труднощі в опануванні дисципліни або інші обставини, що впливають на його участь у навчальному процесі. Якщо інформація про відвідування фіксується систематично, викладач або куратор може своєчасно виявити негативну тенденцію та вжити відповідних заходів: провести індивідуальну бесіду, надати консультацію, повідомити куратора групи, запропонувати додаткові матеріали або організувати підтримку студента. Таким чином, облік відвідування може розглядатися як інструмент раннього попередження можливих навчальних проблем.

У системі моніторингу якості освітнього процесу дані про відвідування відіграють роль одного з індикаторів організації навчання. Моніторинг якості освіти передбачає систематичний збір, обробку та аналіз інформації про перебіг освітнього процесу з метою оцінювання його стану, виявлення проблем і прийняття рішень щодо його вдосконалення. Відвідування занять не є єдиним і достатнім показником якості навчання, однак воно може бути важливим додатковим джерелом інформації. Наприклад, низький рівень відвідування певної дисципліни може свідчити про організаційні проблеми, незручний розклад, недостатню мотивацію студентів або потребу в перегляді методики проведення занять.

Водночас стабільне відвідування занять може бути ознакою належної організації освітнього процесу та зацікавленості студентів у навчанні. Теоретично облік відвідування можна розглядати як процес, що включає кілька взаємопов'язаних етапів: ідентифікацію студента, фіксацію факту участі у занятті, збереження запису, перевірку коректності даних, узагальнення інформації та формування висновків. На першому етапі визначається, який саме студент належить до відповідної навчальної групи. На другому етапі для кожного студента встановлюється певний статус участі у занятті, наприклад «присутній», «відсутній», «запізнівся» або «відсутній з поважної причини». На третьому етапі ці дані зберігаються у журналі відвідування. Далі вони можуть бути використані для перегляду історії занять, формування статистики, підготовки звітів і виявлення студентів із високою кількістю пропусків.

Для більш повного аналізу доцільно використовувати не лише просту фіксацію присутності або відсутності, а й диференційовану систему статусів. Наприклад, студент може бути присутнім на занятті, бути відсутнім без поважної причини, запізнитися або мати підтверджену поважну причину відсутності. Такий підхід дозволяє зробити облік більш гнучким і справедливим, оскільки різні ситуації не повинні оцінюватися однаково.

У програмній системі це може бути реалізовано через перелік стандартних статусів, які обирає користувач під час реєстрації відвідування. Одним із базових показників аналізу є відсоток відвідування, який може розраховуватися як співвідношення кількості відвіданих занять до загальної кількості проведених занять. У загальному вигляді цей показник можна подати так:

$$P = \frac{P}{N} \times 100\%, \quad (1.1)$$

де (P) - відсоток відвідування, (N) - кількість занять, на яких студент був присутній, (N) - загальна кількість занять за певний період. Такий показник дає змогу швидко оцінити рівень участі студента в освітньому процесі. Додатково може розраховуватися кількість пропусків:

$$P_{\text{пропусків}} = N - P, \quad (1.2)$$

де (P) - кількість пропущених занять. Якщо система враховує запізнєння або поважні причини, формула може бути розширена, наприклад шляхом введення коефіцієнтів для різних статусів відвідування.

Для потреб моніторингу може бути доцільним визначення критичного рівня відвідування. Наприклад, якщо відсоток відвідування студента є нижчим за встановлений поріг, система може позначати такого студента як такого, що потребує уваги викладача або куратора. Це можна описати умовою:

$$P < P_{\text{крит}}, \quad (1.3)$$

де $P_{\text{крит}}$ - мінімально допустимий рівень відвідування. Значення такого порогу може встановлюватися відповідно до внутрішніх правил закладу освіти або особливостей конкретної дисципліни. У межах програмного застосування така функція може бути реалізована через автоматичне формування списку студентів із низьким рівнем відвідування.

З організаційної точки зору облік відвідування може використовуватися різними учасниками освітнього процесу. Викладач використовує ці дані для контролю присутності на заняттях і аналізу активності студентів з конкретної дисципліни. Куратор академічної групи може використовувати інформацію про пропуски для загального супроводу студентів, проведення профілактичної роботи та взаємодії з адміністрацією. Адміністрація закладу освіти може використовувати узагальнені дані для аналізу дисципліни в групах, підготовки звітності та прийняття управлінських рішень. Самі студенти також можуть бути зацікавлені в доступі до власної статистики, оскільки це дозволяє їм контролювати свою навчальну активність. Особливого значення облік відвідування набуває в умовах змішаного та дистанційного навчання. У таких форматах факт участі студента у занятті може фіксуватися не лише фізичною присутністю в аудиторії, а й підключенням до онлайн-заняття, виконанням завдань у системі дистанційного навчання, активністю в електронному курсі або участю в обговореннях. Це розширює зміст поняття «відвідування» і потребує більш гнучких підходів до його реєстрації. Проте для навчальної групи, що працює у звичайному або змішаному форматі, базова система реєстрації відвідування все одно залишається важливою, оскільки забезпечує впорядковане збереження інформації та можливість її подальшого аналізу.

Автоматизація обліку відвідування має низку переваг порівняно з ручним веденням журналу. Вона зменшує ймовірність помилок під час підрахунків, прискорює пошук потрібної інформації, забезпечує швидке формування звітів і дозволяє накопичувати дані за тривалий період. Крім того, програмна система може автоматично виконувати розрахунки, які в ручному режимі потребують значного часу: визначати кількість

пропусків, відсоток відвідування, середні показники по групі, рейтинг студентів за рівнем присутності, а також динаміку відвідування за певний період. Завдяки цьому облік відвідування перетворюється з формальної процедури на інструмент аналітичної підтримки освітнього процесу. Водночас слід зазначити, що дані про відвідування не можуть бути єдиною підставою для оцінювання якості навчання або рівня підготовки студента. Вони мають розглядатися у взаємозв'язку з іншими показниками: результатами поточного та підсумкового контролю, виконанням практичних робіт, активністю в електронному курсі, участю в проектній діяльності та рівнем сформованості компетентностей. Тому система реєстрації відвідування повинна розглядатися як складова ширшої системи моніторингу освітнього процесу, а не як самодостатній інструмент оцінювання.

Отже, теоретичні основи обліку відвідування ґрунтуються на систематичному зборі, збереженні та аналізі даних про участь студентів у навчальних заняттях. Такі дані мають значення для викладачів, кураторів, адміністрації та самих студентів, оскільки дозволяють своєчасно виявляти проблеми, формувати звітність і приймати обґрунтовані рішення щодо організації освітнього процесу. У контексті цієї роботи облік відвідування розглядається як важливий елемент моніторингу якості освіти, а розробка програмної системи для його автоматизації є практичним способом підвищення точності, зручності та аналітичної цінності цього процесу.

1.3. Формування функціональних вимог до застосунку реєстрації відвідування занять студентами

Формування функціональних вимог до застосунку реєстрації відвідування занять студентами є важливим етапом розробки програмної системи, оскільки саме на цьому етапі визначається, які завдання має виконувати майбутній програмний засіб, які дані він повинен обробляти, які користувачі будуть з ним працювати та які результати мають бути отримані після його використання. На відміну від загального опису ідеї програмного продукту, функціональні вимоги конкретизують очікувану поведінку системи, її основні модулі, сценарії роботи та правила взаємодії користувача із застосунком. Для системи реєстрації відвідування занять такі вимоги мають бути безпосередньо пов'язані з організацією навчального процесу, веденням журналу відвідування, збереженням інформації про студентів і формуванням аналітичних показників.

Розроблюваний застосунок має бути орієнтований на автоматизацію процесу фіксації присутності або відсутності студентів на заняттях навчальної групи. Основними користувачами такої системи можуть бути викладач, куратор академічної групи, староста або інша відповідальна особа, яка здійснює облік відвідування. У разі подальшого розширення системи можливе введення різних ролей користувачів, наприклад адміністратора, викладача та користувача з правами перегляду. Проте в межах базової версії застосунку доцільно зосередитися на функціях, необхідних для практичного використання в межах однієї або кількох навчальних груп: створення списку студентів, створення занять, реєстрація відвідування, перегляд журналу та формування підсумкової статистики.

Першою важливою вимогою є можливість ведення довідника студентів. Система повинна надавати користувачеві змогу додавати нових студентів, редагувати їхні дані, видаляти помилково створені записи та переглядати список студентів певної навчальної групи. Для кожного студента доцільно зберігати такі основні відомості: прізвище, ім'я, по батькові, навчальну групу, за потреби - номер студентського квитка або інший ідентифікатор. Наявність структурованого списку студентів дозволяє уникнути повторного введення даних під час кожної реєстрації відвідування та забезпечує зручність подальшого аналізу.

Другою функціональною вимогою є можливість роботи з навчальними групами та дисциплінами. Застосунок має дозволяти створювати записи про навчальні групи, до яких належать студенти, а також про дисципліни, за якими проводяться заняття. Це важливо тому, що облік відвідування зазвичай здійснюється не абстрактно, а в межах конкретної дисципліни, конкретної дати та конкретної академічної групи. Наприклад, один і той самий студент може відвідувати різні заняття з різних навчальних компонентів, тому система повинна забезпечувати можливість фільтрації та аналізу даних за дисципліною, групою та періодом.

Третьою ключовою вимогою є створення запису про заняття. Перед реєстрацією відвідування користувач має вказати дату заняття, навчальну групу, дисципліну та, за потреби, тип заняття: лекція, практичне, лабораторне або семінарське заняття. Також можуть фіксуватися додаткові відомості, наприклад тема заняття або примітка викладача. Створення окремого запису про заняття є необхідним для того, щоб кожен факт присутності або відсутності студента був пов'язаний із конкретною навчальною подією. Це забезпечує коректність журналу та дає змогу формувати звітність не лише за студентами, а й за датами, дисциплінами й типами занять.

Основною функцією застосунку є реєстрація відвідування студентів. Після вибору групи та заняття система повинна автоматично відображати список студентів цієї групи, а користувач має мати можливість швидко встановити для кожного студента відповідний статус. До базових статусів доцільно віднести такі: «присутній», «відсутній», «запізнився», «відсутній з поважної причини». Такий підхід є більш гнучким, ніж проста фіксація присутності або відсутності, оскільки дозволяє враховувати різні ситуації. Наприклад, запізнення може впливати на підсумковий показник відвідування частково, а відсутність з поважної причини може обліковуватися окремо від пропусків без поважної причини.

Важливою функціональною вимогою є збереження записів журналу відвідування у базі даних або іншому надійному сховищі. Кожен запис має містити інформацію про студента, заняття, дату, дисципліну, статус відвідування та, за потреби, коментар. Збереження даних у структурованому вигляді дозволяє забезпечити їх подальший пошук, сортування, фільтрацію та аналіз. На відміну від паперового журналу або окремої електронної таблиці, база даних забезпечує цілісність інформації, зменшує ризик дублювання записів і створює передумови для розширення функціональності системи.

Застосунок повинен підтримувати перегляд журналу відвідування за різними параметрами. Користувач має мати можливість переглядати записи за датою, студентом, навчальною групою, дисципліною або певним періодом. Наприклад, викладач може переглянути відвідування конкретного заняття, куратор - статистику за всією групою, а відповідальна особа - інформацію щодо окремого студента. Для зручності роботи доцільно передбачити фільтри, пошук і сортування записів. Такий функціонал дозволяє швидко знаходити потрібну інформацію без необхідності переглядати весь журнал вручну.

Окремою вимогою є формування статистики відвідування. Система повинна автоматично підраховувати загальну кількість занять, кількість відвіданих занять, кількість пропусків, кількість запізнень і відсоток відвідування. Такі показники можуть розраховуватися як для окремого студента, так і для всієї навчальної групи. Наприклад, застосунок може показувати список студентів із найнижчим рівнем відвідування або формувати загальну статистику групи за обраний період. Це дає змогу використовувати систему не лише як електронний журнал, а й як інструмент моніторингу навчальної активності.

Доцільною функцією є можливість формування звітів. Звіт може містити інформацію про відвідування занять за певний період, за конкретною дисципліною або за окремим студентом. Для практичного використання корисною є можливість експорту даних у поширені формати, наприклад CSV, Excel або PDF. Навіть якщо у базовій версії застосунку буде реалізовано лише експорт у CSV, це вже підвищить його практичну цінність, оскільки дозволить використовувати отримані дані в інших програмах, зокрема в електронних таблицях або аналітичних системах.

Важливо також передбачити перевірку коректності введених даних. Система не повинна дозволяти створювати заняття без зазначення дати,

групи або дисципліни, додавати студента без прізвища та імені, зберігати порожні записи відвідування або дублювати одну й ту саму реєстрацію для одного заняття без потреби. Перевірка даних має здійснюватися як на рівні інтерфейсу користувача, так і на рівні логіки застосунку. Це дозволить уникнути помилок, які можуть вплинути на правильність журналу та статистичних показників.

Крім функціональних вимог, важливо визначити й нефункціональні вимоги, які характеризують якість роботи системи. Застосунок має бути простим у використанні, мати зрозумілий інтерфейс і не вимагати від користувача спеціальних технічних знань. Основні дії повинні виконуватися швидко й логічно: користувач має легко перейти від вибору групи до позначення присутності та збереження результатів. Система повинна забезпечувати надійне збереження даних, стабільну роботу, захист від випадкових помилок користувача та можливість подальшого розширення. Наприклад, у майбутньому до застосунку можна додати авторизацію користувачів, інтеграцію з LMS Moodle, імпорт списків студентів або побудову графіків відвідування.

Узагальнено основні функціональні вимоги до застосунку можна подати у вигляді таблиці 1.1.

Таблиця 1.1

Функціональні вимоги до застосунку

№ Функціональна вимога Зміст вимоги

- | | | |
|----|------------------------------|--|
| 1 | Ведення списку студентів | Додавання, редагування, видалення та перегляд даних студентів |
| 2 | Робота з навчальними групами | Створення та вибір груп для подальшої реєстрації відвідування |
| 3 | Робота з дисциплінами | Додавання дисциплін і прив'язка занять до конкретної дисципліни |
| 4 | Створення заняття | Визначення дати, групи, дисципліни та типу заняття |
| 5 | Реєстрація відвідування | Встановлення статусу присутності для кожного студента |
| 6 | Збереження журналу | Запис даних про відвідування у базу даних |
| 7 | Перегляд журналу | Відображення записів за датою, групою, студентом або дисципліною |
| 8 | Фільтрація та пошук | Пошук потрібної інформації за заданими параметрами |
| 9 | Формування статистики | Розрахунок кількості пропусків, присутностей і відсотка відвідування |
| 10 | Формування звітів | Підготовка підсумкової інформації для перегляду або експорту |

Для більш точного опису роботи системи доцільно визначити основні сценарії використання. Перший сценарій пов'язаний із підготовкою системи до роботи: користувач створює навчальну групу, додає студентів і формує перелік дисциплін. Другий сценарій передбачає створення заняття: користувач обирає дату, групу, дисципліну та тип заняття. Третій сценарій є основним і полягає в реєстрації відвідування: система відображає список студентів групи, користувач позначає статус кожного студента та зберігає результат. Четвертий сценарій передбачає перегляд журналу й аналіз статистики за обраними параметрами. П'ятий сценарій пов'язаний із формуванням звіту або експортом даних для подальшого використання.

З погляду структури даних майбутній застосунок має працювати з кількома основними сутностями. До них належать студент, навчальна група, дисципліна, заняття та запис відвідування. Сутність «Студент» містить персональні дані та належність до групи. Сутність «Група» об'єднує студентів за навчальною ознакою. Сутність «Дисципліна» визначає навчальний предмет, у межах якого проводиться заняття. Сутність «Заняття» містить дату, тип заняття, групу та дисципліну. Сутність «Запис відвідування» поєднує конкретного студента з конкретним заняттям і містить статус його участі. Така структура дозволяє уникнути дублювання інформації та забезпечує логічну організацію бази даних.

Формування вимог до застосунку також передбачає визначення меж розробки. У межах цієї роботи система може бути реалізована як локальний настільний застосунок або як вебзастосунок, однак її основна мета залишається однаковою - забезпечити зручний облік відвідування студентів навчальної групи. Базова версія системи не обов'язково повинна містити складну систему авторизації, інтеграцію з університетськими інформаційними ресурсами або автоматичну ідентифікацію студентів за QR-кодами. Такі можливості можуть бути розглянуті як напрями подальшого розвитку. Основну увагу в роботі доцільно зосередити на коректній реалізації базових функцій, зручності інтерфейсу та правильності обробки даних.

Отже, функціональні вимоги до застосунку реєстрації відвідування занять студентами визначають основні можливості майбутньої системи та забезпечують зв'язок між теоретичним аналізом проблеми й подальшою програмною реалізацією. Розроблюваний застосунок має забезпечувати ведення списку студентів, створення занять, фіксацію статусів відвідування, збереження журналу, перегляд і фільтрацію даних, формування статистики та підготовку звітів. Визначені вимоги створюють основу для проєктування архітектури системи, вибору інструментів розробки, побудови моделі даних і реалізації програмного засобу, який може бути використаний для підвищення ефективності обліку відвідування в освітньому процесі.

Діаграма варіантів використання UML (рис. 1.1) відображає основні функціональні можливості застосунку реєстрації відвідування занять студентами та показує, які користувачі взаємодіють із системою. У межах діаграми система подана як окрема область із назвою «Застосунок реєстрації відвідування занять студентами», всередині якої розміщено основні варіанти використання. Зовні системи показано акторів, тобто користувачів або зовнішні ролі, які ініціюють певні дії в застосунку. До таких акторів належать студент, викладач, уповноважена особа, технічний адміністратор і батьки.

Рис. 1.1. Діаграма варіантів використання застосунку

Центральним актором у цій системі є викладач, оскільки саме він безпосередньо працює з журналом відвідування під час проведення занять. Викладач має можливість авторизуватися в системі, створювати запис про заняття, реєструвати відвідування студентів, переглядати журнал відвідування, аналізувати статистику та формувати звіти. Основним варіантом використання для викладача є «Зареєструвати відвідування», оскільки ця дія забезпечує фіксацію факту присутності або відсутності кожного студента на конкретному занятті. Під час виконання цього варіанта використання система передбачає встановлення одного зі статусів: «присутній», «відсутній», «запізнився» або «поважна причина». Актор «Уповноважена особа» представляє користувача, який має організаційні або адміністративні повноваження щодо контролю відвідування. Такою особою може бути куратор академічної групи, завідувач відділення, працівник деканату або інший відповідальний співробітник. Уповноважена особа може авторизуватися в системі, вести список студентів, працювати з даними навчальних груп, вести перелік дисциплін, переглядати журнал відвідування, аналізувати статистику та формувати звіти. На відміну від викладача, її роль більше пов'язана не з поточним проведенням заняття, а з організацією, контролем, узагальненням і використанням даних для прийняття управлінських або педагогічних рішень. Актор «Студент» має обмежені права доступу до системи. Він може авторизуватися та переглядати власну інформацію про відвідування. Такий варіант використання дає студентові змогу контролювати власну навчальну активність, бачити кількість відвіданих і пропущених занять, а також

своєчасно реагувати на можливі проблеми. Студент не має права змінювати журнал відвідування, редагувати дані про заняття або впливати на статистику, оскільки ці функції належать викладачу або уповноваженій особі.

Актор **«Батьки»** також має обмежений доступ до системи. У діаграмі передбачено, що батьки можуть авторизуватися, переглядати інформацію про відвідування конкретного студента та, за потреби, переглядати узагальнену статистику. Така функціональність може бути корисною у випадках, коли заклад освіти передбачає інформування батьків або законних представників про навчальну дисципліну студента. Водночас на діаграмі зазначено, що доступ батьків має бути обмежений лише інформацією щодо конкретного студента, тобто вони не повинні бачити дані інших студентів або всієї навчальної групи.

Окремо на діаграмі представлено актора **«Технічний адміністратор»**. Його функції пов'язані не з освітнім змістом роботи системи, а з її технічним супроводом. Технічний адміністратор може авторизуватися, керувати обліковими записами користувачів, виконувати резервне копіювання та відновлення даних, а також за потреби працювати з даними студентів і навчальних груп. Ця роль є важливою для забезпечення стабільної роботи застосунку, захисту даних, відновлення інформації у разі збою та підтримки користувачів.

У межах системи виділено кілька основних варіантів використання. Варіант **«Авторизуватися в системі»** є спільним для всіх акторів, оскільки перед початком роботи користувач має підтвердити свою особу та отримати доступ до функцій відповідно до своєї ролі. Варіанти **«Вести список студентів»**, **«Вести дані про навчальні групи»** та **«Вести перелік дисциплін»** належать до підготовчих і довідкових функцій системи. Вони забезпечують створення необхідної бази даних, без якої неможливо коректно створювати заняття та реєструвати відвідування.

Варіант використання **«Створити запис про заняття»** описує дію, під час якої користувач вносить інформацію про конкретне заняття: дату, навчальну групу, дисципліну та, за потреби, тип заняття. Після створення такого запису система може перейти до реєстрації відвідування.

Варіант **«Зареєструвати відвідування»** є ключовим у діаграмі, оскільки саме він реалізує основне призначення застосунку. Для кожного студента обраної групи користувач встановлює відповідний статус, після чого ці дані зберігаються в журналі.

На діаграмі також показано зв'язок `<<include>>` між варіантом **«Зареєструвати відвідування»** та варіантом **«Встановити статус»**. Це означає, що встановлення статусу є обов'язковою складовою процесу реєстрації відвідування. Іншими словами, неможливо повноцінно зареєструвати відвідування без визначення статусу кожного студента. До таких статусів належать присутність, відсутність, запізнення або відсутність із поважної причини.

Варіанти **«Переглянути журнал відвідування»**, **«Переглянути статистику відвідування»** та **«Сформувати звіт про відвідування»** пов'язані з аналізом і використанням збережених даних. Журнал дозволяє переглядати первинні записи про відвідування, статистика - отримувати узагальнені показники, а звіт - оформлювати результати у зручному для подальшого використання вигляді. Для цих варіантів використання передбачено зв'язок `<<include>>` із варіантом **«Фільтрувати дані за датою, групою, дисципліною, студентом»**. Це означає, що перегляд журналу, статистики або звіту зазвичай потребує вибору певних параметрів: конкретного студента, групи, дисципліни або періоду.

Варіант використання **«Експортувати дані у CSV / Excel / PDF»** пов'язаний зі звітністю та журналом через зв'язок `<<extend>>`. Це означає, що експорт не є обов'язковою дією під час перегляду журналу або формування звіту, але може виконуватися додатково, якщо користувачеві потрібно зберегти або передати дані в окремому файлі. Наприклад, викладач або уповноважена особа може переглянути журнал у системі, а за потреби експортувати його у формат Excel або PDF для подальшого друку, архівування чи передачі адміністрації.

Таким чином, діаграма варіантів використання на рис. 1.1 демонструє розподіл функцій між різними категоріями користувачів системи. Викладач відповідає переважно за створення занять і фіксацію відвідування, уповноважена особа - за організаційний контроль і формування звітності, студент - за перегляд власних даних, батьки - за обмежений перегляд інформації про конкретного студента, а технічний адміністратор - за підтримку працездатності системи та управління обліковими записами. Така структура дозволяє забезпечити логічний розподіл прав доступу, підвищити зручність роботи із застосунком і створити основу для подальшого проектування архітектури програмної системи.

РОЗДІЛ 2. ПРОЄКТУВАННЯ АРХІТЕКТУРИ ТА ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1. Архітектурна модель програмного застосунку

Архітектурна модель програмного застосунку для реєстрації відвідування занять студентами визначає загальну структуру системи, її основні компоненти, логіку взаємодії між ними та принципи організації програмного коду. На етапі проєктування архітектури важливо не лише визначити, які функції має виконувати система, а й встановити, яким чином ці функції будуть розподілені між окремими модулями застосунку. Такий підхід дозволяє зробити програмний засіб більш зрозумілим, зручним для розробки, тестування, супроводу та подальшого розширення.

Розроблюваний застосунок призначений для автоматизації процесу обліку відвідування занять студентами навчальної групи. Його основними функціями є ведення списку студентів, робота з навчальними групами та дисциплінами, створення записів про заняття, фіксація статусів відвідування, перегляд журналу, фільтрація даних, формування статистики та підготовка звітів. З огляду на це система має бути побудована як багатокomпонентний програмний засіб, у якому кожен модуль відповідає за окрему частину функціональності. Це дає можливість уникнути надмірного ускладнення коду, зменшити дублювання логіки та забезпечити логічну послідовність роботи користувача із застосунком.

Загальну структуру застосунку доцільно побудувати за принципами архітектурного підходу MVC, тобто Model-View-Controller, з додаванням сервісного рівня. Цей підхід передбачає поділ системи на чотири основні частини: модель, представлення, сервіси та контролер. Модель відповідає за дані та правила їх обробки, представлення забезпечує взаємодію користувача із системою, а контролер приймає запити від користувача, обробляє їх і визначає, які дані потрібно передати для відображення, сервіси реалізують зв'язок між контролером та моделлю. Рівень моделі є основою застосунку, оскільки саме він описує основні сутності предметної області. До таких сутностей належать студент, навчальна група, дисципліна, заняття, запис відвідування та користувач системи. Кожна сутність має власний набір властивостей і зв'язків з іншими сутностями. Наприклад, студент належить до певної навчальної групи, заняття пов'язане з конкретною дисципліною та групою, а запис відвідування поєднує студента із заняттям і містить статус його присутності. Завдяки такій структурі система може зберігати дані не у вигляді розрізнених записів, а як логічно пов'язану інформаційну модель.

Модель **«Студент»** містить дані, необхідні для ідентифікації студента в системі: прізвище, ім'я, по батькові, навчальну групу, а за потреби - номер студентського квитка або інший унікальний ідентифікатор. Модель **«Навчальна група»** використовується для об'єднання студентів за академічною ознакою. Модель **«Дисципліна»** містить інформацію про навчальний предмет, у межах якого проводяться заняття. Модель **«Заняття»** фіксує дату, тип заняття, групу та дисципліну. Модель **«Запис відвідування»** зберігає інформацію про те, який студент був присутній або відсутній на конкретному занятті. Саме ця модель є центральною для роботи застосунку, оскільки вона забезпечує накопичення даних, необхідних для формування журналу та статистики.

Рівень представлення відповідає за зовнішню взаємодію користувача із застосунком. У будь-якому разі цей рівень не повинен містити складної

логіки обробки даних. Його головне завдання - надати користувачеві зручний інтерфейс для введення, перегляду та редагування інформації. До основних елементів рівня представлення можна віднести головну сторінку або головну форму застосунку, форму роботи зі студентами, форму роботи з навчальними групами, форму роботи з дисциплінами, форму створення заняття, форму реєстрації відвідування, сторінку або форму перегляду журналу, а також форму статистики та звітності. Головна форма має забезпечувати перехід до основних модулів системи. Форма студентів дозволяє додавати, редагувати та видаляти записи про студентів. Форма реєстрації відвідування є однією з головних, оскільки саме через неї користувач обирає заняття та встановлює статуси для студентів. Форма статистики забезпечує перегляд узагальнених результатів за певний період, групу або дисципліну.

Рівень контролера або керування логікою відповідає за обробку дій користувача. Саме цей рівень приймає інформацію, введену через інтерфейс, перевіряє її коректність, звертається до моделей або сервісів, зберігає дані та повертає результат користувачеві. Наприклад, коли викладач створює нове заняття, контролер перевіряє, чи вказано дату, групу та дисципліну, після чого передає дані до модуля збереження. Коли користувач реєструє відвідування, контролер отримує список студентів і встановлені статуси, перевіряє, чи не було вже створено записів для цього заняття, а потім зберігає інформацію в журналі. Коли користувач переглядає статистику, контролер отримує параметри фільтрації та запускає відповідний алгоритм обробки даних.

Для підвищення зрозумілості та зручності супроводу застосунку доцільно виділити окремий сервісний рівень. Він може містити класи, які реалізують основну бізнес-логіку системи: роботу зі студентами, заняттями, записами відвідування, статистикою та звітами. Наприклад, сервіс студентів може відповідати за додавання, редагування й видалення студентів; сервіс занять - за створення занять і перевірку їх унікальності; сервіс відвідування - за збереження статусів присутності; сервіс статистики - за обчислення кількості пропусків, відсотка відвідування й визначення студентів із низьким рівнем присутності; сервіс звітності - за підготовку даних до експорту. Такий поділ дозволяє винести основну логіку з інтерфейсу й зробити систему більш структурованою.

Окремим важливим компонентом архітектури є рівень доступу до даних. Він забезпечує взаємодію застосунку з **базою даних або іншим сховищем інформації**. Якщо для реалізації використовується Entity Framework Core, цей рівень може бути представлений контекстом бази даних і наборами сутностей, які відповідають таблицям. Головне завдання цього рівня - ізолювати логіку збереження та отримання даних від інших частин системи. Завдяки цьому інтерфейс і бізнес-логіка не повинні напряму працювати з таблицями бази даних.

У загальному вигляді архітектуру застосунку можна подати як послідовність взаємодії кількох рівнів на рис. 2.1: користувач взаємодіє з інтерфейсом, інтерфейс передає запит до контролера або сервісу, сервіс обробляє дані та звертається до рівня доступу до даних, після чого результат повертається користувачеві у вигляді повідомлення, таблиці, журналу або статистичного звіту. Наприклад, під час реєстрації відвідування викладач обирає групу, дисципліну та дату заняття, система показує список студентів, викладач встановлює статуси, а після натискання кнопки збереження дані передаються до сервісу відвідування. Далі сервіс перевіряє коректність записів і передає їх до бази даних. Після успішного збереження користувач отримує повідомлення про виконання операції.

Рис. 2.1 Архітектура системи обліку відвідування студентів

Логічно застосунок можна поділити на кілька основних частин. Першою є модуль автентифікації та керування доступом, який відповідає за вхід користувачів до системи та розмежування прав. У базовій версії він може бути спрощеним або реалізованим лише частково, однак у перспективі саме цей модуль дозволить відокремити права викладача, студента, уповноваженої особи, батьків і технічного адміністратора. Викладач повинен мати доступ до створення занять і реєстрації відвідування, студент - лише до перегляду власних даних, батьки - до перегляду інформації про конкретного студента, уповноважена особа - до звітів і статистики, а технічний адміністратор - до керування обліковими записами та технічного обслуговування системи.

Другою логічною частиною є модуль довідкових даних. Він забезпечує роботу зі студентами, навчальними групами та дисциплінами. Без цього модуля неможливо коректно створювати заняття та формувати журнал відвідування, оскільки система повинна мати попередньо створені списки студентів, груп і навчальних предметів. Цей модуль також забезпечує цілісність даних, адже кожен студент має бути пов'язаний із певною групою, а кожне заняття - з конкретною дисципліною.

Третьою логічною частиною є модуль занять. Він відповідає за створення, редагування та перегляд записів про заняття. У цьому модулі зберігаються дата проведення, дисципліна, навчальна група, тип заняття та додаткові примітки. Створення заняття є проміжним етапом між довідковими даними та реєстрацією відвідування. Без створення заняття неможливо коректно зафіксувати присутність студентів, оскільки кожен запис журналу має бути прив'язаний до конкретної навчальної події.

Четвертою і центральною логічною частиною є модуль реєстрації відвідування. Він забезпечує вибір заняття, автоматичне завантаження списку студентів відповідної групи, встановлення статусів відвідування та збереження результатів. Саме цей модуль реалізує основне призначення застосунку. Для зручності користувача він має підтримувати швидке встановлення статусу для всіх студентів, можливість змінити статус окремого студента, додавання коментаря та перевірку повторного збереження даних для одного й того самого заняття.

П'ятою логічною частиною є модуль журналу та перегляду даних. Він дозволяє користувачеві переглядати раніше збережені записи, застосовувати фільтри за датою, групою, дисципліною або студентом, а також знаходити потрібну інформацію без ручного перегляду всього журналу. Цей модуль є важливим для практичного використання системи, оскільки накопичені дані мають бути доступними для аналізу та перевірки.

Шостою логічною частиною є аналітичний модуль. Його призначення полягає у перетворенні первинних записів журналу на узагальнені показники. До таких показників належать кількість проведених занять, кількість відвіданих занять, кількість пропусків, кількість запізнень, відсоток відвідування окремого студента, середній рівень відвідування групи та список студентів із критично низькою присутністю. Аналітичний модуль підвищує практичну цінність системи, оскільки дозволяє використовувати її не лише як електронний журнал, а й як інструмент моніторингу навчальної активності.

Сьоомою логічною частиною є модуль звітності та експорту даних. Він забезпечує формування підсумкових звітів за обраними параметрами та, за потреби, збереження результатів у зовнішніх форматах, наприклад CSV, Excel або PDF. Такий модуль є корисним для викладачів, кураторів і адміністрації, оскільки дозволяє використовувати дані системи поза межами самого застосунку: для друку, передачі, архівування або подальшого аналізу в електронних таблицях.

Останньою логічною частиною є модуль технічного адміністрування. Він може включати керування користувачами, резервне копіювання бази даних, відновлення інформації, перевірку цілісності даних і налаштування окремих параметрів системи. У межах базової версії застосунку цей модуль може бути реалізований частково, але його наявність в архітектурній моделі є доцільною, оскільки вона показує можливість подальшого розвитку програмного продукту.

Таким чином, архітектурна модель застосунку реєстрації відвідування занять студентами передбачає поділ системи на взаємопов'язані рівні та логічні модулі. Основою організації системи є підхід MVC, який дозволяє відокремити дані, інтерфейс користувача та логіку керування. Додаткове виділення сервісного рівня і рівня доступу до даних забезпечує більшу структурованість коду, спрощує тестування та створює умови для подальшого розширення системи. Запропонована архітектура відповідає функціональним вимогам, визначеним у попередньому розділі, і створює основу для подальшого опису взаємодії користувачів із системою, сценаріїв використання, структури бази даних та UML-моделювання програмного застосунку.

Взаємодія користувача із застосунком реєстрації відвідування занять студентами будується за послідовною логікою: користувач авторизується в системі, обирає необхідний розділ, вводить або переглядає дані, після чого система обробляє запит, звертається до бази даних і повертає результат у зручному для сприйняття вигляді. Така взаємодія відповідає загальній архітектурній моделі, у якій інтерфейс користувача не працює з базою даних безпосередньо, а передає запити до контролерів або сервісів. Це дозволяє розділити відповідальність між окремими частинами застосунку та забезпечити кращу керованість програмного коду.

У типовому сценарії роботи викладач або уповноважена особа спочатку входить до системи, після чого отримує доступ до функцій відповідно до своєї ролі. Якщо користувач має права викладача, він може створити запис про заняття, вибрати навчальну групу, дисципліну, дату та тип заняття. Після цього система автоматично формує список студентів відповідної групи та відображає його у формі реєстрації відвідування. Для кожного студента користувач встановлює статус: «присутній», «відсутній», «запізнився» або «відсутній з поважної причини». Після натискання кнопки збереження система перевіряє коректність введених даних, створює записи журналу відвідування та зберігає їх у базі даних.

Взаємодія між логічними частинами застосунку під час реєстрації відвідування відбувається в кілька етапів. Спочатку рівень представлення отримує від користувача параметри заняття та статуси студентів. Далі контролер приймає ці дані та передає їх до сервісу відвідування. Сервіс перевіряє, чи існує відповідне заняття, чи належать обрані студенти до зазначеної групи, чи не було вже створено записів для цього заняття.

Після перевірки сервіс формує об'єкти записів відвідування та передає їх до рівня доступу до даних. Рівень доступу до даних виконує збереження інформації в базі даних. Після успішного виконання операції результат повертається користувачеві у вигляді повідомлення про збереження або, у разі помилки, у вигляді відповідного попередження.

Окремо слід розглянути взаємодію користувача із журналом відвідування. У цьому випадку користувач відкриває сторінку або форму журналу та задає параметри пошуку: навчальну групу, дисципліну, студента, дату або період. Контролер отримує ці параметри, передає їх до сервісу журналу або сервісу статистики, після чого система виконує вибірку даних із бази. Отримані записи можуть бути відображені у вигляді таблиці, де кожен рядок містить інформацію про заняття, студента, дату, дисципліну та статус відвідування. Якщо користувач обирає статистичний режим, система додатково виконує обчислення кількості занять, кількості відвіданих занять, кількості пропусків, запізнень і відсотка відвідування.

Важливим елементом архітектурної моделі є сценарії використання системи, які описують послідовність дій користувачів під час виконання основних операцій. Перший сценарій пов'язаний із підготовкою системи до роботи. Уповноважена особа або технічний адміністратор створює навчальні групи, додає студентів, формує перелік дисциплін і за потреби створює облікові записи користувачів. Цей сценарій є початковим, оскільки без довідкових даних неможливо створити заняття та виконати реєстрацію відвідування.

Другий сценарій стосується створення заняття. Викладач відкриває відповідну форму, обирає навчальну групу, дисципліну, дату проведення та тип заняття. Система перевіряє, чи заповнено всі обов'язкові поля, і після підтвердження створює запис про заняття. За потреби може бути додано тему заняття або коротку примітку. Створене заняття стає основою для подальшої реєстрації відвідування, оскільки всі записи про присутність студентів мають бути прив'язані до конкретного заняття.

Третій сценарій є основним і описує реєстрацію відвідування. Після вибору створеного заняття система завантажує список студентів відповідної групи. Викладач або уповноважена особа встановлює статус для кожного студента та зберігає результат. Якщо користувач не встановив статус для певного студента, система може або запропонувати заповнити пропущені значення, або автоматично встановити статус за замовчуванням, наприклад «відсутній». Після збереження кожен студент отримує окремий запис у журналі відвідування.

Четвертий сценарій пов'язаний із переглядом журналу. Користувач відкриває журнал відвідування та обирає потрібні параметри фільтрації.

Наприклад, можна переглянути всі записи за певну дату, усі заняття з конкретної дисципліни, історію відвідування певного студента або загальну інформацію по групі. Цей сценарій є важливим для поточного контролю та перевірки правильності внесених даних.

П'ятий сценарій передбачає формування статистики. Користувач обирає групу, дисципліну або період, після чого система аналізує відповідні записи журналу. На основі цих даних обчислюється загальна кількість занять, кількість присутностей, кількість пропусків, кількість запізнень і відсоток відвідування. Також система може формувати перелік студентів із низьким рівнем відвідування, що є корисним для викладача, куратора або адміністрації. Цей сценарій відображає аналітичну складову застосунку.

Шостий сценарій стосується формування звіту та експорту даних. Після перегляду журналу або статистики користувач може сформувати підсумковий звіт. Звіт може бути представлений у вигляді таблиці або окремого документа. У разі реалізації відповідної функції система може експортувати дані у формат CSV, Excel або PDF. Це дозволяє використовувати результати роботи застосунку поза межами системи, наприклад для друку, архівування або подальшого аналізу.

Сьомий сценарій пов'язаний із переглядом власної інформації студентом або батьками. У цьому випадку користувач має обмежений доступ і може переглядати лише ті дані, які стосуються конкретного студента. Такий сценарій може бути реалізований у розширеній версії системи. Його наявність в архітектурній моделі демонструє можливість подальшого розвитку застосунку та впровадження рольового доступу до інформації. Для забезпечення роботи описаних сценаріїв необхідно спроектувати структуру бази даних. База даних є центральним сховищем інформації, у якому зберігаються відомості про користувачів, студентів, навчальні групи, дисципліни, заняття та записи відвідування. Її структура має відповідати логіці предметної області та забезпечувати цілісність, узгодженість і зручність обробки даних. Правильно побудована база даних дозволяє уникнути дублювання інформації, забезпечити зв'язки між сутностями та створити основу для подальшого аналізу відвідування. Основними таблицями бази даних доцільно визначити Users, Roles, Students, Groups, Subjects, Lessons, AttendanceRecord та AttendanceStatuses. За потреби структура може бути розширена таблицями для зберігання звітів, журналу дій користувачів або налаштувань системи. Однак для базової версії застосунку зазначених таблиць достатньо для реалізації основної функціональності.

Таблиця Users призначена для зберігання інформації про користувачів системи. У ній можуть міститися такі поля: ідентифікатор користувача, прізвище, ім'я, електронна пошта або логін, пароль у захищеному вигляді, роль користувача та ознака активності облікового запису. Ця таблиця необхідна для реалізації авторизації та розмежування доступу до функцій системи. Наприклад, викладач може створювати заняття та реєструвати відвідування, студент - переглядати власні дані, батьки - отримувати обмежену інформацію, а технічний адміністратор - керувати обліковими записами.

Таблиця Roles використовується для опису ролей користувачів. Вона може містити ідентифікатор ролі та назву ролі, наприклад «Викладач»,

«Студент», «Уповноважена особа», «Батьки», «Технічний адміністратор». Наявність окремої таблиці ролей дозволяє зробити систему більш гнучкою, оскільки в майбутньому можна буде додавати нові ролі або змінювати права доступу без істотної зміни структури всієї бази даних. Таблиця Groups призначена для зберігання інформації про навчальні групи. Основними полями можуть бути ідентифікатор групи, назва групи, рік вступу або курс, спеціальність чи освітня програма. Ця таблиця використовується для групування студентів і подальшої фільтрації даних журналу відвідування. Зв'язок між групами та студентами дозволяє автоматично завантажувати список студентів під час створення заняття або реєстрації відвідування.

Таблиця Students містить відомості про студентів. До її основних полів належать ідентифікатор студента, прізвище, ім'я, по батькові, ідентифікатор навчальної групи, номер студентського квитка або інший унікальний код, а також, за потреби, зв'язок з обліковим записом користувача. Наявність зв'язку з таблицею Users дозволяє реалізувати особистий кабінет студента, у якому він зможе переглядати власну інформацію про відвідування.

Таблиця Subjects призначена для зберігання переліку навчальних дисциплін. Вона може містити ідентифікатор дисципліни, назву дисципліни, короткий опис або код освітнього компонента. Ця таблиця використовується під час створення занять, перегляду журналу та формування статистики. Завдяки їй система може аналізувати відвідування не лише загалом по групі, а й окремо за кожною дисципліною.

Таблиця Lessons зберігає інформацію про конкретні заняття. Її основними полями можуть бути ідентифікатор заняття, дата проведення, ідентифікатор групи, ідентифікатор дисципліни, тип заняття, тема заняття та ідентифікатор викладача. Ця таблиця є важливою проміжною ланкою між довідковими даними та журналом відвідування. Кожен запис у таблиці AttendanceRecords має бути пов'язаний із конкретним заняттям із таблиці Lessons.

Таблиця AttendanceStatuses містить перелік можливих статусів відвідування. Наприклад, це можуть бути статуси «присутній», «відсутній», «запізнився», «відсутній з поважної причини». Додатково в цій таблиці можна передбачити коефіцієнт, який використовується під час розрахунку статистики. Наприклад, статус «присутній» може мати коефіцієнт 1, «запізнився» - 0,5, «відсутній» - 0, а «відсутній з поважної причини» може не враховуватися як негативний пропуск. Такий підхід робить систему гнучкішою та дозволяє змінювати логіку оцінювання відвідування без зміни програмного коду.

Таблиця AttendanceRecords є центральною таблицею журналу відвідування. Вона містить записи про участь кожного студента в кожному занятті. Основними полями цієї таблиці є ідентифікатор запису, ідентифікатор заняття, ідентифікатор студента, ідентифікатор статусу відвідування, коментар і дата створення запису. Саме на основі цієї таблиці формуються журнал, статистика та звіти. Наприклад, щоб визначити відсоток відвідування студента, система вибирає всі записи цього студента за певний період і підраховує кількість статусів, які відповідають присутності або відсутності.

Узагальнено структуру основних таблиць бази даних можна подати таким чином на табл. 2.1.

Таблиця 2.1

Структура таблиць застосунку

Таблиця Призначення

Users	Зберігання інформації про користувачів системи
Roles	Опис ролей користувачів і основа для розмежування доступу
Groups	Зберігання інформації про навчальні групи
Students	Зберігання персональних і навчальних даних студентів
Subjects	Зберігання переліку навчальних дисциплін
Lessons	Зберігання інформації про проведені або заплановані заняття
AttendanceStatuses	Зберігання можливих статусів відвідування
AttendanceRecords	Зберігання записів журналу відвідування студентів

Зв'язки між таблицями відображають логіку предметної області. Один запис у таблиці Groups може бути пов'язаний із багатьма записами в таблиці Students, оскільки одна навчальна група містить багатьох студентів. Один запис у таблиці Subjects може бути пов'язаний із багатьма заняттями, оскільки з однієї дисципліни може проводитися багато занять. Один запис у таблиці Lessons може бути пов'язаний із багатьма записами в таблиці AttendanceRecords, оскільки на одному занятті фіксується відвідування багатьох студентів. Один студент також може мати багато записів у таблиці AttendanceRecords, оскільки він відвідує багато занять. Таблиця AttendanceStatuses пов'язується з таблицею AttendanceRecords, оскільки кожен запис журналу повинен мати конкретний статус.

З погляду цілісності даних важливо передбачити зовнішні ключі між таблицями. Наприклад, поле GroupId таблиці Students має посилатися на таблицю Groups, поле SubjectId таблиці Lessons - на таблицю Subjects, поле LessonId у таблиці AttendanceRecords - на таблицю Lessons, а поле StudentId - на таблицю Students. Такі зв'язки дозволяють уникнути ситуацій, коли в журналі відвідування з'являється запис для неіснуючого студента або неіснуючого заняття.

Крім того, доцільно передбачити обмеження унікальності. Наприклад, для таблиці AttendanceRecords можна встановити правило, за яким для одного студента не може існувати два записи відвідування в межах одного й того самого заняття. Це запобігає дублюванню даних і підвищує коректність статистичних розрахунків. Також можна обмежити дублювання назв груп або дисциплін, якщо це відповідає логіці системи.

Для зручності подальшої реалізації структуру таблиць можна подати в розширеному вигляді (табл. 2.2.).

Таблиця 2.2

Зв'язки між таблицями бази даних застосунку

Таблиця Основні поля

Roles	RoleId, RoleName
Users	UserId, FullName, Email, PasswordHash, RoleId, IsActive
Groups	GroupId, GroupName, Course, Specialty
Students	StudentId, LastName, FirstName, MiddleName, GroupId, StudentCode, UserId
Subjects	SubjectId, SubjectName, SubjectCode, Description
Lessons	LessonId, LessonDate, GroupId, SubjectId, LessonType, Topic, TeacherId
AttendanceStatuses	StatusId, StatusName, Coefficient, IsNegativeAbsence
AttendanceRecords	RecordId, LessonId, StudentId, StatusId, Comment, CreatedAt

Запропонована структура бази даних (табл. 2.2) відповідає функціональним вимогам застосунку та забезпечує можливість реалізації основних сценаріїв використання. Вона дозволяє створювати навчальні групи, додавати студентів, вести перелік дисциплін, створювати заняття, реєструвати відвідування, переглядати журнал, формувати статистику та готувати звіти. Крім того, така структура є достатньо гнучкою для подальшого розширення системи, наприклад додавання модуля повідомлень, імпорту студентів із файлу, інтеграції з LMS або формування більш складних аналітичних показників.

Рис. 2.3. ER-діаграма бази даних

ER-діаграма на рис. 2.3 відображає структуру бази даних застосунку реєстрації відвідування занять студентами. У центрі моделі розташована таблиця AttendanceRecords, яка зберігає безпосередні записи журналу відвідування. Кожен запис пов'язує конкретного студента, конкретне заняття та встановлений статус відвідування. Саме ця таблиця є основою для подальшого перегляду журналу, фільтрації даних, формування статистики та звітів.

Таблиця Students містить дані про студентів: прізвище, ім'я, по батькові, навчальну групу, студентський код і, за потреби, зв'язок з обліковим записом користувача. Вона пов'язана з таблицею Groups, оскільки кожен студент належить до певної навчальної групи, а одна група може містити багатьох студентів. Таблиця Groups зберігає назву групи, курс і спеціальність.

Таблиця Lessons містить інформацію про заняття: дату, групу, дисципліну, тип заняття, тему та викладача. Вона пов'язана з таблицями Groups, Subjects і Users, оскільки заняття проводиться для конкретної групи, з конкретної дисципліни та певним викладачем. Таблиця Subjects зберігає перелік навчальних дисциплін, їхні коди й описи.

Таблиця AttendanceStatuses містить можливі статуси відвідування: наприклад, «присутній», «відсутній», «запізнився», «відсутній з поважної причини». Для кожного статусу може бути заданий коефіцієнт, який використовується під час розрахунку статистики відвідування. Це дає змогу гнучко враховувати різні ситуації, наприклад часткове зарахування запізнення.

Таблиці Users і Roles забезпечують рольовий доступ до системи. У таблиці Users зберігаються облікові записи користувачів, а таблиця Roles визначає їхні ролі: викладач, студент, уповноважена особа, батьки або технічний адміністратор. Один запис у таблиці Roles може бути пов'язаний із багатьма користувачами. Така структура дозволяє розмежувати права доступу до функцій системи.

Загалом схема побудована за принципом зв'язків «один-до-багатьох». Одна група може мати багато студентів і багато занять, одна дисципліна може бути пов'язана з багатьма заняттями, одне заняття може містити багато записів відвідування, а один студент може мати багато записів у журналі. Для запобігання дублюванню даних передбачено обмеження: у межах одного заняття для одного студента має існувати лише один запис відвідування.

Діаграма компонентів на рис. 2.4 відображає загальну програмну структуру застосунку реєстрації відвідування занять студентами та показує поділ системи на основні рівні: ⁸ рівень представлення, рівень контролерів, сервісний рівень, рівень доступу до даних і базу даних. Користувачі працюють із системою через інтерфейс, після чого їхні запити передаються до відповідних контролерів, які викликають сервіси бізнес-логіки. Сервіси виконують перевірку, обробку та аналіз даних, а для збереження або отримання інформації звертаються до репозиторіїв і контексту бази даних. Окремо виділено додаткові компоненти експорту, резервного копіювання, валідації та журналювання. Така структура забезпечує розділення відповідальності між частинами системи, спрощує супровід програмного коду та створює основу для подальшого розширення застосунку.

Рис. 2.4. Діаграма компонентів застосунку

Отже, взаємодія користувачів із системою, сценарії використання та структура бази даних є взаємопов'язаними елементами архітектурної моделі програмного застосунку. Сценарії використання визначають, які дії виконують користувачі, архітектурні модулі забезпечують обробку цих дій, а база даних зберігає результати роботи системи.

2.2. Обґрунтування вибору інструментів розробки та засобів зберігання даних

Вибір інструментів розробки та засобів зберігання даних є важливим етапом проєктування програмного застосунку, оскільки саме від обраної технологічної основи залежать складність реалізації, зручність супроводу, швидкість розробки, вимоги до обладнання та можливості подальшого розвитку системи. Для застосунку реєстрації відвідування занять студентами навчальної групи необхідно обрати такі засоби, які дозволяють реалізувати основні функції системи без надмірного ускладнення архітектури. До таких функцій належать ведення списку студентів, створення занять, фіксація статусів відвідування, перегляд журналу, формування статистики та збереження даних у структурованому вигляді.

На етапі вибору технологій було розглянуто два можливі підходи до реалізації програмного засобу: створення настільного застосунку на основі Windows Forms або розробка вебзастосунку з використанням ASP.NET Core. Обидва варіанти можуть бути використані для реалізації системи обліку відвідування, оскільки підтримують мову програмування C#, роботу з базами даних, побудову користувацького інтерфейсу та реалізацію бізнес-логіки. Проте ці підходи суттєво відрізняються за складністю розробки, способом розгортання, вимогами до інфраструктури та умовами подальшого використання.

Windows Forms є технологією для створення настільних застосунків у середовищі .NET. Вона дозволяє швидко створювати графічний інтерфейс користувача за допомогою візуального конструктора форм, стандартних елементів керування та подієвої моделі програмування. Для навчального проєкту така технологія є зручною, оскільки дає змогу сконцентруватися на реалізації основної логіки застосунку, роботі з базою даних і побудові зрозумілого інтерфейсу без необхідності налаштовувати серверну інфраструктуру. Windows Forms добре підходить для локальних систем, які використовуються на одному комп'ютері або в межах обмеженого робочого середовища.

ASP.NET Core, навпаки, орієнтований на створення вебзастосунків, які можуть працювати через браузер і бути доступними з різних пристроїв. Такий підхід має низку переваг: централізований доступ до системи, можливість роботи кількох користувачів, перспективи масштабування, інтеграція з іншими вебсервісами, зручніше розгортання в корпоративному середовищі. Для повноцінної системи реєстрації відвідування на рівні закладу освіти вебзастосунок був би перспективним рішенням. Наприклад, ASP.NET Core дозволив би створити систему з особистими кабінетами для викладачів, студентів, батьків та адміністрації, забезпечити доступ через браузер, реалізувати авторизацію, рольове розмежування прав, централізоване зберігання даних і формування звітів у режимі онлайн.

Однак у межах цієї роботи основним завданням є не створення повномасштабної корпоративної інформаційної системи, а розробка працездатного програмного засобу для реєстрації відвідування занять студентами навчальної групи. Тому важливо враховувати не лише

потенційні можливості технології, а й реальні умови виконання роботи: обмежений час, необхідність швидкої реалізації основних функцій, доступність інструментів, простота тестування, відсутність потреби у складному серверному розгортанні та використанні хмарної інфраструктури. Саме ці обставини зумовили вибір Windows Forms як основної технології реалізації застосунку.

Порівняння Windows Forms і ASP.NET Core можна подати за кількома критеріями. За критерієм простоти початкової розробки Windows Forms має перевагу, оскільки дозволяє швидко створити інтерфейс за допомогою конструктора форм, розмістити елементи керування, налаштувати обробники подій і безпосередньо пов'язати їх із логікою застосунку. Для ASP.NET Core потрібно додатково проєктувати структуру вебзастосунку, маршрутизацію, контролери, представлення, налаштовувати роботу вебсервера, обробку HTTP-запитів і механізми взаємодії між клієнтською та серверною частинами. Це робить вебпідхід більш гнучким, але водночас складнішим для реалізації у стислий термін.

За критерієм розгортання Windows Forms також є простішим варіантом. Настільний застосунок може бути запущений на локальному комп'ютері після встановлення необхідного середовища виконання .NET і підключення бази даних. Для демонстрації та апробації достатньо одного робочого місця або локального середовища. У випадку ASP.NET Core необхідно налаштовувати вебсервер або хмарне середовище, наприклад IIS, Docker, Azure App Service чи інші платформи. Також потрібно враховувати питання доменного імені, SSL-сертифікатів, мережевого доступу, захисту серверної частини, резервного копіювання та адміністрування. У межах навчальної роботи такі налаштування можуть суттєво збільшити обсяг завдань і відволікти від основної мети - реалізації функціональності реєстрації відвідування.

За критерієм доступу з різних пристроїв перевагу має ASP.NET Core, оскільки вебзастосунок може бути доступним через браузер із комп'ютера, планшета або смартфона. Windows Forms, як правило, працює на конкретному комп'ютері з операційною системою Windows. Проте для поставленої задачі це обмеження не є критичним, оскільки система розглядається як застосунок для роботи викладача або уповноваженої особи, яка веде журнал відвідування. Для демонстраційної та навчальної реалізації достатньо локального середовища, у якому можна перевірити всі основні функції: додавання студентів, створення занять, реєстрацію відвідування, перегляд журналу та формування статистики.

За критерієм масштабованості ASP.NET Core має кращі перспективи, оскільки може підтримувати одночасну роботу багатьох користувачів, централізоване зберігання інформації та інтеграцію з іншими інформаційними системами. Наприклад, у майбутньому така система могла б бути інтегрована з LMS Moodle, електронним журналом, корпоративною поштою або Microsoft 365. Водночас реалізація таких можливостей потребує додаткових ресурсів, серверної інфраструктури, засобів адміністрування, а також більш складної системи безпеки. Windows Forms має обмеженіші можливості масштабування, але повністю відповідає потребам локального прототипу та дозволяє реалізувати основну логіку системи без надмірного ускладнення.

За критерієм безпеки вебзастосунки потребують більшої уваги, оскільки він потенційно доступний через мережу. Необхідно реалізовувати захист від несанкціонованого доступу, налаштовувати авторизацію, захищати передавання даних, враховувати ризики вебвразливостей і забезпечувати адміністрування користувачів. У Windows Forms ризики також існують, особливо якщо застосунок працює з персональними даними студентів, однак у локальній версії обсяг таких ризиків менший. Дані можуть зберігатися на одному комп'ютері або в локальній базі даних, доступ до якої обмежується користувачами цього робочого середовища. Для навчального проєкту та прототипу це є прийнятним рішенням.

Для реалізації застосунку було обрано мову програмування C#, оскільки вона є сучасною об'єктно-орієнтованою мовою, добре інтегрованою з платформою .NET і зручною для створення настільних застосунків. C# підтримує роботу з класами, колекціями, подіями, винятками, базами даних та іншими механізмами, необхідними для побудови прикладного програмного забезпечення. Крім того, ця мова широко використовується у навчальному процесі під час вивчення програмування, об'єктно-орієнтованого підходу та розробки застосунків для Windows.

Як середовище розробки обрано Visual Studio. Visual Studio є зручним інструментом для створення Windows Forms-застосунків, оскільки містить візуальний дизайнер форм, засоби налагодження, інтеграцію з .NET, інструменти керування пакетами NuGet та можливості роботи з базами даних. За допомогою Visual Studio можна швидко створювати форми, додавати елементи керування, налаштовувати події та перевіряти роботу застосунку.

Для зберігання даних у застосунку доцільно використати локальну базу даних. З огляду на навчальний характер проєкту та відсутність потреби в складній серверній інфраструктурі, найбільш прийнятними варіантами є SQLite.

Для роботи з базою даних обрано Entity Framework Core. Entity Framework Core³ дозволяє працювати з базою даних через об'єкти C#, описувати моделі даних у вигляді класів і зменшити кількість ручного SQL-коду. Це відповідає сучасному підходу до розробки застосунків на платформі .NET і добре поєднується з архітектурною моделлю, описаною в попередньому підрозділі. Для цієї роботи доцільним є використання Entity Framework Core, оскільки він спрощує роботу з моделями даних і полегшує подальше розширення системи.

Важливим аргументом на користь Windows Forms є відповідність цієї технології реальним умовам виконання роботи. Для створення повноцінного вебзастосунку на ASP.NET Core бажано мати більше часу на проєктування серверної частини, інтерфейсу, маршрутизації, авторизації, розгортання та тестування. Крім того, для практичного використання вебсистеми потрібні додаткові інструменти й ресурси: сервер або хмарна платформа, наприклад Azure, налаштування бази даних на сервері, забезпечення захищеного доступу, адміністрування користувачів і резервне копіювання. У межах цієї роботи такі завдання виходять за межі основної мети та можуть значно ускладнити проєкт. Обрання Windows Forms не означає відмову від сучасних принципів побудови програмного забезпечення. Навіть у настільному застосунку можна реалізувати чіткий поділ³ на логічні рівні: інтерфейс користувача, бізнес-логіку, моделі даних і рівень доступу до³ бази даних. Такий підхід дозволяє³ уникнути ситуації, коли весь код зосереджений лише у формах. Форми повинні відповідати переважно за відображення інформації та реакцію на дії користувача, тоді як основні операції з даними мають виконуватися у відповідних сервісах або класах. Це підвищує якість програмної реалізації та створює умови для майбутнього перенесення логіки в інший тип застосунку, наприклад у вебверсію.

Таким чином, у межах цієї роботи доцільно обрати такі інструменти реалізації: мову програмування C#, платформу .NET, технологію Windows Forms для створення графічного інтерфейсу, Visual Studio як основне середовище розробки, SQLite як локальну базу даних, а також Entity Framework Core як засіб доступу до даних. Такий набір інструментів дозволяє реалізувати всі основні функціональні вимоги до застосунку, забезпечити збереження інформації, побудувати зрозумілий інтерфейс і провести тестування системи без необхідності розгортання складної серверної інфраструктури.

2.3. Алгоритм обробки та аналізу даних журналу відвідування занять

Алгоритм обробки та аналізу даних журналу відвідування занять є важливою складовою програмного застосунку, оскільки саме він забезпечує перетворення окремих записів про присутність або відсутність студентів на узагальнені показники, придатні для педагогічного аналізу та моніторингу освітнього процесу. Як було зазначено в п. 1.2, облік відвідування не слід розглядати лише як формальну фіксацію факту присутності студента на занятті. Дані про відвідування можуть виконувати діагностичну, контрольну й аналітичну функції, оскільки дають змогу оцінити рівень залученості студента до навчального процесу, своєчасно виявити ризики зниження навчальної активності та сформувати підстави

для прийняття педагогічних або організаційних рішень.

У традиційному журналі відвідування зазвичай фіксується лише два базові стани: студент був присутній або був відсутній. Такий підхід є простим, але недостатньо гнучким, оскільки він не враховує різні ситуації, що можуть виникати в освітньому процесі. Наприклад, студент міг бути присутнім на занятті повністю, запізнитися, бути відсутнім без поважної причини або мати підтверджену поважну причину відсутності. Якщо всі ці випадки обробляти однаково, статистика відвідування може бути неточною або несправедливою. Тому в межах розроблюваного застосунку доцільно використовувати модифікований алгоритм, який враховує різні статуси відвідування та дозволяє отримати більш змістовний показник участі студента в заняттях.

Вхідними даними для алгоритму є записи журналу відвідування, які зберігаються в базі даних. Кожен запис містить ідентифікатор студента, ідентифікатор заняття, дату, дисципліну, навчальну групу та статус відвідування. До основних статусів можна віднести: «присутній», «відсутній», «запізнився», «відсутній з поважної причини». За потреби перелік статусів може бути розширений, однак для базової версії системи цих значень достатньо для виконання основних аналітичних операцій. Дані можуть аналізуватися за окремим студентом, навчальною групою, дисципліною або періодом.

Першим етапом алгоритму є вибір параметрів аналізу. Користувач задає групу, дисципліну, студента або часовий проміжок, за яким потрібно сформувати статистику. Наприклад, викладач може переглянути відвідування з конкретної дисципліни за місяць, куратор - загальну статистику групи за семестр, а студент - власну історію відвідування. Після цього система виконує запит до бази даних і отримує всі записи журналу, які відповідають заданим параметрам.

Другим етапом є групування отриманих даних. Якщо аналіз виконується для окремого студента, система групує записи за заняттями та підраховує кількість статусів кожного типу. Якщо аналіз виконується для групи, система додатково групує записи за студентами. Якщо користувач обирає дисципліну або період, система враховує лише ті заняття, що відповідають установленим умовам. Така логіка дозволяє формувати як індивідуальну, так і групову статистику.

Третім етапом є підрахунок базових кількісних показників. До них належать загальна кількість занять, кількість занять, на яких студент був присутній, кількість пропусків без поважної причини, кількість запізнень і кількість відсутностей з поважної причини. У простому варіанті відсоток відвідування можна обчислити як відношення кількості відвіданих занять до загальної кількості проведених занять:

$$P = \frac{P}{Z} \times 100\%, \quad (2.1)$$

де (P) - відсоток відвідування, (P) - кількість занять, на яких студент був присутній, (Z) - загальна кількість занять за обраний період.

Однак формула 2.1 не враховує запізнень та поважні причини відсутності. Тому для розроблюваного застосунку доцільно запропонувати модифікований коефіцієнт відвідування, який враховує різні статуси. Наприклад, статус «присутній» може оцінюватися коефіцієнтом 1, статус «запізнився» - коефіцієнтом 0,5, статус «відсутній без поважної причини» - коефіцієнтом 0, а статус «відсутній з поважної причини» може виключатися із загальної кількості занять, які враховуються під час розрахунку. У такому випадку формула може мати вигляд:

$$P = \frac{P}{Z} \times 100\%, \quad (2.2)$$

де (P) - модифікований коефіцієнт відвідування; (P) - кількість занять, на яких студент був присутній; (Z) - кількість занять, на які студент запізнився; (Z) - загальна кількість занять за обраний період; (Z) - кількість відсутностей з поважної причини.

Запропонована формула 2.2 є більш гнучкою, ніж традиційний розрахунок відсотка відвідування, оскільки вона дозволяє диференційовано враховувати різні статуси. Наприклад, запізнень не прирівнюється до повної присутності, але й не вважається повною відсутністю. Відсутність з поважної причини не знижує підсумковий коефіцієнт, оскільки таке заняття фактично вилучається з бази розрахунку. Це дозволяє отримати більш об'єктивний показник навчальної присутності студента.

Для більш універсального опису алгоритму можна подати формулу у зваженому вигляді:

$$P = \frac{P}{Z} \times 100\%, \quad (2.3)$$

де (P) - модифікований коефіцієнт відвідування; (P) - ваговий коефіцієнт певного статусу; (P) - кількість записів із відповідним статусом; (m) - кількість статусів, що враховуються в розрахунку; (Z) - кількість записів, які виключаються з розрахунку, наприклад відсутність з поважної причини. У такій моделі кожному статусу можна надати власну вагу: для присутності (w=1), для запізнень (w=0,5), для відсутності без поважної причини (w=0). Такий підхід дозволяє змінювати правила обчислення без зміни загальної логіки алгоритму.

Практичне значення модифікованої формули полягає в тому, що вона може бути реалізована безпосередньо в структурі бази даних. Як було зазначено в описі ER-моделі, таблиця AttendanceStatuses може містити поле Coefficient, у якому зберігається числове значення статусу.

Наприклад, для статусу «присутній» це значення дорівнює (1), для статусу «запізнився» - (0,5), для статусу «відсутній» - (0). Додаткове поле IsNegativeAbsence або інша службова ознака може визначати, чи враховується цей статус як негативний пропуск. Завдяки цьому алгоритм аналізу не залежить від жорстко заданих текстових назв статусів, а працює з числовими коефіцієнтами.

Після обчислення індивідуального коефіцієнта відвідування система може виконувати додатковий аналіз. Наприклад, вона може визначати студентів із низьким рівнем відвідування. Для цього встановлюється критичне значення коефіцієнта, наприклад (70%). Якщо розрахований показник студента є нижчим за це значення, система може позначити його як такого, що потребує уваги викладача або куратора:

$$P < 0,7; \quad (2.4)$$

де (P) - мінімально допустимий рівень відвідування. Такий підхід відповідає положенням, наведеним у п. 1.2, де облік відвідування розглядався як інструмент раннього виявлення проблем із навчальною активністю студента.

Крім індивідуального аналізу, алгоритм може використовуватися для формування групових показників. Наприклад, середній коефіцієнт відвідування групи можна обчислити як середнє значення індивідуальних коефіцієнтів усіх студентів:

$$\bar{P} = \frac{\sum P_i}{n}, \quad (2.5)$$

де (P) - середній коефіцієнт відвідування навчальної групи; (P) - індивідуальний коефіцієнт відвідування (i)-го студента; (n) - кількість студентів у групі. Цей показник може використовуватися для загального аналізу дисципліни в групі, порівняння відвідування за різними дисциплінами або виявлення періодів, у які відвідування погіршується.

Алгоритм обробки даних журналу відвідування можна подати як послідовність дій. Спочатку користувач обирає параметри аналізу: групу, дисципліну, студента або період. Далі система отримує з бази даних відповідні записи журналу. Після цього записи групуються за студентами та статусами. На наступному етапі для кожного студента підраховується кількість статусів кожного типу, визначається кількість занять, що враховуються в розрахунку, та обчислюється модифікований коефіцієнт відвідування. Потім система порівнює отриманий показник із критичним значенням і, за потреби, формує попередження або список студентів із низьким рівнем відвідування. Завершальним етапом є відображення результатів у журналі, статистичній таблиці або звіті.

У програмній реалізації цей алгоритм може бути розміщений у сервісі статистики, наприклад StatisticsService. Такий сервіс отримує дані з таблиці AttendanceRecords, приєднує до них інформацію про статуси з таблиці AttendanceStatuses, виконує групування, обчислює показники та

передає результат до інтерфейсу користувача. Завдяки цьому інтерфейс не виконує складних розрахунків, а лише відображає вже підготовлену інформацію. Такий підхід відповідає архітектурній моделі, описаній у п. 2.1, де бізнес-логіка була винесена до окремого сервісного рівня. Важливою перевагою запропонованого алгоритму є його гнучкість. Якщо в майбутньому виникне потреба змінити правила підрахунку, наприклад оцінювати запізнення не як (0,5), а як (0,75), це можна буде зробити шляхом зміни коефіцієнта відповідного статусу в базі даних. Якщо потрібно додати новий статус, наприклад «дистанційна участь» або «часткова присутність», його також можна внести до таблиці статусів і задати для нього відповідний коефіцієнт. Це робить систему придатною для подальшого розвитку без повної зміни алгоритму.

Рис. 2.5. Алгоритм обробки даних журналу відвідувань

Рис. 2.5 демонструє три етапи алгоритму аналізу відвідування: вибір параметрів, обробку записів журналу та формування результатів. Окремо показано перевірку критичного рівня відвідування: якщо модифікований коефіцієнт нижчий за встановлений поріг, студент позначається як такий, що потребує уваги викладача або куратора.

Отже, алгоритм обробки та аналізу даних журналу відвідування занять забезпечує перехід від простої фіксації статусів до формування аналітичних показників. На відміну від традиційного підходу, який враховує лише присутність або відсутність, запропонована модифікована формула дозволяє враховувати різні статуси відвідування та їхню вагу. Це підвищує точність аналізу, робить оцінювання більш гнучким і дозволяє використовувати застосунок як інструмент моніторингу навчальної активності студентів. Такий підхід логічно продовжує теоретичні положення, розглянуті в п. 1.2, і створює основу для подальшої програмної реалізації модуля статистики та звітності.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ЗАСТОСУНКУ ТА ЙОГО АПРОБАЦІЯ

3.1. Реалізація програмного застосунку для реєстрації відвідування занять

Реалізація програмного застосунку для реєстрації відвідування занять студентами є практичним етапом роботи, на якому спроектована архітектурна модель, функціональні вимоги, структура бази даних та алгоритм обробки журналу відвідування були втілені у вигляді працездатного програмного засобу. Як було обґрунтовано в підрозділі 2.2, для реалізації застосунку було обрано технологію Windows Forms, мову програмування C# та платформу .NET. Такий вибір дозволив створити настільний застосунок із графічним інтерфейсом користувача, який може працювати локально на комп'ютері викладача або уповноваженої особи без необхідності розгортання складної серверної інфраструктури, хмарних сервісів або вебхостингу.

Основною метою реалізації було створення програмного засобу, який забезпечує ведення списку студентів, створення занять, фіксацію статусів відвідування, збереження даних у базі, перегляд журналу та формування базових статистичних показників. Застосунок розроблено з урахуванням логіки, визначеної у попередніх розділах: дані про студентів, навчальні групи, дисципліни, заняття та записи відвідування мають зберігатися у структурованому вигляді, а користувач повинен мати можливість швидко виконувати основні операції через зрозумілий інтерфейс. Таким чином, програмна реалізація спрямована не лише на створення окремих форм, а й на забезпечення цілісної роботи системи як інструменту автоматизації обліку відвідування.

Проект застосунку було організовано за логікою поділу на окремі функціональні частини. У структурі програмного рішення доцільно виділити папки або групи класів, які відповідають за моделі даних, форми інтерфейсу користувача, роботу з базою даних, бізнес-логіку та допоміжні функції. Такий підхід відповідає архітектурним принципам, описаним у підрозділі 2.1, де система була подана як сукупність рівня представлення, логіки обробки, сервісного рівня та рівня доступу до даних. Навіть у межах Windows Forms-застосунку важливо уникати розміщення всієї логіки безпосередньо у формах, оскільки це ускладнює подальшу підтримку й розширення програми.

Умовно структура проекту може бути подана таким чином: папка Models містить класи предметної області; папка Forms - екранні форми застосунку; папка Data - класи для підключення до бази даних і роботи з нею; папка Services - класи, що реалізують бізнес-логіку; папка Reports або Export - засоби формування звітів чи експорту даних. До основних моделей належать класи Student, Group, Subject, Lesson, AttendanceRecord, AttendanceStatus та User. Ці класи відображають основні таблиці бази даних і забезпечують зв'язок між програмною логікою та збереженими даними.

Клас Student описує студента навчальної групи та містить поля для збереження прізвища, імені, по батькові, ідентифікатора групи та, за потреби, студентського коду. Клас Group містить інформацію про навчальну групу, її назву, курс і спеціальність. Клас Subject використовується для опису навчальної дисципліни. Клас Lesson зберігає інформацію про конкретне заняття: дату, групу, дисципліну, тип заняття та тему. Клас AttendanceStatus описує можливі статуси відвідування, наприклад «присутній», «відсутній», «запізнився», «відсутній з поважної причини», а також може містити коефіцієнт, який використовується для розрахунку модифікованого показника відвідування. Клас AttendanceRecord є центральним у системі, оскільки поєднує студента, заняття та статус відвідування.

Для збереження даних у застосунку використовується локальна база даних. Її структура відповідає ER-моделі, описаній у підрозділі 2.1.

Основними таблицями є Students, Groups, Subjects, Lessons, AttendanceStatuses, AttendanceRecords, а також, за потреби, Users і Roles. Така структура дозволяє зберігати дані не у вигляді розрізнених записів, а як логічно пов'язану систему. Наприклад, студент пов'язаний із навчальною групою, заняття - з дисципліною та групою, а запис відвідування - з конкретним студентом і конкретним заняттям. Це забезпечує можливість фільтрації, пошуку та подальшого аналізу даних.

Рівень доступу до даних реалізує операції створення, читання, оновлення та видалення записів. Для цього використані Entity Framework Core. У проєкті створюється контекст бази даних ApplicationDbContext, у якому визначаються набори даних для кожної сутності: Students, Groups, Subjects, Lessons, AttendanceRecords тощо. Такий підхід дозволяє працювати з таблицями через об'єкти C# і зменшує потребу у великій кількості ручних SQL-запитів.

Важливим етапом реалізації є створення головної форми застосунку. Вона виконує роль центрального вікна, з якого користувач може перейти до основних функціональних модулів. На головній формі доцільно розмістити кнопки або меню для переходу до розділів «Студенти», «Групи», «Дисципліни», «Заняття», «Реєстрація відвідування», «Журнал», «Статистика» та «Звіти». Такий підхід робить інтерфейс зрозумілим і дозволяє користувачеві швидко орієнтуватися в можливостях системи. Головна форма також може відображати коротку службову інформацію: поточного користувача, дату, обрану групу або кількість записів у журналі.

Форма роботи зі студентами забезпечує додавання, редагування та перегляд даних студентів. На ній розміщено таблицю DataGridView, у якій відображається список студентів, а також поля введення для прізвища, імені, по батькові, студентського коду та вибору групи. Користувач заповнює необхідні поля та натискає кнопку збереження, після чого система перевіряє коректність даних і додає нового студента до бази. Якщо користувач обирає вже наявного студента, він може змінити його дані або видалити запис. Така форма реалізує одну з

базових функціональних вимог, сформульованих у підрозділі 1.3.

Форма роботи з навчальними групами та дисциплінами виконує довідкову функцію. Вона дозволяє створювати перелік груп і дисциплін, які надані використовуються під час створення занять і формування журналу відвідування. Наприклад, користувач може додати групу з назвою, курсом і спеціальністю, а також внести дисципліну з назвою та кодом. Збережені групи й дисципліни відображаються у випадкових списках на інших формах, що зменшує кількість ручного введення та знижує ризик помилок.

Форма створення заняття призначена для внесення інформації про конкретне навчальне заняття. Користувач обирає дату, навчальну групу, дисципліну, тип заняття та, за потреби, тему. Після збереження створюється запис у таблиці Lessons. Саме заняття є основою для подальшої реєстрації відвідування, тому система повинна перевіряти, чи заповнені всі обов'язкові поля. Також можна передбачити перевірку на дублювання, щоб уникнути створення двох однакових занять для однієї групи, дисципліни й дати без необхідності.

Центральною частиною програмної реалізації є форма реєстрації відвідування. Вона забезпечує основну функцію застосунку - фіксацію статусів студентів на конкретному занятті. Користувач обирає заняття або задає групу, дисципліну та дату, після чого система автоматично завантажує список студентів відповідної групи. Навпроти кожного студента відображається поле для вибору статусу: «присутній», «відсутній», «запізнився», «відсутній з поважної причини». Після встановлення статусів користувач натискає кнопку збереження, а система створює записи в таблиці AttendanceRecords.

Для зручності роботи на формі реєстрації відвідування можуть бути передбачені додаткові елементи керування. Наприклад, кнопка «Позначити всіх присутніми» дозволяє швидко встановити статус присутності для всієї групи, після чого користувач змінює статус лише для окремих студентів. Також можна додати поле коментаря для кожного запису або загальну примітку до заняття. Це підвищує зручність застосунку й дозволяє адаптувати його до реальних умов роботи викладача.

Форма журналу відвідування забезпечує перегляд раніше збережених записів. На цій формі користувач може обрати групу, дисципліну, студента або період, після чого система відображає відповідні записи в таблиці. Кожен рядок журналу може містити дату заняття, назву дисципліни, прізвище та ім'я студента, статус відвідування і коментар. Для зручності користувача доцільно передбачити фільтрацію, сортування та пошук. Ця форма дозволяє швидко перевірити правильність внесених даних і знайти потрібну інформацію без ручного перегляду всього журналу.

Форма статистики реалізує аналітичну частину застосунку. Вона використовує дані з журналу відвідування та виконує обчислення показників, описаних у підрозділі 2.3. Зокрема, система може визначати кількість проведених занять, кількість присутностей, кількість пропусків, кількість запізнень і модифікований коефіцієнт відвідування. Для цього використовується формула, яка враховує різні статуси відвідування та їхні коефіцієнти. Результати можуть бути подані у вигляді таблиці по студентах або загальної статистики по групі.

Наприклад, для кожного студента система може розраховувати кількість занять, на яких він був присутній, кількість запізнень, кількість пропусків без поважної причини та кількість поважних відсутностей. Після цього обчислюється узагальнений показник відвідування. Якщо цей показник нижчий за встановлений критичний рівень, студент може бути позначений у таблиці як такий, що потребує уваги викладача або куратора. Така реалізація дозволяє використовувати застосунок не лише для ведення журналу, а й для моніторингу навчальної активності студентів.

Окремим елементом реалізації може бути модуль експорту даних. Він дозволяє зберігати журнал або статистику у файл, наприклад у форматі CSV. Це є корисним для подальшої обробки даних в електронних таблицях, друку або передачі звіту відповідальній особі. У базовій версії достатньо реалізувати експорт у CSV, оскільки цей формат є простим, універсальним і підтримується більшістю табличних редакторів. У перспективі можна додати експорт до Excel або PDF.

Під час реалізації важливо забезпечити перевірку коректності введених даних. Наприклад, система не повинна допускати додавати студента без прізвища та імені, створювати заняття без вибору групи або дисципліни, зберігати порожній журнал відвідування або дублювати записи для одного студента в межах одного заняття. Для цього у формах і сервісах застосунку реалізуються перевірки обов'язкових полів, контроль унікальності та повідомлення про помилки. Такі механізми підвищують надійність роботи системи та зменшують ризик некоректних статистичних результатів.

У програмній реалізації важливе значення має обробка подій елементів інтерфейсу. Наприклад, натискання кнопки «Додати студента» викликає метод, який перевіряє введені дані та передає їх до сервісу студентів. Вибір групи на формі реєстрації відвідування запускає завантаження списку студентів цієї групи. Натискання кнопки «Зберегти відвідування» викликає метод збереження записів журналу. Така подієва модель є характерною для Windows Forms і добре відповідає логіці роботи настільного застосунку.

Загальний алгоритм роботи користувача із застосунком можна описати так: спочатку користувач заповнює довідкові дані, тобто створює навчальні групи, додає студентів і дисципліни. Потім створюється заняття з певної дисципліни для конкретної групи. Після цього виконується реєстрація відвідування, під час якої для кожного студента встановлюється відповідний статус. Збережені дані можна переглянути в журналі, проаналізувати у формі статистики або експортувати у файл. Така послідовність дій відповідає сценаріям використання, описаним у попередньому розділі.

Реалізація застосунку на Windows Forms дозволила створити зрозумілий і практичний інструмент для автоматизації обліку відвідування занять. Перевагою такого підходу є швидкість розробки, доступність засобів створення інтерфейсу, можливість локального тестування та відсутність потреби у складному серверному розгортанні. Водночас логічний поділ проєкту на моделі, форми, сервіси та рівень доступу до даних забезпечує структурованість програмного коду й створює передумови для подальшого розвитку системи.

Отже, у процесі реалізації було створено програмний застосунок, який забезпечує основні операції з реєстрації відвідування занять студентами навчальної групи. Застосунок дозволяє вести список студентів, створювати групи й дисципліни, формувати записи про заняття, фіксувати статуси присутності, переглядати журнал, розраховувати статистичні показники та, за потреби, експортувати дані. Реалізація відповідає функціональним вимогам, сформульованим у підрозділі 1.3, архітектурній моделі, описаній у підрозділі 2.1, і алгоритму аналізу даних, поданому в підрозділі 2.3.

Структура проєкту показано на рис. 3.1

```
AttendanceSystem/
|
|— Program.cs
|— AttendanceSystem.csproj
|
|— Data/
|   |— ApplicationDbContext.cs
```

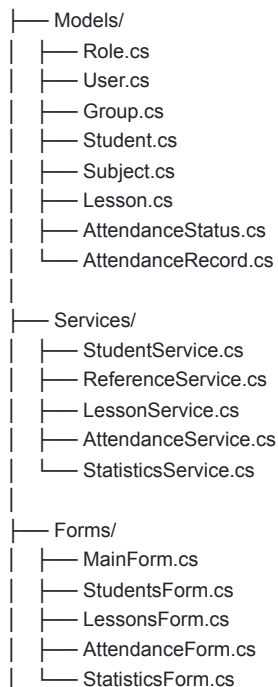



Рис. 3.1 Структура проєкту

У додатку А наведено код основних компонентів системи.

3.2. Опис інтерфейсу користувача та основних сценаріїв роботи із застосунком

Інтерфейс користувача програмного застосунку для реєстрації відвідування занять студентами навчальної групи реалізовано засобами Windows Forms. Такий підхід дозволяє створити настільний застосунок із віконною структурою, зрозумілою для користувача, який має базовий досвід роботи з програмами Windows. Основна ідея інтерфейсу полягає в тому, щоб надати викладачеві або уповноваженій особі швидкий доступ до основних функцій системи: роботи зі студентами, створення занять, реєстрації відвідування, перегляду журналу та формування статистики. Інтерфейс не перевантажується зайвими елементами, а всі дії користувача організовано у вигляді послідовних сценаріїв, що відповідають логіці освітнього процесу.

Після запуску застосунку користувач потрапляє на головну форму. Вона виконує роль навігаційного центру системи та забезпечує перехід до основних модулів. На головній формі розміщено кнопки або пункти меню для відкриття форм «Студенти», «Заняття», «Реєстрація відвідування» та «Статистика». У розширеній версії застосунку тут також можуть бути розміщені кнопки переходу до журналу відвідування, довідників груп і дисциплін, налаштувань користувачів, експорту даних і резервного копіювання. Така структура дає змогу користувачеві швидко зрозуміти призначення програми й перейти до потрібної операції без складної системи вкладених меню.

Форма роботи зі студентами призначена для створення та ведення списку студентів навчальної групи. У лівій або центральній частині форми розміщується таблиця, у якій відображаються вже внесені студенти. Зазвичай така таблиця реалізується за допомогою елемента DataGridView і містить прізвище, ім'я, по батькові, навчальну групу та студентський код. У правій частині форми можуть бути розміщені поля введення для додавання нового студента: прізвище, ім'я, по батькові, студентський код і вибір навчальної групи. Після заповнення необхідних полів користувач натискає кнопку «Додати студента», а система перевіряє коректність даних і зберігає запис у базі даних. Якщо обов'язкові поля не заповнені, користувач отримує повідомлення про помилку.

Форма студентів реалізує один із базових сценаріїв використання - підготовку системи до роботи. Перед тим як реєструвати відвідування, потрібно сформувати перелік студентів, які належать до відповідної навчальної групи. Цей сценарій виконується за такою логікою: користувач відкриває форму «Студенти», обирає або створює навчальну групу, вводить дані студента, зберігає запис і перевіряє його появу в таблиці. У подальшому саме ці дані автоматично використовуються під час створення журналу відвідування, тому користувачеві не потрібно щоразу вводити студентів вручну.

Форма створення занять забезпечує внесення інформації про конкретне навчальне заняття. На ній користувач обирає дату заняття, навчальну групу, дисципліну, викладача та тип заняття. Тип заняття може визначатися зі стандартного списку: лекція, практичне заняття, лабораторне заняття або семінар. Також може бути передбачене поле для введення теми заняття. Після натискання кнопки «Створити заняття» система перевіряє, чи заповнені всі обов'язкові поля, і створює відповідний запис у базі даних. Створене заняття в подальшому використовується як основа для реєстрації відвідування студентів.

Основний сценарій використання форми занять полягає в тому, що викладач або уповноважена особа заздалегідь або безпосередньо перед проведенням заняття створює запис про навчальну подію. Наприклад, користувач обирає групу «ІПЗ-21», дисципліну «Об'єктно-орієнтоване програмування», дату проведення, тип заняття «Лабораторне заняття» та вводить тему. Після збереження це заняття стає доступним у формі реєстрації відвідування. Завдяки такому підходу кожен запис журналу пов'язується не просто з датою, а з конкретною дисципліною, групою та типом навчального заняття.

Центральною частиною інтерфейсу є форма реєстрації відвідування. Вона призначена для фіксації статусу кожного студента на конкретному занятті. Користувач спочатку обирає заняття зі списку. Після цього натискає кнопку «Завантажити студентів», і система автоматично відображає список студентів відповідної групи. Навпроти кожного студента розміщується поле вибору статусу. До основних статусів належать: «присутній», «відсутній», «запізнився», «відсутній з поважної причини». Також передбачено поле для коментаря, у якому можна вказати додаткову інформацію, наприклад причину відсутності або службову примітку.

Для підвищення зручності роботи на формі реєстрації відвідування передбачено кнопку «Позначити всіх присутніми». Вона дозволяє швидко встановити статус «присутній» для всіх студентів групи, після чого користувач може вручну змінити статус лише для тих студентів, які були відсутні, запізнюючись або мали поважну причину. Такий підхід значно скорочує час заповнення журналу, особливо якщо більшість студентів були присутні на занятті. Після перевірки встановлених статусів користувач натискає кнопку «Зберегти відвідування», і система записує дані в таблицю журналу.

Сценарій реєстрації відвідування є головним у роботі застосунку. Він складається з кількох послідовних дій: користувач відкриває форму реєстрації, обирає потрібне заняття, завантажує список студентів, встановлює статуси відвідування, за потреби додає коментарі та зберігає результат. Після збереження кожен студент отримує окремий запис у журналі відвідування, пов'язаний із конкретним заняттям. Якщо для певного студента на цьому занятті вже існує запис, система може не створювати дубль, а оновлювати наявний статус. Це дозволяє уникнути помилок і забезпечити коректність статистичних розрахунків.

Форма статистики призначена для аналізу накопичених даних про відвідування. На ній користувач обирає навчальну групу, після чого натискає кнопку «Розрахувати статистику». Система отримує з бази даних записи відвідування, групує їх за студентами та обчислює основні показники: загальну кількість занять, кількість занять, що враховуються в розрахунку, кількість присутностей, кількість запізнень, кількість пропусків, кількість відсутностей із поважної причини та відсоток відвідування. Результати відображаються у таблиці, де кожен рядок відповідає окремому студенту.

Особливістю статистичної форми є використання модифікованого коефіцієнта відвідування, описаного в підрозділі 2.3. На відміну від простого підрахунку присутніх і відсутніх, система враховує різні статуси відвідування та їхні коефіцієнти. Наприклад, присутність враховується повністю, запізнення може враховуватися частково, а відсутність з поважної причини може виключатися із загальної кількості занять, які беруть участь у розрахунку. Якщо розрахований показник студента є нижчим за критичний поріг, система може позначити його як студента групи ризику. Це дозволяє використовувати застосунок не лише для обліку, а й для моніторингу навчальної активності.

Окремий сценарій пов'язаний із переглядом журналу відвідування. У базовій реалізації журнал може відображатися через окрему форму або бути частиною статистичного модуля. Його завдання полягає у відображенні первинних записів: дата заняття, дисципліна, група, студент, статус і коментар. Для зручності користувача доцільно передбачити фільтри за датою, дисципліною, студентом або групою. Такий функціонал дозволяє швидко знайти потрібну інформацію, перевірити правильність внесених даних або підготувати інформацію для звіту.

Сценарій перегляду журналу може виконуватися після реєстрації відвідування або наприкінці певного періоду. Наприклад, викладач відкриває журнал, обирає дисципліну та групу, після чого переглядає всі записи за конкретний місяць. Якщо виявлено помилку, наприклад неправильний статус студента, у подальшому може бути реалізована функція редагування запису. У базовій версії достатньо забезпечити перегляд і фільтрацію даних, а редагування можна розглядати як напрям подальшого розвитку системи.

Застосунок також може містити сценарій експорту даних. Його призначення полягає в тому, щоб користувач міг зберегти журнал або статистику у зовнішній файл. Найпростішим варіантом є експорт у формат CSV, який підтримується більшістю електронних таблиць. Користувач формує потрібну вибірку, натискає кнопку експорту, обирає місце збереження файлу, після чого система створює файл із відповідними даними. Така функція є корисною для подальшого аналізу, друку або передачі інформації куратору чи адміністрації.

З погляду зручності користування інтерфейс побудовано за принципом мінімальної кількості дій для виконання основної операції. Для реєстрації відвідування користувачеві потрібно лише вибрати заняття, завантажити список студентів, змінити окремі статуси та зберегти результат. Для перегляду статистики достатньо обрати групу й натиснути кнопку розрахунку. Така логіка відповідає практичним умовам роботи викладача, оскільки під час заняття немає можливості витратити багато часу на складні налаштування.

Важливим елементом інтерфейсу є система повідомлень. Якщо користувач виконує дію успішно, програма відображає повідомлення про збереження даних, створення заняття або додавання студента. Якщо виникає помилка, наприклад не обрано групу, не вказано дисципліну, порожнє поле прізвища або спроба створити дубль заняття, система повідомляє про це у зрозумілій формі. Такі повідомлення допомагають користувачеві швидко виправити помилки та підвищують надійність роботи застосунку.

У межах реалізації Windows Forms важливу роль відіграють стандартні елементи керування. Для відображення списків і результатів використовується DataGridView, для вибору груп, дисциплін, занять і статусів - ComboBox, для вибору дати - DateTimePicker, для введення текстових даних - TextBox, для запуску дій - Button, а для пояснювальних підписів - Label. Використання стандартних елементів робить інтерфейс знайомим для користувача й не потребує додаткового навчання.

Таким чином, інтерфейс користувача застосунку побудовано навколо основних сценаріїв роботи: підготовка довідкових даних, створення заняття, реєстрація відвідування, перегляд журналу, розрахунок статистики та можливий експорт результатів. Кожна форма відповідає окремому функціональному модулю, а їхня взаємодія забезпечує послідовну логіку роботи системи. Такий підхід дозволяє зробити застосунок простим у використанні, достатньо функціональним для потреб навчальної групи та придатним для подальшого розширення.

3.3. Налаштування, тестування та апробація програмного засобу

Налаштування, тестування та апробація програмного засобу є завершальним етапом практичної реалізації системи реєстрації відвідування занять студентами навчальної групи. Якщо попередні підрозділи були спрямовані на опис структури застосунку, його інтерфейсу та основних функцій, то в цьому підрозділі основну увагу приділено перевірці працездатності створеної системи, виявленню можливих помилок і підтвердженню відповідності програмного засобу функціональним вимогам, сформульованим у підрозділі 1.3. Тестування дозволяє переконатися, що застосунок коректно додає студентів, створює заняття, зберігає записи відвідування, формує журнал і розраховує статистичні показники.

Під час налагодження перевірялася правильність роботи окремих модулів застосунку та їхня взаємодія між собою. Особлива увага приділялася формам введення даних, обробці дій користувача, збереженню інформації в базі даних і правильності розрахунку показників відвідування.

Оскільки застосунок реалізовано за допомогою Windows Forms, основними об'єктами перевірки були форми, кнопки, таблиці, випадючі списки, поля введення та методи, які виконуються у відповідь на події користувача. Налаштування здійснювалося шляхом покрокового запуску програми, перевірки значень змінних, аналізу повідомлень про помилки та тестування основних сценаріїв роботи.

На першому етапі перевірялася коректність створення та ініціалізації бази даних. Після першого запуску застосунку система повинна створити файл локальної бази даних, сформувати необхідні таблиці та додати початкові довідкові дані, зокрема ролі користувачів і статуси відвідування. До таких статусів належать «присутній», «запізнився», «відсутній» і «відсутній з поважної причини». Перевірка цього етапу є важливою, оскільки без правильної структури бази даних неможлива подальша робота із заняттями, студентами та журналом відвідування.

На другому етапі налагодження перевірялася робота форми студентів. Для цього виконувалося додавання тестових студентів, перевірялася їх поява в таблиці, правильність прив'язки до навчальної групи та збереження в базі даних. Також перевірялося, як система реагує на помилкові дії

користувача: спробу додати студента без прізвища, без імені або без вибору групи. У таких випадках застосунок повинен не зберігати некоректний запис, а виводити користувачеві зрозуміле повідомлення про помилку. Така перевірка підтверджує, що система має базові механізми валідації введених даних.

На третьому етапі перевірялася форма створення занять. Тестування передбачало вибір дати заняття, навчальної групи, дисципліни, типу заняття та викладача. Після збереження перевірялося, чи з'являється нове заняття у списку та чи доступне воно для подальшої реєстрації відвідування. Окремо перевірялася ситуація створення дубльованого заняття для тієї самої групи, дисципліни, дати й типу заняття. У цьому випадку система повинна запобігти повторному створенню однакового запису, оскільки дублювання занять може призвести до помилок у журналі та статистиці.

На четвертому етапі тестувалася основна функція системи - реєстрація відвідування. Користувач обирав створене заняття, завантажував список студентів відповідної групи, встановлював для кожного студента статус відвідування та зберігав результат. Після цього перевірялося, чи були створені записи в журналі, чи правильно збережено статуси та чи не виникають дублікати для одного студента в межах одного заняття. Якщо користувач повторно зберігає відвідування для того самого заняття, система має оновлювати наявні записи, а не створювати нові. Це є важливим для забезпечення цілісності даних.

На п'ятому етапі перевірялася робота статистичного модуля. Для тестової групи створювалися кілька занять, після чого для студентів задавалися різні статуси: присутність, запізнення, відсутність і відсутність з поважної причини. Потім система розраховувала статистику відвідування. Перевірялося, чи правильно підраховується загальна кількість занять, кількість занять, що враховуються в розрахунок, кількість присутностей, пропусків, запізнень і поважних відсутностей. Особлива увага приділялася правильності роботи модифікованої формули, описаної в підрозділі 2.3, яка враховує вагові коефіцієнти різних статусів.

Наприклад, якщо студент мав 4 заняття, з яких на двох був присутній, на одному запізнився, а одне пропустив з поважної причини, то в розрахунок враховується не 4, а 3 заняття. Присутність оцінюється як 1, запізнення - як 0,5, а поважна причина виключається з розрахунку. У такому випадку модифікований коефіцієнт відвідування становить:

$$= \times 100\% = \frac{3}{4} \times 100\% = 83,33\% \quad (3.6)$$

Такий приклад підтверджує, що система не просто рахує кількість присутностей, а використовує більш гнучкий підхід до оцінювання участі студента в заняттях.

Для систематизації результатів тестування було складено набір тестових сценаріїв (табл.3.1), які охоплюють основні функції застосунку.

Таблиця 3.1

Тестові сценарії

No	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
1	Запуск застосунку та створення бази даних	База даних створюється автоматично	Базу даних створено	Успішно
2	Додавання нового студента	Студент з'являється у списку	Студента додано	Успішно
3	Додавання студента без прізвища	Система показує повідомлення про помилку	Повідомлення показано	Успішно
4	Створення нового заняття	Заняття зберігається в базі даних	Заняття створено	Успішно
5	Створення дубльованого заняття	Система забороняє дублювання	Виведено повідомлення про помилку	Успішно
6	Завантаження списку студентів для заняття	Відображається список студентів групи	Список завантажено	Успішно
7	Реєстрація відвідування	Статуси студентів зберігаються в журналі	Дані збережено	Успішно
8	Повторне збереження відвідування	Записи оновлюються без дублювання	Записи оновлено	Успішно
9	Розрахунок статистики	Система обчислює відсоток відвідування	Статистику сформовано	Успішно
10	Виявлення студента з низьким відвідуванням	Студент позначається як група ризику	Ознаку сформовано	Успішно

Окремо перевірялися помилкові та нестандартні ситуації. До них належали спроби зберегти порожню форму, виконати реєстрацію без вибору заняття, створити заняття без дисципліни або групи, а також розрахувати статистику для групи, у якій ще немає записів відвідування. У таких випадках система повинна не завершувати роботу аварійно, а виводити повідомлення, яке пояснює користувачеві причину помилки. Це підвищує надійність застосунку та зменшує ризик втрати або пошкодження даних.

Апробація програмного засобу може бути проведена на прикладі умовної навчальної групи. Для цього в систему було внесено тестову групу, кількох студентів, перелік дисциплін і кілька занять. Після цього для кожного заняття було виконано реєстрацію відвідування з різними статусами студентів. Отримані записи були використані для формування статистики. Така апробація дозволила перевірити повний цикл роботи застосунку: від створення початкових даних до отримання аналітичного результату.

Результати апробації показали, що застосунок дозволяє скоротити час, необхідний для заповнення журналу відвідування, порівняно з ручним обліком. Особливо це помітно завдяки можливості автоматичного завантаження списку студентів групи та функції позначення всіх студентів як присутніх. Користувачеві потрібно змінити статус лише для тих студентів, які були відсутні, запізнилися або мали поважну причину. Це робить процес реєстрації швидшим і зручнішим.

Порівняння ручного та автоматизованого підходів до обліку відвідування можна подати табл. 3.2.

Таблиця 3.2

Порівняння підходів до обліку відвідування

Критерій Ручний облік Розроблений застосунок

Заповнення списку студентів Потрібно повторювати вручну Список завантажується автоматично

Фіксація статусів Виконується в паперовому журналі або таблиці Виконується через форму застосунку

Пошук інформації Ускладнений Можливий через журнал і фільтрацію

Підрахунок пропусків Виконується вручну Виконується автоматично

Розрахунок відсотка відвідування Потребує ручних обчислень Формується системою

Ризик дублювання записів Достатньо високий Зменшується за рахунок перевірок

Формування статистики Потребує додаткового часу Виконується швидко

Проведене тестування підтвердило, що розроблений застосунок виконує основні функції, визначені на етапі формування вимог. Він дозволяє працювати зі студентами, створювати заняття, реєструвати статуси відвідування, зберігати записи в базі даних і формувати статистику. Виявлені

під час налагодження помилки були пов'язані переважно з перевіркою введених даних, дублюванням записів і коректним оновленням таблиць інтерфейсу. Після внесення відповідних змін ці проблеми було усунуто.

Водночас апробація показала й напрямки подальшого вдосконалення системи. Доцільно додати окрему форму для перегляду журналу з розширеними фільтрами, реалізувати експорт статистики у формат CSV або Excel, передбачити повноцінну авторизацію користувачів і розмежування прав доступу, а також додати можливість редагування раніше створених записів. У перспективі система може бути розширена до вебверсії або інтегрована з електронним журналом, LMS чи іншими інформаційними ресурсами закладу освіти.

Отже, налагодження, тестування та апробація підтвердили працездатність розробленого програмного засобу й відповідність його основним функціональним вимогам. Система забезпечує автоматизацію ключових операцій обліку відвідування, зменшує обсяг ручної роботи, підвищує точність збереження даних і дозволяє формувати статистичні показники на основі різних статусів відвідування. Це підтверджує практичну доцільність використання застосунку для реєстрації та аналізу відвідування занять студентами навчальної групи.

ЗАГАЛЬНІ ВИСНОВКИ

У роботі було розглянуто актуальну проблему автоматизації реєстрації відвідування занять студентами навчальної групи. Актуальність теми зумовлена тим, що традиційні способи ведення журналу відвідування, зокрема паперові журнали, окремі електронні таблиці або неструктуровані записи, не завжди забезпечують зручний пошук інформації, швидке формування статистики та своєчасне виявлення студентів із низьким рівнем навчальної активності. В умовах цифровізації освітнього процесу виникає потреба у простих і доступних програмних засобах, які дозволяють не лише фіксувати факт присутності або відсутності студента, а й зберігати ці дані у структурованому вигляді, обробляти їх та використовувати для моніторингу якості навчання. Метою роботи була розробка програмної системи для реєстрації, збереження, перегляду та аналізу відвідування занять студентами навчальної групи.

У процесі виконання роботи було проаналізовано традиційні та цифрові підходи до реєстрації відвідування занять студентами. Показано, що традиційні методи, попри свою простоту, мають низку недоліків, пов'язаних із ризиком помилок, складністю пошуку записів, потребою в ручних підрахунках і незручністю формування звітності. Водночас цифрові підходи, зокрема електронні журнали, таблиці, вебсервіси, LMS та спеціалізовані програмні застосунки, створюють умови для швидшого внесення даних, їх систематизації та подальшого аналізу. Було обґрунтовано, що для потреб окремої навчальної групи доцільним є створення локального програмного засобу, який забезпечує базові операції з обліку відвідування без необхідності розгортання складної корпоративної інформаційної системи.

У роботі було розкрито теоретичні основи обліку відвідування та його роль у моніторингу якості освітнього процесу. Показано, що дані про відвідування можуть виконувати не лише контрольну, а й діагностичну та аналітичну функції. Вони дозволяють оцінювати рівень залученості студентів до навчання, виявляти негативні тенденції, формувати підстави для роботи викладача, куратора або уповноваженої особи. Було сформульовано функціональні вимоги до майбутнього застосунку, серед яких ведення списку студентів, створення навчальних груп і дисциплін, створення занять, реєстрація статусів відвідування, перегляд журналу, формування статистики та можливість подальшого розширення системи. На етапі проєктування було розроблено архітектурну модель програмного застосунку. Запропоновано логічний поділ системи на рівень представлення, рівень керування, сервісний рівень, рівень доступу до даних і базу даних. Показано можливість використання принципів MVC навіть у межах настільного Windows Forms-застосунку, де форми виконують роль представлення, моделі описують предметну область, а сервіси реалізують основну бізнес-логіку. Було визначено основні логічні модулі системи: модуль роботи зі студентами, модуль груп і дисциплін, модуль створення занять, модуль реєстрації відвідування, модуль журналу, модуль статистики та модуль звітності. Такий поділ дозволив зробити структуру застосунку зрозумілою, керованою та придатною до подальшого вдосконалення.

У межах проєктування було сформовано структуру бази даних системи. Розроблено набір основних таблиць, зокрема Users, Roles, Groups, Students, Subjects, Lessons, AttendanceStatuses та AttendanceRecords. Показано зв'язки між ними: навчальна група містить багатьох студентів, дисципліна пов'язана з багатьма заняттями, заняття містить багато записів відвідування, а кожен запис журналу поєднує конкретного студента, конкретне заняття та відповідний статус. Було обґрунтовано необхідність використання зовнішніх ключів і обмежень унікальності, зокрема для запобігання дублюванню записів відвідування одного студента в межах одного заняття. Також було підготовлено UML-діаграми варіантів використання, архітектури, ER-модель, діаграму компонентів, діаграму діяльності алгоритму та діаграму класів.

У роботі було обґрунтовано вибір інструментів розробки та засобів зберігання даних. Розглянуто можливість реалізації системи у вигляді вебзастосунку на ASP.NET Core та настільного застосунку на Windows Forms. Було показано, що ASP.NET Core має ширші перспективи для масштабування, віддаленого доступу та інтеграції з іншими інформаційними системами, однак потребує більше часу, серверної або хмарної інфраструктури, налаштування безпеки та адміністрування. З огляду на обмеженість часу, навчальний характер роботи та потребу створення працездатного локального прототипу було обрано Windows Forms, мову програмування C#, платформу .NET, локальну базу даних SQLite або SQL Server LocalDB та засоби доступу до даних Entity Framework Core.

Окрему увагу приділено алгоритму обробки та аналізу даних журналу відвідування. Було запропоновано модифікований підхід до розрахунку показника відвідування, який враховує різні статуси: «присутній», «запізнівся», «відсутній», «відсутній з поважної причини». На відміну від простого підрахунку присутностей і пропусків, запропонована формула використовує вагові коефіцієнти статусів, що дозволяє більш гнучко оцінювати участь студента в заняттях. Зокрема, присутність враховується повністю, запізнення - частково, а відсутність із поважної причини може виключатися з розрахунку. Такий підхід підвищує аналітичну цінність системи та дозволяє використовувати її як інструмент моніторингу навчальної активності.

У процесі реалізації було створено програмний застосунок для реєстрації відвідування занять студентами навчальної групи. Реалізовано основні моделі даних, зокрема Student, Group, Subject, Lesson, AttendanceStatus, AttendanceRecord, User і Role. Створено контекст бази даних, який забезпечує роботу з таблицями та зв'язками між ними. Реалізовано сервіси для роботи зі студентами, довідковими даними, заняттями, записами відвідування та статистикою. Також було розроблено основні форми інтерфейсу: головну форму, форму роботи зі студентами, форму створення занять, форму реєстрації відвідування та форму статистики. Застосунок дозволяє додавати студентів, створювати заняття, завантажувати список студентів групи, встановлювати статуси відвідування, зберігати записи та розраховувати статистичні показники.

Проведене налагодження, тестування та апробація підтвердили працездатність створеного програмного засобу. Було перевірено запуск застосунку, створення бази даних, додавання студентів, створення занять, реєстрацію відвідування, оновлення записів без дублювання, обробку помилкових ситуацій і формування статистики. Показано, що застосунок скорочує час заповнення журналу, зменшує ризик помилок під час підрахунків і забезпечує швидке отримання узагальненої інформації про відвідування. Апробація на прикладі умовної навчальної групи засвідчила, що система може бути використана як практичний інструмент для підтримки роботи викладача або уповноваженої особи.

Отже, у роботі було виконано поставлені завдання: проаналізовано підходи до реєстрації відвідування занять, визначено роль обліку

відвідування у моніторингу якості освітнього процесу, сформульовано функціональні вимоги до системи, спроектовано архітектуру та структуру бази даних, обґрунтовано вибір інструментів розробки, запропоновано алгоритм аналізу журналу відвідування, реалізовано програмний застосунок і проведено його тестування. Результати роботи підтверджують доцільність розробки системи реєстрації відвідування занять студентами навчальної групи та можливість її подальшого вдосконалення шляхом додавання авторизації користувачів, розширених звітів, експорту даних, інтеграції з електронними освітніми платформами або створення вебверсії системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ab Wahid R. Sustaining ISO 9001-based QMS in higher education: a reality. *The TQM Journal*. 2019. Vol. 31, no. 4. P. 563-577. URL <https://doi.org/10.1108/tqm-12-2018-0185> (date of access: 04.05.2026).
2. Bernstein D., Booch G. The UML and the Rational Unified Process. *IEEE Software*. 2020. Vol. 37, no. 6. P. 12. URL <https://doi.org/10.1109/ms.2020.3019539>.
3. Booch G. UML in action. *Communications of the ACM*. 1999. Vol. 42, no. 10. P. 26-28. URL <https://doi.org/10.1145/317665.317672>.
4. Booch. *The Unified Modeling Language User Guide*. PEARSON EDUCATION, 2002.
5. Brophy J. E., Good T. L. Teachers' communication of differential expectations for children's classroom performance: Some behavioral data. *Journal of Educational Psychology*. 1970. Vol. 61, no. 5. P. 365-374. URL <https://doi.org/10.1037/h0029908>.
6. Brophy J., Good T. *Teacher behavior and student achievement*. New York: McMillan, 1986. 174 p. URL <https://files.eric.ed.gov/fulltext/ED251422.pdf>.
7. Byrne G. .NET Target C#. Berkeley, CA, 2022. P. 1-11. URL https://doi.org/10.1007/978-1-4842-8619-7_1.
8. Contributors to Wikimedia projects. Learning analytics - Wikipedia. *Wikipedia, the free encyclopedia*. URL https://en.wikipedia.org/wiki/Learning_analytics.
9. Danielson C. *Enhancing Professional Practice: A Framework for Teaching*. Association for Supervision & Curriculum Development, 2007.
10. Database Design. Upper Saddle River: Pearson Education, 2003.
11. Duckworth J. M. D. A Mixed Methods Study on the Relationship between JobFit, TeacherFit, Morgan & Associates Video Screener, and iObservation in a Suburban Midwest Public School: thesis. 2019. URL <http://pqdtopen.proquest.com/#viewpdf?dispub=13426676> (date of access: 04.05.2026).
12. ENHANCEMENT TESTING PROCESS IN LMS MOODLE. / V. Blinova et al. *International Journal of Advanced Studies*. 2014. Vol. 4, no. 1. P. 39-43. URL <https://doi.org/10.12731/2227-930x-2014-1-7>.
13. Gorman B. L. *Introduction to Entity Framework*. Practical Entity Framework Core 6. Berkeley, CA, 2021. P. 3-26. URL https://doi.org/10.1007/978-1-4842-7301-2_1.
14. *Handbook for inspecting secondary schools: Effective from September 2003*. ed. by Ofsted. London: Ofsted, 2003. 202 p.
15. Hilberg R. S., Waxman H. C., Sharp R. G. *Purposes and Perspectives on Classroom Observation Research*. Cambridge University Press. URL https://assets.cambridge.org/9780521814539/excerpt/9780521814539_excerpt.pdf.
16. Import test questions into Moodle LMS. / S. Mintii et al. *Освітній вимір*. 2019. Vol. 53, no. 1. P. 111-124. URL <https://doi.org/10.31812/educdim.v53i1.3836>.
17. Jaiswal M. *Software Architecture and Software Design*. SSRN Electronic Journal. 2019. URL <https://doi.org/10.2139/ssrn.3772387>.
18. La Paro K. M., Pianta R. C., Stuhlman M. The Classroom Assessment Scoring System: Findings from the Prekindergarten Year. *The Elementary School Journal*. 2004. Vol. 104, no. 5. P. 409-426. URL <https://doi.org/10.1086/499760> (date of access: 04.05.2026).
19. *Learning advanced SQL*. ed. by W. S. (Firm). Englewood Cliffs, NJ: Prentice Hall, 1992. 213 p.
20. Lengyel P. Practical experiences in Moodle LMS. *Acta Agraria Debreceniensis*. 2008. No. 29. P. 129-133. URL <https://doi.org/10.34101/actaagrar/29/2977>.
21. Martinez F., Taut S., Schaaf K. Classroom observation for evaluating and improving teaching: An international perspective. *Studies in Educational Evaluation*. 2016. Vol. 49. P. 15-29. URL <https://doi.org/10.1016/j.stueduc.2016.03.002>.
22. McGregor D. Software Architecture. *The Journal of Object Technology*. 2004. Vol. 3, no. 5. P. 65. URL <https://doi.org/10.5381/jot.2004.3.5.c7>.
23. Medley D. M., Mitzel H. E. A technique for measuring classroom behavior. *Journal of Educational Psychology*. 1958. Vol. 49, no. 2. P. 86-92. URL <https://doi.org/10.1037/h0040378>.
24. Medley D. M. *Studies of teacher behavior: Inferring classroom behavior from pupil responses*. New York: Division of Teacher Education, Board of Higher Education of the City of New York, 1956. 16 p.
25. Medley D. M. *Studies of teacher behavior: The refinement of two techniques for observing teachers' classroom behaviors*. New York: Division of Teacher Education, Board of Higher Education of the City of New York, 1955. 42 p.
26. Miyachi C. Agile software architecture. *ACM SIGSOFT Software Engineering Notes*. 2011. Vol. 36, no. 2. P. 1-3. URL <https://doi.org/10.1145/1943371.1943388>.
27. Providing Adaptivity in Moodle LMS Courses. / E. Shchedrina et al. *International Journal of Emerging Technologies in Learning (IJET)*. 2021. Vol. 16, no. 02. P. 95. URL <https://doi.org/10.3991/ijet.v16i02.18813>.
28. Sharp J. *Microsoft Visual C# [sharp]. Net step by step*. Redmond, Wash: Microsoft Press, 2002. 621 p.
29. Shuster M. P. *The 2012 Essential Guide to Walk-throughs using eWalk: How to raise student achievement through classroom observations and data analysis*. CreateSpace Independent Publishing Platform, 2012. 184 p.
30. Six models of lesson observation: an international perspective. *University of Bristol*. URL <https://my.chartered.college/wp-content/uploads/2021/10/Six-Models-of-Lesson-Observations.pdf>.
31. Software architecture. / B. Anderson et al. *ACM SIGPLAN Notices*. 1993. Vol. 28, no. 10. P. 356-359. URL <https://doi.org/10.1145/167962.165922>.
32. The Individualized Classroom Assessment Scoring System (inCLASS): Preliminary reliability and validity of a system for observing preschoolers' competence in classroom interactions. / J. T. Downer et al. *Early Childhood Research Quarterly*. 2010. Vol. 25, no. 1. P. 1-16. URL <https://doi.org/10.1016/j.ecresq.2009.08.004> (date of access: 04.05.2026).
33. The Observer XT | Behavioral coding - Event logging software. *Noldus*. URL <https://noldus.com/observer-xt-human>.
34. Unlock the potential in every teacher. *TeachBoost*. URL <https://teachboost.com/>.
35. Wade R. *Reading CSV Files*. Advanced Analytics in Power BI with R and Python. Berkeley, CA, 2020. P. 151-175. URL https://doi.org/10.1007/978-1-4842-5829-3_3.

ДОДАТКИ

ДОДАТОК А. КОД ОСНОВНИХ МОДУЛІВ

Файл проекту AttendanceSystem.csproj

```
&lt;Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>WinExe</OutputType>
    <TargetFramework>net8.0-windows</TargetFramework>
    <UseWindowsForms>true</UseWindowsForms>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="8.0.10" />
    <PackageReference Include="Microsoft.EntityFrameworkCore.Sqlite" Version="8.0.10" />
  </ItemGroup>

</Project>
```

Program.cs

```
using AttendanceSystem.Data;
using AttendanceSystem.Forms;
using Microsoft.EntityFrameworkCore;

namespace AttendanceSystem;

internal static class Program
{
    [STAThread]
    private static void Main()
    {
        ApplicationConfiguration.Initialize();

        using (var db = new ApplicationDbContext())
        {
            db.Database.EnsureCreated();
            DbSeeder.Seed(db);
        }

        Application.Run(new MainForm());
    }
}
```

Моделі даних

Models/Role.cs

```
namespace AttendanceSystem.Models;

public class Role
{
    [1] public int RoleId { get; set; }

    public [1] string RoleName { get; set; } = string.Empty; public List<User> Users { get; set; } = new();
}

Models/User.cs
```

namespace AttendanceSystem.Models;

```
[20] public class User { public int UserId { get; set; } public string FullName { get; set; } = string.Empty;

    public string Email { get; set; } = string[1] Empty;

    public string PasswordHash { get; set; } = string.Empty;

    public int RoleId { get; set; }

    public [8] bool IsActive { get; set; } = true;
```



```
public Role? Role { get; set; }
```

```
public List<Lesson> Lessons { get; set; } = new();  
}
```

Models/Group.cs

namespace AttendanceSystem.Models;

public class Group

```
{  
    public int GroupId { get; set; } public string GroupName { get; set; } = string.Empty;
```

```
    public int Course { get; set; } public string Specialty { get; set; } = string.Empty;
```

```
    public List<Student> Students { get; set; } = new();
```

```
    public List<Lesson> Lessons { get; set; } = new();
```

```
    public override string ToString()
```

```
{  
    return GroupName;  
}
```

Models/Student.cs

namespace AttendanceSystem.Models;

```
public class Student { public int StudentId { get; set; } public string LastName { get; set; } = string.Empty;
```

```
    public string FirstName { get; set; } = string.Empty;
```

```
    public string MiddleName { get; set; } = string.Empty;
```

```
    public int GroupId { get; set; }
```

```
    public string StudentCode { get; set; } = string.Empty;
```

```
    public int? UserId { get; set; } public Group? Group { get; set; }
```

```
    public User? User { get; set; }
```

```
    public List<AttendanceRecord> AttendanceRecords { get; set; } = new();
```

```
    public string FullName => $"{LastName} {FirstName} {MiddleName}".Trim();
```

```
    public override string ToString()
```

```
{  
    return FullName;  
}
```

Models/Subject.cs

```
namespace AttendanceSystem.Models; public class Subject { public int SubjectId { get; set; } public string SubjectName { get; set; } = string.Empty;
```

```
    public string SubjectCode { get; set; } = string.Empty;
```

```
    public string Description { get; set; } = string.Empty;
```

```
    public List<Lesson> Lessons { get; set; } = new();
```

```
    public override string ToString()
```

```
{  
    return SubjectName;  
}
```

Models/Lesson.cs

namespace AttendanceSystem.Models;

public class Lesson

```

1 public int LessonId { get; set; } public DateTime LessonDate { get; set; } public int GroupId { get; set; } public int SubjectId { get; set; } public
string LessonType { get; set; } = string.Empty;

public string Topic { get; set; } = string.Empty;

public int TeacherId { get; set; }

public Group? Group { get; set; }

public Subject? Subject { get; set; }

public User? Teacher { get; set; } public List<AttendanceRecord> AttendanceRecords { get; set; } = new();

public override string ToString()
{
    return $"{LessonDate:dd.MM.yyyy} | {Subject?.SubjectName} | {Group?.GroupName}";
}
}
Models/AttendanceStatus.cs
namespace AttendanceSystem.Models;

public class AttendanceStatus
{
    public int AttendanceStatusId { get; set; }

    public string StatusName { get; set; } = string.Empty;

    public decimal Coefficient { get; set; } public bool IsExcludedFromCalculation { get; set; } public bool IsNegativeAbsence { get; set; } public
List<AttendanceRecord> AttendanceRecords { get; set; } = new();

public override string ToString() { return StatusName;
}
}
Models/AttendanceRecord.cs
namespace AttendanceSystem.Models;

public class AttendanceRecord
{
    public int AttendanceRecordId { get; set; }

    public int LessonId { get; set; }

    public int StudentId { get; set; }

    public int AttendanceStatusId { get; set; }

    public string Comment { get; set; } = string.Empty;

    public DateTime CreatedAt { get; set; } = DateTime.Now;

    public Lesson? Lesson { get; set; } public Student? Student { get; set; } public AttendanceStatus? AttendanceStatus { get; set; }
}

```

Контекст бази даних

Data/ApplicationDbContext.cs

using AttendanceSystem.Models;

using Microsoft.EntityFrameworkCore;

namespace AttendanceSystem.Data;

```

5 public class ApplicationDbContext : DbContext
{
    public DbSet<Role> Roles => Set<Role>();
    public DbSet<User> Users => Set<User>(); public DbSet<Group> Groups => Set<Group>();
    public DbSet<Student> Students => Set<Student>();
    public DbSet<Subject> Subjects => Set<Subject>();
}

```

```

15 public DbSet<Lesson> Lessons => _Set<Lesson>();
public DbSet<AttendanceStatus> AttendanceStatuses => _Set<AttendanceStatus>();
public DbSet<AttendanceRecord> AttendanceRecords => _Set<AttendanceRecord>();

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder) { optionsBuilder. UseSqlite("Data Source=attendance.db");
}

```

```

15 protected override void OnModelCreating(ModelBuilder modelBuilder) { modelBuilder.Entity<Role>().
    .HasMany(r => r.Users)
    .WithOne(u => u.Role)
    .HasForeignKey(u => u.RoleId);

modelBuilder.Entity<Group>().
    .HasMany(g => g.Students)
    .WithOne(s => s.Group)
    .HasForeignKey(s => s.GroupId);

modelBuilder.Entity<Group>().
    .HasMany(g => g.Lessons)
    .WithOne(l => l.Group)
    .HasForeignKey(l => l.GroupId);

modelBuilder.Entity<Subject>().
    .HasMany(s => s.Lessons)
    .WithOne(l => l.Subject)
    .HasForeignKey(l => l.SubjectId);

modelBuilder.Entity<User>().
    .HasMany(u => u.Lessons)
    .WithOne(l => l.Teacher)
    .HasForeignKey(l => l.TeacherId);

modelBuilder.Entity<Lesson>().
    .HasMany(l => l.AttendanceRecords)
    .WithOne(r => r.Lesson)
    .HasForeignKey(r => r.LessonId);

modelBuilder.Entity<Student>().
    .HasMany(s => s.AttendanceRecords)
    .WithOne(r => r.Student)
    .HasForeignKey(r => r.StudentId);

modelBuilder.Entity<AttendanceStatus>().
    .HasMany(s => s.AttendanceRecords)
    .WithOne(r => r.AttendanceStatus)
    .HasForeignKey(r => r.AttendanceStatusId);

modelBuilder.Entity<AttendanceRecord>().
    .HasIndex(r => new { r.LessonId, r.StudentId })
    .IsUnique();
}
}

```

Data/DbSeeder.cs

```
using AttendanceSystem.Models;
```

```
namespace AttendanceSystem.Data;
```

```
public static class DbSeeder
```

```

{
    public static void Seed(ApplicationDbContext db)
    {
        if (!db.Roles.Any())
        {
            db.Roles.AddRange(
                new Role { RoleName = "Викладач" },
                new Role { RoleName = "Студент" },

```

```

        new Role { RoleName = "Уповноважена особа" },
        new Role { RoleName = "Батьки" },
        new Role { RoleName = "Технічний адміністратор" }
    );

    db.SaveChanges();
}

if (!db.Users.Any())
{
    var teacherRole = db.Roles.First(r => r.RoleName == "Викладач");

    db.Users.Add(new User
    {
        FullName = "Тестовий викладач",
        Email = "teacher@test.local",
        PasswordHash = "12345",
        RoleId = teacherRole.RoleId,
        IsActive = true
    });

    db.SaveChanges();
}

if (!db.AttendanceStatuses.Any())
{
    db.AttendanceStatuses.AddRange(
        new AttendanceStatus
        {
            StatusName = "Присутній",
            Coefficient = 1m,
            IsExcludedFromCalculation = false,
            IsNegativeAbsence = false
        },
        new AttendanceStatus
        {
            StatusName = "Запізнився",
            Coefficient = 0.5m,
            IsExcludedFromCalculation = false,
            IsNegativeAbsence = false
        },
        new AttendanceStatus
        {
            StatusName = "Відсутній",
            Coefficient = 0m,
            IsExcludedFromCalculation = false,
            IsNegativeAbsence = true
        },
        new AttendanceStatus
        {
            StatusName = "Відсутній з поважної причини",
            Coefficient = 0m,
            IsExcludedFromCalculation = true,
            IsNegativeAbsence = false
        }
    );

    db.SaveChanges();
}

if (!db.Groups.Any())
{
    db.Groups.Add(new Group
    {
        GroupName = "ІПЗ-21",
        Course = 2,
        Specialty = "Інженерія програмного забезпечення"
    });
}

```

```

    });

    db.SaveChanges();
}

if (!db.Subjects.Any())
{
    db.Subjects.AddRange(
        new Subject
        {
            SubjectName = "Об'єктно-орієнтоване програмування",
            SubjectCode = "ООР",
            Description = "Дисципліна з програмування"
        },
        new Subject
        {
            SubjectName = "Бази даних",
            SubjectCode = "ДВ",
            Description = "Дисципліна з проєктування баз даних"
        }
    );

    db.SaveChanges();
}

if (!db.Students.Any())
{
    var group = db.Groups.First();

    db.Students.AddRange(
        new Student
        {
            LastName = "Іваненко",
            FirstName = "Іван",
            MiddleName = "Іванович",
            GroupId = group.GroupId,
            StudentCode = "S001"
        },
        new Student
        {
            LastName = "Петренко",
            FirstName = "Петро",
            MiddleName = "Петрович",
            GroupId = group.GroupId,
            StudentCode = "S002"
        },
        new Student
        {
            LastName = "Сидоренко",
            FirstName = "Олена",
            MiddleName = "Миколаївна",
            GroupId = group.GroupId,
            StudentCode = "S003"
        }
    );

    db.SaveChanges();
}
}
}

Сервіси
Services/StudentService.cs
using AttendanceSystem.Data;
using AttendanceSystem.Models;
using Microsoft.EntityFrameworkCore;

namespace AttendanceSystem.Services;

```



```

public class StudentService
{
    public List<Student> GetAll()
    {
        using var db = new ApplicationDbContext();

        4return db.Students.Include(s => s.Group).OrderBy(s => s.LastName)
            .ToList();
    }

    public List<Student> GetByGroup(int groupId)
    {
        using var db = new ApplicationDbContext();

        return db.Students
            .Where(s => s.GroupId == groupId)
            .OrderBy(s => s.LastName)
            .ToList();
    }

    public void Add(Student student)
    {
        Validate(student);

        using var db = new ApplicationDbContext();
        db.Students.Add(student);
        db.SaveChanges();
    }

    public void Update(Student student)
    {
        Validate(student);

        using var db = new ApplicationDbContext();
        db.Students.Update(student);
        db.SaveChanges();
    }

    public void Delete(int studentId)
    {
        using var db = new ApplicationDbContext();

        var student = db.Students.Find(studentId);

        if (student == null)
        {
            throw new InvalidOperationException("Студента не знайдено.");
        }

        db.Students.Remove(student);
        db.SaveChanges();
    }

    private static void Validate(Student student)
    {
        if (string.IsNullOrEmpty(student.LastName))
        {
            throw new ArgumentException("Не вказано прізвище студента.");
        }

        if (string.IsNullOrEmpty(student.FirstName))
        {
            throw new ArgumentException("Не вказано ім'я студента.");
        }

        if (student.GroupId <= 0)
    }

```

```

    {
        throw new ArgumentException("Не вибрано навчальну групу.");
    }
}
}

```

```

Services/ReferenceService.cs
using AttendanceSystem.Data;
using AttendanceSystem.Models;
using Microsoft.EntityFrameworkCore;

```

```

namespace AttendanceSystem.Services;

```

```

public class ReferenceService
{
    public List<Group> GetGroups()
    {
        using var db = new ApplicationDbContext();

        return db.Groups
            .OrderBy(g => g.GroupName)
            .ToList();
    }

    public List<Subject> GetSubjects()
    {
        using var db = new ApplicationDbContext();

        return db.Subjects
            .OrderBy(s => s.SubjectName)
            .ToList();
    }

    public List<User> GetTeachers()
    {
        using var db = new ApplicationDbContext();

        return db.Users
            .Include(u => u.Role)
            .Where(u => u.Role != null && u.Role.RoleName == "Викладач")
            .OrderBy(u => u.FullName)
            .ToList();
    }

    public List<AttendanceStatus> GetStatuses()
    {
        using var db = new ApplicationDbContext();

        return db.AttendanceStatuses
            .OrderBy(s => s.AttendanceStatusId)
            .ToList();
    }

    public void AddGroup(4Group group)
    {
        if (string.IsNullOrEmpty(group.GroupName))
        {
4throw new ArgumentException("Назва групи не може бути порожньою.");
        }

        using var db = new ApplicationDbContext();
        db.Groups.Add(group);
        db.SaveChanges();
    }

    public void AddSubject(Subject subject)
    {

```

```

        if (string.IsNullOrEmpty(subject.SubjectName))
        {
            throw new ArgumentException("Назва дисципліни не може бути порожньою.");
        }

        using var db = new ApplicationDbContext();
        db.Subjects.Add(subject);
        db.SaveChanges();
    }
}

```

Services/LessonService.cs

```

using AttendanceSystem.Data;
using AttendanceSystem.Models;
using Microsoft.EntityFrameworkCore;

```

```

namespace AttendanceSystem.Services;

```

```

public class LessonService

```

```

{
    public List<Lesson> GetAll()
    {
        using var db = new ApplicationDbContext();

        return db.Lessons
            .Include(l => l.Group)
            .Include(l => l.Subject)
            .Include(l => l.Teacher)
            .OrderByDescending(l => l.LessonDate)
            .ToList();
    }

    public Lesson? GetById(int lessonId)
    {
        using var db = new ApplicationDbContext();

        return db.Lessons
            .Include(l => l.Group)
            .Include(l => l.Subject)
            .Include(l => l.Teacher)
            .FirstOrDefault(l => l.LessonId == lessonId);
    }

    public void Add(Lesson lesson)
    {
        Validate(lesson);

        using var db = new ApplicationDbContext();

        bool duplicateExists = db.Lessons.Any(l =>
            l.GroupId == lesson.GroupId &&
            l.SubjectId == lesson.SubjectId &&
            l.LessonDate.Date == lesson.LessonDate.Date &&
            l.LessonType == lesson.LessonType);

        if (duplicateExists)
        {
            throw new InvalidOperationException("Таке заняття вже існує.");
        }

        db.Lessons.Add(lesson);
        db.SaveChanges();
    }

    private static void Validate(Lesson lesson)
    {
        if (lesson.GroupId <= 0)

```

```

    {
        throw new ArgumentException("Не вибрано групу.");
    }

    if (lesson.SubjectId &lt;= 0)
    {
        throw new ArgumentException("Не вибрано дисципліну.");
    }

    if (lesson.TeacherId &lt;= 0)
    {
        throw new ArgumentException("Не вибрано викладача.");
    }

    if (string.IsNullOrEmpty(lesson.LessonType))
    {
        throw new ArgumentException("Не вказано тип заняття.");
    }
}
}

```

Services/AttendanceService.cs

```

using AttendanceSystem.Data;
using AttendanceSystem.Models;
using Microsoft.EntityFrameworkCore;

```

```

namespace AttendanceSystem.Services;

```

```

public class AttendanceService

```

```

{
    public void RegisterAttendance(int lessonId, List<AttendanceRecord> records)
    {
        if (lessonId &lt;= 0)
        {
            throw new ArgumentException("Не вибрано заняття.");
        }

        if (records.Count == 0)
        {
            throw new ArgumentException("Список записів відвідування порожній.");
        }

        using var db = new ApplicationDbContext();

        bool lessonExists = db.Lessons.Any(l => l.LessonId == lessonId);

        if (!lessonExists)
        {
            throw new InvalidOperationException("Заняття не знайдено.");
        }

        foreach (var record in records)
        {
            bool recordExists = db.AttendanceRecords.Any(r =>
                r.LessonId == lessonId & &
                r.StudentId == record.StudentId);

            if (recordExists)
            {
                var existing = db.AttendanceRecords.First(r =>
                    r.LessonId == lessonId & &
                    r.StudentId == record.StudentId);

                existing.AttendanceStatusId = record.AttendanceStatusId;
                existing.Comment = record.Comment;
                existing.CreatedAt = DateTime.Now;
            }
        }
    }
}

```

```

        else
        {
            record.LessonId = lessonId;
            record.CreatedAt = DateTime.Now;
            db.AttendanceRecords.Add(record);
        }
    }

    db.SaveChanges();
}

public List<AttendanceRecord> GetJournal()
{
    using var db = new ApplicationDbContext();

    return db.AttendanceRecords
        .Include(r => r.Student)
        .Include(r => r.Lesson)
        .ThenInclude(l => l.Subject)
        .Include(r => r.Lesson)
        .ThenInclude(l => l.Group)
        .Include(r => r.AttendanceStatus)
        .OrderByDescending(r => r.Lesson!.LessonDate)
        .ThenBy(r => r.Student!.LastName)
        .ToList();
}

public List<AttendanceRecord> GetJournalByGroup(int groupId)
{
    using var db = new ApplicationDbContext();

    return db.AttendanceRecords
        .Include(r => r.Student)
        .Include(r => r.Lesson)
        .ThenInclude(l => l.Subject)
        .Include(r => r.AttendanceStatus)
        .Where(r => r.Student != null && r.Student.GroupId == groupId)
        .OrderByDescending(r => r.Lesson!.LessonDate)
        .ToList();
}
}

```

```

Services/StatisticsService.cs
using AttendanceSystem.Data;
using Microsoft.EntityFrameworkCore;

namespace AttendanceSystem.Services;

public class StatisticsService
{
    public List<StudentAttendanceStatistics> GetStatisticsByGroup(int groupId)
    {
        using var db = new ApplicationDbContext();

        var students = db.Students
            .Where(s => s.GroupId == groupId)
            .OrderBy(s => s.LastName)
            .ToList();

        var result = new List<StudentAttendanceStatistics>();

        foreach (var student in students)
        {
            var records = db.AttendanceRecords
                .Include(r => r.AttendanceStatus)
                .Include(r => r.Lesson)
                .Where(r => r.StudentId == student.StudentId)

```



```

        .ToList();

int total = records.Count();
int excluded = records.Count(r => r.AttendanceStatus != null &&
    r.AttendanceStatus.IsExcludedFromCalculation);

int calculatedTotal = total - excluded;

decimal weightedSum = records
    .Where(r => r.AttendanceStatus != null &&
        !r.AttendanceStatus.IsExcludedFromCalculation)
    .Sum(r => r.AttendanceStatus!.Coefficient);

decimal rate = calculatedTotal > 0
    ? Math.Round(weightedSum / calculatedTotal * 100m, 2)
    : 0m;

int present = records.Count(r => r.AttendanceStatus != null &&
    r.AttendanceStatus.StatusName == "Присутній");

int late = records.Count(r => r.AttendanceStatus != null &&
    r.AttendanceStatus.StatusName == "Запізнився");

int absent = records.Count(r => r.AttendanceStatus != null &&
    r.AttendanceStatus.IsNegativeAbsence);

int validAbsent = excluded;

result.Add(new StudentAttendanceStatistics
{
    StudentId = student.StudentId,
    FullName = student.FullName,
    TotalLessons = total,
    CalculatedLessons = calculatedTotal,
    PresentCount = present,
    LateCount = late,
    AbsentCount = absent,
    ValidAbsentCount = validAbsent,
    AttendanceRate = rate,
    IsRiskStudent = rate < 70m && calculatedTotal > 0
});
}

return result;
}
}

21 public class StudentAttendanceStatistics
{
    public int StudentId { get; set; } public string FullName { get; set; } = string.Empty;

    public int TotalLessons { get; set; }

    public int CalculatedLessons { get; set; }

    public int PresentCount { get; set; }

    public int LateCount { get; set; }

    public int AbsentCount { get; set; }

    public int ValidAbsentCount { get; set; }

    public decimal AttendanceRate { get; set; }

    public bool IsRiskStudent { get; set; }
}

```

Форми Windows Forms

Forms/MainForm.cs

using AttendanceSystem.Forms;

namespace AttendanceSystem.Forms;

public class MainForm : Form

{

private Button btnStudents = new();

private Button btnLessons = new();

private Button btnAttendance = new();

private Button btnStatistics = new();

public MainForm()

{

Text = "Система реєстрації відвідування занять";

Width = 600;

Height = 400;

StartPosition = FormStartPosition.CenterScreen;

InitializeControls();

}

private void InitializeControls()

{

var title = new Label

{

Text = "Система реєстрації відвідування занять студентами",

AutoSize = false,

Width = 520,

Height = 40,

Left = 30,

Top = 30,

TextAlign = ContentAlignment.MiddleCenter,

Font = new Font("Segoe UI", 14, FontStyle.Bold)

};

btnStudents = CreateButton("Студенти", 180, 100);

btnLessons = CreateButton("Заняття", 180, 150);

btnAttendance = CreateButton("Реєстрація відвідування", 180, 200);

btnStatistics = CreateButton("Статистика", 180, 250);

btnStudents.Click += (_, _) => new StudentsForm().ShowDialog();

btnLessons.Click += (_, _) => new LessonsForm().ShowDialog();

btnAttendance.Click += (_, _) => new AttendanceForm().ShowDialog();

btnStatistics.Click += (_, _) => new StatisticsForm().ShowDialog();

Controls.Add(title);

Controls.Add(btnStudents);

Controls.Add(btnLessons);

Controls.Add(btnAttendance);

Controls.Add(btnStatistics);

}

private static Button CreateButton(string text, int left, int top)

{

return new Button

{

Text = text,

Left = left,

Top = top,

Width = 220,

Height = 35

};

}

}

```

Forms/StudentsForm.cs
using AttendanceSystem.Models;
using AttendanceSystem.Services;

namespace AttendanceSystem.Forms;

public class StudentsForm : Form
{
    private readonly StudentService _studentService = new();
    private readonly ReferenceService _referenceService = new();

    private DataGridView dgvStudents = new();
    private TextBox txtLastName = new();
    private TextBox txtFirstName = new();
    private TextBox txtMiddleName = new();
    private TextBox txtStudentCode = new();
    private ComboBox cmbGroups = new();
    private Button btnAdd = new();
    private Button btnRefresh = new();

    public StudentsForm()
    {
        Text = "Студенти";
        Width = 900;
        Height = 600;
        StartPosition = FormStartPosition.CenterParent;

        InitializeControls();
        LoadGroups();
        LoadStudents();
    }

    private void InitializeControls()
    {
        dgvStudents.Left = 20;
        dgvStudents.Top = 20;
        dgvStudents.Width = 520;
        dgvStudents.Height = 500;
        dgvStudents.ReadOnly = true;
        dgvStudents.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;

        CreateLabel("Прізвище:", 570, 30);
        txtLastName.SetBounds(570, 55, 250, 25);

        CreateLabel("Ім'я:", 570, 90);
        txtFirstName.SetBounds(570, 115, 250, 25);

        CreateLabel("По батькові:", 570, 150);
        txtMiddleName.SetBounds(570, 175, 250, 25);

        CreateLabel("Код студента:", 570, 210);
        txtStudentCode.SetBounds(570, 235, 250, 25);

        CreateLabel("Група:", 570, 270);
        cmbGroups.SetBounds(570, 295, 250, 25);
        cmbGroups.DropDownStyle = ComboBoxStyle.DropDownList;

        btnAdd.Text = "Додати студента";
        btnAdd.SetBounds(570, 340, 250, 35);
        btnAdd.Click += BtnAdd_Click;

        btnRefresh.Text = "Оновити";
        btnRefresh.SetBounds(570, 385, 250, 35);
        btnRefresh.Click += (_, _) => LoadStudents();

        Controls.AddRange(new Control[]

```

```

    {
        dgvStudents, txtLastName, txtFirstName, txtMiddleName,
        txtStudentCode, cmbGroups, btnAdd, btnRefresh
    });
}

```

```
private void CreateLabel(string text, int left, int top)
```

```

{
    Controls.Add(new Label
    {
        Text = text,
        Left = left,
        Top = top,
        Width = 250,
        Height = 20
    });
}

```

```
private void LoadGroups()
```

```

{
    cmbGroups.DataSource = _referenceService.GetGroups();
    cmbGroups.DisplayMember = "GroupName";
    cmbGroups.ValueMember = "GroupId";
}

```

```
private void LoadStudents()
```

```

{
    dgvStudents.DataSource = _studentService.GetAll()
        .Select(s => new
        {
            s.StudentId,
            ПІБ = s.FullName,
            Група = s.Group != null ? s.Group.GroupName : "",
            Код = s.StudentCode
        })
        .ToList();
}

```

```
private void BtnAdd_Click(object? sender, EventArgs e)
```

```

{
    try
    {
        if (cmbGroups.SelectedValue == null)
        {
            MessageBox.Show("Оберіть групу.");
            return;
        }

        var student = new Student
        {
            LastName = txtLastName.Text.Trim(),
            FirstName = txtFirstName.Text.Trim(),
            MiddleName = txtMiddleName.Text.Trim(),
            StudentCode = txtStudentCode.Text.Trim(),
            GroupId = Convert.ToInt32(cmbGroups.SelectedValue)
        };

        _studentService.Add(student);

        ClearFields();
        LoadStudents();

        MessageBox.Show("Студента додано.");
    }
}

```

```

2 }
catch (Exception ex)

```

```

{
    MessageBox.Show(ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

```
}  
}
```

```
private void ClearFields()  
{  
    txtLastName.Clear();  
    txtFirstName.Clear();  
    txtMiddleName.Clear();  
    txtStudentCode.Clear();  
}  
}
```

Forms/LessonsForm.cs

```
using AttendanceSystem.Models;  
using AttendanceSystem.Services;
```

```
namespace AttendanceSystem.Forms;
```

```
public class LessonsForm : Form  
{  
    private readonly ReferenceService _referenceService = new();  
    private readonly LessonService _lessonService = new();  
  
    private DataGridView dgvLessons = new();  
    private DateTimePicker dtpLessonDate = new();  
    private ComboBox cmbGroups = new();  
    private ComboBox cmbSubjects = new();  
    private ComboBox cmbTeachers = new();  
    private ComboBox cmbLessonType = new();  
    private TextBox txtTopic = new();  
    private Button btnAdd = new();  
  
    public LessonsForm()  
    {  
        Text = "Заняття";  
        Width = 950;  
        Height = 600;  
        StartPosition = FormStartPosition.CenterParent;  
  
        InitializeControls();  
        LoadReferences();  
        LoadLessons();  
    }  
  
    private void InitializeControls()  
    {  
        dgvLessons.SetBounds(20, 20, 560, 500);  
        dgvLessons.ReadOnly = true;  
        dgvLessons.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;  
  
        CreateLabel("Дата:", 610, 30);  
        dtpLessonDate.SetBounds(610, 55, 250, 25);  
  
        CreateLabel("Група:", 610, 90);  
        cmbGroups.SetBounds(610, 115, 250, 25);  
        cmbGroups.DropDownStyle = ComboBoxStyle.DropDownList;  
  
        CreateLabel("Дисципліна:", 610, 150);  
        cmbSubjects.SetBounds(610, 175, 250, 25);  
        cmbSubjects.DropDownStyle = ComboBoxStyle.DropDownList;  
  
        CreateLabel("Викладач:", 610, 210);  
        cmbTeachers.SetBounds(610, 235, 250, 25);  
        cmbTeachers.DropDownStyle = ComboBoxStyle.DropDownList;  
  
        CreateLabel("Тип заняття:", 610, 270);  
        cmbLessonType.SetBounds(610, 295, 250, 25);  
    }  
}
```

```

cmbLessonType.DropDownStyle = ComboBoxStyle.DropDownList;
cmbLessonType.Items.AddRange(new object[]
{
    "Лекція",
    "Практичне заняття",
    "Лабораторне заняття",
    "Семінар"
});

CreateLabel("Тема:", 610, 330);
txtTopic.SetBounds(610, 355, 250, 25);

btnAdd.Text = "Створити заняття";
btnAdd.SetBounds(610, 400, 250, 35);
btnAdd.Click += BtnAdd_Click;

Controls.AddRange(new Control[]
{
    dgvLessons, dtpLessonDate, cmbGroups, cmbSubjects,
    cmbTeachers, cmbLessonType, txtTopic, btnAdd
});
}

private void CreateLabel(string text, int left, int top)
{
    Controls.Add(new Label
    {
        Text = text,
        Left = left,
        Top = top,
        Width = 250,
        Height = 20
    });
}

private void LoadReferences()
{
    cmbGroups.DataSource = _referenceService.GetGroups();
    cmbGroups.DisplayMember = "GroupName";
    cmbGroups.ValueMember = "GroupId";

    cmbSubjects.DataSource = _referenceService.GetSubjects();
    cmbSubjects.DisplayMember = "SubjectName";
    cmbSubjects.ValueMember = "SubjectId";

    cmbTeachers.DataSource = _referenceService.GetTeachers();
    cmbTeachers.DisplayMember = "FullName";
    cmbTeachers.ValueMember = "UserId";

    if (cmbLessonType.Items.Count > 0)
    {
        cmbLessonType.SelectedIndex = 0;
    }
}

private void LoadLessons()
{
    dgvLessons.DataSource = _lessonService.GetAll()
        .Select(l => new
        {
            l.LessonId,
            Дата = l.LessonDate.ToString("dd.MM.yyyy"),
            Група = l.Group?.GroupName,
            Дисципліна = l.Subject?.SubjectName,
            Тип = l.LessonType,
            Тема = l.Topic,
            Викладач = l.Teacher?.FullName
        });
}

```



```

    })
    .ToList();
}

private void BtnAdd_Click(object? sender, EventArgs e)
{
    try
    {
        var lesson = new Lesson
        {
            LessonDate = dtpLessonDate.Value.Date,
            GroupId = Convert.ToInt32(cmbGroups.SelectedValue),
            SubjectId = Convert.ToInt32(cmbSubjects.SelectedValue),
            TeacherId = Convert.ToInt32(cmbTeachers.SelectedValue),
            LessonType = cmbLessonType.Text,
            Topic = txtTopic.Text.Trim()
        };

        _lessonService.Add(lesson);
        LoadLessons();

        MessageBox.Show("Заняття створено.");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
}

```

Forms/AttendanceForm.cs

```

using AttendanceSystem.Models;
using AttendanceSystem.Services;

```

```

namespace AttendanceSystem.Forms;

```

```

public class AttendanceForm : Form

```

```

{
    private readonly LessonService _lessonService = new();
    private readonly StudentService _studentService = new();
    private readonly ReferenceService _referenceService = new();
    private readonly AttendanceService _attendanceService = new();

```

```

    private ComboBox cmbLessons = new();
    private DataGridView dgvAttendance = new();
    private Button btnLoadStudents = new();
    private Button btnSave = new();
    private Button btnAllPresent = new();

```

```

    private List<Student> _students = new();
    private List<AttendanceStatus> _statuses = new();

```

```

    public AttendanceForm()
    {
        Text = "Реєстрація відвідування";
        Width = 1000;
        Height = 650;
        StartPosition = FormStartPosition.CenterParent;

        InitializeControls();
        LoadReferences();
    }

```

```

    private void InitializeControls()
    {
        CreateLabel("Заняття:", 20, 20);
    }

```

```
cmbLessons.SetBounds(20, 45, 700, 25);
cmbLessons.DropDownStyle = ComboBoxStyle.DropDownList;
```

```
btnLoadStudents.Text = "Завантажити студентів";
btnLoadStudents.SetBounds(740, 43, 200, 30);
btnLoadStudents.Click += BtnLoadStudents_Click;
```

```
dgvAttendance.SetBounds(20, 90, 920, 460);
dgvAttendance.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
dgvAttendance.AllowUserToAddRows = false;
```

```
btnAllPresent.Text = "Позначити всіх присутніми";
btnAllPresent.SetBounds(20, 565, 230, 35);
btnAllPresent.Click += BtnAllPresent_Click;
```

```
btnSave.Text = "Зберегти відвідування";
btnSave.SetBounds(710, 565, 230, 35);
btnSave.Click += BtnSave_Click;
```

```
Controls.AddRange(new Control[]
{
    cmbLessons, btnLoadStudents, dgvAttendance, btnAllPresent, btnSave
});
}
```

```
private void CreateLabel(string text, int left, int top)
{
    Controls.Add(new Label
    {
        Text = text,
        Left = left,
        Top = top,
        Width = 250,
        Height = 20
    });
}
```

```
private void LoadReferences()
{
    var lessons = _lessonService.GetAll();
    cmbLessons.DataSource = lessons;
    cmbLessons.DisplayMember = "ToString";
    cmbLessons.ValueMember = "LessonId";

    _statuses = _referenceService.GetStatuses();
}
```

```
private void BtnLoadStudents_Click(object? sender, EventArgs e)
{
    if (cmbLessons.SelectedItem is not Lesson lesson)
    {
        MessageBox.Show("Оберіть заняття.");
        return;
    }
}
```

```
_students = _studentService.GetByGroup(lesson.GroupId);
```

```
dgvAttendance.Columns.Clear();
dgvAttendance.Rows.Clear();
```

```
dgvAttendance.Columns.Add("StudentId", "StudentId");
dgvAttendance.Columns["StudentId"]!.Visible = false;
```

```
dgvAttendance.Columns.Add("FullName", "Студент");
```

```
var statusColumn = new DataGridViewComboBoxColumn
{
```

```

        Name = "Status",
        HeaderText = "Статус",
        DataSource = _statuses,
        DisplayMember = "StatusName",
        ValueMember = "AttendanceStatusId"
    };

    dgvAttendance.Columns.Add(statusColumn);
    dgvAttendance.Columns.Add("Comment", "Коментар");

    var defaultStatus = _statuses.FirstOrDefault(s => s.StatusName == "Присутній");

    foreach (var student in _students)
    {
        dgvAttendance.Rows.Add(
            student.StudentId,
            student.FullName,
            defaultStatus?.AttendanceStatusId,
            ""
        );
    }
}

private void BtnAllPresent_Click(object? sender, EventArgs e)
{
    var present = _statuses.FirstOrDefault(s => s.StatusName == "Присутній");

    if (present == null)
    {
        MessageBox.Show("Статус 'Присутній' не знайдено.");
        return;
    }

    foreach (DataGridViewRow row in dgvAttendance.Rows)
    {
        row.Cells["Status"].Value = present.AttendanceStatusId;
    }
}

private void 12BtnSave_Click(object? sender, EventArgs e)
{
    try.
    {
        if (cmbLessons.SelectedItem is not Lesson lesson)
        {
            MessageBox.Show("Оберіть заняття.");
            return;
        }

        var records = new List<AttendanceRecord>();

        foreach (DataGridViewRow row in dgvAttendance.Rows)
        {
            if (row.Cells["StudentId"].Value == null ||
                row.Cells["Status"].Value == null)
            {
                continue;
            }

            records.Add(new AttendanceRecord
            {
                StudentId = Convert.ToInt32(row.Cells["StudentId"].Value),
                AttendanceStatusId = Convert.ToInt32(row.Cells["Status"].Value),
                Comment = row.Cells["Comment"].Value?.ToString() ?? ""
            });
        }
    }
}

```

```
_attendanceService.RegisterAttendance(lesson.LessonId, records);
```

```
MessageBox.Show("Відвідування збережено.");
```

```
}  
2 catch (Exception ex)
```

```
{  
    2 MessageBox.Show(ex.Message, "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);  
}
```

```
}  
}
```

Forms/StatisticsForm.cs

```
using AttendanceSystem.Services;
```

```
namespace AttendanceSystem.Forms;
```

```
public class StatisticsForm : Form
```

```
{  
    private readonly ReferenceService _referenceService = new();  
    private readonly StatisticsService _statisticsService = new();
```

```
    private ComboBox cmbGroups = new();  
    private Button btnCalculate = new();  
    private DataGridView dgvStatistics = new();
```

```
    public StatisticsForm()  
    {  
        Text = "Статистика відвідування";  
        Width = 1000;  
        Height = 600;  
        StartPosition = FormStartPosition.CenterParent;
```

```
        InitializeControls();  
        LoadGroups();  
    }
```

```
    private void InitializeControls()  
    {  
        CreateLabel("Група.", 20, 20);  
  
        cmbGroups.SetBounds(20, 45, 300, 25);  
        cmbGroups.DropDownStyle = ComboBoxStyle.DropDownList;  
  
        btnCalculate.Text = "Розрахувати статистику";  
        btnCalculate.SetBounds(340, 43, 220, 30);  
        btnCalculate.Click += BtnCalculate_Click;  
  
        dgvStatistics.SetBounds(20, 90, 930, 430);  
        dgvStatistics.ReadOnly = true;  
        dgvStatistics.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
```

```
        Controls.AddRange(new Control[]  
        {  
            cmbGroups, btnCalculate, dgvStatistics  
        });  
    }
```

```
    private void CreateLabel(string text, int left, int top)  
    {  
        Controls.Add(new Label  
        {  
            Text = text,  
            Left = left,  
            Top = top,  
            Width = 250,  
            Height = 20  
        });  
    }
```

```

}

private void LoadGroups()
{
    cmbGroups.DataSource = _referenceService.GetGroups();
    cmbGroups.DisplayMember = "GroupName";
    cmbGroups.ValueMember = "GroupId";
}

18 private void BtnCalculate_Click(object? sender, EventArgs e) { if (cmbGroups.SelectedValue == null)
{
    MessageBox.Show("Оберіть групу.");
    return;
}

    int groupId = Convert.ToInt32(cmbGroups.SelectedValue);

    var statistics = _statisticsService.GetStatisticsByGroup(groupId)
        .Select(s => new
        {
            Студент = s.FullName,
            Усього_занять = s.TotalLessons,
            Враховано_занять = s.CalculatedLessons,
            Присутній = s.PresentCount,
            Запізнився = s.LateCount,
            Відсутній = s.AbsentCount,
            Поважна_причина = s.ValidAbsentCount,
            Відсоток = s.AttendanceRate,
            Група_ризикy = s.IsRiskStudent ? "Так" : "Ні"
        })
        .ToList();

    dgvStatistics.DataSource = statistics;
}
}

```